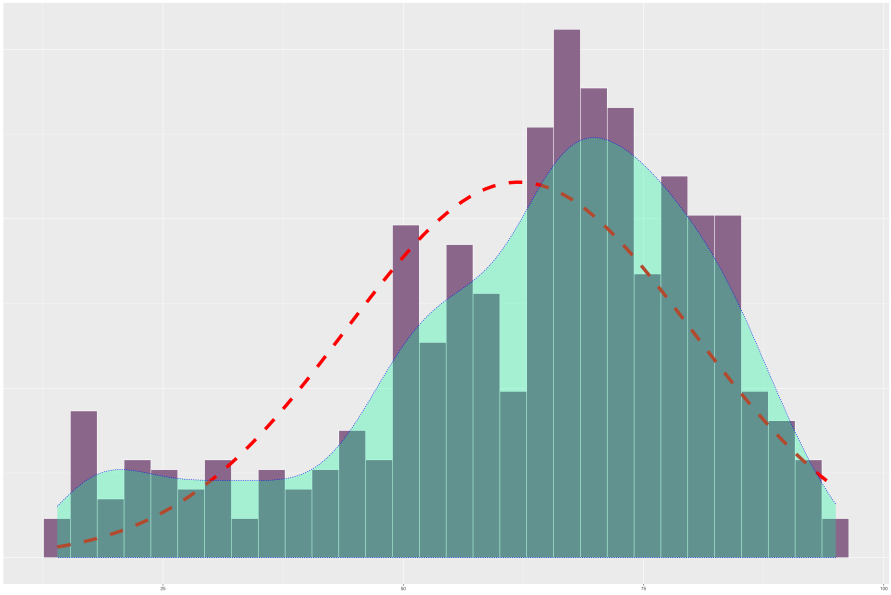
Hochschule Niederrhein
University of Applied Sciences

Gesundheitswesen
Faculty of Health Care

Statistik mit R und RStudio



Ein Nachschlagewerk für Gesundheitsberufe

zusammengestellt von
Jörg große Schlarmann

Version vom 25. November 2024

Das Logo der Hochschule Niederrhein sowie die Logos von `R`, `RStudio` und `quarto` sind Eigentum der jeweiligen Rechteinhaber. Für alles weitere gelten die folgenden Bedingungen.



Dieses Werk ist unter der Creative Commons Lizenz `CC BY-NC-SA 4.0` verfügbar, siehe
https://creativecommons.org/licenses/by-nc-sa/4.0/deed.de

Sie dürfen:

- das Material in jedwedem Format oder Medium vervielfältigen und weiterverbreiten.
- das Material remixen, verändern und darauf aufbauen.

Unter folgenden Bedingungen:

Namensnennung: Sie müssen angemessene Urheber[1]-und Rechteangaben machen, einen Link zur Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Diese Angaben dürfen in jeder angemessenen Art und Weise gemacht werden, allerdings nicht so, dass der Eindruck entsteht, der Lizenzgeber unterstütze gerade Sie oder Ihre Nutzung besonders.

Nicht kommerziell: Sie dürfen das Material nicht für kommerzielle Zwecke nutzen.

Weitergabe unter gleichen Bedingungen: Wenn Sie das Material remixen, verändern oder anderweitig direkt darauf aufbauen, dürfen Sie Ihre Beiträge nur unter derselben Lizenz wie das Original verbreiten.

Keine weiteren Einschränkungen: Sie dürfen keine zusätzlichen Klauseln oder technische Verfahren einsetzen, die anderen rechtlich irgendetwas untersagen, was die Lizenz erlaubt.



Das Nachschlagewerk wird - im Sinne eines *rolling releases* - ständig aktualisiert und erweitert.
Daher hat es keine ISBN- oder DOI-Nummer.
Eine aktuelle PDF-Version dieses Buches finden Sie unter: https://www.produnis.de/R/rbuch.pdf
Eine aktuelle epub-Version dieses Buches finden Sie unter: https://www.produnis.de/R/rbuch.epub
Dieses Buch steht zudem als Webseite bereit unter: https://www.produnis.de/R/

Kritik und Diskussion sind per Mastodon möglich: https://mastodon.social/@rbuch

[1] **Zitationsvorschlag:** große Schlarmann, J (2024): "*Statistik mit R und RStudio - Ein Nachschlagewerk für Gesundheitsberufe*", Hochschule Niederrhein, Krefeld, https://www.produnis.de/R/

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Installation von R und RStudio . . . 2
1.2 Kurze Einführung in R und RStudio . . . 3
1.2.1 R . . . 3
1.2.2 RStudio . . . 4
1.2.2.1 RStudio anpassen . . . 5
1.2.2.2 Eine Sitzung starten . . . 7
1.2.2.3 Projekte in RStudio . . . 11
1.2.3 Grundlegende Befehle in R . . . 14
1.2.3.1 Rechnen mit R . . . 15
1.2.3.2 Variablen . . . 19
1.2.3.3 Wertesequenzen erzeugen . . . 23
1.2.3.4 Datentypen . . . 27
1.2.3.5 Datenklassen . . . 29
1.2.3.5.1 Vektoren . . . 30
1.2.3.5.2 Matrizen . . . 31
1.2.3.5.3 Faktoren . . . 37
1.2.3.5.4 ordinale Faktoren . . . 40
1.2.3.5.5 Datenframes . . . 42
1.2.3.5.6 Datenframes: Fälle hinzufügen . . . 43
1.2.3.5.7 Datenframes: Spalten hinzufügen . . . 45
1.2.3.5.8 Datenframes zusammenführen . . . 46
1.2.3.5.9 Datenframes: Fälle und Variablen auswählen . . . 47
1.2.3.5.10 Listen . . . 54
1.2.3.6 Fehlende Daten . . . 56
1.2.3.7 Umgang mit Datensätzen . . . 57
1.2.3.7.1 Sortieren . . . 58
1.2.3.7.2 Teilgruppen erstellen . . . 60
1.2.3.8 Klassen für Variablen bilden (klassieren) . . . 61
1.2.3.9 Pipe . . . 64
1.2.3.10 Daten laden und speichern . . . 65
1.2.3.11 Arbeitsverzeichnis . . . 67
1.2.3.12 Daten importieren . . . 68
1.2.3.12.1 Import aus Textdateien . . . 68
1.2.3.12.2 Import aus CSV-Datei . . . 71
1.2.3.12.3 Import aus SPSS-Datei . . . 72
1.2.3.12.4 Import aus Excel-Datei . . . 73
1.2.3.12.5 Import aus .ods-Datei . . . 74
1.2.3.12.6 Import mit RStudio . . . 75
1.2.3.12.7 Importierte Daten ins richtige Format bringen . . . 76

1.2.3.13 Daten exportieren . . . 79
1.2.3.13.1 Export als CSV-Datei . . . 79
1.2.3.13.2 Export als Textdatei-Datei . . . 79
1.2.3.13.3 Export als SPSS-Datei . . . 79
1.2.3.14 Zusatzpakete installieren . . . 80
1.2.3.14.1 Zusatzpaket zu diesem Lehrbuch . . . 84
1.2.3.15 Ausgabe in Datei umleiten . . . 85
1.2.4 Verteilungsfunktionen . . . 87
1.2.4.1 Normalverteilung . . . 87
1.2.4.2 t-Verteilung . . . 91
1.2.4.3 Chiquadrat-Verteilung . . . 93
1.2.4.4 Poisson-Verteilung . . . 94
1.2.4.5 Binomial-Verteilung . . . 97
1.2.5 Eigene Funktionen programmieren . . . 99
1.2.5.1 Beispiele für Funktionen . . . 100
1.2.5.1.1 z-Transformation . . . 100
1.2.5.1.2 Sensitivität / Spezifität . . . 100
1.2.5.1.3 Kenngrößen . . . 101
1.2.5.1.4 Häufigkeitstabellen . . . 101
1.2.5.2 Bedingungen . . . 101
1.2.5.2.1 Beispiel Zusatzpakete . . . 102
1.2.5.2.2 Beispiel verschiedene lineare Modelle vergleichen . . . 104
1.2.5.3 Funktionen in Dateien speichern . . . 107
1.2.6 Funktionen über Datenreihen anwenden . . . 108
1.2.6.1 apply() . . . 108
1.2.6.2 Funktionsparameter spezifizieren . . . 110
1.2.6.3 sapply() . . . 113
1.2.6.4 lapply() . . . 114
1.2.6.5 tapply() . . . 115
1.3 Markdown . . . 116
1.4 RMarkdown . . . 117
1.4.1 Syntax . . . 121
1.5 quarto . . . 127
1.5.1 Installation . . . 127
1.5.2 Dokument erstellen . . . 127
1.5.3 Editor-Ansicht . . . 131
1.5.4 Dokumentenaufbau . . . 132
1.5.5 Syntax . . . 132
1.5.5.1 Textformat . . . 133
1.5.5.2 Überschriften . . . 133
1.5.5.3 Links und Bilder . . . 134
1.5.5.4 Listen . . . 134
1.5.5.5 Tabellen . . . 134
1.5.5.6 Gleichungen . . . 135
1.5.6 R-Code integrieren . . . 135
1.5.6.1 Chunk-Optionen . . . 136

1.5.6.2 Intext-Codeintegration . . . 138
1.5.7 PDF-Dokumente erzeugen . . . 139
1.5.8 quarto-Dokumentation . . . 140
1.6 Tidyverse . . . 141
1.6.1 Pakete . . . 143
1.6.2 Pipe . . . 144
1.6.3 Unterschiede zwischen `%>%` und `|>` . . . 145
1.6.4 tibbles . . . 145
1.6.5 Funktionsaufrufe . . . 148
1.6.6 Schritt 1: Daten importieren . . . 149
1.6.6.1 gelabelte SPSS-Daten . . . 149
1.6.7 Schritt 2: Daten *tidy* machen . . . 151
1.6.7.0.1 verschachtelte `wide table` . . . 152
1.6.7.1 Datentypen korrigieren . . . 154
1.6.7.2 fehlende Werte . . . 157
1.6.7.3 Variablen umbenennen . . . 160
1.6.8 Schritt 3: Umgang mit Datensätzen . . . 160
1.6.8.1 Daten filtern und sortieren . . . 160
1.6.8.2 Fälle auswählen . . . 163
1.6.8.2.1 Gruppierungen . . . 166
1.6.8.3 Variablen hinzufügen . . . 167
1.6.8.4 Fälle hinzufügen . . . 168
1.6.8.5 Datensätze verbinden . . . 169
1.6.8.5.1 `inner_join()` . . . 172
1.6.8.5.2 `left_join()` . . . 172
1.6.8.5.3 `right_join()` . . . 173
1.6.8.5.4 `full_join()` . . . 175
1.6.8.6 Variablen auswählen . . . 175
1.6.8.7 Variablen erzeugen . . . 176
1.6.8.8 statistische Berechnungen . . . 177
1.6.8.9 Umgang mit Faktoren . . . 180
1.6.8.9.1 Levels umbenennen . . . 180
1.6.8.9.2 Levelreihenfolge ändern . . . 181
1.6.8.9.3 Levels zusammenfassen . . . 186
1.6.8.10 Daten klassieren . . . 188
1.6.8.11 Daten visualisieren . . . 190
1.6.9 Schritt 4: Ergebisse kommunizieren . . . 190
1.6.10 Die Macht der `Pipe` . . . 191
1.7 data.table . . . 192
1.7.1 Installation . . . 192
1.7.2 Modify-in-Place . . . 192
1.7.3 Grundlegende Syntax . . . 193
1.7.4 Daten einlesen . . . 193
1.7.5 Daten speichern . . . 194
1.7.6 Fälle filtern mit `i` . . . 195
1.7.7 Fälle sortieren mit `i` . . . 196

1.7.8 Daten verarbeiten mit `j` . . . 196
1.7.9 Daten bearbeiten mit `j` . . . 197
1.7.10 `data.table` kopieren . . . 198
1.7.11 pipen . . . 200
1.7.12 Ergebnisse gruppieren mit `by` . . . 201
1.7.13 Weitere Funktionen aus dem `data.table` Paket . . . 202
1.7.13.1 Einzigartige bestimmen mit `uniqueN` . . . 203
1.7.13.2 Anzahl der Fälle mit `.N` . . . 203
1.7.13.3 Lange Tabelle erzeugen mit `melt()` . . . 204
1.7.13.4 Breite Tabelle erzeugen mit `dcast()` . . . 205
1.7.13.5 Subset of Data (`.SD`) . . . 206
1.7.13.5.1 Verwendung von `.SDcols` . . . 207
1.7.14 Cheat Sheet und Übungsaufgaben . . . 208
1.8 Weitere Hilfsmittel . . . 209
1.8.1 Cheatsheets . . . 209
1.8.2 Youtube . . . 210
1.8.3 freie Lehrbücher . . . 210
1.8.4 Internetforen . . . 210

2 Statistik mit `R` und `RStudio` 211
2.1 Deskriptive Statistik . . . 212
2.1.1 Häufigkeiten . . . 212
2.1.2 Lagekenngrößen . . . 213
2.1.3 Streuungskenngrößen . . . 217
2.1.4 Kreuztabellen . . . 218
2.1.5 z-Transformation . . . 219
2.1.6 Korrelation . . . 220
2.2 Schließende Statistik . . . 223
2.2.1 Regressionen . . . 223
2.2.1.1 multiple lineare Regressionen . . . 226
2.2.1.2 nicht-lineare Regression . . . 227
2.2.1.2.1 Funktionen höherer Ordnung . . . 230
2.2.1.2.2 Exponentialfunktion . . . 233
2.2.1.2.3 Sigmoidalfunktion . . . 235
2.2.2 Generalisiert lineare Modelle . . . 237
2.2.2.1 lineare Regression . . . 238
2.2.3 Poissonregression . . . 239
2.2.3.1 Dispersion . . . 243
2.2.3.2 Nullmodell . . . 244
2.2.3.3 Vorhersagen . . . 244
2.2.4 Logistische Regression . . . 247
2.2.5 Ordinale Regression . . . 250
2.2.5.1 Proportional Odds Modelle . . . 250
2.2.5.2 Continuation Ratio Model . . . 255
2.2.6 Konfidenzintervalle . . . 259
2.2.6.1 Normalverteilung . . . 259

2.2.6.2 Binomialverteilung . . . 266
2.2.7 Signifikanztests . . . 275
2.2.7.1 Chiquadrat-Test . . . 275
2.2.7.2 Fisher's Exakttest . . . 275
2.2.7.3 t-Test . . . 275
2.2.7.3.1 Einstichproben t-Test . . . 276
2.2.7.3.2 unabhängiger t-Test . . . 276
2.2.7.3.3 abhängiger t-Test . . . 278
2.2.7.4 F-Test . . . 278
2.2.7.5 Levene-Test . . . 279
2.2.7.6 Mann-Whitney-U-Test . . . 279
2.2.7.7 Wilcoxon-Test . . . 280
2.2.7.8 Shapiro-Wilk-Normalverteilungstest . . . 281
2.2.7.9 Kolmogorov-Smirnov-Test . . . 282
2.2.8 Post-Hoc-Tests . . . 283
2.2.8.1 Paarweise Chiquadrat . . . 283
2.2.8.2 Paarweise Fisher-Test . . . 283
2.2.8.3 Paarweise Wilcoxon-Test (Mann-Whitney-U) . . . 284
2.2.8.4 Paarweise t-Test . . . 284
2.2.9 Varianzanalysen . . . 285
2.2.9.1 ANOVA mit Messwiederholung . . . 285
2.2.9.2 Friedman ANOVA . . . 291
2.2.9.3 einfaktorielle ANOVA . . . 296
2.2.10 Überlebenszeitanalysen . . . 298
2.2.10.1 Kaplan-Meier-Analysen . . . 298
2.2.10.2 Cox-Regression . . . 305
2.2.11 Faktorenanalyse . . . 306
2.2.11.1 explorative Faktorenanalyse . . . 306
2.2.11.2 konfirmatorische Faktorenanalyse . . . 309
2.2.11.3 Hauptkomponentenanalyse . . . 313
2.2.12 Fallzahlkalkulation . . . 315
2.2.12.1 klinische Studien . . . 315
2.2.12.2 Umfragen . . . 316
2.3 Diagramme plotten . . . 320
2.3.1 Punktwolke . . . 322
2.3.2 verfügbare Farben . . . 327
2.3.3 Liniendiagramm . . . 329
2.3.4 Histogram . . . 331
2.3.5 Kreisdiagramm . . . 335
2.3.6 Säulendiagramm . . . 338
2.3.7 Balkendiagramm . . . 340
2.3.8 Boxplot . . . 342
2.3.9 Polygone . . . 344
2.3.10 QQ-Plots . . . 346
2.3.11 Legendenbox . . . 347
2.3.12 Diagramme speichern . . . 350

2.4 ggplot . . . 352
2.4.1 Satzbau . . . 352
2.4.2 Erstes Plot . . . 353
2.4.2.1 Aesthetics . . . 354
2.4.2.2 Geome . . . 355
2.4.2.3 mehr Aesthetics . . . 356
2.4.2.4 mehr Geome . . . 356
2.4.2.5 Facetten . . . 359
2.4.2.6 Geom-Parameter . . . 361
2.4.3 Punktwolke . . . 363
2.4.4 Liniendiagramme . . . 365
2.4.5 Histogramm . . . 369
2.4.5.1 Histogramm und Dichteverteilung . . . 371
2.4.5.2 kumulierte Häufigkeiten . . . 374
2.4.6 Säulendiagramme . . . 376
2.4.6.1 vorgegebene Werte . . . 382
2.4.7 Balkendiagramme . . . 384
2.4.8 Boxplots . . . 386
2.4.9 Kreisdiagramme . . . 391
2.4.10 QQ-Plots . . . 394
2.4.11 Diagramme speichern . . . 396
2.4.11.1 R-Objekt . . . 396
2.4.11.2 in Datei speichern . . . 398
2.4.12 Aussehen ändern . . . 399
2.4.12.1 Titel und Überschriften . . . 399
2.4.12.2 Achsen . . . 400
2.4.12.3 Legendenbox . . . 403
2.4.12.4 Farben . . . 406
2.4.12.5 Linien . . . 409
2.4.12.6 Text . . . 410
2.4.12.7 Sonderzeichen . . . 411
2.4.12.8 Themes . . . 412

3 Was willst du machen? 415
3.1 Daten manipulieren . . . 416
3.1.1 Einer Kreuztabelle Prozentwerte hinzufügen . . . 416
3.1.2 Eine gemeinsame Kreuztabelle mit mehrere Variablen erstellen . . . 416
3.1.2.1 Zeilen- und Spaltensummen . . . 418
3.1.2.2 Prozenttabellen . . . 419
3.1.2.3 Signifikanztests . . . 420
3.1.2.4 eine große Kreuztabelle . . . 420
3.1.3 Zeilen und Spalten tauschen . . . 422
3.1.4 binäre Dummy-Variablen zusammenführen . . . 422
3.2 Diagramme plotten . . . 424
3.2.1 Alle R-Farben . . . 424
3.2.2 Normalverteilung . . . 426

3.2.3 t-Verteilung . . . 436
3.2.4 Chiquadrat-Verteilung . . . 439
3.2.5 Poisson-Verteilung . . . 442
3.2.6 BMI-Gewichtskategorien . . . 445
3.2.6.1 R base . . . 445
3.2.6.2 ggplot() . . . 447
3.2.6.2.1 Alternative . . . 448
3.2.7 Wie hast du das Titelbild erzeugt? . . . 450
3.3 Karten plotten . . . 452
3.3.1 Deutschland . . . 452
3.3.2 Kartendaten einlesen . . . 452
3.3.3 Karten erzeugen . . . 453
3.3.4 Kacheln füllen . . . 455
3.4 Pakete verwalten . . . 458
3.4.1 Pakete beim Start automatisch laden . . . 458
3.5 `RMarkdown` . . . 459
3.5.1 Ausgabeformat per `.css`-Datei ändern . . . 459
3.6 `quarto` . . . 460
3.6.1 eigene LaTeXVorlagen erstellen . . . 460
3.6.1.1 `.tex`-Date . . . 460
3.6.1.2 Vorlage einbinden . . . 461
3.6.1.3 `YAML`-Variablen . . . 461
3.6.2 quarto am Handy oder Tablet . . . 462
3.7 Referenztabellen erstellen . . . 463
3.7.1 Fallzahlen nach Effektstärke und Power . . . 463
3.7.2 Verteilungsfunktion und Quantile der Standardnormalverteilung . . . 466
3.7.3 Kritische Werte $\mathbf{t}^*$ der $\mathbf{t}$-Verteilung . . . 470
3.7.4 Kritische Werte der Chiquadrat-Verteilung . . . 472
3.8 COVID19 Fallzahlen analysieren . . . 474

4 Übungsaufgaben **480**
4.1 Aufgaben . . . 480
4.1.1 Häufigkeitsverteilungen . . . 480
4.1.1.1 Aufgabe 4.1.1.1 Kinder in Familien . . . 480
4.1.1.2 Aufgabe 4.1.1.2 Patienten in der Notaufnahme . . . 481
4.1.1.3 Aufgabe 4.1.1.3 Blutgruppen . . . 481
4.1.1.4 Aufgabe 4.1.1.4 Familienstand . . . 481
4.1.1.5 Aufgabe 4.1.1.5 Handballverletzungen . . . 482
4.1.1.6 Aufgabe 4.1.1.6 Körpergröße . . . 482
4.1.1.7 Aufgabe 4.1.1.7 Neugeborene . . . 483
4.1.2 Stichprobenstatistik . . . 484
4.1.2.1 Aufgabe 4.1.2.1 Kinder in Familien . . . 484
4.1.2.2 Aufgabe 4.1.2.2 Patienten in Notaufnahme . . . 485
4.1.2.3 Aufgabe 4.1.2.3 Studierendenbewertung . . . 485
4.1.2.4 Aufgabe 4.1.2.4 Körpergröße nach Geschlecht . . . 485
4.1.2.5 Aufgabe 4.1.2.5 Handballverletzungen . . . 486

4.1.2.6 Aufgabe 4.1.2.6 Blutdruckmessung . . . 486
4.1.2.7 Aufgabe 4.1.2.7 Alter und Familienstand . . . 487
4.1.2.8 Aufgabe 4.1.2.8 Tabak, Alkohol und Blutdruck . . . 487
4.1.3 Lineare Regression . . . 488
4.1.3.1 Aufgabe 4.1.3.1 X und Y . . . 488
4.1.3.2 Aufgabe 4.1.3.2 Lernen und Durchfallen . . . 488
4.1.3.3 Aufgabe 4.1.3.3 Metabolismus . . . 489
4.1.3.4 Aufgabe 4.1.3.4 Alter und Körpergröße . . . 490
4.1.3.5 Aufgabe 4.1.3.5 Wirksamkeitsverlust . . . 490
4.1.3.6 Aufgabe 4.1.3.6 Dosierung . . . 491
4.1.3.7 Aufgabe 4.1.3.7 Gewicht und Körpergröße . . . 491
4.1.3.8 Aufgabe 4.1.3.8 Neugeborene . . . 492
4.1.4 Nicht-lineare Regression . . . 492
4.1.4.1 Aufgabe 4.1.4.1 Bakterien . . . 492
4.1.4.2 Aufgabe 4.1.4.2 Diät . . . 493
4.1.4.3 Aufgabe 4.1.4.3 Blutkonzentration . . . 494
4.1.5 Wahrscheinlichkeiten . . . 494
4.1.5.1 Aufgabe 4.1.5.1 Glücksspiel . . . 494
4.1.5.2 Aufgabe 4.1.5.2 Münzwürfe . . . 494
4.1.5.3 Aufgabe 4.1.5.3 Medizinschrank . . . 495
4.1.5.4 Aufgabe 4.1.5.4 Kinderkrankheiten . . . 495
4.1.5.5 Aufgabe 4.1.5.5 Schwangerschaftstest . . . 496
4.1.5.6 Aufgabe 4.1.5.6 Glückspielwahrscheinlichkeiten . . . 497
4.1.5.7 Aufgabe 4.1.5.7 Grippeimpfung . . . 497
4.1.5.8 Aufgabe 4.1.5.8 Ebola . . . 497
4.1.6 Diskrete Wahrscheinlichkeitsverteilungen . . . 498
4.1.6.1 Aufgabe 4.1.6.1 Münzwurf . . . 498
4.1.6.2 Aufgabe 4.1.6.2 Geburten pro Tag . . . 498
4.1.6.3 Aufgabe 4.1.6.3 Gesetz der seltenen Ereignisse . . . 499
4.1.6.4 Aufgabe 4.1.6.4 Münzwürfe (II) . . . 499
4.1.6.5 Aufgabe 4.1.6.5 Behandlungserfolg . . . 499
4.1.6.6 Aufgabe 4.1.6.6 Impfreaktion . . . 500
4.1.6.7 Aufgabe 4.1.6.7 Telefonanrufe . . . 500
4.1.7 Kontinuierliche Wahrscheinlichkeitsverteilungen . . . 500
4.1.7.1 Aufgabe 4.1.7.1 Bushaltestelle . . . 500
4.1.7.2 Aufgabe 4.1.7.2 Standardnormalverteilung . . . 501
4.1.7.3 Aufgabe 4.1.7.3 Chiquadratverteilungen . . . 501
4.1.7.4 Aufgabe 4.1.7.4 t-Verteilung . . . 502
4.1.7.5 Aufgabe 4.1.7.5 Fishers F-Verteilung . . . 502
4.1.7.6 Aufgabe 4.1.7.6 Blutzuckerspiegel . . . 502
4.1.7.7 Aufgabe 4.1.7.7 Cholesterinspiegel bei Männern . . . 503
4.1.8 Konfidenzintervalle (eine Stichprobe) . . . 503
4.1.8.1 Aufgabe 4.1.8.1 Wirkstoffkonzentration . . . 503
4.1.8.2 Aufgabe 4.1.8.2 Milchfett . . . 504
4.1.8.3 Aufgabe 4.1.8.3 Bibliotheksnutzung . . . 504
4.1.8.4 Aufgabe 4.1.8.4 Atemwegsprobleme und Impfung . . . 505

4.1.8.5 Aufgabe 4.1.8.5 Cholesterin . . . 505
4.1.8.6 Aufgabe 4.1.8.6 Neurologisches Syndrom . . . 506
4.1.8.7 Aufgabe 4.1.8.7 Neugeborene . . . 506
4.1.9 Konfidenzintervalle (zwei Stichproben) . . . 507
4.1.9.1 Aufgabe 4.1.9.1 Medikamentenwerbung . . . 507
4.1.9.2 Aufgabe 4.1.9.2 Milchfett . . . 507
4.1.9.3 Aufgabe 4.1.9.3 Bibliotheksnutzung nach Geschlecht . . . 508
4.1.9.4 Aufgabe 4.1.9.4 Prüfungen vormittags und nachmittags . . . 509
4.1.9.5 Aufgabe 4.1.9.5 Cholesterin und Sport . . . 509
4.1.9.6 Aufgabe 4.1.9.6 Patientenzufriedenheit . . . 509
4.1.9.7 Aufgabe 4.1.9.7 Neugeborene . . . 510
4.1.10 Signifikanztests . . . 510
4.1.10.1 Aufgabe 4.1.10.1 Wirkstoffkonzentration . . . 510
4.1.10.2 Aufgabe 4.1.10.2 Bibliotheksnutzung . . . 511
4.1.10.3 Aufgabe 4.1.10.3 Laufen lernen . . . 511
4.1.10.4 Aufgabe 4.1.10.4 Bronchialretention . . . 512
4.1.10.5 Aufgabe 4.1.10.5 Prüfungen vormittags und nachmittags . . . 513
4.1.10.6 Aufgabe 4.1.10.6 Pulsmessung . . . 513
4.1.11 Varianzanalysen (ANOVA) . . . 514
4.1.11.1 Aufgabe 4.1.11.1 Aknetherapie . . . 514
4.1.11.2 Aufgabe 4.1.11.2 Schulranking . . . 514
4.1.11.3 Aufgabe 4.1.11.3 Puls und Herzkrankheit . . . 515
4.1.11.4 Aufgabe 4.1.11.4 Kohlenmonoxid . . . 516
4.1.12 Chiquadratests für Anteilswerte . . . 516
4.1.12.1 Aufgabe 4.1.12.1 Magengeschwür . . . 516
4.1.12.2 Aufgabe 4.1.12.2 Blutgruppen . . . 517
4.1.12.3 Aufgabe 4.1.12.3 Rauchen und Geschlecht . . . 517
4.1.12.4 Aufgabe 4.1.12.4 Migräne . . . 518
4.1.12.5 Aufgabe 4.1.12.5 Komatös . . . 518
4.1.12.6 Aufgabe 4.1.12.6 Heilung . . . 519
4.1.12.7 Aufgabe 4.1.12.7 Facherfolg . . . 519
4.1.12.8 Aufgabe 4.1.12.8 Statistikdozent . . . 519
4.2 Lösungen . . . 520
4.2.1 Lösung zur Aufgabe 4.1.1.1 . . . 520
4.2.2 Lösung zur Aufgabe 4.1.1.2 . . . 522
4.2.3 Lösung zur Aufgabe 4.1.1.3 . . . 526
4.2.4 Lösung zur Aufgabe 4.1.1.4 . . . 528
4.2.5 Lösung zur Aufgabe 4.1.1.5 . . . 533
4.2.6 Lösung zur Aufgabe 4.1.1.6 . . . 536
4.2.7 Lösung zur Aufgabe 4.1.1.7 . . . 537
4.2.8 Lösung zur Aufgabe 4.1.2.1 . . . 552
4.2.9 Lösung zur Aufgabe 4.1.2.2 . . . 554
4.2.10 Lösung zur Aufgabe 4.1.2.3 . . . 555
4.2.11 Lösung zur Aufgabe 4.1.2.4 . . . 556
4.2.12 Lösung zur Aufgabe 4.1.2.5 . . . 557
4.2.13 Lösung zur Aufgabe 4.1.2.6 . . . 558

4.2.14 Lösung zur Aufgabe 4.1.2.7 . . . 559
4.2.15 Lösung zur Aufgabe 4.1.2.8 . . . 560
4.2.16 Lösung zur Aufgabe 4.1.3.1 . . . 561
4.2.17 Lösung zur Aufgabe 4.1.3.2 . . . 565
4.2.18 Lösung zur Aufgabe 4.1.3.3 . . . 570
4.2.19 Lösung zur Aufgabe 4.1.3.4 . . . 572
4.2.20 Lösung zur Aufgabe 4.1.3.5 . . . 577
4.2.21 Lösung zur Aufgabe 4.1.3.6 . . . 579
4.2.22 Lösung zur Aufgabe 4.1.3.7 . . . 581
4.2.23 Lösung zur Aufgabe 4.1.3.8 . . . 584
4.2.24 Lösung zur Aufgabe 4.1.4.1 . . . 589
4.2.25 Lösung zur Aufgabe 4.1.4.2 . . . 593
4.2.26 Lösung zur Aufgabe 4.1.4.3 . . . 601
4.2.27 Lösung zur Aufgabe 4.1.5.1 . . . 602
4.2.28 Lösung zur Aufgabe 4.1.5.2 . . . 604
4.2.29 Lösung zur Aufgabe 4.1.5.3 . . . 606
4.2.30 Lösung zur Aufgabe 4.1.5.4 . . . 606
4.2.31 Lösung zur Aufgabe 4.1.5.5 . . . 609
4.2.32 Lösung zur Aufgabe 4.1.5.6 . . . 611
4.2.33 Lösung zur Aufgabe 4.1.5.7 . . . 612
4.2.34 Lösung zur Aufgabe 4.1.5.8 . . . 613
4.2.35 Lösung zur Aufgabe 4.1.6.1 . . . 616
4.2.36 Lösung zur Aufgabe 4.1.6.2 . . . 618
4.2.37 Lösung zur Aufgabe 4.1.6.3 . . . 621
4.2.38 Lösung zur Aufgabe 4.1.6.4 . . . 623
4.2.39 Lösung zur Aufgabe 4.1.6.5 . . . 624
4.2.40 Lösung zur Aufgabe 4.1.6.6 . . . 625
4.2.41 Lösung zur Aufgabe 4.1.6.7 . . . 625
4.2.42 Lösung zur Aufgabe 4.1.7.1 . . . 626
4.2.43 Lösung zur Aufgabe 4.1.7.2 . . . 628
4.2.44 Lösung zur Aufgabe 4.1.7.3 . . . 632
4.2.45 Lösung zur Aufgabe 4.1.7.4 . . . 633
4.2.46 Lösung zur Aufgabe 4.1.7.5 . . . 634
4.2.47 Lösung zur Aufgabe 4.1.7.6 . . . 635
4.2.48 Lösung zur Aufgabe 4.1.7.7 . . . 636
4.2.49 Lösung zur Aufgabe 4.1.8.1 . . . 636
4.2.50 Lösung zur Aufgabe 4.1.8.2 . . . 639
4.2.51 Lösung zur Aufgabe 4.1.8.3 . . . 642
4.2.52 Lösung zur Aufgabe 4.1.8.4 . . . 643
4.2.53 Lösung zur Aufgabe 4.1.8.5 . . . 644
4.2.54 Lösung zur Aufgabe 4.1.8.6 . . . 646
4.2.55 Lösung zur Aufgabe 4.1.8.7 . . . 646
4.2.56 Lösung zur Aufgabe 4.1.9.1 . . . 649
4.2.57 Lösung zur Aufgabe 4.1.9.2 . . . 651
4.2.58 Lösung zur Aufgabe 4.1.9.3 . . . 653
4.2.59 Lösung zur Aufgabe 4.1.9.4 . . . 654

4.2.60 Lösung zur Aufgabe 4.1.9.5 . . . 654
4.2.61 Lösung zur Aufgabe 4.1.9.6 . . . 656
4.2.62 Lösung zur Aufgabe 4.1.9.7 . . . 657
4.2.63 Lösung zur Aufgabe 4.1.10.1 . . . 660
4.2.64 Lösung zur Aufgabe 4.1.10.2 . . . 664
4.2.65 Lösung zur Aufgabe 4.1.10.3 . . . 664
4.2.66 Lösung zur Aufgabe 4.1.10.4 . . . 665
4.2.67 Lösung zur Aufgabe 4.1.10.5 . . . 666
4.2.68 Lösung zur Aufgabe 4.1.10.6 . . . 667
4.2.69 Lösung zur Aufgabe 4.1.11.1 . . . 671
4.2.70 Lösung zur Aufgabe 4.1.11.2 . . . 674
4.2.71 Lösung zur Aufgabe 4.1.11.3 . . . 676
4.2.72 Lösung zur Aufgabe 4.1.11.4 . . . 677
4.2.73 Lösung zur Aufgabe 4.1.12.1 . . . 678
4.2.74 Lösung zur Aufgabe 4.1.12.2 . . . 679
4.2.75 Lösung zur Aufgabe 4.1.12.3 . . . 680
4.2.76 Lösung zur Aufgabe 4.1.12.4 . . . 682
4.2.77 Lösung zur Aufgabe 4.1.12.5 . . . 683
4.2.78 Lösung zur Aufgabe 4.1.12.6 . . . 684
4.2.79 Lösung zur Aufgabe 4.1.12.7 . . . 684
4.2.80 Lösung zur Aufgabe 4.1.12.8 . . . 685

Literaturverzeichnis **687**

Stichwortverzeichnis **688**

Verzeichnis der verwendeten R-Befehle **691**

1 Einleitung

Willkommen zu diesem Nachschlagewerk.

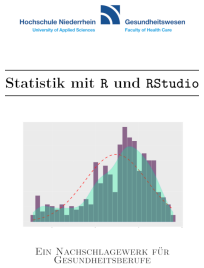
Dieses Buch widmet sich an Studierende und Beschäftigte im Gesundheitswesen und soll als Nachschlagewerk für statistische Fragestellungen bezüglich der freien Software R dienen. Es ist **nicht** dafür geeignet, Einsteigern Statistik zu erklären (hierfür ist das Buch von Grabinger [1] sehr gut geeignet), sondern es richtet sich an jene, die ihre statistischen Auswertungen mit R fahren möchten. Das heisst, Sie müssen schon wissen was Sie eigentlich tun wollen. Dieses Buch möchte Ihnen zeigen, wie Sie Ihre Vorhaben in R umsetzen können.

Dieses Buch ist in vier Teile gegliedert:

1. Im ersten Teil (ab Seite 2) geben wir eine kurze Einführung in R und `RStudio`.
2. Im zweiten Teil (ab Seite 211) betreiben wir Statistik mit R.
3. Der dritte Teil (ab Seite 415) zeigt konkrete Anwedungsbeispiele. Hierbei folgen wir der Ordnung "Was willst du machen?" und zeigen an "echten" Fragen, wie wir das erlernte R-KungFu in der Praxis anwenden können.
4. Im vierten Teil (ab Seite 480) kann das Erlernte anhand von Übungsaufgaben überprüft werden.

Eine aktuelle epub-Version dieses Buches finden Sie unter:
https://www.produnis.de/R/rbuch.epub
Eine aktuelle PDF-Version dieses Buches finden Sie unter:
https://www.produnis.de/R/rbuch.pdf
Dieses Buch steht zudem als Webseite bereit unter:
https://www.produnis.de/R/

Kritik und Diskussion sind per Mastodon möglich:
https://mastodon.social/@rbuch



1.1 Installation von R und RStudio



ist Open Source Software und kann daher weltweit kostenfrei auf vielen Betriebssystemen, z.B. Windows, Apple und Linux, installiert werden. Gehen Sie hierzu auf die Webseite des Projektes unter https://r-project.org und laden sich das passende Installationspaket für Ihr Betriebssystemen herunter und führen Sie die Installation durch.
Für Windows lautet der Link auf das Installationspaket
https://cran.r-project.org/bin/windows/base/

Für MacOSX lautet der Link
https://cran.r-project.org/bin/macosx/

Bei fast allen Linuxdistributionen liegt R im Repository, so dass es z.B. unter Ubuntu mit `apt-get install r-base` installiert werden kann.

Von RStudio steht ebenfalls eine quelloffene (sowie eine kommerzielle) Version für diese Systeme zur Verfügung. Gehen Sie auch hier auf die Projektseite https://rstudio.com/products/rstudio/download/ und laden sich die kostenfreie Variante für Ihr Betriebssystemen herunter. Sie können RStudio erst installieren, wenn R bereits installiert ist.

Falls Sie nicht wissen, wie sich Software auf Ihrem PC installieren lässt, empfehlen wir dieses Youtube-Video
https://www.youtube.com/watch?v=X_Mxya2Fis0
Es zeigt den gesamten Installtionsprozess von und RStudio am Beispiel "Windows" (weitere Youtube-Videos sammeln wir in Abschnitt 1.8.2 auf Seite 210).

1.2 Kurze Einführung in R und RStudio

1.2.1 R

R ist eine Programmiersprache für Statistik. Mit ihr lassen sich statistische Berechnung durchführen und graphisch darstellen. Die klassische R -Software läuft in einem Terminalfenster (Abbildung 1.1) über welches die Befehle abgesetzt werden.

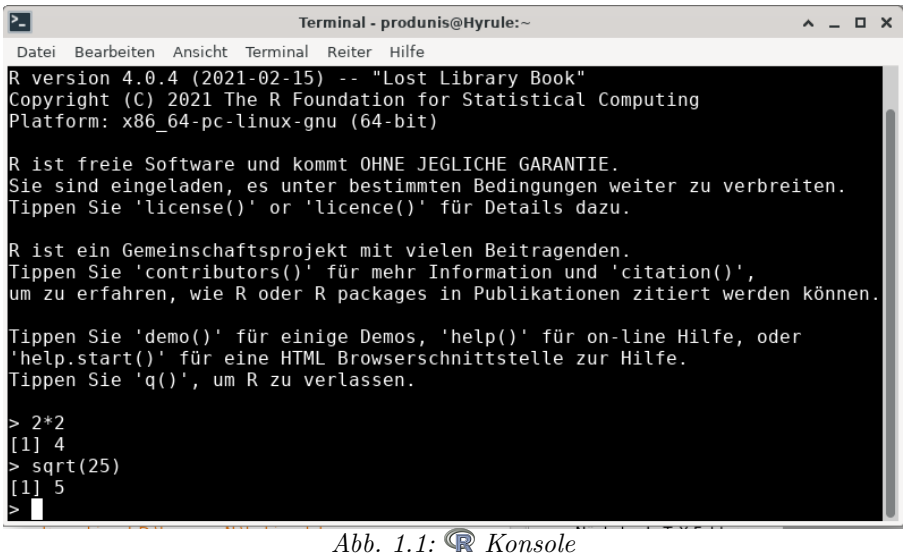


Abb. 1.1: R Konsole

Dies kann am Anfang abschreckend wirken, da wir es eher gewöhnt sind, Programme über eine graphische Nutzeroberfläche zu steuern.
Seit 2012 wird dafür `RStudio` entwickelt, welches eine funktionsreiche Entwicklerumgebung für R bereitstellt. Mittlerweile hat sich `RStudio` als Open Source Standardanwendung für statistische Analysen durchgesetzt.

Sie werden nicht drum herumkommen, Ihre Befehle weiterhin aufzuschreiben und auszuführen, aber `RStudio` bietet Ihnen eine sehr hilfreiche und komfortable Arbeitsumgebung, die viele Arbeitsschritte erleichtern wird.

Auch in R hat sich einiges getan, vor allem in den letzten Jahren. Mit der Paketesammlung *Tidyverse* (siehe Kapitel 1.6 auf Seite 141) und dessen strenger Umsetzung der *Tidy Data* sowie kosequenter Nutzung der *Pipe* (siehe Abschnitt 1.6.2) hat sich ein eigenständiger Dialekt entwickelt, der immer mehr Anhänger findet. Die grundlegenden Schritte der Datenmanipulation und graphischen Darstellung sind im *Tidyverse* viel einfacher als mit klassischen R-Hausmitteln. Die Funktionalität der Pipe wurde mit Version 4.1 ins klassische R übernommen (siehe Abschnitt 1.2.3.9).

In diesem Buch lernen Sie zuerst das "klassische" R, denn die grundlegenden Konzepte sind immer noch gültig, und auch die Funktionen für statistische Analysen sind immer noch die selben. Die Abschnitte zum Umgang mit Datensätzen sowie zur `plot()`-Funktion sind aber nur rudimentär gehalten, da dies mit den modernen Paketen `dplyr` und `ggplot2` wesentlich einfacher funktioniert.

1.2.2 RStudio

Die Nutzeroberfläche von RStudio besteht aus vier Bereichen (siehe Abbildung 1.2), die nach ihrer hauptsächlichen Funktion benannte sind. Jeder Bereich bietet aber über seine Reiter weitere Anwendungsmöglichkeiten (die Sie jetzt evtl. noch nicht verstehen).

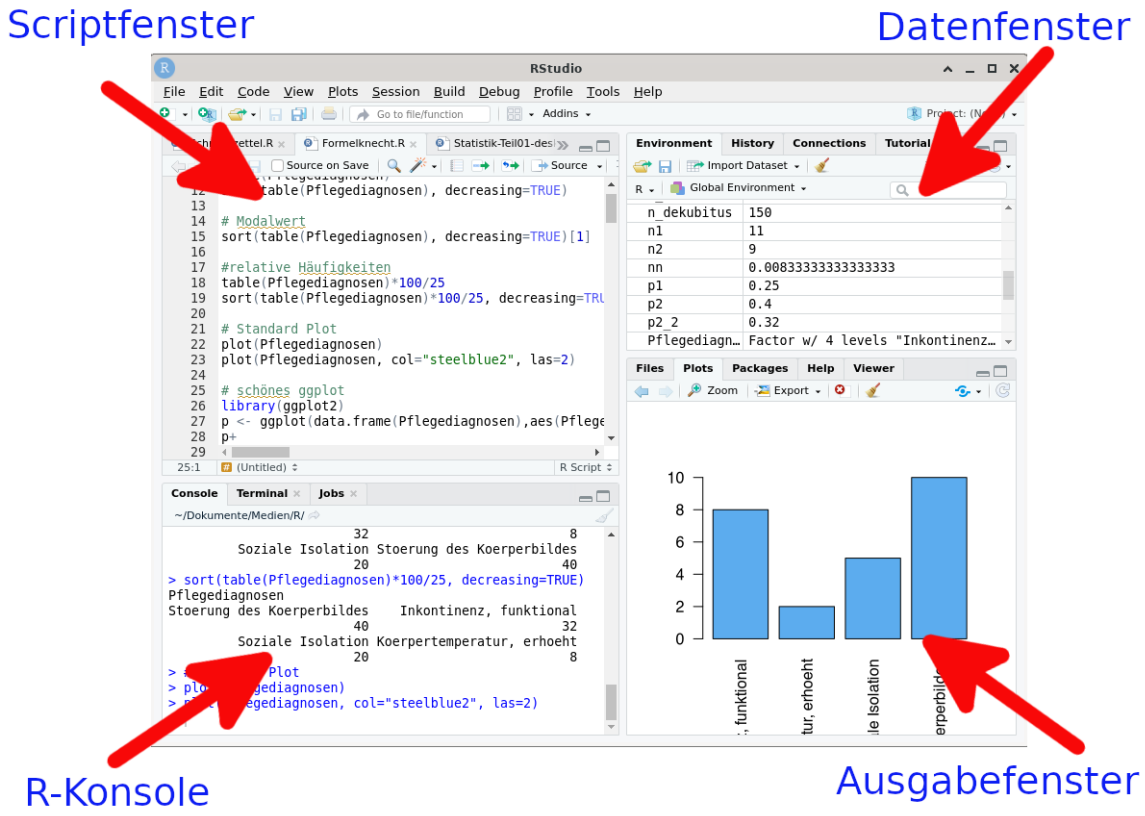


Abb. 1.2: RStudio Bereiche

Oben links befindet sich das "Scriptfenster". In diesem Bereich werden Ihre Scriptdateien angezeigt und Sie können sie dort editieren. In den Scriptdateien schreiben Sie Ihre R-Befehle nacheinander auf, je nachdem, welche Auswertungen oder Datentransformationen Sie fahren, oder welche Diagramme Sie plotten möchten (siehe Kapitel 2). Ihr Code wird dabei von RStudio farblich aufgehübscht, was ihn wesentlich übersichtlicher macht.

Rechts daneben ist das "Datenfenster". Hier werden alle Variablen und Datenframes angezeigt, die derzeit im Speicher geladen sind. Durch Doppelklick auf einen Datensatz werden die Daten im "Scriptfenster" angezeigt.Über den Reiter History können Sie alle Befehle, die Sie in der R-Konsole abgesetzt haben, einsehen. Über den Reiter Tutorials haben Sie Zugriff auf zahlreiche Anleitungsangebote zu RStudio.

Unten links befindet sich die R-Konsole. Dieser Bereich entspricht der "normalen" R-Installation. Hier können Sie von Hand Befehle eingeben, oder aber Befehle aus dem Scriptfenster ausführen lassen. Die Konsole unterscheidet farblich zwischen Eingabe und Ausgabe, was sehr zur Übersicht beiträgt. Der Reiter Terminal öffnet eine Systemkonsole (unter Linux sehr hilfreich), und unter Jobs werden z.B. Installationsprozesse von Zusatzpaketen dargestellt.

Unten rechts ist das "Ausgabefenster". Hier werden unter anderem Plots und Hilfetexte angezeigt. Das Fenster verfügt zudem über einen Dateibrowser (Reiter `Files`), von dem aus Sie bequem Dateien oder Datensätze aus Ihrem Arbeitsvereichnis laden können. Über den Reiter `Packages` können Sie Zusatzpakete aktivieren und deaktivieren.

Für die ersten Schritte in `RStudio` empfehlen wir folgendes Youtube-Video:
https://www.youtube.com/watch?v=tyvEHQszZJs

1.2.2.1 RStudio anpassen

`RStudio` lässt eine Menge an Anpassungen zu, so dass Sie die Software möglichst gut an Ihren Workflow anpassen können. Folgende zwei Einstellungen empfehlen wir nach der Installation vorzunehmen.

- **Arbeitsverzeichnis setzen**: Unter `Tools` → `Global Options` gelangen Sie ins allgemeine Einstellungsmenü. Unter dem Reiter `General` → `Basic` lässt sich das Arbeitsverzeichnis (working directory) festlegen (Abbildung 1.3). In dieses Verzeichnis legt R standardmäßig alle Diagramme, Scripte sowie den aktuellen Workspace (also alle Variablen und Datensätze, die derzeit geladen sind).
  Es bietet sich an, ein eigenes Verzeichnis für R zu erstellen und hier in `RStudio` auszuwählen.

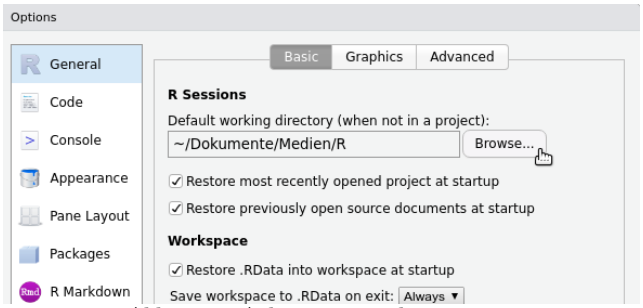


Abb. 1.3: Arbeitsverzeichnis setzen

- **Tastaturkürzel**: Unter `Tools` → `Modify Keayboard Shortcuts` öffnet sich das Dialogfenster zum Festlegen der Tastaturkürzel. Hier geben wir im Feld `Filter` "exec" ein (Abbildung 1.4).
  Bei der Aktion `Execute Current Line` setzen wir das Tastaturkürzel `[STRG]+[R]` (Abbildung 1.5). Die Einstellung bewirkt, dass im Scriptfenster die aktuelle Zeile per `[STRG]+[R]` ausgeführt wird. Dies ist ziemlich hilfreich.

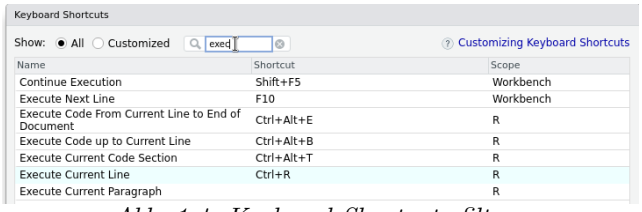


Abb. 1.4: Keyboard Shortcuts filtern

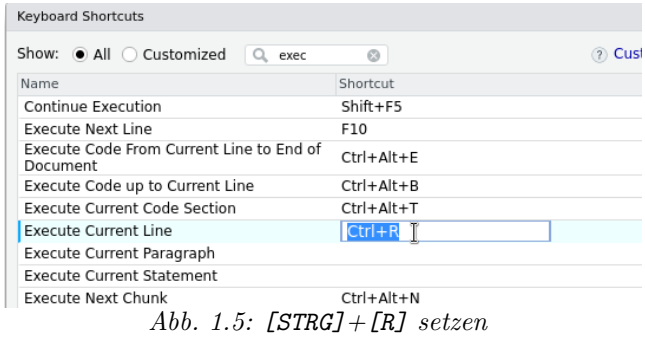


Abb. 1.5: [STRG]+[R] setzen

An dieser Stelle sei darauf hingewiesen, dass das Aussehen von `RStudio` angepasst werden kann. Vielleicht möchten Sie auf einen "augenfreundlicheren" Modus umschalten, in welchem der Hintergrund dunkler ist. Klicken Sie hierfür in der Menüleiste auf `Tools` → `Global Options` klicken.
Wählen Sie hier den Bereich `Appearance` aus (Abbildung 1.6).
Es werden Ihnen zahlreiche "Themes" zur Auswahl gestellt. Probieren Sie alle einmal aus. Als dunkles Theme verwende ich persönlich gerne `Dracula` (Abbildung 1.7).

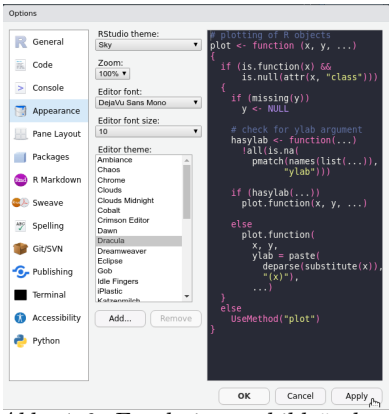


Abb. 1.6: Erscheinungsbild ändern

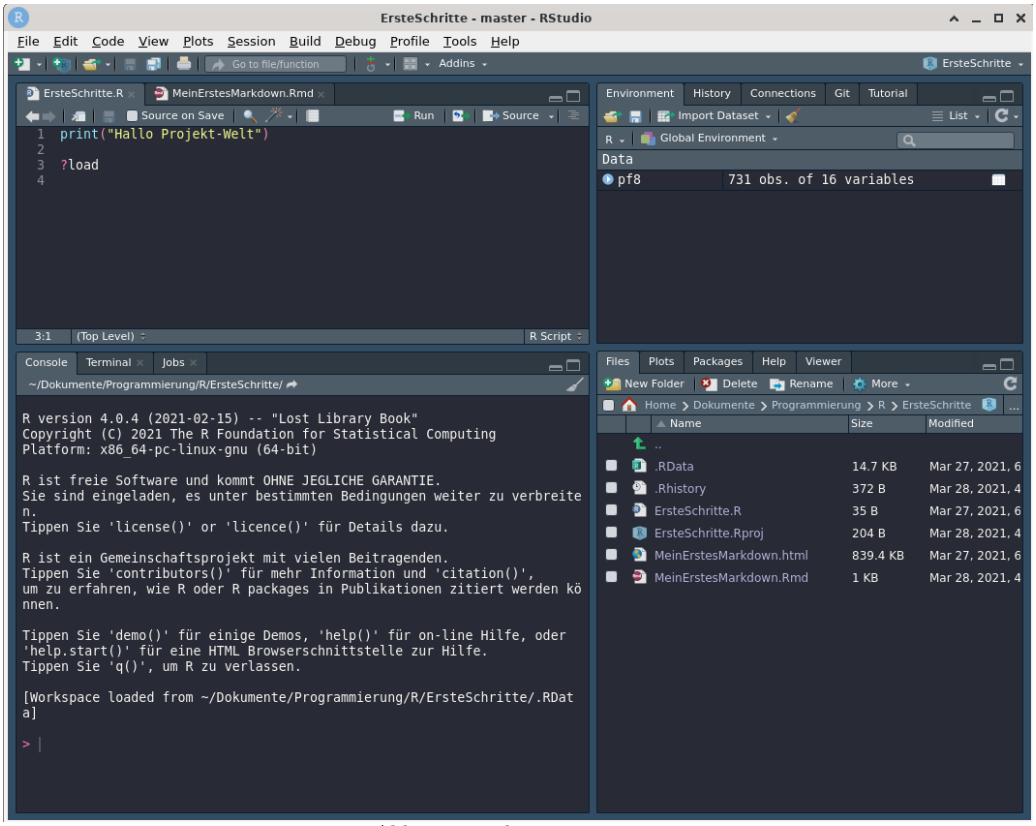


Abb. 1.7: Theme `Dracula`

1.2.2.2 Eine Sitzung starten

Starten Sie `RStudio`. Es begrüßt Sie die `RStudio` Arbeitsfläche. Das Scriptfenster oben links ist noch geschlossen, so dass der gesamte linke Bereich von der -Konsole eingenommen wird (Abb. 1.8).

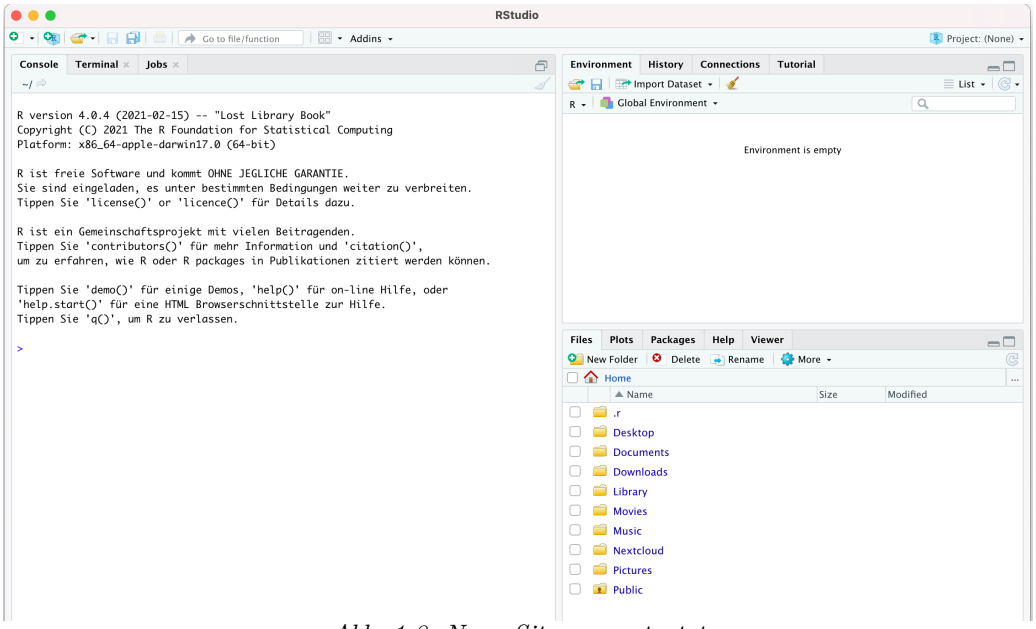


Abb. 1.8: Neue Sitzung gestartet

Als erstes legen Sie eine neue Scriptdatei an. Klicken Sie oben links auf das grüne + Zeichen und wählen Sie `R Script` (Abbildung 1.9).

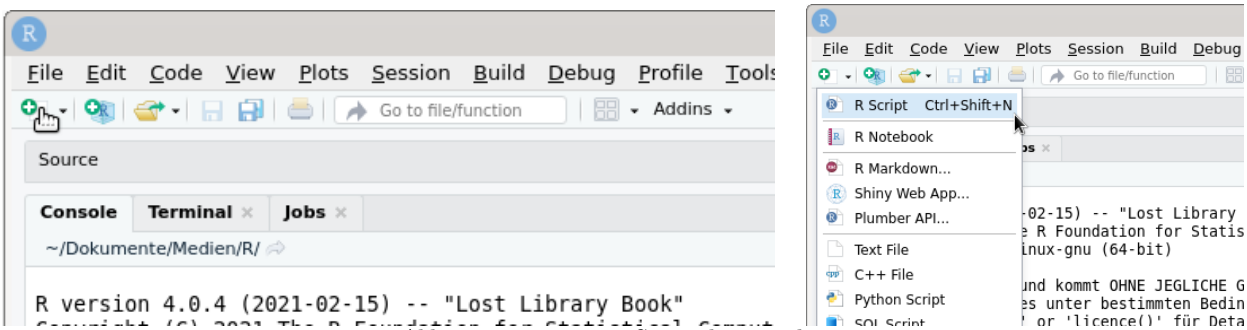


Abb. 1.9: eine neue Scriptdatei anlegen

Wie Abbildung 1.9 zeigt, können Sie auch die Tastenkombination `[STRG]` + `[SHIFT]` + `[N]` verwenden, um eine neue Scriptdatei zu erzeugen.

In dieser Scriptdatei werden wir alle Befehle hinterlegen. Für gewöhnlich beginnt man bei Programmierübungen mit einem "Hallo Welt" Beispiel. Schreiben Sie in die Scriptdatei den Befehl:

```
1  print("Hallo Welt")
```

Diesen Befehl wollen wir nun in der -Konsole ausführen. Das kann auf mehrere Arten geschehen:

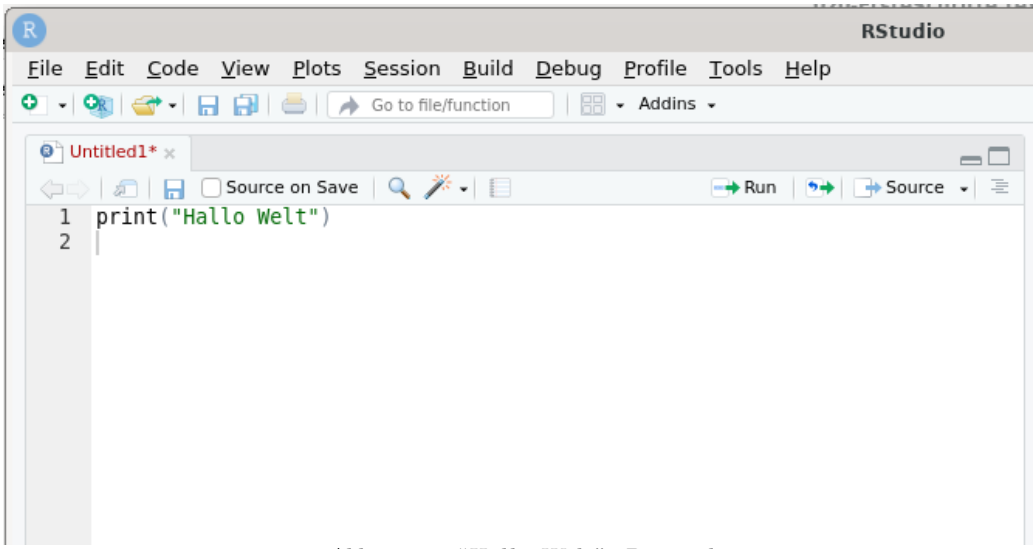


Abb. 1.10: "Hallo Welt" -Beispiel

- gehen Sie mit dem Cursor in die Zeile des Befehls. Drücken Sie die (auf Seite 5 angelegte) Tastenkombination `[STRG] + [R]`.
- oder gehen Sie mit dem Cursor in die Zeile des Befehls und klicken Sie auf den →`Run`-Knopf (siehe Abbildung 1.11).

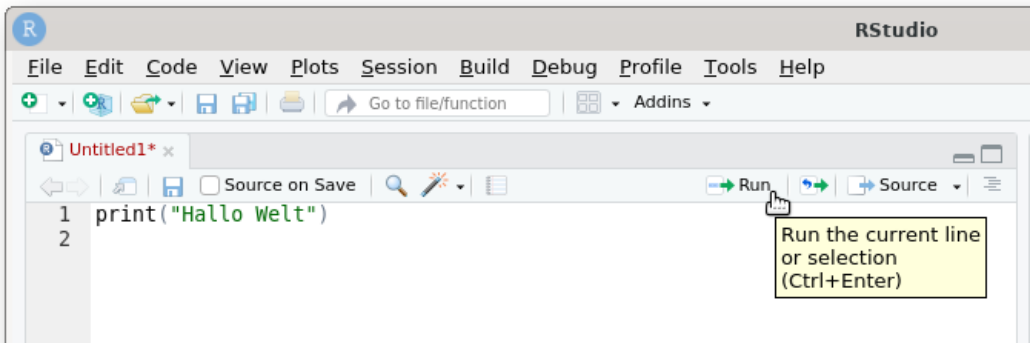


Abb. 1.11: `Run`-Knopf

- oder gehen Sie mit dem Cursor in die Zeile des Befehls und drücken Sie die (voreingestellte) Tastenkombination `[STRG] + [ENTER]`.

Alle drei Aktionen bewirken, dass der Befehl `print("Hallo Welt")` in der R -Konsole unten links ausgeführt wird (siehe Abbildung 1.12).

Und R antwortet entsprechend in der Konsole mit

```
[1] "Hallo Welt"
```

Sie können die Befehle auch jederzeit direkt unten in die R-Konsole eingeben und ausführen.

Prinzipiell sollten Sie aber alles in der Scriptdatei aufschreiben und von dort ausführen. So können Sie leicht Ihre Befehle anpassen oder auch komplizierte R-Sprüche nach und nach

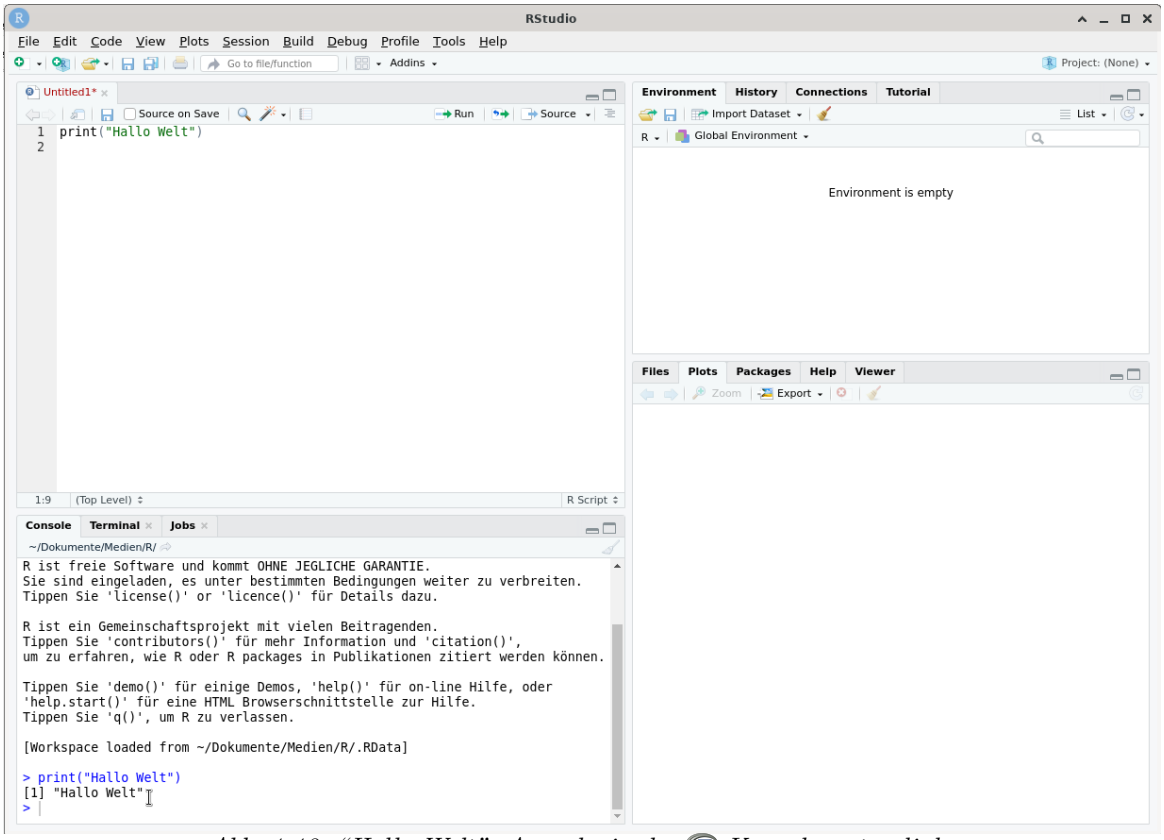


Abb. 1.12: "Hallo Welt" -Ausgabe in der R-Konsole unten links

aufbauen. Manchmal kann es aber auch nützlich sein, ein kurzes `?print` (dazu später mehr) direkt unten in der Konsole einzugeben. Sie werden "den Dreh" sehr schnell heraus haben.

Wir sollten so häufig wie möglich unsere Scriptdatei abspeichern. Hierzu drücken Sie die Tastenkombination `[STRG]` + `[S]`, oder Sie klicken auf das Diskettensymbol (siehe Abbildung 1.13).

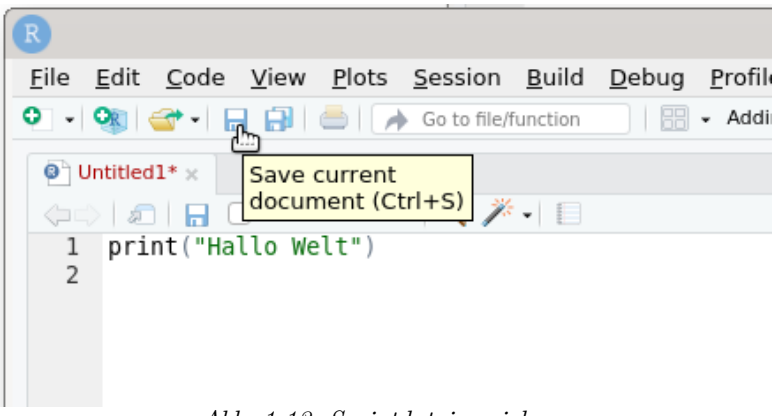


Abb. 1.13: Scriptdatei speichern

Derzeit heisst unsere Scriptdatei noch `Untitled1`. Geben Sie Ihrer Scriptdatei einen schönen Dateinamen mit der Dateiendung `.R`, also z.B. `ErsteSchritte.R`
Der neue Dateiname wird nach dem Speichern über dem Script angezeigt (Abbildung 1.14).

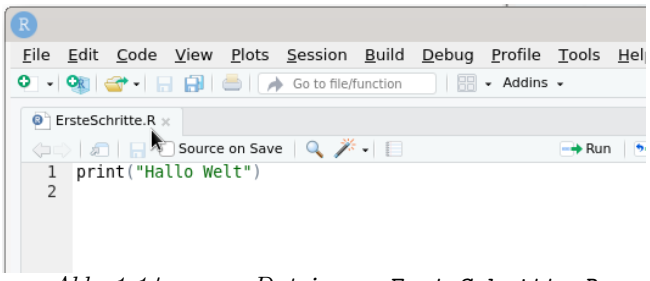


Abb. 1.14: neuer Dateiname `ErsteSchritte.R`

Mit dem `#`-Zeichen können Zeilen *auskommentiert* werden. Das bedeutet, dass alles, was hinter einem `#` steht, von R ignoriert wird (siehe Abbildung 1.15).

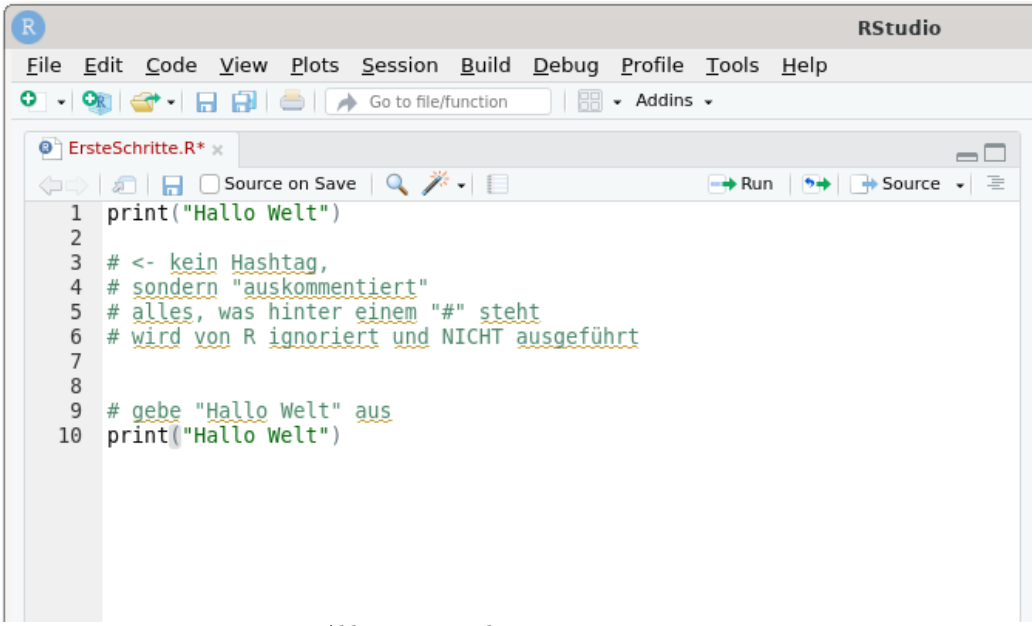


Abb. 1.15: auskommentieren mit `#`

`RStudio` hebt auskommentierte Stelle farblich hervor (Abbildung 1.15, alles Auskommentierte ist hellgrün dargestellt), so dass sie leicht erkannt werden können.
Das Auskommentieren mit `#` ist hilfreich, um den Code zu *kommentieren*, also zu beschreiben, was die nachfolgenden Zeilen machen sollen. So bleibt Ihr Code verständlich und wird auch für andere nachvollziehbar. Zum anderen kann es hilfreich sein, einzelne Befehlszeilen mittels `#` zu "dekativieren", z.B. wenn man auf Fehlersuche ist.

In Abbildung 1.15 sehen Sie auch, dass der Dateiname der Scriptdatei `ErsteSchritte.R*` in rot dargestellt ist und mit einem `*` endet. Auf diese Weise teilt `RStudio` Ihnen mit, dass es ungespeicherte Änderungen an der Scriptdatei gibt, die Sie evtl. speichern möchten. Das verwundert nicht, denn wir haben ja gerade jede Menge Kommentare in unser Script geschrieben. Mit der Tastenkombination `[STRG]` + `[S]` speichern wir die Scriptdatei. Wie Abbildung 1.16 zeigt, ist der Dateiname nun nicht mehr rot und hat auch kein `*` mehr.

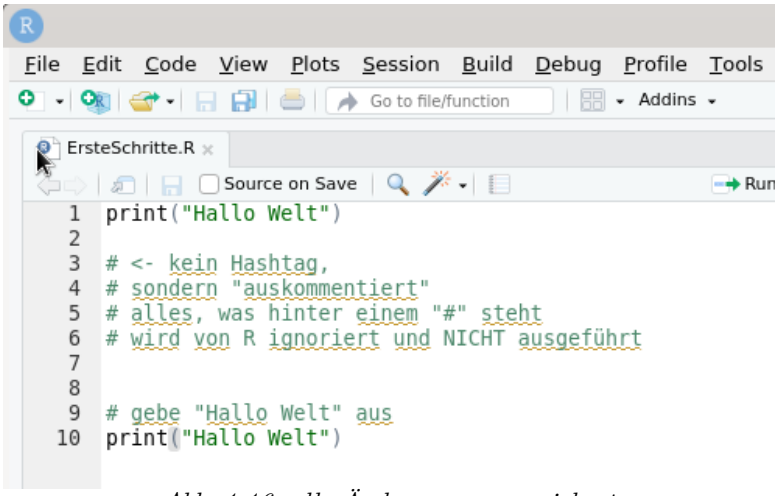


Abb. 1.16: alle Änderungen gespeichert

1.2.2.3 Projekte in RStudio

RStudio erlaubt es Ihnen, so genannte "Projekte" zu erstellen.
Ein neues Projekt startet mit einer leeren R-Konsole und einem (fast) leeren Arbeitsverzeichnis. Sie können wie gewohnt Daten laden, bearbeiten und Ihre Auswertungen fahren. Wenn Sie RStudio schließen, wird Ihr Arbeitsstand im Projektordner gespeichert. Beim nächsten Öffnen steht dann alles wieder so zur Verfügung, wie Sie es verlassen haben.

Es empfiehlt sich sehr, für jedes "Projekt", das Sie mit R durchführen möchten, ein eigenes RStudio-Projekt zu erstellen. Dies erleichtert die Arbeit ungemein. Da jedes Projekt in seinem eigenen Dateiordner "lebt", behalten Sie die Übersicht über Daten, Dateien und Diagramme. Sie können zwischen den Projekten wechseln, ohne dass Ihre Arbeitsspeicher durcheinandergeraten, oder Variablen überschrieben werden. Sofern Sie konsequent alle Dateien in den jeweiligen Ordnern gespeichert lassen, können Sie sich lästige Dateipfadangaben sparen, und Sie können den Projektordner einfach mit Kolleginnen und Kollegen teilen (indem Sie ihn kopieren), und in *deren* RStudio wird alles so funktionieren wie bei Ihnen.

Ich persönlich habe in meinem Dokumenteordner einen Unterordner R, in welchem sich alle Projekte befinden. Es ist aber natürlich auch möglich, Ihre einzelnen Projektordner an beliebigen Stellen des Dateisystems zu platzieren. Projektordner können auch "im Nachhinein" beliebig verschoben werden.

Legen Sie nun ein neues Projekt in RStudio an. Klicken Sie in der Menüleiste auf File → New Project....
Es öffnet sich ein neus Fenster, in welchem Sie gefragt werden, ob Sie einen neuen Projektordner im Dateisystem anlegen möchten, oder ob Sie einen bestehenden Ordner nutzen möchten. Da Sie (wahrscheilich) noch keine R-Projekte durgeführt haben, klicken Sie auf New Directory (Abbildung 1.17).

In RStudio können Sie viel machen, deshalb ist die Auswahlliste des Project Type so lang. Wählen Sie den ersten Eintrag New Project aus. Geben Sie dem Projekt einen Namen. In Abbildung 1.17 haben ich "ErsteSchritte" genannt. Darunter wird der Dateipfad angegeben.

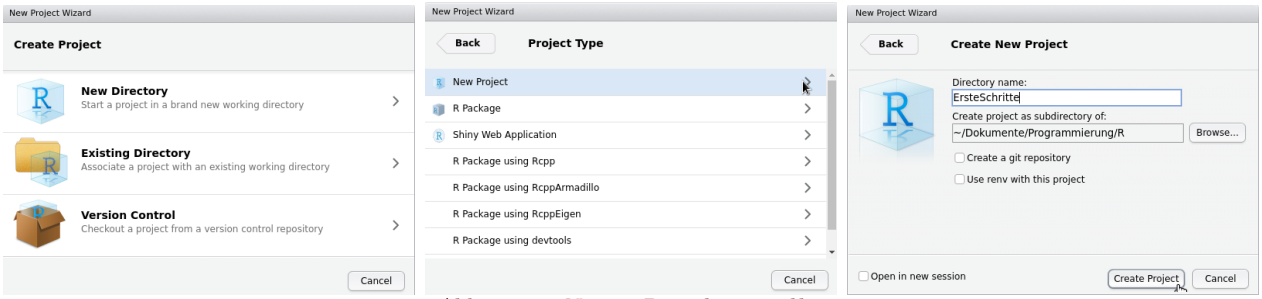


Abb. 1.17: Neues Projekt erstellen

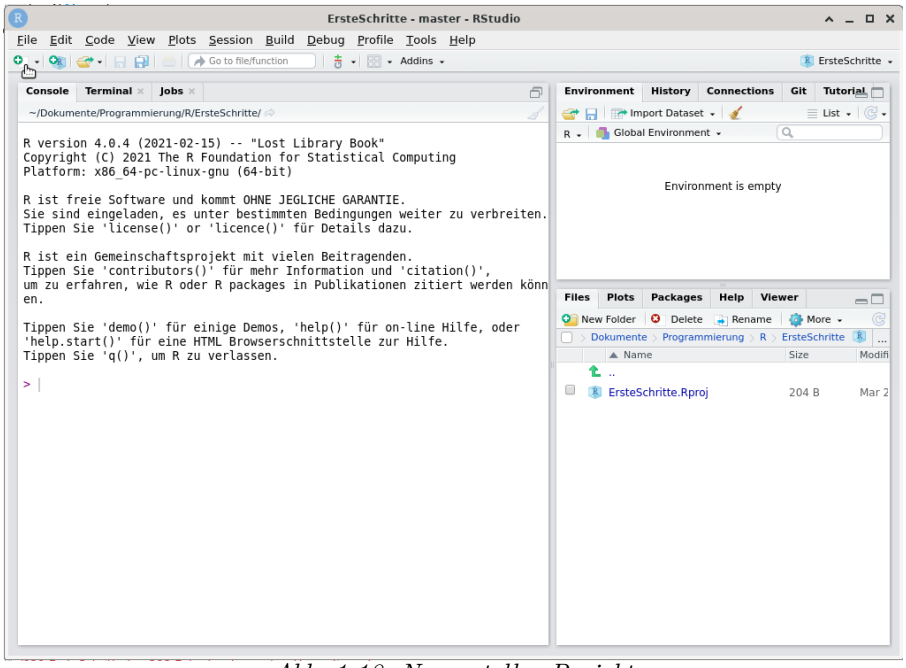


Abb. 1.18: Neuerstelles Projekt

Ignorieren Sie die weitern Häkchenoptionen und klicken Sie unten auf den `Create Project`-Knopf.

Es begrüßt Sie ein "frisches" `RStudio`-Fenster (Abbildung 1.18). Wiederholen Sie nun die Schritte aus dem "Hallo Welt"-Beispiel. Das heisst, legen Sie eine neue Scriptdatei an, indem Sie auf das grüne `+` klicken (siehe Mauszeiger in Abbildung 1.18), und schreiben Sie den Befehl hinein:

```
1 print("Hallo Projekt-Welt")
```

Führen Sie den Befehl mittel `Run`-Knopf oder der Tastenkombination `[STRG]+[R]` aus, und speichern Sie Ihre Scriptdatei anschließend unter dem Namen `ErsteSchritte.R` ab, indem Sie auf das Diskettensymbol klicken oder `[STRG]+[S]` drücken.

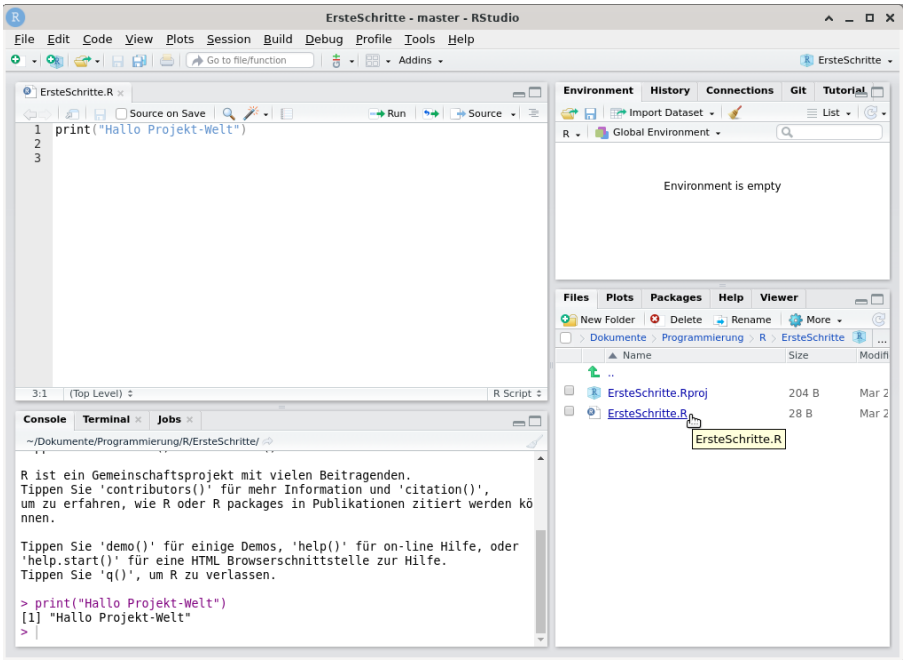


Abb. 1.19: Neues Projekt mit Scriptdatei

Wie Sie sehen, wird Ihre Scriptdatei nun im Ausgabefenster angezeigt (Abbildung 1.19). Daneben sehen Sie die Projektdatei `ErsteSchritte.Rproj`. Wenn Sie aus Ihrem Dateibrowser auf diese Datei klicken, öffnet sich automatisch `RStudio` und es wird der Arbeitsstand des Projektes geladen.

Rechts über dem Datenfenster wird Ihr derzeit aktives Projekt angezeigt (in Abbildung 1.20 `ErsteSchritte`).

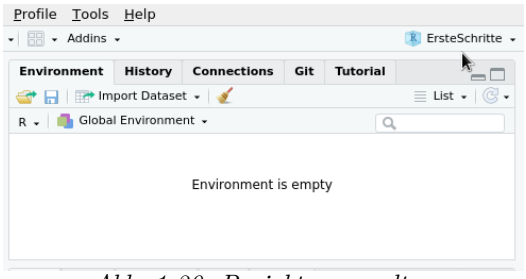


Abb. 1.20: Projekte verwalten

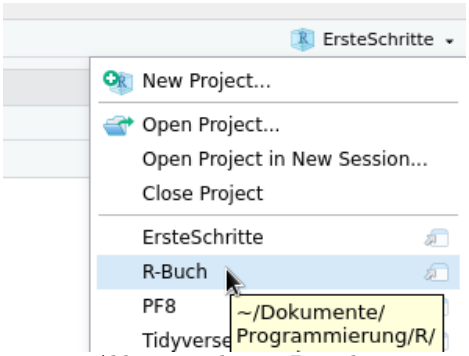


Abb. 1.21: letzte Projekte

Wenn Sie hier klicken, können Sie neue Projekte anlegen, und ihre zuletzt geöffneten Projekte werden Ihnen angezeigt. So können Sie bequem zwischen Ihren Projekten wechseln (Abbildung 1.21).

1.2.3 Grundlegende Befehle in R

Zu jedem R -Befehl lässt sich eine Hilfeseite anzeigen. Diese gibt grundlegende Informationen über Funktionsweisen des Befehls sowie die zugelassenen Befehlsoptionen. Die meisten Hilfeseiten enden mit ein paar Beispielen. Die Hilfeseiten erreicht man mit dem Befehl `help()` oder kurz `?`, wobei der Befehl, dessen Hilfsseite wir aufrufen möchten, innerhalb der Befehlsklammern stehen muss.

```
# gib Hilfeseite für Funktion "mean" aus
help(mean)

# oder einfach
?mean
```

Wenn Sie den Befehl `help(mean)` z.B. mit `[STRG]` + `[R]` ausführen, oder direkt unten in die R-Konsole eingeben, zeigt `RStudio` die Hilfeseiten im unteren rechten Viertel an (siehe Abbildung 1.22).

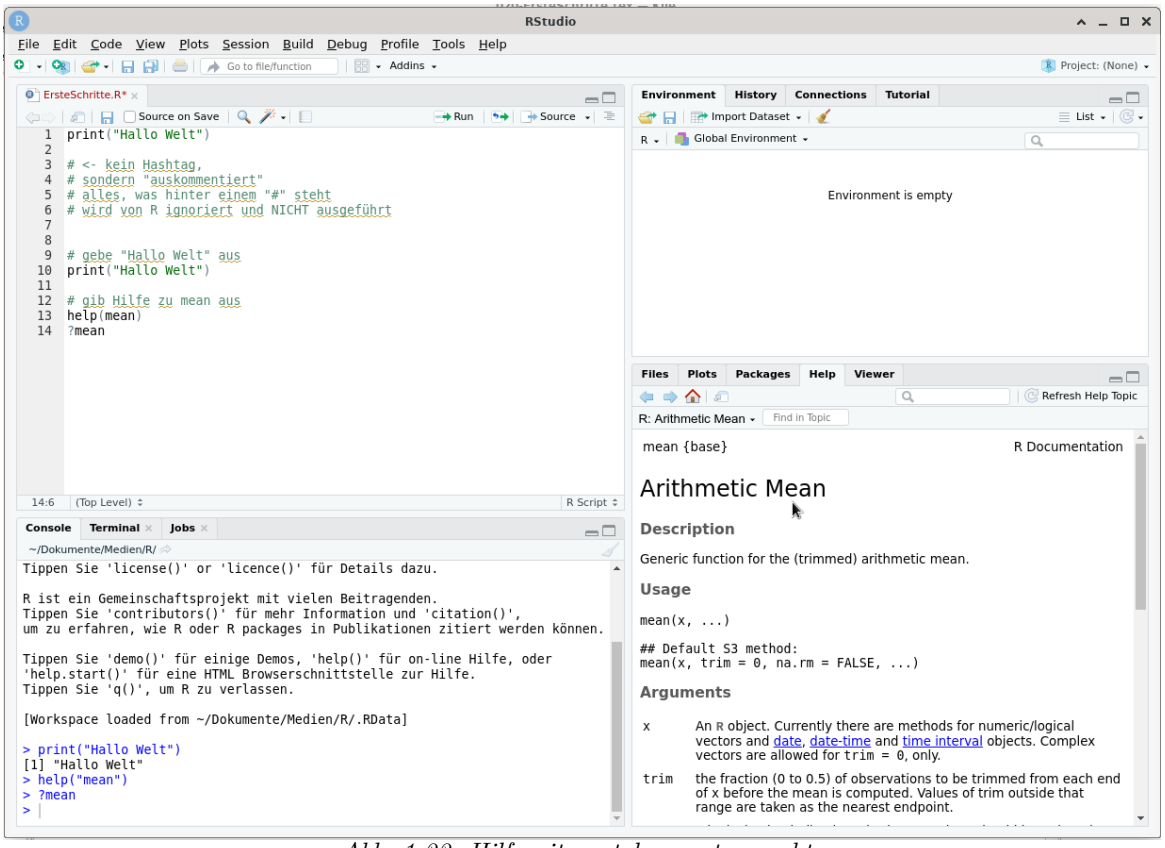


Abb. 1.22: Hilfeseiten stehen unten rechts

In diesem Buch werden die R Befehle nie vollständig mit all ihren Optionen und Einstellungsmöglichkeiten vorgestellt. Daher bietet es sich an, bei jedem "neuen" Befehl, den man lernt oder verwendet, per `help()` oder `?` die Anleitungsseite zumindest zu überfliegen.

```
# bei help() steht die Funktion in Klammern
# gib Hilfeseite für Funktion "c()" aus
```

```r
help(c)

# Bei ? wird sie ohne Klammern direkt drangeschrieben
?c
?mean
?median
```

Mit `quit()` oder kurz `q()` wird die R -Sitzung beendet.

```r
# Beende die R-Sitzung
quit()

# Beende die R-Sitzung
q()

# Rufe Hilfeseite auf
?quit
```

Wenn Sie den Befehl `quit()` in der R -Konsole von `RStudio` ausführen, wird auch `RStudio` beendet.

1.2.3.1 Rechnen mit R

R funktioniert unter anderem wie ein mächtiger Taschenrechner. Folgende grundlegende mathematische Operationen können mit R ausgeführt werden:

Operation	R-Symbol
Addition	+
Subtraktion	−
Multiplikation	*
Division	/
Potenzieren	^oder **
Wurzel	sqrt(x)

Hier ein paar Beispiele:

```r
# Addition
2+8
```

```
[1] 10
```

```r
# Multiplikation
4*4
```

```
[1] 16
```

```r
# Subtraktion
9-7
```

```
[1] 2
```

```r
# Division
6/2
```

```
[1] 3
```

```r
# Exponent
2^3
```

```
[1] 8
```

```r
# Quadratwurzel
sqrt(25)
```

```
[1] 5
```

```r
# Exponent anders notiert
2**3
```

```
[1] 8
```

```r
# Lange Formel
sqrt( (2*10) / (20/2) ) - 9
```

```
[1] -7.585786
```

```r
# Befehl schlägt fehl!
2,3 - 9
```

```
Fehler: Unerwartete(s) ',' in "2,"
```

Achtung, Anfängerfehler!

An der letzten Ausgabe zeigt sich ein wesentliches Merkmal von R:

Dezimalstellen werden mit einem Punkt UND NICHT mit einem Komma dargestellt.

Dies ist ein häufiger Anfängerfehler und meist der Grund, warum am Anfang etwas nicht funktioniert.

```r
# Dezimalstellen mit .
4.25 * 6.25
```

```
[1] 26.5625
```

Vielleicht haben Sie sich schon gefragt, wieso vor jeder Ausgabe eine `[1]` steht. In R beginnt jede Ausgabezeile mit der Position des Wertes, der als nächstes ausgegeben wird. Da wir derzeit nur Rechenoperationen durchführen, die genau ein Ergebnis haben, ist die Position unseres Ergebniswertes immer `[1]`. Hat man längere Datenreihen, z.B. das Gewicht von 10 Probanden in kg, dann könnte die Ausgabe so aussehen:

```
[1] 102.20  76.83  98.50  86.55  121.93
[6]  89.21 123.22  78.23  98.33   83.37
```

Die zweite Ausgabezeile beginnt mit einer `[6]`, was bedeutet, dass nachfolgend die 6. Position der Wertereihe angezeigt wird.

Das Ziehen einer Wurzel erfolgt über die Funktion `sqrt()`. Damit Funktionen arbeiten können, benötigen sie ein "Argument". Das Argument ist in unserem Fall die Zahl 25, da wir die Wurzel aus 25 ziehen wollen.

```r
# Wurzel ziehen
sqrt(25)
```

```
[1] 5
```

Argumente werden der Funktion in runden Klammern übergeben. So erklärt sich der Befehl `sqrt(25)`.

Weitere mathematische Funktionen sind:

Funktion	Befehl	Beispiel	Ausgabe
Exponentialfunktion	`exp()`	`exp(1)`	`[1] 2.718282`
natürlicher Logarithmus	`log()`	`log(2)`	`[1] 0.6931472`
Logarithmus zur Basis 2	`log2()`	`log2(64)`	`[1] 6`
Logarithmus zur Basis 10	`log10()`	`log10(1000)`	`[1] 3`
Logarithmus zur Basis n	`log(x, n)`	`log(343, 7)`	`[1] 3`
Sinus	`sin()`	`sin(2)`	`[1] 0.9092974`
Kosinus	`cos()`	`cos(2)`	`[1] -0.4161468`
Tangens	`tan()`	`tan(2)`	`[1] -2.18504`
Absolutwert	`abs()`	`abs(-7)`	`[1] 7`

Nachkommastellen

R zeigt standardmäßig 6 Nachkommastellen an. Intern rechnet R mit "doppelter Genauigkeit". Damit sind Nummern mit ca. 17 Stellen (1 Ziffer + 16 Nachkommastellen oder eben eine 17 stellige Zahl ohne Nachkommastellen) repräsentierbar. Möchte man die Nachkommastellen entsprechend angezeigt bekommen, so kann man dies mit der options-Funktion einstellen:

```
# gib ab jetzt 16 Nachkommastellen aus
options(digits=17)
# Wurzel aus 3
sqrt(3)
```

```
[1] 1.7320508075688772
```

Nach dieser Eingabe liefert R Ergebnisse mit bis zu 16 Nachkommastellen.
Wir stellen den Wert wieder zurück auf 6 Nachkommastellen.

```
# gib ab jetzt wieder "nur" 6 Nachkommastellen aus
options(digits=7)
```

Runden

Ziehen wir die Wurzel aus 3, erhalten wir ein Ergebnis mit 6 Nachkommastellen.

```
# Wurzel aus 3
sqrt(3)
```

```
[1] 1.732051
```

Dieses Ergebnis kann mit der Funktion `round()` gerundet werden.

```r
# Runde auf 2 Nachkommastellen
round(sqrt(3), digits=2)

# oder einfach
round(sqrt(3), 2)
```

```
[1] 1.73
```

```r
# Runde auf ganze Zahl
round(sqrt(3))
```

```
[1] 2
```

Bitte beachten Sie, dass R *mathematisch*[1] rundet (gemäß dem IEEE 754 Standard), und **nicht** kaufmännisch. Beim mathematischen Runden wird eine 5 am Ende immer auf die nächste *gerade* Zahl gerundet.
Dies soll folgendes Beispiel verdeutlichen:

```r
# mathematisches Runden
round(3.5) # ergibt 4
```

```
[1] 4
```

```r
round(2.5) # ergibt 2
```

```
[1] 2
```

Neben `round()` existieren noch die Spezialfunktionen `ceiling()` und `floor()`. Die Funktion `ceiling()` rundet immer zur nächsten ganzen Zahl **auf**, und die Funktion `floor()` rundet immer zur nächsten ganzen Zahl **ab**.

Man kann sich das so vorstellen:

- Sie pflücken Äpfel und legen diese in Körbe. So erhalten Sie 10 volle und einen halbvollen Korb, oder andersgesagt 10,5 Körbe. Leider können Sie nur volle Körbe verkaufen. In diesem Fall würden Sie mittels `floor()` auf die ganze Zahl abrunden, um zu ermitteln, wieviele Körbe Sie verkaufen können.

- Aus den restlichen Äpfeln pressen Sie Saft, den Sie in Flaschen füllen. In jede Flasche passen 100ml, und Sie pressen insgesamt 565ml Saft aus den Äpfeln. Um zu ermitteln, wieviele Flaschen Sie benötigen, runden Sie mittels `ceiling()` auf die nächste Zahl auf, damit nach 5 vollen Flaschen auch die restlichen 65ml aufgefangen werden können (auch wenn die letzte Flasche nicht voll wird).

[1] siehe https://de.wikipedia.org/wiki/Rundung#Symmetrisches_Runden

```r
# ceiling rundet immer auf
ceiling(sqrt(3))
```

```
[1] 2
```

```r
# floor rundet immer ab
floor(sqrt(3))
```

```
[1] 1
```

```r
# ceiling rundet immer auf
ceiling(1.0008)
```

```
[1] 2
```

```r
# floor rundet immer ab
floor(1.9999)
```

```
[1] 1
```

R nutzt die wissenschaftliche Schreibweise sehr großer und kleiner Zahlen (so genannte *Zehnerpotenzen*).

```r
# erzeuge eine sehr große Zahl
987654321 * 987654321
```

```
[1] 9.754611e+17
```

Um die Ergebniszahl vor der Zehnerpotenz zu runden, kann die Funktion `format()` genutzt werden.

```r
# runde vor der Zehnerpotenz
format(987654321 * 987654321, digits=2)
```

```
[1] "9.8e+17"
```

1.2.3.2 Variablen

Variablen dienen der Speicherung von Werten oder komplexen Daten (streng genommen spricht man in R aber immer von "Objekten"). Variablen bestehen aus einem Namen und dem zugewiesenen Wert (bzw. Werten). Der Name einer Variable in R darf nicht mit einer Zahl beginnen, das heisst Variablennamen wie `1stTry` sind nicht erlaubt. Ein Wert kann einer Variable auf zwei Weisen zugewiesen werden:

```r
# weise Variable "x" den Wert 8 zu
x <- 8
```

Die in R gängige Variante ist das Zuweise mittels "Pfeil", den man mit `<` und - nachahmt. Die Zuweisung per Gleichheitszeichen funktioniert aber ebenso.

```r
# weise Variable "x" den Wert 8 zu
x <- 8

# funktioniert auch mit "="
x = 8

# funktioniert auch in die andere Richtung
8 -> x
```

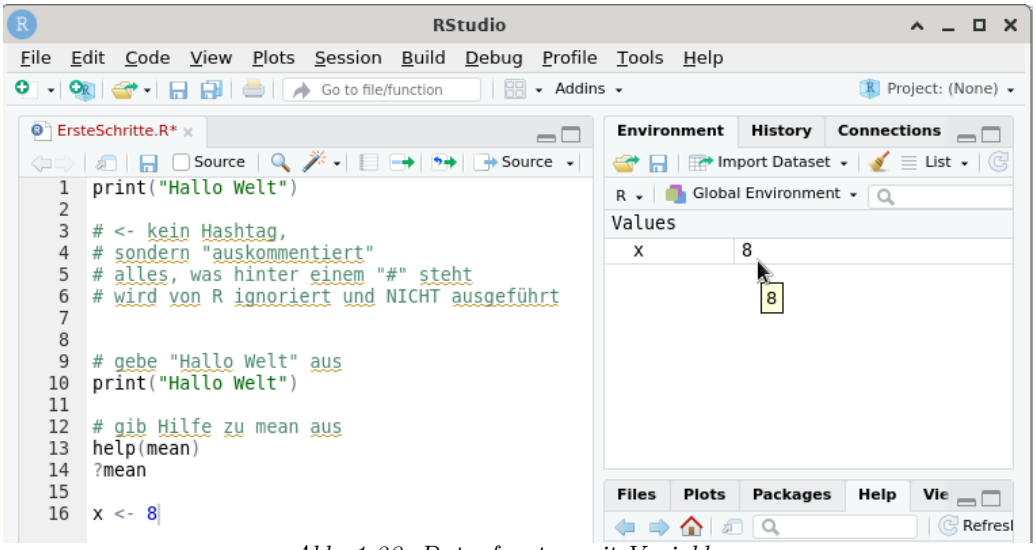


Abb. 1.23: Datenfenster mit Variable `x`

Wir haben nun eine Variable `x` mit dem Wert `8`. Beachten Sie, dass R zwischen Groß- und Kleinschreibung unterscheidet. Unsere Variable heisst `x` und nicht `X`!
In `RStudio` wird uns nun im Datenfenster (oberes rechtes Viertel) die Variable `x` mit ihrem Wert `8` angezeigt (Abbildung 1.23).
Den Wert einer Variable können wir ausgeben lassen, indem wir einfach ihren Namen in die R -Konsole schreiben

```
# gib Wert von x aus
x
```

```
[1] 8
```

Wie erwähnt unterscheidet R zwischen Groß- und Kleinschreibung, und so erhalten wir einen Fehler wenn wir eingeben:

```
# großes "X", kein kleines "x"
X
```

```
Fehler: Objekt 'X' nicht gefunden
```

Mit Variablen kann auch gerechnet werden:

```
# erzeuge Variable "y" mit Wert 10
y <- 10
# erzeuge Variable "z" mit Wert 5
z <- 5
# rechne ein bisschen
x+y+z
```

```
[1] 23
```

```
# noch ein bisschen rechnen
x/(y+z)
```

```
[1] 0.5333333
```

```
# speichere Ergebnis in weitere Variable "Dummy"
Dummy <-x+(y^z)
# Gib Wert(e) von Dummy aus
Dummy
```

```
[1] 100008
```

Wertereihen

Wir können einer Variable auch eine Wertereihe (also mehrere Werte) zuweisen - spätestens jetzt sollten wir nicht mehr von einer "Variable" sprechen, sondern von einem "Objekt". Dies erfolgt für gewöhnlich mit der Funktion `c()` (für ***c**oncatenate*, "aneinanderreihen"), indem man die einzelnen Werte mit einem Komma getrennt der Funktion übergibt.

```
# weise Wertereihe 1 bis 10 mit c() zu
a <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
# Gib Werte von "a" aus
a
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Mit `c()` kann auch Text, also z.B. Wörter, übergeben werden.

```
# es müssen keine Zahlen sein
b <- c("eins", "zwei", "drei", "vier", "fünf", "sechs", "sieben", "acht", "neun", "zehn")
# Gib Werte von "b" aus
b
```

```
 [1] "eins"   "zwei"   "drei"   "vier"
 [5] "fünf"   "sechs"  "sieben" "acht"
 [9] "neun"   "zehn"
```

Der Text muss in Anführungszeichen gesetzt werden, da R ansonsten nicht das Wort `"fünf"` versteht, sondern nach einem Objekt namens `fünf` sucht. Denn Variablen und Objekte können ebenfalls aneinandergereiht werden.

```
# Variablen aneinanderreihen
c(x, y, z)
```

```
[1]  8 10  5
```

💀 Achtung, Anfängerfehler! 💀

Wichtig ist, dass innerhalb der `c()`-Kette nur Werte des selben Typs (entweder Zahlen oder Text) angegeben werden. Mischt man Text mit Zahlen, wandelt `c()` die Wertereihe auf den "kleinsten gemeinsamen Nenner" um, und das ist immer "Text".

```
# c() fällt auf den "kleinsten gemeinsamen Nenner" zurück
c(x, y, z, b)
```

```
 [1] "8"      "10"     "5"      "eins"   "zwei"   "drei"   "vier"
 [8] "fünf"   "sechs"  "sieben" "acht"   "neun"   "zehn"
```

Aus den Zahlen `8`, `10` und `5` sind die *Wörter* `"8"`, `"10"` und `"5"` geworden, und mit denen kann man nun keine Rechenoperationen mehr durchführen.
Das Mischen der Wertetypen ist ein weiterer häufiger Anfängerfehler.

Mit der Funktion `length()` kann die Länge der Wertereihe bestimmt werden

```
1 # länge von Objekt "a"
2 length(a)
```

```
[1] 10
```

Objekt `a` besteht aus `10` Werten.

```
1 # länge von Objekt "Dummy"
2 length(Dummy)
```

```
[1] 1
```

Objekt `Dummy` besteht aus `1` Wert.

Im Datenfenster von `RStudio` (rechtes oberes Viertel) werden alle Objekte angezeigt (Abbildung 1.24).

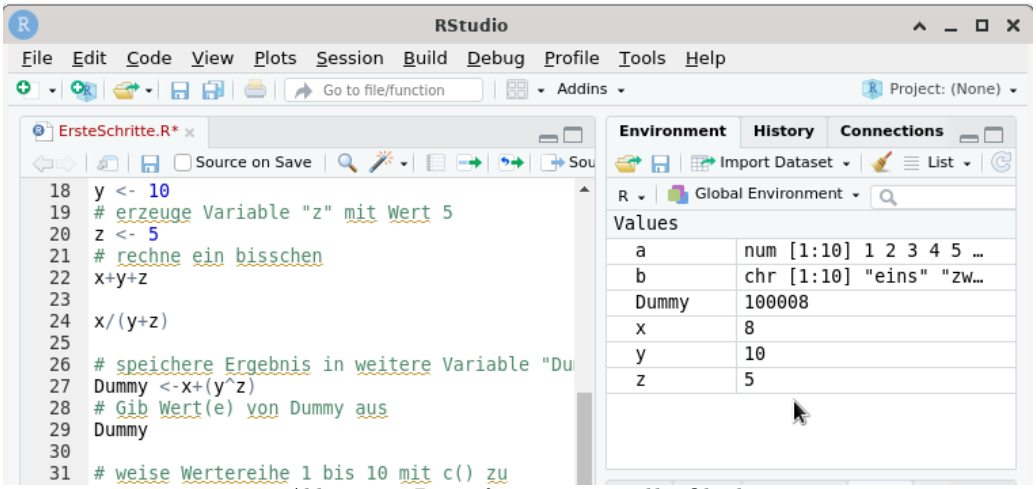


Abb. 1.24: Datenfenster zeigt alle Objekt

Wir sehen dort unsere Objekte `a`, `b`, `Dummy`, `x`, `y` und `z`. Rechts daneben zeigt `RStudio` zusammenfassende Informationen zu den Objekte an. So sieht man direkt, dass der Wert von `Dummy = 100008` ist, oder dass `a` zehn Werte enthält.

In der -Konsole können alle vorhandenen Objekte mit der Funktion `ls()` angezeigt werden.

```
1 # zeige alle Objekte an
2 ls()
```

```
[1] "a"     "b"     "Dummy" "x"     "y"     "z"
```

Objekte lassen sich mit dem Befehl `rm()` löschen.

```
# lösche Objekt "Dummy"
rm(Dummy)

# zeige alle Objekte an
ls()
```

```
[1] "a" "b" "x" "y" "z"
```

Um *alle* Objekte zu löschen, verknüpfen wir den `ls()`-Befehl mit `rm()`. So änder sich der Befehl in:

```
# lösche ALLE Objekte
rm(list=ls())

# zeige alle Objekte an
ls()
```

```
character(0)
```

In `RStudio` löschen Sie alle Objekte aus dem Arbeitsspeicher, indem Sie im Datenfenster auf den kleinen Besen klicken (Abbildung 1.25).

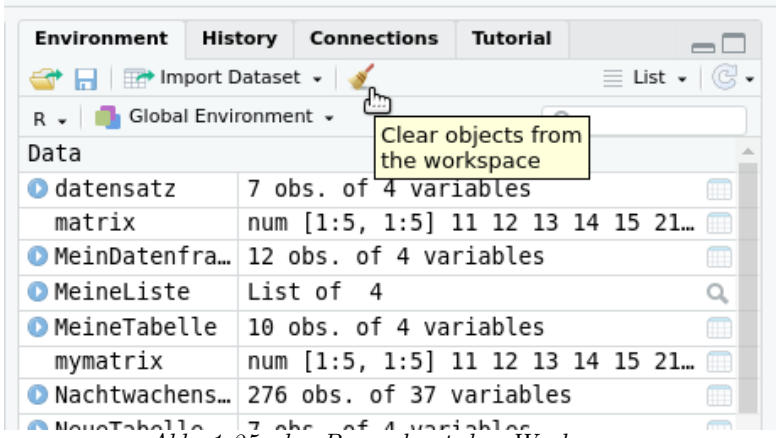


Abb. 1.25: der Besen leert den Workspace

1.2.3.3 Wertesequenzen erzeugen

Bislang haben wir unsere Daten von Hand eingeben. Für die Zahlenreihe von 1 bis 10 sah das so aus:

```
# Zahlenreihe von 1 bis 10
zahlen <- c(1, 2, 3, 4, 5, 6, 7, 8, 9 ,10)

# ausgeben
zahlen
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

In R lassen sich solch kontinuierliche Reihen mit einem : erzeugen:

```
# auch Zahlenreihe von 1 bis 10
zahlen <- 1:10

# ausgeben
zahlen
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

In Kombination mit `c()` geht auch:

```
# neuer Vektor aus 2 Zahlenreihen
zahlen2 <- c(1:10, 100:110)

# ausgeben
zahlen2
```

```
 [1]   1   2   3   4   5   6   7   8   9  10 100 101 102 103 104 105
[17] 106 107 108 109 110
```

Mit der Funktion `seq()` lassen sich `Sequenzen` erstellen. Wir übergeben der Funktion Start- und Endpunkt, und in welchen Schritten gezählt werden soll. Eine Zahlenreihe von `1` bis `10` in den Schritten `0.5` erzeugt man so:

```
# eine Sequenz erstellen
sequenz <- seq(from=1, to=10, by=0.5)

# ausgeben
sequenz
```

```
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
[14]  7.5  8.0  8.5  9.0  9.5 10.0
```

oder schlicht

```
# erstellt die selbe Sequenz
sequenz <- seq(1, 10, 0.5)

# ausgeben
sequenz
```

```
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
[14]  7.5  8.0  8.5  9.0  9.5 10.0
```

Mit der Funktion `rep()` (für ***rep****eat*) lassen sich wiederholte Werte erzeugen

```
# erzeuge 20 wiederholte Werte,
# 20 mal "weiblich"
geschlecht <- rep("weiblich", 20)

# ausgeben
geschlecht
```

```
 [1] "weiblich" "weiblich" "weiblich" "weiblich" "weiblich" "weiblich"
 [7] "weiblich" "weiblich" "weiblich" "weiblich" "weiblich" "weiblich"
[13] "weiblich" "weiblich" "weiblich" "weiblich" "weiblich" "weiblich"
[19] "weiblich" "weiblich"
```

In Kombination mit `c()` geht auch:

```
# erzeuge 20 wiederholte Werte,
# 10 mal "weiblich" und 10 mal "maennlich"
geschlecht <- c(rep("weiblich", 10), rep("maennlich", 10))

# ausgeben
geschlecht
```

```
 [1] "weiblich"  "weiblich"  "weiblich"  "weiblich"  "weiblich"
 [6] "weiblich"  "weiblich"  "weiblich"  "weiblich"  "weiblich"
[11] "maennlich" "maennlich" "maennlich" "maennlich" "maennlich"
[16] "maennlich" "maennlich" "maennlich" "maennlich" "maennlich"
```

Oder anders verschachtelt:

```
# erzeuge 20 wiederholte Werte, anders verschachtelt
# 10 mal das Paar "weiblich" "maennlich"
geschlecht <- rep(c("weiblich", "maennlich"), 10)

# ausgeben
geschlecht
```

```
 [1] "weiblich"  "maennlich" "weiblich"  "maennlich" "weiblich"
 [6] "maennlich" "weiblich"  "maennlich" "weiblich"  "maennlich"
[11] "weiblich"  "maennlich" "weiblich"  "maennlich" "weiblich"
[16] "maennlich" "weiblich"  "maennlich" "weiblich"  "maennlich"
```

Zufallszahlen

In R lassen sich auch Zufallszahlen generieren. Hierfür steht die Funktion `sample()` zur Verfügung.

```
# erzeuge 20 Zufallswerte
sample(20)
```

```
 [1]  4  1 14 17 15  2 18 13 20  6 19 16  9  8 11 12 10  3  5  7
```

Der Bereich, aus welchem die Zahlen gezogen werden sollen, kann ebenfalls angegeben werden.

```
# erzeuge 20 Zufallswerte zwischen 150 und 250
sample(150:250, 20)
```

```
 [1] 161 234 199 210 225 236 247 232 217 179 246 150 220 226
[15] 214 177 188 202 160 224
```

Über die Option `replace` kann angegeben werden, ob Werte auch doppelt vorkommen können (so genanntes *zurücklegen*).

```r
# erzeuge 20 Zufallswerte zwischen 150 und 250
# MIT zurücklegen
sample(150:250, 20, replace=TRUE)
```

```
 [1] 242 163 187 198 204 190 152 184 243 176 170 170 179 217
[15] 150 198 202 227 166 223
```

Zudem bieten die in R implementierten Wahrscheinlichkeitsverteilungen über ihre r-Funktionen (für *random*, siehe Abschnitt 1.2.4 auf Seite 87) die Möglichkeit, zufällige Werte zu erzeugen.
Die Funktion `runif()` erzeugt Zufallswerte aus der "stetigen Gleichverteilung". Standardmäßig liegen die Zufallswerte zwischen `0` und `1`.

```r
# erzeuge 20 Zufallswerte zwischen 0 und 1
runif(20)
```

```
 [1] 0.62383493 0.48679031 0.78736781 0.56125648 0.95977623
 [6] 0.14850696 0.35799878 0.67937020 0.09689319 0.63402240
[11] 0.36186202 0.19484200 0.94330547 0.60192248 0.04309927
[16] 0.70540669 0.75177372 0.72113117 0.60635434 0.50857776
```

Man kann aber (ähnlich wie bei `sample()`) den Bereich festlegen, aus welchem die Werte gezogen werden sollen.

```r
# erzeuge 20 Zufallswerte zwischen 30 und 150
runif(20, min=30, max=150)
```

```
 [1]  92.79867  90.43651  51.79427 137.82589  40.12666
 [6] 123.10785  43.78025 101.20329 147.89779  81.56550
[11] 115.65866  48.43188 109.07900 114.22313  90.78145
[16]  77.35028  43.79792  74.98960 103.78252  45.64845
```

Für ganzzahlige Werte können die Rundungsfunktionen `round()`, `floor()` und `ceiling()` eingeschoben werden.

```r
# erzeuge 20 GANZZAHLIGE Zufallswerte zwischen 30 und 150
floor(runif(20, min=30, max=150))
```

```
 [1] 127  52 142  73 137  59 130  60 101  75  35 119 113 111
[15]  83 138  86 148 113 138
```

Um Zufallszahlen aus der Standardnormalverteilung zu generieren eignet sich die Funktion `rnorm()`.

```r
# erzeuge 40 zufällige Werte aus der
# Standardnormalverteilung
normal <- rnorm(40)

# ausgeben
normal
```

```
 [1] -1.30683973  0.08581781  0.52898949  0.12316733  1.96814473
 [6] -0.03224878  1.24255982  0.80833688  0.12617009 -1.68422729
[11]  2.07087237  0.70833213  0.90960225  1.52042129 -0.61109432
[16] -0.56164887  0.87728035 -0.07644280 -1.09885688 -1.80537158
[21]  0.34978431 -0.93592539 -0.66504186 -0.95879982 -1.60549156
[26] -1.01631415 -0.94082868  0.90715833  0.32633218 -0.11079465
[31] -0.92969431 -1.67418761 -0.20824912  1.21117314 -0.59430724
[36]  0.03055659  0.65196122  0.58226606 -1.33694530  1.16788244
```

Somit erzeugt `rt()` zufällige Werte aus der t-Verteilung.

```
# erzeuge 40 zufällige Werte aus der
# t-Verteilung bei 10 Freiheitsgraden
t <- rt(40, df=10)

# ausgeben
t
```

```
 [1]  0.52311923 -0.67621471  1.04927080 -0.81329198 -0.01622871
 [6] -0.88733849 -0.55410697 -0.37071397  0.79783044 -2.05919751
[11]  0.03762458 -0.09059438  0.49890986  1.02610386  0.72289407
[16]  0.83527344 -1.92007156 -0.00191527  1.51483848 -0.02774711
[21]  1.66549996  0.38796390  0.67036334 -0.72107346 -0.77602159
[26] -0.50466444  0.20603820 -4.16495836 -0.86765434 -2.75438947
[31]  0.71361286 -0.35672527  0.06796026  0.21855036  0.32415552
[36]  1.03790635 -1.53014553 -1.18606832 -0.40752377 -0.54694851
```

1.2.3.4 Datentypen

R ist eine objektorientierte Sprache und kann verschiedene Formen von Datenklassen und -typen verarbeiten.
Die für uns wichtigsten Datentypen sind:

- numerisch (Zahlen, mit denen man rechnen kann)
- character (Zeichenkette, also Text und Wörter)
- logisch (Wahr/Falsch, bzw. `TRUE` und `FALSE`)

Wir haben im oberen Abschnitt (1.2.3.2) schon zwei Daten*typen* kennengelernt, zum einen den Typ `numerisch`, denn wir haben den Variablen `x`, `y` und `z` jeweils Zahlenwerte zugeordnet, mit denen wir rechnen konnten.

```
# numerischer Datetyp
v_num <- c(0, 8, 15)

# rechnen ist möglich
v_num * 100
```

```
[1]    0  800 1500
```

Zum anderen hatten wir der Variable `b` Wörter zugewiesen. Dies entspricht dem Datentyp "character".

```
# Datetyp "Character"
v_char <- c("Hallo", "ihr", "lieben", "Leute")

# rechnen ist nicht möglich
v_char * 100
```

```
Fehler in v_char * 100 : nicht-numerisches Argument für binären Operator
```

Variablen vom Typ `logisch` enthalten die logischen Aussagen `TRUE` und `FALSE` bzw. deren Abkürzungen `T` und `F`.
Eine logische Aussage kann zum Beispiel durch einen Vergleich erzeugt werden

```
# ist 50 = 100
50==100
```

```
[1] FALSE
```

Da `50` eben nicht gleich `100` ist, erhalten wir die logische Aussage `FALSE` zurück.

```
# ist 50 kleiner als 100
50<100
```

```
[1] TRUE
```

Die Zahl `50` ist kleiner als `100`, darum erhalten wir die logische Antwort `TRUE` zurück.

Das kann man auch auf Wertereihen anwenden.

```
# erzeuge 20 werte
reihe <- seq(1, 100, 5)
# ausgeben
reihe
```

```
 [1]  1  6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
# Prüfe jeden Wert in "reihe", ob er kleiner als 75 ist
reihe<75
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[12]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
# Prüfe jeden Wert in "reihe", ob er kleiner als 75 ist
# und speichere das Ergebnis in Variable "logical"
logical <- reihe<75

# zeige den 7. und 18. Wert von "logical" an
logical[c(7,18)]
```

```
[1]  TRUE FALSE
```

Mit der Funktion `class()` kann der Datentyp (und auch die Datenklasse, dazu später mehr) angezeigt werden.

```
# welcher Datentyp ist Variable "logical"
class(logical)
```

```
[1] "logical"
```

```
# welcher Datentyp ist Variable "v_char"
class(v_char)
```

```
[1] "character"
```

```
# welcher Datentyp ist Variable "v_num"
class(v_num)
```

```
[1] "numeric"
```

Mit den Funktionen `is.numeric()`, `is.logical()` und `is.character()` können wir den Datentyp ebenfalls überprüfen. Als Antwort erhalten wir ein logisches `TRUE` oder `FALSE`.

```
# ist Variable "v_num" numerisch?
is.numeric(v_num)
```

```
[1] TRUE
```

```
# ist Variable "v_char" numerisch?
is.numeric(v_char)
```

```
[1] FALSE
```

```
# ist Variable "v_num" logical?
is.logical(v_num)
```

```
[1] FALSE
```

```
# ist Variable "v_char" logical?
is.logical(v_char)
```

```
[1] FALSE
```

```
# ist Variable "v_num" character?
is.character(v_num)
```

```
[1] FALSE
```

```
# ist Variable "v_char" character?
is.character(v_char)
```

```
[1] TRUE
```

```
# ist Variable "logical" character?
is.character(logical)
```

```
[1] FALSE
```

```
# ist Variable "logical" logical?
is.logical(logical)
```

```
[1] TRUE
```

1.2.3.5 Datenklassen

Die Datentypen können wiederum in Daten*klassen* gespeichert werden.
Für uns wichtige Datenklassen sind:

- Vektoren
- Matrizen

- Faktoren (Gruppen, Rangfolge)
- Datenframes
- Listen

1.2.3.5.1 Vektoren

Vektoren haben wir schon kennengelernt. Ein Vektor ist eine einfache Wertereihe vom selben Datentyp. So erzeugt die Funktion `c()` einen Datenvektor. Alle Werte der Datenklasse `Vektor` müssen Werte des selben Werte*typs* enthalen. Kombinieren wir `numerische`, `character` und `logische` Werte in einem Vektor, so wandeln sich alle Werte in den kleinsten gemeinsamen Datentyp (nämlich `character`) um.

```
# c() fällt auf den "kleinsten gemeinsamen Nenner" zurück
c(1, 2, 3, 4, "fünf", "sechs", TRUE, TRUE, FALSE)
```

```
[1] "1"     "2"     "3"     "4"     "fünf"  "sechs" "TRUE"  "TRUE"
[9] "FALSE"
```

Alle Werte sind nun vom Datentyp `character`, erkennbar an den Anführungszeichen.
Auf die Werte kann man zugreifen, indem man den Variablennamen eingibt:

```
# erzeuge einen Vektor
vektor <- seq(1, 20, 1)

# anzeigen
vektor
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Auf die einzelnen Werte des Vektors kann man Zugreifen, indem man die gewünschte Position in eckigen Klammern an den Variablennamen anhgängt.

```
# Zeige den ersten Wert von "vektor"
vektor[1]
```

```
[1] 1
```

Auch hier können wir Positions*bereiche* mit einem : angeben

```
# Zeige die Werte an Position 4 bis 15
vektor[4:15]
```

```
 [1]  4  5  6  7  8  9 10 11 12 13 14 15
```

Mit einem Minuszeichen können auch bestimmte Werte oder Wertbereiche ausgelassen werden.

```
# Zeige "vektor" OHNE den ersten Wert
vektor[-1]
```

```
[1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
# Zeige "vektor" OHNE die Werte 10 bis 14
vektor[-(10:14)]
```

```
[1]  1  2  3  4  5  6  7  8  9 15 16 17 18 19 20
```

Wieviele Werte ein Vektor enthält erfährt man mit der Funktion `lenght()`.

```
# wieviele Werte hat "vektor"?
length(vektor)
```

```
[1] 20
```

Der Vektor `vektor` beinhaltet 20 Werte.

Mit der Funktion `is.vector()` kann geprüft werden, ob ein Objekt ein Vektor ist.

```
# ist "vektor" ein Vektor?
is.vector(vektor)
```

```
[1] TRUE
```

1.2.3.5.2 Matrizen

Matrizen sind zweidimensionale Strukturen (Tabellen) und werden von R intern durch Vektoren dargestellt. Dies impliziert, dass alle Werte der Matrix vom selben Datentyp (z.B. `numerisch`) sein müssen, genau so wie bei Vektoren.
Um besser zu erklären, wie Matrizen funktionieren, erzeugen wir zunächst ein paar Beispielvektoren.

```
# Erzeuge Testwertereihen
a <- c(11, 12, 13, 14, 15)
b <- c(21, 22, 23, 24, 25)
c <- c(31, 32, 33, 34, 35)
d <- c(41, 42, 43, 44, 45)
e <- c(51, 52, 53, 54, 55)
f <- c("eins", "zwei", "drei", "vier", "fünf")

# Füge alle Zahlen zu einem Vektor zusammen
alle <- c(a, b, c, d, e)

# anzeigen
alle
```

```
 [1] 11 12 13 14 15 21 22 23 24 25 31 32 33 34 35 41 42 43 44 45
[21] 51 52 53 54 55
```

Die Funktoin `martix()` setzt aus einem Vektor eine Matrix zusammen. Lässt man alle Parameter im Funktionsaufruf `matrix()` leer, wird eine Matrix mit `1` Spalte erzeugt.

```
# Erzeuge eine Matrix aus Vektor "a"
matrix(a)
```

```
     [,1]
[1,]   11
[2,]   12
[3,]   13
[4,]   14
[5,]   15
```

Mit dem Parameter `ncol` kann die gewünschte Anzahl an Spalten übergeben werden:

```
# Erzeuge eine Matrix aus Vektoren "a" und "b"
# mit 2 Spalten
matrix(c(a,b), ncol=2)
```

```
     [,1] [,2]
[1,]   11   21
[2,]   12   22
[3,]   13   23
[4,]   14   24
[5,]   15   25
```

Mit dem Parameter `nrow` kann die gewünschte Anzahl an Zeilen übergeben werden:

```
# Erzeuge eine Matrix aus Vektoren "a" und "b"
# mit 2 Zeilen
matrix(c(a,b), nrow=2)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   11   13   15   22   24
[2,]   12   14   21   23   25
```

 Achtung, Anfängerfehler! 

Achten Sie auf die Reihenfolge, in der die Werte in der Matrix angelegt wurden. Das ist wahrscheinlich nicht das Ergebnis, das Sie erwartet haben. Die Funktion `matrix()` arbeitet standardmäßig die Werte pro Spalte (*spaltenorientiert*) ab. Wir können das mit dem Parameter `byrow` ändern:

```
# Erzeuge eine Matrix aus Vektoren "a" und "b"
# mit 2 Zeilen, diesmal zeilenorientiert
matrix(c(a,b), nrow=2, byrow=TRUE)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   11   12   13   14   15
[2,]   21   22   23   24   25
```

Sobald eine Matrix mit `2` oder mehr Spalten angelegt werden soll, muss der Datenvektor so viele Werte enthalten, dass die gewünschte Matrix vollständig erstellt werden kann. Sollten zu wenige Werte vorhanden sein, gibt  eine Warnmeldung aus. Für eine Matrix mit `2` Spalten muss also eine `gerade` Anzahl an Werten vorhanden sein.

```
# Matrix mit 2 Spalten benötigt gerade Anzahl an Werten
# daher gibt R (mit nur 9 Werten) eine Warnmeldung aus
matrix(1:9, nrow=2)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8    1
Warnmeldung:
In matrix(1:9, nrow = 2) :
  Datenlänge [9] ist kein Teiler oder Vielfaches der Anzahl der Zeilen [2]
```

Wie Sie sehen versucht R selbstständig die Matrix zu vervollständigen. Hierfür wiederholt R die angegebenen Werte so lange, bis die Matrix "voll" ist. Im obigen Beispiel wurde nach der `9` wieder eine `1` eingetragen.

Wie bereits erwähnt müssen die Werte vom selben Datentyp sein. Mischt man `numeric` mit `character`, fällt auch Matrix auf den kleinsten gemeinsamen Datentyp (`character`) zurück.

```
# Erzeuge eine Matrix aus Vektoren "a" und "f"
# fällt auch Typ "char" zurück
matrix(c(a,f), nrow=2, byrow=TRUE)
```

```
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,] "11"   "12"   "13"   "14"   "15"
[2,] "eins" "zwei" "drei" "vier" "fünf"
```

Neben `matrix()` können auch die Befehle `rbind()` (für ***r**owbind*) und `cbind()` (für ***c**olumnbind*) verwendet werden, um eine Matrix zu erzeugen.
Der Befehl `cbind()` fügt die übergebenen Vektoren *spaltenorientiert* zu einer Matrix zusammen.

```
# Erzeuge eine Matrix aus Vektoren "c" und "d"
# spaltenorientiert
cbind(c, d)
```

```
      c  d
[1,] 31 41
[2,] 32 42
[3,] 33 43
[4,] 34 44
[5,] 35 45
```

Der Befehl `rbind()` fügt die übergebenen Vektoren *zeilenorientiert* zu einer Matrix zusammen.

```
# Erzeuge eine Matrix aus Vektoren "c" und "d"
# zeilenorientiert
rbind(c, d)
```

```
  [,1] [,2] [,3] [,4] [,5]
c   31   32   33   34   35
d   41   42   43   44   45
```

Wir speichern eine Matrix in einer Variable:

```r
# Erzeuge eine Matrix aus Vektor "alle"
# mit 5 Spalten
mymatrix <- matrix(alle, ncol=5)

# ausgeben
mymatrix
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   11   21   31   41   51
[2,]   12   22   32   42   52
[3,]   13   23   33   43   53
[4,]   14   24   34   44   54
[5,]   15   25   35   45   55
```

An der Ausgabe der Spalten- und Zeilentitel lässt sich erahnen, wie die einzelnen Werte einer Matrix refenziert werden können. Bei Vektoren können die einzelnen Werte abgerufen werden, indem in eckigen Klammern die gewünschte Position angegeben wird. Dies funktioniert bei Matrizen ähnlich, jedoch muss innerhalb der eckigen Klammer zwischen *Zeilen* und *Spalten* unterschieden werden. Dies geschieht mit einem Komma, wobei vor dem Komma die Zeilen, und nach dem Komma die Spalten referenziert werden.

```r
# zeige die 1. Zeile der Matrix
# die Zahl vor dem Komma repräsentiert die Zeilen
mymatrix[1,]
```

```
[1] 11 21 31 41 51
```

```r
# zeige die 1. Spalte der Matrix
# die Zahl nach dem Komma repräsentiert die Spalten
mymatrix[,1]
```

```
[1] 11 12 13 14 15
```

Wie in einem Koordinatensystem können nun gezielt einzelne Werte oder Wertbereiche referenziert werden.

```r
# zeige den Wert in Zeile 3 und Spalte 2
mymatrix[3,2]
```

```
[1] 23
```

```r
# zeige die Werte aus Zeile 2 bis 4
# und Spalte 1 bis 3
mymatrix[2:4,1:3]
```

```
     [,1] [,2] [,3]
[1,]   12   22   32
[2,]   13   23   33
[3,]   14   24   34
```

Mit der Funktion `t()` kann die Matrix transponiert werden, das beduetet, es werden Zeilen und Spalten diagonal gespiegelt.

```r
# zeige meine Matrix
mymatrix
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   11   21   31   41   51
[2,]   12   22   32   42   52
[3,]   13   23   33   43   53
[4,]   14   24   34   44   54
[5,]   15   25   35   45   55
```

```
# transponiere meine Matrix
t(mymatrix)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   11   12   13   14   15
[2,]   21   22   23   24   25
[3,]   31   32   33   34   35
[4,]   41   42   43   44   45
[5,]   51   52   53   54   55
```

Mit den Funktionen `colnames()` und `rownames()` können Spalten und Zeilen noch betitelt werden.
Wir benennen unsere Spalten von “a” bis “e”:

```
# benenne Spalten der Matrix
colnames(mymatrix) <- c("a", "b", "c", "d", "e")

# anzeigen
mymatrix
```

```
     a  b  c  d  e
[1,] 11 21 31 41 51
[2,] 12 22 32 42 52
[3,] 13 23 33 43 53
[4,] 14 24 34 44 54
[5,] 15 25 35 45 55
```

Wir benennen unsere Zeilen von römisch `I` bis `V`:

```
# benenne Spalten der Matrix
rownames(mymatrix) <- c("I", "II", "III", "IV", "V")

# anzeigen
mymatrix
```

```
    a  b  c  d  e
I   11 21 31 41 51
II  12 22 32 42 52
III 13 23 33 43 53
IV  14 24 34 44 54
V   15 25 35 45 55
```

Mit der Funktion `class()` kann die Datenklasse angezeigt werden.

```
# welcher Datentyp ist Variable "mymatrix"
class(mymatrix)
```

```
[1] "matrix" "array"
```

Mit der Funktion `is.matrix()` kann zudem geprüft werden, ob ein Objekt eine Matrix ist.

```
# ist "mymatrix" eine Matrix?
is.matrix(mymatrix)
```

```
[1] TRUE
```

```
# ist "vector" eine Matrix?
is.matrix(vector)
```

```
[1] FALSE
```

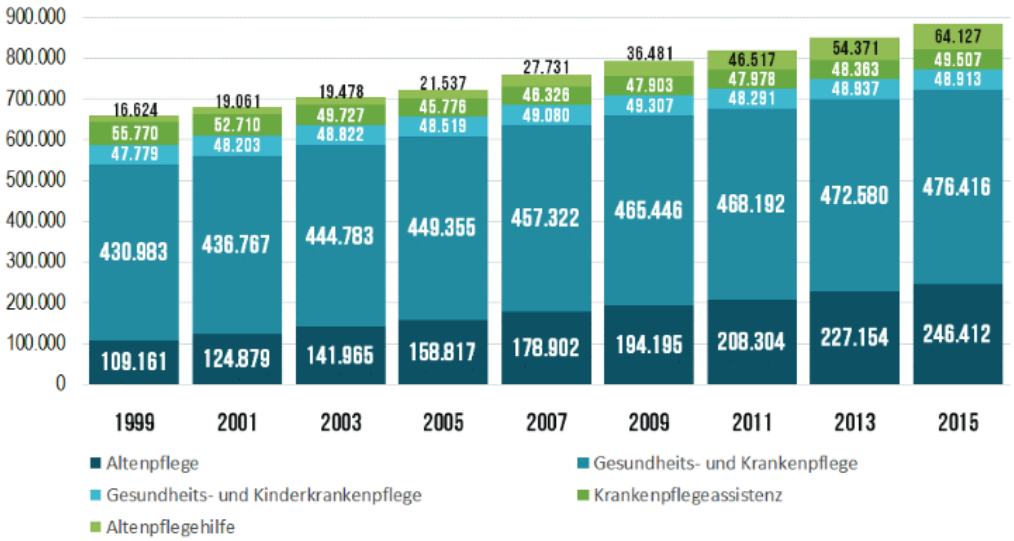


Abb. 1.26: Beschäftigte in Pflegeberufen (aus [2, S.31])

Als Übung können wir nun beispielsweise die Anzahl der Beschäftigten in Pflegeberufen aus dem "Pflegethermometer 2018"[2, S.31] (siehe Abbildung 1.26), als Matrix in R übertragen.

```
# Schreibe die Zahlen reihenweise aus der Grafik ab
Pflegeberufe <- c(16624,  19061,  19478,  21537,  27731,  36481,  46517,  54371,  64127,
                  55770,  52710,  49727,  45776,  48326,  47903,  47978,  48363,  49507,
                  47779,  48203,  48822,  48519,  49080,  49307,  48291,  48937,  48913,
                  430983, 436767, 444783, 449355, 457322, 465446, 468192, 472580, 476416,
                  109161, 124879, 141965, 158817, 178902, 194195, 208304, 227154, 246412
                  )
# überführe in Matrix mit 9 Spalten
# Die Werte kommen reihenmäßig
Pflegeberufe <- matrix(Pflegeberufe, byrow=T,ncol=9)

# benenne die Spalten
colnames(Pflegeberufe) <- c(1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015)

# benenne die Reihen
```

```
16 rownames(Pflegeberufe) <- c("Krankenpflegeassistenz", "Altenpflegehilfe", "Kinderkrankenpflege",
   "Krankenpflege", "Altenpflege")
17
18 # zeige Tabelle
19 Pflegeberufe
```

```
                       1999   2001   2003   2005   2007   2009   2011   2013   2015
Krankenpflegeassistenz  16624  19061  19478  21537  27731  36481  46517  54371  64127
Altenpflegehilfe        55770  52710  49727  45776  48326  47903  47978  48363  49507
Kinderkrankenpflege     47779  48203  48822  48519  49080  49307  48291  48937  48913
Krankenpflege          430983 436767 444783 449355 457322 465446 468192 472580 476416
Altenpflege            109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Wir werden auf diese Matrix später noch zurückkommen.

1.2.3.5.3 Faktoren

Die Datenklasse `factor` beschreibt gruppierte (nominale) oder ranggeordnete (ordinale) Werte. Gruppierte Werte sind beispielsweise "Beruf", "Konfession", "Familienstand" oder "Geschlecht". Es liegt keine Reihenfolge unter den Gruppen vor. Sie werden mit der Funktion `factor()` erstellt.

```
1 # nominalen Faktoren erstellen
2 v_fac1 <- factor(c("maennlich", "weiblich", "divers"))
3
4 # ausgeben
5 v_fac1
```

```
[1] maennlich weiblich  divers
Levels: divers maennlich weiblich
```

Unsere Variable `v_fac1` besteht aus 3 Gruppen, nämlich `divers`, `maennlich` und `weiblich` mit je einem Wert. Erzeugen wir die Daten erneut, diesmal mit mehr Werten und einer abgekürzten Schreibweise für das Geschlecht.

```
1 # Faktoren neu erstellen, mit mehr Werten
2 v_fac1 <- factor(c("m", "w", "d","m", "w", "d","m", "w", "d","m", "w", "d"))
3
4 # ausgeben
5 v_fac1
```

```
 [1] m w d m w d m w d m w d
Levels: d m w
```

In der ersten Zeile sehen wir unsere Datenreihe, in der zweiten Zeile alle `Gruppen` (Levels), in diesem Falle `d`, `m` und `w`.

Für ordninale (also ranggeordnete) Werte nehmen wir ebenfalls die `factor()`-Funktion. Versuchen wir also, klassiche Schulnoten abzubilden. Wir stellen uns eine virtuelle Liste mit Schulnoten von 100 SchülerInnen vor. Wir übertragen die Noten von der virtuellen Liste in R , und zwar in der Reihenfolge, wie sie auf unserer virtuellen Notenliste stehen könnten (sprich: *unsortiert*).

```r
# Faktoren aus Notenliste erstellen
v_fac2 <- factor(c("gut", "ausreichend", "sehr gut", "ausreichend", "befriedigend", "magelhaft", "ungenügend",
"gut", "gut", "sehr gut"))

# ausgeben
v_fac2
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend magelhaft
 [7] ungenügend   gut          gut          sehr gut
Levels: ausreichend befriedigend gut magelhaft sehr gut ungenügend
```

In der ersten Zeile sehen wir unsere Datenreihe, in der letzten Zeile alle `Ränge` (Levels). Ein Blick auf die Levels zeigt aber auch, dass diese in einer falschen Reihehnfolge angelegt wurden. Der erste Rang ist hier `ausreichend`, und der zweite `befriedigend` (es sollte ja eigentlich so sein, dass `sehr gut` der erste Rang ist, und `gut` der zweite, usw). Das liegt daran, dass wir die Noten unsortiert aneinandergereiht haben. R erstellt die Reihenfolge der Levels anhand der Reihenfolge, in der sie eintreffen.

Achtung, Anfängerfehler!

Dies ist ein häufiger Anfängerfehler bei der Erstellung von Faktoren!

Um unsere Schulnoten in der richtigen Levelreihenfolge anzulegen, müssen wir dem Befehl `factor()` eben diese Reihenfolge über die Option `levels` mitgeben (mehr Informationen erhalten Sie über die Hilfeseite `?factor`). Konkret übergeben wir per `levels=c()` die Levelnamen von `sehr gut` bis `ungenügend`.

```r
# Faktoren aus Notenliste erstellen
# diesmal Levelreihenfolge mit "levels=" vorgeben
v_fac2 <- factor(c("gut", "ausreichend", "sehr gut", "ausreichend", "befriedigend", "mangelhaft", "ungenügend",
"gut", "gut", "sehr gut"), levels=c("sehr gut", "gut", "befriedigend", "ausreichend","mangelhaft",
"ungenügend"))

# ausgeben
v_fac2
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend mangelhaft
 [7] ungenügend   gut          gut          sehr gut
Levels: sehr gut gut befriedigend ausreichend mangelhaft ungenügend
```

Die Levels sind nun in der korrekten Reihenfolge.
Wir hätten die Daten aber gar nicht neu eingeben müssen. Der Befehl lässt sich verkürzen, indem man einfach die bestehende Variable als Input nutzt und neu überschreibt:

```r
# Levelreihenfolge in "v_fac2" reparieren und überschreiben
v_fac2 <- factor(v_fac2, levels=c("sehr gut", "gut", "befriedigend", "ausreichend","mangelhaft", "ungenügend"))

# ausgeben
v_fac2
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend mangelhaft
 [7] ungenügend   gut          gut          sehr gut
Levels: sehr gut gut befriedigend ausreichend mangelhaft ungenügend
```

So kann man auch nachträglich die Levelreihenfolge korrigieren.

Mit dem Befehl `revalue()` aus dem `plyr`-Zusatzpaket[2] lassen sich die Werte und Levels von Faktoren umändern. Wir ändern unsere Schulnoten von den ausgeschriebenen Noten hin zu Zahlenwerten. Hierfür erzeugen wir eine neue Variable `v_fac3`.

```
# Lade Zusatzpaket "plyr"
library(plyr)

# Ändere Levelnamen
v_fac3 <- revalue(v_fac2, c("sehr gut"="1",
"gut"="2","befriedigend"="3","ausreichend"="4","mangelhaft"="5","ungenügend"="6"))

# Werte ausgeben
v_fac3
```

```
 [1] 2 4 1 4 3 5 6 2 2 1
Levels: 1 2 3 4 5 6
```

Beachten Sie, dass die "Zahlen"werte nur nominaler Natur sind. Wir können mit ihnen nicht rechnen!

```
# rechnen ist mit factor nicht möglich!
v_fac3 * 100
```

```
 [1] NA NA NA NA NA NA NA NA NA NA
Warnmeldung:
In Ops.factor(v_fac3, 100) : '*' not meaningful for factors
```

Die Werte innerhalb einer Faktorenreihe refenziert man so wie bei Vektoren, indem man die gewünschte Position in eckigen Klammern an den Variablennamen anhgängt.

```
# zeige die Werte von 3 bis 7 von "v_fac3"
v_fac3[3:7]
```

```
[1] 1 4 3 5 6
Levels: 1 2 3 4 5 6
```

Welche Levels in einem Faktor existieren erfährt man mit der Funktion `levels()`.

```
# welche Levels hat "v_fac3"?
levels(v_fac3)
```

```
[1] "1" "2" "3" "4" "5" "6"
```

Durch Kombimation mit der Funktion `length()` können wir die Anzahl der Levels erfahren.

```
# wieviele Levels hat "v_fac3"?
length(levels(v_fac3))
```

[2] zur Installation von Zusatzpaketen siehe Abschnitt 1.2.3.14 auf Seite 80

```
[1] 6
```

Die Variable hat 6 Levels.
Welche Level welche Häufigkeit hat erfahren wir mit der Funktion `table()`.

```
# welche Level hat welche Häufigkeit?
table(v_fac2)
```

```
v_fac2
    sehr gut          gut befriedigend  ausreichend   mangelhaft   ungenügend
           2            3            1            2            1            1
```

Mit der Funktion `class()` kann die Datenklasse angezeigt werden.

```
# welcher Datentyp ist Variable "v_fac2"
class(v_fac2)
```

```
[1] "factor"
```

Mit der Funktion `is.factor()` kann zudem geprüft werden, ob ein Objekt ein Faktor ist.

```
# ist "v_fac2" ein Faktor?
is.factor(v_fac2)
```

```
[1] TRUE
```

```
# ist "mymatrix" ein Faktor?
is.factor(mymatrix)
```

```
[1] FALSE
```

1.2.3.5.4 ordinale Faktoren

Die bislang von uns erzeugten Faktoren haben zwar "scheinbar" eine korrekte Levelreihenfolge. Für R handelt es sich dabei aber weiterhin um "einfache" Faktoren, also *nominale* Daten. Um die Faktoren in *ordinale* Faktoren umzuwandeln kann die Funktion `ordered()` verwendet werden. Nutzen wir hierfür unsere Variable `v_fac2` mit den Schulnoten.

```
# wandle in ordinalen Faktor um
ordered(v_fac2)
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend
 [6] mangelhaft   ungenügend   gut          gut          sehr gut
6 Levels: sehr gut < gut < befriedigend < ausreichend < ... < ungenügend
```

Die Levels haben nun eine Rangfolge, welche durch das Kleinerzeichen < dargestellt wird. Leider sind die Levelränge genau verkehrt herum, denn im obigen Falle ist "`sehr gut`" die kleinste und "`ungenügend`" die größte Note. Wir ändern also nochmal die Levelreihenfolge. Hierzu übergeben wir die Levelnamen in einen Vektor, und kehren mit der Funktion `rev()` dessen Reihenfolge um (die Funktion `fct_rev()` aus dem `forcats`-Paket dreht die Levelreihenfolge ebenfalls um, aber derzeit wissen wir ja noch nicht, wie man Zusatzpakete installiert (siehe Abschnitt 1.2.3.14 auf Seite 80)).

```
# zeige Levelnamen an
levels(v_fac2)
```

```
[1] "sehr gut"     "gut"          "befriedigend" "ausreichend"  "mangelhaft"   "ungenügend"
```

```
# kehre den Vektor um
rev(levels(v_fac2))
```

```
[1] "ungenügend"   "mangelhaft"   "ausreichend"  "befriedigend" "gut"          "sehr gut"
```

In Kombination mit `ordered()` erhalten wir so die korrekte ordinale Darstellung.

```
# überführe die ordinale Reihenfolge in Variable 'v_ord'
v_ord <- ordered(factor(v_fac2,
                        levels= rev(levels(v_fac2))))

v_ord
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend
 [6] mangelhaft   ungenügend   gut          gut          sehr gut
6 Levels: ungenügend < mangelhaft < ausreichend < befriedigend < gut < sehr gut
```

Die Funktion `factor()` nimmt zudem den Parameter `ordered=TRUE` entgegen, der direkt ein ordinales Objekt erzeugt:

```
# erzeuge direkt einen ordinalen factor
# mit Parameter 'ordered=TRUE'
v_ord <- factor(v_fac2,
                levels=rev(levels(v_fac2)),
                ordered=TRUE)

v_ord
```

```
 [1] gut          ausreichend  sehr gut     ausreichend  befriedigend
 [6] mangelhaft   ungenügend   gut          gut          sehr gut
6 Levels: ungenügend < mangelhaft < ausreichend < befriedigend < gut < sehr gut
```

1.2.3.5.5 Datenframes

Die Datenklasse Datenframe (Datensatz) ist wohl die wichtigste in R. Datenframes sind ebenso wie Matrizen zweidimensional. Im Unterschied zu einer Matrix können in einem Datenframe unterschiedliche Daten*typen*, also z.B. `numeric`, `character` und `factor`, zusammengeführt werden. Das Datenframe folgt dabei der Logik "ein Fall pro Zeile" (so genanntes *tidy data* Format[3], siehe Abschnitt 1.6 auf Seite 141). Das bedeutet, dass jede Beobachtung (auch Wiederholungen) in einer eigenen Zeile stehen und die jeweiligen Variablen durch die Spalten repräsentiert werden.
Erzeugen wir uns ein paar Beispielvektoren unterschiedlichen Typs mit je 12 Werten.

```
# erzeuge Testvektoren "factor", "char", "numeric", "logical"
geschlecht <- factor(rep(c("m", "w", "d"), 4))
spitzname  <- c("Hasi", "Ide", "Momsi", "Ryu", "Dave", "Zoid", "Adu", "Efi", "Ole", "Ray", "Sam", "Emi")
hausnummer <- 1:12
angemeldet <- c(TRUE, TRUE, FALSE, T, F, F, F, T, T, T, F, T)
```

Alternativ können die Werte auch direkt dem Datenframe übergeben werden.

```
## Wir schreiben die Werte direkt ins Datenframe
## Das Ergebnis ist das selbe
data.frame(geschlecht = factor(rep(c("m", "w", "d"), 4)),
           spitzname  = c("Hasi", "Ide", "Momsi", "Ryu", "Dave", "Zoid", "Adu", "Efi", "Ole", "Ray", "Sam",
           "Emi"),
           hausnummer = 1:12,
           angemeldet = c(TRUE, TRUE, FALSE, T, F, F, F, T, T, T, F, T)
)
```

Aus den Variablen setzen wir nun mit der Funktion `data.frame()` ein Datenframe zusammen.

```
# erzeuge ein Datenframe aus den Testvektoren
data.frame(geschlecht, spitzname, hausnummer, angemeldet)
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
```

Das Datenframe speichern wir in die Variable `MeinDatenframe`.

```
# speicher Datenframe in Variable
MeinDatenframe <- data.frame(geschlecht, spitzname, hausnummer, angemeldet)

# zeige Datenklasse an
class(MeinDatenframe)
```

```
[1] "data.frame"
```

Die Funktion `class()` weist unsere Variable als Datenframe aus.

Ähnlich wie bei Matrizen müssen die Vektoren jeweils die selbe Anzahl an Werten (die selbe *Länge*) besitzen, damit das Datenframe vollständig aufgebaut werden kann. Entfernen wir z.B. einen Wert aus der Reihe `hausnummer`, schlägt der Befehl fehl.

```
# Datenframe, "hausnummer" ist einen Wert kürzer
data.frame(geschlecht, spitzname, hausnummer[-1], angemeldet)
```

```
Fehler in data.frame(geschlecht, spitzname, hausnummer[-1], angemeldet) :
  Argumente implizieren unterschiedliche Anzahl Zeilen: 12, 11
```

1.2.3.5.6 Datenframes: Fälle hinzufügen

Wenn zwei Datenframes mit den `selben` Spaltennamen exisiteren, können diese per `rbind()` zusammengefasst werden. In unserem Beispiel verdoppeln wir einfach unser Datenframe.

```
# füge 2 Datenframes mittels rbind() zusammen
rbind(MeinDatenframe, MeinDatenframe)
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
13          m      Hasi          1       TRUE
14          w       Ide          2       TRUE
15          d     Momsi          3      FALSE
16          m       Ryu          4       TRUE
17          w      Dave          5      FALSE
18          d      Zoid          6      FALSE
19          m       Adu          7      FALSE
20          w       Efi          8       TRUE
21          d       Ole          9       TRUE
22          m       Ray         10       TRUE
23          w       Sam         11      FALSE
24          d       Emi         12       TRUE
```

💀 Achtung, Anfängerfehler! 💀

Es funktioniert `nicht`, wenn eine neue Zeile mit einem Datenvektor übergeben wird, denn in einem Vektor können nur Werte des selben Datentyps vorkommen.

```
# füge einzelne Zeile mit rbind() hinzu
rbind(MeinDatenframe, c("m", "Joe", 99, TRUE))
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
13          m       Joe         99       TRUE
```

Zwar sieht es so aus, als sei die neue Zeile korrekt eingetragen worden, wenn wir jedoch das neue Datenframe in einer Variable abspeichern und die Datenklassen überprüfen, stellen wir fest, was falsch gelaufen ist.

```
# füge einzelne Zeile mit rbind() hinzu
new <- rbind(MeinDatenframe, c("m", "Joe", 99, TRUE))

# überprüfe Datentyp für Spalte "hausnummer"
class(new$hausnummer)
```

```
[1] "character"
```

Der Datentyp in Spalte `hausnummer` ist auf den "kleinsten gemeinsamen Nenner" (`character`) zurückgefallen. Das liegt daran, dass zunächst der Vektor in `c()` auf `character` zurückfällt. Somit sind alle Werte in der `c()`-Funktion vom Typ `character`. Bei `hausnummer` zieht nun dieser neue Wert die gesamte Spalte auf den Typ `character` zurück. Ebenso verhält es sich bei Variable `angemeldet`, die eigentlich mal vom Typ `logical` war.

```
# überprüfe Datentyp für Spalte "geschlecht"
class(new$angemeldet)
```

```
[1] "character"
```

Wenn wir mit der Spalte `hausnummer` rechnen wollen, schlägt dies fehl.

```
# multipliziere Spalte "hausnummer" mit 2
new$hausnummer * 2
```

```
Fehler in new$hausnummer * 2 :
  nicht-numerisches Argument für binären Operator
```

Ein falscher `rbind()`-Befehl kann Ihnen also das gesamte Datenframe "zerschießen". Dies ist ein häufiger Anfängerfehler, **seien Sie sorgsam**, wenn Sie einem Datenframe neue Zeilen hinzufügen!

Um also eine neue Zeile korrekt dem Datenframe hinzuzufügen, muss diese neue Zeile ebenfalls als Datenframe in der selben Struktur (also mit den selben Variablen (Spalten) vorliegen. Die

```r
# neue Zeile
neuezeile <- data.frame( factor("m"), "Joe", 99, TRUE)

# übergebe die Spaltennamen an die neue Zeile
colnames(neuezeile) <- colnames(MeinDatenframe)

# füge zu Datenframe hinzu
new <- rbind(MeinDatenframe, neuezeile)

# anzeigen
new
```

```
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
13          m       Joe         99       TRUE
```

1.2.3.5.7 Datenframes: Spalten hinzufügen

Mit dem Befehl `cbind()` können dem Datenframe neue Spalten hinzugefügt werden. Hierbei ist wichtig, dass die neue Spalte die selbe Anzahl an Werten aufweist wie die anderen Spalten des Datenframes. Wir erzeugen eine neue Variable und fügen diese als neue Spalte dem Datenframe hinzu.

```r
# neue Variable mit 12 Werten
kinder <- c(1, 4, 3, 1, 2, 3, 2, 1, 4, 2, 3, 4)

# neues Datenframe mit dieser Spalte
new <- cbind(MeinDatenframe, kinder)

# anzeigen
new
```

```
   geschlecht spitzname hausnummer angemeldet kinder
1           m      Hasi          1       TRUE      1
2           w       Ide          2       TRUE      4
3           d     Momsi          3      FALSE      3
4           m       Ryu          4       TRUE      1
5           w      Dave          5      FALSE      2
6           d      Zoid          6      FALSE      3
7           m       Adu          7      FALSE      2
8           w       Efi          8       TRUE      1
9           d       Ole          9       TRUE      4
10          m       Ray         10       TRUE      2
11          w       Sam         11      FALSE      3
12          d       Emi         12       TRUE      4
```

Wir könnten aber auch einfach schreiben:

```r
MeinDatenframe$kinder <- kinder
```

1.2.3.5.8 Datenframes zusammenführen

Wenn die Variablen (Spalten) von zwei Datensätzen zusammengefügt werden sollen, kann alternativ auch die Funktion `merge()` verwendet werden. Die Funktion schaut in beiden Datenframes nach einer ID-Variable, anhand derer sie die Daten einander zuordnen kann. In unserem Beispiel kann das die Variable `spitzname` sein. Ein weiteres Test-Datenframe könnte so aussehen:

```
## Wir schreiben die Werte direkt ins Datenframe
## Das Ergebnis ist das selbe
test <- data.frame(kinder = c(1, 4, 3, 1, 2, 3, 2, 1, 4, 2, 3, 4),
           spitzname  = c( "Emi", "Dave", "Sam", "Hasi", "Zoid", "Ray",
                           "Ide", "Momsi", "Ryu", "Adu", "Efi", "Ole")
        )
# anzeigen
test
```

```
   kinder spitzname
1       1       Emi
2       4      Dave
3       3       Sam
4       1      Hasi
5       2      Zoid
6       3       Ray
7       2       Ide
8       1     Momsi
9       4       Ryu
10      2       Adu
11      3       Efi
12      4       Ole
```

Wie Sie sehen können, ist die Reihenfolge der Spitznamen eine andere als in `MeinDatenframe`. Mittels `merge()` können die beiden Datenframes nun dennoch zusammengefügt werden. Über die Parameter `by.x` und `by.y` legen wir fest, anhand welcher Spalten das Zusammenführen ausgerichtet werden soll. In beiden Datensätzen ist dies die Variable `spitzname`. Daher lautet der Funktionsaufruf:

```
## Vereine MeinDatenframe und test
## anhand der Spalte spitzname
merge(MeinDatenframe, test,
      by.x = "spitzname", by.y = "spitzname")
```

```
   spitzname geschlecht hausnummer angemeldet kinder
1        Adu          m          7      FALSE      2
2       Dave          w          5      FALSE      4
3        Efi          w          8       TRUE      3
4        Emi          d         12       TRUE      1
5       Hasi          m          1       TRUE      1
6        Ide          w          2       TRUE      2
7      Momsi          d          3      FALSE      1
8        Ole          d          9       TRUE      4
9        Ray          m         10       TRUE      3
10       Ryu          m          4       TRUE      4
11       Sam          w         11      FALSE      3
12      Zoid          d          6      FALSE      2
```

Die Daten wurden korrekt anhand der Spitznamen zusammengefügt.

1.2.3.5.9 Datenframes: Fälle und Variablen auswählen

Ebenso wie bei der Matrix lassen sich die einzelnen Spalten und Zeilen referenzieren, indem wir in eckigen Klammern die gewünschte Position angeben

```
# Zeige nur die erste Spalte
MeinDatenframe[,1]
```

```
 [1] m w d m w d m w d m w d
Levels: d m w
```

Die einzelnen Spalten des Datenframes lassen sich auch über ihren Namen referenzieren. Hierfür schreiben wir ein Dollarzeichen $ und hängen den Spaltennamen daran.

```
# Zeige nur Spalte "geschlecht"
MeinDatenframe$geschlecht
```

```
 [1] m w d m w d m w d m w d
Levels: d m w
```

```
# Zeige nur Spalte "angemeldet"
MeinDatenframe$angemeldet
```

```
[1]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
```

So können wir auch die jeweiligen Datentypen der Spaltenwerte überprüfen.

```
# welcher Datentyp liegt in Spalte "angemeldet" vor?
class(MeinDatenframe$angemeldet)
```

```
[1] "logical"
```

```
# welcher Datentyp liegt in Spalte "spitzname" vor?
class(MeinDatenframe$spitzname)
```

```
[1] "character"
```

```
# welcher Datentyp liegt in der 1. Spalte vor?
class(MeinDatenframe[, 1])
```

```
[1] "factor"
```

Möchten wir uns die *Fälle* (also die Zeilen) ausgeben lassen, erfolgt dies mit

```
# Zeige Fall Nummer 4
MeinDatenframe[4,]
```

```
  geschlecht spitzname hausnummer angemeldet
4          m       Ryu          4       TRUE
```

oder für Fallserien per

```
# Zeige Fälle Nummer 2 bis 5
MeinDatenframe[2:5,]
```

```
  geschlecht spitzname hausnummer angemeldet
2          w       Ide          2       TRUE
3          d     Momsi          3      FALSE
4          m       Ryu          4       TRUE
5          w      Dave          5      FALSE
```

Wir können auch bedingte Ausgaben erzeugen.

```
# Zeige nur die Fälle mit "hausnummer" kleiner als 5
MeinDatenframe[MeinDatenframe$hausnummer<5, ]
```

```
  geschlecht spitzname hausnummer angemeldet
1          m      Hasi          1       TRUE
2          w       Ide          2       TRUE
3          d     Momsi          3      FALSE
4          m       Ryu          4       TRUE
```

```
# Zeige nur die Fälle mit "angemeldet" = TRUE
MeinDatenframe[MeinDatenframe$angemeldet==T, ]
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
4           m       Ryu          4       TRUE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
12          d       Emi         12       TRUE
```

Dies kann auch verknüpft werden

```
# Zeige nur die Fälle mit "angemeldet" = TRUE
# und "hausnummer" größer 6
MeinDatenframe[(MeinDatenframe$angemeldet==T) & (MeinDatenframe$hausnummer>6), ]
```

```
   geschlecht spitzname hausnummer angemeldet
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
12          d       Emi         12       TRUE
```

 Achtung, Anfängerfehler! 

Beachten Sie, dass wir jedes Mal, wenn wir auf eine Spalte des Datenframes referenzieren möchten, den Datenframe-Namen mit einem Dollarzeichen `$` schreiben müssen. Tun wir das nicht, sucht R nach einer Variable im Workspace, und nutzt dann *deren* Werte. Dies ist ein häufiger Anfängerfehler!

Die ständige Angabe des Datenframes macht die Befehle recht lang.
Mit der Funktion `with()` können wir uns die Referenzierung mit `$` sparen. Wir übergeben der Funktion `with()` unser Datenframe, und sagen dann, was damit getan werden soll.

```
# mit Funktion with() wird es leichter
with(MeinDatenframe, MeinDatenframe[hausnummer>4 & angemeldet==F, ])
```

```
   geschlecht spitzname hausnummer angemeldet
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
11          w       Sam         11      FALSE
```

```
# mit Funktion with() wird es leichter
with(MeinDatenframe, spitzname[hausnummer>4 & angemeldet==F])
```

```
[1] "Dave" "Zoid" "Adu"  "Sam"
```

Das funktioniert in Kombination mit jeder anderen Funktion.

```
# Summe der Hausnummern
with(MeinDatenframe, sum(hausnummer))
```

```
[1] 78
```

```
# Häufigkeit von geschlecht
with(MeinDatenframe, table(geschlecht))
```

```
geschlecht
d m w
4 4 4
```

Ähnlich wie bei Matrizen können wir die Zeilen- und Spaltentitel anpassen. Mit `colnames()` können wir die Spalten umbenennen.

```
# benenne Spalten des Datenframes
colnames(MeinDatenframe) <- c("Sex", "Nickname", "House", "confirmed")

# anzeigen
MeinDatenframe
```

```
   Sex Nickname House confirmed
1    m     Hasi     1      TRUE
2    w      Ide     2      TRUE
3    d    Momsi     3     FALSE
4    m      Ryu     4      TRUE
5    w     Dave     5     FALSE
6    d     Zoid     6     FALSE
7    m      Adu     7     FALSE
8    w      Efi     8      TRUE
9    d      Ole     9      TRUE
10   m      Ray    10      TRUE
11   w      Sam    11     FALSE
12   d      Emi    12      TRUE
```

Dementsprechend können mit `rownames()` die Zeilen umbenannt werden.

```
# benenne Spalten des Datenframes
rownames(MeinDatenframe) <- c("Eins", "Zwei", "Drei", "Vier", "Fünf", "Sechs", "Sieben", "Acht", "Neun",
"Zehn", "Elf", "Zwölf")

# anzeigen
MeinDatenframe
```

```
       Sex Nickname House confirmed
Eins     m     Hasi     1      TRUE
Zwei     w      Ide     2      TRUE
Drei     d    Momsi     3     FALSE
Vier     m      Ryu     4      TRUE
Fünf     w     Dave     5     FALSE
Sechs    d     Zoid     6     FALSE
Sieben   m      Adu     7     FALSE
Acht     w      Efi     8      TRUE
Neun     d      Ole     9      TRUE
Zehn     m      Ray    10      TRUE
Elf      w      Sam    11     FALSE
Zwölf    d      Emi    12      TRUE
```

Das sollte bei einem Datenframe nach dem *Tidy Data* Prinzip (siehe Kapitel 1.6 auf Seite 141) aber niemals notwendig sein.

Mit der Funktion `class()` kann die Datenklasse angezeigt werden.

```
# welcher Datentyp ist "MeinDatenframe"
class(MeinDatenframe)
```

```
[1] "data.frame"
```

Mit der Funktion `is.data.frame()` kann zudem geprüft werden, ob ein Objekt ein Faktor ist.

```
# ist "MeinDatenframe" ein Datenframe?
is.data.frame(MeinDatenframe)
```

```
[1] TRUE
```

```
# ist "mymatrix" ein Datenframe?
is.data.frame(mymatrix)
```

```
[1] FALSE
```

In RStudio werden die Variablen und Datensätze im Datenfenster oben rechts angezeigt (Abbildung 1.27).

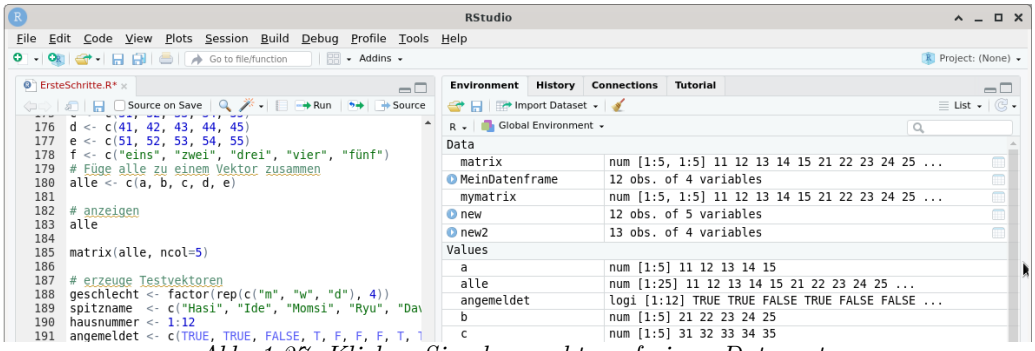
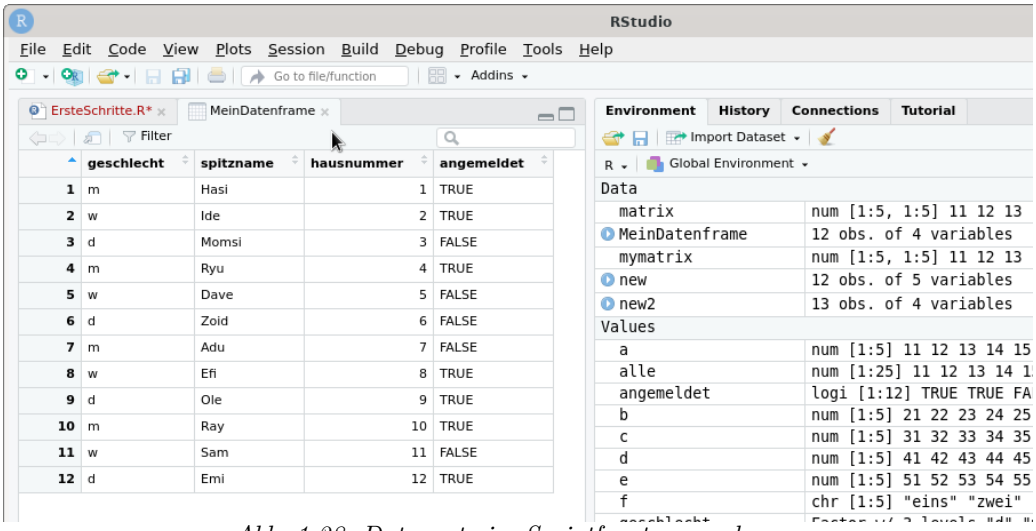


Abb. 1.27: Klicken Sie oben rechts auf einen Datensatz

Wenn Sie hier auf einen Datensatz klicken, z.B. auf MeinDatenframe, so werden Ihnen die *Inhalte* (also die Werte) des Datensatzes im Scriptfenster angezeigt (Abbildung 1.28).



	geschlecht	spitzname	hausnummer	angemeldet
1	m	Hasi	1	TRUE
2	w	Ide	2	TRUE
3	d	Momsi	3	FALSE
4	m	Ryu	4	TRUE
5	w	Dave	5	FALSE
6	d	Zoid	6	FALSE
7	m	Adu	7	FALSE
8	w	Efi	8	TRUE
9	d	Ole	9	TRUE
10	m	Ray	10	TRUE
11	w	Sam	11	FALSE
12	d	Emi	12	TRUE

Abb. 1.28: Datensatz im Scriptfenster anschauen

Wir können mit der Funktion as.data.frame() Objekte in ein Datenframe umwandeln. Schauen wir uns erneut die auf Seite 37 erzeugte Matrix der Beschäftigten in den Pflegeberufen an.

```
# zeige Matrix
Pflegeberufe
```

```
                        1999   2001   2003   2005   2007   2009   2011   2013   2015
Krankenpflegeassistenz 16624  19061  19478  21537  27731  36481  46517  54371  64127
Altenpflegehilfe       55770  52710  49727  45776  48326  47903  47978  48363  49507
Kinderkrankenpflege    47779  48203  48822  48519  49080  49307  48291  48937  48913
Krankenpflege         430983 436767 444783 449355 457322 465446 468192 472580 476416
Altenpflege           109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Mit der Funktion as.data.frame() wandeln wir die Matrix in ein Datenframe um.

```
# wandle Matrix in Datenframe um
Pflegeframe <- as.data.frame(Pflegeberufe)

# anzeigen
Pflegeframe
```

```
                        1999   2001   2003   2005   2007   2009   2011   2013   2015
Krankenpflegeassistenz 16624  19061  19478  21537  27731  36481  46517  54371  64127
Altenpflegehilfe       55770  52710  49727  45776  48326  47903  47978  48363  49507
Kinderkrankenpflege    47779  48203  48822  48519  49080  49307  48291  48937  48913
Krankenpflege         430983 436767 444783 449355 457322 465446 468192 472580 476416
Altenpflege           109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Kommen wir noch einmal auf das Konzept *Tidy Data*[3] zurück (siehe Abschnitt 1.6 auf Seite 141). Ein Datenframe sollte möglichst so aufgebaut sein, dass jeweils ein Fall pro Zeile abgebildet wird. Das ist bei unserem Datenframe `Pflegeframe` aber nicht der Fall.

Ein *Datenframe* würde sich aus der Matrix jede Beschäftigtenzahl einzeln vornehmen, um dann zu fragen "*aus welchem Jahr stammt diese Zahl?*" und "*aus welcher Berufsgruppe stammt diese Zahl?*". Das heisst, es würde hinterher dieser Struktur folgen:

```
    Jahr    Berufsgruppe     Wert
1   1999    Krankenpflege    430983
2   2001    Krankenpflege    436767
3   2001    Altenpflege      124879
4   2003    Altenpflegehilfe 49727
            ( . . . )
```

Man spricht in diesem Zusammenhang von `long table` und `wide table`. Die Matrix der Pflegeberufe stellt dabei die *wide table*, die *breite* Tabelle dar. "Breit" bedeutet, dass die Tabelle, wenn wir ihr nun 10 weitere Jahrgänge hinzufügen würden, immer breiter und breiter werden würde.

Unser angestrebtes Tidy-Data-Datenframe ist vom Typ *long table*, da die Tabelle, wenn wir ihr Daten hinzufügen würden, immer *länger* und länger werden würde.

Wie formen wir unsere Matrix in ein *Tidy Data*-Datenframe, also in eine *long table*, um?

Mit der Funktion `expand.grid()` kann ein Datenframe mit Wertepaaren erzeugt werden. Für unser Beispiel mit den Pflegeberufen brauchen wir im Datenframe je eine Zeile für alle möglichen Kombinationen aus `Jahr` und `Berufsgruppe`. Die Funktion `expand.grid()` erzeugt genau solche Paarungen. Idealerweise sind die benötigten Werte (alle `Jahre` und alle `Berufsgruppen`) als Zeilen- und Spaltennamen in der Matrix `Pflegeberufe` vorhanden.

```
# erzeuge ein Tidy-Data-Dataframe
# mit allen möglichen Kombinationen
# aus Jahren und Berufsgruppen
new <- expand.grid( colnames(Pflegeberufe), rownames(Pflegeberufe))

# anzeigen
new
```

```
   Var1                  Var2
1  1999 Krankenpflegeassistenz
2  2001 Krankenpflegeassistenz
3  2003 Krankenpflegeassistenz
4  2005 Krankenpflegeassistenz
5  2007 Krankenpflegeassistenz
6  2009 Krankenpflegeassistenz
7  2011 Krankenpflegeassistenz
8  2013 Krankenpflegeassistenz
9  2015 Krankenpflegeassistenz
10 1999       Altenpflegehilfe
11 2001       Altenpflegehilfe
12 2003       Altenpflegehilfe
           ( . . . )
```

Mit der Funktion `cbind()` können wir nun die Zahlenwerte aus der Matrix als neue Spalte an das Datenframe anhängen. Dafür müssen die Werte in Form eines Vektors vorliegen.
Um die Matrix als Vektor auszugeben nutzen wir die Funktion `as.vector()`

```
# stelle die Matrix als Vektor dar
as.vector(Pflegeberufe)
```

```
 [1]  16624  55770  47779 430983 109161  19061  52710  48203 436767 124879  19478
[12]  49727  48822 444783 141965  21537  45776  48519 449355 158817  27731  48326
[23]  49080 457322 178902  36481  47903  49307 465446 194195  46517  47978  48291
[34] 468192 208304  54371  48363  48937 472580 227154  64127  49507  48913 476416
[45] 246412
```

Wie Sie sehen, überführt R die Matrix spaltenweise in den Vektor. Für unser neues Datenframe bräuchten wir aber einen zeilenorientierten Vektor, damit er mit der Reihenfolge der Einträge (Paarung aus `Jahr` und `Berufsgruppe`) übereinstimmt. Um einen reihenorientierten Vektor zu erzeugen muss die Matrix mit der Funktion `t()` transpoiniert werden.

```
# stelle die Matrix als Vektor dar
# zeilenorientiert
as.vector(t(Pflegeberufe))
```

```
 [1]  16624  19061  19478  21537  27731  36481  46517  54371  64127  55770  52710
[12]  49727  45776  48326  47903  47978  48363  49507  47779  48203  48822  48519
[23]  49080  49307  48291  48937  48913 430983 436767 444783 449355 457322 465446
[34] 468192 472580 476416 109161 124879 141965 158817 178902 194195 208304 227154
[45] 246412
```

Diesen Vektor fügen wir nun per `cbind()` dem Datenframe als neue Spalte hinzu

```
# füge Spalte hinzu
Pflegeframe <- cbind(new, as.vector(t(Pflegeberufe)))

# benenne die Spalten neue
colnames(Pflegeframe) <- c("Jahr", "Berufsgruppe", "Anzahl")
# zeige an
Pflegeframe
```

```
   Jahr           Berufsgruppe Anzahl
1  1999 Krankenpflegeassistenz  16624
2  2001 Krankenpflegeassistenz  19061
3  2003 Krankenpflegeassistenz  19478
```

```
4  2005 Krankenpflegeassistenz  21537
5  2007 Krankenpflegeassistenz  27731
6  2009 Krankenpflegeassistenz  36481
7  2011 Krankenpflegeassistenz  46517
8  2013 Krankenpflegeassistenz  54371
9  2015 Krankenpflegeassistenz  64127
10 1999       Altenpflegehilfe  55770
11 2001       Altenpflegehilfe  52710
12 2003       Altenpflegehilfe  49727
           ( . . . )
```

Der vollständige Code, ohne Hilfsdatenframe `new`, zur Überführung der Matrix `Pflegeberufe` in das Tidy-Data-Datenframe `Pflegeframe` sieht also so aus:

```
# füge Spalte hinzu
Pflegeframe <- cbind(expand.grid( colnames(Pflegeberufe), rownames(Pflegeberufe)), as.vector(t(Pflegeberufe)))

# benenne die Spalten neue
colnames(Pflegeframe) <- c("Jahr", "Berufsgruppe", "Anzahl")
# zeige erste 12 Fälle an
head(Pflegeframe, 12)
```

```
   Jahr           Berufsgruppe Anzahl
1  1999 Krankenpflegeassistenz  16624
2  2001 Krankenpflegeassistenz  19061
3  2003 Krankenpflegeassistenz  19478
4  2005 Krankenpflegeassistenz  21537
5  2007 Krankenpflegeassistenz  27731
6  2009 Krankenpflegeassistenz  36481
7  2011 Krankenpflegeassistenz  46517
8  2013 Krankenpflegeassistenz  54371
9  2015 Krankenpflegeassistenz  64127
10 1999       Altenpflegehilfe  55770
11 2001       Altenpflegehilfe  52710
12 2003       Altenpflegehilfe  49727
```

1.2.3.5.10 Listen

In der Datenklasse Listen können beliebige Datenobjekte (Vektor, Faktor, Datenframe, Matrizen) zusammengefasst werden. Listen sind also eine Ansammlung an Datenobjekten, ähnlich wie ein Schrank oder ein Koffer, in welchem man "sein Zeug" ablegt. Wir generieren testweise eine Liste aus den Datenobjekten, die wir bislang erzeugt haben. Dies erfolgt in R mit der Funktion `list()`.

```
# erzeuge eine Liste aus den Datenobjekten
# "MeinDatenframe", "mymatrix", "geschlecht" und "logical"
MeineListe <- list(MeinDatenframe, mymatrix, geschlecht, logical)

# anzeigen
MeineListe
```

```
[[1]]
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
```

```
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE

[[2]]
     a  b  c  d  e
I   11 21 31 41 51
II  12 22 32 42 52
III 13 23 33 43 53
IV  14 24 34 44 54
V   15 25 35 45 55

[[3]]
 [1] m w d m w d m w d m w d
Levels: d m w

[[4]]
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Wie Sie sehen, werden die einzelnen Positionen der Datenobjekte durch doppelte eckige Klammer angezeigt (`[[1]]` ist unser Datenframe, `[[2]]` unsere Matrix, usw.) und können über diese auch referenziert werden.

```
1 # zeige Objekt 1 (= unser Datenframe)
2 MeineListe[[1]]
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
```

```
1 # zeige Objekt 3 (= variable "geschlecht")
2 MeineListe[[3]]
```

```
 [1] m w d m w d m w d m w d
Levels: d m w
```

Die Werte der jeweiligen Objekte können anschließend wie gewohnt referenziert werden.

```
1 # zeige Objekt 3, aber nur den 5. Wert
2 MeineListe[[3]][5]
```

```
[1] w
Levels: d m w
```

```
1 # zeige Objekt 2, aber nur die 3. Spalte
2 MeineListe[[2]][, 3]
```

```
  I  II III  IV   V
 31  32  33  34  35
```

1.2.3.6 Fehlende Daten

Fehlende Daten werden in R mit dem Ausdruck `NA` dargestellt. Angenommen Sie erfragen von zehn Personen das Alter, und zwei Personen geben keine Auskunft, dann könnten man das wie folgt in einem Vektor darstellen.

```
# Alter von 10 Personen
alter <- c(28, 30, NA, 21, 27, 43, NA, 35, 22, 18)
# anzeigen
alter
```

```
[1] 28 30 NA 21 27 43 NA 35 22 18
```

Möchten wir nun die Summe der Alterswerte bestimmen, meldet R uns `NA` zurück

```
# Summe der Altersangaben
sum(alter)
```

```
[1] NA
```

Die Berechnung wird nicht durchgeführt, da fehlende Werte in der Reihe vorkommen. Möchte man die Berechnung erzwingen, kann (bei fast jeder R-Funktion) der Parameter `na.rm` (für *NA* ***rem****ove*; entferne NA) auf `TRUE` gesetzt werden. So werden alle fehlenden Daten ignoriert und die Berechnungen nur mit der Restmenge durchgeführt.

```
# Summe der Altersangaben OHNE NA
sum(alter, na.rm=TRUE)
```

```
[1] 224
```

Mit der Funktion `is.na()` kann geprüft werden, ob fehlende Werte enthalten sind

```
# sind Werte in "alter" NA?
is.na(alter)
```

```
[1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

Möchten wir wissen, an welcher Position sich die `NA`-Einträge befinden können wir die Funktion `which()` mit `is.na()` kombinieren

```
# welche Position haben die NAs in "alter"
which(is.na(alter))
```

```
[1] 3 7
```

Sind *keine* NAs enthalten, meldet ein `integer(0)` zurück

```
# NAs in "Pflegeframe"
which(is.na(Pflegeframe))
```

```
integer(0)
```

1.2.3.7 Umgang mit Datensätzen

Mit der Funktion `str()` (für ***str****ucture*) erhalten wir eine erste Beschreibung des Datensatzes.

```
# Übersicht von "MeinDatenframe"
str(MeinDatenframe)
```

```
'data.frame':^^I12 obs. of  4 variables:
 $ geschlecht: Factor w/ 3 levels "d","m","w": 2 3 1 2 3 1 2 3 1 2 ...
 $ spitzname : chr  "Hasi" "Ide" "Momsi" "Ryu" ...
 $ hausnummer: int  1 2 3 4 5 6 7 8 9 10 ...
 $ angemeldet: logi  TRUE TRUE FALSE TRUE FALSE FALSE ...
```

Das Datenframe `MeinDatenframe` enthält 12 Beobachtungen von 4 Variablen. Varibale `geschlecht` ist ein Faktor mit 3 Ausprägungen, usw.

In `RStudio` werden diese Informationen im Datenfenster angezeigt (Abbildung 1.29).

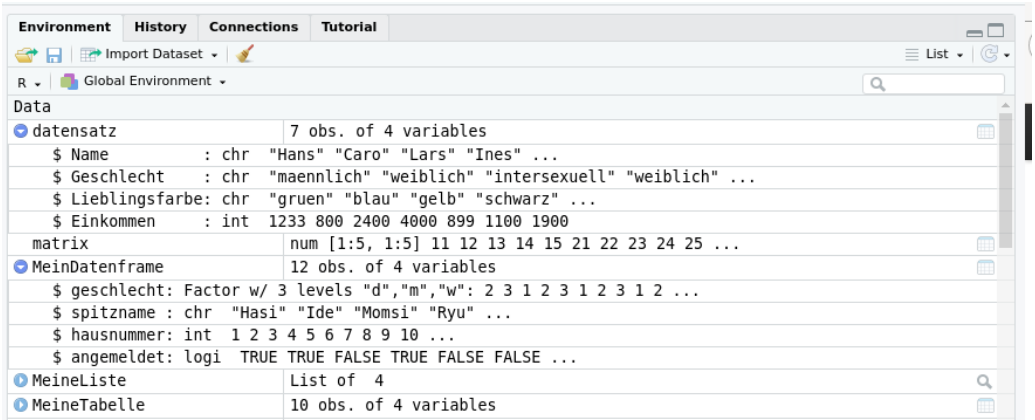


Abb. 1.29: Datenfenster zeigt ***str()****-Ausgabe*

Mit der Funktion `head()` können Sie sich einen weiteren Überblick über den Datensatz verschaffen. Die Funktion `head()` gibt die ersten 6 Zeilen des Datensatzes aus.

```
# zeige die ersten 6 Reihen von "MeinDatenframe"
head(MeinDatenframe)
```

```
  geschlecht spitzname hausnummer angemeldet
1          m      Hasi          1       TRUE
2          w       Ide          2       TRUE
3          d     Momsi          3      FALSE
4          m       Ryu          4       TRUE
5          w      Dave          5      FALSE
6          d      Zoid          6      FALSE
```

Möchten wir mehr als 6 Einträge sehen, können wir dies an `head()` übergeben:

```
# zeige die ersten 10 Reihen von "MeinDatenframe"
head(MeinDatenframe, 10)
```

```
   geschlecht spitzname hausnummer angemeldet
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
4           m       Ryu          4       TRUE
5           w      Dave          5      FALSE
6           d      Zoid          6      FALSE
7           m       Adu          7      FALSE
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
```

Im Gegensatz dazu zeigt `tail()` die letzten Reihen des Datenframes an.

```
# zeige die letzten  5 Reihen von "MeinDatenframe"
tail(MeinDatenframe, 5)
```

```
8           w       Efi          8       TRUE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
11          w       Sam         11      FALSE
12          d       Emi         12       TRUE
```

1.2.3.7.1 Sortieren

Mit der Funktion `sort()` können Datenreihen auf- und absteigend sortiert werden. Standardmäßig wird aufsteigend sortiert

```
# sortiere "hausnummer" von "MeinDatenframe"
# aufsteigend
sort(MeinDatenframe$hausnummer)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Um absteigend zu sortieren, muss der Parameter `decreasing` auf `TRUE` gesetzt werden

```
# sortiere "spitzname" von "MeinDatenframe"
# absteigend
sort(MeinDatenframe$spitzname, decreasing=T)
```

```
 [1] "Zoid"  "Sam"   "Ryu"   "Ray"   "Ole"   "Momsi" "Ide"   "Hasi"
 [9] "Emi"   "Efi"   "Dave"  "Adu"
```

Mit der Funktion `order()` werden die Zeilen*positionen* ausgegeben, an deren Stelle sich die von `sort()` sortierten Daten befinden.

```
# sortiere "spitznamen" von "MeinDatenframe"
# aufsteigend, und zeige Positionen an
order(MeinDatenframe$spitzname)
```

```
[1]  7  5  8 12  1  2  3  9 10  4 11  6
```

Das kann man nun verwenden, um den kompletten Datensatz in dieser Ordnung anzuzeigen. Wir können ja durch Anhängen von eckigen Klammern die gewünschten Zeilen und Spalten des Datenframes referenzieren. Also übergeben wir die Positionsreihenfolge von `order()` als Argument in die eckige Klammer.

```
# sortiere "spitznamen" von "MeinDatenframe"
# aufsteigend, zeige ganzes Datenframe an
MeinDatenframe[ order(MeinDatenframe$spitzname), ]
```

```
   geschlecht spitzname hausnummer angemeldet
7           m       Adu          7      FALSE
5           w      Dave          5      FALSE
8           w       Efi          8       TRUE
12          d       Emi         12       TRUE
1           m      Hasi          1       TRUE
2           w       Ide          2       TRUE
3           d     Momsi          3      FALSE
9           d       Ole          9       TRUE
10          m       Ray         10       TRUE
4           m       Ryu          4       TRUE
11          w       Sam         11      FALSE
6           d      Zoid          6      FALSE
```

Oder andersherum

```
# sortiere "spitznamen" von "MeinDatenframe"
# absteigend, zeige ganzes Datenframe an
MeinDatenframe[ order(MeinDatenframe$spitzname, decreasing=T), ]
```

```
   geschlecht spitzname hausnummer angemeldet
6           d      Zoid          6      FALSE
11          w       Sam         11      FALSE
4           m       Ryu          4       TRUE
10          m       Ray         10       TRUE
9           d       Ole          9       TRUE
3           d     Momsi          3      FALSE
2           w       Ide          2       TRUE
1           m      Hasi          1       TRUE
12          d       Emi         12       TRUE
8           w       Efi          8       TRUE
5           w      Dave          5      FALSE
7           m       Adu          7      FALSE
```

1.2.3.7.2 Teilgruppen erstellen

Mit der Funktion `subset()` können Teilgruppen aus einem Datenframe erstellt werden.

```
# erstelle eine Teilgruppe aus MeinDatenframe
# für "geschlecht" = d
subset(MeinDatenframe, geschlecht=="d")
```

```
   geschlecht spitzname hausnummer angemeldet
3           d     Momsi          3      FALSE
6           d      Zoid          6      FALSE
9           d       Ole          9       TRUE
12          d       Emi         12       TRUE
```

```
# nur männliche Probanden
subset(MeinDatenframe, geschlecht=="m")
```

```
       geschlecht spitzname hausnummer angemeldet
Eins            m      Hasi          1       TRUE
Vier            m       Ryu          4       TRUE
Sieben          m       Adu          7      FALSE
Zehn            m       Ray         10       TRUE
```

```
# nur solche mit hausnummer kleiner 6
subset(MeinDatenframe, hausnummer < 6)
```

```
     geschlecht spitzname hausnummer angemeldet
Eins          m      Hasi          1       TRUE
Zwei          w       Ide          2       TRUE
Drei          d     Momsi          3      FALSE
Vier          m       Ryu          4       TRUE
Fünf          w      Dave          5      FALSE
```

Hierbei können mehrere Bedingungen mittels `&` und `|` verknüpft werden.

```
# mehr Bedingungen
subset(MeinDatenframe, (geschlecht=="m" | geschlecht=="d") &
                        angemeldet==FALSE & hausnummer > 3)
```

```
       geschlecht spitzname hausnummer angemeldet
Sechs           d      Zoid          6      FALSE
Sieben          m       Adu          7      FALSE
```

Über den Parameter `select` kann angegeben werden, welche Spalten übernommen werden sollen.

```
# erstelle eine Teilgruppe aus MeinDatenframe
# für "geschlecht" = d
# aber nur Spalte "spitzname" und "hausnummer"
subset(MeinDatenframe, geschlecht=="d", select = c(spitzname, hausnummer))
```

```
    spitzname hausnummer
3      Momsi          3
6       Zoid          6
9        Ole          9
12       Emi         12
```

Eine andere Möglichkeit zur Erstellung von Teilgruppen kann über die Referenzierung der Zeilen und Spalten des Datenframes (eckige Klammern) erfolgen

```
# erstelle eine Teilgruppe aus MeinDatenframe
# für "geschlecht" = w
MeinDatenframe[ MeinDatenframe$geschlecht == "w", ]
```

```
   geschlecht spitzname hausnummer angemeldet
2           w       Ide          2       TRUE
5           w      Dave          5      FALSE
8           w       Efi          8       TRUE
11          w       Sam         11      FALSE
```

Auf diese Weise können die Bedingungen auch mit einem `&` kombiniert werden

```
# erstelle eine Teilgruppe aus MeinDatenframe
# für "geschlecht" = w und "hausnummer" größer 5
MeinDatenframe[ (MeinDatenframe$geschlecht == "w") & (MeinDatenframe$hausnummer > 5) , ]
```

```
   geschlecht spitzname hausnummer angemeldet
8           w       Efi          8       TRUE
11          w       Sam         11      FALSE
```

Über Angabe der Spaltennummern können wir das Teilset noch weiter zusammenschrumpfen lassen

```
# erstelle eine Teilgruppe aus MeinDatenframe
# für "geschlecht" = w und "angemeldet" = TRUE
# aber nur Spalte "spitzname" und "hausnummer"
MeinDatenframe[(MeinDatenframe$geschlecht == "w") & (MeinDatenframe$angemeldet == T), c(2,3)]
```

```
  spitzname hausnummer
2       Ide          2
8       Efi          8
```

Möchte man auf die Referenzierung per Dollarzeichen `$` verzichten, kann die Funktion `with()` verwendet werden.

```
# with() macht es leichter
with(MeinDatenframe, MeinDatenframe[(geschlecht == "w") & (angemeldet == T), c(2,3)])
```

```
  spitzname hausnummer
2       Ide          2
8       Efi          8
```

1.2.3.8 Klassen für Variablen bilden (klassieren)

Angenommen, eine Variable enthält Werte zwischen 0 und 500, und wir möchten diese Werte in die Gruppen “0-70”, “71-200”, “201-400”, “>400” klassieren.
Erstellen wir zunächst zufällige Zahlen zwischen 0 und 500.

```r
# lade tidyverse
library(tidyverse)
# erzeuge 200 Zufallszahlen von 0 bis 500
dummy <- sample(0:500, 200)
# bzw. direkt als Dataframe
dummy <- data.frame(x = sample(0:500, 200))
```

Nun erzeugen wir die neue Variable `xKAT`, in welcher die Klassierung angegeben werden soll. Dies kann auf mehreren Wegen erfolgen.

A) Klassieren "von Hand"

Zunächst erzeugen wir die Kategorien "von Hand".

```r
# wir bilden Kategorien
#       "0-70"
#       "71-200"
#       "201-400"
#       "> 400"
# und speichern das in die neue Variable xKAT
dummy$xKAT[dummy$x <71] <- "0-70"
dummy$xKAT[dummy$x <201 & dummy$x > 70] <- "71-200"
dummy$xKAT[dummy$x <401 & dummy$x > 200] <- "201-400"
dummy$xKAT[dummy$x >400] <- "größer 400"
```

Anschließend wandeln wir die neue Variable in einen ordinalen Faktor mit korrekter Levelreihenfolge.

```r
dummy$xKAT <- factor(dummy$xKAT, levels=c("0-70",
                                          "71-200",
                                          "201-400",
                                          "größer 400"),
                     ordered=TRUE)
head(dummy$xKAT)
```

```
[1] 201-400    201-400    201-400    größer 400 201-400
[6] 201-400
Levels: 0-70 < 71-200 < 201-400 < größer 400
```

B) Klassieren mittels `cut()`

Mit Hilfe der `cut()`-Funktion kann die Klassierung ebenfalls vorgenommen werden. Sie unterteilt kontinuierliche (numerische) Werte in Kategorien oder Intervalle. Über den Parameter `breaks` können die Klassengrenzen festgelegt werden.

```r
  # bilde Klassen mittels "cut()"
  cut(dummy$x, breaks=c(0, 70, 200, 400, Inf))
```

```
  [1] (200,400] (400,Inf] (70,200]  (400,Inf] (70,200]  (0,70]    (200,400] (70,200]  (200,400]
 [10] (70,200]  (70,200]  (200,400] (70,200]  (400,Inf] (70,200]  (200,400] (400,Inf] (0,70]
 [19] (70,200]  (0,70]    (200,400] (0,70]    (200,400] (200,400] (0,70]    (200,400] (70,200]
 [28] (200,400] (400,Inf] (70,200]  (0,70]    (200,400] (200,400] (400,Inf] (400,Inf] (70,200]
 [37] (70,200]  (70,200]  (70,200]  (200,400] (70,200]  (0,70]    (200,400] (70,200]  (200,400]
 [46] (200,400] (200,400] (70,200]  (200,400] (200,400] (200,400] (70,200]  (70,200]  (400,Inf]
 [55] (0,70]    (200,400] (70,200]  (400,Inf] (400,Inf] (0,70]    (0,70]    (200,400] (70,200]
 [64] (400,Inf] (400,Inf] (0,70]    (70,200]  (200,400] (0,70]    (200,400] (70,200]  (0,70]
```

```
  [73] (70,200]  (200,400] (0,70]    (70,200]  (400,Inf] (70,200]  (200,400] (200,400] (70,200]
  [82] (70,200]  (0,70]    (200,400] (400,Inf] (0,70]    (0,70]    (200,400] (200,400] (200,400]
  [91] (0,70]    (200,400] (400,Inf] (200,400] (200,400] (200,400] (200,400] (0,70]    (200,400]
 [100] (400,Inf] (400,Inf] (200,400] (200,400] (200,400] (200,400] (200,400] (70,200]  (0,70]
 [109] (200,400] (400,Inf] (400,Inf] (0,70]    (0,70]    (200,400] (200,400] (400,Inf] (400,Inf]
 [118] (400,Inf] (200,400] (200,400] (200,400] (200,400] (0,70]    (200,400] (200,400] (200,400]
 [127] (400,Inf] (200,400] (200,400] (200,400] (400,Inf] (200,400] (70,200]  (200,400] (70,200]
 [136] (200,400] (200,400] (70,200]  (70,200]  (70,200]  (400,Inf] (400,Inf] (70,200]  (70,200]
 [145] (200,400] (70,200]  (200,400] (400,Inf] (0,70]    (0,70]    (400,Inf] (200,400] (400,Inf]
 [154] (400,Inf] (200,400] (400,Inf] (0,70]    (0,70]    (0,70]    (400,Inf] (200,400] (70,200]
 [163] (200,400] (200,400] (0,70]    (400,Inf] (200,400] (70,200]  (70,200]  (70,200]  (70,200]
 [172] (200,400] (0,70]    (70,200]  (0,70]    (0,70]    (200,400] (200,400] (70,200]  (200,400]
 [181] (400,Inf] (200,400] (0,70]    (70,200]  (200,400] (400,Inf] (0,70]    (400,Inf] (70,200]
 [190] (70,200]  (200,400] (70,200]  (70,200]  (200,400] (400,Inf] (0,70]    (400,Inf] (70,200]
 [199] (200,400] (400,Inf]
Levels: (0,70] (70,200] (200,400] (400,Inf]
```

Die erzeugten Werte werden als Factor wiedergegeben. Die Factorlevels entsprechen dabei den Kategorien (Klassen). Die `cut()`-Funktion benennt die Klassen mit der üblichen (]-Notation. Die eckigen Klammern geben an, dass die nebenstehende Zahl in der Klasse eingeschlossen wird. Runde Klammern hingegen schließen die nebenstehende Zahl aus.

- Eine Klasse mit der Notation $(10, 20]$ schließt die 10 **nicht** mit ein, sondern jede höhere Zahl. Sie endet bei der Zahl 20, wobei 20 in der Klasse enthalten ist.
- Eine Klasse mit der Notation $[10, 20)$ schließt die 10 mit ein. Sie endet bei der Zahl 20, wobei 20 **nicht** in der Klasse enthalten ist, sondern zur nächst-höheren Klasse zählt.

Über den Parameter `right` kann bestimmt werden, ob die Zahlen an der rechten Klassengrenze mit eingeschlossen werden sollen, oder nicht.

```
# erzeuge Dummy-Werte
x <- 1:20

# cut-Beispiele für Parameter "right"
# die rechte Grenze wird mit eingeschlossen. Entspricht "(]"
cut(x, breaks=c(0,5,10,15,20))
```

```
 [1] (0,5]   (0,5]   (0,5]   (0,5]   (0,5]   (5,10]  (5,10]  (5,10]  (5,10]  (5,10]  (10,15] (10,15]
[13] (10,15] (10,15] (10,15] (15,20] (15,20] (15,20] (15,20] (15,20]
Levels: (0,5] (5,10] (10,15] (15,20]
```

```
  # die rechte Grenze wird NICHT mit eingeschlossen. Entspricht "[)"
  cut(x, breaks=c(0,5,10,15,20),
      right=FALSE)
```

```
 [1] [0,5)   [0,5)   [0,5)   [0,5)   [5,10)  [5,10)  [5,10)  [5,10)  [5,10)  [10,15) [10,15) [10,15)
[13] [10,15) [10,15) [15,20) [15,20) [15,20) [15,20) <NA>
Levels: [0,5) [5,10) [10,15) [15,20)
```

Über den Parameter `labels` können eigene Bezeichnungen für die Klassen gesetzt werden.

```
# die rechte Grenze wird NICHT mit eingeschlossen. Entspricht "[)"
cut(x, breaks=c(0,5,10,15,20),
labels=c("0 bis 5", "6 bis 10", "11 bis 15", "16 bis 20"))
```

```
 [1] 0 bis 5   0 bis 5   0 bis 5   0 bis 5   0 bis 5   6 bis 10  6 bis 10  6 bis 10  6 bis 10
[10] 6 bis 10  11 bis 15 11 bis 15 11 bis 15 11 bis 15 11 bis 15 16 bis 20 16 bis 20 16 bis 20
[19] 16 bis 20 16 bis 20
Levels: 0 bis 5 6 bis 10 11 bis 15 16 bis 20
```

Auch hier kann man sich etwas Schreibarbeit sparen. Wir speichern die Klassengrenzen in das Objekt `mybreaks`. Innerhalb des `labels`-Parameter greifen wir darauf zurück. Für die unteren Grenzen nehmen wir die `mybreaks`-Einträge (ohne den letzten) und addieren jeweils 1. Für die oberen Grenzen nehmen wir `mybreaks` ohne den ersten Eintrag. Mittels `paste()` können die Grenzwerte aneinander geklebt werden.

```
# die rechte Grenze wird NICHT mit eingeschlossen. Entspricht "[)"
# erzeuge eigenes Objekt für "breaks"
mybreaks <- c(0,5,10,15,20)

# eigene Klassenbezeichnung
cut(x, breaks=mybreaks,
       labels = paste(mybreaks[-length(mybreaks)]+1, "bis", mybreaks[-1]))
```

```
 [1] 1 bis 5   1 bis 5   1 bis 5   1 bis 5   1 bis 5   6 bis 10  6 bis 10  6 bis 10  6 bis 10
[10] 6 bis 10  11 bis 15 11 bis 15 11 bis 15 11 bis 15 11 bis 15 16 bis 20 16 bis 20 16 bis 20
[19] 16 bis 20 16 bis 20
Levels: 1 bis 5 6 bis 10 11 bis 15 16 bis 20
```

Über den Parameter `ordered_result` kann die Ausgabe als ordinaler Factor erfolgen.

```
cut(x, breaks=mybreaks,
       labels = paste(mybreaks[-length(mybreaks)]+1, "bis", mybreaks[-1]),
       ordered_result = TRUE)
```

```
 [1] 1 bis 5   1 bis 5   1 bis 5   1 bis 5   1 bis 5   6 bis 10  6 bis 10  6 bis 10  6 bis 10
[10] 6 bis 10  11 bis 15 11 bis 15 11 bis 15 11 bis 15 11 bis 15 16 bis 20 16 bis 20 16 bis 20
[19] 16 bis 20 16 bis 20
Levels: 1 bis 5 < 6 bis 10 < 11 bis 15 < 16 bis 20
```

1.2.3.9 Pipe

Ein wichtiger Operator für den Umgang mit Datensätzen ist die `Pipe`. Diese ist fester Bestandteil des Tidyverse (siehe Abschnitt 1.6.2) und hat sich dort so bewährt, dass sie seit R Version 4.1 ebenfalls im "Standard"-R implementiert ist.
Die `Pipe` erlaubt es, *Datenströme* weiterzuleiten.
Hierfür wurde ein eigener Operator implementiert, die Zeichenkette `|>`
Sie bedeutet so viel wie "*und dann*".
Zur Eklärung sei der "klassische Weg"der Arbeitsschritte in R aufgezeigt:

```
# wir machen etwas, und speichern es in ein Objekt
habe.getan <- mache.etwas(Datensatz)

# dann machen wir etwas weiteres,
# und speichern wieder in ein Objekt
habe.weiteres.getan <- mache.weiteres(habe.getan)

# und kommen schließlich zum Endergebnis
endergebnis <- mache.noch.letzte.Sache(habe.weiteres.getan)
```

Mit Einsatz der `Pipe` wird dieser Prozess quasi umgekehrt in

```
# Speichere Endergebnis
endergebnis <- Datensatz |>
                gruppiere.nach.geschlecht() |>
                  sortiere Alter aufsteigend() |>
                    nimm die letzten 4 Werte()
```

Die liest sich in etwa so:

1. "Speichere etwas ins Objekt `endergebnis`, und das geht so...
2. nimm den `Datensatz` *und dann*
3. gruppiere die Daten nach `geschlecht` *und dann*
4. sortiere nach `Alter` aufsteigend *und dann*
5. nimm die letzten 4 Werte."

Die `Pipe` reicht das jeweilige Ergebnis (den *Datenstrom*) an die nächste Code-Zeile weiter. Erst wenn die letzte Zeile durchgelaufen ist, wird das Resultat in `endergebnis` gespeichert.
Dies liest sich zu Beginn evtl. etwas ungewohnt, aber Sie können erkennen, dass die Befehle so wesentlich übersichtlicher und die einzelnen *Manipulations*schritte nachvollziehbarer geworden sind. Auch kann man sich diese Art der "Grammatik" relativ leicht merken.

Standardmäßig übergibt die Pipe das Objekt auf der linken Seite an das erste Argument der Funktion auf der rechten Seite, so dass `x |> f(y)` äquivalent ist zu `f(x, y)`, und `x |> f(y) |> g(z)` ist äquivalent zu `g(f(x, y), z)`.

Mehr über die "Original"-Pipe `%>%` des Tidyverse sowie zu den Unterschieden zur Standard-Pipe `|>` finden Sie im Abschnitt **??** auf Seite **??**.

In RStudio können Sie einstellen, ob Sie mit der Tastenkombination `[STRG]` + `[ALT]` + `[M]` die Tidyverse-Pipe oder die Standard-Pipe erzeugen möchten.
Wählen Sie hierzu im Menü "Tools" → "Global Options" → "Code" aus, siehe Abbildung 1.30.

1.2.3.10 Daten laden und speichern

In R können Daten leicht geladen und gespeichert werden. Eine R-Datei hat die Dateiendung `.RData`. In der Datei können "beliebig" viele Objekte (Variablen, Datenframes, usw.) enthalten sein.

Mit der Funktion `save.image()` werden alle Objekte im Speicher, also die aktuelle "Worksession", als "unsichtbare" Datei `.RData` ins aktuelle Arbeitsverzeichnis geschrieben.

```
# speichere die aktuelle Sitzung
save.image()
```

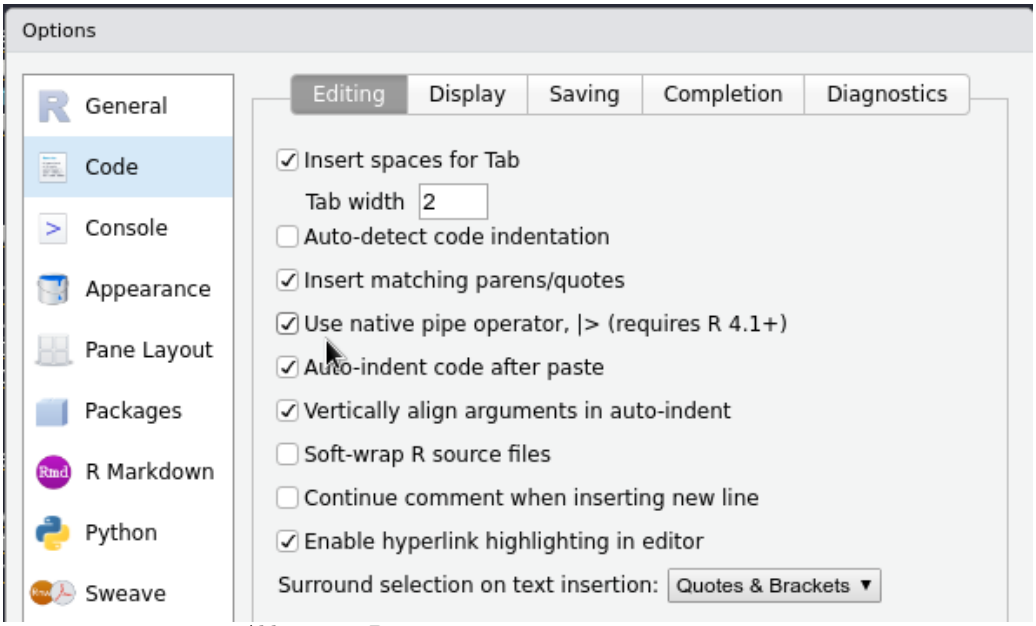


Abb. 1.30: Datei Tastenkombination für Pipe

In RStudio können Sie im Ausgabefenster (unten rechts) auf den Reiter Files klicken. Sie sehen Ihr aktuelles Arbeitsverzeichnis, und darin die -Datei .RData (siehe Abbildung 1.31). Das ist übrigens die selbe Datei, die angelegt wird, wenn Sie RStudio beenden, und "speichere Arbeitsspeicher" klicken. Das heisst, dass diese Datei unter Umständen bald wieder überschrieben wird.

Der Funktion save.image() kann auch ein gewünschter Dateiname übergeben werden.

```
# speichere die aktuelle Sitzung in Datei "MeineSitzung.RData"
save.image("MeineSitzung.RData")
```

Die Datei MeineSitzung.RData wird in RStudio im Ausgabefenster angezeigt (siehe Abbildung 1.32).

Wenn nur einzelne Objekte gespeichert werden sollen, wird die save()-Funktion angewandt. Mit dem file-Parameter wird die gewünschte Zieldatei angegeben.

```
# speichere nur die Variable "sequenz"
# in Datei "Sequenz.RData"
save(sequenz, file="Sequenz.RData")
```

```
# speichere die Objekte "sequenz" "normal" und "geschlecht"
# in Datei "MeineVariablen.RData"
save(sequenz, normal, geschlecht, file="MeineVariablen.RData")
```

Daten können mit dem load()-Befehl geladen werden.

```
# lade Datei "MeineVariablen.RData"
load("MeineVariablen.RData")
```

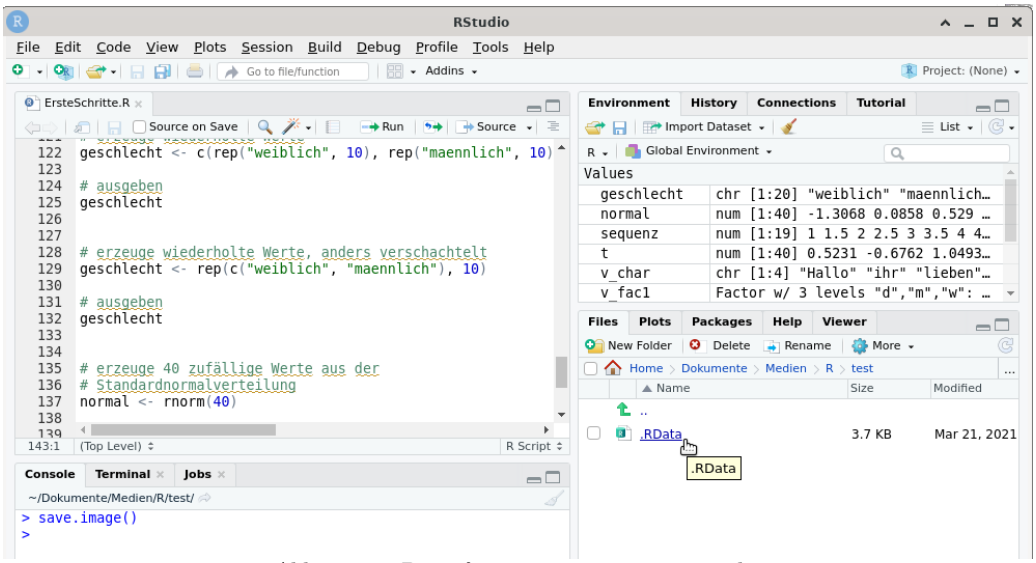


Abb. 1.31: Dateifenster `Files` unten rechts



Abb. 1.32: Datei `MeineSitzung.RData`

In `RStudio` kann im Ausgabefenster unter dem Reiter `Files` einfach auf eine -Datei geklickt werden, um diese zu laden (Abbildung 1.33).

Wenn die Datei im Internet liegt, muss sie mit Hilfe der `url()`-Funktion geladen werden

```
# Test-Datensatz aus dem Internet laden
# (dieser enthält das Datenframe "ordinalSample")
load(url("https://www.produnis.de/R/data/OrdinalSample.RData"))

# Daten anschauen mit head(ordinalSample)
head(ordinalSample)
```

1.2.3.11 Arbeitsverzeichnis

Das aktuelle Arbeitsverzeichnis kann per `getwd()` angezeigt werden.

```
# Arbeitsverzeichnis ausgeben
getwd()
```

```
[1] "/home/produnis/Dokumente/Medien/R/test"
```

Mit dem Befehl dir() werden die Dateien des aktuellen Arbeitsverzeichnisses angezeigt.

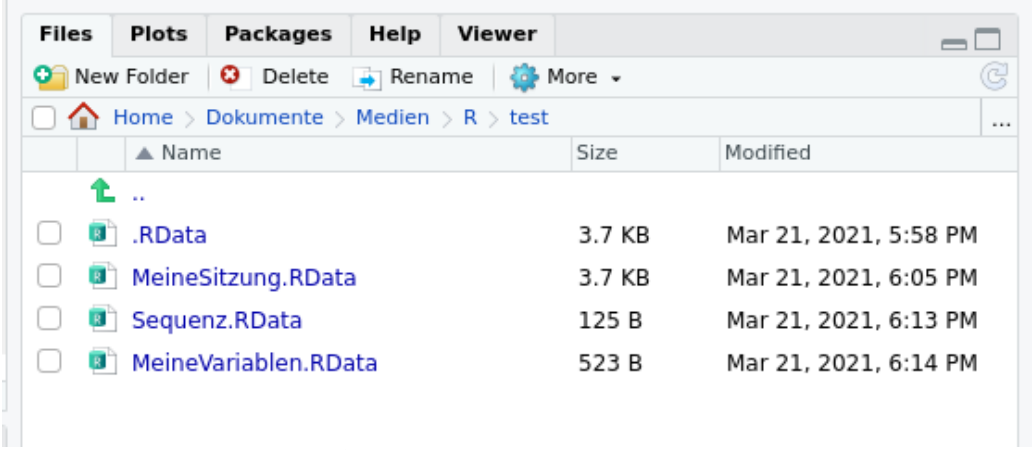


Abb. 1.33: verfügbare R-Dateien

```
# Inhalt des Arbeitsverzeichnis ausgeben
dir()
```

```
[1] "MeineSitzung.RData"   "MeineVariablen.RData" "Sequenz.RData"
```

Mit dem Befehl `setwd()` kann das aktuelle Arbeitsverzeichnis gewechselt werden.

```
# Arbeitsverzeichnis wechseln
setwd("/tmp/R/")
```

Auf Dauer kann es einfacher sein, ein eigenes Arbeitsverzeichnis beim Startvorgang einzustellen (siehe Abschnitt 1.2.2.1 auf Seite 5).

1.2.3.12 Daten importieren

In R können Daten auf vielfältige Weise importiert werden. Das Grundproblem beim "Tausch" von Daten zwischen verschiedenen Anwendungen (z.B. `SPSS` oder `Statistica`) liegt im verwendeten Format. So können native `SPSS`-Dateien nur mit einem Zusatzpaket importiert werden, für `Statistica` gibt es ein solches Zusatzpaket (derzeit) nicht. In beiden Fällen ist es aber möglich, die Daten über das `CSV`-Format zu tauschen. Hierfür speichert man in `SPSS` den Datensatz als `CSV`-Datei ab und liest diese in R ein.

☠ Achtung, Anfängerfehler! ☠

Bedenken Sie ebenfalls, dass R Dezimalstellen mit einem Punkt angibt, und nicht mit einem Komma. So kann es zu Fehlern kommen, wenn die zu importierenden Werte nicht ebenfalls die Dezimalstellen mit einem Punkt angeben! Dies ist ein weiterer häufiger Anfängerfehler!

1.2.3.12.1 Import aus Textdateien

In einer Textdatei (mit Dateiendung `.txt`) liegen unsere Daten wie folgt vor:

```
Geschlecht Alter Gewicht Groesse
m 28 80 170
w 18 55 174
w 25 74 183
m 29 101 190
m 21 84 185
w 19 74 178
w 27 65 169
w 26 56 163
m 31 88 189
m 22 78 184
```

Angenommen, diese Textdatei hieße `Datentabelle.txt`, und läge in unserem Arbeitsverzeichnis, dann können wir die Daten mit der Funktion `read.table()` in R einlesen.

```
1  # importiere Daten aus "Datentabelle.txt"
2  MeineTabelle <- read.table("Datentabelle.txt", header=TRUE)
3
4  # ausgeben
5  MeineTabelle
```

```
   Geschlecht Alter Gewicht Groesse
1           m    28      80     170
2           w    18      55     174
3           w    25      74     183
4           m    29     101     190
5           m    21      84     185
6           w    19      74     178
7           w    27      65     169
8           w    26      56     163
9           m    31      88     189
10          m    22      78     184
```

Liegt die Datei im Internet, kann die Adresse mit der Funktion `url()` übergeben werden.

```
1  # importiere Daten aus "Datentabelle.txt"
2  MeineTabelle <- read.table(url("https://www.produnis.de/R/Datentabelle.txt"), header=TRUE)
3
4  # ausgeben
5  MeineTabelle
```

```
   Geschlecht Alter Gewicht Groesse
1           m    28      80     170
2           w    18      55     174
3           w    25      74     183
4           m    29     101     190
5           m    21      84     185
6           w    19      74     178
7           w    27      65     169
8           w    26      56     163
9           m    31      88     189
10          m    22      78     184
```

Der Parameter `header=TRUE` gibt an, dass in der ersten Zeile der Textdatei die Spaltennamen (Variablennamen) angegeben sind.
Da wir sonst keine Parameter übergeben haben, geht R davon aus, dass die Werte durch ein Leerzeichen getrennt sind, was in unserem Beispiel auch stimmt. Sollte die Tabelle in diesem Format vorliegen (die Daten sind mit einem Komma getrennt) …

```
Geschlecht,Alter,Gewicht,Groesse
m,28,80,170
w,18,55,174
w,25,74,183
m,29,101,190
m,21,84,185
w,19,74,178
w,27,65,169
w,26,56,163
m,31,88,189
m,22,78,184
```

...dann müssen wir mit der Option `sep` (für *Daten**sep**arator*) angeben, dass die Daten mit einem Komma getrennt sind.

```
# importiere Daten aus "Datentabelle.txt"
# Daten sind mit Komma getrennt
MeineTabelle <- read.table("Datentabelle.txt", sep=",", header=TRUE)
```

In beiden Fällen wurde die Texttabelle als Datenframe in R importiert. Die Funktion `class()` bestätigt uns dies.

```
# welche Datenklasse wurde erzeugt?
class(MeineTabelle)
```

```
[1] "data.frame"
```

Aber welche Datentypen wurden in den *Spalten* erzeugt?

```
# welcher Datentyp liegt in Spalte 1 vor?
class(MeineTabelle[,1])
```

```
[1] "character"
```

```
# welcher Datentyp liegt in Spalte "Alter" vor?
class(MeineTabelle$Alter)
```

```
[1] "integer"
```

Die Spalte `Geschlecht` wurde als `character` angelegt, die Variable `Alter` ist `numerisch`. Möchten wir `Geschlecht` in einen Faktor umwandeln, "überschreiben" wir einfach die Spalte entsprechend:

```
# "Geschlecht" soll factor werden
MeineTabelle$Geschlecht <- factor(MeineTabelle$Geschlecht)

# überprüfen, ob es geklappt hat
class(MeineTabelle$Geschlecht)
```

```
[1] "factor"
```

1.2.3.12.2 Import aus CSV-Datei

Das CSV-Format (für **c**omma-**s**eparated-**v**alues) wird von vielen Anwendungen unterstützt. So können auch Tabellen, die in `LibreOffice` oder `Excel` angelegt wurden, als CSV-Datei gespeichert werden. Wenn Sie eine Datei im CSV-Format speichern, fragt Sie Ihre Anwendung nach dem gewünschten *Datenseparator*, also nach dem Zeichen, durch welches die Werte voneinander getrennt sind. Standardmäßig wird hier ein Komma "`,`" verwendet (wie der Name **C**SV vermuten lässt, und was zusätzlich sicherstellt, dass die zu importierenden Daten ihre Dezimalstellen mit einem **Punkt** abbilden, denn sonst würde nach jedem Dezimalkomma ein neuer Wert beginnen!) oder ein Semikolon "`;`" . Ich persönlich verwende gerne das Semikolon.
Der Import in R erfolgt ebenfalls über die Funktion `read.table()`. Neben dem Parameter `header` muss ihr der eingestellte *Datenseparator* übergeben werden.
Angenommen, die CSV-Datei hieße `DieDaten.csv`, läge in unserem Arbeitsverzeichnis und hätte den Datenseparator "`;`", dann erfolgt der Import per `read.table()` mit dem Befehl:

```
# importiere Daten aus "DieDaten.csv"
NeueTabelle <- read.table("DieDaten.csv", sep=";", header=TRUE)
```

Liegt die Datei im Internet, kann die Adresse per `url()` Funktion übergeben werden.

```
# importiere Daten aus dem Internet
NeueTabelle <- read.table(url("http://www.produnis.de/R/DieDaten.csv"), sep=";", header=TRUE)

# anzeigen
NeueTabelle
```

```
    Name   Geschlecht Lieblingsfarbe Einkommen
1   Hans    maennlich          gruen      1233
2   Caro     weiblich           blau       800
3   Lars intersexuell           gelb      2400
4   Ines     weiblich         schwarz      4000
5 Samira     weiblich           gelb       899
6  Peter    maennlich          gruen      1100
7  Sarah     weiblich           blau      1900
```

Sollte die CSV-Datei mit dem Datenseparator `TAB` (für *Tabulator*) erstellt worden sein, lautet der Befehl:

```
# importiere Daten
# Daten sind mit TAB separiert!
NeueTabelle <- read.table("DieDaten.csv", sep="\t", header=TRUE)
```

(Separieren Sie Ihre Daten möglichst niemals mittels Tabulator, da es hier zu ungewollten Effekten kommen kann.)
Sollte `read.table)()` keine brauchbaren Daten erzeugen (z.B. wegen Sonderzeichen in den Daten oder wenn die Daten per `TAB` separiert wurden), kann auf die Funktion `read.csv()` zurückgegriffen werden. Letztendlich ruft diese Funktion intern auch "nur" `read.table()` auf, jedoch mit jeder Menge voreingestellter Parameter.

```
# importiere Daten mit read.csv()
# Daten sind mit TAB separiert!
NeueTabelle <- read.csv("DieDaten.csv", sep="\t", header=TRUE)
```

1.2.3.12.3 Import aus SPSS-Datei

Um native SPSS-Dateien in R zu importieren, müssen zunächste Zusatzpakete installiert werden, siehe hierzu den Abschnitt 1.2.3.14 auf Seite 80.

Zum Import von SPSS-Dateien installieren und aktivieren wir das Paket `haven`.

```
# installiere das Zusatzpaket "haven"
install.packages("haven")

# aktiviere das Paket "haven"
library("haven")
```

Das Paket `haven` bietet für jeden SPSS-Dateityp die passende Funktion an. Angenommen, die SPSS-Datei hieße "`alteDaten.sav`" und läge in unserem Arbeitsverzeichnis, dann kann sie mit der Funktion `read_sav()` in R importiert werden.

```
# importiere SPSS-Datei "alteDaten.sav"
alteDaten <- read_sav("alteDaten.sav")
```

In SPSS werden kategoriale Daten häufig mit *Labels* versehen.

```
# Lade Test-SPSS-Datei mit Labels
spss <- read_sav(url("https://www.produnis.de/R/alteDaten-lang.sav"))
head(spss)
```

```
  Frage_1             Frage_2        Frage_3             Frage_4
  <dbl+lbl>           <dbl+lbl>      <dbl+lbl>           <dbl+lbl>
1 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
2 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
3 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
4 4 [stimme zu]       3 [weiß nicht] 3 [weiß nicht]      4 [stimme zu]
5 3 [weiß nicht]      3 [weiß nicht] 2 [stimme nicht zu] 2 [stimme nicht zu]
6 2 [stimme nicht zu] 3 [weiß nicht] 3 [weiß nicht]      2 [stimme nicht zu]
```

Wir sehen, dass die Werte der Spalten gelabelt sind (4=*stimme zu*, 3=*weiß nicht*, usw.). Die Labels stehen in eckigen Klammern neben dem eingentlichen Wert der Variable.

Die enthaltenen Labels kann man sich mit der Funktion `attr()` anzeigen lassen.

```
# Namen der Variablen
attr(spss, "names")
```

```
[1] "Frage_1" "Frage_2" "Frage_3" "Frage_4"
```

Das Label einer Variable wird angezeigt mit

```
# Label der Variablen
attr(spss$v1, "label")
```

```
[1] "Statistik ist mein Lieblingsfach?"
```

Die Labels innerhalb einer Variable mit

```
  # Label der Variablenwerte für "Frage_1"
  attr(spss$Frage_1, "labels")
```

```
nicht vorhanden stimme gar nicht zu     stimme nicht zu
  0                   1                   2
weiß nicht           stimme zu      stimme voll zu
  3                   4                   5
```

In R ist die Verwendung von Wertelabels eher untypisch. Es empfiehlt sich, die Ausprägungsstufen *so wie sie sind* als Werte einzutragen, also z.B. direkt "`männlich`" - "`weiblich`" - "`divers`" anstatt `0 - 1 - 2` und anschließender Labelung. So werden die Ausprägungen auch auf den Grafiken entsprechend "aussagekräftig" angezeigt.

Wir wandeln die Variablen in Faktoren um, welche die Labels als Levelnamen nutzen[3].

```
library(tidyverse)
spss2 <- spss %>%
             mutate(as_factor(.))
head(spss2, 10)
```

```
   Frage_1         Frage_2             Frage_3         Frage_4
   <fct>           <fct>               <fct>           <fct>
 1 stimme zu       stimme zu           stimme zu       nicht vorhanden
 2 stimme zu       stimme zu           stimme zu       nicht vorhanden
 3 stimme zu       stimme zu           stimme zu       nicht vorhanden
 4 stimme zu       weiß nicht          weiß nicht      stimme zu
 5 weiß nicht      weiß nicht          stimme nicht zu stimme nicht zu
 6 stimme nicht zu weiß nicht          weiß nicht      stimme nicht zu
 7 stimme nicht zu stimme gar nicht zu weiß nicht      stimme nicht zu
 8 stimme nicht zu stimme voll zu      weiß nicht      weiß nicht
 9 stimme zu       stimme zu           stimme zu       stimme voll zu
10 weiß nicht      weiß nicht          weiß nicht      stimme zu
```

Weitere Informationen zum Umgang mit gelabelten Datensätzen erhalten Sie im Tidyverse-Abschnitt 1.6.6.1 auf Seite 149.

1.2.3.12.4 Import aus `Excel`-Datei

Um `xlsx`-Dateien in R zu importieren, muss das Zusatzpakete `readxl` installiert werden. Der Datenimport erfolgt dann mit der Funktion `read_excel()`:

```
# installiere das Zusatzpaket "readxl"
install.packages("readxl")

# aktiviere das Paket "readxl"
library("readxl")

# importiere Excel-Datei "alteDaten.xlsx"
alteDaten <- read_excel("alteDaten.xlsx")
```

Soll ein bestimmtes Tabellenblatt der `xlsx`-Datei importiert werden, kann diese mit dem Parameter `sheet` übergeben werden.

```
# importiere Tabelle "Tabelle1" aus Excel-Datei "alteDaten.xlsx"
alteDaten <- read_excel("alteDaten.xlsx", sheet="Tabelle1")
```

[3] Den folgenden Code können Sie *jetzt* noch nicht verstehen. Er stammt aus dem *Tidyverse*, dessen Funktionen in Kapitel 1.6 ab Seite 141 beschrieben werden.

Soll nur ein bestimmter Bereich der Tabelle importiert werden, kann diese mit dem Parameter `range` übergeben werden.

```
# importiere aus Excel-Datei "alteDaten.xlsx"
# Tabellenblatt "Tabelle1"
# im Bereich A2 bis D3
alteDaten <- read_excel("alteDaten.xlsx", sheet="Tabelle1", range="A2:D3")
```

1.2.3.12.5 Import aus `.ods`-Datei

Um `ods`-Dateien (z.B. aus `Libreoffice` oder `OpenOffice`) in R zu importieren, muss das Zusatzpakete `readODS` installiert werden. Der Datenimport erfolgt dann mit der Funktion `read_ods()`:

```
# installiere das Zusatzpaket "readODS"
install.packages("readODS")

# aktiviere das Paket "readODS"
library("readODS")

# importiere ods-Datei "tolleDaten.ods"
tolleDaten <- read_ods("tolleDaten.ods")
```

Soll ein bestimmtes Tabellenblatt der `ods`-Datei importiert werden, kann diese mit dem Parameter `sheet` übergeben werden.

```
# importiere Tabelle "Tabelle1" aus ods-Datei "cooleDaten.ods"
cooleDaten <- read_ods("cooleDaten.ods", sheet="Tabelle1")
```

Soll nur ein bestimmter Bereich der Tabelle importiert werden, kann diese mit dem Parameter `range` übergeben werden.

```
# importiere aus ods-Datei "cooleDaten.ods"
# Tabellenblatt "Tabelle1"
# im Bereich A2 bis D3
alteDaten <- read.ods("cooleDaten.ods", sheet="Tabelle1", range="A2:D3")
```

1.2.3.12.6 Import mit RStudio

Mit RStudio können Daten sehr leicht importiert werden. Im Datenfenster klicken Sie oben auf Import Dataset.

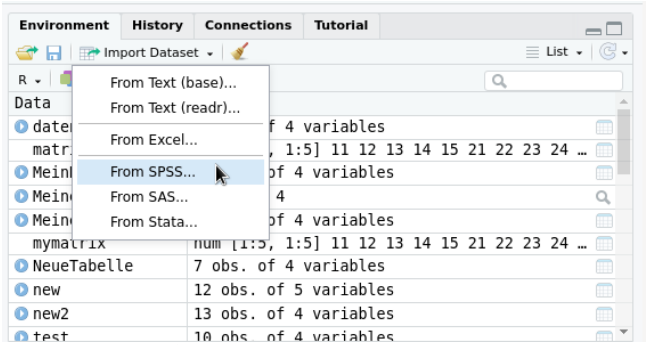


Abb. 1.34: Import Dataset

Es öffnet sich ein kleines Fenster, in welchem Sie das gewünschte Import-Format auswählen können (Abbildung 1.34). Wie Sie sehen, werden native Dateien aus Excel, SPSS, SAS und Stata unterstützt.
Wir nehmen in diesem Beispiel From SPSS.

Da der Datenimport das Zusatzpaket haven erfordert, wird es zur Installation vorgeschlagen, sofern es noch nicht installiert ist (Abbildung 1.35). Wir bestätigen in diesem Fall mit Yes.

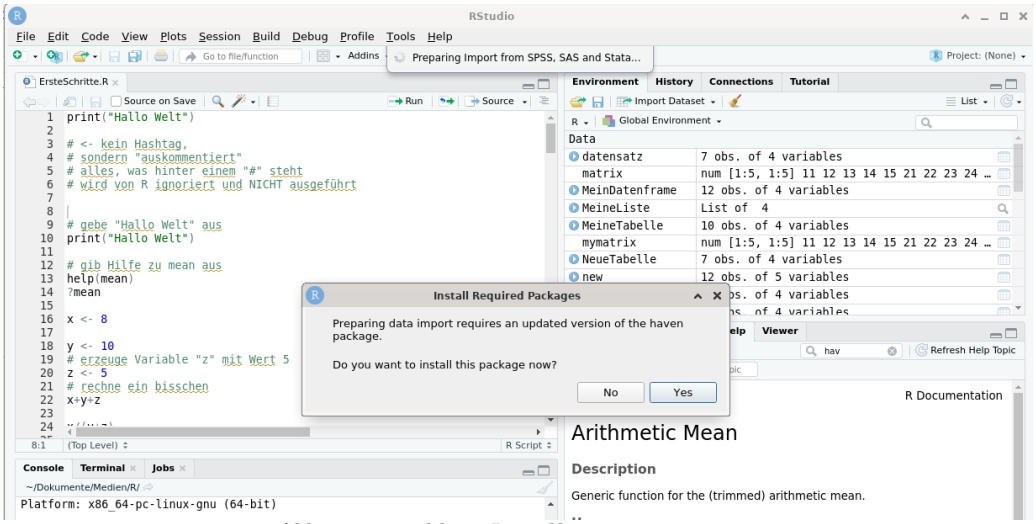


Abb. 1.35: schlägt Installation von haven vor

Nun öffnet sich das eigentliche Importierfenster (Abbildung 1.36).

Klicken Sie oben rechts auf den Browse-Knopf und wählen Sie die SPSS-Datei aus, die Sie importieren möchten. Für Abbildung 1.36 habe ich die Datei alteDaten-I.sav in meinem Arbeitsverzeichnis ausgewählt. Sobald Sie eine Datei ausgewählt haben, wird unter Data Preview eine Vorschau des zu importierenden Datensatzes angezeigt. Wie Sie sehen hat mein alter Datensatz die Variablen v1, v2, v3 …. Darunter sehen Sie die Ausprägungen der Variablen.

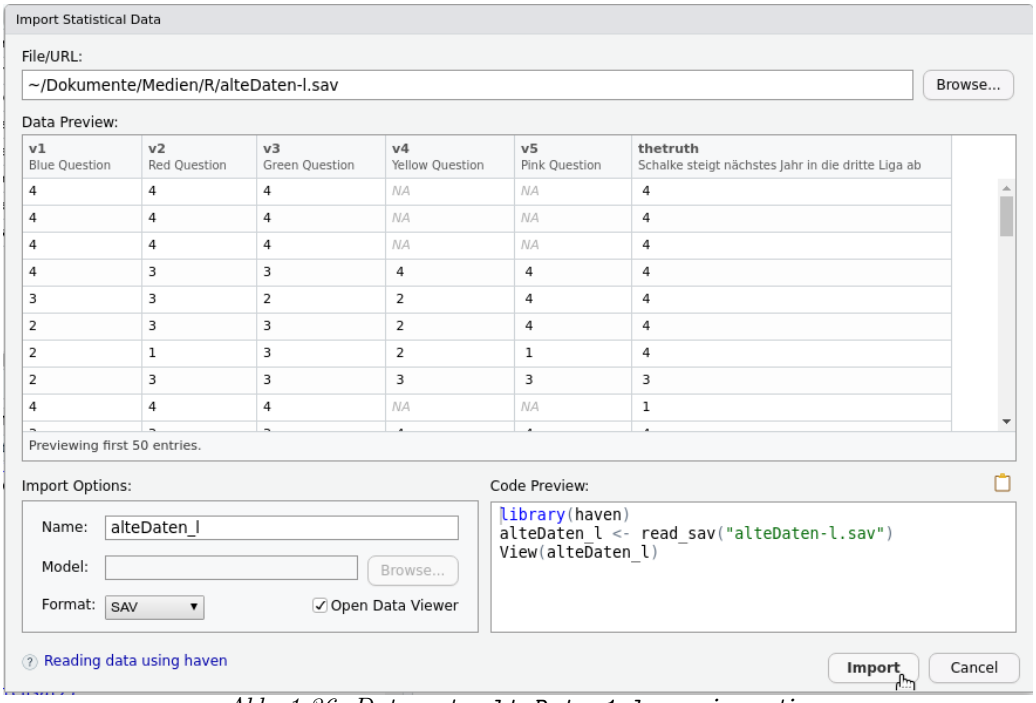


Abb. 1.36: Datensatz `alteDaten1-l.sav` importieren

Unter dem Vorschaufenster können Sie im linken Abschnitt (`Import Options`) den Objektnamen eintragen, den der Datensatz in R haben soll. Ich habe in Abbildung 1.36 den Namen `alteDaten_l` gewählt. Das `Format` ist schon automatisch auf `SAV` eingestellt (denn der Dateiname endet mit `.sav`). Der Haken bei `Open Data Viewer` bewirkt, dass der Datensatz nach dem Import im Scritpfenster angezeigt wird.

Im rechten Abschnitt (`Code Preview`) sehen Sie die R -Befehle, die in der Konosle ausgeführt werden, um den Import durchzuführen.

Klicken Sie unten rechts auf `Import`.

1.2.3.12.7 Importierte Daten ins richtige Format bringen

In R werden Dezimalstellen mit einem Punkt angegeben. Dies kann beim Importieren von Daten Probleme bereiten, insbesondere dann, wenn die zu importierenden Daten ihre Dezimalstelle mit einem Komma angeben.
Als Beispiel soll folgende CSV-Tabelle[4] dienen. Sie enthält die beiden Variablen `Anwesenheit` und `Note` (mehr Informationen gibt es in diesem Blogpost[5]).
Mittels `read.table()` in Kombination mit `url()` lesen wir die CSV-Datei in unsere R-Session.

```
# einlesen
df <- read.table(url("http://www.produnis.de/nagut/data/anwesenheitnoten.csv"),
            sep=";",
            header=TRUE)
# anzeigen
df
```

[4] http://www.produnis.de/nagut/data/anwesenheitnoten.csv
[5] https://www.produnis.de/blog/posts/2023-09-29-AnwesenheitNoten/

```
##    Anwesenheit Note
## 1         90,9  1,7
## 2         40,9    5
## 3          100    1
## 4         81,8
## 5            0    5
## 6          100    2
## 7         27,3  3,7
## 8         27,3    5
## 9         54,5  3,3
## 10        54,5
## 11        45,5    4
## 12         100  1,3
## 13          50  3,3
## 14        27,3    4
## 15        68,2  2,3
## 16         100
## 17         100    1
## 18          50    2
## 19        63,6  2,7
## 20        16,7
## 21        31,8    4
## 22        72,7  3,3
## 23        27,3    4
## 24        31,8    3
## 25        18,2    4
## 26          80  1,7
## 27        90,9  1,3
```

```
# Datentyp
str(df)
```

```
## 'data.frame':    27 obs. of  2 variables:
##  $ Anwesenheit: chr  "90,9" "40,9" "100" "81,8" ...
##  $ Note       : chr  "1,7" "5" "1" "" ...
```

Die Daten sind als `chr` eingelesen worden. Mit der Funktion `as.numeric()` können wir solche Daten in `numeric` umwandlen, eigentlich...

```
dummy <-c("1", "5", "9", "18")
# anzeigen
dummy
```

```
## [1] "1"  "5"  "9"  "18"
```

```
# wandle chr in numeric um:
as.numeric(dummy)
```

```
## [1]  1  5  9 18
```

Bei den zuvor importierten Daten funktioniert dies aber nicht:

```
as.numeric(df$Anwesenheit)
```

```
## Warning: NAs durch Umwandlung erzeugt
##  [1]  NA  NA 100  NA   0 100  NA  NA  NA  NA  NA 100  50  NA  NA 100 100  50  NA
## [20]  NA  NA  NA  NA  NA  NA  80  NA
```

Das liegt an der Dezimaltrennung per Kommata. Wir müssen zunächst alle Kommata in Punkte umwandeln. Dies erfolgt mit der Funktion `gsub()`, die als Parameter den zu suchenden Wert gefolgt vom zu ersetzenden Wert entgegennimmt.

```
# ersetze alle Kommata durch Punkte in df$Note
gsub(",", ".", df$Note)
```

```
##  [1] "1.7" "5"   "1"   ""    "5"   "2"   "3.7" "5"   "3.3" ""    "4"   "1.3"
## [13] "3.3" "4"   "2.3" ""    "1"   "2"   "2.7" ""    "4"   "3.3" "4"   "3"
## [25] "4"   "1.7" "1.3"
```

Jetzt bauen wir die beiden Funktionen zusammen:

```
as.numeric(gsub(",", ".", df$Note))
```

```
##  [1] 1.7 5.0 1.0  NA 5.0 2.0 3.7 5.0 3.3  NA 4.0 1.3 3.3 4.0 2.3  NA 1.0 2.0 2.7
## [20]  NA 4.0 3.3 4.0 3.0 4.0 1.7 1.3
```

Genau so wollten wir es haben. Ändern wir nun das Datenframe entsprechend:

```
df$Anwesenheit <- as.numeric(gsub(",", ".", df$Anwesenheit))
df$Note <- as.numeric(gsub(",", ".", df$Note))
#anzeigen
df
```

```
##    Anwesenheit Note
## 1         90.9  1.7
## 2         40.9  5.0
## 3        100.0  1.0
## 4         81.8   NA
## 5          0.0  5.0
## 6        100.0  2.0
## 7         27.3  3.7
## 8         27.3  5.0
## 9         54.5  3.3
## 10        54.5   NA
## 11        45.5  4.0
## 12       100.0  1.3
## 13        50.0  3.3
## 14        27.3  4.0
## 15        68.2  2.3
## 16       100.0   NA
## 17       100.0  1.0
## 18        50.0  2.0
## 19        63.6  2.7
## 20        16.7   NA
## 21        31.8  4.0
## 22        72.7  3.3
## 23        27.3  4.0
## 24        31.8  3.0
## 25        18.2  4.0
## 26        80.0  1.7
## 27        90.9  1.3
```

```
# Datentyp
str(df)
```

```
## 'data.frame':    27 obs. of  2 variables:
##  $ Anwesenheit: num  90.9 40.9 100 81.8 0 100 27.3 27.3 54.5 54.5 ...
##  $ Note       : num  1.7 5 1 NA 5 2 3.7 5 3.3 NA ...
```

Jetzt ist alles richtig eingelesen.

1.2.3.13 Daten exportieren

In Abschnitt 1.2.3.10 auf Seite 65 wird beschrieben, wie Sie Ihre R-Objekte im `.RData`-Format abspeichern können.
Es ist aber natürlich ebenso möglich, die Daten in anderen Formaten zu exportieren.

1.2.3.13.1 Export als CSV-Datei

Mit dem CSV-Format kommen die meisten Anwendungen zurecht. Daher empfehlen wir, dieses zum Tausch zwischen Anwendungen zu verwenden. Wir benutzen hierfür die Funktion `write.csv()` und speichern das Datenframe `MeinDatenframe` in die Datei `MeinDatenframe.csv`. Als Datenseparator wird standardmäßig ein Leerzeichen verwendet. Soll ein anderer Separator verwendet werden, kann dies über den Parameter `sep` angegeben werden.

```
# exportiere "MeinDatenframe" als CSV-Datei
# mit "," als Separator
write.csv(MeinDatenframe, "MeinDatenframe.csv", sep=",")
```

1.2.3.13.2 Export als Textdatei-Datei

Mit der Funktion `write.table()` können die Daten als simple Textdatei gespeichert werden. Als Datenseparator wird auch hier standardmäßig ein Leerzeichen verwendet. Soll ein anderer Separator verwendet werden, kann dies über den Parameter `sep` angegeben werden.

```
# exportiere "MeinDatenframe" als Text-Datei
# mit "," als Separator
write.table(MeinDatenframe, "MeinDatenframe.txt", sep=",")
```

Sollen die Daten an eine bestehende Datei angehängt werden, muss der Parameter `append` auf `TRUE` gesetzt werden.

1.2.3.13.3 Export als SPSS-Datei

Mit dem Zusatzpaket `haven` können R-Objekte als SPSS-Datei abgespeichert werden.
Wir benutzen hierfür die Funktion `write_sav()` und speichern das Datenframe `MeinDatenframe` in die Datei `MeinDatenframe.sav`.

```
# exportiere "MeinDatenframe" als SPSS-Datei
write_sav(MeinDatenframe, "MeinDatenframe.sav")
```

Um eine komprimierte `.zsav`-Datei zu erstellen, setzen wir den Parameter `compress` auf `TRUE`.

```
# exportiere "MeinDatenframe" als komprimierte SPSS-Datei
write_sav(MeinDatenframe, "MeinDatenframe.sav", compressp=TRUE)
```

1.2.3.14 Zusatzpakete installieren

In R stehen viele Zusatzpakete bereit, die den Funktionsumfang erweitern. Über das *Comprehensive R Archive Network*, kurz `CRAN`, können diese mit dem Befehl `install.packages()` installiert werden.
Möchte man beispielsweise "ggplot" (siehe Abschnitt 2.4 auf Seite 352) installieren, lautet der Befehl entsprechend (Achtung, das Paket für ggplot heisst `ggplot2` !)

```
# ggplot installieren, ACHTUNG, das Paket heisst ggplot2 !!!
install.packages("ggplot2")
```

Manche Pakete benötigen weitere Zusatzpakete, um vollständig laufen zu können. Man spricht hier von "Abhängigkeiten", da das eine Paket vom anderen abhängt. Möchte man das gewünschte Paket inklusive aller Abhängigkeiten installieren, ändert sich der Befehl in:

```
# Paket ggplot mit allen Abhängigkeiten installieren
install.packages("ggplot2", dependencies=TRUE)
```

Sollen mehrere Pakete auf einmal installiert werden, können diese mit der `c()`-Funktion übergeben werden.

```
# Pakete ggplot, wikibooks, pwr mit allen Abhängigkeiten installieren
install.packages(c("ggplot2", "wikibooks", "pwr"), dependencies=TRUE)
```

Um die installierten Zusatzpakete benutzen zu können, müssen sie in einer R-Sitzung zunächst geladen bzw. aktiviert werden. Dies erfolgt mit dem Befehl `library()`.

```
# aktiviere das ggplot2 Paket
library(ggplot2)
```

Mit dem Befehl `detach()` können die Zusatzpakete wieder deaktiviert werden.

```
# aktiviere das ggplot2 Paket
detach("package:ggplot2")
```

Ein anderer Weg besteht darin, den Paketnamen mit zwei Doppelpunkten getrennt vor die Funktion zu schreiben.

```
# nutze Funktion "read_sav" aus dem Paket "haven"
haven::read_sav("SPSSdaten.sav")
```

Die beiden Doppelpunkte weisen R an, die Funktion `read_sav()` aus dem Paket `haven` zu nutzen. So kann man sich den `library(haven)`-Aufruf sparen. Dies bietet sich an, wenn man "nur mal kurz" eine bestimmte Funktion eines bestimmten Pakets nutzen möchte. Hat man mehrere Funktionsaufrufe, kann das Laden per `library()` die Schreibarbeit verkürzen.

In RStudio lassen sich Pakete auf mehrere Arten installieren. Zum einen kann man in der Menüzeile auf Tools → Install packages klicken. Es öffnet sich ein neues Fenster, in welches wir einfach die Paketnamen eintragen können (Abbildung 1.37).

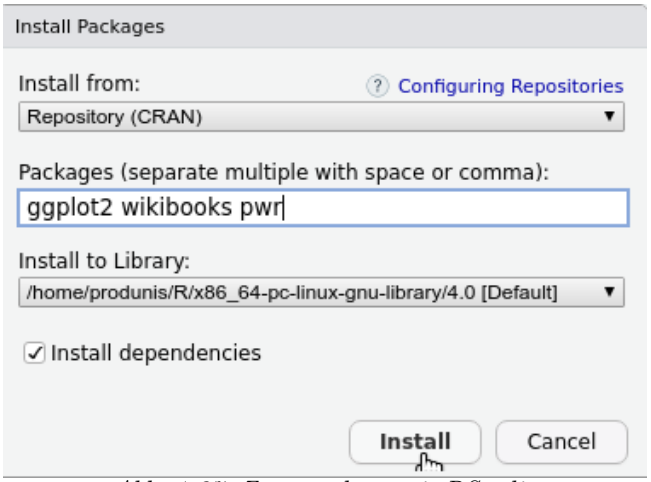


Abb. 1.37: Zusatzpakete mit RStudio

RStudio bietet eine weitere recht einfache und nützliche Methode zur Installation von Zusatzpaketen. Die Software durchsucht Ihre Scriptdatei nach dem Befehl library(). Ist das dort referenzierte Paket noch nicht installiert, schlägt RStudio die Installation vor (siehe gelber Balken in Abbildung 1.38). Klicken Sie im gelben Balken auf Install um das Paket zu instal-

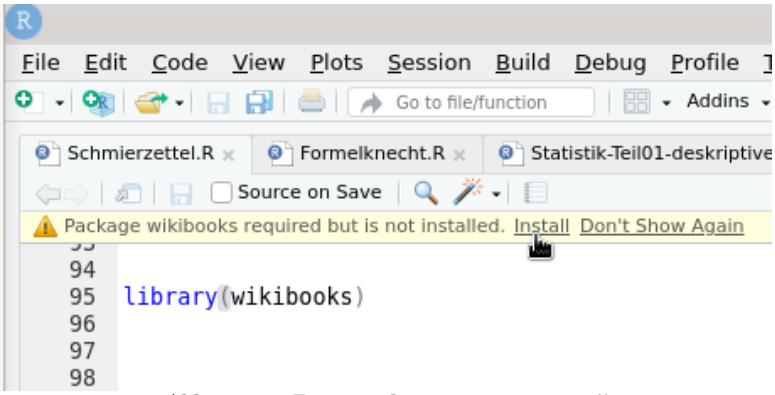


Abb. 1.38: Zusatzpakete per `library()`

lieren. Dies ist nützlich, wenn man z.B. an einem "neuen" PC *alte* R-Scriptdateien öffnet, oder z.B. *fremde* R-Scripte verwendet. Nach dem Öffnen schlägt RStudio alle fehlenden Pakete zur Installation vor.
Natürlich können Sie auch einfach install.packages("ggplot2") (oder jeden anderen Paketnamen) in Ihre Scriptdatei schreiben und diese dann in der R-Konsole ausführen lassen.

Das Aktivieren von installierten Zusatzpaketen wird von RStudio ebenfalls ermöglicht. Klicken Sie im Ausgabefenster auf den Reiter Packages. So werden Ihnen alle installierten Zusatzpakete angezeigt (Abbildung 1.39).

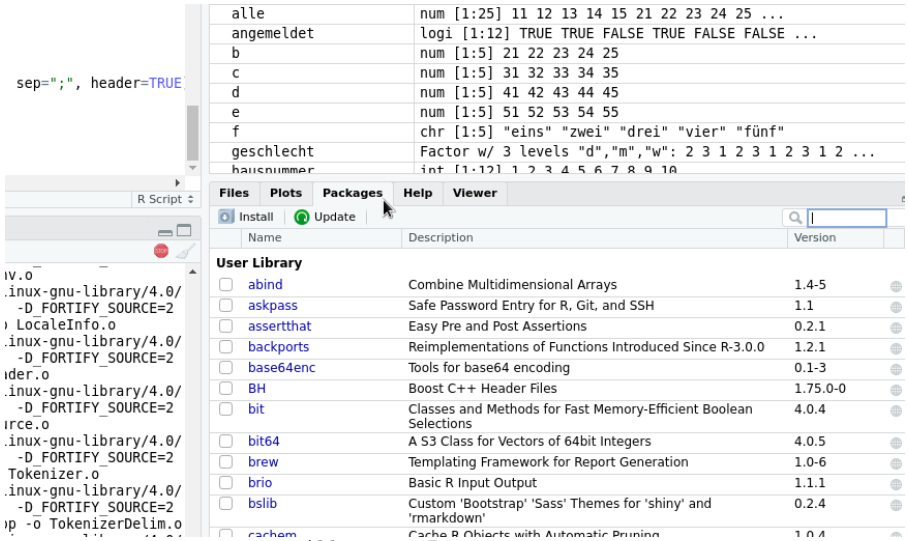


Abb. 1.39: Reiter `Packages`

Wenn Sie ein Paket aktivieren möchten, klicken Sie einfach dessen Häkchen an.

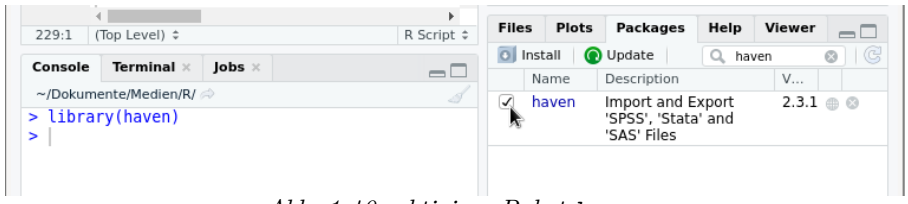


Abb. 1.40: aktiviere Paket `haven`

Wie Abbildung 1.40 zeigt, wurde durch den Klick automatisch der Befehl `library(haven)` in die R-Konsole übertragen und ausgeführt (linke Seite). Zum Deaktivieren müssen Sie einfach wieder auf das Kästchen klicken und den Haken entfernen.

Viele Paket enthalten Datensätze, die für Übungs- und Auswertungsbeispiele genutzt werden können.
Die enthaltenen Datensätze lassen sich mit dem Befehl
`data(package="PAKETNAME")` auflisten. In R ist das Basispaket `datasets` enthalten, dessen Inhalte sich anzeigen lassen per:

```
1  # zeige die Datensätze des Pakets "datasets" an
2  data(package = "datasets")
```

```
Data sets in package 'datasets':

AirPassengers               Monthly Airline Passenger Numbers 1949-1960
BJsales                     Sales Data with Leading Indicator
BJsales.lead (BJsales)      Sales Data with Leading Indicator
BOD                         Biochemical Oxygen Demand
CO2                         Carbon Dioxide Uptake in Grass Plants
ChickWeight                 Weight versus age of chicks on different diets
DNase                       Elisa assay of DNase
EuStockMarkets              Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde                Determination of Formaldehyde
HairEyeColor                Hair and Eye Color of Statistics Students
Harman23.cor                Harman Example 2.3
Harman74.cor                Harman Example 7.4
Indometh                    Pharmacokinetics of Indomethacin
InsectSprays                Effectiveness of Insect Sprays
JohnsonJohnson              Quarterly Earnings per Johnson & Johnson Share
LakeHuron                   Level of Lake Huron 1875-1972
LifeCycleSavings            Intercountry Life-Cycle Savings Data
Loblolly                    Growth of Loblolly pine trees
Nile                        Flow of the River Nile
Orange                      Growth of Orange Trees
OrchardSprays               Potency of Orchard Sprays
PlantGrowth                 Results from an Experiment on Plant Growth
Puromycin                   Reaction Velocity of an Enzymatic Reaction
Seatbelts                   Road Casualties in Great Britain 1969-84
Theoph                      Pharmacokinetics of Theophylline
Titanic                     Survival of passengers on the Titanic
```

Abb. 1.41: Datensätze im Basispaket `datasets`

Um einen Datensatz eizubinden, wird er mit der `data()`-Funktion aufgerufen.

```r
# aktiviere den Datensatz "iris"
data(iris)

# zeige den Inhalt von 'iris'
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Eine alternative Zugriffsweise besteht darin, die Datensätze per `paketname::datensatz` einem Objekt zuzuweisen.

```r
# weise den Datensatz psych::bfi dem Objekt testdf zu
testdf <- psych::bfi

# zeige erste Zeilen an
head(testdf)
```

```
      A1 A2 A3 A4 A5 C1 C2 C3 C4 C5 E1 E2 E3 E4 E5 N1 N2 N3 N4 N5
61617  2  4  3  4  4  2  3  3  4  4  3  3  3  4  4  3  4  2  2  3
61618  2  4  5  2  5  5  4  4  3  4  1  1  6  4  3  3  3  3  5  5
61620  5  4  5  4  4  4  5  4  2  5  2  4  4  4  5  4  5  4  2  3
61621  4  4  6  5  5  4  4  3  5  5  5  3  4  4  4  2  5  2  4  1
61622  2  3  3  4  5  4  4  5  3  2  2  2  5  4  5  2  3  4  4  3
61623  6  6  5  6  5  6  6  6  1  3  2  1  6  5  6  3  5  2  2  3
      O1 O2 O3 O4 O5 gender education age
61617  3  6  3  4  3      1        NA  16
61618  4  2  4  3  3      2        NA  18
61620  4  2  5  5  2      2        NA  17
61621  3  3  4  3  5      2        NA  17
61622  3  3  4  3  3      1        NA  17
61623  4  3  5  6  1      2         3  21
```

1.2.3.14.1 Zusatzpaket zu diesem Lehrbuch

Zu diesem Lehrbuch existiert ein eigenes Zusatzpaket, in welchem alle verwendeten Datensätze und Zusatzfunktionen enthalten sind. Das Paket heisst `jgsbook` und kann über das CRAN[6] installliert werden:

```
# installiere das Zusatzpaket dieses Lehrbuchs
install.packages("jgsbook")
```

Die enthaltenen Datensätze lassen sich mit dem Befehl `data()` auflisten:

```
# zeige die Datensätze des Pakets "jgsbook" an
data(package = "jgsbook")
```

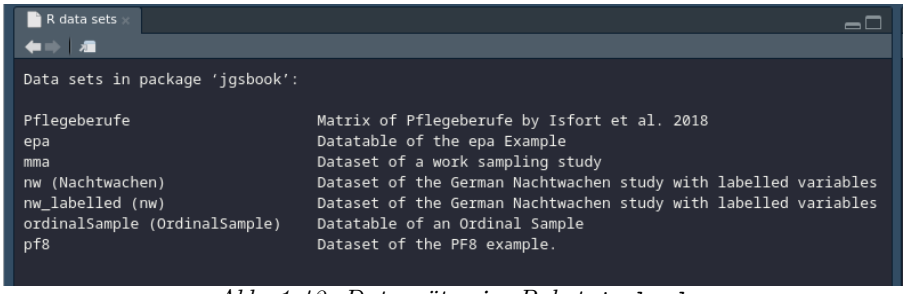


Abb. 1.42: Datensätze im Paket `jgsbook`

...und in die Arbeitsumgebung laden:

```
# aktiviere jgsbook
library(jgsbook)

# aktiviere den Datensatz "epa"
data(epa)

# zeige die ersten 6 Datenreihen von "epa"
head(epa)
```

```
  age sex cms risk expert decu
1  50   m  40    0      0    0
2  75   m  40    0      0    0
3  86   w  13    1      1    0
4  77   w  24    1      1    0
5  66   w  40    0      0    0
6  51   m  37    0      0    0
```

Sie können auch über die Kombination `paketname::datensatz` auf die Datensätze zugreifen. So muss das Paket nicht erst per `library()` aktiviert werden.

```
# Datensatz "epa" aus dem Paket "jgsbook"
epa <- jgsbook::epa

# zeige die ersten 6 Datenreihen von "epa"
head(epa)
```

[6] https://cran.r-project.org/package=jgsbook

```
  age sex cms risk expert decu
1  50   m  40    0      0    0
2  75   m  40    0      0    0
3  86   w  13    1      1    0
4  77   w  24    1      1    0
5  66   w  40    0      0    0
6  51   m  37    0      0    0
```

1.2.3.15 Ausgabe in Datei umleiten

Das Umleiten der R-Ausgabe in eine Textdatei ist evtl. ein etwas angestaubtes Verfahren, welches vor allem in Zeiten *vor* `RStudio` Anwendung gefunden hat. Heutzutage steht mit `quarto` eine komfortable Lösung zur Verfügung, hübsche R-Auswertungen z.B. in einem ansprechenden `html`-Dokument (sprich einer Webseite) zu präsentieren (siehe hierzu Abschnitt 1.5 auf Seite 127).

Standardmäßig schreibt R seine Ausgaben in das Konsolfenster. Manchmal kann es aber auch vorteilhaft sein, die Ausgaben in eine Datei umzulenken. Dies kann mit der Funktion `sink()` erreicht werden. Der folgende Befehl leitet alle Ausgaben in die Textdatei `Ausgabe.txt` im Arbeitsverzeichnis um.

```
1  # leite ab jetzt alle Ausgaben in die Datei "Ausgabe.txt" um
2  sink("Ausgabe.txt")
```

Soll die Umleitung aufgehoben werden, erfolgt dies mit einem leeren Funktionsaufruf.

```
1  # beende die Umleitung
2  sink()
```

So kann man in der Scriptdatei genau festlegen, ab wann die Ausgaben in eine Datei geschrieben werden sollen, und wann nicht. Beachten Sie, dass R die Textdatei immer wieder überschreiben wird, z.B. wenn der `sink()` Aufruf beendet und dann erneut gestartet wird. Um die Ausgabe an eine bestehende Datei *anzuhängen* (ohne diese zu überschreiben) muss der Parameter `append` auf `TRUE` gesetzt werden.

```
1  # hänge ab jetzt alle Ausgaben an die Datei "Ausgabe.txt" an
2  sink("Ausgabe.txt", append=TRUE)
```

An dieser Stelle sei auf die Befehle `print()` und `cat()` hingewiesen. Wenn Sie die R-Ausgabe in eine Textdatei umleiten, möchten Sie evtl. ebenfalls Kommentare oder anderen Text in die Textdatei schreiben lassen, z.B. bevor Sie Befehle ausführen (oder nachdem).

Der Befehl `print()` gibt den Text dabei in der R -bekannten Weise aus.

```
1  # schreibe mit print() in die Textdatei
2  print("Guten Tag, beginnen wir mit der Analyse.")
```

In die Textdatei wird geschrieben:

```
[1] "Guten Tag, beginnen wir mit der Analyse."
```

Mit dem Befehl `cat()` wird tatsächlich nur der Text ausgegeben (ohne den Positionsanzeiger `[1]`).

```
1  # schreibe mit cat() in die Textdatei
2  cat("Guten Tag, beginnen wir mit der Analyse.\nLos geht es!")
```

In die Textdatei wird geschrieben:

```
Guten Tag, beginnen wir mit der Analyse.
Los geht es!
```

Der Parameter `\n` gibt an, dass eine neue Zeile beginnen soll.

1.2.4 Verteilungsfunktionen

In R sind zahlreiche Verteilungsfunktionen implementiert. Wir beschränken uns in diesem Buch auf die für uns wichtigen Wahrscheinlichkeisverteilungen.
Eine vollständige Liste aller implementierten Verteilungen findet sich unter
https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html

Der Funktionsumfang folgt dabei einer gewissen Logik, für jede Verteilung existiert je eine `d`, `p`, `q` und `r`-Funktion. Für die Normalverteilung (siehe Näheres in Abschnitt 1.2.4.1) sind das `dnorm()`, `pnorm()`, `qnorm()` und `rnorm()`. Für die t-Verteilung (siehe Abschnitt 1.2.4.2) heißen die Equivalente `dt()`, `pt()`, `qt()` und `rt()`, für χ^2 entsprechend `dchisq()`, `pchisq()`, `qchisq()` und `rchisq()`.

`d` ist jeweils die Dichtefunktion der Verteilung

`p` berechnet die Werte der Verteilungsfunktion (kumulierte Dichte)

`q` bestimmt die Quantile der Verteilung

`r` erzeugt Zufallswerte aus der Verteilung

Dies möchten wir am Beispiel der Normalverteilung verdeutlichen.

1.2.4.1 Normalverteilung

Die **Normalverteilung** ist mit den oben beschriebenen Funktionen in R implementiert.

Mit der Funktion `dnorm()` berechnet man die Werte der Wahrscheinlichkeitsdichte. Möchte man beispielsweise die Wahrscheinlichkeitsdichte für $x = 2$ aus der Standardnormalverteilung berechnen lautet der Befehl:

```
# Wahrscheinlichkeitsdichte für x=2
# in der Standardnormalverteilung
dnorm(2)
```

```
[1] 0.05399097
```

Standardmäßig stammen diese Werte aus der Standardnormalverteilung, sie streuen also mit einer Standardabweichung von $s = 1$ um den Mittelwert $\bar{x} = 0$.
Möchte man dies ändern, können der Funktion entsprechende Werte für Mittelwert und Standardabweichung übergeben werden:

```
# Wahrscheinlichkeitsdichte für x=4
# aus einer Verteilung mit Mittelwert 8 und Standardabweichung 2
dnorm(4, mean=8, sd=2)
```

```
[1] 0.02699548
```

Die Werte der Verteilungsfunktion lassen sich mit `pnorm()` berechnen. Die Funktion gibt die Wahrscheinlichkeit an, dass ein Zufallsvariable einen Wert von $\leq x$ (= "*höchstens*" x) annimmt.

```
# Wahrscheinlichkeit dass Wert kleiner-gleich 2
# in Standardnormalverteilung
pnorm(2)
```

```
[1] 0.9772499
```

Für andere Normalverteilungen ändert sich der Befehl entsprechend:

```
# Wahrscheinlichkeit dass Wert kleiner-gleich 8
# in Normalverteilung mit xquer=12 und s=3
pnorm(8, mean=12, s=3)
```

```
[1] 0.09121122
```

Quantile lassen sich mit der Funktion `qnorm()` berechnen. Quantile stellen Grenzen dar, die ein zufälliger Wert mit einer vorgegebenen Wahrscheinlichkeit nicht *überschreiten* wird. Wenn das 95%-Quantil beispielsweise den Wert 5 hat, wird eine zufällig gezogene Zahl mit 95% Sicherheit *kleiner* als 5 sein.

```
# Grenze des 95% Quantils
# in Standardnormalverteilung
qnorm(p=0.95)
```

```
[1] 1.644854
```

Auch `qnorm()` können Werte für andere Normalverteilungen übergeben werden.

```
# Grenze des 95% Quantils
# in Normalverteilung mit xquer=9 und sd=5
qnorm(p=0.95, mean=9, sd=5)
```

```
[1] 17.22427
```

Mit `rnorm()` lassen sich normalverteilte Zufallswerte erzeugen:

```
# erzeuge "zufällig" 10  Werte
# aus der Standardnormalverteilung
rnorm(10)
```

```
[1] -0.0127120  1.2604385  1.7311170  1.5315658  1.2866661
[6] -2.2716082  0.7971483  0.6404524  0.1764541 -0.1808511
```

Auch mit `rnorm()` können wir andere Normalverteilungen für unsere Zufallszahlen angeben:

```
# erzeuge "zufällig" 10 normalverteilte Werte
# mit Mittelwert 8 und Standardabweichung 2
rnorm(10, mean=8, sd=2)
```

```
[1] 12.904742  9.272356  8.503763  4.955420  9.617479  7.018170
[7]  9.160690  5.292520  7.270711  7.172105
```

Plot-Beispiel mit der `plot()`-Funktion (siehe Kapitel **??**).

```
# Erzeuge Werte von -3 bis 6
x <- seq(-3, 6, by = 0.005)

# Plot erstellen
plot(x, dnorm(x, mean = 0, sd = 1),
     type = "l", xlim = c(-3, 6), ylim = c(0, 0.8),
     xlab = "x", ylab = "Dichtefunktion", main = "Normalverteilungen")
lines(x, dnorm(x, mean = 2, sd = 0.5), col = "blue")
lines(x, dnorm(x, mean = 2, sd = 2), col = "darkorchid")

# Annotationen hinzufügen
text(0, 0.42, "N(0;1)", col = "black")
text(3, 0.6, "N(2;0.5)", col = "blue")
text(5, 0.1, "N(2;2)", col = "darkorchid")
```

Der Befehl erzeugt Abbildung 1.43.

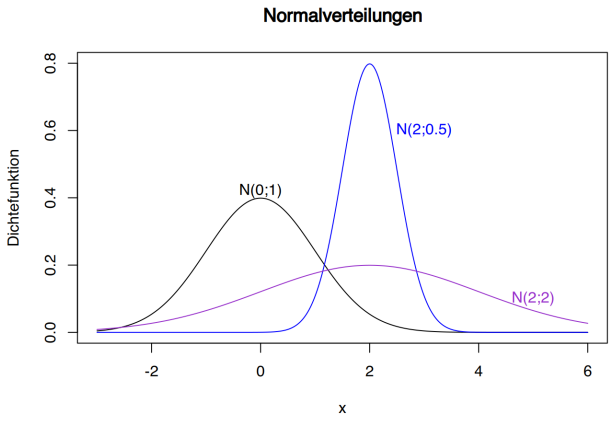


Abb. 1.43

Beispiel mit `ggplot()` (siehe Kapitel 2.4).

```
# erzeuge Werte von -3 bis 6
x <- seq(-3,6, by=0.005)
# übergebe in ein data.frame
```

```
    df <- data.frame(x)
    # ggplot erstellen
    p <- ggplot(data=df, aes(x)) +
                         xlim(-3,6) + ylim(0, 0.8) +
                         xlab("x") + ylab("Dichtefunktion") +
                         ggtitle("Normalverteilungen")
    p + stat_function(fun=dnorm, args=(c(mean=0,sd=1)), colour="black")+
      annotate(geom="text", x=0, y=0.42, label="N(0;1)", color="black")+
      stat_function(fun=dnorm, args=(c(mean=2,sd=0.5)), colour="blue") +
      annotate(geom="text", x=3, y=0.6, label="N(2;0.5)", color="blue")+
      stat_function(fun=dnorm, args=(c(mean=2,sd=2)), colour="darkorchid") +
      annotate(geom="text", x=5, y=0.1, label="N(2;2)", color="darkorchid")
```

Der Befehl erzeugt Abbildung 1.44.

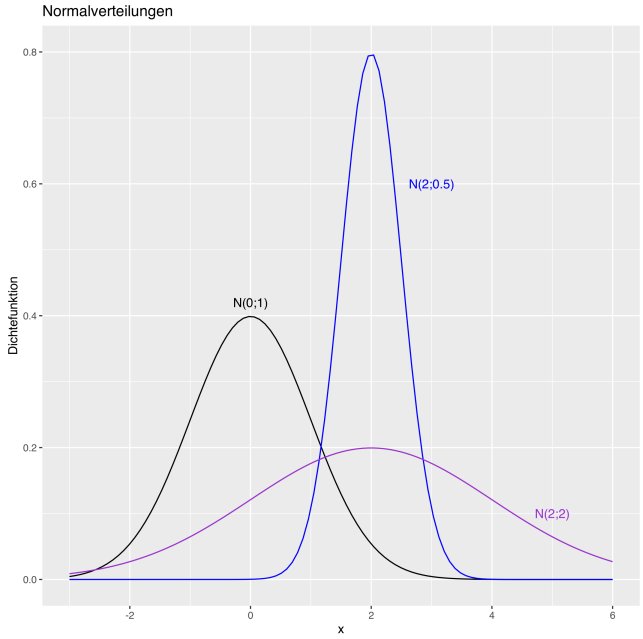


Abb. 1.44

Für weitere Plot-Beispiele zur Normalverteilung siehe Abschnitt 3.2.2 auf Seite 426.

1.2.4.2 t-Verteilung

Die "kleine Schwester" der Standardnormalverteilung ist die **Student-t-Verteilung** (oder einfach kurz **t-Verteilung**). Für sie stehen die selben Funktionstypen zur Verfügung wie für die Normalverteilung (siehe Abschnitt 1.2.4.1). Diese heissen entsprechend `dt()`, `pt()`, `qt()` und `rt()` und funktionieren genau wie ihre Pendants der Standardnormalverteilung. Jedoch muss ihnen noch die Anzahl der Freiheitsgrade (*df*) übergeben werden:

```r
# Wahrscheinlichkeitsdichte für x=2
# in der t-Verteilung mit 4 Freiheitsgraden
dt(2, df=4)
```

```
[1] 0.06629126
```

```r
# Wahrscheinlichkeit dass Wert kleiner-gleich 2
# in der t-Verteilung mit 3 Freiheitsgraden
pt(2, df=3)
```

```
[1] 0.930337
```

```r
# Grenze des 95% Quantils
# in der t-Verteilung mit 3 Freiheitsgraden
qt(p=0.95, df=3)
```

```
[1] 2.353363
```

```r
# erzeuge "zufällig" 10 Werte
# aus der t-Verteilung mit 13 Freiheitsgraden
rt(10, df=13)
```

```
 [1] -0.9328004  1.4917239  0.0955167 -0.3438326 -0.7593989
 [6] -0.0556377  0.1085506  1.5149383  0.4664587 -2.8477079
```

Möchte man die t-Verteilung plotten, kann das mit `plot()` (siehe Kapitel **??**) so erfolgen.

```r
# Erzeuge x-Werte
x <- seq(-3, 3, by = 0.005)

# Plot erstellen
plot(x, dt(x, df = 1), type = "l",
     xlim = c(-3, 3), ylim = c(0, 0.4),
     xlab = "x", ylab = "Dichtefunktion",
     main = "t-Verteilungen nach Freiheitsgraden")

# Text hinzufügen
text(0, 0.25, "df=1", col = "black")
```

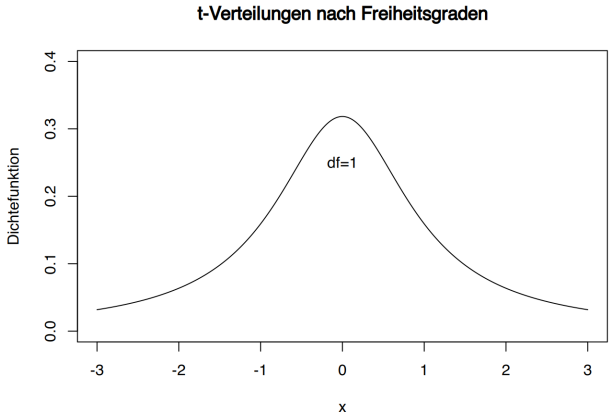


Abb. 1.45

... oder mittels mit `ggplot` (siehe Kapitel 2.4) wie folgt.

```
# Erzeuge x-werte
df <- data.frame(x=seq(-3,3, by=0.005))

# Grundlegene Plotangaben
p <- ggplot(data=df, aes(x)) +
  # begrenze die Achsen
  xlim(-3,3) + ylim(0, 0.4) +
  # Achsen-Titel
  xlab("x") + ylab("Dichtefunktion") +
  # Plot-Titel
  ggtitle("t-Verteilungen", subtitle = "nach Freiheitsgraden")

  # t-Verteilung plotten
p +
  stat_function(fun=dt, args=list(df=1), col="black") +
  # Textfeld hinzufügen
  annotate(geom="text", x=0, y=0.25, label="df=1", color="black")
```

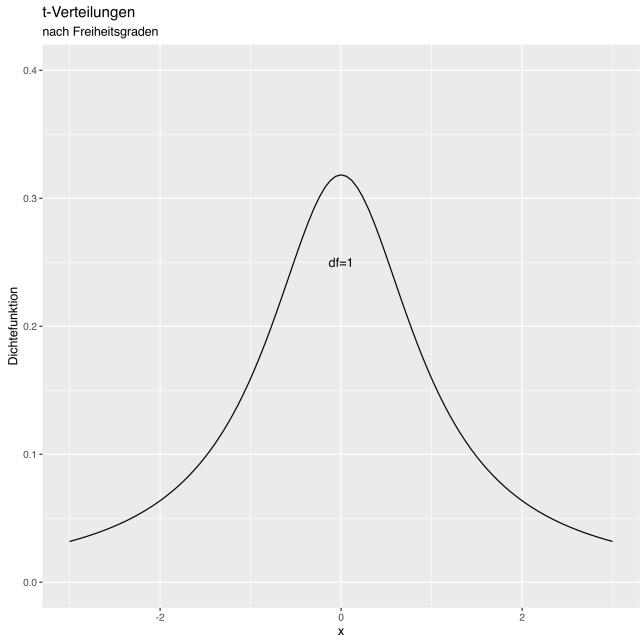


Abb. 1.46

Für weitere Plot-Beispiele zur t-Verteilung siehe Abschnitt 3.2.3 auf Seite 436.

1.2.4.3 χ^2-Verteilung

Die **χ^2-Verteilung** ist mit den oben beschriebenen Funktionen in R implementiert. Diese heissen entsprechend `dchisq()`, `pchisq()`, `qchisq()` und `rchisq()` und funktionieren genau wie ihre Pendants der Standardnormalverteilung. Jedoch muss ihnen (wie bei der t-Verteilung) noch die Anzahl der Freiheitsgrade (*df*) übergeben werden:

```r
# Wahrscheinlichkeitsdichte für x=2
# in der Chi^2-Verteilung mit 4 Freiheitsgraden
dchisq(2, df=4)
```

```
[1] 0.1839397
```

Möchte man die χ^2-Verteilung plotten, kann das mit `plot()` (siehe Kapitel **??**) so erfolgen.

```r
# Erzeuge x-Werte
x <- seq(0, 25, by = 0.005)

# Dichte-Funktionen berechnen
df01 <- dchisq(x, df = 1)
df05 <- dchisq(x, df = 5)
df10 <- dchisq(x, df = 10)
df15 <- dchisq(x, df = 15)

# Plot erstellen
plot(x, df01, type = "l",
     xlim = c(0, 25), ylim = c(0, 0.2),
     xlab = "x", ylab = "Dichtefunktion",
     main = "Chi^2-Verteilungen nach Freiheitsgraden")

# Linien für andere Freiheitsgrade hinzufügen
lines(x, df05, col = "blue")
lines(x, df10, col = "darkgreen")
lines(x, df15, col = "darkred")

# Legende hinzufügen
legend("topright", legend = c("df=1", "df=5", "df=10", "df=15"),
       col = c("black", "blue", "darkgreen", "darkred"),
       lty = 1, cex = 0.8)
```

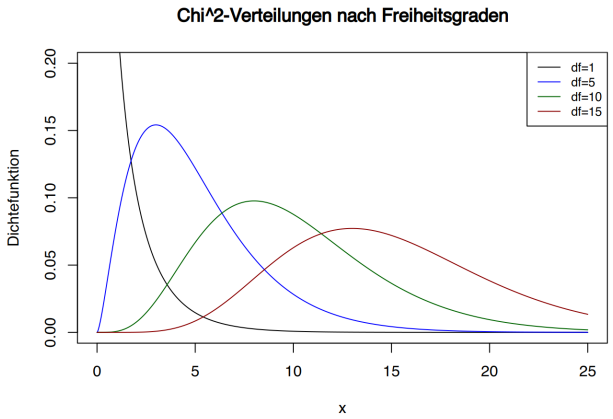


Abb. 1.47

... oder mittels mit `ggplot` (siehe Kapitel 2.4) wie folgt.

```
# Erzeuge x-werte
x=seq(0,25, by=0.005)
df <- data.frame(  x,
  df01 = dchisq(x, df=1),
  df05 = dchisq(x, df=5),
  df10 = dchisq(x, df=10),
  df15 = dchisq(x, df=15)
)

# erzeuge long-table
df <- pivot_longer(df, cols=c(df01, df05, df10, df15))

p <- ggplot(data=df, aes(x, value, fill=name)) +
  xlim(0,25) + ylim(0, 0.2) +
  xlab("x") + ylab("Dichtefunktion") +
  ggtitle("Chi^2-Verteilungen", subtitle = "nach Freiheitsgraden")
p + geom_line(aes(col=name,linetype=name))+
labs(col="Freiheitsgrade",linetype="")
```

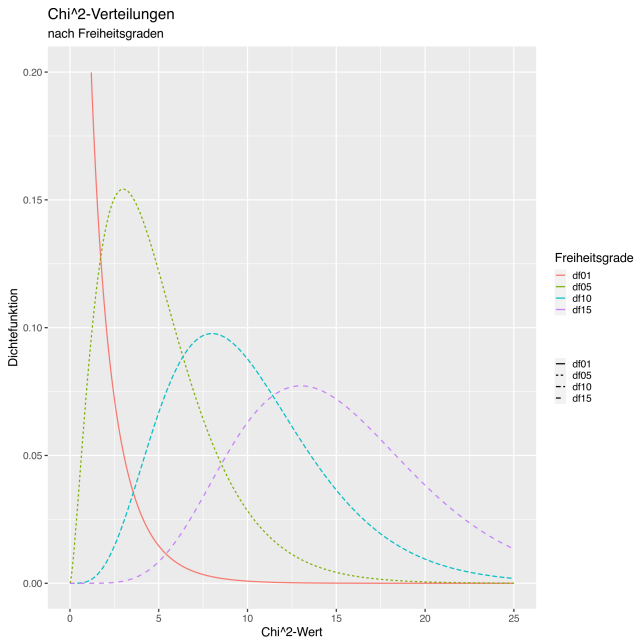


Abb. 1.48

Für weitere Plot-Beispiele zur χ^2-Verteilung siehe Abschnitt 3.2.4 auf Seite 439.

1.2.4.4 Poisson-Verteilung

Die **Poisson-Verteilung** ist mit den oben beschriebenen Funktionen in R implementiert. Diese heissen entsprechend `dpois()`, `ppois()`, `qpois()` und `rpois()` und funktionieren genau wie ihre Pendants der Standardnormalverteilung. Jedoch muss ihnen der Wert für λ übergeben werden.

```
# Wahrscheinlichkeitsdichte für x=2
# in der Poisson-Verteilung mit Lambda=4
dpois(2, lambda=4)
```

```
[1] 0.1465251
```

Möchte man die Poisson-Verteilung plotten, kann das mit `plot()` (siehe Kapitel **??**) so erfolgen. Wir plotten mit $\lambda = 1, 2, 4$ und 9.

```
# Erzeuge x-Werte
x <- seq(0, 25)

# Dichtefunktionen berechnen
l1 <- dpois(x, lambda = 1)
l2 <- dpois(x, lambda = 2)
l4 <- dpois(x, lambda = 4)
l9 <- dpois(x, lambda = 9)

# Plot erstellen
plot(x, l1, type = "l",
     xlim = c(0, 17), ylim = c(0, 0.4),
     xlab = "x", ylab = "Dichtefunktion",
     main = "Poisson-Verteilungen nach Lambda")

# Linien für andere Lambdas hinzufügen
lines(x, l2, col = "blue")
lines(x, l4, col = "darkgreen")
lines(x, l9, col = "darkred")

# Legende hinzufügen
legend("topright", legend = c("lambda=1", "lambda=2", "lambda=4", "lambda=9"),
       col = c("black", "blue", "darkgreen", "darkred"),
       lty = 1, cex = 0.8)
```

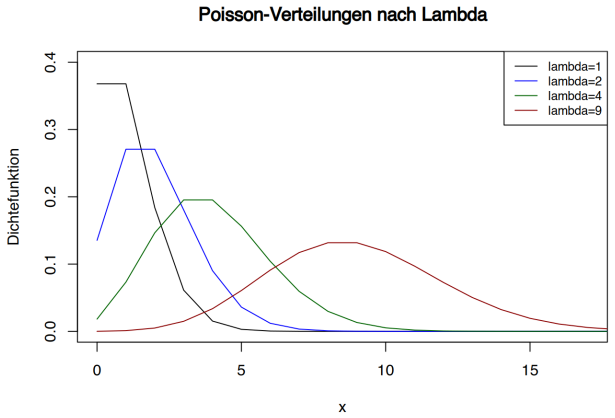


Abb. 1.49

... oder mittels mit `ggplot` (siehe Kapitel 2.4) wie folgt.

```
# Erzeuge x-Werte
x=seq(0,25)
df <- data.frame(  x,
                   l1 = dpois(x, 1),
                   l2 = dpois(x, 2),
                   l4 = dpois(x, 4),
                   l9 = dpois(x, 9)
)
```

```
9
10    # erzeuge eine long-table
11    df <- pivot_longer(df, cols=c(l1, l2, l4, l9))
12
13    # plot vorbereiten
14    p <- ggplot(data=df, aes(x, value, fill=name)) +
15      xlim(0,17) + ylim(0, 0.4) +
16      xlab("x") + ylab("Dichtefunktion") +
17      ggtitle("Poisson-Verteilungen", subtitle = "nach Lambda")
18    p + geom_line(aes(col=name,linetype=name))+
19      labs(col="Lambda",linetype="")
```

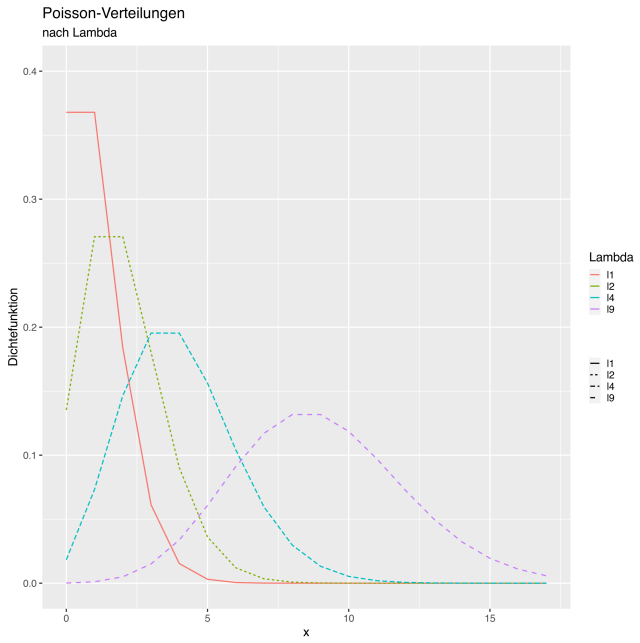


Abb. 1.50

Für weitere Plot-Beispiele zur Poisson-Verteilung siehe Abschnitt 3.2.5 auf Seite 442.

1.2.4.5 Binomial-Verteilung

Die Binomial-Verteilung ist mit den oben beschriebenen Funktionen in R implementiert. Diese heissen entsprechend `dbinom()`, `pbinom()`, `qbinom()` und `rbinom()` und funktionieren genau wie ihre Pendants der Standardnormalverteilung. Als Parameter werden übergeben:

- `x` = *günstige* Werte
- `n` = Anzahl der Beobachtungen
- `prob` = Wahrscheinleichkeit der Ereignisse

Die Wahrscheinlichkeit, dass ich in einem 4-tägigen Seminar mit insgesamt 10 Teilnehmern 2mal das Tagesprotokoll schreiben muss[4] errechnet sich wie folgt:

```
# Wahrscheinlichkeit, insgesamt 2mal das Tagesprotokoll
# schreiben zu müssen,
# bei 10 Teilnehmern und 4 Seminartagen
dbinom(x=2, size=4, prob=0.1)
```

```
[1] 0.0486
```

Die Wahrscheinlichkeitswerte können für jedes Ereignis "`Protokoll schreiben`" errechnet und geplottet werden:

```
# Erzeuge Ereignisraum
protokoll <- c(0:4)

# überführe in Datenframe und berechne alle
# Wahrscheinlichkeiten, das Tagesprotokoll schreiben zu müssen,
# bei 10 Teilnehmern an 4 Seminartagen
df <- data.frame(protokoll, y=dbinom(x=protokoll, size=4, prob=0.1)*100)

# plotten
ggplot(df, aes(x=protokoll, y=y)) + geom_bar(stat="identity", col="black",fill="skyblue3")+
  geom_text(aes(label = y),size = 5,vjust = -0.5)+
  xlab("Anzahl Protokolle")+
  ylab("Wahrscheinlichkeit")+ ylim(0,100)
```

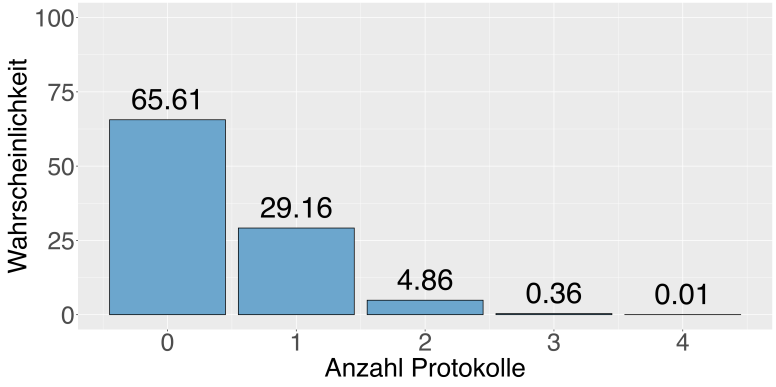


Abb. 1.51

Die Wahrscheinlichkeit, bei 10 Münzwürfen k-mal `Kopf` zu werfen, errechnet sich wie folgt:

```
# Ereignisraum erzeugen
münze <- c(1:10)

# überführe in Datenframe und berechne
# alle Wahrscheinlichkeiten, k-mal "Kopf" zu werfen
df <- data.frame(münze, y=round(dbinom(x=münze, size=10, prob=0.5)*100,1))

# plotten
ggplot(df, aes(x=münze, y=y)) +
  geom_bar(stat="identity", col="black",fill="skyblue3")+
  geom_text(aes(label = y),size = 5,vjust = -0.5)+
  xlab("Anzahl 'Kopf'")+
  ylab("Wahrscheinlichkeit")+ ylim(0,33) +
  scale_x_continuous(breaks = 0:10)
```

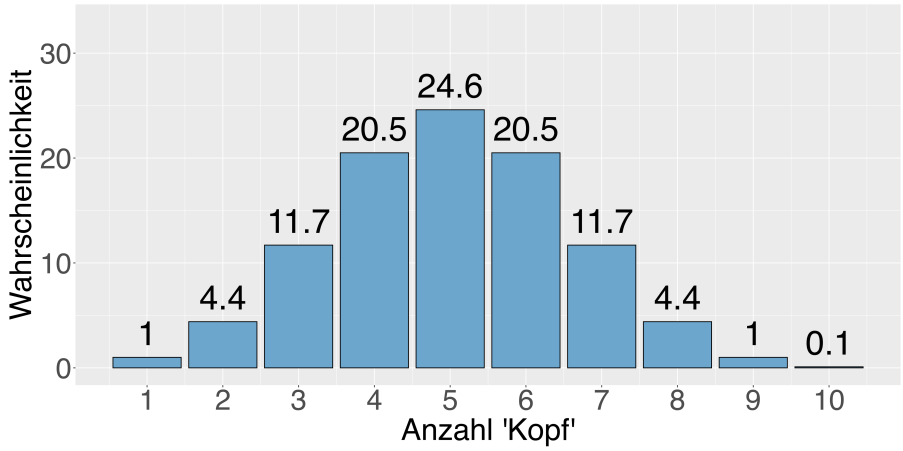


Abb. 1.52

1.2.5 Eigene Funktionen programmieren

In R lassen sich auch eigene Funktionen integrieren. Die wohl "einfachste" Methode hierbei ist, die Funktionen in der R-Sprache selber zu schreiben. Es besteht aber auch die Möglichkeit, neue Funktionen in Programmiersprachen wie z.B. C++ zu programmieren.

Zum Erstellen einer eigenen Funktion steht der Befehl `function()` zur Verfügung. Dieser ist wie folgt aufzurufen:

```
function(Parameter1, Parameter2, ...) {
        FUNKTIONSWEISE
        }
```

In den Klammern direkt nach `function` können Parameter benannt werden, welche dann beim Funktionsaufruf der Funktion zur Verfügung stehen. Innerhalb der geschweiften Klammern wird die eigentliche Funktionsweise geschrieben.
Um eine neue Funktion in R aufrufen zu können, wird die Funktionsweise einem neuen Objekt (z.b. `myfunc`) zugeordnet.

```
myfunc <- function(x,y){ # Neue Funktion mit den Parametern x und y
          z <- x+y       # Die Summe von x und y wird in z gespeichert
          return(z)      # z wird zurückgemeldet
          }
```

Der obigen Dummy-Funktion werden im Funktionsaufruf die Parameter `x` und `y` übergeben. Innerhalb der Funktion wird die Summe aus `x` und `y` gebildet, und per `return()` an die R-Konsole zurückgemeldet. Jetzt können wir diese Funktion aufrufen:

```
myfunc(4,6)
```

```
[1] 10
```

Es besteht ferner die Möglichkeit, Standardwerte für die Parameter zu setzen, welche verwendet werden, falls der Funktionsaufruf ohne Parameter erfolgt.

```
myfunc <- function(x=3,y=7){ # Neue Funktion mit Standardwerten für die Parameter x und y
          z <- x+y           # Die Summe von x und y wird in z gespeichert
          return(z)          # z wird zurückgemeldet
          }
```

Rufen wird die Funktion ohne Parameter auf, so werden die Standardwerte genutzt.

```
# Funktionsaufruf OHNE Parameter
myfunc()
```

```
[1] 10
```

Rufen wir die Funktion mit Parametern auf, werden diese verwendet.

```
myfunc(12,4)
```

```
[1] 16
```

1.2.5.1 Beispiele für Funktionen

Alle Funktionen (und Datensätze) dieses Lehrbuches sind auch im Zusatzpaket `jgsbook` enthalten, siehe Abschnitt 1.2.3.14.1 auf Seite 84.

1.2.5.1.1 z-Transformation

Mit dieser Funktion wird die z-Transformation für gegebene Werte errechnet. Die Funktion folgt der Formel
$z_i = \frac{x_i - \mu}{\sigma}$
Als Standardparameter werden die Werte der Standardnormalverteilung gesetzt.

```
ztrans <- function(x, mu=0, sd=1){
  z = (x-mu)/sd
  return(z)
}

# x ist 120, aus einer normalverteilten Reihe mit
# Mittelwert 118 und Standardabweichung 20
ztrans(120,mu=118,sd=20)
```

```
[1] 0.1
```

1.2.5.1.2 Sensitivität / Spezifität

Die folgende Funktion errechnet Sensitivität, Spezifität sowie positiv- und negativ-prädiktive Werte für gegebene Werte.

- `rp` = Anzahl richtig positive
- `rn` = Anzahl richtig negative
- `fp` = Anzahl falsch positive
- `fn` = Anzahl falsch negative

```
sens.spec <- function(rp, rn, fp, fn){
    x <- data.frame(
              sens=round(rp/(rp+fn)*100, 2),
              spec=round(rn/(rn+fp)*100, 2),
              ppw =round(rp/(rp+fp)*100, 2),
              npw =round(rn/(rn+fn)*100, 2)
              )
   return(x)
 }


 sens.spec(40, 17, 85, 4)
```

```
   sens  spec ppw   npw
1 90.91 16.67  32 80.95
```

1.2.5.1.3 Kenngrößen

Die folgende Funktion gibt die gebräuchlichsten Kenngrößen einer Wertereihe zurück:

```
kenngroessen <- function(werte){
  bla <- data.frame(0)
  bla$modus=paste(as.character(statip::mfv(werte)), collapse="|")
  bla$mean=mean(werte, na.rm=T)
  bla$median=median(werte, na.rm=T)
  bla$p25=quantile(werte,0.25,type=6)
  bla$p75=quantile(werte,0.75,type=6)
  bla$iqr=IQR(werte,type=6)
  bla$sd=sd(werte, na.rm=T)
  bla$var=var(werte, na.rm=T)
  bla$VK= (sd(werte, na.rm=T)/mean(werte,na.rm=T))
  return(bla[-1])
}

# erzeuge zufällige Werte
x <- ceiling(rnorm(100, 10,5))

# Kenngrößen anzeigen
kenngroessen(x)
```

```
  modus  mean median p25 p75 iqr       sd      var        VK
1  6|11 10.65     11   7  14   7 4.710562 22.18939 0.4423063
```

1.2.5.1.4 Häufigkeitstabellen

Die folgende Funktion gibt eine vollständige Häufigkeitstabelle mit absoluten und relativen Häufigkeiten sowie kummulierten Werten zurück.

```
freqTable <- function(werte){
  x <- table(werte)
  tabelle <- data.frame(x)
  tabelle$freqcum <- cumsum(x)
  tabelle$relfreq <- round(x/length(werte)*100,2)
  tabelle$relcum  <- cumsum(round(x/length(werte)*100,2))
  colnames(tabelle) <- c("Wert", "Häufig", "Hkum", "Relativ", "Rkum")
  tabelle$Wert <- as.numeric(as.vector(tabelle$Wert))
  return(tabelle)
}

x <- ceiling(rnorm(13, 10,2))
freqTable(x)
```

```
  Wert Häufig Hkum Relativ   Rkum
1    8      1    1    7.69   7.69
2    9      3    4   23.08  30.77
3   10      4    8   30.77  61.54
4   12      4   12   30.77  92.31
5   13      1   13    7.69 100.00
```

1.2.5.2 Bedingungen

Innerhalb der Funktion können Variablenbedingungen mit dem `if()`-Befehl abgefragt werden. Der Aufruf erfolgt etwa so:

```
if(VARIABLENBEDINGUNG) {FUNKTIONSWEISE}
```

Innerhalb der Klammern des `if()`-Befehls werden die Variablenbedingungen gesetzt. Falls diese Bedingungen erfüllt sind, wird der Code innerhalb der geschweiften Klammern ausgeführt. Folgende Bedingungen können abgefragt werden

Zeichen	Bedingung
==	gleich
!=	ungleich
<	kleiner
<=	kleiner-gleich
>	größer
>=	größer-gleich
&	UND
|	ODER

Innerhalb von Funktionen kann man dies wie folgt anwenden

```
myfunc <- function(x=3,y=7){
          z <- x+y                      # Die Summe von x und y wird in z gespeichert
          if(z>20) {                    # Abfrage, ob die Summer größer als 20 ist
             z <- "wow, bist du gross"  # wenn ja, dann schreibe einen Text in das Objekt z
             }
          return(z)                     # z wird zurückgemeldet
          }
```

Mehrere Bedingungen können verknüpft werden, z.B. so:

```
myfunc <- function(x=3,y=7){
          if(x<0 & y<0) {          # Abfrage, ob x und y negativ sind
             x <- x*(-1)           # wenn ja, dann mache beide positiv
             y <- y*(-1)
             }
          z <- x+y                 # Die Summe von x und y wird in z gespeichert
          if(z==0 | z>50)  {       # Abfrage, ob z gleich 0 oder größer 50 ist
            z <- "Summe ist 0 oder größer 50" # wenn ja, schreibe einen Text...
            }
          return(z)                # z wird zurückgemeldet
          }
```

1.2.5.2.1 Beispiel Zusatzpakete

Die folgende Funktion installiert die vorgegebenen Pakete, sofern sie noch nicht installiert sind. Dies ist hilfreich, wenn z.B. auf eine höhere R-Version geupdatet wurde, und alle Zusatzpakete neu installiert werden müssen.

```
install.my.packages <- function(){
  # Liste meiner favorisierten Pakete
  my_packages <- c("blogdown", "bookdown",
                   "car",
                   "foreign",
                   "gghighlight", "ggplot2",
                   "haven",
                   "likert",
```

```
                  "prettyR", "psych",
                  "reshape", "reshape2",
                  "samplingbook", "scales", "statip",
                  "tidyverse",
                  "VGAM",
                  "xtable"
)

#--------------
# Überprüfe, ob die Pakete bereits installiert sind
not_installed <- my_packages[!(my_packages %in% installed.packages()[ , "Package"])]
# installiere solche, die noch nicht installiert sind
if(length(not_installed)) install.packages(not_installed, dependencies = TRUE)
return(paste(length(not_installed), "Pakete wurden installiert (plus dependencies)."))
}
```

Die Funktion kann dann wie folgt aufgerufen werden:

```
install.my.packages()
```

```
[1] "0 Pakete wurden installiert (plus dependencies)."
```

Bei mir ist alles up-to-date, so dass kein Paket installiert werden muss.

Mit einer leichten Änderung können weitere Pakete an den Parameter `p` übergeben werden:

```
install.my.packages <- function(p=""){
  # Liste meiner favorisierten Pakete
  my_packages <- c("blogdown", "bookdown",
                   "car",
                   "foreign",
                   "gghighlight", "ggplot2",
                   "haven",
                   "likert",
                   "prettyR", "psych",
                   "reshape", "reshape2",
                   "samplingbook", "scales", "statip",
                   "tidyverse",
                   "VGAM",
                   "xtable"
)

# Falls Pakete über den Parameter "p" übergeben wurden,
# füge sie der Liste hinzu
if(p!=""){
  my_packages <- c(p, my_packages)
    }
#--------------
# Überprüfe, ob die Pakete bereits installiert sind
not_installed <- my_packages[!(my_packages %in% installed.packages()[ , "Package"])]
# installiere solche, die noch nicht installiert sind
if(length(not_installed)) install.packages(not_installed, dependencies = TRUE)
return(paste(length(not_installed), "Pakete wurden installiert (plus dependencies)."))
}
```

Der Funktion kann so ein Vektor weiterer Pakete übergeben werden, die zusätzlich zur vorgegebenen Liste installiert werden, falls sie noch nicht installiert sind:

```
install.my.packages(c("ggpubr", "qqplotr"))
```

```
[1] "2 Pakete wurden installiert (plus dependencies)."
```

1.2.5.2.2 Beispiel verschiedene lineare Modelle vergleichen

Wir programmieren eine Funktion, welche verschiedene lineare Modelle vergleicht. Hierbei sollen die Modelle mittels Bestimmtheitsmaß (R^2) verglichen werden, und auf Wunsch können Vorhersagewerte erzeugt werden.

```
  compare.lm <- function(dep, ind, predict=FALSE, steps=0.01){
    # erzeuge lineares Modell
    lin <- lm(dep ~ ind)
    # erzeuge quadratisches Modell
    q <- lm(dep ~ ind + I(ind^2))
    # erzeuge kubisches Modell
    c <- lm(dep ~ ind + I(ind^2) + I(ind^3))
    # erzeuge exponentielles Modell
    e <- lm(log(dep) ~ ind)
    # erzeuge logarithmisches Modell
    l <- lm(dep ~ log(ind))
    # erzeuge sigmoidales Modell
    s <- lm(log(dep) ~ I(1/ind))
    # erzeuge Potenzmodell
```

```
  p <- lm(log(dep) ~ log(ind))
  # erzeuge logistische Funktion
  logit <- nls(dep ~ SSlogis(ind, Asym, xmid, scal),
               start = list(Asym = max(dep),
               xmid = mean(ind), scal = 1))

  # Speichere die R²-Ergebnisse in einem Datenframe
  result <- data.frame(Modell = c("linear", "quadratisch", "kubisch", "exponentiell",
                                  "logarithmisch", "sigmoidal", "potenz"),
                       R.square = c(summary(lin)$r.squared,
                                    summary(q)$r.squared,
                                    summary(c)$r.squared,
                                    summary(e)$r.squared,
                                    summary(l)$r.squared,
                                    summary(s)$r.squared,
                                    summary(p)$r.squared))
  # Sollen Vorhersagewerte erzeugt werden?
  if(predict==TRUE){
    # x-Werte
    pred.x <- seq(min(ind), max(ind), steps)
    # lineare Vorhersagewerte
    pred.lin <- predict(lin, list(ind=pred.x))
    # quadratische Vorhersagewerte
    pred.q <- predict(q, list(ind=pred.x))
    # kubische Vorhersagewerte
    pred.c <- predict(c, list(ind=pred.x))
    # exponentielle Vorhersagewerte
    pred.e <- exp(predict(e, list(ind=pred.x)))
    # logarithmische Vorhersagewerte
    pred.l <- predict(l, list(ind=pred.x))
    # sigmoidale Vorhersagewerte
    pred.s <- predict(s, list(ind=pred.x))
    pred.s[-1] <- exp(pred.s[-1])
    # potenzvorhersagewerte
    pred.p <- exp(predict(p, list(ind=pred.x)))
    # logistische Vorhersagewerte
    pred.logit <- predict(logit, list(ind=pred.x))

    # Vorhersagewerte zurückgeben
    return(data.frame(x = ind, y=dep,
                      pred.x = pred.x,
                      line = pred.lin,
                      quad = pred.q,
                      cube = pred.c,
                      expo = pred.e,
                      loga = pred.l,
                      sigm = pred.s,
                      power = pred.p,
                      logistic = pred.logit))
  } else {
    return(result[order(result$R.square, decreasing = TRUE),])
  }
}

```

Probieren wir die Funktion aus:

```
# Dummy-Daten
x <- c(6, 9, 12, 14, 30, 35, 40, 47, 51, 55, 60)
y <- c(14, 28, 50, 70, 89, 94, 90, 75, 59, 44, 27)
# Modellvergleich
compare.lm(y, x)
```

```
         Modell   R.square
3       kubisch 0.97480010
2   quadratisch 0.96019615
6     sigmoidal 0.47930323
```

```
7        potenz 0.26118539
5 logarithmisch 0.18835564
4  exponentiell 0.08309738
1        linear 0.04459826
```

```
# Vorhersagewerte
head(compare.lm(y, x, predict=TRUE))
```

```
  x  y pred.x     line     quad     cube     expo     loga     sigm     power   logistic
1  6 14   6.00 50.24169 18.56932 13.44564 39.81109 36.63143  2.949079 28.94804 10.14320
2  9 28   6.01 50.24467 18.62461 13.51854 39.81464 36.65621 19.134154 28.96656 10.18668
3 12 50   6.02 50.24765 18.67988 13.59140 39.81820 36.68095 19.179906 28.98506 10.23031
4 14 70   6.03 50.25063 18.73513 13.66423 39.82175 36.70564 19.225615 29.00354 10.27410
5 30 89   6.04 50.25361 18.79036 13.73702 39.82530 36.73029 19.271282 29.02200 10.31804
6 35 94   6.05 50.25660 18.84557 13.80977 39.82885 36.75491 19.316905 29.04044 10.36214
```

Mit den Vorhersagedaten können alle Modelle in eine Punktwolke geplottet werden.

```
df <- compare.lm(y, x, predict=TRUE)

ggplot(df) +
  geom_point(aes(x=x, y=y)) +
  scale_linetype("Regression model") +
  geom_line(aes(x=pred.x, y=line, linetype="lineal"), col="blue") +
  geom_line(aes(x=pred.x, y=quad, linetype="quadratisch"), col="skyblue") +
  geom_line(aes(x=pred.x, y=expo, linetype="exponential"), col="seagreen") +
  geom_line(aes(x=pred.x, y=loga, linetype="logarithmisch"), col="coral") +
  geom_line(aes(x=pred.x, y=sigm, linetype="sigmoidal"), col="violet") +
  geom_line(aes(x=pred.x, y=cube, linetype="kubisch"), col="burlywood") +
  geom_line(aes(x=pred.x, y=power, linetype="potenz"), col="maroon")  +
  geom_line(aes(x=pred.x, y=logistic, linetype="logis"), col="cornflowerblue")
```

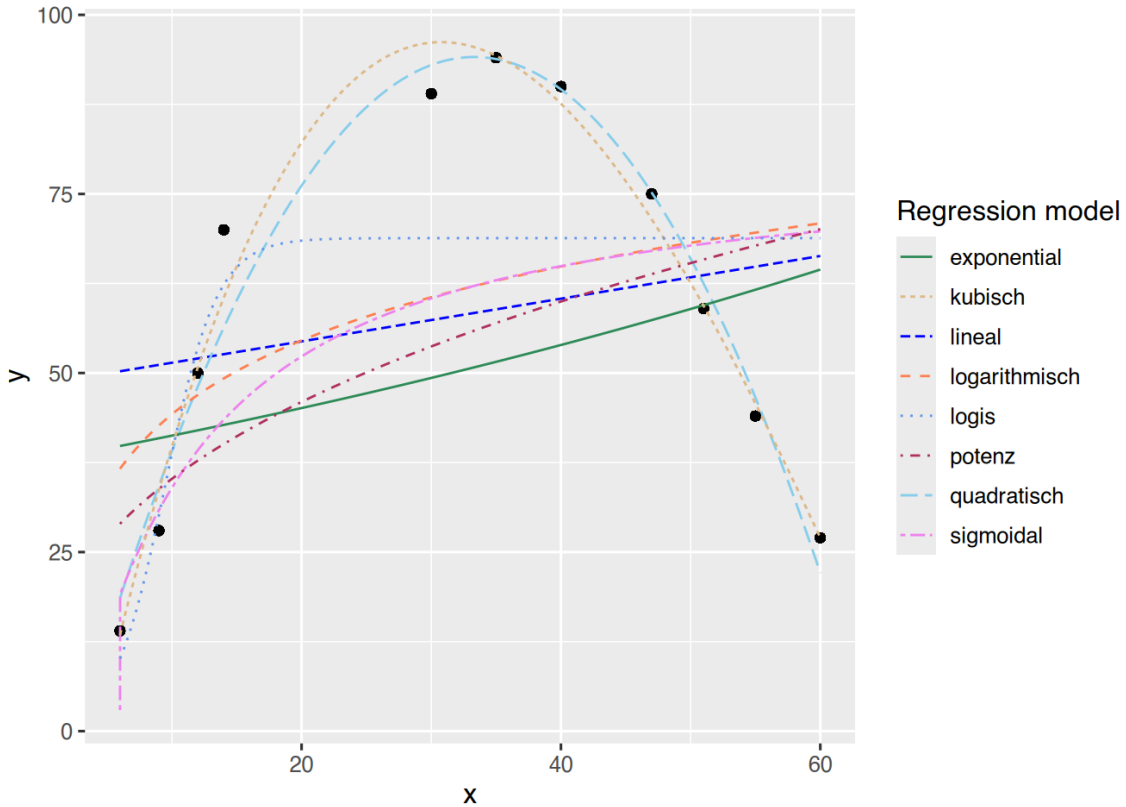


Abb. 1.53

Die Funktion ist auch im `jgsbook`-Zusatzpaket enthalten.

1.2.5.3 Funktionen in Dateien speichern

Eigene Funktionen sind Objekte wie alle anderen Variablen im Workspace. Wenn sie nicht abgespeichert werden, stehen Sie u.U. nach einem Neustart (oder wenn der Workspace überschrieben oder geleert wird) nicht mehr zur Verfügung.

Es bietet sich an, die eigenen Funktionen in einer `.R`-Datei zu speichern. Mit dem `source()`-Befehl kann die Datei dann in neuen Projekten eingelesen werden, und die Funktionen stehen zur Verfügung.

```
1 source("/Pfad/zu/MeineFunktionen.R")
```

1.2.6 Funktionen über Datenreihen anwenden

1.2.6.1 `apply()`

Eine wichtige Funktion zur Datenmanipulation ist die Funktion `apply()` (wie auch ihre Tochterfunktionen `sapply()`, `lapply()` und `tapply()`). Mit ihr können beliebige Funktionen auf die Reihen und/oder Spalten eines Datenframes (oder einer Matrix) angewendet werden. Dadurch erlaubt sie es in vielen Fällen, Loop-Schleifen und Redundanzen zu vermeiden.
Zur Verdeutlichung sei folgendes Beispiel gegeben:

```
# erzeuge Testdatensatz
df <- data.frame(Name = c("Hans", "Gerda", "Kurt", "Maria"),
                 Alter = c(56, 59, 58, 58),
                 Groesse = c(178, 172, 181, 166),
                 Gewicht = c(105, 67, 95, 89))
# anzeigen
df
```

```
   Name Alter Groesse Gewicht
1  Hans    56     178     105
2 Gerda    59     172      67
3  Kurt    58     181      95
4 Maria    58     166      89
```

Angenommen, wir wollen die Mittelwerte der drei numerischen Variablen bestimmen, dann könnten wir so vorgehen:

```
mean(df$Alter)
```

```
[1] 57.75
```

```
mean(df$Groesse)
```

```
[1] 174.25
```

```
mean(df$Gewicht)
```

```
[1] 89
```

Die Funktion `apply()` erlaubt es, die Werte in nur einem Aufruf zu erzeugen, indem die Funktion `mean()` auf die Werte jeder Spalte angewendet wird.

```
apply(df[,-1], MARGIN=2, FUN=mean)
```

```
  Alter Groesse Gewicht
  57.75  174.25   89.00
```

Wir übergeben der Funktion zunächst das Datenframe df ohne die Spalte Name (`df[,-1]`). Mit dem Parameter `MARGIN` wird festgelegt, wie `apply()` auf das Datenframe schauen soll:

1. `MARGIN=1` - Reihe für Reihe
2. `MARGIN=2` - Spalte für Spalte
3. `MARGIN=3` - beides zusammen

Über den Parameter `FUN` wird die zu verwendende Funktion (in diesem Falle `mean()`) **ohne Klammern** angegeben.
Möchten wir nun die Standardabweichung der drei Variablen berechnen, lautet der Aufruf entsprechend:

```
1 apply(df[,-1], MARGIN=2, FUN=mean)
```

```
  Alter Groesse Gewicht
  57.75  174.25   89.00
```

Es ist wichtig, die nicht-numerische Variable `Name` vor dem `apply()`-Aufruf herauszufiltern. Denn wenn die zu verwendende Funktion bei nur einer Reihe fehl schlägt, liefert `apply()` nur eine Reihe `NA`s und Warnmeldungen zurück.

```
1 # aufruf mit Spalte "Name" schlägt fehl
2 apply(df, MARGIN=2, FUN=mean)
```

```
Warning in mean.default(newX[, i], ...): Argument ist weder numerisch noch
boolesch: gebe NA zurück

Warning in mean.default(newX[, i], ...): Argument ist weder numerisch noch
boolesch: gebe NA zurück

Warning in mean.default(newX[, i], ...): Argument ist weder numerisch noch
boolesch: gebe NA zurück

Warning in mean.default(newX[, i], ...): Argument ist weder numerisch noch
boolesch: gebe NA zurück

   Name   Alter Groesse Gewicht
     NA      NA      NA      NA
```

Möchten wir die Werte der Probanden beispielsweise aufaddieren, lassen wir per `apply()` die Funktion `sum()` reihenweise über die Daten laufen. Hierzu setzen wir den Parameter `MARGIN=1`.

```
1 # laufe pro-Reihe über die Daten
2 apply(df[,-1], MARGIN=1, FUN=sum)
```

```
[1] 339 298 334 313
```

Wir können jede Funktion per `apply()` auf die Variablen loslassen:

```
# Wurzel aus jedem Wert ziehen
apply(df[,-1], MARGIN=2, sqrt)
```

```
        Alter  Groesse   Gewicht
[1,] 7.483315 13.34166 10.246951
[2,] 7.681146 13.11488  8.185353
[3,] 7.615773 13.45362  9.746794
[4,] 7.615773 12.88410  9.433981
```

Da wir wissen, dass der Parameter `MARGIN` an zweiter Stelle kommt, müssen wir ihn dort nicht immer ausschreiben, sondern können einfach 1, 2 oder 3 schreiben.

```
# Häufigkeitstabelle für jede Variable
# MARGIN=2 muss nicht ausgeschrieben werden
apply(df[,-1], 2, FUN=jgsbook::freqTable)
```

```
$Alter
  Wert Haeufig Hkum Relativ Rkum
1   56       1    1      25   25
2   58       2    3      50   75
3   59       1    4      25  100

$Groesse
  Wert Haeufig Hkum Relativ Rkum
1  166       1    1      25   25
2  172       1    2      25   50
3  178       1    3      25   75
4  181       1    4      25  100

$Gewicht
  Wert Haeufig Hkum Relativ Rkum
1   67       1    1      25   25
2   89       1    2      25   50
3   95       1    3      25   75
4  105       1    4      25  100
```

1.2.6.2 Funktionsparameter spezifizieren

Parameter für die auszuführende Funktion können einfach mit einem Komma angehängt werden.

```
# Führen quantile(x, type=6) aus
apply(df[,-1], MARGIN=2, FUN=quantile, type=6)
```

```
     Alter Groesse Gewicht
0%   56.00  166.00    67.0
25%  56.50  167.50    72.5
50%  58.00  175.00    92.0
75%  58.75  180.25   102.5
100% 59.00  181.00   105.0
```

Das funktioniert auch mit mehreren Parametern.

```r
# Führen quantile(x, type=6) aus
apply(df[,-1], MARGIN=2, FUN=quantile, type=6, probs=c(0.1, 0.6, 0.8))
```

```
    Alter Groesse Gewicht
10%    56     166      67
60%    58     178      95
80%    59     181     105
```

Es ist möglich, die Parameter, welche an die auszuführende Funktion übergeben werden müssen, aus dem Datenframe selbst zu ziehen. Erzeugen wir zur Verdeutlichung folgendes Datenframe:

```r
# Testdaten
df <- data.frame(Name = c("Jerome", "Nadine", "Dominik", "Ella"),
                 Geschwindigkeit = c(120, 25, 32, 80),
                 Strecke = c(200, 34, 50, 100))
# anzeigen
df
```

```
     Name Geschwindigkeit Strecke
1  Jerome             120     200
2  Nadine              25      34
3 Dominik              32      50
4    Ella              80     100
```

Jetzt programmieren wir die Funktion `zeit()`, welche ausrechnet, wie lange die Personen für die Strecke mit der angegebenen Geschwindigkeit benötigt haben. Die Funktion nimmt die Werte `velo` (Geschwindigkeit) und `dist` (Strecke) entgegen.

```r
# Funktion programmieren
zeit <- function(velo, dist){
  # rechne Distanz / Geschwindigkeit
  # runde auf 2 Stellen
  zeit <- round(dist/velo, 2)
  return(zeit)
}
```

Um die Funktion `zeit()` auf jede Reihe des Datenframes anzuwenden, wobei die Werte von `Geschwindigkeit` und `Strecke` jeweils als Parameter übergeben werden, müssen wir zunächst per `MARGIN=1` angeben, dass `apply()` die Daten Reihe-für-Reihe verarbeiten soll.
Der Trick besteht darin, im Paramter `FUN` eine temporäre Funktion `function(line)` anzugeben. Diese ruft ihrerseits die Funktion `zeit()` (diesmal **mit** Klammern) auf, und übergibt die Parameter per `line("NAME_DER_SPALTE")`.

```r
# Übergebe die Werte an die Funktion zeit()
apply(df[,-1], MARGIN=1, function(line) zeit(line["Geschwindigkeit"],
                                             line["Strecke"]))
```

```
[1] 1.67 1.36 1.56 1.25
```

Wenn die Parameter `dist` und `velo` mit ihrem Namen angegeben werden, ist die Reihenfolge der Parameter egal.

```
apply(df[,-1], MARGIN=1, FUN=function(line) zeit(dist=line["Strecke"],
                                                 velo=line["Geschwindigkeit"]))
```

```
[1] 1.67 1.36 1.56 1.25
```

Als temporäre Hilfsfunktionen können verschiedene Arten verwendet werden:

1. `function(x)` - übergibt jeden Wert einzeln und nacheinander jeweils als Parameter an die Funktion, wobei `x` für spezifische Parameter referenziert werden kann (siehe Abschnitt 1.2.6.3 auf Seite 113).

2. `function(line)` - übergibt die Werte der gesamte Datenreihe als Parameter an die Funktion, wobei die einzelnen Variablenwerte per `line("NAME_DER_SPALTE")` referenziert werden können.

In einem weiteren Beispiel wollen wir Fallzahlen für spezifische Werte von `e`, `P` und dem Konfidenzlevel berechnen (siehe Abschnitt 2.2.12.2 auf Seite 316). Hierzu nutzen wir die Funktion `samplingbook::sample.size.prop()`, wobei die Erweiterung `samplingbook::sample.size.prop()$n` nur die Fallzahlen zurückgibt.

```
# beispielhafter Funktionsaufruf
samplingbook::sample.size.prop(e=0.05, P=0.65, level=0.95, N=1500)$n
```

```
[1] 284
```

```
# Erzeuge Kombinationen von e, P und level
df <- data.frame(e = c(0.05, 0.04, 0.03),
                 P = c(0.65, 0.6, 0.5),
                 level = c(0.9, 0.95, 0.99))
# anzeigen
df
```

```
     e    P level
1 0.05 0.65  0.90
2 0.04 0.60  0.95
3 0.03 0.50  0.99
```

```
# Berechne Fallzahlen für jede Reihe (Kombination)
apply(df, 1, function(line) samplingbook::sample.size.prop(e = line["e"],
                                                           P = line["P"],
                                                           level = line["level"])$n
      )
```

```
[1]  247  577 1844
```

1.2.6.3 `sapply()`

Die Tochterfunktion `sapply()` kann auf Vektoren angewendet werden.

```
# erzeuge einen Vektor mit 3 Werten für N
N <- c(10, 30, 100)

# ziehe für jeden Wert die Wurzel
sapply(N, sqrt)
```

```
[1]  3.162278  5.477226 10.000000
```

Wenn die verwendete Funktion Wertereihen zurückgibt, wird das Ergebnis als Liste zurückgegeben.

```
# erzeuge jewils N Zufallszahlen
sapply(N, sample)
```

```
[[1]]
 [1]  5  6 10  7  9  1  2  3  8  4

[[2]]
 [1] 14 27 10 16 26  2 21 28 20 23 12 29 18 19 17  3  8 25 11 22  9 15  6 30 13
[26]  4  1 24  5  7

[[3]]
  [1]  87  45  31  15  48  65  58  24  53  96  27  36  85  41  21  19  63  38
 [19]  11  82  88  59  69  35  80  68  57   9  74  86  83   4  51  32  77  81
 [37]  94  99  43  78   8  30  16  76   6  71  75  17  95  66  26  14  93  13
 [55]  92  33   1  47  49  62   7   5  55  28  91  20  98  18  61  60  50  64
 [73]  72  97  29  70  89  34  22  37  23  73  90 100  46  84  52   2  56  79
 [91]  42  54  44  67  10  12  40  25   3  39
```

Da die Funktion `sample()` jeweils einen Datenvektor zurückgibt, liegt das Ergebnis als Liste vor.

Auch bei `sapply()` werden Funktionsparameter per Komma angehängt.

```
# erzeuge jewils N Zufallszahlen
# mit doppelten
sapply(N, sample, replace=TRUE)
```

```
[[1]]
 [1] 4 7 2 3 4 6 2 6 2 7

[[2]]
 [1]  4  5  7 30  9  8 19  9  8  4 15 21 28 22 12 29 29 28 21  1  1 13  2 14 25
[26] 11  6  4  7 13

[[3]]
  [1]  62  28  38  81  44  88  40  52  48  95  51  40  51  73  62  83  36  20
 [19]  61  79  29  34  74  17  53  75  51  48  44  31  73  33  49  23  83  71
 [37]  99  78  77  73  54  24   3  47  62  99  50  43  56  27  65  75  62  89
 [55]  62  22  75  41  57  66  42  36  23  19  81  96  46   8  91  67  37  17
 [73]  69  74  29  60  72  83  64  78  68  76  25  70  61  75  98  17  14  49
 [91]   6  34  70   1 100  95  86   3   8  47
```

Sollen die Werte des Ausgangsvektors als spezifischer Parameter der auszuführenden Funktion übergeben werden, muss wieder auf eine temporäre Hilfsfunktion `function(x)` zurückgegriffen werden. Im folgenden Aufruf sollen die Werte des Vektors `N` jeweils als Paramenter `N` an die Funktion `samplingbook::sample.size.mean()` übergeben werden.

```
# Rufe für jeden Wert x im Vektor N die Funktion auf,
# wobei Parameter N=x ist
sapply(N, FUN=function(x) samplingbook::sample.size.mean(e=0.05,
                                                         S=0.3,
                                                         level=0.90,
                                                         N=x)
       )
```

```
     [,1]   [,2]   [,3]
call list,4 list,4 list,4
n    10     23     50
```

Per `samplingbook::sample.size.mean()$n` wird nur die benötigte Fallzahl zurückgegeben. Diese Anweisung können wir auch auf `sapply()` übertragen:

```
# mit $n nur Fallzahlen
sapply(N, FUN=function(x) samplingbook::sample.size.mean(e=0.05,
                                                         S=0.3,
                                                         level=0.90,
                                                         N=x)$n
       )
```

```
[1] 10 23 50
```

1.2.6.4 `lapply()`

Die Funktion `lapply()` liefert ihr Ergebnis immer als Liste zurück, wobei für jedes `x` ein eigener Listeneintrag erzeugt wird.

```
# erzeuge einen Vektor mit 3 Werten für N
N <- c(10, 30, 100)

# ziehe für jeden Wert die Wurzel
lapply(N, sqrt)
```

```
[[1]]
[1] 3.162278

[[2]]
[1] 5.477226

[[3]]
[1] 10
```

Mit der Funktion `unlist()` kann die Liste in Vektoren überführt werden.

```
unlist(lapply(N, sqrt))
```

```
[1]  3.162278  5.477226 10.000000
```

1.2.6.5 `tapply()`

Mit der Tochterfunktion `tapply()` lassen sich gruppierte Auswertungen erzeugen. Die Gruppierung erfolgt anhand eines Factors im Datenframe.

```
df <- data.frame(Geschlecht = factor(c("m", "w", "m", "m", "w",
                                       "w", "d", "d", "d")),
                 Alter = c(45, 34, 46, 41, 31, 29, 24, 25, 26))
# anzeigen
df
```

```
  Geschlecht Alter
1          m    45
2          w    34
3          m    46
4          m    41
5          w    31
6          w    29
7          d    24
8          d    25
9          d    26
```

Angenommen, wir möchten nun die Mittelwerte des Alters in Abhängigkeit zum Geschlecht bestimmen, so können wir `tapply()` verwenden.

```
tapply(df$Alter, df$Geschlecht, mean)
```

```
       d        m        w
25.00000 44.00000 31.33333
```

Wie bei den Schwesterfunktionen lassen sich weitere Parameter per Komma übergeben.

```
# Berecne IQR so wie SPSS (type=6)
tapply(df$Alter, df$Geschlecht, IQR, type=6)
```

```
d m w
2 5 5
```

1.3 Markdown

Markdown ist eine "Auszeichnungssprache", die es erlaubt, mit einer recht einfachen Syntax ansprechende Dokumente zu erstellen. Sie wurde dabei so konzipiert, dass schon der *Quellcode* leicht lesbar ist.

In Kombination mit `RMarkdown` oder `quarto` lassen sich Berichte, Artikel, Präsentationen und Bücher erstellen, die  -Code und -Diagramme verwenden.

Mehr Informationen erhalten Sie hier:
https://quarto.org/docs/authoring/markdown-basics.html

1.4 RMarkdown

Im Sommer 2022 wurde `RMarkdown` durch den Nachfolger `quarto` abgelöst, weshalb dieses Kapitel eher historische Informationen enthält. Wenn Sie `RMarkdown` noch nie benutzt haben, sollten Sie dieses Kapitel überspringen und direkt mit `quarto` einsteigen, siehe Kapitel 1.5 auf Seite 127.

`RMarkdown` ermöglicht es, R-Code, R-Output und -Diagramme, sowie Formeln und externe Quellen (z.B. Bilder aus dem Internet) in ansprechenden Dokumenten (z.B. Webseite, Word, Powerpoint, PDF) zu exportieren. Es ist sehr gut dafür geeignet, die eigenen statistischen Auswertungen und Ergebnisse für *Endanwender* (z.b. Auftragsgeber oder Dozenten) darzustellen. Die Funktionalität hat sich so sehr weiterentwickelt, dass mittlerweile auch Abschlussarbeiten (Bachelor, Master) komplett mit `RMarkdown` geschrieben werden.

Tatsächlich ist Abschnitt 2.2.5 zur Ordinalen Regression zunächst mit `RMarkdown` erstellt und veröffentlicht worden[7], das ist aber schon sehr lange her.

Wir erstellen in unserem Projekt `ErsteSchritte` nun das erste Markdown-Dokument. Klicken Sie hierzu wieder oben im Scriptfenster auf das grüne + Symbol und wählen Sie `R Markdown...` (Abbildung 1.54).

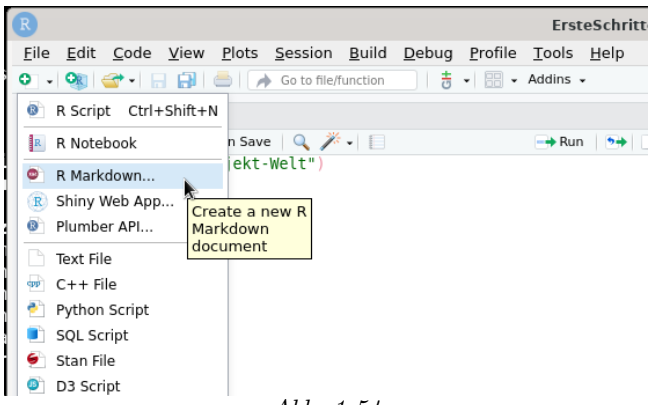


Abb. 1.54

Falls Sie dies zum ersten Mal tun, schlägt `RStudio` die notwendigen Zusatzpaket zur Installation vor (Abbildung 1.55). Die Installation dauert ein paar Minuten, und Sie können den Prozess unten im Konsolfenster verfolgen (Abbildung 1.56, siehe grüner Balken).

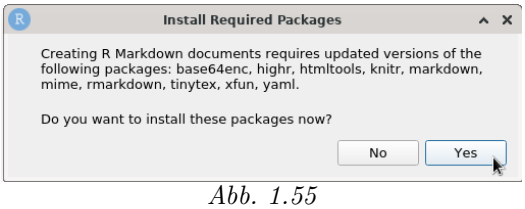


Abb. 1.55

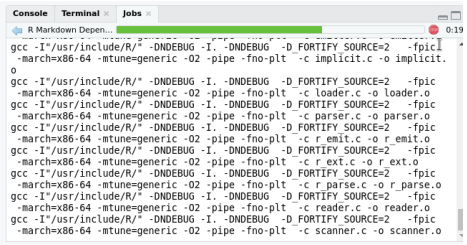


Abb. 1.56: Installationsprozess

Es öffnet sich ein neus Fenster (Abbildung 1.57). Wir erstellen hier ein `Dokument`. Sie sehen aber schon, dass `RMarkdown` weitere Formate (z.B. Präsentationen) ebenfalls unterstützt.

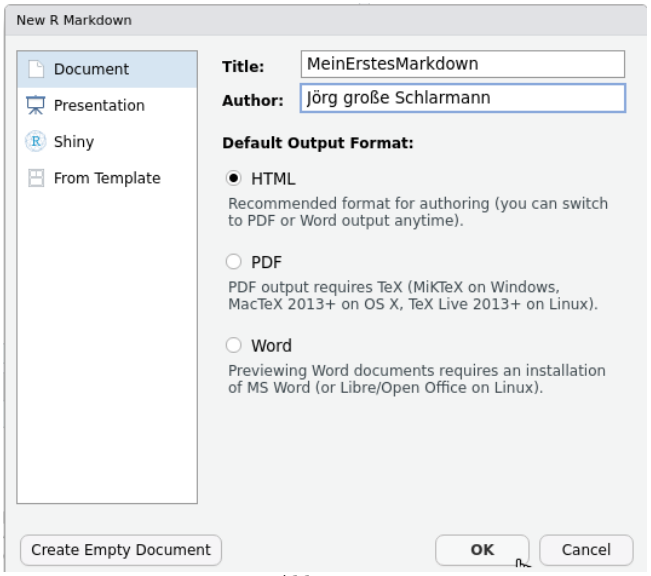


Abb. 1.57

Tragen Sie den Titel des Dokumentes sowie den Autoren ein. Wir belassen in diesem Beispiel die `Default Output Option` auf `HTML`. Sie können aber auch `Word` oder `PDF` wählen.

Es wird nun eine Standardvorlage des Dokumentes erzeugt (Abbildung 1.58). Bevor wir uns dem Inhalt zuwenden speichern wir das Dokument zunächst ab. Klicken Sie hierzu auf das Diskettensymbol (Mauszeiger in Abbildung 1.58) oder drücken Sie die Tastenkombination `[STRG]` + `[S]`. Geben Sie Ihrem Dokument einen Dateinamen mit der Endung `.Rmd` (für ***RM****ark****d****own*). Ich habe meine Datei `MeinErstesMarkdown.Rmd` genannt.

Die Datei ist nun ebenfalls im Dateiverzeichnis zu sehen (Abbildung 1.59) und kann von dort aus geöffnet werden.

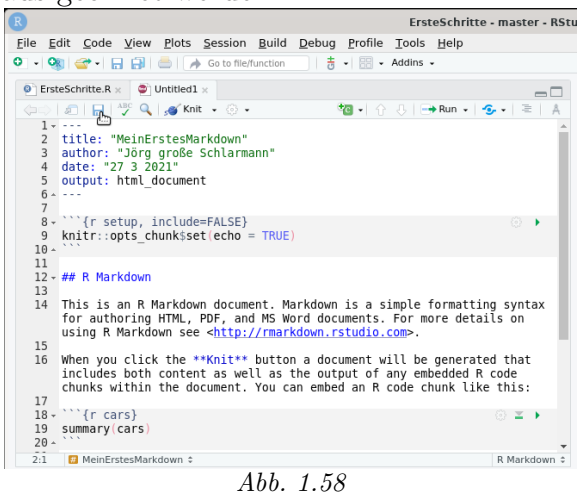


Abb. 1.58

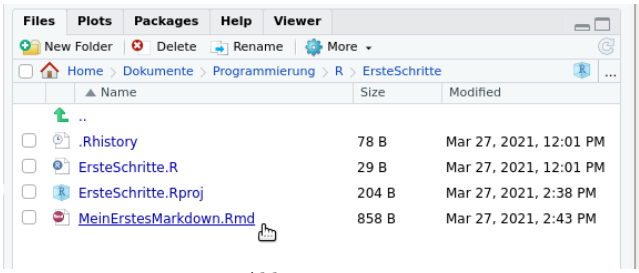


Abb. 1.59

Über dem Scriptfenster befindet sich der `Knit`-Knopf (Abbildung 1.60, Mauszeigen).
Wenn Sie hier klicken, oder die Tastenkombination `[STRG]` + `[SHIFT]` + `[K]` benutzen, wird aus Ihrem `RMarkdown` die gewünschte Ausgabedatei (in diesem Beispiel `html`, also eine Webseite) erzeugt und im Arbeitsverzeichnis gespeichert.

[7] https://rpubs.com/produnis/oreg

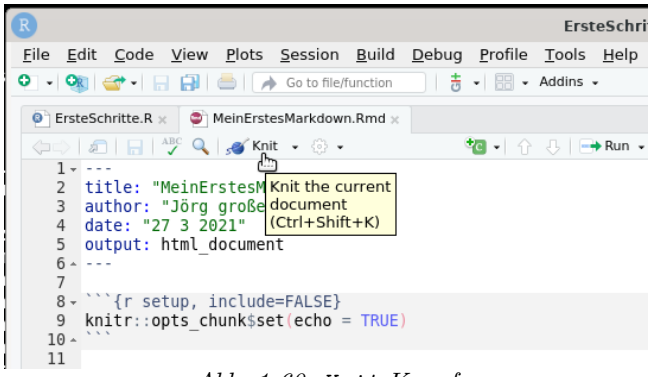


Abb. 1.60: `Knit`-Knopf

Da die Standardvorlage alle wichtigen Dinge enthält können wir direkt auf den `Knit`-Knopf klicken. Das `RMarkdown`-Dokument wird nun "übersetzt", und Sie können den Prozess im Konsolenfenster verfolgen (Abbildung 1.61).

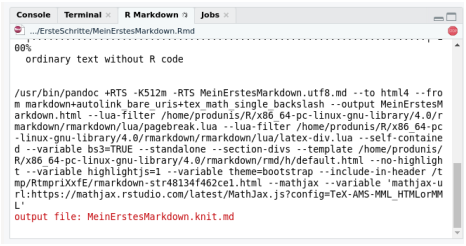


Abb. 1.61: `knit`-Prozess

Ist dieser Schritt erfolgreich, öffnet sich ein neues Fenster mit dem Endresultat (Abbildung 1.62).

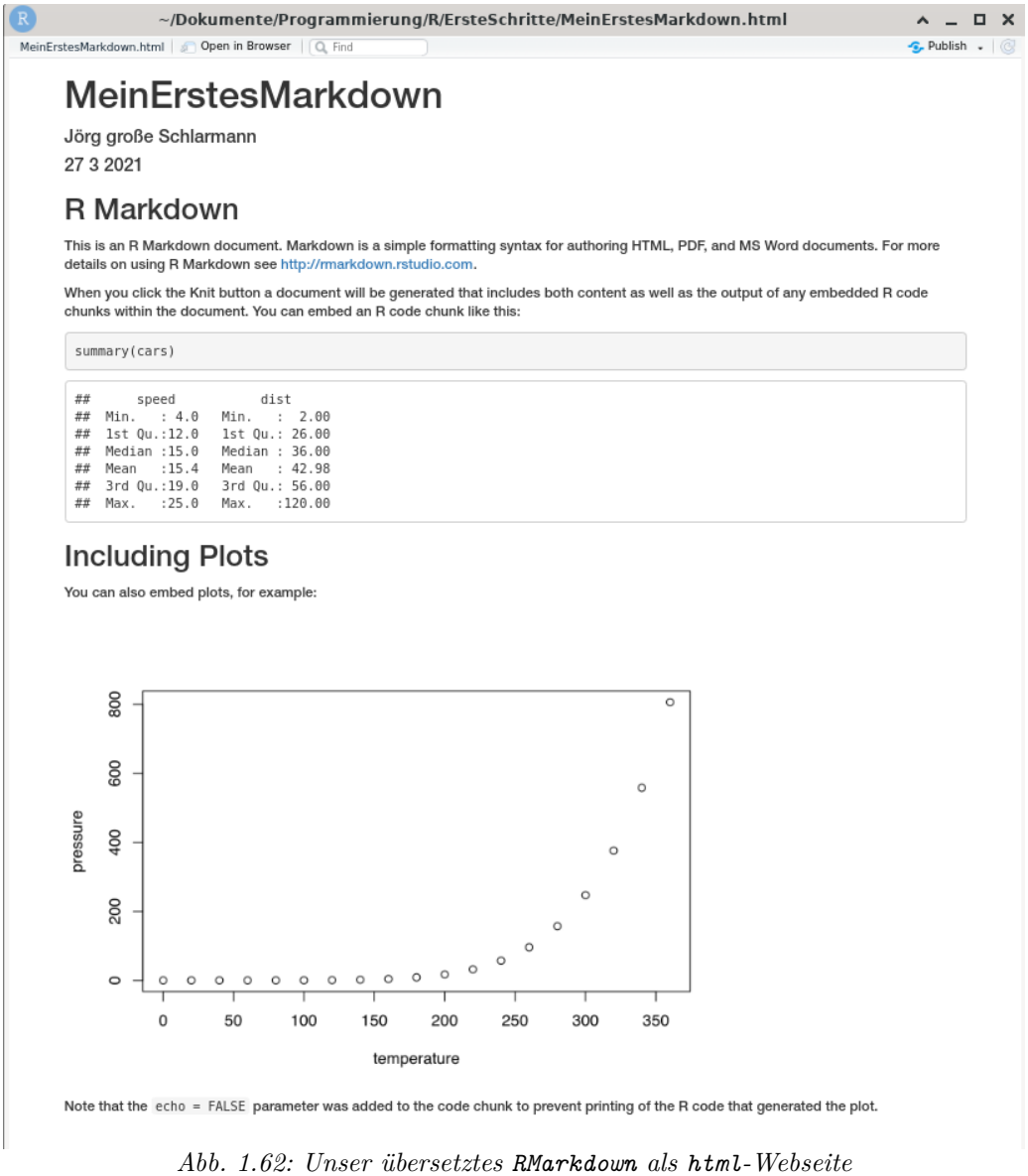


Abb. 1.62: Unser übersetztes `RMarkdown` als `html`-Webseite

In unserem Arbeitsverzeichnis sehen wir ebenfalls die soeben erzeugte Ausgabedatei, in Abbildung 1.63 ist das `MeinErstsMarkdown.html`.

Beachten Sie, dass in der `html`-Datei bereits alle nötigen Ressourcen (z.B. alle Bilder) gespeichert sind. Möchten Sie Ihr `html`-Dokument mit Kolleginnen und Kollegen teilen, brauchen Sie keine weiteren Dateien kopieren oder versenden. Die `html`-Webseite wird auf allen Geräten "gleich" aussehen, und es werden z.B. keine Grafiken oder Bilder fehlen.

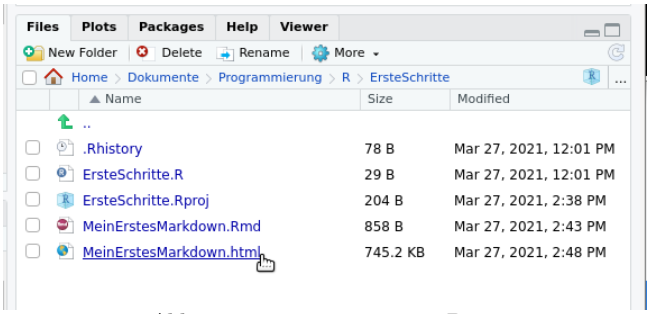


Abb. 1.63: erzeugte `html`*-Datei*

1.4.1 Syntax

Die Syntax von `RMardown` ist recht leicht, und es existiert zudem ein Cheatsheet (siehe Abschnitt 1.8.1 auf Seite 209) unter
`https://raw.githubusercontent.com/rstudio/cheatsheets/master/rmarkdown-2.0.pdf`.
Schauen wir uns unsere eben erzeugte Markdowndatei noch einmal an. Wir löschen alles, bis auf die ersten Zeilen, und beginnen ganz von vorne.

```
---
title: "Ein neuer Titel"
author: "Jörg große Schlarmann"
date: "27 3 2021"
output: html_document
---
```

Dies ist die *Kopfzeile* des Dokuments. Wir können hier den Titel, Autor und das Datum ändern. Unter den drei Strichen `---` beginnen wir unser Dokument. Schreiben Sie folgende Zeilen.

```
---
title: "Dies ist ein Test"
author: "Timm Thaler"
date: "27 3 2021"
output: html_document
---

# Überschrift

Willkommen zu meinem RMarkdown-Dokument

- dies ist ein Punkt
- dies ein weitere

1. erstens
2. zweitens
3. drittens

## Unter-Überschrift

Dies ist **fett**, dies *kursiv*, dies ~~durchgestrichen~~.
```

Drücken Sie den `Knit`-Knopf und erzeugen so einen Testoutput.

Das Zeichen `#` steht für Überschriften und deren Hierarchie. Die Anzahl der `#` legt die Überschriftenhierarchie fest. Aus den Spiegelstrichen ist eine Liste geworden, die Zahlen wurden in eine nummerierte Aufzählungsliste umgewandelt. Schauen Sie im `RMarkdown`-Cheatsheet nach, welche weiteren Zeichen für welchen Effekt stehen.

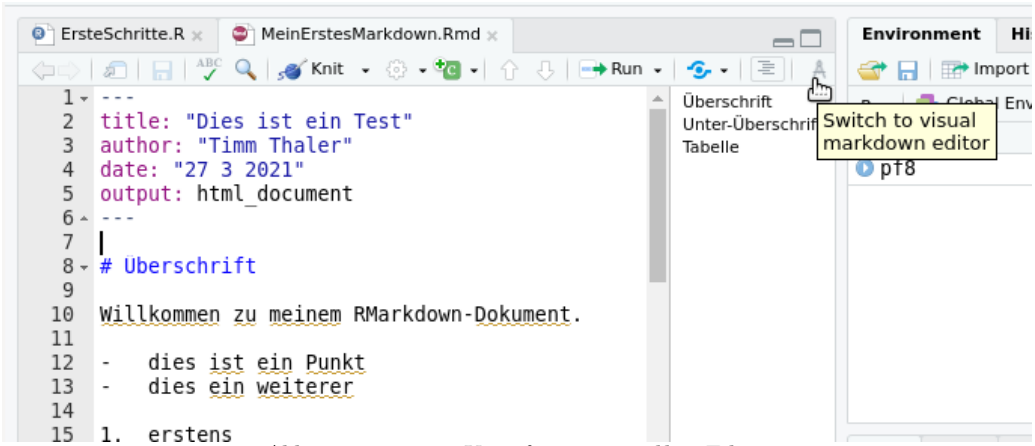


Abb. 1.64: mit A-Knopf zum visueller Editor

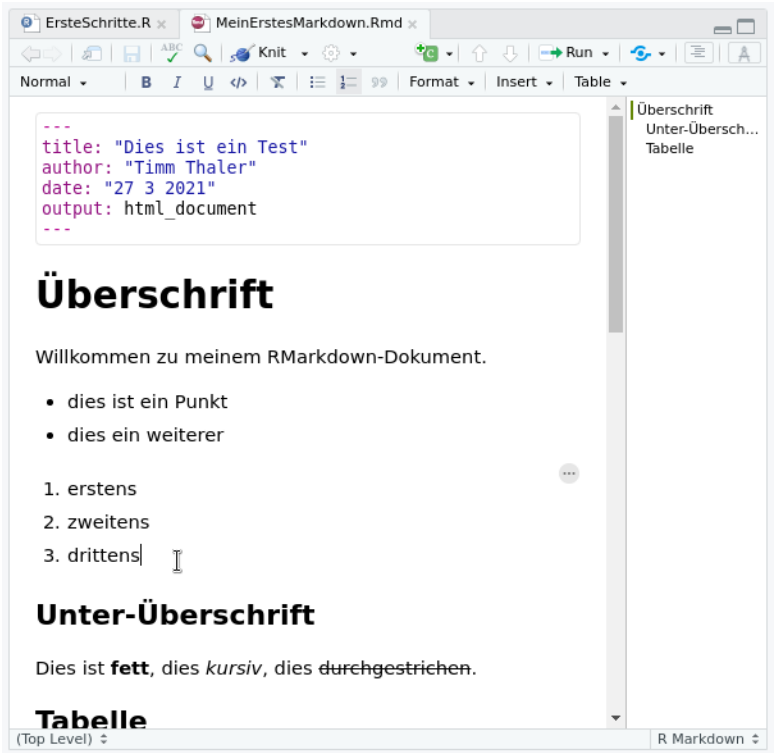


Abb. 1.65

In `RStudio` gibt es auch einen visuellen Markdown-Modus, der so ähnlich funktioniert wie `Word` oder `LibreOffice`. So können Sie Ihren Text "wie gewohnt" gestalten. Klicken Sie hierzu auf das `A`-Symbol rechts oben (siehe Mauszeiger in Abbildung 1.64).

Der Text sieht nun "fast" schon so aus, wie auf dem Endoutput (Abbildung 1.65).

Sie haben zudem oben eine Menüzeile, mit der Sie die Schriftart wechseln, externe Bilder einbinden oder Tabellen erzeugen können.

Durch Klick auf das `A`-Symbol wechseln Sie zwischen visueller und "klassischer" Ansicht.

💀 Achtung, Anfängerfehler! 💀

Denken Sie daran, in die *klassische* Ansicht zurück zu wechseln, bevor Sie Markdown-Code kopieren und einfügen. Dies ist ein häufiger Anfgängerfehler, und der visuelle Editor kann mit dem Code nichts anfangen.

In Markdown können sehr leicht Tabellen erzeugt werden. Versuchen Sie diesen Code aus (vorher vom visuellen Editor zurückwechseln!), und beachten Sie, wie "gut lesbar" und "leicht zu merken" das ist.

```
## Tabelle

| Name   | Beruf       | Alter |
|--------|-------------|-------|
| Heinz  | Kraftfahrer | 46    |
| Lisa   | Professorin | 32    |
| Jürgen | Klemptner   | 48    |

: Meine Tabelle
```

Alles weitere finden Sie auf den erwähnten Cheatsheets.

Kommen wir nun zum eigentlichen Kernpunkt: der Darstellung von R-Code und R-Output.

Hierfür muss im Markdowndokument der Bereich für den R-Code festgelegt werden. Dies erfolgt über die Zeichenkette ``` , die den Codebereicht einschließt.

````
```{r}

# hier steht der R-Code

```
````

Die Anführungsstriche grenzen den Codebereich ein. Dieser wird *Chunk* genannt. Das `{r}` bedeutet, dass R-Code verwendet wird.

Achtung, Anfängerfehler!

Achten Sie darauf, die korrekten `` zu verwenden, und nicht etwa `'''` oder ´´´. Dies ist ein häufiger Anfängerfehler.

In `RStudio` können Sie einfach die Tastenkombination `[STRG]` + `[ALT]` + `[i]` drücken, oder im Scriptfenster oben auf das grüne `c` klicken (Abbildung 1.66).
Beides fügt einen neuen *Chunk* (Codeblock) ein.

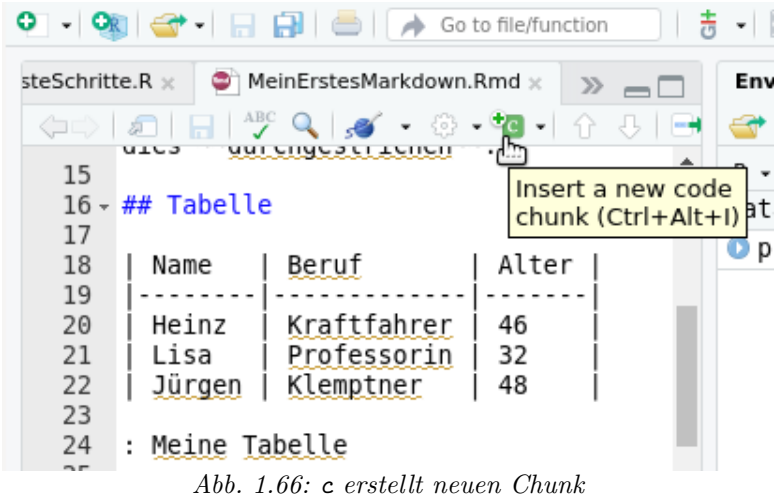


Abb. 1.66: `c` erstellt neuen Chunk

Schreiben Sie in Ihr Dokument:

````
Ich lade den pf8-Datensatz mit:
```{r}
load(url("https://www.produnis.de/R/data/pf8.RData"))
```
````

Wenn Sie den `Knit`-Knopf drücken, wird der `load()`-Befehl im Dokument angezeigt und in der Konsole ausgeführt.

Ergänzen Sie jeweils die nun folgende Zeilen, und schauen Sie sich das Ergebnis nach einem `Knit` an.

````
Ich lade den mma-Datensatz,
der Befehl wird aber im Dokument nicht angezeigt.
```{r include=FALSE}
load(url("https://www.produnis.de/R/data/pf8.RData"))
```
````

Durch den Parameter `include=FALSE` wird der Befehl zwar *intern* ausgeführt, jedoch nicht im Dokument angezeigt. Sollte der Befehl einen Output erzeugen, wird dieser ebenfalls nicht angezeigt.

Wenn Sie nur das Ergebnis anzeigen möchten, aber nicht die -Befehle, setzen Sie den Parameter `echo=FALSE`.

````
Wir plotten Alter und gewicht aus dem pf8-Datensatz
```{r echo=FALSE}
plot(pf8$Alter, pf8$Gewicht)
```
````

In der "Standardeinstellung" werden immer Befehl und Output des Chunks im Dokument angezeigt. Sie können dieses Verhalten auch erzwingen durch den Parameter `echo=TRUE`

````
Wir plotten Alter und Gewicht aus dem pf8-Datensatz
```{r echo=TRUE}
plot(pf8$Alter, pf8$Gewicht)
```
````

nur Code, kein Output	`results="hide"`
nur Code, nicht ausführen	`eval=FALSE`
kein Code, nur Output	`echo=FALSE`
zeige beides	`echo=TRUE`
zeige nichts	`include=FALSE`
mache auch nichts	`eval=FALSE, include=FALSE`

Tabelle 1.1: Darstellungsoptionen der Chunks

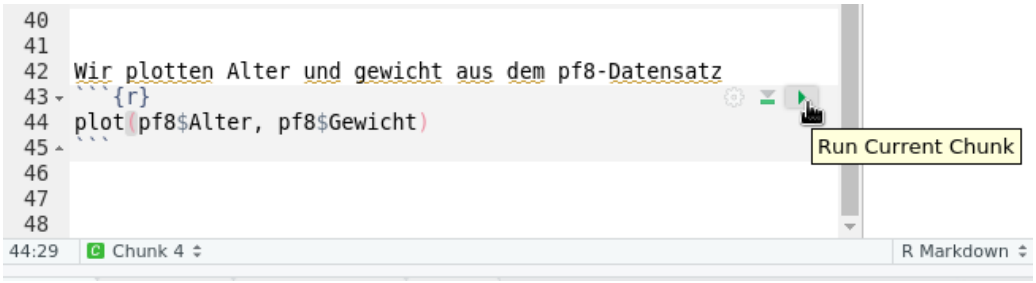


Abb. 1.67: Chunks können im Editor "aktiviert" werden

Die *Chunks* werden im Editorfenster farblich hervorgehoben (Abbildung 1.67).
Sie haben hier die Möglichkeit, den Code "ohne `Knit`" auszuführen und sich das Ergebnis anzeigen zu lassen. Klicken Sie dazu rechts auf das grüne Dreieck.

In diesem Beispiel wird der Plot nun direkt im Markdownfenster angezeigt (Abbildung 1.68). Dies ist wirklich hilfreich beim Erstellen des Dokuments, denn Sie müssen nicht immer erst das gesamte Dokument knitten oder überhaupt geöffnet haben, um zu sehen, was R an dieser Stelle ausgeben würde.

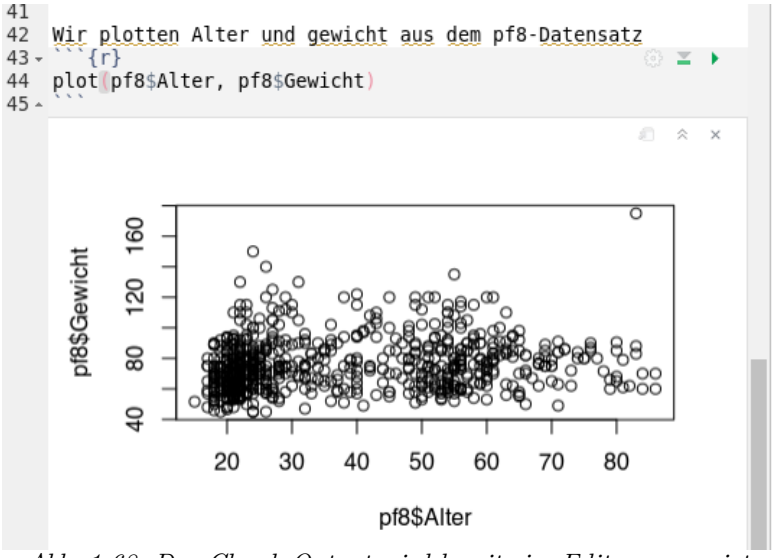


Abb. 1.68: Der Chunk-Output wird bereits im Editor angezeigt

Über das Zahnrad gelangen Sie zu weiteren Einstellungsmöglichkeiten. So können Sie die Chunks internet mit Namen versehen, so dass Sie später im Dokument darauf referenzieren können.

Die vielen weiteren Möglichkeiten von `RMarkdown` werden in diesem Buch nicht weiter behandelt,

weitere Informationen erhalten Sie im RMarkdown-Cheatsheet (siehe Seite 209) oder im frei verfügbaren Buch *R Markdown: The Definitive Guide*[5], welches Sie unter https://bookdown.org/yihui/rmarkdown/ lesen können.

Grundsätzlich sollten Sie es so halten, dass Sie in den Scriptdateien Ihre Auswertungen ausprobieren und mit den Befehlen "herumspielen". Die Quintessenz Ihrer Analyseschritte können Sie dann in "aufgräumter" Form in einem RMarkdown festhalten und mit anderen teilen.

1.5 quarto



quarto ermöglicht es, -Code, -Output und -Diagramme, sowie Formeln und externe Quellen (z.B. Bilder aus dem Internet) in ansprechenden Dokumenten (z.B. Webseite, Word, Powerpoint, PDF) zu exportieren. Es ist sehr gut dafür geeignet, die eigenen statistischen Auswertungen und Ergebnisse für Endanwender (z.b. Auftragsgeber oder Dozenten) darzustellen.

quarto ist der Nachfolger von RMarkdown und wurde 2022 veröffentlicht. Basierend auf den Erkenntnissen der letzten 10 Jahre mit RMarkdown wurde es von Grund auf neu aufgebaut, um mehr Sprachen und Umgebungen zu unterstützen.

quarto unterstützt die Publikationsformate:

- Artikel, Berichte
- Präsentationen
- Webseiten, Blogs
- Bücher

in den Ausgabeformaten HTML, PDF, ePub und Office.

Dieses Buch wurde jahrelang mit RMarkdown (bookdown) erzeugt und ist im September 2022 auf quarto umgezogen.

Mehr Infos erhalten Sie unter https://www.quarto.org.

1.5.1 Installation

In einer aktuellen Installation von RStudio ist quarto bereits enthalten.
Unter Linux muss zusätzlich das Paket quarto-cli installiert werden. Für Ubuntu steht ein .deb-Paket unter https://quarto.org/docs/get-started/ bereit, unter Archlinux lautet der Befehl:

```
yay -S quarto-cli-bin
```

1.5.2 Dokument erstellen

Wir erstellen in unserem Projekt ErsteSchritte nun das erste Markdown-Dokument. Klicken Sie hierzu wieder oben im Scriptfenster auf das grüne + Symbol und wählen Sie Quarto Document, siehe Abbildung 1.69.

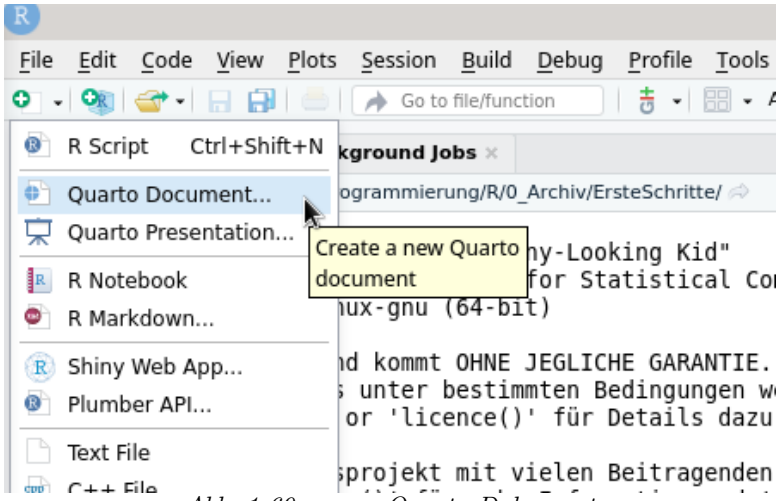


Abb. 1.69: neues Quarto Dokument

Falls Sie dies zum ersten Mal tun, schlägt `RStudio` die notwendigen Zusatzpaket zur Installation vor. Die Installation dauert ein paar Minuten, und Sie können den Prozess unten im Konsolfenster verfolgen.

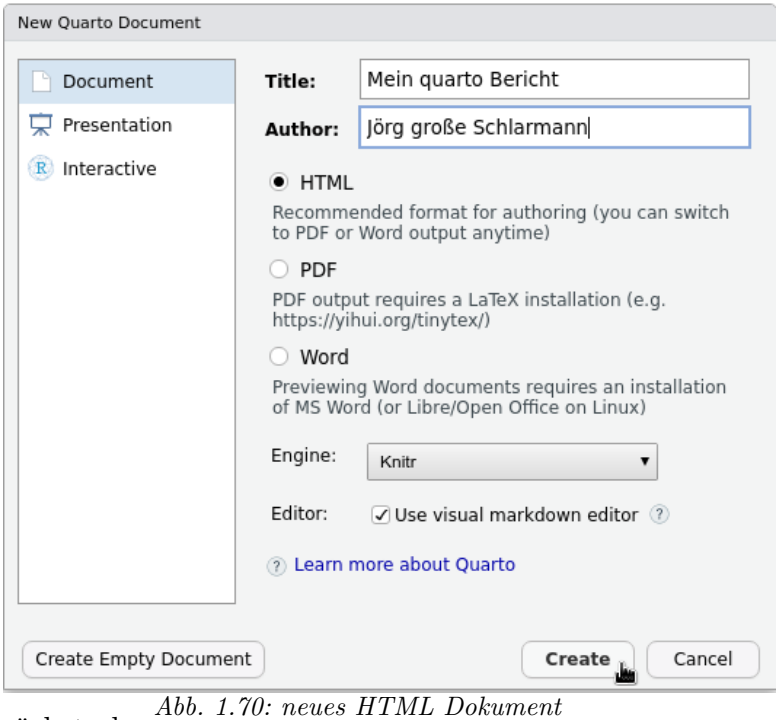


Abb. 1.70: neues HTML Dokument

Es öffnet sich ein neues Fenster (Abbildung 1.70).
Wir erstellen hier ein Dokument, Sie sehen aber schon, dass `quarto` weitere Formate (z.B. Präsentationen) ebenfalls unterstützt. Tragen Sie den Titel des Dokumentes sowie den Autoren ein. Wir belassen in diesem Beispiel die Default Output Option auf `HTML`.
Es wird nun eine Standardvorlage des Dokumentes erzeugt (Abbildung 1.71).
Bevor wir uns dem Inhalt zuwenden speichern wir das Dokument zunächst ab.
Klicken Sie hierzu auf das Diskettensymbol (Mauszeiger in Abbildung 1.71) oder drücken Sie die Tastenkombination `[STRG]` + `[S]`. Geben Sie Ihrem Dokument einen Dateinamen mit der Endung `.qmd` (für **q**uarto-**M**ark**d**own). Ich habe meine Datei `Mein-quarto-Dokument.qmd` genannt.

Die Datei ist nun ebenfalls im Dateiverzeichnis zu sehen (Abbildung 1.72) und kann von dort aus geöffnet werden.

Über dem Scriptfenster befindet sich der `Render`-Knopf (Abbildung 1.73, Mauszeiger).

Wenn Sie hier klicken, oder die Tastenkombination `[STRG]` + `[SHIFT]` + `[K]` benutzen, wird aus Ihrem `quarto`-Markdown die gewünschte Ausgabedatei (in diesem Beispiel `html`, also eine Webseite) erzeugt und im Arbeitsverzeichnis gespeichert.

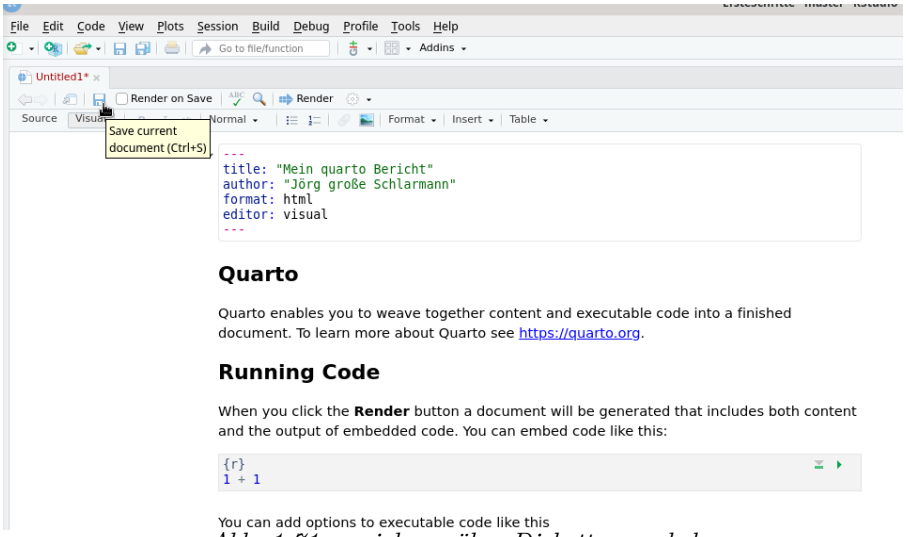


Abb. 1.71: speichern über Diskettensymbol

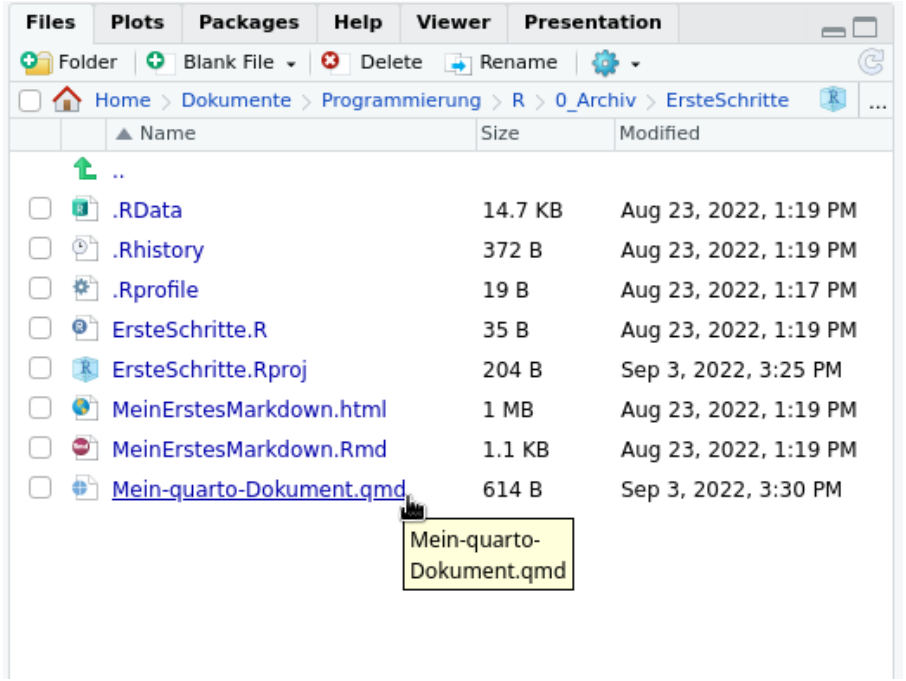


Abb. 1.72: quarto-Dokument im Arbeitsverzeichnis

Da die Standardvorlage alle wichtigen Dinge enthält können wir direkt auf den `Render`-Knopf klicken. Das `quarto`- Dokument wird nun "übersetzt", und Sie können den Prozess im Konsolenfenster verfolgen (Abbildung 1.74).

Ist dieser Schritt erfolgreich, öffnet sich ein neues Fenster mit dem Endresultat (Abbildung 1.76).

In unserem Arbeitsverzeichnis sehen wir ebenfalls die soeben erzeugte Ausgabedatei, in Abbildung 1.75 ist das `Mein-quarto-Dokument.html`.

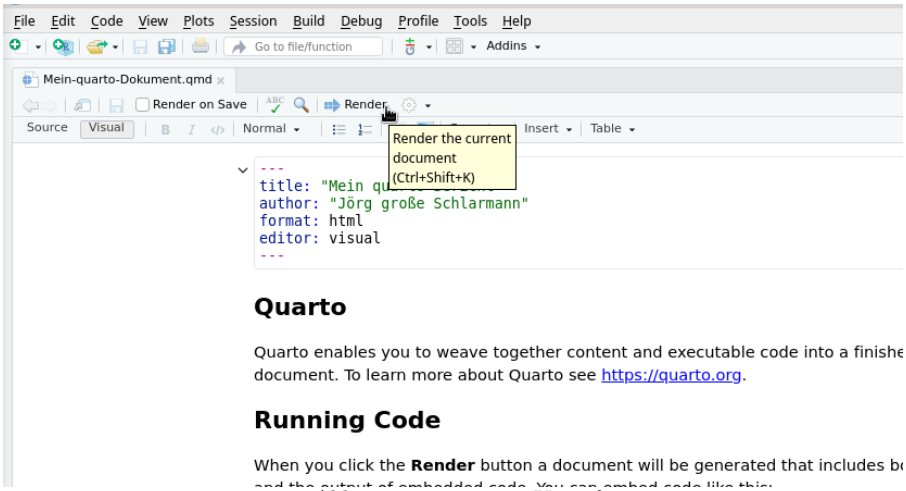


Abb. 1.73: Render-Knopf

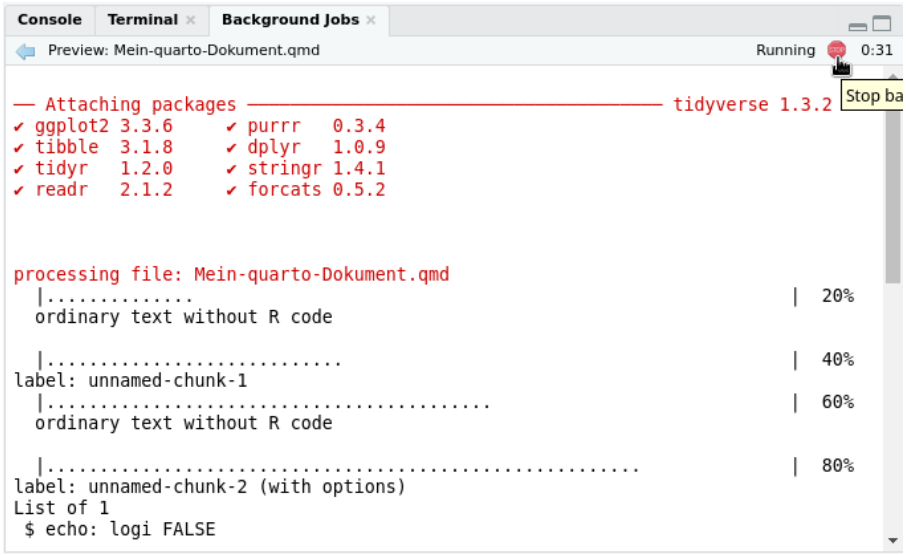


Abb. 1.74: Render-Prozess

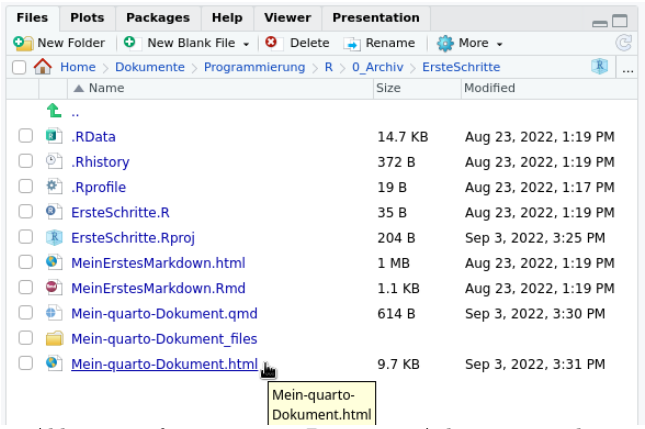


Abb. 1.75: fertige HTML-Datei im Arbeitsverzeichnis

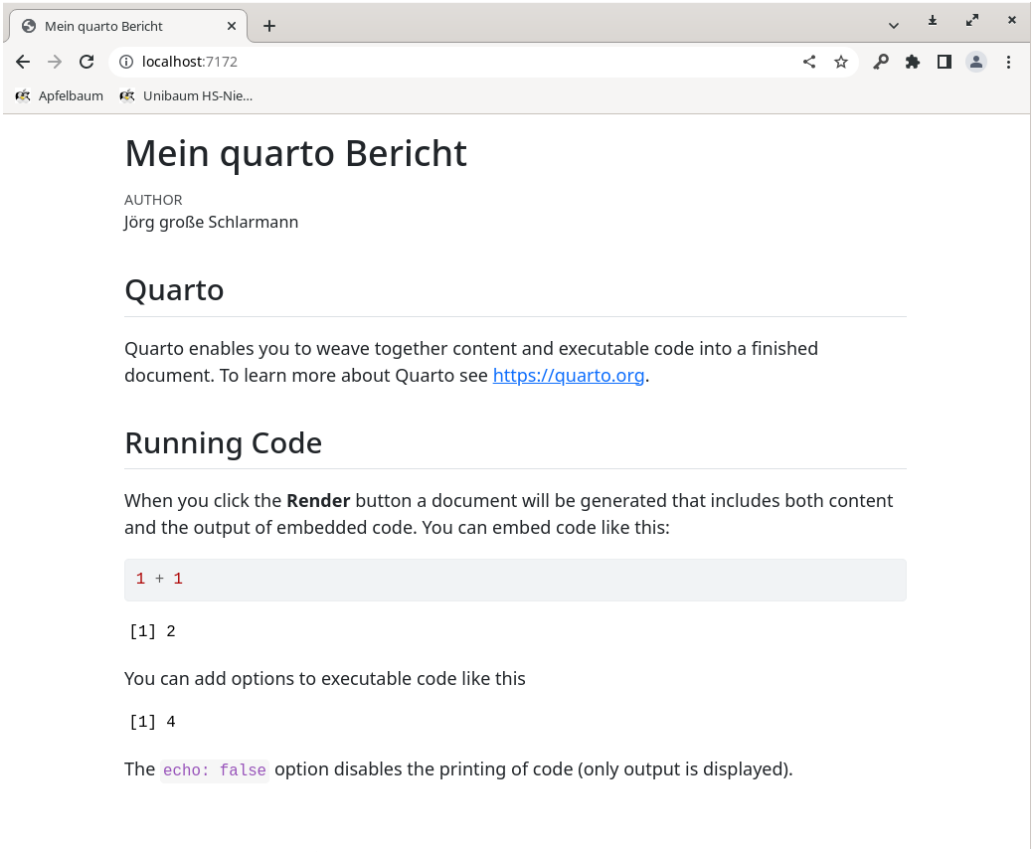
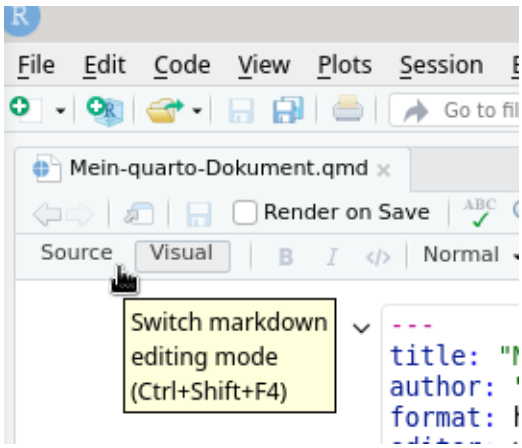
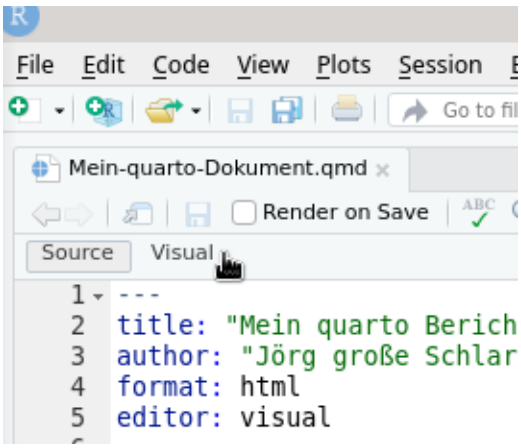


Abb. 1.76: fertiges `quarto` Dokument

1.5.3 Editor-Ansicht

Das `quarto`-Dokument startet im "visuellen Editor", der so ähnlich funktioniert wie Word oder LibreOffice. So können Sie Ihren Text "wie gewohnt" gestalten. Sie haben zudem oben eine Menüzeile, mit der Sie die Schriftart wechseln, externe Bilder einbinden oder Tabellen erzeugen können.
Auf der linken Seite können Sie zwischen dem visuellen Editor und der Code-Ansicht wechseln.





☠ Achtung, Anfängerfehler! ☠

Denken Sie daran, in die klassische Code-Ansicht zurück zu wechseln, bevor Sie Markdown-Code

kopieren und einfügen. Dies ist ein häufiger Anfgängerfehler, und der visuelle Editor kann mit dem Code nichts anfangen.

1.5.4 Dokumentenaufbau

Schauen wir uns unsere eben erzeugte Markdowndatei noch einmal an. Wechseln Sie hierzu in die Code-Ansicht.

```
---
title: "Mein quarto Bericht"
author: "Jörg große Schlarmann"
format: html
editor: visual
---
```

Dies ist die *Kopfzeile* des Dokuments, die im `yaml`-Format angegeben ist (so genannter *yaml-Header*). Wir können hier den Titel, Autor und das Datum ändern. Alle Parameter des Headers finden Sie unter https://quarto.org/docs/reference/formats/html.html

Unter den drei Strichen `---` beginnen wir unser eigentliches Dokument. Schreiben Sie folgende Zeilen in Ihr Dokument:

```
---
title: "Mein quarto Bericht"
author: "Jörg große Schlarmann"
date: "01 09 2022"
format: html
editor: visual
---

# Überschrift

Willkommen zu meinem quarto-Markdown-Dokument

- dies ist ein Punkt

- dies ein weiterer

1. erstens

2. zweitens

3. drittens

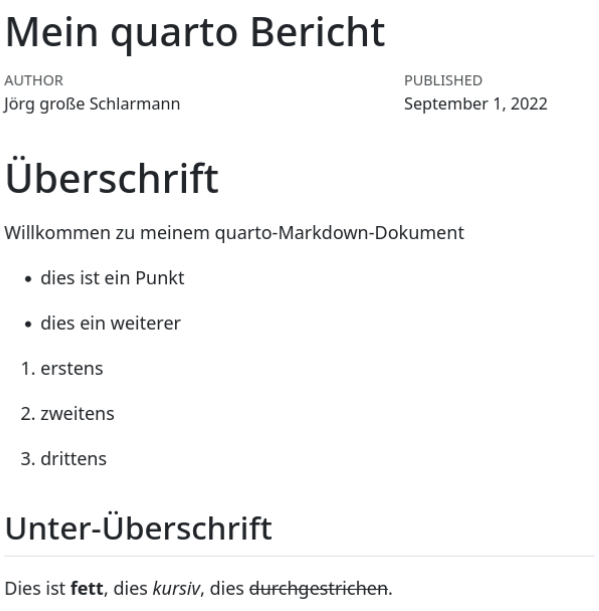
## Unter-Überschrift

Dies ist **fett**, dies *kursiv*, dies ~~durchgestrichen~~.
```

Drücken Sie den `Render`-Knopf und erzeugen so einen Testoutput (vgl. Abbildung 1.77).
Das Zeichen `#` steht für Überschriften und deren Hierarchie. Die Anzahl der `#` legt die Überschriftenhierarchie fest. Aus den Spiegelstrichen ist eine Liste geworden, die Zahlen wurden in eine nummerierte Aufzählungsliste umgewandelt.

1.5.5 Syntax

Die Syntax von *Markdown* ist recht leicht, die folgenden Beispiele wurden aus dem Quarto-Guide hierhin übernommen:



Mein quarto Bericht

AUTHOR
Jörg große Schlarmann

PUBLISHED
September 1, 2022

Überschrift

Willkommen zu meinem quarto-Markdown-Dokument

- dies ist ein Punkt
- dies ein weiterer

1. erstens
2. zweitens
3. drittens

Unter-Überschrift

Dies ist **fett**, dies *kursiv*, dies ~~durchgestrichen~~.

Abb. 1.77: fertig gerendertes Testdokument

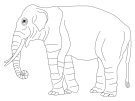
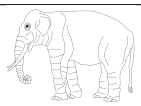
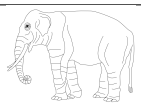
1.5.5.1 Textformat

`*kursiv*` und `**fett**`	*kursiv* und **fett**
`superscript^2^` und `subscript~2~`	superscript^2 und subscript$_2$
`~~durchgestrichen~~`	~~durchgestrichen~~
`` `verbatim code` ``	`verbatim code`

1.5.5.2 Überschriften

`# Header 1`	Überschrift 1
`## Header 2`	Überschrift 2
`### Header 3`	Überschrift 3
`#### Header 4`	Überschrift 4
`##### Header 5`	Überschrift 5
`###### Header 6`	Überschrift 6

1.5.5.3 Links und Bilder

`<https://www.produnis.de>`	https://www.produnis.de
`[Produnis](https://www.produnis.de)`	Produnis
`![Caption](elephant.png)`	
`![Caption](elephant.png)(https://www.produnis.de)`	
`![Caption](elephant.png)(https://www.produnis.de ¨Ein Elefant¨)`	

1.5.5.4 Listen

```
* ungeordnete Liste
    + sub-Item
    + sub-Item 2
        - sub-sub-Item
```

- ungeordnete Liste
  - sub-Item
  - sub-Item 2
    - sub-sub-Item

```
1. nummerierte Liste
2. zweites Item
    i) sub-Item
        A. sub-sub-Item
```

1. nummerierte Liste
2. zweites Item
   - i) sub-Item
     - A. sub-sub-Item

1.5.5.5 Tabellen

```
## Tabelle

| Name   | Beruf       | Alter |
|--------|-------------|-------|
| Heinz  | Kraftfahrer | 46    |
| Lisa   | Professorin | 32    |
| Jürgen | Klemptner   | 48    |

: Meine Tabelle
```

Name	Beruf	Alter
Heinz	Kraftfahrer	46
Lisa	Professorin	32
Jürgen	Klemptner	48

Tabelle 1.2: Meine Tabelle

1.5.5.6 Gleichungen

Mit `$` können In-Text-Formeln erzeugt werden, mit `$$` schließt man Matheumgebungen ein:

`inline math: $E = mc^2$`	inline math: $E = mc^2$
`display math: $$E = mc^2$$`	display math:

$$E = mc^2$$

Mehr Informationen erhalten Sie hier:
https://quarto.org/docs/authoring/markdown-basics.html

1.5.6 R-Code integrieren

Kommen wir nun zum eigentlichen Kernpunkt: der Darstellung von -Code und -Output. Hierfür muss im Markdowndokument der Bereich für den Code festgelegt werden. Dies erfolgt über die Zeichenkette ```, die den Codebereicht einschließt.

````
```{r}

# hier steht der R-Code

```
````

Die Anführungsstriche grenzen den Codebereich ein. Dieser wird *Chunk* genannt. Das `{r}` bedeutet, dass -Code verwendet wird.

Achtung, Anfängerfehler!

Achten Sie darauf, die korrekten `` zu verwenden, und nicht etwa `'''` oder ´´´. Dies ist ein häufiger Anfängerfehler.

In ‘RStudio‘ können Sie einfach die Tastenkombination `[STRG]` + `[ALT]` + `[i]` drücken, oder im Scriptfenster oben auf das grüne `c` klicken, siehe Mauszeiger in Abbildung 1.78.

Beides fügt einen neuen *Chunk* (Codeblock) ein.
Die *Chunks* werden im Editorfenster farblich hervorgehoben. Tragen Sie folgenden Code in den Chunk ein:

```
load(url("https://www.produnis.de/R/data/pf8.RData"))
plot(pf8$Alter, pf8$Gewicht)
```

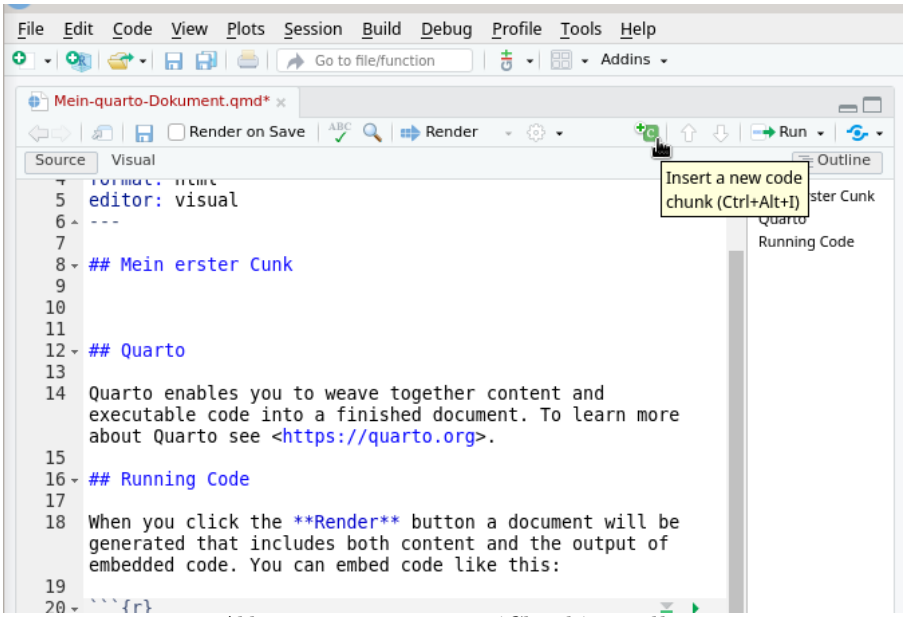


Abb. 1.78: einen neuen 'Chunk' erstellen

Wenn Sie den *Render*-Knopf drücken, wird der *load()*-Befehl im Dokument angezeigt und in der Konsole ausgeführt. Das neu gerenderte HTML-Dokument öffnet sich automatisch.



Abb. 1.79: Chunk-Code ausführen

Sie haben auch die Möglichkeit, den Code auszuführen und sich das Ergebnis anzeigen zu lassen, ohne das komplette Dokument zu *Rendern* . Klicken Sie dazu rechts auf das grüne Dreieck (Abbildung 1.79).

In diesem Beispiel wird der Plot nun direkt im Markdownfenster angezeigt (Abbildung 1.81). Dies ist wirklich hilfreich beim Erstellen des Dokuments, denn Sie müssen nicht immer erst das gesamte Dokument rendern oder überhaupt geöffnet haben, um zu sehen, was R an dieser Stelle ausgeben würde.

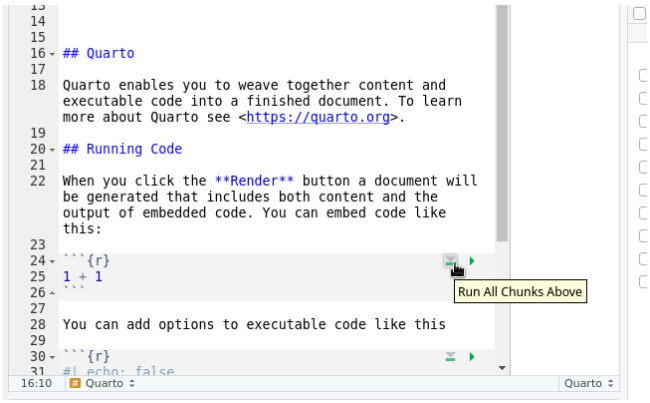


Abb. 1.80: Chunks bis hier ausführen

Über das Symbol links daneben (Abbildung 1.80, siehe Mauszeiger) lassen sich älle Chunks bis hier hinäusführen.

Dies ist zum Beispiel hilfreich, wenn in den vorherigen Chunks Objekte oder Variablen erzeugt wurden, auf die nun zugegriffen werden soll.

1.5.6.1 Chunk-Optionen

Den Chunks können Parameter übergeben, um ihr Verhalten zu steuern. Dies erfolgt über die Zeichenfolge `#|` direkt am Zeilenanfang.

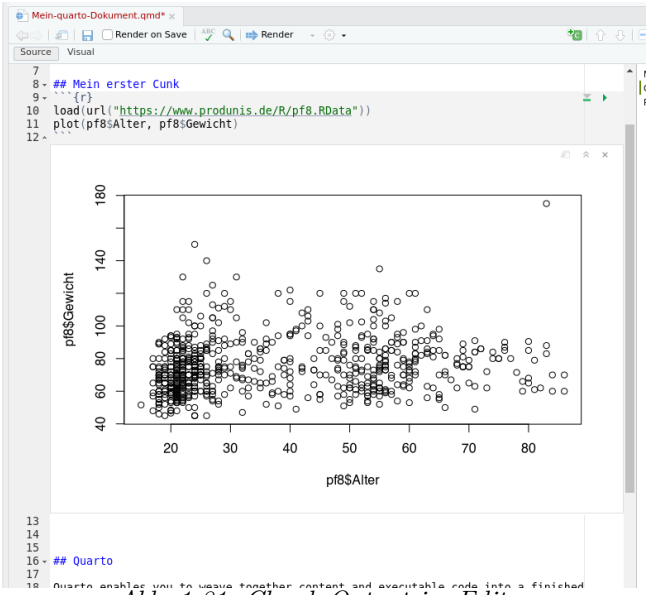


Abb. 1.81: Chunk-Output im Editor

````
```{r}
#| label: fig-testplot # zum verlinken der Plots
#| fig-cap: "Testplot" # Plotunterschrift
#| echo: true          # R-Befehle anzeigen
#| output: true        # R-Output anzeigen
#| warning: false      # Warnung unterdrücken
#| include: true       # Chunk anzeigen
#| eval: true          # Code ausführen
load(url("https://www.produnis.de/R/data/pf8.RData"))
plot(pf8$Alter, pf8$Gewicht)
```
````

Der Plot ist über den Referenzierer `@fig-testplot` ansprechbar. Dieser wird beim rendern umgewandelt in → "Abbildung 1.81".
Die Standard-Parameter lauten:

- `label` - gibt dem Chunk einen Namen. Dies ist beim Debugging sehr hilfreich. Soll auf die Plots des Chunks referenziert werden, muss das Label zwingend mit `fig-` anfangen.
- `fig-cap` - gibt den Plottitel an.
- `echo` - gibt an, ob die Befehle des Chunks im Dokument angezeigt werden sollen (`true` / `false`).
- `output` - gibt an, ob die R-Ausgaben im Dokument angezeigt werden sollen (`true` / `false`).
- `warning` - gibt an, ob die Warnungen, die möglicherweise beim Ausführen des Codes ausgegeben werden, im Dokument angezeigt werden sollen (`true` / `false`).
- `include` - gibt an, ob der Chunk (Befehle und Ausgabe) im Dokument angezeigt werden soll (`true` / `false`).
- `eval` - gibt an, ob der Chunk ausgeführt (`true`) werden, oder nur der Chunkinhalt angezeigt werden soll (`false`).

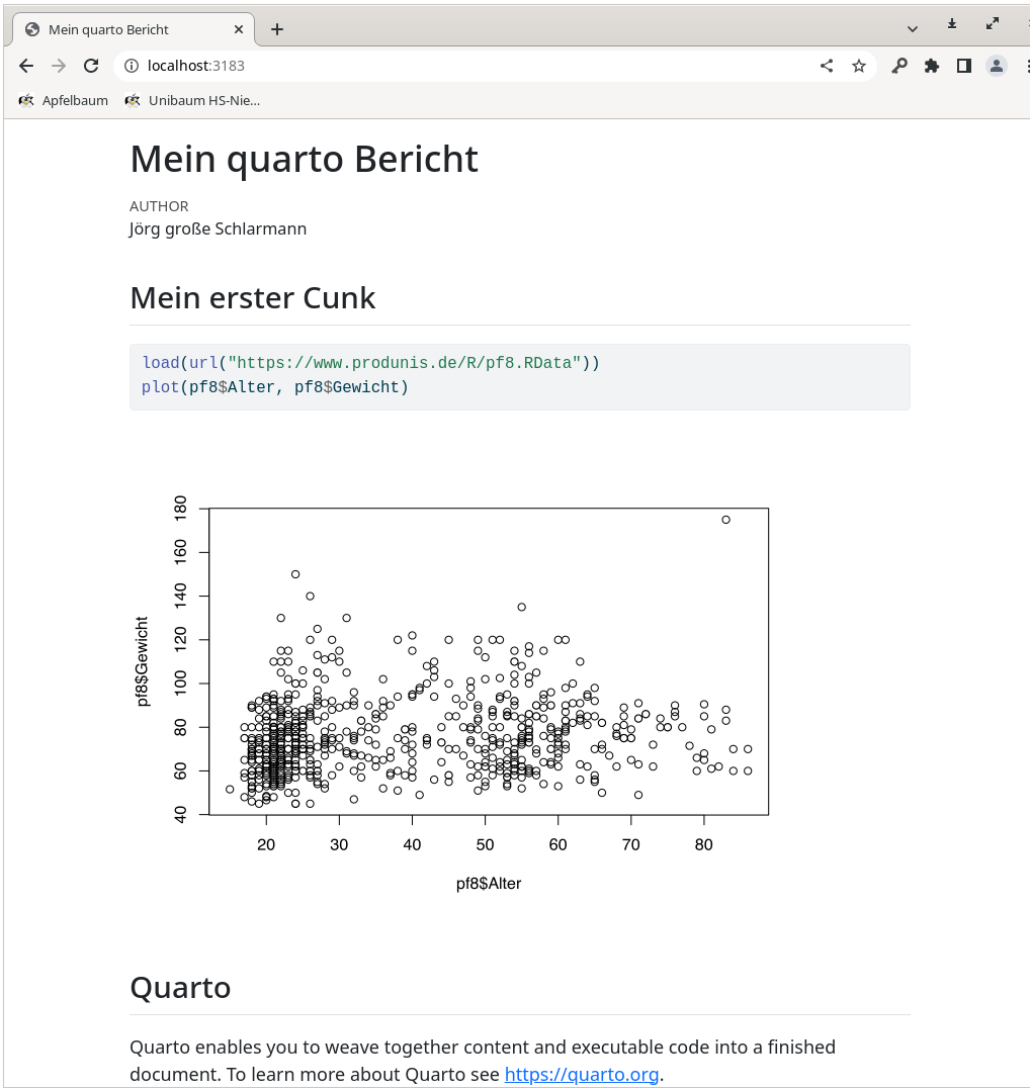


Abb. 1.82: Dokument mit Plot

1.5.6.2 Intext-Codeintegration

Auch im Fließtext können R-Objekte referenziert werden. Die Syntax dazu lautet `` `r BEFEHL` ``.

```
Der Mittelwert des Alters beträgt `r mean(pf8$Alter, na.rm=TRUE)`, wobei
die Standardabweichumg mit `r sd(pf8$Alter, na.rm=TRUE)` hoch ist.
```

ergibt:

Der Mittelwert des Alters beträgt 37.0027397, wobei die Standardabweichumg mit 17.9997712 hoch ist.

1.5.7 PDF-Dokumente erzeugen

Da PDF-Dokumente von `quarto` intern mittels LaTeX erzeugt werden, müssen Sie auf Ihrem PC eine LaTeX Distribution installiert haben. Falls dies nicht der Fall ist, können Sie `TinyTeX` mit folgendem Behfehl im RStudio Terminal installieren:

```
quarto install tool tinytex
```

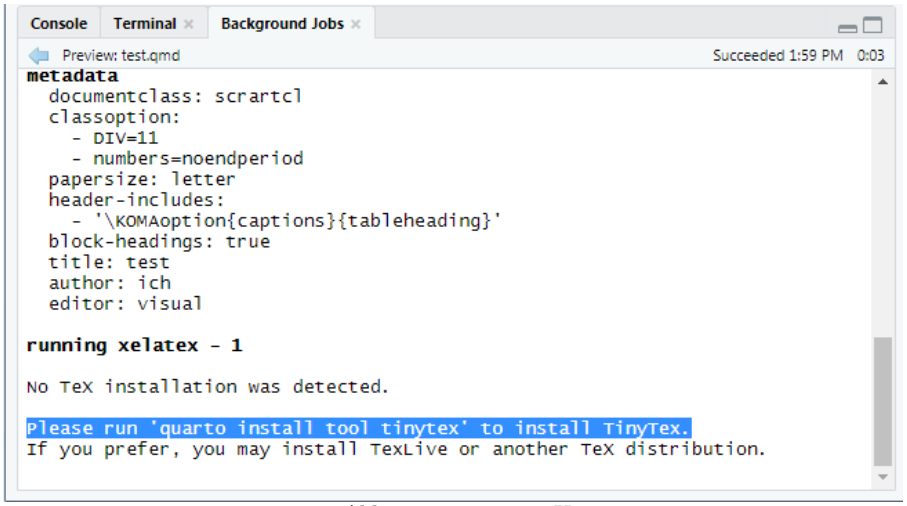


Abb. 1.83: TinyTeX

Mit dieser Methode installiert quarto alle weiteren Pakete automatisch, sofern diese benötigt werden. Sollten Sie bereits eine LaTeX-Distribution installiert haben, müssen Sie das Paket `fontawesome` installieren, damit alles funktioniert.

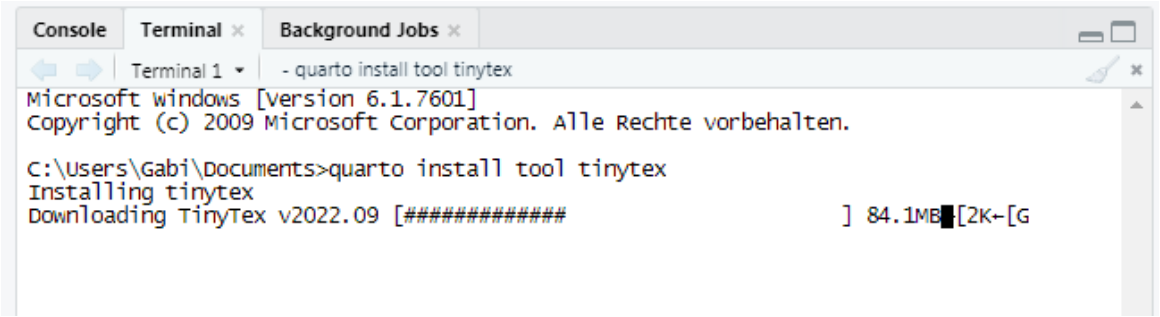


Abb. 1.84: TinyTeX installieren

1.5.8 quarto-Dokumentation

Die Webseite von quarto bietet umfangreiche Anleitungs- und Referenzierungsdokumente an.

- Der **Guide** leitet Sie durch die Erstellung Ihrer Dokumente, siehe https://quarto.org/docs/guide/, und ist für Einsteiger gut geeignet.
  - Für HTML z.B. https://quarto.org/docs/output-formats/html-basics.html
- Die **References** listen alle Parameter und Einstellungsmöglichkeiten der quarto-Dokumente auf, siehe https://quarto.org/docs/reference/.
  - Für HTML z.B. https://quarto.org/docs/reference/formats/html.html
- Auf Youtube gibt es eine offizielle Quarto-Playlist
  https://www.youtube.com/playlist?list=PL9HYL-VRX0oRupficE2l5DGgVIzpypTHs

In der *Awesome Quarto List* auf Github https://github.com/mcanouil/awesome-quarto werden quartobezogene Anleitungen, Präsentationen, Erweiterungen und Beispiele gesammelt. Beachten Sie auch Kapitel 3.6 "Was willst du machen“ ab Seite 460.

1.6 Tidyverse

Das `Tidyverse` ist eine Weiterentwicklung von R, die maßgeblich von Hadley Wickham vorangetrieben wurde (weshalb auch die Bezeichung *Hadleyverse* noch gebräuchlich ist. Wickham ist mittlerweile Chefentwickler von `RStudio`). Das Standardwerk zum Tiddyverse "*R for Data Science*" (kurz *R4DS*) stammt ebenfalls von Wickham[6], und ist online frei verfügbar unter `https://r4ds.had.co.nz/` (eine physische Kopie des Buches kann ebenfalls gekauft werden).

Das Tidyverse folgt streng dem Konzept *Tidy Data*[3], welches besagt, dass ein Datensatz so aufgebaut sein muss, dass jeweils ein Fall pro Zeile abgebildet wird. Das bedeutet, dass jede Beobachtung (auch Wiederholungen) in einer eigenen Zeile steht, und die jeweiligen Variablen durch die Spalten repräsentiert werden.

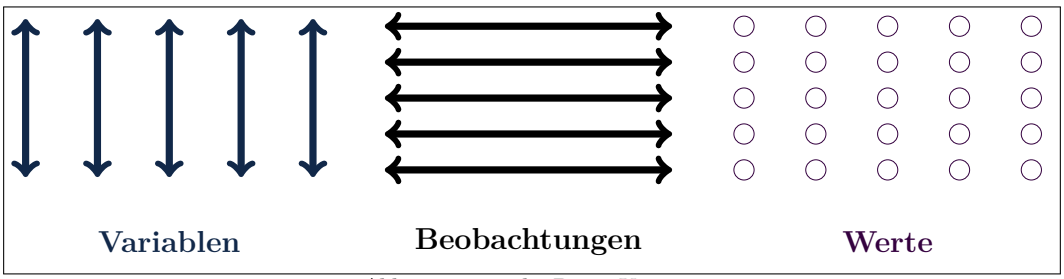


Abb. 1.85: Tidy Data Konzept

Abbildung 1.85 zeigt die Sicht auf ein Datenobjekt nach dem Tidy Data Konzept. Jede Variable wird als eigene Spalte dargestellt. Jede Reihe entspricht einer Beobachtung. Somit entsprechen die einzelnen Werte in der Tabelle genau einer Variable zu einer Beobachtung.

Man spricht in diesem Zusammenhang von "`long table`" und "`wide table`".
Die Matrix der Pflegeberufe, die wir aus Abbildung 1.26 übernommen haben, stellt dabei die *wide table*, die *breite* Tabelle dar.

```
                        1999   2001   2003   2005   2007   2009   2011   2013   2015
Krankenpflegeassistenz 16624  19061  19478  21537  27731  36481  46517  54371  64127
Altenpflegehilfe       55770  52710  49727  45776  48326  47903  47978  48363  49507
Kinderkrankenpflege    47779  48203  48822  48519  49080  49307  48291  48937  48913
Krankenpflege         430983 436767 444783 449355 457322 465446 468192 472580 476416
Altenpflege           109161 124879 141965 158817 178902 194195 208304 227154 246412
```

"Breit" bedeutet, dass die Tabelle, wenn wir ihr nun 10 weitere Jahrgänge hinzufügen würden, immer breiter und breiter werden würde. Die *wide table* ist als *Darstellung* der Daten sicherlich gut geeignet, und die Leser erhalten eine gute Übersicht. Für die Daten*analyse* und -verarbeitung ist dieses Format eher ungeeignet, da mit jedem Wert in der Tabelle eigentlich *zwei* Variablen andressiert werden (konkrete Berufsgruppe in einem konkreten Jahr). Welche beiden Variablen das nun genau sind, lässt sich am Wert alleine nicht ablesen, und wir hätten große Probleme, wenn Zeilen und Spalten nicht benannt wären!

Als Faustregel kann man sich merken, dass eine *wide table* dann vorliegt, wenn auch (bedeutsame) Zeilennamen im Datensatz vergeben wurden (was bei einer `matrix` häufig der Fall ist).

Ein Datensatz nach dem Tidy Data-Konzept ist vom Typ *long table*.
Für die Matrix der Pflegeberufe sähe die `long table`-Version das Datensatzes so aus:

```
   Jahr           Berufsgruppe Anzahl
1  1999 Krankenpflegeassistenz  16624
2  2001 Krankenpflegeassistenz  19061
3  2003 Krankenpflegeassistenz  19478
4  2005 Krankenpflegeassistenz  21537
5  2007 Krankenpflegeassistenz  27731
6  2009 Krankenpflegeassistenz  36481
7  2011 Krankenpflegeassistenz  46517
8  2013 Krankenpflegeassistenz  54371
9  2015 Krankenpflegeassistenz  64127
10 1999       Altenpflegehilfe  55770
11 2001       Altenpflegehilfe  52710
12 2003       Altenpflegehilfe  49727
           ( . . . )
```

Jede Spalte repräsentiert eine Variable, und jede Zeile repräsentiert eine Beobachtung. Auch ohne die Spaltennamen (Zeilennamen gibt es gar keine) könnten wir erkennen, worum es in dem Datensatz geht.

Der Name `long table` leitet sich davon ab, dass die Tabelle, wenn wir ihr mehr Daten hinzufügen würden, immer *länger* und länger werden würde.

Die Tatsache, dass die Verarbeitung solcher `long tables` wesentlich einfacher ist, bildet den Grundgedanken des Tidyverse, und es wurden spezielle R-Erweitrungen implementiert, die speziell an Tidy Data angepasst sind. Die Erweiterungen wurden nach dieser gemeinsame Philosophie entworfen, sie verwenden die selbe "Grammatik" und greifen auf die selbe Datenstruktur (*tibbles*, dazu später mehr) zurück.

Abbildung 1.86 zeigt den typischen Ablauf einer Datenauswertung in 4 Schritten.

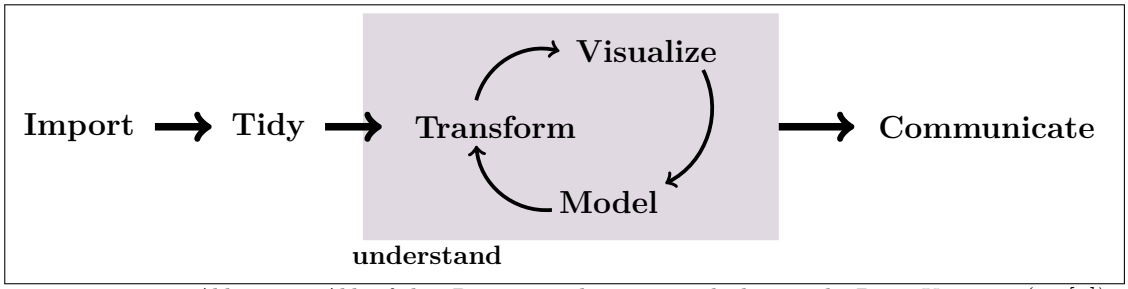


Abb. 1.86: Ablauf der Datenverarbeitung nach dem Tidy Data Konzept (in [6])

1. Die Daten müssen importiert und
2. gegebenenfalls angepasst (ins `tidy data` bzw. `long table` Format überführt werden).
3. Anschließend werden die Daten "statistisch ausgewertet", d.h. wir versuchen zu verstehen, was wir in den Daten sehen und welche spezifischen Informationen extrahiert werden können. Das heisst die Daten werden transformiert (z.B. indem man Untergruppen bildet) und visualisiert (z.B. in Form von Diagrammen und Tabellen) oder wir rechnen Modelle wie Korrelationen, Regressionen und Signifikanztests.

4. Unsere Erkenntnisse nutzen wir dann zur Kommunikation (Präsentation) unserer Forschungsergebnisse.

Für all diese Arbeitsschritte stellt das Tidyverse passende Pakete und Funktionen in R zur Verfügung, und wir werden die Schritte nun einzeln durchlaufen.

1.6.1 Pakete

In R besteht das Tidyverse im Wesentlichen aus dem Zusatzpaket `tidyverse`, welches alle notwendingen Unterpakete als Abhängigkeit mit-installiert:

1. Tibble, die Datenklasse im Tidyverse, im Paket
   - `tibble`, mit den Funktionen
     `tibble()` `as_tibble()`
2. Daten importieren:
   - `readr`, mit den Funktionen
     `read_csv()`, `read_tsv()`, `read_delim()`, `read_table()`
3. Daten *tidy* machen:
   - `tidyr`, mit den Funktionen
     `pivot_longer()`, `pivot_wider()`, `separate()`, `unite()`, `drop_na()`, `replace_na()`, `fill()`, `complete()`, `expand()`
   - `forcats`, mit den Funktionen
     `as_factor()`
4. Umgang mit Datensätzen:
   - `dplyr`, mit den Funktionen
     `arrange()`, `filter()`, `select()`, `pull()`, `mutate()`, `transmute()`, `rename()`, `summarise()`, `bind_cols()`, `bind_rows()`
   - `purrr`, mit den Funktionen
     `modify()`, `map()`, `reduce()`, `nest()`
   - `stringr`, mit den Funktionen
     `str_sub()`, `bind_locate()`, `bind_extract()`, `bind_count()`, `bind_match()`
5. Daten visualisieren:
   - `ggplot`, mit den Funktionen
     `ggplot()`, `qplot()`

Dazu später mehr, denn um die Funktionen vorzustellen, müssen wir uns zunächst der `Pipe` zuwenden.

1.6.2 Pipe

Ein wichtiger Operator für die Arbeit im Tidyverse ist die `Pipe` aus dem `magrittr`-Zusatzpaket[8]. Sie erlaubt es, *Datenströme* weiterzuleiten. Ihre Funktionalität hat so sehr überzeugt, dass sie im Standard-R ab Version 4.1 enthalten ist, siehe Abschnitt 1.2.3.9.
Hierfür wurde ein eigener Operator implementiert, die Zeichenkette `%>%` (im Standard-R lautet die Zeichenkette `|>`).

Sie bedeutet so viel wie "*und dann*".

Zur Eklärung sei nochmals der "klassische Weg" der Arbeitsschritte in  aufgezeigt:

```
# wir machen etwas, und speichern es in ein Objekt
habe.getan <- mache.etwas(Datensatz)

# dann machen wir etwas weiteres,
# und speichern wieder in ein Objekt
habe.weiteres.getan <- mache.weiteres(habe.getan)

# und kommen schließlich zum Endergebnis
endergebnis <- mache.noch.letzte.Sache(habe.weiteres.getan)
```

Mit Einsatz der `Pipe` wird dieser Prozess quasi umgekehrt in

```
# Speichere Endergebnis
endergebnis <- Datensatz %>%
                    gruppiere.nach.geschlecht %>%
                        sortiere Alter aufsteigend  %>%
                            nimm die letzten 4 Werte
```

Die liest sich in etwa so:

- "Speichere etwas ins Objekt `endergebnis`, und das geht so…
- nimm den `Datensatz` *und dann*
- gruppiere die Daten nach `geschlecht` *und dann*
- sortiere nach `Alter` aufsteigend *und dann*
- nimm die letzten 4 Werte. "

Die `Pipe` reicht das jeweilige Ergebnis (den *Datenstrom*) an die nächste Code-Zeile weiter. Erst wenn die letzte Zeile durchgelaufen ist, wird das Resultat in `endergebnis` gespeichert.
Dies liest sich zu Beginn evtl. etwas ungewohnt, aber Sie können erkennen, dass die Befehle so wesentlich übersichtlicher und die einzelnen *Manipulations*schritte nachvollziehbarer geworden sind. Auch kann man sich diese Art der "Grammatik" relativ leicht merken.

Benötigt eine Funktion die Angabe eines Datensets, so kann mit einem Punkt `.` auf den Pipe-Datenstrom verwiesen werden.

[8] Die Pipe wird mit `tidyverse` aktiviert, Sie müssen `magrittr` nicht extra installieren.

```r
# Der Punkt übergibt den Pipe-Strom an die Funktion sum()
endergebnis <- Datensatz %>%
                    wähle Spalte "Alter" %>%
                        # berechne Summe
                        sum(.)
```

Die `Pipe` funktioniert bei jedem R-Objekt, egal in welchem Format (`long table` vs. `wide table`, Faktor, Vektor, Matrix, usw.) es vorliegt.

1.6.3 Unterschiede zwischen %>% und |>

Während sich `|>` und `%>%` in einfachen Fällen identisch verhalten, gibt es ein paar wichtige Unterschiede. Diese betreffen Sie aber nur, wenn Sie `%>%` schon lange nutzen und einige der fortgeschrittenen Funktionen verwendet haben.

1. Standardmäßig übergibt die Pipe das Objekt auf ihrer linken Seite an das erste Argument der Funktion auf der rechten Seite. Mit `%>%` können Sie die Platzierung des Datenstroms mit dem Platzhalter "`.`" ändern. Zum Beispiel ist `x %>% f(1)` äquivalent zu `f(x, 1)`, aber `x %>% f(1, .)` ist äquivalent zu `f(1, x)`. Seit R 4.2.0 ist ein ähnlicher Platzhalte über `_` zur Basis-Pipe hinzugefügt worden, jedoch muss hier immer ein Argument benannt werden. Zum Beispiel ist `x |> f(1, y = _)` äquivalent zu `f(1, y = x)`.

2. Mit `%>%` kann man die Klammern weglassen, wenn man eine Funktion ohne weitere Argumente aufruft; `|>` erfordert immer die Klammern.

3. Der Platzhalter `|>` ist absichtlich einfach gehalten und kann viele Eigenschaften des Platzhalters `%>%` nicht wiedergeben. Sie können ihn nicht an mehrere Argumente übergeben, und er hat kein besonderes Verhalten, wenn er innerhalb einer anderen Funktion verwendet wird. Zum Beispiel ist `df %>% split(.$var)` äquivalent zu `split(df, df$var)` und `df %>% {plot(.$x, .$y)}` ist äquivalent zu `plot(df$x, df$y)`.

4. Mit `%>%` können Sie den Punkt `.` auf der linken Seite von Operatoren verwenden, so dass Sie eine einzelne Spalte aus einem Datenframe mit (z. B.) `mtcars %>% .$cyl` extrahieren können. Dies geht mit `|>` nicht.

Glücklicherweise gibt es keinen Grund, sich ganz auf die eine oder andere Pipe festzulegen - Sie können die Basis-Pipe für die meisten Fälle verwenden, und die Magrittr-Pipe benutzen, wenn Sie deren speziellen Eigenschaften wirklich benötigen.

1.6.4 tibbles

Da dieses Konzept der `long table` so essenziell ist, wurde im Tidyverse eine eigene Datenklasse eingeführt, die `tibble` heisst[9].
Sie entspricht in etwa dem Datenframe, hat aber intern noch weitere wichtige Änderungen eingebaut.

[9] Das kommt daher, dass `tidy data` immer Tabellen sind, was häufig per *tbl* abgekürzt wird. Liest man im Englischn *tbl* laut und schnell, klingt das wie - genau - *tibble*.

Tibbles werden so wie Datenframes erzeugt, indem man gleichlange Vektoren übergibt. Der Funktionsname ist `tibble()`.

```
# aktiviere Tidyverse
library(tidyverse)

# erzeuge ein tibble
tibble(a=c(1,2,3), b=c("a", "b", "c"), c=c(T, F, T))
```

```
# A tibble: 3 x 3
      a b     c
  <dbl> <chr> <lgl>
1     1 a     TRUE
2     2 b     FALSE
3     3 c     TRUE
```

Die Schwesterfunktion `tribble()` (mit r) erlaubt diese Schreibweise

```
# erzeuge ein tibble mit tRibble
tribble(
  ~Text, ~Zahl, ~Logic,
  "a",     1,     T,
  "b",     2,     F,
  "c",     3,     T
  )
```

```
# A tibble: 3 x 3
  Text   Zahl Logic
  <chr> <dbl> <lgl>
1 a         1 TRUE
2 b         2 FALSE
3 c         3 TRUE
```

Sind die übergebenen Reihen nicht gleich lang, werden sie als `list` gespeichert.

```
# Datenreihen NICHT gleich lang
tribble(
  ~Text, ~Zahl, ~Logic,
  "a",     1:3,     T,
  "b",     4:6,     F,
  "c",     7:9,     T
)
```

```
# A tibble: 3 x 3
  Text  Zahl      Logic
  <chr> <list>    <lgl>
1 a     <int [3]> TRUE
2 b     <int [3]> FALSE
3 c     <int [3]> TRUE
```

Die meisten Funktionen des Tiddyverse funktioneren auch mit Datenframes, eben weil sie sich so ähnlich sind. Dennoch ergibt es (schon ideologisch) Sinn, die Datenframes in `tibbles` umzuwandeln.
Schauen wir uns hierfür die Daten der Pflegeberufe als `long table` an:

```
# Pflegeframe
head(Pflegeframe)
```

```
  Jahr           Berufsgruppe Anzahl
1 1999 Krankenpflegeassistenz  16624
2 2001 Krankenpflegeassistenz  19061
3 2003 Krankenpflegeassistenz  19478
4 2005 Krankenpflegeassistenz  21537
5 2007 Krankenpflegeassistenz  27731
6 2009 Krankenpflegeassistenz  36481
```

```r
# Übersicht von "Pflegeframe"
str(Pflegeframe)
```

```
'data.frame':^^I45 obs. of  3 variables:
 $ Jahr        : Factor w/ 9 levels "1999","2001",..: 1 2 3 4 5 6 7 8 9 1 ...
 $ Berufsgruppe: Factor w/ 5 levels "Krankenpflegeassistenz",..: 1 1 1 1 1 1 1 1 1 2 ...
 $ Anzahl      : num  16624 19061 19478 21537 27731 ...
```

Wir erhalten eine Zusammenstellung von wichtigen Informationen. Unser Pflegetibble hat 45 Beobachtungen (Reihen) von 3 Variablen (Spalten). Die Variablen `Jahr` und `Berufsgruppe` liegen als Faktoren vor, `Anzahl` ist numerisch.

```r
# Datenklasse von "Pflegeframe"
class(Pflegeframe)
```

```
[1] "data.frame"
```

Jetzt wandeln wir das Pflegeframe in ein Pflegetibble um.

```r
# erzeuge Pflegetibble aus Pflegeframe
Pflegetibble <- tibble(Pflegeframe)

# Datenklasse von "Pflegeframe"
class(Pflegetibble)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Wie Sie sehen, ist unser Tibble immernoch ein Datenframe. Es ist aber auch ein "Tibble Datenframe" (`tbl_df`) und einfach ein Tibble (`tbl`).

Die Funktion zum Einsehen von Tibbles heisst `glimpse()` (englisch für *Blick*).

```r
# werfe einen Blick auf "Pflegetibble"
glimpse(Pflegetibble)
```

```
Rows: 45
Columns: 3
$ Jahr         <fct> 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015, 199…
$ Berufsgruppe <fct> Krankenpflegeassistenz, Krankenpflegeassistenz, Krankenpf…
$ Anzahl       <dbl> 16624, 19061, 19478, 21537, 27731, 36481, 46517, 54371, 6…
```

Die Ausgabe sieht fast genau so aus wie der `str()`-Aufruf. Neben den Variablennamen steht der Datentyp, wobei `<fct>` für Faktor, `<dbl>` für Dezimalzahlen und `<lgl>` für logical steht.

Im klassischen R hat man ein `data.frame` mit `str()` angeschaut, im Tidyverse schaut man mit `glimpse()` auf ein `tibble`.

1.6.5 Funktionsaufrufe

In R gibt es jede Menge Zusatzpakete, und obschon die Programmierer sich Mühe geben, einen Funktionsnamen nicht doppelt zu vergeben (vielleicht ist Ihnen schon aufgefallen, dass im Tidyverse viele Funktionen mit einem Unterstrich _ geschrieben werden, z.B. `read_sav()` oder `as_factor()`), kann dies durchaus vorkommen.

Wenn wir das Tidyverse per `library()` aktivieren, erhalten wir folgende Informationen:

```
1 library(tidyverse)
```

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────── tidyverse 1.3.1 ──
✓ ggplot2 3.3.5     ✓ purrr   0.3.4
✓ tibble  3.1.6     ✓ dplyr   1.0.7
✓ tidyr   1.1.4     ✓ stringr 1.4.0
✓ readr   2.1.1     ✓ forcats 0.5.1
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
>
```

Abb. 1.87: Tidyverse aktivieren

Wie Sie sehen, gibt es nun Konflikte, da die Funktionen `filter()` und `lag()` jeweils in den Paketen `tidyverse` und `stats` (R-Basispaket) existieren. Das heisst, möchte man die Funktion `filter()` verwenden, muss man R mitteilen, aus welchem Paket sie stammt. Dies erfolgt in R indem man den Paketnamen getrenn durch zwei Doppelpunkte vor die Funktion schreibt.

```
1 # nutze Funktion filter() aus dem Paket "dplyr"
2 dplyr::filter()
3
4 # nutze Funktion filter() aus dem "stats" Paket
5 stats::filter()
```

Im Tidyverse ist es daher durchaus üblich, den Paketnamen mit zwei Doppelpunkten getrennt vor den Funktionsnamen zu setzen. So ist immer klar, welche Funktion aus welchem Paket denn nun aufgerufen werden soll.

```
1 # Paketname::Funktionsname
2 haven::read_sav("spss.sav")
```

Durch die Refernzierung des Paketnamens sparen Sie sich das Einbinden des gesamten Paketes per `library()`.

Dies funktioniert auch bei den zahlreichen Datensätzen, die R mitliefert. Möchten Sie den Datensatz `flights` aus dem Paket `nycflights13` nutzen, können Sie (nachdem das Paket `nycflights13` installiert wurde) wie folgt referenzieren:

```
1 # Installiere Paket "nycflights13"
2 install.packages("nycflights13")
3
4 # Paketname::Datensatz
5 head(nycflights13::flights)
```

1.6.6 Schritt 1: Daten importieren

Das Importieren von Daten unterscheidet sich nicht von den Methoden aus Abschnitt 1.2.3.12. Tidyverse bringt mit `readr()` ein eigenes Paket zum Importieren mit, dies ist aber nur für Textdateien wirklich geeignet. Mit den Zusatzpaketen `haven` und `readxl` können Sie wie in 1.2.3.12 beschrieben über die Funktionen `read_sav()`, `read_xlsx()` und `read_csv()` Ihre Daten einlesen.

In `RStudio` können Sie einfach im Datenfenster auf `Import Dataset` klicken, siehe Seite 75.

1.6.6.1 gelabelte SPSS-Daten

Hier sei noch einmal erwähnt, dass in `SPSS` kategoriale Daten häufig mit *Labels* versehen werden.

```
# Lade Test-SPSS-Datei mit Labels
spss <- haven::read_sav(url("https://www.produnis.de/R/alteDaten-lang.sav"))
head(spss)
```

```
  Frage_1             Frage_2        Frage_3             Frage_4
  <dbl+lbl>           <dbl+lbl>      <dbl+lbl>           <dbl+lbl>
1 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
2 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
3 4 [stimme zu]       4 [stimme zu]  4 [stimme zu]       0 [nicht vorhanden]
4 4 [stimme zu]       3 [weiß nicht] 3 [weiß nicht]      4 [stimme zu]
5 3 [weiß nicht]      3 [weiß nicht] 2 [stimme nicht zu] 2 [stimme nicht zu]
6 2 [stimme nicht zu] 3 [weiß nicht] 3 [weiß nicht]      2 [stimme nicht zu]
```

Mit `head()` sehen wir, dass die Werte der Spalten gelabelt sind (`4`=*stimme zu*, `3`=*weiß nicht*, usw.). Die Labels stehen in eckigen Klammern neben dem eingentlichen Wert der Variable.

Mit `glimpse()` werden die Labels ignoriert

```
# glimpse() zeigt KEINE Labels
glimpse(spss)
```

```
Rows: 1,000
Columns: 4
$ Frage_1 <dbl+lbl> 4, 4, 4, 4, 3, 2, 2, 2, 4, 3, 2, 5, 2, 2,…
$ Frage_2 <dbl+lbl> 4, 4, 4, 3, 3, 3, 1, 5, 4, 3, 2, 2, 2, 2,…
$ Frage_3 <dbl+lbl> 4, 4, 4, 3, 2, 3, 3, 3, 4, 3, 2, 2, 2, 3,…
$ Frage_4 <dbl+lbl> 0, 0, 0, 4, 2, 2, 2, 3, 5, 4, 2, 4, 4, 2,…
```

In R ist die Verwendung von Wertelabels eher untypisch. Es empfiehlt sich, die Ausprägungsstufen *so wie sie sind* als Werte einzutragen, also z.B. direkt "`männlich`" - "`weiblich`" - "`divers`" anstatt `0 - 1 - 2` und anschließender Labelung. So werden die Ausprägungen auch auf den Grafiken entsprechend "aussagekräftig" angezeigt.

Wir wandeln alle Variablen in Faktoren um, welche die Labels als Levelnamen nutzen[10].

```
spss %>%
        mutate(as_factor(.))
```

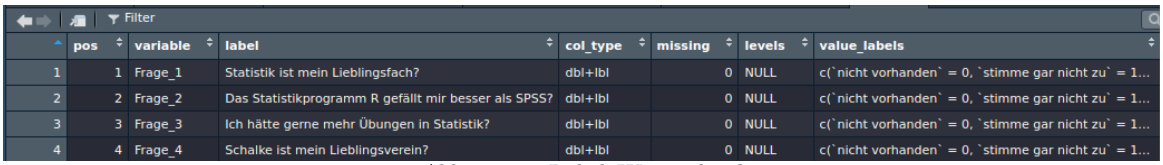
[10] Die Funktion `mutate()` wird später genauer erläutert.

```
  # A tibble: 1,000 × 4
   Frage_1         Frage_2             Frage_3         Frage_4
   <fct>           <fct>               <fct>           <fct>
 1 stimme zu       stimme zu           stimme zu       nicht vorhanden
 2 stimme zu       stimme zu           stimme zu       nicht vorhanden
 3 stimme zu       stimme zu           stimme zu       nicht vorhanden
 4 stimme zu       weiß nicht          weiß nicht      stimme zu
 5 weiß nicht      weiß nicht          stimme nicht zu stimme nicht zu
 6 stimme nicht zu weiß nicht          weiß nicht      stimme nicht zu
 7 stimme nicht zu stimme gar nicht zu weiß nicht      stimme nicht zu
 8 stimme nicht zu stimme voll zu      weiß nicht      weiß nicht
 9 stimme zu       stimme zu           stimme zu       stimme voll zu
10 weiß nicht      weiß nicht          weiß nicht      stimme zu
```

Wenn die Daten umfangreich gelabelt sind, kann es hilfreich sein, sich ein "Wörterbuch"der Labels anzulegen. Hierfür nutzt man die Funktion `generate_dictionary()` aus dem `labelled`-Paket.

```
1 dict <- labelled::generate_dictionary(spss)
```

Das Objekt `dict` ist ein Datenframe, in welchem die Labels der Variablen gespeichert sind. In RStudio kann man die Labels so im Datenviewer durchsuchen.



Filter

	pos	variable	label	col_type	missing	levels	value_labels
1	1	Frage_1	Statistik ist mein Lieblingsfach?	dbl+lbl	0	NULL	c(`nicht vorhanden` = 0, `stimme gar nicht zu` = 1...
2	2	Frage_2	Das Statistikprogramm R gefällt mir besser als SPSS?	dbl+lbl	0	NULL	c(`nicht vorhanden` = 0, `stimme gar nicht zu` = 1...
3	3	Frage_3	Ich hätte gerne mehr Übungen in Statistik?	dbl+lbl	0	NULL	c(`nicht vorhanden` = 0, `stimme gar nicht zu` = 1...
4	4	Frage_4	Schalke ist mein Lieblingsverein?	dbl+lbl	0	NULL	c(`nicht vorhanden` = 0, `stimme gar nicht zu` = 1...

Abb. 1.88: Label-Wörterbuch

Da wir die SPSS-Datei mit den Funktionen des `haven`-Pakets importiert haben, liegen gelabelte Werte und Variablen als Klasse `haven_labelled` vor. Diese Eigenschaft machen wir uns in den nächsten Code-Schnipseln zu Nutze.

Mit diesen Befehlen werden alle Werte der im Datensatz enthaltenen Variablen, die als `haven_labelled` klassifiziert ist, in einen Factor umgewandelt:

```
1 spssfct <- spss %>%
2   mutate_if(haven::is.labelled, haven::as_factor)
3 glimpse(spssfct)
```

```
  Rows: 1,000
  Columns: 4
  $ Frage_1 <fct> stimme zu, stimme zu, stimme zu, stimme zu, weiß nicht, stimme nicht zu, stimme n…
  $ Frage_2 <fct> stimme zu, stimme zu, stimme zu, weiß nicht, weiß nicht, weiß nicht, stimme gar n…
  $ Frage_3 <fct> stimme zu, stimme zu, stimme zu, weiß nicht, stimme nicht zu, weiß nicht, weiß ni…
  $ Frage_4 <fct> nicht vorhanden, nicht vorhanden, nicht vorhanden, stimme zu, stimme nicht zu, st…
```

Und folgende Befehle wandeln Variablen*labels* in Variablen*namen* um, hierzu muss das Paket `sjlabelled` installiert sein:

```
1 spssn <- spssfct %>%
2   sjlabelled::label_to_colnames()
3 glimpse(spssn)
```

```
  Rows: 1,000
  Columns: 4
  $ `Statistik ist mein Lieblingsfach?`                    <fct> stimme zu, stimme zu, stimme zu, s…
  $ `Das Statistikprogramm R gefällt mir besser als SPSS?` <fct> stimme zu, stimme zu, stimme zu, w…
  $ `Ich hätte gerne mehr Übungen in Statistik?`           <fct> stimme zu, stimme zu, stimme zu, w…
  $ `Schalke ist mein Lieblingsverein?`                    <fct> nicht vorhanden, nicht vorhanden, …
```

1.6.7 Schritt 2: Daten *tidy* machen

Der zweite Schritt besteht darin, die Daten *tidy* zu machen, das heisst, sie ins `long table` Format zu überführen. Schauen wir uns hierfür die Matrix der Pflegeberufe an, die ja im *wide* Format vorliegt:

```
1 Pflegeberufe
```

```
                       1999   2001   2003   2005   2007   2009   2011   2013   2015
Krankenpflegeassistenz  16624  19061  19478  21537  27731  36481  46517  54371  64127
Altenpflegehilfe        55770  52710  49727  45776  48326  47903  47978  48363  49507
Kinderkrankenpflege     47779  48203  48822  48519  49080  49307  48291  48937  48913
Krankenpflege          430983 436767 444783 449355 457322 465446 468192 472580 476416
Altenpflege            109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Die Reihennamen repräsentieren Variablen.
Da im Tidyverse fast alles über `tibble`s bzw. Datenframes abläuft, muss die Matrix zunächst in ein `tibble` überführt werden. Hierfür verwenden wir die Funktion `as_tibble()`

```
1 tbl <- as_tibble(Pflegeberufe)
2 head(tbl)
```

```
# A tibble: 5 x 9
  `1999` `2001` `2003` `2005` `2007` `2009` `2011` `2013` `2015`
   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1  16624  19061  19478  21537  27731  36481  46517  54371  64127
2  55770  52710  49727  45776  48326  47903  47978  48363  49507
3  47779  48203  48822  48519  49080  49307  48291  48937  48913
4 430983 436767 444783 449355 457322 465446 468192 472580 476416
5 109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Wie Sie sehen, wurden die Spalten (Jahre) als Variablen übernommen. Die Informationen aus den Zeilen (Berufsgruppe) ist verloren gegangen. Für fügen sie daher per `cbind()` unserem `tibble` hinzu.

```
1 tbl <- cbind(tbl, Berufsgruppe = rownames(Pflegeberufe))
2 head(tbl)
```

```
    1999   2001   2003   2005   2007   2009   2011   2013   2015           Berufsgruppe
1  16624  19061  19478  21537  27731  36481  46517  54371  64127 Krankenpflegeassistenz
2  55770  52710  49727  45776  48326  47903  47978  48363  49507       Altenpflegehilfe
3  47779  48203  48822  48519  49080  49307  48291  48937  48913    Kinderkrankenpflege
4 430983 436767 444783 449355 457322 465446 468192 472580 476416          Krankenpflege
5 109161 124879 141965 158817 178902 194195 208304 227154 246412            Altenpflege
```

Zur Überführung in ein korrektes tibble vom Format *long table* wird die Funktion `pivot_longer()` aus dem Tidyverse-Paket `tidyr` verwendet. Die Logik der Funktion besteht darin, die Variable mit den ehemaligen Reihennamen (die per `cbind()` hinzugefügt wurde), bestehen zu lassen. Aus den *übrigen* Spalten werden neue Variablen erzeugt, deren Werte passend zu den Reihenwerten (der `cbind()`-Spalte) eingetragen werden. In unserem Beispiel ist die (neue) Variable `Berufsgruppe` in Spalte 10. Wir belassen also Spalte 10, und wenden die Funktion auf die Spalten 1-9 an.

```
Pflegetibble <- pivot_longer(tbl,
             # wähle die Spalten 1-9 aus
             cols=1:9,
             # speichere Spaltennamen als Werte in neuer Variable "Jahr"
             names_to="Jahr",
             # Die derzeitigen Werte werden in der neuen Variable "Anzahl" gespeichert
             values_to = "Anzahl")

head(Pflegetibble)
```

```
  Berufsgruppe           Jahr  Anzahl
  <chr>                  <chr>  <dbl>
1 Krankenpflegeassistenz 1999   16624
2 Krankenpflegeassistenz 2001   19061
3 Krankenpflegeassistenz 2003   19478
4 Krankenpflegeassistenz 2005   21537
5 Krankenpflegeassistenz 2007   27731
6 Krankenpflegeassistenz 2009   36481
```

Mit der Funktion `pivot_wider()` kann im Gegenzug eine *wide table* erstellt werden. Ihr wird übergeben, aus welcher Variable die neuen Spalten gebildet werden sollen, und aus welcher Variable die anzuzeigenden Werte zu nehmen sind. Für unser Pflegetibble lautet der Befehl:

```
pivot_wider(Pflegetibble, names_from = Jahr, values_from = Anzahl)
```

```
# A tibble: 5 x 10
  Berufsgruppe           `1999` `2001` `2003` `2005` `2007` `2009` `2011` `2013` `2015`
  <chr>                   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Krankenpflegeassistenz  16624  19061  19478  21537  27731  36481  46517  54371  64127
2 Altenpflegehilfe         55770  52710  49727  45776  48326  47903  47978  48363  49507
3 Kinderkrankenpflege      47779  48203  48822  48519  49080  49307  48291  48937  48913
4 Krankenpflege           430983 436767 444783 449355 457322 465446 468192 472580 476416
5 Altenpflege             109161 124879 141965 158817 178902 194195 208304 227154 246412
```

1.6.7.0.1 verschachtelte `wide table`

Schauen wir uns folgende Tabelle über ausgedachte Einkommen an:

```
untidy_df
```

```
  Altersgruppe Maenner_2018 Frauen_2018 Maenner_2019 Frauen_2019 Maenner_2020 Frauen_2020
1         < 18        22000       20000        22000       20000        23000       21000
2        18-30        36000       35000        36000       35000        38000       36000
3        31-50        50000       40000        50000       40000        52000       47000
4        51-60        62000       60000        62000       60000        65000       62000
5         > 60        75000       72000        75000       72000        78000       75000
```

Diese Tabelle ist mehrfach verschachtelt. Zum einen haben wir “bedeutsame Reihennamen” in Form der `Altersgruppe`. Zum anderen sind auch in den Spaltennamen jeweils 2 Variablen ”versteckt”, nämlich das Geschlecht und das Jahr.
Um diese Tabelle *tidy* zu machen, müssen wir in 2 Schritten vor gehen.

Mit der Funktion `pivot_longer()` bringen wir die Daten in eine erste `long table`-Version. Für den Datensatz `untidy_df` belassen wir die Variable `Altersgruppe` und wenden die Funktion auf die übrigen Spalten an.

```
untidy_df %>%
  pivot_longer(cols = Maenner_2018:Frauen_2020,
               names_to="Geschlecht_Jahr",
               values_to = "Einkommen") %>%
  head(10)
```

```
# A tibble: 10 x 3
   Altersgruppe Geschlecht_Jahr Einkommen
   <chr>        <chr>                <dbl>
 1 < 18         Maenner_2018         22000
 2 < 18         Frauen_2018          20000
 3 < 18         Maenner_2019         22000
 4 < 18         Frauen_2019          20000
 5 < 18         Maenner_2020         23000
 6 < 18         Frauen_2020          21000
 7 18-30        Maenner_2018         36000
 8 18-30        Frauen_2018          35000
 9 18-30        Maenner_2019         36000
10 18-30        Frauen_2019          35000
```

Wie Sie sehen, hat `pivot_longer()` die `Altersgruppen` ebenfalls richtig zugeordnet.

Im zweiten Schritt teilen wir die Variable `Geschlecht_Jahr` in zwei eigene Variablen `Geschlecht` und `Jahr` auf. Hierfür steht die Funktion `separate()` zur Verfügung.

```
untidy_df %>%
  pivot_longer(cols = Maenner_2018:Frauen_2020,
               names_to="Geschlecht_Jahr",
               values_to = "Einkommen") %>%
  separate(Geschlecht_Jahr,
           into = c("Geschlecht", "Jahr")) %>%
  head(10)
```

```
# A tibble: 10 x 4
   Altersgruppe Geschlecht Jahr  Einkommen
   <chr>        <chr>      <chr>     <dbl>
 1 < 18         Maenner    2018      22000
 2 < 18         Frauen     2018      20000
 3 < 18         Maenner    2019      22000
 4 < 18         Frauen     2019      20000
 5 < 18         Maenner    2020      23000
 6 < 18         Frauen     2020      21000
 7 18-30        Maenner    2018      36000
 8 18-30        Frauen     2018      35000
 9 18-30        Maenner    2019      36000
10 18-30        Frauen     2019      35000
```

Die Daten liegen nun vollständig als *tidy data* vor.

1.6.7.1 Datentypen korrigieren

Gerade bei importierten Daten sollte überprüft werden, ob den Variablen der korrekte Datentyp (numerisch, faktorial, logisch) zugewiesen ist. Dies ist wichtig, wenn wir mit ihnen später die Statistikfunktionen aufrufen.
Schauen wir unser Pflegetibble an

```
glimpse(Pflegetibble)
```

```
Rows: 45
Columns: 3
$ Berufsgruppe <chr> "Krankenpflegeassistenz", "Krankenpflegeassistenz", "Krankenpfl
$ Jahr         <chr> "1999", "2001", "2003", "2005", "2007", "2009", "2011", "2013",
$ Anzahl       <dbl> 16624, 19061, 19478, 21537, 27731, 36481, 46517, 54371, 64127,
```

Die Variablen `Berufsgruppe` und `Jahr` sind beide vom Typ `chr`, was falsch ist. Überführen wir die Berufsgruppen in einen Faktor, und die Jahre in numerische Werte. Im klassischen R geht dies so:

```
Pflegetibble$Berufsgruppe <- as.factor(Pflegetibble$Berufsgruppe)
Pflegetibble$Jahr <- as.numeric(Pflegetibble$Jahr)
glimpse(Pflegetibble)
```

```
Rows: 45
Columns: 3
$ Berufsgruppe <fct> Krankenpflegeassistenz, Krankenpflegeassistenz, Krankenpflegeas…
$ Jahr         <dbl> 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015, 1999, 200…
$ Anzahl       <dbl> 16624, 19061, 19478, 21537, 27731, 36481, 46517, 54371, 64127, …
```

Im Tidyverse kann man die Datentypen mit den Funktionen `mutate()` (wird später erleutert) und `across()` umwandeln. Letztere ist das Pendant zum klassischen `apply()` und erlaubt es, Funktionen auf ausgewählte Spalten (Variablen) des Datensatzes anzuwenden.

```
Pflegetibble <- Pflegetibble %>%
  # wende auf Spalte "Berufsgruppe" die Funktion as.factor an.
  mutate(across(Berufsgruppe, as.factor)) %>%
  # wende auf Spalte "Jar" die Funktion as.numeric an.
  mutate(across(Jahr, as.numeric))

glimpse(Pflegetibble)
```

```
Rows: 45
Columns: 3
$ Berufsgruppe <fct> Krankenpflegeassistenz, Krankenpflegeassistenz, Krankenpflegeas…
$ Jahr         <dbl> 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015, 1999, 200…
$ Anzahl       <dbl> 16624, 19061, 19478, 21537, 27731, 36481, 46517, 54371, 64127, …
```

Sollen der auszuführenden Funktion Parameter übergeben werden, ändert sich der Aufruf - z.B. um einen ordinalen Faktore zu erzeugen, wie folgt:

```
Pflegetibble %>%
  # wende auf Spalte "Berufsgruppe" die Funktion factor an.
  # Um Parameter an 'factor()' übergeben zu können,
  # muss folgende Notation erfolgen.
  # Erstelle ordinale Faktoren mit Parameter "ordered=T"
  mutate(across(Berufsgruppe, ~factor(.x, ordered=T))) %>%
  head()
```

```
# A tibble: 6 × 3
  Berufsgruppe            Jahr Anzahl
  <ord>                  <dbl>  <dbl>
1 Krankenpflegeassistenz  1999  16624
2 Krankenpflegeassistenz  2001  19061
3 Krankenpflegeassistenz  2003  19478
4 Krankenpflegeassistenz  2005  21537
5 Krankenpflegeassistenz  2007  27731
6 Krankenpflegeassistenz  2009  36481
```

Schauen wir uns als weiteres Beispiel den gelabelten Datensazt der Nachtwachenstudie an.

```
# lade den Nachtwachendatensatz
load(url("https://www.produnis.de/R/data/nw.RData"))
#nutze die gelabelte Version
test <- nw_labelled

# wenn Sie das Paket jgsbook installiert haben,
# können Sie den Datensatz auch so laden:
test <- jgsbook::nw_labelled
glimpse(test)
```

```
Rows: 276
Columns: 49
$ sex              <fct> männlich, weiblich, weiblich, weiblich, weiblich…
$ age              <labelled> 56, 53, 55, 28, 53, 62, 55, 61, 34, 50, 54,…
$ land             <fct> Nordrhein-Westfalen, Nordrhein-Westfalen, Nordrh…
$ host             <fct> "privater Träger", "Wohlfahrtsverband (z.B. AWO,…
$ bed              <fct> weniger als 10, 75 - 100, 100 - 150, 100 - 150, …
$ patient          <fct> 10 - 20, 75 - 100, 100 - 150, 100 - 150, 75 - 10…
$ liability        <labelled> 5, 30, 34, 23, 30, 41, 79, 79, 52, 32, 71, …
$ Stufe0           <labelled> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 25, 25, 0, 0,…
$ Stufe1           <labelled> 3, 0, 10, 4, 0, 14, 15, 8, 6, 9, 15, 15, 9,…
$ Stufe2           <labelled> 9, 10, 19, 17, 11, 19, 29, 14, 32, 14, 19, …
$ Stufe3           <labelled> 12, 20, 5, 2, 19, 7, 23, 18, 14, 10, 12, 12…
$ Demenz           <labelled> 7, 20, 16, 8, 28, 7, 60, 31, 32, 22, 21, 21…
$ Bettgitter       <labelled> 2, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1…
$ Bettgurt         <labelled> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Schlafmittel     <labelled> 7, 6, 5, 0, 5, 9, 7, 2, 15, 4, 4, 4, 4, 0, …
$ colleagueall     <labelled> 2, 3, 5, 3, 3, 3, 2, 2, 2, 3, 3, 3, 3, 2, 2…
$ colleague        <labelled> 1, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2…
$ pause            <fct> selten, immer, häufig, immer, immer, manchmal, i…
$ pausecolleague   <fct> nie, immer, nie, immer, immer, nie, immer, immer…
$ hintergrunddienst <fct> NA, Ja, Ja, Ja, Ja, NA, NA, Ja, Ja, Ja, Ja, Ja, …
$ daynights        <fct> "hauptsächlich im Nachtdienst, mit vereinzelten …
$ whynight         <fct> NA, die Arbeitszeiten sagen mir zu, NA, familiär…
$ nights           <labelled> 6, 7, 8, 5, 7, 7, 7, 7, 5, 5, 7, 5, 9, 9, 8…
$ timework         <fct> Vollzeit, Teilzeit, NA, Teilzeit, Teilzeit, NA, …
$ nightstarts      <labelled> "1970-01-01 20:33:00", "1970-01-01 20:30:00…
$ nightstops       <labelled> "1970-01-01 07:07:00", "1970-01-01 07:00:00…
$ MagNacht         <fct> eher ja, ja, ja, ja, ja, ja, ja, ja, ja, ja, ja,…
$ WiederTag        <fct> neutral, nein, nein, neutral, nein, nein, nein, …
$ ueberfordert     <fct> selten, nie, nie, nie, selten, selten, nie, selt…
$ Zwischenfall     <fct> selten, selten, häufig, selten, häufig, selten, …
$ Bew_herumirren   <fct> sehr oft, häufig, selten, häufig, häufig, selten…
$ Bew_sterben      <fct> selten, häufig, häufig, selten, häufig, selten, …
$ muede            <fct> selten, nie, nie, häufig, selten, nie, nie, selt…
$ schwer_wach      <fct> nie, nie, nie, selten, selten, nie, nie, nie, ni…
$ Demenz_aktiv     <labelled> 10, 7, 5, 5, 7, 7, 3, 5, NA, 5, 5, 5, 4, 4,…
$ Demenz_stoeren   <labelled> 8, 7, 3, 3, 7, 0, 3, 2, NA, 5, 5, 5, 2, 1, …
$ Demenz_sediert   <labelled> 3, 7, 1, 0, 2, 0, 0, 1, NA, 2, 2, 2, 0, 0, …
$ Demenz_nv        <labelled> 10, 4, 4, 0, 1, 7, 0, 0, NA, 0, 0, 0, 0, 0,…
$ Sorge_weglaufen  <fct> eher ja, ja, eher ja, nein, eher nicht, eher nic…
$ Sorge_Sturz      <fct> ja, nein, ja, ja, eher ja, eher nicht, neutral, …
$ Sorge_FEM        <fct> ja, NA, NA, nein, neutral, eher ja, NA, neutral,…
```

```
$ Sorge_nv          <fct> ja, neutral, eher ja, eher nicht, nein, eher nic…
$ Sorge_sterben     <fct> eher ja, ja, ja, eher nicht, eher ja, ja, eher n…
$ workyear          <labelled> 35, 37, 37, 10, 30, 34, 13, 28, 4, 32, 10, …
$ worknight         <labelled> 12, 11, 32, 1, 14, 29, 7, 13, 2, 5, 10, 5, …
$ meeting           <fct> selten, immer, NA, nie, immer, manchmal, selten,…
$ fortbildung       <fct> Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, …
$ wannfort          <labelled> 2004, 2014, 2013, 2013, 2014, 2015, 2014, 2…
$ edu               <fct> dreijährige Pflegeausbildung, zweijährige Pflege…
```

Wir sehen, dass viele Variablen vom Typ `labelled` sind. Schauen wir uns die Variablen `age` und `sex` genauer an.

```
1 class(test$age)
```

```
[1] "labelled" "numeric"
```

```
1 class(test$sex)
```

```
[1] "labelled" "factor"
```

Die Variablen ist sowhohl vom Typ `labelled` als auch vom Typ `numeric` bzw. `factor`. Manche -Funktionen haben mit dieser Doppelbelegung ihre Probleme, z.B. `vtable::sumtable()`.

```
1 vtable::sumtable(test$age)
```

```
Fehler in vtable::sumtable(test$age) :
It doesn't look like you have any variables that belong in a sumtable.
Check your data. Use vars to explicitly choose variables, or convert
things to numeric or factor before sending to sumtable.
```

Wir sollten daher alle Variablentypen eindeutig setzen. Die Funktion `mutate_if()` wendet eine Funktion an, wenn die gesetzten Bedingungen erfüllt sind. So möchten wir im ersten Schritt nur solche Spalten, die (zusätzlich zum Typ `labelled`) vom Typ `numeric` sind, den eindeutigen Typ `numeric` zuweisen. Im zweiten Schritt wiederholen wir dies für Faktoren.

```
1 test %>%
2   # wende nur auf labelled-numeric an
3   mutate_if(is.numeric, as.numeric) %>%
4   # wende nur auf labelled-factor an
5   mutate_if(is.factor, as.factor) %>%
6   # wende nur auf die Spalet nightstarts nightstops an
7   mutate(across(nightstarts:nightstops, as.character)) %>%
8   glimpse()
```

```
Rows: 276
Columns: 49
$ sex               <fct> männlich, weiblich, weiblich, weiblich, weiblich…
$ age               <dbl> 56, 53, 55, 28, 53, 62, 55, 61, 34, 50, 54, 50, …
$ land              <fct> Nordrhein-Westfalen, Nordrhein-Westfalen, Nordrh…
$ host              <fct> "privater Träger", "Wohlfahrtsverband (z.B. AWO,…
$ bed               <fct> weniger als 10, 75 - 100, 100 - 150, 100 - 150, …
$ patient           <fct> 10 - 20, 75 - 100, 100 - 150, 100 - 150, 75 - 10…
$ liability         <dbl> 5, 30, 34, 23, 30, 41, 79, 79, 52, 32, 71, 71, 4…
$ Stufe0            <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 25, 25, 0, 0, 0, 0…
$ Stufe1            <dbl> 3, 0, 10, 4, 0, 14, 15, 8, 6, 9, 15, 15, 9, 10, …
$ Stufe2            <dbl> 9, 10, 19, 17, 11, 19, 29, 14, 32, 14, 19, 19, 2…
$ Stufe3            <dbl> 12, 20, 5, 2, 19, 7, 23, 18, 14, 10, 12, 12, 14,…
```

```
$ Demenz            <dbl> 7, 20, 16, 8, 28, 7, 60, 31, 32, 22, 21, 21, 20,…
$ Bettgitter        <dbl> 2, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 11, 1,…
$ Bettgurt          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ Schlafmittel      <dbl> 7, 6, 5, 0, 5, 9, 7, 2, 15, 4, 4, 4, 4, 0, 26, 1…
$ colleagueall      <dbl> 2, 3, 5, 3, 3, 3, 2, 2, 2, 3, 3, 3, 3, 2, 2, 6, …
$ colleague         <dbl> 1, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, …
$ pause             <fct> selten, immer, häufig, immer, immer, manchmal, i…
$ pausecolleague    <fct> nie, immer, nie, immer, immer, nie, immer, immer…
$ hintergrunddienst <fct> NA, Ja, Ja, Ja, Ja, NA, NA, Ja, Ja, Ja, Ja, Ja, …
$ daynights         <fct> "hauptsächlich im Nachtdienst, mit vereinzelten …
$ whynight          <fct> NA, die Arbeitszeiten sagen mir zu, NA, familiär…
$ nights            <dbl> 6, 7, 8, 5, 7, 7, 7, 7, 5, 5, 7, 5, 9, 9, 8, 8, …
$ timework          <fct> Vollzeit, Teilzeit, NA, Teilzeit, Teilzeit, NA, …
$ nightstarts       <chr> "1970-01-01 20:33:00", "1970-01-01 20:30:00", "1…
$ nightstops        <chr> "1970-01-01 07:07:00", "1970-01-01 07:00:00", "1…
$ MagNacht          <fct> eher ja, ja, ja, ja, ja, ja, ja, ja, ja, ja, ja,…
$ WiederTag         <fct> neutral, nein, nein, neutral, nein, nein, nein, …
$ ueberfordert      <fct> selten, nie, nie, nie, selten, selten, nie, selt…
$ Zwischenfall      <fct> selten, selten, häufig, selten, häufig, selten, …
$ Bew_herumirren    <fct> sehr oft, häufig, selten, häufig, häufig, selten…
$ Bew_sterben       <fct> selten, häufig, häufig, selten, häufig, selten, …
$ muede             <fct> selten, nie, nie, häufig, selten, nie, nie, selt…
$ schwer_wach       <fct> nie, nie, nie, selten, selten, nie, nie, nie, ni…
$ Demenz_aktiv      <dbl> 10, 7, 5, 5, 7, 7, 3, 5, NA, 5, 5, 5, 4, 4, 7, 7…
$ Demenz_stoeren    <dbl> 8, 7, 3, 3, 7, 0, 3, 2, NA, 5, 5, 5, 2, 1, 7, 7,…
$ Demenz_sediert    <dbl> 3, 7, 1, 0, 2, 0, 0, 1, NA, 2, 2, 2, 0, 0, 5, 7,…
$ Demenz_nv         <dbl> 10, 4, 4, 0, 1, 7, 0, 0, NA, 0, 0, 0, 0, 0, 7, 7…
$ Sorge_weglaufen   <fct> eher ja, ja, eher ja, nein, eher nicht, eher nic…
$ Sorge_Sturz       <fct> ja, nein, ja, ja, eher ja, eher nicht, neutral, …
$ Sorge_FEM         <fct> ja, NA, NA, nein, neutral, eher ja, NA, neutral,…
$ Sorge_nv          <fct> ja, neutral, eher ja, eher nicht, nein, eher nic…
$ Sorge_sterben     <fct> eher ja, ja, ja, eher nicht, eher ja, ja, eher n…
$ workyear          <dbl> 35, 37, 37, 10, 30, 34, 13, 28, 4, 32, 10, 32, 2…
$ worknight         <dbl> 12, 11, 32, 1, 14, 29, 7, 13, 2, 5, 10, 5, 17, 3…
$ meeting           <fct> selten, immer, NA, nie, immer, manchmal, selten,…
$ fortbildung       <fct> Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, Ja, …
$ wannfort          <dbl> 2004, 2014, 2013, 2013, 2014, 2015, 2014, 2015, …
$ edu               <fct> dreijährige Pflegeausbildung, zweijährige Pflege…
```

1.6.7.2 fehlende Werte

Fehlende Werte (`NA`) können mit der Funktion `drop_na()` entfernt werden. Die Funktion "dropt" dabei die komplette Datenreihe, wenn ein `NA` in einer angegebenen Spalte enthalten ist.

```
# lade Beispieldatensatz
load(url("https://www.produnis.de/R/data/pf8.RData" ))

# Schaue Variable "Gewicht" an
glimpse(pf8$Gewicht)
```

```
num [1:731] 69 67 NA 90 68 60 80 60 NA 60 ...
```

Variable `Gewicht` enthält 731 Werte, und wir sehen, dass auch `NA` enthalten sind. Jetzt "droppen"wir alle Reihen, in denen bei der Variable `Gewicht` ein `NA` steht

```
pf8 %>%
  drop_na(Gewicht) %>%
      glimpse(.)
```

```
Rows: 720
Columns: 16
$ Standort       <fct> Münster, Münster, Münster, Münster, Münster, Münster, Münster…
$ Alter          <int> 18, 67, 61, 24, 21, 59, 56, 52, 79, 22, 19, 18, 17, 21, 28, 7…
$ Geschlecht     <fct> weiblich, weiblich, männlich, männlich, weiblich, weiblich, w…
$ Größe          <int> 172, 165, 182, 173, 177, 168, 156, 166, 161, 206, 163, 172, 1…
$ Gewicht        <dbl> 69, 67, 90, 68, 60, 80, 60, 60, 66, 130, 52, 68, 48, 54, 70, …
$ Bildung        <fct> Abitur, mittlere Reife, mittlere Reife, Abitur, Abitur, mittl…
$ Beruf          <fct> Inspektor*in, Rentner*in, Beamter*in, Student*in, Student*in,…
$ Familienstand  <fct> Partnerschaft, geschieden, ledig, ledig, Partnerschaft, verhe…
$ Kinder         <int> 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Wohnort        <fct> ländlich, ländlich, ländlich, städtisch, städtisch, ländlich,…
$ Rauchen        <fct> nein, nein, nein, nein, ja, nein, ja, nein, nein, nein, nein,…
$ SportHäufig    <dbl> NA, 2.0, 4.0, 4.0, 1.0, 2.0, 1.0, 2.0, NA, 4.0, 5.0, 2.5, 1.0…
$ SportMinuten   <dbl> NA, 60, 120, 60, 60, 45, 90, 45, NA, 90, 20, 120, 120, 80, NA…
$ SportWie       <fct> Allein, Gruppe, Allein, Allein, Gruppe, Gruppe, beides, Allei…
$ SportWarum     <fct> Arbeitsweg, Vorbeugung, Fitness, Gewichtsreduktion, Fitness, …
$ LebenZufrieden <dbl> 5, 7, 2, 9, 8, 5, 8, 8, 8, 8, 7, 8, 7, 8, 7, 7, 8, 7, 10, 8, …
```

Der Datensatz `pf8` wird mittels `Pipe` weitergeleitet. In der zweiten Zeile droppen wir alle Reihen mit `Gewicht NA` und geben das Resultat per `Pipe` weiter.
Mit `glimpse()` sehen wir das Resultat. Datensatz `pf8` hat jetzt nur noch 720 Reihen, und nicht mehr 731.

Sollen *alle* `NA`s in allen Spalten entfernt werden, lautet der Befehl

```
pf8 %>%
  drop_na(.) %>%
      glimpse(.)
```

```
Rows: 398
Columns: 16
$ Standort       <fct> Münster, Münster, Münster, Münster, Münster, Münster, Münster…
$ Alter          <int> 67, 61, 24, 21, 59, 56, 52, 22, 19, 18, 17, 21, 72, 19, 69, 2…
$ Geschlecht     <fct> weiblich, männlich, männlich, weiblich, weiblich, weiblich, w…
$ Größe          <int> 165, 182, 173, 177, 168, 156, 166, 206, 163, 172, 158, 163, 1…
$ Gewicht        <dbl> 67, 90, 68, 60, 80, 60, 60, 130, 52, 68, 48, 54, 86, 65, 89, …
$ Bildung        <fct> mittlere Reife, mittlere Reife, Abitur, Abitur, mittlere Reif…
$ Beruf          <fct> Rentner*in, Beamter*in, Student*in, Student*in, arbeitlos, Pf…
$ Familienstand  <fct> geschieden, ledig, ledig, Partnerschaft, verheiratet, geschie…
$ Kinder         <int> 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Wohnort        <fct> ländlich, ländlich, städtisch, städtisch, ländlich, städtisch…
$ Rauchen        <fct> nein, nein, nein, ja, nein, ja, nein, nein, nein, nein, nein,…
$ SportHäufig    <dbl> 2.0, 4.0, 4.0, 1.0, 2.0, 1.0, 2.0, 4.0, 5.0, 2.5, 1.0, 4.0, 4…
$ SportMinuten   <dbl> 60, 120, 60, 60, 45, 90, 45, 90, 20, 120, 120, 80, 160, 120, …
$ SportWie       <fct> Gruppe, Allein, Allein, Gruppe, Gruppe, beides, Allein, Grupp…
$ SportWarum     <fct> Vorbeugung, Fitness, Gewichtsreduktion, Fitness, Vorbeugung, …
$ LebenZufrieden <dbl> 7, 2, 9, 8, 5, 8, 8, 8, 7, 8, 7, 8, 7, 8, 10, 8, 7, 9, 8, 8, …
```

Mit dieser radikalen Methode schrumpft der Datensatz pf8 auf 398 vollständige Datenreihen zusammen.

Sie können mit den Funktionen `fill()` und `replace_na()` fehlende Werte auffüllen oder ersetzen. Hierbei ersetzt `replace_na()` alle `NA` durch einen vorgegebenen Wert, und `fill()` versucht, die `NA` durch "wahrscheinliche" Werte zu ersetzen. Dies ist hilfreich, wenn die Daten z.B. nach einem Kriterium sortiert sind.

```
test <- tibble(Beruf = factor(c("Bäcker", NA, "Bäcker", "Ärztin", NA, "Ärztin")))

# vorher
test
```

```
## # A tibble: 6 x 1
##   Beruf
##   <fct>
## 1 Bäcker
## 2 <NA>
## 3 Bäcker
## 4 Ärztin
## 5 <NA>
## 6 Ärztin
```

```
# ersetze NA
test %>%
  tidyr::fill(Beruf)
```

```
## # A tibble: 6 x 1
##   Beruf
##   <fct>
## 1 Bäcker
## 2 Bäcker
## 3 Bäcker
## 4 Ärztin
## 5 Ärztin
## 6 Ärztin
```

Aufsteigende Werte können **nicht** erzeugt werden.

```
test <- tibble(Jahr = c(2014, NA, NA, NA, 2018, 2019, NA , NA , 2022))

# vorher
test
```

```
## # A tibble: 9 x 1
##    Jahr
##   <dbl>
## 1  2014
## 2    NA
## 3    NA
## 4    NA
## 5  2018
## 6  2019
## 7    NA
## 8    NA
## 9  2022
```

```
test %>%
  tidyr::fill(Jahr)
```

```
## # A tibble: 9 x 1
##    Jahr
##   <dbl>
## 1  2014
## 2  2014
## 3  2014
## 4  2014
## 5  2018
## 6  2019
## 7  2019
## 8  2019
## 9  2022
```

1.6.7.3 Variablen umbenennen

Mit der Funktion `rename()` könne Sie die Variablennamen ändern.

```
Pflegetibble %>%
  rename(Neu=Berufsgruppe) %>%
    head()
```

```
# A tibble: 6 x 3
  Neu                    Jahr Anzahl
  <fct>                 <dbl>  <dbl>
1 Krankenpflegeassistenz  1999  16624
2 Krankenpflegeassistenz  2001  19061
3 Krankenpflegeassistenz  2003  19478
4 Krankenpflegeassistenz  2005  21537
5 Krankenpflegeassistenz  2007  27731
6 Krankenpflegeassistenz  2009  36481
```

```
Pflegetibble %>%
  rename(graduate=Berufsgruppe,
         year=Jahr,
         n=Anzahl) %>%
            head()
```

```
# A tibble: 6 x 3
  graduate               year     n
  <fct>                 <dbl> <dbl>
1 Krankenpflegeassistenz  1999 16624
2 Krankenpflegeassistenz  2001 19061
3 Krankenpflegeassistenz  2003 19478
4 Krankenpflegeassistenz  2005 21537
5 Krankenpflegeassistenz  2007 27731
6 Krankenpflegeassistenz  2009 36481
```

1.6.8 Schritt 3: Umgang mit Datensätzen

Das Paket `dplyr` bietet zahlreiche Funktionen, die den Umgang mit Datensätzen erleichtern.

1.6.8.1 Daten filtern und sortieren

Mit der Funktion `filter()` kann der Datensatz nach Kriterien gefiltert werden. Beachten Sie, dass die Funktion mit `stats::filter()` kollidiert. Rufen Sie sie daher sicherheitshalber mit `dplyr::filter()` auf.

```
pf8 %>%
    # Nur Daten von Personen jünger 40
    dplyr::filter(Alter<40) %>%
      glimpse()
```

```
## Rows: 443
## Columns: 16
## $ Standort       <fct> Münster, Münster, Münster, Münster, Münster, Münster, M…
## $ Alter          <int> 18, 24, 21, 22, 19, 18, 17, 21, 28, 19, 21, 28, 26, 21,…
## $ Geschlecht     <fct> weiblich, männlich, weiblich, männlich, weiblich, weibl…
## $ Größe          <int> 172, 173, 177, 206, 163, 172, 158, 163, 181, 175, 186, …
## $ Gewicht        <dbl> 69, 68, 60, 130, 52, 68, 48, 54, 70, 65, 70, 85, 60, 65…
```

```
## $ Bildung        <fct> Abitur, Abitur, Abitur, Fachabitur, Abitur, mittlere Re…
## $ Beruf          <fct> Inspektor*in, Student*in, Student*in, Kaufmann/frau, St…
## $ Familienstand  <fct> Partnerschaft, ledig, Partnerschaft, ledig, ledig, ledi…
## $ Kinder         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Wohnort        <fct> ländlich, städtisch, städtisch, ländlich, städtisch, st…
## $ Rauchen        <fct> nein, nein, ja, nein, nein, nein, nein, nein, nein, nei…
## $ SportHäufig    <dbl> NA, 4.0, 1.0, 4.0, 5.0, 2.5, 1.0, 4.0, NA, 0.5, 1.0, 5.…
## $ SportMinuten   <dbl> NA, 60, 60, 90, 20, 120, 120, 80, NA, 120, 60, 50, 30, …
## $ SportWie       <fct> Allein, Allein, Gruppe, Gruppe, Allein, Gruppe, Gruppe,…
## $ SportWarum     <fct> Arbeitsweg, Gewichtsreduktion, Fitness, Fitness, Vorbeu…
## $ LebenZufrieden <dbl> 5, 9, 8, 8, 7, 8, 7, 8, 7, 8, 7, 8, 7, 9, 8, 8, 6, 7, 8…
```

Die Kriterien können auch kombiniert werden

```
pf8 %>%
  dplyr::filter(Alter>20 , Geschlecht == "weiblich", Bildung=="Abitur") %>%
      glimpse()
```

```
## Rows: 56
## Columns: 16
## $ Standort       <fct> Münster, Münster, Münster, Bahn, Bahn, Rheine, Rheine, …
## $ Alter          <int> 21, 21, 21, 21, 21, 22, 23, 22, 30, 52, 45, 23, 49, 41,…
## $ Geschlecht     <fct> weiblich, weiblich, weiblich, weiblich, weiblich, weibl…
## $ Größe          <int> 177, 163, 175, 170, 170, 170, 168, 163, 173, 174, 168, …
## $ Gewicht        <dbl> 60, 54, 61, 58, 80, 59, 70, 63, 71, 72, 70, 56, 70, 49,…
## $ Bildung        <fct> Abitur, Abitur, Abitur, Abitur, Abitur, Abitur, Abitur,…
## $ Beruf          <fct> Student*in, Student*in, Student*in, Student*in, Student…
## $ Familienstand  <fct> Partnerschaft, Partnerschaft, Partnerschaft, NA, ledig,…
## $ Kinder         <int> 0, 0, 0, NA, 0, 0, 0, 0, 0, 1, 2, 0, 2, 0, 3, 2, 0, 1, …
## $ Wohnort        <fct> städtisch, städtisch, städtisch, ländlich, ländlich, st…
## $ Rauchen        <fct> ja, nein, nein, nein, nein, nein, nein, nein, nein, nei…
## $ SportHäufig    <dbl> 1.0, 4.0, 3.0, 5.0, 1.0, 1.0, 2.0, 4.0, 3.0, 4.0, 1.0, …
## $ SportMinuten   <dbl> 60, 80, 60, 75, 60, 90, 45, 60, 60, 20, 60, 90, 30, 120…
## $ SportWie       <fct> Gruppe, beides, Gruppe, Allein, Allein, Allein, beides,…
## $ SportWarum     <fct> Fitness, Vorbeugung, Fitness, Fitness, Fitness, Gewicht…
## $ LebenZufrieden <dbl> 8, 8, 8, 9, 8, 8, 10, 8, 7, 6, 7, 8, 7, 7, 7, 7, 8, 8, …
```

Beachten Sie, dass wir doppelte Gleichheitszeichen (==) verwenden müssen, da mit *einem* Gleichheitszeichen (=) Variablen zugewiesen werden (so wie mit <-).

Möchte man nach mehreren bestimmten Werte filtern, kann der `%in%` Operator verwendet werden. Hierdurch übergeben Sie einen Vektor an möglichen Ausprägungen. Angenommen, Sie möchten nur solche Fälle auswählen, in denen das Alter 20 *oder* 25 *oder* 40 Jahre beträgt, lautet der Aufruf:

```
# nach mehreren Werten filtern
# Suche Fälle, die 20 oder 25 oder 40 Jahre alt sind
pf8 %>%
  dplyr::filter(Alter %in% c(20, 25, 40)) %>%
  head(10)
```

```
   Standort Alter Geschlecht Größe Gewicht     Bildung          Beruf Familienstand
1   Münster    25   männlich   182    75.0  Ausbildung     Finanzwirt         ledig
2      Bahn    25   männlich   180    72.0  Hochschule    etriebswirt         ledig
3    Rheine    25   weiblich   168    75.0  Hochschule     Student*in         ledig
4    Rheine    40   weiblich   163    68.0 Hauptschule  Angestellte*r         ledig
5    Rheine    20   weiblich   158    70.0 Hauptschule     Schüler*in         ledig
6    Rheine    20   weiblich   170    85.0  Hochschule          Azubi         ledig
7    Rheine    20   weiblich   170    65.0      Abitur          Azubi         ledig
8    Rheine    20   weiblich   160    48.5      Abitur     Student*in         ledig
9    Rheine    25   männlich   186    70.0  Hochschule     Student*in         ledig
10     Bahn    40   weiblich   181    64.0  Hochschule    gestellte*r   verheiratet
```

Über den Operator `!=` können Fälle ausgeschlossen werden. Wir schließen für dieses Beispiel alle Fälle aus, die verheiratet sind:

```
# schließe alle verheirateten Fälle aus
pf8 %>%
  dplyr::filter(Familienstand != "verheiratet") %>%
  head(10)
```

```
   Standort Alter Geschlecht Größe Gewicht        Bildung            Beruf Familienstand
1   Münster    18   weiblich   172      69         Abitur     Inspektor*in Partnerschaft
2   Münster    67   weiblich   165      67 mittlere Reife       Rentner*in    geschieden
3   Münster    60   weiblich   175      NA     Hochschule Ergotherapeut*in Partnerschaft
4   Münster    61   männlich   182      90 mittlere Reife       Beamter*in         ledig
5   Münster    24   männlich   173      68         Abitur       Student*in         ledig
6   Münster    21   weiblich   177      60         Abitur       Student*in Partnerschaft
7   Münster    56   weiblich   156      60     Ausbildung  Pflegehelfer*in    geschieden
8   Münster    22   männlich   206     130     Fachabitur    Kaufmann/frau         ledig
9   Münster    19   weiblich   163      52         Abitur       Student*in         ledig
10  Münster    18   weiblich   172      68 mittlere Reife       Schüler*in         ledig
```

Mit der Funktion `arrange()` können die Daten sortiert werden. Dabei wird standardmäßig *aufsteigend* sortiert.

```
# sortiere Datensatz "pf8"
pf8 %>%
  arrange(Alter, Größe, Gewicht) %>%
     head(.)
```

```
  Standort Alter Geschlecht Größe Gewicht        Bildung      Beruf Familienstand
1 Internet    15   weiblich   165    51.6           <NA>            Partnerschaft
2  Münster    17   weiblich   158    48.0 mittlere Reife Schüler*in         ledig
3 Internet    17   weiblich   160    75.0           <NA>            Partnerschaft
4 Internet    17   weiblich   163    57.0           <NA>            Partnerschaft
5 Internet    17   weiblich   168    59.0           <NA>                    ledig
6 Internet    17   weiblich   175    80.0           <NA>                    ledig
```

Die Daten werden zunächst aufsteigend nach `Alter` sortiert. Gibt es Fälle mit dem selben Alter, wird nach dem zweiten Kriterium (in diesem Falle `Größe`) sortiert. Gibt es Fälle mit der selben `Größe` und dem selben `Alter`, wird innerhalb dieser Gruppe nach `Gewicht` sortiert.

Für eine *absteigende* Sortierung muss die Funktion `desc()` verwendet werden

```
# sortiere Datensatz "pf8"
# diesmal absteigend
pf8 %>%
  arrange(desc(Alter), desc(Größe), desc(Gewicht)) %>%
     head(.)
```

```
   Standort Alter Geschlecht Größe Gewicht     Bildung        Beruf Familienstand
1   Münster    86   weiblich   162      70 Hauptschule   Rentner*in     verwitwet
2    Rheine    86   weiblich   160      60      Abitur  Postbote*in     verwitwet
3    Rheine    84   weiblich   164      70 Hauptschule Verkäufer*in     verwitwet
4    Rheine    84   weiblich   160      60 Hauptschule   Rentner*in     verwitwet
5 Ladbergen    83   männlich   182      88 Hauptschule  Rentener*in   verheiratet
6   Münster    83       <NA>   165     175        <NA>   Rentner*in         ledig
```

1.6.8.2 Fälle auswählen

Mit der Funktion `slice()` können gewünschte Datenreihen (Fälle) ausgegeben werden

```
pf8 %>%
  # zeige Fälle (Reihen) 98 bis 105
  slice(98:105) %>%
    glimpse()
```

```
## Rows: 8
## Columns: 16
## $ Standort       <fct> Rheine, Rheine, Rheine, Rheine, Rheine, Rheine, Rheine,…
## $ Alter          <int> 23, 30, 26, 35, 22, 50, 35, 22
## $ Geschlecht     <fct> männlich, weiblich, männlich, männlich, männlich, männl…
## $ Größe          <int> 194, 173, 189, 184, 181, 204, 172, 175
## $ Gewicht        <dbl> 110, 71, 89, 79, 92, 102, 86, 75
## $ Bildung        <fct> Abitur, Abitur, Hauptschule, mittlere Reife, mittlere R…
## $ Beruf          <fct> Außendienst, Innendienst, Angestellte*r,  Produktionsle…
## $ Familienstand  <fct> Partnerschaft, verheiratet, Partnerschaft, verheiratet,…
## $ Kinder         <int> 0, 0, 0, 3, 0, 1, 2, 0
## $ Wohnort        <fct> ländlich, ländlich, städtisch, ländlich, ländlich, länd…
## $ Rauchen        <fct> ja, nein, ja, ja, nein, nein, ja, nein
## $ SportHäufig    <dbl> 1, 3, 1, 3, 2, 4, 1, 0
## $ SportMinuten   <dbl> 30, 60, 60, 90, 30, 30, 60, 0
## $ SportWie       <fct> Allein, Allein, Gruppe, Gruppe, Gruppe, beides, Gruppe,…
## $ SportWarum     <fct> Fitness, Fitness, Freizeit, Fitness, Fitness, Vorbeugun…
## $ LebenZufrieden <dbl> 9, 7, 6, 6, 7, 8, 7, 7
```

Mit `slice_min()` und `slice_max()` können die kleinsten und größten Werte eingesehen werden.

```
pf8 %>%
  # zeige die leichtesten Gewichte
  slice_min(Gewicht)%>%
      glimpse()
```

```
## Rows: 5
## Columns: 16
## $ Standort       <fct> Münster, Münster, Münster, Internet, Internet
## $ Alter          <int> 26, 24, 24, 24, 19
## $ Geschlecht     <fct> männlich, weiblich, weiblich, weiblich, weiblich
## $ Größe          <int> 180, 164, 160, 159, 155
## $ Gewicht        <dbl> 45, 45, 45, 45, 45
## $ Bildung        <fct> Hauptschule, Abitur, Abitur, NA, NA
## $ Beruf          <fct> Kaufmann/frau, Student*in, Student*in, ,
## $ Familienstand  <fct> ledig, Partnerschaft, Partnerschaft, Partnerschaft, led…
## $ Kinder         <int> 0, 0, 0, 0, 0
## $ Wohnort        <fct> ländlich, städtisch, städtisch, ländlich, städtisch
## $ Rauchen        <fct> ja, nein, nein, nein, nein
## $ SportHäufig    <dbl> NA, 2.0, 3.0, 4.0, 2.5
## $ SportMinuten   <dbl> NA, 60, 60, 60, 90
## $ SportWie       <fct> NA, Gruppe, Allein, Allein, Allein
## $ SportWarum     <fct> NA, Vorbeugung, Vorbeugung, NA, NA
## $ LebenZufrieden <dbl> 10, 8, 9, 10, 9
```

Die Tabelle wird nach `Gewicht` sortiert ausgegeben. Im Datensatz `pf8` haben insgesamt 5 Fälle das kleinste `Gewicht`.

Der Parameter `n` gibt an, wieviele "Stufen" gezählt werden sollen, also z.B. der *kleinste* (n=1) oder auch der *zweit-kleinste* Wert (n=2) usw..
Intern zählt `n` allerdings mit, wieviele Fälle (Reihen) bereits ausgegeben wurde. Das ist am Anfang etwas un-intuitiv.

```
pf8 %>%
  # zeigen die leichtesten Gewichte
  slice_min(Gewicht, n=3)%>%
      glimpse()
```

```
## Rows: 5
## Columns: 16
## $ Standort       <fct> Münster, Münster, Münster, Internet, Internet
## $ Alter          <int> 26, 24, 24, 24, 19
## $ Geschlecht     <fct> männlich, weiblich, weiblich, weiblich, weiblich
## $ Größe          <int> 180, 164, 160, 159, 155
## $ Gewicht        <dbl> 45, 45, 45, 45, 45
## $ Bildung        <fct> Hauptschule, Abitur, Abitur, NA, NA
## $ Beruf          <fct> Kaufmann/frau, Student*in, Student*in, ,
## $ Familienstand  <fct> ledig, Partnerschaft, Partnerschaft, Partnerschaft, led…
## $ Kinder         <int> 0, 0, 0, 0, 0
## $ Wohnort        <fct> ländlich, städtisch, städtisch, ländlich, städtisch
## $ Rauchen        <fct> ja, nein, nein, nein, nein
## $ SportHäufig    <dbl> NA, 2.0, 3.0, 4.0, 2.5
## $ SportMinuten   <dbl> NA, 60, 60, 60, 90
## $ SportWie       <fct> NA, Gruppe, Allein, Allein, Allein
## $ SportWarum     <fct> NA, Vorbeugung, Vorbeugung, NA, NA
## $ LebenZufrieden <dbl> 10, 8, 9, 10, 9
```

Das Ergebnis ist evtl. anders, als Sie es erwarten. Es werden weiterhin 5 Fälle angezeigt, obwohl `n` auf 3 gesetzt wurde. Das liegt daran, dass 5 Fälle das kleinste `Gewicht` haben, und diese werden vollständig angezeigt. Intern ist `n` dabei auf 5 angewachsen, weshalb keine *zweit-kleinsten* Gewichte mehr ausgegeben werden.

Lassen wir uns die Älteste Probanden auswählen.

```
pf8 %>%
  # zeige die ältesten
  slice_max(Alter, n=2) %>%
      glimpse()
```

```
## Rows: 2
## Columns: 16
## $ Standort       <fct> Münster, Rheine
## $ Alter          <int> 86, 86
## $ Geschlecht     <fct> weiblich, weiblich
## $ Größe          <int> 162, 160
## $ Gewicht        <dbl> 70, 60
## $ Bildung        <fct> Hauptschule, Abitur
## $ Beruf          <fct> Rentner*in, Postbote*in
## $ Familienstand  <fct> verwitwet, verwitwet
## $ Kinder         <int> 0, 1
## $ Wohnort        <fct> ländlich, städtisch
## $ Rauchen        <fct> nein, nein
## $ SportHäufig    <dbl> 2, 2
## $ SportMinuten   <dbl> 45, 45
## $ SportWie       <fct> Gruppe, Gruppe
## $ SportWarum     <fct> Fitness, Freizeit
## $ LebenZufrieden <dbl> 7, 8
```

Die Tabelle besteht zwar aus 2 Fällen, aber beide Fälle haben das selbe `Alter`. Die *zweit-ältesten* werden nicht mit angezeigt, da `n` intern bereits auf 2 angewachsen ist.

Wir eröhen `n` auf 3:

```
pf8 %>%
  # zeige die ältesten
  slice_max(Alter, n=3)%>%
      glimpse()
```

```
## Rows: 4
## Columns: 16
## $ Standort       <fct> Münster, Rheine, Rheine, Rheine
## $ Alter          <int> 86, 86, 84, 84
## $ Geschlecht     <fct> weiblich, weiblich, weiblich, weiblich
## $ Größe          <int> 162, 160, 160, 164
## $ Gewicht        <dbl> 70, 60, 60, 70
## $ Bildung        <fct> Hauptschule, Abitur, Hauptschule, Hauptschule
## $ Beruf          <fct> Rentner*in, Postbote*in, Rentner*in, Verkäufer*in
## $ Familienstand  <fct> verwitwet, verwitwet, verwitwet, verwitwet
## $ Kinder         <int> 0, 1, 0, 0
## $ Wohnort        <fct> ländlich, städtisch, städtisch, städtisch
## $ Rauchen        <fct> nein, nein, nein, nein
## $ SportHäufig    <dbl> 2, 2, 2, NA
## $ SportMinuten   <dbl> 45, 45, 60, NA
## $ SportWie       <fct> Gruppe, Gruppe, Allein, NA
## $ SportWarum     <fct> Fitness, Freizeit, Fitness, NA
## $ LebenZufrieden <dbl> 7, 8, 8, 5
```

Jetzt haben wir insgesamt 4 Fälle. Es werden zuerst die *ältesten* angezeigt. Da es 2 Fälle sind, steigt `n` intern auf 2 an. Da wir `n` mit 3 aufgerufen haben, ist noch "Platz" für die *zweitältesten*. Da dies ebenfalls 2 Fälle sind, werden beide ausgegeben. Insgesamt sehen wir also 4 Fälle, obwohl wir `n` mit 3 aufgerufen haben.

Ähnlich wie `head()` und `tail()` zeigen `slice_head()` und `slice_tail()` die ersten bzw. letzten Fälle an.

```
# zeige die ersten 5 Fälle
pf8 %>% slice_head(n=5)
```

```
  Standort Alter Geschlecht Größe Gewicht        Bildung            Beruf Familienstand
1  Münster    18   weiblich   172      69         Abitur     Inspektor*in Partnerschaft
2  Münster    67   weiblich   165      67 mittlere Reife       Rentner*in    geschieden
3  Münster    60   weiblich   175      NA     Hochschule Ergotherapeut*in Partnerschaft
4  Münster    61   männlich   182      90 mittlere Reife       Beamter*in         ledig
5  Münster    24   männlich   173      68         Abitur       Student*in         ledig
```

```
# zeige die letzten 2 Fälle
pf8 %>% slice_tail(n=2)
```

```
  Standort Alter Geschlecht Größe Gewicht Bildung Beruf Familienstand Kinder   Wohnort
1 Internet    20   weiblich   168      58    <NA>              ledig      0 städtisch
2 Internet    40   männlich   183      78    <NA>         verheiratet      2 städtisch
```

Mit `slice_sample()` können zufällig Fälle aus dem Datensatz gezogen werden.

```
# ziehe zufällig 7 Fälle
pf8 %>% slice_sample(n=7)
```

```
  Standort Alter Geschlecht Größe Gewicht     Bildung             Beruf Familienstand Kinder
1 Internet    70   weiblich   158      75        <NA>                   geschieden      0
2 Internet    27   männlich   184      83        <NA>                        ledig      0
3 Internet    28   weiblich   168      74        <NA>                  verheiratet      0
4  Münster    86   weiblich   162      70 Hauptschule       Rentner*in     verwitwet      0
5  Münster    25   weiblich   180      77  Fachabitur           Pflege        ledig      0
6  Münster    20   weiblich   165      85      Abitur            Azubi        ledig      0
7  Münster    57   männlich   161      60 Hauptschule Sicherheitsdienst   verheiratet      0
```

1.6.8.2.1 Gruppierungen

Mit der Funktion `group_by()` können Gruppierungen vorgenommen werden.
In Kombination mit `group_by()` zeigen die `slice()`-Funktionen die jeweilige Auswahl an Fällen pro Gruppe.

```
pf8 %>%
  drop_na() %>%
  # Gruppiere nach Geschlecht
  group_by(Geschlecht) %>%
  #  zeige
  slice(1)
```

```
# A tibble: 3 x 16
# Groups:   Geschlecht [3]
  Standort Alter Geschlecht Größe Gewicht Bildung  Beruf Familienstand Kinder
  <fct>    <int> <fct>      <int>   <dbl> <fct>    <fct> <fct>           <int>
1 Münster     61 männlich     182      90 mittler… Beam… ledig               0
2 Münster     67 weiblich     165      67 mittler… Rent… geschieden          0
3 Rheine      32 divers       186      92 Hochsch… Sozi… verheiratet         0
```

```
pf8 %>%
  drop_na() %>%
  # Gruppiere nach Geschlecht
  group_by(Geschlecht) %>%
  # sortiere nach Alter
  arrange(desc(Alter)) %>%
  #  zeige die ersten 3 Fälle pro Gruppe
  slice(1:3)
```

```
# A tibble: 7 x 16
# Groups:   Geschlecht [3]
  Standort  Alter Geschlecht Größe Gewicht Bildung Beruf Familienstand Kinder
  <fct>     <int> <fct>      <int>   <dbl> <fct>   <fct> <fct>           <int>
1 Ladbergen    83 männlich     182    88   Haupts… Rent… verheiratet         0
2 Münster      78 männlich     178    71.5 Hochsc… Lehr… verheiratet         0
3 Bahn         76 männlich     180    90   Hochsc… Arzt… verheiratet         0
4 Münster      86 weiblich     162    70   Haupts… Rent… verwitwet           0
5 Rheine       86 weiblich     160    60   Abitur  Post… verwitwet           1
6 Rheine       84 weiblich     160    60   Haupts… Rent… verwitwet           0
7 Rheine       32 divers       186    92   Hochsc… Sozi… verheiratet         0
```

Innerhalb der Gruppierungen kann auch gefiltert werden.

```
pf8 %>%
  # gruppiere nach Standort
  group_by(Geschlecht) %>%
  # filtere "divers"
  dplyr::filter(Geschlecht != "divers") %>%
```

```
  # filtere "Internet"
  dplyr::filter(Standort != "Internet") %>%
  # sortiere nach "Gewicht"
  arrange(desc(Gewicht)) %>%
  #  zeige die ersten 2 Fälle pro Gruppe
  slice(1,2)
```

```
# A tibble: 4 x 16
# Groups:   Geschlecht [2]
  Standort Alter Geschlecht Größe Gewicht Bildung  Beruf Familienstand Kinder
  <fct>    <int> <fct>      <int>   <dbl> <fct>    <fct> <fct>          <int>
1 Münster     22 männlich     206     130 Fachabi… Kauf… ledig              0
2 Bahn        31 männlich     184     130 Fachabi… Scha… Partnerschaft      0
3 Münster     40 weiblich     171     122 Hochsch… Pres… Partnerschaft      0
4 Rheine      45 weiblich     168     120 mittler… kauf… geschieden         1
```

```
pf8 %>%
  group_by(Standort) %>%
  dplyr::filter(Familienstand == "ledig" & Geschlecht == "männlich") %>%
  arrange(desc(Alter)) %>%
  head()
```

```
# A tibble: 6 x 16
# Groups:   Standort [2]
  Standort Alter Geschlecht Größe Gewicht Bildung  Beruf Familienstand Kinder
  <fct>    <int> <fct>      <int>   <dbl> <fct>    <fct> <fct>          <int>
1 Rheine      81 männlich     179      79 Hauptsc… "Han… ledig              0
2 Rheine      74 männlich     175      84 mittler… "Koc… ledig              0
3 Münster     72 männlich     175      86 mittler… "Sic… ledig              0
4 Münster     69 männlich     179      89 mittler… "Ren… ledig              0
5 Münster     61 männlich     182      90 mittler… "Bea… ledig              0
6 Rheine      59 männlich     175      70 Abitur   ""    ledig              1
```

1.6.8.3 Variablen hinzufügen

Variablen können mit der Funktion `bind_cols()` hinzugefügt werden. Dabei muss die Länge der enthaltenen Werte mit der Länge des `tibble` übereinstimmen, ansonsten wiederholt die Wertereihe so lange, bis sie mit der Länge des `tibble` übereinstimmt.

```
pf8 %>%
  # Fügt eine Variable "test" hinzu, in der alle Werte TRUE sind.
  bind_cols(test = TRUE) %>%
    glimpse()
```

```
Rows: 731
Columns: 17
$ Standort       <fct> Münster, Münster, Münster, Münster, Münster, Münster, Münster…
$ Alter          <int> 18, 67, 60, 61, 24, 21, 59, 56, 82, 52, 79, 22, 19, 18, 17, 2…
$ Geschlecht     <fct> weiblich, weiblich, weiblich, männlich, männlich, weiblich, w…
$ Größe          <int> 172, 165, 175, 182, 173, 177, 168, 156, 184, 166, 161, 206, 1…
$ Gewicht        <dbl> 69, 67, NA, 90, 68, 60, 80, 60, NA, 60, 66, 130, 52, 68, 48, …
$ Bildung        <fct> Abitur, mittlere Reife, Hochschule, mittlere Reife, Abitur, A…
$ Beruf          <fct> Inspektor*in, Rentner*in, Ergotherapeut*in, Beamter*in, Stude…
$ Familienstand  <fct> Partnerschaft, geschieden, Partnerschaft, ledig, ledig, Partn…
$ Kinder         <int> 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Wohnort        <fct> ländlich, ländlich, städtisch, ländlich, städtisch, städtisch…
$ Rauchen        <fct> nein, nein, nein, nein, nein, ja, nein, ja, nein, nein, nein,…
$ SportHäufig    <dbl> NA, 2.0, 2.0, 4.0, 4.0, 1.0, 2.0, 1.0, 1.0, 2.0, NA, 4.0, 5.0…
$ SportMinuten   <dbl> NA, 60, 45, 120, 60, 60, 45, 90, NA, 45, NA, 90, 20, 120, 120…
```

```
$ SportWie       <fct> Allein, Gruppe, beides, Allein, Allein, Gruppe, Gruppe, beide…
$ SportWarum     <fct> Arbeitsweg, Vorbeugung, Vorbeugung, Fitness, Gewichtsreduktio…
$ LebenZufrieden <dbl> 5, 7, 7, 2, 9, 8, 5, 8, 10, 8, 8, 8, 7, 8, 7, 8, 7, 7, 8, 7, …
$ test           <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, T…
```

Unsere neue Variable `test` sehen wir ganz unten. Alle Werte stehen auf `TRUE`.

1.6.8.4 Fälle hinzufügen

Neue Fälle können mit der Funktion `bind_rows()` hinzugefügt werden. Dies klappt nur, wenn die neuen Fälle als `tibble` vorliegen, das über die selben Variablen verfügt wie der Originaldatensatz. Die Reihenfolge der Spalten ist egal, da über die Spalten*namen* gematcht wird.

 Achtung, Anfängerfehler! 

Wichtig ist jedoch, dass innerhalb der Variablen auch der selbe Daten`typ` (numerisch, faktor, logisch) vorliegt, da ansonten die Datentypen auf den kleinsten gemeinsamen Nenner (`character`) zurückfallen. Wir machen es hier einmal falsch:

```
# erzeuge neuen Fall
# ohne Angabe des Datentyps
neu <- tibble("Internet", 44, "weiblich", 166, 70, NA, "Pilot", "verheiratet", 0, "ländlich","ja")
# kopiere die Spaltennamen
colnames(neu) <- colnames(pf8)

# füge zum Datensazu hinzu
# Achtung, zerschießt die Datentypen des tibble!
pf8 %>%
   bind_rows( neu ) %>%
    tail(pf8)
```

```
    Standort Alter Geschlecht Größe Gewicht Bildung Beruf Familienstand Kinder
727 Internet    19   weiblich   158      58    <NA>              ledig      0
728 Internet    37   weiblich   160      59    <NA>        verheiratet      1
729 Internet    56   männlich   185      79    <NA>      Partnerschaft      0
730 Internet    20   weiblich   168      58    <NA>              ledig      0
731 Internet    40   männlich   183      78    <NA>        verheiratet      2
732 Internet    44   weiblich   166      70    <NA> Pilot  verheiratet      0
      Wohnort Rauchen SportHäufig SportMinuten SportWie SportWarum LebenZufrieden
727 städtisch    nein         7.0           10   Allein       <NA>             7
728  ländlich    nein         3.0           60   Allein       <NA>             9
729  ländlich    nein         2.0           90   Gruppe       <NA>             9
730 städtisch    nein         1.0           60   Allein       <NA>             8
731 städtisch      ja         2.5          120   Gruppe       <NA>            10
732  ländlich      ja          NA           NA     <NA>       <NA>            NA
```

Unsere Zeile ist ganz unten zu sehen. Die fehlenden Werte wurden mit `NA` aufgefüllt. Alles scheint gut gelaufen zu sein. Wie ein Blick mit `glimpse()` jedoch zeigt, haben wir die Datentypen unseres `tibbles` zerschossen, da unsere hinzugefügte Zeile nicht die korrekten Daten*typen* beinhaltete.

```
pf8 %>%
   bind_rows( neu ) %>%
    glimpse()
```

```
Rows: 732
Columns: 16
$ Standort       <chr> "Münster", "Münster", "Münster", "Münster", "Münster", "Münst…
$ Alter          <dbl> 18, 67, 60, 61, 24, 21, 59, 56, 82, 52, 79, 22, 19, 18, 17, 2…
$ Geschlecht     <chr> "weiblich", "weiblich", "weiblich", "männlich", "männlich", "…
$ Größe          <dbl> 172, 165, 175, 182, 173, 177, 168, 156, 184, 166, 161, 206, 1…
$ Gewicht        <dbl> 69, 67, NA, 90, 68, 60, 80, 60, NA, 60, 66, 130, 52, 68, 48, …
$ Bildung        <fct> Abitur, mittlere Reife, Hochschule, mittlere Reife, Abitur, A…
$ Beruf          <chr> "Inspektor*in", "Rentner*in", "Ergotherapeut*in", "Beamter*in…
$ Familienstand  <chr> "Partnerschaft", "geschieden", "Partnerschaft", "ledig", "led…
$ Kinder         <dbl> 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Wohnort        <chr> "ländlich", "ländlich", "städtisch", "ländlich", "städtisch",…
$ Rauchen        <chr> "nein", "nein", "nein", "nein", "nein", "ja", "nein", "ja", "…
$ SportHäufig    <dbl> NA, 2.0, 2.0, 4.0, 4.0, 1.0, 2.0, 1.0, 1.0, 2.0, NA, 4.0, 5.0…
$ SportMinuten   <dbl> NA, 60, 45, 120, 60, 60, 45, 90, NA, 45, NA, 90, 20, 120, 120…
$ SportWie       <fct> Allein, Gruppe, beides, Allein, Allein, Gruppe, Gruppe, beide…
$ SportWarum     <fct> Arbeitsweg, Vorbeugung, Vorbeugung, Fitness, Gewichtsreduktio…
$ LebenZufrieden <dbl> 5, 7, 7, 2, 9, 8, 5, 8, 10, 8, 8, 8, 7, 8, 7, 8, 7, 7, 8, 7, …
```

Wie Sie sehen, sind unsere ehemaligen Faktoren `Standort` und `Geschlecht` auf `<chr>` zurückgefallen.

Ein falscher `bind_rows()`-Befehl kann Ihnen also das gesamte tibble "zerschießen"!

Fügen wir nun eine **korrekte** neue Zeile hinzu

```
# erzeuge korrekte neue Zeile
neu <- tibble(factor("Internet"), 44, factor("weiblich"), 166, 70, NA, factor("Pilot"), factor("verheiratet"),
0, factor("ländlich"), factor("ja"))
# Spaltennamen übernehmen
colnames(neu) <- colnames(pf8)

pf8 %>%
   bind_rows( neu ) %>%
    glimpse()
```

```
Rows: 732
Columns: 16
$ Standort       <fct> Münster, Münster, Münster, Münster, Münster, Münster, Münster…
$ Alter          <dbl> 18, 67, 60, 61, 24, 21, 59, 56, 82, 52, 79, 22, 19, 18, 17, 2…
$ Geschlecht     <fct> weiblich, weiblich, weiblich, männlich, männlich, weiblich, w…
$ Größe          <dbl> 172, 165, 175, 182, 173, 177, 168, 156, 184, 166, 161, 206, 1…
$ Gewicht        <dbl> 69, 67, NA, 90, 68, 60, 80, 60, NA, 60, 66, 130, 52, 68, 48, …
$ Bildung        <fct> Abitur, mittlere Reife, Hochschule, mittlere Reife, Abitur, A…
$ Beruf          <fct> Inspektor*in, Rentner*in, Ergotherapeut*in, Beamter*in, Stude…
$ Familienstand  <fct> Partnerschaft, geschieden, Partnerschaft, ledig, ledig, Partn…
$ Kinder         <dbl> 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
$ Wohnort        <fct> ländlich, ländlich, städtisch, ländlich, städtisch, städtisch…
$ Rauchen        <fct> nein, nein, nein, nein, nein, ja, nein, ja, nein, nein, nein,…
$ SportHäufig    <dbl> NA, 2.0, 2.0, 4.0, 4.0, 1.0, 2.0, 1.0, 1.0, 2.0, NA, 4.0, 5.0…
$ SportMinuten   <dbl> NA, 60, 45, 120, 60, 60, 45, 90, NA, 45, NA, 90, 20, 120, 120…
$ SportWie       <fct> Allein, Gruppe, beides, Allein, Allein, Gruppe, Gruppe, beide…
$ SportWarum     <fct> Arbeitsweg, Vorbeugung, Vorbeugung, Fitness, Gewichtsreduktio…
$ LebenZufrieden <dbl> 5, 7, 7, 2, 9, 8, 5, 8, 10, 8, 8, 8, 7, 8, 7, 8, 7, 7, 8, 7, …
```

Die Datentypen sind erhalten geblieben.

1.6.8.5 Datensätze verbinden

Um Datenframes oder Tibbles miteinander zu verbinden, stehen neben `bind_rows()` und `bind_cols()` die `join`-Funktionen zur Verfügung. Sie kommen dann zur Anwendung, wenn die Zusammenführung anhand von "Übereinstimmungen" erfolgen soll.

Stellen wir uns die beiden Datenframes `X` und `Y` vor.

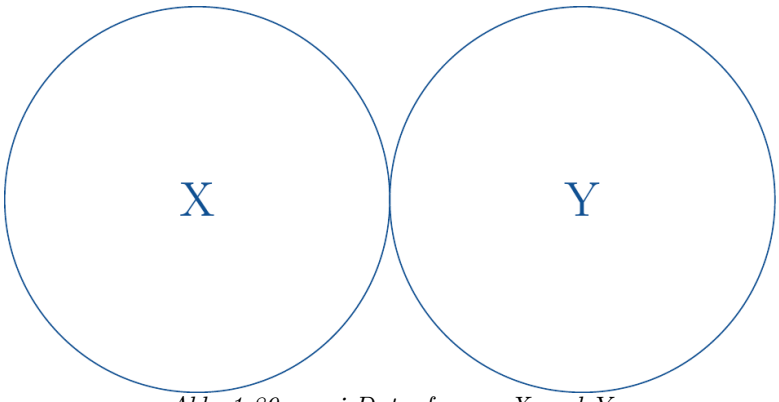


Abb. 1.89: zwei Datenframes X und Y

Lassen Sie uns nun zwei konkrete Testdatensätze für `X` und `Y` erzeugen.
Das Tibble `patient` soll Daten über 10 Testpatienten enthalten.

```
patient <- tribble(
  ~Patient, ~Alter, ~Geschlecht, ~Diagnose,
    "AB25",     51,         "w",   "00119",
    "JA26",     61,         "w",   "00030",
    "BG81",     44,         "m",   "00011",
    "ZT42",     50,         "m",   "00199",
    "AL63",     80,         "w",   "00223",
    "XV96",     88,         "w",   "00016",
    "QR49",     66,         "w",   "00027",
    "FE31",     77,         "w",   "00027",
    "PP23",     64,         "m",   "00098",
    "WU53",     86,         "w",   "00016"
  )
```

In `nanda` sollen 10 NANDA-Pflegediagnosen enthalten sein.

```
nanda <- tribble(
  ~Code,                               ~Diagnosetitel, ~Evidenzlevel,
"00110", "Selbstversorgungsdefizit Toilettenbenutzung",          2.1,
"00108",       "Selbstversorgungsdefizit Körperpflege",          2.1,
"00027",             "Defizitäres Flüssigkeitsvolumen",          2.1,
"00011",                                 "Obstipation",          3.1,
"00030",               "Beeinträchtigter Gasaustausch",          3.3,
"00223",                        "Ineffektive Beziehung",          2.1,
"00016",            "Beeinträchtigte Harnausscheidung",          3.1,
"00119",         "Chronisch geringes Selbstwertgefühl",          3.2,
"00199",              "Ineffektive Aktivitätenplanung",          2.1,
"00095",                                 "Schlafstörung",          3.3
)
```

Schauen wir uns die beiden Tibbles noch einmal an:

```
head(patient)
```

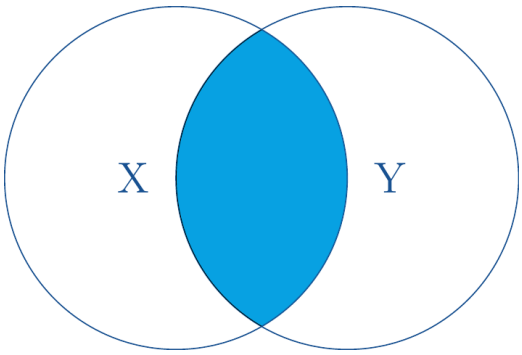
```
  # A tibble: 6 × 4
  Patient Alter Geschlecht Diagnose
  <chr>   <dbl> <chr>      <chr>
1 AB25       51 w          00119
2 JA26       61 w          00030
```

```
3 BG81       44 m          00011
4 ZT42       50 m          00199
5 AL63       80 w          00223
6 XV96       88 w          00016
```

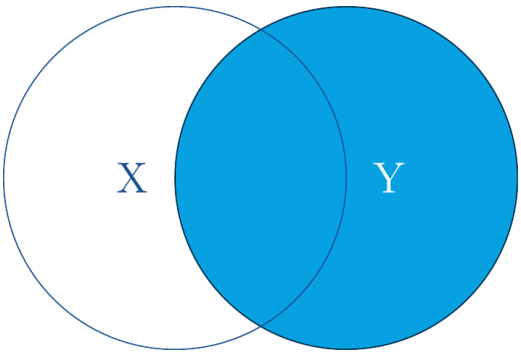
```
1 head(nanda)
```

```
  # A tibble: 6 × 3
  Code  Diagnosetitel                                  Evidenzlevel
  <chr> <chr>                                                 <dbl>
1 00110 Selbstversorgungsdefizit Toilettenbenutzung           2.1
2 00108 Selbstversorgungsdefizit Körperpflege                 2.1
3 00027 Defizitäres Flüssigkeitsvolumen                       2.1
4 00011 Obstipation                                           3.1
5 00030 Beeinträchtigter Gasaustausch                         3.3
6 00223 Ineffektive Beziehung                                 2.1
```

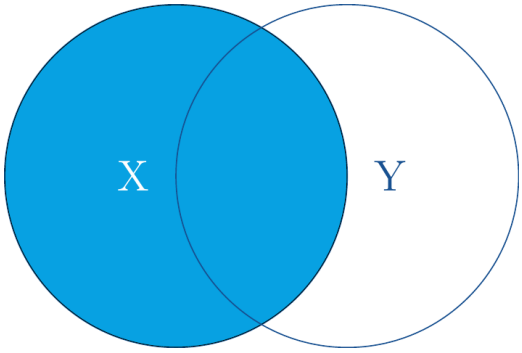
Es ist erkennbar, dass `patient$Diagnose` und `nanda$Code` Daten über das selbe Item beinhalten (in vielen Projekten wird eine eindeutige `ID` vergeben, über die später referenziert werden kann). Diese Spalten können wir also nutzen, um beim Verschmelzen die jeweiligen Datenreihen der beiden Tibbles einander zuordnen zu können. Dies haben wir schon bei der `merge()`-Funktion so gemacht, siehe Abschnitt 1.2.3.5.8 auf Seite 46. Im Tidyverse muss diese Information mittels `join_by( X$Spalte == Y$Spalte)` an die `join`-Funktionen übergeben werden. In userem Beispiel würde die Übereinstimmung wie folgt festgelegt: `join_by(Diagnose==Code)`. Die *eigentliche* Zusammenführung kann dann auf verschiedene Arten erfolgen:



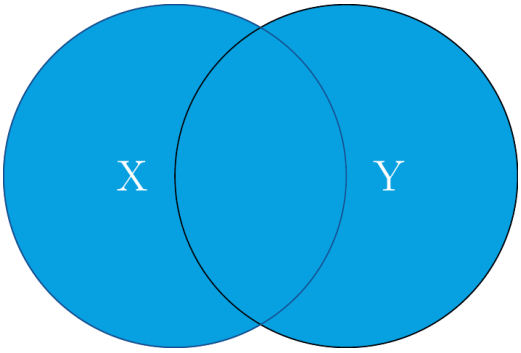
(a) *`inner_join()` übernimmt nur übereinstimmende Zeilen*



(b) *`right_join()` behält alle Y und ergänz passende X*



(c) *`left_join()` behält alle X und ergänz passende Y*



(d) *`full_join()` behält alle X und Y*

- `inner_join()`
- `left_join()`
- `right_join()`
- `full_join()`

1.6.8.5.1 `inner_join()`

Die Funktion `inner_join()` verschmilzt nur solche Datenreihen aus `X` und `Y`, die in den Übereinstimmungsspalten gleiche Werte haben.
Welcher Datensatz für `X` oder `Y` steht entscheidet die Reihenfolge innerhalb des Funktionsaufrufs.
Im folgenden Beispiel setzen wir `patient` als `X` und `nanda` als `Y`.

```
# behalte nur solche, die matchen
inner_join(patient, nanda, join_by(Diagnose == Code))
```

```
  # A tibble: 9 × 6
  Patient Alter Geschlecht Diagnose Diagnosetitel                       Evidenzlevel
  <chr>   <dbl> <chr>      <chr>    <chr>                                     <dbl>
1 AB25       51 w          00119    Chronisch geringes Selbstwertgefühl         3.2
2 JA26       61 w          00030    Beeinträchtigter Gasaustausch               3.3
3 BG81       44 m          00011    Obstipation                                 3.1
4 ZT42       50 m          00199    Ineffektive Aktivitätenplanung              2.1
5 AL63       80 w          00223    Ineffektive Beziehung                       2.1
6 XV96       88 w          00016    Beeinträchtigte Harnausscheidung            3.1
7 QR49       66 w          00027    Defizitäres Flüssigkeitsvolumen             2.1
8 FE31       77 w          00027    Defizitäres Flüssigkeitsvolumen             2.1
9 WU53       86 w          00016    Beeinträchtigte Harnausscheidung            3.1
```

Das so erzeugte Tibble enthält nur 9 Patientendaten, da der Diagnosecode von Patient `PP23` (`00098`) nicht in der `nanda`-Liste enthalten ist. Weil aber nur vollständige Matches beibehalten werden, fehlt Patient `PP23` in der Ausgabe.
Das Merkmal von `inner_join()` ist also, dass nur vollständige Datenreihen erhalten bleiben.

1.6.8.5.2 `left_join()`

Die Funktion `left_join()` behält alle Datenreihen von `X`, und fügt nur solche `Y` hinzu, die über die Übereinstimmungsspalte matchen.
Auch in diesem Beispiel setzen wir `patient` als `X` und `nanda` als `Y`.

```
# behalte alle patienten (x), und packe nur
# matchende y (NANDA) hinzu
left_join(patient, nanda, join_by(Diagnose == Code))
```

```
  # A tibble: 10 × 6
   Patient Alter Geschlecht Diagnose Diagnosetitel                      Evidenzlevel
   <chr>   <dbl> <chr>      <chr>    <chr>                                    <dbl>
1  AB25       51 w          00119    Chronisch geringes Selbstwertgefühl        3.2
2  JA26       61 w          00030    Beeinträchtigter Gasaustausch              3.3
3  BG81       44 m          00011    Obstipation                                3.1
4  ZT42       50 m          00199    Ineffektive Aktivitätenplanung             2.1
5  AL63       80 w          00223    Ineffektive Beziehung                      2.1
6  XV96       88 w          00016    Beeinträchtigte Harnausscheidung           3.1
7  QR49       66 w          00027    Defizitäres Flüssigkeitsvolumen            2.1
```

```
8  FE31       77 w          00027    Defizitäres Flüssigkeitsvolumen             2.1
9  PP23       64 m          00098    NA                                          NA
10 WU53       86 w          00016    Beeinträchtigte Harnausscheidung            3.1
```

Alle Patienten (`X`) sind erhalten geblieben. Da der Diagnosecode von Patient `PP23` (`00098`) nicht in der `nanda`-Liste (`Y`) enthalten ist, wurden `NA`s ergänzt.

1.6.8.5.3 `right_join()`

Die Funktion `right_join()` ist das spiegelverkehrte Pendant. Sie behält alle Datenreihen von `Y`, und fügt nur solche `X` hinzu, die über die Übereinstimmungsspalte matchen.

```
1  # behalte alle NANDA (y) und packe nur
2  # matchende Patienten (x) hinzu
3  # erzeugt doppelte Y, wenn x mehrfach y enthält
4  right_join(patient, nanda, join_by(Diagnose == Code))
```

```
  # A tibble: 12 × 6
   Patient Alter Geschlecht Diagnose Diagnosetitel                              Evidenzlevel
   <chr>   <dbl> <chr>      <chr>    <chr>                                           <dbl>
1  AB25       51 w          00119    Chronisch geringes Selbstwertgefühl               3.2
2  JA26       61 w          00030    Beeinträchtigter Gasaustausch                     3.3
3  BG81       44 m          00011    Obstipation                                       3.1
4  ZT42       50 m          00199    Ineffektive Aktivitätenplanung                    2.1
5  AL63       80 w          00223    Ineffektive Beziehung                             2.1
6  XV96       88 w          00016    Beeinträchtigte Harnausscheidung                  3.1
7  QR49       66 w          00027    Defizitäres Flüssigkeitsvolumen                   2.1
8  FE31       77 w          00027    Defizitäres Flüssigkeitsvolumen                   2.1
9  WU53       86 w          00016    Beeinträchtigte Harnausscheidung                  3.1
10 NA         NA NA         00110    Selbstversorgungsdefizit Toilettenbenutzung       2.1
11 NA         NA NA         00108    Selbstversorgungsdefizit Körperpflege             2.1
12 NA         NA NA         00095    Schlafstörung                                     3.3
```

Das neue Tibble ist 12 Reihen lang, da zwei Pflegediagnosen (`Y`) jeweils 2mal im Datensatz `patient` (`X`) vorkommen.
Das Ergebnis von `right_join()` lässt sich prinzipiell auch erreichen, indem innerhalb von `left_join()` einfach die Werte umgedreht werden. Die Reihenfolge der Spalten und Zeilen im erzeugten Tibble ist dann aber anders, obwohl insgesamt die selben Daten enthalten sind.

```
1  # geht auch mit left_join(),
2  # indem man einfach X und Y vertauscht
3  # join_by() muss ebenfalls vertauscht werden
4  left_join(nanda, patient, join_by(Code == Diagnose))
```

```
  # A tibble: 12 × 6
  Code  Diagnosetitel                              Evidenzlevel Patient Alter Geschlecht
  <chr> <chr>                                            <dbl> <chr>   <dbl> <chr>
1 00110 Selbstversorgungsdefizit Toilettenbenutzung        2.1 NA         NA  NA
2 00108 Selbstversorgungsdefizit Körperpflege              2.1 NA         NA  NA
3 00027 Defizitäres Flüssigkeitsvolumen                    2.1 QR49       66  w
4 00027 Defizitäres Flüssigkeitsvolumen                    2.1 FE31       77  w
5 00011 Obstipation                                        3.1 BG81       44  m
6 00030 Beeinträchtigter Gasaustausch                      3.3 JA26       61  w
7 00223 Ineffektive Beziehung                              2.1 AL63       80  w
8 00016 Beeinträchtigte Harnausscheidung                   3.1 XV96       88  w
9 00016 Beeinträchtigte Harnausscheidung                   3.1 WU53       86  w
```

```
10 00119 Chronisch geringes Selbstwertgefühl                3.2 AB25       51  w
11 00199 Ineffektive Aktivitätenplanung                     2.1 ZT42       50  m
12 00095 Schlafstörung                                      3.3 NA         NA  NA
```

Das neue Tibble ist 12 Reihen lang, da zwei Pflegediagnosen jeweils 2mal im Datensatz `patient` vorkommen.
Dieses Verhalten kann über den Parameter `multiple` bestimmt werden. So lässt sich einstellen, ob aus `Y` alle (`all`), nur der erste (`first`) oder letzte (`last`), oder *irgendein* (`any`) passender Treffer übernommen werden soll.

```
# wenn mehrfaches Vorkommen, nimm den letzten
left_join(nanda, patient, join_by(Code == Diagnose),
          multiple="last")
```

```
   # A tibble: 10 × 6
   Code  Diagnosetitel                                Evidenzlevel Patient Alter Geschlecht
   <chr> <chr>                                              <dbl> <chr>   <dbl> <chr>
1  00110 Selbstversorgungsdefizit Toilettenbenutzung          2.1  NA         NA NA
2  00108 Selbstversorgungsdefizit Körperpflege                2.1  NA         NA NA
3  00027 Defizitäres Flüssigkeitsvolumen                      2.1  FE31       77 w
4  00011 Obstipation                                          3.1  BG81       44 m
5  00030 Beeinträchtigter Gasaustausch                        3.3  JA26       61 w
6  00223 Ineffektive Beziehung                                2.1  AL63       80 w
7  00016 Beeinträchtigte Harnausscheidung                     3.1  WU53       86 w
8  00119 Chronisch geringes Selbstwertgefühl                  3.2  AB25       51 w
9  00199 Ineffektive Aktivitätenplanung                       2.1  ZT42       50 m
10 00095 Schlafstörung                                        3.3  NA         NA NA
```

Ausgangspunkt für den Parameter `multiple` ist immer Datensatz `X`. Im letzten Beispiel wurde `nanda` als `X` angegeben. Da jeder Diagnosecode in `nanda` genau einmal vorkommt, ist das neue Tibble 10 Reihen lang. Sollte in `Y` (`patient`) die Diagnose mehrfach vorkommen, wird (im Beispiel oben) die letzte vorkommende Datenreihen aus `Y` verwendet.

Kehren wir zum ursprünglichen `right_join()`-Aufruf zurück und übergeben den Parameter...

```
right_join(patient, nanda, join_by(Diagnose == Code),
           multiple="last")
```

```
   # A tibble: 12 × 6
   Patient Alter Geschlecht Diagnose Diagnosetitel                                Evidenzlevel
   <chr>   <dbl> <chr>      <chr>    <chr>                                             <dbl>
1  AB25       51  w          00119    Chronisch geringes Selbstwertgefühl                3.2
2  JA26       61  w          00030    Beeinträchtigter Gasaustausch                      3.3
3  BG81       44  m          00011    Obstipation                                        3.1
4  ZT42       50  m          00199    Ineffektive Aktivitätenplanung                     2.1
5  AL63       80  w          00223    Ineffektive Beziehung                              2.1
6  XV96       88  w          00016    Beeinträchtigte Harnausscheidung                   3.1
7  QR49       66  w          00027    Defizitäres Flüssigkeitsvolumen                    2.1
8  FE31       77  w          00027    Defizitäres Flüssigkeitsvolumen                    2.1
9  WU53       86  w          00016    Beeinträchtigte Harnausscheidung                   3.1
10 NA         NA  NA         00110    Selbstversorgungsdefizit Toilettenbenutzung        2.1
11 NA         NA  NA         00108    Selbstversorgungsdefizit Körperpflege              2.1
12 NA         NA  NA         00095    Schlafstörung                                      3.3
```

... so ist das Tibble wieder 12 Reihen lang, denn Ausgangspunkt für `multiple` ist **immer** Datensatz `X` (auch bei `right_join()`). Im Funktionsaufruf oben steht `patient` als `X`. Daher

schaut die Funktion für jede Reihe in X, ob ein oder mehrere passende Einträge in `Y` vorhanden sind. Da in `Y` (`nanda`) jede Diagnose nur einmal definiert wird, kommen keine multiplen Zeilen vor. Der Parameter `multiple` findet daher keine Anwendung.
Nun will `right_join()` aber alle `Y` beibehalten und mit allen passenden `X` (`patient`) verschmelzen. Da bei zwei Diagnosen in `Y` jeweils zwei Patienten aus `X` matchen, ist die Ausgabe 12 Zeilen lang.

1.6.8.5.4 `full_join()`

Beim `full_join()` werden alle Daten von `X` und `Y` beibehalten.

```
# behält alle x und alle y
full_join(patient, nanda, join_by(Diagnose == Code))
```

```
   # A tibble: 13 × 6
   Patient Alter Geschlecht Diagnose Diagnosetitel                                Evidenzlevel
   <chr>   <dbl> <chr>      <chr>    <chr>                                             <dbl>
1  AB25       51 w          00119    Chronisch geringes Selbstwertgefühl                 3.2
2  JA26       61 w          00030    Beeinträchtigter Gasaustausch                       3.3
3  BG81       44 m          00011    Obstipation                                         3.1
4  ZT42       50 m          00199    Ineffektive Aktivitätenplanung                      2.1
5  AL63       80 w          00223    Ineffektive Beziehung                               2.1
6  XV96       88 w          00016    Beeinträchtigte Harnausscheidung                    3.1
7  QR49       66 w          00027    Defizitäres Flüssigkeitsvolumen                     2.1
8  FE31       77 w          00027    Defizitäres Flüssigkeitsvolumen                     2.1
9  PP23       64 m          00098    NA                                                  NA
10 WU53       86 w          00016    Beeinträchtigte Harnausscheidung                    3.1
11 NA         NA NA         00110    Selbstversorgungsdefizit Toilettenbenutzung         2.1
12 NA         NA NA         00108    Selbstversorgungsdefizit Körperpflege               2.1
13 NA         NA NA         00095    Schlafstörung                                       3.3
```

1.6.8.6 Variablen auswählen

Mit der Funktion `select()` können Variablen (Spalten) des `tibble` ausgewählt werden.

```
# wähle Variablen "Alter", "Größe" und "Gewicht"
# aus Datensatz pf8
 pf8 %>%
   select(Alter,Größe,Gewicht) %>%
   head(.)
```

```
   Alter Größe Gewicht
11    18   172      69
12    67   165      67
13    60   175      NA
14    61   182      90
15    24   173      68
16    21   177      60
```

In Kombination mit der Funktion `everything()` kann die Reihenfolge der Variablen im Datensatz geändert werden. Die Funktion `everything()` hängt alle weiteren Variablen an unsere Auswahl an. Angenommen, wir möchten `Größe` und `Gewicht` "als erstes" sehen, und alle anderen Variablen danach, so lautet der Befehl:

```r
# zeige erst "Größe" und "Gewicht"
# und danach alles andere.
pf8 %>%
  select(Größe, Gewicht, everything()) %>%
  as_tibble()
```

```
# A tibble: 731 x 16
   Größe Gewicht Standort Alter Geschlecht Bildung   Beruf    Familienstand
   <int>   <dbl> <fct>    <int> <fct>      <fct>     <fct>    <fct>
 1   172      69 Münster     18 weiblich   Abitur    Inspekt… Partnerschaft
 2   165      67 Münster     67 weiblich   mittlere… Rentner… geschieden
 3   175      NA Münster     60 weiblich   Hochschu… Ergothe… Partnerschaft
 4   182      90 Münster     61 männlich   mittlere… Beamter… ledig
 5   173      68 Münster     24 männlich   Abitur    Student… ledig
 6   177      60 Münster     21 weiblich   Abitur    Student… Partnerschaft
 7   168      80 Münster     59 weiblich   mittlere… arbeitl… verheiratet
 8   156      60 Münster     56 weiblich   Ausbildu… Pflegeh… geschieden
 9   184      NA Münster     82 männlich   mittlere… Kaufman… verheiratet
10   166      60 Münster     52 weiblich   Fachabit… Kaufman… verheiratet
```

Mit der Funktion `pull()` werden die Spaltenwerte als Vektor wiedergegeben

```r
# gib Werte von "Alter" als Vektor
pf8 %>%
  pull(Alter) %>%
   head(10)
```

```
 [1] 18 67 60 61 24 21 59 56 82 52
```

1.6.8.7 Variablen erzeugen

Die Funktion `mutate()` erlaubt es, neue Variablen (Spalten) zu erzeugen, die durch Interaktion mit den anderen Variablen enstanden sind. So können wir im Datensatz `pf8` aus den Variablen `Größe` und `Gewicht` den Body-Maß-Index errechnen, und diese Werte als neue Variable (Spalte) speichern.

```r
# erzeuge BMI
pf8 %>%
  mutate(BMI = Gewicht/Größe*100) %>%
    head()
```

```
   Standort Alter Geschlecht Größe Gewicht        Bildung            Beruf
11  Münster    18   weiblich   172      69         Abitur     Inspektor*in
12  Münster    67   weiblich   165      67 mittlere Reife       Rentner*in
13  Münster    60   weiblich   175      NA     Hochschule Ergotherapeut*in
14  Münster    61   männlich   182      90 mittlere Reife       Beamter*in
15  Münster    24   männlich   173      68         Abitur       Student*in
16  Münster    21   weiblich   177      60         Abitur       Student*in
          SportWarum LebenZufrieden      BMI
11        Arbeitsweg              5 40.11628
12        Vorbeugung              7 40.60606
13        Vorbeugung              7       NA
14           Fitness              2 49.45055
15 Gewichtsreduktion              9 39.30636
16           Fitness              8 33.89831
```

Mit der Schwesterfunktion `transmute()` wird nur die neue Variable in einem eigenen `tibble` zurückgegeben

```
# erzeuge BMI und gib nur BMI zurück
pf8 %>%
  transmute(BMI = Gewicht/Größe*100) %>%
    head()
```

```
        BMI
11 40.11628
12 40.60606
13       NA
14 49.45055
15 39.30636
16 33.89831
```

1.6.8.8 statistische Berechnungen

Mit der Funktion `summarise()` können "zusammenfassenden Statistiken" berechnet werden. Sie fungiert als eine Art "Vermittlungsfunktion", um Statistikfunktionen (siehe Kapitel 2 ab Seite 211) in den Tidyverse-*Workflow* zu integrieren.

Klassische zusammenfassende Kennzahlen sind Mittelwert (über Funktion `mean()`), Median (über Funktion `median()`) und Standardabweichung (über Funktion `sd()`). Es funktioniert aber auch mit allen anderen Statistikfunktionen. Wir beschränken uns im Weiteren auf diese drei, alle anderen werden in Kapitel 2 vorgestellt.

Berechnen wir mit `mean()` den Mittelwert für `Alter`.

```
# Mittelwert von "Alter"
pf8 %>%
  summarise(Mittelwert = mean(Alter))
```

```
  Mittelwert
1         NA
```

Wir erhalten ein `NA` zurück, weil in der Variable `Alter` fehlende Werte enthalten sind. Fast alle statistischen Funktionen erwarten von uns, dass `NA`s weggefilter wurden. Nutzen wir die Macht der Pipe und ändern den Befehl in

```
# Mittelwert von "Alter"
pf8 %>%
  drop_na() %>%
    summarise(Mittelwert = mean(Alter))
```

```
  Mittelwert
1   38.25126
```

Mit der Funktion `group_by()` können Gruppierungen vorgenommen werden.

```
# Gruppiere nach Geschlecht
pf8 %>%
  group_by(Geschlecht) %>%
    drop_na() %>%
      summarise(Mittelwert = mean(Alter))
```

```
# A tibble: 3 x 2
  Geschlecht Mittelwert
  <fct>           <dbl>
1 männlich         37.3
2 weiblich         39.1
3 divers           32
```

Die Liste der "zusammenfassenden Statistiken" lässt sich beliebig erweitern.

```
pf8 %>%
  group_by(Geschlecht) %>%
     drop_na() %>%
       summarise(Mittelwert = mean(Alter),
                 Median = median(Alter),
                 Stdabw = sd(Alter))
```

```
# A tibble: 3 x 4
  Geschlecht Mittelwert Median Stdabw
  <fct>           <dbl>  <dbl>  <dbl>
1 männlich         37.3     29   17.5
2 weiblich         39.1     33   18.9
3 divers           32       32   NA
```

Auch, indem Gruppierungen komplexer definiert werden.

```
# mit mehr Gruppierungen
pf8 %>%
  group_by(Geschlecht, Standort) %>%
     drop_na() %>%
        summarise(Mittelwert = mean(Alter),
                  Median = median(Alter),
                  Stdabw = sd(Alter))
```

```
# A tibble: 9 x 5
# Groups:   Geschlecht [3]
  Geschlecht Standort  Mittelwert Median Stdabw
  <fct>      <fct>          <dbl>  <dbl>  <dbl>
1 männlich   Rheine          34.3   26     15.4
2 männlich   Münster         37.9   28     18.5
3 männlich   Bahn            39.9   36     19.8
4 männlich   Ladbergen       41.4   40     17.6
5 weiblich   Rheine          37.6   33     17.8
6 weiblich   Münster         38.8   27     20.0
7 weiblich   Bahn            41.3   37.5   19.4
8 weiblich   Ladbergen       43     40.5   18.5
9 divers     Rheine          32     32     NA
```

Die Funktion `count()` zählt die Häufigkeit der Variablenwerte. Dies ist nicht nur bei Faktoren hilfreich.

```
# mit mehr Gruppierungen
pf8 %>%
  count(Geschlecht)
```

```
  Geschlecht   n
1   männlich 287
2   weiblich 437
3     divers   1
4       <NA>   6
```

In Kombination mit `group_by()` lassen sich Ausgaben ähnlich der Kreuztabelle erstellen.

```
pf8 %>%
  drop_na() %>%
  group_by(Familienstand, Geschlecht) %>%
  count()
```

```
# A tibble: 13 x 3
# Groups:   Familienstand, Geschlecht [13]
   Familienstand Geschlecht     n
   <fct>         <fct>      <int>
 1 ledig         männlich      74
 2 ledig         weiblich      82
 3 Partnerschaft männlich      32
 4 Partnerschaft weiblich      40
 5 verheiratet   männlich      61
 6 verheiratet   weiblich      78
 7 verheiratet   divers         1
 8 geschieden    männlich       5
 9 geschieden    weiblich       7
10 verwitwet     männlich       4
11 verwitwet     weiblich       9
12 getrennt      männlich       3
13 getrennt      weiblich       2
```

Mit der Funktion `add_count()` werden die Häufigkeitswerte als eigene Variable im Datensatz gespeichert.

```
pf8 %>%
  # Speichere die Levelhäufigkeit als eigene Variable
  add_count(Geschlecht, name = "AnzahlGeschlecht") %>%
  # Zeige diese Variablen als erstes, dann den Rest
  select(Geschlecht, AnzahlGeschlecht, everything()) %>%
  # Ausgabe als tibble ist "schöner"
  as_tibble()
```

```
# A tibble: 731 x 17
   Geschlecht AnzahlGeschlecht Standort Alter Größe Gewicht Bildung
   <fct>                 <int> <fct>    <int> <int>   <dbl> <fct>
 1 weiblich                437 Münster     18   172      69 Abitur
 2 weiblich                437 Münster     67   165      67 mittlere Re…
 3 weiblich                437 Münster     60   175      NA Hochschule
 4 männlich                287 Münster     61   182      90 mittlere Re…
 5 männlich                287 Münster     24   173      68 Abitur
 6 weiblich                437 Münster     21   177      60 Abitur
 7 weiblich                437 Münster     59   168      80 mittlere Re…
 8 weiblich                437 Münster     56   156      60 Ausbildung
 9 männlich                287 Münster     82   184      NA mittlere Re…
10 weiblich                437 Münster     52   166      60 Fachabitur
```

Die Funktion `n()` fliefert die Anzahl der Fälle.

```
pf8 %>%
  group_by(Standort, Geschlecht) %>%
  drop_na() %>%
  summarise(xquer = mean(Alter),
            sd = sd(Alter),
            median = median(Alter),
            n = n())
```

```
# A tibble: 9 x 6
# Groups:   Standort [4]
  Standort  Geschlecht xquer    sd median     n
  <fct>     <fct>      <dbl> <dbl>  <dbl> <int>
1 Rheine    männlich    34.3  15.4   26      67
2 Rheine    weiblich    37.6  17.8   33      83
3 Rheine    divers      32    NA     32       1
4 Münster   männlich    37.9  18.5   28      62
5 Münster   weiblich    38.8  20.0   27      83
6 Bahn      männlich    39.9  19.8   36      29
7 Bahn      weiblich    41.3  19.4   37.5    38
8 Ladbergen männlich    41.4  17.6   40      21
9 Ladbergen weiblich    43    18.5   40.5    14
```

1.6.8.9 Umgang mit Faktoren

Mit dem Paket `forcats` stehen im Tidyverse viele hilfreiche Funktionen für den Umgang mit Faktoren zur Verfügung.

1.6.8.9.1 Levels umbenennen

Mit der Funktion `fct_recode()` können Faktorenlevels umbenannt werden.

```
pf8 %>%
  mutate(Geschlecht = fct_recode(Geschlecht,
                      m ="männlich",
                      w = "weiblich",
                      d = "divers")) %>%
  as_tibble()
```

```
# A tibble: 731 x 16
   Standort Alter Geschlecht Größe Gewicht Bildung Beruf Familienstand
   <fct>    <int> <fct>      <int>   <dbl> <fct>   <fct> <fct>
 1 Münster     18 w            172      69 Abitur  Insp… Partnerschaft
 2 Münster     67 w            165      67 mittle… Rent… geschieden
 3 Münster     60 w            175      NA Hochsc… Ergo… Partnerschaft
 4 Münster     61 m            182      90 mittle… Beam… ledig
 5 Münster     24 m            173      68 Abitur  Stud… ledig
 6 Münster     21 w            177      60 Abitur  Stud… Partnerschaft
 7 Münster     59 w            168      80 mittle… arbe… verheiratet
 8 Münster     56 w            156      60 Ausbil… Pfle… geschieden
 9 Münster     82 m            184      NA mittle… Kauf… verheiratet
10 Münster     52 w            166      60 Fachab… Kauf… verheiratet
```

Enthalten die neuen Levelnamen Sonderzeichen (wie das Leerzeichen), müssen sie in Anführungsstrichen gesetzt werden.

```
pf8 %>%
  mutate(Geschlecht = fct_recode(Geschlecht,
                      "ein Mann" ="männlich",
                      "eine Frau" = "weiblich",
                      "ein(e) Divers" = "divers")) %>%
  as_tibble()
```

```
# A tibble: 731 x 16
   Standort Alter Geschlecht Größe Gewicht Bildung Beruf Familienstand
   <fct>    <int> <fct>      <int>   <dbl> <fct>   <fct> <fct>
 1 Münster     18 eine Frau    172      69 Abitur  Insp… Partnerschaft
 2 Münster     67 eine Frau    165      67 mittle… Rent… geschieden
```

```
 3 Münster     60 eine Frau    175      NA Hochsc… Ergo… Partnerschaft
 4 Münster     61 ein Mann     182      90 mittle… Beam… ledig
 5 Münster     24 ein Mann     173      68 Abitur  Stud… ledig
 6 Münster     21 eine Frau    177      60 Abitur  Stud… Partnerschaft
 7 Münster     59 eine Frau    168      80 mittle… arbe… verheiratet
 8 Münster     56 eine Frau    156      60 Ausbil… Pfle… geschieden
 9 Münster     82 ein Mann     184      NA mittle… Kauf… verheiratet
10 Münster     52 eine Frau    166      60 Fachab… Kauf… verheiratet
```

1.6.8.9.2 Levelreihenfolge ändern

Bei kategorialen Daten existiert keine geordnete Reihe der Werte. Dennoch kann es hilfreich sein, kategoriale Levels in eine bestimmte Reihenfolge zu bringen. Über die Hausfunktion `factor()` können über den Parameter `levels` die Levelreihenfolgen von Hand geändert werden.

```
x <- factor(c("vielleicht", "ja", "nein"))
x
```

```
[1] vielleicht ja         nein
Levels: ja nein vielleicht
```

```
# ändere Levelreihenfolge
factor(x, levels=c("nein", "vielleicht", "ja"))
```

```
[1] vielleicht ja         nein
Levels: nein vielleicht ja
```

Für häufige Anwendungsfälle bietet `forcats` Funktionen, die uns diese Arbeit abnehmen.

Schauen wir uns im Datensatz `pf8` die Anlässe für Sport an. Das Skalenniveau ist nominal. Wenn wir die Daten plotten (zu `ggplot` siehe Kapitel 2.4), werden die Daten (die Diagrammsäulen) in der Reihenfolge der Levels angezeigt, und nicht in der Reihenfolge der Häufigkeiten.

```
pf8 %>%
  select(SportWarum) %>%
  drop_na() %>%
  # dies ist der Plotbefehl, den Sie jetzt noch nicht verstehen.
  # Lesen Sie das Kapitel zu "ggplot".
  ggplot(aes(x=SportWarum)) + geom_bar()
```

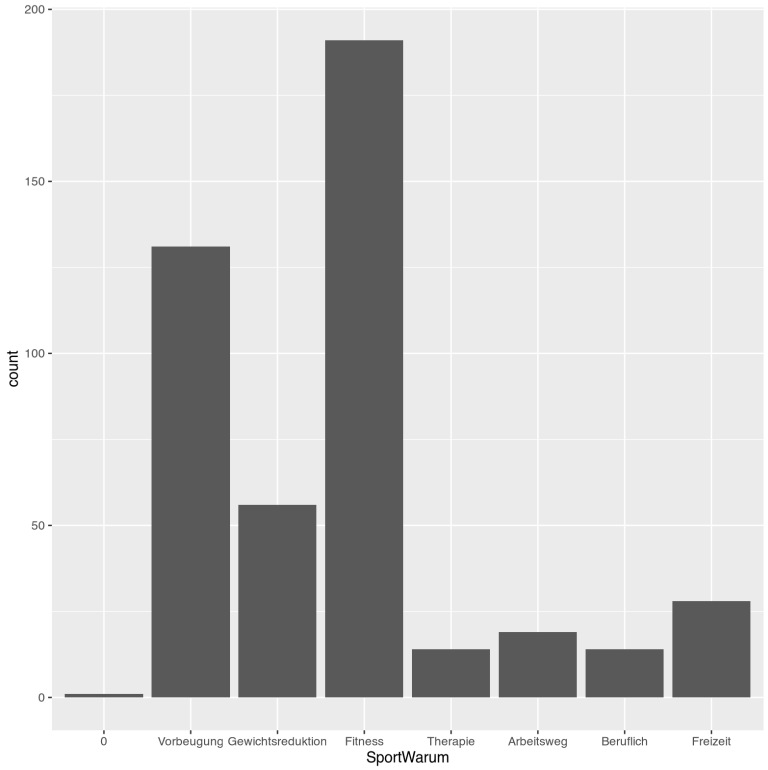


Abb. 1.91

Soll die Reihenfolgde der Levels so verändert werden, dass die mit der höchsten Ausprägung zuerst angezeigt werden, kann die Funktion `fct_infreq()` verwendet werden.

```
pf8 %>%
  drop_na() %>%
  # Sortiere nach Häufigkeiten
  mutate(SportWarum = fct_infreq(SportWarum)) %>%
  # plotten
  ggplot(aes(x=SportWarum)) + geom_bar()
```

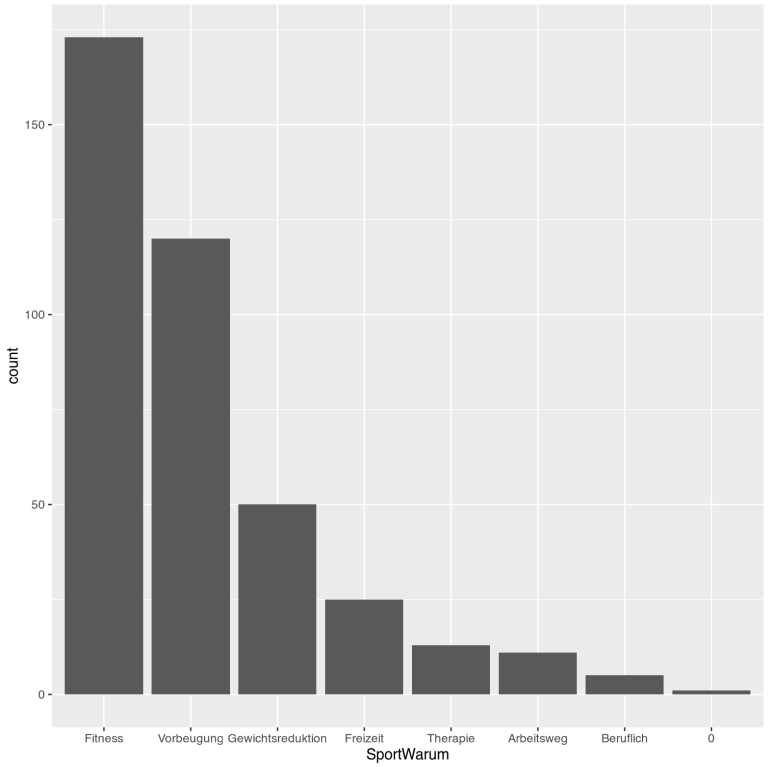


Abb. 1.92

Mit der Funktion `fct_rev()` wird die Levelreihenfolge umgekehrt.

```
pf8 %>%
  drop_na() %>%
  # Sortiere nach Häufigkeiten
  mutate(SportWarum = fct_infreq(SportWarum)) %>%
  # kehre Levelreihenfolge um
  mutate(SportWarum = fct_rev(SportWarum)) %>%
  # plotten
  ggplot(aes(x=SportWarum)) + geom_bar()
```

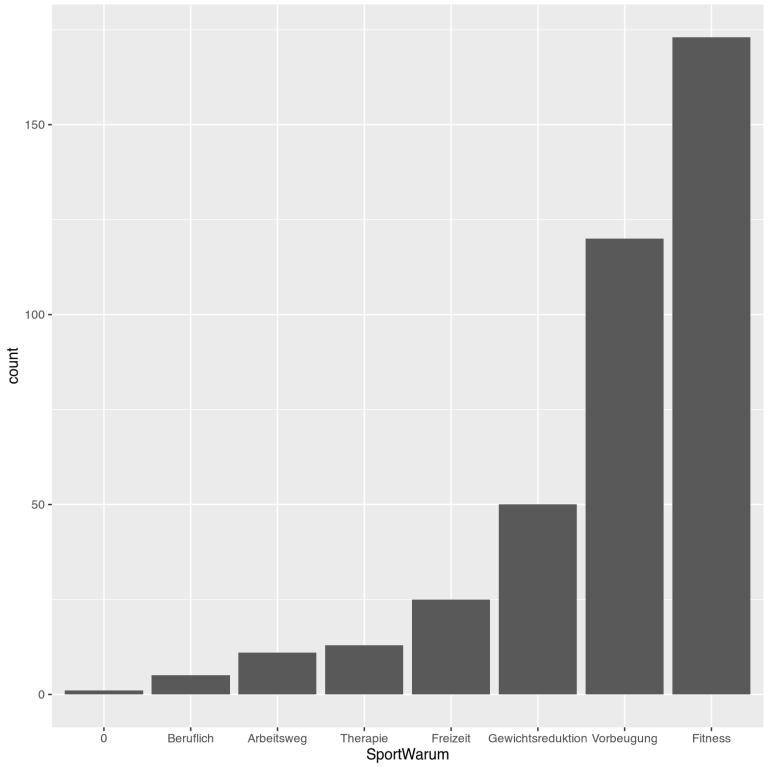


Abb. 1.93

Die Levelreihenfolge kann auch auf Grundlage einer anderen Variable erfolgen.
Erstellen wir uns hierfür ein Subset des `pf8`-Datensatzes, indem wir für jede Sport-Kategorie die Mittelwerte von `Alter` und `Gewicht` ermitteln.

```
pf8sub <- pf8 %>%
            group_by(SportWarum) %>%
            summarise(
              Alter = mean(Alter, na.rm=T),
              Gewicht = mean(Gewicht, na.rm=T),
              n = n()
            )
pf8sub
```

```
# A tibble: 9 x 4
  SportWarum        Alter Gewicht     n
  <fct>             <dbl>   <dbl> <int>
1 0                  19      65       1
2 Vorbeugung         42.9    78.1   131
3 Gewichtsreduktion  31.9    81.7    56
4 Fitness            37.4    73.2   191
5 Therapie           52.4    80.8    14
6 Arbeitsweg         34.6    82.1    19
7 Beruflich          43.9    80.6    14
8 Freizeit           36.7    74.0    28
9 NA                 34.1    73.9   277
```

Wenn wir die Daten plotten, folgen die Datenpunkte auch hier der Levelreihenfolge.

```
pf8sub %>%
  # plotten
  ggplot(aes(x=Alter, y=SportWarum)) + geom_point()
```

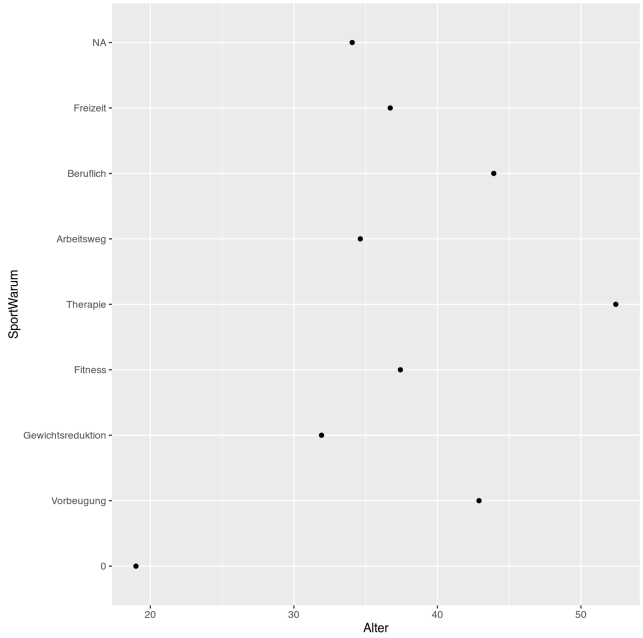


Abb. 1.94

Soll auch hier die Reihenfolge nach Mittelwerten erfolgen, kann die Funktion `fct_reorder()` verwendet werden.

```
1  pf8sub %>%
2    # überschreibe Variable "SportWarum"
3    # ordne die Levels von "SportWarum" nach "Alter"
4    mutate(SportWarum = fct_reorder(SportWarum, Alter)) %>%
5    # plotten
6    ggplot(aes(x=Alter, y=SportWarum)) + geom_point()
```

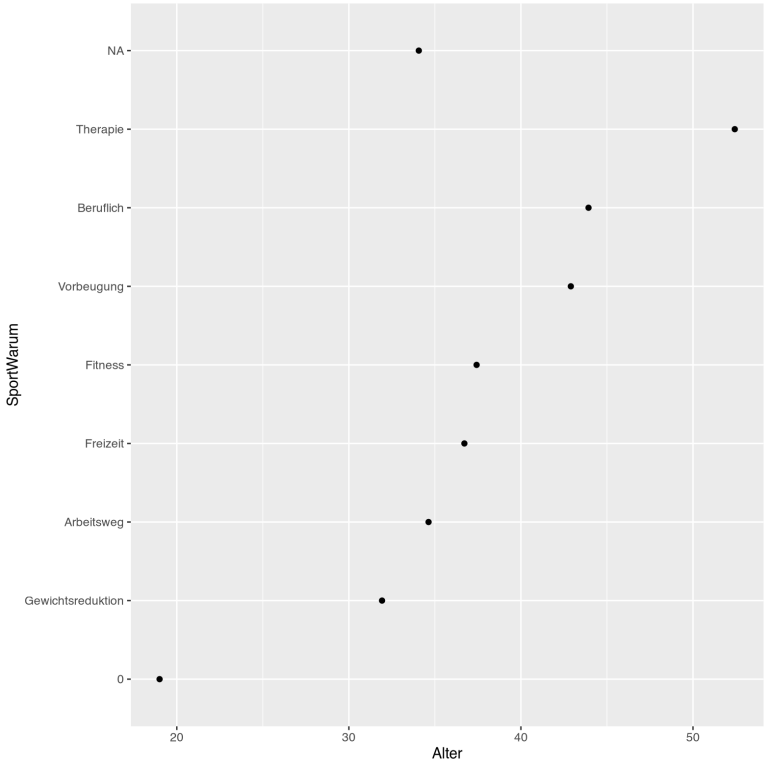


Abb. 1.95

Beachten Sie, dass ‘NA‘ weiterhin ganz oben angezeigt werden.

Mit der Funktion `fct_relevel()` können einzelne Levels “nach vorne” geholt werden. In diesem Beispiel möchten wir, dass der Grund `Freizeit` als erstes angezeigt wird.

```
pf8sub %>%
  # überschreibe Variable "SportWarum"
  # ordne die Levels von "SportWarum" nach "Alter"
  mutate(SportWarum = fct_reorder(SportWarum, Alter)) %>%
  # hole Level "Freizeit" nach vorne
  mutate(SportWarum = fct_relevel(SportWarum, "Freizeit")) %>%
  # plotten
  ggplot(aes(x=Alter, y=SportWarum)) + geom_point()
```

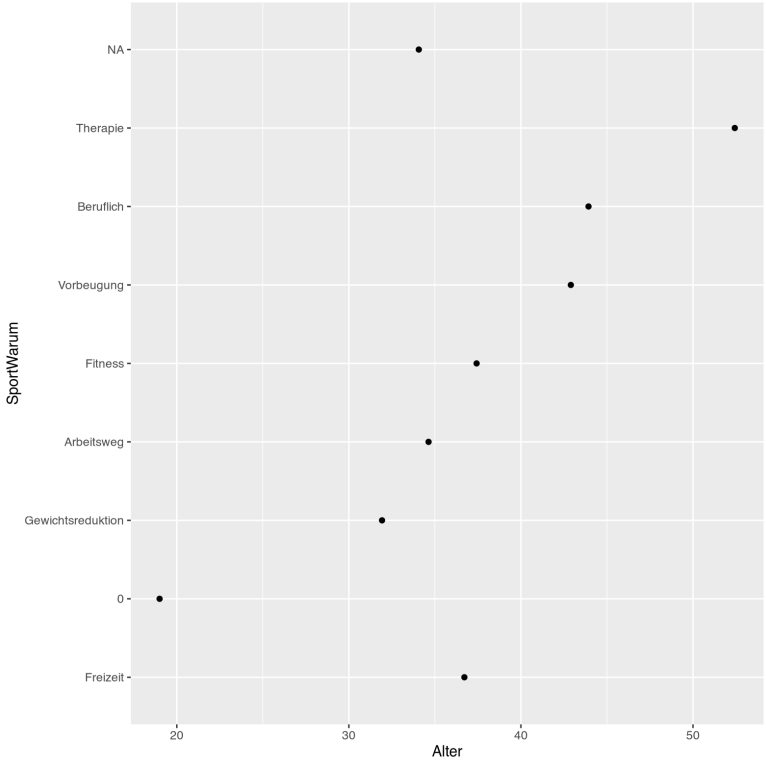


Abb. 1.96

1.6.8.9.3 Levels zusammenfassen

Schauen wir uns die Berufe im Datenatz `pf8` an.

```
pf8 %>%
  dplyr::select(Beruf) %>%
  dplyr::count(Beruf) %>%
  arrange(n) %>%
  as_tibble()
```

```
# A tibble: 104 x 2
   Beruf                  n
   <fct>              <int>
 1 " Produktionsleiter"     1
```

```
 2 "Angestellte"           1
 3 "Bäcker*in"             1
 4 "Bauzeichner*in"        1
 5 "Berater*in"            1
 6 "Betriebsprüfer*in"     1
 7 "Bibliothekar*in"       1
 8 "Büchereileiter*in"     1
 9 "Controller"            1
10 "Ergotherapeut*in"      1
# … with 94 more rows
```

Wie Sie sehen, haben wir sehr viele Levels, die nur einmal oder zweimal vorkommen.

Mit der Funktion `fct_lump_min()` können wir niedrigausgeprägte Levels zusammenfassen.

```
pf8 %>%
  drop_na() %>%
  # fasse alle Levels mit Ausprägung kleiner 4 zusammen
  mutate(Beruf = fct_lump_min(Beruf, 4)) %>%
  dplyr::select(Beruf) %>%
  dplyr::count(Beruf) %>%
  arrange(desc(n)) %>%
  as_tibble()
```

```
# A tibble: 22 x 2
   Beruf             n
   <fct>         <int>
 1 Other           101
 2 Student*in       82
 3 kaufmann/frau    29
 4 Handwerker*in    19
 5 Angestellte*r    18
 6 Azubi            17
 7 Pflege           16
 8 Rentner*in       14
 9 Schüler*in       14
10 Lehrer*in        12
# … with 12 more rows
```

Alle Levels mit Ausprägungen kleiner 4 wurden in der neuen Level ‘Other‘ zusammengefasst.

Mit der Funktion `fct_lump_n()` bleiben die höchsten `n` Levels erhalten, während der Rest unter `Other` zusammengefasst wird.

```
pf8 %>%
  drop_na() %>%
  # behalte die 3 höchstausgeprägtesten Levels
  # und fasse den Rest zusammen
  mutate(Beruf = fct_lump_n(Beruf, 3)) %>%
  dplyr::select(Beruf) %>%
  dplyr::count(Beruf) %>%
  arrange(desc(n)) %>%
  as_tibble()
```

```
# A tibble: 4 x 2
  Beruf             n
  <fct>         <int>
1 Other           268
2 Student*in       82
3 kaufmann/frau    29
4 Handwerker*in    19
```

Mit der Funktion `fct_collapse()` können die Levels "von Hand" zusammengefasst werden. Schauen wir uns den Familienstand im Datensatz `pf8` an.

```
levels(pf8$Familienstand)
```

```
[1] "ledig" "Partnerschaft" "verheiratet" "geschieden" "verwitwet" "getrennt"
```

Fassen wir nun alle "Verpartnerte" und alle "Singles" zu zwei Levels zusammen.

```
```{r}
pf8 %>%
  # fasse Singles und Partner in eigenen Levels zusammen
  mutate(Partnerschaft = fct_collapse(Familienstand,
         "Partnerschaft" = c("Partnerschaft", "verheiratet"),
         "Single" = c("ledig", "geschieden", "verwitwet", "getrennt"))) %>%
  dplyr::select(Partnerschaft, Familienstand, Alter) %>%
  as_tibble()
```

```
# A tibble: 731 x 3
   Partnerschaft Familienstand Alter
   <fct>         <fct>         <int>
 1 Partnerschaft Partnerschaft    18
 2 Single        geschieden       67
 3 Partnerschaft Partnerschaft    60
 4 Single        ledig            61
 5 Single        ledig            24
 6 Partnerschaft Partnerschaft    21
 7 Partnerschaft verheiratet      59
 8 Single        geschieden       56
 9 Partnerschaft verheiratet      82
10 Partnerschaft verheiratet      52
# … with 721 more rows
```

1.6.8.10 Daten klassieren

Angenommen, eine Variable enthält Werte zwischen 0 und 500, und wir möchten diese Werte in die Gruppen "0-70", "71-200", "201-400", ">400" klassieren.
Erstellen wir zunächst zufällige Zahlen zwischen 0 und 500.

```
# lade tidyverse
library(tidyverse)
# erzeuge 200 Zufallszahlen von 0 bis 500
dummy <- sample(0:500, 200)
# bzw. direkt als Tibble
dummy <- tibble(x = sample(0:500, 200))
```

Nun erzeugen wir die neue Variable `xKAT`, in welcher die Klassierung angegeben werden soll. Dies kann auf unterschiedliche Weisen erfolgen.

A) mittels `ifelse()`"

Wir können innerhalb von `mutate()` die `ifelse()`-Funktion benutzen. Diese folgte der Logik "`WELCHE - WAS - ANSONSTEN`".
In einem ersten beispielhaften Schritt wählen wir all "$x < 71$" aus, speichern für diese Fälle den `character`-Wert "0-70", und bei allen anderen Fällen ein `NA`.
```

```
dummy %>%
                # ifelse(WELCHE, WAS, ANSONSTEN)
    mutate(xKAT = ifelse(x < 71 , "0-70", NA))
```

Es lassen sich mehrere `ifelse()`-Ausdrücke kombinieren, indem diese vor den `NA`-Ausdruck geschrieben werden. Dabei sammeln sich die Klammer-Zu Symbole) an, kommen Sie hier nicht durcheinander!

```
# Das lässt sich erweitern,
# indem der ", NA"-Ausdruck nach hinten wandert
dummy %>%
  mutate(xKAT = ifelse(x < 71 , "0-70",
                ifelse(x < 201 & x >70 , "71-200",
                       NA)))
```

Der vollständige Befehl zur Klassierung lautet demnach:

```
# wir bilden Kategorien
#       "0-70"
#       "71-200"
#       "201-400"
#       "> 400"
# und speichern das in die neue Variable xKAT
dummy %>%
  mutate(xKAT = ifelse(x < 71 , "0-70",
                ifelse(x < 201 & x >70 ,  "71-200",
                ifelse(x < 401 & x >200 , "201-400",
                ifelse(x > 400 , "> 400",
                             NA)))))
```

B) mittels `case_when()`"

Wenn - so wie hier - mehrere Konditionen angegeben werden, ist die Funktion `case_when()` etwas einfacher zu schreiben und zu lesen als die `ifelse()`-Staffelungen.

```
# nutze case_when() an Stelle von ifelse()
dummy %>%
  mutate(xKAT = case_when(x < 71 ~ "0-70",
                          x < 201 & x > 70  ~ "71-200",
                          x < 401 & x > 200 ~ "201-400",
                          x > 400 ~ "größer 400")
         )
```

Mit dem Parameter `.default` kann ein Wert festgelegt werden, der vergeben wird, wenn keine der Bedingungen zutrifft. Standardmäßig werden `NA`s vergeben.

```
  # nutze case_when() an Stelle von ifelse()
  dummy %>%
    mutate(xKAT = case_when(x < 71 ~ "0-70",
                            x < 201 & x > 70  ~ "71-200",
                            x < 401 & x > 200 ~ "201-400",
                            x > 400 ~ "größer 400",
                            .default="keines davon")
           )
```

Anschließend wandeln wir noch die neue Variable in einen ordinalen Faktor mit korrekter Levelreihenfolge.

```
# die neue Variable als ordinalen Factor mit korrekter
# Levelreihenfolge speichern
dummy <- dummy %>%
  mutate(xKAT = case_when(x < 71 ~ "0-70",
                          x < 201 & x > 70  ~ "71-200",
                          x < 401 & x > 200 ~ "201-400",
                          x > 400 ~ "größer 400",
                          .default="keines davon"),
         xKAT = factor(xKAT, levels=c("0-70",
                                      "71-200",
                                      "201-400",
                                      "größer 400",
                                      "keines davon"),
                       ordered=TRUE)
         )
head(dummy$xKAT)
```

```
[1] 201-400    201-400    201-400    größer 400 201-400
[6] 201-400
Levels: 0-70 < 71-200 < 201-400 < größer 400 < keines davon
```

C) mittels `cut()`"

Innerhalb der `mutate()`-Funktion kann aber auch `cut()` zur Klassierung verwendet werden.

```
  dummy <- dummy %>%
  mutate(xKAT = cut(x, breaks=c(0, 70, 200, 400, Inf),
                       ordered_result = TRUE)
         )
head(dummy$xKAT)
```

```
 [1] (200,400] (200,400] (400,Inf] (70,200]  (400,Inf] (400,Inf]
 Levels: (0,70] < (70,200] < (200,400] < (400,Inf]
```

1.6.8.11 Daten visualisieren

Zum Visualisieren von Daten sind neben Tabellen vor allem Diagramme geeignet. Wie Sie im Tidyverse Diagramme erstellen behandeln wir im Kapitel `ggplot` ab Seite 352.

1.6.9 Schritt 4: Ergebisse kommunizieren

Ihre Ergebnisse können Sie Ihrem Publikum am besten mit `quarto` kommunizieren. Lesen Sie hierzu Abschnitt 1.5 ab Seite 127.

1.6.10 Die Macht der `Pipe`

Die volle Macht entfaltet das Tidyverse durch die Aneinanderreihung von Funktionen und Befehlen mittels der `Pipe`. Im Vergleich zum klassischen kommen Sie so viel leichter zum Ziel, und Ihr Code ist gut les- und nachvollziehbar.
Versuchen Sie den folgenden Code nachzuvollziehen

```
mma %>%
  drop_na() %>%
  dplyr::filter(ward == "Chirurgie") %>%
  group_by(day) %>%
  ggplot(aes(x=category, fill=category)) + geom_bar(aes(fill=qual))
```

```
pf8 %>%
  group_by(Geschlecht, Standort) %>%
  drop_na() %>%
  dplyr::filter(Rauchen == "ja") %>%
  summarise(Mittelwert = mean(Alter),
            Median = median(Alter),
            Stdabw = sd(Alter))
```

1.7 data.table

Neben dem zuvor besprochenen `tidyverse` steht mit `data.table` ein weiterer -Dialekt zur Verfügung, der sich immer größerer Beliebtheit erfreut. Im Kern sind `data.tables` verbesserte Versionen von `data.frames`, die schneller und speichereffizienter arbeiten und mit einer prägnanteren Syntax manipuliert werden können. Das Paket stellt außerdem eine Reihe zusätzlicher Funktionen zum Lesen und Schreiben von tabellarischen Dateien, zum Umformen von Daten zwischen langen und breiten Formaten und zum Verbinden von Datensätzen zur Verfügung.

1.7.1 Installation

Alle Funktionen sind über das Paket `data.table` implementiert, welches wie gewohnt installiert und aktiviert werden kann.

```
# installiere data.table
install.packages("data.table", dependencies=TRUE)
# aktivieren
library(data.table)
```

1.7.2 Modify-in-Place

Der größte Unterschied besteht darin, dass `data.table` die *Modify-in-Place*-Methode verwendet. Das klassische und auch das Tidyverse verwenden die *Copy-on-Modify*-Methode, welche besagt, dass bei der Manipulation eines Objektes das Ergebnis in einem neuen Objekt gespeichert wird.

```
# klassisches "Copy-on-Modify"
meine.daten %>%
  mutate(Neu = Alt*10)
```

Bei oben stehendem Code wird das Objekt `meine.daten` nicht verändert. Das Ergebnis der `mutate()`-Funktion wird als neues Objekt ausgegeben. Dieses neue Objekt ist eine Kopie der Ursprungsdaten `meine.daten`, an welcher die Veränderungen vorgenommen werden.
Mit `data.table` wird der Ansatz *Modify-in-Place* verfolgt.

```
meine.daten[, Neu := Alt*10]
```

Der oben stehende Code erzeugt keine Kopie von `meine.daten`. Vielmehr wird das Objekt `meine.daten` **direkt** verändert. Im klassischen entspricht diese Vorgehensweise dem Code

```
meine.daten$Neu <- meine.daten$Alt*10
```

Durch *Modify-in-Place* wird `data.table` sehr effizient, wenn größere Datenmengen verarbeitet werden sollen. Es kann jedoch auch dazu führen, dass der Code schwieriger zu verstehen ist und überraschende Ergebnisse liefert (insbesondere, wenn ein ‘data.table‘ innerhalb einer Funktion modifiziert wird).

1.7.3 Grundlegende Syntax

Die generelle Syntax von `data.table` lautet

```
dt[i, j, by]
```

wobei

- `dt` ein `data.table`-Objekt ist.
- `i` zum Filtern und für join-Funktionen genutzt wird.
- `j` zum Manipulieren, Transformieren und Zusammenfassen der Daten verwendet wird.
- `by` zum Gruppieren genutzt wird.

Man kann die Syntax lesen als:

> „In diesen Zeilen, mache dies, gruppiert nach jenem“.

1.7.4 Daten einlesen

Der erste Schritt der meisten Datenanalysen besteht darin, Daten in den Speicher zu laden. Wir können die Funktion `data.table::fread()` verwenden (das `f` steht für *fast* (schnell)), um reguläre, durch Trennzeichen getrennte Dateien wie `txt`- oder `csv`-Dateien zu lesen. Diese Funktion ist nicht nur schnell, sondern erkennt automatisch das Trennzeichen und errät die Klasse jeder Spalte sowie die Anzahl der Zeilen in der Datei.

```
dt <- fread("data/Befragung22.csv")

# anschauen
str(dt)
```

```
Classes ‘data.table’ and 'data.frame':^^I37 obs. of  6 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : chr  "selten" "selten" "selten" "oft" ...
- attr(*, ".internal.selfref")=<externalptr>
```

Das Objekt `dt` gehört sowohl zur Klasse `data.frame` als auch zu der neuen Klasse `data.table`.

```
dt <- fread("https://www.produnis.de/R/data/Befragung22.csv")
```

Liegen die Daten bereits als `data.frame` vor, können sie per `as.data.table()` umgewandelt werden.

```
df <- read.table("https://www.produnis.de/R/data/Datentabelle.txt",
header=TRUE)

# wandle in data.table um
dt2 <- as.data.table(df)

# anschauen
str(dt2)
```

```
  Classes ‘data.table’ and 'data.frame':^^I10 obs. of  4 variables:
  $ Geschlecht: chr  "m" "w" "w" "m" ...
  $ Alter     : int  28 18 25 29 21 19 27 26 31 22
  $ Gewicht   : int  80 55 74 101 84 74 65 56 88 78
  $ Groesse   : int  170 174 183 190 185 178 169 163 189 184
  - attr(*, ".internal.selfref")=<externalptr>
```

Sollen die Daten von Hand eingegeben werden, wird die Funktion `data.table()` verwendet.

```
  dt3 <- data.table(x = 1:10,
  y = 11:20,
  z = factor(rep(c("foo", "bar"), 5))
  )

# anschauen
str(dt3)
```

```
  Classes ‘data.table’ and 'data.frame':^^I10 obs. of  3 variables:
  $ x: int  1 2 3 4 5 6 7 8 9 10
  $ y: int  11 12 13 14 15 16 17 18 19 20
  $ z: Factor w/ 2 levels "bar","foo": 2 1 2 1 2 1 2 1 2 1
  - attr(*, ".internal.selfref")=<externalptr>
```

1.7.5 Daten speichern

Mit der Funktion `fwrite()` können `data.tables` (aber auch `data.frames`) in eine Datei gespeichert werden. Sie funktioniert ähnlich wie `write.csv`, ist aber wesentlich schneller. Wird kein Dateiname angegeben, erfolgt die Ausgabe in der Konsole. So kann überprüft werden, was in die Datei geschrieben würde.

```
  # schreibe Objekt "dt2" in die Konsole
  fwrite(dt2)
```

```
Geschlecht,Alter,Gewicht,Groesse
m,28,80,170
w,18,55,174
w,25,74,183
m,29,101,190
m,21,84,185
w,19,74,178
w,27,65,169
w,26,56,163
m,31,88,189
m,22,78,184
```

```
  # schreibe Objekt "dt" in datei "dt.csv"
  fwrite(dt2, "dt2.csv")
  # schreibe Objekt "dt" in datei "dt.txt"
  fwrite(dt2, "dt2.txt")
```

1.7.6 Fälle filtern mit `i`

Wir erinnern uns, dass die allgemeine Syntax `dt[i, j, by]` lautet. Über den Parameter `i` können die Daten gefilter werden, so dass nur bestimmte Fälle berücksichtigt werden.
Beispielsweise könnten wir im Objekt `dt` nur solche Fälle auswählen, bei denen das Alter größer als 30 ist.

```
1  dt[alter > 30]
```

```
   alter geschlecht stifte      geburtsort fahrzeit  podcast
   <int>     <char>  <int>          <char>    <int>   <char>
1:    41   männlich      1            Bonn       60   selten
2:    34   weiblich     13      Düsseldorf       25      oft
3:    38   weiblich     25       Dinslaken       50      oft
4:    38   männlich      5          Donezk       57 manchmal
5:    31   weiblich     16 Charkov Ukraine      135      oft
6:    36   weiblich      1          Rybnik       90 manchmal
7:    45   männlich      1   Gelsenkirchen       85      oft
```

Dies ist Vergleichbar mit dem klassischen -Aufruf

```
1  dt[dt$alter > 30]
```

```
   alter geschlecht stifte      geburtsort fahrzeit  podcast
   <int>     <char>  <int>          <char>    <int>   <char>
1:    41   männlich      1            Bonn       60   selten
2:    34   weiblich     13      Düsseldorf       25      oft
3:    38   weiblich     25       Dinslaken       50      oft
4:    38   männlich      5          Donezk       57 manchmal
5:    31   weiblich     16 Charkov Ukraine      135      oft
6:    36   weiblich      1          Rybnik       90 manchmal
7:    45   männlich      1   Gelsenkirchen       85      oft
```

Da alle Ausdrücke in `i` im Kontext der `data.table` ausgewertet werden, müssen wir den (eventuell sehr langen) Namen des Objektes nicht erneut eingeben. Dies ist vor allem bei längeren Ausdrücken sehr bequem.

```
1  # erzeuge langen Objektnamen
2  langer.Objekt.name <- dt
```

Der klassische ‘R‘-Aufruf

```
1  langer.Objekt.name[langer.Objekt.name$alter > 25 &
2                     langer.Objekt.name$geschlecht=="männlich" |
3                     langer.Objekt.name$stifte > 30]
```

```
   alter geschlecht stifte    geburtsort fahrzeit  podcast
   <int>     <char>  <int>        <char>    <int>   <char>
1:    41   männlich      1          Bonn       60   selten
2:    26   männlich      5    Düsseldorf       40      nie
3:    38   männlich      5        Donezk       57 manchmal
4:    20   weiblich     32         Wesel       89      nie
5:    45   männlich      1 Gelsenkirchen       85      oft
```

verkürzt sich auf

```
1  langer.Objekt.name[alter > 25 & geschlecht=="männlich" | stifte > 30]
```

1.7.7 Fälle sortieren mit `i`

Dem Parameter `i` können auch Funktionen übergeben werden. So lassen sich die Daten beispielsweise über die `order()`-Funktion sortieren.

```
# nehme anderen (kürzeren) Datensatz  zur Demonstration
dt2[order(Alter)]

# absteigend
dt2[order(Gewicht, decreasing = TRUE)]
```

```
   Geschlecht Alter Gewicht Groesse
   <char> <int>   <int>   <int>
1:          m    29     101     190
2:          m    31      88     189
3:          m    21      84     185
4:          m    28      80     170
5:          m    22      78     184
6:          w    25      74     183
7:          w    19      74     178
8:          w    27      65     169
9:          w    26      56     163
10:         w    18      55     174
```

1.7.8 Daten verarbeiten mit `j`

Nachdem der Datensatz mittels `i` eventuell vorsortiert und -gefiltert wurde, erfolgen die eigentlichen Operationen über den Parameter `j`. So können wir den Mittelwert des Alters der Probanden wie folgt bestimmen:

```
# Mittelwert des Alters
dt[, mean(alter)]
```

```
[1] 25.2973
```

```
# Mittelwert des Alters der Männer
dt[geschlecht == "männlich", mean(alter)]
```

```
[1] 29
```

Innerhalb von `j` kann jede Funktion verwendet werden. So könnten wir überprüfen, ob die Variablen `fahrzeit` und `alter` miteinander korrelieren (ja, das ist quatsch).

```
# korrelieren alter und fahrzeit?
dt[, cor(alter, fahrzeit)]
```

```
[1] 0.1504465
```

Es können auch mehrere Funktionen angewendet werden. Hierfür müssen diese per `list()` an den Parameter `j` übergeben werden. Auf diese Weise könnten wir Median, Mittelwert und Standardabweichung des Alters der Probanden bestimmen.

```
  # mehrere Funktionen per list()
  dt[, list(Median = median(alter),
            Mittelw = mean(alter),
            Stdabw = sd(alter))]
```

```
   Median Mittelw   Stdabw
   <int>   <num>    <num>
1:     22 25.2973 6.765373
```

Da der Parameter `j` immer eine Liste erwartet, kann die Funktion `list()` mit einem Punkt abgekürzt werden.

```
  # geht auch mit "."
  dt[, .(Median = median(alter),
         Mittelw = mean(alter),
         Stdabw = sd(alter),
         InterquA = IQR(alter))]
```

```
   Median Mittelw   Stdabw InterquA
   <int>   <num>    <num>    <num>
1:     22 25.2973 6.765373        6
```

1.7.9 Daten bearbeiten mit `j`

Über den Parameter `j` können die Daten auch manipuliert werden, ähnlich wie bei der `mutate()`-Funktion des Tidyverse. Eine neue Variable kann über die Zeichenkette `:=` definiert werden (dem so genannten *Walrus Operator* (Walross-Operator), der so heisst, weil die Zeichenfolge `:=` an die Stoßzähne eines Walrosses erinnert. Das Logo des `data.table`-Pakets zeigt eine Robbe, was zur humorvollen Verbindung beigetragen hat).
Mit folgendem Aufruf erzeugen wir eine neue Variable `FahrzeitH`, welche die `fahrzeit` in Stunden beinhalten soll.

```
  dt[, FahrzeitH := fahrzeit/60]

  # anzeigen
  str(dt)
```

```
 Classes ‘data.table’ and 'data.frame':^^I37 obs. of  7 variables:
 $ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
 $ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
 $ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
 $ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
 $ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
 $ podcast   : chr  "selten" "selten" "selten" "oft" ...
 $ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "index")= int(0)
  ..- attr(*, "__geschlecht")= int [1:37] 3 8 10 11 13 31 33 34 37 1 ...
```

So können wir auch mittels der `cut()`-Funktion die Daten klassieren, zum Beispiel das Alter:

```
  dt[, alterK := cut(alter, breaks=c(0,20,25,30,40,50),
                            ordered=TRUE)]

# anzeigen
str(dt)
```

```
Classes 'data.table' and 'data.frame':^^I37 obs. of  8 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : chr  "selten" "selten" "selten" "oft" ...
$ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
$ alterK    : Ord.factor w/ 5 levels "(0,20]"<"(20,25]"<..: 1 3 5 4 3 4 3 2 3 3 ...
- attr(*, ".internal.selfref")=<externalptr>
- attr(*, "index")= int(0)
 ..- attr(*, "__geschlecht")= int [1:37] 3 8 10 11 13 31 33 34 37 1 ...
```

Pro Aufruf kann der Walross-Operator nur einmal verwendet werden. Sollen mehrere Variablen verändert oder hinzugefügt werden, steht die `let()`-Funktion bereit. Innerhalb von `let()` werden wie gewohnt *einfache* Gleichheitszeichen verwendet.

```
# mehrere Manipulationen per let()
dt[, let(geschlecht = factor(geschlecht),
         geburtsort = factor(geburtsort),
         podcast = factor(podcast, ordered=TRUE,
                          levels=c("nie", "selten", "manchmal",
                                   "oft", "immer")))]
# anzeigen
str(dt)
```

```
Classes 'data.table' and 'data.frame':^^I37 obs. of  9 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: Factor w/ 2 levels "männlich","weiblich": 2 2 1 2 2 2 2 1 2 1 ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: Factor w/ 26 levels "Bagdad","Bonn",..: 9 21 2 10 8 5 21 7 18 10 ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : Ord.factor w/ 5 levels "nie"<"selten"<..: 2 2 2 4 NA 4 4 3 1 1 ...
$ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
$ alterK    : Ord.factor w/ 5 levels "(0,20]"<"(20,25]"<..: 1 3 5 4 3 4 3 2 3 3 ...
$ kuckuck   : int  12 315 60 325 270 1250 1160 60 120 200 ...
- attr(*, ".internal.selfref")=<externalptr>
- attr(*, "index")= int(0)
```

Die Änderungen wurden direkt im Objekt `dt` gespeichert.

1.7.10 `data.table` kopieren

Eine weitere wesentliche Eigenschaft von `data.table`-Objekten besteht darin, dass man sie gesondert kopieren muss. Wir eine `data.table` auf klassischem Wege in ein neues Objekt "kopiert", so erfolgt keine echte Kopie, sondern lediglich ein *symbolischer Link* auf das ursprüngliche Objekt.

```
neu <- dt

str(neu)
```

```
Classes 'data.table' and 'data.frame':^^I37 obs. of  8 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : chr  "selten" "selten" "selten" "oft" ...
```

```
 $ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
 $ alterK    : Ord.factor w/ 5 levels "(0,20]"<"(20,25]"<..: 1 3 5 4 3 4 3 2 3 3 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "index")= int(0)
  ..- attr(*, "__geschlecht")= int [1:37] 3 8 10 11 13 31 33 34 37 1 ...
```

Wir haben das Objekt `dt` nur *scheinbar* in das neue Objekt `neu` kopiert. Wenn wir Änderungen am Objekt `neu` vornehmen, so sind diese auch im Objekt `dt` präsent, weil eben **nicht** kopiert, sondern nur ein *Verweis* erstellt wurde.

```
# erstelle neue Variable in "neu"
neu[, kuckuck := fahrzeit * stifte]

# die neue Variable ist auch in "dt" enthalten
str(dt)
```

```
Classes ‘data.table’ and 'data.frame':^^I37 obs. of  9 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : chr  "selten" "selten" "selten" "oft" ...
$ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
$ alterK    : Ord.factor w/ 5 levels "(0,20]"<"(20,25]"<..: 1 3 5 4 3 4 3 2 3 3 ...
$ kuckuck   : int  12 315 60 325 270 1250 1160 60 120 200 ...
- attr(*, ".internal.selfref")=<externalptr>
- attr(*, "index")= int(0)
 ..- attr(*, "__geschlecht")= int [1:37] 3 8 10 11 13 31 33 34 37 1 ...
```

💀 Achtung, Anfängerfehler! 💀

Dies ist ein häufiger fataler Anfängerfehler, der zum Datenverlust führen kann!

Um das Objekt tatsächlich zu kopieren, muss die Funktion `copy()` verwendet werden.

```
# kopieren dt2 nach neu2
neu2 <- copy(dt2)

# anzeigen
str(neu2)
```

```
Classes ‘data.table’ and 'data.frame':^^I10 obs. of  4 variables:
$ Geschlecht: chr  "m" "w" "w" "m" ...
$ Alter     : int  28 18 25 29 21 19 27 26 31 22
$ Gewicht   : int  80 55 74 101 84 74 65 56 88 78
$ Groesse   : int  170 174 183 190 185 178 169 163 189 184
- attr(*, ".internal.selfref")=<externalptr>
```

```
neu2[, Kuckuck := Groesse/Gewicht]

# dt2 ist unverändert
str(dt2)
```

```
Classes ‘data.table’ and 'data.frame':^^I10 obs. of  4 variables:
$ Geschlecht: chr  "m" "w" "w" "m" ...
$ Alter     : int  28 18 25 29 21 19 27 26 31 22
$ Gewicht   : int  80 55 74 101 84 74 65 56 88 78
$ Groesse   : int  170 174 183 190 185 178 169 163 189 184
- attr(*, ".internal.selfref")=<externalptr>
```

1.7.11 pipen

Innerhalb von `data.table` kann auch die Pipe verwendet werden. Wird die R -Base-Pipe `|>` verwendet, kann mittels Unterstrich `_` auf den weitergeleiteten Datenstrom zugegriffen werden. Bei der Tidyverse-Pipe (eigentlich von `magrittr`) mit der Zeichenfolge `%>%` muss ein Punkt `.` verwendet werden.
Folgende Aufrufe filtern das `geschlecht` und pipen den Datenstrom weiter. Anschließend wird nach `alter` sortiert.

```
1  # Daten pipen mit R_Base
2  dt2[Geschlecht=="m"] |>
3    _[order(Alter)]
```

```
  Geschlecht Alter Gewicht Groesse
  <char> <int>   <int>   <int>
1:          m    21      84     185
2:          m    22      78     184
3:          m    28      80     170
4:          m    29     101     190
5:          m    31      88     189
```

```
1  # Daten pipen mit magrittr
2  dt2[Geschlecht=="m"] %>%
3    .[order(Alter)]
```

```
  Geschlecht Alter Gewicht Groesse
  <char> <int>   <int>   <int>
1:          m    21      84     185
2:          m    22      78     184
3:          m    28      80     170
4:          m    29     101     190
5:          m    31      88     189
```

Oder wir erstellen ein linerares Modell und pipen es an die `summary()`-Funktion weiter.

```
1  dt2[, lm(Gewicht ~ Groesse)] |>
2  summary()
```

```
Call:
lm(formula = Gewicht ~ Groesse)

Residuals:
     Min       1Q   Median       3Q      Max
-14.9024  -3.4756  -0.3902   1.0915  15.0732

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -146.5366    58.3503  -2.511  0.03630 *
```

```
Groesse        1.2439     0.3265   3.810  0.00516 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 8.992 on 8 degrees of freedom
Multiple R-squared:  0.6447,^^IAdjusted R-squared:  0.6003
F-statistic: 14.51 on 1 and 8 DF,  p-value: 0.005164
```

Wir können den Ausdruck aber auch direkt in die `summary()`-Funktion schreiben.

```
summary(dt2[, lm(Gewicht ~ Groesse)])
```

```
Call:
lm(formula = Gewicht ~ Groesse)

Residuals:
     Min       1Q   Median       3Q      Max
-14.9024  -3.4756  -0.3902   1.0915  15.0732

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -146.5366    58.3503  -2.511  0.03630 *
Groesse        1.2439     0.3265   3.810  0.00516 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 8.992 on 8 degrees of freedom
Multiple R-squared:  0.6447,^^IAdjusted R-squared:  0.6003
F-statistic: 14.51 on 1 and 8 DF,  p-value: 0.005164
```

1.7.12 Ergebnisse gruppieren mit `by`

Über den Paramter `by` können die Ergebnisse gruppiert werden.

```
# gruppiert nach Geschlecht
dt[, .(Median = median(alter),
       Mittelw = mean(alter),
       Stdabw = sd(alter)),
   by = geschlecht]
```

```
  geschlecht Median  Mittelw   Stdabw
  <char>  <num>    <num>    <num>
1:   weiblich   21.5 24.10714 5.251732
2:   männlich   25.0 29.00000 9.617692
```

Die Ausgabe kann gepipet und weiterverarbeitet werden. In folgendem Beispiel berechnen wir den Variationskoeffizienten ($sd/\bar{x}$) aus den gruppierten Ergebnissen.

```
dt[, .(Median = median(alter),
       Mittelw = mean(alter),
       Stdabw = sd(alter)),
    by = geschlecht] |>
# berechnen
_[, VK := Stdabw / Mittelw] |>
# anzeigen
_[]
```

```
   geschlecht Median  Mittelw   Stdabw        VK
       <char>  <num>    <num>    <num>     <num>
1:   weiblich   21.5 24.10714 5.251732 0.2178496
2:   männlich   25.0 29.00000 9.617692 0.3316446
```

Bitte beachten Sie, dass wir in diesem Beispiel die Anzeige der Endergebnisse mittels `|> _[]` erzwingen mussten. Dies ist notwendig, wenn per `by` gruppierte Ergebnisse weiter manipuliert werden sollen. `Data.table` speichert Änderungen durch `:=` immer direkt im Objekt, wobei keine Ausgabe der Daten erfolgt. Im vorliegenden Fall von `VK := Stdabw / Mittelw` ist diese Speicherung jedoch nicht möglich (ausgegeben wird ja eh nichts), da sich das Endergebnis nicht mehr auf das ursprüngliche Objekt `dt` bezieht.
In diesem Fall ist es (sogar) möglich und üblich, das Ergebnis wie gewohnt in einem neuen Objekt zu *speichern*, ohne dass dabei ein symbolischer Link angelegt wird.

```
neu3 <- dt[, .(Median = median(alter),
               Mittelw = mean(alter),
               Stdabw = sd(alter)),
            by = geschlecht] |>
_[, VK := Stdabw / Mittelw] |>
_[]

# anzeigen
neu3
```

```
   geschlecht Median  Mittelw   Stdabw        VK
       <char>  <num>    <num>    <num>     <num>
1:   weiblich   21.5 24.10714 5.251732 0.2178496
2:   männlich   25.0 29.00000 9.617692 0.3316446
```

Wir können den letzten Pipevorgang abkürzen, indem wir einfach eckige Klammern `[]` an unseren Aufruf anhängen.

```
neu4 <- dt[, .(Median = median(alter),
               Mittelw = mean(alter),
               Stdabw = sd(alter)),
            by = geschlecht] |>
_[, VK := Stdabw / Mittelw][]

# anzeigen
neu4
```

```
   geschlecht Median  Mittelw   Stdabw        VK
       <char>  <num>    <num>    <num>     <num>
1:   weiblich   21.5 24.10714 5.251732 0.2178496
2:   männlich   25.0 29.00000 9.617692 0.3316446
```

1.7.13 Weitere Funktionen aus dem `data.table` Paket

Das Paket `data.table` bringt zahlreiche eigene Funktionen mit, um typische Aufgabenstellungen effizienter bearbeiten zu können.

1.7.13.1 Einzigartige bestimmen mit `uniqueN`

Um zum Beispiel die Anzahl verschiedener Städte innerhalb der Variable `geburtsort` zu bestimmen, können wir auf die paketeigene Funktion `uniqueN()` zurückgreifen:

```
# wieviele unterschiedliche Städte sind in "geburtsort"?
dt[, uniqueN(geburtsort)]
```

```
[1] 26
```

1.7.13.2 Anzahl der Fälle mit `.N`

Mit der Funktion `.N` kann die Anzahl der Fälle ermittelt werden.

```
dt[, .(Anzahl = .N),
     by = geschlecht]
```

```
   geschlecht Anzahl
       <char>  <int>
1:   weiblich     28
2:   männlich      9
```

Mit Hilfe von `nrow()` können so prozentuale Anteile berechnet werden.

```
dt[, .(Anzahl = .N,
       Prozent = .N/nrow(dt)*100),
     by = alterK]
```

```
    alterK Anzahl   Prozent
     <ord>  <int>     <num>
1:  (0,20]      9 24.324324
2: (25,30]      6 16.216216
3: (40,50]      2  5.405405
4: (30,40]      5 13.513514
5: (20,25]     15 40.540541
```

Die Ergebnisse können an `ggplot()` weitergereicht werden.

```
library(ggplot2)

dt[, .(Anzahl = .N,
       Prozent = .N/nrow(dt)*100),
       by = alterK] |>
  ggplot(aes(x=alterK, y=Prozent)) +
  geom_col(color="black", fill="orchid")
```

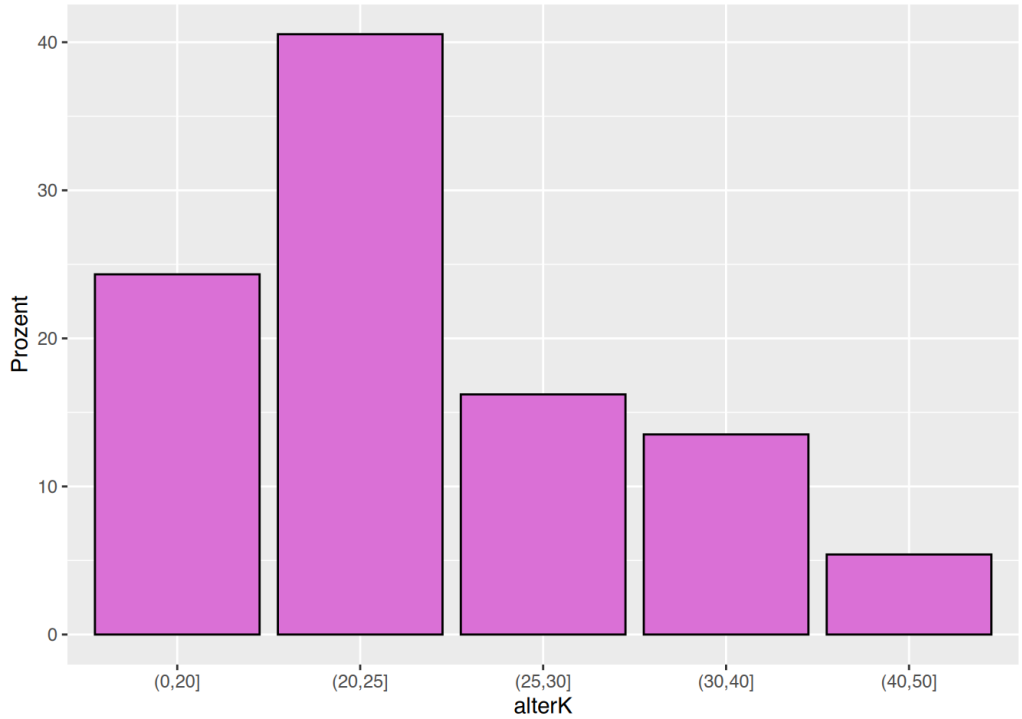


Abb. 1.97

1.7.13.3 Lange Tabelle erzeugen mit `melt()`

Mit der Funktion `melt()` können breite Tabellen in lange (tidy) umgewandelt werden, ähnlich wie mit `dplyr::pivot_longer()`.
Zur Demonstration verwenden wir die Pflegetabelle von Isfort [2].

```
load("https://www.produnis.de/R/data/Pflegeberufe.RData")

# anschauen
Pflegeberufe
```

```
                          1999   2001   2003   2005   2007   2009   2011   2013   2015
  Krankenpflegeassistenz 16624  19061  19478  21537  27731  36481  46517  54371  64127
  Altenpflegehilfe       55770  52710  49727  45776  48326  47903  47978  48363  49507
  Kinderkrankenpflege    47779  48203  48822  48519  49080  49307  48291  48937  48913
  Krankenpflege         430983 436767 444783 449355 457322 465446 468192 472580 476416
  Altenpflege           109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Die Tabelle ist nicht *tidy* und liegt im breiten Format vor. Ausserdem ist sie von der Klasse `matrix`.

```
  # wandle um in data.table
  pf <- as.data.table(Pflegeberufe, keep.rownames = "Berufsgruppe")

  # anzeigen
  pf
```

```
             Berufsgruppe   1999   2001   2003   2005   2007   2009   2011   2013   2015
                   <char>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: Krankenpflegeassistenz  16624  19061  19478  21537  27731  36481  46517  54371  64127
2:       Altenpflegehilfe  55770  52710  49727  45776  48326  47903  47978  48363  49507
```

```
3:    Kinderkrankenpflege  47779  48203  48822  48519  49080  49307  48291  48937  48913
4:          Krankenpflege 430983 436767 444783 449355 457322 465446 468192 472580 476416
5:            Altenpflege 109161 124879 141965 158817 178902 194195 208304 227154 246412
```

Mittels `melt()` transformieren wir `pf` in eine lange (tidy) Tabelle. Dabei übergeben wir dem Parameter

- `id.vars` alle Variabelen, welche "Identifikatoren" beinhalten. Damit sind alle Spalten gemeint, die keine konkrekten Messwerte enhalten, sondern weitere *bezeichnende* Kennwerte. Klassischer Weise sind dies vor allem die **Zeilen**namen, in unserem Falle also `Berufsgruppe`. Es können mehrere `id.vars` mittels `c()` aneinandergereiht werden.
- `measure.vars` alle Spalten, welche die eigentlichen Messwerte enthalten, in unserem Falle 1999:2015 (alles außer `Berufsgruppe`). Wird dieser Parameter leer gelassen, nimmt `data.table` automatisch alle Spalten, die keine `id.vars` sind.
- `variable.name` den Name der neuen Spalte, in welche die Bezeichnungen der `measure.vars` überführt werden sollen, in unserem Fall `Jahr`.
- `value.name` den Name der neuen Spalte, in welche die Werte der `measure.vars` überführt werden sollen, in unserem Fall `Anzahl`.

Da wir alle Spalten außer Berufsgruppe *melten* wollen, kann der Parameter `measure.vars` weggelassen werden.

```r
# pf mit melt() tidy machen
pf_tidy <- melt(pf, id.vars = "Berufsgruppe",
                variable.name = "Jahr",
                value.name = "Anzahl")

# anschauen
head(pf_tidy)
```

```
             Berufsgruppe   Jahr Anzahl
                   <char> <fctr>  <num>
1: Krankenpflegeassistenz   1999  16624
2:       Altenpflegehilfe   1999  55770
3:    Kinderkrankenpflege   1999  47779
4:          Krankenpflege   1999 430983
5:            Altenpflege   1999 109161
6: Krankenpflegeassistenz   2001  19061
```

1.7.13.4 Breite Tabelle erzeugen mit `dcast()`

Mittels `dcast()` können lange Tabellen wieder in breite Tabellen transformiert werden, so wie bei `dplyr::pivot_wider()`. Der Aufruf folgt der Semantik:

```
dcast(Bezeichner ~ Spaltenname, value.var = "Wertename")
```

wobei

- `Bezeichner` die Spalten der `id.vars` (Identifikatoren) meint.

- `Spaltenname` die Spalte mit der `variable.name` meint.
- `value.var` den Namen der Spalte meint, welche die konkreten Messwerte enthält. Diese muss in Anführungszeichen angegeben werden. Wird dieser Parameter weggelassen, versucht `data.table` die korrekte Spalte zu erraten (was einfach ist, wenn nur noch eine Spalte übrig bleibt).

```
# wandle pf_tdiy mit dcast() in breite Tabelle
pf_wide <- dcast(pf_tidy, Berufsgruppe ~ Jahr,
                         value.var = "Anzahl")


# anschauen
head(pf_wide)
```

```
  Key: <Berufsgruppe>
             Berufsgruppe   1999   2001   2003   2005   2007   2009   2011   2013   2015
                   <char>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1:            Altenpflege 109161 124879 141965 158817 178902 194195 208304 227154 246412
2:       Altenpflegehilfe  55770  52710  49727  45776  48326  47903  47978  48363  49507
3:    Kinderkrankenpflege  47779  48203  48822  48519  49080  49307  48291  48937  48913
4:          Krankenpflege 430983 436767 444783 449355 457322 465446 468192 472580 476416
5: Krankenpflegeassistenz  16624  19061  19478  21537  27731  36481  46517  54371  64127
```

1.7.13.5 Subset of Data (`.SD`)

Subset of Data wird in der `data.table`-Syntax verwendet, um eine Teilmenge der Daten in einer speziellen Umgebung zu referenzieren. Die Funktion hierfür heisst `.SD` und enthält eine Auswahl an Spalten der `data.table`, die weiterverarbeitet werden können, z. B. durch Berechnungen oder Transformationen.
`.SD` ist besonders nützlich, wenn Sie Berechnungen oder Transformationen nur auf bestimmte Spalten anwenden möchten, während die anderen Spalten beibehalten werden sollen.
Schauen wir uns nochmal unser Objekt `pf` an.

```
str(pf)
```

```
Classes 'data.table' and 'data.frame':^^I5 obs. of  10 variables:
$ Berufsgruppe: chr  "Krankenpflegeassistenz" "Altenpflegehilfe" "Kinderkrankenpflege" "Krankenpflege" ...
$ 1999        : num  16624 55770 47779 430983 109161
$ 2001        : num  19061 52710 48203 436767 124879
$ 2003        : num  19478 49727 48822 444783 141965
$ 2005        : num  21537 45776 48519 449355 158817
$ 2007        : num  27731 48326 49080 457322 178902
$ 2009        : num  36481 47903 49307 465446 194195
$ 2011        : num  46517 47978 48291 468192 208304
$ 2013        : num  54371 48363 48937 472580 227154
$ 2015        : num  64127 49507 48913 476416 246412
- attr(*, ".internal.selfref")=<externalptr>
```

Nun verwenden wir `.SD`, um für jede Spalte den Mittelwert zu berechnen.

```
pf[, lapply(.SD, mean)]
```

```
  Berufsgruppe     1999   2001   2003     2005     2007     2009     2011   2013   2015
          <num>    <num>  <num>  <num>    <num>    <num>    <num>    <num>  <num>  <num>
1:           NA 132063.4 136324 140955 144800.8 152272.2 158666.4 163856.4 170281 177075
```

Der Aufruf `lapply(.SD, mean)` wendet die Funktion `mean` auf jede Spalte in `.SD` an.

1.7.13.5.1 Verwendung von `.SDcols`

`.SDcols` ist ein optionaler Parameter, mit dem die Spalten, die in `.SD` enthalten sind, gezielt ausgewählt werden können.
Schauen wir uns das Objekt `dt` an.

```
1  str(dt)
```

```
Classes 'data.table' and 'data.frame':^^I37 obs. of  9 variables:
$ alter     : int  20 28 41 34 26 38 28 21 27 26 ...
$ geschlecht: chr  "weiblich" "weiblich" "männlich" "weiblich" ...
$ stifte    : int  12 7 1 13 18 25 29 1 2 5 ...
$ geburtsort: chr  "Düren" "Neuss" "Bonn" "Düsseldorf" ...
$ fahrzeit  : int  1 45 60 25 15 50 40 60 60 40 ...
$ podcast   : chr  "selten" "selten" "selten" "oft" ...
$ FahrzeitH : num  0.0167 0.75 1 0.4167 0.25 ...
$ alterK    : Ord.factor w/ 5 levels "(0,20]"<"(20,25]"<..: 1 3 5 4 3 4 3 2 3 3 ...
$ kuckuck   : int  12 315 60 325 270 1250 1160 60 120 200 ...
- attr(*, ".internal.selfref")=<externalptr>
- attr(*, "index")= int(0)
 ..- attr(*, "__geschlecht")= int [1:37] 3 8 10 11 13 31 33 34 37 1 ...
```

Wir können nun den Median für alle mindestens ordinalskalierten Variablen (`alter`, `stifte`, `fahrzeit`) berechnen, indem wir diese Variablen per `.SDcols` angeben.

```
1  # Wähle die Spalten aus, die verwendet werden sollen
2  dt[, lapply(.SD, median), .SDcols=c("alter", "stifte", "fahrzeit")]
```

```
   alter stifte fahrzeit
   <int>  <int>    <int>
1:    22      8       60
```

Dies klappt auch mit Gruppierungen. Wir berechnen die Werte erneut, diesmal aber getrennt nach `geschlecht`.

```
1  dt[, lapply(.SD, median),
2  by=geschlecht, .SDcols=c("alter", "stifte", "fahrzeit")]
```

```
   geschlecht alter stifte fahrzeit
       <char> <num>  <num>    <num>
1:   weiblich  21.5     12       60
2:   männlich  25.0      5       57
```

Auf diese Weise können Spalten auch transformiert werden. Angenommen, wir möchten die Werte für `alter` und `stifte` verdoppeln, dann lautet der Aufruf:

```
dt[, (c("alter", "stifte")) := lapply(.SD, function(x) x*2),
.SDcols = c("alter", "stifte")]

head(dt)
```

```
   alter geschlecht stifte geburtsort fahrzeit podcast  FahrzeitH  alterK kuckuck
   <num>     <char>  <num>     <char>    <int>  <char>      <num>   <ord>   <int>
1:    40   weiblich     24      Düren        1  selten 0.01666667  (0,20]      12
2:    56   weiblich     14      Neuss       45  selten 0.75000000 (25,30]     315
3:    82   männlich      2       Bonn       60  selten 1.00000000 (40,50]      60
4:    68   weiblich     26 Düsseldorf       25     oft 0.41666667 (30,40]     325
5:    52   weiblich     36   Duisburg       15    <NA> 0.25000000 (25,30]     270
6:    76   weiblich     50  Dinslaken       50     oft 0.83333333 (30,40]    1250
```

1.7.14 Cheat Sheet und Übungsaufgaben

Auf GitHub ist ein schöner Cheat-Sheet für `data.table` vorhanden. Das PDF können Sie unter https://raw.githubusercontent.com/rstudio/cheatsheets/master/datatable.pdf herunterladen.
Des weiteren stehen mit dem *table traineR* [7] eine Reihe an Übungsaufgaben bereit, an denen Sie Ihre `data.table`-Fähigkeiten ausprobieren können, siehe https://www.produnis.de/tabletrainer/.

1.8 Weitere Hilfsmittel

1.8.1 Cheatsheets

In `RStudio` ist ein Liste mit "Cheatsheets" enthalten. Ein Cheatsheet (*Schummelzettel*) ist eine Seite, auf der (möglichs) alle wichtigen Befehle zu einem Thema (z.B. LaTeX oder R oder `ggplot`) enthalten sind. Diese Seite kann z.B. ausgedruckt und neben den PC bzw. ans Pinnboard geheftet werden. So hat man immer einen guten Überblick über die wichtigsten Befehle oder Funktionen.
Klicken Sie in der Menüzeile von `RStudio` auf `Help` → `Cheatsheets` , um eine Liste der enthaltenen Cheatsheets zu erhalten. Alle Sheets sind auf dieser Webseite verfügbar:
https://www.rstudio.com/resources/cheatsheets/

Folgende Cheatsheets sind besonders empfehlenswert:

- Klassisches `R`
  http://github.com/rstudio/cheatsheets/blob/main/base-r.pdf
- Datentransformation mit `dplyr`
  https://raw.githubusercontent.com/rstudio/cheatsheets/master/data-transformation.pdf
- Grafiken erstellen mit `ggplot`
  https://raw.githubusercontent.com/rstudio/cheatsheets/master/data-visualization-2.1.pdf
- `RMarkdown`
  https://raw.githubusercontent.com/rstudio/cheatsheets/master/rmarkdown-2.0.pdf
- `RMarkdown` references
  https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

1.8.2 Youtube

Auf Youtube gibt es zahlreiche Anleitungsvideos zu R und `RStudio`. In der folgenden Liste möchten wir ein paar sehenswerte Videos vorschlagen.

- R und RStudio installieren
  https://www.youtube.com/watch?v=X_Mxya2Fis0
- Erste Schritte in RStudio
  https://www.youtube.com/watch?v=tyvEHQszZJs
- Einfaches Rechnen und Variablen in R
  https://www.youtube.com/watch?v=qOVhiOu_ARs
- Zusatzpakete installieren
  https://www.youtube.com/watch?v=YD8OLpvkHD8
- Intro to the Tinyverse (englisch)
  https://www.youtube.com/watch?v=MKwyauo8nSI

Ebenso gibt es empfehlenswerte Playlists:

- "Statistik in R" von Björn Walther
  https://www.youtube.com/playlist?list=PLJ8d6dduOfrpqxUw6Y7lGV4SoEkI9gIAi
- "Statistik in R" von Daniela Keller
  https://www.youtube.com/watch?v=dEsm3PHrAHE&list=PLTMQeIvhjir5BEI7DfSAbTmHREI8Zqpnv
- "Statistische Methoden 1" von Florian Heiss
  https://www.youtube.com/playlist?list=PLVOKtPjp4poWkwg73O5ykfWu2mPviYnCq
- Quarto-Playlist (englisch)
  https://www.youtube.com/playlist?list=PL9HYL-VRXOoRupficE2l5DGgVIzpypTHs

1.8.3 freie Lehrbücher

Weitere freie R-Bücher

- *Einführung in R* von Ellis & Mayer
  https://methodenlehre.github.io/einfuehrung-in-R/
- *R für Psychos* von Burk & Anton
  https://r-intro.tadaa-data.de/book/

1.8.4 Internetforen

- *Deutsches* `R` *Forum*
  http://forum.r-statistik.de/



2 Statistik mit `R` und `RStudio`

Dieses Kapitel ist in folgende Bereiche unterteilt:

1. Deskriptive Statistik (ab Seite 212)
2. Schließende Statistik (ab Seite 223)

Zu Demonstrationszwecken werden wir in diesem Kapitel auf die Beispieldatensätze `epa`, `mma` , `nw` und `pf8` zurückgreifen, die Sie sich aus dem Internet in Ihre R-Session laden können.

```
# lade Datensatz "epa"
load(url("https://www.produnis.de/R/data/epa.RData"))
# lade Datensatz "mma"
load(url("https://www.produnis.de/R/data/mma.RData"))
# lade Datensatz "pf8"
load(url("https://www.produnis.de/R/data/pf8.RData"))
# lade Datensatz "nw"
load(url("https://www.produnis.de/R/data/nw.RData"))

# Wenn Sie das Paket 'jgsbook' installiert haben,
# können Sie auch wie folgt auf die Datensätze zugreifen
epa <- jgsbook::epa
mma <- jgsbook::mma
pf8 <- jgsbook::pf8
nw  <- jgsbook::nw
```

2.1 Deskriptive Statistik

Zur Beschreibung von Verteilungen eigenen sich Tabellen, Lage- und Streuungskenngrößen.

2.1.1 Häufigkeiten

Eine Häufigkeitstabelle wird mit der Funktion `table()` erstellt

```
# Daten erzeugen
x <- c(1, 3, 5, 7, 7, 7, 6, 34)

# Häufigkeitstabelle
table(x)
```

```
x
 1  3  5  6  7 34
 1  1  1  1  3  1
```

Für relative Häufigkeiten teilt man einfach durch die Länge der Variable `x`. So ändert sich der Befehl in:

```
# relative Häufigkeitstabelle
table(x)/length(x)
```

```
x
    1     3     5     6     7    34
0.125 0.125 0.125 0.125 0.375 0.125
```

Kumulierte Häufigkeiten erhält man mit der Funktion `cumsum()`

```
# kumulierte Häufigkeitstabelle
cumsum(table(x))
```

```
x
 1  3  5  6  7 34
 1  2  3  4  7  8
```

Und entsprechend auch kumulierte relative Häufigkeiten

```
# kumulierte relative Häufigkeitstabelle
cumsum(table(x)/length(x))
```

```
x
    1     3     5     6     7    34
0.125 0.250 0.375 0.500 0.875 1.000
```

Eine weitere Option bietet die Funktion `freq()` aus dem `rettyR`-Paket. Sie erstellt eine Tabelle mit absoluten und relativen Häufigkeiten sowie relativen Häufikeiten ohne fehlende Wert (`NA`).

```r
# Paket "prettyR" installieren
install.packages("prettyR", dependencies=TRUE)

# Paket laden
library(prettyR)

# Häufigkeitstabelle mit absoluten und relative Werten erstellen
freq(x)
```

```
Frequencies for x
        7    1    3    5    6   34   NA
        3    1    1    1    1    1    0
%    37.5 12.5 12.5 12.5 12.5 12.5    0
%!NA 37.5 12.5 12.5 12.5 12.5 12.5
```

Die Funktion `freqTable()` aus dem `jgsbook`-Paket erstellt für einen Vektor eine vollständige Häufigkeitstabelle mit absoluten, relativen und kumulierten Werten.

```r
# 234 Zufallswerte zwischen 21 und 30
x <- sample(21:30, 234, replace = TRUE)
# Häufigkeitstabelle
jgsbook::freqTable(x)
```

```
   Wert Haeufig Hkum Relativ   Rkum
1    21      19   19    8.12   8.12
2    22      18   37    7.69  15.81
3    23      30   67   12.82  28.63
4    24      23   90    9.83  38.46
5    25      24  114   10.26  48.72
6    26      36  150   15.38  64.10
7    27      15  165    6.41  70.51
8    28      24  189   10.26  80.77
9    29      23  212    9.83  90.60
10   30      22  234    9.40 100.00
```

2.1.2 Lagekenngrößen

Mit dem Befehl `summary()` erhält man eine erste Zusammenfassung der Werteverteilung mit den wichtigsten Lagekenngrößen.

```r
# Daten erzeugen
x <- c(1, 3, 5, 7, 7, 7, 6, 34)

# Zusammenfassung
summary(x)
```

```
   Min.   1st Qu.  Median  Mean   3rd Qu.  Max.
   1.00    4.50    6.50    8.75    7.00   34.00
```

Die Funktion `describe()` aus dem `psych`-Zusatzpaket liefert noch detailiertere Übersichten:

```
# Zusammenfassung
psych::describe(x)
```

```
   vars n mean    sd median trimmed  mad min max range
X1    1 8 8.75 10.43    6.5    8.75 1.48   1  34    33

skew kurtosis   se
1.69     1.35 3.69
```

Das Funktioniert auch bei kompletten Datensätzen.

```
# Zusammenfassung von pf8
psych::describe(pf8)
```

```
               vars   n   mean    sd median trimmed   mad min max
Standort*         1 731   2.85  1.61      2    2.81  1.48   1   5
Alter             2 730  37.00 18.00     28   34.98 12.60  15  86
Geschlecht*       3 725   1.61  0.49      2    1.63  0.00   1   3
Größe             4 730 174.04  9.62    173  173.84 10.38 151 206
Gewicht           5 720  75.53 16.88     73   73.93 16.31  45 175
Bildung*          6 506   4.88  1.77      5    4.98  2.97   1   7
Beruf*            7 731  40.42 37.04     34   38.58 48.93   1 104
Familienstand*    8 729   2.11  1.12      2    1.99  1.48   1   6
Kinder            9 727   0.38  0.79      0    0.18  0.00   0   5
Wohnort*         10 729   1.41  0.49      1    1.39  0.00   1   2
Rauchen*         11 729   1.21  0.40      1    1.13  0.00   1   2
SportHäufig      12 677   2.56  1.88      2    2.41  1.48   0  21
SportMinuten     13 667  62.42 40.14     60   60.47 44.48   0 400
SportWie*        14 656   1.64  0.69      2    1.55  1.48   1   3
SportWarum*      15 454   3.75  1.64      4    3.49  1.48   1   8
LebenZufrieden   16 724   7.91  1.59      8    8.07  1.48   1  11
               range  skew kurtosis   se
Standort*          4  0.28    -1.53 0.06
Alter             71  0.74    -0.70 0.67
Geschlecht*        2 -0.40    -1.75 0.02
Größe             55  0.26    -0.26 0.36
Gewicht          130  1.12     2.40 0.63
Bildung*           6 -0.39    -1.26 0.08
Beruf*           103  0.26    -1.55 1.37
Familienstand*     5  0.83     0.48 0.04
Kinder             5  2.19     4.45 0.03
Wohnort*           1  0.36    -1.87 0.02
Rauchen*           1  1.45     0.11 0.01
SportHäufig       21  1.91    12.75 0.07
SportMinuten     400  1.53     8.64 1.55
SportWie*          2  0.62    -0.77 0.03
SportWarum*        7  1.06     0.79 0.08
LebenZufrieden    10 -1.10     1.86 0.06
```

Die Lagewerte lassen sich auch einzeln bestimmen.

Mit der Funktion `min()` wird das Minimum der Verteilung ausgegeben.

```
# Minimum bestimmen
min(x)
```

```
[1] 1
```

Dementsprechend liefert `max()` das Maximum.

```
# Maximum bestimmen
max(x)
```

```
[1] 34
```

Die Spannweite wird mit `range()` abgefragt.

```
# Spannweite bestimmen
range(x)
```

```
[1]  1 34
```

Das arithmetische Mittel wird mit der Funktion `mean()` bestimmt

```
# x-quer bestimmen
mean(x)
```

```
[1] 8.75
```

Sind fehlende Werte in der Reihe enthalten, gibt R nur `NA` zurück

```
#  mean für "Gewicht" im Datensatz "pf8"
mean(pf8$Gewicht)
```

```
[1] NA
```

Beheben kann man dies, indem der Parameter `na.rm` auf `TRUE` gesetzt wird.

```
#  mean für "Gewicht" im Datensatz "pf8"
# entferne NAs
mean(pf8$Gewicht, na.rm=TRUE)
```

```
[1] 75.5325
```

Der Median wird mit der Funktion `median()` ermittelt.

```
# Median bestimmen
median(x)
```

```
[1] 6.5
```

Auch hier müssen enthaltene `NA`s ausgeblendet werden:

```r
# Median bestimmen, OHNE NAs
median(pf8$Gewicht, na.rm=TRUE)
```

```
[1] 73
```

In den R -Hausmitteln existiert keine eigene Funktion zur Bestimmung des Modalwertes. Wir können ihn daher "auf Umwegen" bestimmen. Wir lassen mittels `table()` eine Häufigkeitstabelle der Variable ausgeben. Diese sortieren wir mittels `sort()` absteigend. Der Modus ist dann der erste Wert der sortierten Tabelle.

```r
# Daten als Häufigkeitstabelle
table(x)

# absteigend sortieren
sort(table(x), decreasing=TRUE)

# Modalwert
sort(table(x), decreasing=TRUE)[1]
```

```
7
3
```

Es ist der Wert `7` und er kommt `3` mal vor.

Eine weitere Möglichkeit bietet das Zusatzpaket `statip`. Es enthält die Funktion `mfv()`, mit welcher der Modalwert bestimmt werden kann.

```r
# Daten als Häufigkeitstabelle
library(statip)

# Modalwert bestimmen
mfv(x)
# bzw. auch
statip::mfv(x)
```

```
[1] 7
```

Die vier Quartile werden mit der Funktion `quantile()` ermittelt (ja, der Befehl für Quartile heisst in R quantile()).

```r
# Quartile bestimmen, ohne NAs
quantile(x, na.rm=T)
```

```
  0%  25%  50%  75% 100%
 1.0  4.5  6.5  7.0 34.0
```

Es lassen sich beliebige Quantile berechnen.

```
# Quartile bestimmen, ohne NAs
quantile(x, probs = c(.333, .666))
```

```
33.3%   66.6%
5.331   7.000
```

Und auch die Art der Berechnung lässt sich über den Parameter `type` festlegen. Beispielsweise verwendet `type=6` die Methode von `SPSS`.

```
# Quartile bestimmen, ohne NAs
# so wie SPSS
quantile(x, probs = c(.333, .666), type=6)
```

```
33.3% 66.6%
4.994 7.000
```

Die Schiefe der Verteilung (Skewness) kann mittels `skew()` aus dem `psych`-Paket bestimmt werden. Ein positiver Wert bedeutet eine rechtsschiefe Verteilung, ein negativer Wert eine linksschiefe Verteilung.

```
psych::skew(x, type=2) # rechne wie SPSS oder SAS
```

```
[1] 2.577408
```

Ebenso enthält das Paket die Funktion `kurtosi()`, mit welcher die *Spitzigkeit* der Verteilung bestimmt werden kann.

```
psych::kurtosi(x, type=2) # rechne wie SPSS oder SAS
```

```
[1] 7.020852
```

2.1.3 Streuungskenngrößen

Die wichtigste Streuungskenngröße ist wahrscheinlich die Standardabweichung. Sie wird mittels der Funktion `sd()` bestimmt.

```
# Standardabweichung bestimmen
sd(x, na.rm=T)
```

```
[1] 10.43004
```

Die Varianz der Stichprobe wird mit `var()` berechnet.

```r
# Varianz von x bestimmen
var(x, na.rm=T)
```

```
[1] 108.7857
```

Der Interquartilsabstand (oder einfach nur Quartilsabstand) wird mit der Funktion `IQR()` bestimmt.

```r
# Quartilsabstand bestimmen
IQR(x, na.rm=T)
```

```
[1] 2.5
```

Ebenso wie bei `quantile()` lässt sich über den Parameter `type` die Berechnungsart ändern. Mit `type=6` nutzt `R` die Methode wie in `SPSS`:

```r
# Quartilsabstand bestimmen
# Methode wie in SPSS
IQR(x, na.rm=T, type=6)
```

```
[1] 3.5
```

2.1.4 Kreuztabellen

Mit der Funktion `table()` lassen sich auch Kreuztabellen erstellen, indem einfach beide Variablen übergeben werden.

```r
# erzeuge Daten
Punktwert <- c(2 , 12 , 7 , 15 , 10 , 4 , 13 , 9 , 16 , 19)
Geschlecht <- rep(c("maennlich" , "weiblich") ,5)

# Kreuztabelle
table(Punktwert, Geschlecht)
```

```
           Geschlecht
Punktwert maennlich weiblich
       2          1        0
       4          0        1
       7          1        0
       9          0        1
       10         1        0
       12         0        1
       13         1        0
       15         0        1
       16         1        0
       19         0        1
```

Ebenso steht die Funktion `xtabs()` zur Verfügung.

```
# Kreuztabelle mit "xtabs()"
xtabs(~ Punktwert + Geschlecht)
```

```
         Geschlecht
Punktwert maennlich weiblich
       2          1        0
       4          0        1
       7          1        0
       9          0        1
       10         1        0
       12         0        1
       13         1        0
       15         0        1
       16         1        0
       19         0        1
```

Siehe Kapitel 3.1.1 auf Seite 416 für weitere Beispiele zur Kreuztabelle.

2.1.5 z-Transformation

Die z-Transformation erfolgt mit der Funktion `scale()`.

```
# Wertereihe
Punktwerte <- c(2 , 12 , 7 , 15 , 10 , 4 , 13 , 9 , 16 , 19)

# z-Transformation
scale(Punktwerte)
```

```
            [,1]
 [1,] -1.6183421
 [2,]  0.2418212
 [3,] -0.6882604
 [4,]  0.7998702
 [5,] -0.1302114
 [6,] -1.2463094
 [7,]  0.4278376
 [8,] -0.3162278
```

```
 [9,]  0.9858866
[10,]  1.5439356
attr(,"scaled:center")
[1] 10.7
attr(,"scaled:scale")
[1] 5.375872
```

Hierbei berechnet `scale()` automatisch Mittelwert und Standardabweichung der übergebenen Werte. In der Ausgabe erkennen Sie diese unter `attr(,"scaled:center")` (in unserem Fall 10.7) und `attr(,"scaled:scale")` (in unserem Fall 5.375872).
Über die Parameter `center` und `scale` können Mittelwert und Standardabweichung aber auch beliebig spezifiziert werden.

```
1  scale(Punktwerte, center=15, scale=3)
```

```
            [,1]
 [1,] -4.3333333
 [2,] -1.0000000
 [3,] -2.6666667
 [4,]  0.0000000
 [5,] -1.6666667
 [6,] -3.6666667
 [7,] -0.6666667
 [8,] -2.0000000
 [9,]  0.3333333
[10,]  1.3333333
attr(,"scaled:center")
[1] 15
attr(,"scaled:scale")
[1] 3
```

Das Ergebnis ist ein `matrix`-Objekt. Wenn Sie nur den z-Wert als numerischen Wert ausgeben möchten, kann die `as.numeric()`-Funktion verwendet werden.

```
1  # gebe nur die z-Werte aus
2  as.numeric(scale(Punktwerte))
```

```
 [1] -1.6183421  0.2418212 -0.6882604  0.7998702 -0.1302114
 [6] -1.2463094  0.4278376 -0.3162278  0.9858866  1.5439356
```

2.1.6 Korrelation

Die allgemeine Funktion zur Berechnung von Korrelationen heisst `cor()`.

Die Korrelation nach Pearson berechnet sich mit der Funktion `cor()` und deren Parameter `method="pearson"`

```
1  # erzeuge Daten
2  Lesetest <- c(2 , 12 , 7 , 15 , 10 , 4 , 13 , 9 , 16 , 19)
3  Rechtschreibung <- c(3 , 14 , 9 , 17 , 12 , 4 , 16 , 12 , 18 , 20)
4
5  # Pearson Maßkorrelationskoeffizient
6  cor(Lesetest, Rechtschreibung, method="pearson")
```

```
[1] 0.9884616
```

Mit der Funktion `cor.test()` kann direkt ein Siginifkanztest mitgefahren werden

```
# Pearson Maßkorrelationskoeffizient
# mit Signifikanztest
cor.test(Lesetest, Rechtschreibung, method="pearson")
```

```
^^IPearson's product-moment correlation

data:  Lesetest and Rechtschreibung
t = 18.458, df = 8, p-value = 7.648e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9502085 0.9973659
sample estimates:
    cor
0.9884616
```

Sollten fehlende Wert (`NA`) enthalten sein, gibt der Befehl nur ein `NA` zurück.

```
# Pearson Maßkorrelationskoeffizient
# mit Datensatz "pf8", da sind NAs enthalten!!!
cor(pf8$Alter, pf8$Größe, method="pearson")
```

```
[1] NA
```

Mit dem Parameter `use="complete.obs"` (*nutze nur vollständige Beobachtungen*) können wir dies beheben.

```
# Pearson Maßkorrelationskoeffizient
# mit Datensatz "pf8", verwende nur "komplette" Daten (ohne NA)
cor(pf8$Alter, pf8$Größe, method="pearson", use="complete.obs")
```

```
[1] -0.1067484
```

Für die Korrelationsberechnung nach Spearmen (Spearman's ρ) wird der Parameter `method` auf `spearman` gesetzt

```
# Spearman's Korrelationsberechnung
cor(Lesetest, Rechtschreibung, method="spearman")
```

```
[1] 0.9969651
```

Auch hier müssen `NA`s mit dem Parameter `use="complete.obs"` unterdrückt werden

```r
# Spearman's Rho
# mit Datensatz "pf8", verwende nur "komplette" Daten (ohne NA)
cor(pf8$Alter, pf8$Größe, method="spearman", use="complete.obs")
```

```
[1] -0.05920716
```

Auch für Spearman's Rho kann ein Signifikanztest mitgefahren werden, indem `cor.test()` aufgerufen wird.

```r
# Spearman's Korrelationsberechnung
# mit Signifikanztest
cor.test(Lesetest, Rechtschreibung, method="spearman")
```

```
^^ISpearman's rank correlation rho

data:  Lesetest and Rechtschreibung
S = 0.50076, p-value = 3.698e-10
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9969651

Warnmeldung:
In cor.test.default(Lesetest, Rechtschreibung, method = "spearman") :
  Kann exakten p-Wert bei Bindungen nicht berechnen
```

Die Warnmeldung kann ignoriert werden, der p-Wert wird in der Ausgabe angezeigt.

Da `cor.test()` ein Objekt der Klasse `Liste` zurückliefert, kann auf die einzelnen Komponenten des Tests zugegriffen werden.

```r
# schreibe Testergebnis in Variable "test"
test <- cor.test(Lesetest, Rechtschreibung, method="spearman")

# zeige nur p-Wert an
test$p.value
```

```
[1] 3.698093e-10
```

```r
# zeige nur Teststatistik an
test$statistic
```

```
        S
0.5007599
```

```r
# zeige nur Spearmans's Rho an
test$estimate
```

```
      rho
0.9969651
```

2.2 Schließende Statistik

2.2.1 Regressionen

Die Funktion zur Berechnung linearer Regressionen ist `lm()` (für "linear model"). Sie folgt der Syntax *x erklärt durch y*, was durch eine Tilde ~ ausgedrückt wird.

```
# erzeuge Daten
Lesetest <- c(2 , 12 , 7 , 15 , 10 , 4 , 13 , 9 , 16 , 19)
Rechtschreibung <- c(3 , 14 , 9 , 17 , 12 , 4 , 16 , 12 , 18 , 20)

# lineare Regression "Rechtschreibung erklärt durch Lesetest"
lm(Rechtschreibung~Lesetest)
```

```
Call:
lm(formula = Rechtschreibung ~ Lesetest)

Coefficients:
(Intercept)     Lesetest
      1.208        1.055
```

Hierbei ist `Intercept` der Schnitt durch die Y-Achse.
Die Formel der Modellgeraden lautet also $y = 1,208 + 1,055 \cdot x$.

Wenn Sie über den Parameter `data` das Datenframe angeben, können Sie auf die Variablen referenzieren, ohne `$` verwenden zu müssen.

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/nw.RData"))

# entweder Variablen per "$" referenzieren
lm(nw$workyear ~ nw$worknight)

# oder Datenframe mit angeben
lm(workyear ~ worknight, data=nw)

# speichere Modell in "fit"
fit <- lm(workyear~worknight, data=nw)
# schaue mit "summary()" an
summary(fit)
```

Es ist üblich, Modelle in ein Objekt zu speichern und über die `summary()`-Funktion aufzurufen. Die Informationen der Modellzusammenfassung sind so detailierter.

```
# speichere Modell in "fit"
fit <- lm(workyear~worknight, data=nw)
# schaue mit "summary()" an
summary(fit)
```

```
Call:
lm(formula = workyear ~ worknight, data = nw)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-12.889  -5.743  -2.351   3.660  26.427

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.91425    0.74088   13.38   <2e-16 ***
worknight    0.82912    0.05958   13.91   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.463 on 274 degrees of freedom
Multiple R-squared:  0.4141,^^IAdjusted R-squared:  0.4119
F-statistic: 193.6 on 1 and 274 DF,  p-value: < 2.2e-16
```

Außerdem lässt sich das Modell so leicht plotten.

```
1  # plotte das Modell
2  plot(fit)
```

Dies erzeugt insgesamt 4 Plots (Residualplot, Q-Q, Scale-Location und Einfluss). Durch drücken der `[Enter]`-Taste schalten Sie zum nächsten Plot weiter.
Sie können aber auch alle 4 Plots in einem ausgeben lassen.

```
1  # plotte alle 4 auf einer Seite
2  par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
3  plot(fit)
```

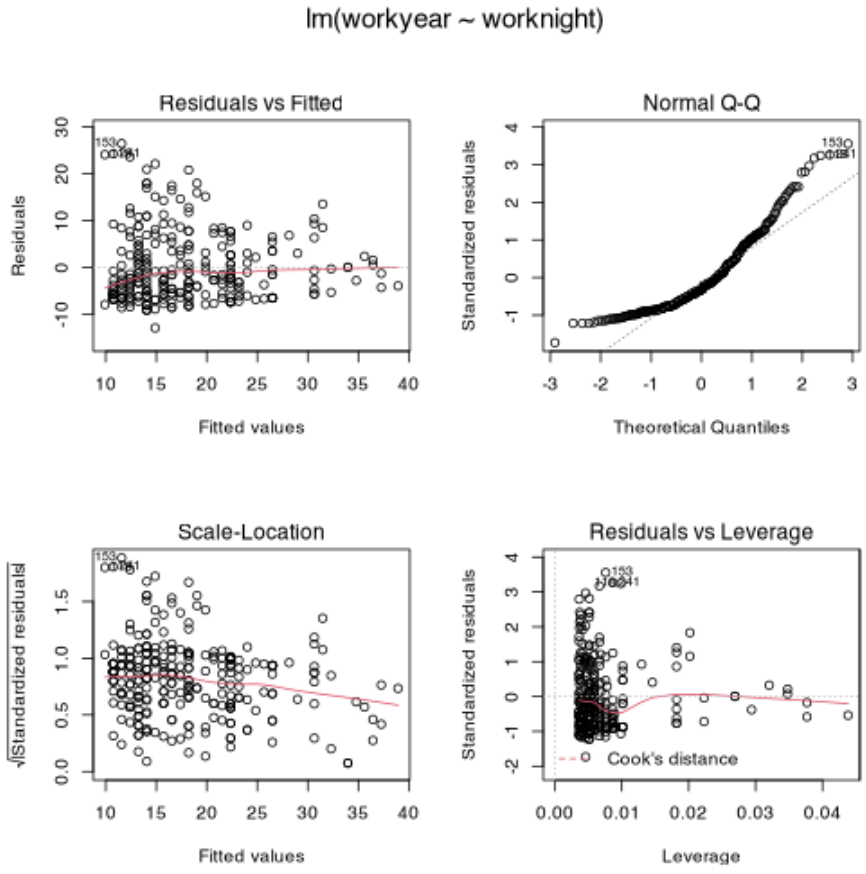


Abb. 2.1

Denken Sie aber daran, anschließend die Plotausgabe wieder zurückzusetzen:

```
par(mfrow = c(1, 1))
```

Die Regressionsgerade lässt sich mittels `abline()` der Punktwolke hinzufügen. Achten Sie dabei auf die jeweils korrekte Zuordnung von `x` und `y`. Beim Plotten muss `x, y` verwendet werden, bei der Regressiongerade ist es `y ~ x`.

```
# so ist es richtig
plot(nw$worknight, nw$workyear)
abline(lm(nw$workyear ~ nw$worknight))
```

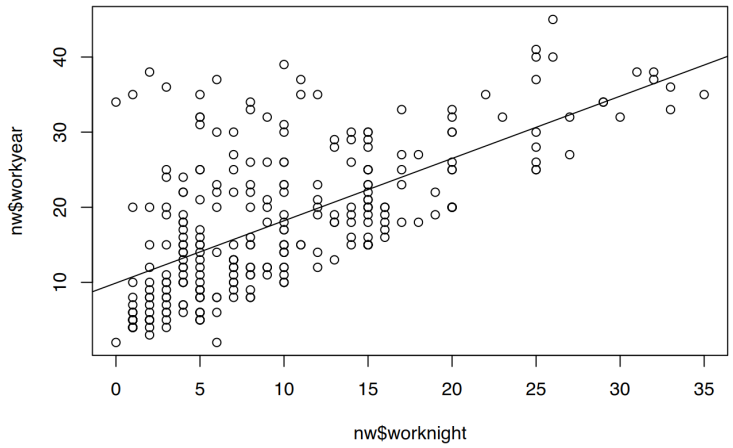


Abb. 2.2

```
# so ist es FALSCH
plot(nw$workyear, nw$worknight)
abline(lm(nw$workyear ~ nw$worknight), col="red")
```

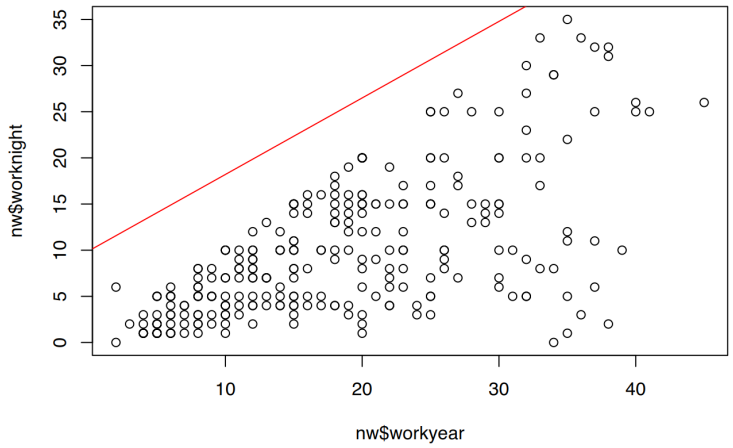


Abb. 2.3

Mit der Funktion `predict()` lassen sich nun für jedes `x` die entsprechenden `y`-Werte des Modells bestimmen (vorhersagen). Der Funktion wird das gefittete Objekt übergeben, sowie eine Liste der gewünschten Prädiktorenwerte.

```
# erstelle Modell
fit <- lm(workyear ~ worknight, data=nw)
# sage workyear für worknight=7 vorher
predict(fit, list(worknight=7))

```

```
  1
  15.71809
```

Für `worknight=7` wird `workyear=15` vorhergesagt.

2.2.1.1 multiple lineare Regressionen

Multiple lineare Regressionen werden ebenfalls mit der Funktion `lm()` berechnet. Hierbei werden die erklärenden Variablen per + Zeichen aneinandergereiht.

```
# multiple lineare Regression
# "workyear" erklärt durch "worknight" UND "age"
fit <- lm(workyear ~ worknight+age, data=nw)
summary(fit)#
```

```
Call:
lm(formula = workyear ~ worknight + age, data = nw)

Residuals:
     Min       1Q   Median       3Q      Max
-14.7077  -3.9748  -0.3066   3.5936  19.9569

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.99856    1.72557  -3.476 0.000591 ***
worknight    0.57707    0.05714  10.100  < 2e-16 ***
age          0.41614    0.04195   9.921  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.41 on 273 degrees of freedom
Multiple R-squared:  0.5693,^^IAdjusted R-squared:  0.5662
F-statistic: 180.5 on 2 and 273 DF,  p-value: < 2.2e-16
```

Ebenfalls können hier die 4 Plots erzeugt werden.

```
# plotte alle 4 auf einer Seite
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
plot(fit)
par(mfrow = c(1, 1))
```

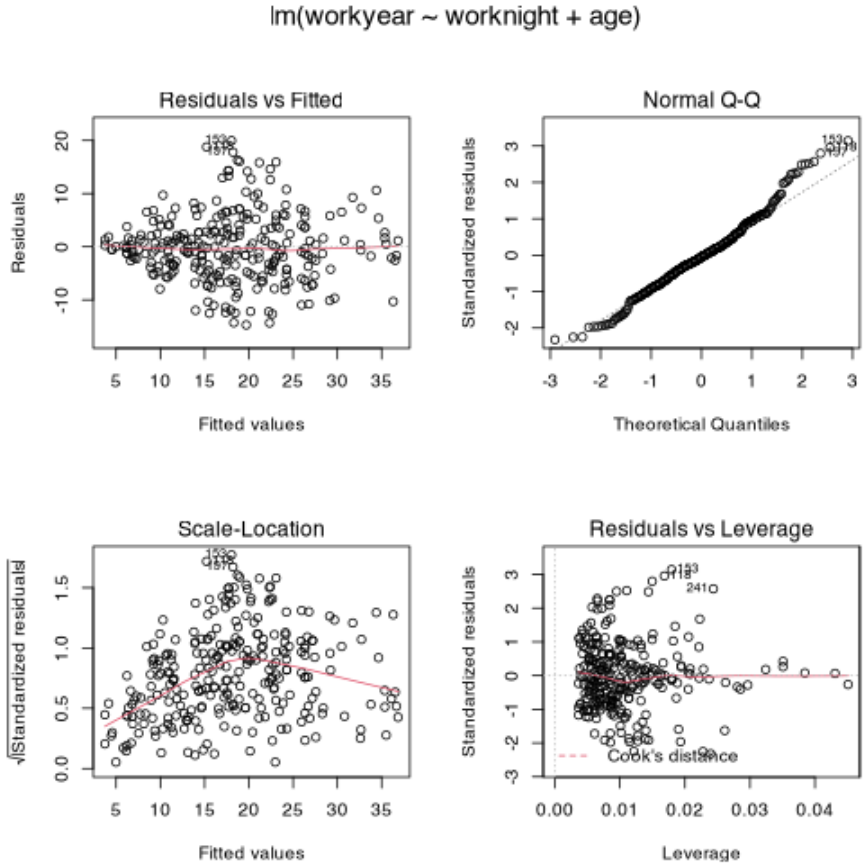


Abb. 2.4

2.2.1.2 nicht-lineare Regression

Häufig folgen die vorhandenen Daten keinem linearen Zusammenhang.

```
# erzeuge dummy-Werte
x <- c(6, 9, 12, 14, 30, 35, 40, 47, 51, 55, 60)
y <- c(14, 28, 50, 70, 89, 94, 90, 75, 59, 44, 27)

# Plotte die Daten mit Regressionsgeraden
plot(x, y, col="blue")
abline(lm(y~x), col="red", lty=3)
```

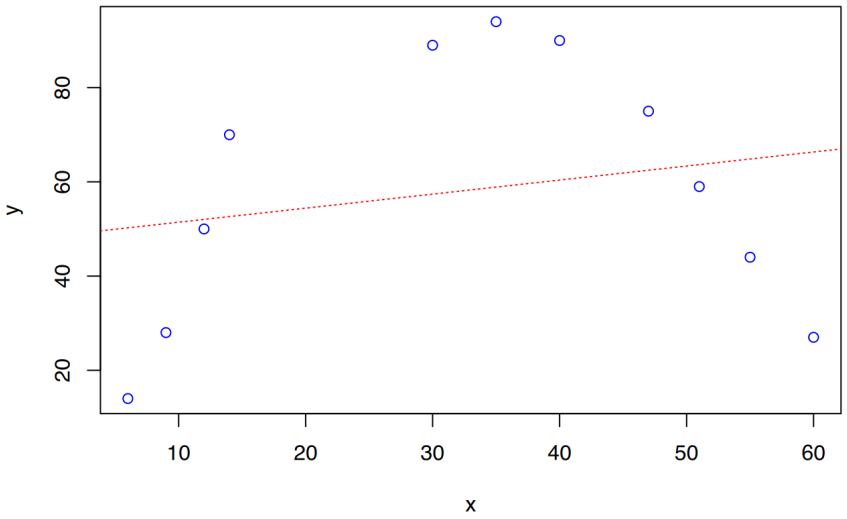


Abb. 2.5

Das Aussehen des Plots lässt schon erahnen, dass kein linearer Zusammenhang vorliegt. Wenn wir ein lineares Modell auf die Daten fitten, ist R^2 sehr niedrig.

```
# Modell fitten
fit <- lm(y~x)
# Modelldaten ausgeben
summary(fit)
```

```
Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max
-39.34 -21.99  -2.03  23.50  35.11

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  48.4531    17.3288   2.796   0.0208 *
x             0.2981     0.4599   0.648   0.5331
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 28.72 on 9 degrees of freedom
Multiple R-squared:  0.0446,^^IAdjusted R-squared:  -0.06156
F-statistic: 0.4201 on 1 and 9 DF,  p-value: 0.5331
```

R^2 ist hier 0,0446, was bedeutet, dass nur 4% der Daten durch das Modell erklärt werden können.
Das Plot legt nahe, dass ein quadratischer Zusammenhang besteht, denn die Punkte scheinen einer Parabel zu folgen. Die Formel einer quadratischen Funktion lautet vereinfacht $y = x^2 + x$. Den rechten Teil der Formel müssen wir innerhalb von `lm()` verwenden. Wir können die Formel nicht *direkt* in `lm()` schreiben, weil innerhalb von `lm()` nach dem ~-Zeichen keine weiteren Rechenoperationen möglich sind. Das liegt daran, dass die Prädiktoren mittels + und - übergeben werden. Unser Trick besteht nun darin, die Formel in die `I()`-Funktion einzupacken::

```r
# Modell erstellen mit quadratischer Formel
fit <- lm(y ~ x + I(x^2))
# Modellwerte ausgeben
summary(fit)
```

```
Call:
lm(formula = y ~ x + I(x^2))

Residuals:
    Min      1Q  Median      3Q     Max
-6.2484 -3.7429 -0.1812  1.1464 13.6678

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -18.25364    6.18507  -2.951   0.0184 *
x             6.74436    0.48551  13.891 6.98e-07 ***
I(x^2)       -0.10120    0.00746 -13.565 8.38e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.218 on 8 degrees of freedom
Multiple R-squared:  0.9602,^^IAdjusted R-squared:  0.9502
F-statistic: 96.49 on 2 and 8 DF,  p-value: 2.51e-06
```

R^2 liegt nun bei 0,9602, was bedeutet, dass 96% der Daten durch das Modell erklärt werden können. Die Funktionsformel lautet $y = -0,10120 \cdot x^2 + 6,74436 \cdot x - 18,25364$.
Mit der Funktion `predict()` lassen sich nun für jedes `x` die entsprechenden `y`-Werte des Modells bestimmen (vorhersagen). Der Funktion wird das gefittete Objekt übergeben, sowie eine Liste der gewünschten Prädiktorenwerte. Die so erzeugten Modellwerte können dann ins Plot gezeichnet werden.

```r
vorhersage <- predict(fit, list(x=x))
plot(x, y, col="blue")
lines(x, vorhersage, col='darkgreen')
```

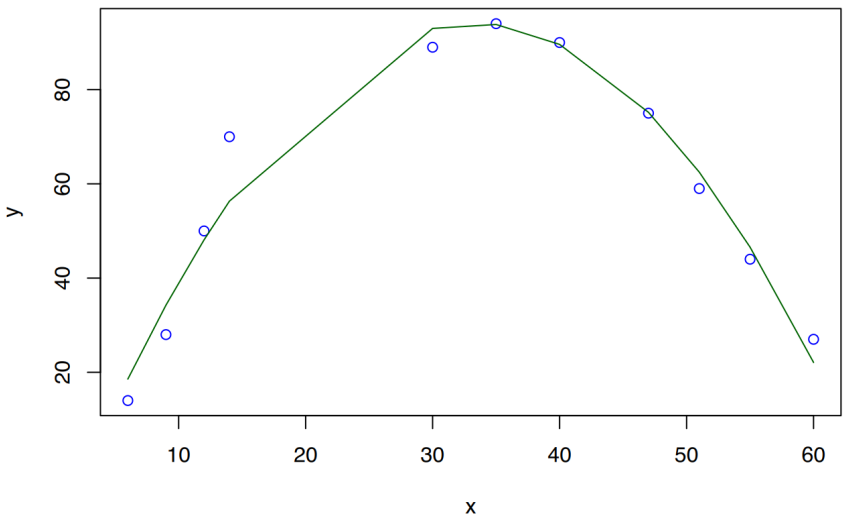


Abb. 2.6

Um eine *schöne* Parabel zu zeichnen, haben wir zu wenige `x`-Werte. Daher erstellen wir einen neuen Vektor, der kontinuierliche Werte für `x` enthält. Mit diesem können wir dann die entsprechenden (vielen) `y`-Werte mittels `predict()` bestimmen und in das Plot einzeichnen. So entsteht - bei genügend Werten - eine schöne Modellparabel.

```
vorhersage <- predict(fit, list(x, x2))
plot(x, y, col="blue")
lines(x, vorhersage, col='darkgreen')
# falsche Regressionsgerade
abline(lm(y~x), col="red", lty=3)
```

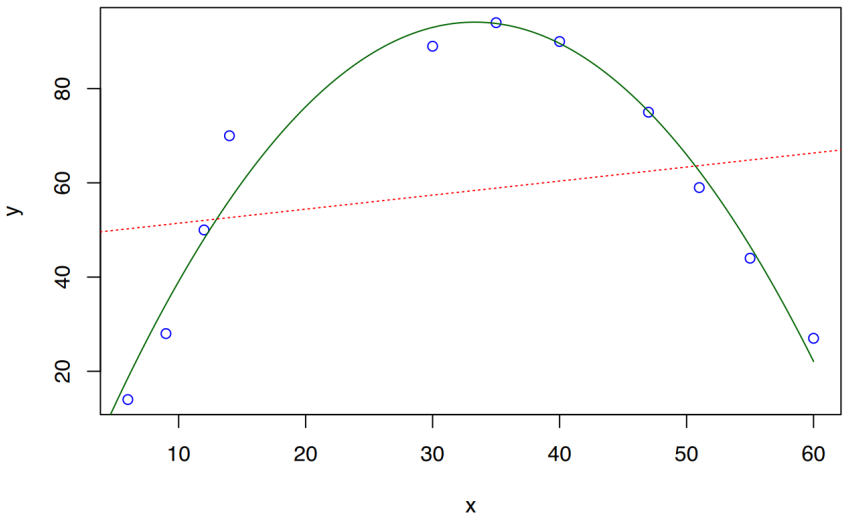


Abb. 2.7

2.2.1.2.1 Funktionen höherer Ordnung

Wird ein Zusammenhang mit einer Funktion höherer Ordnung vermutet, gehen wir analog zur quadratischen Funktion vor. In diesem Beispiel nehmen wir an, der Zusammenhang ließe sich vereinfacht durch die Formel $y = x^3 + x^2 + x$ beschreiben. Die Daten verhalten sich wie folgt:

```
# erzeuge Testwerte
x <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5)
y <- c(1, 46, 71, 82,  82,  86,  91, 106, 137,  17,  51,  68, 77,  76,  82.1,  87, 108)

# plotten
plot(x, y, col="blue")

# falsche Regressionsgerade
abline(lm(y~x), col="red", lty=3)
```

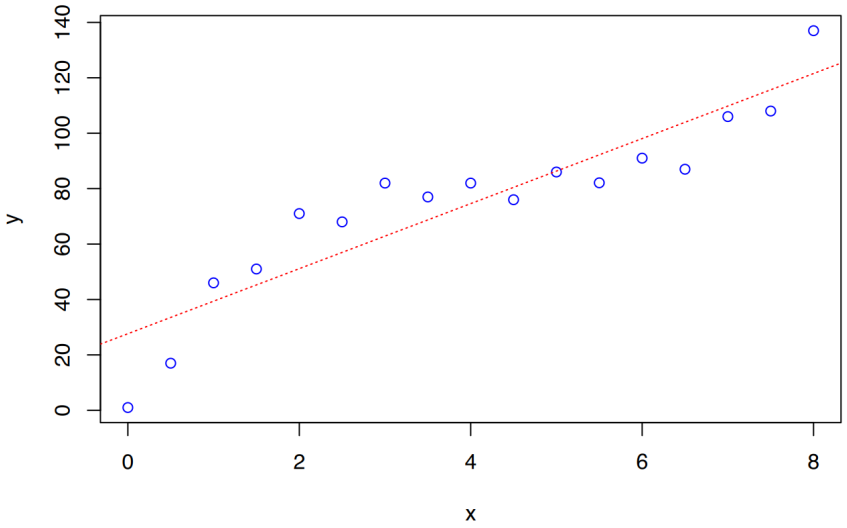


Abb. 2.8

```
Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-26.6471  -7.6728  -0.3309   8.2743  19.8794

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   27.647      6.387   4.329 0.000596 ***
x             11.737      1.362   8.619  3.4e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13.75 on 15 degrees of freedom
Multiple R-squared:  0.832,^^IAdjusted R-squared:  0.8208
F-statistic: 74.29 on 1 and 15 DF,  p-value: 3.398e-07
```

Die Regressionsgerade beschreibt die Daten nicht gut, aber das R^2 des linearen Modells ist mit 0,832 recht hoch. Das Modell beschreibt also 83% der vorhandenen Daten.
Versuchen wir nun, die Funktion dritten Grades ins Modell zu übernehmen. Wie bei den quadratischen Funktionen müssen wir die Werte für x^3 und x^2 in `I()`-Funktionen einwickeln, die wir dann als Prädiktoren innerhalb von `lm()` übergeben.

```
1  # Modell fitten
2  fit <- lm(y ~ x + I(x^2) + I(x^3))
3  summary(fit)
```

```
Call:
lm(formula = y ~ x + I(x^2) + I(x^3))

Residuals:
   Min     1Q Median     3Q    Max
```

```
-7.038 -4.905  1.872  4.254  5.005

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.0945     4.0612  -0.516    0.615
x            55.5108     4.5344  12.242 1.64e-08 ***
I(x^2)      -12.6470     1.3414  -9.428 3.55e-07 ***
I(x^3)        0.9771     0.1101   8.876 7.04e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.14 on 13 degrees of freedom
Multiple R-squared:  0.9797,^^IAdjusted R-squared:  0.975
F-statistic: 208.8 on 3 and 13 DF,  p-value: 3.038e-11
```

Das R^2 liegt bei 0,9797, was bedeutet, dass knapp 98% der Daten durch unser Modell erklärt werden können. Dieses Modell ist also wesentlich besser als das zuvor berechnete lineare Modell. Die Modellformel lautet $y = 0,9772 \cdot x^3 - 12,647 \cdot x^2 + 55,5108 \cdot x - 2,0945$.
Um die Funktionskurve des Modells in das Plot zu zeichnen, erstellen wir zunächst einen neuen Vektor, der kontinuierliche Werte für `x` enthält. Über die Funktion `predict()` können dann die Modellwerte für `y` berechnet werden.

```
1  # Erstelle kontinuierliche Werte
2  # für x von 0 bis 8
3  xref <- seq(0, 8, 0.1)
4  # berechne dazugehörige y-Modellwerte
5  yref <- predict(fit, list(x = xref))
6
7  # Plotten
8  plot(x, y, col="blue")
9  # Modell-Parabel einzeichnen
10 lines(xref, yref, col='darkgreen')
11 abline(lm(y~x), col="red", lty=3 )
```

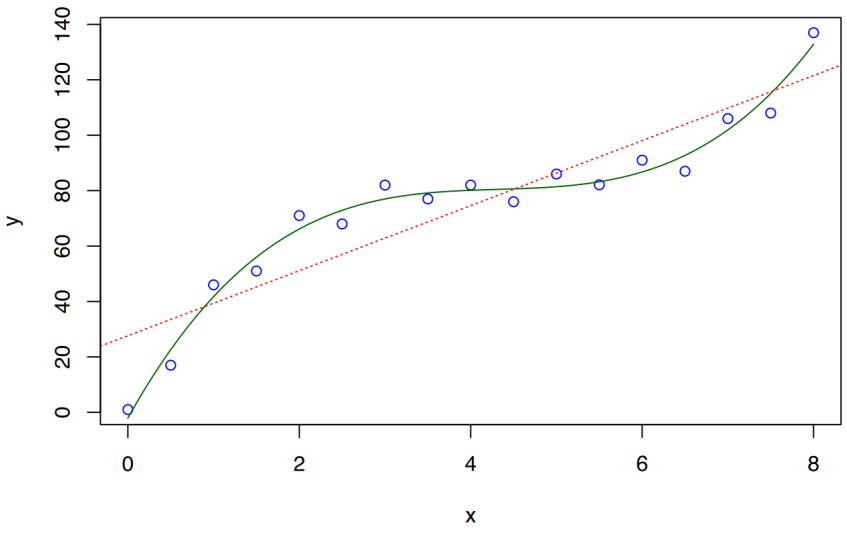


Abb. 2.9

2.2.1.2.2 Exponentialfunktion

Wird ein exponentieller Zusammenhang vermutet, gehen wir analog zur quadratischen Regression vor.
In diesem Beispiel nehmen wir an, der Zusammenhang ließe sich vereinfacht durch die Formel $y = a^x$ beschreiben. Die Daten verhalten sich wie folgt:

```
# Testdaten
x <- c(2, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16, 17, 18, 20, 21, 22, 22, 23, 24, 24, 25)
y <- c(1, 2, 2, 1, 3, 3, 2, 3, 2, 5, 4, 7, 10, 7, 15, 12, 18, 20, 17, 19, 20)

# plotten
plot(x, y, col="blue")
# falsche Regressionsgerade
abline(lm(y~x), col="red", lty=3)
summary(lm(y~x))
```

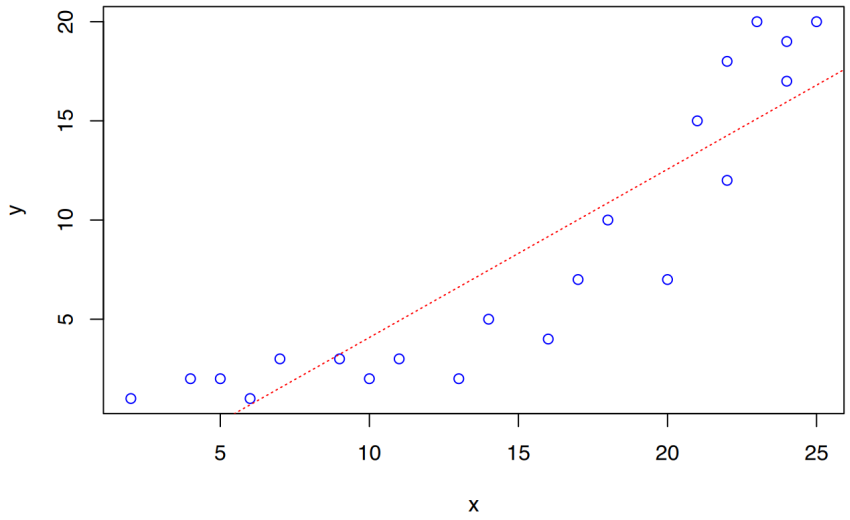


Abb. 2.10

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-5.5601 -2.2566  0.3153  3.0118  4.8952

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.40475    1.60736  -2.740    0.013 *
x            0.84824    0.09688   8.756 4.27e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.236 on 19 degrees of freedom
Multiple R-squared:  0.8014,^^IAdjusted R-squared:  0.7909
F-statistic: 76.67 on 1 and 19 DF,  p-value: 4.275e-08
```

Die Regressionsgerade beschreibt die Daten nicht gut, aber das R^2 des linearen Modells ist mit 0,8014 recht hoch. Das Modell beschreibt also 80% der vorhandenen Daten.
Versuchen wir nun, die Exponentialfunktion ins Modell zu übernehmen. Hierfür fitten wir das Modell nicht auf `y`, sondern auf den natürlichen Logarithmus yon `y`.

```
# Modell fitten
fit <- lm(log(y) ~ x)
summary(fit)
```

```
Call:
lm(formula = log(y) ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73254 -0.10440  0.01856  0.24804  0.44867

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.25510    0.16442  -1.551    0.137
x            0.12929    0.00991  13.047 6.23e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.331 on 19 degrees of freedom
Multiple R-squared:  0.8996,^^IAdjusted R-squared:  0.8943
F-statistic: 170.2 on 1 and 19 DF,  p-value: 6.232e-11
```

Mit $R^2 = 0{,}8996$ beschreibt das Modell die Daten besser, als das zuvor erstellte lineare Modell. Die Funktionsformel lautet $ln(y) = 0,12929 \cdot x - 0,2551$.
Um die Funktionskurve des Modells in das Plot zu zeichnen, erstellen wir zunächst einen neuen Vektor, der kontinuierliche Werte für `x` enthält. Über die Funktion `predict()` können dann die Modellwerte für `y` berechnet werden. Da wir das Modell auf `log(y)` gefittet haben, müssen die Werte von `predict()` mittels `exp()` umgewandelt werden.

```
# Erstelle kontinuierliche Werte
# für x von 0 bis 8
xref <- seq(0, 25, 0.1)
# berechne dazugehörige y-Modellwerte
yref <- exp(predict(fit, list(x = xref)))

# Plotten
plot(x, y, col="blue")
# Modell-Parabel einzeichnen
lines(xref, yref, col='darkgreen')
abline(lm(y~x), col="red", lty=3 )
```

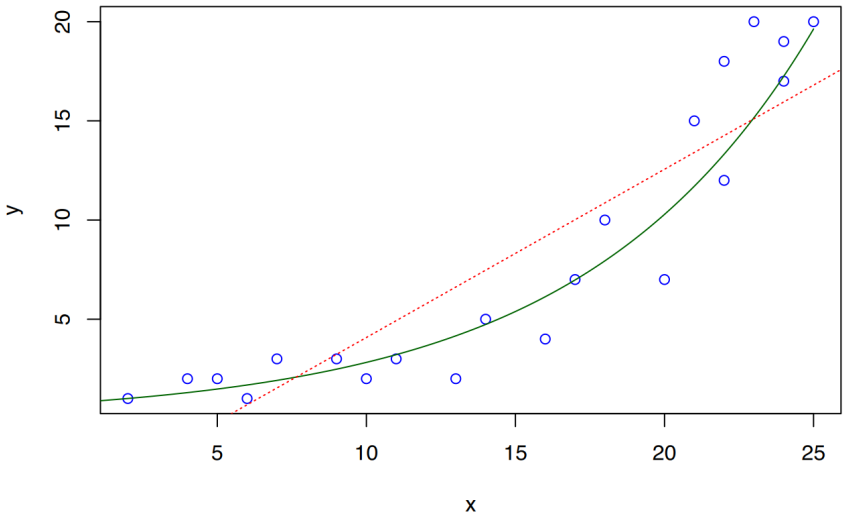


Abb. 2.11

2.2.1.2.3 Sigmoidalfunktion

Auch für sigmoidale Verteilungen können wir ähnlich wie in den vorherigen Kapiteln vorgehen. Als Beispieldatensatz verwenden wir die (ausgedachte) Fangwahrscheinlichkeiten für Pokémons in Abhängigkeit zu ihrer Lebensenergie.

```
# laden Daten
load(url("https://www.produnis.de/R/data/pokemon.RData"))
# anschauen
summary(pokemon)
```

```
 remaining_hp      catch_prob
Min.   :  1.00   Min.   :0.08607
1st Qu.: 25.75   1st Qu.:0.14526
Median : 50.50   Median :0.16318
Mean   : 50.50   Mean   :0.19460
3rd Qu.: 75.25   3rd Qu.:0.18905
Max.   :100.00   Max.   :0.99999
```

```
# Plot anschauen
plot(pokemon$remaining_hp, pokemon$catch_prob)
```

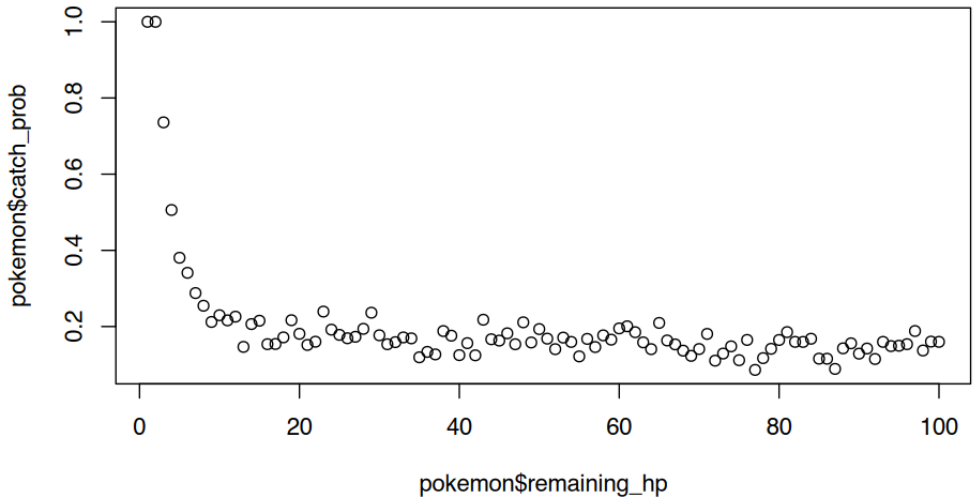


Abb. 2.12

Die Punktwole legt ein sigmoidales Modell nahe, welches wir nun fitten möchten.

```r
# Modell "Fangwahrscheinlichkeit erklärt durch Level"
fit <- lm(log(catch_prob) ~ I(1/remaining_hp), data=pokemon)

# anschauen
summary(fit)
```

```
Call:
lm(formula = log(catch_prob) ~ I(1/remaining_hp), data = pokemon)

Residuals:
     Min       1Q   Median       3Q      Max
-0.83416 -0.11957  0.02162  0.12840  0.67374

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       -1.88747    0.02514  -75.08   <2e-16 ***
I(1/remaining_hp)  2.72162    0.19661   13.84   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2298 on 98 degrees of freedom
Multiple R-squared:  0.6616,^^IAdjusted R-squared:  0.6582
F-statistic: 191.6 on 1 and 98 DF,  p-value: < 2.2e-16
```

Mit dem Modell können wir nun Werte vorhersagen. Hierbei muss die `predict()`-Funktion in die `exp()`-Funktion gewickelt werden, um den Logarithmus des Modells zu neutralisieren. Anschließend können wir die Modelllinie plotten.

```r
# vorhersage für alle Levels von 1 bis 100
px = 1:100
py = exp(predict(fit, list(remaining_hp=1:100)))

# plotten
plot(pokemon$remaining_hp, pokemon$catch_prob)
# Modelllinie hinzufügen
lines(px, py, col="cyan")
```

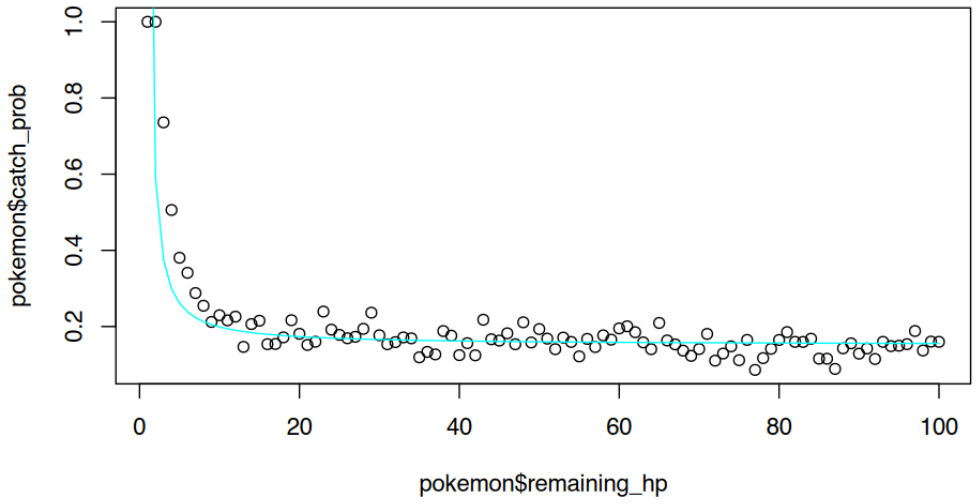


Abb. 2.13

2.2.2 Generalisiert lineare Modelle

Mit der Funktion `glm()` können generalisierte lineare Modelle gebildet werden. Strenggenommen können *alle* Regressionsmodelle mit `glm()` erzeugt werden. In R wird sie vor allem verwendet für

1. Poisson-Regressionen
2. logistische Regressionen
3. binomiale Regressionen

Das Standardvorgehen ist in allen Fällen gleich, das gefittete Modell wird in ein Objekt geschrieben

```
1  fit <- glm(ZIEL ~ Prädiktor1 + Prädiktor2, family= gaussian)
```

wobei über den Parameter `family` die Art des Modells festgelgt wird (z.B. `binomial`, `poisson` oder `gaussian`).

```
1  # Modellübersicht
2  summary(fit)
3
4  # Koeffizientenwerte
5  coef(fit)
6  #oder
7  fit$coefficients
8
9  # Konfidenzintervalle der Koeffizienten
10 confint(fit)
11
12 # Residuen / Devianz
```

```r
residuals(fit, type="deviance")

# Vorhersagewerte
predict(fit, type="response")
```

2.2.2.1 lineare Regression

Für lineare Regressionen haben wir die Funktion `lm()` verwendet. Tatsächlich könnten wir die Modelle auch per `gml()` erstellen. Kommen wir zurück zu dem Beispiel des Lese- und Rechtschreibetests.

```r
# erzeuge Daten
Lesetest <- c(2 , 12 , 7 , 15 , 10 , 4 , 13 , 9 , 16 , 19)
Rechtschreibung <- c(3 , 14 , 9 , 17 , 12 , 4 , 16 , 12 , 18 , 20)

# lineare Regression "Rechtschreibung erklärt durch Lesetest"
summary(lm(Rechtschreibung~Lesetest))
```

```
Call:
lm(formula = Rechtschreibung ~ Lesetest)

Residuals:
     Min       1Q   Median       3Q      Max
-1.42907 -0.26211  0.04498  0.36332  1.29412

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.20761    0.67774   1.782    0.113
Lesetest     1.05536    0.05718  18.458 7.65e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9221 on 8 degrees of freedom
Multiple R-squared:  0.9771,^^IAdjusted R-squared:  0.9742
F-statistic: 340.7 on 1 and 8 DF,  p-value: 7.648e-08
```

Der Aufruf in `glm()` erfolgt analog, wobei wir den Parameter `family` auf `gaussian` stellen müssen.

```r
fit <- glm(Rechtschreibung~Lesetest, family = gaussian)
summary(fit)
```

```
Call:
glm(formula = Rechtschreibung ~ Lesetest, family = gaussian)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.20761    0.67774   1.782    0.113
Lesetest     1.05536    0.05718  18.458 7.65e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.850346)
```

```
    Null deviance: 296.5000  on 9  degrees of freedom
Residual deviance:   6.8028  on 8  degrees of freedom
AIC: 30.526

Number of Fisher Scoring iterations: 2
```

Die errechneten Parameter stimmen mit `lm()` überein. Es fällt auf, dass `glm()` keinen Wert für R^2 zeigt, dafür aber einen AIC-Wert liefert.

Eine alternative Modellübersicht bietet die Funktion `summ()` (mit 2 m) aus dem Paket `jtools`.

```
jtools::summ(fit, digits=4)
```

```
MODEL INFO:
Observations: 10
Dependent Variable: Rechtschreibung
Type: Linear regression

MODEL FIT:
Chi²(1) = 289.6972, p = 0.0000
Pseudo-R² (Cragg-Uhler) = 0.9790
Pseudo-R² (McFadden) = 0.6062
AIC = 30.5262, BIC = 31.4340

Standard errors: MLE
-------------------------------------------------------
                      Est.     S.E.    t val.        p
----------------- -------- -------- --------- --------
(Intercept)         1.2076   0.6777    1.7818   0.1126
Lesetest            1.0554   0.0572   18.4576   0.0000
-------------------------------------------------------

Estimated dispersion parameter = 0.8503
```

2.2.3 Poissonregression

Die Poisson-Regression wird beispielsweise verwendet, um die Anzahl an Ereignissen (Kunden in der Schlange, Patienten in der Notaufnahme) auf ihre Abhängigkeit zu bestimmten Prädiktoren zu untersuchen.
Wir verwenden den Testdatensatz "`Notaufnahme.txt`", der als Texttabelle im Internet verfügbar ist.

```
df <- read.table(url("https://www.produnis.de/R/data/Notaufnahme.txt"),
                 header=TRUE, sep=",")
```

Er besteht aus 350 Beobachtungen der Patientenanzahl pro Tag in einer Notaufnahme. Zusätzlich wurden in der Spalte `Wochentag` die Wochentage erhoben. In Spalte `Event` wurde dichtotom erfasst, ob eine "größere" Veranstaltung in der Stadt stattgefunden hat.

```
head(df)
```

```
  Patienten Wochentag Event
1         8        Mo  nein
2         9        Di  nein
3        14        Mi  nein
4        10        Do  nein
5        10        Fr  nein
6        15        Mo  nein
```

Untersuchen wir nun, inwieweit der Wochentag und das Stattfinden einer größeren Veranstaltung die Anzahl der Notfälle beeinflusst. Das Modell kann wie gewohnt mit `glm()` erstellt werden, wobei wir den Parameter `family = poisson` setzen müssen.

```
# Poissonregressionsmodell erzeugen
fit <- glm(Patienten ~ Wochentag + Event, data = df, family = poisson)

# Zusammenfassung des Modells
summary(fit)
```

```
Call:
glm(formula = Patienten ~ Wochentag + Event, family = poisson,
    data = df)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.528011   0.050672  49.889  < 2e-16 ***
WochentagDo  0.071575   0.057063   1.254   0.2097
WochentagFr  0.050956   0.057349   0.889   0.3743
WochentagMi  0.119094   0.056421   2.111   0.0348 *
WochentagMo  0.003367   0.058026   0.058   0.9537
WochentagSa  0.429869   0.065146   6.599 4.15e-11 ***
WochentagSo  0.365318   0.066489   5.494 3.92e-08 ***
Eventnein   -0.292176   0.037640  -7.762 8.33e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 512.02  on 349  degrees of freedom
Residual deviance: 345.57  on 342  degrees of freedom
AIC: 1830.7

Number of Fisher Scoring iterations: 4
```

Das Modell zeigt an, dass die Patientenzahl signifikant an den Wochentagen `Sa` und `So` erhöht ist. Auch der Prädiktor `Event` ist signifikant. Der negative Prädiktorenwert bei `Eventnein` gibt an, dass sich die Patientenzahl verringert, wenn kein Event stattfindet. In der Modellzusammenfassung ist `Event=ja` der Bezugswert, so dass der Schätzwerte für `Event=nein` ausgegeben wird. Schöner anzusehen wäre es genau andersherum. Um `Event=nein` als Basiswert zu setzen, und den Schätzwert für `Event=ja` anzuzeigen, können wir die Events in einen Factor umwandeln, und dann mittels `relevel()` den Wert `nein` als Bezugspunkt setzen.

```
# Wir wollen Event="nein" als Basis
df$Event <- factor(df$Event)
df$Event <- relevel(df$Event, "nein")

# Poissonregressionsmodell neu erzeugen
fit <- glm(Patienten ~ Wochentag + Event, data = df, family = poisson)
```

```
7
8  # Zusammenfassung des Modells
9  summary(fit)
```

```
Call:
glm(formula = Patienten ~ Wochentag + Event, family = poisson,
    data = df)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 2.235835   0.041828  53.453  < 2e-16 ***
WochentagDo 0.071575   0.057063   1.254   0.2097
WochentagFr 0.050956   0.057349   0.889   0.3743
WochentagMi 0.119094   0.056421   2.111   0.0348 *
WochentagMo 0.003367   0.058026   0.058   0.9537
WochentagSa 0.429869   0.065146   6.599 4.15e-11 ***
WochentagSo 0.365318   0.066489   5.494 3.92e-08 ***
Eventja     0.292176   0.037640   7.762 8.33e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 512.02  on 349  degrees of freedom
Residual deviance: 345.57  on 342  degrees of freedom
AIC: 1830.7

Number of Fisher Scoring iterations: 4
```

Eine alternative Übersicht bietet die Funktion `summ()` (mit 2 m) aus dem Paket `jtools`.

```
1  jtools::summ(fit, digits=4)
```

```
MODEL INFO:
Observations: 350
Dependent Variable: Patienten
Type: Generalized linear model
  Family: poisson
  Link function: log

MODEL FIT:
Chi²(7) = 166.4510, p = 0.0000
Pseudo-R² (Cragg-Uhler) = 0.3798
Pseudo-R² (McFadden) = 0.0840
AIC = 1830.7077, BIC = 1861.5712

Standard errors: MLE
-------------------------------------------------------
                      Est.     S.E.    z val.        p
----------------- -------- -------- --------- --------
(Intercept)         2.2358   0.0418   53.4531   0.0000
WochentagDo         0.0716   0.0571    1.2543   0.2097
WochentagFr         0.0510   0.0573    0.8885   0.3743
WochentagMi         0.1191   0.0564    2.1108   0.0348
WochentagMo         0.0034   0.0580    0.0580   0.9537
WochentagSa         0.4299   0.0651    6.5986   0.0000
WochentagSo         0.3653   0.0665    5.4944   0.0000
Eventja             0.2922   0.0376    7.7624   0.0000
-------------------------------------------------------
```

Die Werte der Koeffizienten sind als Log-Odds dargestellt. Eine bessere Lese- und Interpretierbarkeit bietet die Darstellung als *Odds-Ratio*. Hierfür muss der Logarithmus mittels `exp()` aufgehoben werden.

```
# Zeige die Koeffizienten als Odds-Ratio
exp(fit$coefficients)
```

```
(Intercept) WochentagDo WochentagFr WochentagMi WochentagMo WochentagSa WochentagSo     Eventja
   9.354288    1.074199    1.052277    1.126476    1.003373    1.537056    1.440972    1.339339
```

Die Konfidenzintervalle der Prädiktorenwerte erhalten wir mit der Funktion `confint()`.

```
confint(fit)
```

```
                  2.5 %    97.5 %
(Intercept)  2.152783594 2.3167746
WochentagDo -0.040220978 0.1835209
WochentagFr -0.061421948 0.1634414
WochentagMi  0.008603452 0.2298270
WochentagMo -0.110388958 0.1171302
WochentagSa  0.301707989 0.5571528
WochentagSo  0.234302393 0.4950263
Eventja      0.217989976 0.3655505
```

Bzw. mit vorgeschalteter `exp()`-Funktion, um die Werte als Odds-Ratio auszugeben.

```
# Konfidenzintervalle als Odds Ratio
exp(confint(fit))
```

```
                2.5 %    97.5 %
(Intercept) 8.6087884 10.142906
WochentagDo 0.9605771  1.201440
WochentagFr 0.9404263  1.177556
WochentagMi 1.0086406  1.258382
WochentagMo 0.8954858  1.124266
WochentagSa 1.3521663  1.745695
WochentagSo 1.2640267  1.640541
Eventja     1.2435746  1.441307
```

Die Funktion `tab_model()` aus dem `sjPlot`-Paket erstellt ihrerseits eine schöne Übersichtstabelle, wobei die Werte in Odds-Ratios angegeben und die Konfidenzintervalle enthalten sind.

```
sjPlot::tab_model(fit)
```

	Patienten		
Predictors	*Incidence Rate Ratios*	*CI*	*p*
(Intercept)	9.35	8.61 – 10.14	**<0.001**
Wochentag [Do]	1.07	0.96 – 1.20	0.210
Wochentag [Fr]	1.05	0.94 – 1.18	0.374
Wochentag [Mi]	1.13	1.01 – 1.26	**0.035**
Wochentag [Mo]	1.00	0.90 – 1.12	0.954
Wochentag [Sa]	1.54	1.35 – 1.75	**<0.001**
Wochentag [So]	1.44	1.26 – 1.64	**<0.001**
Event [ja]	1.34	1.24 – 1.44	**<0.001**
Observations	350		
R^2 Nagelkerke	0.493		

Abb. 2.14

Um die Güte des Modells zu bestimmen, können nun weitere Berechnungen vorgenommen werden.

2.2.3.1 Dispersion

Die Equidistation (Dispersion) des Modells sollte zwischen $0,9$ und $1,1$ liegen. Um sie zu bestimmen teilen wir die Residualdeviance durch die Anzahl der Feiheitsgrade.

```
fit$deviance / fit$df.residual
```

```
[1] 1.010433
```

Unser Wert liegt innerhalb des Bereichs, d.h. das Modell ist nicht überdispersioniert. Mit der Funktion `dispersiontest()` aus dem `AER`-Paket lässt sich zusätzlich ein Signifikanztest auf Überdispersion fahren. Die Nullhypothese besagt, das **keine** Überdispersion vorliegt.

```
AER::dispersiontest(fit, trafo=1)
```

```
^^IOverdispersion test

data:  fit
z = -0.64112, p-value = 0.7393
alternative hypothesis: true alpha is greater than 0
sample estimates:
      alpha
-0.04479655
```

Der Test ist nicht signifikant, es liegt keine Überdispersion vor.
Sollte in dem Modell Überdispersion vorliegen, erfolgt der Aufruf von `gml()` mit dem Parameter `family = quasipoisson`.

```
glm(Patienten ~ Wochentag + Event, data = df, family = quasipoisson)
```

2.2.3.2 Nullmodell

Mit einem *Nullmodell* können wir testen, ob unser gefittetes Modell die Daten besser erklären kann. Hierfür ersetzen wir alle Prädiktoren durch die Ziffer 1.

```
fit0 <- glm(Patienten ~ 1, data = df, family = poisson)
```

Vergleichen wir die AIC-Werte beider Modelle.

```
# Ziehe von unserem Modell-AIC den Nullmodell-AIC ab
fit$aic - fit0$aic
```

```
[1] -152.451
```

Das Ergebnis ist negativ. Das beudetet, dass der AIC-Wert des Nullmodells größer ist. Somit ist unser AIC-Modellwert *kleiner*, und das bedeutet, dass es *besser* ist als das Nullmodell. Zusätzlich können wir die beiden Modelle mittels Varianzanalyse (siehe Abschnitt 2.2.9 auf Seite 285) untersuchen, wobei wir als Testeinstellung den Chi^2-Test wählen.

```
anova(fit0, fit, test="Chisq")
```

```
Analysis of Deviance Table

Model 1: Patienten ~ 1
Model 2: Patienten ~ Wochentag + Event
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1       349     512.02
2       342     345.57  7   166.45 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Unser gefittetes Modell erklärt die Daten signifikant besser als das Nullmodell.

2.2.3.3 Vorhersagen

Mit der Funktion `predict()` können wir nun die wahrscheinliche Patientenzahl für Kombinationen aus Wochentag und Event vorhersagen.

```
# Dienstag mit Event in der Stadt zu rechnen?
predict(fit, data.frame(Wochentag="Di", Event="ja"), type = "response")

# ohne Label "1" ausgeben
as.numeric(predict(fit, data.frame(Wochentag="Di", Event="ja"),
                   type = "response"))
```

```
[1] 12.52856
```

Um gut vorbereitet zu sein, runden wir die Patientenzahl mittels `ceiling()` auf die nächste ganze Zahl auf.

```
as.numeric(ceiling(predict(fit, data.frame(Wochentag="Di", Event="ja"),
                           type = "response")))
```

```
[1] 13
```

Wir können davon ausgehen, dass an einem Dienstag mit Event in der Stadt etwa 13 Patienten in die Notaufnahme kommen werden.
Die Wahrscheinlichkeiten für verschiedene Patientenanzahlen an beliebigen Wochentagen mit oder ohne Stadtevent lassen sich mit der Funktion `ppois()` bestimmen (siehe 1.2.4.4 auf Seite 94).

```
# Lambda-Wert für "Dienstag" und Event="ja"
Modelllambda <- predict(fit, data.frame(Wochentag="Di", Event="ja"),
                        type = "response")

## Wahrscheinlichkeit, dass an einem Dienstag mit Event...
## - bis zu 16 Patienten eingeliefert werden (nicht mehr als 16)
ppois(16, Modelllambda)
```

```
[1] 0.8674935
```

```
## - mindestens 16 Patienten eingeliefert werden (16 oder mehr)
1 - ppois(16, Modelllambda)
```

```
[1] 0.1325065
```

```
## - exakt 16 Patienten eingeliefert werden
ppois(16, Modelllambda) - ppois(15, Modelllambda)
```

```
[1] 0.0637844
```

So können wir einen Plot erstellen, auf dem die Wahrscheinlichkeiten angegeben sind, dass an einem Dienstag mit oder ohne Event in der Stadt *mindestens* `x` Patienten aufgenommen werden.

```
# Lambda für Event="nein"
Modelllambda_ohne <- predict(fit, data.frame(Wochentag="Di", Event="nein"),
                             type = "response")

# Patienten N 1 bis 25
x <- 1:25

# plotten für Event="ja"
plot(x, 1-ppois(x, Modelllambda),
     type="b", col="blue",
     main = "Wahrscheinlichkeiten für Mindestpatientenanzahl",
     sub="an einem Dienstag mit und ohne Event ",
```

```
     ylab="Wahrscheinlichkeit",
     xlab="Mindestanzahl Patienten",  xaxt = "n")
# X-Achsenticks in 2er-Schritten
axis(1, at=seq(0, 25, 2))
# füge Punktlinie für Event="nein" hinzu
lines(x, 1-ppois(x, Modelllambda_ohne),
      type="b", col="red")
# füge Hilfslinie hinzu
abline(h=.5, lty=5, col="grey65")
# füge Legendenbox hinzu
legend(x="topright", inset=0.005,
       legend=c("ohne Event", "mit Event"),
       col=c("red", "blue"), lwd=6, bg="grey95")
```

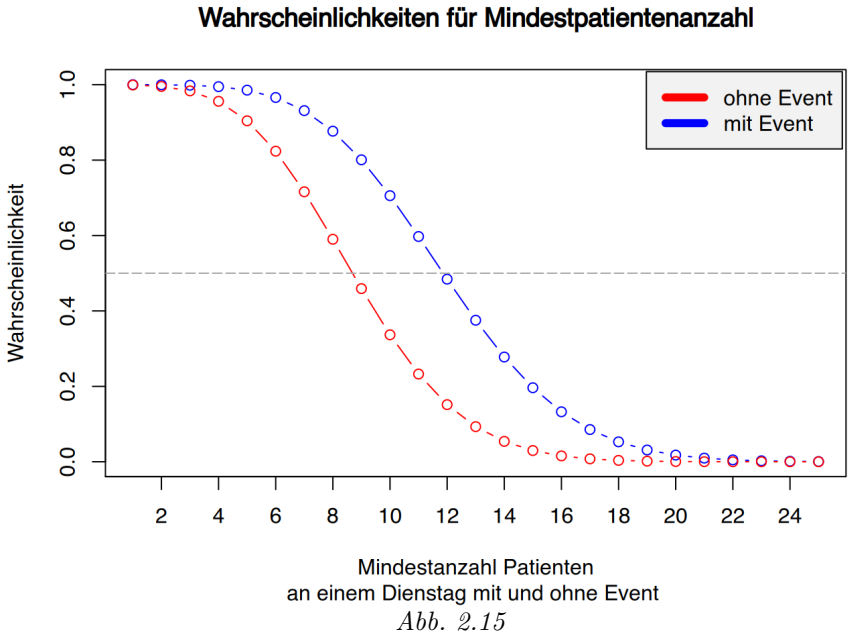


Abb. 2.15

Das geht ebenso für die Wahrscheinlichkeiten der `exakten` Patientenanzahl.

```
# Wahrscheinlichkeit für exakte Patienten-N mit Event="ja"
y1 <- ppois(x, Modelllambda) - ppois(x-1, Modelllambda)
# Wahrscheinlichkeit für exakte Patienten-N mit Event="nein"
y2 <- ppois(x, Modelllambda_ohne) - ppois(x-1, Modelllambda_ohne)

# plotte für Event="nein"
plot(x, y2, type="b",  col="red",
     main = "Wahrscheinlichkeiten für exakte Patientenanzahl",
     sub="an einem Dienstag mit und ohne Event ",
     ylab="Wahrscheinlichkeit",
     xlab="Anzahl Patienten",  xaxt = "n")
# X-Achsenticks in 2er-Schritten
axis(1, at=seq(0, 25, 2))
# Hilfslinie für Event="nein"
abline(v=9, lty=5, col="pink", lwd=2)
# Punktlinie für Event="ja" einzeichnen
lines(x, y1, type="b", col="blue")
# Hilfslinie für Event="ja"
abline(v=12, lty=5, col="paleturquoise", lwd=2)
# Legendenbox hinzufügen
legend(x="topright", inset=0.005,
       legend=c("ohne Event", "mit Event"),
       col=c("red", "blue"), lwd=6, bg="grey95")
```

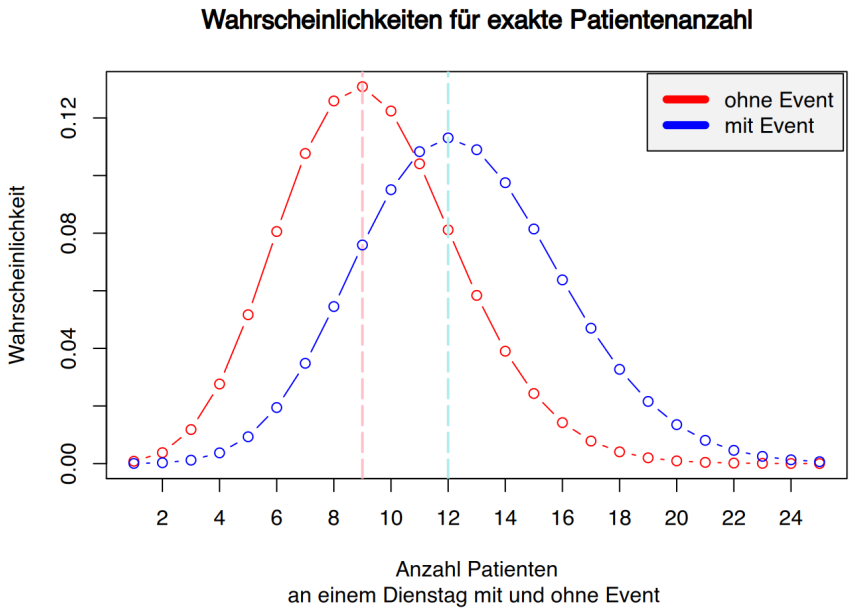


Abb. 2.16

2.2.4 Logistische Regression

Als Beispiel dienen Daten der Challenger-Katastrophe[1]. Der Ausfall eines oder mehrerer Dichtungsringe in einer der seitlichen Feststoffraketen wurde als Grund der Katastrophe ermittelt. Probleme mit den Dichtungsringen sind auch vorher am Boden aufgetreten. So liegen Erhebungsdaten vor, die den Defekt eines Rings in Zusammenhang mit der Außentemperatur darstellen können: Je kälter es draußen ist, desto höher die Ausfallwahrscheinlichkeit eines Rings. Bedauerlicher Weise wurden diese Daten erst rückwirkend ausgewertet.
Diese Daten übertragen wir wie folgt in R:

```
1  Temp <- c(66, 67, 68, 70, 72, 75, 76, 79, 53, 58, 70, 75, 67, 67, 69, 70, 73, 76, 78, 81, 57, 63, 70)
2  Ausfall <- factor(c(0, 0,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1, 1))
3  shuttle <- data.frame(Temp, Ausfall)
4  colnames(shuttle) <- c("Temp", "Ausfall")
```

Die Kodierung des factor `Ausfall` lautet: `0`=kein Ausfall | `1`=Ausfall
Die Einheit für `Temp` lautet Fahrenheit.
Der Aufruf zur logistischen Regression (`Ausfall` erklärt durch `Temp`) lautet:

```
1  fit <- glm(Ausfall~Temp,data=shuttle, family=binomial)
2  fit
```

Dies liefert die Ausgabe

[1] Am 28. Januar 1986 explodierte die Raumfähre Challenger 73 Sekunden nach dem Start.

```
Call:  glm(formula = Ausfall ~ Temp, family = binomial, data = shuttle)

Coefficients:
(Intercept)         Temp
    15.0429      -0.2322

Degrees of Freedom: 22 Total (i.e. Null);  21 Residual
Null Deviance:^^I    28.27
Residual Deviance: 20.32 ^^IAIC: 24.32
```

Die Odds Ratio (OR) für $\beta_{Temp=-0,2322}$ errechnet sich mit der `exp()`-Funktion:

```
1  exp(-0.2322)
```

```
[1] 0.7927875
```

Dies liefert das Ergebnis $OR = 0.7927875$. Die Interpretation lautet: Mit jedem höhren Grad an Temparatur ändert sich die Chance auf einem Ausfall um den Faktor 0.7927. Das bedeutet, dass die Chancen eines Ausfalls niedriger werden, je höher die Außentemperatur steigt.
In der Nacht vor dem Challengerstart konnte eine Temperatur von 37° Fahrenheit gemessen werden. Auf Grundlage der logistischen Regression lässt sich nun die Ausfallwahrscheinlichkeit bei 37° Außentemperatur berechnen

```
1  predict(fit, data.frame(Temp=37),type="response")
```

```
[1] 0.9984243
```

Das Modell sagt bei einer Außentemperatur von 37° eine Ausfallwahrscheinlichkeit von 99,84% voraus.
Jetzt kann man dies für jede mögliche Temperatur durchmachen, und schon erhält man dieses schöne Plot (Abbildung 2.17).

```
1  y <- predict(fit, data.frame(Temp=30:90), type="response")
2  plot(30:90, y, type="l", ylab="Ausfallswahrscheinlichkeit", xlab="Temperatur in F")
3  abline(v=seq(20,100,1), h=seq(-0.2,1.2,0.05),col="gray") #Mathegitter
4  abline(v=37, col="red") # 37-Grad
5  abline(h=0.5, col="black") # Mittellinie
6  points(30:90, y, type="l",lwd=2, col="blue") # Schätzwerte in "blau"
```

Der Befehl erzeugt Abbildung 2.17.

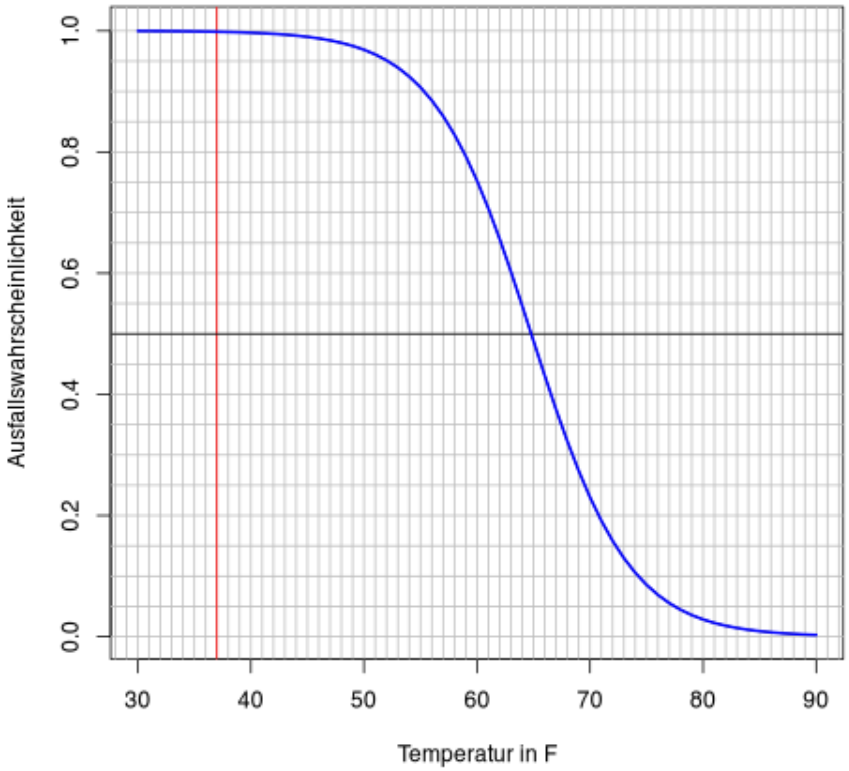

Ausfallswahrscheinlichkeit
0.0
0.2
0.4
0.6
0.8
1.0
30
40
50
60
70
80
90
Temperatur in F

Abb. 2.17

2.2.5 Ordinale Regression

Zur Bestimmung des Einflusses von Prädiktoren auf ein ordinales Ziel stehen verschiedene Modelle der ordinalen Regression zur Verfügung. Namentlich werden hier vor allem "Proportional Odds Modelle" (siehe Seite 250) und "Continuation Ratio Modelle" (siehe Seite 255) verwendet. Eine verständliche Einführung in ordinale Modelle findet sich im Open Access Artikel von große Schlarmann und Galatsch [8].

Für die ordinalen Regressionen verwenden wir das R-Paket `VGAM`. Für die Berechnungen laden wir mit `load()` einen Beispieldatensatz aus dem Internet:

```
# VGAM installieren
install.packages("VGAM", dependencies = T)

# Test-Datensatz aus dem Internet laden
load(url("http://www.produnis.de/R/OrdinalSample.RData"))

# kopiere Originaldaten für Testzwecke
mydata <- ordinalSample

# zeigen den Datensatz an
head(mydata)
```

```
   Konflikt Zufriedenh Geschlecht Stimmung
1       4.2       2.50          1  maessig
6       1.8       3.00          1 sehr gut
15      3.4       1.75          1  maessig
16      4.0       2.50          2  maessig
22      2.2       2.50          1      gut
23      3.4       2.25          1  maessig
```

Der Test-Datensatz enthält das Datenframe `ordinalSample`. Dieses besteht aus den vier Variablen `Konflikt`, `Zufriedenh`, `Geschlecht` und `Stimmung`.

Die Variable `Stimmung` dient hier als ordinales Ziel mit den Ausprägungen `schlecht < maessig < gut < sehr gut`.

Die Variable `Konflikt` dient als Prädiktor. Sie gibt an, wieviele Konflikte derzeit im Arbeitsleben vorliegen.

Die Variable `Zufriedenh` soll ebenfalls Prädiktor sein. Sie beschreibt, wie zufrieden Probanden mit ihrem Job sind.

Wir wollen nun wissen, wie stark `Stimmung` durch `Konflikt` und `Zufriedenh` beinflusst wird. Das heisst, in unserem Modell soll die Variable `Stimmung` durch `Konflikt` und `Zufriedenh` beschrieben werden.

2.2.5.1 Proportional Odds Modelle

Diese errechnet sich leicht mit dem `VGAM`-Paket und der Funktion `vglm()`:

```r
# aktiviere VGAM
library(VGAM)

# erstelle Modell
pom <- vglm(Stimmung ~ Konflikt + Zufriedenh, data = mydata, family = cumulative(parallel = T))

# oder abgekürzt
pom <- vglm(Stimmung ~ Konflikt + Zufriedenh, data = mydata, family = propodds)

# gebe Modellzusammenfassung aus
summary(pom)
```

```
Call:
vglm(formula = Stimmung ~ Konflikt + Zufriedenh, family = propodds,
    data = mydata)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1   0.6210     0.6482   0.958   0.3381
(Intercept):2  -1.7703     0.6540  -2.707   0.0068 **
(Intercept):3  -4.0578     0.6759  -6.004 1.93e-09 ***
Konflikt       -0.5762     0.0970  -5.940 2.86e-09 ***
Zufriedenh      1.3643     0.2021   6.751 1.47e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Names of linear predictors: logitlink(P[Y>=2]), logitlink(P[Y>=3]), logitlink(P[Y>=4])

Residual deviance: 920.5493 on 1240 degrees of freedom

Log-likelihood: -460.2746 on 1240 degrees of freedom

Number of Fisher scoring iterations: 5

No Hauck-Donner effect found in any of the estimates

Exponentiated coefficients:
  Konflikt Zufriedenh
 0.5620549  3.9129275
```

Mit dem Modell können nun weitere Parameter errechnet werden:

```r
# Koeffizienten
pom.coef <- (coef(summary(pom)))
pom.coef
```

```
                Estimate Std. Error    z value     Pr(>|z|)
(Intercept):1  0.6209962 0.64821816  0.9580049 3.380603e-01
(Intercept):2 -1.7702692 0.65404383 -2.7066523 6.796539e-03
(Intercept):3 -4.0577657 0.67586741 -6.0037895 1.927645e-09
Konflikt      -0.5761558 0.09700358 -5.9395317 2.858375e-09
Zufriedenh     1.3642858 0.20208974  6.7508909 1.469400e-11
```

Die Odds Ratio errechnet sich mit der `exp()`-Funktion:

```r
# Odds Ratio
pom.odds =exp(coef(pom))
pom.odds
```

```
(Intercept):1 (Intercept):2 (Intercept):3      Konflikt    Zufriedenh
    1.8607808     0.1702871     0.0172876     0.5620549     3.9129275
```

```
# Devianz
pom.devi =deviance(pom)
pom.devi
```

```
(Intercept):1 (Intercept):2 (Intercept):3      Konflikt    Zufriedenh
    1.8607808     0.1702871     0.0172876     0.5620549     3.9129275
```

```
# AIC
pom.aic =AIC(pom)
pom.aic
```

```
[1] 930.5493
```

```
# logLikelihood
pom.ll =logLik(pom)
pom.ll
```

```
[1] -460.2746
```

```
# 0-Modell (fuer pseudo R^2)
p0 =vglm(Stimmung ~ 1, data = mydata, family = propodds)
p0.ll =  logLik(p0)
```

```
# R^2 McFadden
pom.mcfad =as.vector(1 - (pom.ll/p0.ll))
pom.mcfad
```

```
[1] 0.1240613
```

```
# R^2 Cox&Snell
N =length(mydata[, 1])  # Anzahl der Fälle
pom.cox =  as.vector(1 - exp((2/N) * (p0.ll - pom.ll)))
pom.cox
```

```
[1] 0.2696034
```

```
# R^2 Nagelkerke
pom.nagel =as.vector((1 - exp((2/N) * (p0.ll - pom.ll)))/(1 - exp(p0.ll)^(2/N)))
pom.nagel
```

```
[1] 0.2928789
```

Das Proportional Odds Modell geht von der "equal slopes assumption" (auch: "proportional odds assumption") aus. Diese Annahme muss geprüft werden, bevor das Modell als gültig angesehen werden kann. Dies geht mit dem `VGAM`-Paket recht einfach. Der Befehl

```
vglm(Stimmung ~ Konflikt+Zufriedenh, data=mydata, family=propodds)
```

ist eine Abkürzung für

```
vglm(Stimmung ~ Konflikt+Zufriedenh, data=mydata, family=cumulative(parallel=T))
```

Der Parameter `parallel=TRUE/FALSE` stellt ein, ob das Modell mit equal slopes (=`TRUE`), oder ohne equal slopes assumption (=`FALSE`) erstellt werden soll.
Zur Überprüfung der "Equal Slopes Assumption" erstellt man jeweils ein `TRUE` und ein `FALSE`-Modell, und führt dann einen Likelihood-Test durch. Die Nullhypothese lautet, dass equal slopes vorliegen.

```
# Modell OHNE equal slopes
npom =vglm(Stimmung ~ Konflikt+Zufriedenh, data=mydata, family=cumulative(parallel=F))

# log-likelihood-Test auf Equal Slopes Assumption
lrtest(pom, npom)  # Test
```

```
Likelihood ratio test

Model 1: Stimmung ~ Konflikt + Zufriedenh
Model 2: Stimmung ~ Konflikt + Zufriedenh
   #Df  LogLik Df  Chisq Pr(>Chisq)
1 1240 -460.27
2 1236 -456.46 -4 7.6202     0.1065
```

Der Test ist mit $p = 0,1065$ nicht signifikant. Das heisst, es liegen equal slopes vor. Das Modell darf also verwendet werden.
Alternativ kann dieser Test auch auf die Devianz der Modelle bestimmt werden:

```
# Devianz-Test auf Equal Slopes Assumption
pom.pdevi = (1 - pchisq(deviance(pom) - deviance(npom), df = df.residual(pom) - df.residual(npom)))
pom.pdevi
```

```
[1] 0.106526
```

Ist der Test signifikant, liegen keine equal slopes vor. Hier kann ein Partial Proportional Odds Modell erstellt werden, welches die equal slopes nur für bestimmte Prädiktoren annimmt.
Mit dem `VGAM`-Paket kann recht einfach bestimmt werden, wie ein solches Modell erstellt werden soll. Hierfür erweitertet man die `parallel`-Option wie folgt:

```
# Equal Slopes NUR für 'Konflikt'
ppom =vglm(Stimmung ~ Konflikt+Zufriedenh, data=mydata, family=cumulative(parallel=T ~ Konflikt - 1))
ppom
```

```
Call:
vglm(formula = Stimmung ~ Konflikt+Zufriedenh, family=cumulative(parallel=T ~ Konflikt - 1), data=mydata)


Coefficients:
(Intercept):1 (Intercept):2 (Intercept):3      Konflikt  Zufriedenh:1  Zufriedenh:2  Zufriedenh:3
    0.6614816     1.5373699     2.7337389     0.5705897    -1.9684047    -1.2805318    -0.8865225

Degrees of Freedom: 1245 Total; 1238 Residual
Residual deviance: 914.4929
Log-likelihood: -457.2464
```

```
# Nochmals EqualSlopes NUR für 'Konflikt'
ppom =vglm(Stimmung ~ Konflikt+Zufriedenh, data=mydata, family=cumulative(parallel=F ~Zufriedenh))
ppom
```

```
Call:
vglm(formula = Stimmung ~ Konflikt+Zufriedenh, family=cumulative(parallel=F ~
    Zufriedenh), data = mydata)


Coefficients:
(Intercept):1 (Intercept):2 (Intercept):3      Konflikt  Zufriedenh:1  Zufriedenh:2  Zufriedenh:3
    0.6614816     1.5373699     2.7337389     0.5705897    -1.9684047    -1.2805318    -0.8865225

Degrees of Freedom: 1245 Total; 1238 Residual
Residual deviance: 914.4929
Log-likelihood: -457.2464
```

`parallel=T Konflikt-1` bedeutet, dass equal slopes nur für `Konflikt` angenommen wird.

`parallel=F Zufriedenh` bedeutet, dass equal slopes nur für `Zufriedenh` **nicht** angenommen wird.

Beide Befehle bedeuten also das selbe. Daher ist die R-Ausgabe bei beiden Befehlen gleich. Eine Koeffizientenübersicht erhält man per

```
ppom.ce =coef(summary(ppom))
ppom.ce
```

```
               Estimate Std. Error    z value     Pr(>|z|)
(Intercept):1 0.6614816 0.85866412  0.7703613 4.410856e-01
(Intercept):2 1.5373699 0.71997824  2.1353005 3.273647e-02
(Intercept):3 2.7337389 0.98084390  2.7871294 5.317723e-03
Konflikt      0.5705897 0.09715456  5.8730094 4.279541e-09
```

```
Zufriedenh:1  -1.9684047 0.34535219 -5.6997024 1.200167e-08
Zufriedenh:2  -1.2805318 0.23585229 -5.4293805 5.654999e-08
Zufriedenh:3  -0.8865225 0.32542569 -2.7241933 6.445877e-03
```

Mit einem kleinen Script können Konfidenzintervalle und andere Werte ausgegeben werden:

```
get.ci <- function(x){
   back <- cbind( x[1], # estimate
                  (x[1] - (1.96 * x[2])), #lCI
                  (x[1] + (1.96 * x[2])), #uCI
                  x[2], x[3], # SD und z
                  (2*(1 -pnorm(abs(x[3])))  ),# p-wert
                  exp(x[1]))
   colnames(back) <-  c("Estimate", "lCI", "uCI", "SD", "z", "p-value", "OR")
   return(back)
}
```

Der Aufruf erfolgt z.B. so:

```
get.ci(as.numeric(ppom.ce[4, ]))
```

```
      Estimate       lCI       uCI         SD        z      p-value      OR
[1,] 0.5705897 0.3801667 0.7610126 0.09715456 5.873009 4.279541e-09 1.76931
```

2.2.5.2 Continuation Ratio Model

Diese lassen sich ebenfalls mit der Funktion `vglm()` erstellen. Um ein Continuation Ratio Modell zu rechnen, muss der Parameter `family` auf `cratio` gesetzt werden.

```
# aktiviere VGAM
library(VGAM)

# CR-Modell MIT PO-Assumption (Vorwärts)
crm =vglm(Stimmung ~ Konflikt + Zufriedenh, data = mydata, family = cratio(parallel = T, reverse = F))
summary(crm)
```

```
Call:
vglm(formula = Stimmung ~ Konflikt + Zufriedenh, family = cratio(parallel = T,
    reverse = F), data = mydata)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1  0.92610    0.57405   1.613   0.1067
(Intercept):2 -1.14132    0.57924  -1.970   0.0488 *
(Intercept):3 -3.01660    0.60706  -4.969 6.72e-07 ***
Konflikt      -0.53645    0.08639  -6.209 5.32e-10 ***
Zufriedenh     1.16010    0.17944   6.465 1.01e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Names of linear predictors: logitlink(P[Y>1|Y>=1]), logitlink(P[Y>2|Y>=2]), logitlink(P[Y>3|Y>=3])

Residual deviance: 924.3824 on 1240 degrees of freedom

Log-likelihood: -462.1912 on 1240 degrees of freedom

Number of Fisher scoring iterations: 6

No Hauck-Donner effect found in any of the estimates
```

Dies berechnet standardmäßig die Vorwärts-Methode. Möchte man die Rückwärts-Methode rechnen, setzt man den Parameter `reverse` auf `TRUE`.

```
# CR-Modell MIT PO-Assumption (Rückwärts)
crm =vglm(Stimmung ~ Konflikt + Zufriedenh, data = mydata, family = cratio(parallel = T, reverse = TRUE))
summary(crm)
```

```
Call:
vglm(formula = Stimmung ~ Konflikt + Zufriedenh, family = cratio(parallel = T,
    reverse = TRUE), data = mydata)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1  0.12822    0.57259   0.224 0.822809
(Intercept):2  2.08751    0.58094   3.593 0.000326 ***
(Intercept):3  4.04245    0.60636   6.667 2.62e-11 ***
Konflikt       0.45234    0.08488   5.329 9.88e-08 ***
Zufriedenh    -1.25631    0.18108  -6.938 3.98e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Names of linear predictors: logitlink(P[Y<2|Y<=2]), logitlink(P[Y<3|Y<=3]), logitlink(P[Y<4|Y<=4])

Residual deviance: 920.4627 on 1240 degrees of freedom

Log-likelihood: -460.2314 on 1240 degrees of freedom

Number of Fisher scoring iterations: 5

No Hauck-Donner effect found in any of the estimates
```

Hat man die passende Methode gewählt, folgen die selben Befehle wie beim Proportional Odds Modell (siehe Seite 250). Zunächst muss überprüft werden, ob equal slopes vorliegen:

```
# CR-Modell OHNE PO-Assumption (Rückwärts)
ncrm =vglm(Stimmung ~ Konflikt + Zufriedenh, data = mydata, family = cratio(parallel = F, reverse = T))
summary(ncrm)
```

```
Call:
vglm(formula = Stimmung ~ Konflikt + Zufriedenh, family = cratio(parallel = F,
    reverse = T), data = mydata)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1   2.3649     1.1740   2.014 0.043969 *
(Intercept):2   1.9998     0.8117   2.464 0.013753 *
(Intercept):3   2.4670     1.1190   2.205 0.027474 *
Konflikt:1      0.1465     0.1811   0.809 0.418633
Konflikt:2      0.4864     0.1194   4.073 4.65e-05 ***
Konflikt:3      0.6308     0.1689   3.736 0.000187 ***
```

```
Zufriedenh:1   -1.8008     0.3919  -4.596 4.32e-06 ***
Zufriedenh:2   -1.2602     0.2578  -4.889 1.01e-06 ***
Zufriedenh:3   -0.8351     0.3382  -2.469 0.013543 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Names of linear predictors: logitlink(P[Y<2|Y<=2]), logitlink(P[Y<3|Y<=3]), logitlink(P[Y<4|Y<=4])

Residual deviance: 914.7924 on 1236 degrees of freedom

Log-likelihood: -457.3962 on 1236 degrees of freedom

Number of Fisher scoring iterations: 6

No Hauck-Donner effect found in any of the estimates
```

```
# loglikelihood test auf equal slopes assumption
lrtest(ncrm, crm)  # Test
```

```
Likelihood ratio test

Model 1: Stimmung ~ Konflikt + Zufriedenh
Model 2: Stimmung ~ Konflikt + Zufriedenh
   #Df  LogLik Df  Chisq Pr(>Chisq)
1 1236 -457.40
2 1240 -460.23  4 5.6703     0.2252
```

Der Test ist nicht signifikant (p=0,23). Das bedeutet, dass die Annahme der equal slopes für die hier getesteten Rückwärts-Modelle beibehalten werden kann.
Nun können weitere Modellparameter bestimmt werden:

```
# 0-Modell (fuer pseudo R^2) Vorbereitungen
c0 =vglm(Stimmung ~ 1, data = mydata, family = cratio(parallel = T))
c0.ll =  logLik(c0)
crm.ll =  logLik(crm)
N =  length(mydata[, 1])  # Anzahl der Fälle

# R^2 McFadden
crm.mcfad =as.vector(1 - (crm.ll/c0.ll))
crm.mcfad
```

```
[1] 0.1241436
```

```
# R^2 Cox&Snell
crm.cox =as.vector(1 - exp((2/N) * (c0.ll - crm.ll)))
crm.cox
```

```
[1] 0.2697557
```

```
# R^2 Nagelkerke
crm.nagel =as.vector((1 - exp((2/N) * (c0.ll - crm.ll)))/(1 - exp(c0.ll)^(2/N)))
crm.nagel
```

```
[1] 0.2930443
```

```
# Devianz
crm.devi =deviance(crm)
crm.devi
```

```
[1] 920.4627
```

```
# AIC
crm.aic =AIC(crm)
crm.aic
```

```
[1] 930.4627
```

Mit unserer Funktion `get.ci()` von Seite 255 können wir uns die Modellparameter der Koeffizienten ausgeben lassen

```
# Konfidenzinztervalle
crm.ce =coef(summary(crm))
crm.ce
```

```
                Estimate Std. Error    z value     Pr(>|z|)
(Intercept):1  0.1282227 0.57259420  0.2239329 8.228095e-01
(Intercept):2  2.0875126 0.58094099  3.5933299 3.264788e-04
(Intercept):3  4.0424508 0.60636000  6.6667504 2.615293e-11
Konflikt       0.4523353 0.08488357  5.3288915 9.881398e-08
Zufriedenh    -1.2563060 0.18108066 -6.9378255 3.981811e-12
```

```
get.ci(as.numeric(crm.ce[4, ]))
```

```
      Estimate       lCI       uCI         SD        z      p-value       OR
[1,] 0.4523353 0.2859635 0.6187071 0.08488357 5.328892 9.881398e-08 1.571979
```

```
get.ci(as.numeric(crm.ce[5, ]))
```

```
      Estimate       lCI        uCI         SD         z      p-value        OR
[1,] -1.256306 -1.611224 -0.9013879 0.1810807 -6.937825 3.981704e-12 0.2847038
```

2.2.6 Konfidenzintervalle

Konfidenzintervalle folgen der Logik, dass zum Punktschätzwert ein gewisser Fehlerbereich addiert und subtrahiert wird. Hierdurch wird ein Intervall mit Unter- und Obergrenze aufgezogen, das mit einer bestimmten Irrtumswahrscheinlichkeit den "wahren Wert" der Grundgesamtheit enthält.

2.2.6.1 Normalverteilung

Um ein Konfidenzintervall **für einen Mittelwert** aus einer normalverteilten Wertemenge zu berechnen wird folgende Formel angewendet:

$$\left.\begin{matrix}\mu_u \\ \mu_o\end{matrix}\right\} = \bar{x} \pm t \cdot \frac{s}{\sqrt{n}} \tag{2.1}$$

$\bar{x}$ = arithmetisches Mittel
n = Stichprobenumfang
s = Standardabweichung
t = Wert der t-Verteilung, (mit Freiheitsgraden $df = n - 1$)

Die Formel ist in der Funktion `t.test()` implementiert. Die Funktion kann angewendet werden, wenn die Daten in Rohform vorliegen.

```
# Testwerte
x <-  c(50, 45, 49, 48, 53, 69, 50, 54, 50, 36, 56,
        48, 48, 50, 50, 53, 34, 68, 47, 51)
# Konfidenzintervall
t.test(x)
```

```
^^IOne Sample t-test

data:  x
t = 27.806, df = 19, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 46.65246 54.24754
sample estimates:
mean of x
    50.45
```

Über die Zusatzparameter `$estimate` und `$conf.int` kann die Ausgabe auf Median und Konfidenzgrenzen beschränkt werden.

```
t.test(x)$estimate
```

```
mean of x
50.45
```

```
t.test(x)$conf.int
```

```
[1] 46.65246 54.24754
attr(,"conf.level")
[1] 0.95
```

Damit die Ausgabe *schöner* wird, wickeln wir noch `as.numeric()` außen herum.

```r
as.numeric(t.test(x)$conf.int)
```

```
[1] 46.65246 54.24754
```

Standardmäßig wird das 95%-Intervall berechnet. Über den Parameter `conf.level` können beliebige Konfidenzniveaus angegeben werden.

```r
# 99%-Intervall
t.test(x, conf.level=0.99)
```

```
^^IOne Sample t-test

data:  x
t = 27.806, df = 19, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 45.25918 55.64082
sample estimates:
mean of x
    50.45
```

Wenn die Daten selbsst **nicht** vorliegen, sondern lediglich die Kennwerte bekannt sind, müssen wir die Grenzen von Hand ausrechnen. Nehmen wir an, eine Zufalls-Erhebung über die jährlichen Ausgaben für Pflegehilfsmittel ergab bei 100 Befragten ($n = 100$) einen Mittelwert $\bar{x}$=400,-€ bei einer Standardabweichung von s=20,-€.
Das Konfidenzintervall für $\alpha = 0.05$ wird nun wie folgt bestimmt.

```r
# schreibe gegebene Werte in Variablen
n     <- 100
xquer <- 400
s     <- 20

# 95%KI-Fehler berechnen mit (alpha/2)
fehler95 <- qt(0.975, df=n-1)*s/sqrt(n)

# Fehlerwert ausgeben
fehler95
```

```
[1] 3.968434
```

Beachten Sie, dass in der Funktion `qt()` zur Bestimmung des kritischen t-Wertes die Irrtumswahrscheinlichkeit α halbiert werden muss, da es sich um eine zweiseitige Fragestellung handelt. So muss `qt()` nicht mit dem Wert `0.95` ($\alpha = 0.05$) aufgerufen werden, sondern mit `0.975` ($\frac{\alpha}{2} = 0.025$).

```r
# untere Grenze des 95%KI
xquer - fehler95
```

```
[1] 396.0316
```

Die untere Grenze des 95%-Konfidenzintervalls liegt bei `396.0316` €.

```
# obere Grenze des 95%KI
xquer + fehler95
```

```
[1] 403.9684
```

Die obere Grenze des 95%-Konfidenzintervalls liegt bei `403.9684` €.

```
# Länge des 95%KI
(xquer + fehler95) - (xquer - fehler95)
# oder schlauer
2*fehler95
```

```
[1] 7.936868
```

Das 95%-Konfidenzintervall hat eine Länge von `7.936868` €.

Für das 99% Konfidenzintervall ändern wir entsprechend um. Auch hier muss, da es sich um eine zweiseitige Fragestellung handelt, mit $\frac{\alpha}{2}$, also 0.005, gerechnet werden. In der Funktion `qt()` ändert sich also der Wert auf `0,995`.

```
# 99%KI-Fehler berechnen mit (alpha/2)
fehler99 <- qt(0.995, df=n-1)*s/sqrt(n)

# zeige fehler99 an
fehler99
```

```
[1] 5.252811
```

```
# untere Grenze des 99%KI
xquer - fehler99
```

```
[1] 394.7472
```

Die untere Grenze des 99%-Konfidenzintervalls liegt bei `394.7472` €.

```
# obere Grenze des 99%KI
xquer + fehler99
```

```
[1] 405.2528
```

Die obere Grenze des 99%-Konfidenzintervalls liegt bei `405.2528` €.

```
# Länge des 99%KI
(xquer + fehler99) - (xquer - fehler99)
# oder schlauer
2*fehler99
```

```
[1] 10.50562
```

Das 99%-Konfidenzintervall hat eine Länge von `10.50562` €.
Die Funktion `KInormal_a()` aus dem `jgsbook`-Paket nimmt uns die vorgestellten Rechenschritte ab.

```
# Berechne das 95%-KI für
# n=100, xquer=400 und s=20
jgsbook::KInormal_a(xquer=400, n=100, s=20, alpha=.05)
```

```
                      x          y
1 0.95 KI untere Grenze 396.031566
2  0.95 KI obere Grenze 403.968434
3        0.95 KI Laenge   7.936868
```

```
# Berechne das 99%-KI für
# n=100, xquer=400 und s=20
jgsbook::KInormal_a(xquer=400, n=100, s=20, alpha=.01)
```

```
                      x         y
1 0.99 KI untere Grenze 394.74719
2  0.99 KI obere Grenze 405.25281
3        0.99 KI Laenge  10.50562
```

Unterschied von Mittelwerten

Das Konfidenzintervall für den **Unterschied von Mittelwerten** normalverteilter Daten berechnet sich mit der Formel

$$\left.\begin{matrix}\mu_u \\ \mu_o\end{matrix}\right\} = \hat{d} \pm t \cdot \sqrt{\frac{s_{pool}^2}{n_x} + \frac{s_{pool}^2}{n_y}} \tag{2.2}$$

$\hat{d}$ = Differenz der Mittelwerte $\bar{x}$,$\bar{y}$
n_x, n_y = die Umfänge der beiden Stichproben
s_{pool} = gepoolte Standardabweichungen = $\frac{(n_x-1)\cdot s_x^2 \; + \; (n_y-1)\cdot s_y^2}{n_x+n_y-2}$
t = Wert der t-Verteilung (mit Freiheitsgraden $df = n_x + n_y - 1$)

Liegen die Verteilungsdaten vor, können wir wieder die `t.test()`-Funktion nutzen.

```
# Testwerte
x <- c(50, 45, 49, 48, 53, 69, 50, 54, 50, 36, 56,
       48, 48, 50, 50, 53, 34, 68, 47, 51)
y <- c(57, 50, 57, 52, 48, 49, 57, 53, 54, 46, 47,
       54, 58, 53, 58, 55, 55, 52, 53, 52)
# Konfidenzintervall
t.test(x, y)
```

```
^^IWelch Two Sample t-test

data:  x and y
t = -1.2848, df = 26.194, p-value = 0.2101
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.628295  1.528295
sample estimates:
mean of x mean of y
    50.45     53.00
```

Über die Parameter `$estimate` und `$conf.int` können wir die Ausgabe auf unsere Werte beschränken.

```
u <- t.test(x,y)

# Unterschied der Mittelwerte
as.numeric(u$estimate[2]) - as.numeric(u$estimate[1])
```

```
[1] 2.55
```

```
# Konfidenzintervall
as.numeric(u$conf.int)
```

```
[1] -6.628295  1.528295
```

Standardmäßig wird das 95%-Intervall berechnet. Über den Parameter `conf.level` können beliebige Konfidenzniveaus angegeben werden.

```
# 99%-Intervall
t.test(x, y, conf.level=0.99)
```

```
^^IWelch Two Sample t-test

data:  x and y
t = -1.2848, df = 26.194, p-value = 0.2101
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -8.061928  2.961928
sample estimates:
mean of x mean of y
    50.45     53.00
```

Liegen uns lediglich die Kennwerte der Verteilungen vor, müssen wir das Intervall von Hand ausrechnen. Nehmen wir an, in einer Studie wurde die Wirkung von zwei Medikamenten auf den Magnesiumgehalt des Blutserums (in mg/dl) untersucht. 13 Personen erhielten Medikament A und 10 Personen Medikament B. Nach 6 Wochen wurden folgende Daten bestimmt:

- $\bar{x}_A = 2{,}22$ mg/dl
- $\bar{x}_B = 2{,}70$ mg/dl
- $s_A = 0{,}255$ mg/dl
- $s_B = 0{,}306$ mg/dl

- $n_A = 13$
- $n_B = 10$

Das Konfidenzintervall für den Unterschied der Mittelwerte bei $\alpha = 0,05$ wird nun wie folgt bestimmt (Achtung, da es sich um eine zweiseitige Fragestellung handelt, müssen wir mit $\frac{\alpha}{2} = 0,025$ rechnen).

```
# erzeuge die Daten
xa <- 2.22
xb <- 2.7
sa <- 0.255
sb <- 0.306
na <- 13
nb <- 10

# Differenz der Mittelwerte
d   <- xb - xa

# ausgeben
d
```

```
[1] 0.48
```

```
# berechne die gepoolten Standardabweichungen
s_pool = ((na-1)*sa^2 + (nb-1)*sb^2) / (na+nb-2)

# ausgeben
s_pool
```

```
[1] 0.07728686
```

```
#Fehlerquote berechnen
fehler95 <- qt(0.975, df=na+nb-1) * sqrt(s_pool^2/na + s_pool^2/nb)

# ausgeben
fehler95
```

```
[1] 0.06741865
```

```
# untere Grenze des 95%KI
d - fehler95
```

```
[1] 0.4125813
```

Die untere Grenze des 95%-Konfidenzintervalls liegt bei `0.4125813` mg/dl

```
# obere Grenze des 95%KI
d + fehler95
```

```
[1] 0.5474187
```

Die obere Grenze des 95%-Konfidenzintervalls liegt bei `0.5474187` mg/dl

```
# Länge des Intervalls
2*fehler95
```

```
[1] 0.1348373
```

Das 95%-Konfidenzintervall hat eine Länge von `0.1348373` mg/dl

Für das 99%-Konfidenzintervall wird der Wert für die Funktion `qt()` auf `0.995` gesetzt (da es sich um eine zweiseitige Fragestellung handelt, müssen wir mit $\frac{\alpha}{2} = 0,005$ rechnen).

```
#Fehlerquote berechnen
fehler99 <- qt(0.995, df=na+nb-1) * sqrt(s_pool^2/na + s_pool^2/nb)

# ausgeben
fehler99
```

```
[1] 0.09163373
```

```
# untere Grenze des 99%KI
d - fehler99
```

```
[1] 0.3883663
```

Die untere Grenze des 99%-Konfidenzintervalls liegt bei `0.3883663` mg/dl

```
# obere Grenze des 99%KI
d + fehler99
```

```
[1] 0.5716337
```

Die obere Grenze des 99%-Konfidenzintervalls liegt bei `0.5716337` mg/dl

```
# Länge des Intervalls
2*fehler99
```

```
[1] 0.1832675
```

Das 99%-Konfidenzintervall hat eine Länge von `0.1832675` mg/dl

Die Funktion `KInormal_u()` aus dem `jgsbook`-Paket nimmt uns auch hier die vorgestellten Rechenschritte ab.

```
# Berechne das 95%-KI
jgsbook::KInormal_u(xa, sa, na, xb, sb, nb, alpha=0.05)
```

```
                          x         y
1 Differenz der Mittelwerte 0.4800000
2     0.95 KI untere Grenze 0.4125813
3      0.95 KI obere Grenze 0.5474187
4            0.95 KI Laenge 0.1348373
```

```
# Berechne das 99%-KI
jgsbook::KInormal_u(xa, sa, na, xb, sb, nb, alpha=0.01)
```

```
                          x         y
1 Differenz der Mittelwerte 0.4800000
2     0.99 KI untere Grenze 0.3883663
3      0.99 KI obere Grenze 0.5716337
4            0.99 KI Laenge 0.1832675
```

2.2.6.2 Binomialverteilung

Zur Berechnung von Konfidenzintervallen **für Anteilswerte** wird folgende Formel verwendet.

$$\left.\begin{matrix} p_o \\ p_u \end{matrix}\right\} = k \pm c \cdot \sqrt{\frac{k \cdot (1-k)}{n}} \tag{2.3}$$

n = Umfang der Stichprobe
k = geschätzter Anteil
c = Quartilswert für Irrtumswahrscheinlichkeit α

Liegen die Verteilungswerte vor, kann die Funktion `prop.test()` in Kombination mit `table()` verwendet werden.

```
antwort <- c("ja", "nein", "nein", "ja", "ja", "ja", "ja", "ja", "ja",
"ja", "ja", "ja", "nein", "ja", "ja", "ja", "nein", "ja")
# erstelle Häufigkeitstabelle
freq <- table(antwort)
# Tabelle anzeigen
freq
```

```
antwort
  ja nein
  14    4
```

```
# Konfidenzintervall für Anteil "ja"
prop.test(freq[["ja"]], length(antwort))
```

```
1-sample proportions test with continuity correction

data:  freq[["ja"]] out of length(antwort), null probability 0.5
X-squared = 4.5, df = 1, p-value = 0.03389
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.5191861 0.9262769
sample estimates:
        p
0.7777778
```

```
# Konfidenzintervall für Anteil "nein"
prop.test(freq[["nein"]], sum(freq))
```

```
^^I1-sample proportions test with continuity correction

data:  freq[["nein"]] out of sum(freq), null probability 0.5
X-squared = 4.5, df = 1, p-value = 0.03389
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.0737231 0.4808139
sample estimates:
        p
0.2222222
```

Wenn die Werte der absoluten Häufigkeitstabelle bekannt sind, können die Werte auch direkt an die Funktion übergeben werden. Angenommen, von 120 Studierenden hätten 95 die Abschlussprüfung bestanden, dann lautet der Funktionsaufruf für den Anteil ”bestandenëntsprechend:

```
prop.test(95, 120)
```

```
^^I1-sample proportions test with continuity correction

data:  95 out of 120, null probability 0.5
X-squared = 39.675, df = 1, p-value = 2.999e-10
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.7059845 0.8582409
sample estimates:
        p
0.7916667
```

Über die Zusatzparameter `$estimate` und `$conf.int` kann die Ausgabe auf die Konfidenzgrenzen beschränkt werden.

```
prop.test(95, 120)$conf.int
```

```
  [1] 0.7059845 0.8582409
  attr(,"conf.level")
  [1] 0.95
```

Standardmäßig wird das 95%-Intervall berechnet. Über den Parameter `conf.level` können beliebige Konfidenzniveaus angegeben werden.

```
# 99%-Intervall
prop.test(95, 120, conf.level=0.99)
```

```
^^I1-sample proportions test with continuity correction

data:  95 out of 120, null probability 0.5
X-squared = 39.675, df = 1, p-value = 2.999e-10
alternative hypothesis: true p is not equal to 0.5
99 percent confidence interval:
 0.6775964 0.8738928
sample estimates:
        p
0.7916667
```

Wenn die absolute Häufigkeitstabelle nicht vollständig vorliegt, müssen wir von Hand rechnen. Nehmen wir an, in einer klinischen Studie wurde an 150 Patienten die Dekubitusprävalenz ermittelt. Dabei wurde eine Prävalenz von 35% festgestellt.
Die Konfidenzintervalle für den "wahren Wert" der Grundgesamtheit wird wie folgt bestimmt.

```
# Daten erzeugen
n <- 150
k <- 0.35

# Fehler berechnen mit (alpha/2)!
fehler95 <- qnorm(0.975)*sqrt(k*(1-k)/n)

# anzeigen
fehler95
```

```
0.07632963
```

Beachten Sie, dass in der Funktion `qnorm()` die Irrtumswahrscheinlichkeit α halbiert werden muss, da es sich um eine zweiseitige Fragestellung handelt. So muss `qnorm()` nicht mit dem Wert `0.95` ($\alpha = 0.05$) aufgerufen werden, sondern mit `0.975` ($\frac{\alpha}{2} = 0.025$).

```
# untere Grenze des KI
k - fehler95
```

```
0.2736704
```

Die untere Grenze des 95%-Konfidenzintervalls liegt bei `0.2736704` = `27,36704`%

```
# obere Grenze des KI
k + fehler95
```

```
[1] 0.4263296
```

Die obere Grenze des 95%-Konfidenzintervalls liegt bei `0.4263296` = `42,63296`%

```
# Länge des KI
2*fehler95
```

```
[1] 0.1526593
```

Das 95%-Konfidenzintervall hat eine Länge von `0.1526593` = `15,26593`%.

Für das 99%-Konfidenzintervall muss der Wert der Funktion `qnorm()` auf `0.995` ($\frac{\alpha}{2} = 0,005$) angepasst werden.

```
# Fehler berechnen mit (alpha/2)!
fehler99 <- qnorm(0.995)*sqrt(k*(1-k)/n)

# anzeigen
fehler99
```

```
[1] 0.1003141
```

```
# untere Grenze des KI
k - fehler99
```

```
0.2496859
```

Die untere Grenze des 99%-Konfidenzintervalls liegt bei `0.2496859` = `24,96859`%

```
# obere Grenze des KI
k + fehler99
```

```
0.4503141
```

Die obere Grenze des 99%-Konfidenzintervalls liegt bei `0.4503141` = `45,03141`%

```
# Länge des KI
2*fehler99
```

```
0.2006283
```

Das 99%-Konfidenzintervall hat eine Länge von `0.2006283` = `20,06283`%.

Die Funktion `KIbinomial_a()` aus dem `jgsbook`-Paket nimmt uns auch hier die vorgestellten Rechenschritte ab.

```
# Berechne das 95%-KI
jgsbook::KIbinomial_a(p=0.35, n=150, alpha=0.05)
```

```
                       x         y
1 0.95 KI untere Grenze 0.2736704
2  0.95 KI obere Grenze 0.4263296
3        0.95 KI Laenge 0.1526593
```

```
# Berechne das 95%-KI
jgsbook::KIbinomial_a(p=0.35, n=150, alpha=0.01)
```

```
                       x         y
1 0.99 KI untere Grenze 0.2496859
2  0.99 KI obere Grenze 0.4503141
3        0.99 KI Laenge 0.2006283
```

Unterschied von Anteilswerten

Zur Berechnung der Konfidenzintervalle bei **Unterschieden von Anteilswerten** wird die folgende Formel verwendet.

$$\left.\begin{matrix} d_o \\ d_u \end{matrix}\right\} = \hat{d} \pm c \cdot \sqrt{\frac{\hat{p}_1 \cdot (1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2 \cdot (1-\hat{p}_2)}{n_2}} \tag{2.4}$$

$\hat{d}$ = die Differenz der Anteile
n_1, n_2 = die Umfänge der beiden Stichproben
$\hat{p}_1$, $\hat{p}_2$ = die geschätzten Anteile in beiden Stichproben
c = Quartilswert für Irrtumswahrscheinlichkeit α

Liegen die erhobenen Daten als Kreuztabelle der absoluten Häufigkeiten vor, kann die Funktion `prop.test()` verwendet werden.

```
# Testwerte
antwort <- c("ja", "nein", "nein", "ja", "ja", "ja", "ja", "nein", "ja",
             "ja", "ja", "ja", "nein", "ja", "ja", "ja", "nein", "ja")
geschlecht <- c("w", "w", "m", "w", "m", "m", "m", "m", "w",
                "m", "w", "m", "w", "w", "m", "m", "w", "w")
# erstelle Kreuztabelle
freq <- xtabs(~ antwort + geschlecht)
# Tabelle anzeigen
freq
```

```
       geschlecht
  antwort m w
     ja   7 6
     nein 2 3
```

```
# Konfidenzintervall für Anteil "ja" zwischen "m" und "w"
prop.test(c(freq[["ja","m"]], freq[["ja","w"]]),
          c(sum(freq[,"m"]), sum(freq[,"w"])))
```

```
^^I2-sample test for equality of proportions with continuity correction

data:  c(freq[["ja", "m"]], freq[["ja", "w"]]) out of
       c(sum(freq[, "m"]), sum(freq[, "w"]))
X-squared = 0, df = 1, p-value = 1
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.4106382  0.6328604
sample estimates:
   prop 1    prop 2
0.7777778 0.6666667
```

Wenn alle Daten der Kreuztabelle vorliegen, können die Werte auch direkt an die Funktion übergeben werden. Angenommen wir wissen, dass von 9 Frauen 6 mit ”ja”geantwortet haben, und von 9 Männern 7, dann können wir das Konfidenzintervall für den Anteilsunterschied wie folgt berechnen:

```
# Unterschied im Anteil "ja"
prop.test(c(7, 6), c(9, 9))
```

```
^^I2-sample test for equality of proportions with continuity correction

data:  c(7, 6) out of c(9, 9)
X-squared = 0, df = 1, p-value = 1
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.4106382  0.6328604
sample estimates:
   prop 1    prop 2
0.7777778 0.6666667
```

Über die Zusatzparameter `$estimate` und `$conf.int` kann die Ausgabe auf die Konfidenzgrenzen beschränkt werden.

```
# Konfidenzintervall für Anteilsunterschied "ja"
prop.test(c(7, 6), c(9, 9))$conf.int
```

```
 [1] -0.4106382  0.6328604
 attr(,"conf.level")
 [1] 0.95
```

Standardmäßig wird das 95%-Intervall berechnet. Über den Parameter `conf.level` können beliebige Konfidenzniveaus angegeben werden.

```
# 99%-Intervall
prop.test(c(7, 6), c(9, 9), conf.level=0.99)
```

```
^^I2-sample test for equality of proportions with continuity correction

data:  c(7, 6) out of c(9, 9)
X-squared = 0, df = 1, p-value = 1
alternative hypothesis: two.sided
99 percent confidence interval:
 -0.5396700  0.7618923
sample estimates:
   prop 1    prop 2
0.7777778 0.6666667
```

Wenn wir keine absoluten Zahlen vorliegen haben, müssen wir von Hand rechnen. Nehmen wir an, in Bundesland A gaben 25% von 100 Befragten an, Zuzahlungen bei Pflegehilfsmitteln leisten zu müssen. In Bundesland B wurden 150 Personen befragt, von denen 40% von Zuzahlungen betroffen waren. Der Punktschätzwert für die Differenz $\hat{d}$ beträgt folglich 0,40-0,25= **0,15**.
Die Konfidenzintervalle für die wahre Differenz d kann nun wie folgt bestimmt werden.

```r
# Daten erzeugen
n1 <- 100
p1 <- 0.25

n2 <- 150
p2 <- 0.4

# Differenz der Anteile
d <- p2-p1
```

Beachten Sie, dass in der Funktion `qnorm()` die Irrtumswahrscheinlichkeit α halbiert werden muss, da es sich um eine zweiseitige Fragestellung handelt. So muss `qnorm()` nicht mit dem Wert `0.95` ($\alpha = 0.05$) aufgerufen werden, sondern mit `0.975` ($\frac{\alpha}{2} = 0.025$).

```r
# 95%KI-Fehler berechnen mit (alpha/2)
fehler95 <- qnorm(0.975)*sqrt(p1*(1-p1)/n1 + p2*(1-p2)/n2)

#anzeigen
fehler95
```

```
[1] 0.1155382
```

```r
# untere Grenze des 95%KI
d - fehler95
```

```
[1] 0.03446183
```

Die untere Grenze des 95%-Konfidenzintervalls liegt bei `0.03446183` = `3,446183`%

```r
# obere Grenze des 95%KI
d + fehler95
```

```
[1] 0.2655382
```

Die obere Grenze des 95%-Konfidenzintervalls liegt bei `0.2655382` = `26,55382`%

```
# Länge des KI
2*fehler95
```

```
[1] 0.2310763
```

Die Länge des 95%-Konfidenzintervalls liegt bei `0.2310763` = `23,10763`%

Für das 99%-Konfidenzintervall muss der Wert der Funktion `qnorm()` auf `0.995` ($\frac{\alpha}{2} = 0,005$) angepasst werden.

```
# 99%KI-Fehler berechnen mit (alpha/2)
fehler99 <- qnorm(0.995)*sqrt(p1*(1-p1)/n1 + p2*(1-p2)/n2)

#anzeigen
fehler99
```

```
[1] 0.1518429
```

```
# untere Grenze des 99%KI
d - fehler99
```

```
[1] -0.001842898
```

Die untere Grenze des 99%-Konfidenzintervalls liegt bei `-0.0018429` = `-0,18429`%

```
# obere Grenze des 99%KI
d + fehler99
```

```
[1] 0.3018429
```

Die obere Grenze des 99%-Konfidenzintervalls liegt bei `0.3018429` = `30,18429`%

```
# Länge des KI
2*fehler99
```

```
[1] 0.3036858
```

Die Länge des 99%-Konfidenzintervalls liegt bei `0.3036858` = `30,36858`%

Wie Sie sehen, liegt die untere Grenze des 99%-Konfidenzintervalls im negativen Bereich. Errechnet sich ein Intervall, welches die `0` einschließt, bedeutet dies, dass der wahre Unterschied auch `0` sein könnte.

Die Funktion `KIbinomial_u()` aus dem `jgsbook`-Paket nimmt uns auch hier die vorgestellten Rechenschritte ab.

```
# Berechne das 95%-KI
jgsbook::KIbinomial_u(p1, n1, p2, n2, alpha=0.05)
```

```
                       x          y
1 Differenz der Anteile 0.15000000
2 0.95 KI untere Grenze 0.03446183
3  0.95 KI obere Grenze 0.26553817
4        0.95 KI Laenge 0.23107635
```

```
# Berechne das 99%-KI
jgsbook::KIbinomial_u(p1, n1, p2, n2, alpha=0.01)
```

```
                       x           y
1 Differenz der Anteile  0.150000000
2 0.99 KI untere Grenze -0.001842898
3  0.99 KI obere Grenze  0.301842898
4        0.99 KI Laenge  0.303685796
```

2.2.7 Signifikanztests

2.2.7.1 χ^2-Test

Der χ^2-Test wird mit der Funktion `chisq.test()` durchgeführt. Hierfür müssen die Daten als Kreuztabelle vorliegen.

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/nw.RData"))

# erstelle Kreuztabelle aus Alter und Geschlecht
xtabelle <- xtabs(~ nw$age + nw$sex)

# Führe Chiquadrat-Test durch
chisq.test(xtabelle)
```

```
^^IPearson's Chi-squared test

data:  xtabelle
X-squared = 43.886, df = 44, p-value = 0.4765

Warnmeldung:
In chisq.test(xtabelle) : Chi-Quadrat-Approximation kann inkorrekt sein
```

Der p-Wert ist nicht signifikant, das heisst, das kein Zusammenhang gefunden werden konnte. Die Warnung bedeutet, dass der p-Wert aus sehr kleinen Zellenhäufigkeiten (kleiner 5) geschätzt wurde, und daher fehlerhaft sein kann. Wir sollten daher einen *exakten* Test durchführen, zum Beispiel den *Fisher-Test*.

Ein gutes Anleitungsvideo zum χ^2-Test findet sich bei Youtube:
https://www.youtube.com/watch?v=4lFyRcXoJB8

2.2.7.2 Fisher's Exakttest

Der *exakte Fisher-Test* wird mit der Funktion `fisher.test()` durchgeführt.

```
# Führe Fisher-Test durch
fisher.test(xtabelle)
```

```
Fisher's Exact Test for Count Data with simulated p-value (based on 2000 replicates)

data:  nw$sex and nw$age
p-value = 0.3848
alternative hypothesis: two.sided
```

Der exakte Fisher-Test bestätigt, dass es keinen Unterschied gibt.

2.2.7.3 t-Test

Der t-Test ist wohl einer der bekanntesten Signifikanztests. In  wird er mit der Funktion `t.test()` ausgeführt.

2.2.7.3.1 Einstichproben t-Test

Beim Einstichproben t-Test wird der Mittelwert der Grundgesamtheit über den Parameter `mu` mitgegeben.

Schauen wir, ob im Nachtwachendatensatz das Alter von einem gedachten Mittelwert `40` in der Grundgesamtheit abweicht.

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/nw.RData"))

# Führe t-Test für "Alter" auf Mittelwert 40 durch
t.test(nw$age, mu=40)
```

```
^^IOne Sample t-test

data:  nw$age
t = 6.8274, df = 275, p-value = 5.524e-11
alternative hypothesis: true mean is not equal to 40
95 percent confidence interval:
 43.00908 45.44744
sample estimates:
mean of x
 44.22826
```

Der Test ist signifikant, das bedeutet, das Alter im Datensatz `nw` weicht von einem Mittelwert in der Grundgesamtheit von 40 ab. Es liegt ein Unterschied vor.

Zur Beschreibung der Effekt*stärke* des Unterschieds können wir *Cohen's* `d` berechnen. Dies kann über das Zusatzpaket `lsr` und der Funktion `cohensD()` erfolgen.

```
# aktiviere Zusatzpaket "lsr"
library(lsr)

# berechne Effektsärke Cohen's D
cohensD(nw$age, mu=40)
```

```
[1] 0.4109627
```

Dieser Wert kann nun nach Cohen's Angaben[9] interpretiert werden (0.2=kleiner Effekt, 0.5=mittel, 0.8=stark). Unsere Effektgröße ist eher "mittel".

Ein gutes Anleitungsvideo zum Einstichproben t-Test findet sich bei Youtube:
https://www.youtube.com/watch?v=7vcyjuSwfzI

2.2.7.3.2 unabhängiger t-Test

Für den Vergleich der Mittelwerte bei unabhängige Stichproben werden die Werte beider Gruppen als Variable übergeben. So können wir z.B. schauen, ob sich die Datensätze `nw` und `epa` hinsichtlich des Alters unterscheiden

```
# t-test auf Alter in Datensatz "epa" und "nw"
t.test(nw$age, epa$age)
```

```
^^IWelch Two Sample t-test

data:  nw$age and epa$age
t = -19.316, df = 854.37, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -20.65610 -16.84545
sample estimates:
mean of x mean of y
 44.22826  62.97903
```

Das Ergebnis ist signifikant, die Gruppen unterscheiden sich also.

Standardmäßig wird ein zweiseitiger Test durchgeführt. Über den Parameter `alternative` kann dies geändert werden.

```
# t-test einseitig, kleiner
t.test(nw$age, epa$age, alternative = "less")
# t-test einseitig, größer
t.test(nw$age, epa$age, alternative = "greater")
# t-test zweiseitig
t.test(nw$age, epa$age, alternative = "two.sided")
```

Die Funktion nimmt auch die Syntax *erklärt durch* entgegen. Wir schauen, ob sich das Alter im Datensatz `epa` in den Geschlechtergruppen unterscheidet, also *Alter erklärt durch Geschlecht*.

```
# t-test Alter zwischen Geschlechtern im Datensatz "epa"
t.test(age ~ sex, data=epa)
```

```
^^IWelch Two Sample t-test

data:  age by sex
t = -4.7535, df = 595.03, p-value = 2.51e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -9.923268 -4.120822
sample estimates:
mean in group m mean in group w
       59.79646        66.81851
```

Das Ergebnis ist signifikant. Auch hier lässt sich wieder die Effekt*stärke* Cohen's D berechnen.

```
# Effektsärke
lsr::cohensD(age ~ sex, data=epa)
```

```
[1] 0.3837919
```

Der Effekt ist mit 0,38 eher klein.

Ein gutes Anleitungsvideo zum unabhängigen t-Test findet sich bei Youtube:
https://www.youtube.com/watch?v=3Yz-XNRasGM

2.2.7.3.3 abhängiger t-Test

Bei gepaarten Werten muss ein abhängiger t-Test berechnet werden. Dies erfolgt über den Parameter `paired`, der auf `TRUE` gesetzt wird. Auch hier können wir wieder die Richtung (größer, kleiner, zweiseitig) über den Parameter `alternative` verändern.

```
# erzeuge Testwerte
t0 <- rnorm(100, mean=12, sd=1)
t1 <- rnorm(100, mean=11, sd=3)

# gepaarter t-Test
t.test(t0, t1, paired=TRUE)

# gepaarter  t-test einseitig, kleiner
t.test(t0, t1, paired=TRUE, alternative = "less")
# gepaarter t-test einseitig, größer
t.test(t0, t1, paired=TRUE, alternative = "greater")
# gepaarter t-test zweiseitig
t.test(t0, t1, paired=TRUE, alternative = "two.sided")
```

```
^^IPaired t-test

data:  t0 and t1
t = 3.012, df = 99, p-value = 0.003295
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.3195899 1.5536199
sample estimates:
mean of the differences
              0.9366049
```

Die Effektstärke Cohen's D lässt sich entsprechend berechnen, indem der Parameter `method` auf `paired` gesetzt wird:

```
# Effektsärke
lsr::cohensD(t0, t1, method="paired")
```

```
[1] 0.3258625
```

Der Effekt ist mit 0,33 eher gering.

Ein gutes Anleitungsvideo zum verbundenen t-Test findet sich bei Youtube:
https://www.youtube.com/watch?v=H15M_8gh1Ok

2.2.7.4 F-Test

Der F-Test wird mit der Funktion `var.test()` berechnet.
So können wir prüfen, ob die Varianz des Alters im Datensatz `epa` in den Geschlechtergruppen unterschiedlich ist.

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/epa.RData"))

# F-Test
var.test(age ~ sex , data=epa)
```

```
^^IF test to compare two variances

data:  age by sex
F = 0.98352, num df = 338, denom df = 280, p-value = 0.8816
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.7848842 1.2294811
sample estimates:
ratio of variances
          0.983521
```

Der Test ist nicht signifikant, es liegt kein Unterschied in den Varianzen vor.

2.2.7.5 Levene-Test

Ebenso kann mit dem Levene-Test auf Varianzunterschied geprüft werden. Hierfür rufen wir die Funktion `leveneTest()` aus dem Zusatzpaket `car` auf.

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/epa.RData"))

# Levene-Test
car::leveneTest(epa$age, epa$sex)
```

```
Levene's Test for Homogeneity of Variance (center = median)
       Df F value Pr(>F)
group   1  0.0232 0.8789
      618
```

Der Test ist nicht signifikant, es liegt kein Unterschied in den Varianzen vor.

Ein gutes Anleitungsvideo zum verbundenen Levene-Test findet sich bei Youtube:
https://www.youtube.com/watch?v=717UWGnyQGA

2.2.7.6 Mann-Whitney-U-Test

Dies ist die nicht-parametrische alternative zum t-Test für unabhängige Stichproben. Er wird mit der Funktion `wilcox.test()` durchgeführt, wobei der Parameter `paired` auf `FALSE` gesetzt wird (ist die Standardeinstellung).

```
# lade Testdaten
load(url("https://www.produnis.de/R/data/epa.RData"))

# Man-Whitney-U-Test
wilcox.test(epa$age ~ epa$sex)
```

```
^^IWilcoxon rank sum test with continuity correction

data:  epa$age by epa$sex
W = 35862, p-value = 1.151e-07
alternative hypothesis: true location shift is not equal to 0
```

Das Ergebnis ist signifikant, das Alter im Datensatz `epa` unterscheidet sich zwischen den Geschlechtern.

Mit dem Parameter `conf.int` kann das Konfidenzintervall des Schätzwertes mit ausgegeben werden.

```
# Man-Whitney-U-Test mit Konfidenzintervall
wilcox.test(epa$age ~ epa$sex, conf.int=TRUE)
```

```
^^IWilcoxon rank sum test with continuity correction

data:  epa$age by epa$sex
W = 35862, p-value = 1.151e-07
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
 -9.999986 -4.999952
sample estimates:
difference in location
             -7.000029
```

Ein gutes Anleitungsvideo zum Mann-Whitney-U-Test finden Sei bei Youtube:
https://www.youtube.com/watch?v=4lFyRcXoJB8

2.2.7.7 Wilcoxon-Test

Dies ist die nicht-parametrische alternative zum abhängigen t-Test. Er wird mit der Funktion `wilcox.test()` durchfegührt, wobei der Parameter `paired` auf `TRUE` gesetzt werden muss. Andernfalls rechnet R einen Man-Whitney-U-Test.

```
# erzeuge Testwerte
t0 <- sample(70:140, 100, replace=T)
t1 <- sample(74:170, 100, replace=T)

wilcox.test(t0, t1, paired=TRUE)
```

```
^^IWilcoxon signed rank test with continuity correction

data:  t0 and t1
V = 1036.5, p-value = 1.435e-06
alternative hypothesis: true location shift is not equal to 0
```

Mit dem Parameter `conf.int` kann das Konfidenzintervall des Schätzwertes mit ausgegeben werden.

```
# erzeuge Testwerte
t0 <- sample(70:140, 100, replace=T)
t1 <- sample(74:170, 100, replace=T)

wilcox.test(t0, t1, paired=TRUE, conf.int=TRUE)
```

```
^^IWilcoxon signed rank test with continuity correction

data:  t0 and t1
V = 1036.5, p-value = 1.435e-06
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
 -26.99994 -12.50000
sample estimates:
(pseudo)median
         -19.5
```

Ein gutes Anleitungsvideo zum Wilcoxon-Test finden Sie bei Youtube:
https://www.youtube.com/watch?v=4lFyRcXoJB8

2.2.7.8 Shapiro-Wilk-Normalverteilungstest

Zum Test auf Normalverteilung kann der *Shapiro-Wilk*-Test über die Funktion `shapiro.test()` verwendet werden. Die Nullhypothese lautet, *dass* eine Normalverteilung vorliegt. Testen wir, ob im Datensatz `nw` das Alter (`age`) normalverteilt ist.

```
1  # lade Testdaten
2  load(url("https://www.produnis.de/R/data/nw.RData"))
3
4  # Teste "age" auf Normalverteilung
5  shapiro.test(nw$age)
```

```
^^IShapiro-Wilk normality test

data:  nw$age
W = 0.96636, p-value = 4.677e-06
```

Das Ergebnis ist signifikant, die Daten sind **nicht** normalverteilt.

Erzeugen wir normalverteilte Werte zur Gegenprobe.

```
1  # Test mit 100 zufälligen und normalverteilten Werten
2  shapiro.test(rnorm(100))
```

```
^^IShapiro-Wilk normality test

data:  rnorm(100)
W = 0.98488, p-value = 0.3115
```

Das Ergebnis ist (wie erwartet) nicht signifikant, das heisst, unsere Zufallszahlen sind wirklich normalverteilt.

Der Shapiro-Wilk-Test funktioniert nur bis zu einer Stichprobengröße von $n = 5000$. Besteht der Datensatz aus mehr Fällen, kann auf den Kolmogorov-Smirnov-Test ausgewichen werden.

2.2.7.9 Kolmogorov-Smirnov-Test

Der Kolmogorov-Smirnov-Test prüft die Nullhypothese, dass die Daten einer bestimmten Verteilung (z.B. der Normalverteilung) folgen. In R ist er über die Funktion `ks.test()` implementiert.
Prüfen wir nun also mittels Kolmogorov-Smirnov-Test, ob im Datensatz `nw` das Alter (`age`) normalverteilt ist.

```
# lade nw Datensatz
load(url("https://www.produnis.de/R/data/nw.RData"))

# Teste, ob Alter normalverteilt ist
ks.test(nw$age, "pnorm")
```

```
    Asymptotic one-sample Kolmogorov-Smirnov test

    data:  nw$age
    D = 1, p-value < 2.2e-16
    alternative hypothesis: two-sided
```

Der Test ist signifikant, das heisst, es liegt **keine** Normalverteilung vor.
Die Testparameter können zudem auf die Lage- und Streuungskenngrößen der Wertereihe angepasst werden.

```
# passe auf Kenngrößen an
ks.test(nw$age, "pnorm",
        mean = mean(nw$age, na.rm = TRUE),
        sd = sd(nw$age, na.rm = TRUE))
```

```
    Asymptotic one-sample Kolmogorov-Smirnov test

    data:  nw$age
    D = 0.11477, p-value = 0.001391
    alternative hypothesis: two-sided
```

Auch dieser Test ist signifikant, das heisst, es liegt keine Normalverteilung vor.
Machen wir nun den Gegentest mit normalverteilten Werten:

```
# erzeuge normalverteilte Dummywerte
dummy <- rnorm(10000)

# Test auf Normalverteilung
ks.test(dummy, "pnorm",
        mean = mean(dummy),
        sd = sd(dummy))
```

```
    Asymptotic one-sample Kolmogorov-Smirnov test

    data:  dummy
    D = 0.0063771, p-value = 0.8108
    alternative hypothesis: two-sided
```

Der Test ist nicht signifikant, das heisst, es liegen normalverteilte Werte vor.

2.2.8 Post-Hoc-Tests

Post-hoc-Tests werden verwendet, um paarweise Vergleiche nach einem initialen Test (wie ANOVA (siehe Kapitel 2.2.9)) durchzuführen, der signifikante Unterschiede zwischen Gruppenmittelwerten anzeigt. Durch die Mehrfachvergleiche (multiples Testen) entsteht das erhöhte Risiko von falsch-positiven Ergebnissen (Typ-I-Fehler).
Post-Hoc-Tests helfen dabei, spezifische Gruppen zu identifizieren, die sich voneinander unterscheiden, während sie das Problem der Mehrfachtestung berücksichtigen.
R bietet mehrere Funktionen zur Durchführung von Post-hoc-Tests. In diesem Abschnitt bschränken wir uns auf die weit verbreiteten Funktionen `reporttools::pairwise.fisher.test()`, `pairwise.wilcox.test()` und `pairwise.t.test()`.

2.2.8.1 Paarweise Chiquadrat

In R-base ist keine Funktion implementiert, die einen paarweisen Chiquadrat-Test durchführt. Das mag daran liegen, dass der Chiquadrat-Test als (weniger genaue) Alternative zu dem rechenaufwändigen exakten Fisher-Test entwickelt wurde. Mit der heutigen Computertechnologie ist es jedoch in den meisten Fällen kein Problem, den exakten Test nach Fisher auch auf älterer Hardware anzuwenden.

2.2.8.2 Paarweise Fisher-Test

Der `pairwise.fisher.test()` aus dem Paket `reporttools` wird verwendet, um paarweise Vergleiche zwischen Gruppen durchzuführen, wenn nonimale Daten vorliegen.
Als Beispiel untersuchen wir im Datensatz `pf8`, ob sich die Geschlechter je nach Standort unterscheiden.

```
1  # verwende pf8 Datensatz
2  pf8 <- jgsbook::pf8
3
4  # führe exakten Test durch
5  reporttools::pairwise.fisher.test(pf8$Geschlecht, pf8$Standort, p.adjust.method = "holm")
```

```
    Pairwise comparisons using

    data:  pf8$Geschlecht and pf8$Standort

              Rheine  Münster Bahn    Ladbergen
    Münster   1.00000 -       -       -
    Bahn      1.00000 1.00000 -       -
    Ladbergen 0.56615 0.24097 0.32261 -
    Internet  0.00262 0.02282 0.09245 0.00065

    P value adjustment method: holm
```

Als Ergebnis erhalten wir eine Tabelle mit p-Werten. Es ist zu erkennen, dass sich die Geschlechterverteilungen zwischen den Standortvergleichen `Internet-Rheine`, `Internet-Münster` und `Internet-Landbergen` unterscheiden.

2.2.8.3 Paarweise Wilcoxon-Test (Mann-Whitney-U)

Die Funktion `pairwise.wilcox.test()` wird verwendet, um paarweise Vergleiche zwischen Gruppen durchzuführen, wenn die Daten nicht normalverteilt sind. Dieser Test basiert auf dem Wilcoxon-Rangsummentest (Mann-Whitney-U-Test), und ist eine nichtparametrische Methode, die keine Annahmen über die Verteilung der Daten erfordert.
Als Beispiel untersuchen wir im Datensatz `pf8`, ob sich das Alter der Probanden je nach Standort unterscheidet.

```
# verwende pf8 Datensatz
pf8 <- jgsbook::pf8

# führe paarweisen Wilcox-Test durch
pairwise.wilcox.test(pf8$Alter, pf8$Standort, p.adjust.method = "bonferroni")
```

```
    Pairwise comparisons using Wilcoxon rank sum test with continuity correction

    data:  pf8$Alter and pf8$Standort

              Rheine  Münster Bahn    Ladbergen
    Münster   1.00000 -       -       -
    Bahn      0.55704 1.00000 -       -
    Ladbergen 0.56286 1.00000 1.00000 -
    Internet  0.02780 3.5e-05 0.00027 0.00062

    P value adjustment method: bonferroni
```

Als Ergebnis erhalten wir eine Tabelle mit p-Werten. Es ist zu erkennen, dass sich die Altersverteilungen zwischen den Standortvergleichen `Internet-Rheine`, `Internet-Münster` und `Internet-Landbergen` unterscheiden.

2.2.8.4 Paarweise t-Test

Die Funktion `pairwise.t.test()` wird verwendet, um paarweise Vergleiche zwischen Gruppen durchzuführen, wenn die Daten normalverteilt sind.

```
# erzeuge Testdaten
Gruppe <- factor(rep(c("Handball", "Fussball", "Volleyball"), each = 10))
Punkte <- c(rnorm(10, mean = 5), rnorm(10, mean = 6), rnorm(10, mean = 7))

# Führe paarweise t-Tests durch
pairwise.t.test(Punkte, Gruppe, p.adjust.method = "holm")
```

```
    Pairwise comparisons using t tests with pooled SD

    data:  Punkte and Gruppe

               Fussball Handball
    Handball   0.0027   -
    Volleyball 0.0551   2e-05

    P value adjustment method: holm
```

Als Ergebnis erhalten wir eine Tabelle mit p-Werten. Es ist zu erkennen, dass sich die Punkteverteilungen zwischen `Handball-Fussball`, und `Handball-Volleyball` unterscheiden.

2.2.9 Varianzanalysen

2.2.9.1 ANOVA mit Messwiederholung

Eine ANOVA mit Messwiederholung kann wie folgt in R durchgeführt werden. Wir nutzen den Testdatensatz `Messwiederholung.rda`.

```
load(url("https://www.produnis.de/R/data/Messwiederholung.rda"))
```

Der Datensatz besteht aus 200 Probanden, bei denen zu drei Zeitpunkten ein Wert erhoben wurde.

```
head(Messwiederholung)
```

```
     Name        t1       t2       t3
1  Brycen 11.728110 8.190290 6.202859
2   Waail  9.781605 8.646142 6.311455
3 Abigail 11.937967 9.386623 4.577478
4   Holly  4.597950 8.204762 7.250849
5  Sheila  8.277670 8.322449 6.952457
6 Rebekah  7.891258 9.104856 7.191639
```

Zunächst müssen wir die Daten ins 'long table'-Format überführen.

```
# Überführe die Daten ins "long table"-Format
df <- Messwiederholung %>%
  pivot_longer(cols=t1:t3,
               names_to = "Zeitpunkt",
               values_to = "Wert") %>%
  mutate(Zeitpunkt = factor(Zeitpunkt),
         Name = factor(Name))

# anschauen
head(df)
```

```
# A tibble: 6 × 3
  Name   Zeitpunkt  Wert
  <fct>  <fct>     <dbl>
1 Brycen t1        11.7
2 Brycen t2         8.19
3 Brycen t3         6.20
4 Waail  t1         9.78
5 Waail  t2         8.65
6 Waail  t3         6.31
```

Zunächst verschaffen wir uns einen deskriptiven Überblick über die Daten.

```
# mal deskriptiv anschauen
df %>%
  group_by(Zeitpunkt) %>%
  summarise(M = mean(Wert),
            SD = sd(Wert),
            Q1 = quantile(Wert, probs = 0.25),
            Q2 = quantile(Wert, probs = 0.5),
            Q3 = quantile(Wert, probs = 0.75),
            IQR = IQR(Wert),
            N = n())
```

```
# A tibble: 3 × 8
  Zeitpunkt     M    SD    Q1    Q2    Q3   IQR     N
  <fct>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
1 t1         9.88 2.14   8.41  9.77 11.3   2.91   200
2 t2         7.99 0.935  7.24  8.09  8.60  1.36   200
3 t3         6.92 0.868  6.31  6.89  7.51  1.21   200
```

Das geht auch graphisch:

```
# mal graphisch anschauen
boxplot(df$Wert ~ df$Zeitpunkt)
```

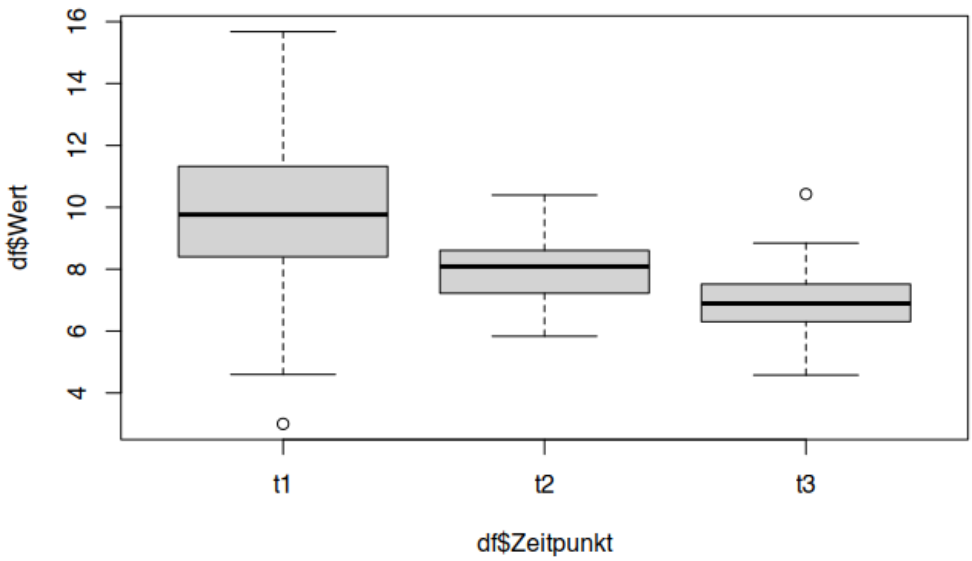


Abb. 2.18

```
ez::ezPlot(df, Wert, Name, within = Zeitpunkt, x = Zeitpunkt)
```

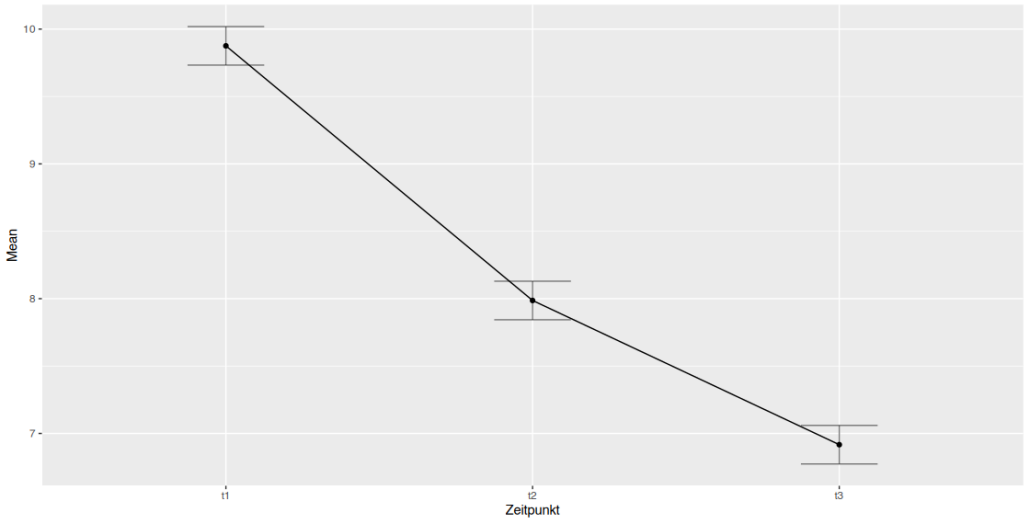


Abb. 2.19

```
# mit Regressionsgerade
ez::ezPlot(df, Wert, Name, within = Zeitpunkt, x = Zeitpunkt) +
  geom_smooth(aes(x = as.numeric(Zeitpunkt)), method = 'lm', se = F)
```

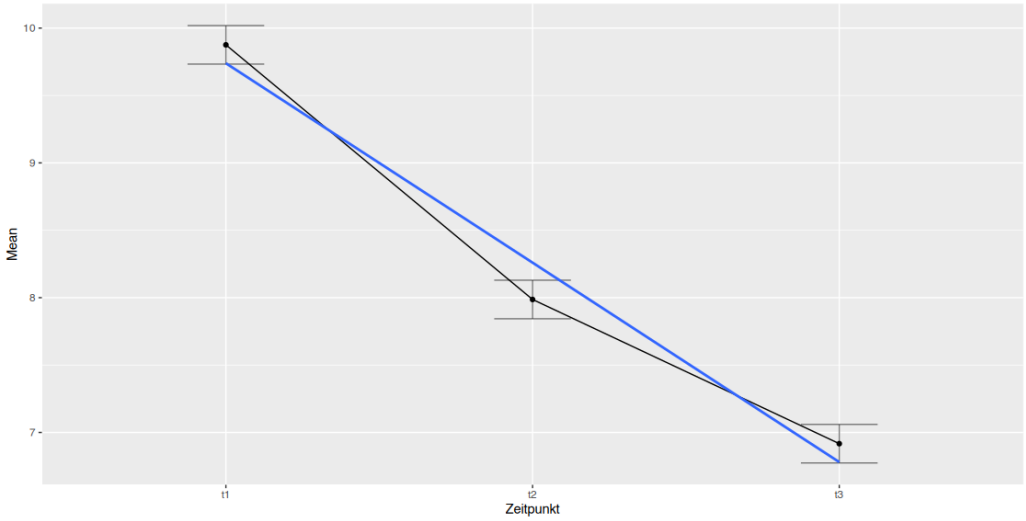


Abb. 2.20

Die Plots legen nahe, dass es einen Unterschied in den Zeitpunktgruppen gibt.
Wir überprüfen mittels Shapiro-Test und QQ-Plot, ob die Daten normalverteilt sind.

```
## qq-Plot (Normalverteilung)
ggpubr::ggqqplot(df, x="Wert",
                 facet.by = "Zeitpunkt",
                 color = "Zeitpunkt",
                 scales = "free")
```

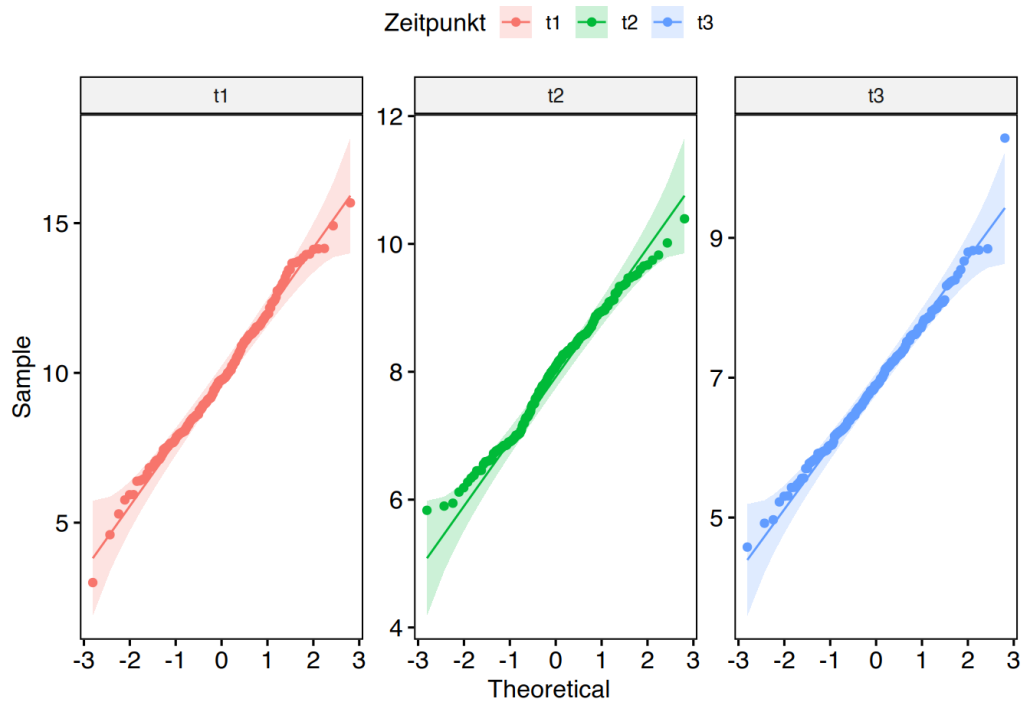


Abb. 2.21

```
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t1"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t1"]
W = 0.99364, p-value = 0.5481
```

```
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t2"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t2"]
W = 0.98766, p-value = 0.08025
```

```
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t3"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t3"]
W = 0.99116, p-value = 0.2625
```

Die Testergebnisse sind nicht signifikant, also liegt Normalverteilung vor.
Wir dürfen eine ANOVA mit Messwiederholung rechnen. Mit -Hausmittel könnten wir so vorgehen:

```
# ANOVA mit Messwiederholung
fit <- aov(Wert ~ Zeitpunkt + Error(Name), data=df)
summary(fit)
```

```
Error: Name
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 199  395.8   1.989

Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
Zeitpunkt   2  898.0   449.0   212.3 <2e-16 ***
Residuals 398  841.6     2.1
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Das Ergebnis ist signifikant. Mit der Funktion `pairwise.t.test()` können wir nun schauen, welche Gruppenunterschiede es konkret gibt:

```
# geht auch mit R-Hausmitteln so:
pairwise.t.test(df$Wert, df$Zeitpunkt,
                p.adjust="bonferroni")
```

```
^^IPairwise comparisons using t tests with pooled SD

data:  df$Wert and df$Zeitpunkt

   t1      t2
t2 < 2e-16 -
t3 < 2e-16 1.1e-12

P value adjustment method: bonferroni
```

Die Unterschiede sind zwischen allen Gruppen signifikant.

Das Zusatzpaket `rstatix` bietet mit der Funktion `anova_test()` eine alternative Vorgehensweise.

```
# ANOVA mit Messwiederholung
fit <- rstatix::anova_test(data = df, dv = Wert, wid = Name,
                           within = Zeitpunkt, effect.size = "pes")

# auf Sphärizität prüfen
fit$`Mauchly's Test for Sphericity`
```

```
     Effect     W        p p<.05
1 Zeitpunkt 0.668 4.29e-18     *
```

Mauchly's Test ist signifikant, das heisst, es liegt *keine* Sphärizität vor. R hat in diesem Falle bereits die Freiheitsgrade automatisch korrigiert (mittels Greenhouse-Geisser-Verfahren). Schauen wir uns die ANOVA an:

```
rstatix::get_anova_table(fit)Das geht auch graphisch wie oben beschrieben:
```

```
ANOVA Table (type III tests)

     Effect DFn    DFd      F        p p<.05   pes
1 Zeitpunkt 1.5 298.73 212.34 2.18e-48     * 0.516
```

Die ANOVA ist signifikant. Jetzt können wir analysieren, welche konkreten Gruppenunterschiede vorliegen.

```
# post-hoc Analyse: paarweise Gruppenvergleiche
df %>%
  rstatix::pairwise_t_test(Wert ~ Zeitpunkt, paired = TRUE,
                           p.adjust.method = "bonferroni") %>%
  as.data.frame()
```

```
   .y. group1 group2  n1  n2 statistic  df        p    p.adj p.adj.signif
1 Wert     t1     t2 200 200  11.24384 199 5.02e-23 1.51e-22         ****
2 Wert     t1     t3 200 200  18.28475 199 1.79e-44 5.37e-44         ****
3 Wert     t2     t3 200 200  11.26870 199 4.23e-23 1.27e-22         ****
```

Die Unterschiede sind zwischen allen Gruppen signifikant.
Die Effektstärke kann wie folgt berechnet werden.

```
# Effektstärke (nur solche, die post-hoc signifikant sind, interpretieren)
df %>%
  rstatix::cohens_d(Wert ~ Zeitpunkt, paired = TRUE) %>%
  as.data.frame()
```

```
   .y. group1 group2   effsize  n1  n2 magnitude
1 Wert     t1     t2 0.7950598 200 200  moderate
2 Wert     t1     t3 1.2929274 200 200     large
3 Wert     t2     t3 0.7968171 200 200  moderate
```

Achtung, wir dürfen nur solche Werte interpretieren, die oben signifkant waren. In unserem Beispiel sind alle Werte signifikant, darum dürfen wir alle Effekte interpretieren.

2.2.9.2 Friedman ANOVA

Sind die Werte nicht normalverteilt, kann ein Friedman-ANOVA gerechnet werden. Wir nutzen den Testdatensatz `MarioANOVA.rda`. Er enthält Informationen über 47 Super Mario Charaktere, deren Fehlerquote beim Spielen eines schweren Levels zu 3 Zeitpunkten erfasst wurde.

```
load(url("https://www.produnis.de/R/data/MarioANOVA.rda"))

# anschauen
head(MarioANOVA)
```

```
# A tibble: 6 × 8
  Name              Alter Kingdom          Geschlecht BadGuy    t1    t2    t3
  <fct>             <dbl> <fct>            <fct>      <lgl>  <dbl> <dbl> <dbl>
1 Mario                35 Mushroom Kingdom männlich   FALSE  0.372 0.472 0.257
2 Luigi                35 Mushroom Kingdom männlich   FALSE  1.27  1.34  1.14
3 Prinzessin Peach     25 Mushroom Kingdom weiblich   FALSE  6.15  1.40  1.98
4 Bowser               45 Bowser's Castle  männlich   TRUE   4.85  0.648 1.27
5 Toad                 20 Mushroom Kingdom männlich   FALSE  1.34  0.174 1.77
6 Toadette             19 Mushroom Kingdom weiblich   FALSE  0.727 0.413 2.09
```

In den Variablen `t1` bis `t3` liegen die Werte der 3 Messzeitpunkte. Auch hier überführen wir die Daten ins `long table`-Format.

```
df <- MarioANOVA %>%
  select(Name, t1:t3) %>%
  pivot_longer(cols=t1:t3,
               names_to = "Zeitpunkt",
               values_to = "Wert")
```

Schauen wir uns die Daten zunächst deskriptiv an.

```
# mal deskriptiv anschauen
df %>%
  group_by(Zeitpunkt) %>%
  summarise(M = mean(Wert),
            SD = sd(Wert),
            Q1 = quantile(Wert, probs = 0.25),
            Q2 = quantile(Wert, probs = 0.5),
            Q3 = quantile(Wert, probs = 0.75),
            IQR = IQR(Wert),
            N = n())
```

```
# A tibble: 3 × 8
  Zeitpunkt     M    SD    Q1    Q2    Q3   IQR     N
  <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
1 t1        2.75  4.61  0.750 1.34   2.46 1.71     47
2 t2        0.762 0.585 0.356 0.509  1.15 0.791    47
3 t3        0.901 0.701 0.266 0.650  1.34 1.07     47
```

Das geht auch graphisch wie oben beschrieben:

```
# mal graphisch anschauen
boxplot(df$Wert ~ df$Zeitpunkt)
```

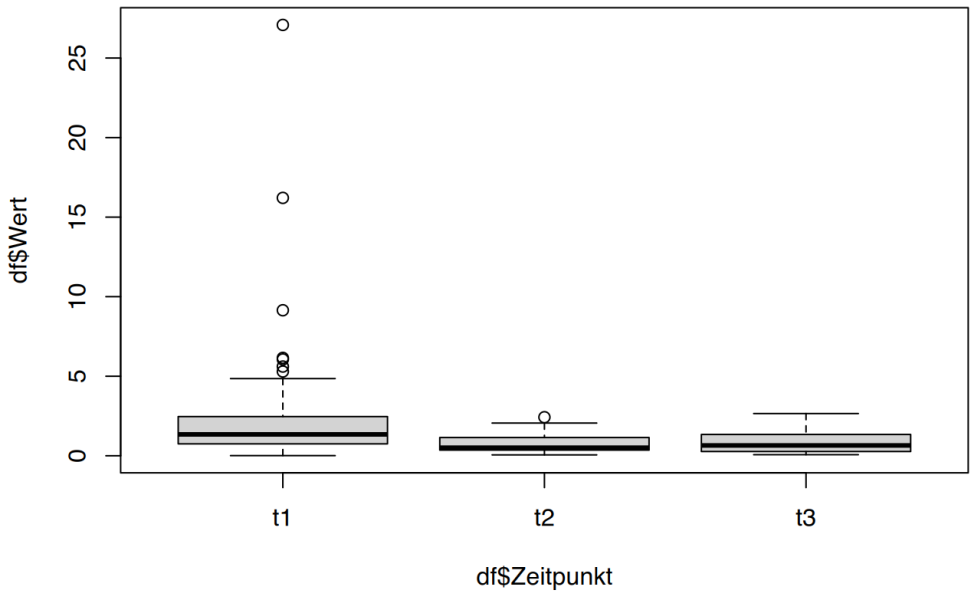


Abb. 2.22

```
ez::ezPlot(df, Wert, Name, within = Zeitpunkt, x = Zeitpunkt)
```

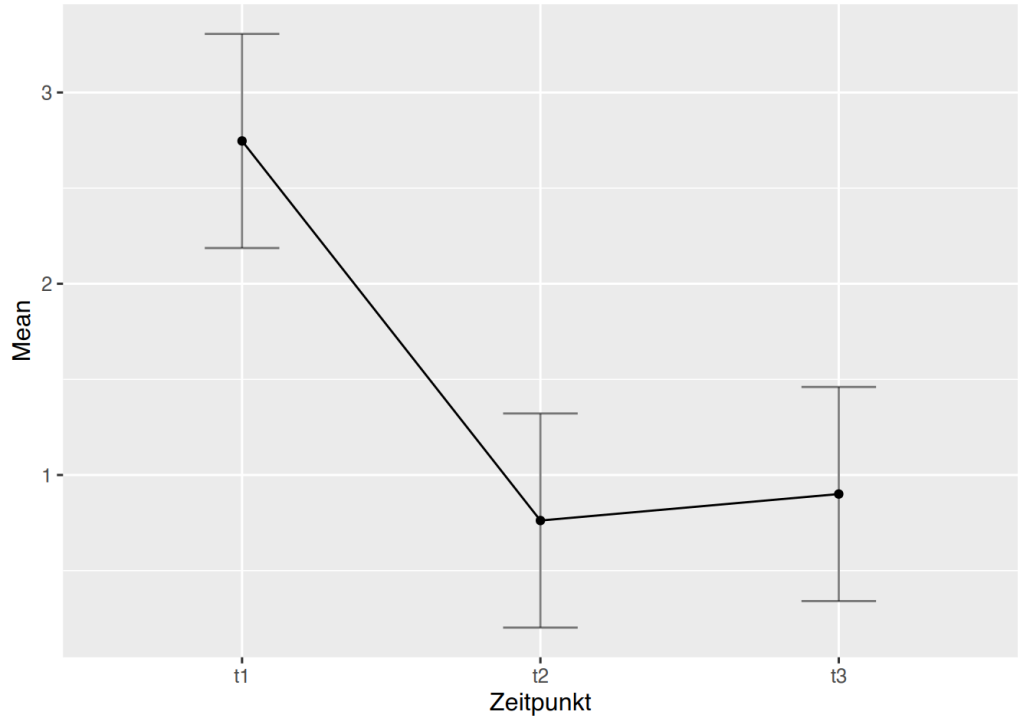


Abb. 2.23

```
# mit Regressionsgerade
ez::ezPlot(df, Wert, Name, within = Zeitpunkt, x = Zeitpunkt) +
  geom_smooth(aes(x = as.numeric(Zeitpunkt)), method = 'lm', se = F)
```

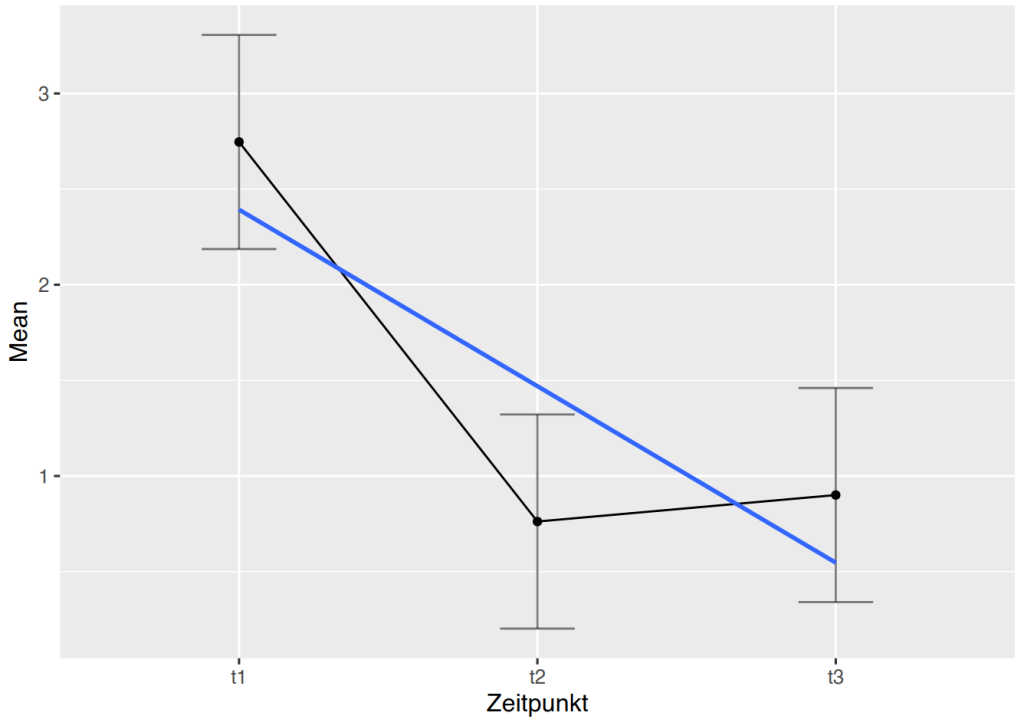


Abb. 2.24

Nun prüfen wir mittels QQ-Plot und Shapiro-Test auf Normalverteilung.

```
## qq-Plot (Normalverteilung)
ggpubr::ggqqplot(df, x="Wert",
                 facet.by = "Zeitpunkt",
                 color = "Zeitpunkt",
                 scales = "free")
```

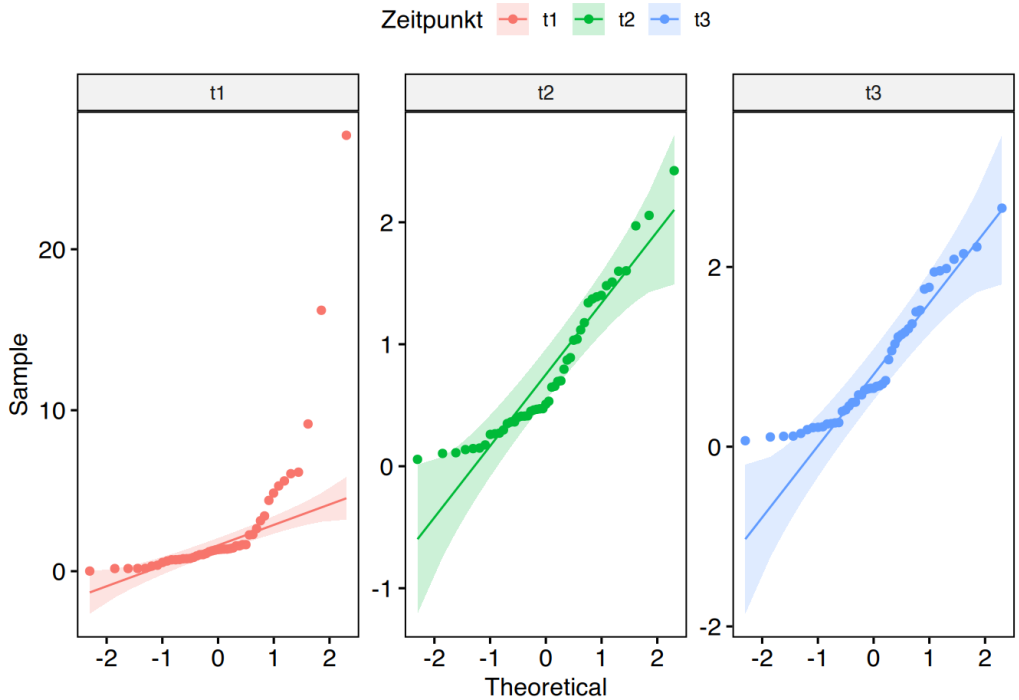


Abb. 2.25

```r
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t1"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t1"]
W = 0.52189, p-value = 3.729e-11
```

```r
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t2"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t2"]
W = 0.89427, p-value = 0.0004751
```

```r
# Teste "Wert" auf Normalverteilung
shapiro.test(df$Wert[df$Zeitpunkt=="t3"])
```

```
^^IShapiro-Wilk normality test

data:  df$Wert[df$Zeitpunkt == "t3"]
W = 0.90101, p-value = 0.0007756
```

Die Werte für `t1` bis `t3` sind signifikant, also **nicht** normalverteilt.

Wir führen daher eine Friedman-ANOVA mit Messwiederholung durch.

```r
rstatix::friedman_test(Wert ~ Zeitpunkt | Name, data=df)
```

```
# A tibble: 1 × 6
  .y.       n statistic    df       p method
* <chr> <int>     <dbl> <dbl>   <dbl> <chr>
1 Wert     47      12.8     2 0.00165 Friedman test
```

Das Ergebnis ist signifikant, es gibt also einen Unterschied.

```r
# post-hoc-test
df %>%
  rstatix::wilcox_test(Wert ~ Zeitpunkt, paired=T,
                       p.adjust.method = "bonferroni")
```

```
# A tibble: 3 × 9
  .y.   group1 group2    n1    n2 statistic        p    p.adj p.adj.signif
* <chr> <chr>  <chr>  <int> <int>     <dbl>    <dbl>    <dbl> <chr>
1 Wert  t1     t2        47    47       917 0.000102 0.000306 ***
2 Wert  t1     t3        47    47       882 0.000536 0.002    **
3 Wert  t2     t3        47    47       506 0.546    1        ns
```

Der Unterschied existiert zwischen `t1` und `t2` sowie zwischen `t1` und `t3`.

Die Effktstärke wird wie folgt berechnet:

```
# Effektstärke
rstatix::friedman_effsize(Wert ~ Zeitpunkt | Name, data = df)
```

```
# A tibble: 1 × 5
  .y.       n effsize method    magnitude
* <chr> <int>   <dbl> <chr>     <ord>
1 Wert     47   0.136 Kendall W small
```

Der Effekt des Gesamtmodells ist eher gering. Schauen wir uns die einzelnen Gruppeneffekte an. Achtung, wir dürfen nur solche Werte interpretieren, die oben signifkant waren. In unserem Beispiel ist das der Unterschied von `t1` zu `t2` sowie von `t1` zu `t3`.

```
# nur die signifikanten anschauen (t1 -> t2 und t1 -> t3)
df %>%
  rstatix::wilcox_effsize(Wert ~ Zeitpunkt, paired = TRUE)
```

```
# A tibble: 3 × 7
  .y.   group1 group2 effsize    n1    n2 magnitude
* <chr> <chr>  <chr>    <dbl> <int> <int> <ord>
1 Wert  t1     t2      0.545     47    47 large
2 Wert  t1     t3      0.491     47    47 moderate
3 Wert  t2     t3      0.0895    47    47 small
```

Der Effekt von `t1` nach `t2` ist “groß”, der von `t1` nach `t3` ist “moderat”.

2.2.9.3 einfaktorielle ANOVA

Eine einfaktorielle Varianzanalyse (ANOVA) wird mit der Funktion `aov()` durchgeführt. Sie wird nach der Syntax *x erklärt durch y* aufgerufen. Wie bei anderen Modellen ist es ratsam, die Ergebnisse der Berechnung in ein Objekt zu speichern, und dieses per `summary()` anzuschauen

```
# Lade Testdatensatz
load(url("https://www.produnis.de/R/data/pf8.RData"))

# erstelle ANOVA  "Alter" erklärt durch "Standort"
fit <- aov(Alter ~ Standort, data=pf8)

# schaue Modellzusammenfassung an
summary(fit)
```

```
             Df Sum Sq Mean Sq F value   Pr(>F)
Standort      4  10557  2639.3    8.48 1.09e-06 ***
Residuals   725 225633   311.2
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
1 observation deleted due to missingness
```

Das Ergebnis ist signifikant, wir haben unterschiedliche Varianzen im Alter erklärt durch Standort.

Wir wissen aber noch nicht, innerhalb welche Standorte die Unterschiede bestehen. Dazu führen wir post hoc Tests durch, in denen alle Gruppen paarweise verglichen werden. Wegen dieses multiplen Testens muss eine Fehlerkorrektur für Alpha durchgeführt werden. Dazu steht die Funktion `pairwise.t.test()` zur Verfügung, die mit den selben Angaben aufgerufen wird.

```
pairwise.t.test(pf8$Alter, pf8$Standort)
```

```
^^IPairwise comparisons using t tests with pooled SD

data:  pf8$Alter and pf8$Standort

          Rheine Münster Bahn   Ladbergen
Münster   0.5042 -       -      -
Bahn      0.4354 1.0000  -      -
Ladbergen 0.5432 1.0000  1.0000 -
Internet  0.0090 1.8e-05 0.0002 0.0059

P value adjustment method: holm
```

Die gewählte Fehlerkorrektur ist standardmäßig `holm`. Möchten Sie auf die Bonferroni-Korrekturmethode wechseln, übergeben Sie den Parameter
`p.adjust="bonferroni"`

```
pairwise.t.test(pf8$Alter, pf8$Standort, p.adjust="bonferroni")
```

```
^^IPairwise comparisons using t tests with pooled SD

data:  pf8$Alter and pf8$Standort
```

```
          Rheine  Münster Bahn    Ladbergen
Münster   1.00000 -       -       -
Bahn      0.72572 1.00000 -       -
Ladbergen 1.00000 1.00000 1.00000 -
Internet  0.01292 1.8e-05 0.00022 0.00733

P value adjustment method: bonferroni
```

Die signifikanten Unterschiede befinden sich zwischen dem Standort `Internet` und allen anderen Standorten.

Ein anschauliches Video zur ANOVA in R findet sich bei Youtube:
https://www.youtube.com/watch?v=GctGncQrJew

2.2.10 Überlebenszeitanalysen

2.2.10.1 Kaplan-Meier-Analysen

Zur Überlebenszeitanalyse nach dem Kaplan-Meier-Verfahren steht das Zusatzpaket `survival` zur Verfügung. Zum Ausprobieren der Funktionen liegt dem Paket der Datensatz `ovarian` bei.

```
# Beschreibung des Datensatzes lesen
?survival::ovarian

# speichere Testdatensatz "ovarian" in Objekt "ov"
ov <- survival::ovarian
head(ov)
```

```
  futime fustat     age resid.ds rx ecog.ps
1     59      1 72.3315        2  1       1
2    115      1 74.4932        2  1       1
3    156      1 66.4658        2  1       2
4    421      0 53.3644        2  2       1
5    431      1 50.3397        2  1       1
6    448      0 56.4301        1  1       2
```

Bevor wir mit der Analyse beginnen, sollten wir den Datensatz noch aufbereiten. Wir überführen alle kategorialen Variablen in einen factor.

```
### Datensatz aufbereiten
## alle kategorialen Variablen
# rx ist die Behandlungsgruppe
ov$rx <- factor(ov$rx,
                levels = c("1", "2"),
                labels = c("A", "B"))

# resid.ds sind Nebenerkrankungen
ov$resid.ds <- factor(ov$resid.ds,
                      levels = c("1", "2"),
                      labels = c("nein", "ja"))

# ecog.ps ist der ECOG-Status
ov$ecog.ps <- factor(ov$ecog.ps,
                     levels = c("1", "2"),
                     labels = c("gut", "schlecht"))
```

Als ersten Analyseschritt erzeugen wir mit der Funktion `Surv()` ein "Survivalobjekt". Der Funktion muss eine Zeitvariable mit der Beobachtungszeit sowie eine dichotome Statusvariable mit dem Ereignis (0 = nicht eingetreten; 1 = eingetreten) übergeben werden. Im `ovarian`-Datensatz ist die Zeit in `futime` und der Status in `fustat` gespeichert. Der Funktionsaufruf lautet dementsprechend:

```
# Survivalobjet erzeugen
survObjekt <- survival::Surv(time=ov$futime, event=ov$fustat)
```

Mit der Funktion `survfit()` wird nun das Kaplan-Meier-Modell gefittet. Der Funktion muss das eben erstellte Survivalobjekt übergeben werden.

```
### Kaplan-Meier
#  ~1 bedeutet "Gesamtmodell", ohne Gruppierung
km_fit <- survival::survfit(survObjekt ~ 1, data=ov)
```

Das Modell kann als Tabelle oder Plot ausgegeben werden.

```
## Tabelle anschauen
#  "survival"-Spalte ist die Überlebenswahrscheinlichkeit
summary(km_fit)
```

```
Call: survfit(formula = survObjekt ~ 1, data = ov)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
   59     26       1    0.962  0.0377        0.890        1.000
  115     25       1    0.923  0.0523        0.826        1.000
  156     24       1    0.885  0.0627        0.770        1.000
  268     23       1    0.846  0.0708        0.718        0.997
  329     22       1    0.808  0.0773        0.670        0.974
  353     21       1    0.769  0.0826        0.623        0.949
  365     20       1    0.731  0.0870        0.579        0.923
  431     17       1    0.688  0.0919        0.529        0.894
  464     15       1    0.642  0.0965        0.478        0.862
  475     14       1    0.596  0.0999        0.429        0.828
  563     12       1    0.546  0.1032        0.377        0.791
  638     11       1    0.497  0.1051        0.328        0.752
```

Die Spalte `survival` gibt die entsprechenden Überlebenswahrscheinlichkeiten an.

```
# erzeuge Testwerte
## plot()  (sehr hässlich)
plot(km_fit)
```

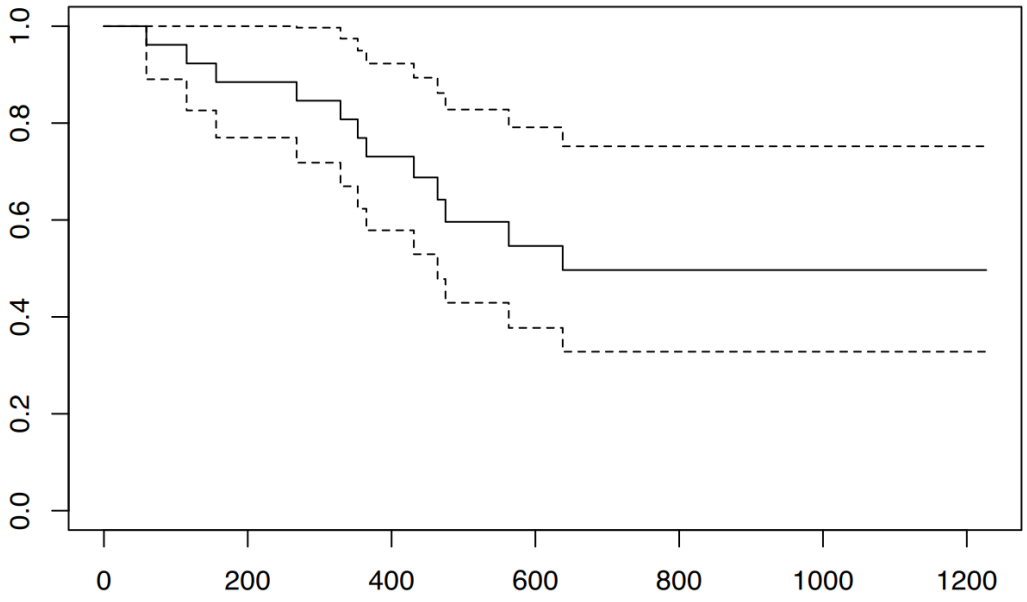


Abb. 2.26

Etwas hübschere Plots lassen sich mit der Funktion `autoplot()` erzeugen. Damit die Funktion die Modelldaten versteht, muss das Zusatzpaket `ggfortify` geladen werden.

```
# etwas hübscher
library(ggfortify)
autoplot(km_fit)
```

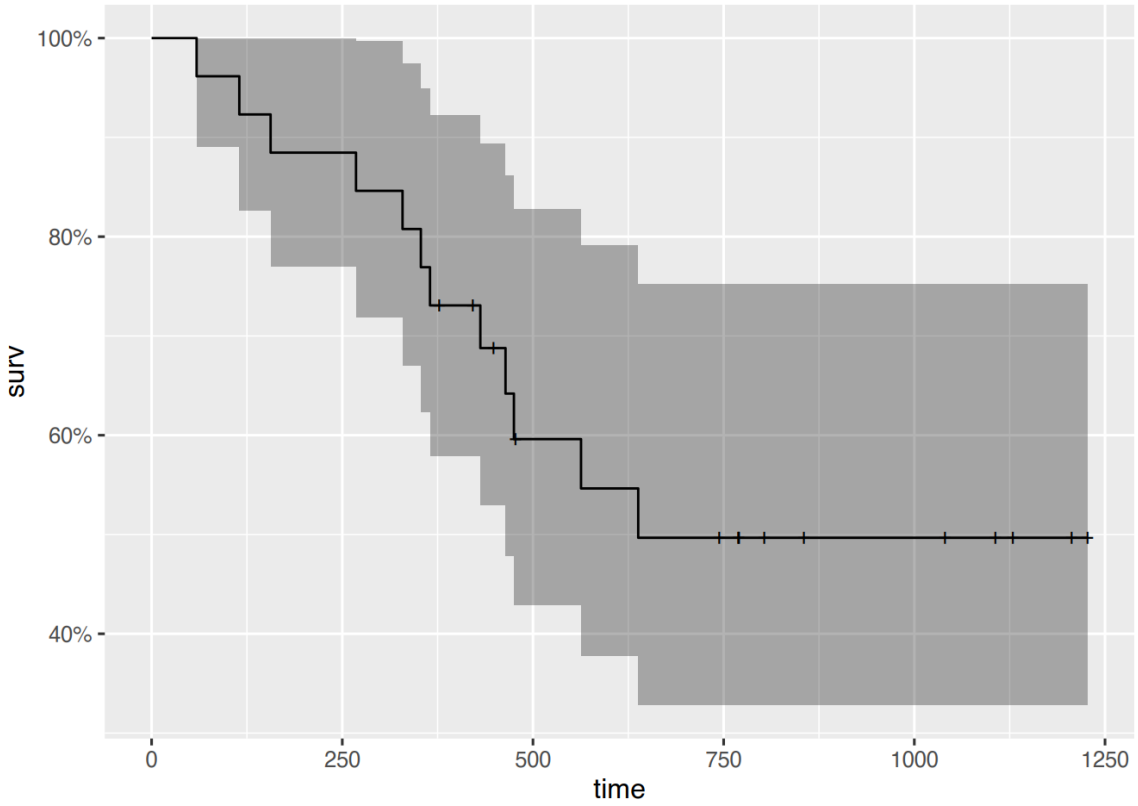


Abb. 2.27

Darüber hinaus bietet das Paket `survminer` die Funktion `ggsurvplot()`:

```
## survminer ggsurvplot
survminer::ggsurvplot(km_fit)
```

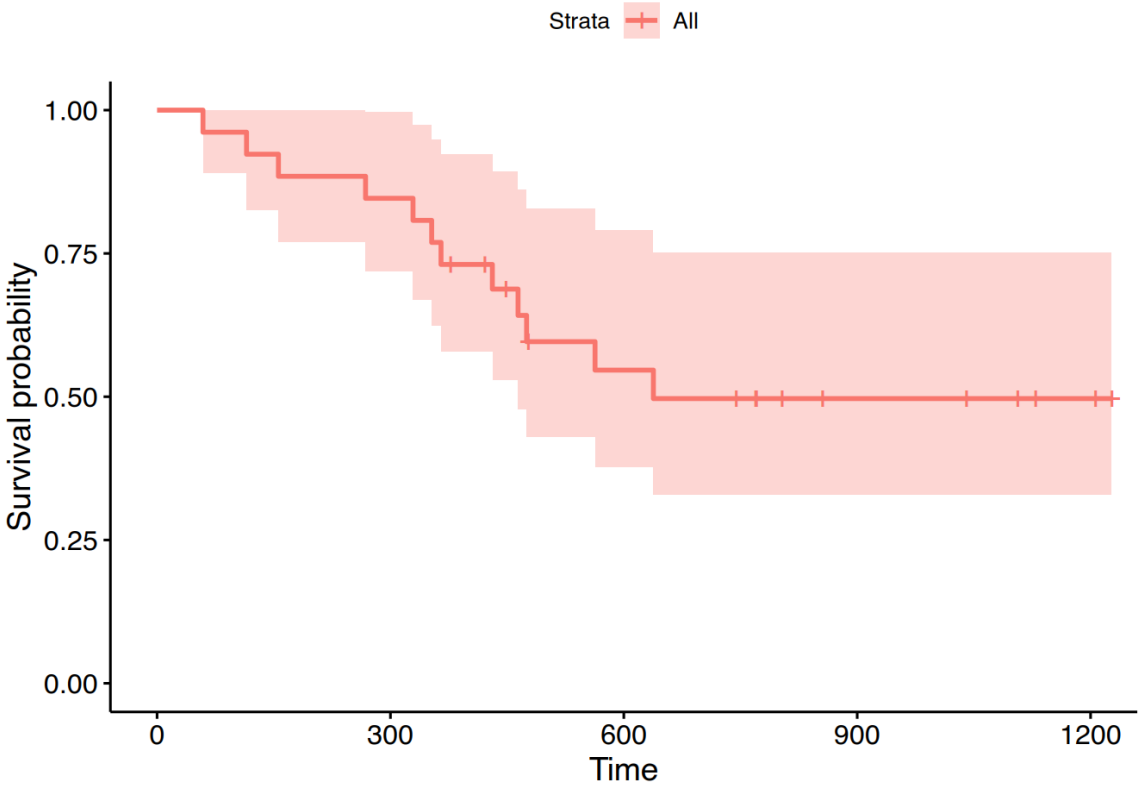


Abb. 2.28

Das Modell lässt sich auch für Gruppenvergleiche erstellen. Die Variable `ov$rx` unterteilt den Datensatz in zwei Behandlungsgruppen. Das Überlebensmodell pro Gruppe lässt sich wie folgt erzeugen:

```
### Gruppierungen
# nach Behandlungsgruppe "rx" gruppieren
km_fit <- survival::survfit(survObjekt ~ rx, data=ov)
```

Mittels `summary()` werden die Überlebenstabellen pro Gruppe ausgegeben.

```
### Gruppierungen
# nach Behandlungsgruppe "rx" gruppieren
km_fit <- survival::survfit(survObjekt ~ rx, data=ov)
```

```
Call: survfit(formula = survObjekt ~ rx, data = ov)

                rx=A
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
   59     13       1    0.923  0.0739        0.789        1.000
  115     12       1    0.846  0.1001        0.671        1.000
  156     11       1    0.769  0.1169        0.571        1.000
  268     10       1    0.692  0.1280        0.482        0.995
  329      9       1    0.615  0.1349        0.400        0.946
  431      8       1    0.538  0.1383        0.326        0.891
  638      5       1    0.431  0.1467        0.221        0.840

                rx=B
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
```

```
    353      13       1   0.923  0.0739       0.789       1.000
    365      12       1   0.846  0.1001       0.671       1.000
    464       9       1   0.752  0.1256       0.542       1.000
    475       8       1   0.658  0.1407       0.433       1.000
    563       7       1   0.564  0.1488       0.336       0.946
```

Im Plot werden die Kaplan-Meier-Kurven beider Gruppen angezeigt.

```
1 # plot (hässlich)
2 plot(km_fit, conf.int=T) # mit Konfidenzintervallen
```

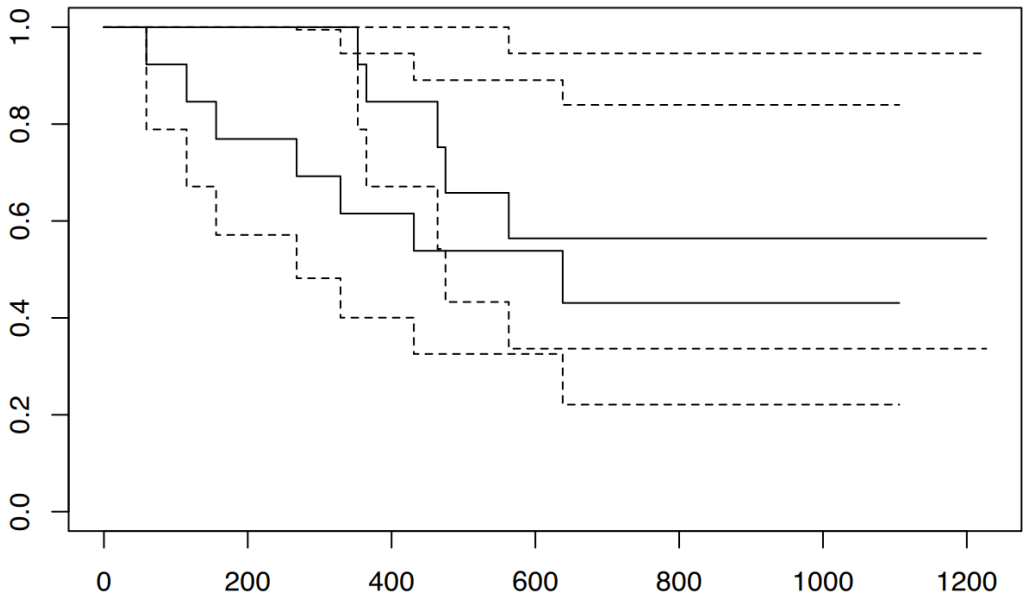


Abb. 2.29

Das Diagramm kann noch etwas aufgehübscht werden.

```
# plot (hübscher)
plot(km_fit, conf.int=TRUE, col=c("red", "blue"), yscale=100,
     xlab = "Überlebenszeit in Tagen", ylab = "Prozent Überleben")
legend("bottomright", title="Gruppe", c("1", "2"),
       fill=c("red", "blue"))
```

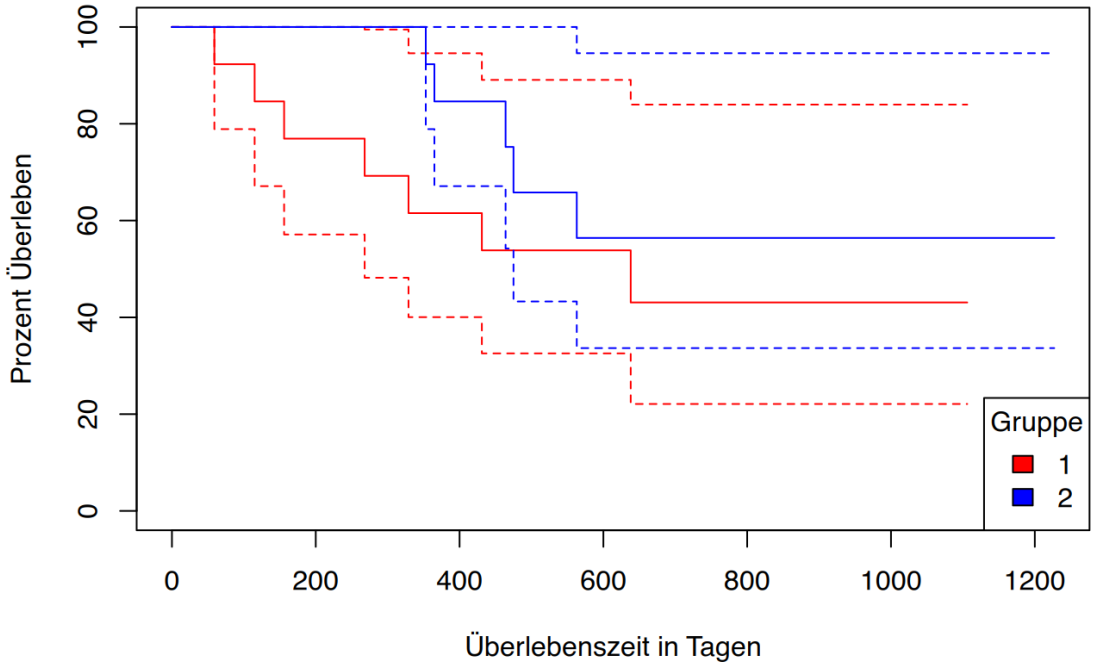


Abb. 2.30

Die Funktion `ggsurvplot()` liefert auf Wunsch den p-Wert des Gruppenunterschieds mit:

```
## survminer ggsurvplot mit p-Wert
survminer::ggsurvplot(km_fit, pval=TRUE, conf.int = TRUE)
```

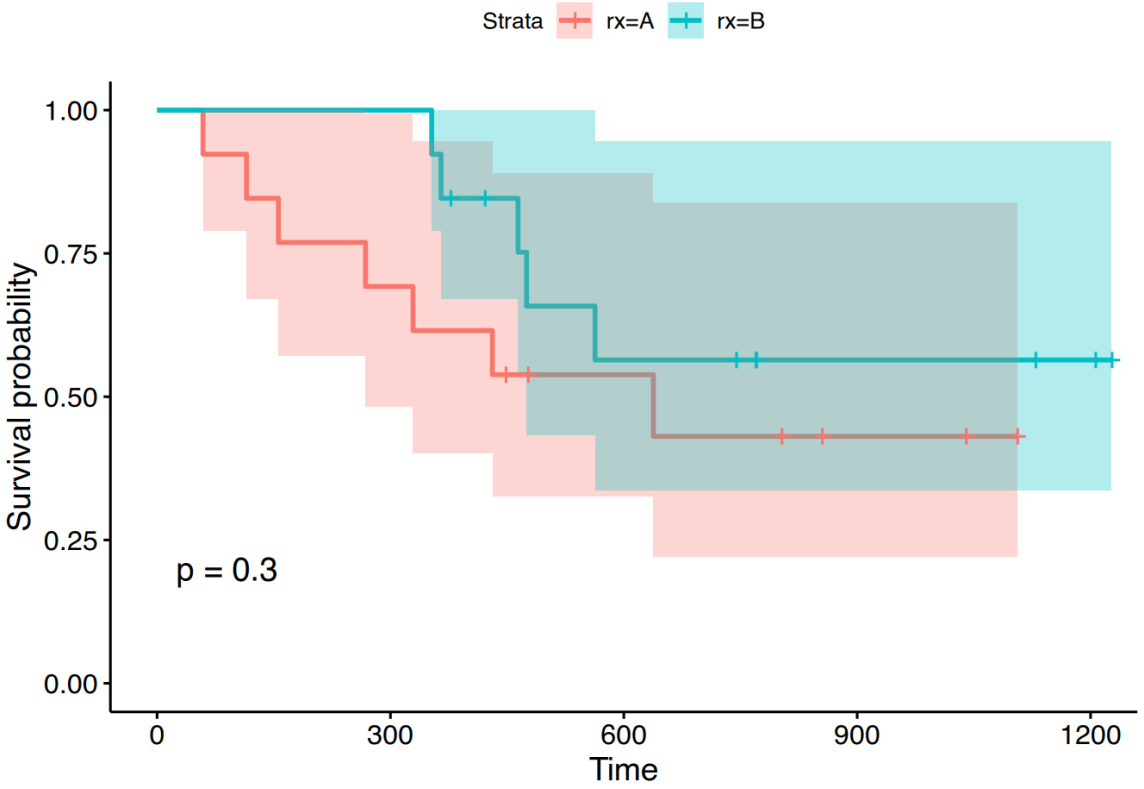


Abb. 2.31

Der p-Wert lässt sich mit dem Lograng-Test auch selbst berechnen. Hierfür steht die Funktion `survdiff()` zur Verfügung.

```
# p-Wert selbst berechnen
## Lograng-Test
survival::survdiff(survObjekt ~rx, data=ov)
```

```
Call:
survival::survdiff(formula = survObjekt ~ rx, data = ov)

      N Observed Expected (O-E)^2/E (O-E)^2/V
rx=A 13        7     5.23     0.596      1.06
rx=B 13        5     6.77     0.461      1.06

 Chisq= 1.1  on 1 degrees of freedom, p= 0.3
```

2.2.10.2 Cox-Regression

Neben der Kaplan-Meier-Analyse können die Daten auch per Cox-Regression analysiert werden. Der Vorteil besteht darin, dass Prädiktoren verschiedensten Datenniveaus in die Analyse integriert werden können. Das Paket `survival` bietet hierzu die Funktion `coxph()`.

```
cox <- survival::coxph(survObjekt ~ rx+resid.ds+age+ecog.ps, data=ov)
# Modell ausgeben
cox
```

```
Call:
survival::coxph(formula = survObjekt ~ rx + resid.ds + age +
    ecog.ps, data = ov)

             coef exp(coef) se(coef)      z       p
rx       -0.91450   0.40072  0.65332 -1.400 0.16158
resid.ds  0.82619   2.28459  0.78961  1.046 0.29541
age       0.12481   1.13294  0.04689  2.662 0.00777
ecog.ps   0.33621   1.39964  0.64392  0.522 0.60158

Likelihood ratio test=17.04  on 4 df, p=0.001896
n= 26, number of events= 12
```

Die Spalte `exp(coef)` entspricht der Hazard-Ratio, mit welcher Richtung und Stärke des jeweiligen Einflusses interpretiert werden kann.

Das Modell kann mittels `ggforest()` geplottet werden:

```
# Forestplot der Hazard-Ratio
survminer::ggforest(cox, data=ov)
```

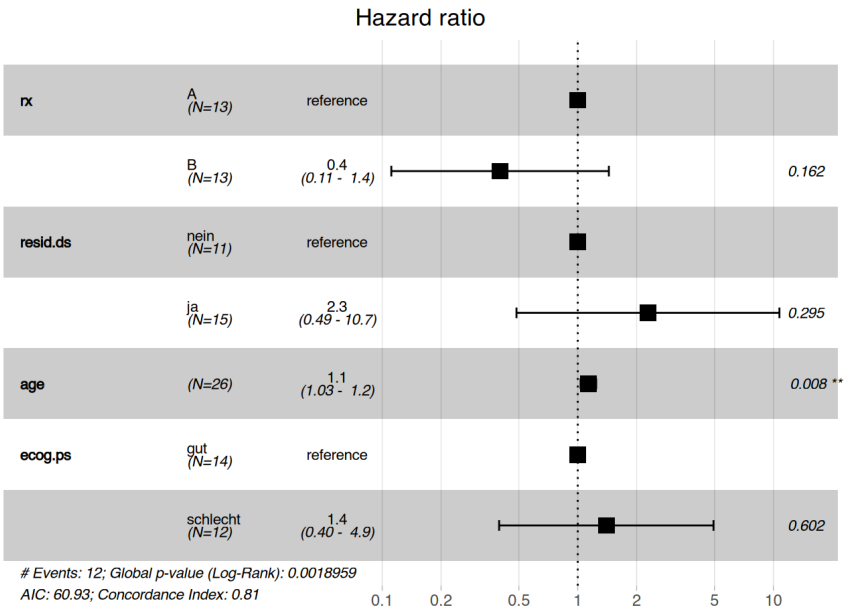


Abb. 2.32

2.2.11 Faktorenanalyse

Die beiden am Häufigsten genutzen Verfahren sind die Haupt*komponenten*analyse (PCA) und die Haupt*achsen*analyse (PAF). Sie unterscheiden sich in den Varianzanteilen, die in den Items enthalten sein können. Die PCA versucht, die *gesamte* Varianz auf Faktoren zurückzuführen, die PAF die *gemeinsame* Varianz der Items. In psychologischen Anwendungen (= am Menschen) wird daher eher die PAF verwendet.

2.2.11.1 explorative Faktorenanalyse

Bei der explorativen Faktorenanalyse (EFA) erfolgt die Auswertung (häufig) nicht theoriegeleitet. Ziel ist es vielmehr zu erfahren, mit wievielen Faktoren die Daten bestmöglich erklärt werden können.
Wir verwenden den Datensatz `Faktorenbogen`, der wie folgt geladen werden kann.

```
# Datensatz laden
load(url("https://www.produnis.de/R/data/Faktorenbogen.rda"))

# anschauen
psych::describe(Faktorenbogen)
```

```
         vars   n  mean    sd median trimmed   mad min max range  skew kurtosis   se
age         1 150 53.11 18.82   53.0   54.16 17.05   1 101   100 -0.47     0.29 1.54
gender*     2 150  1.65  0.55    2.0    1.64  0.00   1   3     2  0.02    -0.87 0.04
A           3 150  4.41  2.03    5.0    4.40  2.97   0  10    10  0.05    -0.43 0.17
B           4 150  1.92  0.84    2.0    1.84  1.48   1   5     4  0.76     0.43 0.07
C           5 150  7.91  6.19    7.0    7.22  5.93   0  30    30  1.12     1.31 0.51
D           6 150 12.23  5.05   12.0   11.95  4.45   3  27    24  0.51    -0.04 0.41
E           7 150 14.77  2.39   15.0   14.69  2.97  10  21    11  0.28    -0.13 0.20
F           8 150  5.00  1.32    5.0    5.03  1.48   2   8     6 -0.12    -0.67 0.11
G           9 150 10.35  4.08    9.5   10.06  3.71   3  24    21  0.78     0.76 0.33
H          10 150 15.92  2.93   16.0   15.84  2.97  10  24    14  0.21    -0.39 0.24
I          11 150 85.02 68.19   65.0   75.93 51.15   0 356   356  1.40     2.05 5.57
J          12 150  5.02  2.25    5.0    4.93  1.48   1  11    10  0.33    -0.12 0.18
K          13 150  5.44  2.08    6.0    5.38  2.97   1  12    11  0.31    -0.10 0.17
L          14 150  1.47  0.65    1.0    1.36  0.00   1   4     3  1.20     0.86 0.05
```

Die Items für die Faktorenanalyse befinden sich in den Variablen `A` bis `L`.

```
items <- Faktorenbogen %>%
  select(A:L)
```

Mit dem Bartlett-Test kann überprüft werden, ob die Items miteinander korrelieren (das sollten sie tun).

```
psych::cortest.bartlett(items)$p.value
```

```
R was not square, finding R from data
[1] 8.843226e-151
```

Das Ergebnis ist signifikant.

Mittels *Kaiser-Meyer-Olkin*-Verfahren (KMO) wird geprüft, wie gut die Daten für eine Faktorenanalyse geeignet sind, wobei Werte zwischen 0 (für "nicht geeignet") und 1 (für "perfekt geeignet") angenommen werden. Die *Minimum Average Partial* (MSA) ist ein Maß der internen Konsistenz, und nimmt ebenfalls Werte zwischen 0 (nicht konsistent) und 1 (perfekt konsistent) an. Beide Werte sollte größer als 0,5 sein.

```
# berechne KMO und MSA
psych::KMO(items)
```

```
Kaiser-Meyer-Olkin factor adequacy
Call: psych::KMO(r = items)
Overall MSA =  0.69
MSA for each item =
   A    B    C    D    E    F    G    H    I    J    K    L
0.36 0.72 0.66 0.68 0.60 0.73 0.24 0.73 0.75 0.62 0.78 0.61
```

Die "Overall MSA" entspricht dem KMO.

Mittels Screeplot kann die Faktorenzahl graphisch entschieden werden. Dabei sollte der letzte "gültige" Faktor vor dem deutlichen Knick zu einer abflachenden Gerade genutzt werden. Darüber hinaus sollten die Eigenwerte der Faktoren größer 1 sein.

Mit -Hausmitteln kann ein Screeplot wie folgt erzeugt werden.

```
plot(eigen(cor(items))$values, type="b")
abline(h=1)
```

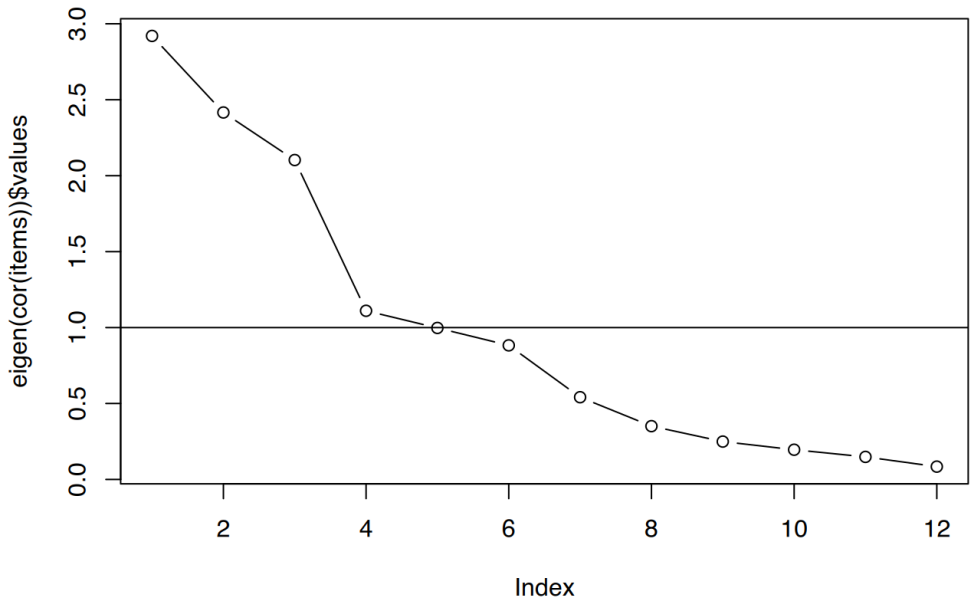


Abb. 2.33

Schöner geht es mit der Funktion `fa.parallel()`.

```
psych::fa.parallel(items)
```

```
Parallel analysis suggests that the number of factors =  3
                         and the number of components =  3
```

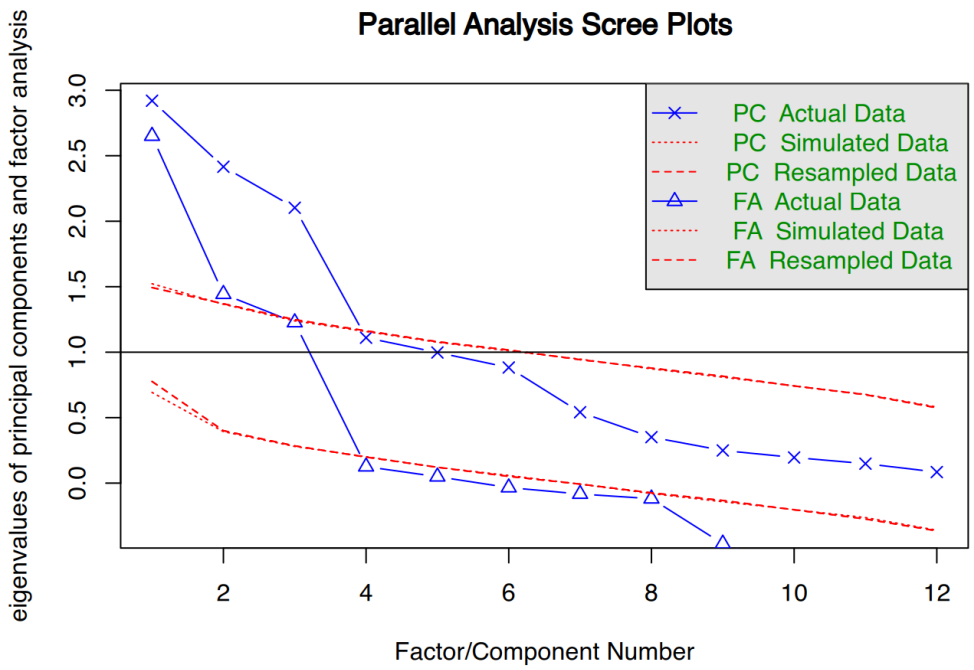


Abb. 2.34

Es werden 3 Faktoren/Komponenten vorgeschlagen.
Die Funktion berechnet das Screeplot sowohl für PAC (Sternchen) als auch PAF (Dreieck). Mit dem Parameter `fa = "fa"` kann die PAC-Methode entfernt werden.

```
psych::fa.parallel(items, fa = "fa")
```

```
Parallel analysis suggests that the number of factors =  3
```

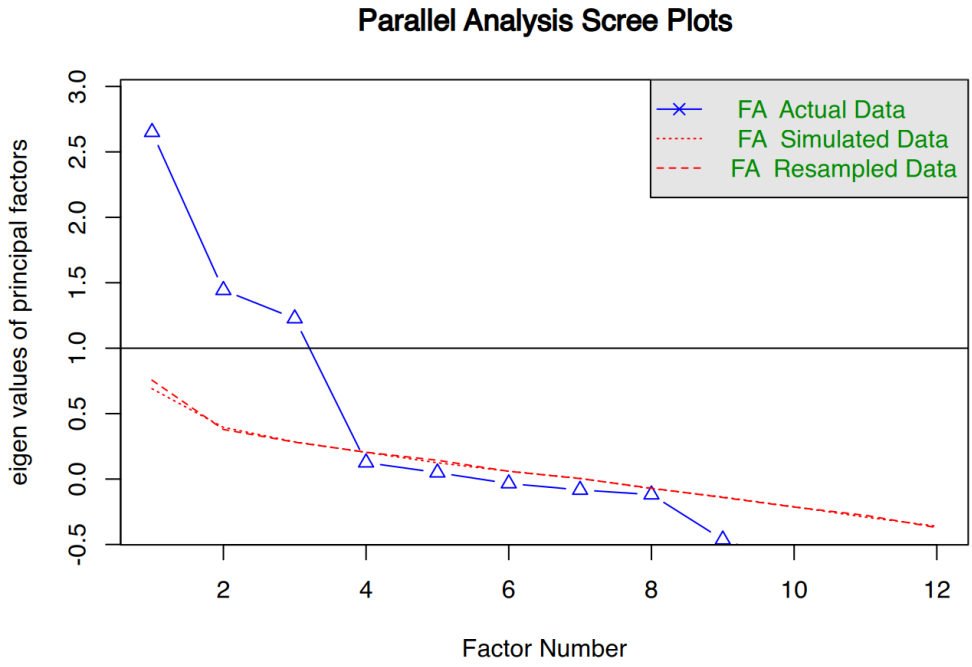


Abb. 2.35

Die Daten lassen sich am besten mit 3 Faktoren erklären.

2.2.11.2 konfirmatorische Faktorenanalyse

Ist die Anzahl der Faktoren bekannt (z.B. theoriegeleitet), wird die Hauptachsenanaylse wie folgt durchgeführt.

```
# Hauptachsen-Faktorenanalyse (PAF)
# mit 3 Faktoren, Varimax-Rotation
fit <- psych::fa(items, nfactors=3,
                 rotate = "varimax")
fit
```

```
Factor Analysis using method =  minres
Call: psych::fa(r = items, nfactors = 3, rotate = "varimax")
Standardized loadings (pattern matrix) based upon correlation matrix
    MR1   MR2   MR3     h2    u2 com
A  0.00  0.08  0.03 0.0069 0.993 1.2
B -0.01  0.83 -0.08 0.6997 0.300 1.0
C -0.05  0.88 -0.01 0.7770 0.223 1.0
D  0.96 -0.09 -0.07 0.9266 0.073 1.0
E -0.02  0.10  0.88 0.7820 0.218 1.0
F -0.04 -0.12  0.59 0.3655 0.635 1.1
G  0.00  0.00  0.07 0.0048 0.995 1.0
H  0.94 -0.02 -0.08 0.8855 0.115 1.0
I -0.03  0.78 -0.05 0.6107 0.389 1.0
J -0.08  0.08  0.88 0.7866 0.213 1.0
K  0.89  0.03 -0.01 0.7896 0.210 1.0
L -0.10  0.09  0.14 0.0393 0.961 2.6

                       MR1  MR2  MR3
SS loadings           2.61 2.13 1.94
Proportion Var        0.22 0.18 0.16
```

```
Cumulative Var        0.22 0.39 0.56
Proportion Explained  0.39 0.32 0.29
Cumulative Proportion 0.39 0.71 1.00

Mean item complexity =  1.2
Test of the hypothesis that 3 factors are sufficient.

The degrees of freedom for the null model are  66  and the objective function was
                                                6.38 with Chi Square of  920.48
The degrees of freedom for the model are 33  and the objective function was  0.32

The root mean square of the residuals (RMSR) is  0.03
The df corrected root mean square of the residuals is  0.05

The harmonic number of observations is  150 with the empirical chi square  20.2
                                                              with prob <  0.96
The total number of observations was  150  with Likelihood Chi Square =  45.01
                                                            with prob <  0.079

Tucker Lewis Index of factoring reliability =  0.971
RMSEA index =  0.049  and the 90 % confidence intervals are  0 0.083
BIC =  -120.34
Fit based upon off diagonal values = 0.99
Measures of factor score adequacy
                                                 MR1  MR2  MR3
Correlation of (regression) scores with factors   0.98 0.94 0.94
Multiple R square of scores with factors          0.96 0.88 0.88
Minimum correlation of possible factor scores     0.92 0.76 0.77
```

Die Ausgabe von `fit` ist ziemlich lang. Wie gewohnt besteht die Möglichkeit, mittels `$` auf die einzelnen Ergebnisse zuzugreifen. Eine Übersicht erhält man mittels `names()`.

```
1 names(fit)
```

```
 [1] "residual"      "dof"           "chi"           "nh"            "rms"
 [6] "EPVAL"         "crms"          "EBIC"          "ESABIC"        "fit"
[11] "fit.off"       "sd"            "factors"       "complexity"    "n.obs"
[16] "objective"     "criteria"      "STATISTIC"     "PVAL"          "Call"
[21] "null.model"    "null.dof"      "null.chisq"    "TLI"           "RMSEA"
[26] "BIC"           "SABIC"         "r.scores"      "R2"            "valid"
[31] "score.cor"     "weights"       "rotation"      "hyperplane"    "communality"
[36] "communalities" "uniquenesses"  "values"        "e.values"      "loadings"
[41] "model"         "fm"            "rot.mat"       "Structure"     "method"
[46] "scores"        "R2.scores"     "r"             "np.obs"        "fn"
[51] "Vaccounted"
```

```
1 fit$communality
```

```
          A           B           C           D           E           F
0.006940720 0.699704328 0.776989353 0.926609287 0.782014553 0.365479522
          G           H           I           J           K           L
0.004811051 0.885466837 0.610670890 0.786557566 0.789638724 0.039273937
```

```
1 fit$loadings
```

```
Loadings:
  MR1    MR2    MR3
A
B         0.833
C         0.880
D  0.956
E         0.103  0.878
F        -0.118  0.592
G
H  0.938
I         0.779
J                0.879
K  0.888
L -0.104         0.141

                 MR1   MR2   MR3
SS loadings    2.605 2.129 1.940
Proportion Var 0.217 0.177 0.162
Cumulative Var 0.217 0.394 0.556
```

Mit diesen Informationen können die Items ein bisschen umsortiert...

- Faktor 1: `D`, `H`, `K`
- Faktor 2: `B`, `C`, `I`
- Faktor 3: `E`, `F`, `J`

... und ein Korrelationsmatrix erstellt werden.

```
# ändere Reihenfolgt
items2 <- items %>%
  select(D, H, K, B, C, I, E, F, J)

# Korreltationsmatrix plotten
corrplot::corrplot(corr = cor(items2), # Korrelationsmatrix
         method = "color",   # Ausprägung farblich kodieren
         addCoef.col = "black", # schreibe Werte in schwarz
         number.cex = 0.7)   # Schriftgröße
```


D H K B C I E F J
D 1 0.9 0.85 -0.06 -0.14 -0.1 -0.08 -0.09 -0.15
H 0.9 1 0.83 -0.02 -0.05 -0.07 -0.08 -0.11 -0.14
K 0.85 0.83 1 -0.01 -0.01 0.02 -0.06 0.01 -0.1
B -0.06 -0.02 -0.01 1 0.73 0.65 0.03 -0.19 0.02
C -0.14 -0.05 -0.01 0.73 1 0.69 0.07 -0.08 0.05
I -0.1 -0.07 0.02 0.65 0.69 1 0.04 -0.12 0.01
E -0.08 -0.08 -0.06 0.03 0.07 0.04 1 0.5 0.8
F -0.09 -0.11 0.01 -0.19 -0.08 -0.12 0.5 1 0.49
J -0.15 -0.14 -0.1 0.02 0.05 0.01 0.8 0.49 1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

Abb. 2.36

2.2.11.3 Hauptkomponentenanalyse

Die Hauptkomponentenanalyse kann mit der Funktion `principal()` durchgeführt werden.

```
# Hauptkomponentenanalyse (PCA)
# mit 3 Hauptkomponenten und
# Varimax-Rotation
fit2 <- psych::principal(items, nfactors=3, rotate="varimax")

# anschauen
fit2
```

```
Principal Components Analysis
Call: psych::principal(r = items, nfactors = 3, rotate = "varimax")
Standardized loadings (pattern matrix) based upon correlation matrix
    RC1   RC2   RC3    h2    u2 com
A  0.01  0.13  0.05 0.019 0.981 1.3
B -0.01  0.89 -0.08 0.793 0.207 1.0
C -0.05  0.90  0.00 0.821 0.179 1.0
D  0.96 -0.08 -0.07 0.928 0.072 1.0
E -0.03  0.10  0.89 0.803 0.197 1.0
F -0.03 -0.15  0.76 0.606 0.394 1.1
G  0.01  0.00  0.12 0.015 0.985 1.0
H  0.95 -0.02 -0.07 0.916 0.084 1.0
I -0.04  0.86 -0.05 0.752 0.248 1.0
J -0.09  0.08  0.89 0.806 0.194 1.0
K  0.94  0.04  0.00 0.884 0.116 1.0
L -0.14  0.14  0.24 0.095 0.905 2.3

                       RC1  RC2  RC3
SS loadings           2.74 2.44 2.26
Proportion Var        0.23 0.20 0.19
Cumulative Var        0.23 0.43 0.62
Proportion Explained  0.37 0.33 0.30
Cumulative Proportion 0.37 0.70 1.00

Mean item complexity =  1.2
Test of the hypothesis that 3 components are sufficient.

The root mean square of the residuals (RMSR) is  0.06
 with the empirical chi square  61.27  with prob <  0.002

Fit based upon off diagonal values = 0.96
```

```
# welche $-Werte sind verfügbar?
names(fit2)
```

```
 [1] "values"       "rotation"     "n.obs"        "communality"  "loadings"
 [6] "fit"          "fit.off"      "fn"           "Call"         "uniquenesses"
[11] "complexity"   "chi"          "EPVAL"        "R2"           "objective"
[16] "residual"     "rms"          "factors"      "dof"          "null.dof"
[21] "null.model"   "criteria"     "STATISTIC"    "PVAL"         "weights"
[26] "r.scores"     "rot.mat"      "Vaccounted"   "Structure"    "scores"
```

```
# Faktorenladungen
fit2$loadings
```

```
Loadings:
  RC1    RC2    RC3
A          0.128
B          0.887
C          0.905
D  0.958
E          0.102  0.890
F         -0.146  0.764
G                 0.121
H  0.954
I          0.865
J                 0.890
K  0.940
L -0.135  0.143  0.238

                 RC1   RC2   RC3
SS loadings    2.744 2.435 2.260
Proportion Var 0.229 0.203 0.188
Cumulative Var 0.229 0.432 0.620
```

Ein passendes Video von Daniela Keller findet sich bei Youtube unter https://www.youtube.com/watch?v=NFPGQcq1fO8.

2.2.12 Fallzahlkalkulation

2.2.12.1 klinische Studien

Zur Fallzahlkalkulation bei klinischen Studien (z.B. RCTs) kann das Zusatzpaket `pwr` installiert werden. Mit der Funktion `pwr.t.test()` können alle Werte berechnet werden. Ihr müssen die entsprechenden Werte für das Signifikanzlevel, Power und Cohen's `d` (Effektstärke) übergeben werden.

```
# erechne Fallzahl mit
# Cohens d = 0.1
# alpha = 0.05
# Power = 0.8
pwr::pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="two.sample", alternative="two.sided")
```

```
     Two-sample t test power calculation

              n = 1570.733
              d = 0.1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

Über die Parameter `type` und `alternative` lassen sich die Testbedingungen festlegen.

```
# Lade Paket
library(pwr)

# unabhängige Stichprobe, zweiseitig
pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="two.sample", alternative="two.sided")

# Einstichproben-Test, kleiner
pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="one.sample", alternative="less")

# gepaarte Stichprobe, größer
pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="paired", alternative="greater")
```

Möchten Sie nicht immer die gesamte Testausgabe angezeigt bekommen, sondern nur den Wert für `n`, können Sie dies mit einem Dollarzeichen `$` hinter dem Funktionsaufruf spezifizieren.

```
# zeige nur "n"
pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="two.sample")$n
```

```
[1] 1570.733
```

```
# runde auf nächste ganze Zahl
ceiling(pwr.t.test(d=0.1, sig.level=0.05, power=0.8, type="two.sample")$n)
```

```
[1] 1571
```

Sie benötigen 1571 Probanden pro Untersuchungsgruppe.

2.2.12.2 Umfragen

Soll eine Befragung (ein *Survey*) durchgeführt werden, kann die benötigte Fallzahl über das Zusatzpaket `samplingbook` errechnet werden.
Geht es um *numerische* Werte, wird die Funktion `sample.size.mean()` verwendet. Ihr übergibt man Fehlerspanne `e` (ein Präzisionswert, der besagt, wie breit das Konfidenzintervall - nach oben sowie nach unten - sein darf, für gewöhnlich 5% bzw. `0.05`), das Konfidenzniveau, die Standardabweichung `S` innerhalb der Grundgesamtheit (füre gewöhnlich 50% bzw. `0.5`) sowie die Populationsgröße. Standardmäßig ist die Populationsgröße auf `N=Inf` (für unendlich) gesetzt.

```
# Berechne Fallzahl für einen Survey
# Konfidenzniveau auf 95%, Fehlerbereich 5%.
# In der Grundgesamtheit hat der Wert eine
# Standardabweichung von S=0.5
samplingbook::sample.size.mean(e=0.05, S=0.5, level=0.95 )
```

```
sample.size.mean object: Sample size for mean estimate
Without finite population correction: N=Inf, precision e=0.05 and standard deviation S=0.5

Sample size needed: 385
```

Ist die Grundgesamtheit bekannt, kann dies mit `N` angegeben werden.

```
# Die Grundgesamtheit ist 1.500 Personen groß
samplingbook::sample.size.mean(e=0.05, S=0.5, level=0.95, N=1500 )
```

```
sample.size.mean object: Sample size for mean estimate
With finite population correction: N=1500, precision e=0.05 and standard deviation S=0.5

Sample size needed: 306
```

Sollen Prozentwerte **kategorialer** Variablen erfragt werden (z.B. der Anteil weiblicher Pflegepersonen), wird die Funktion `sample.size.prop()` verwendet. Ihr werden ebenso `e`, `N` und `level` (Konfidenzniveau) übergeben. Statt der Standardabweichung wird über den Parameter `P` die prozentuale Verteilung in der Grundgesamtheit angegeben.

```
# Berechne Fallzahl für einen Survey
# unsere Werte düren maximal 5% abweichen
# Konfidenzniveau auf 95%
# In der Grundgesamtheit hat der Wert eine
# Verteilung von 65%
# Die Grundgesamtheit ist 1.500 Personen groß
samplingbook::sample.size.prop(e=0.05, P=0.65, level=0.95, N=1500)
```

```
sample.size.prop object: Sample size for proportion estimate
With finite population correction: N=1500, precision e=0.05 and expected proportion P=0.65

Sample size needed: 284
```

Tabellen

Um ein Gefühl für die benötigten Stichprobengrößen zu erhalten, können wir Tabellen erzeugen, in denen die benötigten Fallzahlen beispielsweise in Abhängigkeit zur Populationsgröße angezeigt werden. Die Tabelle soll Werte für metrische und kategoriale Items anzeigen, wobei wir jeweils das 90%-, 95% und 99%-Konfidenzintervall einsehen wollen. In diesem Beispiel gehen wir für metrische Daten von 30% Streuung aus (`S=0.3`) und setzen die Fehlerspanne `e` auf 5%. Für kategoriale Items nehmen wir eine Verteilung von 50% an (`P=0.5`).

```
# gewünschte Populationsgrößen
Population <- c(100, 200, 300, 400, 500, 600, 700, 800, 900, 1000,
                1500, 2000, 4000, 6000, 8000, 10000, Inf)

# überführe in Datenframe
df <- data.frame(Population)
```

Mit der Funktion `sapply()` können wir `sample.size.mean()` und `sample.size.prop()` auf jeden Wert des Vectors `Population` anwenden. Die Ergebnisse schreiben wir als neue Variable in unser Datenframes `df`.

```
## Wende sample.size.mean() auf jeden Wert in "Population" an
# 90%-Konfidenzintervall
df$ConA10 <- sapply(Population, FUN=function(x) samplingbook::sample.size.mean(e=0.05, S=0.3, level=0.90,
N=x)$n)
# 95%-Konfidenzintervall
df$ConA05 <- sapply(Population, FUN=function(x) samplingbook::sample.size.mean(e=0.05, S=0.3, level=0.95,
N=x)$n)
# 99%-Konfidenzintervall
df$ConA01 <- sapply(Population, FUN=function(x) samplingbook::sample.size.mean(e=0.05, S=0.3, level=0.99,
N=x)$n)

## Wende sample.size.prop() auf jeden Wert in "Population" an
# 90%-Konfidenzintervall
df$CatA10 <- sapply(Population, FUN=function(x) samplingbook::sample.size.prop(e=0.05, P=0.5, level=0.9,
N=x)$n)
# 95%-Konfidenzintervall
df$CatA05 <- sapply(Population, FUN=function(x) samplingbook::sample.size.prop(e=0.05, P=0.5, level=0.95,
N=x)$n)
# 99%-Konfidenzintervall
df$CatA01 <- sapply(Population, FUN=function(x) samplingbook::sample.size.prop(e=0.05, P=0.5, level=0.99,
N=x)$n)

```

Die Variablennamen `ConA10` - `CatA01` sind etwas kryptisch und für eine *schöne* Tabelle eher ungeeignet. Daher ändern wir die Variablennamen mittels `colnames()` in leserliche Spaltennamen um. Zuvor speichern wir aber unser Datenframe in ein neues Objekt `df2`, denn mit den veränderten Variablennamen können wir später viel schlechter weiterarbeiten.

```
# Mache Sicherungskopie von df
df2 <- df

# ändere die Variablennamen für eine "schöne" Tabelle
Kopfzeile <- c("Population N", "metrisch 90%", "metrisch 95%", "metrisch 99%",
               "kategor 90%", "kategor 95%", "kategor 99%")
colnames(df) <- Kopfzeile
```

Mit den neuen Variablennamen sieht die Tabelle nun so aus:

```
   Population N metrisch 90% metrisch 95% metrisch 99% kategor 90% kategor 95% kategor 99%
1           100           50           59           71          74          80          87
2           200           66           82          109         115         132         154
3           300           74           95          133         143         169         207
4           400           79          103          150         162         196         250
5           500           82          109          162         176         218         286
6           600           84          113          171         187         235         316
7           700           86          116          179         196         249         341
8           800           87          118          184         203         260         363
9           900           88          120          189         209         270         382
10         1000           89          122          193         213         278         399
11         1500           92          127          207         230         306         461
12         2000           93          130          214         239         323         499
13         4000           96          134          226         254         351         570
14         6000           96          136          230         259         362         598
15         8000           97          136          232         262         367         613
16        10000           97          137          234         264         370         623
17          Inf           98          139          239         271         385         664
```

Und um auf das nächste Kapitel vorzugreifen, können wir die Daten der Tabelle in einem einfachen Plot graphisch darstellen, siehe Abbildung 2.37.

```
plot(Population, df2$ConA01, type="b", ylim=c(0,670),
        ylab="Fallzahl", col="darkgreen", pch=19, lty=2)

# Füge weitere y-Variablen mit anderer Farbe hinzu
lines(Population, df2$ConA05, col="seagreen", type="b", pch=19, lty=1)
lines(Population, df2$ConA10, col="lightgreen", type="b", pch=19, lty=3)
lines(Population, df2$CatA01, col="darkblue", type="b", pch=18, lty=2)
lines(Population, df2$CatA05, col="blue", type="b", pch=18, lty=1)
lines(Population, df2$CatA10, col="lightblue", type="b", pch=18, lty=3)

# Füge eine Legendenbox hinzu
legend(x=0, y=670,
       legend = c("metrisch 90%", "metrisch 95%", "metrisch 99%",
                  "kategori 90%", "kategori 95%", "kategori 99%"),
       col = c("lightgreen", "seagreen", "darkgreen",
                  "lightblue", "blue", "darkblue"),
       lty=c(3,1,2), cex=0.8)
```

Sieht kompliziert aus? Warten Sie es ab, es wird alles im folgenden Kapitel erklärt.

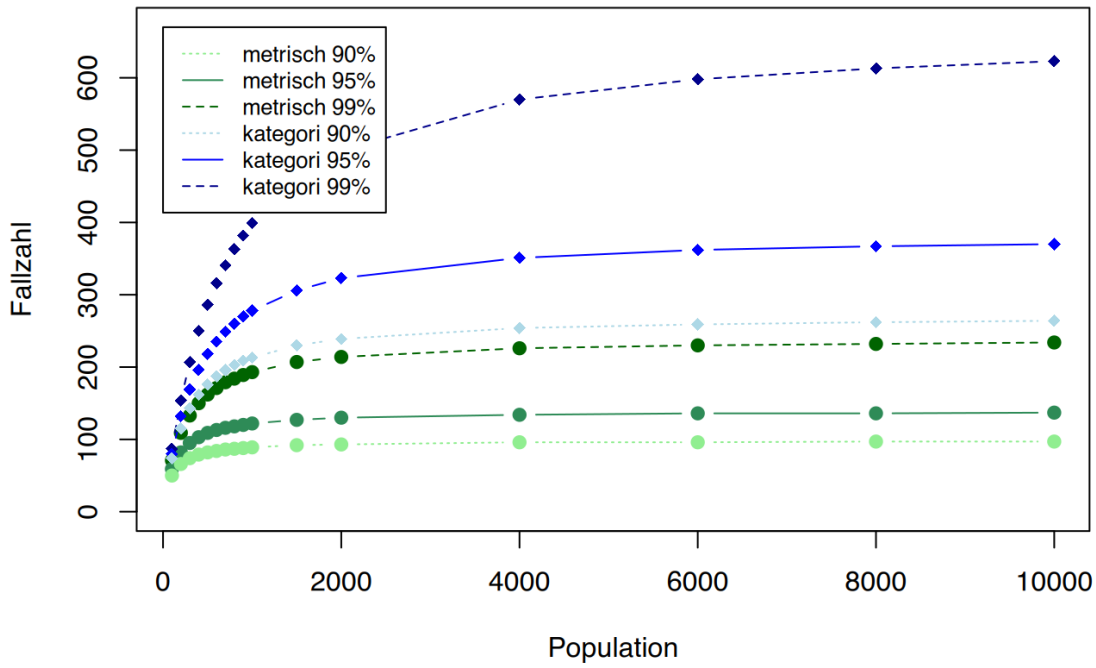


Abb. 2.37: Fallzahl nach Populationsgröße

2.3 Diagramme plotten

labelKapitel:plot

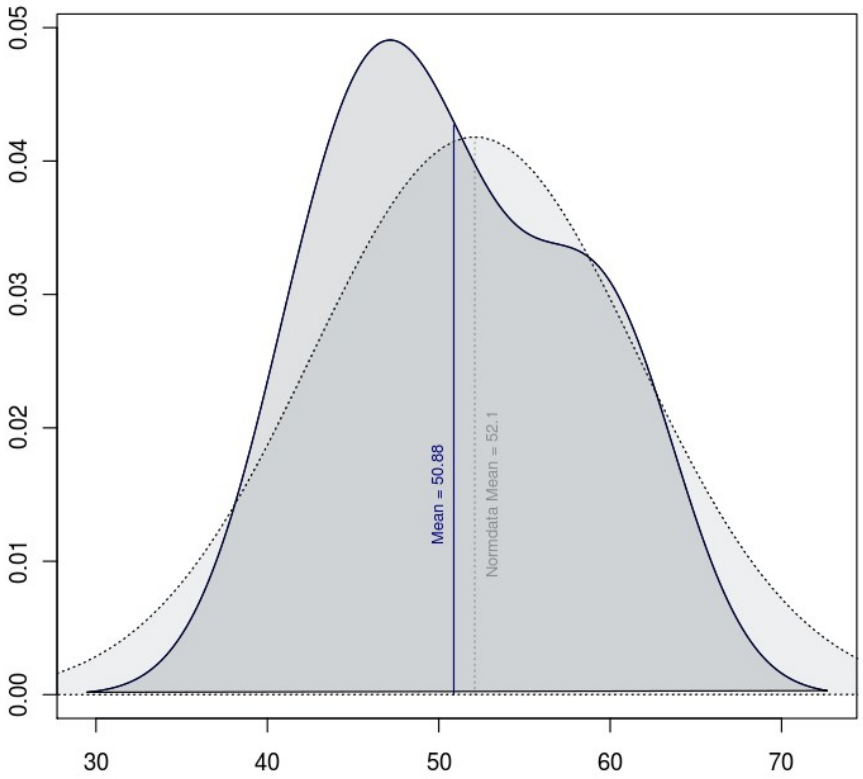


Abb. 2.38: Ein Plot mit klassischem R erstellt

In diesem Kapitel werden die Plotfunktionen des *klassischen* R vorgestellt. Sie erzeugen immernoch brauchbare Ergebnisse, und man kann viele Einstellungen vornehmen. Wenn Sie allerdings ein etwas komplexeres Diagramm aufbauen möchten, lohnt es sich nicht, die Spezialeinstellungen der klassischen Plotfunktionen zu durchstöbern. Schwenken Sie in diesem Falle lieber gleich zum moderneren `ggplot()` um, welches in Kapitel 2.4 ab Seite 352 vorgestellt wird. So kommen Sie viel "einfacher" an Ihre Ziel.

Die universelle Funktion zum erstellen von Diagrammen heisst `plot()`.
Mit ihr werden ein Großteil der Diagramme erstellt, und viele Funktionen rufen letztendlich auch nur `plot()` auf.

Da `plot()` so universal ist, nimmt sie viele Parameter entgegen. Tabelle 2.1 (Seite 321) zeigt diese Parameter in einer Übersicht. Viele der Parameter sind auch in weiteren Plotfunktionen implementiert. Es kann dennoch nie schaden, sich mit `?barplot` in der R-Konsole die explizieten Parameter zu vergegenwärtigen.

Parameter	Argumente / Beschreibung	Beispiel
`type` (mittlerweile veraltet, funktioniert aber noch)	legt den Diagrammtyp fest `l` = Liniendiagramm `p` = Punktdiagramm `b` = (**b**oth) Kombination aus `l` und `p` `s` = Stufendiagramm `h` = Histogramm-*artig(!)* `n` = keine Darstellung	`type="p"`

`col`	Füllfarbe
	(`colors()` zeigt alle Farben)
	`gray0` = schwarz bis `gray99` = weiß
		`col="blue"`
		`col="#112233"` (RGB)
		`col="gray77"`

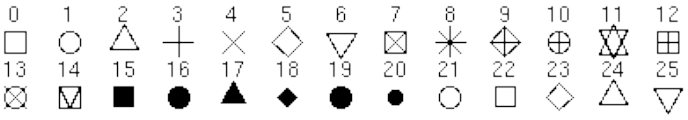
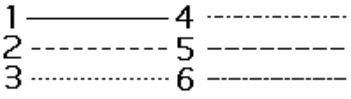
`main`	Überschrift	`main="Mein Plot"`
`sub`	Untertitel	`main="Abbildung 12"`
`xlab`	Beschriftung der X-Achse	`xlab="Jahrgänge"`
`ylab`	Beschriftung der Y-Achse	`ylab="Häufigkeit"`
`xlim`	Start- und Endwert der X-Achse	`xlim=c(2,10)`
`ylim`	Start- und Endwert der Y-Achse	`ylim=c(0,100)`
`xaxt`	X-Achse komplett unbeschriftet	`xaxt="n"`
`yaxt`	Y-Achse komplett unbeschriftet	`yaxt="n"`
`pch`	Darstellungssymbol 	`pch=4`
`cex`	Symbolgröße	`cex=2`
`lty`	Linientyp 	`lty=1`
`lwd`	Strichstärke der Linien	`lwd=3`
`bty`	Boxtyp, setzt Rahmen ums Plot `n` = kein Rahmen `o` = komplett `C` = oben, unten und links `L` = unten und links `u` = unten, rechts und links `7` = oben und rechts (die Buchstaben sehen so aus wie der Rahmen)	`bty="o"`
`font`	Fontformat `1` = normal `2` = fett `3` = kursiv `5` = griechische Symbole	font=2

Tabelle 2.1: Parameter der Funktion `plot()`

2.3.1 Punktwolke

Die Punktwolke wird mit besagter Funktion `plot()` erstellt. Ihr werden zwei Vektoren mit den entsprechenden x und y Werten übergeben.

```
# erzeuge Testwerte
x <- sort(rnorm(100, mean=12, sd=2))
y <- sort(rnorm(100, mean=140, sd=15))

# Plotten
plot(x,y)
```

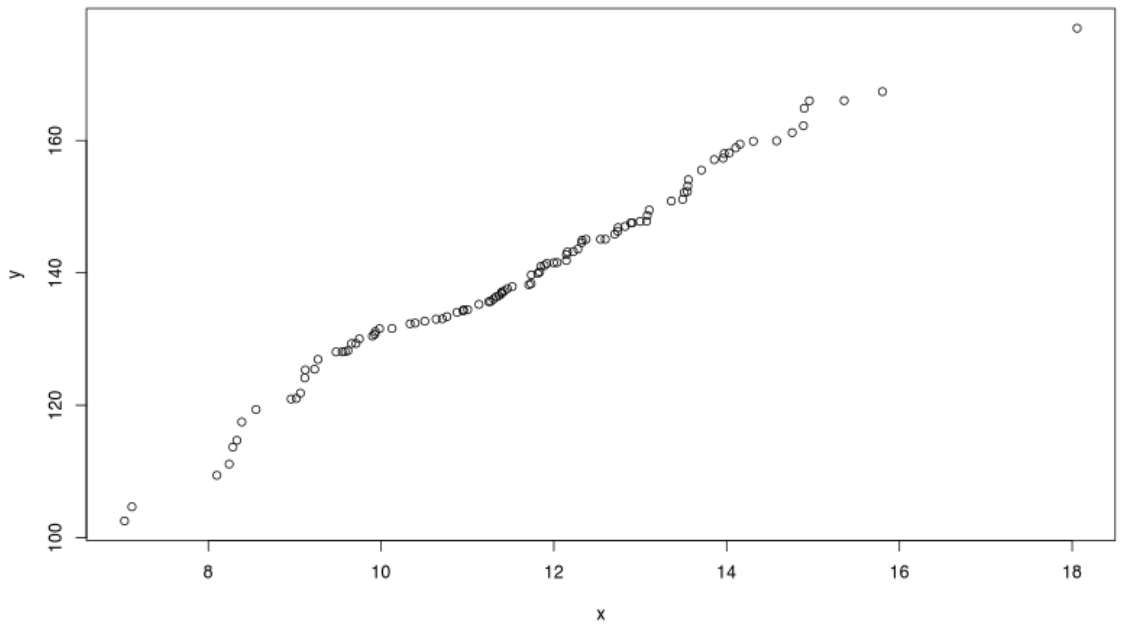


Abb. 2.39

Über die Parameter aus Tabelle 2.1 kann das Plot aufgehübscht werden

```
plot(x,y, col="blue",
          ylab="Körpergröße", xlab="Alter",
          main="Ein Plot mit R base",
          sub="Dieses Plot wurde aus Zufallswerten erstellt",
          pch=13)
```

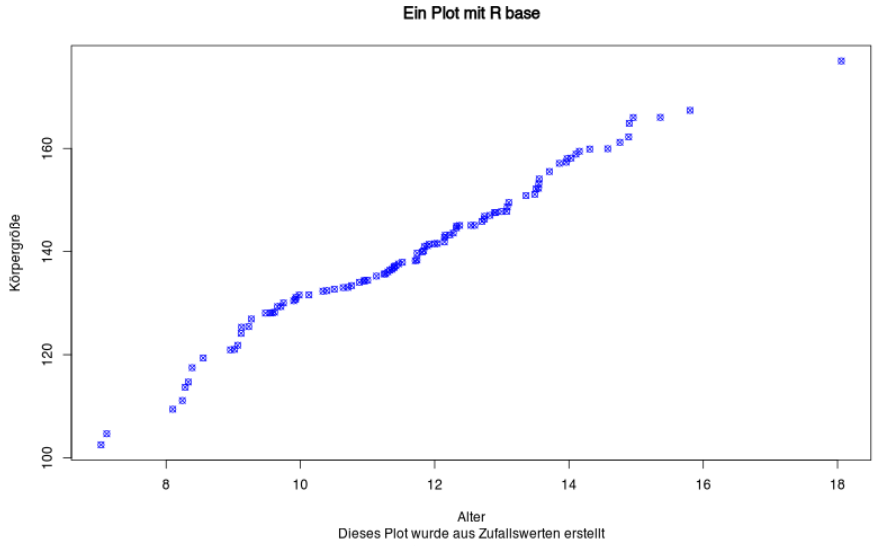


Abb. 2.40

Mit der Funktion `abline()` kann dem Plot eine Gerade hinzugefügt werden, z.B. eine Regressionsgerade. Die Funktion benötigt als Angaben den Schnitt durch die Y-Achse sowie den Steigungswert. Die Funktion `lm()` liefert für Regressionsgeraden genau diese Werte.

```
# Regressionswerte von Körpergröße ~ Alter
lm(y~x)
```

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      59.50         6.89
```

Das heisst, wir rufen die Funktion innerhalb von `abline()` auf

```
plot(x,y, col="blue",
          ylab="Körpergröße", xlab="Alter",
          main="Ein Plot mit R base",
          sub="Dieses Plot wurde aus Zufallswerten erstellt",
          pch=13)
abline(lm(y~x))
```

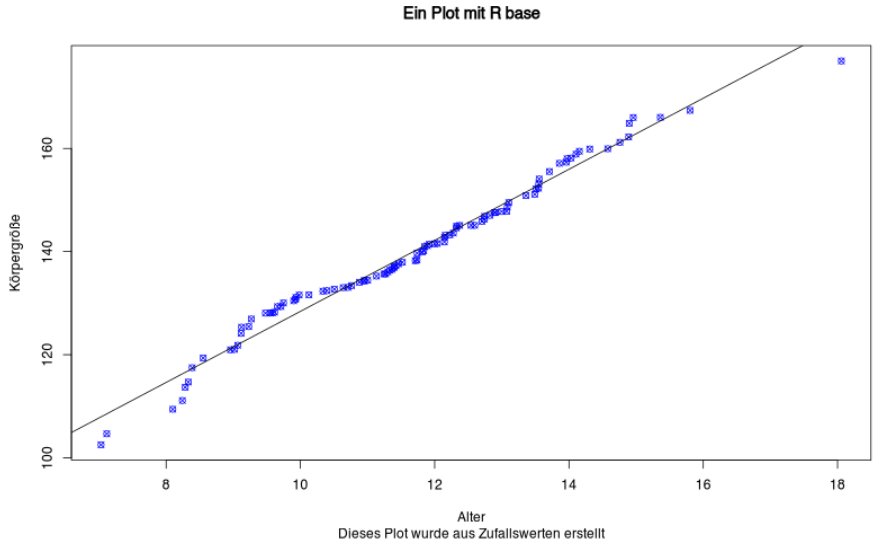


Abb. 2.41

In rot sieht sie evtl. besser aus:

```
plot(x,y, col="blue",
          ylab="Körpergröße", xlab="Alter",
          main="Ein Plot mit R base",
          sub="Dieses Plot wurde aus Zufallswerten erstellt",
          pch=13)
abline(lm(y~x), col="red")
```

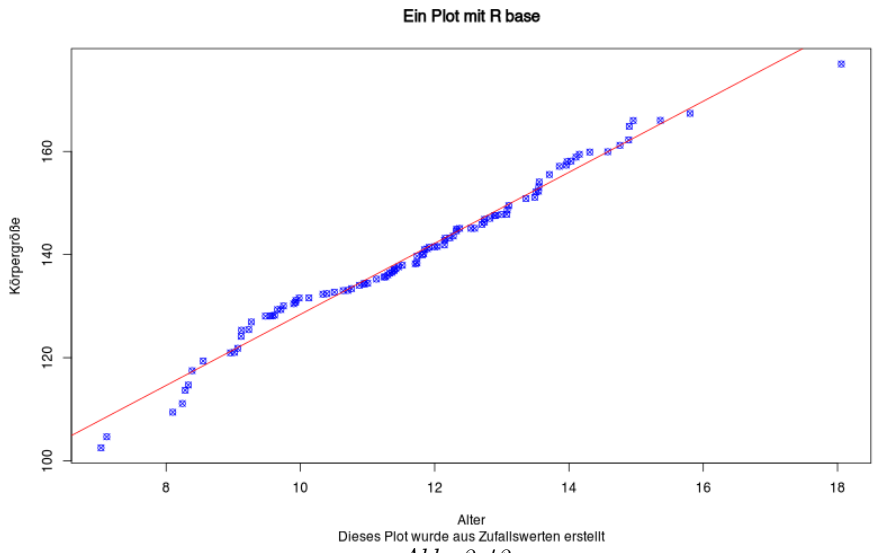


Abb. 2.42

Mit der Funktion `lines()` können Linien hinzugefügt werden. Dafür übergeben wir die entsprechenden Koordinaten des Start- und Endpunkts. So können wir z.B. den Mittelwert für `Alter` markieren

```
# Mittelwer als x-Koordinate
mx <- mean(x)

# Berechne Schnittpunkt mit Regressionsgeraden
```

```
ymx <-  coef(lm(y~x))[1] + mx*coef(lm(y~x))[2]

plot(x,y, col="blue",
          ylab="Körpergröße", xlab="Alter",
          main="Ein Plot mit R base",
          sub="Dieses Plot wurde aus Zufallswerten erstellt",
          pch=13)
abline(lm(y~x), col="red")

# Füge Mittelwertlinie hinzu
lines(x=c(mx, mx), y=c(0, ymx), lty=3)
```

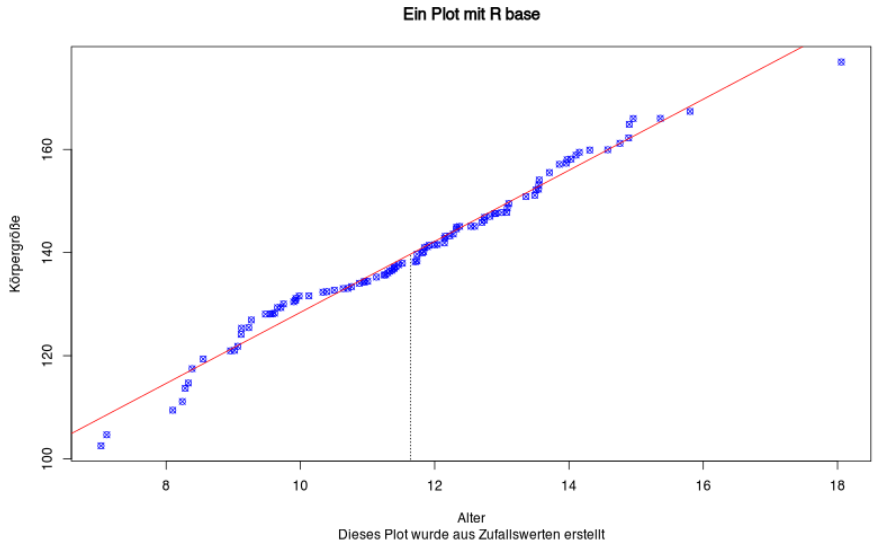


Abb. 2.43

Liegen mehrere Variablen in einem Datenframe vor, stellt `plot()` nach Kombinationen getrennt dar.

```
x <- data.frame(Lesetest = sort(rnorm(100, mean=20, sd=6)),
                Rechentest = rnorm(100, mean=15, sd=12),
                Rechtschreibung = sort(rnorm(100, mean=21, sd=4))
)

plot(x)
```

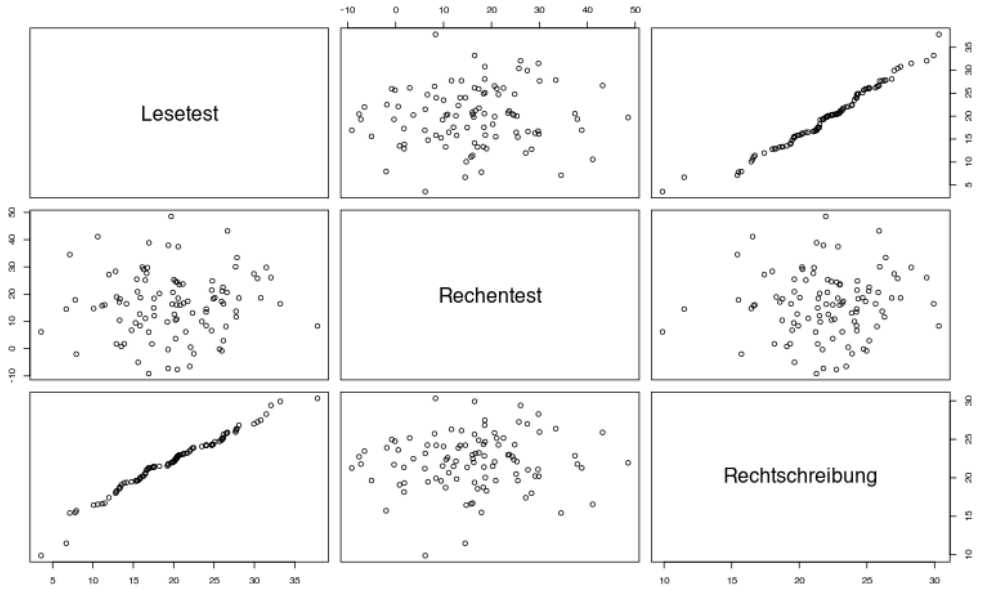


Abb. 2.44

Je mehr Variablen vorliegen, desto komplexer wird der Plot

```
x <- data.frame(Lesetest = sort(rnorm(100, mean=20, sd=6)),
                Rechentest = rnorm(100, mean=15, sd=12),
                Rechtschreibung = sort(rnorm(100, mean=21, sd=4)),
                Grammatikfehler = sort(rnorm(100, mean=21, sd=4), decreasing=T)
)

plot(x)
```

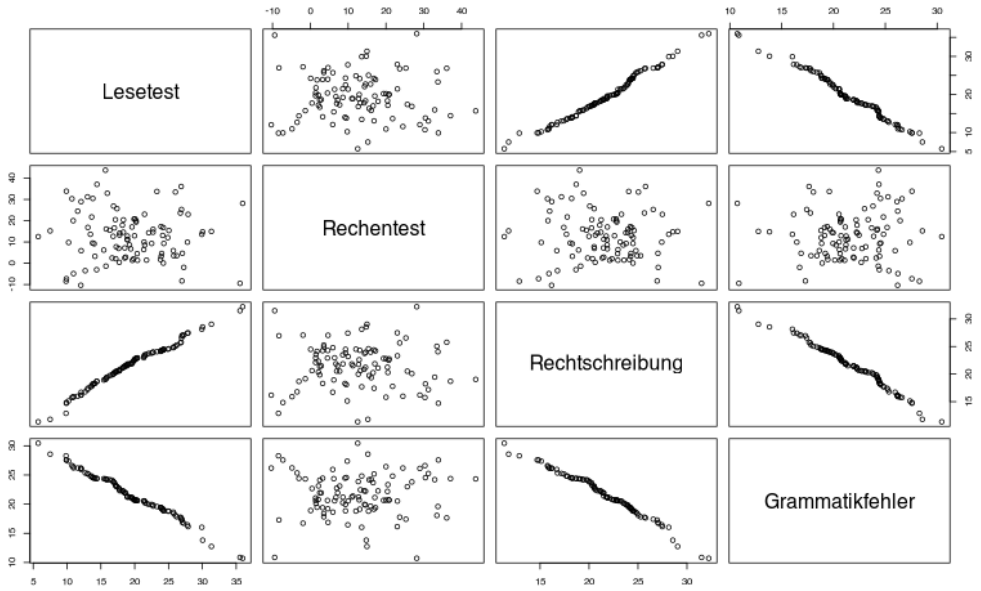


Abb. 2.45

Mit Farben kann man es etwas aufhübschen.

```
x <- data.frame(Lesetest = sort(rnorm(100, mean=20, sd=6)),
                Rechentest = rnorm(100, mean=15, sd=12),
                Rechtschreibung = sort(rnorm(100, mean=21, sd=4)),
                Grammatikfehler = sort(rnorm(100, mean=21, sd=4), decreasing=T)
)

plot(x, col=rainbow(3))
```

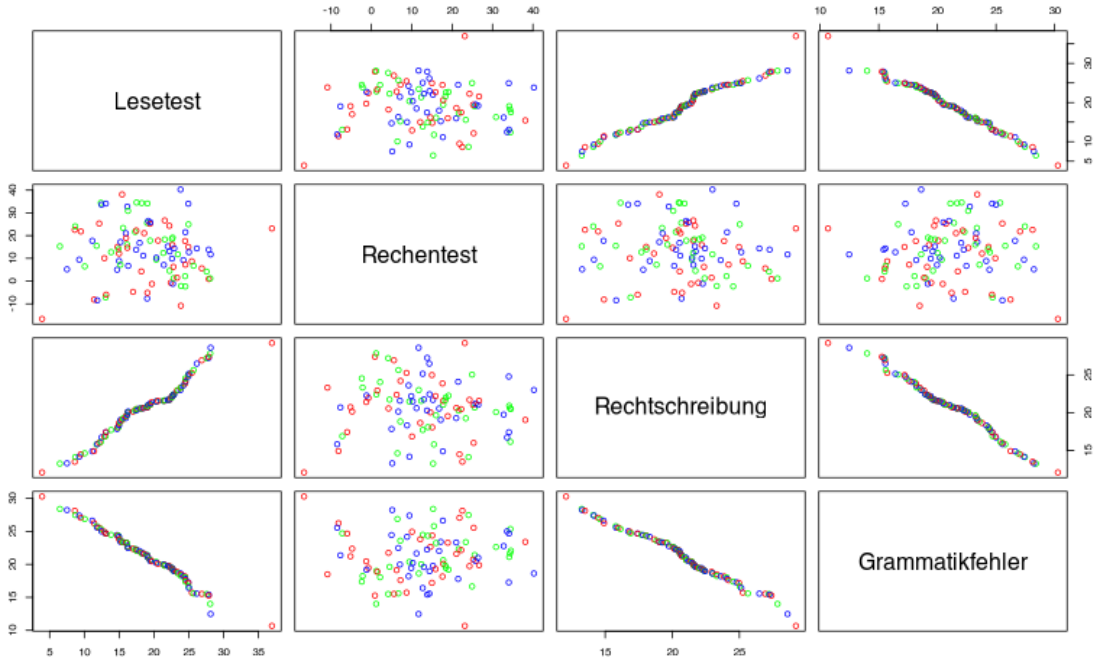


Abb. 2.46

2.3.2 verfügbare Farben

Die Funktion `colors()` zeigt die verfügbaren Farben an.

```
# zeige verfügbare Farben an
colors()
```

```
  [1] "white"                "aliceblue"            "antiquewhite"
  [4] "antiquewhite1"        "antiquewhite2"        "antiquewhite3"
  [7] "antiquewhite4"        "aquamarine"           "aquamarine1"
 [10] "aquamarine2"          "aquamarine3"          "aquamarine4"
 [13] "azure"                "azure1"               "azure2"
 [16] "azure3"               "azure4"               "beige"
 [19] "bisque"               "bisque1"              "bisque2"
 [22] "bisque3"              "bisque4"              "black"
 [25] "blanchedalmond"       "blue"                 "blue1"
 [28] "blue2"                "blue3"                "blue4"
 [31] "blueviolet"           "brown"                "brown1"
 [34] "brown2"               "brown3"               "brown4"
 [37] "burlywood"            "burlywood1"           "burlywood2"
 [40] "burlywood3"           "burlywood4"           "cadetblue"
 [43] "cadetblue1"           "cadetblue2"           "cadetblue3"
 [46] "cadetblue4"           "chartreuse"           "chartreuse1"
```

```
[49] "chartreuse2"          "chartreuse3"          "chartreuse4"
[52] "chocolate"            "chocolate1"           "chocolate2"
[55] "chocolate3"           "chocolate4"           "coral"
                    ( . . . )
```

Der Output ist 657 Farben lang.

```
1  # wieviele Farben gibt es?
2  length(colors())
```

```
[1] 657
```

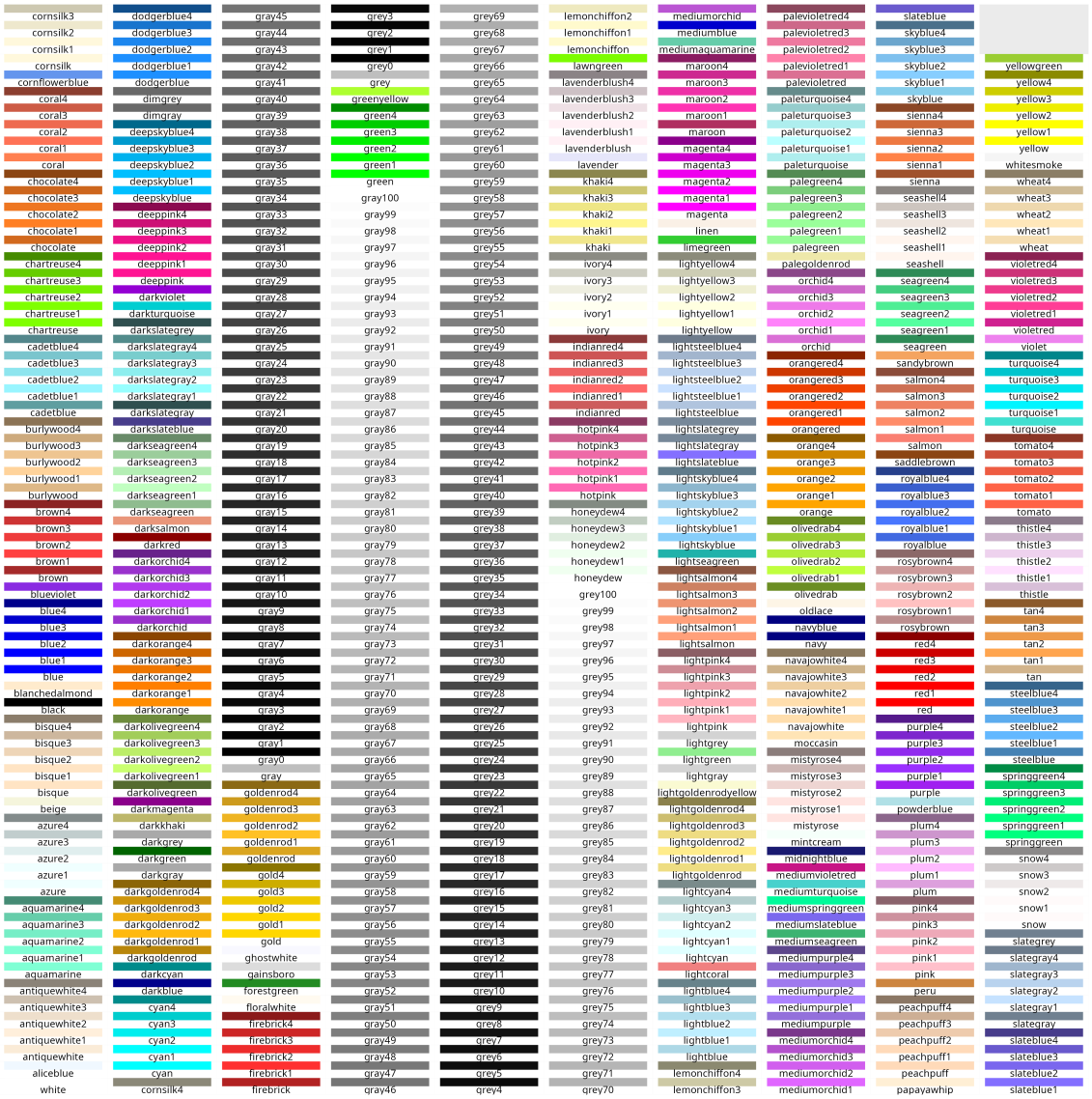


Abb. 2.47: Alle R Farben

Der R-Code für Abbildung 2.47 ist in Abschnitt 3.2.1 auf Seite 424 hinterlegt.
Eine weitere Übersicht der Farben in "echten Farbtönen" findet sich bei Björn Walther:
http://www.bjoernwalther.com/farben-in-r-der-col-befehl/

Zu erwähnen ist noch die Farbe `gray`, die in den Abstufungen `gray0` (=schwarz) bis `gray99` (= weiß) knapp 99 Grauschattierungen ermöglicht. Dies ist bei schwarz-weiß Publikationen sehr hilfreich.

Soll es möglichst bunt sein, erzeugt die Funktion `rainbow()` die gewünschte Anzahl an Farben.

2.3.3 Liniendiagramm

Ein Liniendiagramm wird mit der Parametereinstellung `type="l"` erzeugt.

```
# Erzeuge ein paar Daten
x  <-1:20
y1 <- x*x+1

# Erzeuge Liniendiagramm
plot(x, y1, type="l", pch=19, col="red", xlab="x", ylab="y")
```

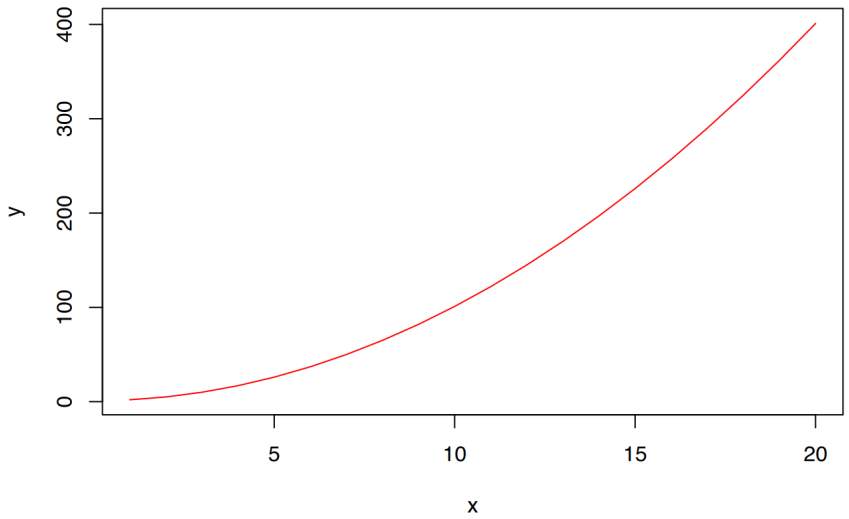


Abb. 2.48

Mit `type="b"` (für `both`) wird sowohl Punkt- als auch Liniendiagramm erzeugt.

```
plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
```

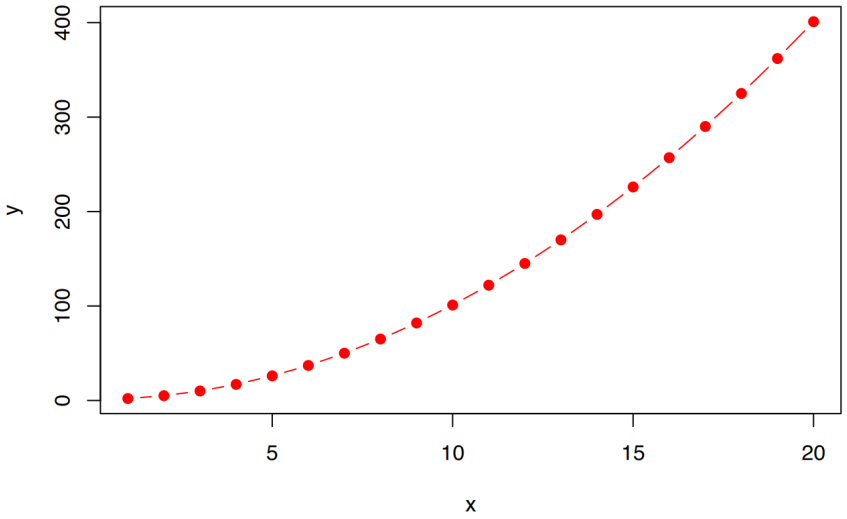


Abb. 2.49

Mit der Funktion `lines()` können weitere Linien hinzugefügt werden.

```
# erzeuge zweite Y-Reihe
y2 <- 2*y1

plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
# Füge Linie hinzu
lines(x, y2, pch=18, col="blue", type="b", lty=2)
```

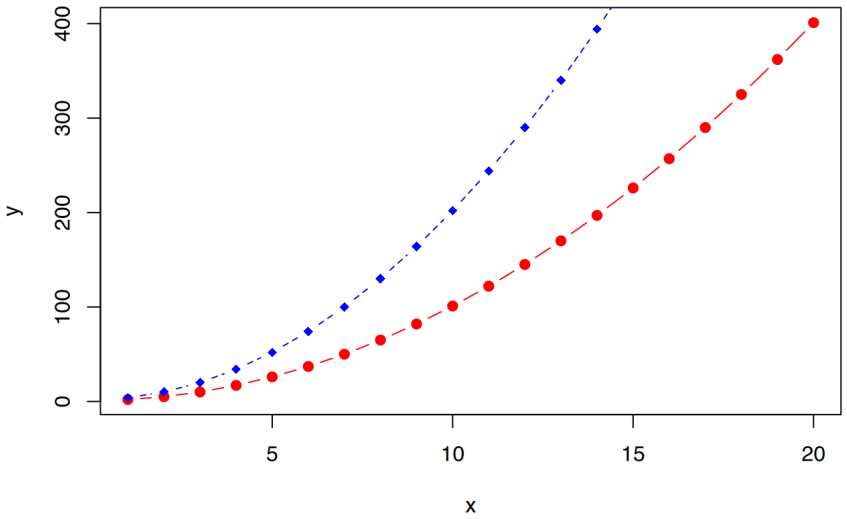


Abb. 2.50

2.3.4 Histogram

Ein Histogram wird mit der Funktion `hist()` erstellt. Plotten wir ein Histogramm der Variable `Alter` im Beispieldatensatz `pf8`.

```
# lade Datensatz "pf8"
load(url("https://www.produnis.de/R/data/pf8.RData"))

# erstelle Histogramm von Variable "Alter" aus Datensatz "pf8"
hist(pf8$Alter)
```

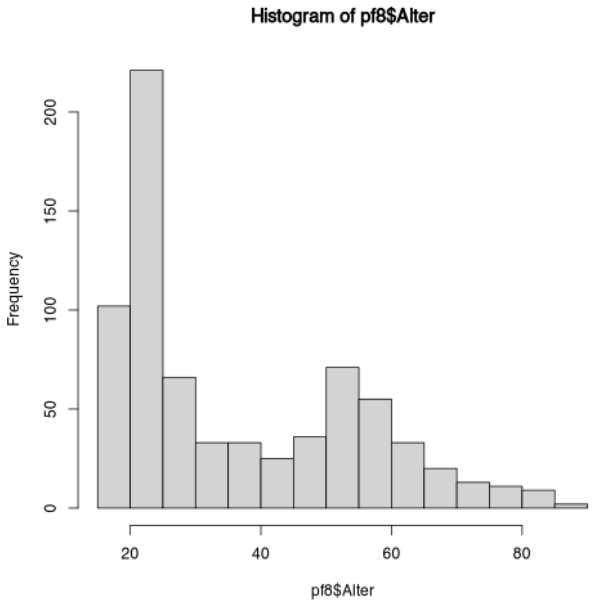


Abb. 2.51

```
# zum Vergleich plot()
plot(pf8$Alter, type="h")
```

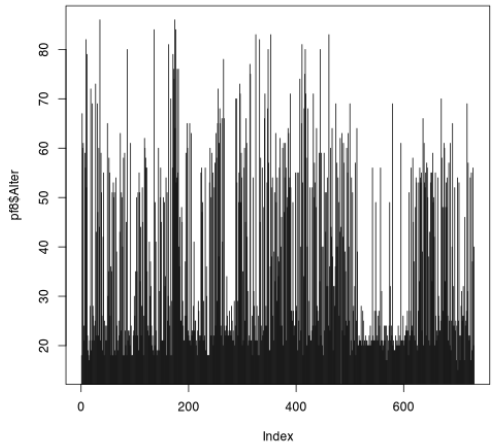


Abb. 2.52

Wie Sie sehen ist das `plot()`-Resultat nur entfernt dem Histogram ähnlich.

Die Klassengröße des Histograms kann mit dem Parameter `breaks()` übergeben werden. Entweder schreiben wir die Anzahl der "Balken", dann versucht R das "möglichst gut" auszuführen, oder wir übergeben einen Vektor mit den konkreten Breakpunkten.

```
1  # Histogramm mit "möglichst" 4 Balken
2  hist(pf8$Alter, breaks=4)
```

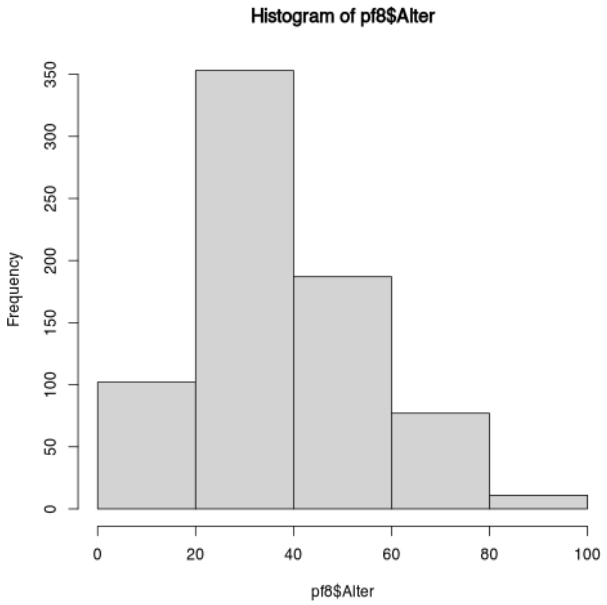


Abb. 2.53

Wie Sie sehen stellt R dennoch 5 Balken dar.
Über den Parameter `breaks` können die Klassengrenzen manuell gesetzt werden.

```
1  hist(pf8$Alter, breaks=c(10, 30, 60, 90, 100))
```

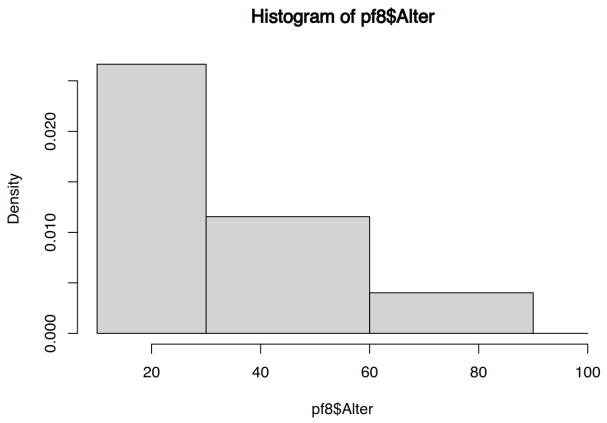


Abb. 2.54

Über die Parameter (siehe Tabelle 2.1 (Seite 321) kann der Plot nun aufgehübscht werden.

```
# Histogramm mit "möglichst" 4 Balken
# in schön
hist(pf8$Alter, breaks=4,
     # Überschrift
     main="Mein Histogramm",
     # Unterschrift
     sub="Abbildung x.y: Ein Histogramm",
               # X-achse
               xlab="Alter in Jahren",
               # Y-Achse
               ylab="Häufigkeit",
               # blaue Balken
               col="blue")
```

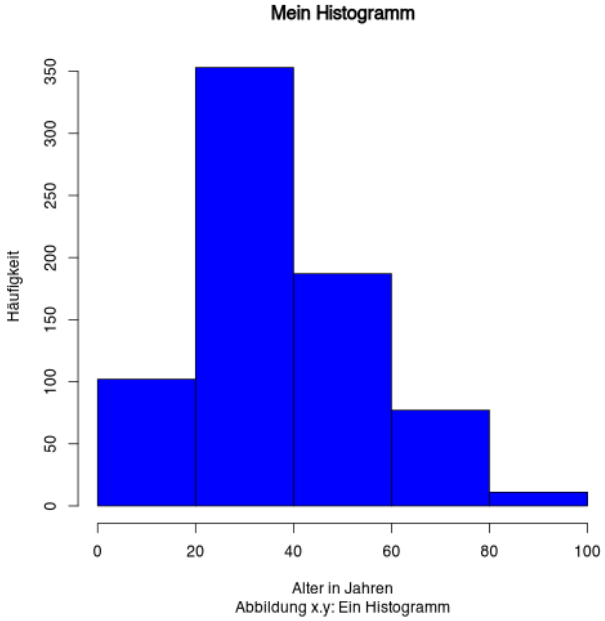


Abb. 2.55

Mittels des Parameters `freq=FALSE` kann auf relative Häufigkeiten umgeschaltet werden.

```
hist(pf8$Alter, breaks=c(0, 20, 40, 60, 90), freq=FALSE)
```

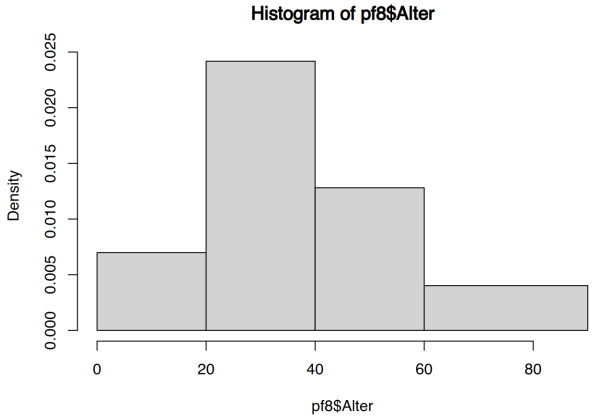


Abb. 2.56

Für kumulierte Häufigkeiten muss das Histogramm zunächst in ein Objekt geschrieben werden.

```
# speichere in Objekt h
h <- hist(pf8$Alter, breaks=5, plot=FALSE)
```

Nun können wir auf alle Werte des Histograms zugreifen. Wir ersetzen die gezählten Häufigkeiten (`h$counts`) mittels `cumsum()` durch kumulierte Werte.

```
# ersetze die Zellen durch kumulierte Häufigkeiten
h$counts <- cumsum(h$counts)

# plotte das kumulative Histogram
plot(h, col="hotpink", main = "kumulierte Häufigkeiten")
# füge Polygonzug hinzu
lines(c(h$breaks),c(0,h$counts), col="blue") # type="s" für `Steps`
```

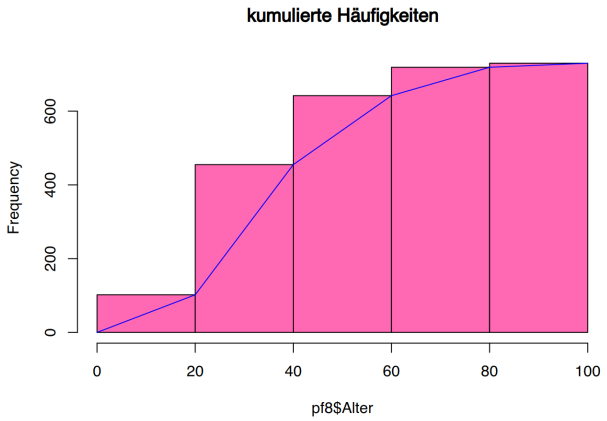


Abb. 2.57

Bei den Polygonzügen können wir die X-Werte noch mittig setzen (`h$mids`), und den Linienstil auf `steps` ändern.

```
# ersetze die Zellen durch kumulierte Häufigkeiten
# plotte das kumulative Histogram
plot(h, col="hotpink", main = "kumulierte Häufigkeiten")
# füge Polygonzug hinzu
lines(c(0, h$mids),c(0,h$counts), col="blue", type="s")
```

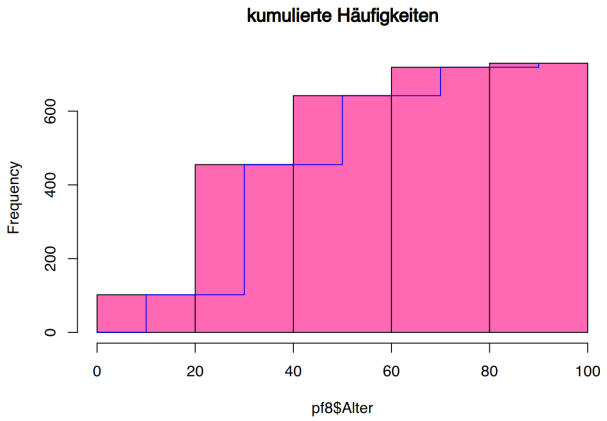


Abb. 2.58

2.3.5 Kreisdiagramm

Ein Kreisdiagramm wird mit der Funktion `pie()` erzeugt. Die Funktion benötigt einen Vektor mit den Anteilsgrößen.

```
# Kreisdiagramm 60 / 40
pie(c(60,40))
```

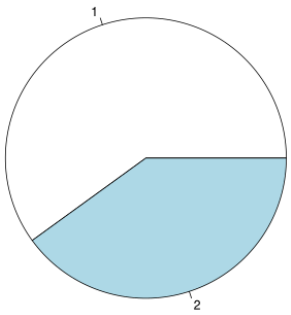


Abb. 2.59

Soll z.B. ein Kreisdiagramm aus einem Faktor erzeugt werden, gelingt dies über die Integration der Funktion `table()`

```
# Kreisdiagramm von "Geschlecht" in "pf8"
pie(table(pf8$Geschlecht))
```

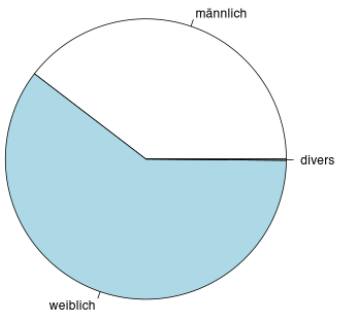


Abb. 2.60

Mit den Parametern `col`, `clockwise` und `labels` können Sie weitere Anpassungen vornehmen.

```r
pie(c(60,40), labels=c("ja", "nein"),
col=c("azure", "coral"))
```

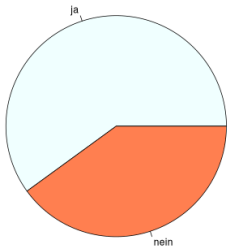


Abb. 2.61

```r
pie(table(pf8$Geschlecht), clockwise=TRUE,
col=c("burlywood", "chartreuse","black"))
```

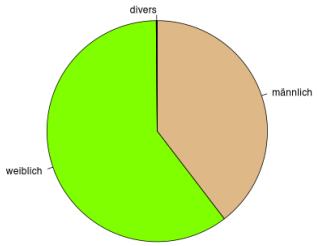


Abb. 2.62

Und auch lustige Dinge kann man plotten[2]:

```r
par(mar = c(0, 1, 0, 1))
pie(
  c(280, 60, 20),
  c('Sky', 'Sunny side of pyramid', 'Shady side of pyramid'),
  col = c('#0292D8', '#F7EA39', '#C4B632'),
  init.angle = -50, border = NA
)
```

[2] https://www.shirin-glander.de/2017/09/moving-my-blog-to-blogdown/

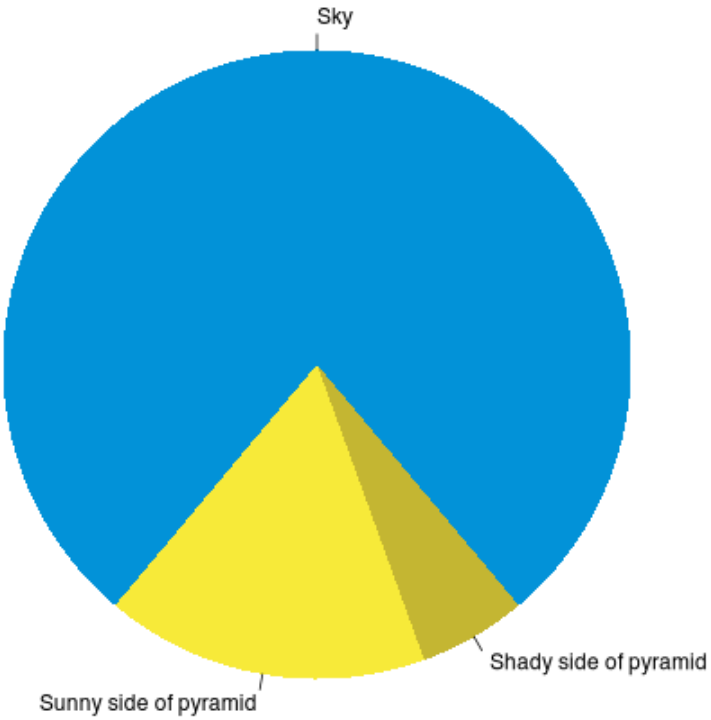


Abb. 2.63

2.3.6 Säulendiagramm

Säulendiagramme werden mit der Funktion `barplot()` erstellt. Sie benötigt einen Vektor mit den jeweiligen Ausprägungen der Säulen sowie optional die Namen der Säulen über den Parameter `names.arg`:

```
# Säulendiagramm
barplot(c(10,40,20), names.arg=c("Dritter", "Erster", "Zweiter"))
```

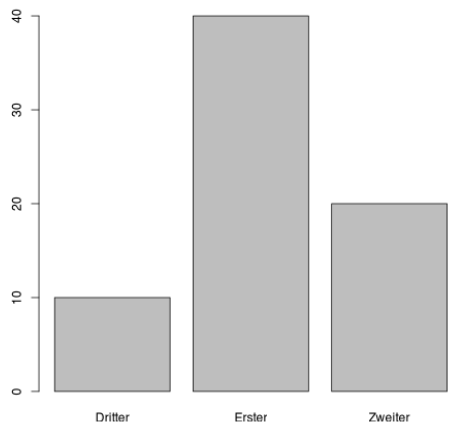


Abb. 2.64

Liegen die Daten als Faktor vor, können sie mit Hilfe der Funktion `table()` übergeben werden

```
# Säulendiagramm von "Standort" in "pf8"
barplot(table(pf8$Standort))
```

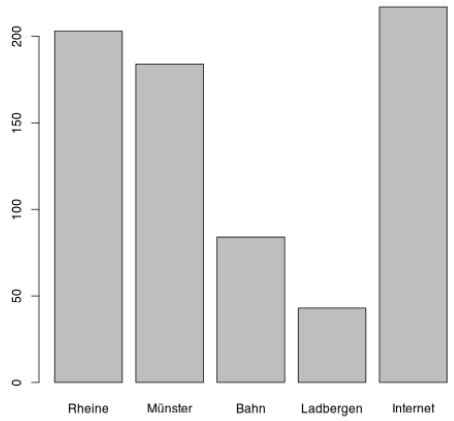


Abb. 2.65

Liegen die Daten als `wide table` vor (z.B als Matrix mit bedeutsamen `rownames`), so wie unsere Matrix der Pflegeberufe (siehe Seite 36), erstellt `barplot()` gruppierte Säulen. Über das Argument `legend.text` können die Reihennamen übergeben werden.

```
# Säulendiagramm der Matrix Pflegeberufe
barplot(Pflegeberufe, beside=T, col=rainbow(5), legend.text = rownames(Pflegeberufe))
```

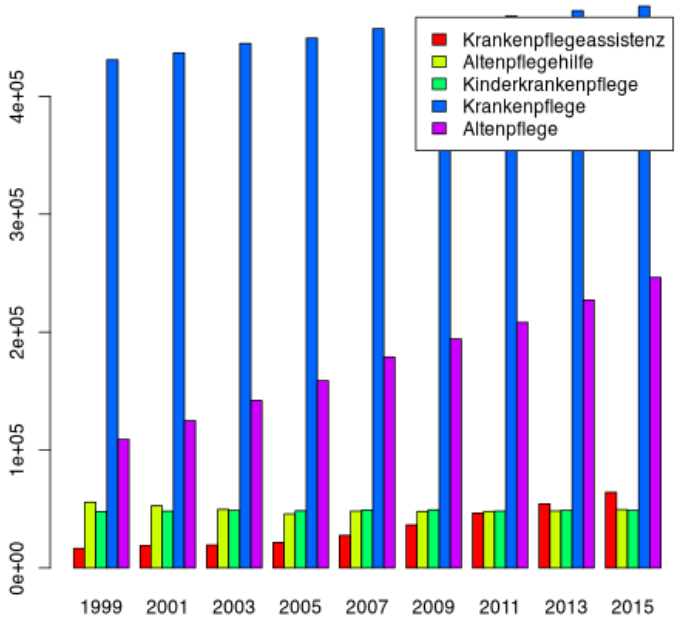


Abb. 2.66

Wird der Parameter `beside` weggelassen (entspricht `FALSE`), werden die Reihen in einer gemeinsamen Säule zusammengefasst.

```
# Säulendiagramm der Matrix Pflegeberufe
barplot(Pflegeberufe, col=rainbow(5), legend.text = rownames(Pflegeberufe))
```

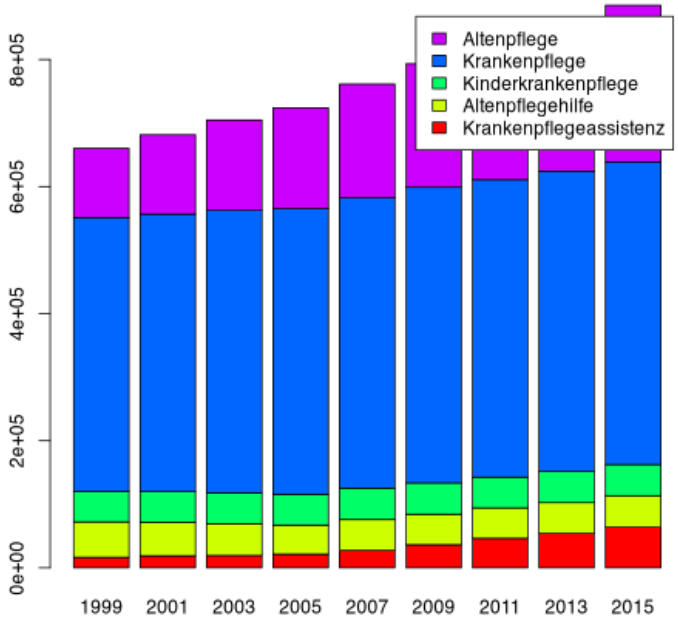


Abb. 2.67

Das entspricht vom Prinzip her der Abbildung auf Seite 36.

2.3.7 Balkendiagramm

Balkendiagramme werden genau so erstellt wie Säulendiagramme, nämlich über die Funktion `barplot()` mit dem Parameter `horiz=TRUE`.

```
# Balkendiagramm der Matrix Pflegeberufe
# diesmal Schwarz-weiss
barplot(Pflegeberufe,
            # schalte um auf "Balkendiagramm"
            horiz=TRUE,
            # Graustufen   führende 0 kann weggelassen werden
            col=gray(c(.1, .2, .3, .4, .5)),
            legend.text = rownames(Pflegeberufe))
```

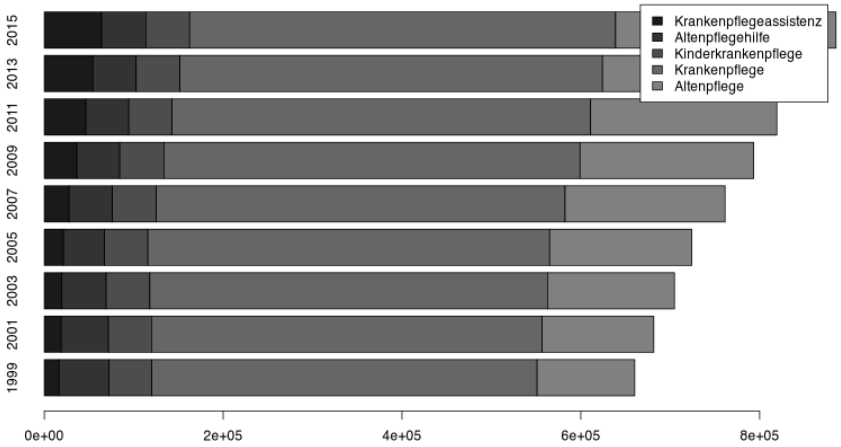


Abb. 2.68

```
# Balkendiagramm "category" aus "mma"
# diesmal Schwarz-weiss
barplot(table(mma$category),
        horiz=TRUE,
        col=rainbow(7),
        # Legende aus Faktorenlevels
        legend.text = levels(mma$category))

```

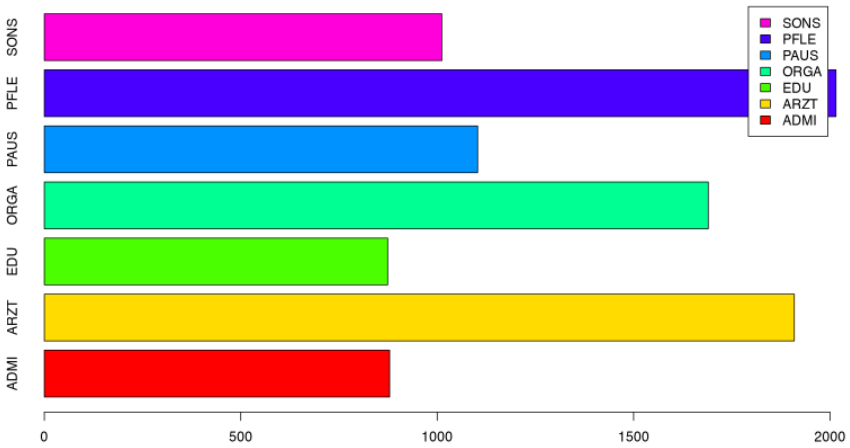


Abb. 2.69

2.3.8 Boxplot

Boxplots können mit der Funktion `boxplot()` erstellt werden.

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/nw.RData"))

# Boxplot des Alters in Datensatz "nw"
boxplot(nw$age)
```

Die Funktion erkennt auch die Syntax "*erklärt durch*" (Abbildung 2.71).

```
# Boxplot des Alters in Datensatz "nw" erklärt durch "sex"
boxplot(age ~ sex, data=nw)
```

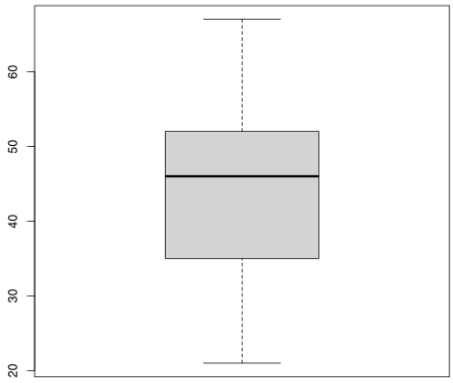


Abb. 2.70

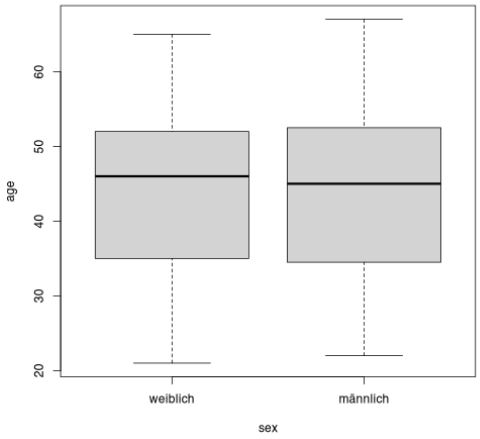


Abb. 2.71

Über die Parameter `frame`, `border`, `col` und `notch` können weitere Anpassungen erfolgen.

```
boxplot(age ~ sex, data=nw, frame = FALSE, border="blue", horizontal = TRUE)
boxplot(age ~ sex, data=nw, frame = FALSE, col=c("green", "red"), notch = TRUE)
```

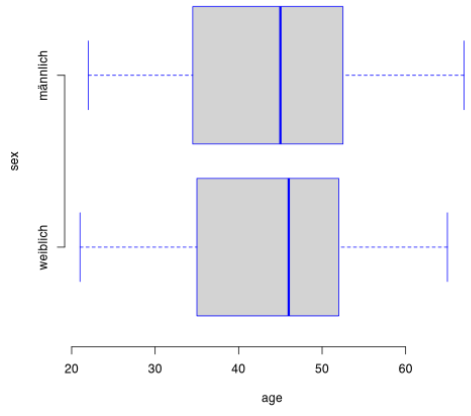


Abb. 2.72

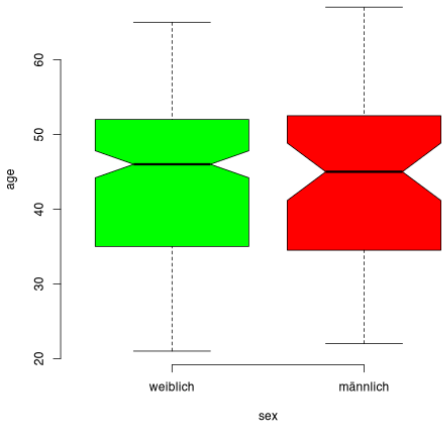


Abb. 2.73

Weitere *innere* Gruppierungen können mit einem Sternchen * übergeben werden.

```
1  boxplot(age~sex*timework, data=nw, col=c("blue", "skyblue"), notch = TRUE)
```

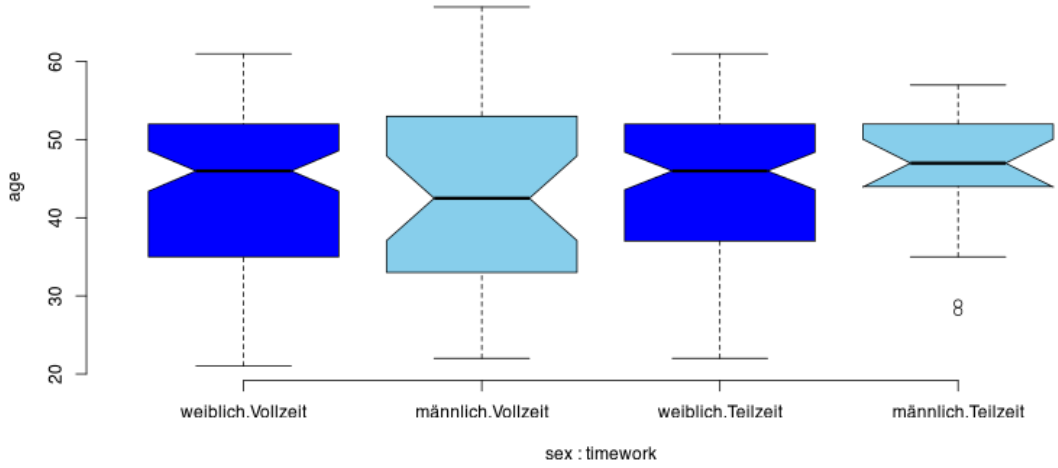


Abb. 2.74

2.3.9 Polygone

Mit der Funktion `polygon()` können Polygone hinzugefügt werden. Ein Polygon wird so gezogen, als würden wir es mit einem Stift auf Papier zeichnen, ohne den Stift dabei abzusetzen. Das heisst, wir müssen "hin und zurück" zeichnen.

```
# Mittelwer als x-Koordinate
mx <- mean(x)

# Berechne Schnittpunkt mit Regressionsgeraden
ymx <-  coef(lm(y~x))[1] + mx*coef(lm(y~x))[2]

plot(x,y, col="blue",
          ylab="Körpergröße", xlab="Alter",
          main="Ein Plot mit R base",
          sub="Dieses Plot wurde aus Zufallswerten erstellt",
          pch=13)
abline(lm(y~x), col="red")

# Füge Mittelwertlinie hinzu
lines(x=c(mx, mx), y=c(0, ymx), lty=3)

# füge Polygon hinzu
polygon(x=c(x, max(x), min(x)), y=c(y, 0, 0), col="snow2")
```

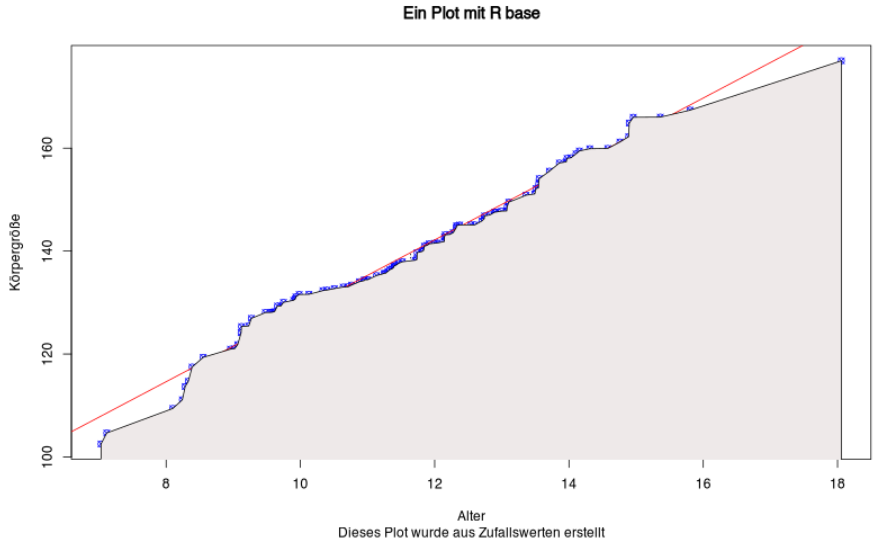


Abb. 2.75

So kann ein Diagramm der Standardnormalverteilung erstellt werden, wobei beliebige Bereiche hervorgehoben werden können.

```
# erzeuge 10.000 Zufallswerte der Standardnormalverteilung
x <- sort(rnorm(10000))

# erzeuge die passenden Dichterwerte
y <- dnorm(x)

# beginne mit Plot
plot(x, y, ylab="Dichtefunktion", xlab="x", main="Die Standardnormalverteilung",
     sub="Bereich von 1 Standardabweichung")

# füge gefülltes Polygon hinzu
polygon(x=c(x, max(x), min(x)), y=c(y, 0,0 ), col="snow2")
```

```
13
14  # füge 1 Standardabweichung hinzu
15  lines(x=c(-1, -1), y=c(0, dnorm(-1)), lty=3)
16  lines(x=c(1, 1), y=c(0, dnorm(1)), lty=3)
17
18  # füge Polygon für 1. Standardbweichung hinzu
19  xsd1 <- x[x<1 & x>-1]
20  ysd1 <- dnorm(xsd1)
21  polygon(x=c(xsd1, max(xsd1), min(xsd1)), y=c(ysd1, 0, 0 ), col="green")
```

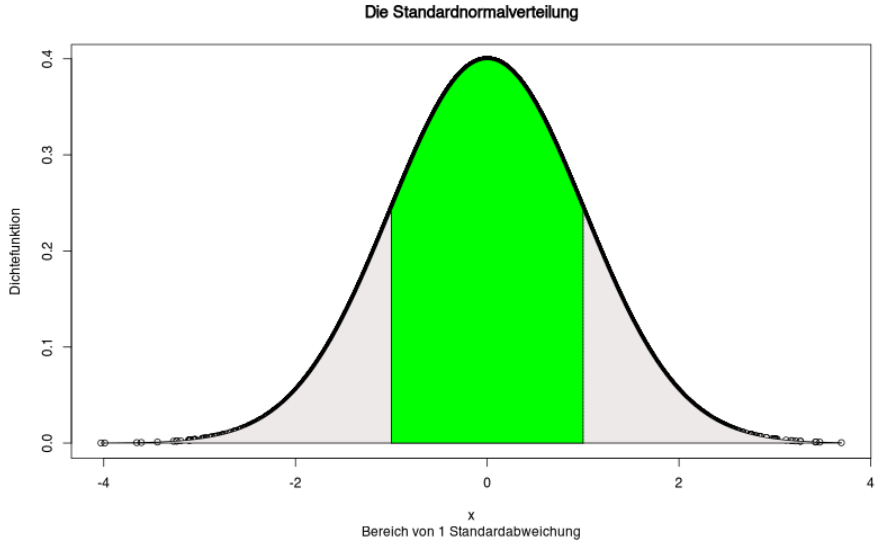


Abb. 2.76

Der Bereich von zwei Standardabweichungen kann wie folgt hinzugefügt werden. Dies "übermalt" jedoch die grüne Fläche.

```
1  # füge 2. Standardabweichung hinzu
2  lines(x=c(-2, -2), y=c(0, dnorm(-2)), lty=6)
3  lines(x=c(2, 2), y=c(0, dnorm(2)), lty=6)
4
5  # füge Polygon für 2. Standardabweichung
6  # dies übermalt das 1. Polygon
7  xsd2 <- x[x<2 & x>-2]
8  ysd2 <- dnorm(xsd2)
9  polygon(x=c(xsd2, max(xsd2), min(xsd2)), y=c(ysd2, 0, 0 ), col="blue")
```

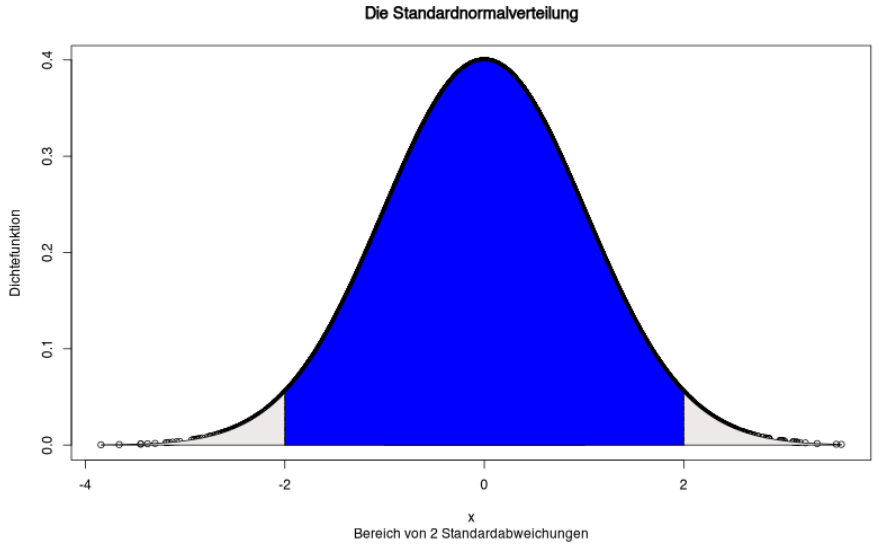


Abb. 2.77

In Kapitel 3.2.6 auf Seite 445 werden Polygone verwendet, um eine Übersichtsgrafik der BMI-Klassen in Abhängigkeit von Körpergröße und Körpergewicht zu erstellen.

2.3.10 QQ-Plots

Zur Überprüfung, ob eine Variable normalverteilt ist, werden häufig Quantil-Quantil-Diagramme (QQ-Plots) erzeugt. In R lassen sich diese mit den Funktionen `qqnorm()` und `qqline()` erstellen.

```
# QQ-Plot für "Alter" aus dem Datensatz "epa"
qqnorm(epa$age, col = "steelblue")
qqline(epa$age, col = "purple", lwd = 2)
```

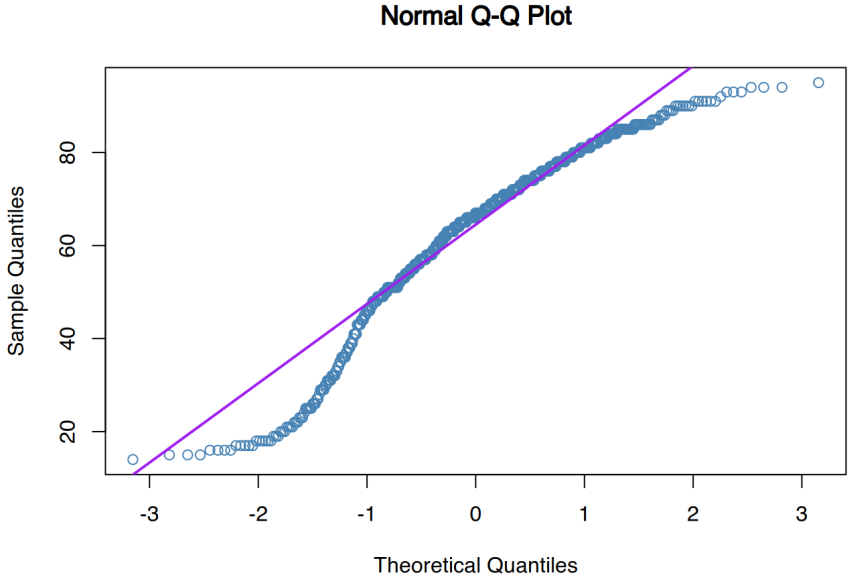


Abb. 2.78

Zusätzlich steht die abgewandelte Funktion `qqPlot()` aus dem `car`-Paket zur Verfügung. Dieses zeichnet standardmäßig das Konfidenzintervall mit ein, und kann neben der Normalverteilung auch anderen Verteilungsformen prüfen.

```
car::qqPlot(epa$age, col="darkblue",
                     col.lines = "red")
```

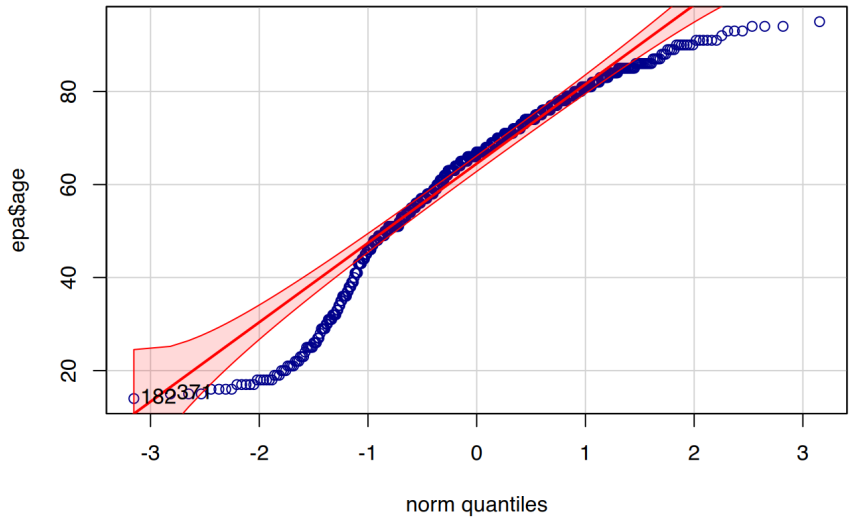


Abb. 2.79

2.3.11 Legendenbox

Mit der Funktion `legend()` kann dem Plot eine Legendenbox hinzugefügt werden. Die ersten beiden Parameter legen dabei die X- und Y-Achsenpositionen der Box fest.

```
# Erzeuge ein paar Daten
x  <-1:20
y1 <- x*x+1
y2 <- 2*y1

#
plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
# Füge Linie hinzu
lines(x, y2, pch=18, col="blue", type="b", lty=2)
# Füge Legendenbox hinzu
legend(1, 95, legend=c("Männer", "Frauen"),
       col=c("red", "blue"), lty=1:2, cex=0.8)
```

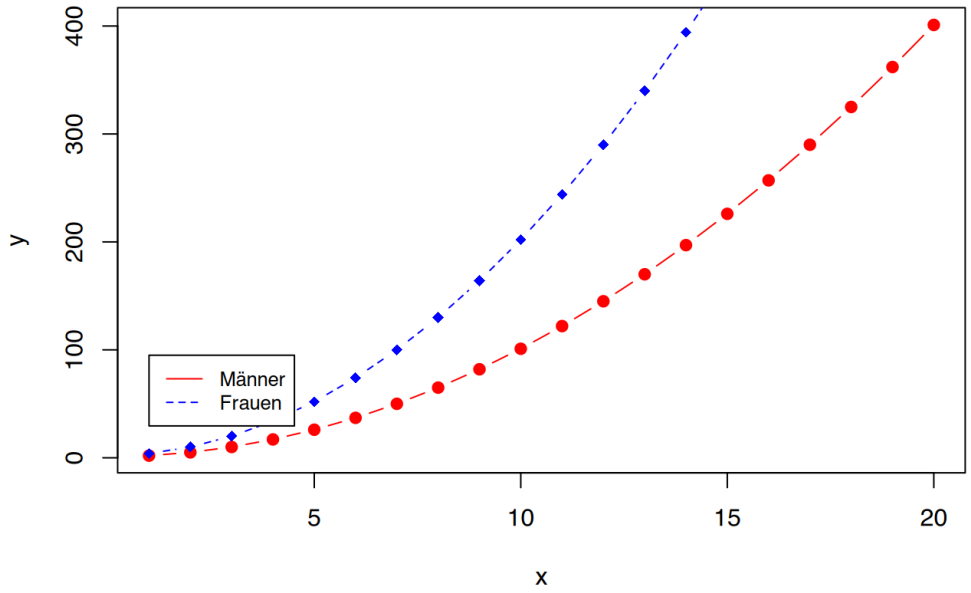


Abb. 2.80

Zusätzlich nimmt `legend()` folgende Parameter entgegen:

- `title` - Titel der Legendenbox
- `text.font` - ein Integer der das Aussehen festlegt:
  - `1` = normal
  - `2` = fett
  - `3` = kursiv
  - `4` = fett und kursiv
- `bg` - die Hintergrundfarbe der Legendenbox
- `box.lty` - die Strichart des Legenrahmens
- `box.lwd` - die Strichstärke des Rahmens
- `box.col` - die Farbe des Rahmens
- `horiz` - Box horizontal ausrichten (`TRUE`/ `FALSE`)
- `legend` - Variablennamen der Objekte
- `lty` - Strichart der Variablen
- `col` / `fill` - Füll- und Strichfarben der Variablen

```
plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
lines(x, y2, pch=18, col="blue", type="b", lty=2)
# Füge Legendenbox hinzu
legend(1, 95, legend=c("Männer", "Frauen"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       bg="lightgreen", box.lty = 4, box.lwd = 2, box.col = "purple")
```

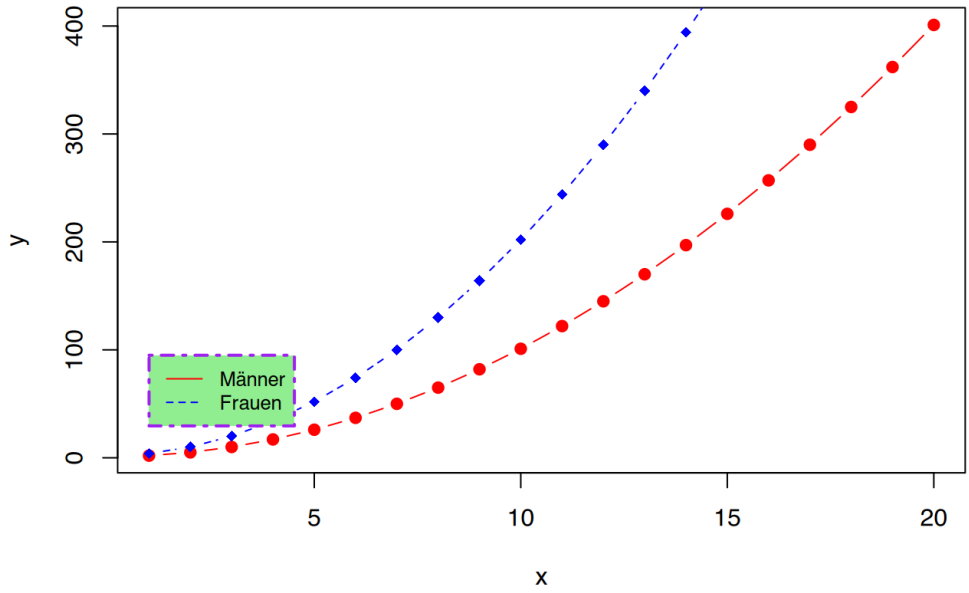


Abb. 2.81

Die Position der Box kann auch mit den Keywords `bottomright`, `bottom`, `bottomleft`, `left`, `topleft`, `top`, `topright`, `right` und `center` bestimmt werden.

```
plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
lines(x, y2, pch=18, col="blue", type="b", lty=2)
# Füge Legendenbox hinzu
legend("topleft", legend=c("Männer", "Frauen"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       bg="lightgreen", box.lty = 4, box.lwd = 2, box.col = "purple")
```

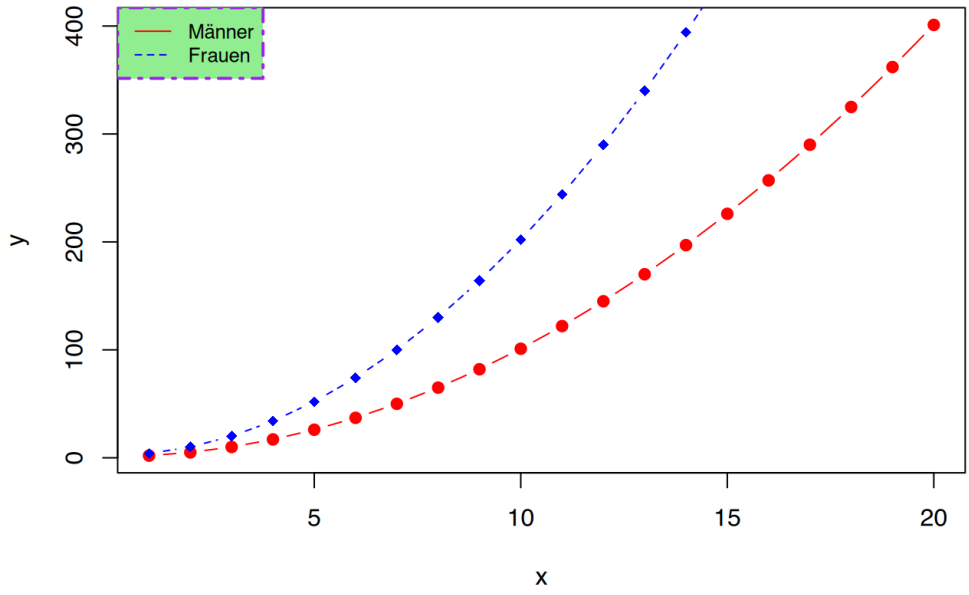


Abb. 2.82

Mit dem Parameter `horiz=TRUE` werden die Variablennamen horizontal aneinandergereiht.

```
plot(x, y1, type="b", pch=19, col="red", xlab="x", ylab="y")
lines(x, y2, pch=18, col="blue", type="b", lty=2)
# Füge Legendenbox hinzu
legend("topleft", legend=c("Männer", "Frauen"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       bg="lightgreen", box.lty = 4, box.lwd = 2, box.col = "purple",
       horiz=TRUE)
```

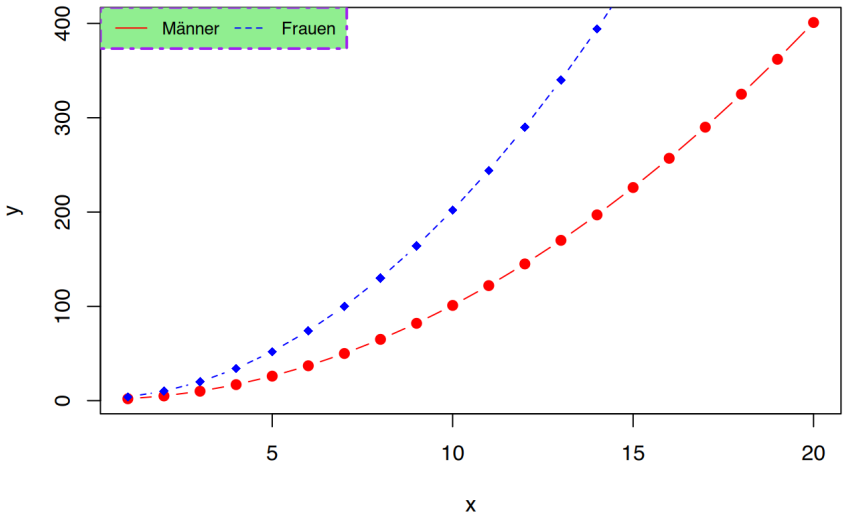


Abb. 2.83

2.3.12 Diagramme speichern

Zum Speichern der Diagramme bietet R die Formate `jpeg`, `png`, `bmp` und `tiff` an. Und genau so heissen auch die jeweiligen Speicherfunktionen. Dieser Befehl speichert das Diagramm im Format `png`:

```
# Speichere nachfolgendes als PNG
png("Testplot.png", width = 500, height = 500, units = "px", pointsize = 12)

# erzeuge Diagramm
pie(c(60,40))

# speichern abschließen
dev.off()
```

Wie Sie sehen wird zunächst die Funktion `png()` aufgerufen. In ihr wird der Dateiname des Diagramms sowie dessen Dimensionen angegeben. Anschließend folgt der Aufruf der Plotbefehle, in diesem Falle `pie()`. Mit dem Befehl `dev.off()` (für ***dev**ice off*) wird das Speichern abgeschlossen. Die eigentlichen Diagrammbefehle müssen also *zwischen* der Speicherfunktion (`png()`) und `dev.off()` stehen.

Soll das Diagramm als `jpeg` gespeichert werden, lautet die Befehlskette entsprechend:

```
# Speichere nachfolgendes als JPEG
jpeg("Testplot2.jpg", width = 500, height = 500, units = "px", pointsize = 12)

# erzeuge Diagramm
barplot(c(10,40,20), names.arg=c("Dritter", "Erster", "Zweiter"))

# speichern abschließen
dev.off()
```

Genau so funktionieren auch die Funktionen `bmp()` und `tiff()`, aber diese Formate verwendet heute eigentlich niemand mehr.

2.4 ggplot

Das Zusatzpaket **ggplot** (für ***g**rammar of **g**raphics plot*) ist Hadley Wickhams[10] -Implementation der *Grammar of Graphics* von Leland Wilkinson[11]. Es ist eines der ersten Pakete des Tidyverse (welches damals noch nicht so hieß).

Die Funktion zum plotten heisst "`ggplot()`", das Installationspaket in lautet jedoch `ggplot2`. Unter `https://ggplot2-book.org` ist eine detaillierte Anleitung in englischer Sprache von Hadely Wickham et al.[12] verfügbar.

```
# ggplot2 installieren
install.packages("ggplot2", dependencies=T)

# ggplot2 aktivieren
library(ggplot2)
```

Wenn Sie zuvor das Tidyverse (siehe Kapitel 1.6 auf Seite 141) aktiviert haben…

```
library(tidyverse)
```

… wird das `ggplot2`-Paket ebenfalls aktiviert.

Da `ggplot` aus dem *Tidyverse* stammt, ist es von Vorteil, wenn die zu verabreitenden Datensätze dem Prinzip "ein Fall pro Zeile" (*tidy data* bzw. `long table`) folgen. Das bedeutet, dass jede Beobachtung (auch Wiederholungen) in einer eigenen Zeile steht, und die jeweiligen Variablen durch die Spalten repräsentiert werden.

2.4.1 Satzbau

Die Idee einer Grammatik für Diagramme ist, dass jedes Diagramm aus den selben Komponenten (*Satzteilen*) aufgebaut werden kann. Benötigt werden:

1. Daten (möglichst im *Tidy Data* Format)
2. **Aes**thetische bzw. visuelle Zuordnungen der Daten zu den Komponenten des Plots
3. **Geom**etrische Objekte, die im Plot erscheinen sollen
4. **Stat**istische Transformation der Daten, bevor sie geplottet werden
5. **Coord**inates (Koordinaten), um die Lage der geometrischen Objekte zu bestimmen
6. **Scale**s (Skalen), um den Wertebereich der Datenzuordnung zu bestimmen
7. **Facette**n, um den Plot in Teil-Diagramme zu gruppieren

Diese Komponenten werden in `ggplot()` mit dem `+` Zeichen aneinander gereiht.

```
# sinnbildlich
ggplot(daten) +
   visuelle Zuordnung +
     Geometrische Objekte +
       statistische Transformationen +
         Koordinaten +
           Skalen +
             Facetten
```

Die enthaltenen Funktionen folgen dabei dieser Komponentenstruktur:

- Geome (Flächen, Linien, Punkte, usw) beginnen mit `geom_`, z.B.:
  - `gemo_area()`, `gemo_density()`, `gemo_smooth()`, `gemo_histogram()`, `gemo_point()`, `gemo_jitter()`, `gemo_line()`, `gemo_boxplot()`, `gemo_col()`, `gemo_text()`
- statistische Funktionen (Häufigkeiten, Mittelwerte, Dichte, usw.) beginnen mit `stat_`, z.B.:
  - `stat_count()`, `stat_density()`, `stat_function()`, `stat_summary()`
- Koordinatensystemfunktionen beginnen mit `coor_`, z.B.:
  - `coord_flip()`, `coord_trans()`
- Funktionen zum Manipulieren des grundlegenden Aussehens (Theme), beginnend mit `theme_`, z.B.:
  - `theme()`, `theme_bw()`, `theme_classic()`

In `RStudio` ist ein "Cheatsheet" (siehe Abschnitt 1.8.1 auf Seite 209) zu `ggplot` verlinkt. Hier finden Sie (fast) alle verfügbaren Komponenten zusammengefasst. Klicken Sie in der Menüzeile von `RStudio` auf `Help` → `Cheatsheets` → `Data Visualization with ggplot2`.

2.4.2 Erstes Plot

Mit dem Datensatz `jgsbook::pf8` erzeugen wir nun unser erstes ggplot.

```
# Lade Datensatz
load(url("https://www.produnis.de/R/data/pf8.RData"))
# oder
pf8 <- jgsbook::pf8

# Tidyverse aktivieren!
library(tidyverse)

# unser erstes Plot
ggplot(pf8)
```

Abb. 2.84

"*Wie Sie sehen, sehen Sie nichts.*"

Wir sehen deshalb nichts, weil wir noch nicht bestimmt haben, *was* (welche Variablen) *wie* (Punkte, Linien, Boxplots) *wo* (x-Achse, y-Achse) geplottet werden soll. Wir müssen diese "Aesthetics" erst definieren.

2.4.2.1 Aesthetics

Angenommen, uns interessiert das `Gewicht` sowie das `Alter` der Probanden. Mit der Funktion `aes()` können wir die Daten visuell zuordnen. Das `Alter` soll auf der x-Achse liegen, das `Gewicht` auf der y-Achse.

```
1  # erster Plot mit "Aesthetics"
2  ggplot(pf8) +
3    aes(x = Alter,   # x-Achse
4        y = Gewicht) # y-Achse
```

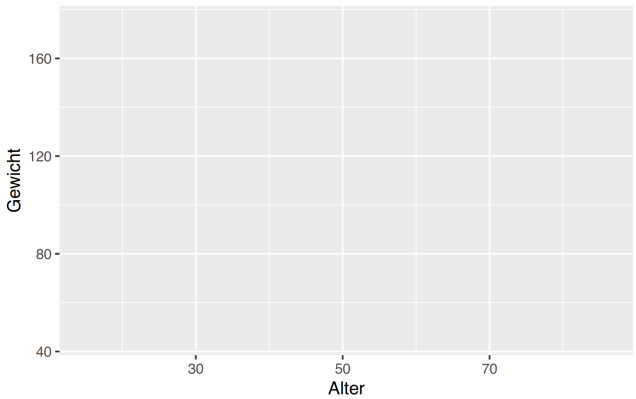


Abb. 2.85

Ggplot hat die Achsen entsprechend beschriftet und die Wertebereiche anhand der vorhandenen Daten (Minimum, Maximum) ausgewählt.

2.4.2.2 Geome

Zur Darstellung von `Gewicht` und `Alter` werden geometrische Objekte (*Geome*) benötigt. Diese werden über die Funktionen `geom_*()` erzeugt, wobei das Sternchen durch die gewünschte gemoetrischen Form ersetzt werden muss.
Die 6 meist-genutzten Geome sind sicherlich:

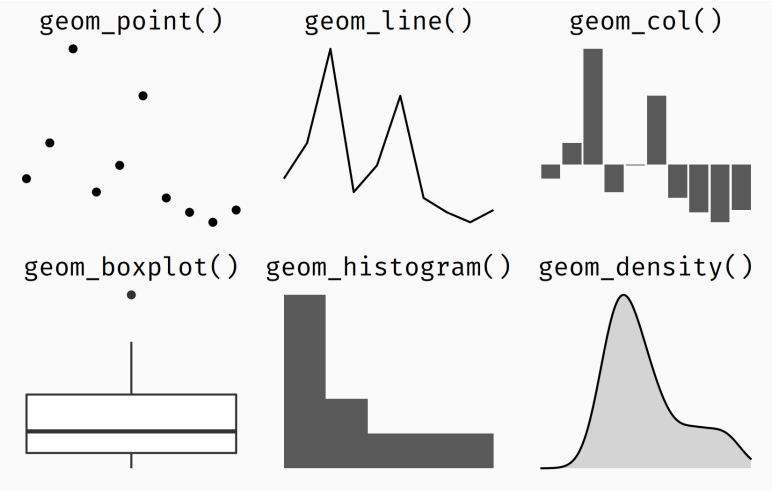


Abb. 2.86

Insgesamt sind 53 Geome verfügbar:

```
 [1] "geom_abline"            "geom_area"              "geom_bar"
 [4] "geom_bin_2d"            "geom_bin2d"             "geom_blank"
 [7] "geom_boxplot"           "geom_col"               "geom_contour"
[10] "geom_contour_filled"    "geom_count"             "geom_crossbar"
[13] "geom_curve"             "geom_density"           "geom_density_2d"
[16] "geom_density_2d_filled" "geom_density2d"         "geom_density2d_filled"
[19] "geom_dotplot"           "geom_errorbar"          "geom_errorbarh"
[22] "geom_freqpoly"          "geom_function"          "geom_hex"
[25] "geom_histogram"         "geom_hline"             "geom_jitter"
[28] "geom_label"             "geom_line"              "geom_linerange"
[31] "geom_map"               "geom_path"              "geom_point"
[34] "geom_pointrange"        "geom_polygon"           "geom_qq"
[37] "geom_qq_line"           "geom_quantile"          "geom_raster"
[40] "geom_rect"              "geom_ribbon"            "geom_rug"
[43] "geom_segment"           "geom_sf"                "geom_sf_label"
[46] "geom_sf_text"           "geom_smooth"            "geom_spoke"
[49] "geom_step"              "geom_text"              "geom_tile"
[52] "geom_violin"            "geom_vline"
```

In unserem Beispiel möchten wir das Gewicht sowie das Alter der Probanden als Punktwolke darstellen. Das geeignete Geom ist `geom_point()`.

```
# unser erstes Plot, mit "Aesthetics" und "Geom"
ggplot(pf8) +
  aes(x = Alter,
      y = Gewicht) +
  geom_point()
```

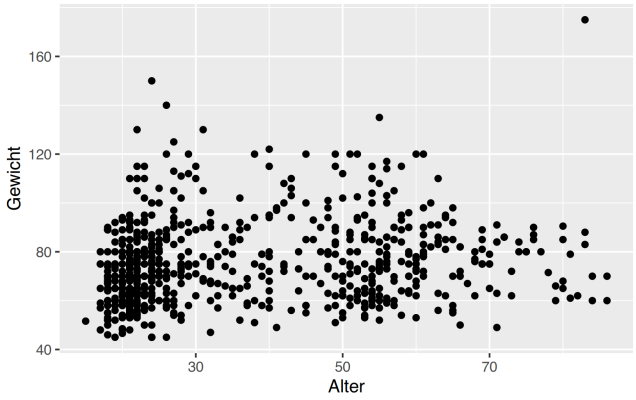


Abb. 2.87

2.4.2.3 mehr Aesthetics

Angenommen, wir möchten das `Gewicht` sowie das `Alter` der Probanden als Punktwolke darstellen, aber graphisch zwischen den Geschlechtern unterscheiden. Hierzu erweitern wir dies `aes()`-Funktion um den Parameter `color`.

```
1  # unser erstes Plot, mit "Aesthetics" und "Geom",
2  # farblich nach Geschlecht
3  ggplot(pf8) +
4    aes(x = Alter,
5        y = Gewicht,
6        color = Geschlecht) +
7    geom_point()
```

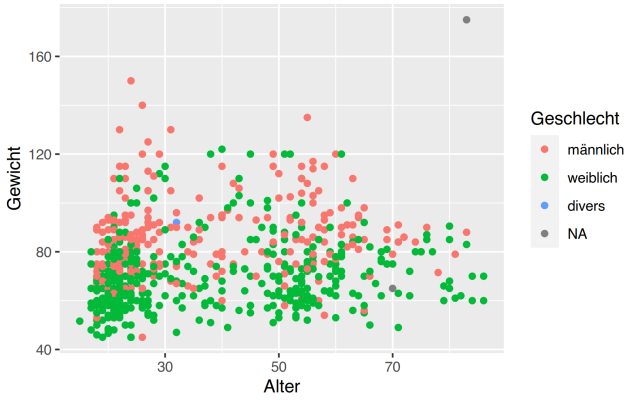


Abb. 2.88

Ggplot stellt die Werte in unterschiedlichen Farben pro Geschlechterkategorie dar. Ebenso wurde das Plot um eine Legendenbox erweitert, in welcher die Farbzuordnungen pro Geschlechtskategorie angezeigt ist.

2.4.2.4 mehr Geome

Über das + Zeichen können beliebig viele Geome dem Plot hinzugefügt werden. Dabei werden die *aesthetics* automatisch an die Geome weitergegeben, sie können aber auch in jedem Geom per `aes()` neu definiert werden.
In unserem Beispiel möchten wir nun eine Regressionsgerade mit Konfidenzintervall einzeichnen. Dies kann mit `geom_smooth()` umgesetzt werden. Über den Parameter `method` können wir angeben, dass ein lineares Modell ("`lm`") eingezeichnet werden soll.

```
# unser erstes Plot, mit "Aesthetics" und zwei "Geomen"
# und Regressionsmodell

ggplot(pf8) +
  aes(x = Alter,
      y = Gewicht,
      color = Geschlecht) +
  geom_point()  +
  geom_smooth(method="lm")
```

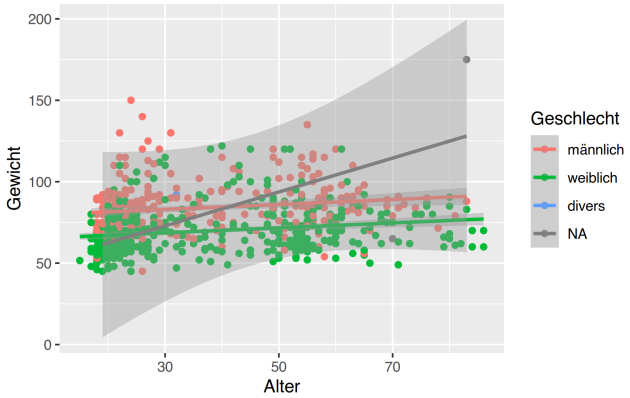


Abb. 2.89

In der Variablen `Geschlecht` sind `NA` enthalten. Diese können wir mit der Funktion `drop_na()` aussortieren und direkt an `ggplot()` weiter-pipen.

```
# unser erstes Plot, mit "Aesthetics" und zwei "Geomen"
# und Regressionsmodell und
# OHNE NA
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point()  +
    geom_smooth(method="lm")
```

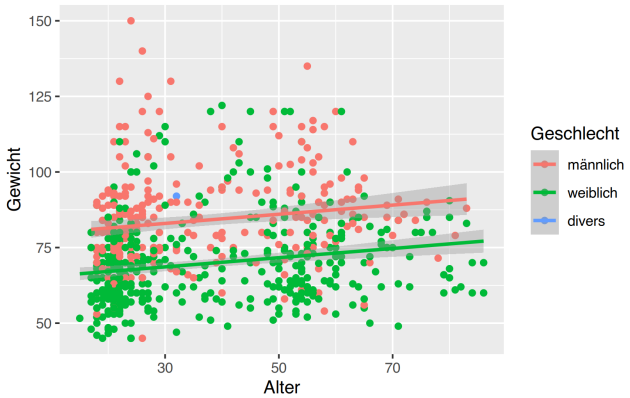


Abb. 2.90

Es wird ausdrücklich empfohlen, sich über die Hilfeseiten, z.B. `?geom_smooth`, mit den Funktionen vertraut zu machen. So erfährt man dort beispielsweise, dass der Konfidenzbereich per `se=FALSE` ausgeblendet werden kann.

```r
# unser erstes Plot, mit "Aesthetics" und zwei "Geomen"
# und Regressionsmodell ohne Konfidenzbereich
# OHNE NA
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point()  +
    geom_smooth(method="lm", se=FALSE)
```

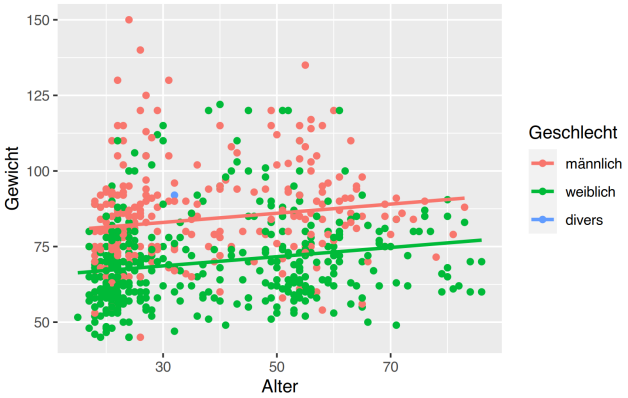


Abb. 2.91

2.4.2.5 Facetten

Mit den Funktionen `facet_wrap()` und `facet_grid()` können Teil-Diagramme innerhalb des Gesamtplots erzeugt werden.
So können wir beispielsweise ein Plot pro `Standort` erstellen.

```
# unser erstes Plot, mit "Aesthetics",
# "Geom" und Facetten

pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point() +
    geom_smooth(method="lm") +
    facet_wrap(~ Standort)
```

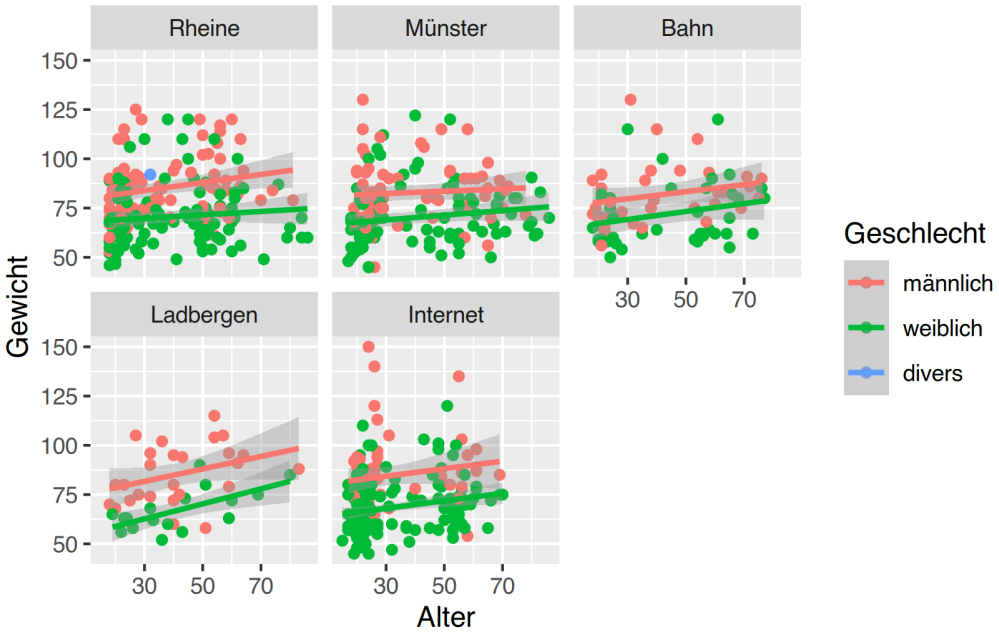


Abb. 2.92

Oder wir erstellen ein Plot pro `Rauchen`-Kategorie.

```
# unser erstes Plot, mit "Aesthetics",
# "Geom" und Facetten

pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point() +
    geom_smooth(method="lm") +
    facet_wrap(~ Rauchen)
```

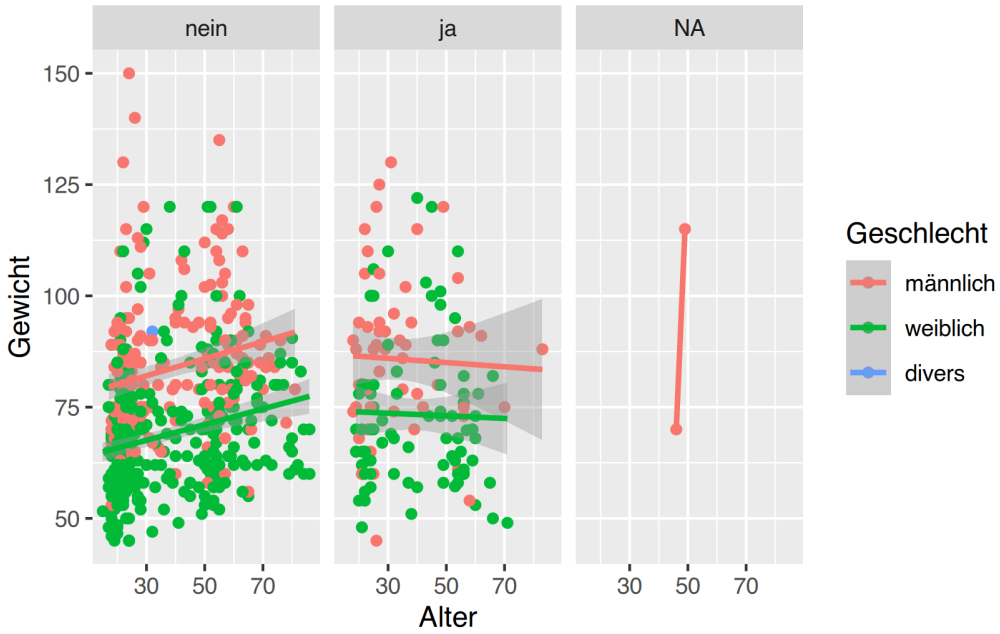


Abb. 2.93

Mit der Funktion `facet_grid()` können Plot-Gitter erzeugt werden. So können wir beispielsweise ein Plot pro Standort und Rauchen erstellen.

```
# unser erstes Plot, mit "Aesthetics",
# "Geom" und Facetten

pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point() +
    geom_smooth(method="lm") +
    facet_grid(Rauchen ~ Standort)
```

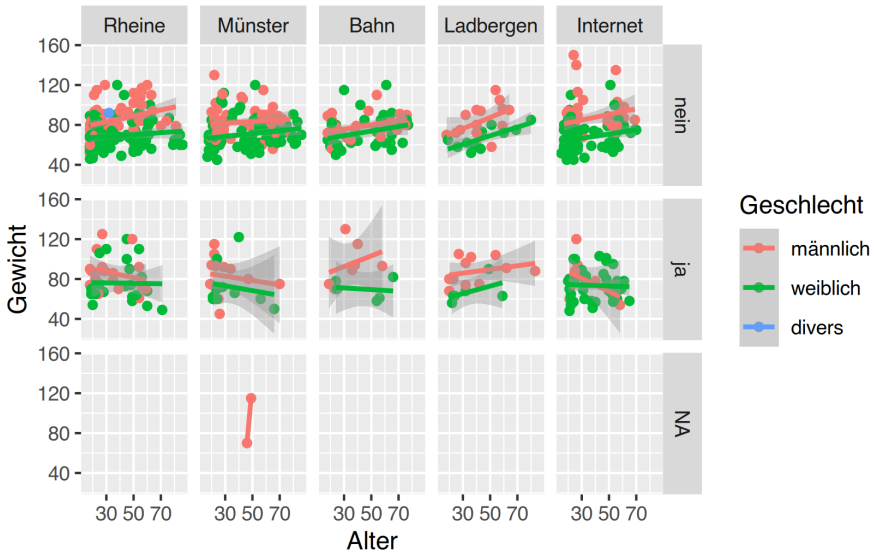


Abb. 2.94

Mittels `drop_na()` können wir uns der `NA`-Zeile entledigen.

```
# unser erstes Plot, mit "Aesthetics",
# "Geom" und Facetten

pf8 %>%
  drop_na(Geschlecht, Rauchen) %>%
  ggplot() +
    aes(x = Alter,
        y = Gewicht,
        color = Geschlecht) +
    geom_point() +
    geom_smooth(method="lm") +
    facet_grid(Rauchen ~ Standort)
```

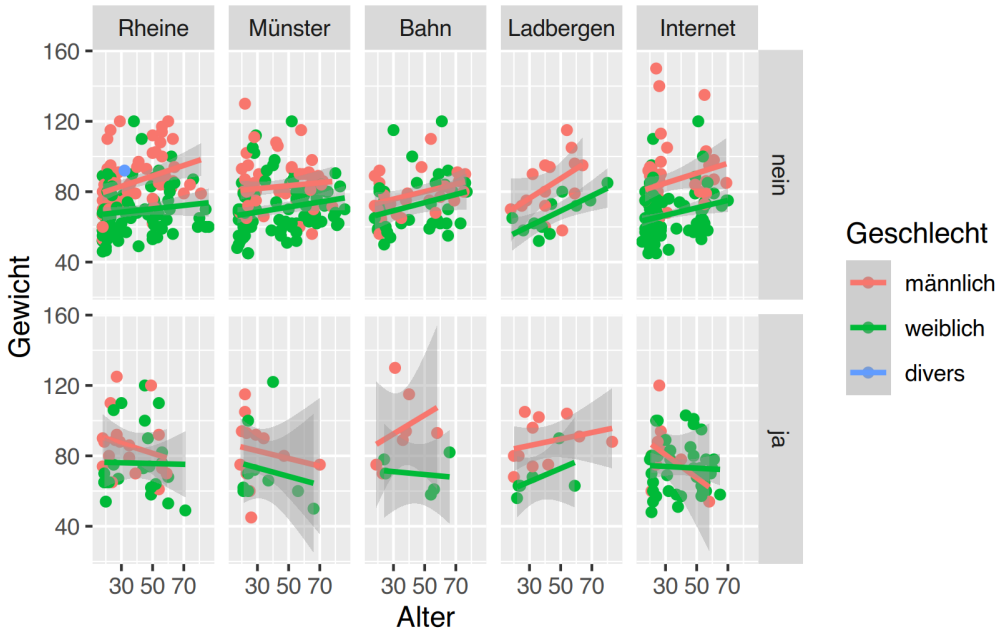


Abb. 2.95

2.4.2.6 Geom-Parameter

Geome können mit verschiedenen Parametern beeinflusst werden. Welche Parameter zur Verfügung stehen, verrät die Hilfeseite der Geome. Die meisten reagieren auf:

geom_*(data, aes, stat, position, PARAMETER)

- `data`: Geome übernehmen das Datenobjekt aus der `ggplot()`-Funktion. Sie können aber auch eigene Daten verarbeiten
- `aes`: Geome erben die Zuordnungen aus der `aes()`-Funktion. Sie können aber auch eigene aesthetics haben.
- `stat`: Geome können statistische Transformationen vornehmen, z.B. `stat='identity'`.
- `position`: die Positionierung der Objekte lässt sich anpassen, z.B. `'dodge'`, `'stack'`, `'jitter'`, `'fill'`.

- weitere PARAMETER sind:
  - `color`: (Rahmen)farbe
  - `fill`: Füllfarbe
  - `size`: Objektgröße
  - `shape`: Objektformen

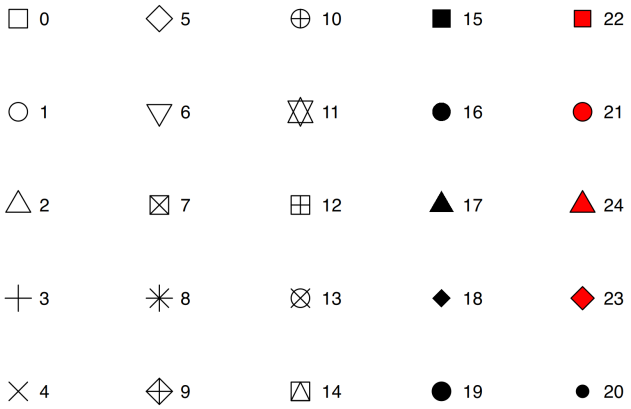


Abb. 2.96: `shape` nach Zahlen

Die Parameter können auch innerhalb der `aes()`-Funktion verwendet werden, so dass anhand der Daten entschieden wird, welche Größe, Farbe und Form die Geome haben.

  - `linetype`: Linientypen

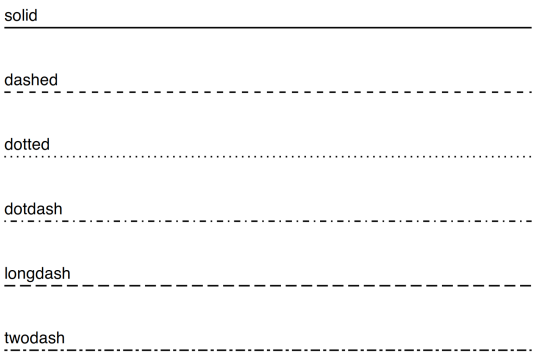


Abb. 2.97: Linientypen

2.4.3 Punktwolke

Punktwolken (Scatterplots) werden mit der Funktion `geom_point()` erzeugt. Wir plotten die Variablen `Größe` und `Gewicht` aus dem `pf8`-Datensatz.

```
ggplot(pf8) +
  aes(x=Größe, y=Gewicht) +
  geom_point()
```

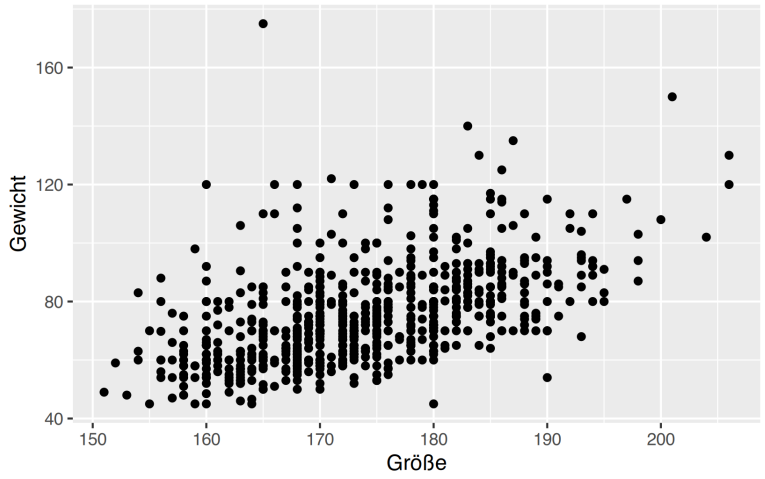


Abb. 2.98

Über die Parameter `color`, `shape` und `size` können wir die Punkte manipulieren.

```
ggplot(pf8) +
  aes(x=Größe, y=Gewicht) +
  geom_point(color="blue",
             shape=13,
             size=10)
```

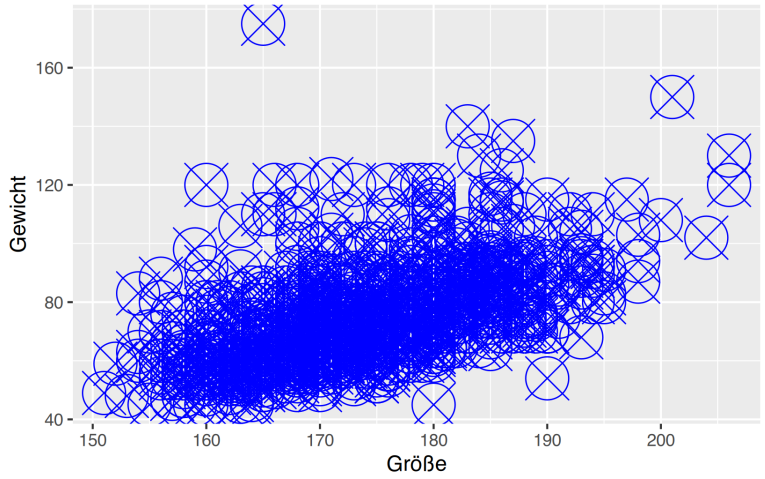


Abb. 2.99

Die Parameter können auch als *aesthetics* verwendet werden, so dass anhand der Daten entschieden wird, welche Größe, Farbe und Form die Punkte haben.

```
ggplot(pf8) +
  aes(x=Größe, y=Gewicht,
      color=Geschlecht,
      shape=Geschlecht,
      size=Alter) +
  geom_point()
```

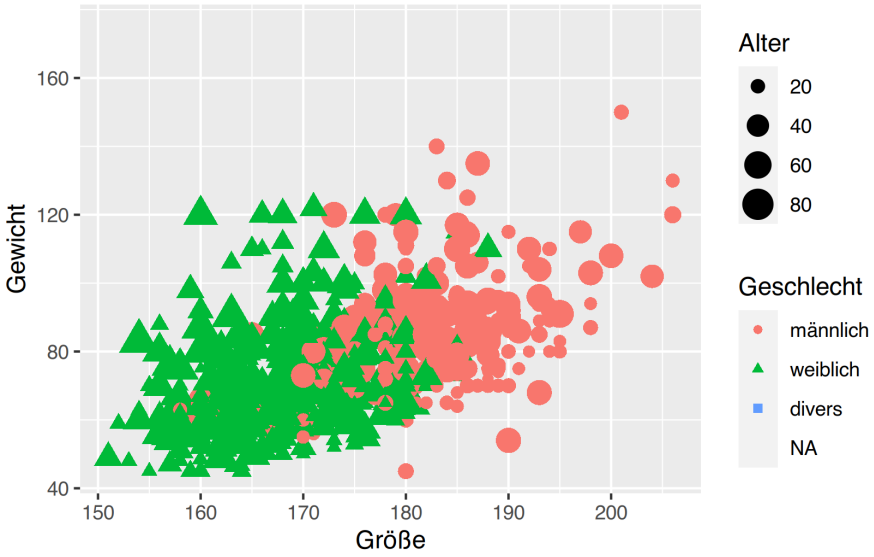


Abb. 2.100

2.4.4 Liniendiagramme

Liniendiagramme werden mit der Funktion `geom_line()` erzeugt.

```
# Erzeuge ein paar Daten
df <- data.frame(Jahr=c(2020:2023),
                 Wert=c(3.2, 11, 31.1, 78.3))

# Plotte als Liniendiagramm
ggplot(df, aes(x=Jahr, y=Wert)) +
  geom_line()
```

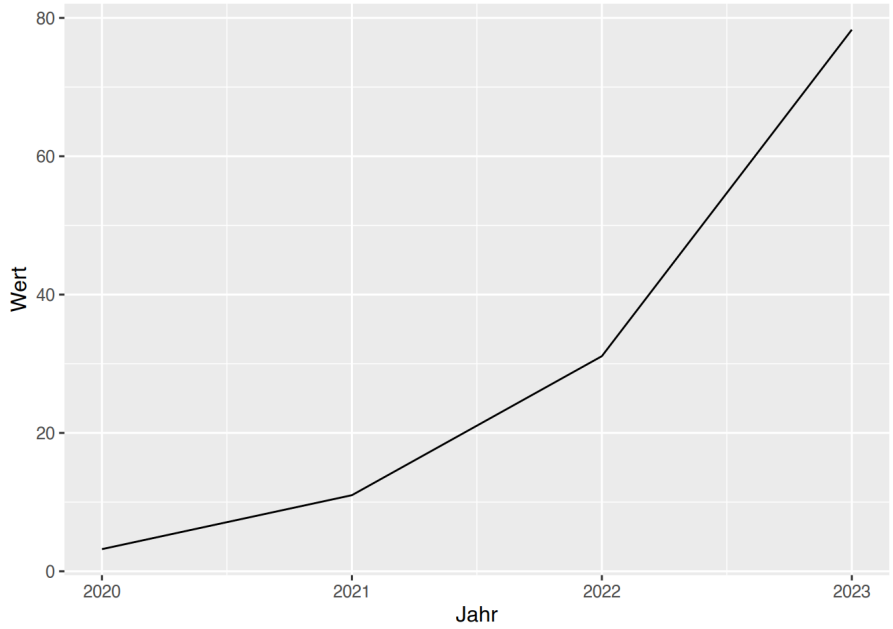


Abb. 2.101

Über den Parameter `linetype` kann der Linientyp festgelegt werden.

```
ggplot(df, aes(x=Jahr, y=Wert)) +
  geom_line(linetype = "dashed")
```

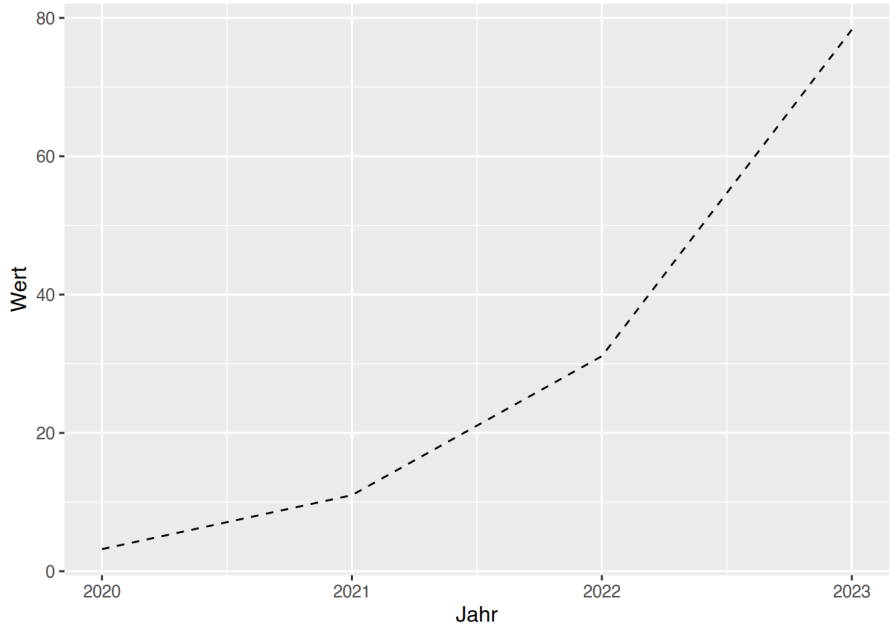


Abb. 2.102

Sollen zusätzlich die Datenpunkte eingezeichnet werden (vgl. `plot(type="b")`), kann einfach das `geom_point()` angehängt werden.

```
ggplot(df, aes(x=Jahr, y=Wert)) +
  geom_line(linetype = "dashed")+
  geom_point()
```

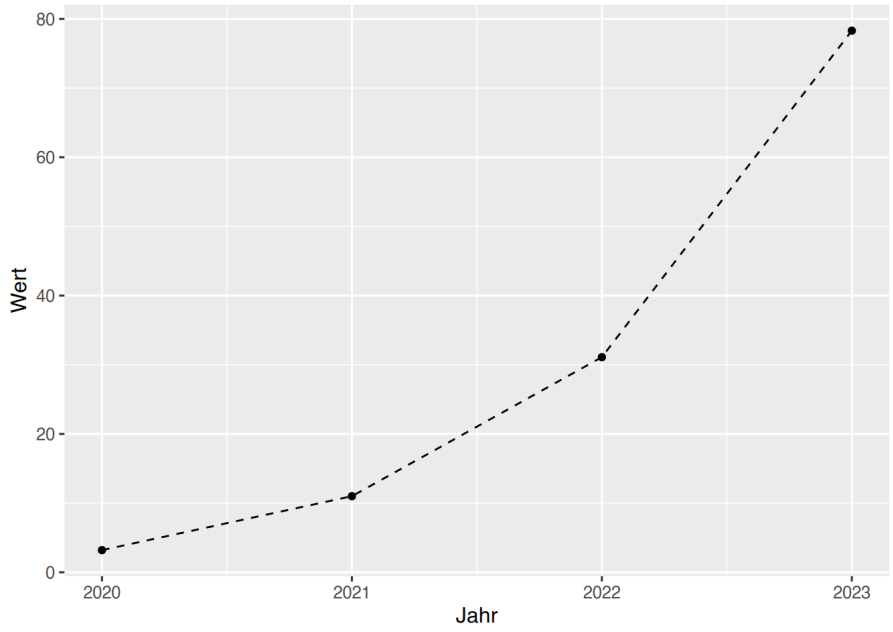


Abb. 2.103

Farben können entweder per `aes()` oder direkt per `col=` gesetzt werden.

```
ggplot(df, aes(x=Jahr, y=Wert)) +
  geom_line(color="red")+
  geom_point(color="darkblue")
```

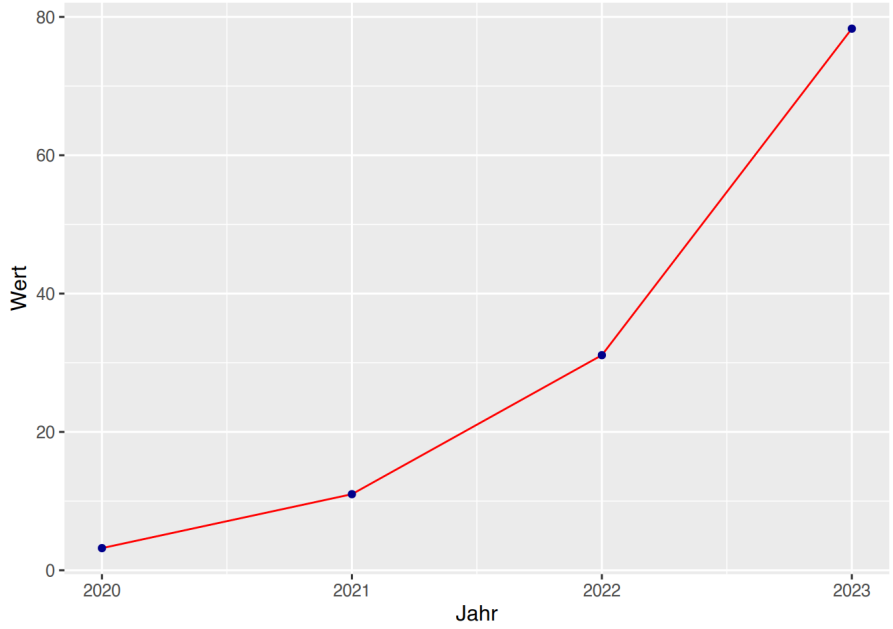


Abb. 2.104

Liegen gruppierte Daten vor, kann die Farbe der Gruppen per `aes()` übergeben werden.

```
# erzeuge gruppierte Daten
df <- data.frame(Jahr=rep(c(2020:2023), 2),
                 Wert=c(3.2, 11, 31.1, 78.3,
                        4.1, 21, 53.5, 120),
                 Person=c(rep("Erika", 4), rep("Hans", 4)))

# Plotte mit unterschiedlichen Farben
ggplot(df, aes(x=Jahr, y=Wert, color=Person)) +
  geom_line()+
  geom_point()
```

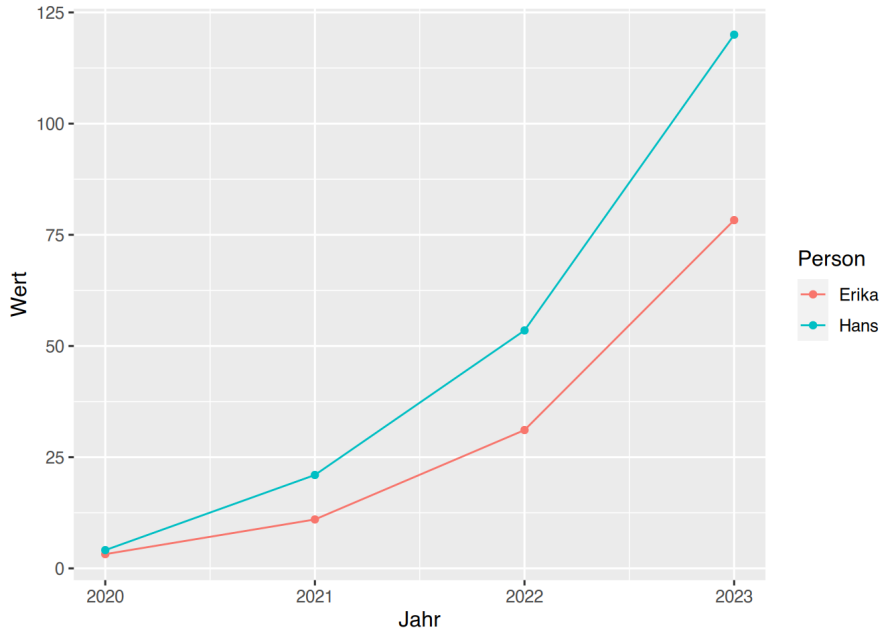


Abb. 2.105

Dies funktioniert nicht nur mit Farben, sondern auch mit allen anderen Parametern.

```
ggplot(df, aes(x=Jahr, y=Wert, color=Person,
               linetype=Person)) +
  geom_line()+
  geom_point()
```

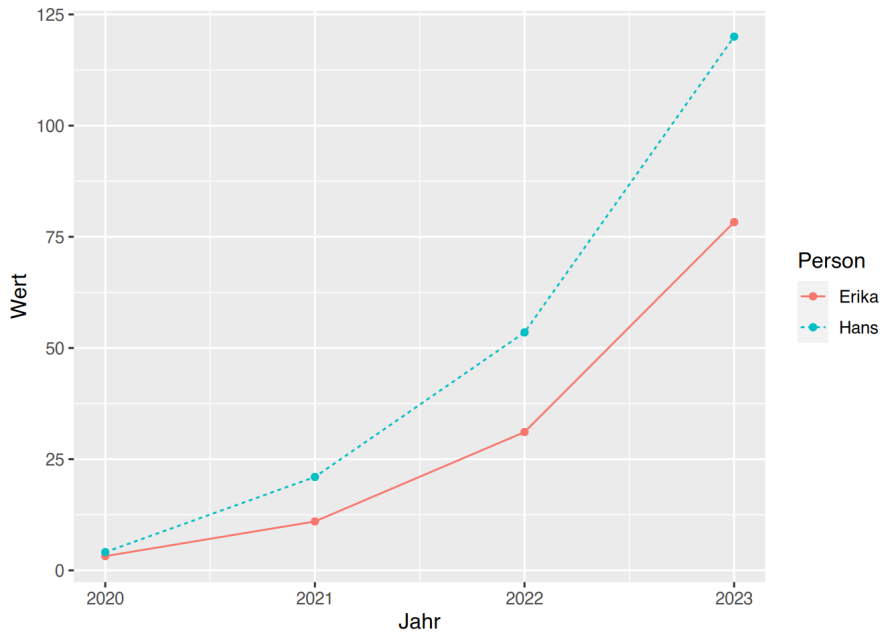


Abb. 2.106

Innerhalb der Geome können die Parameter überschrieben werden, z.B. die Geom-Farbe.

```
ggplot(df, aes(x=Jahr, y=Wert, color=Person,
               linetype=Person)) +
  geom_line()+
  geom_point()
```

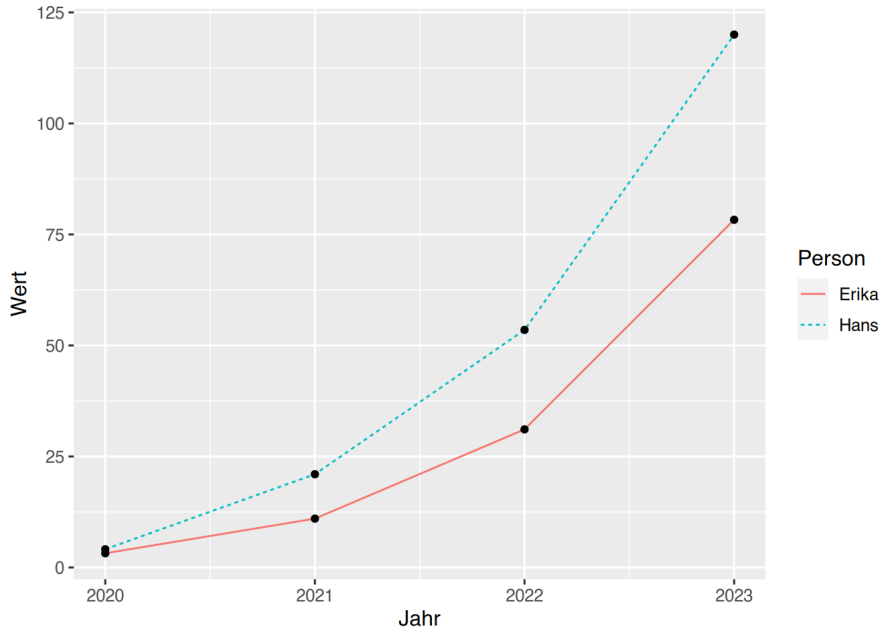


Abb. 2.107

2.4.5 Histogramm

Histogramme werden mit der Funktion `geom_histogram()` erzeugt.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram()
```

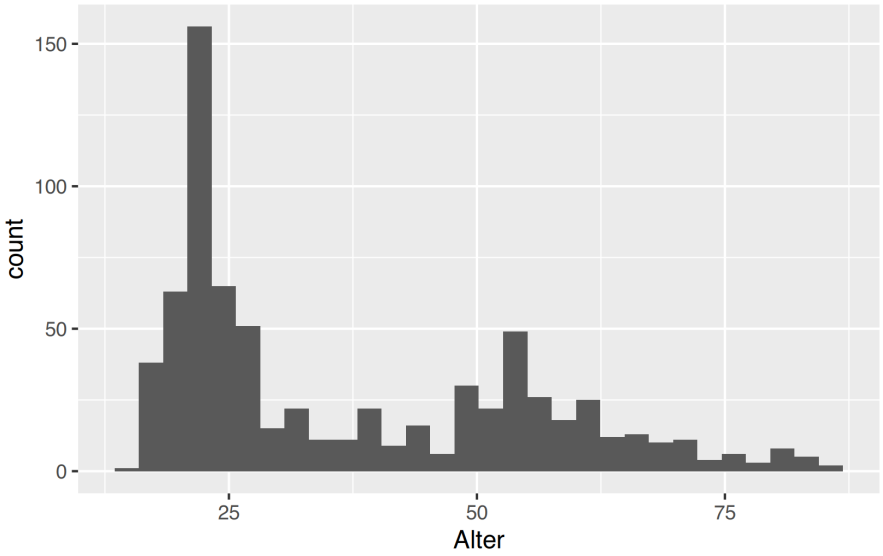


Abb. 2.108

Mit den Parametern `color` und `fill` können wir das Histogramm etwas aufhübschen.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",    # Rahmenfarbe
                 fill="skyblue") # Füllfarbe
```

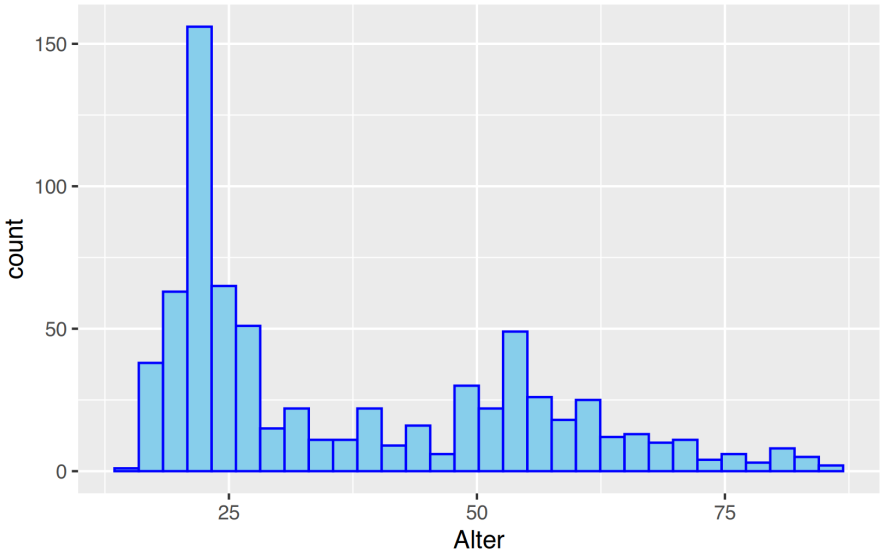


Abb. 2.109

Über den Parameter `binwidth` kann die Säulenbreite bestimmt werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",    # Rahmenfarbe
                 fill="skyblue", # Füllfarbe
                 binwidth=10)    # Klassenbreite 10
```

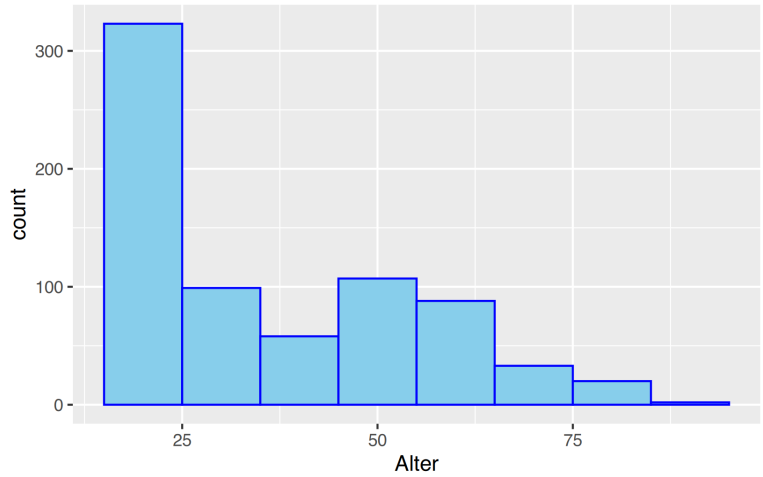


Abb. 2.110

Mit dem Parameter `bins` kann die Anzahl der Säulen bestimmt werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",    # Rahmenfarbe
                 fill="skyblue", # Füllfarbe
                 bins=5)         # 5 Säulen
```

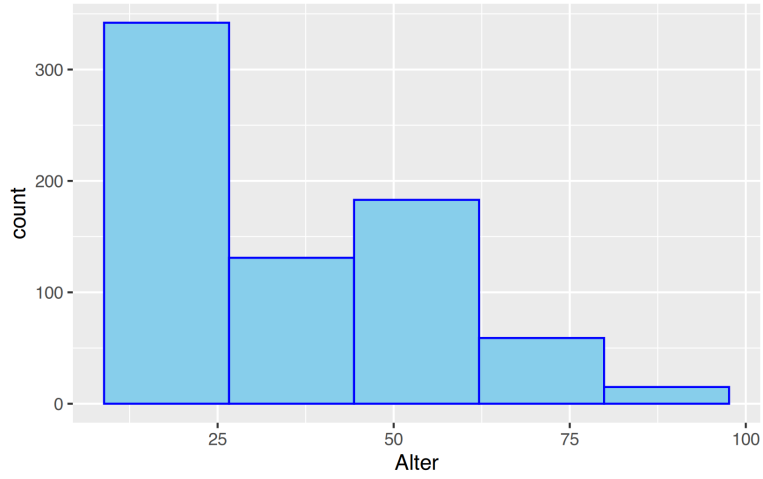


Abb. 2.111

Mit der Funktion `stat_bin()` kann das "Originalhistogramm" hinzugefügt werden. So ist leicht zu erkennen, wie grob die Klassen zusammengefasst wurden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",    # Rahmenfarbe
                 fill="skyblue", # Füllfarbe
                 bins=5) +       # 5 Säulen
  stat_bin(fill="darkblue")
```

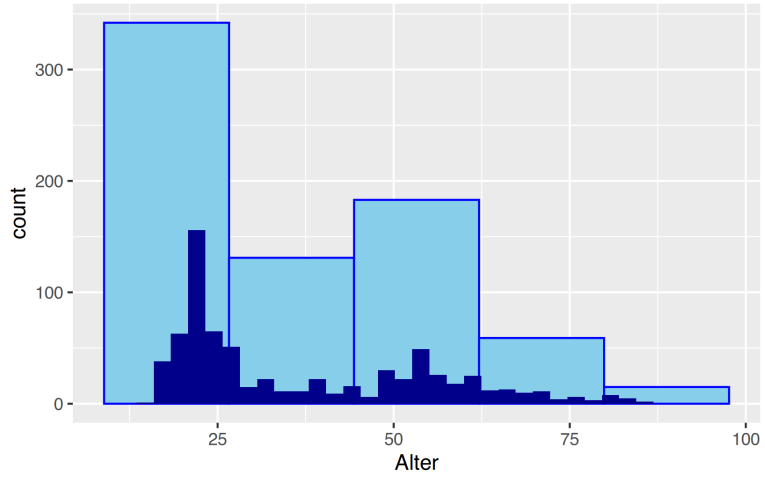


Abb. 2.112

2.4.5.1 Histogramm und Dichteverteilung

Über die Aesthetic `..density..` können wir auf die Skala der Dichtefunktion wechseln.

```
ggplot(pf8) +
  aes(x=Alter,
      y=..density..) +
  geom_histogram(color="blue",
                 fill="skyblue")
```

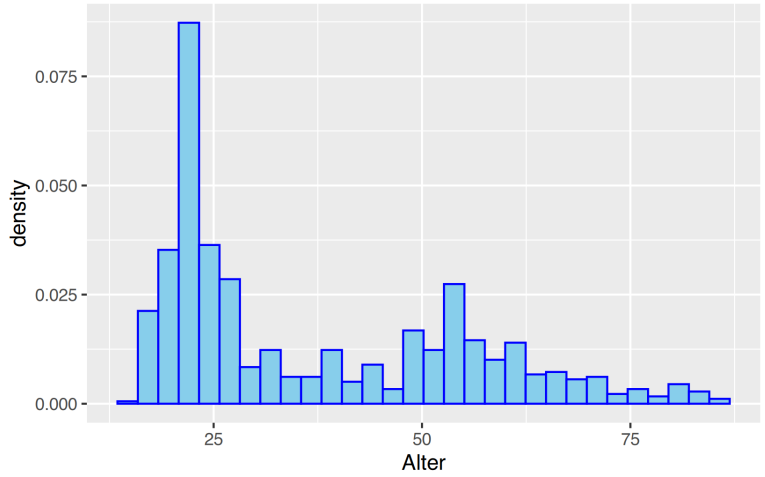


Abb. 2.113

Beachten Sie die Achseneinträge der y-Achse.
Das funktioniert auch, wenn die aesthetics im `geom_histogram()` angegeben werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(aes(y=..density..),
                 color="blue",
                 fill="skyblue")
```

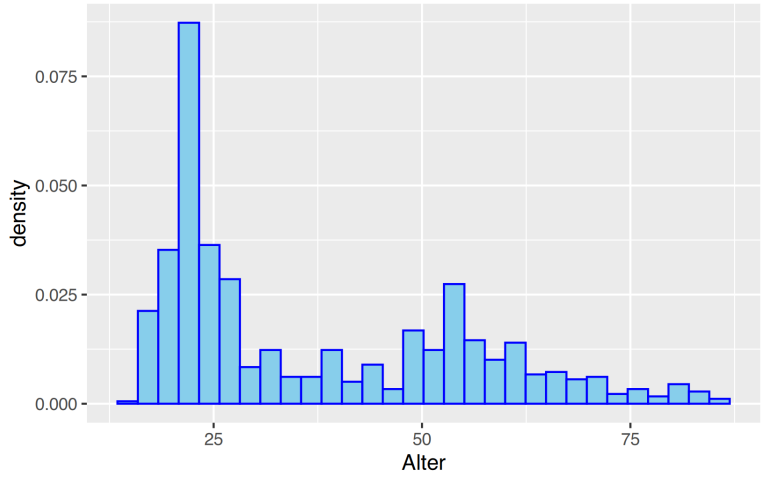


Abb. 2.114

Die Dichtefunktion des Histogramms lässt sich nun mit der Funktion `stat_density()` einzeichnen.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(aes(y=..density..),
                 color="blue",
                 fill="skyblue") +
  stat_density(geom="line",
               color="red",
               linetype="dotted")
```

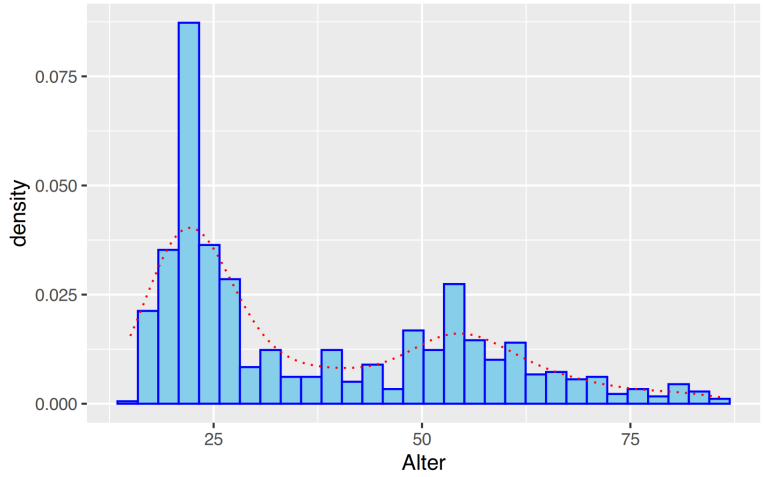


Abb. 2.115

Wir wechseln von `geom="line"` auf `geom="area"`:

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(aes(y=..density..),
                 color="blue",
                 fill="skyblue") +
  stat_density(geom="area",
               color="red",
               fill="pink",
               linetype="dotted")
```

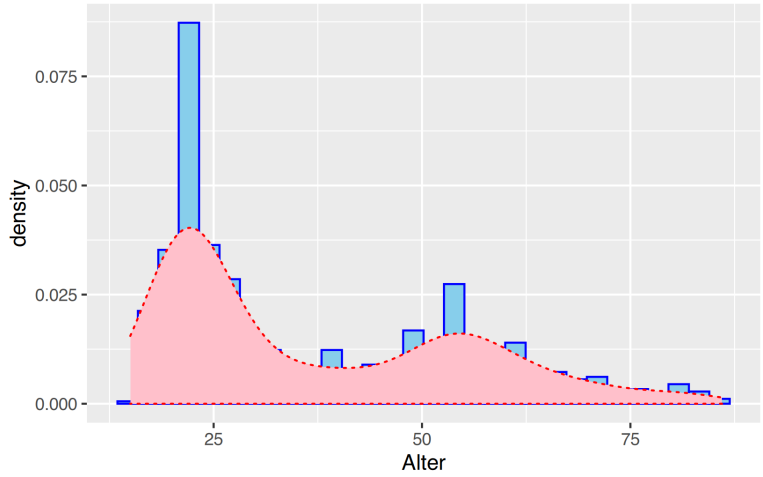


Abb. 2.116

Über den Paramter `alpha` lässt sich die Durchsichtigkeit (so genannter Alphakanal) des Objekts eingeben.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(aes(y=..density..),
                 color="blue",
                 fill="skyblue") +
  stat_density(geom="area",
               color="red",
               fill="pink",
               linetype="dotted",
               alpha=0.5)
```

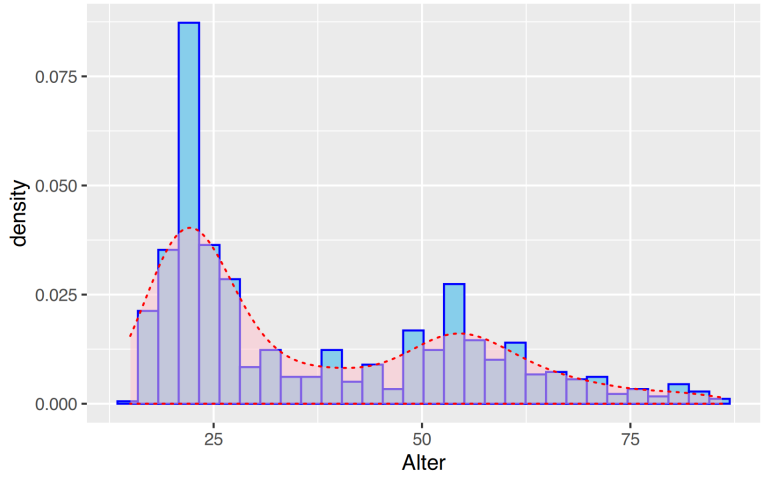


Abb. 2.117

Mittels `stat_function()` und `dnorm()` lässt sich die Normalverteilung ergänzen. Hierfür bestimmen wir zunächst Mittelwert und Standardabweichung von `pf8$Alter`.

```
round(mean(pf8$Alter, na.rm=TRUE))
round(sd(pf8$Alter, na.rm=TRUE))
```

Die Werte werden nun wie folgt übertragen:

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(aes(y=..density..),
                 color="blue",
                 fill="skyblue") +
  stat_density(geom="area",
               color="red",
               fill="pink",
               linetype="dotted",
               alpha=0.5)+
  stat_function(fun=dnorm,
                args=(c(mean=37,sd=18)),
                color = "darkgreen",
                linetype = "dashed")
```

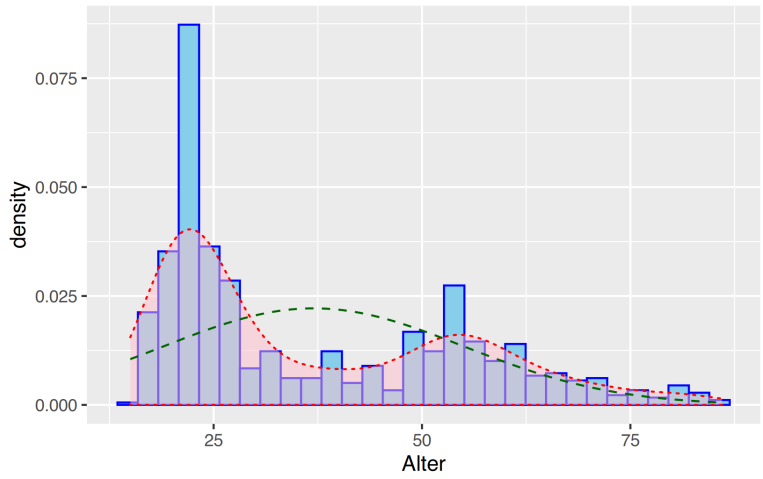


Abb. 2.118

Es ist deutlich erkennbar, dass **keine** Normalverteilung vorliegt.

2.4.5.2 kumulierte Häufigkeiten

Für kumulierte Häufigkeiten nutzen wir innerhalb von aes() die Funktionen `after_stat(count)`, um die Häufigkeiten entsprechen mittels `cumsum()` und `sum()` zu manipulieren.

```
  # kumulierte absolute Häufigkeiten
  ggplot(pf8) +
    aes(x=Alter) + ylab("absolute Häufigkeiten") +
    geom_histogram(aes(y=cumsum(after_stat(count))),
                   bins=5, fill="hotpink3", color="black") +
    # füge Polygonzug hinzu
    stat_bin(aes(y=cumsum(after_stat(count))),
                   bins=5,
                   geom="line", color="blue", linewidth=1.5) # oder geom="step"
```

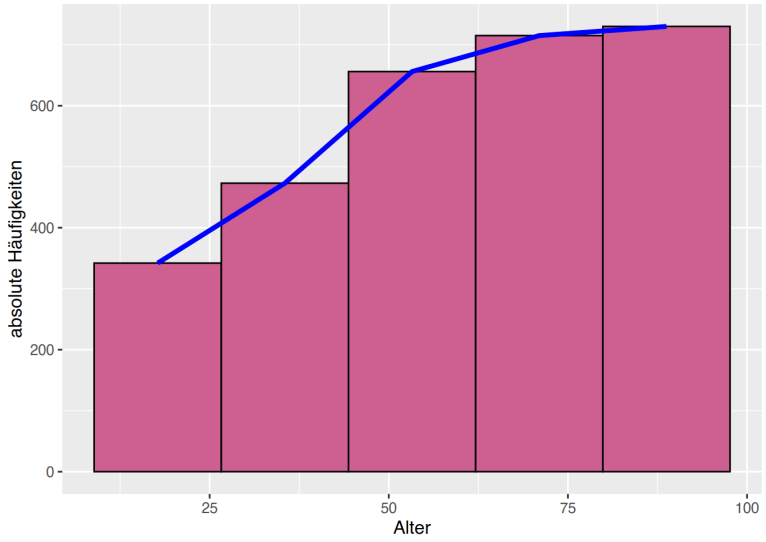


Abb. 2.119

```
# kumulierte relative Häufigkeiten
ggplot(pf8) +
  aes(x=Alter) +
  ylab("relative Häufigkeiten")+
  geom_histogram(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))),
                 bins=5, fill="hotpink3", color="black") +
  # füge Polygonzug hinzu
  stat_bin(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))),
                 bins=5,
                 geom="step", color="blue", linewidth=1.5) # oder geom="line"
```

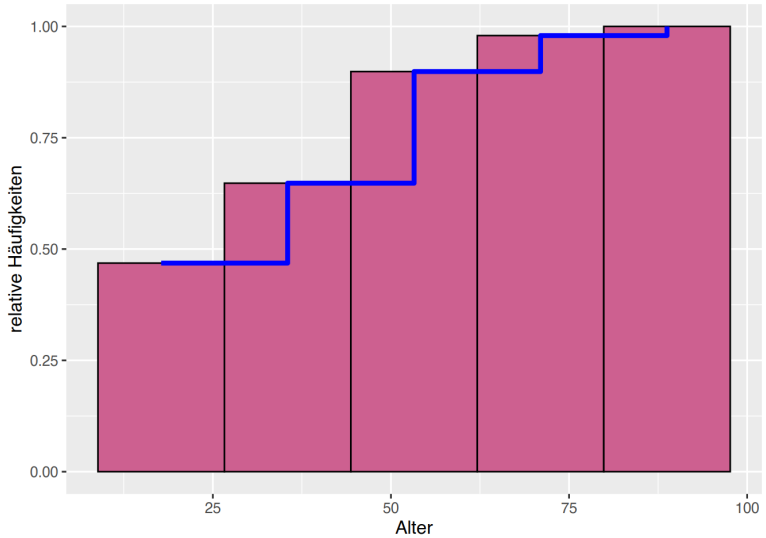


Abb. 2.120

2.4.6 Säulendiagramme

Säulendiagramme werden mit den Funktionen `geom_bar()` und `geom_col()` erzeugt.

- Für `geom_bar()` müssen die Variablen als Factor vorliegen. Die Größe der Säulen wird automatisch mittels `stat_count()` berechnet.
- `geom_col()` erstellt die Größe der Säulen anhand der tatsächlichen Werte im Datenframe (vgl. `stat_identity()`; absolute Häufigkeit)

Die Variable `pf8$SportWarum` ist ein Faktor und soll mittels `geom_bar()` geplottet werden.

```
ggplot(pf8) +
  aes(x=SportWarum ) +
  geom_bar(color="blue",
           fill="skyblue")
```

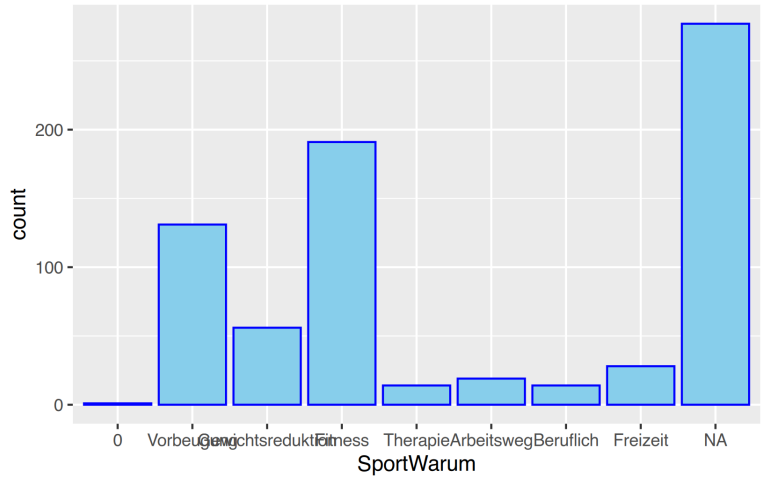


Abb. 2.121

Mittels `drop_na()` kann der große `NA`-Balken entfernt werden.

```
pf8 %>%
  drop_na(SportWarum) %>%
    ggplot() +
      aes(x=SportWarum ) +
      geom_bar(color="blue",
               fill="skyblue")
```

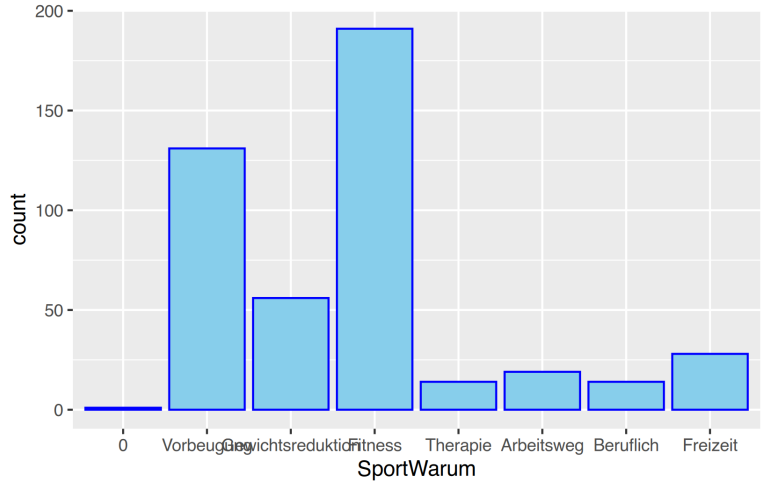


Abb. 2.122

Mit Hilfe von der Funktionen `fct_infreq()` und `fct_rev()` können wir die Levels auf- und absteigend sortieren. Hierzu erzeugen wir eine neue Spalte `S`, welche wir in der `aes()`-Funktion auf die x-Achse legen.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  ggplot() +
    aes(x=S ) +
    geom_bar(color="blue",
             fill="skyblue")
```

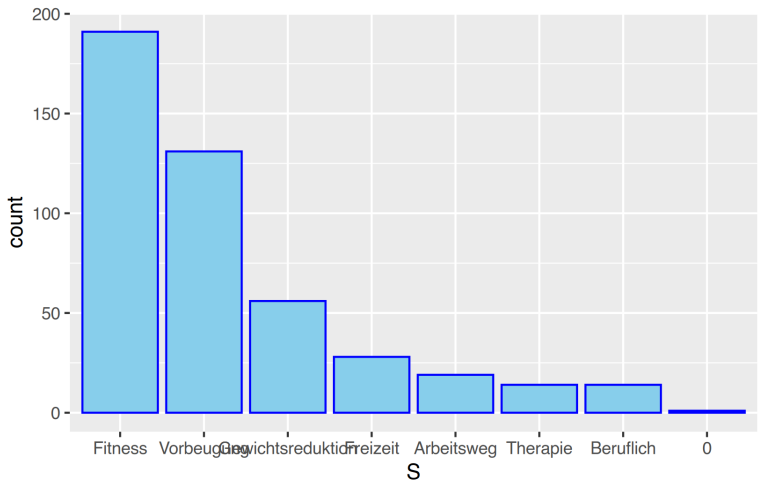


Abb. 2.123

Um aufsteigend zu sortieren erweitert sich der `mutate()`-Aufruf wie folgt:

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum),
         S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S) +
    geom_bar(color="blue",
             fill="skyblue")
```

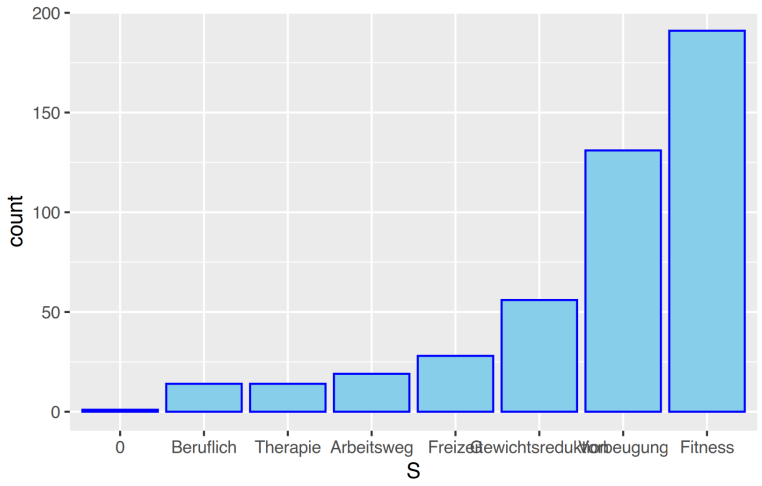


Abb. 2.124

Über die Aesthetic `fill` können wir gruppieren, z.B. nach `Geschlecht`.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue")
```

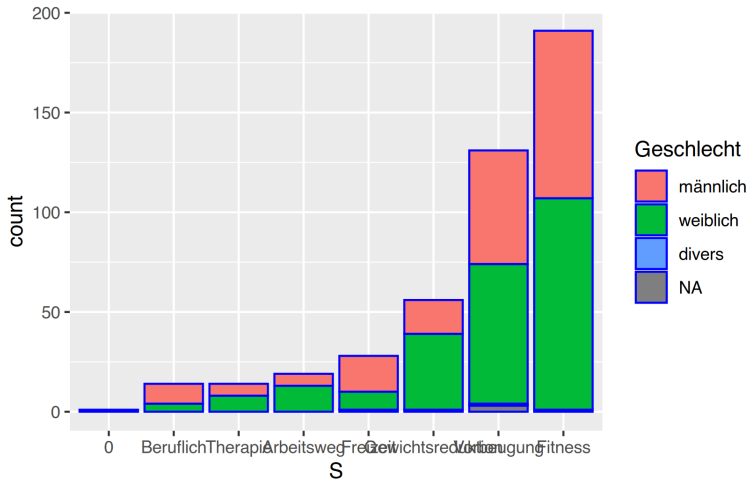


Abb. 2.125

Mit dem Parameter `position` können wir die Säulenausgabe weiter beeinflussen. Die Angabe `position="fill"` erzeugt Prozentanteile.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue",
             position="fill")
```

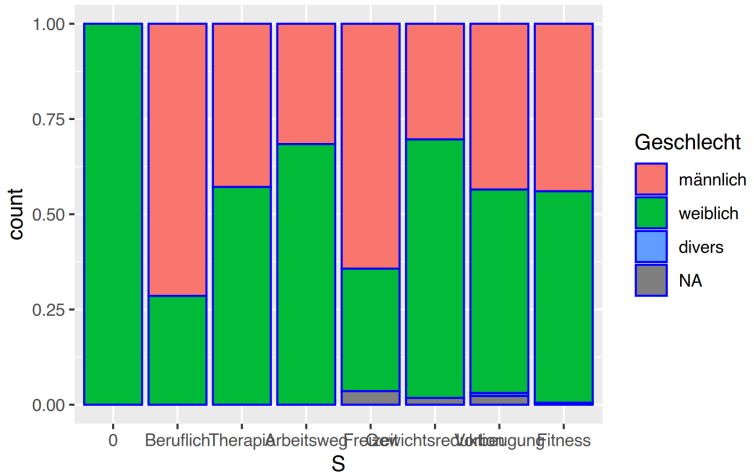


Abb. 2.126

Mit `position="dodge"` werden die Säulen der Gruppen nebeneinander gestellt.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue",
             position="dodge")
```

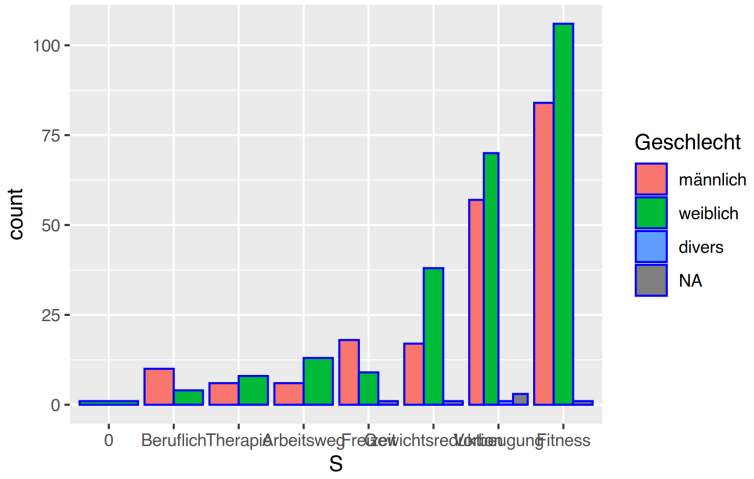


Abb. 2.127

Mit `position="jitter"` wird etwas Rauschen bei der Positionierung verwendet. Das sieht nicht immer schön aus.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue",
             position="jitter")
```

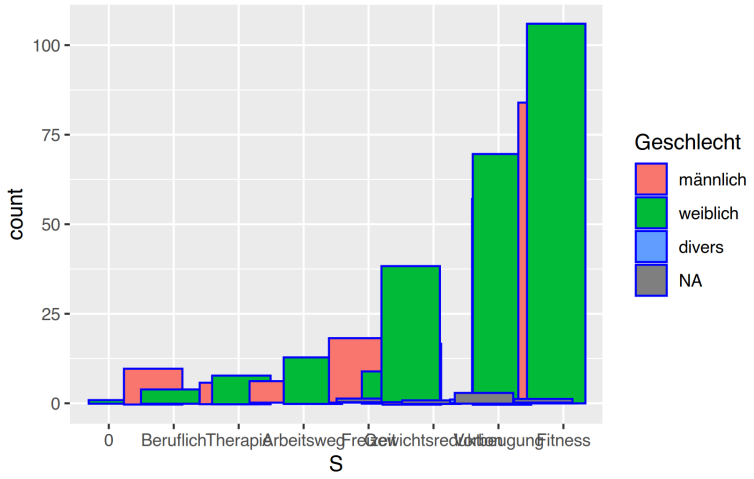


Abb. 2.128

Die Default-Einstellung ist `position="stack"`.

```
pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue",
             position="stack")
```

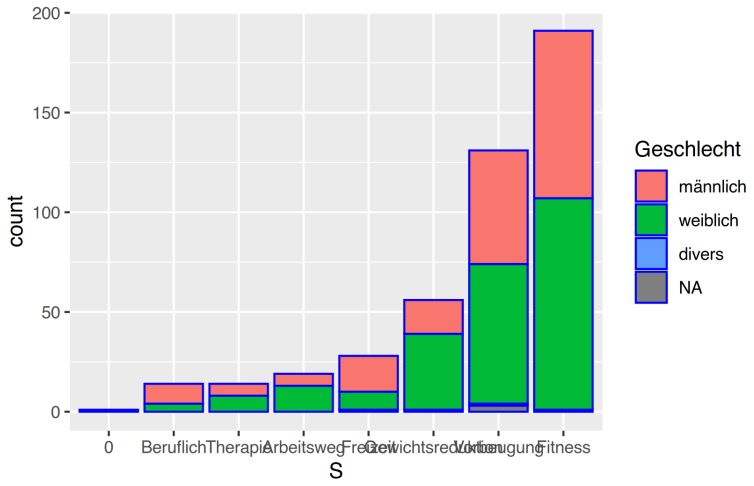


Abb. 2.129

Mit der Funktion `stat_count()` können die jeweiligen Werte der Balken ausgeschrieben werden.

```
  pf8 %>%
  drop_na(SportWarum) %>%
  mutate(S = fct_infreq(SportWarum)) %>%
  mutate(S = fct_rev(S)) %>%
  ggplot() +
    aes(x=S,
        fill=Geschlecht) +
    geom_bar(color="blue",
             position="stack")  +
    stat_count(geom = "text",
               aes(label = after_stat(count)),
               position=position_stack(),
               vjust=-0.5)
```

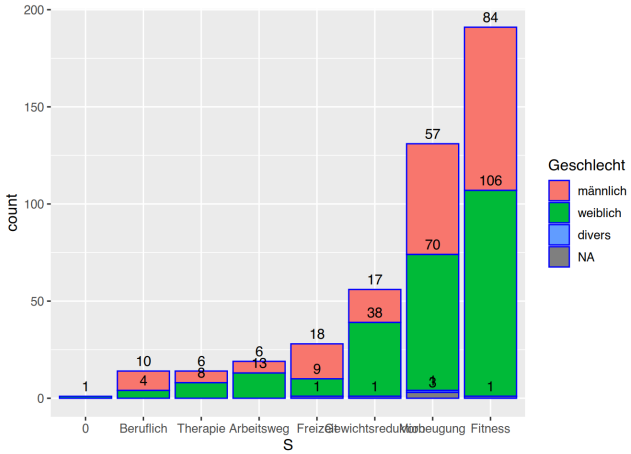


Abb. 2.130

Mit dem Parameter `vjust` kann die Höhe der Beschriftungen verändert werden.
Beachten Sie, dass die Parameter `position` aus `geom_bar()` und `stat_count()` übereinstimmen müssen. Da in `geom_bar(position=ßtack")` gesetzt ist, muss innerhalb von `stat_count()` die Funktion `position_stack()` übergeben werden.
Ändern wir in `geom_bar()` den Parameter auf `position="dodge"`, so muss innerhalb von `stat_count()` die Funktion `position_dodge()` übergeben werden.

```
pf8 %>%
drop_na(SportWarum) %>%
mutate(S = fct_infreq(SportWarum)) %>%
mutate(S = fct_rev(S)) %>%
ggplot() +
  aes(x=S,
      fill=Geschlecht) +
  geom_bar(color="blue",
           position="dodge")  +
  stat_count(geom = "text",
             aes(label = after_stat(count)),
             position=position_dodge(width=1),
             vjust=-0.5)
```

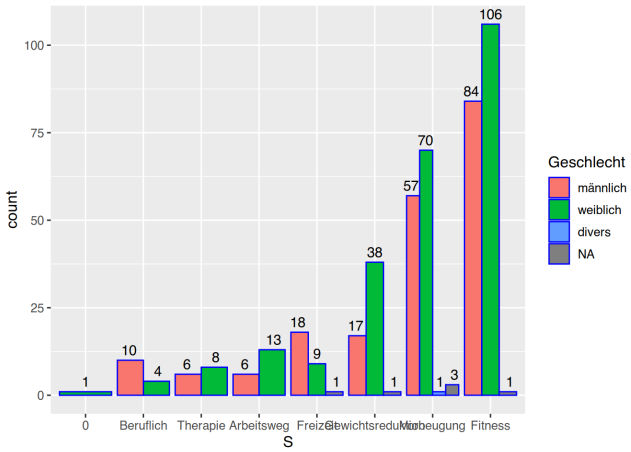


Abb. 2.131

```
pf8 %>%
drop_na(SportWarum) %>%
mutate(S = fct_infreq(SportWarum)) %>%
mutate(S = fct_rev(S)) %>%
ggplot() +
  aes(x=S,
      fill=Geschlecht) +
  geom_bar(color="blue",
           position="fill")  +
  stat_count(geom = "text",
             aes(label = after_stat(count)),
             position=position_fill(vjust=0.5),
             color="white")
```

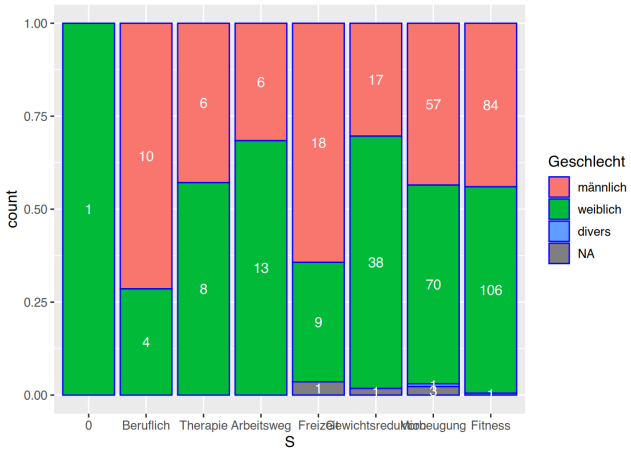


Abb. 2.132

2.4.6.1 vorgegebene Werte

Mit dem Parameter `stat="identity"` können vorberechnete Werte (absolute Häufigkeiten) geplottet werden.

```
# vorgegebene Werte
df <- data.frame(Gruppe=c("Vanille","Schokolade","Erdbeer"),
                 Anteil=c(60, 25, 15))

# plotten
ggplot(df, aes(x=Gruppe, y=Anteil,
               fill=Gruppe)) +
    geom_bar(stat="identity",
             position="dodge")
```

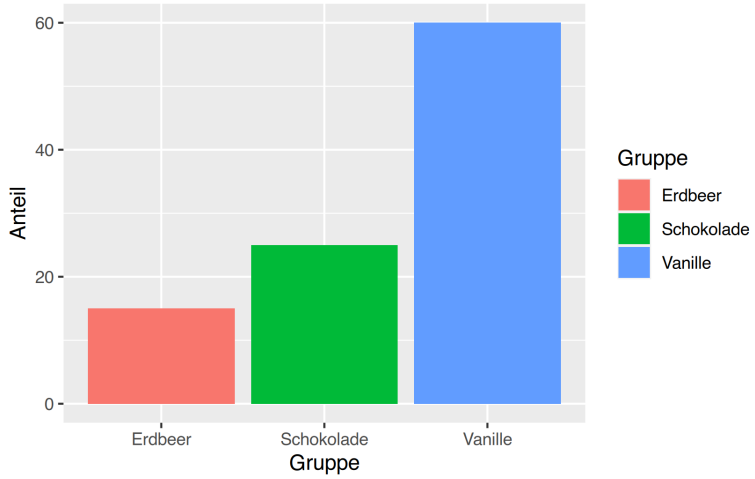


Abb. 2.133

Alternativ kann das Plot mit der Funktion `geom_col()` erzeugt werden, die von Hause aus `stat_identity()` nutzt.

```
# vorgegebene Werte
df <- data.frame(Gruppe=c("Vanille","Schokolade","Erdbeer"),
                 Anteil=c(60, 25, 15))

# plotten mit geom_col()
```

```
ggplot(df, aes(x=Gruppe, y=Anteil,
               fill=Gruppe)) +
    geom_col(position="dodge")
```

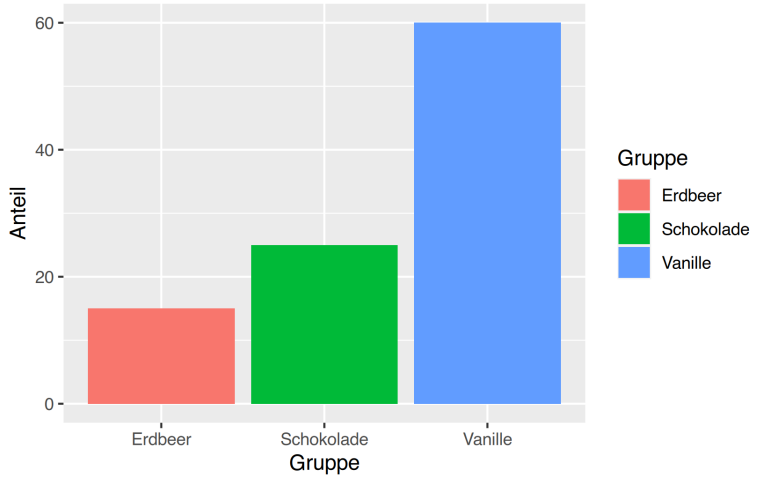


Abb. 2.134

Mit der Funktion `geom_text()` können die Werte den Plot-Balken hinzugefügt werden. Über den Parameter `vjust` kann bestimmt werden, ob die Werte über oder in den Säulen stehen sollen.

```
ggplot(df, aes(x=Gruppe, y=Anteil,
               fill=Gruppe)) +
  geom_col(position="dodge")+
  # Werte stehen über den Balken
  geom_text(aes(label = Anteil), vjust = -0.2)
```

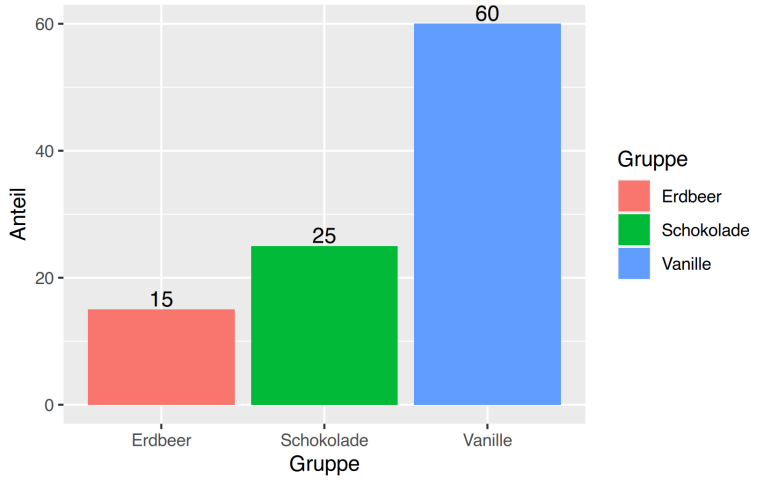


Abb. 2.135: Werte stehen über den Balken

```
ggplot(df, aes(x=Gruppe, y=Anteil,
               fill=Gruppe)) +
  geom_col(position="dodge")+
  # Werte stehen über den Balken
  geom_text(aes(label = Anteil), vjust = 2)
```

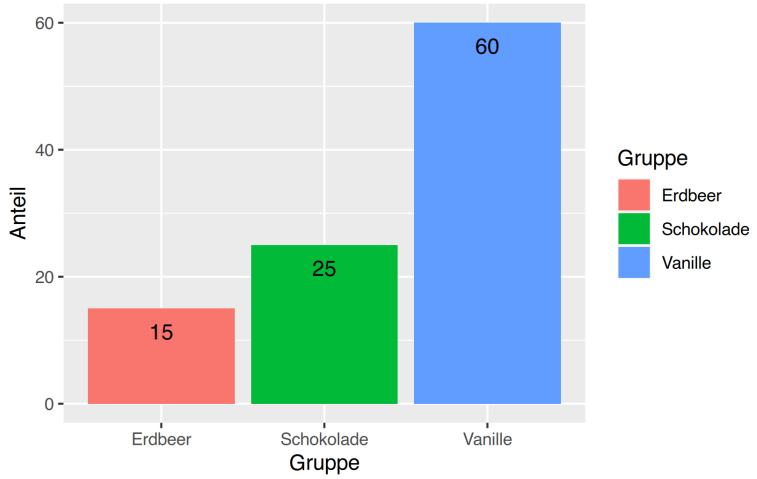


Abb. 2.136: Werte stehen in den Balken

Das selbe Ergebis kann auch mit der Funktion `stat_identity()` erzielt werden. Achten Sie auch hier darauf, dass die Parameter `position` von `geom_col()` (bzw. `geom_bar()`) und von `stat_identity()` gleich sein müssen.

```
ggplot(df, aes(x=Gruppe, y=Anteil,
               fill=Gruppe)) +
  geom_col(position="dodge") +
  stat_identity(geom = "text",
                aes(label=Anteil),
                position = position_dodge(width=1),
                vjust=2, size=5, color="white")
```

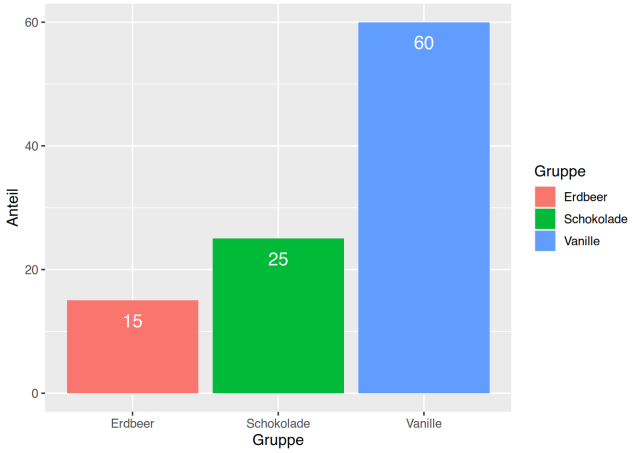


Abb. 2.137

2.4.7 Balkendiagramme

Balkendiagramme können auf zwei Arten erstellt werden. In der `aes()`-Funktion tauschen wir einfach `x` durch `y` aus:

```
ggplot(pf8) +
  aes(y=SportWarum ) +
  geom_bar(color="blue",
           fill="skyblue")
```

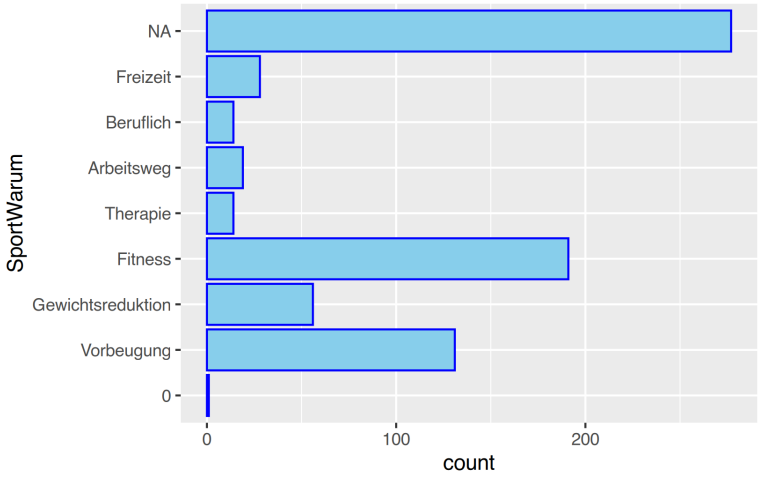


Abb. 2.138

Alternativ kann dem Plotaufruf die Funktion `coord_flip()` angehängt werden.

```
ggplot(pf8) +
  aes(x=SportWarum ) +
  geom_bar(color="blue",
           fill="skyblue") +
  coord_flip()
```

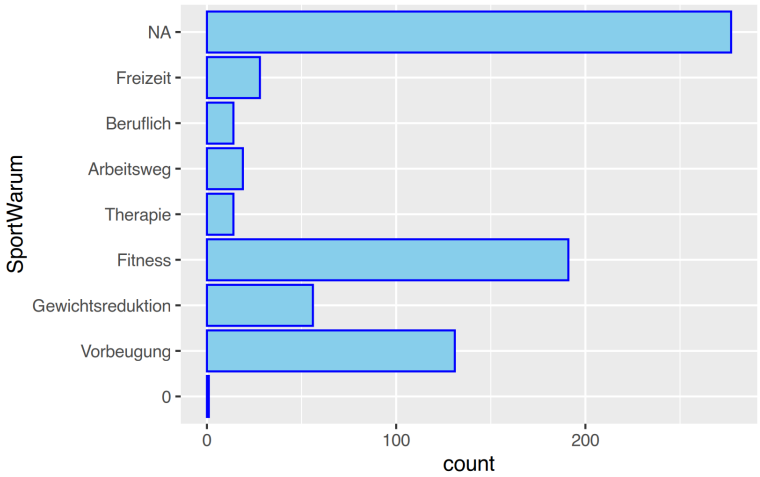


Abb. 2.139

2.4.8 Boxplots

Boxplots werden mit der Funktion `geom_boxplot()` erzeugt.

```
pf8 %>%
  ggplot() +
    aes(x = Alter) +
    geom_boxplot(fill="seagreen3")
```

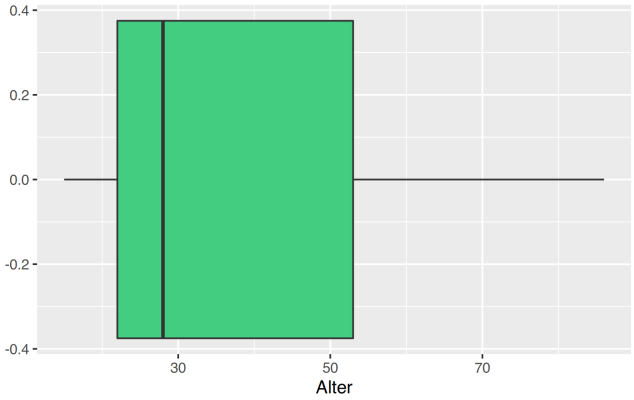


Abb. 2.140

Das Boxplot liegt auf dem Bauch, das `Alter` auf der x-Achse abgebildet werden soll. Um das Boxplot zu drehen, muss `aes()` entsprechend angepasst werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter) +
    geom_boxplot(fill="seagreen3")
```

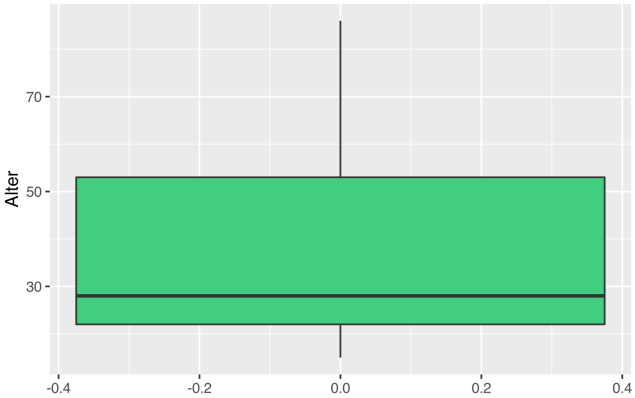


Abb. 2.141

Mit der Funktion `stat_boxplot()` können wir die Whiskers hinzugefügt werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter) +
    geom_boxplot(fill="seagreen3") +
    stat_boxplot(geom="errorbar")
```

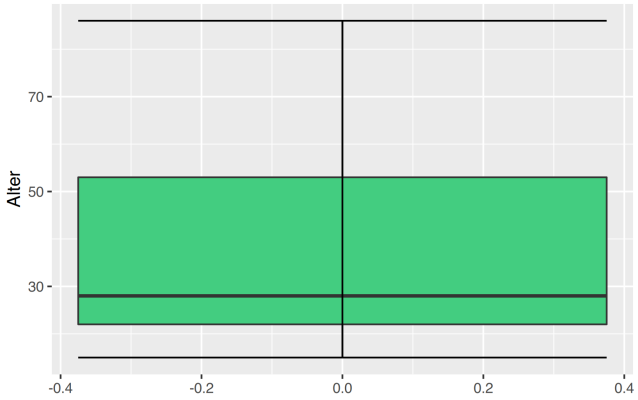


Abb. 2.142

Über den `color`-Parameter können die Farben angepasst werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter) +
    geom_boxplot(fill="seagreen3",
                 color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")
```

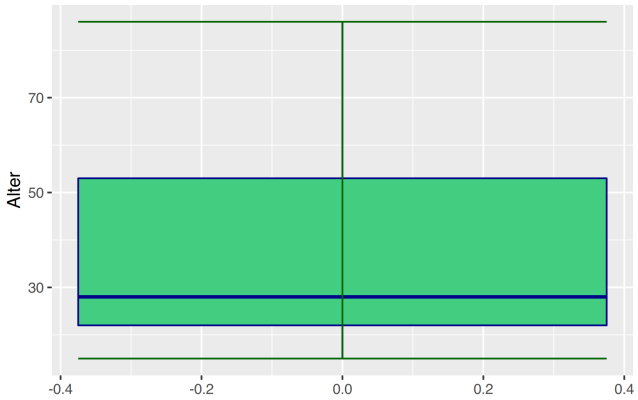


Abb. 2.143

Über die Aesthetics können wir nach `Geschlecht` gruppieren.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        x = Geschlecht) +
    geom_boxplot(fill="seagreen3",
                 color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")
```

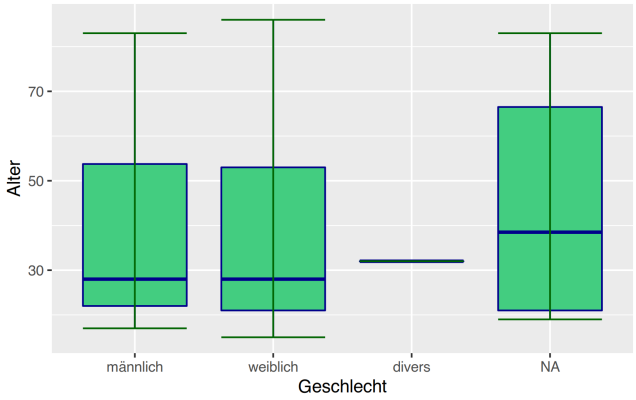


Abb. 2.144

Bzw. ohne `NA`:

```
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(y = Alter,
        x = Geschlecht) +
    geom_boxplot(fill="seagreen3",
                 color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")
```

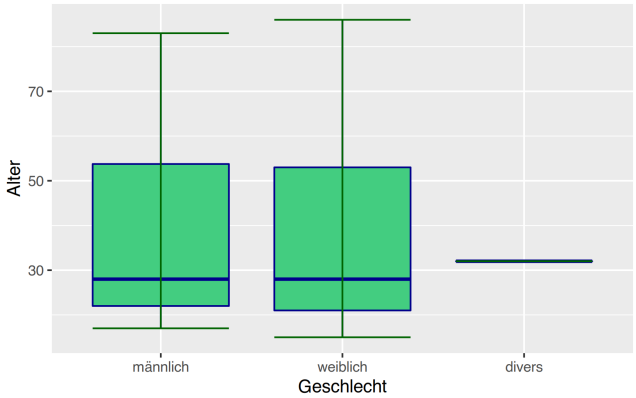


Abb. 2.145

Über die Aesthetics können wir die Boxplots auch nach `Geschlecht` *färben*. Dazu muss der Parameter aus `geom_boxplot()` entfernt werden.

```
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(y = Alter,
        x = Geschlecht,
        fill = Geschlecht) +
    geom_boxplot(color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")
```

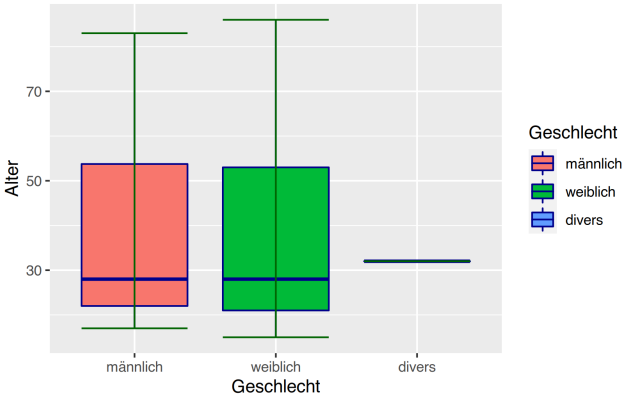


Abb. 2.146

In manchen Journals ist es üblich, ebenfalls das arithmetische Mittel per Raute den Boxplots hinzuzufügen. Dies kann mit der Funktion `stat_summary()` umgesetzt werden.

```
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(y = Alter,
        x = Geschlecht,
        fill = Geschlecht) +
    geom_boxplot(color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")+
    stat_summary(fun=mean,
                 colour="darkred",
                 geom="point",
                 shape=18,
                 size=5,
                 show.legend = F)
```

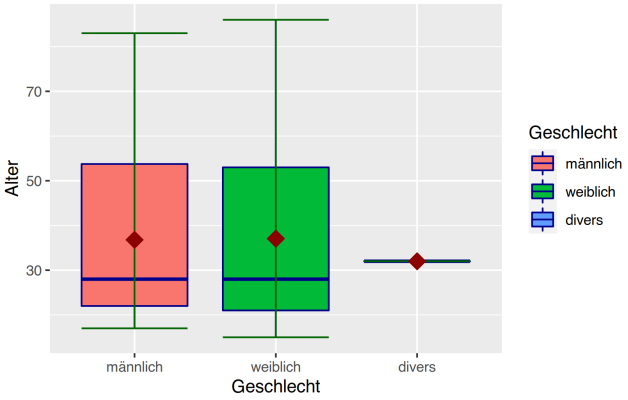


Abb. 2.147

Mit `facet_wrap()` können wir beispielweise nach `Standort` gruppieren.

```
pf8 %>%
  drop_na(Geschlecht) %>%
  ggplot() +
    aes(y = Alter,
        x = Geschlecht,
        fill = Geschlecht) +
    geom_boxplot(color="darkblue") +
    stat_boxplot(geom="errorbar",
                 color="darkgreen")+
    stat_summary(fun=mean,
                 colour="darkred",
                 geom="point",
                 shape=18,
                 size=5,
                 show.legend = F)+
    facet_wrap(~ Standort)
```

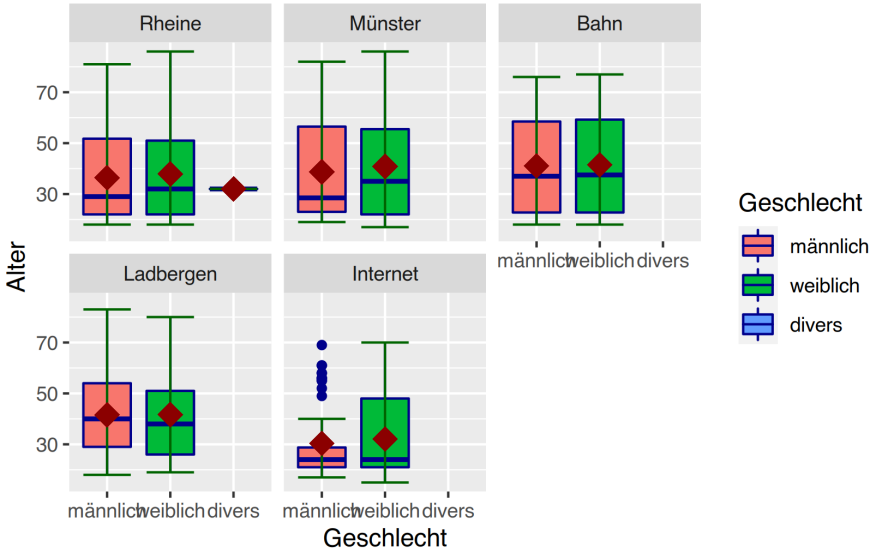


Abb. 2.148

2.4.9 Kreisdiagramme

Kreisdiagramme werden von `ggplot()` nicht direkt unterstützt, können jedoch mit einem Trick erzeugt werden. Hierbei wird das Koordinatensystem eines Balkendiagramms mittels `coord_polar()` so verbogen, dass ein Kreis dabei herauskommt.

```
# pfusche Pie-Chart zurecht
pf8 %>%
  ggplot() +
    aes(x="", fill=Geschlecht) +
    geom_bar(color="white") +
    # verbiege Koordinatensystem
    coord_polar("y") +
    # entferne Achsen und Ticks
    theme_void()
```

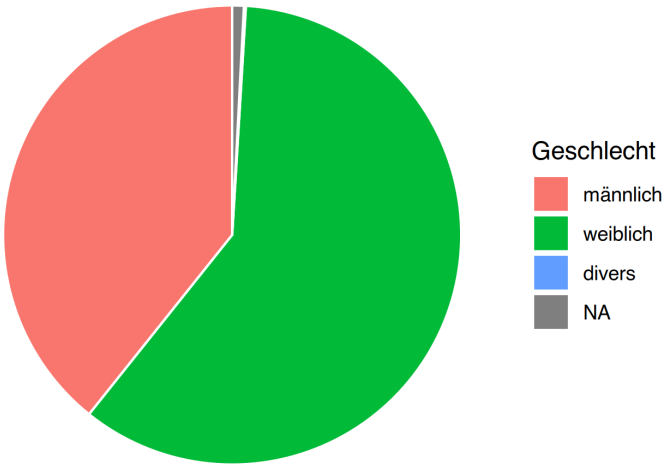


Abb. 2.149

Das funktioniert auch mit `geom_col()` und absoluten Häufigkeiten.

```
# vorgegebene Werte
df <- data.frame(Gruppe=c("Vanille","Schokolade","Erdbeer"),
                 Anteil=c(60, 25, 15))

# pfusche Pie-Chart zurecht
ggplot(df, aes(x="",
               y=Anteil,
               fill=Gruppe)) +
    geom_col(color="white") +
    # verbiege Koordinatensystem
    coord_polar("y", start=0) +
    # entferne Achsen und Ticks
    theme_void()
```

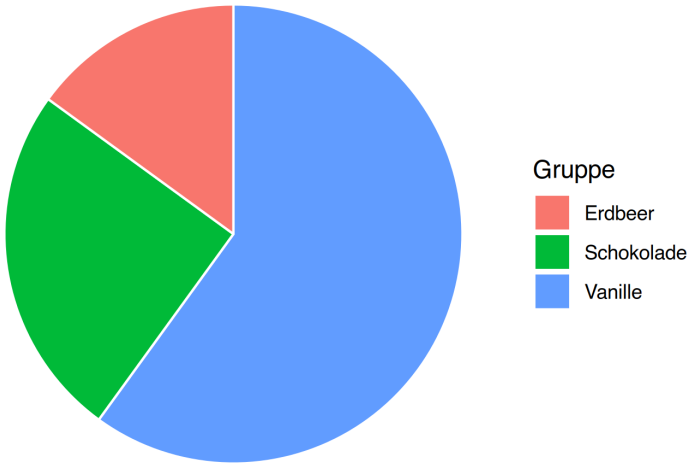


Abb. 2.150

Mittels `geom_text()` können die Werte in den Kreis geschrieben werden.

```
ggplot(df, aes(x="", y=Anteil, fill=Gruppe)) +
    geom_col(color="white") +
    # füge Werte hinzu
    geom_text(aes(label = Anteil),
              position = position_stack(vjust = 0.5)) +
    # verbiege Koordinatensystem
    coord_polar("y", start=0) +
    # entferne Achsen und Ticks
    theme_void()
```

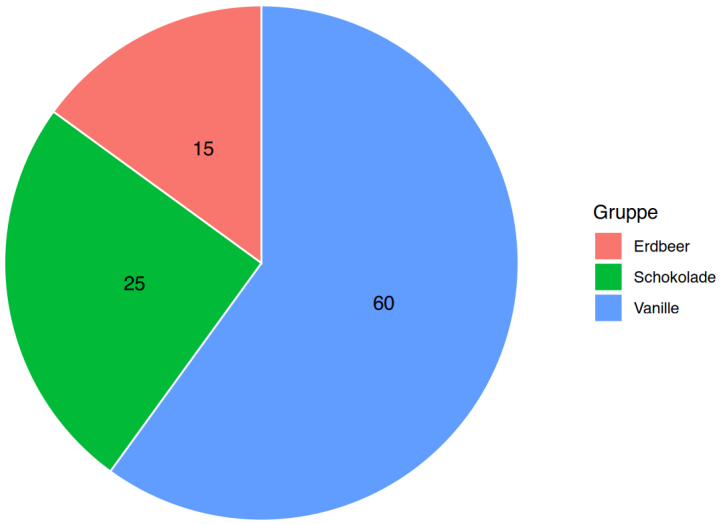


Abb. 2.151

2.4.10 QQ-Plots

Zur Überprüfung, ob eine Variable normalverteilt ist, werden häufig Quantil-Quantil-Diagramme (QQ-Plots) erzeugt. Für `ggplot()` steht hierfür die Funktion `stat_qq()` zur Verfügung.

```
# erstelle QQ-Plot für Variable "age"
# gruppiert nach "sex"
ggplot(epa, aes(sample=age)) +
 stat_qq(aes(color=sex))
```

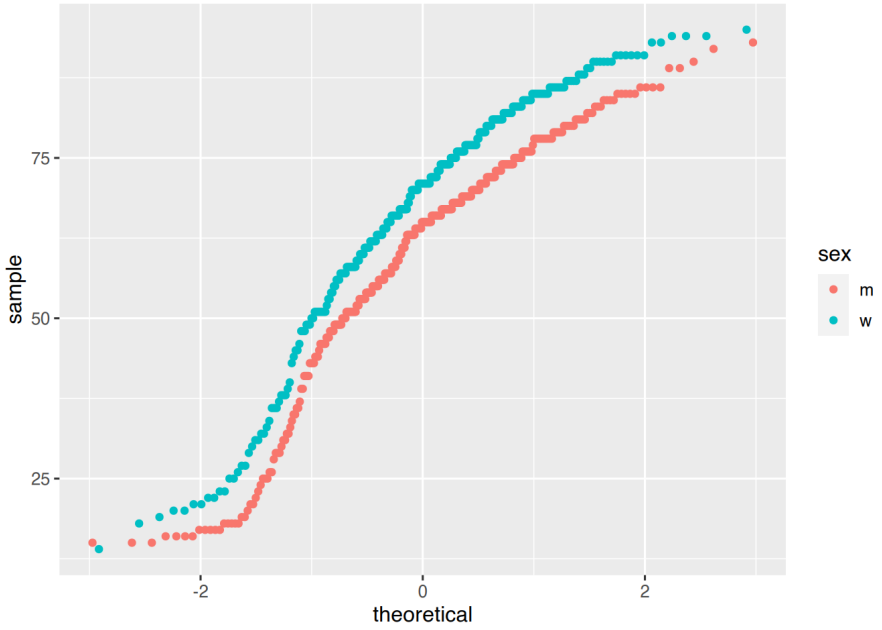


Abb. 2.152

Mit Zusatzpaketen kann auch hier die Funktionalität erhöht werden. Beispielsweise ist im Paket `ggpubr` die Funktion `ggqqplot()` enthalten, die automatisch das Konfidenzintervall hinzufügt. Der Aufruf ist etwas anders als bei `ggplot()`:

```
ggpubr::ggqqplot(epa, x="age", color="sex")
```

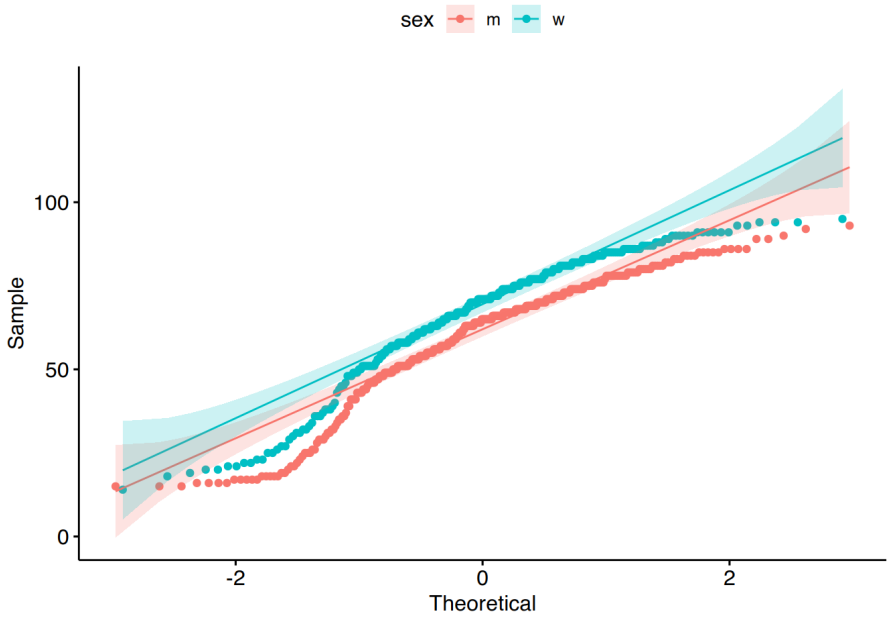


Abb. 2.153

Weitere Möglichkeiten bietet das Zusatzpaket `qqplotr`. Es fügt weitere Geome und Funktionen für `ggplot` hinzu.

```
# aktiviere die Zusatzfunktionen
library(qqplotr)

ggplot(epa, aes(sample=age)) +
  # zeichne Punkte wie im "Original"
  stat_qq_point(aes(color=sex)) +
  # füge Linie hinzu
  stat_qq_line(aes(color=sex)) +
  # füge Konfidenzgrenzen hinzu
  stat_qq_band(aes(fill=sex))
```

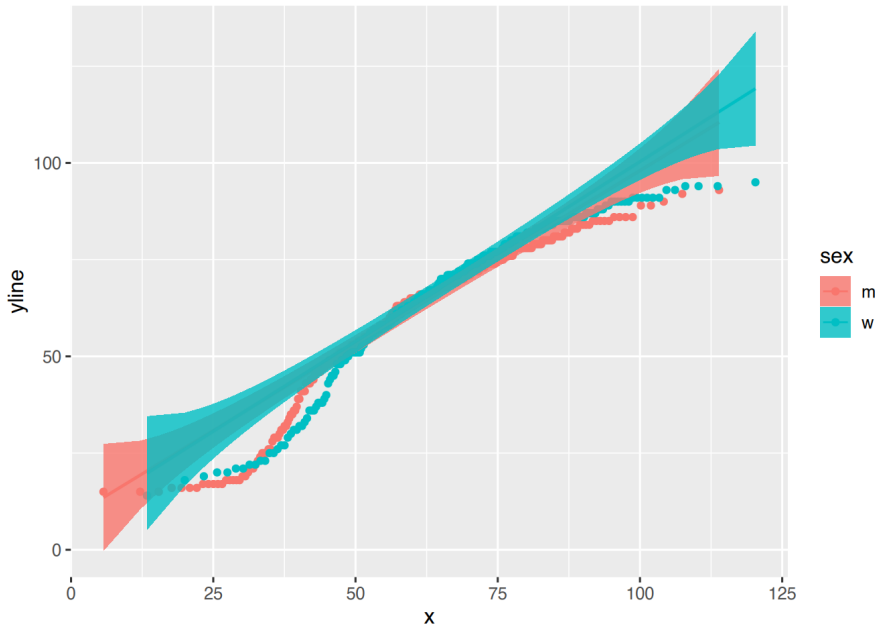


Abb. 2.154

Dieses Plot ist aber nicht ganz so “schön” wie die vorangegangenen. Es wird besser, wenn wir die Layerreihenfolge vertauschen und den Alpha-Wert der Farbe herbasetzen.

```
ggplot(epa, aes(sample=age)) +
  # fange mit Konfidenzgrenzen an
  # setzte Alpha-Wert für Farbe
  stat_qq_band(aes(fill=sex), alpha=2/10) +
  # füge Linie hinzu
  stat_qq_line(aes(color=sex)) +
  # zeichne Punkte wie im "Original"
  stat_qq_point(aes(color=sex))
```

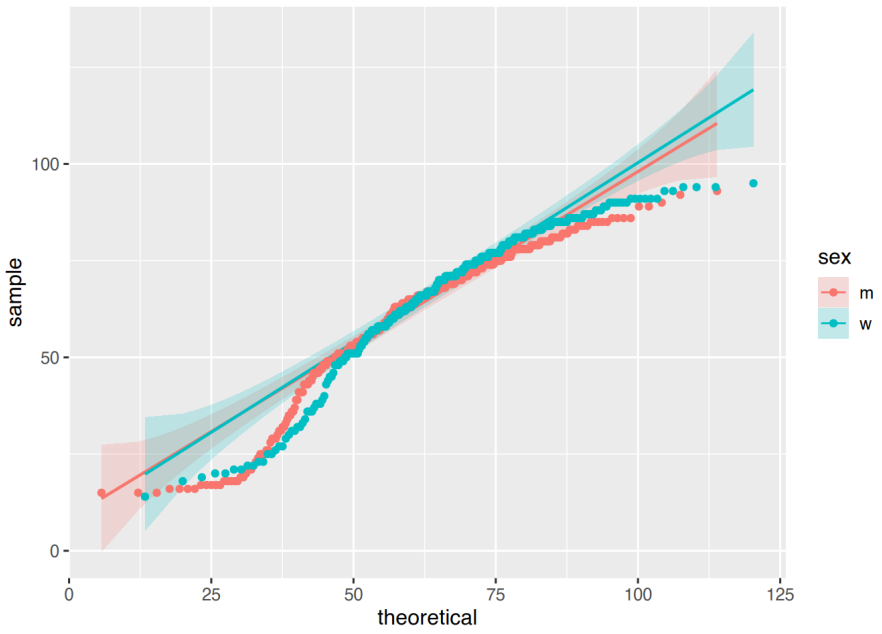


Abb. 2.155

2.4.11 Diagramme speichern

2.4.11.1 R-Objekt

`ggplot()`-Diagramme können in R-Objekte (zwischen)gespeichert werden.

```
# Plot in Obkjekt "p" speichern
p <- ggplot(pf8) +
       aes(x="", fill=Geschlecht) +
       geom_bar(color="white")
# Plot anzeigen
p
```

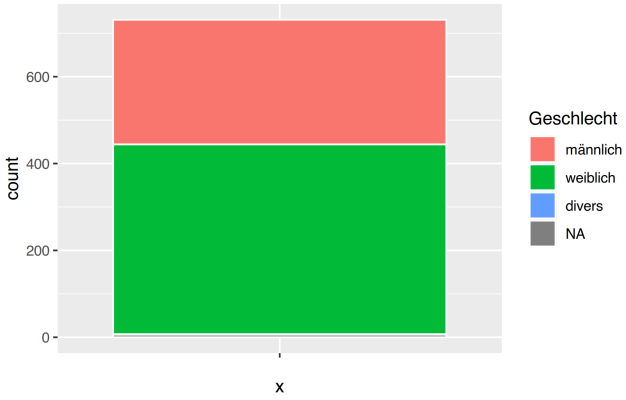


Abb. 2.156

Das Objekt kann nun weitere Funktionen verarbeiten.

```
# Plot in Obkjekt "p" speichern
p <- ggplot(pf8) +
       aes(x="", fill=Geschlecht) +
       geom_bar(color="white")
# Plot weiter bearbeiten
p + coord_polar("y", start=0) +
    theme_void()
```

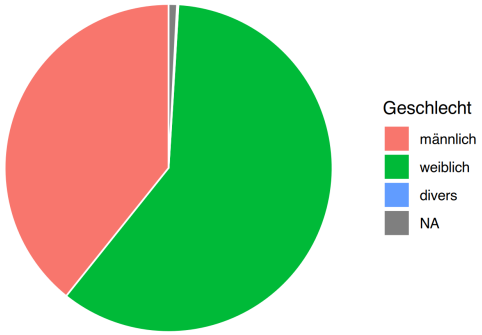


Abb. 2.157

2.4.11.2 in Datei speichern

Mit der Funktion `ggsave()` können `ggplot()`-Diagramme als Datei gespeichert werden.

```
# Plot erzeugen
ggplot(pf8) +
    aes(x="", fill=Geschlecht) +
    geom_bar(color="white") +
    coord_polar("y", start=0) +
    theme_void()

# Plot in Datei speichern
ggsave("MeinPlot.png",
       units="mm",
       width=400,
       height=200,
       dpi=300,
       pointsize=7)
```

Liegt der Plot in einem R-Objekt, kann dies mit dem Parameter `plot` angegeben werden.

```
# Plot ist in Obkjekt "p" gespeichert
ggsave("MeinPlot.png",
       plot= p,
       units="mm",
       width=400,
       height=200,
       dpi=300,
       pointsize=7)
```

2.4.12 Aussehen ändern

2.4.12.1 Titel und Überschriften

Die Plottitel können mit der Funktion `labs()` verändert werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  labs(title="Dies ist Überschrift",
       subtitle="Dies ist die Unter-Überschrift",
       caption="Dies ist die Bildunterschrift")
```

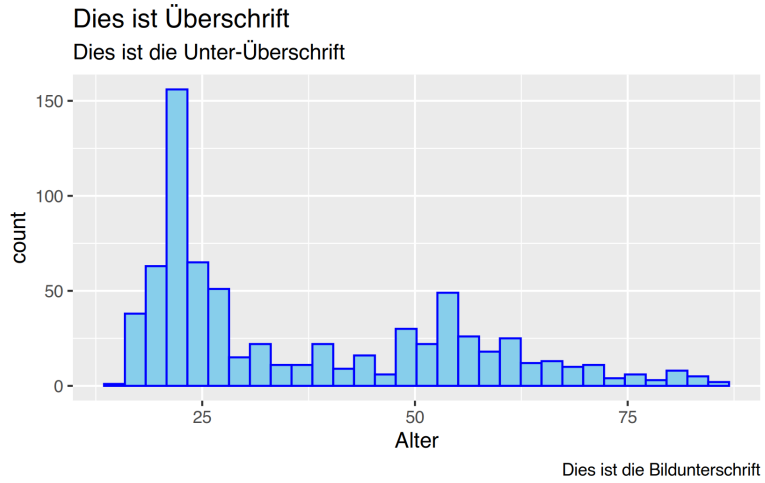


Abb. 2.158

Die Schriftart kann mittels `theme()`-Funktion geändert werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  labs(title="Dies ist Überschrift",
       subtitle="Dies ist die Unter-Überschrift in fett",
       caption="Dies ist die Bildunterschrift in kursiv") +
  theme(plot.title = element_text(size=18)) +
  theme(plot.subtitle = element_text(size=10, face="bold")) +
  theme(plot.caption = element_text(size=8, face="italic"))
```

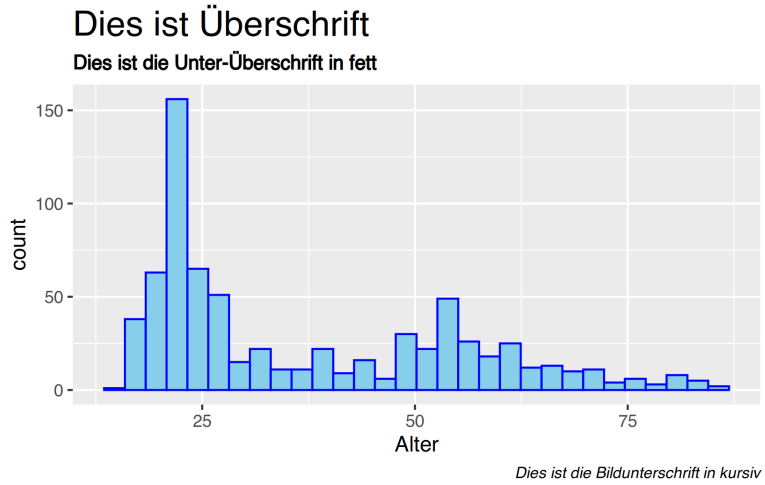


Abb. 2.159

2.4.12.2 Achsen

Die Achsenbeschriftung kann mittels `xlab()` und `ylab()` verändert werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  xlab("Ich bin die x-Achse") +
  ylab("Ich bin die y-Achse")
```

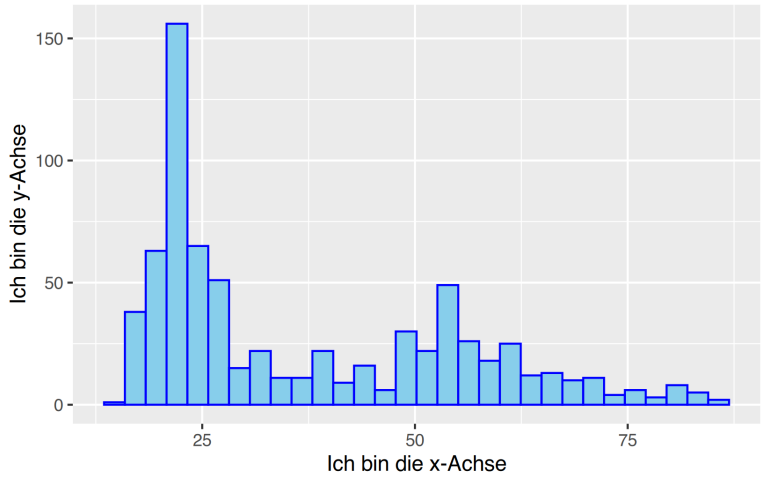


Abb. 2.160

Die Längen der Achsenausschnitte können mit `xlim()` und `ylim()` angegeben werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  xlab("Ich bin die x-Achse") +
  ylab("Ich bin die y-Achse")+
  xlim(20, 50) +
  ylim(-20, 75)
```

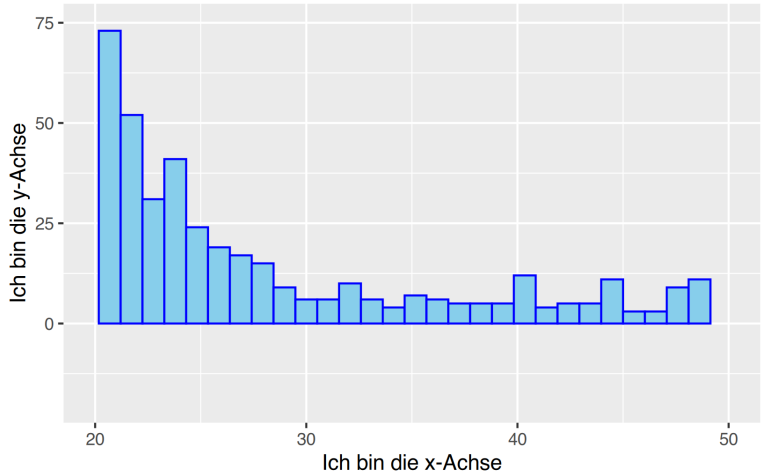


Abb. 2.161

Die Anzahl und Position der Achsenmarkierungen können mit `scale_x_continuous()` und `scale_y_continuous()` angepasst werden.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  xlab("Ich bin die x-Achse") +
  ylab("Ich bin die y-Achse")+
  xlim(20, 50) +
  ylim(-20, 75) +
  scale_x_continuous(breaks = c(20, 25, 30, 35, 40)) +
  scale_y_continuous(breaks = seq(-20, 100, 10) )
```

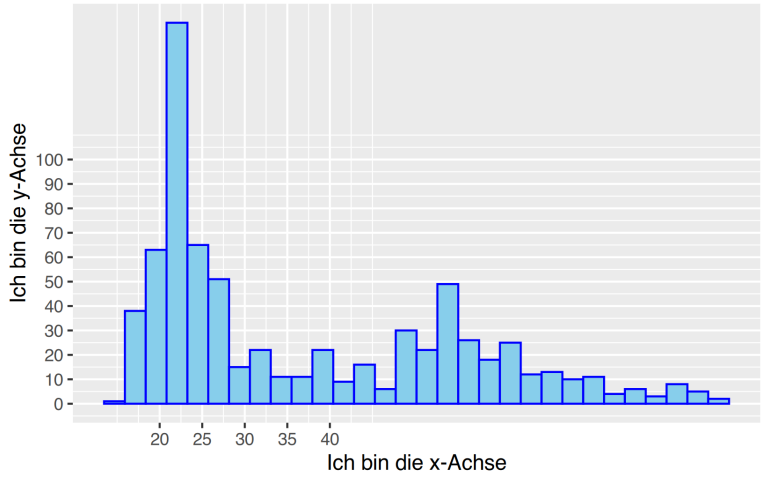


Abb. 2.162

Die Schriftarten werden ebenfalls mit der `theme()`-Funktion gesteuert.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
  xlab("Ich bin die x-Achse") +
  ylab("Ich bin die y-Achse")+
  xlim(20, 50) +
  ylim(-20, 75) +
  scale_x_continuous(breaks = c(20, 25, 30, 35, 40)) +
  scale_y_continuous(breaks = seq(-20, 100, 10) ) +
  theme(axis.title.x = element_text(size=18, face="bold")) +
  theme(axis.title.y = element_text(size=14, face="italic")) +
  theme(axis.text.x = element_text(size=8, color="red")) +
  theme(axis.text.y = element_text(size=4, color="blue")) +
  theme(axis.ticks.x = element_line(size=8, color="darkgreen")) +
  theme(axis.ticks.y = element_line(size=8, color="orange"))
```

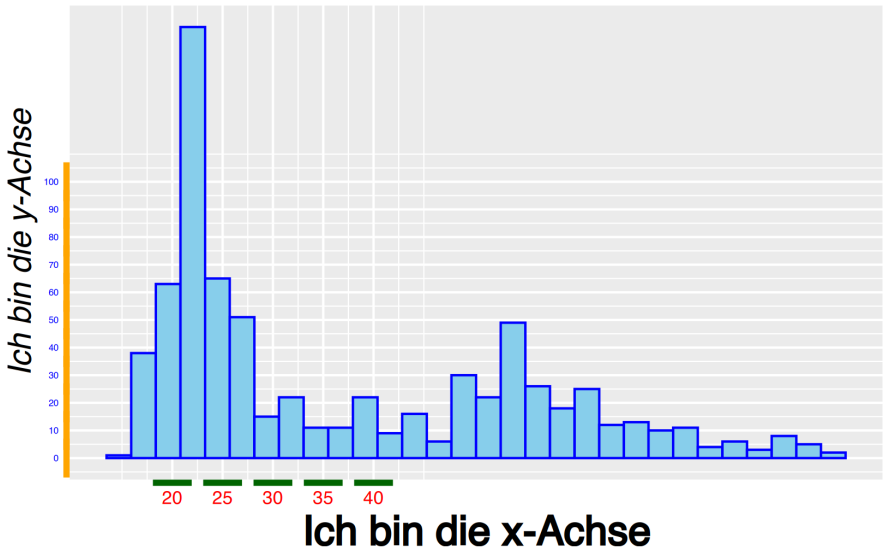


Abb. 2.163

So können die Achsenbeschriftungen auch über den Parameter `angle` gedreht werden.

```
ggplot(pf8) +
  aes(x=SportWarum ) +
  geom_bar(color="blue",
           fill="skyblue") +
  theme(axis.text.x = element_text(angle = 45,
                                   hjust = 1,
                                   size=10))
```

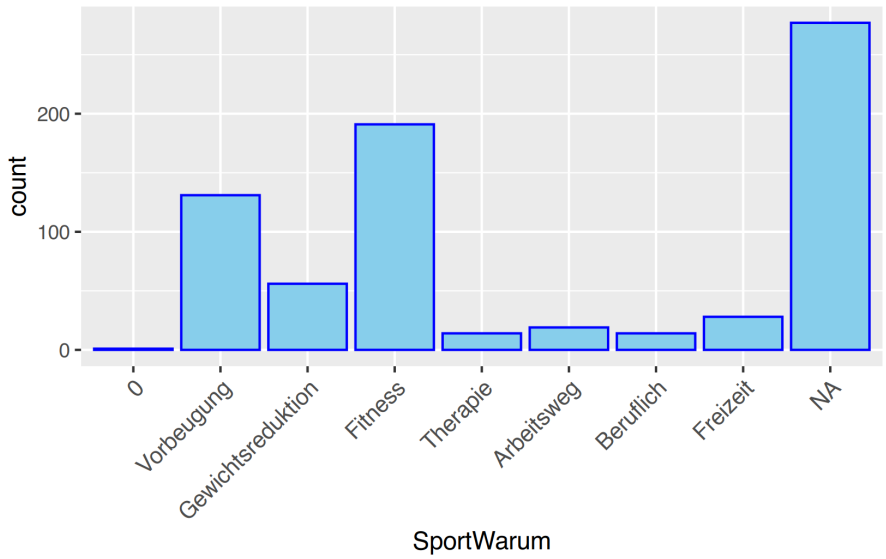


Abb. 2.164

Sollen die Ticks und ihre Striche vollständig entfernt werden, so geht dies mit den Funktionen `theme()` und `element_blank()`.

```
ggplot(pf8) +
  aes(x=Alter) +
  geom_histogram(color="blue",
                 fill="skyblue") +
```

```
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```

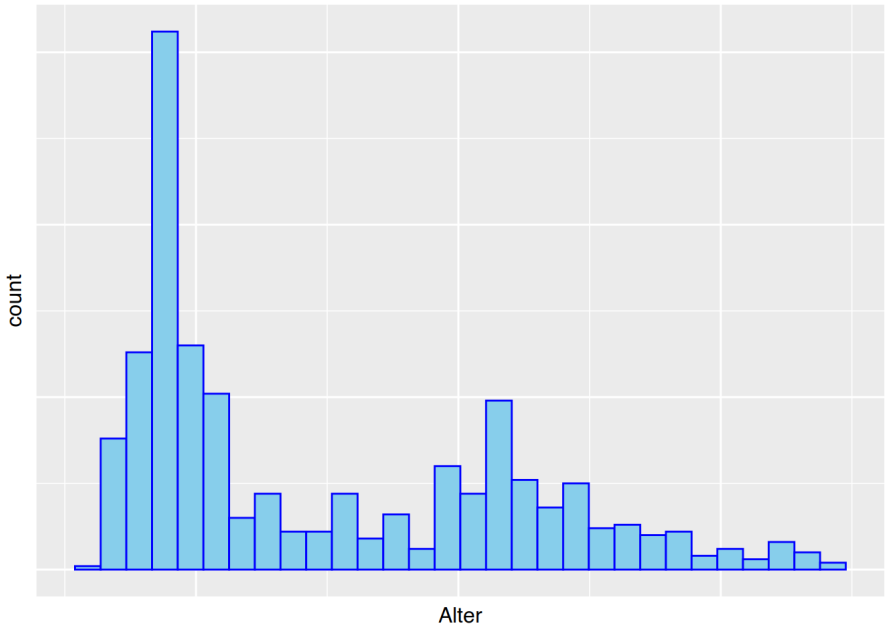


Abb. 2.165

2.4.12.3 Legendenbox

Der **Title der Legendbox** kann prinzipiell geändert werden, indem die ursprüngliche Variable umbenannt wird. Dann ändert sich der Legendenboxtitel ebenfalls entsprechend. Sollte dies keine "gute" Lösung sein, z.B. weil mehrere Wörter als Titel verwendet werden sollen, kann der Title über die `labs()`-Funktion geändert werden. Hierbei besteht eine Abhängigkeit zur Gruppierung über die `aes()`-Funktion. Im folgenden Beispiel wird die Gruppierung über den Parameter `fill=Geschlecht` erzeugt. Daher muss der Legendentitel mittels `labs(fill="NEU")` geändert werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Geschlecht) +
    geom_boxplot() +
    labs(fill="Neuer Titel")
```

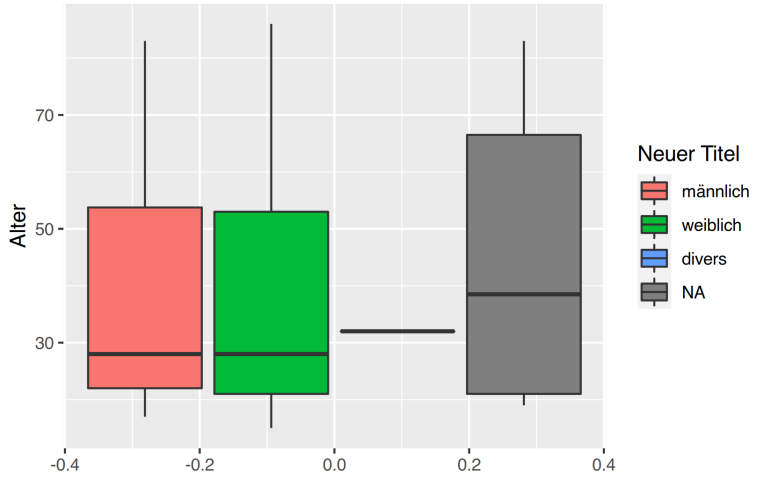


Abb. 2.166

Sollen die **Einträge der Legenbox** geändert werden, kann dies zum einen erfolgen, indem die Levels des Originalfactor entsprechend umgeschrieben werden. Dann ändern sich auch die Einträge in der Legendenbox. Sollte dies keine "gute" Lösung sein, können die Einträge von Hand über die Funktion `scale_fill_discrete()` überschrieben werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Geschlecht) +
    geom_boxplot() +
    labs(fill="Neuer Titel") +
    scale_fill_discrete(labels=c('men', 'woman', 'divers', 'no answer'))
```

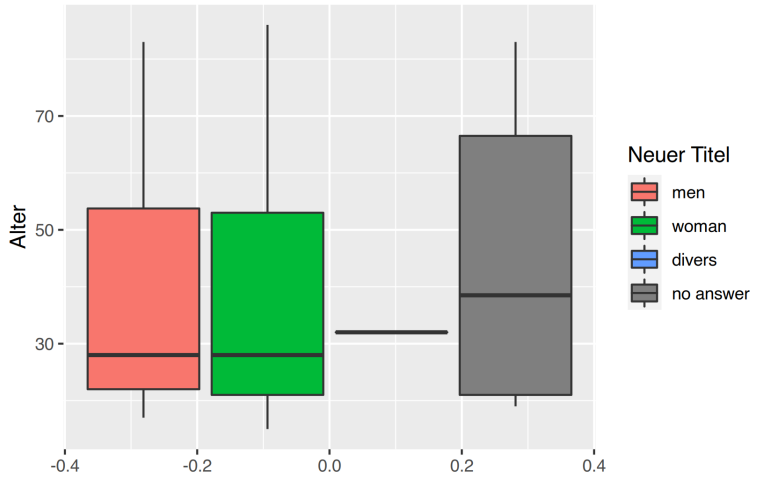


Abb. 2.167

Soll die **Reihenfolge der Levelboxelemte** geändert werden, kann ebenfalls auf die Funktion `scale_fill_discrete()` zurückgegriffen werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Geschlecht) +
    geom_boxplot() +
    labs(fill="Neuer Titel") +
    scale_fill_discrete(breaks=c(NA, "weiblich", "divers", "männlich"))
```

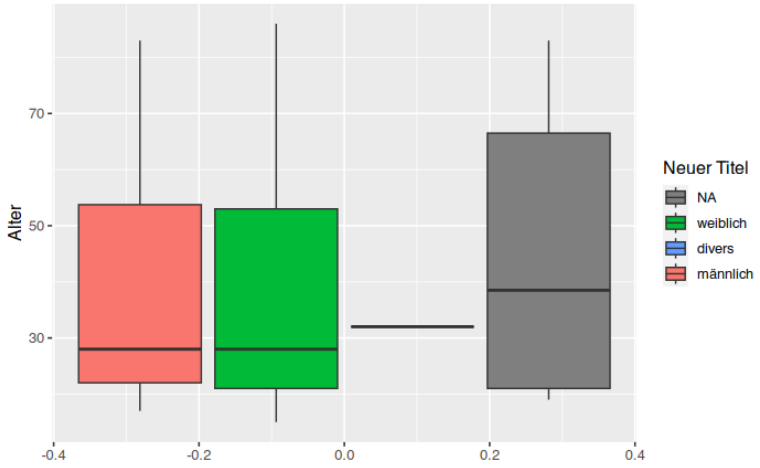


Abb. 2.168

Die **Schriftarten** der Legendenbox können ebenfalls mit der `theme()`-Funktion verändert werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Geschlecht) +
    geom_boxplot() +
    theme(legend.title = element_text(color="red", face="bold")) +
    theme(legend.text = element_text(color="blue", face="italic"))
```

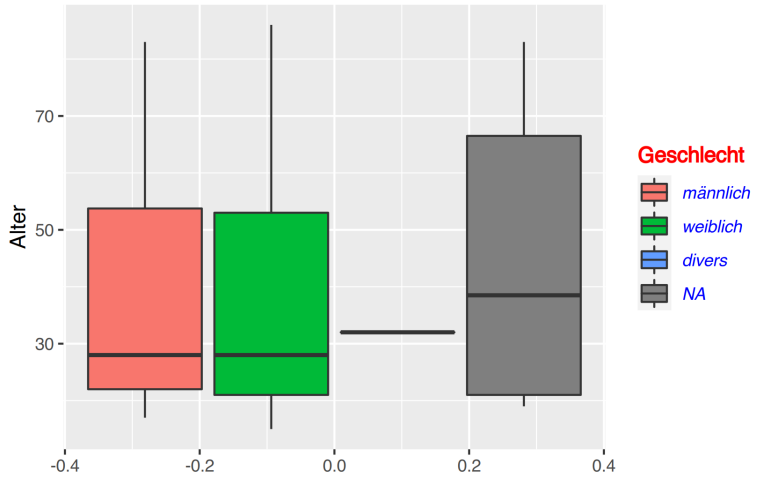


Abb. 2.169

Soll die Legendenbox vollständig entfernt werden, so geht dies über die Funktion `theme(legend.position = "none")`

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Geschlecht) +
    geom_boxplot() +
    theme(legend.position = "none")
```

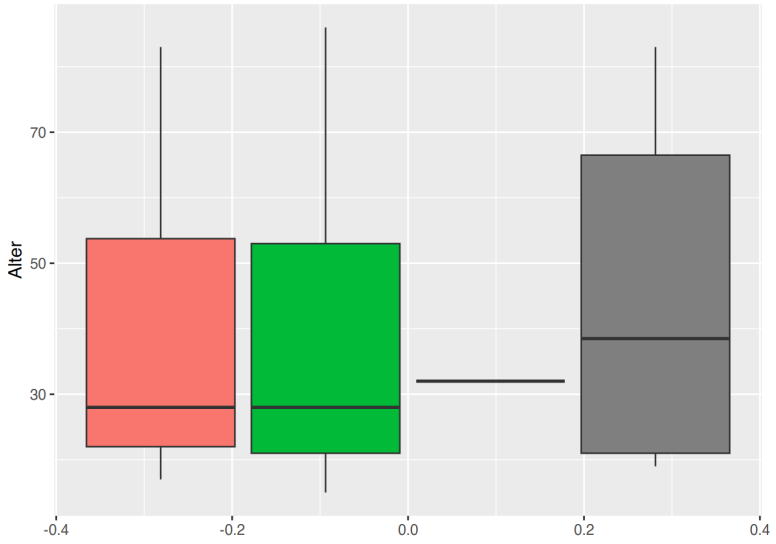


Abb. 2.170

2.4.12.4 Farben

Ohne weitere Angaben verwendet `ggplot()` seine "eingebauten" Standardfarben.

Wenn wir das `Alter` der Probanden nach `Standort` gruppieren, wählt `ggplot()` diese Farben aus:

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        color= Standort,
        fill= Standort) +
    geom_boxplot()
```

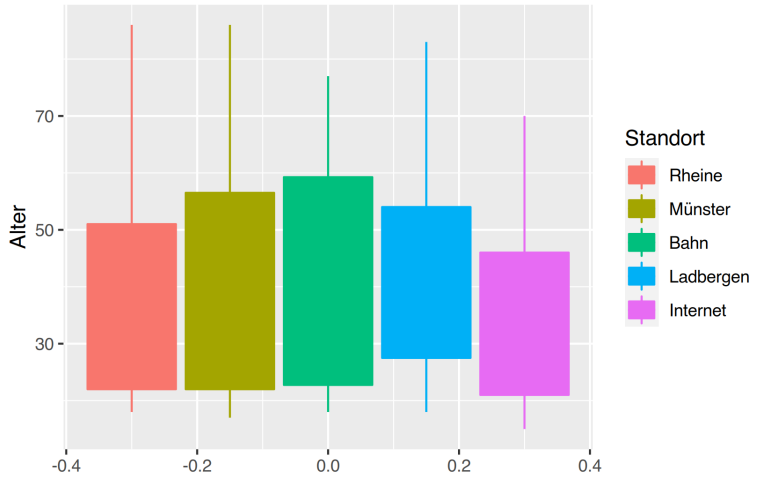


Abb. 2.171

Die Farbintensität und -helligkeit der Standardfarben können über die Funktionen `scale_color_hue()` und `scale_fill_hue()` angepasst werden. Hierbei steht der Parameter `c` für die Farbintensität, und `l` für die Helligkeit.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        color= Standort,
        fill= Standort) +
    geom_boxplot() +
    scale_color_hue(l=40, c=35) +
    scale_fill_hue(l=40, c=35)
```

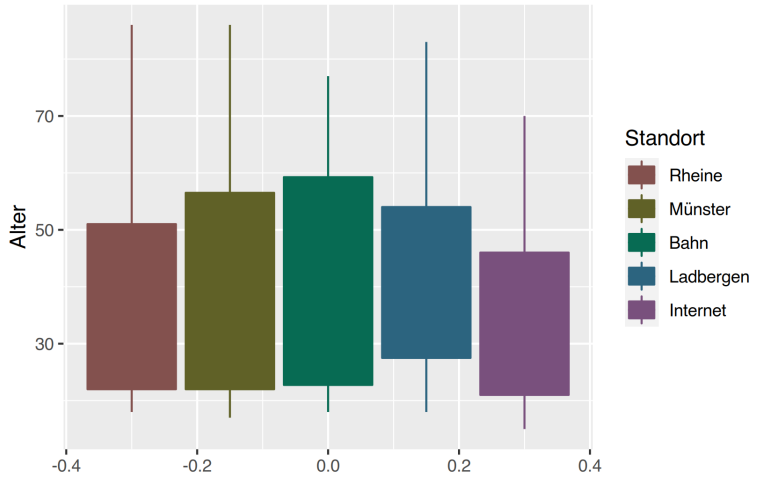


Abb. 2.172

Über die Funktionen `scale_color_manual()` und `scale_fill_manual()` können die Farben manuell ausgewählt werden (eine Liste aller Farbnamen liefert die Funktion `colors()`):

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        color= Standort,
        fill= Standort) +
    geom_boxplot() +
    scale_color_manual(values=c("darkblue", "navy", "blue",
                                "skyblue2", "skyblue3")) +
    scale_fill_manual(values=c("hotpink", "pink", "pink3",
                               "red3", "red4"))
```

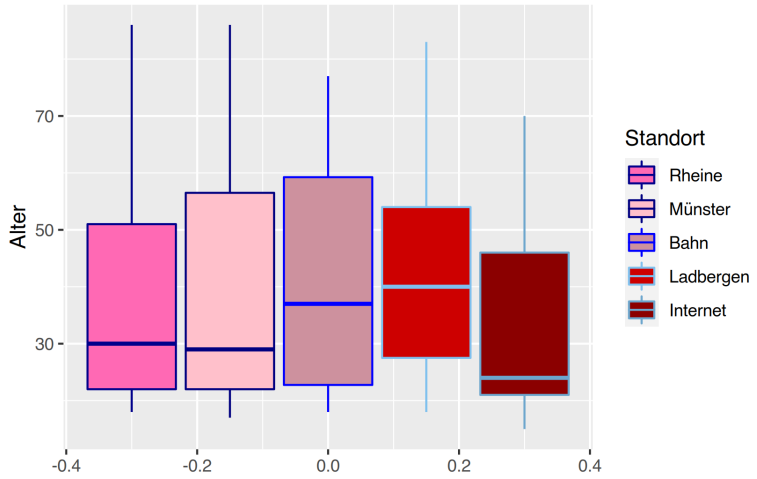


Abb. 2.173

Die Farben können über den `breaks`-Parameter auch direkt den Levelausprägungen zugewiesen werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        color= Standort,
        fill= Standort) +
    geom_boxplot() +
    scale_color_manual(values=c("darkblue", "navy", "blue",
                                "skyblue2", "skyblue3"),
                       breaks=c("Internet", "Ladbergen", "Münster",
                                "Rheine", "Bahn")) +
    scale_fill_manual(values=c("hotpink", "pink", "pink3",
                               "red3", "red4"),
                      breaks=c("Internet", "Ladbergen", "Münster",
                               "Rheine", "Bahn"))
```

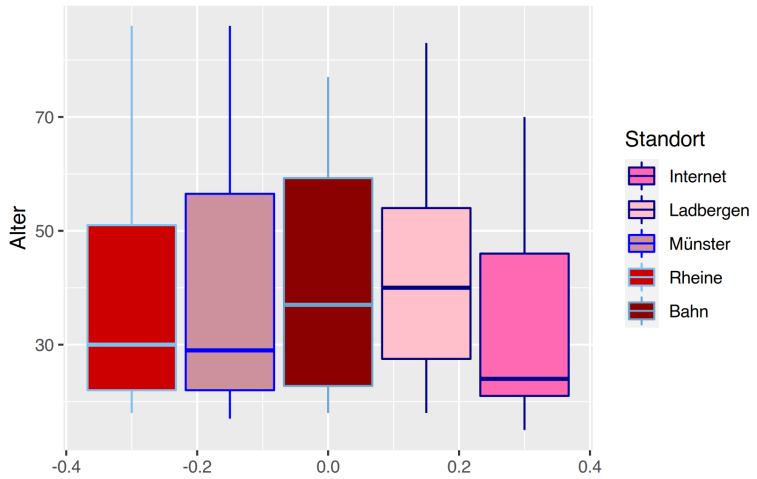


Abb. 2.174

Hierdurch hat sich auch die Reihenfolge der Levelboxelemte geändert.

Soll die Reihenfolge der Levelboxelemte geändert werden, ohne dass eigene Farben definiert werden, kann auf die Funktion `scale_fill_discrete()` zurückgegriffen werden.

```
pf8 %>%
  ggplot() +
    aes(y = Alter,
        fill= Standort) +
    geom_boxplot() +
    scale_fill_discrete(breaks=c("Internet", "Ladbergen", "Bahn",
                                 "Rheine", "Münster"))
```

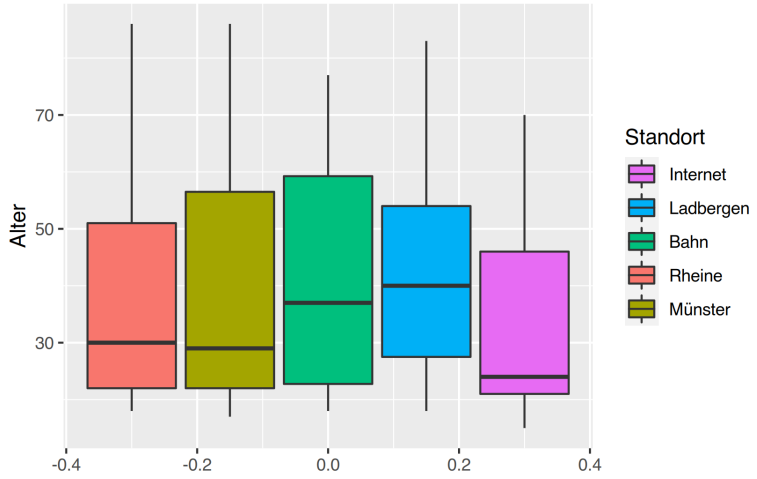


Abb. 2.175

2.4.12.5 Linien

Mit den Funktionen `geom_vline()` und `geom_hline()` können vertikale und horizontale Linien hinzugefügt werden. Mit den Parametern `xintercept` und `yintercept` wird der Schnitt durch die jeweilige Achse festegelegt.

```
ggplot(pf8) +
    aes(x = Alter,
        y = Gewicht) +
    geom_point() +
    geom_hline(yintercept = 60,
               linetype = "solid",
               color = "blue",
               size = 3) +
    geom_vline(xintercept=40,
               linetype="dashed",
               color="red")
```

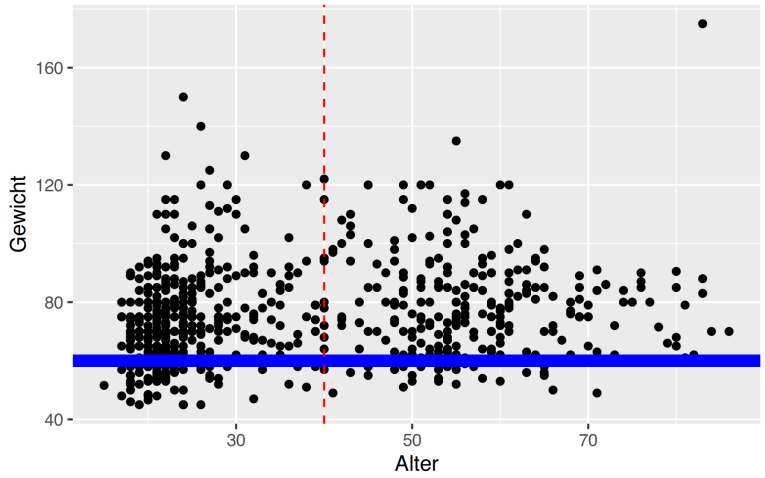


Abb. 2.176

Um zwei beliebige Punkte mit einer Linie zu verbinden, muss zunächst ein Hilfsdatenframe mit den Koordinaten der Punkte erstellt werden. Dieses neue Datenframe wird dann der Funktion `geom_line()` übergeben.

```
# Hilfsdatenframe
strich <- data.frame(x = c(30,50),
                     y = c(10,90))

ggplot(pf8) +
  aes(x = Alter,
      y = Gewicht) +
  geom_point() +
  geom_line(data = strich,
            aes(x=x, y=y),
            color="darkgreen",
            size=1)
```

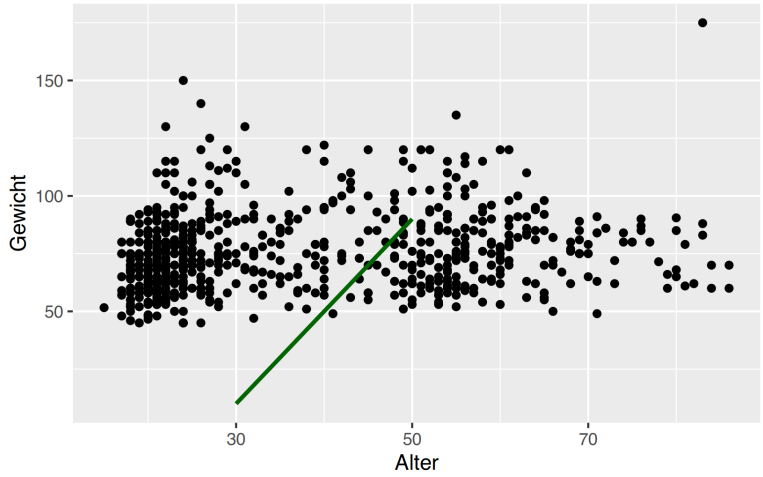


Abb. 2.177

2.4.12.6 Text

Texte können mit der `annotate()`-Funktion hinzugfügt werden.

```
ggplot(pf8) +
  aes(x = Alter,
      y = Gewicht) +
  geom_point() +
  annotate(geom="text",
           x=30, y=120,
           label="Huhu",
           color="darkgreen",
           size=14)
```

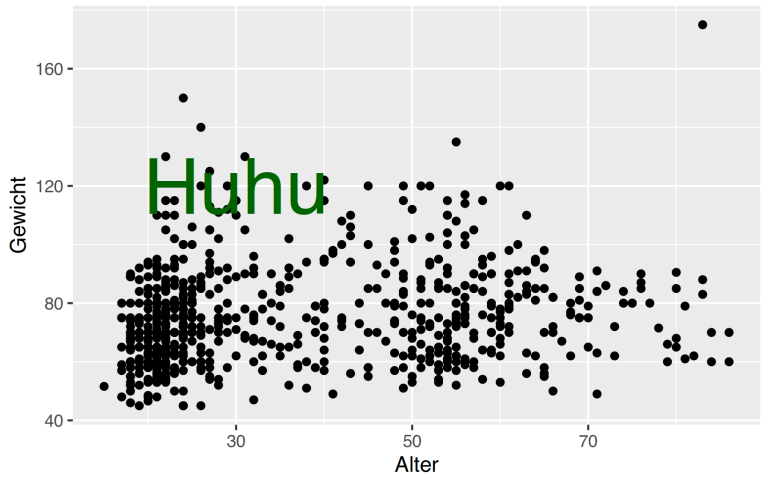


Abb. 2.178

2.4.12.7 Sonderzeichen

Sonderzeichen und Mathematiksymbole können mittels der `expression()`-Funktion hinzugefügt werden.

```
ggplot(pf8) +
  aes(x = Alter,
      y = Gewicht) +
  geom_point() +
  labs(title = expression("Dies ist ein Text mit"~a^2~+~b^2~"="~c^2),
       subtitle = expression(sigma~sqrt(alpha)~beta^3~"sind hier"~x[1])
       )
```

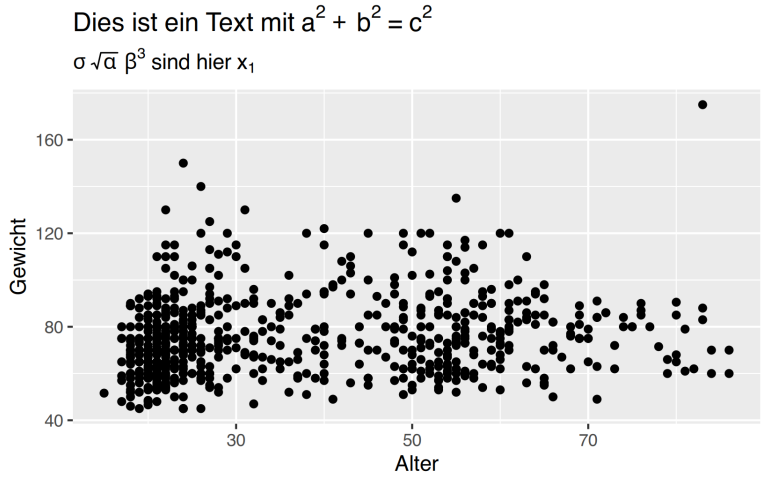


Abb. 2.179

2.4.12.8 Themes

In `ggplot()` sind 8 verschiedene "Themes" implementiert, die das Aussehen der nicht-datenbezogenen Anteile des Plots festlegen. Das Standardtheme ist `theme_grey()`, die anderen sind:

- `theme_bw()`: Eine Variante von `theme_grey()` mit weißem Hintergrund und dünnen grauen Gitterlinien.
- `theme_linedraw()`: Ein Theme mit nur schwarzen Linien unterschiedlicher Stärke auf weißem Hintergrund, das an eine Zeichnung erinnert.
- `theme_light()`: Ähnlich wie `theme_linedraw()`, aber mit hellgrauen Linien und Achsen, um die Aufmerksamkeit stärker auf die Daten zu lenken.
- `theme_dark()`: Das dunkle Pendant von `theme_light()`, mit ähnlichen Linienstärken, aber dunklem Hintergrund. Nützlich, um dünne farbige Linien hervorzuheben.
- `theme_minimal()`: Ein minimalistisches Theme ohne Hintergrundanmerkungen.
- `theme_classic()`: Ein klassisch aussehendes Theme mit x- und y-Achsen-Linien und ohne Gitterlinien.
- `theme_void()`: Ein vollständig leeres Theme.

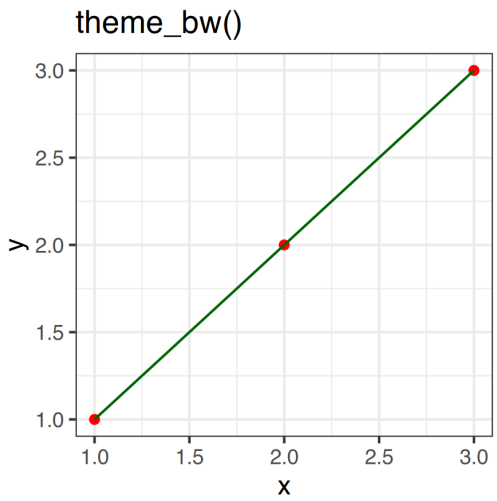
```r
# dummy Daten
df <- data.frame(x = 1:3, y = 1:3)

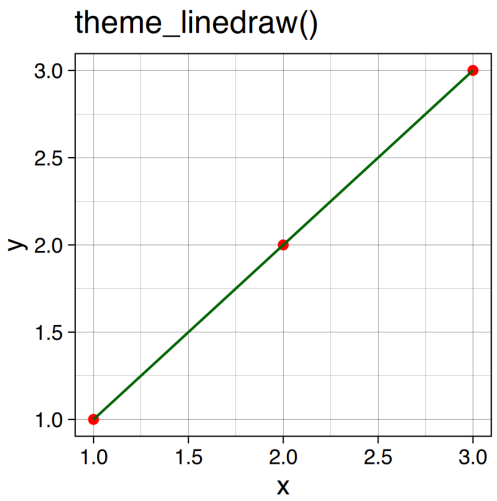
# Basisplot
base <- ggplot(df, aes(x, y)) +
  geom_point(color="red") +
  geom_line(color="darkgreen")

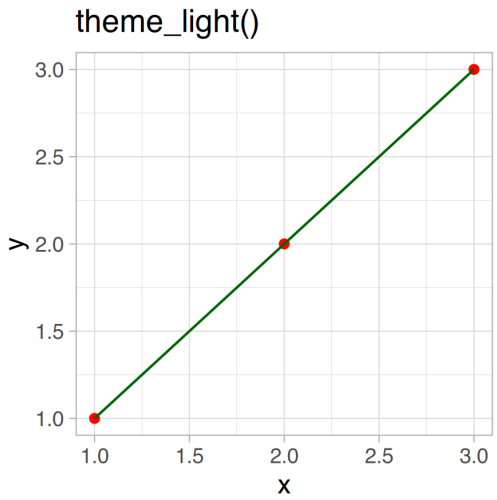
# Themepark
base +  ggtitle("theme_grey()")
base + theme_grey() + ggtitle("theme_grey()")
base + theme_bw() + ggtitle("theme_bw()")
base + theme_linedraw() + ggtitle("theme_linedraw()")
base + theme_light() + ggtitle("theme_light()")
base + theme_dark() + ggtitle("theme_dark()")
base + theme_minimal()  + ggtitle("theme_minimal()")
base + theme_classic() + ggtitle("theme_classic()")
base + theme_void() + ggtitle("theme_void()")
```

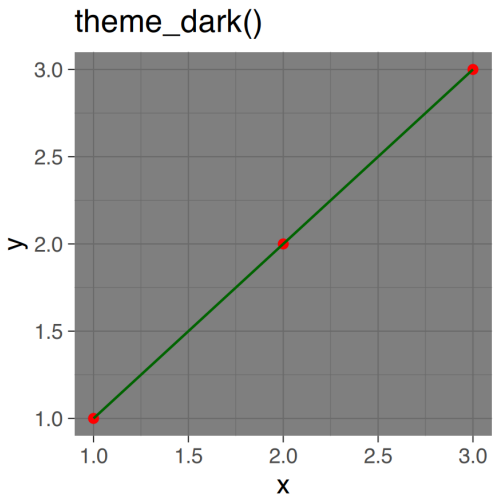


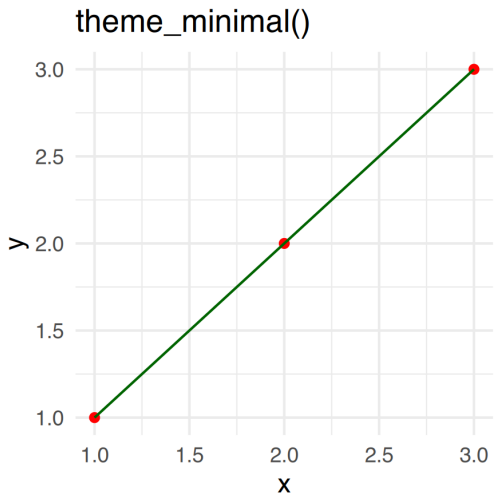


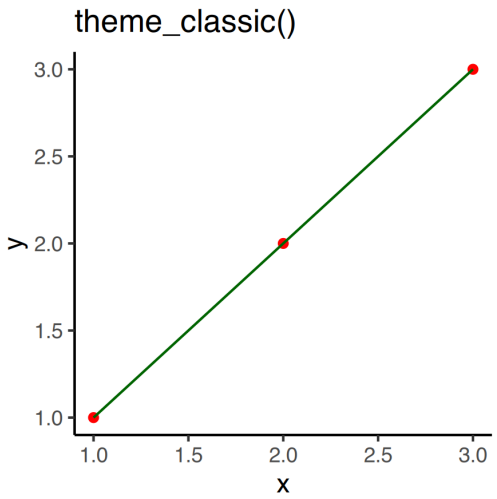

theme_bw()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x

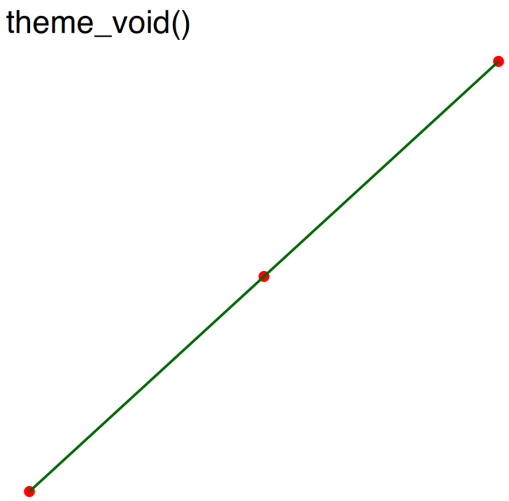

theme_linedraw()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x


theme_light()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x


theme_dark()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x


theme_minimal()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x


theme_classic()
3.0
2.5
2.0
1.5
1.0
y
1.0
1.5
2.0
2.5
3.0
x


theme_void()

3 Was willst du machen?

3.1 Daten manipulieren

Ich möchte meine Datenobjekte manipulieren.

3.1.1 Einer Kreuztabelle Prozentwerte hinzufügen

Ich möchte eine Kreuztabelle ausgeben, die sowohl absolute Häufigkeiten als auch Prozentwerte enthält.
Ich erstelle folgende Kreuztabelle:

```
A <- rep(c("A1", "A2", "A3"), 10)
B <- sample(1:5, size=30, replace=TRUE)
C <- sample(c("ja", "nein"), size=30, replace=TRUE)
df <- data.frame(rbind(table(B, A),
                 table(C, A)))
df
```

```
      A1 A2 A3
1      0  2  3
2      1  1  1
3      1  2  1
4      6  1  4
5      2  4  1
ja     7  6  3
nein   3  4  7
```

Die einzelnen Werte sollen nun mit den jeweiligen Prozentwerten ergänzt werden. Dies kann mit den Funktionen `transmute()` und `across()` erfolgen.

```
df %>%
            # über jede Spalte
  transmute(across(everything(),
                   # x ausgeben und " (x%)" anhängen
                   ~ paste0(.x, " (", round(.x / sum(df), 2) * 100, "%)")
                  )
            )
```

```
           A1       A2       A3
1     0 (0%)  2 (3%)  3 (5%)
2     1 (2%)  1 (2%)  1 (2%)
3     1 (2%)  2 (3%)  1 (2%)
4    6 (10%)  1 (2%)  4 (7%)
5     2 (3%)  4 (7%)  1 (2%)
ja   7 (12%) 6 (10%)  3 (5%)
nein  3 (5%)  4 (7%) 7 (12%)
```

3.1.2 Eine gemeinsame Kreuztabelle mit mehrere Variablen erstellen

Häufig erstellen wir verschiedene Kreuztabellen mit der selben "Gruppierungsvariable", z.B. `Geschlecht` oder `Kontroll-` vs. `Interventionsgruppe`.
Nehmen wir als Beispiel den Datensatz `pf8`.

```
load(url("https://www.produnis.de/R/data/pf8.RData"))
# oder
pf8 <- jgsbook::pf8
```

Möchten wir hinsichtlich der Variable `Geschlecht` unterscheiden, würden wir die Kreuztabellen mit anderen Variablen einzeln per `xtabs()` oder `table()` erstellen.

```
table(pf8$Standort, pf8$Geschlecht)
```

```
           männlich weiblich divers
  Rheine         90      112      1
  Münster        75      103      0
  Bahn           36       48      0
  Ladbergen      26       17      0
  Internet       60      157      0
```

```
table(pf8$Bildung, pf8$Geschlecht)
```

```
                männlich weiblich divers
  keinen               2        3      0
  Hauptschule         30       26      0
  mittlere Reife      46       53      0
  Ausbildung          10       19      0
  Fachabitur          23       42      0
  Abitur              64       76      0
  Hochschule          50       56      1
```

```
table(pf8$Familienstand, pf8$Geschlecht)
```

```
               männlich weiblich divers
  ledig             125      162      0
  Partnerschaft      56       98      0
  verheiratet        90      136      1
  geschieden          6       23      0
  verwitwet           5       13      0
  getrennt            4        4      0
```

Eine alternative Vorgehensweise besteht darin, mit der Funktion `lapply()` eine `list` der Kreuztabellen zu erstellen.

```
tbl.list <- lapply(pf8[, c("Standort", "Bildung", "Familienstand")],
                   function(x) xtabs(~ x + pf8$Geschlecht))

tbl.list
```

```
$Standort
           pf8$Geschlecht
x           männlich weiblich divers
  Rheine          90      112      1
  Münster         75      103      0
  Bahn            36       48      0
  Ladbergen       26       17      0
  Internet        60      157      0

$Bildung
                pf8$Geschlecht
x                männlich weiblich divers
  keinen                2        3      0
  Hauptschule          30       26      0
  mittlere Reife       46       53      0
  Ausbildung           10       19      0
  Fachabitur           23       42      0
```

```
  Abitur               64       76      0
  Hochschule           50       56      1

$Familienstand
               pf8$Geschlecht
x               männlich weiblich divers
  ledig              125      162      0
  Partnerschaft       56       98      0
  verheiratet         90      136      1
  geschieden           6       23      0
  verwitwet            5       13      0
  getrennt             4        4      0
```

Wir haben Zugriff auf die einzelnen Kreuztabellen per `tbl.list[[1]]` oder `tbl.list[[2]]`.

```
1 tbl.list[[3]]
```

```
               pf8$Geschlecht
x               männlich weiblich divers
  ledig              125      162      0
  Partnerschaft       56       98      0
  verheiratet         90      136      1
  geschieden           6       23      0
  verwitwet            5       13      0
  getrennt             4        4      0
```

3.1.2.1 Zeilen- und Spaltensummen

Mit der Liste können wir nun weiterarbeiten. Zum Beispiel können wir Spalten- und Zeilensummen hinzufügen...

```
1 lapply(tbl.list, addmargins)
```

```
               pf8$Geschlecht
x               männlich weiblich divers
  ledig              125      162      0
  Partnerschaft       56       98      0
  verheiratet         90      136      1
  geschieden           6       23      0
  verwitwet            5       13      0
  getrennt             4        4      0
> lapply(tbl.list, addmargins)
$Standort
           pf8$Geschlecht
x           männlich weiblich divers Sum
  Rheine          90      112      1 203
  Münster         75      103      0 178
  Bahn            36       48      0  84
  Ladbergen       26       17      0  43
  Internet        60      157      0 217
  Sum            287      437      1 725

$Bildung
                pf8$Geschlecht
x                männlich weiblich divers Sum
  keinen                2        3      0   5
  Hauptschule          30       26      0  56
  mittlere Reife       46       53      0  99
  Ausbildung           10       19      0  29
```

```
  Fachabitur           23       42      0  65
  Abitur               64       76      0 140
  Hochschule           50       56      1 107
  Sum                 225      275      1 501

$Familienstand
               pf8$Geschlecht
x               männlich weiblich divers Sum
  ledig              125      162      0 287
  Partnerschaft       56       98      0 154
  verheiratet         90      136      1 227
  geschieden           6       23      0  29
  verwitwet            5       13      0  18
  getrennt             4        4      0   8
  Sum                286      436      1 723
```

3.1.2.2 Prozenttabellen

... oder Prozenttabellen erstellen. Um die Prozentwerte *zeilenweise* zu bilden, muss der Parameter `margin=1` angegeben werden. Für *spaltenweises* Vorgehen gilt `margin=2`.

```
1 lapply(tbl.list, prop.table, margin=1)
```

```
$Standort
           pf8$Geschlecht
x             männlich    weiblich      divers
  Rheine    0.443349754 0.551724138 0.004926108
  Münster   0.421348315 0.578651685 0.000000000
  Bahn      0.428571429 0.571428571 0.000000000
  Ladbergen 0.604651163 0.395348837 0.000000000
  Internet  0.276497696 0.723502304 0.000000000

$Bildung
                pf8$Geschlecht
x                   männlich    weiblich      divers
  keinen         0.400000000 0.600000000 0.000000000
  Hauptschule    0.535714286 0.464285714 0.000000000
  mittlere Reife 0.464646465 0.535353535 0.000000000
  Ausbildung     0.344827586 0.655172414 0.000000000
  Fachabitur     0.353846154 0.646153846 0.000000000
  Abitur         0.457142857 0.542857143 0.000000000
  Hochschule     0.467289720 0.523364486 0.009345794

$Familienstand
               pf8$Geschlecht
x                  männlich    weiblich      divers
  ledig         0.435540070 0.564459930 0.000000000
  Partnerschaft 0.363636364 0.636363636 0.000000000
  verheiratet   0.396475771 0.599118943 0.004405286
  geschieden    0.206896552 0.793103448 0.000000000
  verwitwet     0.277777778 0.722222222 0.000000000
  getrennt      0.500000000 0.500000000 0.000000000
```

3.1.2.3 Signifikanztests

Zudem können wir Signifikanztests rechnen.

```
lapply(tbl.list, chisq.test)
```

```
$Standort
^^IPearson's Chi-squared test
data:  X[[i]]
X-squared = 26.296, df = 8, p-value = 0.0009347


$Bildung
^^IPearson's Chi-squared test
data:  X[[i]]
X-squared = 9.4526, df = 12, p-value = 0.6639


$Familienstand
^^IPearson's Chi-squared test
data:  X[[i]]
X-squared = 10.503, df = 10, p-value = 0.3975
```

Wenn wir die Liste in ein Objekt speichern, können wir uns nur die p-Werte ausgeben lassen.

```
chi.list <- lapply(tbl.list, chisq.test)
chi.list[[1]]$p.value
```

```
[1] 0.0009347078
```

3.1.2.4 eine große Kreuztabelle

Zu guter Letzt können wir die einzelnen Kreuztabellen zu einer ”großen”Tabelle vereinen.

```
dummy <- rbind(tbl.list[[1]],
      tbl.list[[2]],
      tbl.list[[3]])
# erzeuge ein Datenframe aus den Zeilennamen und den Werten
dummy <- data.frame( row.names(dummy), dummy )

# lösche die alten Rownames
rownames(dummy) <- NULL
dummy
```

```
   row.names.dummy. männlich weiblich divers
1            Rheine       90      112      1
2           Münster       75      103      0
3              Bahn       36       48      0
4         Ladbergen       26       17      0
5          Internet       60      157      0
6            keinen        2        3      0
7       Hauptschule       30       26      0
8    mittlere Reife       46       53      0
9         Ausbildung       10       19      0
10       Fachabitur       23       42      0
11           Abitur       64       76      0
12       Hochschule       50       56      1
13            ledig      125      162      0
```

```
14    Partnerschaft       56       98      0
15      verheiratet       90      136      1
16       geschieden        6       23      0
17        verwitwet        5       13      0
18          getrennt       4        4      0
```

3.1.3 Zeilen und Spalten tauschen

Ich möchte bei meinem Datensatz Zeilen und Spalten vertauschen.

Dies kann mit der Funktion `apply()` gemacht werden. Angenommen das Datenframe `datensatz` sieht wie folgt aus ...

```
# lade Testdatensatz
datensatz <- read.table(url("http://www.produnis.de/R/DieDaten.csv"), sep=";", header=TRUE)
# zeige "datensaz" an
datensatz
```

```
    Name   Geschlecht Lieblingsfarbe Einkommen
1   Hans    maennlich          gruen      1233
2   Caro     weiblich           blau       800
3   Lars intersexuell           gelb      2400
4   Ines     weiblich        schwarz      4000
5 Samira     weiblich           gelb       899
6  Peter    maennlich          gruen      1100
7  Sarah     weiblich           blau      1900
```

...so tauschen wir Zeilen und Spalten mittels

```
# vertausche Spalten und Zeilen
apply(datensatz, MARGIN=1, FUN=function(x) {x})
```

```
               [,1]        [,2]       [,3]       [,4]       [,5]        [,6]        [,7]
Name           "Hans"      "Caro"     "Lars"     "Ines"     "Samira"    "Peter"     "Sarah"
Geschlecht     "maennlich" "weiblich" "isexuell" "weiblich" "weiblich"  "maennlich" "weiblich"
Lieblingsfarbe "gruen"     "blau"     "gelb"     "schwarz"  "gelb"      "gruen"     "blau"
Einkommen      "1233"      " 800"     "2400"     "4000"     " 899"      "1100"      "1900"
```

Da die Funktion `apply()` eine Typkonversion in die Klasse `matrix` vornimmt, (und somit alle Datentypen auf den kleinsten gemeinsamen Nenner `character` zurückfallen, siehe die Anführungszeichen im Output) muss bei Bedarf zurück in die Klasse `data.frame` konvertiert werden.

```
# vertausche Spalten und Zeilen
as.data.frame(apply(datensatz, MARGIN=1, FUN=function(x) {x}))
```

```
                     V1       V2  V3       V4       V5       V6       V7
Name                Hans     Caro  Lars     Ines     Samira   Peter    Sarah
Geschlecht     maennlich weiblich  isexuell weiblich weiblich maennlich weiblich
Lieblingsfarbe     gruen     blau  gelb     schwarz  gelb     gruen    blau
Einkommen           1233      800  2400     4000     899      1100     1900
```

3.1.4 binäre Dummy-Variablen zusammenführen

Manchmal liegen Daten in Dummy-Variablen vor. So wird beispielsweise der Beruf einer Person nicht als Ausprägung der Variable `Beruf` gespeichert, sondern es gibt für jeden Beruf eine Dummy-Variable, die `Ja` oder `Nein` enthält, je nachdem, welchen Beruf die Person ausübt.

```r
# erzeuge Testdaten
df <- data.frame(
  id = 1:5,
  Busfahrer_in = factor(c("ja", "nein", "nein", "ja", "nein")),
  Aerztin = factor(c("nein", "ja", "nein", "nein", "nein")),
  Lehrer_in = factor(c("nein", "nein", "ja", "nein", "nein")),
  Ingenieur_in = factor(c("nein", "nein", "nein", "nein", "ja"))
)
# anzeigen
df
```

```
  id Busfahrer_in Aerztin Lehrer_in Ingenieur_in
1  1           ja    nein      nein         nein
2  2         nein      ja      nein         nein
3  3         nein    nein        ja         nein
4  4           ja    nein      nein         nein
5  5         nein    nein      nein           ja
```

Die Aufteilung in solche Dummy-Variablen wird beispielsweise gemacht, um logistische Regression zu berechnen.
In manchen Fällen kann es unerwünscht sein, solche Dummys im Datensatz zu haben.
Mit den Funktionen `mutate()` und `case_when()` können die Dummyeinträge in einer neuen Variable vereinigt werden. Das klappt allerdings nur, wenn sich die Dummyeinträge gegenseitig ausschließen (also keine Mehrfachauswahl).

```r
library(tidyverse)
# führe Dummys in "Beruf" zusammen
df %>%
  mutate(Beruf = case_when(
                    Busfahrer_in == "ja" ~ "Busfahrer_in",
                    Aerztin == "ja" ~ "Arzt/Ärztin",
                    Lehrer_in == "ja" ~ "Lehrer/in",
                    Ingenieur_in == "ja" ~ "Ingenieur/in")
    )

```

```
  id Busfahrer_in Aerztin Lehrer_in Ingenieur_in        Beruf
1  1           ja    nein      nein         nein Busfahrer_in
2  2         nein      ja      nein         nein  Arzt/Ärztin
3  3         nein    nein        ja         nein    Lehrer/in
4  4           ja    nein      nein         nein Busfahrer_in
5  5         nein    nein      nein           ja Ingenieur/in
```

3.2 Diagramme plotten

Ich möchte Diagramme plotten!

3.2.1 Alle R-Farben

Ich möchte alle Farben, die per `colors()` ausgegeben werden, in eine Übersichtsgrafik plotten.

```
# ggplot laden
library(ggplot2)

# Datenframe vorbereiten
d=data.frame(c=colors(),
             y=seq(0, length(colors())-1)%%66,
             x=seq(0, length(colors())-1)%/%66)
# Alle Farben plotten
p <- ggplot() +
  scale_x_continuous(name="", breaks=NULL, expand=c(0, 0)) +
  scale_y_continuous(name="", breaks=NULL, expand=c(0, 0)) +
  scale_fill_identity() +
  # weisse Boxen für Text
  geom_rect(data=d, mapping=aes(xmin=x,
                                xmax=x+1,
                                ymin=y,
                                ymax=y+1),
            fill="white") +
  # Farbboxen
  geom_rect(data=d, mapping=aes(xmin=x+0.05,
                                xmax=x+0.95,
                                ymin=y+0.5,
                                ymax=y+1,
                                fill=c)) +
  # Farbnamen
  geom_text(data=d, mapping=aes(x=x+0.5, y=y+0.5, label=c),
            colour="black", hjust=0.5, vjust=1, size=3)

# Plot ansehen
p
```

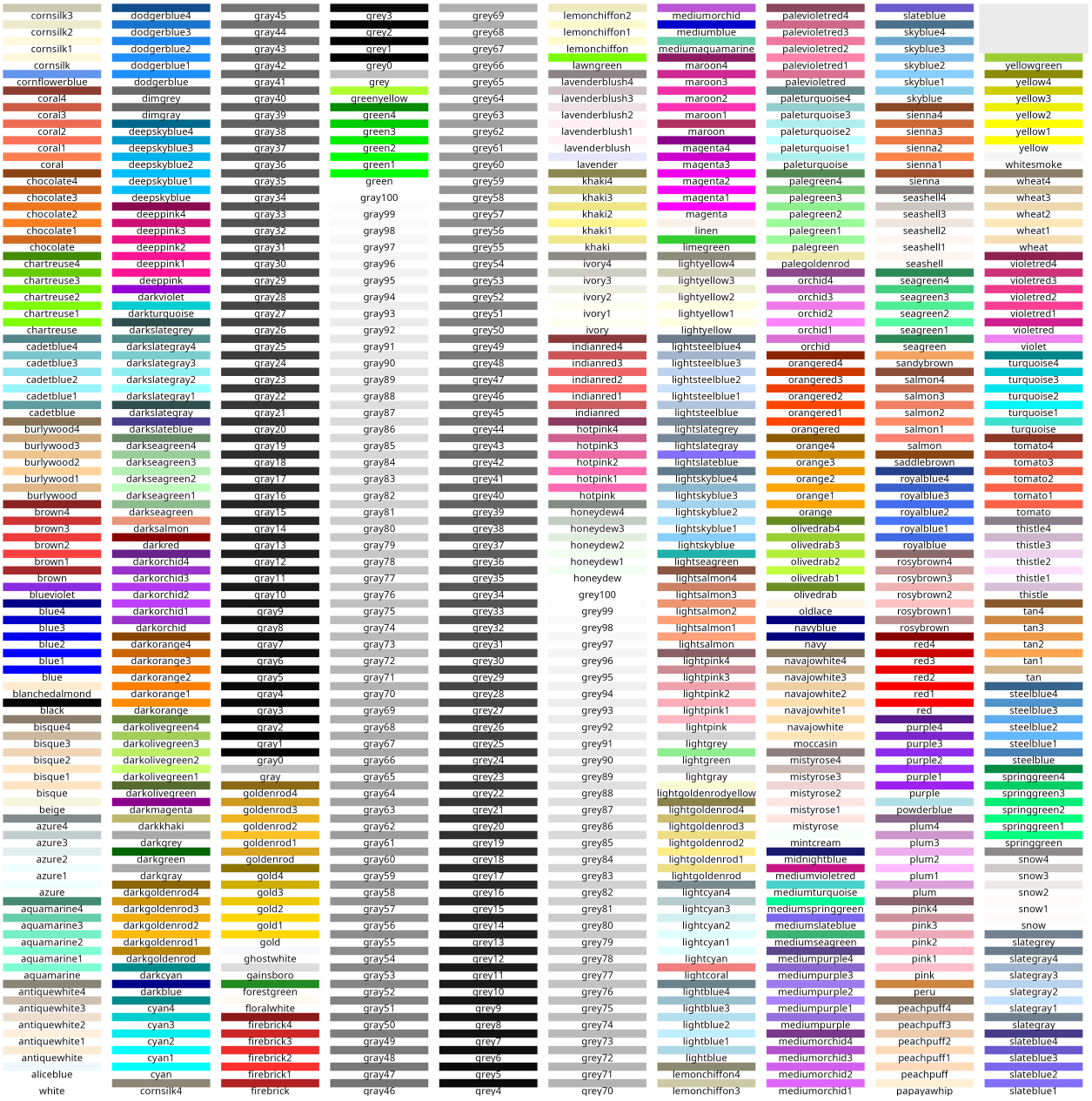


Abb. 3.1: Alle R Farben

```
# Plot speichern
ggsave("rbase-Farben.png",
       plot= p,
       units="px",
       width=4000,
       height=4000,
       dpi=300)
```

3.2.2 Normalverteilung

Zum Plotten von Normalverteilungen (siehe auch Abschnitt 2.3.1 auf Seite 344) kann man mit der Funktion `plot()` so vorgehen:

```
# erstelle Werte von -4 bis 4 in 0.005er-Schritten
x <- seq(-4, 4, by=0.005)

# plotte die Standardnormalverteilung
plot(x,dnorm(x))
```

Der Befehl erzeugt Abbildung 3.2.

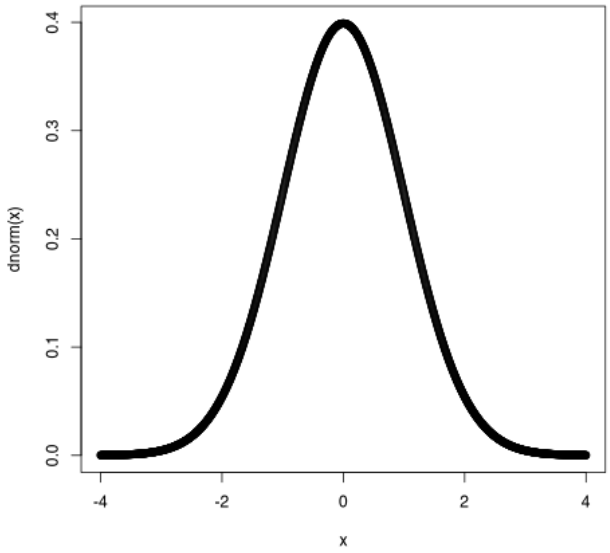


Abb. 3.2

```
# etwas hübscher
plot(x,dnorm(x), col="blue", type="l", xlab="x", ylab="f(x)", main="Standardnormalverteilungen")
```

Der Befehl erzeugt Abbildung 3.3.

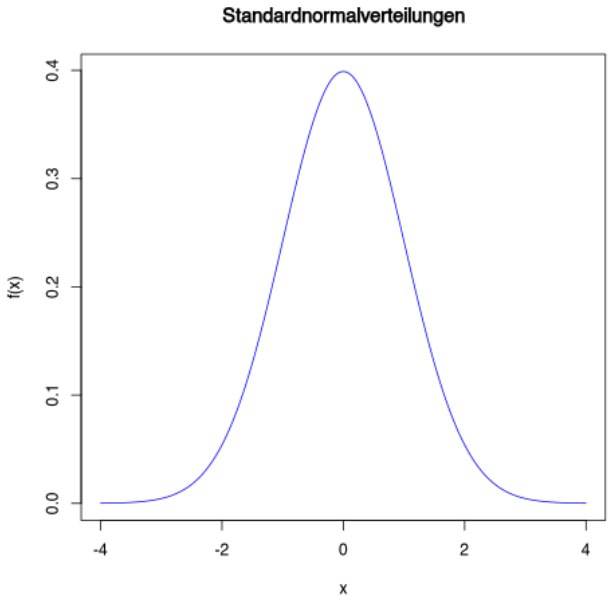


Abb. 3.3

```
# mit Mittelwert 2 und sd = 0,5
plot(x,dnorm(x,mean=2,s=0.5), col="darkblue", type="l", xlab="x", ylab="f(x)", main="Normalverteilungen")
```

Der Befehl erzeugt Abbildung 3.4.

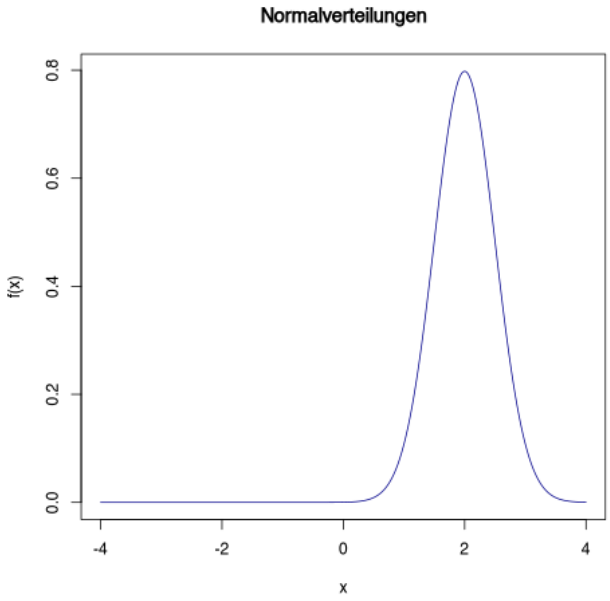


Abb. 3.4

```
# mit Mittelwert 2 und sd = 2
plot(x,dnorm(x,mean=2,s=2), col="darkorchid", type="l", xlab="x", ylab="f(x)", main="Normalverteilungen")
```

Der Befehl erzeugt Abbildung 3.5.

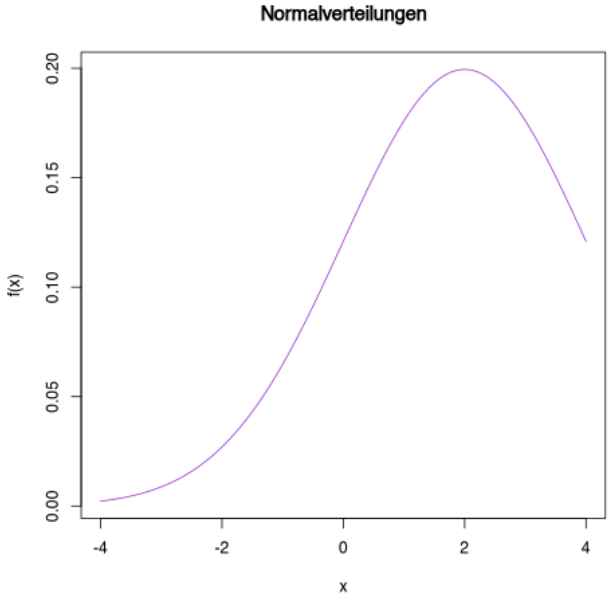


Abb. 3.5

```
# erzeuge neue Werte von -3 bis 6
x <- seq(-3,6, by=0.005)

# Alles zusammen plotten
plot(x,dnorm(x,mean=2,s=0.5), col="blue", type="l", xlab="x", ylab="f(x)",main="Normalverteilungen")
lines(x,dnorm(x,mean=0,s=1), col="black")
lines(x,dnorm(x,mean=2,s=2), col="darkorchid")
text(0,.45,"N(0;1)")
text(2.8, 0.6, "N(2;0.5)", col="blue")
text(5, 0.1, "N(2;2)", col="darkorchid")
```

Der Befehl erzeugt Abbildung 3.6.

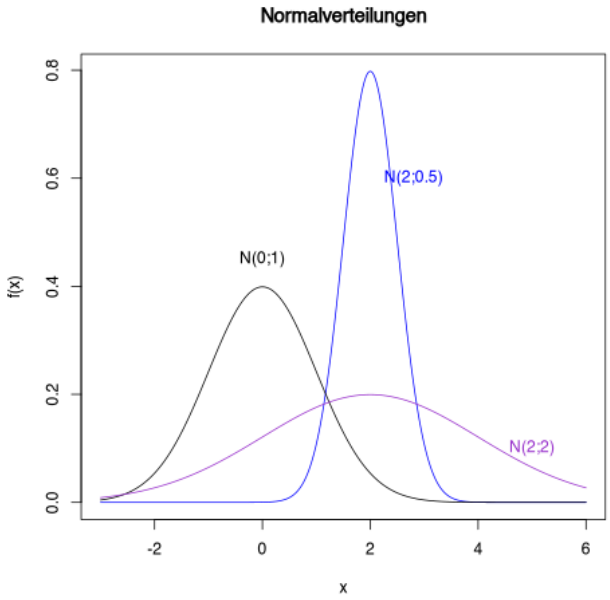


Abb. 3.6

Mit `ggplot` kann es so aussehen

```
# aktiviere ggplot
library(ggplot2)

# erzeuge Werte von -3 bis 3
x <- seq(-3,3, by=0.005)

# übergebe in ein data.frame
df <- data.frame(x)

# ggplot erstellen
p <- ggplot(data=df, aes(x)) +
                     xlim(-4,4) + ylim(0,0.5) +
                     xlab("x") + ylab("Dichtefunktion") +
                     ggtitle("Normalverteilungen")
p + stat_function(fun=dnorm, args=(c(mean=0,sd=1)), colour="black")+
    annotate(geom="text", x=0, y=0.42, label="N(0;1)", color="black")
```

Der Befehl erzeugt Abbildung 3.7.

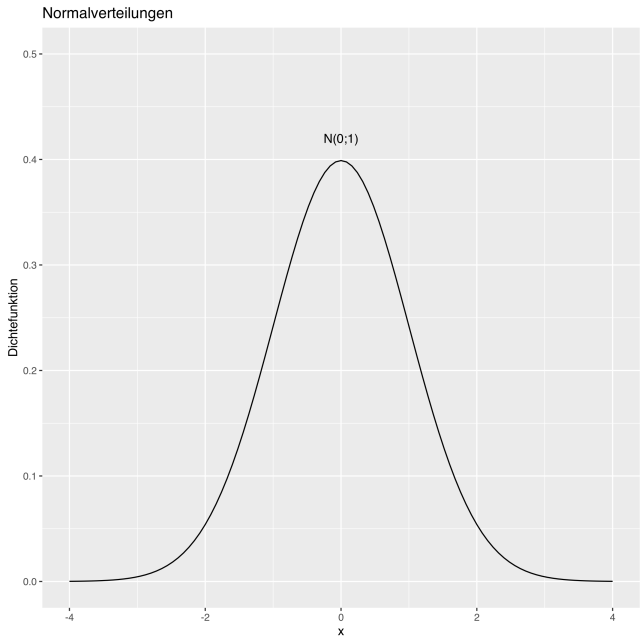


Abb. 3.7

```r
# erzeuge Werte von -3 bis 6
x <- seq(-3,6, by=0.005)

# übergebe in ein data.frame
df <- data.frame(x)

# ggplot erstellen
p <- ggplot(data=df, aes(x)) +
                     xlim(-3,6) + ylim(0, 0.8) +
                     xlab("x") + ylab("Dichtefunktion") +
                     ggtitle("Normalverteilungen")
p + stat_function(fun=dnorm, args=(c(mean=0,sd=1)), colour="black")+
  annotate(geom="text", x=0, y=0.42, label="N(0;1)", color="black")+
  stat_function(fun=dnorm, args=(c(mean=2,sd=0.5)), colour="blue") +
  annotate(geom="text", x=3, y=0.6, label="N(2;0.5)", color="blue")+
  stat_function(fun=dnorm, args=(c(mean=2,sd=2)), colour="darkorchid") +
  annotate(geom="text", x=5, y=0.1, label="N(2;2)", color="darkorchid")
```

Der Befehl erzeugt Abbildung 3.8.

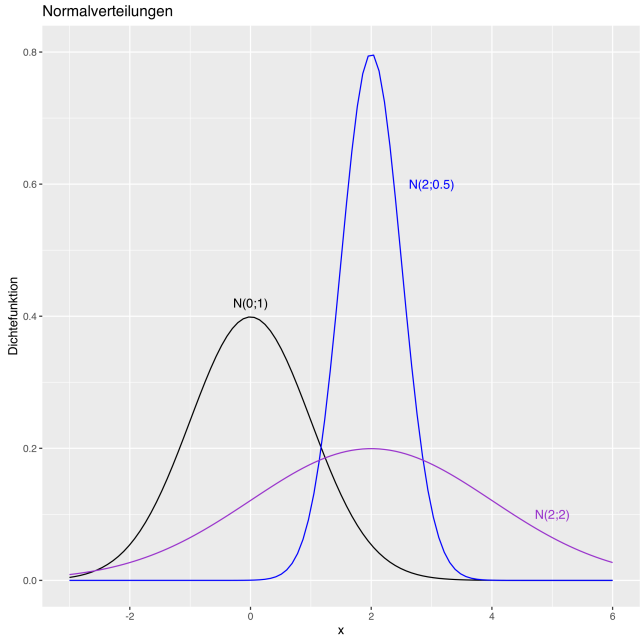


Abb. 3.8

Die Bereiche der Standardabweichung können mit `geom_area()` visualisiert werden.

```
# erzeuge X-Werte
df    <- data.frame(x=seq(-3,3, by=0.005))
# berechne Y-Werte
df$y  <- dnorm(df$x)
# Setze Cuts
df$sd <- cut(df$x, breaks = c(-Inf, 1 *(-2:2), Inf))

# plotte als area
ggplot(df, aes(x, y, fill=sd)) + geom_area()
```

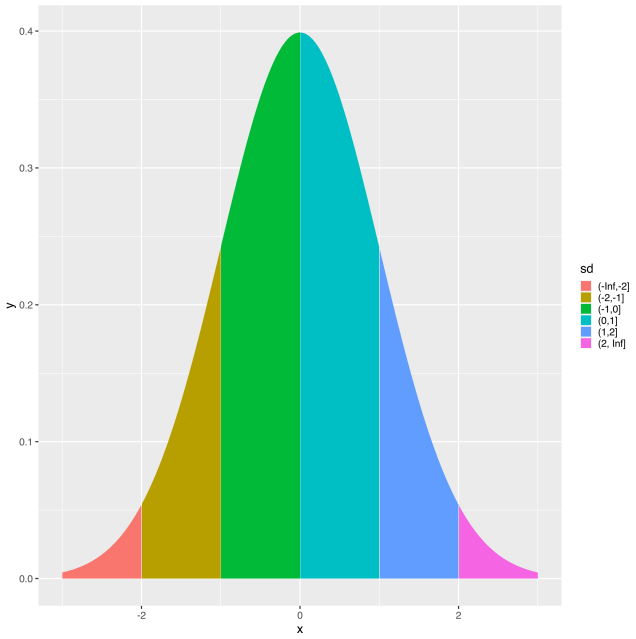


Abb. 3.9

Erzeugen wir den Bereich von 1 Standardabweichung.

```
df    <- data.frame(x=seq(-3,3, by=0.005))
df$y  <- dnorm(df$x)
df$sd <- cut(df$x, breaks = c(-1,1))
df <- rbind(df, data.frame(x=c(-1, 1), y=c(0,0),sd=c(-1,1)))
df$sd <- forcats::fct_explicit_na(df$sd, na_level="x")

ggplot(df, aes(x, y, fill=sd)) +
  ggtitle("Standardnormalverteilung", subtitle = "Bereich von 1 Standardabweichung") +
  geom_area() + theme(legend.position = "none") +
  geom_vline(xintercept=0, linetype="dotted")+
  geom_vline(xintercept=-1, linetype="dashed", col="blue", size=1)+
  geom_vline(xintercept=1, linetype="dashed", col="blue", size=1)+
  scale_fill_manual(name=c("x","(-1,1]"), values=c("skyblue", "snow3"))
```

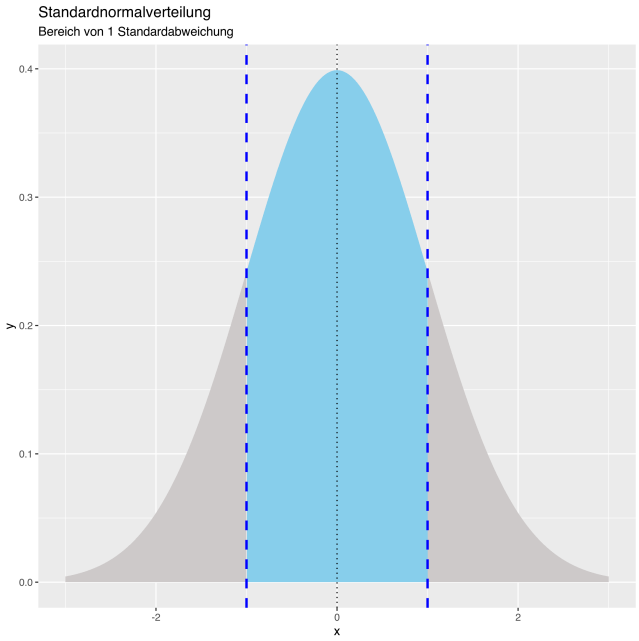


Abb. 3.10

Nun erzeugen wir den Bereich von 2 Standardabweichungen.

```
df    <- data.frame(x=seq(-3,3, by=0.005))
df$y  <- dnorm(df$x)
df$sd <- cut(df$x, breaks = c(-2,2))
df    <- rbind(df, data.frame(x=c(-2, 2), y=c(0,0),sd=c(-2,2)))
df$sd <- forcats::fct_explicit_na(df$sd, na_level="x")

ggplot(df, aes(x, y, fill=sd)) +
  ggtitle("Standardnormalverteilung", subtitle = "Bereich von 2 Standardabweichung") +
  geom_area() + theme(legend.position = "none") +
  geom_vline(xintercept=0, linetype="dotted")+
  geom_vline(xintercept=-2, linetype="dashed", col="purple3", size=1)+
  geom_vline(xintercept=2, linetype="dashed", col="purple3", size=1)+
  scale_fill_manual(name=c("x","(-2,2]"), values=c("purple3", "snow3"))
```

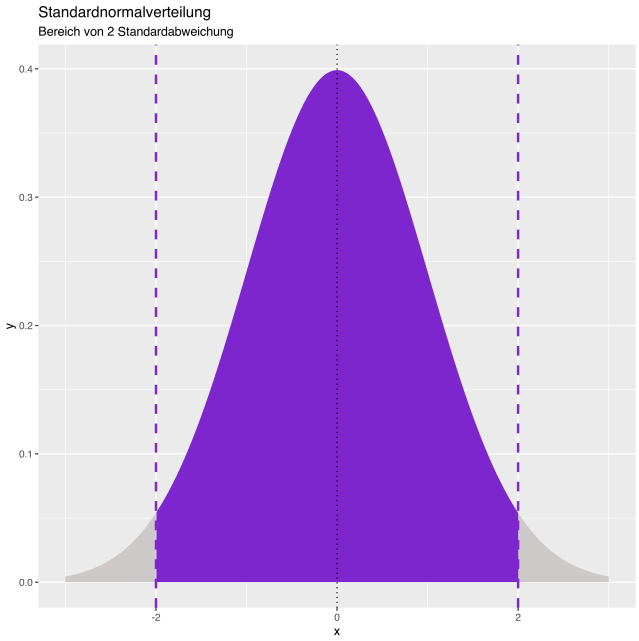


Abb. 3.11

Erzeugen wir die Fläche, in der 95% der Werte liegen, also von 1,96 Standardabweichungen.

```
df    <- data.frame(x=seq(-3,3, by=0.005))
df$y  <- dnorm(df$x)
df$sd <- cut(df$x, breaks = c(-1.96, 1.96))
df    <- rbind(df, data.frame(x=c(-1.96, 1.96), y=c(0,0),sd=c(-1.96,1.96)))
df$sd <- forcats::fct_explicit_na(df$sd, na_level="x")

ggplot(df, aes(x, y, fill=sd)) +
  ggtitle("Standardnormalverteilung", subtitle = "95% der Werte bei 1,96 sd") +
  geom_area() + theme(legend.position = "none") +
  geom_vline(xintercept=0, linetype="dotted")+
  geom_vline(xintercept=-2, linetype="dotted", col="purple3", size=0.5)+
  geom_vline(xintercept=2, linetype="dotted", col="purple3", size=0.5)+
  geom_vline(xintercept=-1.96, col="chartreuse4", size=0.5)+
  geom_vline(xintercept=1.96, col="chartreuse4", size=0.5)+
  scale_fill_manual(name=c("x","(-1.96,1.96]"), values=c("seagreen3", "snow3"))
```

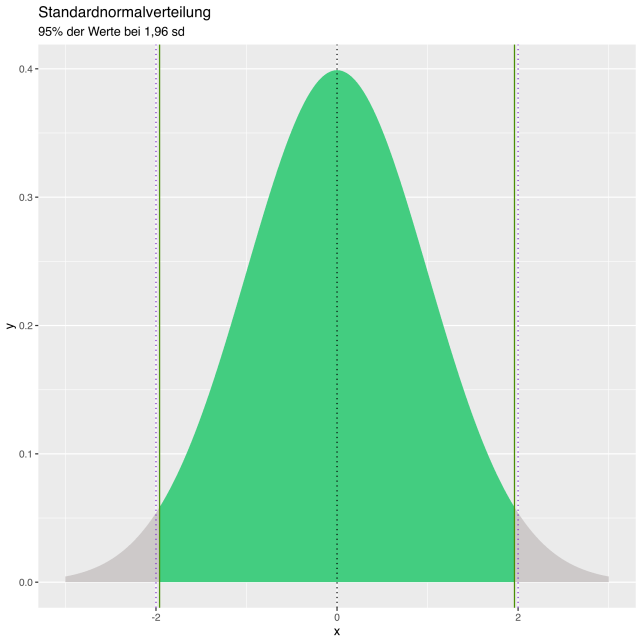


Abb. 3.12

Dieser Code kann für die Erzeugung der Grafiken von Tabelle 3.3 und 3.4 verwendet werden.

```
df    <- data.frame(x=seq(-3,3, by=0.005))
df$y  <- dnorm(df$x)
df$sd <- "B"
df$sd[df$x < 1.4] <- "A"

ggplot(df, aes(x, y, fill=sd)) +
  geom_area() + theme(legend.position = "none") +
  geom_vline(xintercept=0, linetype="dotted")+
  geom_vline(xintercept=1.4, col="blue", size=0.5)+
  ylab("") + scale_fill_manual(values=c("skyblue", "snow3"))
```

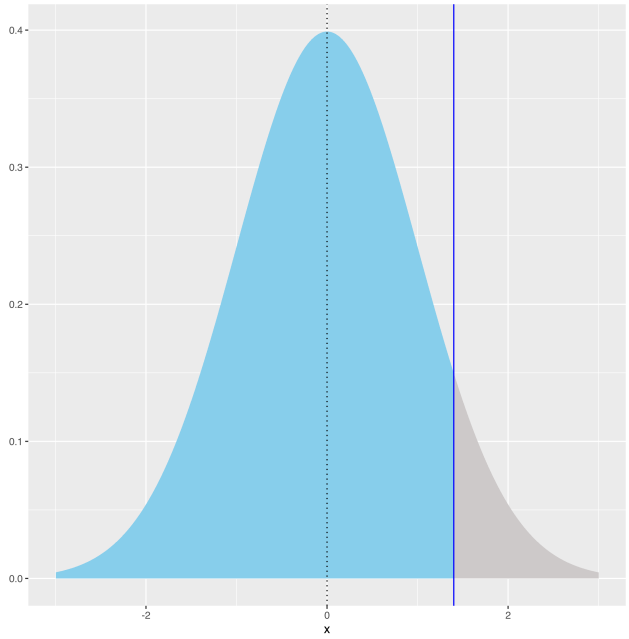


Abb. 3.13

```
df    <- data.frame(x=seq(-3,3, by=0.005))
df$y  <- dnorm(df$x)
df$sd <- "B"
df$sd[df$x < 1.4] <- "A"

ggplot(df, aes(x, y, fill=sd)) +
  geom_area() + theme(legend.position = "none") +
  geom_vline(xintercept=0, linetype="dotted")+
  geom_vline(xintercept=1.4, col="blue", size=0.5)+
  ylab("") + scale_fill_manual(values=c("snow3", "skyblue"))
```

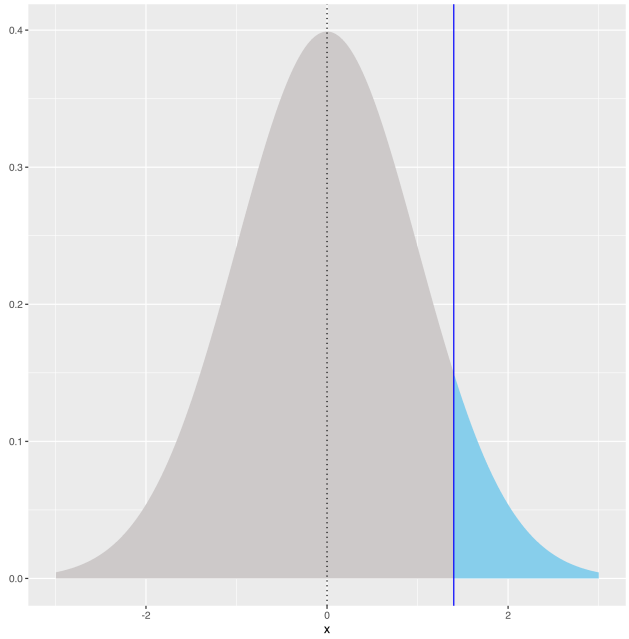


Abb. 3.14

3.2.3 t-Verteilung

Die t-Verteilung kann mit `ggplot` geplottet werden.

```
# Erzeuge x-werte
df <- data.frame(x=seq(-3,3, by=0.005))

# Grundlegene Plotangaben
p <- ggplot(data=df, aes(x)) +
  # begrenze die Achsen
  xlim(-3,3) + ylim(0, 0.4) +
  # Achsen-Titel
  xlab("x") + ylab("Dichtefunktion") +
  # Plot-Titel
  ggtitle("t-Verteilungen", subtitle = "nach Freiheitsgraden")

  # t-Verteilung plotten
p +
  stat_function(fun=dt, args=list(df=1), col="black") +
  # Textfeld hinzufügen
  annotate(geom="text", x=0, y=0.25, label="df=1", color="black")
```

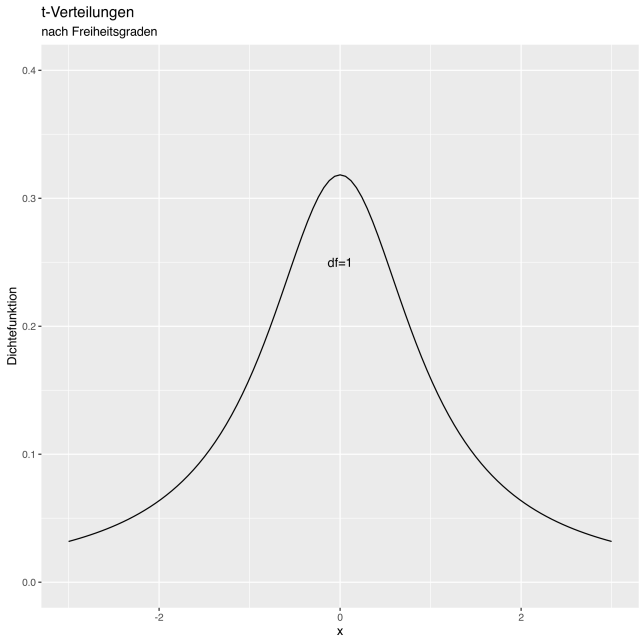


Abb. 3.15

Dem Plott können weitere Freiheitsgrade hinzugefügt werden.

```
# t-Verteilungen plotten
p +
  stat_function(fun=dt, args=list(df=1), col="black") +
  annotate(geom="text", x=0, y=0.25, label="df=1", color="black")+
  stat_function(fun=dt, args=list(df=2), col="blue") +
  annotate(geom="text", x=0, y=0.37, label="df=2", color="blue")
```

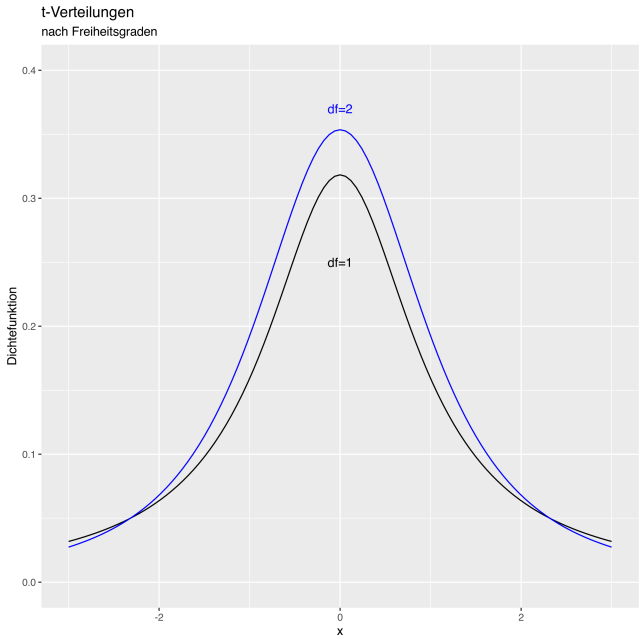


Abb. 3.16

Wenn die t-Werte bereits berechnet wurden, kann eine alternative Vorgehensweise so aussehen:

```r
# berechne t-Werte für Freiheitsgrade 1 bis 5
x=seq(-3,3, by=0.005)
df <- data.frame(
    x,
    df1 = dt(x,df=1),
    df2 = dt(x,df=2),
    df3 = dt(x,df=3),
    df4 = dt(x,df=4),
    df5 = dt(x,df=5)
)

# wandle ins Format "long table" um
df <- pivot_longer(df, cols=c(df1, df2, df3, df4, df5))

# grundlegende Ploteinstellungen
p <- ggplot(data=df, aes(x,value)) +
  xlim(-3,3) + ylim(0, 0.4) +
  xlab("x") + ylab("Dichtefunktion") +
  ggtitle("t-Verteilungen", subtitle = "nach Freiheitsgraden")

# plotte t-Verteilungen
p + geom_line(aes(col=name))+
  labs(col="Freiheitsgrade")
```

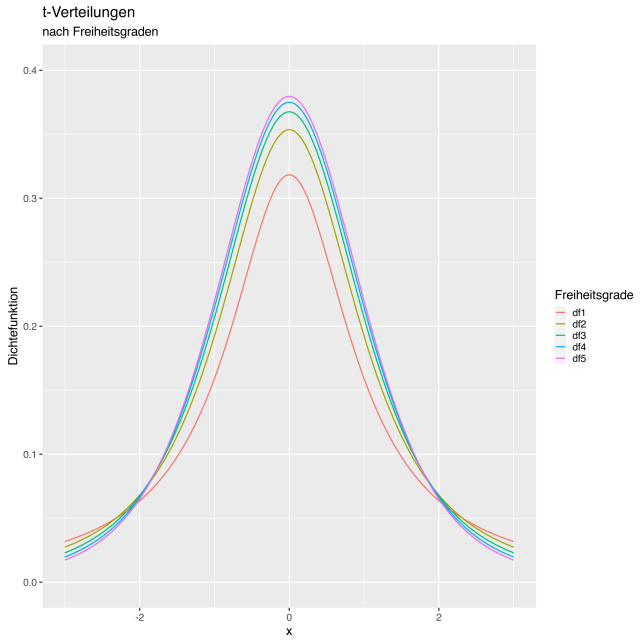


Abb. 3.17

3.2.4 χ^2-Verteilung

Die χ^2-Verteilung kann mit `ggplot` geplottet werden.

```
# Erzeuge x-werte
x=seq(0,25, by=0.005)
df <- data.frame(  x,
  df01 = dchisq(x, df=1),
  df05 = dchisq(x, df=5),
  df10 = dchisq(x, df=10),
  df15 = dchisq(x, df=15)
)

# erzeuge long-table
df <- pivot_longer(df, cols=c(df01, df05, df10, df15))

p <- ggplot(data=df, aes(x, value, fill=name)) +
  xlim(0,25) + ylim(0, 0.2) +
  xlab("x") + ylab("Dichtefunktion") +
  ggtitle("Chi^2-Verteilungen", subtitle = "nach Freiheitsgraden")
p + geom_line(aes(col=name,linetype=name))+
labs(col="Freiheitsgrade",linetype="")
```

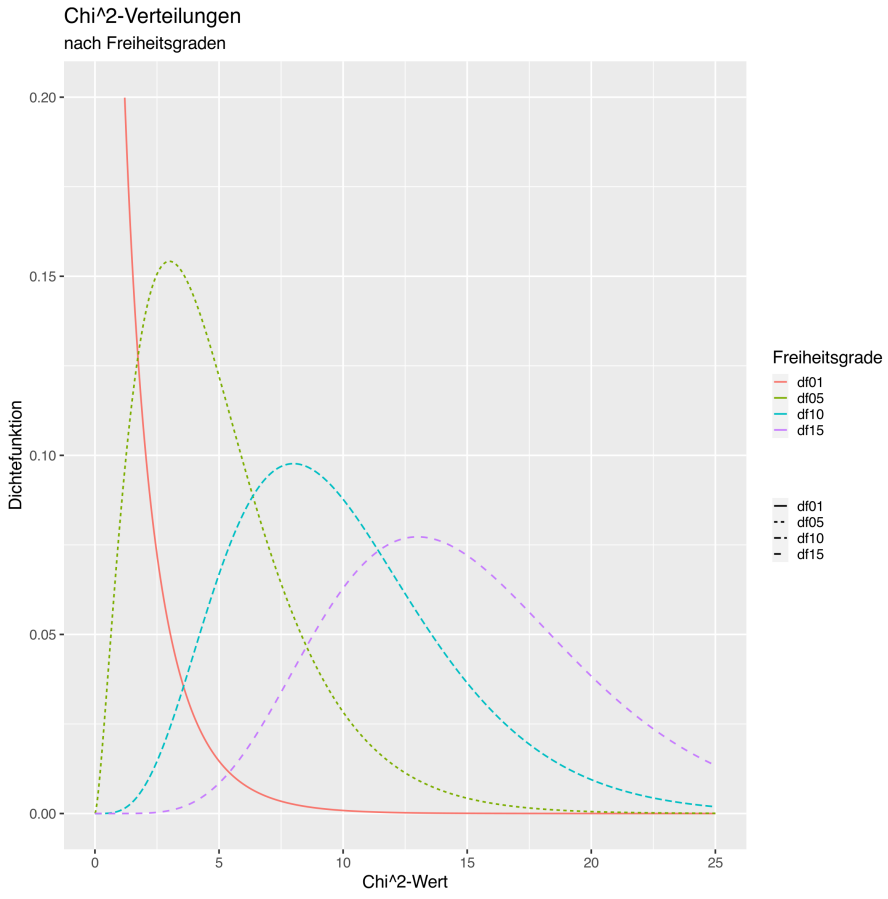


Abb. 3.18

Die Fläche unterhalb der Kurve kann mit `geom_area()` erzeugt werden. Für `df=1` lautet der Aufruf:

```
# erzeuge Dummy-Werte
x=seq(0,25, by=0.005)
# überführe in Datenframe
df <- data.frame(  x, df01 = dchisq(x, df=1))
ggplot(data=df, aes(x, df01)) +
  xlim(0,25)  + coord_cartesian(ylim=c(0, 0.2)) +
  xlab("Chi^2-Wert") + ylab("Dichtefunktion") +
  ggtitle("Chi^2-Verteilung", subtitle = "mit 1 Freiheitsgrad") +
  geom_line(col="#F8766D") +geom_area(fill="skyblue")
```

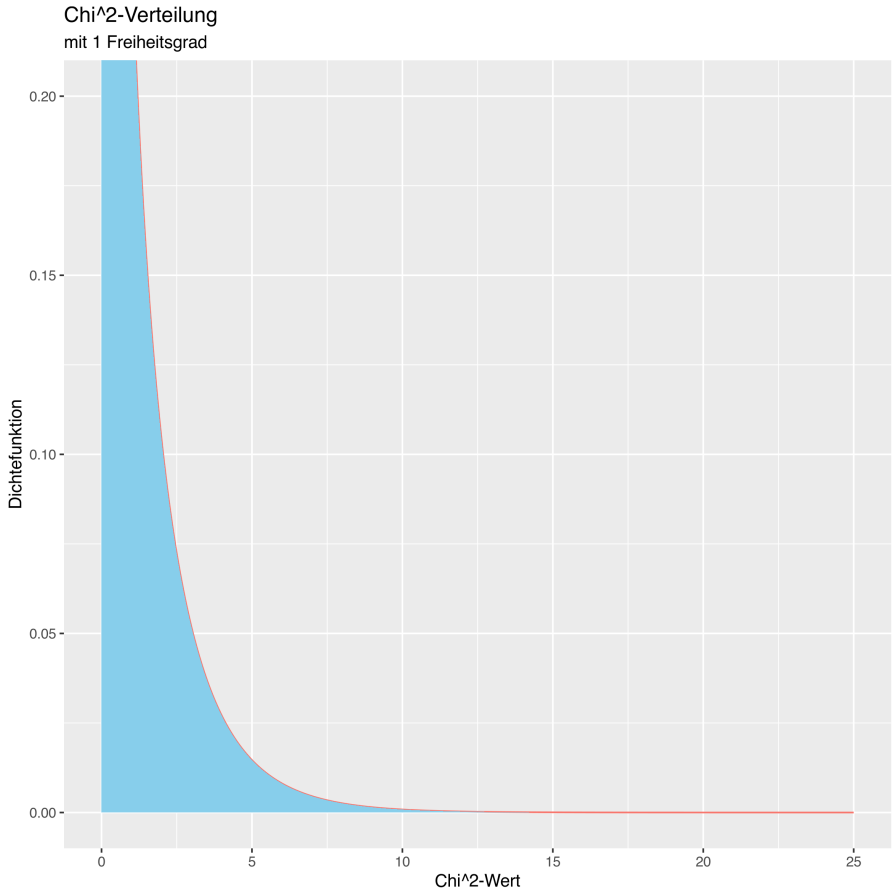


Abb. 3.19

Ähnlich wie bei der Normalverteilung können wir die Fläche für bestimmte χ^2-Werte einfärben. Bei einem Freiheitsgrad und $\alpha = 0,05$ ergibt sich ein kritischer χ^2-Wert von 3,84. Der Flächenanteil unterhalb dieses Wertes lässt sich wie folgt darstellen:

```
# Dummy-Werte erzeugen
df <- data.frame(x=seq(0.005,25, by=0.005))
# Chi^2-Werte für df=1 erstellen
df$y  <- dchisq(df$x, df=1)
# Grenze bei 3.84 einziehen
df$sd <- cut(df$x, breaks = c(0.00, 3.84))
ggplot(df, aes(x, y, fill=sd)) +
  geom_area() + theme(legend.position = "none") +
  geom_line(aes(col="#F8766D")) +
  xlim(0,25) + coord_cartesian(ylim=c(0, 0.2)) +
  xlab("Chi^2-Wert") + ylab("Dichtefunktion") +
  ggtitle("Chi^2-Verteilung", subtitle = "mit 1 Freiheitsgrad")+
  geom_vline(xintercept=3.84, col="blue", size=0.5, linetype="dashed")+
  ylab("") + scale_fill_manual(values=c("skyblue", "snow3")) +
  annotate(geom="text", x=2, y=0.028, size=8,label="95%", color="blue")
```

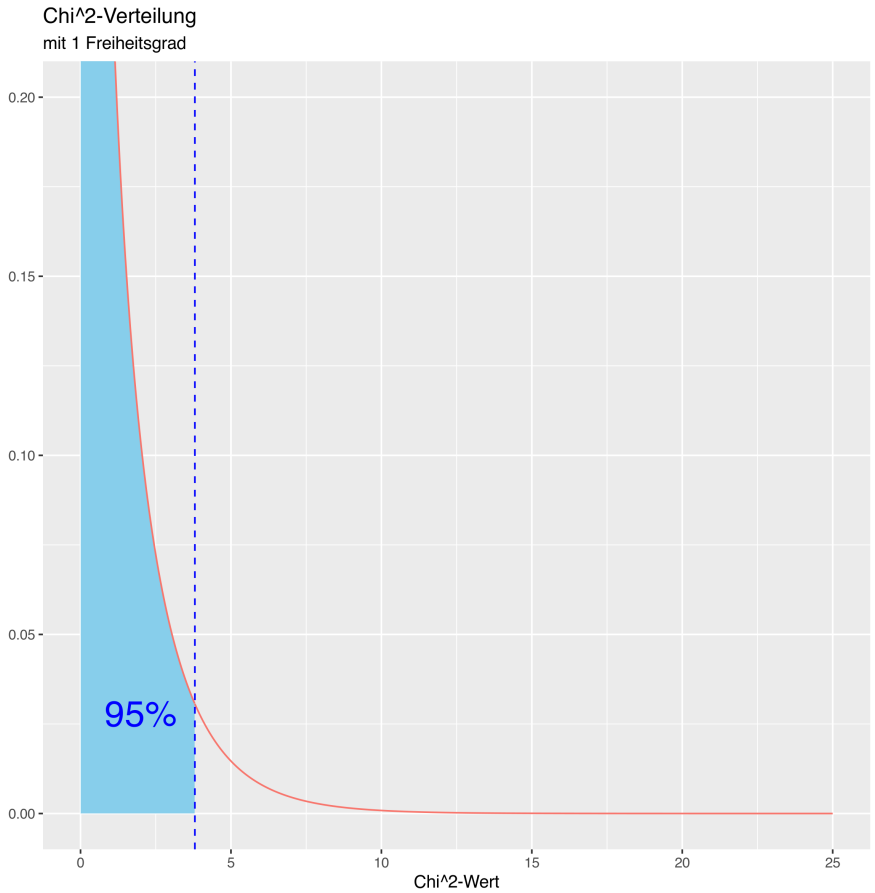


Abb. 3.20

3.2.5 Poisson-Verteilung

Die Poisson-Verteilung kann mit `ggplot` geplottet werden.

```
# Erzeuge x-Werte
x=seq(0,25)
df <- data.frame(  x,
                   l1 = dpois(x, 1),
                   l2 = dpois(x, 2),
                   l4 = dpois(x, 4),
                   l9 = dpois(x, 9)
)

# erzeuge eine long-table
df <- pivot_longer(df, cols=c(l1, l2, l4, l9))

# plot vorbereiten
p <- ggplot(data=df, aes(x, value, fill=name)) +
  xlim(0,17) + ylim(0, 0.4) +
  xlab("x") + ylab("Dichtefunktion") +
  ggtitle("Poisson-Verteilungen", subtitle = "nach Lambda")
p + geom_line(aes(col=name,linetype=name))+
  labs(col="Lambda",linetype="")
```

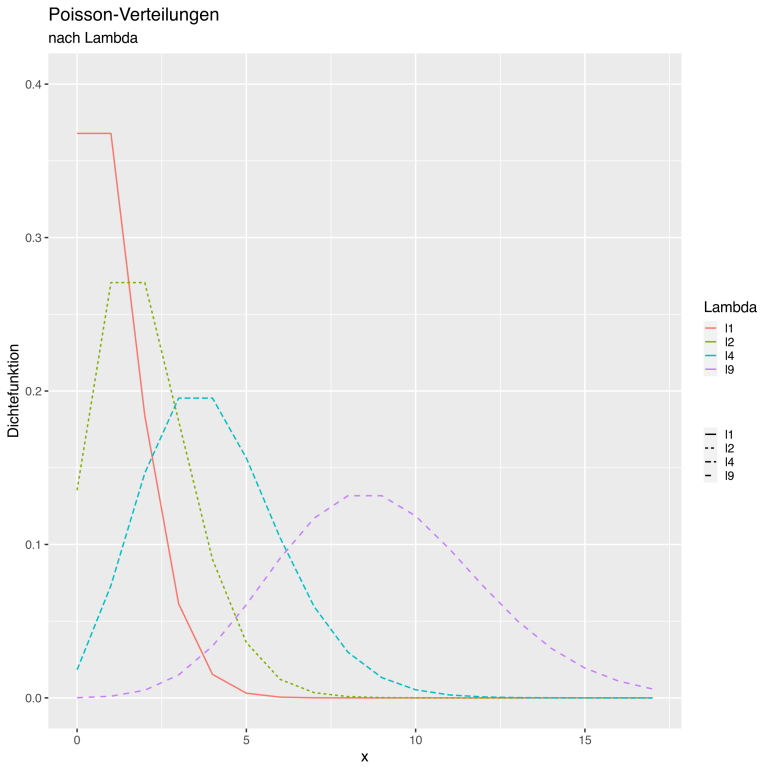


Abb. 3.21

Da wir die Plot-Grundlagen in `p` gespeichert haben, können wir ergänzen:

```
p + geom_bar(aes(col=name,linetype=name), stat="identity", position="dodge")+
    geom_line(aes(col=name,linetype=name))+
  labs(col="Lambda",linetype="")
```

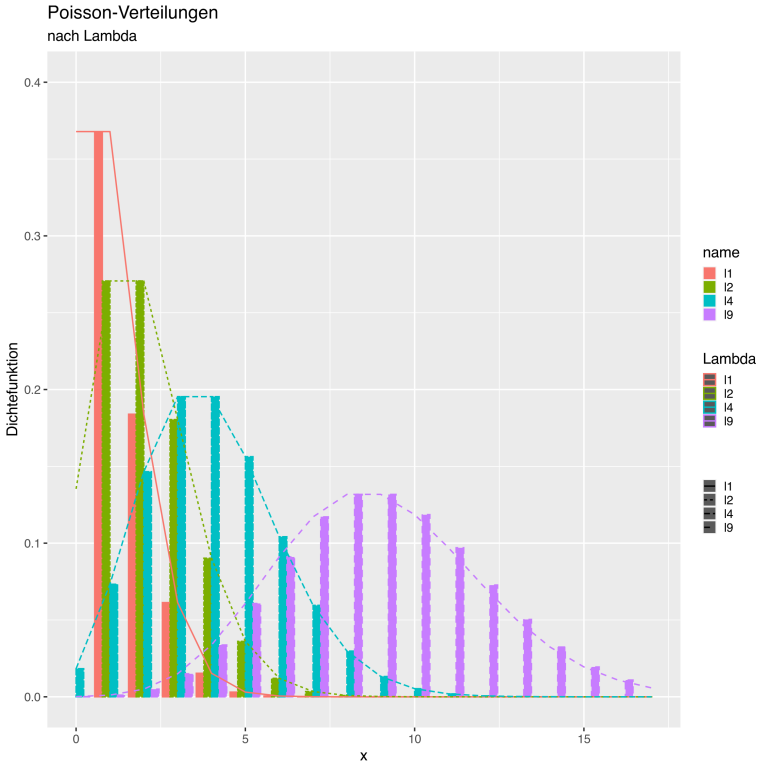


Abb. 3.22

Möchte man einen Polygonzug, müssen als Ausgangspunkt die Koordinaten (0,0) festegelegt werden.

```
x=seq(0,25)#, by=0.005)
df <- data.frame(  x=c(0,x),
                   l1 = c(0, dpois(x, 1)),
                   l2 = c(0, dpois(x, 2)),
                   l4 = c(0, dpois(x, 4)),
                   l9 = c(0, dpois(x, 9))
)
df <- pivot_longer(df, cols=c(l1, l2, l4, l9))

p <- ggplot(data=df, aes(x, value, fill=name)) +
  xlim(0,17) + ylim(0, 0.4) +
  xlab("x") + ylab("Dichtefunktion") +
  ggtitle("Poisson-Verteilungen", subtitle = "nach Lambda")
p + geom_polygon(aes(col=name,linetype=name))+
  geom_line(aes(col=name),linetype="dashed")+
  labs(col="Lambda", fill="Lambda", linetype="Lambda")
```

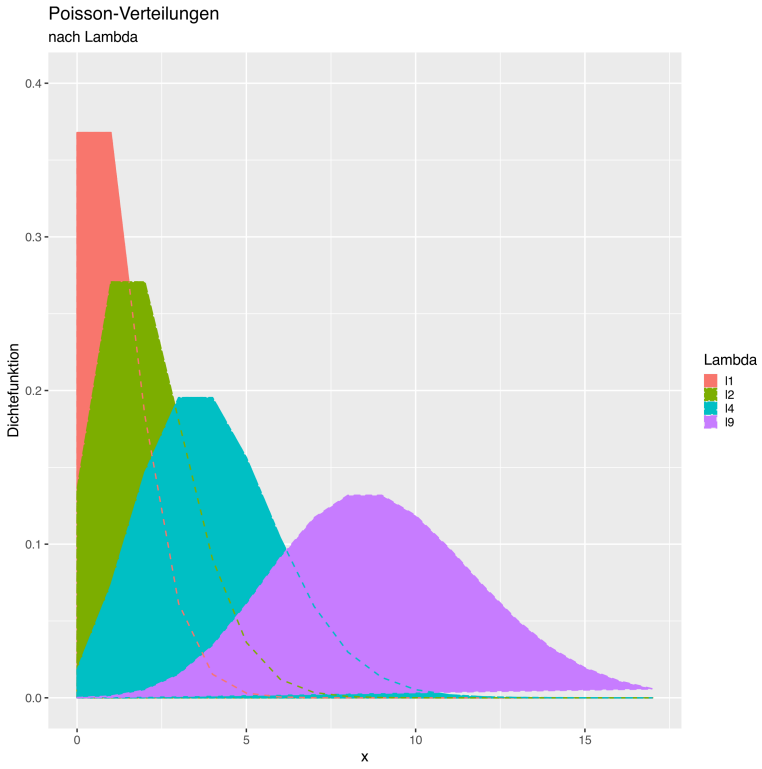


Abb. 3.23

3.2.6 BMI-Gewichtskategorien

Wir benötigen eine Übersichtsgrafik, aus der hervorgeht, in welcher BMI-Klasse sich ein Patient in Abhängigkeit von Körpergröße und Körpergewicht befindet.
Erzeugen wir zunächst unsere Daten.

```
# BMI-Gewichtsklassen
Gewichtsklassen <- c(0, 18.5, 25, 30, 35, 40, 100)
# Schöne Farben
Farben <- c("skyblue4", "darkgreen", "yellow", "orange", "red", "darkred", "black")
# 100 Körpergrößen von 1,4m bis 2,2m
Koerpergroesse <- seq(1.4, 2.2, length = 100)

# Funktion um das BMI-Grenzgewicht pro Klasse
# für jede Körpergröße zu bestimmen
bmi.k <- function(groesse, konstant) {
  return(groesse^2 * konstant)
}
```

3.2.6.1 R base

Wir erzeugen das Diagramm mit der `plot()`-Funktion, wobei wir die Klassen als Polygone einzeichnen.
Ein Polygon wird so gezogen, als würden wir es mit einem Stift auf Papier zeichnen, ohne den Stift dabei abzusetzen. Das heisst, wir müssen "hin und zurück" zeichnen. Wir beginnen unten links, zeichnen von dort nach rechts, dann nach oben, und von dort wieder zurück nach links und wieder hinunter.
Für die Körpergröße (x-Achse) bedeutet das, dass wir die Körpergrößen in umgekehrter Reihenfolge an die "Originalreihe" kleben müssen, um wieder "zurück" zum Ausgangspunkt zu kommen.

```
Gross <- c(Koerpergroesse, rev(Koerpergroesse))
```

Für die Klassengrenzen (y-Achse) benötigen wir für das "Zurückkehren" die Werte der nächsten Klasse in umgekehrter Reihenfolge.

```
# Berechne alle Klassengrenzen
Klasse1 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[1]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[2])))
Klasse2 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[2]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[3])))
Klasse3 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[3]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[4])))
Klasse4 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[4]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[5])))
Klasse5 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[5]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[6])))
Klasse6 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[6]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[7])))
Klasse7 <- c(bmi.k(Koerpergroesse, Gewichtsklassen[7]), rev(bmi.k(Koerpergroesse, Gewichtsklassen[8])))
```

Jetzt können wir das Diagramm plotten.

```
# Plotparameter festlegen
par(bg="whitesmoke")
plot(Koerpergroesse, bmi.k(Koerpergroesse, 18), type="n",
     xlim=c(1.60, 1.99), ylim=c(40, 125),
     xaxt="n", yaxt="n",
```

```
6       cex.axis=1.4, cex.lab=1.3, cex.main=1.7,
7       xlab="Größe [in m]", ylab="Gewicht [in kg]", main="Body Mass Index")
8
9   # Polygone der Gewichtsklassen einzeichnen
10  polygon(Gross, Klasse1, col=Farben[1])
11  polygon(Gross, Klasse2, col=Farben[2])
12  polygon(Gross, Klasse3, col=Farben[3])
13  polygon(Gross, Klasse4, col=Farben[4])
14  polygon(Gross, Klasse5, col=Farben[5])
15  polygon(Gross, Klasse6, col=Farben[6])
16  polygon(Gross, Klasse7, col=Farben[7])
17
18  # graues Gitter
19  box()
20  grid(lty="dotdash" ,col="darkgrey")
21  abline(v=seq(1.65, 1.95, by=0.1), h=seq(50, 110, by=20), lty="dotted", col="grey")
22
23  # Legendenbox
24  legend(x="bottomright", inset=0.005,
25         legend=c("Untergewicht", "Normalgewicht", "Präadipositas",
26                  "Adipositas Grad I", "Adipositas Grad II", "Adipositas Grad III"),
27         col=Farben, lwd=6, bg="skyblue")
28
29  # X-Achse in Groß
30  axis(1, at=format(seq(1.60, 2, by=0.1), nsmall=2),
31       labels=format(seq(1.60, 2, by=0.1), nsmall=2), cex.axis=1.5)
32  # X-Achse Zwischenschritte in Klein und Grau
33  axis(1, at=seq(1.65, 2, by=0.1), cex.axis=1.2, col.axis="grey")
34  # Y-Achse Ticks
35  axis(2, at=seq(40, 120, by=10), cex.axis=1.5)
```

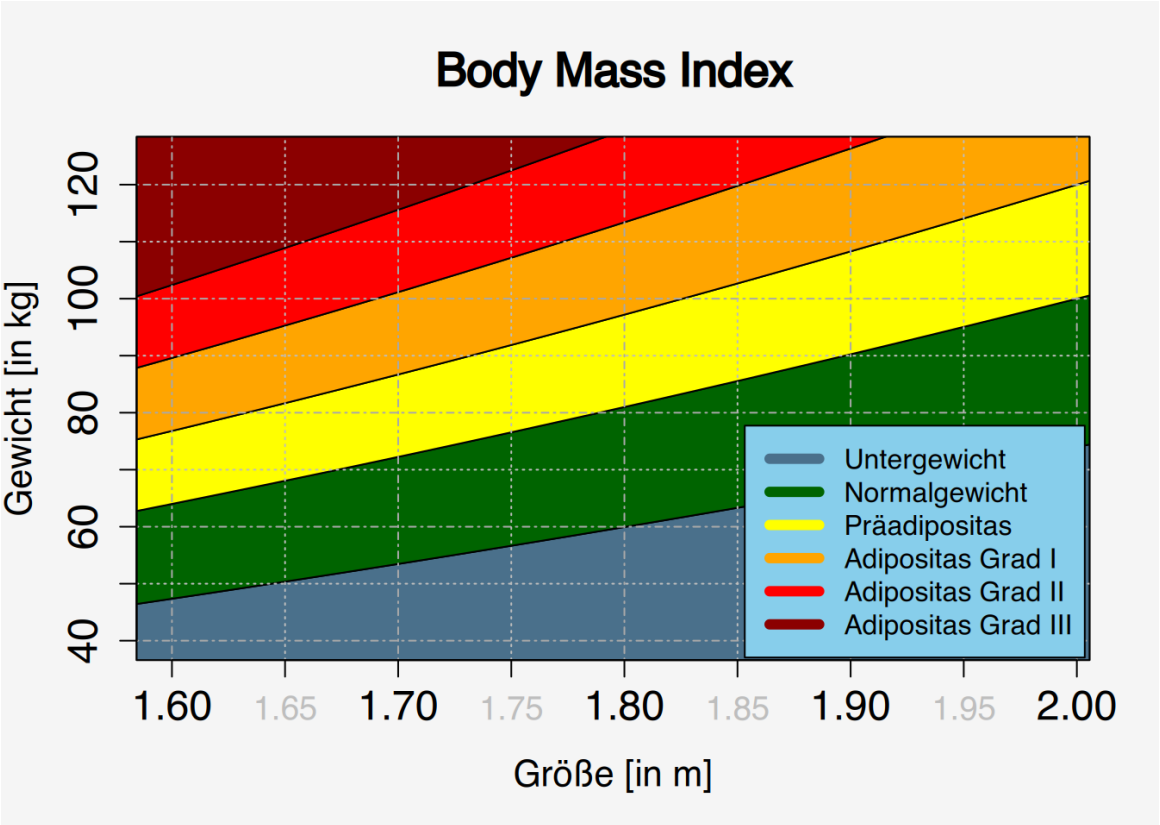


Abb. 3.24

3.2.6.2 ggplot()

Schauen wir nun, wie wir das Diagramm mit `ggplot()` zeichnen würden. Dafür müssen wir die Daten zunächst als "long table" schreiben (siehe 1.6 auf Seite 141). Wie Sie evtl. im obigen Beispiel bemerkt haben, kommt Klasse7 in den Plotdimensionen gar nicht vor, daher lassen wir sie direkt weg.

```
# bereite "long table" vor
df1 <- data.frame(Gross, Wert=Klasse1, BMI="Untergewicht")
df2 <- data.frame(Gross, Wert=Klasse2, BMI="Normalgewicht")
df3 <- data.frame(Gross, Wert=Klasse3, BMI="Präadipositas")
df4 <- data.frame(Gross, Wert=Klasse4, BMI="Adipositas I")
df5 <- data.frame(Gross, Wert=Klasse5, BMI="Adipositas II")
df6 <- data.frame(Gross, Wert=Klasse6, BMI="Adipositas III")
# schreibe alles ins Datenframe df
df <- rbind(df1, df2, df3, df4, df5, df6)
df$BMI <- factor(df$BMI, ordered = TRUE,
                 levels=c("Untergewicht", "Normalgewicht", "Präadipositas",
                          "Adipositas I", "Adipositas II", "Adipositas III"))
```

Das geht auch auf diese Weise:

```
df <- data.frame(Gross=rep(Gross, 6),
                 Wert=c(Klasse1, Klasse2, Klasse3, Klasse4, Klasse5, Klasse6),
                 BMI=factor(c(rep("Untergewicht", 200), rep("Normalgewicht", 200),
                              rep("Präadipositas", 200), rep("Adipositas I", 200),
                              rep("Adipositas II", 200), rep("Adipositas III", 200)),
                            ordered=TRUE,
                            levels=c("Untergewicht", "Normalgewicht", "Präadipositas",
                                     "Adipositas I", "Adipositas II", "Adipositas III")
                            )
                 )
```

Nun können wir die Daten an `ggplot()` übergeben.

```
## Plotten mittels ggplot
library(ggplot2)
ggplot(df) + aes(x=Gross, y=Wert, fill=BMI) +
  geom_polygon()+
  coord_cartesian(xlim = c(1.55, 2.05), ylim = c(35, 125), expand = FALSE)+
  scale_fill_manual(values = Farben) +
  theme(legend.position = c(0.9, 0.2))
```

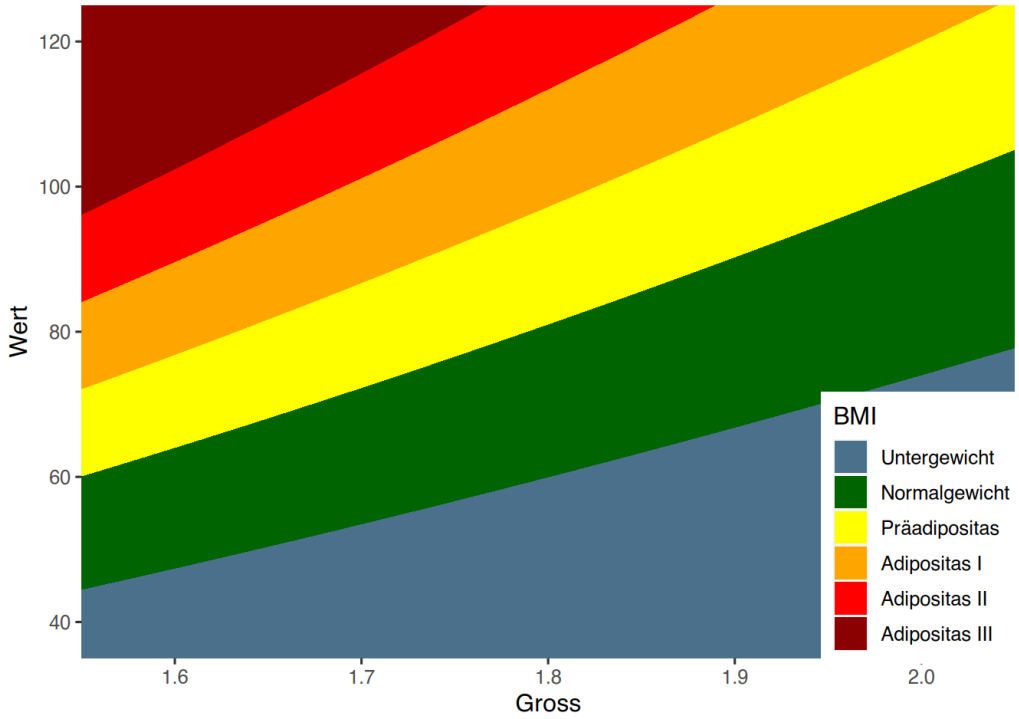


Abb. 3.25

3.2.6.2.1 Alternative

Eine alternative Lösung kann mit `geom_contour_filled()` erreicht werden. Dieses Geom kennt außer `x` und `y` noch eine zusätzliche virtuelle `z`-Achse. Wenn wir das Gewicht auf der `y`-Achse darstellen wollen, und die Körpergröße auf der `x`-Achse, so können wir auf der virtuellen `z`-Achse den jeweils dazugehörigen BMI-Wert angeben. Dem Geom `geom_contour_filled()` können wir dann über den `breaks`-Parameter die BMI-Grenzwerte mitteilen, und die jeweiligen Bereiche manuell einfärben.
Zunächst erzeugen wir mittels `expand.grid()` eine Tabelle mit Kombinationen von Körpergröße und Gewicht.

```
# erzeuge alle Kombinationen von
# Körpergröße und Gewicht
df <- expand.grid(Gewicht = seq(30, 130, 0.1),
                  Größe = seq(1.5, 2.1, 0.01))
# erste Zeilen anzeigen
head(df)
```

```
  Gewicht Größe
1    30.0   1.5
2    30.1   1.5
3    30.2   1.5
4    30.3   1.5
5    30.4   1.5
6    30.5   1.5
```

Jetzt können wir `ggplot()` wie folgt aufrufen:

```r
library(ggplot2)
ggplot(df) +
  aes(Größe, Gewicht) +
  # berechne für "z" den BMI (kg/m^2)
  geom_contour_filled(aes(z = Gewicht/Größe^2),
                      # schwarzer Rahmen
                      color = "black",
                      # BMI Grenzwerte
                      breaks = c(0, 18.5, 25, 30, 35, 40, 100)) +
  # manuell umfärben und umbenennen
  scale_fill_manual("BMI",
                    values = rev(c("#8b0000", "red", "#ffa500", "yellow",
                                   "#006400", "#4a708b")),
                    labels=c("Untergewicht", "Normalgewicht", "Präadipositas",
                             "Adipositas I", "Adipositas II", "Adipositas III")) +
  # Plot-Dimensionen festlegen
  coord_cartesian(xlim = c(1.55, 2.05),
                  ylim = c(35, 125),
                  expand = FALSE) +
  # X-Ticks
  scale_x_continuous(breaks = seq(1.6, 2.0, 0.05)) +
  #Y-Ticks
  scale_y_continuous(breaks = seq(40, 120, 10) )+
  # legendenbox innerhalb des Plots platzieren
  theme(legend.position = c(0.88, 0.25)) +
  # Graues Gitter per vline und hline hinzufügen
  geom_vline(xintercept = seq(1.55, 2.05, 0.01),
             linetype = "solid",
             color = "gray",
             size = 0.1,
             alpha=0.3) +
  geom_hline(yintercept=seq(36, 124, 2),
             linetype="solid",
             color="gray",
             alpha=0.3)
```

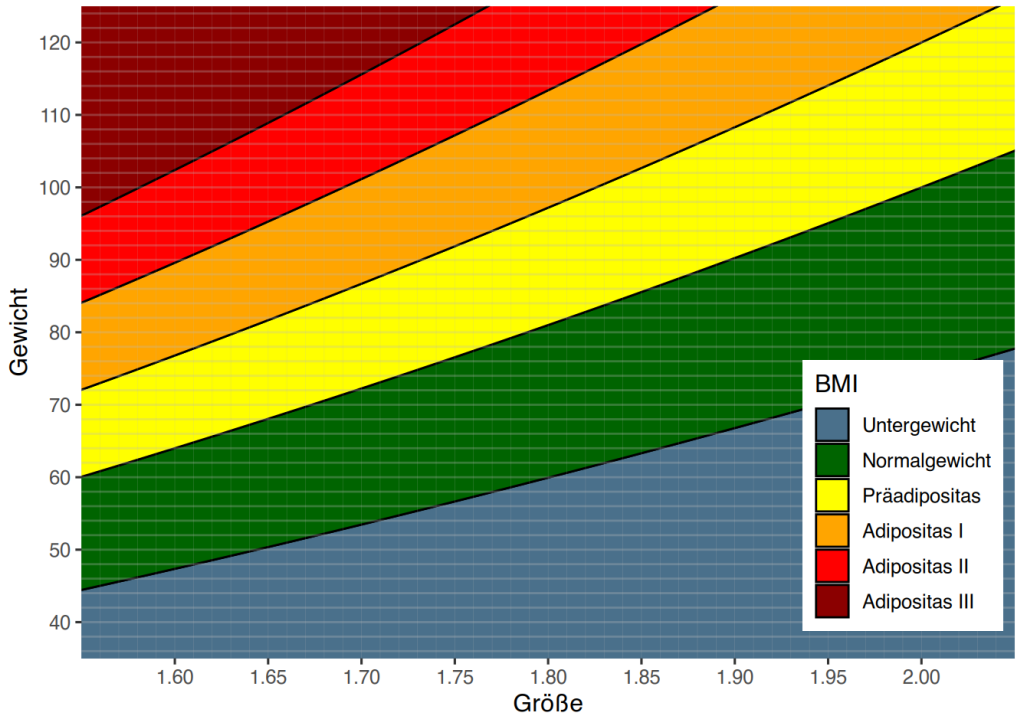


Abb. 3.26

3.2.7 Wie hast du das Titelbild erzeugt?

Das Titelbild ist mit der Altersverteilung aus dem epa-Datensatz erstellt.

```
# datensatz laden
epa <- jgsbook::epa

# ggplot
library(ggplot2)

# plotten
ggplot(epa) +
  aes(x=age) +
  geom_histogram(aes(y=after_stat(density)),
                 color="white",
                 fill="#8a658a") +
  stat_density(geom="area",
               color="blue",
               fill="seagreen3",
               linetype="dotted",
               alpha=0.5)+
  stat_function(fun=dnorm,
                args=(c(mean=mean(epa$age),sd=sd(epa$age))),
                color = "red", lwd=1.5,
                linetype = "dashed") +
  theme_void()
```

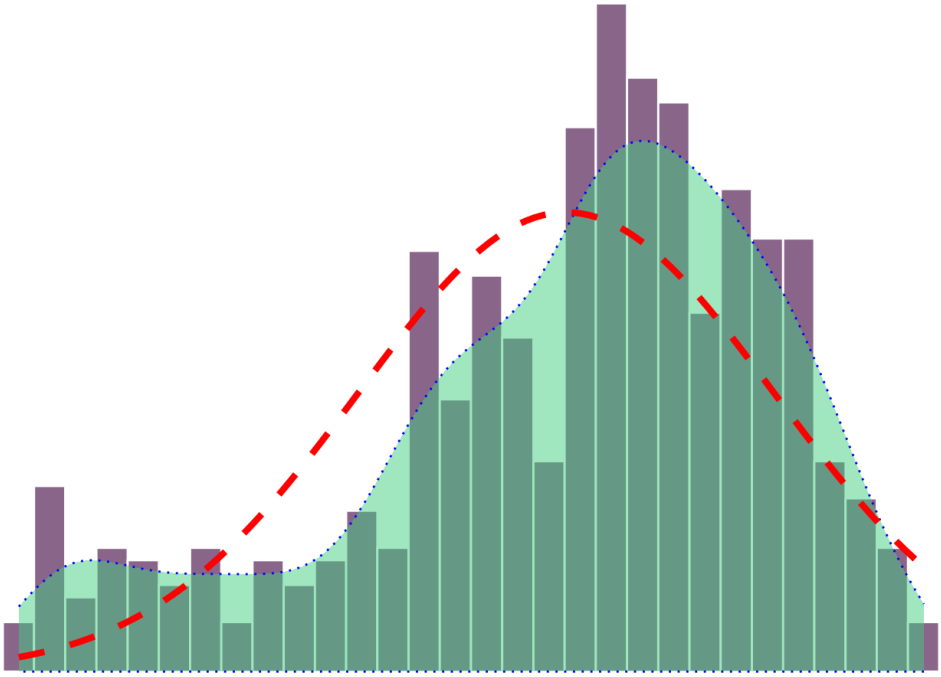


Abb. 3.27: Titelbild

Mit dem Paket `hexSticker` kann daraus ein schöner Hex-Sticker erstellt werden.

```
library(hexSticker)
# Titelbild in Objekt speichern
titelbild <- ggplot(epa) +
  aes(x=age) +
  geom_histogram(aes(y=after_stat(density)),
                 color="white", lwd=0.3,
                 fill="#8a658a") +
  stat_density(geom="area",
               color="blue",
               fill="seagreen3",
               linetype="dotted",
               alpha=0.5)+
  stat_function(fun=dnorm,
                args=(c(mean=mean(epa$age),sd=sd(epa$age))),
                color = "red",
                linetype = "dashed") +
  theme_void()

# Sticker erzeugen
sticker(titelbild, package="jgsbook", p_color="white", p_x=0.7, p_size=20,
        s_x=1.05, s_y=1.06, s_width=1.5, s_height=1.4,
        h_fill="#07A1E2", h_color="#185191",
        url="produnis.de/R", u_size=5, u_color="white",
        filename="jgsbook-hexsticker.png")
```



Abb. 3.28: Hex-Sticker

3.3 Karten plotten

Ich möchte meine Daten mit Landschaftskarten kombinieren!

3.3.1 Deutschland

Wir verwenden zum plotten das Zusatzpaket `sf`, mit dessen Hilfe `shape geopackages` verarbeitet werden können. Das zu diesem Format passende Kartenmaterial wird vom Bundesamt für Kartographie und Geodäsie kostenfrei zur Verfügung gestellt, siehe https://gdz.bkg.bund.de/index.php/default/digitale-geodaten/verwaltungsgebiete.html. Als Starterpaket ist der Datensatz der Verwaltungsgebiete `VG2500` zu empfehlen. Dieser enthält die Verwaltungsebenen vom Staat bis zu den Kreisen mit den jeweiligen Grenzen:

- `VG2500_STA` - Staat
- `VG2500_LAN` - Bundesländer
- `VG2500_RBZ` - Regierungsbezirke
- `VG2500_KRS` - Kreise

Dabei besteht jeder Kartensatz aus den Unterdateien

- `*.shp` - Shape-Datei. Diese lesen wir in R ein
- `*.shx` - Geometrieindex
- `*.prj` - Projektion
- `*.dbf` - Attribute
- `*.cpg` - Zeichensatz

3.3.2 Kartendaten einlesen

Sind die Kartendaten heruntergeladen, müssen **alle** Dateien (nicht nur die `.shp` Datei) mit Dateinamen `VG2500_KRS` (beispielhaft für Kreise) in den R-Studio-Projektordner gelegt werden. Ich persönlich habe dort den Unterordner `data` erstellt, von dem aus ich die Daten einlese. Der Import erfolgt mit der Funktion `read_sf()` wie folgt:

```
1  # Paket aktivieren
2  library(sf)
3
4  # Lade Kartendaten der Bundesländer
5  bundeslaender <- read_sf("data/VG2500_LAN.shp")
6
7  # anschauen
8  head(bundeslaender)
```

```
Simple feature collection with 6 features and 24 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 280353.1 ymin: 5471359 xmax: 674168.3 ymax: 6101335
Projected CRS: ETRS89 / UTM zone 32N
# A tibble: 6 × 25
  OBJID BEGINN       ADE    GF   BSG ARS   AGS   SDV_ARS GEN   BEZ     IBZ BEM
  <chr> <date>     <int> <int> <int> <chr> <chr> <chr>   <chr> <chr> <int> <chr>
1 DEBK… 2021-09-01     2     9     1 01    01    010020… Schl… Land     20 --
2 DEBK… 2021-09-01     2     9     1 02    02    020000… Hamb… Frei…    22 --
3 DEBK… 2021-09-01     2     9     1 03    03    032410… Nied… Land     20 --
4 DEBK… 2021-09-01     2     9     1 04    04    040110… Brem… Frei…    23 --
5 DEBK… 2021-09-01     2     9     1 05    05    051110… Nord… Land     20 --
6 DEBK… 2021-09-01     2     9     1 06    06    064140… Hess… Land     20 --
#   13 more variables: NBD <chr>, SN_L <chr>, SN_R <chr>, SN_K <chr>,
#    SN_V1 <chr>, SN_V2 <chr>, SN_G <chr>, FK_S3 <chr>, NUTS <chr>, ARS_0 <chr>,
#    AGS_0 <chr>, WSK <date>, geometry <MULTIPOLYGON [m]>
```

```r
# Lade Kartendaten der Kreise
kreise <- read_sf("data/VG2500_KRS.shp")
head(kreise)
```

```
Simple feature collection with 6 features and 24 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 478836.8 ymin: 5913378 xmax: 629246.8 ymax: 6075267
Projected CRS: ETRS89 / UTM zone 32N
# A tibble: 6 × 25
  OBJID BEGINN       ADE    GF   BSG ARS   AGS   SDV_ARS GEN   BEZ     IBZ BEM
  <chr> <date>     <int> <int> <int> <chr> <chr> <chr>   <chr> <chr> <int> <chr>
1 DEBK… 2021-09-01     4     9     1 01001 01001 010010… Flen… Krei…    40 --
2 DEBK… 2021-09-01     4     9     1 01002 01002 010020… Kiel  Krei…    40 --
3 DEBK… 2021-09-01     4     9     1 01003 01003 010030… Lübe… Krei…    40 --
4 DEBK… 2021-09-01     4     9     1 01004 01004 010040… Neum… Krei…    40 --
5 DEBK… 2021-09-01     4     9     1 01051 01051 010510… Dith… Kreis    42 --
6 DEBK… 2021-09-01     4     9     1 01053 01053 010530… Herz… Kreis    42 --
#   13 more variables: NBD <chr>, SN_L <chr>, SN_R <chr>, SN_K <chr>,
#    SN_V1 <chr>, SN_V2 <chr>, SN_G <chr>, FK_S3 <chr>, NUTS <chr>, ARS_0 <chr>,
#    AGS_0 <chr>, WSK <date>, geometry <MULTIPOLYGON [m]>
```

Die in den Datensätzen für uns interessanten Variablen sind `ARS` (Amtlicher Regionalschlüssel), `AGS` (Amtlicher Gemeindeschlüssel) und `GEN` (geographischer Name), denn über diese können die entsprechenden Flächen der Karte manipuliert werden. Die eigentlichen Positionsangaben sind in der Variable `geometry` enthalten.

3.3.3 Karten erzeugen

Erzeugen wir zunächst eine grobe Deutschlandkarte der Bundesländer mit `ggplot()`. Hierfür kann die Funktion `geom_sf()` genutzt werden.

```r
library(ggplot2)
# Hier wird eine weiße Karte der Bundesländer
# mit schwarzen Grenzen und dünner Linie gezeichnet
ggplot(bundeslaender) +
  geom_sf(fill="white", color="black", linewidth=0.3) +
  theme_void()
# Hier eine Karte der Kreise
ggplot(kreise) +
  geom_sf(fill="white", color="black", linewidth=0.3) +
  theme_void()
```

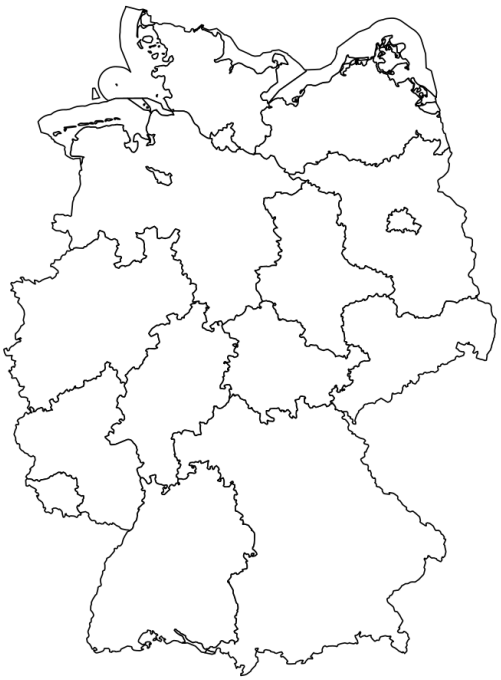


Abb. 3.29

```
# Hier eine Karte der Kreise
ggplot(kreise) +
  geom_sf(fill="white", color="black", linewidth=0.3) +
  theme_void()
```

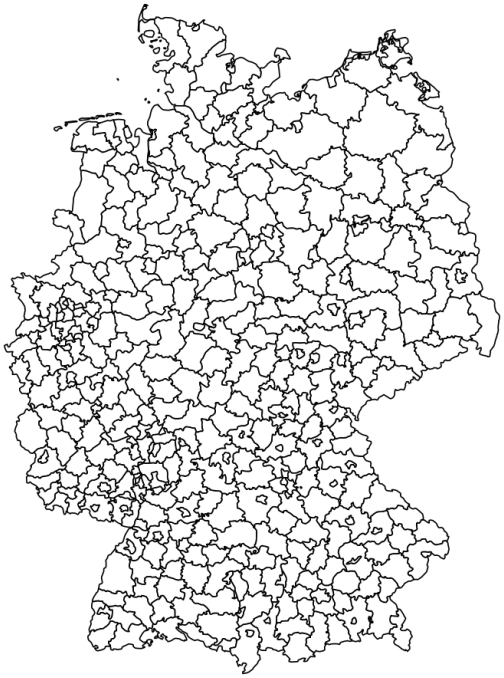


Abb. 3.30

3.3.4 Kacheln füllen

Auf den Seiten des Bundesinstituts für Bau-, Stadt-, und Raumforschung können interessante Kennzahlen heruntergeladen werden, siehe https://www.inkar.de. Die Daten aus Tabelle `Bevoelkerung2022.txt` stammen aus diesem riesigen Datensatz und beschreiben die Bevölkerungszahlen der Kreise von 2021 und 2022. Diese Daten sollen genutzt werden, um die Kartenkacheln einzufärben.

```
# Tabelle einlesen
Einwohner <- read.table(url("https://www.produnis.de/R/data/Bevoelkerung2022.txt"),
                        header=TRUE, sep=",")
# Daten anschauen
head(Einwohner)
```

```
  Kennziffer            Raumeinheit Aggregat  J2021  J2022
1       1001       Flensburg, Stadt   Kreise  91113  90050
2       1002 Kiel, Landeshauptstadt   Kreise 246243 245217
3       1003     Lübeck, Hansestadt   Kreise 216277 215595
4       1004      Neumünster, Stadt   Kreise  79496  77002
5       1051           Dithmarschen   Kreise 133969 132752
6       1053    Herzogtum Lauenburg   Kreise 200819 201212
```

In der Spalte `Kennziffer` der Tabelle sind die amtlichen Regionalschlüssel enthalten. Diese stimmen mit den `ARS` und `AGS` Kennziffern aus den Kartendaten überein. Wir können also über diese Variable die Datensätze zusammenführen.
Beachten Sie, dass in der ersten Spalte führende Nullen angegeben sind. Um sicherzustellen, dass die führenden Nullen beim Einlesen in R erhalten bleiben, muss die Option `colClasses="character"` verwendet werden.

```
# Daten "richtrig" einlesen
Einwohner <- read.table(url("https://www.produnis.de/R/data/Bevoelkerung2022.txt"),
                        header=TRUE, sep=",", colClasses = "character")
# anschauen
head(Einwohner)
```

```
  Kennziffer            Raumeinheit Aggregat  J2021  J2022
1      01001       Flensburg, Stadt   Kreise  91113  90050
2      01002 Kiel, Landeshauptstadt   Kreise 246243 245217
3      01003     Lübeck, Hansestadt   Kreise 216277 215595
4      01004      Neumünster, Stadt   Kreise  79496  77002
5      01051           Dithmarschen   Kreise 133969 132752
6      01053    Herzogtum Lauenburg   Kreise 200819 201212
```

In einer neuen Variable `Diff` soll die prozentuale Veränderung der Bevölkerungsanzahl von 2021 nach 2022 gespeichert werden.

```
Einwohner$Diff <- ((Einwohner$J2022 - Einwohner$J2021) / Einwohner$J2021)*100
```

Jetzt können wir die beiden Datensätze mittels `left_join()` über die Variablen `Kennziffer` und `ARS` zusammenführen.

```
library(dplyr)
# Einwohnerzahlen und Kartenmaterial verbinden
kreise2 <- left_join(kreise, Einwohner,
                     join_by(ARS==Kennziffer))
```

Das neue Objekt `kreise2` kann nun zum Plotten verwendet werden. Hierbei werden die Kartensegmente in Abhängigkeit zur prozentualen Bevölkerungsveränderung eingefärbt. Um einen schönen Farbverlauf zu erzeugen, nutzen wir die `muted()` Funktion aus dem `scales` Paket.

```
library(scales)
ggplot(kreise2) +
  geom_sf(aes(fill=Diff),linewidth =0, alpha=0.9) +
  # Färbefarben festlegen
  scale_fill_gradient2(low=muted("red"),
                       mid="lightyellow", high= muted("green"),
                       midpoint=0, na.value="black") +
  coord_sf() +
  # Legende der Färbung
  guides(fill=guide_colorbar(barwidth=0.5,
                             barheight=10,
                             title="Veränderung in %"))
```

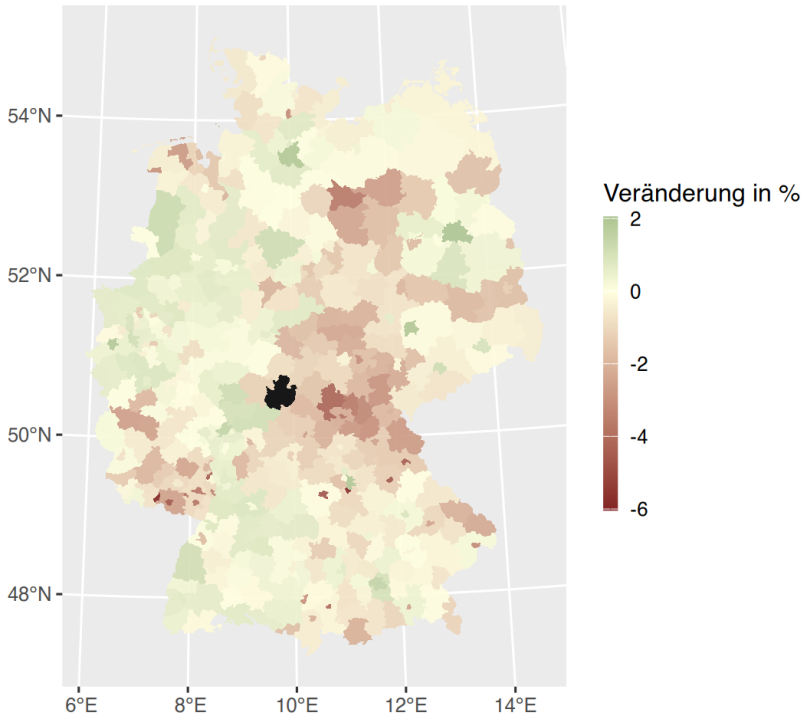


Abb. 3.31

Die Karten können auch kombiniert werden. So könnten wir unserer Karte noch die Bundsländergrenzen hinzufügen.

```
# nochmal der selbe Plot
ggplot(kreise2) +
  geom_sf(aes(fill=Diff),linewidth =0, alpha=0.9) +
  # Färbefarben festlegen
  scale_fill_gradient2(low=muted("red"),
                       mid="lightyellow", high= muted("green"),
                       midpoint=0, na.value="black") +
  coord_sf() +
  # Legende der Färbung
  guides(fill=guide_colorbar(barwidth=0.5,
                             barheight=10,
                             title="Veränderung in %")) +
  # Grenzen der Bundesländer drüberlegen
  geom_sf(data=bundeslaender, fill=NA, color="black", linewidth=0.3)
```

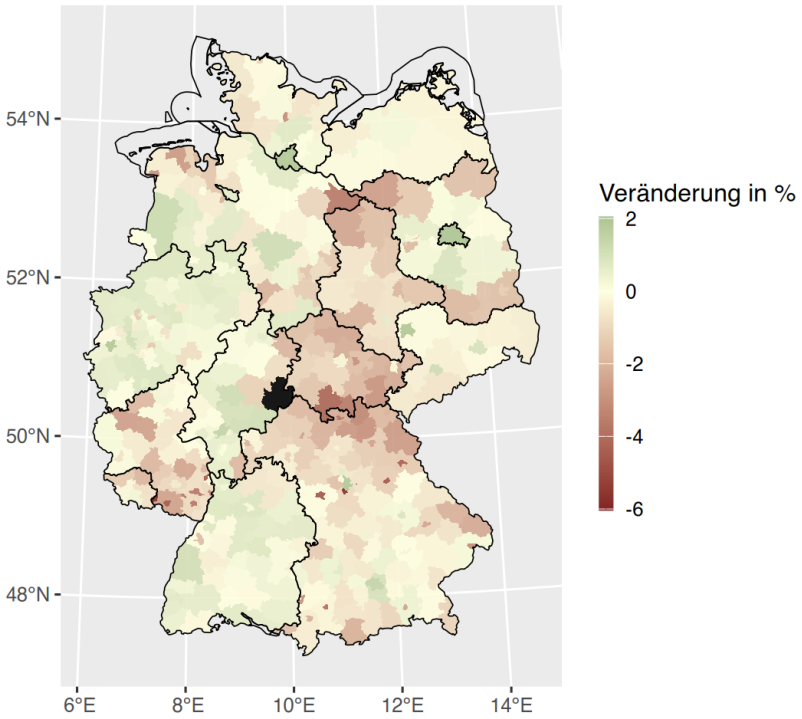


Abb. 3.32

3.4 Pakete verwalten

Ich möchte Zusatzpakete verwalten.

3.4.1 Pakete beim Start automatisch laden

Ich möchte, dass Pakete direkt beim Start von `R` geladen werden.

Legen Sie hierzu in Ihrem Arbeitsverzichnis die Datei `.Rprofile` an. In diese Datei können Sie alle Befehle schreiben, die beim Start ausgeführt werden sollen. Das geht auch direkt in R :

```
file.edit(".Rprofile")

# Inhalt von .Rprofile
print("Willkommen zurück!")
print("Ich lade Paket tidyverse")
library(tidyverse)
```

3.5 RMarkdown

Ich möchte RMarkdown individualisieren.

3.5.1 Ausgabeformat per .css-Datei ändern

Ich möchte die Ausgabe der html-Datei ändern.

Hierfür kann eine css-Datei angelegt werden. Im Kopf des RMarkdown-Dokumentes wird die Datei MeinStyle.css wie folgt eingebunden:

```
---
title: "Test"
output:
  html_document:
       css:  MeinStyle.css
---
```

In der css-Datei können dann wie gewohnt Style-Angaben gemacht werden, z.B. das Aussehen der Codebox. Diese sind über die Parameter class.source (R-Befehle) und class.output (R-Ausgabe) referenzierbar. In der CSS-Datei definieren wir unsere eigene Klasse "meine" und hinterlegen gewünschte Parameter.

```
 .meine {
    background-color: rgb(100, 0, 0);
}
```

Auf diese Klasse können wir in den Chunks verweisen mit

```
```{r class.output="meine"}
# unser R Code
```
```

Möchte man alle Chunks automatisch auf class.output="meine" setzen, kann dies im YAML-Kopfbereich wie folgt angegeben werden:

```
knitr::opts_chunk$set(echo = TRUE, class.output="meine")
```

Es gibt auch vordefinierte Klassen für die Chunks, probieren Sie bg-primary, bg-success, bg-info, bg-warning, und bg-danger aus.

3.6 quarto

Ich möchte quarto individualisieren.

3.6.1 eigene LaTeXVorlagen erstellen

Quarto ermöglicht es, zusätzlichen LaTeX-Code einzufügen[1] oder nur einzelne Teile des Templates auszutauschen[2].
Es können aber auch eigene vollständige LaTeX-Vorlagen erzeugt werden.
Folgende Punkte sind dabei zu beachten:

- Möchten Sie alle Funktionen von quarto verfügbar und Ihre Vorlage so flexibel wie möglich machen, sollten Sie die quarto-Vorlage kopieren[3] und Ihre Änderungen direkt dort einfügen.
- Möchten Sie sich z.B. eine Briefvorlage erstellen, deren wesentlichen Parameter (Dokumentenklasse, Schriftgröße, Geomtery, etc.) sich eh nie ändern werden, ist es durchaus legitim, eine eigene "rudimentäre" LaTeX-Datei zu erstellen.

3.6.1.1 .tex-Date

Ich habe mir einen ßentralenÖrdner erstellt (z.B. Dokumente/Vorlagen/quarto), in welchem ich alle Vorlagendateien ablege. So kann ich sie später bequem in jeder .qmd-Datei einbinden.
Die LaTeX-Vorlage sollte mindestens folgende Werte enthalten, damit die rudimentären quarto-Funktionen unterstützt werden.

```
\documentclass[a4paper, 10pt]{scrartcl} % Dokumentenklasse wählen
\usepackage[utf8]{inputenc}
\usepackage[ngerman]{babel}             % deutsche Sprache harcoded
\usepackage{lmodern}
\usepackage[numbered]{bookmark}
\usepackage{graphicx}
\usepackage{longtable}
\usepackage{booktabs}
\usepackage{calc}
\usepackage{lastpage}
\usepackage{geometry}
\usepackage{tabularx}
\usepackage{colortbl}
\usepackage{multirow}
\usepackage{float}
\usepackage{amsmath}
\usepackage{hhline}
\usepackage{blindtext}
\usepackage{xcolor}
\usepackage[normalem]{ulem}

\definecolor{linkblue}{RGB}{17, 40, 73}

\hypersetup{
colorlinks,
linkcolor=linkblue,
```

[1] https://quarto.org/docs/output-formats/pdf-basics.html
[2] https://quarto.org/docs/journals/templates.html
[3] https://github.com/quarto-dev/quarto-cli/tree/main/src/resources/formats/pdf/pandoc

```
urlcolor=blue,
bookmarks=true,
plainpages=false,
hypertexnames=true,
pdftitle={$title$},
pdfauthor={$author$}
}

\providecommand{\tightlist}{%
  \setlength{\itemsep}{0pt}\setlength{\parskip}{0pt}}

%% Callout-Boxen
\makeatletter
\@ifpackageloaded{tcolorbox}{}{\usepackage[many]{tcolorbox}}
\@ifpackageloaded{fontawesome5}{}{\usepackage{fontawesome5}}
\definecolor{quarto-callout-color}{HTML}{909090}
\definecolor{quarto-callout-note-color}{HTML}{0758E5}
\definecolor{quarto-callout-important-color}{HTML}{CC1914}
\definecolor{quarto-callout-warning-color}{HTML}{EB9113}
\definecolor{quarto-callout-tip-color}{HTML}{00A047}
\definecolor{quarto-callout-caution-color}{HTML}{FC5300}
\definecolor{quarto-callout-color-frame}{HTML}{acacac}
\definecolor{quarto-callout-note-color-frame}{HTML}{4582ec}
\definecolor{quarto-callout-important-color-frame}{HTML}{d9534f}
\definecolor{quarto-callout-warning-color-frame}{HTML}{f0ad4e}
\definecolor{quarto-callout-tip-color-frame}{HTML}{02b875}
\definecolor{quarto-callout-caution-color-frame}{HTML}{fd7e14}
\makeatother

$if(highlighting-macros)$
$highlighting-macros$
$endif$

\begin{document}

$body$

\end{document}
```

Die Variable *body* fügt den Inhalte der `qmd`-Datei (alles unterhalb des YAML-Headers) ein.

3.6.1.2 Vorlage einbinden

Die Vorlage wird wie folgt im YAML-Header eingebunden:

```
---
format:
  pdf:
    template: /pfad/zur/vorlage.tex
---
```

3.6.1.3 YAML-Variablen

Im YAML-Header können Variablen gesetzt werden, welche in der `.tex`-Datei abgerufen werden können.

```
---
title: Meine Vorlage
variable1: "Test"
variable2: "Noch ein Test"

```

```
format:
  pdf:
    template: /pfad/zur/vorlage.tex
---
```

In der `.tex`-Datei sind die oben definierten Variablen verfügbar per `$variable1$` und `$variable2$`.

```
\begin{document}

$title$

$body$

Dies ist ein $variable1$.
Und dies ist $variable2$.

\end{document}
```

Eine Beispiel für die oben erwähnte Briefvorlage finden Sie unter https://www.produnis.de/blog/post/quarto-briefvorlage/.

3.6.2 quarto am Handy oder Tablet

Am Handy oder am Tablet sind `quarto` und LaTeXnicht verfügbar.
Eine Möglichkeit, dennoch `.qmd`-Dateien zu rendern, besteht darin, das Rendern an einen Server auszulagern. Wenn Sie zu Hause einen kleinen Linux-Heimserver betreiben, könnten Ihnen diese beiden Bots weiterhelfen. Beide setzen voraus, dass `quarto` auf dem Server installiert ist. Ebenfalls sollte LaTeXvia `quarto install tool tinytex` installiert worden sein, damit eventuell benötigte Pakete automatisch nachgeladen werden.

1. quarto-Bot für Nextcloud
   - Der Bot ist ein kleines Shell-Script.
   - Der Bot mountet ein Nextcloud-Verzeichnis per Webdav und schaut nach, ob in dem Verzeichnis eine `.qmd`-Datei enthalten ist.
   - Sollte dies der Fall sein, führt der Bot den Befehl `quarto render DATEI.qmd` aus.
   - Da die erzeugt PDF-Datei im selben Verzeichnis liegt, ist sie über die Nextcloud-Dateiansicht verfügbar und wird ebenfalls an alle Clients synchronisiert.
   - https://www.produnis.de/blog/post/simpler-quarto-bot-fuer-nextcloud/
2. quarto-matrix-bot
   - Der Bot ist in python geschrieben.
   - Der Bot lauscht in vorher angegebenen Matrix-Räumen.
   - Wird eine `.qmd`-Datei in den Raum gesendet, führt der Bot den Befehl `quarto render DATEI.qmd` aus.
   - Wenn eine PDF-Datei erzeugt wurde, sendet der Bot diese Datei in den Matrix-Raum.
   - Sollte das Rendern fehlschlagen, postet der Bot die Fehlermeldungen in den Raum.
   - https://github.com/rgomez90/matrix-bot

3.7 Referenztabellen erstellen

Ich möchte Referenztabellen erstellen und benötige die Werte.

3.7.1 Fallzahlen nach Effektstärke und Power

$\alpha = 0.05$															
	Effektstärke														
Power	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**	**0.55**	**0.6**	**0.65**	**0.7**	**0.75**	**0.8**
0.6	981	436	246	158	110	81	62	49	40	33	28	24	21	18	16
0.7	1235	550	310	198	138	102	78	62	50	42	35	30	26	23	20
0.8	1571	699	393	252	175	129	99	78	64	53	45	38	33	29	26
0.9	2102	935	526	337	234	173	132	105	85	70	59	51	44	38	34
0.95	2600	1156	651	417	290	213	163	129	105	87	73	62	54	47	42
0.99	3675	1634	920	589	409	301	231	182	148	122	103	88	76	66	58

Tabelle 3.1: Anzahl Probanden pro Gruppe nach Power und Effektstärke bei $\alpha = 0,05$

$\alpha = 0.01$															
	Effektstärke														
Power	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**	**0.55**	**0.6**	**0.65**	**0.7**	**0.75**	**0.8**
0.6	1603	713	402	258	180	132	102	81	66	55	46	40	34	30	27
0.7	1924	856	482	309	215	159	122	97	79	65	55	47	41	36	32
0.8	2337	1040	586	375	261	192	148	117	95	79	67	57	49	43	38
0.9	2978	1324	746	478	332	245	188	149	121	100	84	72	62	55	48
0.95	3564	1585	892	572	398	293	224	178	144	119	101	86	74	65	57
0.99	4808	2138	1203	771	536	394	302	239	194	161	135	115	100	87	77

Tabelle 3.2: Anzahl Probanden pro Gruppe nach Power und Effektstärke bei $\alpha = 0,01$

Der Quellcode für die Werte in Tabelle 3.1 und 3.2 ist etwas komplizierter, da wir zunächst eine Funktion programmieren müssen, um die Tabelle ausgeben zu können.

```
# lade Paket zur Poweranalyse
library(pwr)
# erzeuge Hilfsfunktion, die nur die errechnete Fallzahl ausgibt
#----------------------------------
powertabelle <- function(alpha, d, power) { round(pwr::pwr.t.test(d = d, sig.level = alpha, power = power,
type = "two.sample", alternative = "two.sided")$n)}
# Ende der Funktion


# ein Datenframe aus allen gewünschten Kombinationen
# von power, alpha und d
scen <- expand.grid( alpha = c(0.05, 0.01),
                     d = seq(from = 0.1, to = 0.8, by = 0.05),
                     power = c(seq(0.6, 0.9, 0.1), 0.95, 0.99)
                   )

# füge Datenframe "scen" eine neue Spalte "n" hinzu
# per apply() wird auf jede Datenreihe die Funktion "powertabelle" angewendet.
# line() hilft dabei, die richtigen Werte aus dem
# Datenframe "scen" an die Funktion "powertabelle" zu übergeben
scen$n <- apply(scen, 1, function(line) powertabelle(line["alpha"], line["d"], line["power"]))

# zeige die ersten 20 Datenreihen an
head(scen, 20)
```

```
   alpha    d power    n
1   0.05 0.10   0.6  981
2   0.01 0.10   0.6 1603
3   0.05 0.15   0.6  436
4   0.01 0.15   0.6  713
5   0.05 0.20   0.6  246
6   0.01 0.20   0.6  402
7   0.05 0.25   0.6  158
8   0.01 0.25   0.6  258
9   0.05 0.30   0.6  110
10  0.01 0.30   0.6  180
11  0.05 0.35   0.6   81
12  0.01 0.35   0.6  132
13  0.05 0.40   0.6   62
14  0.01 0.40   0.6  102
15  0.05 0.45   0.6   49
16  0.01 0.45   0.6   81
17  0.05 0.50   0.6   40
18  0.01 0.50   0.6   66
19  0.05 0.55   0.6   33
20  0.01 0.55   0.6   55
```

Jetzt können Teilmengen von `scen` erstellt werden, um die Werte entsprechend der gewünschten Tabellen ($\alpha = 0.05$ und $\alpha = 0.01$) zu erzeugen.

```
1  # erstelle ein subset für "alpha=0.05"
2  subset05 <- subset(scen, alpha == .05)
3
4  # benenne die Spalten um
5  # und bringe per "reshape" die Tabelle ins "wide"-Format
6  tabelle05 <- reshape(subset05, v.names = "n", timevar = "d",  idvar = "power", direction = "wide")
7
8  # gib tabelle aus
9  print(tabelle05)
```

```
    alpha power n.0.1 n.0.15 n.0.2 n.0.25 n.0.3 n.0.35 n.0.4 n.0.45 n.0.5 n.0.55 n.0.6 n.0.65 n.0.7 n.0.75 n.0.8
1    0.05  0.60   981    436   246    158   110     81    62     49    40     33    28     24    21     18    16
31   0.05  0.70  1235    550   310    198   138    102    78     62    50     42    35     30    26     23    20
61   0.05  0.80  1571    699   393    252   175    129    99     78    64     53    45     38    33     29    26
91   0.05  0.90  2102    935   526    337   234    173   132    105    85     70    59     51    44     38    34
121  0.05  0.95  2600   1156   651    417   290    213   163    129   105     87    73     62    54     47    42
151  0.05  0.99  3675   1634   920    589   409    301   231    182   148    122   103     88    76     66    58
```

```
1  # erstelle ein subset für "alpha=0.01"
2  subset01 <- subset(scen, alpha == .01)
3
4  # benenne die Spalten um
5  # und bringe per "reshape" die Tabelle ins "wide"-Format
6  tabelle01 <- reshape(subset01, v.names = "n", timevar = "d",  idvar = "power", direction = "wide")
7
8  # gib tabelle aus
9  print(tabelle01)
```

```
    alpha power n.0.1 n.0.15 n.0.2 n.0.25 n.0.3 n.0.35 n.0.4 n.0.45 n.0.5 n.0.55 n.0.6 n.0.65 n.0.7 n.0.75 n.0.8
2    0.01  0.60  1603    713   402    258   180    132   102     81    66     55    46     40    34     30    27
32   0.01  0.70  1924    856   482    309   215    159   122     97    79     65    55     47    41     36    32
62   0.01  0.80  2337   1040   586    375   261    192   148    117    95     79    67     57    49     43    38
92   0.01  0.90  2978   1324   746    478   332    245   188    149   121    100    84     72    62     55    48
122  0.01  0.95  3564   1585   892    572   398    293   224    178   144    119   101     86    74     65    57
152  0.01  0.99  4808   2138  1203    771   536    394   302    239   194    161   135    115   100     87    77
```

Wenn Sie den Abschnitt über *Tidy Data* auf Seite 141 gelesen haben, ist Ihnen evtl. aufgefallen, dass wir hier eine `long table` in eine `wide table` umgeformt haben. Das erwähne ich deshalb, da wir sonst immer genau das Gegenteil machen, nämlich `wide table` ins Format `long table` zu überführen.

Übrigens, dieses Buch ist mit LaTeX erstellt worden, und ich hatte keine Lust, die Werte für die Tabellen von Hand in LaTeX zu übertragen.
Mit der Funktion `xtable()` aus dem Zusatzpaket `xtable` kann die Ausgabe der Tabelle in LaTeX-Quellcode erfolgen:

```
# lade Paket "xtable"
library(xtable)

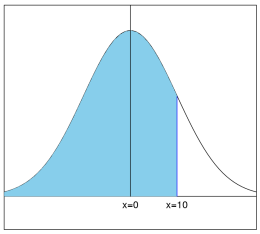
# gib die Tabelle als LaTeX-Code aus
xtable(tabelle01)
```

```
% latex table generated in R 4.0.4 by xtable 1.8-4 package
% Wed Mar 24 15:06:56 2021
\begin{table}[ht]
\centering
\begin{tabular}{rrrrrrrrrrrrrrrrr}
  \hline
 & alpha & power & n.0.1 & n.0.15 & n.0.2 & n.0.25 & n.0.3 & n.0.35 & n.0.4 & n.0.45 & n.0.5 & n.0.55 & n.0.6 & n.0.65 & n.0.7 & n.0.75 & n.0.8 \\
  \hline
2 & 0.01 & 0.60 & 1603.00 & 713.00 & 402.00 & 258.00 & 180.00 & 132.00 & 102.00 & 81.00 & 66.00 & 55.00 & 46.00 & 40.00 & 34.00 & 30.00 & 27.00 \\
  32 & 0.01 & 0.70 & 1924.00 & 856.00 & 482.00 & 309.00 & 215.00 & 159.00 & 122.00 & 97.00 & 79.00 & 65.00 & 55.00 & 47.00 & 41.00 & 36.00 & 32.00 \\
  62 & 0.01 & 0.80 & 2337.00 & 1040.00 & 586.00 & 375.00 & 261.00 & 192.00 & 148.00 & 117.00 & 95.00 & 79.00 & 67.00 & 57.00 & 49.00 & 43.00 & 38.00 \\
  92 & 0.01 & 0.90 & 2978.00 & 1324.00 & 746.00 & 478.00 & 332.00 & 245.00 & 188.00 & 149.00 & 121.00 & 100.00 & 84.00 & 72.00 & 62.00 & 55.00 & 48.00 \\
  122 & 0.01 & 0.95 & 3564.00 & 1585.00 & 892.00 & 572.00 & 398.00 & 293.00 & 224.00 & 178.00 & 144.00 & 119.00 & 101.00 & 86.00 & 74.00 & 65.00 & 57.00 \\
  152 & 0.01 & 0.99 & 4808.00 & 2138.00 & 1203.00 & 771.00 & 536.00 & 394.00 & 302.00 & 239.00 & 194.00 & 161.00 & 135.00 & 115.00 & 100.00 & 87.00 & 77.00 \\
   \hline
\end{tabular}
\end{table}
```

3.7.2 Verteilungsfunktion und Quantile der Standardnormalverteilung

ϕ **z**	**.00**	**.01**	**.02**	**.03**	**.04**	**.05**	**.06**	**.07**	**.08**	**.09**
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812	.9817
2.1	.9821	.9826	.9830	.9834	.9838	.9842	.9846	.9850	.9854	.9857
2.2	.9861	.9864	.9868	.9871	.9875	.9878	.9881	.9884	.9887	.9890
2.3	.9893	.9896	.9898	.9901	.9904	.9906	.9909	.9911	.9913	.9916
2.4	.9918	.9920	.9922	.9925	.9927	.9929	.9931	.9932	.9934	.9936
2.5	.9938	.9940	.9941	.9943	.9945	.9946	.9948	.9949	.9951	.9952
2.6	.9953	.9955	.9956	.9957	.9959	.9960	.9961	.9962	.9963	.9964
2.7	.9965	.9966	.9967	.9968	.9969	.9970	.9971	.9972	.9973	.9974
2.8	.9974	.9975	.9976	.9977	.9977	.9978	.9979	.9979	.9980	.9981
2.9	.9981	.9982	.9982	.9983	.9984	.9984	.9985	.9985	.9986	.9986
3.0	.9987	.9987	.9987	.9988	.9988	.9989	.9989	.9989	.9990	.9990
p =	0.5000	0.7500	0.8000	0.9000	0.9500	0.9750	0.9900	0.9950	0.9975	0.9990
z_p =	0.0000	0.6745	0.8416	1.2816	1.6449	1.9600	2.3263	2.5758	2.8070	3.0902

Tabelle 3.3: Verteilungsfunktion und Quantile der Standardnormalverteilung



Bei der z-Transformation benötigt man die Werte dieser Tabelle beispielsweise für Fragen wie "*Wie hoch ist der prozentuale Anteil für Werte, die* ***höchstens*** *X sind*"

Die Werte für Tabelle 3.3 werden erzeugt mit:

```
# erstelle Werte von 0.00 bis 3.09
z <- seq(from=0, to=3.09, by = 0.01)

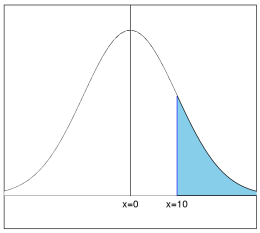
# P-Werte berechnen
pnorm(z)

# In Matrix mit 10 Spalten übertragen
matrix(pnorm(z),ncol=10, byrow=TRUE)
# auf 4 Stellen runden
matrix(round(pnorm(z),4),ncol=10, byrow=TRUE)
# In Dataframe übertragen
tabelle1 <- data.frame(matrix(round(pnorm(z),4),ncol=10, byrow=TRUE))
# benenne die Spalten
colnames(tabelle1) <- c(seq(0.00, 0.09, 0.01))
# benenne die Zeilen
row.names(tabelle1) <- c(seq(0, 3, 0.1))
# gib fertige Tabelle aus
tabelle1
```

Im Gegensatz zu Tabelle 3.3 zeigt Tabelle 3.4 die Überschreitungswahrscheinlichkeiten P für Abszissenwerte z der Standardnormalverteilung ($\mu = 0$ und $\sigma = 1$). Die p-Werte entsprechen der Fläche unter der Standardnormalverteilung zwischen z und $+\infty$ und damit einer *einseitigen* Fragestellung. Sie müssen bei *zweiseitigen* Fragestellungen verdoppelt werden!

ϕ **z**	**.00**	**.01**	**.02**	**.03**	**.04**	**.05**	**.06**	**.07**	**.08**	**.09**
0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641
0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2887	.2843	.2810	.2776
0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1464	.1423	.1401	.1379
1.1	.1357	.1335	.1314	.1291	.1271	.1251	.1230	.1210	.1190	.1170
1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
p =	0.5000	0.7500	0.8000	0.9000	0.9500	0.9750	0.9900	0.9950	0.9975	0.9990
z_p =	0.0000	0.6745	0.8416	1.2816	1.6449	1.9600	2.3263	2.5758	2.8070	3.0902

Tabelle 3.4: Überschreitungswahrscheinlichkeiten P für Abszissenwerte z der Standardnormalverteilung



Bei der z-Transformation benötigt man die Werte dieser Tabelle beispielsweise für Fragen wie "*Wie hoch ist der prozentuale Anteil für Werte, die* ***mindestens*** *X sind*"

Die Werte für Tabelle 3.4 werden erzeugt mit:

```
# erstelle Werte von 0.00 bis 3.09
z <- seq(from=0, to=3.09, by = 0.01)

# P-Werte berechnen
1 - pnorm(z)

# In Matrix mit 10 Spalten übertragen
matrix(1-pnorm(z),ncol=10, byrow=TRUE)
# auf 4 Stellen runden
matrix(round(1-pnorm(z),4),ncol=10, byrow=TRUE)

# In Dataframe übertragen
tabelle2 <- data.frame(matrix(round(1-pnorm(z),4),ncol=10, byrow=TRUE))
# benenne die Spalten
colnames(tabelle2) <- c(seq(0.00, 0.09, 0.01))
# benenne die Zeilen
row.names(tabelle2) <- c(seq(0, 3, 0.1))
# gib fertige Tabelle aus
tabelle2
```

3.7.3 Kritische Werte t^* der t-Verteilung

	Irrtumswahrscheinlichkeit $\alpha/2$ für einseitige Fragestellung					
	0.10	**0.05**	**0.025**	**0.01**	**0.005**	**0.0005**
df	Irrtumswahrscheinlichkeit α für zweiseitige Fragestellung					
	0.20	**0.10**	**0.050**	**0.020**	**0.010**	**0.0010**
1	3.078	6.314	12.706	31.821	63.657	318.309
2	1.886	2.920	4.303	6.965	9.925	22.327
3	1.638	2.353	3.182	4.541	5.841	10.215
4	1.533	2.132	2.776	3.747	4.604	7.173
5	1.476	2.015	2.571	3.365	4.032	5.893
6	1.440	1.943	2.447	3.143	3.707	5.208
7	1.415	1.895	2.365	2.998	3.499	4.785
8	1.397	1.860	2.306	2.896	3.355	4.501
9	1.383	1.833	2.262	2.821	3.250	4.297
10	1.372	1.812	2.228	2.764	3.169	4.144
11	1.363	1.796	2.201	2.718	3.106	4.025
12	1.356	1.782	2.179	2.681	3.055	3.930
13	1.350	1.771	2.160	2.650	3.012	3.852
14	1.345	1.761	2.145	2.624	2.977	3.787
15	1.341	1.753	2.131	2.602	2.947	3.733
16	1.337	1.746	2.120	2.583	2.921	3.686
17	1.333	1.740	2.110	2.567	2.898	3.646
18	1.330	1.734	2.101	2.552	2.878	3.611
19	1.328	1.729	2.093	2.539	2.861	3.579
20	1.325	1.725	2.086	2.528	2.845	3.552
21	1.323	1.721	2.080	2.518	2.831	3.527
22	1.321	1.717	2.074	2.508	2.819	3.505
23	1.319	1.714	2.069	2.500	2.807	3.485
24	1.318	1.711	2.064	2.492	2.797	3.467
25	1.316	1.708	2.060	2.485	2.787	3.450
26	1.315	1.706	2.056	2.479	2.779	3.435
27	1.314	1.703	2.052	2.473	2.771	3.421
28	1.313	1.701	2.048	2.467	2.763	3.408
29	1.311	1.699	2.045	2.462	2.756	3.396
30	1.310	1.697	2.042	2.457	2.750	3.385
40	1.303	1.684	2.021	2.423	2.704	3.307
50	1.299	1.676	2.009	2.403	2.678	3.261
60	1.296	1.671	2.000	2.390	2.660	3.232
70	1.294	1.667	1.994	2.381	2.648	3.211
80	1.292	1.664	1.990	2.374	2.639	3.195
90	1.291	1.662	1.987	2.368	2.632	3.183
100	1.290	1.660	1.984	2.364	2.626	3.174
200	1.286	1.653	1.972	2.345	2.601	3.131
300	1.284	1.650	1.968	2.339	2.592	3.118
400	1.284	1.649	1.966	2.336	2.588	3.111
500	1.283	1.648	1.965	2.334	2.586	3.107
∞	1.282	1.645	1.960	2.326	2.576	3.090

Tabelle 3.5: Kritische Werte t^* der t-Verteilung

Die Werte für Tabelle 3.5 werden erzeugt mit:

```
# erstelle Werte von 1 bis 100, und 200, 300, 400, 500
df <- c(seq(from=1, to=100, by = 1), 200, 300, 400, 500)

# für alpha= 0.2
qt(0.9,df=df)

# für alpha= 0.1
qt(0.95,df=df)

```

```
10  # für alpha= 0.05
11  qt(0.975,df=df)
12
13  # für alpha= 0.02
14  qt(0.99,df=df)
15
16  # für alpha= 0.01
17  qt(0.995,df=df)
18
19  # für alpha= 0.001
20  qt(0.9995,df=df)
21
22  # erstelle Datenframe
23  tabelle_t <- data.frame(qt(0.9,df=df), qt(0.95,df=df), qt(0.975,df=df), qt(0.99,df=df), qt(0.995,df=df),
    qt(0.9995,df=df))
24  # benenne Spalten
25  colnames(tabelle_t) <- c("0.01","0.05","0.025","0.01","0.005","0.0005")
26  # benenne Zeilen
27  row.names(tabelle_t) <- df
28  # gib Tabelle aus
29  tabelle_t
```

3.7.4 Kritische Werte der Chiquadrat-Verteilung

df	Irrtumswahrscheinlichkeit α					
	0.10	**0.05**	**0.025**	**0.01**	**0.005**	**0.0005**
1	2.71	3.84	5.02	6.63	7.88	12.12
2	4.61	5.99	7.38	9.21	10.60	15.20
3	6.25	7.81	9.35	11.34	12.84	17.73
4	7.78	9.49	11.14	13.28	14.86	20.00
5	9.24	11.07	12.83	15.09	16.75	22.11
6	10.64	12.59	14.45	16.81	18.55	24.10
7	12.02	14.07	16.01	18.48	20.28	26.02
8	13.36	15.51	17.53	20.09	21.95	27.87
9	14.68	16.92	19.02	21.67	23.59	29.67
10	15.99	18.31	20.48	23.21	25.19	31.42
11	17.28	19.68	21.92	24.72	26.76	33.14
12	18.55	21.03	23.34	26.22	28.30	34.82
13	19.81	22.36	24.74	27.69	29.82	36.48
14	21.06	23.68	26.12	29.14	31.32	38.11
15	22.31	25.00	27.49	30.58	32.80	39.72
16	23.54	26.30	28.85	32.00	34.27	41.31
17	24.77	27.59	30.19	33.41	35.72	42.88
18	25.99	28.87	31.53	34.81	37.16	44.43
19	27.20	30.14	32.85	36.19	38.58	45.97
20	28.41	31.41	34.17	37.57	40.00	47.50
21	29.62	32.67	35.48	38.93	41.40	49.01
22	30.81	33.92	36.78	40.29	42.80	50.51
23	32.01	35.17	38.08	41.64	44.18	52.00
24	33.20	36.42	39.36	42.98	45.56	53.48
25	34.38	37.65	40.65	44.31	46.93	54.95
26	35.56	38.89	41.92	45.64	48.29	56.41
27	36.74	40.11	43.19	46.96	49.64	57.86
28	37.92	41.34	44.46	48.28	50.99	59.30
29	39.09	42.56	45.72	49.59	52.34	60.73
30	40.26	43.77	46.98	50.89	53.67	62.16
31	41.42	44.99	48.23	52.19	55.00	63.58
32	42.58	46.19	49.48	53.49	56.33	65.00
33	43.75	47.40	50.73	54.78	57.65	66.40
34	44.90	48.60	51.97	56.06	58.96	67.80
35	46.06	49.80	53.20	57.34	60.27	69.20
36	47.21	51.00	54.44	58.62	61.58	70.59
37	48.36	52.19	55.67	59.89	62.88	71.97
38	49.51	53.38	56.90	61.16	64.18	73.35
39	50.66	54.57	58.12	62.43	65.48	74.73
40	51.81	55.76	59.34	63.69	66.77	76.09

Tabelle 3.6: Kritische Werte der χ^2-Verteilung

Die Werte für Tabelle 3.6 werden erzeugt mit:

```
# werte von 1 bis 40
df <- c(seq(from=1, to=40, by = 1))
# chi^2-Werte für unterschiedliche alpha berechnen
tabelle_chi <- data.frame(qchisq(0.9,df=df), qchisq(0.95,df=df), qchisq(0.975,df=df), qchisq(0.99,df=df),
qchisq(0.995,df=df), qchisq(0.9995,df=df))
# benenne Spalten nach alpha-Werten
colnames(tabelle_chi) <- c("0.01","0.05","0.025","0.01","0.005","0.0005")
# benenne Zeilen nach df-Werten
row.names(tabelle_chi) <- df
# gib Tabelle aus
tabelle_chi
```

3.8 COVID19 Fallzahlen analysieren

Ich möchte die COVID19-Fallahlen analysieren.

Hierfür gibt es eine schöne Anleitung von der Universität Toronto:
https://mdl.library.utoronto.ca/technology/tutorials/covid-19-data-r

Zunächst holen wir uns die aktuellen Daten.

```
# aktiviere das Tidyverse
library(tidyverse)

# Importiere Johns Hopkins Github data
confirmedraw <- read.csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv")
deathsraw <- read.csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv")
recoveredraw <- read.csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_recovered_global.csv")
```

Dann bringen wir sie ins richtige Format.

```
confirmed <- confirmedraw %>% gather(key="date", value="confirmed", -c(Country.Region, Province.State, Lat, Long)) %>% group_by(Country.Region, date) %>% summarize(confirmed=sum(confirmed))
deaths <- deathsraw %>% gather(key="date", value="deaths", -c(Country.Region, Province.State, Lat, Long)) %>% group_by(Country.Region, date) %>% summarize(deaths=sum(deaths))
recovered <- recoveredraw %>% gather(key="date", value="recovered", -c(Country.Region, Province.State, Lat, Long)) %>% group_by(Country.Region, date) %>% summarize(recovered=sum(recovered))
summary(confirmed)
```

```
 Country.Region         date             confirmed
 Length:83712       Length:83712       Min.   :       0
 Class :character   Class :character   1st Qu.:     141
 Mode  :character   Mode  :character   Median :    4521
                                       Mean   :  208962
                                       3rd Qu.:   58988
                                       Max.   :30539759
```

Jetzt kombinieren wir alles in ein Datenframe und korrigieren die Datumsangaben

```
# Final data: combine all three
country <- full_join(confirmed, deaths) %>% full_join(recovered)
# Date variable
# repariere Datumsangaben von character nach date
country$date <- country$date %>% sub("X", "", .) %>% as.Date("%m.%d.%y")

# Neue variable: Anzahl der Tage
country <- country %>% group_by(Country.Region) %>% mutate(cumconfirmed=cumsum(confirmed), days = date - first(date) + 1)

```

Jetzt aggregieren wir auf Weltperspektive und Deutschland.

```
world <- country %>% group_by(date) %>% summarize(confirmed=sum(confirmed), cumconfirmed=sum(cumconfirmed), deaths=sum(deaths), recovered=sum(recovered)) %>% mutate(days = date - first(date) + 1)
# Extract specific country: Germany
germany <- country %>% dplyr::filter(Country.Region=="Germany")
```

So vorbereitet können wir Statistiken ausgeben ...

```
# SUMMARY STATISTICS
summary(country)
by(country$confirmed, country$Country.Region, summary)
by(country$cumconfirmed, country$Country.Region, summary)
by(country$deaths, country$Country.Region, summary)
by(country$recovered, country$Country.Region, summary)
summary(world)
summary(germany)
```

...und Grafiken plotten

```
# World confirmed
ggplot(world, aes(x=date, y=confirmed)) + geom_bar(stat="identity", width=0.1) +
  theme_classic() +
  labs(title = "Covid-19 Global Confirmed Cases", x= "Date", y= "Daily confirmed cases") +
  theme(plot.title = element_text(hjust = 0.5))
```

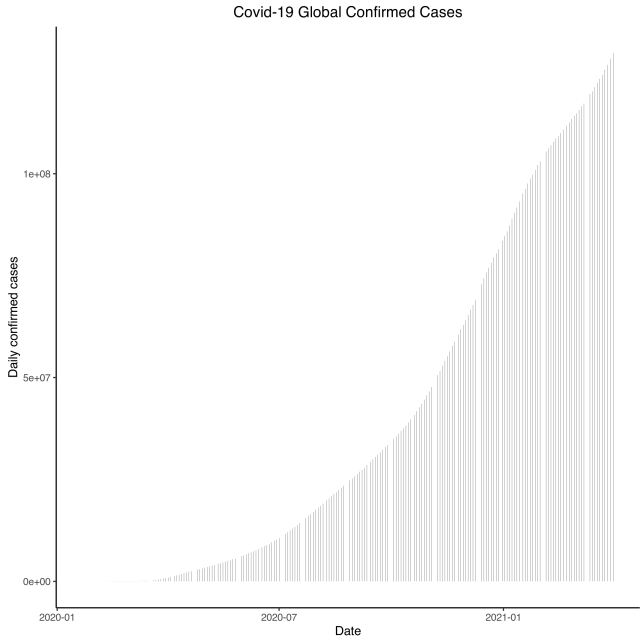


Abb. 3.33

```
# Germany confirmed
ggplot(germany, aes(x=date, y=confirmed)) + geom_bar(stat="identity", width=0.1) +
  labs(title = "Covid-19 Confirmed Cases in Germany", x= "Date", y= "Daily confirmed cases") +
  theme(plot.title = element_text(hjust = 0.5))
```

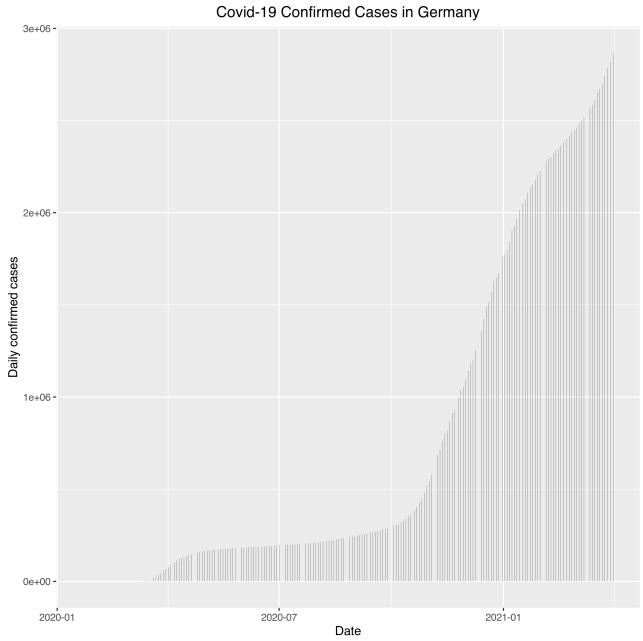


Abb. 3.34

```
# Line graph of cases over time
# World confirmed
ggplot(world, aes(x=days, y=confirmed)) + geom_line() +
  labs(title = "Covid-19 Global Confirmed Cases", x= "Days", y= "Daily confirmed cases") +
  theme(plot.title = element_text(hjust = 0.5))
# Ignore warning
```

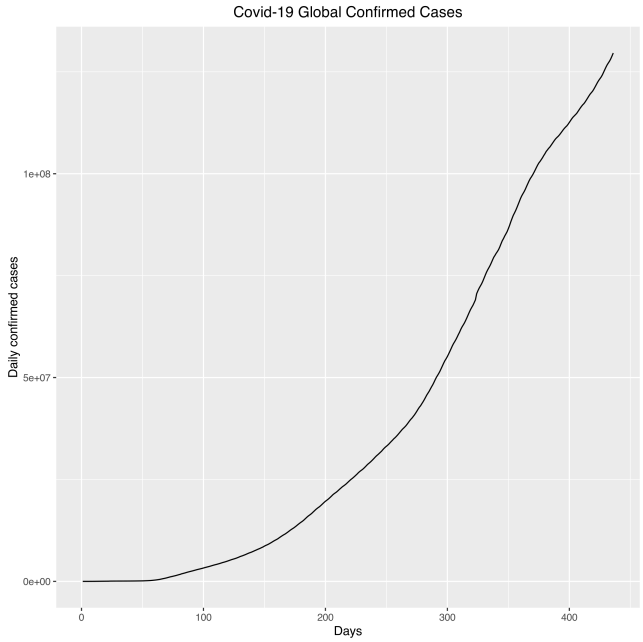


Abb. 3.35

```
# World confirmed with counts in log10 scale
ggplot(world, aes(x=days, y=confirmed)) + geom_line() +
  labs(title = "Covid-19 Global Confirmed Cases", x= "Days", y= "Daily confirmed cases  (log scale)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(trans="log10")
```

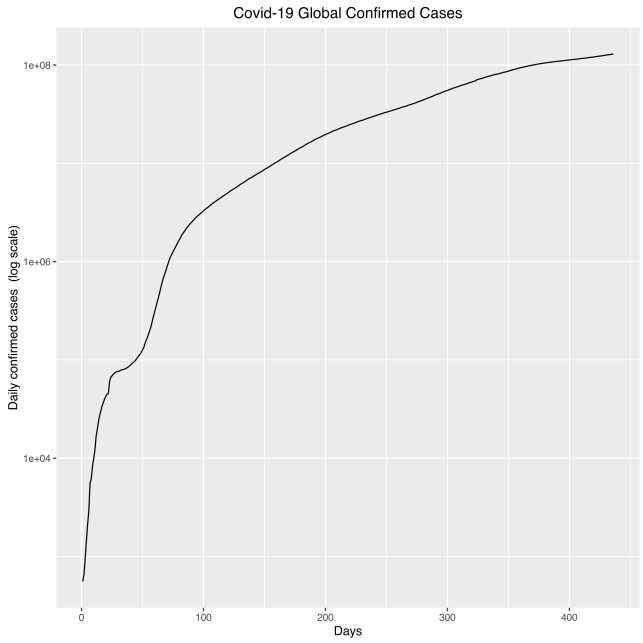


Abb. 3.36

```
# Confirmed by country for select countries with counts in log10 scale
countryselection <- country %>% filter(Country.Region==c("US", "Italy", "China", "France", "United Kingdom",
"Germany"))
ggplot(countryselection, aes(x=days, y=confirmed, colour=Country.Region)) + geom_line(size=1) +
  labs(title = "Covid-19 Confirmed Cases by Country", x= "Days", y= "Daily confirmed cases (log scale)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(trans="log10")
```

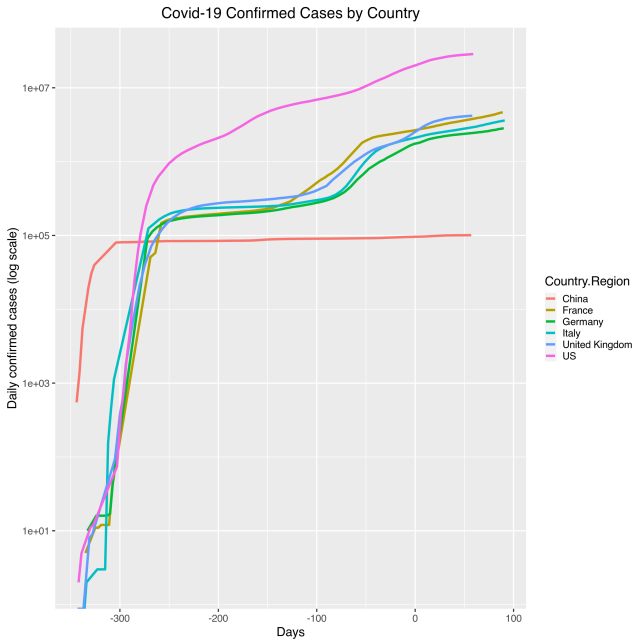


Abb. 3.37

```r
# Matrix of line graphs of confirmed, deaths and recovered for select countries in log10 scale
countryselection %>% gather("Type", "Cases", -c(date, days, Country.Region)) %>%
  ggplot(aes(x=days, y=Cases, colour=Country.Region)) + geom_line(size=1) +
  labs(title = "Covid-19 Cases by Country", x= "Days", y= "Daily cases (log scale)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(trans="log10") +
  facet_grid(rows=vars(Type))
```

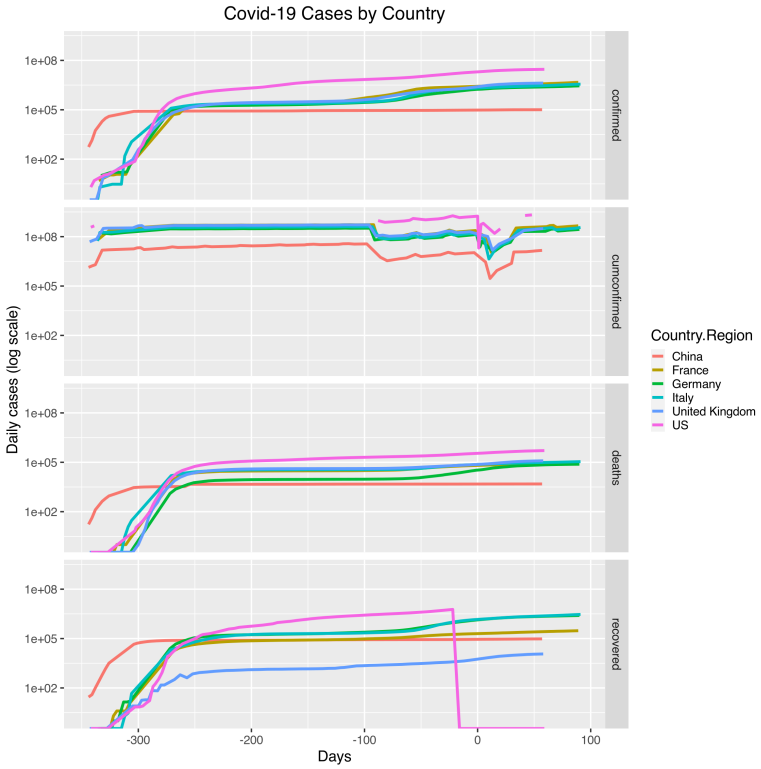


Abb. 3.38

4 Übungsaufgaben

In diesem Kapitel werden Übungsaufgaben zu verschiedenen Teilen der Statistik vorgestellt und gelöst.

Die Aufgaben stammen von Gimeno et al. (2022)[13][1]. Dort werden die Lösungswege nur teilweise und nur unter Verwendung der Software RKWard[2], aber ohne konkreten -Code besprochen.

Versuchen Sie zunächst selbst eine Lösung zu finden bevor Sie sich die Auflösungen anschauen. Die Aufgaben und Lösungen stehen auch als Quartodokument auf GitHub zur Verfügung, siehe https://github.com/produnis/angewandte_uebungen_in_R.

4.1 Aufgaben

Die Aufgaben sind nach unterschiedlichen Bereichen der Statistik gegliedert.

4.1.1 Häufigkeitsverteilungen

4.1.1.1 Aufgabe 4.1.1.1 Kinder in Familien

Für 25 Familien liegt die Anzahl an Kindern vor:

1, 2, 4, 2, 2, 2, 3, 2, 1, 1, 0, 2, 2, 0, 2, 2, 1, 2, 2, 3, 1, 2, 2, 1, 2

a) Erstellen Sie ein Datenframe mit der Variable `Kinder` und übertragen Sie die Daten.
b) Erzeugen Sie eine einfache Häufigkeitstabelle
c) Erzeugen Sie ein Balkendiagramm der Häufigkeiten
d) Erzeugen Sie eine vollständige Häufigkeitstabelle, inklusive absoluter, relativer und jeweils kumulativer Häufigkeiten

Für die Lösung siehe Abschnitt 4.2.1

[1] siehe https://github.com/asalber/statistics_practice_rkteaching
[2] siehe https://rkward.kde.org/

4.1.1.2 Aufgabe 4.1.1.2 Patienten in der Notaufnahme

Den gesamten November über wurde die Anzahl an Patienten in der Notaufnahme erhoben

15 23 12 10 28 50 12 17 20 21 18 13 11 12 26
30 6 16 19 22 14 17 21 28 9 16 13 11 16 20

a) Erstellen Sie ein Datenframe mit der Variable `Patienten` und übertragen Sie die Daten.
b) Erzeugen Sie ein Boxplot. Gibt es Ausreißer? Wenn ja, entfernen Sie diese, bevor Sie weitermachen.
c) Erzeugen Sie eine Häufigkeitstabelle, welche die Daten in 5 Klassen gruppiert.
d) Erzeugen Sie ein Histogram der klassierten absoluten Häufigkeiten.
e) Erzeugen Sie ebenso Histogramme der relativen und jeweils kumulativen Häufigkeiten, inklusive Polygonzügen.

Für die Lösung siehe Abschnitt 4.2.2

4.1.1.3 Aufgabe 4.1.1.3 Blutgruppen

Von 30 Personen wurden die Blutgruppen wie folgt bestimmt:

A, B, B, A, AB, 0, 0, A, B, B, A, A, A, A, AB,
A, A, A, B, 0, B, B, B, A, A, A, 0, A, AB, 0.

a) Erstellen Sie ein Datenframe mit der Variable `Blutgruppe` und übertragen Sie die Daten.
b) Erzeugen Sie eine Häufigkeitstabelle
c) Erzeugen Sie ein Kreisdiagramm

Für die Lösung siehe Abschnitt 4.2.3

4.1.1.4 Aufgabe 4.1.1.4 Familienstand

Das Alter und der Familienstand von 28 Personen wurden wie folgt erhoben:

Familienstand	Alter								
Single	31	45	35	65	21	38	62	22	31
Verheiratet	72	39	62	59	25	44	54		

Verwitwet	80	68	65	40	78	69	75
Geschieden	31	65	59	58	50		

a) Erstellen Sie ein Datenframe mit den Variablen `Alter` und `Familienstand` und übertragen Sie die Daten.
b) Erzeugen Sie für jeden `Familienstand` eine Häufigkeitstabelle des `Alters`.
c) Erzeugen Sie für jeden `Familienstand` eine Boxplot des `Alters`. Gibt es Ausreißer? In welcher Gruppe streut das Alter am meisten?
d) Erzeugen Sie für jeden `Familienstand` eine Histogram des `Alters`. Wie unterscheiden sich die Histogramme?

Für die Lösung siehe Abschnitt 4.2.4

4.1.1.5 Aufgabe 4.1.1.5 Handballverletzungen

Die Anzahl der Verletzungen von Handballspielern eines Teams wurden wie folgt erhoben:

0, 1, 2, 1, 3, 0, 1, 0, 1, 2, 0, 1, 1, 1, 2, 0, 1, 3, 2, 1, 2, 1, 0, 1

a) Erstellen Sie eine Häufigkeitstabelle
b) Erzeugen Sie ein Säulendiagramm der relativen und kumulativen relativen Häufigkeiten.
c) Erzeugen Sie ein Boxplot

Für die Lösung siehe Abschnitt 4.2.5

4.1.1.6 Aufgabe 4.1.1.6 Körpergröße

Von 30 Studierenden wurde die Körpergröße gemessen

179, 173, 181, 170, 158, 174, 172, 166, 194, 185,
162, 187, 198, 177, 178, 165, 154, 188, 166, 171,
175, 182, 167, 169, 172, 186, 172, 176, 168, 187

a) Erstellen Sie ein Histogram der Körpergröße mit Klassen von 150cm bis 200cm, die jeweils 10cm breit sind.
b) Gibt es Ausreißer?

Für die Lösung siehe Abschnitt 4.2.6

4.1.1.7 Aufgabe 4.1.1.7 Neugeborene

Der Datensatz `neonates` von rk.Teaching[3] enthält Informationen über eine Stichprobe von 320 Neugeborenen, die im Laufe eines Jahres nach normaler Schwangerschaftsdauer geboren wurden.

a) Erstellen Sie die Häufigkeitstabelle des APGAR-Scores nach 1 Minute. Wenn ein Score von 3 oder weniger anzeigt, dass das Neugeborene in einem kritischen Zusatand ist, wie viel Prozent der Neugeborenen in der Stichprobe sind dann in einem kritischen Zustand?

b) Erstellen Sie die Häufigkeitstabelle des Geburtsgewichts der Neugeborenen, indem Sie die Daten in Klassen mit einer Breite von 0,5 kg von 2 bis 4,5 kg einteilen. Welches Intervall enthält die meisten Neugeborenen?

c) Vergleichen Sie die Häufigkeitsverteilung des APGAR-Scores nach 1 Minute für Mütter unter 20 Jahren und für Mütter über 20 Jahren. Welche Gruppe hat mehr Neugeborene in kritischem Zustand?

d) Vergleichen Sie die relative Häufigkeitsverteilung des Geburtsgewichts der Neugeborenen, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Wenn ein Gewicht unter 2,5 kg als niedriges Gewicht gilt, welche Gruppe hat einen höheren Prozentsatz an Neugeborenen mit niedrigem Gewicht?

e) Berechnen Sie die Prävalenz von Neugeborenen mit niedrigem Gewicht für Mütter, die vor der Schwangerschaft geraucht haben, und den Nichtraucherinnen.

f) Berechnen Sie das relative Risiko eines niedrigen Geburtsgewichts des Neugeborenen, wenn die Mutter während der Schwangerschaft raucht, im Vergleich dazu, wenn die Mutter nicht raucht.

g) Erstellen Sie ein Balkendiagramm des APGAR-Scores nach 1 Minute. Welcher Score ist am häufigsten?

h) Erstellen Sie das Balkendiagramm der kumulierten relativen Häufigkeit des APGAR-Scores nach 1 Minute. Unter welchem Wert liegen die Hälfte der Neugeborenen?

i) Vergleichen Sie die Balkendiagramme der relativen Häufigkeitsverteilungen des APGAR-Scores nach 1 Minute, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Welche Schlussfolgerungen können gezogen werden?

j) Erstellen Sie ein Histogramm der Geburtsgewichte der Neugeborenen mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg. Welche Klasse enthält die meisten Neugeborenen?

k) Vergleichen Sie die relativen Häufigkeitshistogramme der Geburtsgewichte der Neugeborenen, mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Welche Gruppe hat Neugeborene mit geringeren Gewichten?

l) Vergleichen Sie die relativen Häufigkeitshistogramme der Geburtsgewichte der Neugeborenen, mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg, je nachdem, ob die Mutter vor der Schwangerschaft geraucht hat oder nicht. Welche Schlussfolgerungen können gezogen werden?

m) Erstellen Sie ein Boxplot der Geburtsgewichte der Neugeborenen. Welcher Gewichtsbereich kann in der Stichprobe als normal angesehen werden? Gibt es Ausreißer in der Stichprobe?

n) Vergleichen Sie die Boxplots der Geburtsgewichte der Neugeborenen je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht und ob die Mutter unter 20 oder über 20 Jahre alt war. Welche Gruppe hat eine größere zentrale Streuung? Welche Gruppe hat Neugeborene mit geringerem Gewicht?

o) Vergleichen Sie die Boxplots der APGAR-Scores nach 1 Minute und nach 5 Minuten. Welche Variable hat eine größere zentrale Streuung?

Für die Lösung siehe Abschnitt 4.2.7

4.1.2 Stichprobenstatistik

Bei diesen Aufgaben geht es vor allem um Lage- und Streuungskenngrößen.

4.1.2.1 Aufgabe 4.1.2.1 Kinder in Familien

Die Anzahl an Kindern in einer Stichprobe aus 25 Familien sind:

1, 2, 4, 2, 2, 2, 3, 2, 1, 1, 0, 2, 2, 0, 2, 2, 1, 2, 2, 3, 1, 2, 2, 1, 2

a) Erstellen Sie ein Datenframe mit der Variable `Kinder` und übertragen Sie die Daten.
b) Berechnen Sie das arithmetische Mittel, die Varianz sowie die Standardabweichung für die Anzahl an Kindern.
c) Berechnen Sie die Quartile, die Spannweite, den Interquartilsabstand, das dritte Dezil sowie das 68te Perzentil.

Für die Lösung siehe Abschnitt 4.2.8

[3] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/neonates.RData

4.1.2.2 Aufgabe 4.1.2.2 Patienten in Notaufnahme

Den gesamten November über wurde die Anzahl an Patienten in der Notaufnahme erhoben

15	23	12	10	28	50	12	17	20	21	18	13	11	12	26
30	6	16	19	22	14	17	21	28	9	16	13	11	16	20

a) Erstellen Sie ein Datenframe mit der Variable `Patienten` und übertragen Sie die Daten.
b) Berechnen Sie das arithmetische Mittel, die Varianz, die Standardabweichung und den Variationskoeffizienten.
c) Berechnen Sie die Skewness (Schiefe) und Kurtosis ("Spitzigkeit") und interpretieren Sie die Werte.

Für die Lösung siehe Abschnitt 4.2.9

4.1.2.3 Aufgabe 4.1.2.3 Studierendenbewertung

Im letzten R-Kurs haben 20 Studierende folgende Abschlussbewertungen erhalten

SS, AP, SS, AP, AP, NT, NT, AP, SB, SS
SB, SS, AP, AP, NT, AP, SS, NT, SS, NT

a) Erstellen Sie ein Datenframe mit der Variable `Bewertung` und übertragen Sie die Daten.
b) Wandeln Sie die `Bewertung` in Punkte um, nach dem Schema "`SS`" = 2,5 | "`AP`" = 6 | "`NT`" = 8 | "`SB`" = 9,5.
c) Bestimmen Sie den Median und den Interquartilsabstand.

Für die Lösung siehe Abschnitt 4.2.10

4.1.2.4 Aufgabe 4.1.2.4 Körpergröße nach Geschlecht

Von 30 Studierenden wurde die Körpergröße wie folgt gemessen:

Geschlecht	Größe
weiblich	173, 158, 174, 166, 162, 177, 165, 154, 166, 182, 169, 172, 170, 168
männlich	179, 181, 172, 194, 185, 187, 198, 178, 188, 171, 175, 167, 186, 172, 176, 187

a) Erstellen Sie ein Datenframe mit den Variable `Geschlecht` und `Koerpergroesse` und übertragen Sie die Daten.
b) Bestimmen Sie in Abhängigkeit zum `Geschlecht` das arithmetische Mittel, den Median, die Varianz, die Standardabweichung sowie die Quartile.

Für die Lösung siehe Abschnitt 4.2.11

4.1.2.5 Aufgabe 4.1.2.5 Handballverletzungen

Die Anzahl der Verletzungen von Handballspielern eines Teams wurden wie folgt erhoben:

0, 1, 2, 1, 3, 0, 1, 0, 1, 2, 0, 1, 1, 1, 2, 0, 1, 3, 2, 1, 2, 1, 0, 1

a) Bestimmen Sie das arithmetische Mittel, den Median, die Varianz sowie die Standardabweichung der Verletzungen.
b) Bestimmen Sie die Skewness und Kortosis der Verteilung.
c) Berechnen Sie das vierte und achte Dezil der Verteilung.

Für die Lösung siehe Abschnitt 4.2.12

4.1.2.6 Aufgabe 4.1.2.6 Blutdruckmessung

Wir möchten die Zuverlässigkeit zweier Blutdruckmonitore bestimmen. Gerät 1 misst den Blutdruck am Handgelenk, Gerät 2 am Unterarm. Es wurden 8 Messungen mit jedem Gerät bei der selben Person durchgeführt, wobei folgende systolischen Werte gemessen wurden:

Position	Messdaten
Unterarm	111, 109, 112, 111, 113, 113, 114, 111
Handgelenk	115, 113, 117, 116, 112, 112, 117, 112

Welcher Monitor funktioniert besser?

Für die Lösung siehe Abschnitt 4.2.13

4.1.2.7 Aufgabe 4.1.2.7 Alter und Familienstand

Das Alter und der Familienstand von 28 Personen wurden wie folgt erhoben:

Familienstand	Alter								
Single	31	45	35	65	21	38	62	22	31
Verheiratet	72	39	62	59	25	44	54		
Verwitwet	80	68	65	40	78	69	75		
Geschieden	31	65	59	58	50				

a) Bestimmen Sie das arithmetische Mittel, den Median, die Varianz sowie die Standardabweichung des `Alters` für jeden `Familienstand`.
b) Welche Gruppe hat den "besten" Mittelwert?

Für die Lösung siehe Abschnitt 4.2.14

4.1.2.8 Aufgabe 4.1.2.8 Tabak, Alkohol und Blutdruck

Eine Studie möchte den möglichen Zusammenhang zwischen dem Blutdruck und dem Alkohol- und Tabakkonsum untersuchen. Hierzu wurden folgende Daten von 25 Personen erhoben.

Rauchen	ja	nein	ja	ja	ja	nein	nein	ja	nein	ja	nein	ja	nein
Alkohol	nein	nein	ja	ja	nein	nein	ja	ja	nein	ja	nein	ja	ja
Blutdruck	80	92	75	56	89	93	101	67	89	63	98	58	91

Rauchen	ja	nein	nein	ja	nein	nein	nein	ja	nein	ja	nein	ja
Alkohol	ja	nein	ja	ja	nein	nein	ja	ja	ja	nein	ja	nein
Blutdruck	71	52	98	104	57	89	70	93	69	82	70	49

a) Vergleichen Sie das arithmetische Mittel, die Standardabweichung, die Skewness und Kurtosis des `Blutdrucks` zwischen Rauchern und Nichtrauchern.
b) Vergleichen Sie die selben Werte zwischen der Alkohol- und Nicht-Alkoholgruppe.
c) Vergleichen Sie die selben Werte zwischen der Raucher- und Alkoholgruppe, zwischen der Raucher- und Nicht-Alkoholgruppe, der Nichtraucher- und Alkoholgruppe sowie der Nichtraucher- und Nicht-Alkoholgruppe.

Für die Lösung siehe Abschnitt 4.2.14

4.1.3 Lineare Regression

4.1.3.1 Aufgabe 4.1.3.1 X und Y

Bei 10 Personen wurden `x` und `y` erhoben.

x	0	1	2	3	4	5	6	7	8	9
y	2	5	8	11	14	17	20	23	26	29

a) Erstellen Sie ein Datenframe mit den Variablen `x` und `y`.
b) Erzeugen Sie ein Scatterplot von `x` und `y`. Bestimmen Sie anhand des Plots, welche Regressionsfunktion die Daten am besten erklären würde.
c) Führen Sie die Regression durch.
d) Fügen Sie die Regressionsfunktion `y erklärt durch x` dem Plot hinzu.
e) Fügen Sie die Regressionsfunktion `x erklärt durch y` ebenfalls dem Plot hinzu, aber in roter Farbe.
f) Wie groß sind die Residuen?

Für die Lösung siehe Abschnitt 4.2.16

4.1.3.2 Aufgabe 4.1.3.2 Lernen und Durchfallen

Eine Studie gibt vor, den Zusammenhang zwischen den täglichen Lernstunden und der Anzahl nicht bestandener Prüfungen im Semester zu untersuchen. Bei 30 Studierenden wurden folgende Werte erhoben:

Lernzeit	durchgefallen	Lernzeit	durchgefallen	Lernzeit	durchgefallen
3.5	1	2.2	2	1.3	4
0.6	5	3.3	0	3.1	0
2.8	1	1.7	3	2.3	2
2.5	3	1.1	3	3.2	2
2.6	1	2.0	3	0.9	4
3.9	0	3.5	0	1.7	2
1.5	3	2.1	2	0.2	5
0.7	3	1.8	2	2.9	1
3.6	1	1.1	4	1.0	3
3.7	1	0.7	4	2.3	2

a) Erstellen Sie ein Datenframe mit den Variablen `Lernen` und `Durchgefallen`.
b) Erzeugen Sie eine Kreuztabelle der Variablen `Lernen` und `Durchgefallen`.
c) Führen Sie eine lineare Regression `Durchgefallen erklärt durch Lernen` durch und plotten Sie Ihr Ergebnis.
d) Wie lauten die Regressionskoeffizient des Modells, und wie ist er zu interpretieren?

e) Ist das soeben erstellte Modell besser als das in Abschnitt 4.1.3.1 berechnete? Vergleichen Sie zur Beantwortung die Residuen beider Modelle.
f) Berechnen Sie den linearen Bestimmungskoeffizient und den Korrelationskoeffizient. Ist das lineare Modell ein gutes Modell, um die Beziehung zwischen den gescheiterten Prüfungen und den täglichen Studienzeiten zu erklären? Wie viel Prozent der Variabilität der durchgefallenen Prüfungen wird durch das lineare Modell erklärt?
g) Benutzen Sie das lineare Modell, um die Anzahl an durchgefallenen Prüfungen für einen Studierenden zu bestimmen, der 3 Stunden Lernzeit investiert hat. Wie glaubwürdig ist die Vorhersage?
h) Wie viele Stunden Lernzeit wird benötigt, um alle Kurse zu bestehen?

Für die Lösung siehe Abschnitt 4.2.17

4.1.3.3 Aufgabe 4.1.3.3 Metabolismus

Um herauszufinden, wie der Körper Alkohol verstoffwechselt, hat ein Proband einen Liter Wein zügig getrunken. Anschließend wurde alle 30 Minuten der Blutalkoholspiegel gemessen.

Minuten	30	60	90	120	150	180	210
Alkohol (g/l)	1.6	1.7	1.5	1.1	0.7	0.2	2.1

a) Erstellen Sie ein Datenframe mit den Variablen `Minuten` und `Alkohol`.
b) Bestimmen Sie den passenden Korrelationskoeffizienten. Werden die Daten ausreichend gut durch das Modell beschrieben?
c) Plotten Sie das lineare Regressionsmodell `Alkohol erklärt durch Minuten`. Gibt es Punkte mit großen Residuen? Wenn ja, entfernen Sie diese und führen die Berechnungen erneut durch. Hat sich der Korrelationskoeffizient verbessert?
d) Mit welcher Geschwindigkeit wird der Alkohol pro Minute verstoffwechselt?
e) Wenn es gesetzlich erlaubt wäre, mit einem Blutalkoholwert von $0,3$ g/l Auto zu fahren, wie lange muss die Person warten, nachdem sie 1 Liter Weingetrunken hat, um wieder fahrtüchtig zu sein? Wie zuverlässig ist diese Vorhersage?

Für die Lösung siehe Abschnitt 4.2.18

4.1.3.4 Aufgabe 4.1.3.4 Alter und Körpergröße

Im Datensatz `age.height` von rk.Teaching[4] sind Alter und Körpergröße von 30 Probanden enthalten.

a) Laden Sie den Datensatz `age.height` in Ihre R-Session.
b) Berechnen Sie die Regressionsgerade `Größe erklärt durch Alter`. Ist das lineare Modell geeignet, den Zusammenhang zwischen `Alter` und `Körpergröße` zu erklären?
c) Erstellen Sie eine Punktwolke inklusive der Regressionsgeraden. Ab welchem Alter ändert sich die Punktetendenz?
d) Erstellen Sie eine Gruppierungsvariable, welche `Alter` in einen ordinalen Faktor mit den Ausprägungen "`jünger als 20`" und "`20 und älter`" einteilt.
e) Führen Sie die lineare Regressionsanalyse für beide Gruppen erneut durch. In welcher Gruppe wird der Zusammenhang zwischen `Alter` und `Körpergröße` am besten erklärt?
f) Plotten Sie die Modelle.
g) Welche Körpergröße sagt Ihr Modell für eine 14jährige Person vorher, und welche für eine 38jährige Person?

Für die Lösung siehe Abschnitt 4.2.19

4.1.3.5 Aufgabe 4.1.3.5 Wirksamkeitsverlust

Eine Studie Untersucht den Wirksamkeitsverlust eines Medikaments, das über Jahre von vielen Probanden eingenommen wurde. Folgende Aussagen zur Wirksamkeit konnten aus den Daten ermittelt werden.

Jahr	1	2	3	4	5
Wirksamkeit (%)	96	84	70	58	52

a) Führen Sie eine lineare Regression `Wirksamkeit erklärt durch Jahr` durch und plotten Sie Ihr Ergebnis.
b) Wie große ist der jährliche Wirksamkeitsverlust in %?
c) Nach wie vielen Jahren ist die Wirksamkeit bei 80%, und nach wie vielen bei 0%? Sind beide Werte gleich zuverlässig?

Für die Lösung siehe Abschnitt 4.2.20

[4] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/age.height.RData

4.1.3.6 Aufgabe 4.1.3.6 Dosierung

In einer Studie über die Wirkung verschiedener Dosen eines Medikaments erhielten 2 Patienten 2 mg und benötigten 5 Tage zur Heilung, 4 Patienten erhielten 2 mg und benötigten 6 Tage zur Heilung, 2 Patienten erhielten 3 mg und benötigten 3 Tage zur Heilung, 4 Patienten erhielten 3 mg und benötigten 5 Tage zur Heilung, 1 Patient erhielt 3 mg und benötigte 6 Tage zur Heilung, 5 Patienten erhielten 4 mg und benötigten 3 Tage zur Heilung und 2 Patienten erhielten 4 mg und benötigten 5 Tage zur Heilung.

a) Berechnen Sie die Regressionsgerade der Heilungstage in Abhängigkeit von der Dosis.
b) Berechnen Sie den Regressionskoeffizienten der Heilungstage in Abhängigkeit von der Dosis und interpretieren Sie ihn.
c) Berechnen Sie den Korrelationskoeffizienten und interpretieren Sie ihn.
d) Bestimmen Sie die erwartete Zeit, die für die Heilung mit einer Dosis von 5 mg benötigt wird. Ist diese Vorhersage zuverlässig? Begründen Sie die Antwort.
e) Welche Dosis muss angewendet werden, um in 4 Tagen zu heilen? Ist diese Vorhersage zuverlässig? Begründen Sie die Antwort.

Für die Lösung siehe Abschnitt 4.2.21

4.1.3.7 Aufgabe 4.1.3.7 Gewicht und Körpergröße

Im Datensatz `heigths.weights.students` von rk.Teaching[5] sind Gewicht und Körpergröße von 100 Probanden enthalten.

a) Laden Sie den Datensatz `heigths.weights.students` in Ihre R-Session.
b) Führen Sie eine lineare Regression `Gewicht erklärt durch Größe` durch.
c) Erstellen Sie eine Punktwolke inklusive Regressionsgeraden jeweils für Männer und Frauen getrennt.
d) Berechnen Sie die Bestimmtheitskoeffizienten (R^2) für beide Modelle. Welches Modell erklärt besser die Beziehung zwischen Gewicht und Größe, das der Männer oder das der Frauen? Begründen Sie die Antwort.
e) Was ist das zu erwartende Gewicht für einen Mann mit 170cm Körpergröße? Und für eine Frau der selben Größe?

Für die Lösung siehe Abschnitt 4.2.22

[5] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/heights.weights.students.RData

4.1.3.8 Aufgabe 4.1.3.8 Neugeborene

Der Datensatz `neonates` von rk.Teaching[6] enthält Informationen über eine Stichprobe von 320 Neugeborenen, die im Laufe eines Jahres nach normaler Schwangerschaftsdauer geboren wurden.

a) Erstellen Sie eine Kreuztabelle vom APGAR-Wert nach 1 Minute und dem Rauchverhalten der Mütter während der Schwangerschaft. Welche Schlüsse lassen sich ziehen?
b) Erstellen Sie eine Kreuztabelle vom APGAR-Wert nach 1 Minute und der Alterskategorie der Mütter. Welche Schlüsse lassen sich ziehen?
c) Führen Sie eine lineare Regression für `Geburtsgewicht erklärt durch Anzahl täglich gerauchter Zigaretten` durch. Gibt es einen starken linearen Zusammenhang?
d) Plotten Sie Ihre Regression. Passt die Regressionsgerade gut zur Punktwolke?
e) Wiederholen Sie die Regression, aber nutzen Sie dieses Mal nur Daten von Raucherinnen. Ist dieses Modell besser oder schlechter als das vorherige? Wieviel Gewicht verliert ein Neugeborenes nach diesem Modell pro täglich gerauchter Zigarette?
f) Welches Geburtsgewicht sagt dieses Modell für ein Neugeborenes vorher, dessen Mutter 5 Zigaretten täglich während der Schwangerschaft geraucht hat? Wieviel für eine Mutter, die 30 Zigaretten täglich raucht. Wie zuverlässich sind diese Ergebnisse?
g) Ändert sich der lineare Zusammenhang, wenn die Daten nach Altersgruppen getrennt untersucht werden?

Für die Lösung siehe Abschnitt 4.2.23

4.1.4 Nicht-lineare Regression

4.1.4.1 Aufgabe 4.1.4.1 Bakterien

Die Anzahl an Bakterien in einer Kultur vermehrt sich wie folgt:

Stunden	1	2	3	4	5	6	7	8	9
Bakterien	25	28	47	65	86	121	190	290	362

[6] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/neonates.RData

a) Erstellen Sie ein Datenframe mit den Variablen `Stunden` und `Bakterien`.
b) Erzeugen Sie ein Scatterplot. Welche Regression würden Sie auf Grundlage des Plots vorschlagen?
c) Berechnen Sie die quadratischen und exponentiellen Modelle für die Bakterienvermehrung über die Zeit.
d) Plotten Sie das bessere Modell in die Punktwolke.
e) Wie viele Bakterien werden nach dem besten Modell 3 Stunden nach Anlegen der Kultur vorhanden sein? Und nach 10 Stunden? Sind diese Vorhersagen zuverlässig?
f) Machen Sie eine möglichst zuverlässige Vorhersage über die Zeit, die benötigt wird, um 100 Bakterien in der Kultur zu haben.

Für die Lösung siehe Abschnitt 4.2.24

4.1.4.2 Aufgabe 4.1.4.2 Diät

Der Datensatz `diet` von rk.Teaching[7] enthält Informationen über eine Diätenuntersuchung. Für jede Person wurde die Anzahl der Diättage, der Gewichtsverlust und die regelmäßige körperliche Betätigung gemessen.

a) Laden Sie den Datensatz `diet` in Ihre R-Session.
b) Erstellen Sie eine Punktwolke. Welche Art von Modell erklärt auf Grundlage der Punktwolke den Gewichtsverlust pro Diättag besser?
c) Berechnen Sie das Regressionsmodell, welches den Gewichtsverlust mit der Anzahl an Diättagen am besten (im Vergleich zu anderen) erklären kann. Wird das Modell zuverlässige Vorhersagen machen?
d) Plotten Sie Ihr Modell.
e) Berechnen Sie das Regressionsmodell, das den Gewichtsverlust mit den Tagen der Diät für die Gruppe der Personen, die sich nicht regelmäßig körperlich betätigen, am besten erklären kann.
f) Wiederholen Sie die Analyse für die Gruppe, die sich regelmäßig körperlich betätigt.
g) Benutzen Sie die erstellen Modelle, um den Gewichtsverlust nach 30 und nach 100 Tagen Diät für Personen, die sich körperlich betätigen, und für solche, die dies nicht tun, vorherzusagen. Sind diese Vorhersagen zuverlässig?

Für die Lösung siehe Abschnitt 4.2.25

[7] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/diet.RData

4.1.4.3 Aufgabe 4.1.4.3 Blutkonzentration

Die Konzentration eines Arzneimittels im Blut in mg/dl hängt von der Zeit ab, wie aus den folgenden Daten hervorgeht.

Stunden	2	3	4	5	6	7	8
Konzentration	25	36	48	64	86	114	168

a) Benutzen Sie ein exponentielles Modell, um die Konzentration nach 10 Stunden vorherzusagen. Ist die Vorhersage zuverlässig?
b) Benutzen Sie ein logarithmisches Modell um zu bestimmen, nach wie vielen Stunden eine Konzentration von 100 mg/dl erreicht sein wird.

Für die Lösung siehe Abschnitt 4.2.26

4.1.5 Wahrscheinlichkeiten

4.1.5.1 Aufgabe 4.1.5.1 Glücksspiel

Lassen Sie in R ...

a) eine beliebige Poker-Spielkarte[8] ziehen.
b) 2 Münzen werfen.
c) 2 Würfeln werfen.

Für die Lösung siehe Abschnitt 4.2.27

4.1.5.2 Aufgabe 4.1.5.2 Münzwürfe

Wiederholen Sie die Zufallsexperimente und lassen Sie R 10 mal, 100 mal 1.000 mal und 1.000.000 mal zwei Münzen werfen.

a) Erstellen Sie je eine relative Häufigkeitstabelle der Ergebnisse. Wie sind die Tabellen zu bewerten?
b) Welche theoretischen Wahrscheinlichkeiten haben die möglichen Wurfergebnisse? Stimmen diese mit den beobachteten Ergebnissen überein?

[8] Den Datensatz für ein Pokerkartenspiel erhalten Sie unter https://www.produnis.de/R/data/cards.RData

Für die Lösung siehe Abschnitt 4.2.28

4.1.5.3 Aufgabe 4.1.5.3 Medizinschrank

In einem Medizinschrank befinden sich drei Boxen mit Medikament A, zwei Boxen mit Medikament B und eine Box mit Medikament C.

a) Ziehen Sie zufällig 3 Boxen, ohne zurücklegen.
b) Ziehen Sie zufällig 3 Boxen, diesmal mit zurücklegen.

Für die Lösung siehe Abschnitt 4.2.29

4.1.5.4 Aufgabe 4.1.5.4 Kinderkrankheiten

Eine epidemiologische Untersuchung wurde durchgeführt, um die Lebenszeitprävalenz von drei häufigen Kinderkrankheiten zu ermitteln: Windpocken, Masern und Röteln. Die beobachteten Häufigkeiten sind in der nachstehenden Tabelle aufgeführt.

Windpocken	Masern	Röteln	Häufigkeit
No	No	No	2654
No	No	Yes	1436
No	Yes	No	1682
No	Yes	Yes	668
Yes	No	No	1747
Yes	No	Yes	476
Yes	Yes	No	876
Yes	Yes	Yes	265

a) Erstellen Sie ein Datenframe mit den Variablen `Windpocken`, `Masern`, `Röteln` und `Häufigkeit` und übertragen Sie die Daten.
b) Erstellen Sie den Wahrscheinlichkeitsraum der Lebenszeitprävalenz.
c) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Windpocken hatte?
d) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Windpocken oder Masern hatte?
e) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Masern und Röteln hatte?
f) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person, die bereits an Masern erkrankte, nun an Windpocken erkrankt?
g) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person, die keine Masern und keine Röteln hatte, an Windpocken erkrankt?

Für die Lösung siehe Abschnitt 4.2.30

4.1.5.5 Aufgabe 4.1.5.5 Schwangerschaftstest

Ein Schwangerschaftstest, der von vielen Frauen angewendet wurde, erzielte folgende Ergebnisse.

Schwanger	Test	Häufigkeit
Nein	-	3876
Nein	+	47
Ja	-	12
Ja	+	131

a) Erstellen Sie ein Datenframe mit den Variablen `Schwanger`, `Testergebnis` und `Häufigkeit`.
b) Erstellen Sie den Wahrscheinlichkeitsraum.
c) Berechnen Sie die Prävalenz der Schwangerschaften.
d) Wie groß ist die Wahrscheinlichkeit, ein positives Testergebnis zu ziehen?
e) Bestimmen Sie die Sensitivität des Tests
f) Bestimmen Sie die Spezifität des Tests
g) Bestimmen Sie den positiv prädiktiven Wert des Tests
h) Bestimmen Sie den negativ prädiktiven Wert des Tests

Für die Lösung siehe Abschnitt 4.2.31

4.1.5.6 Aufgabe 4.1.5.6 Glückspielwahrscheinlichkeiten

Erstelle den Ereignisraum des Zufallsexperiments, das aus dem Werfen einer Münze, dem Werfen eines Würfels und dem Ziehen einer Karte aus einem spanischen Kartenspiel besteht.

Für die Lösung siehe Abschnitt 4.2.32

4.1.5.7 Aufgabe 4.1.5.7 Grippeimpfung

Die Wirksamkeit eines Grippeimpfstoffs wurde an 1.000 Probanden erprobt.

Impfung	Grippe	Häufigkeit
Nein	Nein	418
Nein	Ja	312
Ja	Nein	233
Ja	Ja	37

a) Erzeugen Sie den Wahrscheinlichkeitsraum
b) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Person geimpft ist?
c) Wie hoch ist die Prävalenz der Grippe?
d) Wie groß ist die Wahrscheinlichkeit, dass geimpfte Personen an Grippe erkranken? Ist die Impfung effektiv?

Für die Lösung siehe Abschnitt 4.2.33

4.1.5.8 Aufgabe 4.1.5.8 Ebola

Um die Wirksamkeit eines Diagnosetests zur Feststellung von Ebola in einem zentralafrikanischen Land zu ermitteln, wurde der Test an vielen Personen durchgeführt. Das Ergebnis des Tests war positiv bei 147 Personen mit Ebola, aber aber auch bei 28 Personen ohne Ebola. Negativ war das Ergebnis des Tests bei 97465 Personen ohne Ebola, aber auch bei 65 Personen mit Ebola.

a) Erzeugen Sie den Wahrscheinlichkeitsraum des Tests.
b) Berechnen Sie die Prävalenz von Ebola in der Bevölkerung.
c) Wie hoch ist die Wahrscheinlichkeit, ein negatives Testergebnis zu erhalten?
d) Berechnen Sie die Sensitivität und Spezifität des Tests.
e) Kann der Test besser Erkrankte erkennen, oder Gesunde?

f) Wenn eine Person einen positiven Test erhält, wie hoch ist dann die Wahrscheinlichkeit, dass er tatsächlich krank ist?
g) Wenn eine Person einen negativen Test erhält, wie hoch ist dann die Wahrscheinlichkeit, dass er tatsächlich gesund ist?

Für die Lösung siehe Abschnitt 4.2.34

4.1.6 Diskrete Wahrscheinlichkeitsverteilungen

4.1.6.1 Aufgabe 4.1.6.1 Münzwurf

Wir haben 10 mal eine Münze geworfen, wobei das Ergebnis der Binomialverteilung B(10;0.5) folgt. Die Variable `X` misst, wie häufig dabei "`Kopf`" geworfen wurde.

a) Berechnen Sie die Wahrscheinlichkeitsverteilung von `X`
b) Plotten Sie die Wahrscheinlichkeitsfunktion von `X`
c) Plotten Sie die Verteilungsfunktion.
d) Berechnen Sie die Wahrscheinlichkeit, 7 mal `Kopf` zu werfen.
e) Berchnen Sie die Wahrscheinlichkeit, weniger als als 4 mal `Kopf` zu werfen.
f) Berchnen Sie die Wahrscheinlichkeit, mehr als als 5 mal `Kopf` zu werfen.
g) Berechnen Sie die Wahrscheinlichkeit, 2 bis 8 mal `Kopf` zu werfen.

Für die Lösung siehe Abschnitt 4.2.35

4.1.6.2 Aufgabe 4.1.6.2 Geburten pro Tag

Die Anzahl an täglichen Geburten `X` in unserer Stadt folgt einer Poissonverteilung mit durchschnittlich 6 Geburten am Tag.

a) Plotten Sie die Wahrscheinlichkeitsfunktion von `X`
b) Plotten Sie die Verteilungsfunktion von `X`
c) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag (nur) 1 Geburt stattfindet?
d) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag weniger als 6 Geburten stattfinden?
e) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag 4 oder mehr Geburten stattfinden?
f) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag 4 bis 8 Geburten stattfinden?
g) Wie groß ist die Wahrscheinlicheit, dass in einer Woche zwischen 30 und 40 Geburten stattfinden?

Für die Lösung siehe Abschnitt 4.2.36

4.1.6.3 Aufgabe 4.1.6.3 Gesetz der seltenen Ereignisse

Kommen wir nochmal auf das Münzwurfbeispiel aus Abschnitt 4.1.6.1 zurück.
Das Gesetz der seltenen Ereignisse besagt, dass das Binomial-Verteilungsmodell $B(n, p)$ zum Poisson-Wahrscheinlichkeitsverteilungsmodell $P(np)$ tendiert, wenn n gegen ∞ und p gegen 0 tendiert. Insbesondere ist das Poisson-Modell eine gute Annäherung an das Binomialmodell für $n \geq 30$ und $p \leq 0,1$.
Zur Überprüfung dieses Gesetz,

a) berechnen Sie die Wahrscheinlichkeitsverteilung des binomialen Modells $B(30, 0.1)$.
b) berechnen Sie die Wahrscheinlichkeitsverteilung des Poissonmodells $P(3)$ und vergleichen Sie es mit dem binomialen Modell $B(30, 0.1)$.
c) berechnen Sie die Wahrscheinlichkeitsverteilung des binomialen Modells $B(100, 0.3)$ und vergleichen Sie es es mit dem Modell $P(3)$. Sind diese Modelle ähnlicher als die vorherigen?
d) Plotten Sie die Wahrscheinlichkeitsfunktionen der vorherigen Modelle. Erhöhen Sie die Anzahl der Wiederholungen und verringern Sie die Erfolgswahrscheinlichkeit im Binomialmodell und beobachten Sie, wie sich die Wahrscheinlichkeiten des Binomialmodells und des Poissonmodells annähern.

Für die Lösung siehe Abschnitt 4.2.37

4.1.6.4 Aufgabe 4.1.6.4 Münzwürfe (II)

Wie groß ist die Wahrscheinlichkeit, beim Werfen von 100 Münzen zwischen 40 und 60 Mal `Kopf` zu erhalten (beide Werte eingeschlossen)?

Für die Lösung siehe Abschnitt 4.2.38

4.1.6.5 Aufgabe 4.1.6.5 Behandlungserfolg

Die Wahrscheinlichkeit, dass eine Behandlung Erfolg hat, liegt bei 85%. Wenn wir an 6 Personen die Behandlung durchführen,

a) wie groß ist die Wahrscheinlichkeit, dass die Hälfte der Patienten geheilt wird?
b) wie groß ist die Wahrscheinlichkeit, dass mindestens 4 Patienten geheilt werden?
c) plotten Sie die Wahrscheinlichkeitsfunktion für die Anzahl geheilter Patienten.

Für die Lösung siehe Abschnitt 4.2.39

4.1.6.6 Aufgabe 4.1.6.6 Impfreaktion

Die Wahrscheinlichkeit einer starken Impfreaktion beträgt $0,001$. Wenn 2.000 Personen geimpft werden, wie hoch ist die Wahrscheinlichkeit für starke Reaktionen?

Für die Lösung siehe Abschnitt 4.2.40

4.1.6.7 Aufgabe 4.1.6.7 Telefonanrufe

Die durchschnittliche Anzahl an Telefonanrufen in unserer Telefonzentrale beträgt 120 Anrufe pro Minute.

a) Wie hoch ist die Wahrscheinlichkeit, dass weniger als 4 Anrufe in 2 Sekunden eintreffen?
b) Wie hoch ist die Wahrscheinlichkeit, dass mindestens 3 Anrufe in 3 Sekunden eintreffen?

Für die Lösung siehe Abschnitt 4.2.41

4.1.7 Kontinuierliche Wahrscheinlichkeitsverteilungen

4.1.7.1 Aufgabe 4.1.7.1 Bushaltestelle

Nehmen wir an, dass ein Bus alle 15 Minuten an einer Haltestelle vorbeifährt und dass eine Person zu jedem Zeitpunkt mit der gleichen Wahrscheinlichkeit eintreffen kann. Dann folgt die Variable, die die Wartezeit auf den Bus misst, einer gleichmäßigen Wahrscheinlichkeitsverteilung $U(0, 15)$, da jede Wartezeit zwischen 0 und 15 Minuten die gleiche Wahrscheinlichkeit hat.

a) Plotten Sie die Dichtefunktion der Wartezeit.
b) Plotten Sie die Verteilungsfunktion der Wartezeit.
c) Berechnen Sie die Wahrscheinlichkeit, weniger als 5 Minuten auf den Bus zu warten.
d) Berechnen Sie die Wahrscheinlichkeit, länger als 12 Minuten auf den Bus zu warten.
e) Berechnen Sie die Wahrscheinlichkeit, zwischen 5 und 10 Minuten auf den Bus zu warten.
f) Bei welcher Zeit zwischen 0 und 15 Minuten muss die Hälft der Personen kürzer auf

den Bus warten als die angegebene Zeit?
g) Bei welcher Zeit zwischen 0 und 15 Minuten müssen 10% der Personen länger auf den Bus warten als die angegebene Zeit?

Für die Lösung siehe Abschnitt 4.2.42

4.1.7.2 Aufgabe 4.1.7.2 Standardnormalverteilung

Eine Variable folgt in ihren Ausprägungen der Standardnormalverteilung ($Z \sim N(0,1)$)

a) Plotten Sie die Dichtefunktion von Z.
b) Wie beeinflussen Mittelwert und Standardabweichung die Form der Gausschen Glockenkurve?
c) Plotten Sie die Verteilungsfunktion von Z.
d) Berechnen Sie die Wahrscheinlichkeit $P(Z < -1)$.
e) Berechnen Sie die Wahrscheinlichkeit $P(Z > 1)$
f) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus der Standardabweichung und dem Mittelwert plus der Standardabweichung liegt, d. h. $P(-1 \leq Z \leq 1)$.
g) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus zwei Standardabweichungen und dem Mittelwert plus zwei Standardabweichungen liegt, d. h. $P(-2 \leq Z \leq 2)$.
h) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus drei Standardabweichungen und dem Mittelwert plus drei Standardabweichungen liegt, d. h. $P(-3 \leq Z \leq 3)$.
i) Berechnen Sie die Quartile.
j) Bei welchem Z-Wert liegen 95% der Fläche unterhalb des Wertes?
k) Bei welchem Z-Wert liegen 2,5% der Fläche oberhalb des Wertes?

Für die Lösung siehe Abschnitt 4.2.43

4.1.7.3 Aufgabe 4.1.7.3 Chiquadratverteilungen

Wenn $X_1, \ldots, X_n$ unabhängige standardnormalverteilte Werte sind, dann folgt die Variable $X = X_1^2 + \cdots + X_n^2$ einer Chiquadratverteilung mit n Freiheitsgraden ($\chi^2(n)$). Nehmen wir nun an, X würde der Chiquadratverteilung mit 6 Freiheitsgraden folgen ($\chi^2(6)$).

a) Plotten Sie die Dichtefunktion dieser Verteilung
b) Wie groß ist die Wahrscheinlichkeit für $P(X < 6)$?
c) Berechnen Sie das fünfte Perzentil der Verteilung.

d) Bei welchem Wert liegen 10% der Fläche oberhalb des Wertes?

Für die Lösung siehe Abschnitt 4.2.44

4.1.7.4 Aufgabe 4.1.7.4 t-Verteilung

Wenn Y einer Chiquadratverteilung mit n Freiheitsgraden folgt ($\chi^2(n)$) und Z der Standardnormalverteilung ($N(0,1)$), dann folgt die Variable $X = \frac{Z}{\sqrt{Y/n}}$ einer Student-t-Verteilung mit 8 Freiheitsgraden ($T(8)$).

a) Plotten Sie die Dichtefunktion von X und vergleichen Sie diese mit der Dichtefunktion der Standardnormalverteilung.
b) Berechnen Sie das 8te Perzentil von X.
c) Bei welchem Wert von X liegen 5% aller Fälle oberhalb dieses Wertes?

Für die Lösung siehe Abschnitt 4.2.45

4.1.7.5 Aufgabe 4.1.7.5 Fishers F-Verteilung

Wenn Y_1 und Y_2 zwei unabhängige Variablen aus den Chiquadratverteilungen mit n und m Freiheitsgraden stammen, dann folgt die Variable $X = \frac{Y_1/n}{Y_2/m}$ einer Fisher-F-Verteilung mit n und m Freiheitsgraden ($F(n,m)$). Nehmen wir an, X folge einer Fisher-F-Verteilung mit 10 und 20 Freiheitsgeraden ($F(10,20)$).

a) Plotten Sie die Dichtefunktion von X.
b) Berechnen Sie Wahrscheinlichkeit $P(X > 1)$.
c) Berechnen Sie den Interquartilsabstand.

Für die Lösung siehe Abschnitt 4.2.46

4.1.7.6 Aufgabe 4.1.7.6 Blutzuckerspiegel

Es ist bekannt, dass der Glukosespiegel im Blut von Diabetikern einem Normalverteilungsmodell mit einem Mittelwert von 106 mg/100 ml und einer Standardabweichung von 8 mg/100 ml folgt.

a) Berechnen Sie die Wahrscheinlichkeit, dass ein zufällig ausgewählter Diabetiker einen

Glukosespiegel von weniger als 120 mg/100 ml hat.
b) Wie viel Prozent der Personen haben einen Glukosespiegel zwischen 90 und 120 mg/100 ml?
c) Berechnen und interpretieren Sie das erste Quartil des Glukosespiegels.

Für die Lösung siehe Abschnitt 4.2.47

4.1.7.7 Aufgabe 4.1.7.7 Cholesterinspiegel bei Männern

Es ist bekannt, dass der Cholesterinspiegel bei Männern im Alter von 30 Jahren einer Normalverteilung folgt mit Mittelwert 220 mg/dl und einer Standardabweichung von 30 mg/dl. In einer bestimmten Population gibt es 20.000 Männer im Alter von 30 Jahren.

a) Wie viele von ihnen haben einen Cholesterinspiegel zwischen 210 und 240 mg/dl?
b) Wenn ein Cholesterinspiegel von mehr als 250 mg/dl eine Thrombose auslösen kann, wie viele von ihnen sind thrombosegefährdet?
c) Welcher Cholesterinwert wird von mindestens 20% der Männer erreicht?

Für die Lösung siehe Abschnitt 4.2.48

4.1.8 Konfidenzintervalle (eine Stichprobe)

4.1.8.1 Aufgabe 4.1.8.1 Wirkstoffkonzentration

Die Wirkstoffkonzentration einer Zufallsstichprobe von 10 Arzneimittelbehältern aus einer Charge beträgt (in mg/mm^3)

17.6 19.2 21.3 15.1 17.6 18.9 16.2 18.3 19.0 16.4

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Konzentration`.
b) Berechnen Sie das Konfidenzintervall für die mittlere Konzentration bei einem Konfidenzniveau von 95% (Signifikanzlevel $\alpha = 0,05$).
c) Berechnen Sie das Konfidenzintervall für die mittlere Konzentration bei einem Konfidenzniveau von 99% (Signifikanzlevel $\alpha = 0,01$).
d) Wenn wir die Genauigkeit des Intervalls als den Kehrwert seiner Breite definieren, wie ändert sich die Genauigkeit eines Intervalls, wenn wir das Konfidenzniveau erhöhen? Warum?
e) Welche Stichprobengröße wird benötigt, um den mittleren Konzentrationswert mit einem Fehler von $\pm 0.5 mg/mm^3$ und einem Konfidenzniveau von 95% Sicherheit zu bestimmen?

f) Wenn die Konzentration des Wirkstoffs mindestens 16 mg/mm^3 betragen muss, um wirksam zu sein, ist dann unsere Medikamentencharge wirksam?

Für die Lösung siehe Abschnitt 4.2.49

4.1.8.2 Aufgabe 4.1.8.2 Milchfett

Ein Molkereibetrieb erhält Milch von zwei Bauernhöfen X und Y. Um die Qualität der Milch zu analysieren, wird das Milchfett für zwei Milchproben, eine von jedem Betrieb, gemessen. Die Ergebnisse sind in der nachstehenden Tabelle aufgeführt.

X		Y	
0.34	0.34	0.28	0.29
0.32	0.35	0.30	0.32
0.33	0.33	0.32	0.31
0.32	0.32	0.29	0.29
0.33	0.30	0.31	0.32
0.31	0.32	0.29	0.31
		0.33	0.32
		0.32	0.33

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Hof1` und `Hof2`.
b) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt.
c) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt, getrennt nach Höfen.
d) Plotten Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt, getrennt nach Höfen.
e) Lässt sich aus den Konfidenzintervallen ein signifikanter Untschied zwischen den Höfen feststellen?

Für die Lösung siehe Abschnitt 4.2.50

4.1.8.3 Aufgabe 4.1.8.3 Bibliotheksnutzung

In einer von einer Universität durchgeführten Umfrage über die Nutzung der Bibliothek wurde eine Stichprobe von 34 Studierenden gefragt, ob sie mindestens einmal pro Woche in die Bibliothek gehen.

nein ja nein nein nein ja nein ja ja ja ja nein ja nein ja nein nein nein ja ja ja
nein nein ja nein nein ja ja nein nein ja nein ja nein

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Antwort`.
b) Berechnen Sie das Konfidenzintervall für den Anteil an Studierenden, welche die Bibliothek wöchentlich nutzen mit einem Signifikanzlevel von $\alpha = 0,01$.
c) Wie präzise ist das Intervall?
d) Welcher Stichprobenumfang ist erforderlich, um eine Schätzung des Anteils der Studierenden zu erhalten, die die Bibliothek mindestens einmal pro Woche nutzen, mit einem Fehler von $\pm 1\%$ und einem Konfidenzniveau von 95%?

Für die Lösung siehe Abschnitt 4.2.51

4.1.8.4 Aufgabe 4.1.8.4 Atemwegsprobleme und Impfung

Das Gesundheitsministerium möchte ein Konfidenzintervall für den Anteil der Personen über 65 Jahre mit Atemwegsproblemen berechnen, die geimpft worden sind. In einer Zufallsstichprobe von 200 Personen über 65 mit Atemwegsproblemen wurden 154 geimpft.

a) Berechnen Sie das 95%-Konfidenzintervall für den Anteil an geimpften Probanden in der Grundgesamtheit.
b) Wenn das Gesundheitsministerium das Ziel verfolgt, dass mindestens 70% der Menschen über 65 mit Atemwegserkrankungen geimpft sind, können wir dann sagen, dass das Ministerium das Ziel erreicht hat?

Für die Lösung siehe Abschnitt 4.2.52

4.1.8.5 Aufgabe 4.1.8.5 Cholesterin

Der Cholesterinspiegel (in mg/dl) in einer Zufallsstichprobe mit 8 Probanden beträgt

196 212 188 206 203 210 201 198

a) Berechnen Sie die Konfidenzintervalle für den Mittelwert mit den Signifikanzniveaus 0.1, 0.05 und 0.01.

B) Kann man schließen, dass der Mittelwert des Cholesterinspiegels der Bevölkerung unter 210 mg/dl liegt?

Für die Lösung siehe Abschnitt 4.2.53

4.1.8.6 Aufgabe 4.1.8.6 Neurologisches Syndrom

Zur Behandlung eines neurologischen Syndroms gibt es zwei Therapien, A und B. In einer Studie wurde eine Stichprobe von 60 Personen gezogen. Bei 25 von ihnen wurde Therapie A angewandt, bei den anderen 35 Therapie B. Insgesmant 18 der mit A behandelten Personen wurden geheilt, während 21 der mit B behandelten Personen geheilt wurden.

a) Berechnen Sie für jede Therapie das 95% Konfidenzintervall für den Anteil an Personen, die geheilt wurden.
b) Welches Intervall ist präziser?

Für die Lösung siehe Abschnitt 4.2.54

4.1.8.7 Aufgabe 4.1.8.7 Neugeborene

Der Datensatz `neonates` von rk.Teaching[9] enthält Informationen über eine Stichprobe von 320 Neugeborenen, die im Laufe eines Jahres nach normaler Schwangerschaftsdauer geboren wurden.

a) Berechnen Sie das 99% Konfidenzintervall für den Mittelwert des Gewichts der Neugeborenen.
b) Berechnen Sie die Konfidenzintervalle für den APGAR-Score nach 1 Minute und für den APGAR-Score nach 5 Minuten und vergleiche sie beide Intervalle. Gibt es auf Grundlage der Konfidenzintervalle einen signifikanten Unterschied zwischen den Mittelwerten der beiden Scores?
c) Berechnen Sie die Konfidenzintervalle für den Prozentsatz der Neugeborenen mit einem Gewicht von $\leq 2,5$ kg für Raucher- und Nichtrauchermütter und vergleichen Sie die Intervalle.

Für die Lösung siehe Abschnitt 4.2.55

[9] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/neonates.RData

4.1.9 Konfidenzintervalle (zwei Stichproben)

4.1.9.1 Aufgabe 4.1.9.1 Medikamentenwerbung

Um festzustellen, ob eine Werbekampagne den Absatz eines Arzneimittels erhöht hat, wurde eine Stichprobe von 8 Apotheken aus einer Stadt gezogen. In jeder Apotheke wurden die monatlichen Verkäufe des Arzneimittels vor und nach der Kampagne in der folgenden Tabelle erfasst.

Vorher	147	163	121	205	132	190	176	147
Nachher	150	171	132	208	141	184	182	145

a) Erstellen Sie ein Datenframe mit den Variablen `vorher` und `nachher` und übertragen Sie die Daten.
b) Berechnen Sie den Mittelwert der monatlichen Umsätze vor und nach der Kampagne. Sind die Mittelwerte unterschiedlich? Hat die Kampagne den Absatz des Arzneimittels erhöht?
c) Berechnen Sie die Konfidenzintervalle für den durchschnittlichen Unterschied mit $\alpha = 0,05$ und $\alpha = 0,01$.
d) Können wir dieselbe Schlussfolgerung ziehen, wenn wir die Verkäufe nach der Kampagne der beiden letzten Apotheken ändern und 190 statt 182 und 165 statt 145 angeben? Was passiert mit den Konfidenzintervallen?

Für die Lösung siehe Abschnitt 4.2.56

4.1.9.2 Aufgabe 4.1.9.2 Milchfett

Ein Molkereibetrieb erhält Milch von zwei Bauernhöfen X und Y. Um die Qualität der Milch zu analysieren, wird das Milchfett für zwei Milchproben, eine von jedem Betrieb, gemessen. Die Ergebnisse sind in der nachstehenden Tabelle aufgeführt.

X		Y	
0.34	0.34	0.28	0.29
0.32	0.35	0.30	0.32
0.33	0.33	0.32	0.31
0.32	0.32	0.29	0.29
0.33	0.30	0.31	0.32
0.31	0.32	0.29	0.31
		0.33	0.32
		0.32	0.33

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Hof1` und `Hof2`.
b) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettunterschied in der Milch von `Hof1` und `Hof2`.

c) Kann man daraus schließen, dass der Unterschied zwischen den Milchfettmittelwerten der Betriebe signifikant ist? Welcher Betrieb hat Milch mit mehr Fett? Wie viel mehr Fett hat die Milch von `Hof1` als die Milch von `Hof2`?

Für die Lösung siehe Abschnitt 4.2.57

4.1.9.3 Aufgabe 4.1.9.3 Bibliotheksnutzung nach Geschlecht

In einer von einer Universität durchgeführten Umfrage über die Nutzung der Bibliothek wurde eine Stichprobe von 34 Studierenden gefragt, ob sie mindestens einmal pro Woche in die Bibliothek gehen.

Antwort	nein	ja	nein	nein	nein	ja	nein	ja	ja	ja	ja	nein
Geschlecht	m	w	w	m	m	m	w	w	w	w	m	m

Antwort	nein	ja	nein	nein	nein	ja	ja	ja	nein	nein	ja	nein
Geschlecht	m	w	m	m	w	m	w	w	w	m	w	m

Antwort	ja	ja	nein	nein	ja	nein	ja	nein	ja	nein
Geschlecht	w	w	m	m	w	w	w	m	w	m

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Antwort` und `Geschlecht`.
b) Berechnen Sie das Konfidenzintervall für den Unterschied zwischen den Anteilen der Frauen und Männern, die die Bibliothek mindestens einmal pro Woche nutzen.

Für die Lösung siehe Abschnitt 4.2.58

4.1.9.4 Aufgabe 4.1.9.4 Prüfungen vormittags und nachmittags

In einem Kurs gibt es zwei Gruppen von Studierenden, eine am Vormittag und die andere am Nachmittag. In der Vormittagsgruppe haben 55 von 80 Studierenden bestanden, während in der Nachmittagsgruppe 32 von 90 Studierenden bestanden haben.

a) Gibt es signifikante Unterschiede zwischen den Prozentsätzen der Studierenden, die am Vormittag und am Nachmittag bestanden haben? Kann man daraus schließen, dass der Stundenplan die Ursache für diese Unterschiede ist?

Für die Lösung siehe Abschnitt 4.2.59

4.1.9.5 Aufgabe 4.1.9.5 Cholesterin und Sport

In einer Studie zur Ermittlung des Zusammenhangs zwischen körperlicher Betätigung und dem Cholesterinspiegel im Blut wurde eine wurde eine Stichprobe von 11 Personen gezogen. Der Cholesterinspiegel der Teilnehmer (in mg/dl) vor und nach der Teilnahme an einem Programm mit körperlichen Übungen ist unten dargestellt.

vorher	182	232	191	200	148	249	276	213	241	280	262
nachher	198	210	194	220	138	220	219	161	210	213	226

a) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Unterschied der Cholesterinwerte vor und nach den körperlichen Übungen
b) Berechnen Sie das 99%-Konfidenzintervall für den durchschnittlichen Unterschied der Cholesterinwerte vor und nach den körperlichen Übungen
c) Auf Grundlage der zuvor berechneten Intervalle, welchen Schluss bezüglich des Einflusses von körperlichen Aktivitäten auf den Cholesterinspiegel können Sie ziehen?

Für die Lösung siehe Abschnitt 4.2.60

4.1.9.6 Aufgabe 4.1.9.6 Patientenzufriedenheit

Insgesamt 500 Patienten aus zwei Krankenhäusern wurden zu ihrer Zufriedenheitbefragt. In Krankenhaus 1 wurden 200 Patienten befragt, von denen 140 zufrieden waren. In Krankenhaus 2 wurden 300 Patienten befragt, von denen 180 zufrieden waren.

a) Berechnen Sie das 95%-Konfidenzintervall für den Anteilsunterschied an zufriedenen Patienten in beiden Häusern.
b) Wenn $\alpha = 0,01$ ist, können dann Rückschlüsse gezogen werden, ob der Unterschied

der Anteile zufriedener Patienten signifikant ist?

Für die Lösung siehe Abschnitt 4.2.61

4.1.9.7 Aufgabe 4.1.9.7 Neugeborene

Der Datensatz `neonates` von rk.Teaching[10] enthält Informationen über eine Stichprobe von 320 Neugeborenen, die im Laufe eines Jahres nach normaler Schwangerschaftsdauer geboren wurden.

a) Berechnen Sie das 95%-Konfidenzintervall für den durchscnnittlichen Unterschied des Geburtsgewichts zwischen Kindern von Raucherinnen und Nichtraucherinnen. Wie groß ist der durchschnittliche Gewichtsunterschied?
b) Berücksichtigen Sie nur die Daten der Mütter, die während der Schwangerschaft nicht geraucht haben. Berechnen Sie das 95%-Konfidenzintervall für den durchscnnittlichen Unterschied des Geburtsgewichts zwischen Kindern von Müttern, die vor der Schwangerschaft geraucht haben, und den Nichtraucherinnen.
c) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Unterschied von APGAR-1-Werten und APGAR-5-Werten. Wie entwickeln sich Neugeborene in den ersten 5 Minuten nach der Geburt?
d) Wenn Neugeborene mit einem APGAR-1-Wert ≤ 3 in einem kritischen Zustand sind, berechnen Sie das 90%-Konfidenzintervall für den Unterschied der Anteile von Neugeborenen in kritischem Zustand zwischen Müttern, die während der Schwangerschaft geraucht haben und den Nichtraucherinnen.
e) Hat das Alter der Mutter einen signifikanten Einfluss auf den Anteil an Neugeborenen in kritischem Zustand?

Für die Lösung siehe Abschnitt 4.2.62

4.1.10 Signifikanztests

4.1.10.1 Aufgabe 4.1.10.1 Wirkstoffkonzentration

Die Wirkstoffkonzentration einer Zufallsstichprobe von 10 Arzneimittelbehältern aus einer Charge beträgt (in mg/mm^3)

17.6 19.2 21.3 15.1 17.6 18.9 16.2 18.3 19.0 16.4

[10] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/neonates.RData

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Konzentration`.
b) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 18$ versus $H_1 : \mu \neq 18$ mit einem Signifikanzniveau von $\alpha = 0,05$.
c) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 19,5$ versus $H_1 : \mu \neq 19,5$ mit den Signifikanzniveaus von $\alpha = 0,05$ und $0,01$. Wie beeinflusst das Signifikanzniveau das Testergebnis?
d) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 17$ versus $H_1 : \mu \neq 17$ mit einem Signifikanzniveau von $\alpha = 0,05$. Testen Sie ebenfalls die Hypothesen $H_0 : \mu = 17$ versus $H_1 : \mu > 17$ mit $\alpha = 0,05$. Was ist der Unterschied zwischen den p-Werten des zweiseitigen und des einseitigen Tests?
e) Wenn der Hersteller angibt, die Konzentration des Wirkstoffs erhöht zu haben (im Vergleich zu früheren Chargen, bei denen der Mittelwert der Konzentration 17 mg/mm^3 war), können wir ihm glauben?
f) Welche Fallzahl würde benötigt, um einen Konzentrationsanstieg von 0,5 mg/mm^3 zu erkennen (mit $\alpha = 0,05$ und einer Power von $1 - \beta = 0,8$)?

Für die Lösung siehe Abschnitt 4.2.63

4.1.10.2 Aufgabe 4.1.10.2 Bibliotheksnutzung

In einer von einer Universität durchgeführten Umfrage über die Nutzung der Bibliothek wurde eine Stichprobe von 34 Studierenden gefragt, ob sie mindestens einmal pro Woche in die Bibliothek gehen.

nein ja nein nein nein ja nein ja ja ja ja nein ja nein ja nein nein nein ja ja ja
nein nein ja nein nein ja ja nein nein ja nein ja nein

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `bib`.
b) Testen Sie die Hypothese, dass der Anteil an Studierenden, die wöchentlich die Bibliothek nutzen, größer als 40% ist.

Für die Lösung siehe Abschnitt 4.2.64

4.1.10.3 Aufgabe 4.1.10.3 Laufen lernen

Eine Studie möchte untersuchen, ob Babies aus den unterschiedlichen Populationen A und B zu unterschiedlichen Zeiten anfangen zu laufen. In folgender Tabelle ist das Alter der Babies in Monaten aufgeführt, zu welchem sie mit dem Laufen anfingen.

A	9.5	10.5	9.0	9.8	10.0	13.0	10.0	13.5	10.0	9.8		
B	12.5	9.5	13.5	13.8	12.0	13.8	12.5	9.5	12.0	13.5	12.0	12.0

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Alter` und `Population`.
b) Testen Sie die Hypothese, dass das durchschnittliche Alter in den Populationen unterschiedlich ist, mit $\alpha = 0,05$.

Für die Lösung siehe Abschnitt 4.2.65

4.1.10.4 Aufgabe 4.1.10.4 Bronchialretention

Forschende haben bei Rauchern einen größeren Atemwegswiderstand festgestellt als bei Nichtrauchern. Zur Überprüfung wurde bei 12 Probanden der Prozentsatz der tracheobronchialen Retention gemessen als sie Raucher waren und ein Jahr nach dem Rauchstopp.

Rauchen	Nichrauchen
60.6	47.5
12.0	13.3
56.0	33.0
75.2	55.2
12.5	21.9
29.7	27.9
57.2	54.3
62.7	13.9
28.7	8.90
66.0	46.1
25.2	29.8
40.1	36.2

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `vorher` und `nachher`.
b) Testen Sie, ob sich die Bronchialretention nach dem Rauchstopp verringert.

Für die Lösung siehe Abschnitt 4.2.66

4.1.10.5 Aufgabe 4.1.10.5 Prüfungen vormittags und nachmittags

In einem Kurs gibt es zwei Gruppen von Studierenden, eine am Vormittag und die andere am Nachmittag. Unter der Vormittagsgruppe haben 55 von 80 Studierenden bestanden, während in der Nachmittagsgruppe 32 von 90 Studierenden bestanden haben.

a) Gibt es signifikante Unterschiede zwischen den Prozentsätzen der Studiereden, die am Vormittag und am Nachmittag bestanden haben? Kann man daraus schließen, dass der Stundenplan die Ursache für diese Unterschiede ist?

Für die Lösung siehe Abschnitt 4.2.67

4.1.10.6 Aufgabe 4.1.10.6 Pulsmessung

Der Datensatz `pulse` von rk.Teaching[11] enthält Informationen über den Puls einer Stichprobe von Personen nach verschiedenen Übungen:

- Ruhepuls in Schlägen pro Minute (`pulse1`),
- Puls nach Bewegung in Schlägen pro Minute (`pulse2`),
- Art der Bewegung (`type`),
- Geschlecht (`sex`) und Gewicht (`weight`)

a) Testen Sie, ob der Ruhepuls weniger als 75 Schläge pro Minute beträgt.
b) Welcher Stichprobenumfang ist erforderlich, um einen Anstieg des Ruhepulses um 2 Schläge pro Minute mit einem Signifikanzniveau von 0,05 und einer Power von 0,9 festzustellen?
c) Testen Sie, ob der Puls nach dem Laufen größer als 85 Schläge pro Minute ist.
d) Eine Person hat eine leichte Tachykardie, wenn der Ruhepuls größer als 90 Schläge pro Minute ist. Prüfen Sie, ob der Prozentsatz der Personen mit leichter Tachykardie größer als 5% ist.
e) Kann man mit 95%iger Sicherheit schließen, dass Bewegung den Puls erhöht? Und bei einem Signifikanzniveau von $\alpha = 0,01$?
f) Gibt es einen Unterschied zwischen den durchschnittlichen Pulsschlägen nach dem Gehen und dem Laufen?
g) Gibt es einen Unterschied zwischen den Mittelwerten des Ruhepulses von Männern und Frauen? Und nach dem Laufen?

Für die Lösung siehe Abschnitt 4.2.68

[11] https://github.com/rkward-community/rk.Teaching, auch verfügbar unter https://www.produnis.de/R/data/pulse.RData

4.1.11 Varianzanalysen (ANOVA)

4.1.11.1 Aufgabe 4.1.11.1 Aknetherapie

In einer Studie wird versucht, die Wirksamkeit von drei Therapieprogrammen A, B und C zur Behandlung von von Akne zu bestimmen. Die Teilnehmer der Studie wurden nach dem Zufallsprinzip in drei Gruppen eingeteilt, und in jeder Gruppe wurde eine der Behandlungen durchgeführt. Nach 16 Wochen Behandlung wurde der prozentuale Rückgang der Akneläsionen gemessen.

Reduction in percentage of lesions

Treatment A		Treatment B		Treatment C	
48.6	50.8	68.0	71.9	67.5	61.4
49.4	47.1	67.0	71.5	62.5	67.4
50.1	52.5	70.1	69.9	64.2	65.4
49.8	49.0	64.5	68.9	62.5	63.2
50.6	46.7	68.0	67.8	63.9	61.2
		68.3	68.9	64.8	60.5
				62.3	

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Therapie` und `Aknereduktion`.
b) Plotten Sie die Aknereduktion für jede Therapie. Sind Unterschiede erkennbar?
c) Führen Sie eine ANOVA durch. Gibt es signifikante Unterschiede zwischen den Therapien?
d) Berechnen Sie die Konfidenzintervalle für die paarweisen Unterschiede zwischen den drei Behandlungen. Bei welchen Behandlungen gibt es signifikante Unterschiede?
e) Plotten Sie diese Konfidenzintervalle.

Für die Lösung siehe Abschnitt 4.2.69

4.1.11.2 Aufgabe 4.1.11.2 Schulranking

Um zu prüfen, ob es zwischen den Schulen einer Stadt Unterschiede in den sportlichen Leistungen gibt, wurde eine Zufallsstichprobe von 8 Schülern jeder Schule gezogen. Die erreichten Punkte bei einem Sportwettkampf (von 1 bis 10) der jeweiligen Schüler sind in der folgenden Tabelle dargestellt.

A	B	C	D	E
5.5	6.1	4.9	3.2	6.7
5.2	7.2	5.5	3.3	5.8
5.9	5.5	6.1	5.5	5.4

7.1	6.7	6.1	5.7	5.5
6.2	7.6	6.2	6.0	4.9
5.9	5.9	6.4	6.1	6.2
5.3	8.1	6.9	4.7	6.1
6.2	8.3	4.5	5.1	7.0

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Schule` und `Punkte`.
b) Plotten Sie die durchschnittlich erreichten Punkte pro Schule. Sind Unterschiede erkennbar?
c) Führen Sie eine ANOVA durch. Gibt es signifikante Unterschiede zwischen den Schulen?
d) In welcher Schule sind die sportlichen Leistungen am besten?

Für die Lösung siehe Abschnitt 4.2.70

4.1.11.3 Aufgabe 4.1.11.3 Puls und Herzkrankheit

Die nachstehende Tabelle zeigt den Puls (in Schlägen pro Minute) von vier Patientengruppen: Kontrollen (`A`), Patienten mit Angina pectoris (`B`), Patienten mit Herzrhythmusstörungen (`C`) und Patienten, die sich von einem Herzinfarkt erholt haben (`D`).

A	B	C	D
83	81	75	61
61	65	68	75
80	77	80	78
63	87	80	80
67	95	74	68
89	89	78	65
71	103	69	68
73	89	72	69
70	78	76	70
66	83	75	79
57	91	69	61

a) Gibt es laut den Daten signifikante Unterschiede zwischen den vier Gruppen?

Für die Lösung siehe Abschnitt 4.2.71

4.1.11.4 Aufgabe 4.1.11.4 Kohlenmonoxid

Die folgende Tabelle zeigt die Atemfrequenz (Atemzüge pro Minute) bei einer Stichprobe von Laborratten, die drei Konzentrationen von Kohlenmonoxid ausgesetzt waren.

Low	Medium	High
36	43	45
33	38	39
35	41	33
39	34	39
41	28	33
41	44	26
44	30	39
45	31	29

a) Gibt es laut den Daten signifikante Unterschiede zwischen den drei Gruppen?

Für die Lösung siehe Abschnitt 4.2.72

4.1.12 Chiquadratests für Anteilswerte

4.1.12.1 Aufgabe 4.1.12.1 Magengeschwür

Die folgende Tabelle enthält die Blutgruppe einer Stichprobe von 1655 Patienten mit Magengeschwüren und 10000 Patienten ohne Magengeschwüre Patienten.

	0	A	B	AB
Geschwür	911	579	124	41
kein Geschwür	4578	4219	890	313

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Geschwuer` und `Blutgruppe`.
b) Führen Sie einen Chiquadrattest auf die Hypothese durch, dass die Geschwüre von der Blutgruppe abhängig sind.
c) Gibt es in Anbetracht der Ergebnisse des Vergleichs einen Zusammenhang zwischen dem Magengeschwür und der Blutgruppe? Können wir behaupten, dass der Anteil der Ulkuspatienten je nach Blutgruppe unterschiedlich ist?

Für die Lösung siehe Abschnitt 4.2.73

4.1.12.2 Aufgabe 4.1.12.2 Blutgruppen

Mitchell et al. (1976)[14] untersuchten die Verteilung der Blutgruppen in einer Stichprobe von 478 Personen aus verschiedenen Regionen im Südwesten Schottlands. Sie erhielten die folgenden Ergebnisse:

	Eskdale	Annandale	Nithsdale	
A	33	54	98	185
B	6	14	35	55
O	56	52	115	223
AB	5	5	5	15
	100	125	253	478

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Region` und `Blutgruppe`.
b) Führen Sie einen Chiquadrattest auf die Hypothese durch, dass die Blutgruppe von der Region abhängig sind.
c) Gibt es in Anbetracht der Ergebnisse einen Zusammenhang zwischen der Blutgruppe und der Region? Können wir behaupten, dass die Region keinen Einfluss auf die Blutgruppe hat?

Für die Lösung siehe Abschnitt 4.2.74

4.1.12.3 Aufgabe 4.1.12.3 Rauchen und Geschlecht

In einer Studie wurd versucht festzustellen, ob das Rauchen mit dem Geschlecht zusammenhängt. Es wurden 9 Männer und 17 Frauen befragt. Unter den männlichen Probanden gab es 2 Raucher, während in der weiblichen Stichprobe 6 Raucherinnen waren.

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Rauchen` und `Geschlecht`.
b) Führen Sie einen Chi-Quadrat-Test durch, um festzustellen, ob das Rauchen mit dem Geschlecht zusammenhängt.
c) Ist die Verteilung der Raucher bei beiden Geschlechtern gleich?

Für die Lösung siehe Abschnitt 4.2.75

4.1.12.4 Aufgabe 4.1.12.4 Migräne

Um die Wirksamkeit von zwei Medikamenten gegen Migräne zu vergleichen, wurden 20 Personen, die häufig unter Migräne litten, ausgewählt und die beiden Medikamente zu verschiedenen Zeitpunkten ausprobiert. Die folgende Tabelle zeigt die Anzahl der Personen, die eine gewisse Linderung erfuhren.

Drug 1	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Drug 2	No	No	Yes	No	Yes	Yes	No	No	No	No

Drug 1	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Drug 2	Yes	No	Yes	No	No	Yes	No	Yes	No	No

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `drug1` und `drug2`.
b) Führen Sie einen McNemar-Test durch, um festzustellen, ob die Linderung mit dem Medikament zusammenhängt.
c) Können wir nach dem Ergebnis des Tests behaupten, dass die Linderung der Migräne vom Medikament abhängt? Wenn ja, welches Medikament bewirkt eine signifikant höhere Linderung?

Für die Lösung siehe Abschnitt 4.2.76

4.1.12.5 Aufgabe 4.1.12.5 Komatös

Eine Studie versucht zu bestimmen, ob Patienten, die bei der Ankunft im Krankenhaus komatös sind, eine schlechtere Prognose (Überleben oder Sterben) haben.

	nicht komatös	komatös	
überleben	484	37	521
verstorben	118	89	207
	602	126	728

a) Ist ein komatöser Zustand bei der Ankunft im Krankenhaus ein Risikofaktor zu versterben?

Für die Lösung siehe Abschnitt 4.2.77

4.1.12.6 Aufgabe 4.1.12.6 Heilung

Die Heilung einer Krankheit, die durch zwei Behandlungen A und B hervorgerufen wird, wird in drei Kategorien eingeteilt: sehr gut, gut und schlecht.
Die Behandlung A wird bei 32 Patienten angewandt und B bei 28. Bei Medikament A konnten 10 von insgesamt 22 `sehr guten` Heilungen, 14 von insgesamt 24 `guten` Heilungen und 8 von insgesamt 14 `schlechten` Heilungen beobachtet werden. Ist die Wirksamkeit der beiden Behandlungen die gleiche?

Für die Lösung siehe Abschnitt 4.2.78

4.1.12.7 Aufgabe 4.1.12.7 Facherfolg

Um festzustellen, ob Frauen in einem Fach erfolgreicher sind als Männer, wurde eine Stichprobe von 10 Frauen und 10 Männern gezogen. Beide Gruppen wurden von einem Lehrer geprüft, der immer 40% der Prüflinge durchfallen lässt. Wenn man weiß, dass nur 2 Männer bestanden haben, können wir dann behaupten, dass Frauen in diesem Fach erfolgreicher sind als Männer?

Für die Lösung siehe Abschnitt 4.2.79

4.1.12.8 Aufgabe 4.1.12.8 Statistikdozent

150 Studierende wurden befragt, ob ihnen die Lehrmethoden von zwei Biostatistik-Dozenten (Hans und Erna) gefallen. Die Ergebnisse sind in der nachstehenden Tabelle aufgeführt:

	like Hans	dislike Hans
like Erna	37	48
dislike Erna	44	21

Können wir bestätigen, dass es unterschiedliche Meinungen über Hans und Erna gibt?

Für die Lösung siehe Abschnitt 4.2.80

4.2 Lösungen

Die hier vorgestellten Lösungen stellen immer nur *eine mögliche* Vorgehensweisen dar und sind sicherlich nicht der Weisheit letzter Schluss. In `R` führen viele Wege nach Rom, und wenn Sie mit anderem Code zu den richtigen Ergebnissen kommen, dann ist das völlig in Ordnung.

4.2.1 Lösung zur Aufgabe 4.1.1.1

a) Erstellen Sie ein Datenframe mit der Variable `Kinder` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Kinder = c(1, 2, 4, 2, 2, 2, 3, 2, 1, 1, 0, 2, 2, 0,
                            2, 2, 1, 2, 2, 3, 1, 2, 2, 1, 2))
```

b) Erzeugen Sie eine einfache Häufigkeitstabelle

```
# erzeuge Datenframe
table(df$Kinder)
```

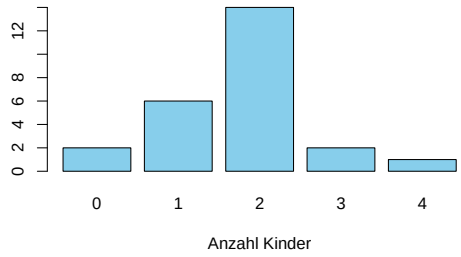
```
 0  1  2  3  4
 2  6 14  2  1
```

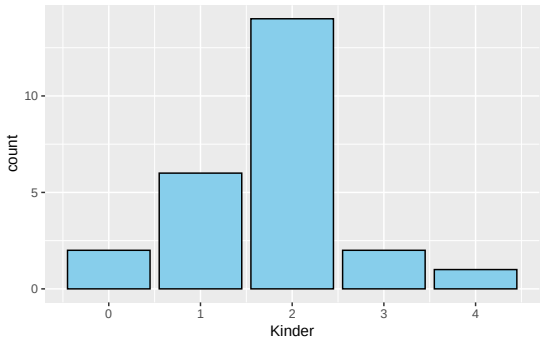
```
# oder
xtabs(~Kinder, data=df)
```

```
Kinder
 0  1  2  3  4
 2  6 14  2  1
```

c) Erzeugen Sie ein Balkendiagramm der Häufigkeiten

```
# Balkendiagramm mit R-Base
barplot(table(df$Kinder), col="skyblue", xlab="Anzahl Kinder")
# mit ggplot
ggplot(df, aes(x=Kinder)) +
          geom_bar(fill="skyblue", color="black")
```





d) Erzeugen Sie eine vollständige Häufigkeitstabelle, inklusive absoluter, relativer und jeweils kumulativer Häufigkeiten

```
## zu Fuß
# kumulierte absoulte Häufigkeiten
cumsum(table(df$Kinder))
```

```
 0  1  2  3  4
 2  8 22 24 25
```

```
# relative Häufigkeiten
(table(df$Kinder)/length(df$Kinder))*100
```

```
 0  1  2  3  4
 8 24 56  8  4
```

```
# kumulierterelative Häufigkeiten
cumsum(table(df$Kinder)/length(df$Kinder))*100
```

```
  0   1   2   3   4
  8  32  88  96 100
```

```
## einfacher
# erzeuge vollständige Häufigkeitstabelle
jgsbook::freqTable(df$Kinder)
```

Wert	Haeufig	Hkum	Relativ	Rkum
0	2	2	8	8

1	6	8	24	32
2	14	22	56	88
3	2	24	8	96
4	1	25	4	100

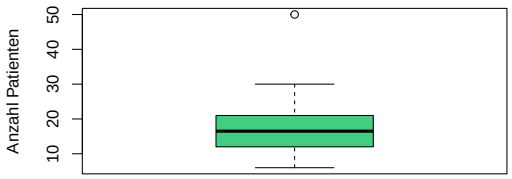
4.2.2 Lösung zur Aufgabe 4.1.1.2

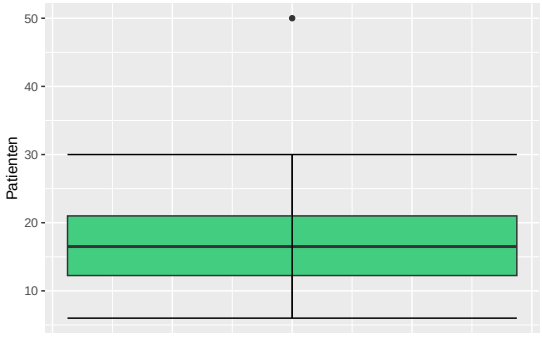
a) Erstellen Sie ein Datenframe mit der Variable `Patienten` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Patienten = c(15, 23, 12, 10, 28, 50, 12, 17, 20,
                               21, 18, 13, 11, 12, 26, 30,  6, 16,
                               19, 22, 14, 17, 21, 28,  9, 16, 13,
                               11, 16, 20))
```

b) Erzeugen Sie ein Boxplot und entfernen Sie etwaige Ausreißer.

```
# Boxplot mit Rbase
boxplot(df$Patienten, col="seagreen3", ylab="Anzahl Patienten")
# Boxplot mit ggplot
ggplot(df, aes(y=Patienten)) +
    geom_boxplot(fill="seagreen3") +
    # whiskers
    stat_boxplot(geom="errorbar") +
    theme(axis.ticks.x=element_blank(),
          axis.text.x=element_blank())
```





Es ist ein Ausreißer enthalten.

```
# entferne Ausreißer für weiteres Vorgehen
df <- subset(df, Patienten < 50)
```

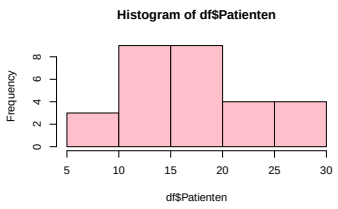
c) Erzeugen Sie eine Häufigkeitstabelle, welche die Daten in 5 Klassen gruppiert.

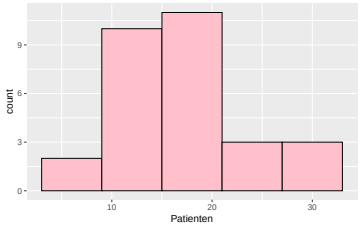
```
# klassiere in 5 Gruppen
gruppen <- cut(df$Patienten, breaks = 5, ordered_result = TRUE)
# Häufigkeitstabelle
table(gruppen)
```

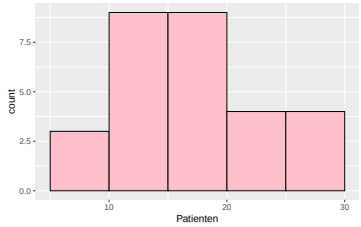
```
gruppen
(5.98,10.8] (10.8,15.6] (15.6,20.4] (20.4,25.2]   (25.2,30]
          3           9           9           4           4
```

d) Erzeugen Sie ein Histogram der klassierten absoluten Häufigkeiten.

```
# Histogram mit Rbase
hist(df$Patienten, col="pink")
# mit ggplot werden andere Breaks erzeugt
ggplot(df, aes(x=Patienten)) +
   geom_histogram(fill="pink", color="black",
                  bins=5)
# also die Klassengrenzen manuell festlegen
ggplot(df, aes(x=Patienten)) +
   geom_histogram(fill="pink", color="black",
                  breaks=c(5, 10, 15, 20, 25, 30))
```







e) Erzeugen Sie ebenso Histogramme der relativen und jeweils kumulativen Häufigkeiten, inklusive Polygonzügen.

Mit R base können wir wie folgt vorgehen.

```
  # 1. kumulierte absolute Häufigkeiten
  #-----------------------------------
  # speichere Histogramm in Objekt h
  h <- hist(df$Patienten, plot=FALSE)

  # ersetze die Zellen durch kumulierte Häufigkeiten
  h$counts <- cumsum(h$counts)

  # plotte das kumulative Histogram
  plot(h, col="hotpink", main = "kumulierte Häufigkeiten")
  # füge Polygonzug hinzu
  lines(c(0, h$mids),c(0,h$counts), col="blue") # type="s"
  ##---------------------------------------#

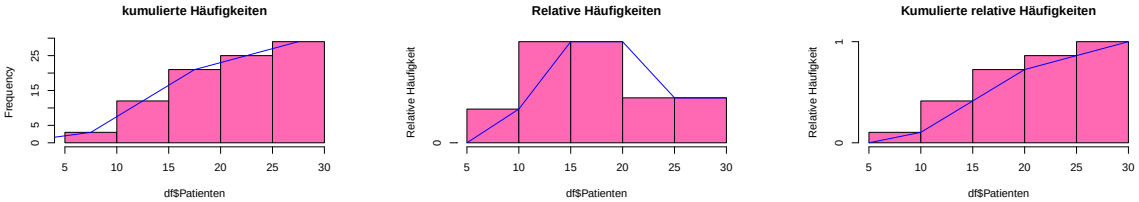

  # 2. Histogram der relativen Häufigkeiten #
  ##---------------------------------------#
  # speichere Histogramm in Objekt h
  h <- hist(df$Patienten, plot=FALSE)

%-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

%-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  # relative Häufigkeiten
  h$counts <- h$counts/sum(h$counts)
  ### plot
  plot(h, col="hotpink", main = "Relative Häufigkeiten",
       ylab = "Relative Häufigkeit" )
  # Polygon hinzufügen
  lines(h$breaks, c(0, h$counts), col = "blue") # add type="s" if you like
  ##-----------------------------------------------------#


  # 3. Histogram der kumulierten relativen Häufigkeiten #
  ##-----------------------------------------------------#
  # speichere Histogramm in Objekt h
  h <- hist(df$Patienten, plot=FALSE)

  # kumulative relative Häufigkeiten
  h$counts <- cumsum(h$counts)/sum(h$counts)
  ### plot
  plot(h, col="hotpink", main = "Kumulierte relative Häufigkeiten",
       ylab = "Relative Häufigkeit" )
  # Polygon hinzufügen
  lines(h$breaks, c(0, h$counts), col = "blue") # add type="s" if you like
```



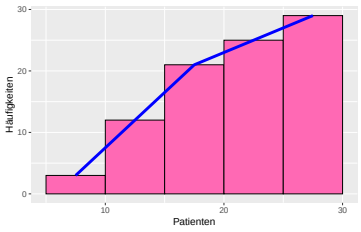
Im Tidyverse können wir so vorgehen.

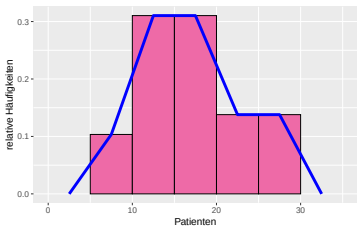
```
### Mittels ggplot()
# Klassengrenzen festlegen
breaks = c(5, 10, 15, 20, 25, 30)

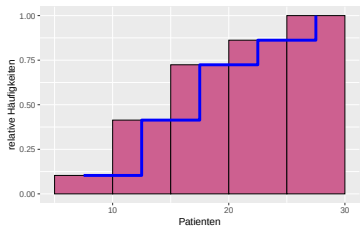
# kumulierte Häufigkeiten
ggplot(df, aes(x=Patienten)) +
  ylab("Häufigkeiten")+
  geom_histogram(aes(y=cumsum(after_stat(count))),
                 fill="hotpink", color="black",
                 breaks=breaks) +
  stat_bin(aes(y=cumsum(after_stat(count))),
           breaks=breaks,
           geom="line", color="blue", linewidth=1.5) # oder geom="step"
#-----------------

# relative Häufigkeiten
ggplot(df, aes(x=Patienten))+
  ylab("relative Häufigkeiten")+
  geom_histogram(aes(y=after_stat(count)/sum(after_stat(count))),
                 breaks=breaks, fill="hotpink2", color="black") +
  geom_freqpoly(aes(y=after_stat(count)/sum(after_stat(count))),
                 breaks=breaks, color="blue", linewidth=1.5)
#-----------------

# kumulierte relative Häufigkeiten
ggplot(df, aes(x=Patienten)) +
  ylab("relative Häufigkeiten")+
  geom_histogram(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))),
                 breaks=breaks, fill="hotpink3", color="black") +
  stat_bin(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))),
                 breaks=breaks,
                 geom="step", color="blue", linewidth=1.5) # oder geom="line"
```







4.2.3 Lösung zur Aufgabe 4.1.1.3

a) Erstellen Sie ein Datenframe mit der Variable `Blutgruppe` und übertragen Sie die Daten.

```
# Übertrage Daten
df <- data.frame(Blutgruppe = factor(c("A", "B", "B", "A", "AB", "0", "0", "A",
                                       "B", "B", "A", "A", "A", "A", "AB", "A",
                                       "A", "A", "B", "0", "B", "B", "B", "A",
                                       "A", "A", "0", "A", "AB", "0")))
```

b) Erzeugen Sie eine Häufigkeitstabelle

```
table(df$Blutgruppe)
```

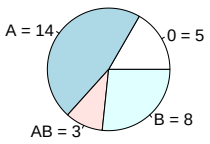
```
 0  A AB  B
 5 14  3  8
```

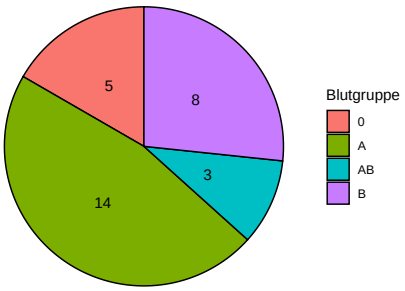
c) Erzeugen Sie ein Kreisdiagramm

```
# mit R base
pie(table(df$Blutgruppe),
    labels = paste(levels(df$Blutgruppe),"=",
                   as.numeric(table(df$Blutgruppe))
                  )
    )
#------

# mit ggplot benötigen wir ein Hilfsdatenframe
df2 <- as.data.frame(table(df$Blutgruppe))
colnames(df2) <- c("Blutgruppe", "Wert")

ggplot(df2, aes(x="", y=Wert, fill=Blutgruppe)) +
  geom_col(color="black") +
  # Werte schreiben
  geom_text(aes(label = Wert),
            position = position_stack(vjust = 0.5)) +
  # verbiege zu Kreisdiagramm
  coord_polar(theta="y") +
  # entferne Achsen und Ticks
  theme_void()
```





4.2.4 Lösung zur Aufgabe 4.1.1.4

a) Erstellen Sie ein Datenframe mit den Variablen `Alter` und `Familienstand` und übertragen Sie die Daten.

```
df <- data.frame(Alter = c(31, 45, 35, 65, 21, 38, 62, 22, 31,
                           72, 39, 62, 59, 25, 44, 54,
                           80, 68, 65, 40, 78, 69, 75,
                           31, 65, 59, 58, 50),
                 Familienstand = c( rep("Single", 9),
                                    rep("Verheiratet", 7),
                                    rep("Verwitwet", 7),
                                    rep("Geschieden", 5)
                                  )
                 )
```

b) Erzeugen Sie für jeden `Familienstand` eine Häufigkeitstabelle des `Alters`.

```
# Singles
df2 <- subset(df, Familienstand=="Single")
table(df2$Alter)
```

```
21 22 31 35 38 45 62 65
 1  1  2  1  1  1  1  1
```

```
# Verheiratet
df2 <- subset(df, Familienstand=="Verheiratet")
table(df2$Alter)
```

```
25 39 44 54 59 62 72
 1  1  1  1  1  1  1
```

```
# Verwitwet
df2 <- subset(df, Familienstand=="Verwitwet")
table(df2$Alter)
```

```
40 65 68 69 75 78 80
 1  1  1  1  1  1  1
```

```r
# Geschieden
df2 <- subset(df, Familienstand=="Geschieden")
table(df2$Alter)
```

```
31 50 58 59 65
 1  1  1  1  1
```

c) Erzeugen Sie für jeden `Familienstand` eine Boxplot des `Alters`. Gibt es Ausreißer? In welcher Gruppe streut das Alter am meisten?

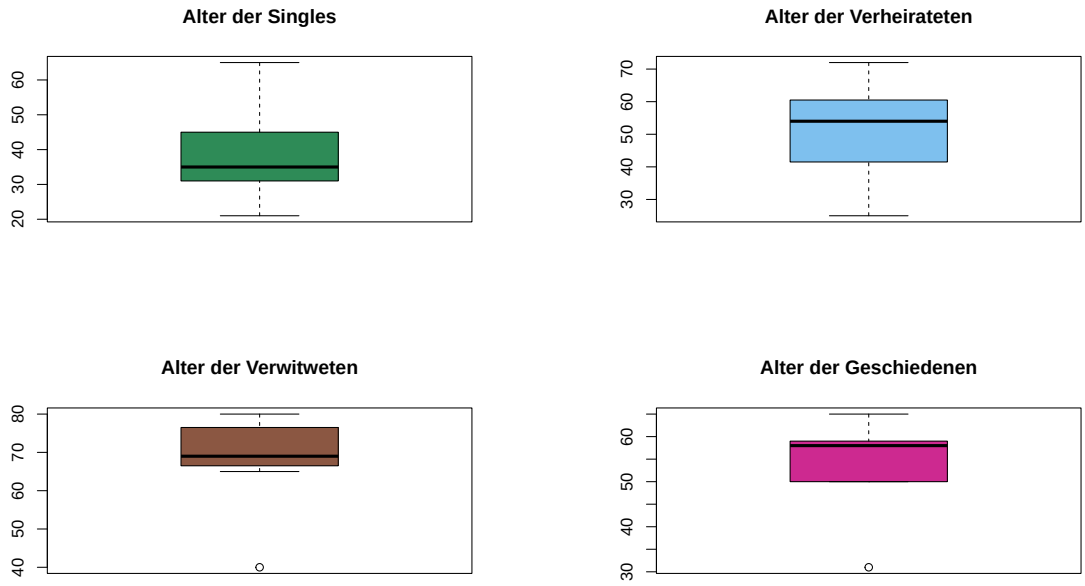
Mit R base können wir so vorgehen.

```r
# Singles
df2 <- subset(df, Familienstand=="Single")
boxplot(df2$Alter, main="Alter der Singles", col="seagreen")
#---------------------------------------

# Verheiratet
df2 <- subset(df, Familienstand=="Verheiratet")
boxplot(df2$Alter, main="Alter der Verheirateten", col="skyblue2")
#---------------------------------------

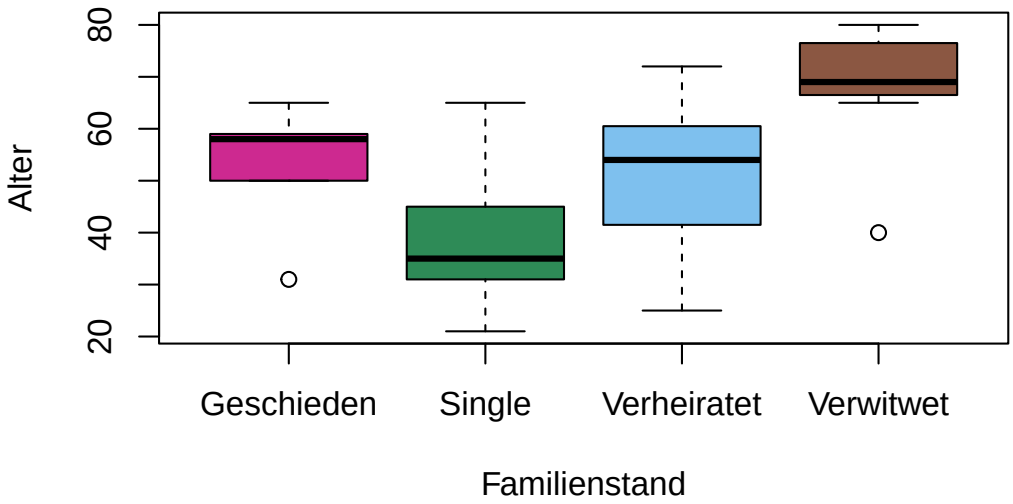
# Verwitwet
df2 <- subset(df, Familienstand=="Verwitwet")
boxplot(df2$Alter, main="Alter der Verwitweten", col="lightsalmon4")
#---------------------------------------

# Geschieden
df2 <- subset(df, Familienstand=="Geschieden")
boxplot(df2$Alter, main="Alter der Geschiedenen", col="maroon3")
#---------------------------------------
```



Oder alle auf einmal:

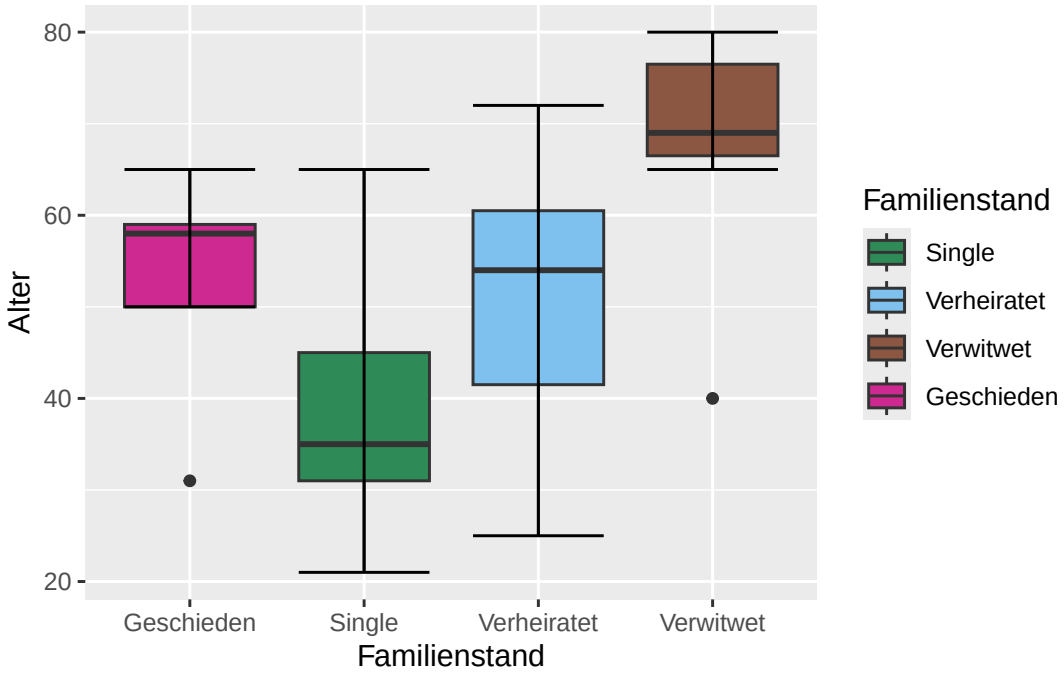
```
boxplot(Alter ~ Familienstand , data=df,
        col=c("maroon3", "seagreen", "skyblue2", "lightsalmon4"))
```



Es sind Ausreißer erkennbar in der Gruppe der Verwitweten und der Geschiedenen.

Im Tidyverse können wir so vorgehen:

```
ggplot(df, aes(y=Alter, x=Familienstand)) +
  geom_boxplot(aes(fill=Familienstand)) +
  stat_boxplot(geom="errorbar")+
  scale_fill_manual(values=c("seagreen", "skyblue2",
                             "lightsalmon4", "maroon3"),
                    breaks=c("Single", "Verheiratet",
                             "Verwitwet", "Geschieden"))
```



Auch hier sind die Ausreißer in den Gruppen der Verwitweten und der Geschiedenen erkennbar.

d) Erzeugen Sie für jeden `Familienstand` eine Histogram des `Alters`. Wie unterscheiden sich die Histogramme?

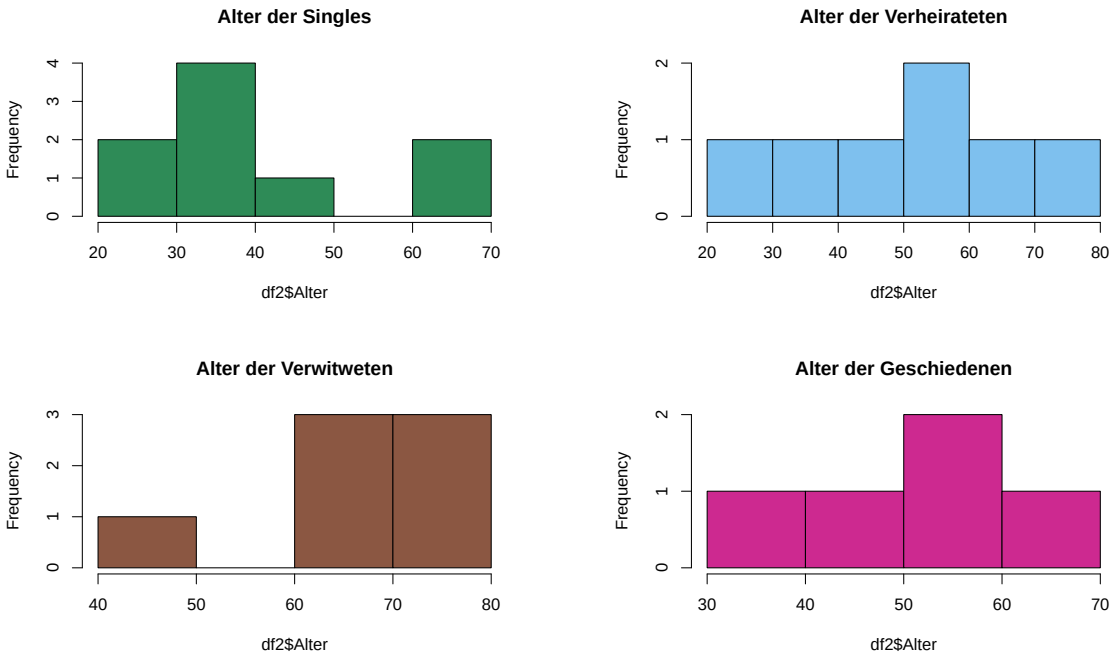
Mit R base können wir so vorgehen

```
# Singles
df2 <- subset(df, Familienstand=="Single")
hist(df2$Alter, main="Alter der Singles", col="seagreen")
#---------------------------------------

# Verheiratet
df2 <- subset(df, Familienstand=="Verheiratet")
hist(df2$Alter, main="Alter der Verheirateten", col="skyblue2")
#---------------------------------------

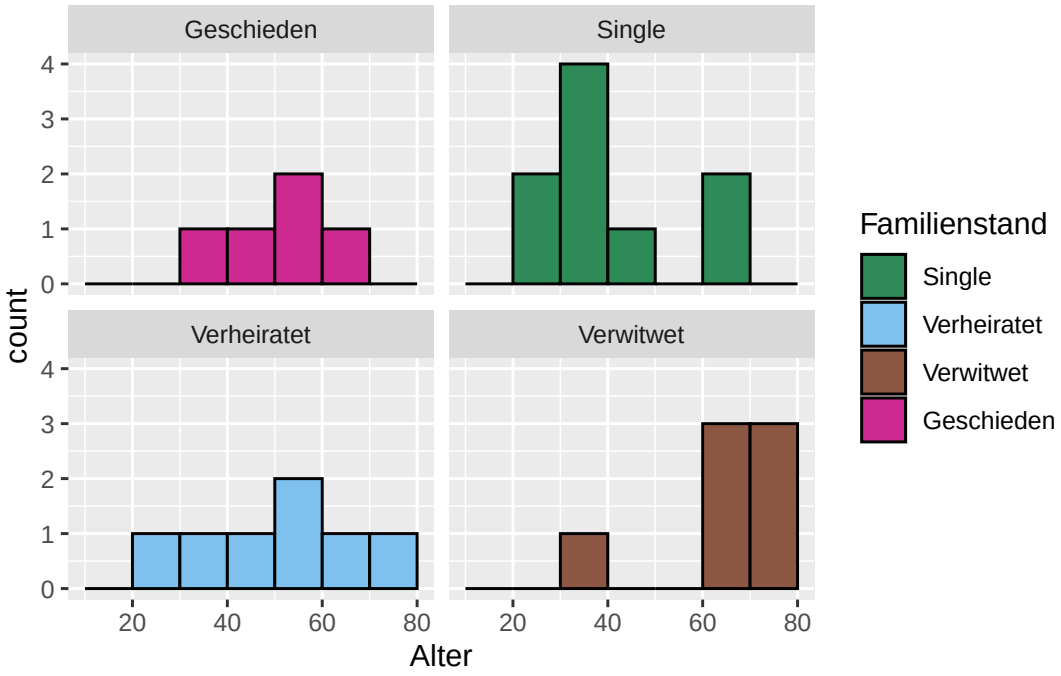
# Verwitwet
df2 <- subset(df, Familienstand=="Verwitwet")
hist(df2$Alter, main="Alter der Verwitweten", col="lightsalmon4")
#---------------------------------------

# Geschieden
df2 <- subset(df, Familienstand=="Geschieden")
hist(df2$Alter, main="Alter der Geschiedenen", col="maroon3")
#---------------------------------------
```



Im Tidyverse können wir so vorgehen:

```
breaks = c(seq(10,80,10))
ggplot(df, aes(x=Alter, fill=Familienstand)) +
    geom_histogram(breaks=breaks, color="black")+
    scale_fill_manual(values=c("seagreen", "skyblue2",
                               "lightsalmon4", "maroon3"),
                      breaks=c("Single", "Verheiratet",
                               "Verwitwet", "Geschieden"))+
    facet_wrap(~Familienstand)
```



4.2.5 Lösung zur Aufgabe 4.1.1.5

a) Erstellen Sie eine Häufigkeitstabelle

```
# erzeuge Daten
Verletzung <- c(0, 1, 2, 1, 3, 0, 1, 0, 1, 2, 0, 1, 1, 1, 2, 0,
                1, 3, 2, 1, 2, 1, 0, 1)

# Häufigkeitstabelle
xtabs(~Verletzung)
```

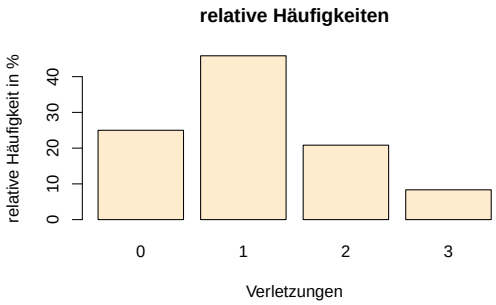
```
Verletzung
 0  1  2  3
```

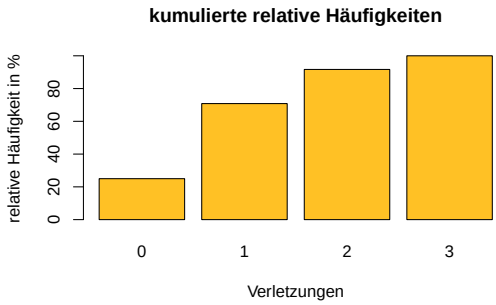
```
 6 11  5  2
```

b) Erzeugen Sie ein Säulendiagramm der relativen und kumulativen relativen Häufigkeiten.

```
# relative Häufigkeiten als Barplot
barplot( table(Verletzung) / length(Verletzung)*100,
        col="blanchedalmond", main="relative Häufigkeiten",
        ylab="relative Häufigkeit in %", xlab="Verletzungen")
#-------------------------------------------------------

# relative kumulierte Häufigkeiten als Barplot
barplot( cumsum(table(Verletzung)) / sum(table(Verletzung))*100,
        col="goldenrod1", main="kumulierte relative Häufigkeiten",
        ylab="relative Häufigkeit in %", xlab="Verletzungen")
```

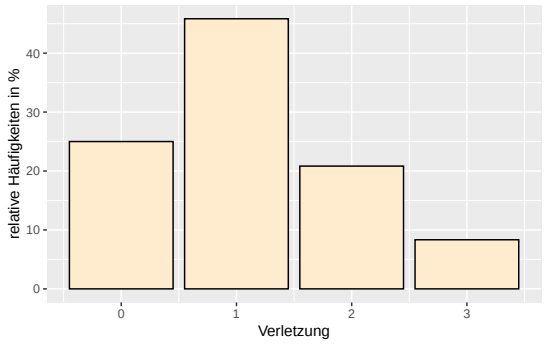


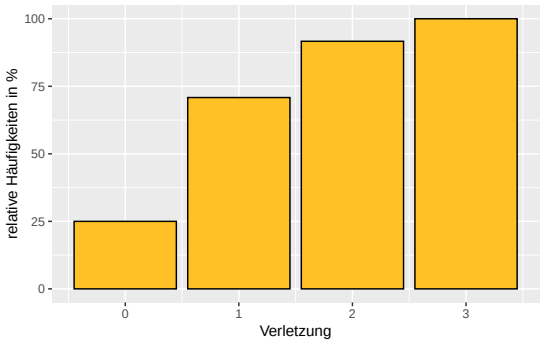


Mit ggplot

```
df <- data.frame(Verletzung)
ggplot(df, aes(x=Verletzung))+
    geom_bar(aes(y=after_stat(count)/sum(after_stat(count))*100),
            fill="blanchedalmond", color="black") +
    ylab("relative Häufigkeiten in %")
#----------------------------

# kumulierte relative Häufigkeiten
ggplot(df, aes(x=Verletzung))+
    ylab("relative Häufigkeiten in %")+
    geom_bar(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))*100),
            fill="goldenrod1", color="black")
```

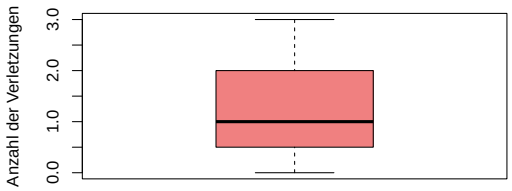


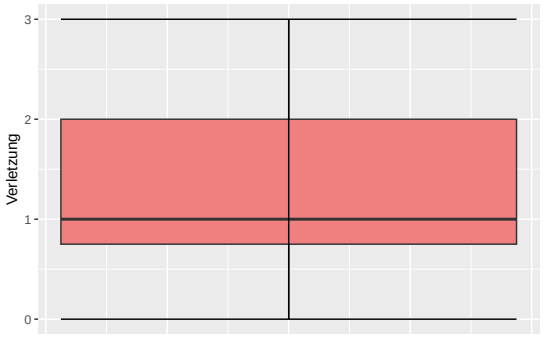


c) Erzeugen Sie ein Boxplot

```
boxplot(Verletzung, col="lightcoral", ylab="Anzahl der Verletzungen")
#--------------------

# mit ggplot
ggplot(df, aes(y=Verletzung)) +
    geom_boxplot(fill="lightcoral") +
    stat_boxplot(geom="errorbar") +
    theme(axis.ticks.x=element_blank(),
          axis.text.x=element_blank())
```





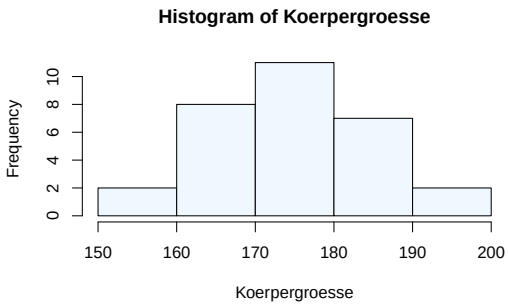
4.2.6 Lösung zur Aufgabe 4.1.1.6

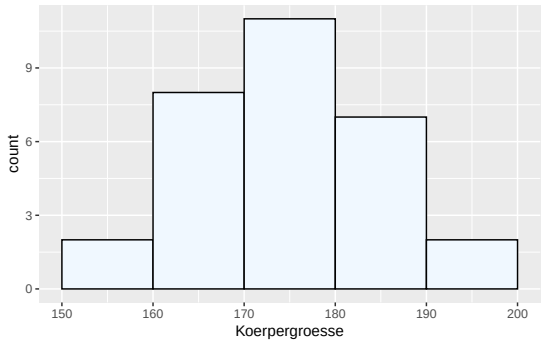
a) Erstellen Sie ein Histogram der Körpergröße mit Klassen von 150cm bis 200cm, die jeweils 10cm breit sind.

```
Koerpergroesse <- c(179, 173, 181, 170, 158, 174, 172, 166, 194, 185,
                    162, 187, 198, 177, 178, 165, 154, 188, 166, 171,
                    175, 182, 167, 169, 172, 186, 172, 176, 168, 187)

hist(Koerpergroesse, breaks=seq(150, 200, 10),
                     col="aliceblue")
#-----------------------

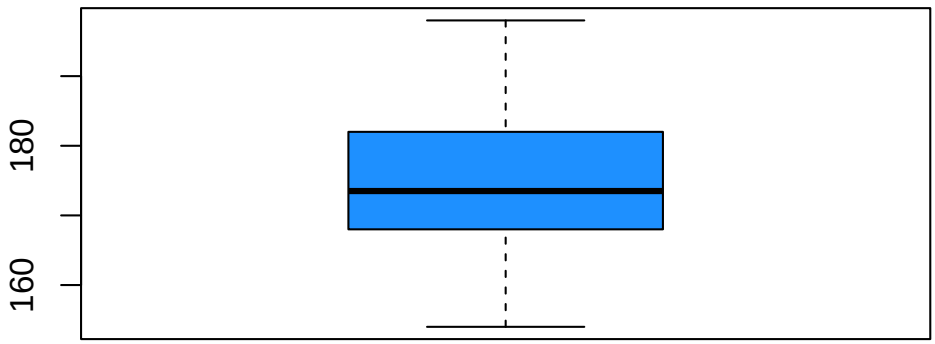
# mit ggplot
as.data.frame(Koerpergroesse) %>%
    ggplot(aes(x=Koerpergroesse)) +
        geom_histogram(breaks=seq(150, 200, 10),
                       fill="aliceblue", color="black")
```





b) Gibt es Ausreißer?

```
boxplot(Koerpergroesse, col="dodgerblue1")
```



Es sind keine Ausreißer erkennbar.

4.2.7 Lösung zur Aufgabe 4.1.1.7

```
# lade Daten
load("https://www.produnis.de/R/data/neonates.RData")
```

```
# Datensatz anschauen
summary(neonates)
```

```
     weight          gender                    age       smoke       cigarettes
 Min.   :2.021   male  :157   greater than 20:218   No :220   Min.   : 0.000
 1st Qu.:2.794   female:163   less than 20   :102   Yes:100   1st Qu.: 0.000
 Median :3.030                                               Median : 0.000
 Mean   :3.026                                               Mean   : 3.891
 3rd Qu.:3.267                                               3rd Qu.: 8.250
 Max.   :4.182                                               Max.   :22.000
 smoke.before     apgar1          apgar5
 No :185      Min.   :2.000   Min.   : 2.000
 Yes:135      1st Qu.:5.000   1st Qu.: 5.000
              Median :6.000   Median : 6.000
              Mean   :5.628   Mean   : 6.213
              3rd Qu.:6.000   3rd Qu.: 7.000
              Max.   :9.000   Max.   :10.000
```

a) Erstellen Sie die Häufigkeitstabelle des APGAR-Scores nach 1 Minute. Wenn ein Score von 3 oder weniger anzeigt, dass das Neugeborene in einem kritischen Zusatand ist, wie viel Prozent der Neugeborenen in der Stichprobe sind dann in einem kritischen Zustand?

```
# neue Variable "kritisch"
neonates$kritisch <- FALSE
# nur solche mit APGAR<4 sind kritisch
neonates$kritisch[neonates$apgar1<4] <- TRUE
# relative Häufigkeiten
table(neonates$kritisch) / length(neonates$kritisch) * 100
```

```
  FALSE    TRUE
92.1875  7.8125
```

7,81% der Neugeborenen sind in einem kritischen Zustand.

b) Erstellen Sie die Häufigkeitstabelle des Geburtsgewichts der Neugeborenen, indem Sie die Daten in Klassen mit einer Breite von 0,5 kg von 2 bis 4,5 kg einteilen. Welches Intervall enthält die meisten Neugeborenen?

```
# neue Variable für die Gewichtsklassifikation
neonates$gewiKat <- cut(neonates$weight, breaks=seq(2, 4.5, 0.5),
                        ordered_results=TRUE)
# einfache Häufigkeitstabelle
table(neonates$gewiKat)
```

```
(2,2.5] (2.5,3] (3,3.5] (3.5,4] (4,4.5]
     22     127     146      24       1
```

```
# oder vollständige
jgsbook::freqTable(neonates$gewiKat)
```

Wert	Haeufig	Hkum	Relativ	Rkum
(2,2.5]	22	22	6.88	6.88
(2.5,3]	127	149	39.69	46.57
(3,3.5]	146	295	45.62	92.19
(3.5,4]	24	319	7.50	99.69
(4,4.5]	1	320	0.31	100.00

Das Intervall von 3-3,5kg enthält die meisten Neugeborenen.

c) Vergleichen Sie die Häufigkeitsverteilung des APGAR-Scores nach 1 Minute für Mütter unter 20 Jahren und für Mütter über 20 Jahren. Welche Gruppe hat mehr Neugeborene in kritischem Zustand?

```
# gruppieren
gruppe1 <- neonates$apgar1[neonates$age=="less than 20"]
gruppe2 <- neonates$apgar1[neonates$age=="greater than 20"]

# Jünger als 20
jgsbook::freqTable(gruppe1)
```

Wert	Haeufig	Hkum	Relativ	Rkum
2	2	2	1.96	1.96
3	11	13	10.78	12.74
4	16	29	15.69	28.43
5	28	57	27.45	55.88
6	28	85	27.45	83.33
7	12	97	11.76	95.09
8	4	101	3.92	99.01
9	1	102	0.98	99.99

```
# Älter als 20
jgsbook::freqTable(gruppe2)
```

Wert	Haeufig	Hkum	Relativ	Rkum
2	2	2	0.92	0.92
3	10	12	4.59	5.51
4	22	34	10.09	15.60
5	53	87	24.31	39.91
6	69	156	31.65	71.56
7	34	190	15.60	87.16
8	24	214	11.01	98.17
9	4	218	1.83	100.00

In der Gruppe der unter-20-jährigen liegt der Prozentsatz an Neugeborenen mit APGAR-Werten kleinergleich 3 bei 12,74%. In der Gruppe der über-20-jährigen liegt der Prozentwert bei 5,51%. Es tritt also in der Gruppe der jüngeren Mütter häufiger auf.

d) Vergleichen Sie die relative Häufigkeitsverteilung des Geburtsgewichts der Neugeborenen, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Wenn ein Gewicht unter 2,5 kg als niedriges Gewicht gilt, welche Gruppe hat einen höheren Prozentsatz an Neugeborenen mit niedrigem Gewicht?

```
# (hatten wir oben schon)
# Gewichtsklassifikation
neonates$gewiKat <- cut(neonates$weight, breaks=seq(2, 4.5, 0.5),
                        right=FALSE, ordered_results=TRUE)
gruppe1 <- neonates$gewiKat[neonates$smoke=="No"]
gruppe2 <- neonates$gewiKat[neonates$smoke=="Yes"]

# Nichtraucherinnen
jgsbook::freqTable(gruppe1)
```

Wert	Haeufig	Hkum	Relativ	Rkum
[2,2.5)	5	5	2.27	2.27
[2.5,3)	75	80	34.09	36.36
[3,3.5)	119	199	54.09	90.45
[3.5,4)	20	219	9.09	99.54
[4,4.5)	1	220	0.45	99.99

```
# Raucherinnen
jgsbook::freqTable(gruppe2)
```

Wert	Haeufig	Hkum	Relativ	Rkum
[2,2.5)	17	17	17	17
[2.5,3)	52	69	52	69
[3,3.5)	27	96	27	96
[3.5,4)	4	100	4	100
[4,4.5)	0	100	0	100

In der Gruppe der Nichtraucherinnen trat ein Geburtsgewicht kleiner 2,5kg in 2,27% der Fälle auf. Bei den Raucherinnen waren es 17%.

e) Berechnen Sie die Prävalenz von Neugeborenen mit niedrigem Gewicht für Mütter, die vor der Schwangerschaft geraucht haben, und den Nichtraucherinnen.

```
gruppe1 <- neonates$gewiKat[neonates$smoke.before=="No"]
gruppe2 <- neonates$gewiKat[neonates$smoke.before=="Yes"]

# Nichtraucherinnen
jgsbook::freqTable(gruppe1)
```

Wert	Haeufig	Hkum	Relativ	Rkum
[2,2.5)	2	2	1.08	1.08
[2.5,3)	60	62	32.43	33.51
[3,3.5)	105	167	56.76	90.27
[3.5,4)	18	185	9.73	100.00
[4,4.5)	0	185	0.00	100.00

```
# Raucherinnen
jgsbook::freqTable(gruppe2)
```

Wert	Haeufig	Hkum	Relativ	Rkum
[2,2.5)	20	20	14.81	14.81
[2.5,3)	67	87	49.63	64.44
[3,3.5)	41	128	30.37	94.81
[3.5,4)	6	134	4.44	99.25
[4,4.5)	1	135	0.74	99.99

Die Prävalenz beträgt unter den Nichtraucherinnen 1,08% und unter den Raucherinnen 14,81%.

f) Berechnen Sie das relative Risiko eines niedrigen Geburtsgewichts des Neugeborenen, wenn die Mutter während der Schwangerschaft raucht, im Vergleich dazu, wenn die Mutter nicht raucht.

```
# neue binäre Variable, ob Gewicht niedrig ist
neonates$gewiLow <- FALSE
neonates$gewiLow[neonates$gewiKat=="[2,2.5)"] <- TRUE

# Kreuztabelle
table(neonates$smoke, neonates$gewiLow)
```

```
       FALSE TRUE
```

```
    No    215    5
    Yes    83   17
```

relatives Risiko $= \frac{a \cdot (c+d)}{c \cdot (a+b)}$

```
# Kreuztabelle als numerische Werte
tab <- as.numeric(table(neonates$smoke, neonates$gewiLow))

# rechne das relative Risiko nach der obigen Formel
( tab[1] * (tab[2]+tab[4]) ) / ( tab[2] * (tab[1]+tab[3]) )
```

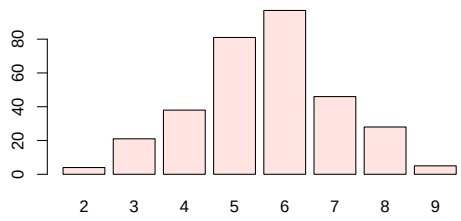
```
[1] 1.177437
```

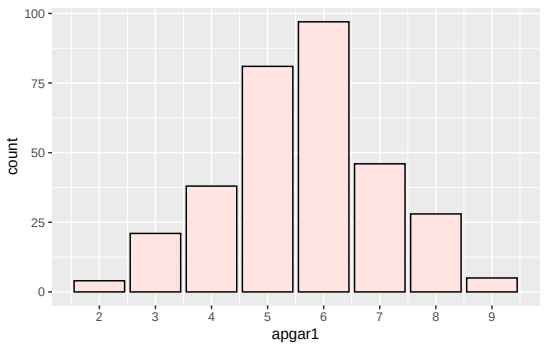
Raucherinnen haben ein 1,177437-fach höheres Risiko ein Kind mit niedrigem Gewicht zugebären als Nichtraucherinnen. Die Wahrscheinlichkeit ist in der Raucherinnengruppe also 17,74% höher als bei den Nichtraucherinnen.

g) Erstellen Sie ein Balkendiagramm des APGAR-Scores nach 1 Minute. Welcher Score ist am häufigsten?

```
# mit R base
barplot(table(neonates$apgar1), col="mistyrose")

# mit ggplot
ggplot(neonates, aes(x=apgar1)) +
  geom_bar(color="black", fill="mistyrose")+
  scale_x_continuous(breaks=seq(2, 9, 1))
```



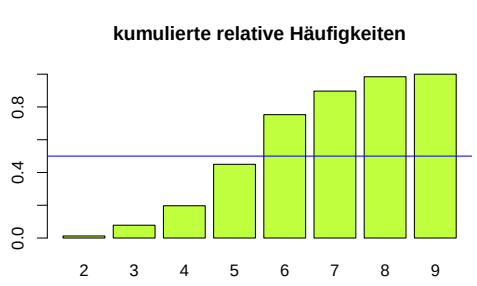


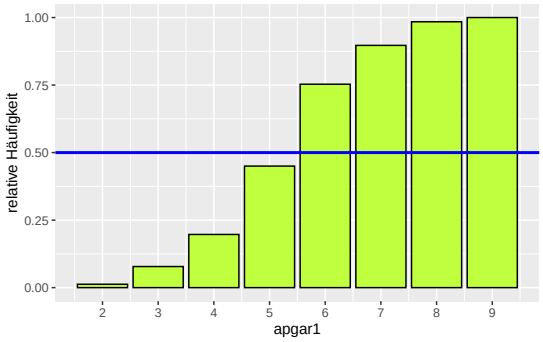
Am häufigsten tritt Wert 6 auf.

h) Erstellen Sie das Balkendiagramm der kumulierten relativen Häufigkeit des APGAR-Scores nach 1 Minute. Unter welchem Wert liegen die Hälfte der Neugeborenen?

```
# mit R base
# plotte das kumulative Balkendiagramm
barplot(cumsum(table(neonates$apgar1))/sum(table(neonates$apgar1)),
        col="olivedrab1", main = "kumulierte relative Häufigkeiten")
# Linie bei 50% ziehen
abline(h=0.5, col="blue")
#------------------------

# mit ggplot()
ggplot(neonates, aes(x=apgar1)) +
  geom_bar(aes(y=cumsum(after_stat(count)/sum(after_stat(count)))),
        fill="olivedrab1", color="black") +
  ylab("relative Häufigkeit") +
  geom_hline(yintercept= 0.5, color="blue", linewidth=1) +
  scale_x_continuous(breaks=seq(2, 9, 1))
```



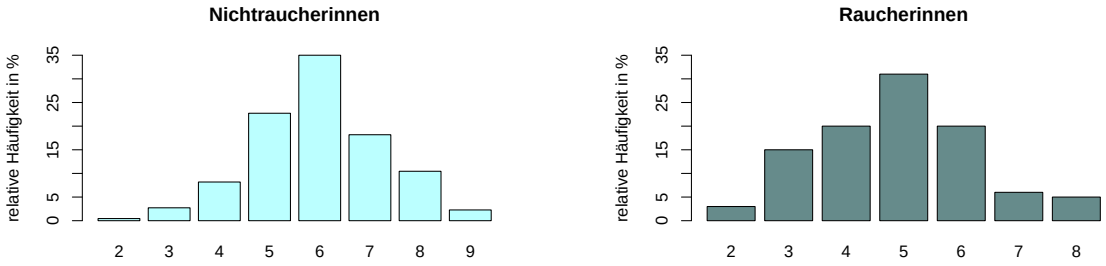


Der Median liegt bei 6.

i) Vergleichen Sie die Balkendiagramme der relativen Häufigkeitsverteilungen des APGAR-Scores nach 1 Minute, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Welche Schlussfolgerungen können gezogen werden?

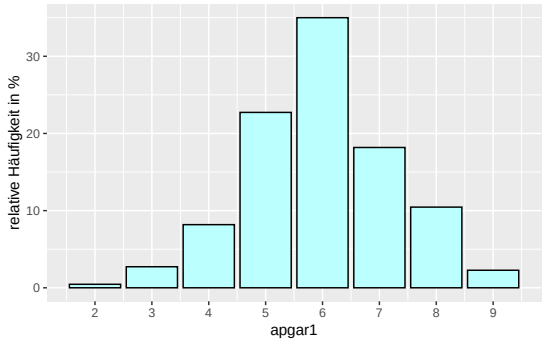
```
# mit R base
gruppe1 <- neonates$apgar1[neonates$smoke=="No"]
gruppe2 <- neonates$apgar1[neonates$smoke=="Yes"]

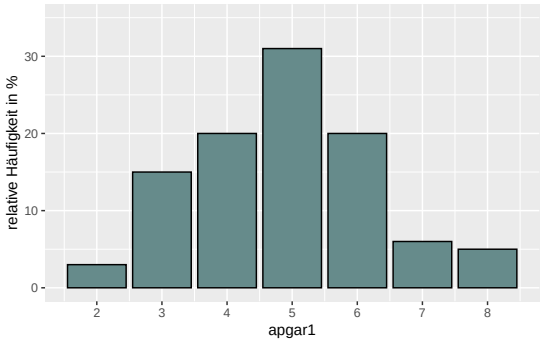
barplot( table(gruppe1)/sum(table(gruppe1)) *100,
        ylab="relative Häufigkeit in %", main="Nichtraucherinnen",
        ylim=c(0,35), col="paleturquoise1")
barplot( table(gruppe2)/sum(table(gruppe2)) *100,
        ylab="relative Häufigkeit in %", main="Raucherinnen",
        ylim=c(0,35), col="paleturquoise4")
```



```r
# mit ggplot
# Nichtraucherinnen
neonates %>%
  filter(smoke=="No") %>%
  ggplot(aes(x=apgar1))+
  geom_bar(aes(y=after_stat(count)/sum(after_stat(count))*100),
           color="black", fill="paleturquoise1")+
  scale_x_continuous(breaks=seq(2, 9, 1)) +
  ylab("relative Häufigkeit in %") +
  ylim(0,35)
#------------------------------

# Raucherinnen
neonates %>%
  filter(smoke=="Yes") %>%
  ggplot(aes(x=apgar1))+
  geom_bar(aes(y=after_stat(count)/sum(after_stat(count))*100),
           color="black", fill="paleturquoise4") +
  scale_x_continuous(breaks=seq(2, 9, 1)) +
  ylab("relative Häufigkeit in %") +
  ylim(0,35)
```



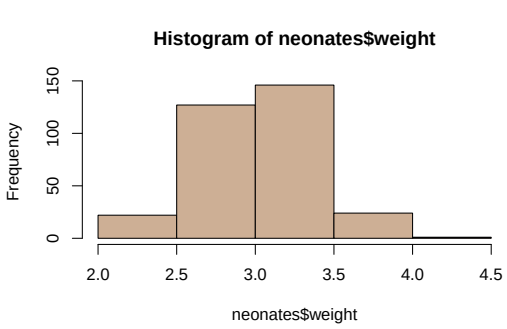


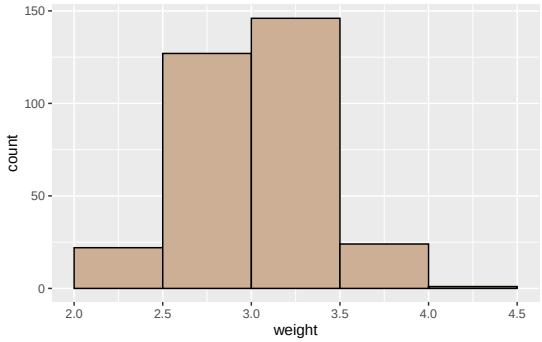
Die Kinder der Raucherinnen haben geringere APGAR-Werte.

j) Erstellen Sie ein Histogramm der Geburtsgewichte der Neugeborenen mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg. Welche Klasse enthält die meisten Neugeborenen?

```
# mit R base
hist(neonates$weight, breaks = seq(2, 4.5, 0.5),
    col="peachpuff3")
#---------------------

# mit ggplot
ggplot(neonates, aes(x=weight)) +
  geom_histogram( breaks = seq(2, 4.5, 0.5),
                  fill="peachpuff3", color="black")
```



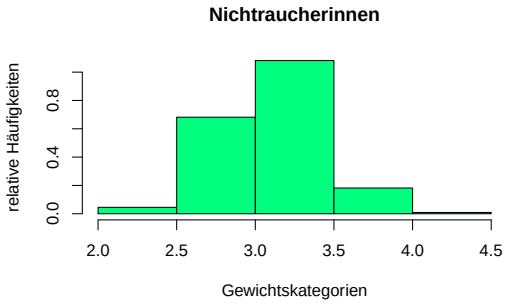


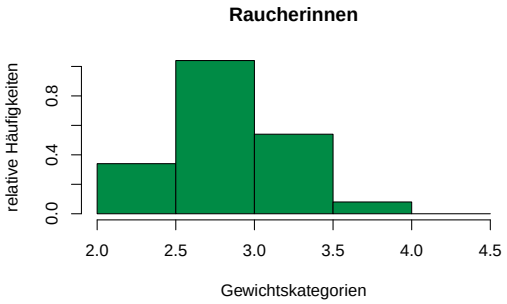
Die Gewichtsklasse 3kg-3,5kg enthält die meisten Neugeborenen

k) Vergleichen Sie die relativen Häufigkeitshistogramme der Geburtsgewichte der Neugeborenen, mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg, je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht. Welche Gruppe hat Neugeborene mit geringeren Gewichten?

```
# mit R base
hist(neonates$weight[neonates$smoke=="No"],
     breaks=seq(2, 4.5, 0.5), col="springgreen1",
     main="Nichtraucherinnen", xlab="Gewichtskategorien",
     ylab="relative Häufigkeiten", freq=FALSE)
#---------------------

# Raucherinnen
hist(neonates$weight[neonates$smoke=="Yes"],
     breaks=seq(2, 4.5, 0.5), col="springgreen4",
     main="Raucherinnen", xlab="Gewichtskategorien",
     ylab="relative Häufigkeiten", freq=FALSE)
```

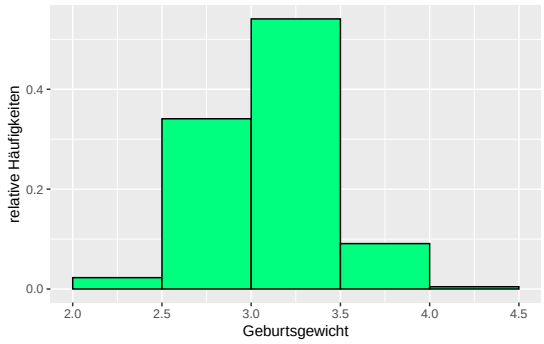


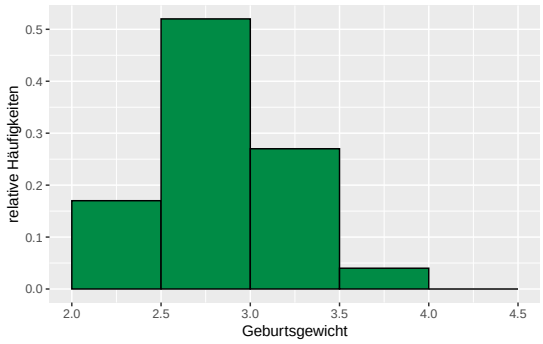



```
# mit ggplot

neonates %>%
  filter(smoke=="No") %>%
  ggplot(aes(x=weight)) +
    geom_histogram(aes(y=after_stat(count)/sum(after_stat(count))),
                   breaks=seq(2, 4.5, 0.5),
                   fill="springgreen1", color="black") +
    ylab("relative Häufigkeiten") + xlab("Geburtsgewicht")
#-----------

# Raucherinnen
neonates %>%
  filter(smoke=="Yes") %>%
  ggplot(aes(x=weight)) +
    geom_histogram(aes(y=after_stat(count)/sum(after_stat(count))),
                   breaks=seq(2, 4.5, 0.5),
                   fill="springgreen4", color="black") +
    ylab("relative Häufigkeiten") + xlab("Geburtsgewicht")
```



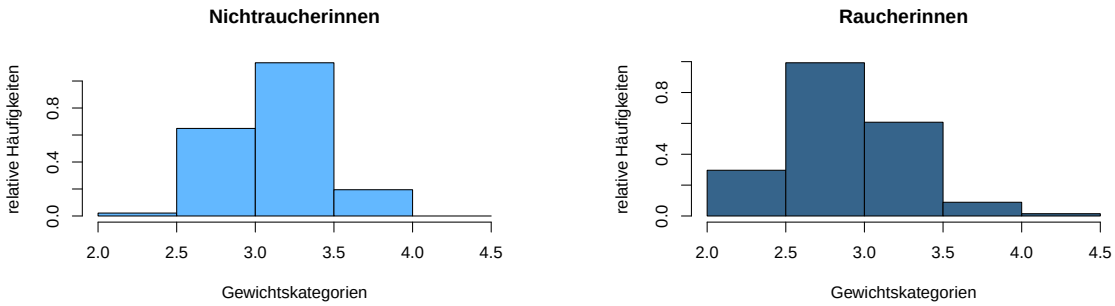


Kinder von Raucherinnen haben durchschnittlich weniger Geburtsgewicht.

l) Vergleichen Sie die relativen Häufigkeitshistogramme der Geburtsgewichte der Neugeborenen, mit Klassenbreiten von 0,5 kg von 2 bis 4,5 kg, je nachdem, ob die Mutter vor der Schwangerschaft geraucht hat oder nicht. Welche Schlussfolgerungen können gezogen werden?

```
# mit R base
hist(neonates$weight[neonates$smoke.before=="No"],
     breaks=seq(2, 4.5, 0.5), col="steelblue1",
     main="Nichtraucherinnen", xlab="Gewichtskategorien",
     ylab="relative Häufigkeiten", freq=FALSE)
#---------------------

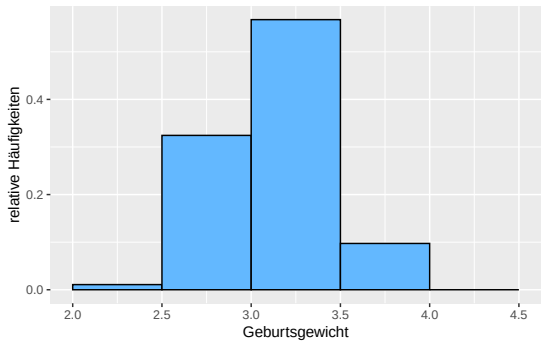
# Raucherinnen
hist(neonates$weight[neonates$smoke.before=="Yes"],
     breaks=seq(2, 4.5, 0.5), col="steelblue4",
     main="Raucherinnen", xlab="Gewichtskategorien",
     ylab="relative Häufigkeiten", freq=FALSE)
```

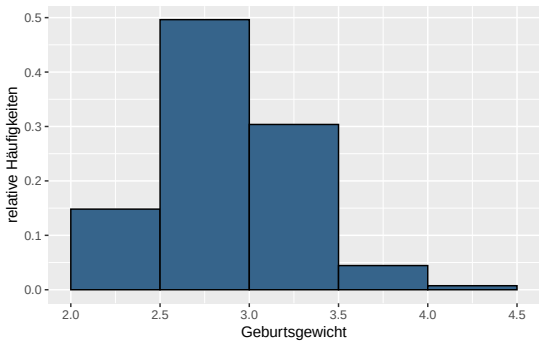


```
# mit ggplot

neonates %>%
  filter(smoke.before=="No") %>%
  ggplot(aes(x=weight)) +
    geom_histogram(aes(y=after_stat(count)/sum(after_stat(count))),
                   breaks=seq(2, 4.5, 0.5),
                   fill="steelblue1", color="black") +
    ylab("relative Häufigkeiten") + xlab("Geburtsgewicht")
#-----------

# Raucherinnen
neonates %>%
  filter(smoke.before=="Yes") %>%
  ggplot(aes(x=weight)) +
    geom_histogram(aes(y=after_stat(count)/sum(after_stat(count))),
                   breaks=seq(2, 4.5, 0.5),
                   fill="steelblue4", color="black") +
    ylab("relative Häufigkeiten") + xlab("Geburtsgewicht")
```

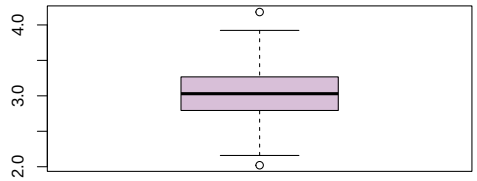


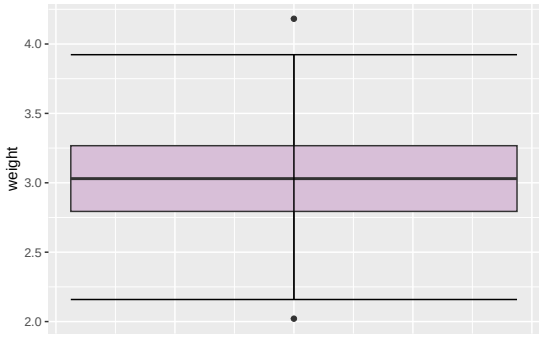


Kinder von Müttern, die vor der Schwangerschaft geraucht haben, haben durchschnittlich weniger Geburtsgewicht.

m) Erstellen Sie ein Boxplot der Geburtsgewichte der Neugeborenen. Welcher Gewichtsbereich kann in der Stichprobe als normal angesehen werden? Gibt es Ausreißer in der Stichprobe?

```
# mit R base
boxplot(neonates$weight, col="thistle")
# mit ggplot()
ggplot(neonates, aes(y=weight)) +
  geom_boxplot(fill="thistle") +
  stat_boxplot(geom="errorbar") +
  theme(axis.ticks.x=element_blank(),
        axis.text.x=element_blank())
```





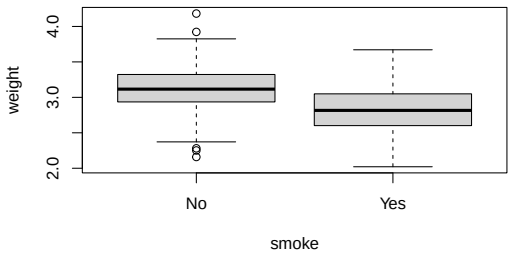

```
# Zusammenfassung
summary(neonates$weight)
```

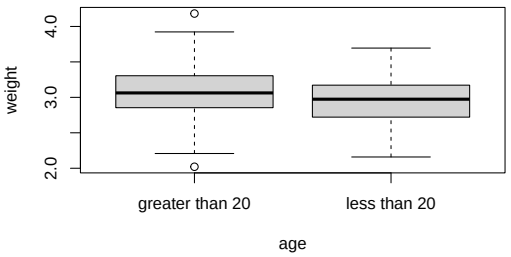
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.021   2.794   3.030   3.026   3.267   4.182
```

Gewichte zwischen 2,794kg und 3,267kg können als normal angesehen werden. Es gibt je einen Ausreißer nach oben und nach unten.

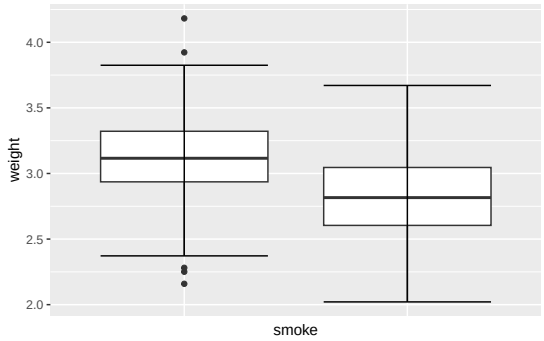
n) Vergleichen Sie die Boxplots der Geburtsgewichte der Neugeborenen je nachdem, ob die Mutter während der Schwangerschaft geraucht hat oder nicht und ob die Mutter unter 20 oder über 20 Jahre alt war. Welche Gruppe hat eine größere zentrale Streuung? Welche Gruppe hat Neugeborene mit geringerem Gewicht?

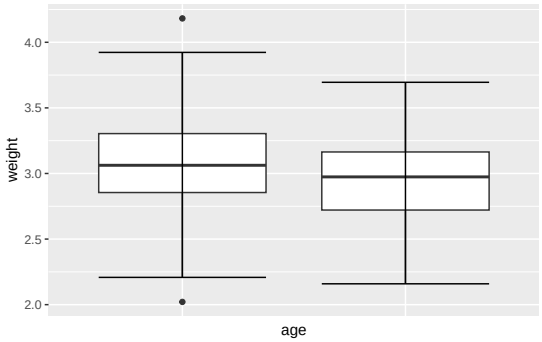
```
# R base
boxplot(weight ~ smoke, data=neonates)
# Alterkategorie
boxplot(weight ~ age, data=neonates)
```






```
# ggplot Rauchen
ggplot(neonates, aes(y=weight, x=smoke)) +
  geom_boxplot() +
  stat_boxplot(geom="errorbar") +
  theme(axis.ticks.x=element_blank(),
        axis.text.x=element_blank())
#-------------------------------------
# 20 Jahre alt
ggplot(neonates, aes(y=weight, x=age)) +
  geom_boxplot() +
  stat_boxplot(geom="errorbar") +
  theme(axis.ticks.x=element_blank(),
        axis.text.x=element_blank())
```





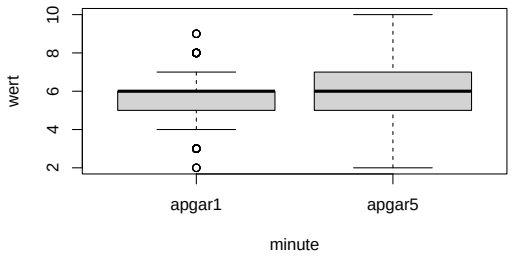
Das Gewicht ist in der Gruppe der Raucherinnen und in der Gruppe der unter-20jährigen geringer.

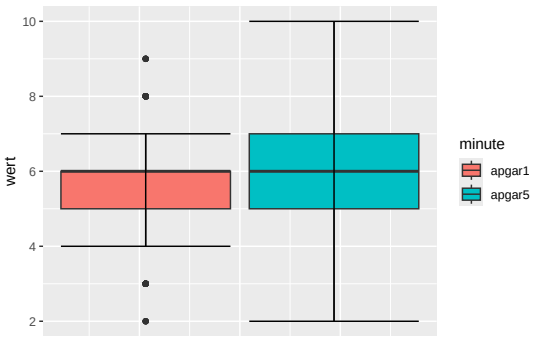
o) Vergleichen Sie die Boxplots der APGAR-Scores nach 1 Minute und nach 5 Minuten. Welche Variable hat eine größere zentrale Streuung?

```
# daten tidy machen
df <- pivot_longer(neonates, apgar1:apgar5,
                   names_to  = "minute",
                   values_to = "wert")

# boxplot
boxplot(wert ~ minute, data=df)
#--------------------------------

# ggplot
ggplot(df, aes(y=wert, fill=minute)) +
  geom_boxplot() +
  stat_boxplot(geom="errorbar") +
  theme(axis.ticks.x=element_blank(),
        axis.text.x=element_blank())
```





Die Streuung in `apgar5` ist größer.

4.2.8 Lösung zur Aufgabe 4.1.2.1

a) Erstellen Sie ein Datenframe mit der Variable `Kinder` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Kinder = c(1, 2, 4, 2, 2, 2, 3, 2, 1, 1, 0, 2, 2, 0,
                            2, 2, 1, 2, 2, 3, 1, 2, 2, 1, 2))
```

b) Berechnen Sie das arithmetische Mittel, die Varianz sowie die Standardabweichung für die Anzahl an Kindern.

```
# arithmetisches Mittel
mean(df$Kinder)
```

```
[1] 1.76
```

```
# Varianz
var(df$Kinder)
```

```
[1] 0.7733333
```

```
# Standardabweichung
sd(df$Kinder)
```

```
[1] 0.8793937
```

c) Berechnen Sie die Quartile, die Spannweite, den Interquartilsabstand, das dritte Dezil sowie das 68te Perzentil.

```
# Quartile (so wie SPSS rechnet)
quantile(df$Kinder, type=6)
```

```
  0%  25%  50%  75% 100%
   0    1    2    2    4
```

```
# Spannweite
range(df$Kinder)
```

```
[1] 0 4
```

```
# Interquartilsabstand (so wie SPSS rechnet)
IQR(df$Kinder, type=6)
```

```
[1] 1
```

```
# drittes Dezil und 68tes Perzentil
quantile(df$Kinder, probs=c(0.3, 0.68), type=6)
```

```
30% 68%
  1   2
```

4.2.9 Lösung zur Aufgabe 4.1.2.2

a) Erstellen Sie ein Datenframe mit der Variable `Patienten` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Patienten = c(15, 23, 12, 10, 28, 50, 12, 17, 20,
                               21, 18, 13, 11, 12, 26, 30,  6, 16,
                               19, 22, 14, 17, 21, 28,  9, 16, 13,
                               11, 16, 20))
```

b) Berechnen Sie das arithmetische Mittel, die Varianz, die Standardabweichung und den Variationskoeffizienten.

```
# arithmetisches Mittel
mean(df$Patienten)
```

```
[1] 18.2
```

```
# Varianz
var(df$Patienten)
```

```
[1] 71.82069
```

```
# Standardabweichung
sd(df$Patienten)
```

```
[1] 8.474709
```

```
# Variationskoeffizient in Prozent
(sd(df$Patienten) / mean(df$Patienten)) * 100
```

```
[1] 46.56433
```

c) Berechnen Sie die Skewness (Schiefe) und Kurtosis (“Spitzigkeit”) und interpretieren Sie die Werte.

```
# Skewness
psych::skew(df$Patienten, type=2) # rechne wie SPSS
```

```
[1] 1.902838
```

```
# Kurtosis
psych::kurtosi(df$Patienten, type=2) # rechne wie SPSS
```

```
[1] 5.796082
```

Die Skewness beträgt 1.902838, was für eine rechtsschiefe Verteilung spricht. Die Kurtosis beträgt 5.796082 und ist somit größer als 3. Das bedeutet, die Verteilung hat eine schmale Spitze und fette Ränder.

4.2.10 Lösung zur Aufgabe 4.1.2.3

a) Erstellen Sie ein Datenframe mit der Variable `Bewertung` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Bewertung = c("SS", "AP", "SS", "AP", "AP", "NT", "NT", "AP",
                               "SB", "SS", "SB", "SS", "AP", "AP", "NT", "AP",
                               "SS", "NT", "SS", "NT"))
```

b) Wandeln Sie die `Bewertung` in Punkte um nach dem Schema “SS” = 2,5 | “AP” = 6 | “NT”= 8 | “SB” = 9,5.

```
# Umkodieren
df$Punkte[df$Bewertung=="SS"] <- 2.5
df$Punkte[df$Bewertung=="AP"] <- 6
df$Punkte[df$Bewertung=="NT"] <- 8
df$Punkte[df$Bewertung=="SB"] <- 9.5
```

c) Bestimmen Sie den Median und den Interquartilsabstand.

```
# Median
median(df$Punkte)
```

```
[1] 6
```

```
# Interquartilsabstand, so wie SPSS ihn rechnen würde
IQR(df$Punkte, type=6)
```

```
[1] 5.5
```

4.2.11 Lösung zur Aufgabe 4.1.2.4

a) Erstellen Sie ein Datenframe mit der Variablen `Geschlecht` und `Koerpergroesse` und übertragen Sie die Daten.

```
# erzeuge Datenframe
df <- data.frame(Geschlecht = c(rep("weiblich", 14),
                                rep("männlich", 16)),
                 Koerpergroesse = c(173, 158, 174, 166, 162, 177, 165, 154, 166,
                 182, 169, 172, 170, 168, 179, 181, 172, 194, 185, 187, 198,
                 178, 188, 171, 175, 167, 186, 172, 176, 187))
```

b) Bestimmen Sie in Abhängigkeit zum `Geschlecht` das arithmetische Mittel, den Median, die Varianz, die Standardabweichung sowie die Quartile.

```
# mit dplyr
df %>% group_by(Geschlecht) %>%
  reframe(aritMittel = mean(Koerpergroesse),
          Median = median(Koerpergroesse),
          Varianz = var(Koerpergroesse),
          StdAbw  = sd(Koerpergroesse),
          Q1 = quantile(Koerpergroesse, probs=0.25, type=6),
          Q3 = quantile(Koerpergroesse, probs=0.75, type=6)
  )
```

Geschlecht	aritMittel	Median	Varianz	StdAbw	Q1	Q3
männlich	181.0000	180.0	76.80000	8.763561	172.75	187.00

weiblich	168.2857	168.5	54.37363	7.373848	164.25	173.25

4.2.12 Lösung zur Aufgabe 4.1.2.5

a) Bestimmen Sie das arithmetische Mittel, den Median, die Varianz sowie die Standardabweichung der Verletzungen.

```
# Daten übertragen
Handball <- c(0, 1, 2, 1, 3, 0, 1, 0, 1, 2, 0, 1, 1,
              1, 2, 0, 1, 3, 2, 1, 2, 1, 0, 1)

# Mittelwert
mean(Handball)
```

```
[1] 1.125
```

```
# Median
median(Handball)
```

```
[1] 1
```

```
# Varianz
var(Handball)
```

```
[1] 0.8097826
```

```
# Standardabweichung
sd(Handball)
```

```
[1] 0.8998792
```

b) Bestimmen Sie die Skewness und Kortosis der Verteilung.

```
# Skewness, wie SPSS
psych::skew(Handball, type=2)
```

```
[1] 0.5186785
```

```r
# Kurtosis, wie SPSS
psych::kurtosi(Handball, type=2)
```

```
[1] -0.226357
```

c) Berechnen Sie das vierte und achte Dezil der Verteilung.

```r
quantile(Handball, probs=c(0.4, 0.8), type=6)
```

```
40% 80%
  1   2
```

4.2.13 Lösung zur Aufgabe 4.1.2.6

Welcher Monitor funktioniert besser?

```r
# Daten übertragen
monitor <- data.frame(Unterarm=c(111, 109, 112, 111, 113, 113, 114, 111),
                      Handgelenk=c(115, 113, 117, 116, 112, 112, 117, 112)
)

# Standardabweichungen vergleichen
sd(monitor$Unterarm)
```

```
[1] 1.581139
```

```r
sd(monitor$Handgelenk)
```

```
[1] 2.251983
```

Die Daten des Monitors am Handgelenk haben eine größere Streuung als jene vom Monitor am Unterarm.

4.2.14 Lösung zur Aufgabe 4.1.2.7

a) Bestimmen Sie das arithmetische Mittel, den Median, die Varianz sowie die Standardabweichung des `Alters` für jeden `Familienstand`.

```
# Daten übertragen
df <- data.frame(Familienstand = c(rep("Single", 9),
                                   rep("Verheiratet", 7),
                                   rep("Verwitwet", 7),
                                   rep("Geschieden", 5)),
                 Alter = c(31, 45, 35, 65, 21, 38, 62, 22, 31,
                           72, 39, 62, 59, 25, 44, 54,
                           80, 68, 65, 40, 78, 69, 75,
                           31, 65, 59, 58, 50)
                 )

# dplyr
df %>% group_by(Familienstand) %>%
       reframe(Mittel = mean(Alter),
               Median = median(Alter),
               Varianz = var(Alter),
               StdAbw = sd(Alter))
```

Familienstand	Mittel	Median	Varianz	StdAbw
Geschieden	52.60000	58	174.3000	13.20227
Single	38.88889	35	249.8611	15.80700
Verheiratet	50.71429	54	250.5714	15.82945
Verwitwet	67.85714	69	181.1429	13.45893

Welche Gruppe hat den "besten" Mittelwert?

```
df %>% group_by(Familienstand) %>%
       reframe(Mittel = mean(Alter),
               Median = median(Alter),
               Varianz = var(Alter),
               StdAbw = sd(Alter),
               Skew = psych::skew(Alter, type=2),
               Kurto = psych::kurtosi(Alter, type=2))
```

Familienstand	Mittel	Median	Varianz	StdAbw	Skew	Kurto
Geschieden	52.60000	58	174.3000	13.20227	-1.4067277	2.0349803
Single	38.88889	35	249.8611	15.80700	0.7646514	-0.5243790
Verheiratet	50.71429	54	250.5714	15.82945	-0.4251562	-0.3129758

Verwitwet	67.85714	69	181.1429	13.45893	-1.7649555	3.6623934

In der Gruppe der Geschiedenen ist die Streuung am geringsten, aber das arithmetische Mittel ist weit vom Median entfernt. Insofern scheint der Mittelwert der Verwitweten am "besten" zu sein, da er sowohl eine geringe Differenz zum Median als auch eine niedrige Varianz und Standardabweichung aufweist. Die Gruppe der Verheirateten weist hingegen die geringste Schiefe auf. Ach hier könnten wir argumentieren, dass dies für den "besten" Mittelwert spräche.

4.2.15 Lösung zur Aufgabe 4.1.2.8

a) Vergleichen Sie das arithmetische Mittel, die Standardabweichung, die Skewness und Kurtosis des `Blutdrucks` zwischen Rauchern und Nichtrauchern.

```
# Daten übertragen
df <- data.frame(Rauchen = c("ja", "nein", "ja", "ja", "ja", "nein", "nein",
                             "ja", "nein", "ja", "nein", "ja", "nein",
                             "ja", "nein", "nein", "ja", "nein", "nein",
                             "nein", "ja", "nein", "ja", "nein", "ja"  ),
                 Alkohol = c("nein", "nein", "ja", "ja", "nein", "nein", "ja",
                             "ja", "nein", "ja", "nein", "ja", "ja" , "ja",
                             "nein", "ja", "ja", "nein", "nein", "ja", "ja",
                             "ja", "nein", "ja", "nein" ),
                 Blutdruck = c(80, 92, 75, 56, 89, 93, 101, 67, 89, 63, 98, 58,
                               91, 71, 52, 98, 104, 57, 89, 70, 93, 69, 82, 70,
                               49 )
                 )

# dplyr
df %>% group_by(Rauchen) %>%
       reframe(Mittel = mean(Blutdruck),
               StdAbw = sd(Blutdruck),
               Skew =  psych::skew(Blutdruck, type=2),
               Kurto = psych::kurtosi(Blutdruck, type=2))
```

Rauchen	Mittel	StdAbw	Skew	Kurto
ja	73.91667	16.43421	0.2814578	-0.6239070
nein	82.23077	16.46792	-0.6998100	-0.9345984

b) Vergleichen Sie die selben Werte zwischen der Alkohol- und Nicht-Alkoholgruppe.

```
# dplyr
df %>% group_by(Alkohol) %>%
       reframe(Mittel = mean(Blutdruck),
               StdAbw = sd(Blutdruck),
               Skew =  psych::skew(Blutdruck, type=2),
               Kurto = psych::kurtosi(Blutdruck, type=2))
```

Alkohol	Mittel	StdAbw	Skew	Kurto
ja	77.57143	16.39284	0.4549274	-1.3206125
nein	79.09091	17.74517	-0.9484563	-0.8040328

c) Vergleichen Sie die selben Werte zwischen der Raucher- und Alkoholgruppe, zwischen der Raucher- und Nicht-Alkoholgruppe, der Nichtraucher- und Alkoholgruppe sowie der Nichtraucher- und Nicht-Alkoholgruppe.

```
# dplyr
df %>% group_by(Alkohol, Rauchen) %>%
       reframe(Mittel = mean(Blutdruck),
               StdAbw = sd(Blutdruck),
               Skew =  psych::skew(Blutdruck, type=2),
               Kurto = psych::kurtosi(Blutdruck, type=2))
```

Alkohol	Rauchen	Mittel	StdAbw	Skew	Kurto
ja	ja	73.37500	16.97845	1.0124993	0.0026964
ja	nein	83.16667	15.14486	0.1733778	-2.8742732
nein	ja	75.00000	17.75763	-1.7101307	3.2083168
nein	nein	81.42857	18.69810	-1.1391312	-0.7477992

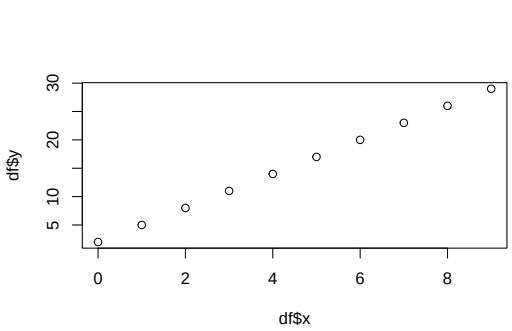
4.2.16 Lösung zur Aufgabe 4.1.3.1

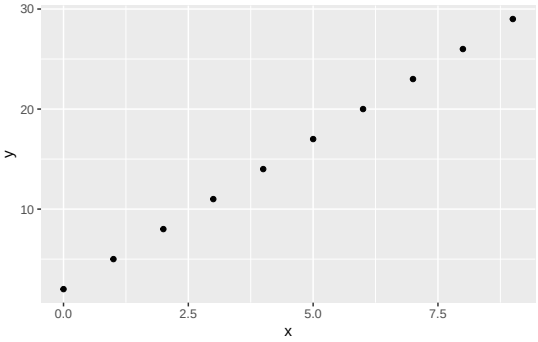
a) Erstellen Sie ein Datenframe mit den Variablen `x` und `y`.

```
# erzeuge Datenframe
df <- data.frame(x = c( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
                 y = c( 2, 5, 8, 11, 14, 17, 20, 23, 26, 29))
```

b) Erzeugen Sie ein Scatterplot von `x` und `y`. Bestimmen Sie anhand des Plots, welche Regressionsfunktion die Daten am besten erklären würde.

```
# plot()
plot(df$x, df$y)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point()
```





Es ist ein deutlicher linearer Zusammenhang erkennbar.

c) Führen Sie die Regression durch.

```
# lineares Modell
fit <- lm(y ~ x, data=df)

# anschauen
summary(fit)
```

```
Warning in summary.lm(fit): im Wesentlichen ein perfekter Fit: summary kann
unzuverlässig sein



Call:
lm(formula = y ~ x, data = df)

Residuals:
       Min          1Q      Median          3Q         Max
-3.675e-15 -8.783e-16  5.168e-16  9.646e-16  1.944e-15

Coefficients:
             Estimate Std. Error   t value Pr(>|t|)
(Intercept) 2.000e+00  1.049e-15 1.906e+15   <2e-16 ***
```

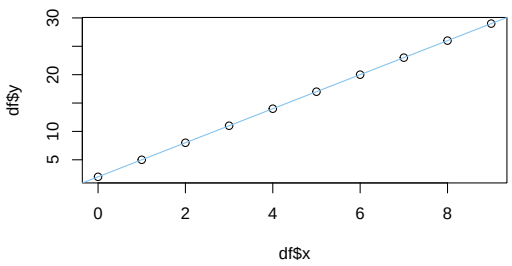
```
x           3.000e+00  1.965e-16 1.527e+16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

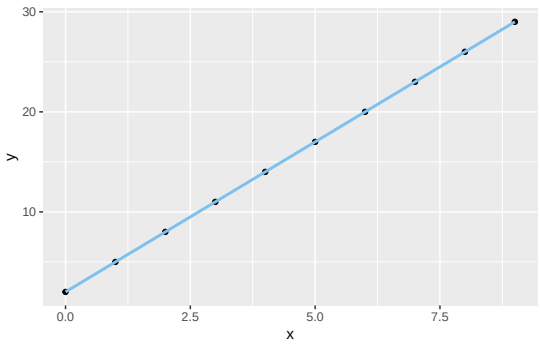
Residual standard error: 1.785e-15 on 8 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:      1
F-statistic: 2.33e+32 on 1 and 8 DF,  p-value: < 2.2e-16
```

d) Fügen Sie die Regressionsfunktion `y erklärt durch x` dem Plot hinzu.

```
# plot()
plot(df$x, df$y)
# Regressionsgerade
abline(lm(y~x, data=df), col="skyblue2")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point() +
  geom_smooth(method="lm", color="skyblue2")
```

```
`geom_smooth()` using formula = 'y ~ x'
```

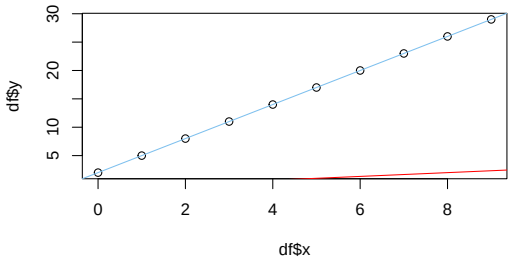


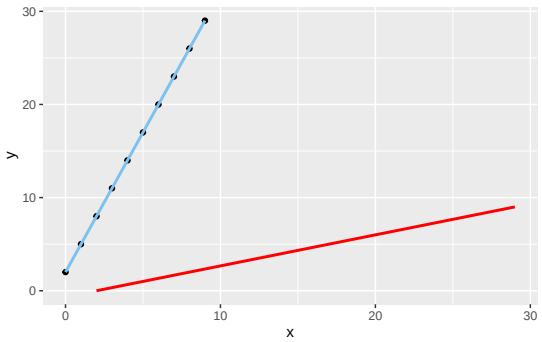


e) Fügen Sie die Regressionsfunktion `x erklärt durch y` ebenfalls dem Plot hinzu, aber in roter Farbe.

```
# plot()
plot(df$x, df$y)
# Regressionsgeraden
abline(lm(y~x, data=df), col="skyblue2")
abline(lm(x~y, data=df), col="red")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point() +
  geom_smooth(method="lm", color="skyblue2") +
  geom_smooth(aes(x=y, y=x), method="lm", color="red")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```





Achten Sie darauf, die Variablen nicht zu vertauschen!

f) Wie große sind die Residuen?

```
fit <- lm(y ~ x, data=df)

# Residuen
fit$residuals
```

```
            1             2             3             4             5
-3.675227e-15  4.362024e-16  1.944285e-15  1.385502e-15  1.048764e-15
            6             7             8             9            10
 7.120252e-16  5.973314e-16 -1.293719e-15  3.679437e-16 -1.523107e-15
```

Die Residuen sind sehr klein. Die Regressionsgerade scheint sehr gut zu passen.

4.2.17 Lösung zur Aufgabe 4.1.3.2

a) Erstellen Sie ein Datenframe mit den Variablen `Lernen` und `Durchgefallen`.

```
# erzeuge Datenframe
df <- data.frame(Lernen = c( 3.5, 0.6, 2.8, 2.5, 2.6, 3.9, 1.5, 0.7, 3.6, 3.7,
                             2.2, 3.3, 1.7, 1.1, 2.0, 3.5, 2.1, 1.8, 1.1, 0.7,
                             1.3, 3.1, 2.3, 3.2, 0.9, 1.7, 0.2, 2.9, 1.0, 2.3),
                 Durchgefallen = c( 1, 5, 1, 3, 1, 0, 3, 3, 1, 1,
                                    2, 0, 3, 3, 3, 0, 2, 2, 4, 4,
                                    4, 0, 2, 2, 4, 2, 5, 1, 3, 2))
```

b) Erzeugen Sie eine Kreuztabelle der Variablen `Lernen` und `Durchgefallen`.

```
# entweder
table(df$Lernen, df$Durchgefallen)
```

```
     0 1 2 3 4 5
  0.2 0 0 0 0 0 1
  0.6 0 0 0 0 0 1
  0.7 0 0 0 1 1 0
  0.9 0 0 0 0 1 0
  1   0 0 0 1 0 0
  1.1 0 0 0 1 1 0
  1.3 0 0 0 0 1 0
  1.5 0 0 0 1 0 0
  1.7 0 0 1 1 0 0
  1.8 0 0 1 0 0 0
  2   0 0 0 1 0 0
  2.1 0 0 1 0 0 0
  2.2 0 0 1 0 0 0
  2.3 0 0 2 0 0 0
  2.5 0 0 0 1 0 0
  2.6 0 1 0 0 0 0
  2.8 0 1 0 0 0 0
  2.9 0 1 0 0 0 0
  3.1 1 0 0 0 0 0
  3.2 0 0 1 0 0 0
  3.3 1 0 0 0 0 0
  3.5 1 1 0 0 0 0
  3.6 0 1 0 0 0 0
  3.7 0 1 0 0 0 0
  3.9 1 0 0 0 0 0
```

```
# oder
xtabs(~ Lernen + Durchgefallen, data=df)
```

```
      Durchgefallen
Lernen 0 1 2 3 4 5
   0.2 0 0 0 0 0 1
   0.6 0 0 0 0 0 1
   0.7 0 0 0 1 1 0
   0.9 0 0 0 0 1 0
   1   0 0 0 1 0 0
   1.1 0 0 0 1 1 0
   1.3 0 0 0 0 1 0
   1.5 0 0 0 1 0 0
   1.7 0 0 1 1 0 0
   1.8 0 0 1 0 0 0
   2   0 0 0 1 0 0
   2.1 0 0 1 0 0 0
   2.2 0 0 1 0 0 0
   2.3 0 0 2 0 0 0
   2.5 0 0 0 1 0 0
   2.6 0 1 0 0 0 0
   2.8 0 1 0 0 0 0
   2.9 0 1 0 0 0 0
   3.1 1 0 0 0 0 0
   3.2 0 0 1 0 0 0
   3.3 1 0 0 0 0 0
   3.5 1 1 0 0 0 0
   3.6 0 1 0 0 0 0
   3.7 0 1 0 0 0 0
   3.9 1 0 0 0 0 0
```

c) Führen Sie eine lineare Regression `Durchgefallen erklärt durch Lernen` durch und plotten Sie Ihr Ergebnis.

```
# lineare Regression
fit <- lm(Durchgefallen ~ Lernen , data=df)
summary(fit)
```

```
Call:
lm(formula = Durchgefallen ~ Lernen, data = df)

Residuals:
```

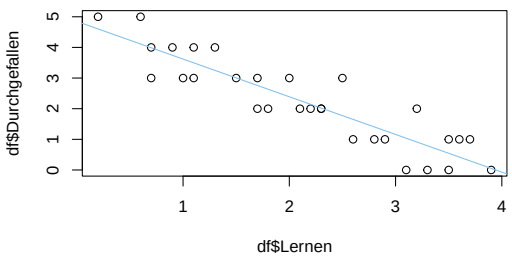
```
     Min       1Q   Median       3Q      Max
-1.03614 -0.53214 -0.02013  0.49187  1.22587

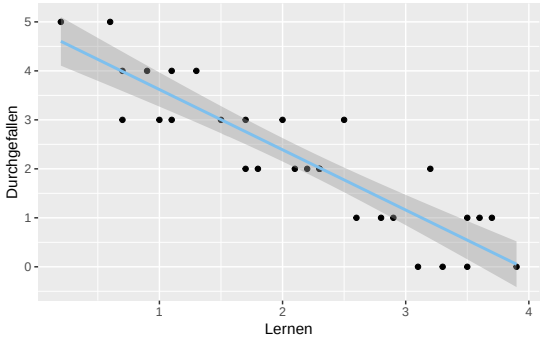
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.8491     0.2622   18.49  < 2e-16 ***
Lernen       -1.2300     0.1106  -11.12  8.7e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6359 on 28 degrees of freedom
Multiple R-squared:  0.8155,    Adjusted R-squared:  0.8089
F-statistic: 123.8 on 1 and 28 DF,  p-value: 8.7e-12
```

```
# plot()
plot(df$Lernen, df$Durchgefallen)
# Regressionsgerade
abline(lm(Durchgefallen ~ Lernen, data=df), col="skyblue2")
# ggplot()
ggplot(df, aes(x=Lernen, y=Durchgefallen)) +
  geom_point() +
  geom_smooth(method="lm", color="skyblue2")
```

```
`geom_smooth()` using formula = 'y ~ x'
```





d) Wie lauten die Regressionskoeffizient des Modells, und wie ist er zu interpretieren?

```
# Koeffizienten anzeigen
fit$coefficients
```

```
(Intercept)      Lernen
   4.849127   -1.229997
```

Der Regressionskoeffizient für `Lernen` beträgt -1.2299972. Das bedeutet, dass mit ungefähr jeder Stunde Lernen ein Kurs weniger nicht bestanden wird.

e) Ist das soeben erstellte Modell *besser* als das in Abschnitt 4.2.16 berechnete? Vergleichen Sie zur Beantwortung die Residuen beider Modelle.

```
# aktuelles Modell
# Residuen anschauen
fit$residuals
```

```
          1           2           3           4           5           6
 0.45586279  0.88887097 -0.40513523  1.22586561 -0.65113467 -0.05213834
          7           8           9          10          11          12
-0.00413156 -0.98812931  0.57886251  0.70186223 -0.14313354 -0.79013664
         13          14          15          16          17          18
 0.24186787 -0.49613044  0.61086702 -0.54413721 -0.26613326 -0.63513241
         19          20          21          22          23          24
 0.50386956  0.01187069  0.74986900 -1.03613608 -0.02013382  1.08686364
         25          26          27          28          29          30
 0.25787013 -0.75813213  0.39687210 -0.28213552 -0.61913015 -0.02013382
```

```
# Modell aus anderer Aufgabe
df2 <- data.frame(x = c( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
                  y = c( 2, 5, 8, 11, 14, 17, 20, 23, 26, 29))

fit2 <- lm(y~x, data=df2)
fit2$residuals
```

```
           1            2            3            4            5
-3.675227e-15  4.362024e-16  1.944285e-15  1.385502e-15  1.048764e-15
           6            7            8            9           10
 7.120252e-16  5.973314e-16 -1.293719e-15  3.679437e-16 -1.523107e-15
```

Im aktuellen Modell sind die Residuen größer als im vorherigen Modell. Somit ist das vorherige Modell besser.

f) Berechnen Sie den linearen Bestimmungskoeffizient und den Korrelationskoeffizient. Ist das lineare Modell ein gutes Modell, um die Beziehung zwischen den gescheiterten Prüfungen und den täglichen Studienzeiten zu erklären? Wie viel Prozent der Variabilität der durchgefallenen Prüfungen wird durch das lineare Modell erklärt?

```
# aktuelles Modell
lernen <- summary(fit)
# R^2 anschauen
lernen$r.squared
```

```
[1] 0.8154995
```

```
# Korrelationskoeffizient
cor.test(df$Lernen, df$Durchgefallen)
```

```
    Pearson's product-moment correlation

data:  df$Lernen and df$Durchgefallen
t = -11.125, df = 28, p-value = 8.7e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9532031 -0.8045264
sample estimates:
       cor
-0.9030501
```

Das Bestimmtheitsmaß R^2 beträgt 0.8154995. Somit können 81.55% des Rauschens im aktuellen Modell erklärt werden.
Der Korrelationskoeffizient von -0.9030501 ist nahe an -1. Dies spricht für einen starken negativen Zusammenhang.

g) Benutzen Sie das lineare Modell, um die Anzahl an durchgefallenen Prüfungen für einen Studenten zu bestimmen, der 3 Stunden Lernzeit investiert hat. Wie glaubwürdig ist die Vorhersage?

```
# aktuelles Modell
predict(fit, list(Lernen=3))
```

```
       1
1.159136
```

Wenn der Student 3 Stunden lernt, wird er wahrscheinlich "nur" durch 1 Kurs durchfallen.

h) Wie viele Stunden Lernzeit wird benötigt, um alle Kurse zu bestehen?

```
# neues Modell
fit <- lm(Lernen ~ Durchgefallen, data = df)
# Wieviel lernen für Durchgefallen=0?
predict(fit, list(Durchgefallen=0))
```

```
       1
3.607387
```

Wenn der Student 3 Stunden lernt, wird er wahrscheinlich "nur" durch 1 Kurs durchfallen.

4.2.18 Lösung zur Aufgabe 4.1.3.3

a) Erstellen Sie ein Datenframe mit den Variablen `Minuten` und `Alkohol`.

```
# erzeuge Datenframe
df <- data.frame(Alkohol = c(1.6, 1.7, 1.5, 1.1, 0.7, 0.2, 2.1),
                 Minuten = c(30, 60, 90, 120, 150, 180, 210))
```

b) Bestimmen Sie den passenden Korrelationskoeffizienten. Werden die Daten ausreichend gut durch das Modell beschrieben?

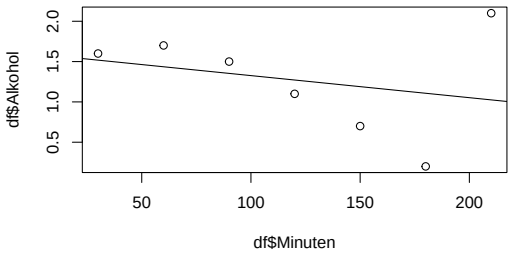
```
# Korrelation
cor(df$Minuten, df$Alkohol)
```

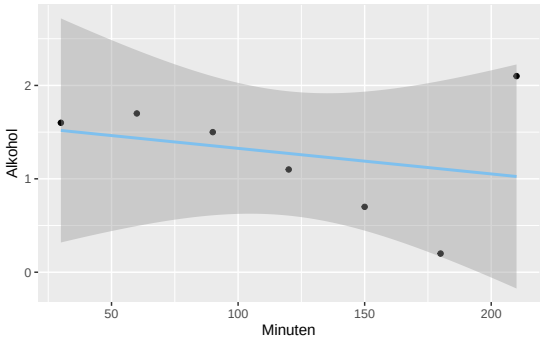
```
[1] -0.2730367
```

Der Korrelationskoeffizient ist eher gering. Das spricht für keinen starken Zusammenhang.

c) Plotten Sie das lineare Regressionsmodell `Alkohol erklärt durch Minuten`. Gibt es Punkte mit großen Residuen? Wenn ja, entfernen Sie diese und führen die Berechnungen erneut durch. Hat sich der Korrelationskoeffizient verbessert?

```
# plot()
plot(df$Minuten, df$Alkohol)
abline(lm(Alkohol ~ Minuten, data=df))
# ggplot()
ggplot(df, aes(x=Minuten, y=Alkohol)) +
  geom_point()+
  geom_smooth(method="lm", color="skyblue2")
```





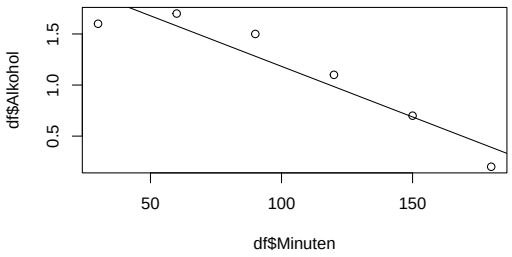
Der letzte Wert ist ein deutlicher Ausreißer, wahrscheinlich ein Tippfehler bei der Dateneingabe.

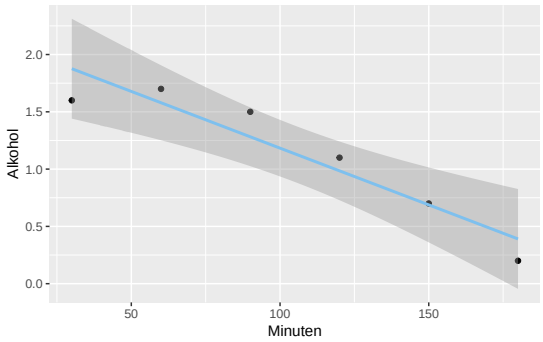
```
# entferne letzten Wert
df <- df[-7,]
# Korrelation
cor(df$Minuten, df$Alkohol)
```

```
[1] -0.944155
```

Der Korrelationskoeffizient ist nun sehr nah an -1. Das spricht für einen starken Zusammenhang.

```
# Modell
# plot()
plot(df$Minuten, df$Alkohol)
abline(lm(Alkohol ~ Minuten, data=df))
# ggplot()
ggplot(df, aes(x=Minuten, y=Alkohol)) +
  geom_point()+
  geom_smooth(method="lm", color="skyblue2")
```





d) Mit welcher Geschwindigkeit wird der Alkohol pro Minute verstoffwechselt?

```
# Koeffizienten
fit <- lm(Alkohol ~ Minuten, data=df)
fit$coefficient
```

```
 (Intercept)      Minuten
2.173333333 -0.009904762
```

Der Alkoholspiegel sinkt pro Minute um -0.0099048 g/l.

e) Wenn es gesetzlich erlaubt wäre, mit einem Blutalkoholwert von $0,3$ g/l Auto zu fahren, wie lange muss die Person warten, nachdem sie 1 Liter Weingetrunken hat, um wieder fahrtüchtig zu sein? Wie zuverlässig ist diese Vorhersage?

```
# Koeffizienten
fit <- lm(Minuten ~ Alkohol, data=df)
predict(fit, list(Alkohol=0.3))
```

```
  1
180
```

Der Alkoholspiegel wird nach 180 Minuten auf $0,3$ g/l fallen.

4.2.19 Lösung zur Aufgabe 4.1.3.4

a) Laden Sie den Datensatz `age.height` in Ihre R-Session.

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/age.height.RData"))
```

b) Berechnen Sie die Regressionsgerade `Größe erklärt durch Alter`. Ist das lineare Modell geeignet, den Zusammenhang zwischen `Alter` und `Körpergröße` zu erklären?

```
# Regression
fit <- lm(height ~ age, data=age.height)
summary(fit)
```

```
Call:
lm(formula = height ~ age, data = age.height)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9137 -0.1018  0.0449  0.1644  0.4202

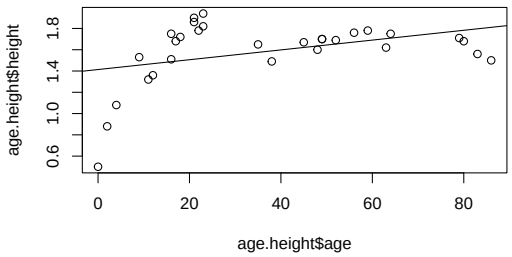
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.413724   0.091080  15.522 2.77e-15 ***
age         0.004612   0.002036   2.265   0.0314 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

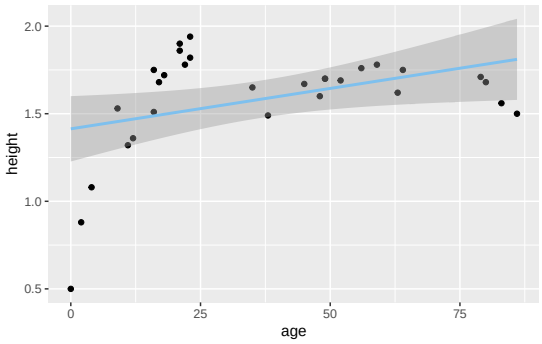
Residual standard error: 0.2852 on 28 degrees of freedom
Multiple R-squared:  0.1549,    Adjusted R-squared:  0.1247
F-statistic: 5.131 on 1 and 28 DF,  p-value: 0.03142
```

R^2 ist eher gering, es können nur 15.49% des Rauschens mit dem Modell erklärt werden.

c) Erstellen Sie eine Punktwolke inklusive der Regressionsgeraden. Ab welchem Alter ändert sich die Punktetendenz?

```
# plot()
plot(age.height$age, age.height$height)
abline(fit)
# ggplot()
ggplot(age.height, aes(x=age, y=height)) +
  geom_point()+
  geom_smooth(method="lm", color="skyblue2")
```





Ab etwa 20 Jahren ändert sich die Punktetendenz.

d) Erstellen Sie eine Gruppierungsvariable, welche `Alter` in einen ordinalen Faktor mit den Ausprägungen "`jünger als 20`" und "`20 und älter`" einteilt.

```
# klassieren
age.height$ageK <- cut(age.height$age,
                       breaks = c(0,20, Inf),
                       right=FALSE,
                       labels = c("jünger als 20",
                                  "20 und älter"))
# anschauen

head(age.height)
```

age	height	ageK
18	1.72	jünger als 20
21	1.90	20 und älter
45	1.67	20 und älter
59	1.78	20 und älter
21	1.86	20 und älter
22	1.78	20 und älter

e) Führen Sie die lineare Regressionsanalyse für beide Gruppen erneut durch. In welcher Gruppe wird der Zusammenhang zwischen `Alter` und `Körpergröße` am besten erklärt?

```
# Gruppen
df1 <- subset(age.height, ageK=="jünger als 20")
# Regression
fit1 <- lm(height ~ age, data=df1)
summary(fit1)
```

```
Call:
lm(formula = height ~ age, data = df1)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22746 -0.05601 -0.03485  0.08416  0.28351

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.727459   0.094487   7.699 5.75e-05 ***
age         0.057671   0.007738   7.453 7.25e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1525 on 8 degrees of freedom
Multiple R-squared:  0.8741,    Adjusted R-squared:  0.8584
F-statistic: 55.54 on 1 and 8 DF,  p-value: 7.245e-05
```

Das Bestimmtheitsmaß in der Gruppe “jünger als 20” liegt bei 0.8741033, d.h. es werden 87.41% des Rauschens erklärt.

```
# Gruppen
df2 <- subset(age.height, ageK=="20 und älter")
# Regression
fit2 <- lm(height ~ age, data=df2)
summary(fit2)
```

```
Call:
lm(formula = height ~ age, data = df2)

Residuals:
     Min       1Q   Median       3Q      Max
-0.25783 -0.04614 -0.01064  0.07793  0.14155

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.876084   0.056839  33.007  < 2e-16 ***
age         -0.003375   0.001051  -3.213  0.00483 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

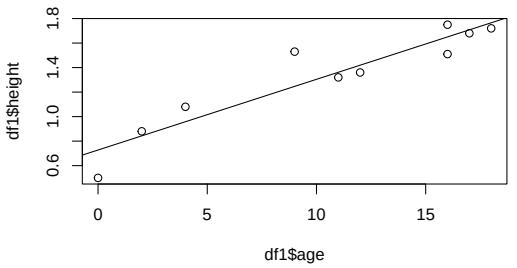
Residual standard error: 0.09931 on 18 degrees of freedom
Multiple R-squared:  0.3644,    Adjusted R-squared:  0.3291
```

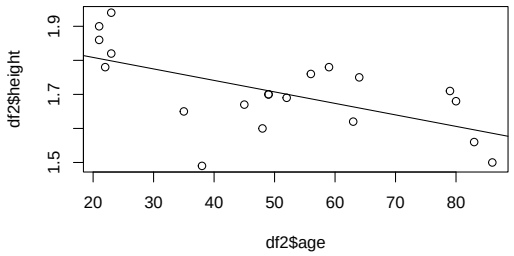
```
F-statistic: 10.32 on 1 and 18 DF,  p-value: 0.004827
```

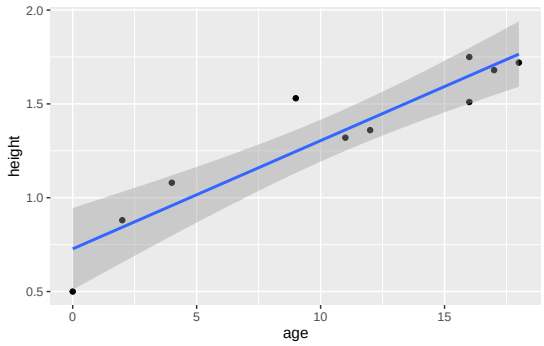
Das Bestimmtheitsmaß in der Gruppe “20 und älter” liegt bei 0.3644171, d.h. es werden 36.44% des Rauschens erklärt.

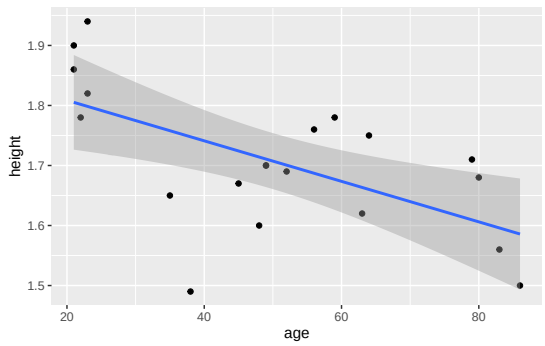
f) Plotten Sie Ihre Modelle

```
# < 20 Jahre
plot(df1$age, df1$height)
abline(fit1)
# >= 20 Jahre
plot(df2$age, df2$height)
abline(fit2)
## ggplot()
# < 20 Jahre
ggplot(df1, aes(x=age, y=height)) +
  geom_point() +
  geom_smooth(method="lm")
# >= 20 Jahre
ggplot(df2, aes(x=age, y=height)) +
  geom_point() +
  geom_smooth(method="lm")
```









g) Welche Körpergröße sagt Ihr Modell für eine 14jährige Person vorher, und welche für eine 38jährige Person?

```
# 14 jährige Person
predict(fit1, list(age=14))
```

```
       1
1.534847
```

```
# 38 jährige Person
predict(fit2, list(age=38))
```

```
       1
1.747827
```

4.2.20 Lösung zur Aufgabe 4.1.3.5

```
df <- data.frame(Jahr=c(1:5),
                 Wirksamkeit=c(96, 84, 70, 58, 52)
)
```

a) Führen Sie eine lineare Regression `Wirksamkeit erklärt durch Jahr` durch und plotten Sie Ihr Ergebnis.

```
# Regression
fit <- lm(Wirksamkeit~Jahr, data=df)
summary(fit)
```

```

Call:
lm(formula = Wirksamkeit ~ Jahr, data = df)

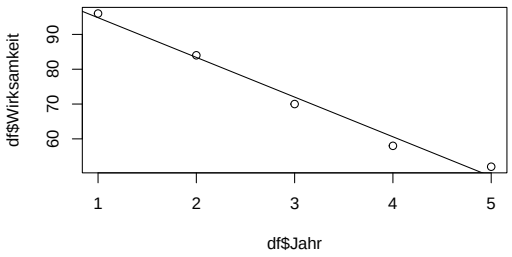
Residuals:
   1    2    3    4    5
 1.2  0.6 -2.0 -2.6  2.8

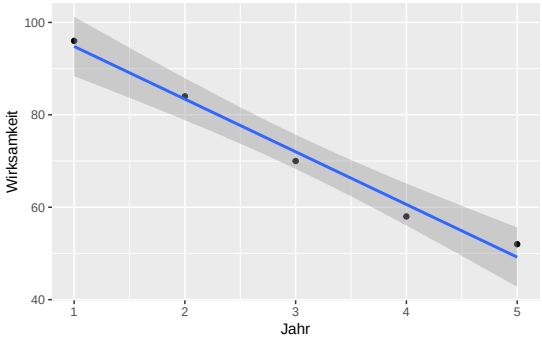
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 106.2000     2.7350   38.83 3.76e-05 ***
Jahr        -11.4000     0.8246  -13.82 0.000819 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 2.608 on 3 degrees of freedom
Multiple R-squared:  0.9845,    Adjusted R-squared:  0.9794
F-statistic: 191.1 on 1 and 3 DF,  p-value: 0.0008192
```

```r
# plot()
plot(df$Jahr, df$Wirksamkeit)
abline(fit)
# ggplot()
ggplot(df, aes(x=Jahr, y=Wirksamkeit)) +
  geom_point() +
  geom_smooth(method="lm")
```

```
`geom_smooth()` using formula = 'y ~ x'
```





b) Wie große ist der jährliche Wirksamkeitsverlust in %?

```r
# Regression
fit$coefficients
```

```
(Intercept)        Jahr
      106.2       -11.4
```

Der Wirksamkeitsverlust beträgt 11.4% pro Jahr.

c) Nach wie vielen Jahren ist die Wirksamkeit bei 80%, und nach wie vielen bei 0%? Sind beide Werte gleich zuverlässig?

```
# anderes Modell
fit2 <- lm(Jahr ~ Wirksamkeit, data=df)
# 80% und 0%
predict(fit2, list(Wirksamkeit=c(80,0)))
```

```
       1        2
2.309091 9.218182
```

Nach 2.31 Jahren ist die Wirksamkeit bei 80%, nach 9.22 Jahren bei 0%.

4.2.21 Lösung zur Aufgabe 4.1.3.6

```
df <- data.frame(Dosis=c(2,2, 2,2,2,2,
                         3,3, 3,3,3,3,
                         3, 4,4,4,4,4, 4,4),
                 Tage =c(5,5, 6,6,6,6,
                         3,3, 5,5,5,5,
                         6, 3,3,3,3,3, 5,5))
```

a) Berechnen Sie die Regressionsgerade der Heilungstage in Abhängigkeit von der Dosis.

```
# Regression
fit <- lm(Tage~Dosis, data=df)
summary(fit)
```

```
Call:
lm(formula = Tage ~ Dosis, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-1.6023 -0.5560  0.3513  0.3977  1.4440

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.7413     0.7941   9.749 1.32e-08 ***
Dosis        -1.0463     0.2517  -4.156 0.000593 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.9059 on 18 degrees of freedom
Multiple R-squared:  0.4897,    Adjusted R-squared:  0.4614
F-statistic: 17.28 on 1 and 18 DF,  p-value: 0.000593
```

b) Berechnen Sie den Regressionskoeffizienten der Heilungstage in Abhängigkeit von der Dosis und interpretieren Sie ihn.

```
# Regression
fit$coefficients
```

```
(Intercept)       Dosis
   7.741313   -1.046332
```

Mit jeder Dosiserhöhung um 1 verkürzt sich die Heilungsdauer um ca. 1 Tag.

c) Berechnen Sie den Korrelationskoeffizienten und interpretieren Sie ihn.

```
# Regression
cor(df$Dosis, df$Tage)
```

```
[1] -0.69981
```

Der Korrelationskoeffizient ist größer als 0,5. Es liegt ein mittelstarker Zusammenhang vor.

d) Bestimmen Sie die erwartete Zeit, die für die Heilung mit einer Dosis von 5 mg benötigt wird. Ist diese Vorhersage zuverlässig? Begründen Sie die Antwort.

```
# Vorhersage
predict(fit, list(Dosis=5))
```

```
       1
2.509653
```

e) Welche Dosis muss angewendet werden, um in 4 Tagen zu heilen? Ist diese Vorhersage zuverlässig? Begründen Sie die Antwort.

```
# neues Modell
fit2 <- lm(Dosis~Tage, data=df)
# Vorhersage
predict(fit2, list(Tage=4))
```

```
       1
3.307427
```

4.2.22 Lösung zur Aufgabe 4.1.3.7

a) Laden Sie den Datensatz `heigths.weights.students` in Ihre R-Session.

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/heights.weights.students.RData"))
```

b) Führen Sie eine lineare Regression `Gewicht erklärt durch Größe` durch und plotten Sie Ihr Modell.

```
# Regression
fit <- lm(weight ~ height, data=heights.weights.students)
summary(fit)
```

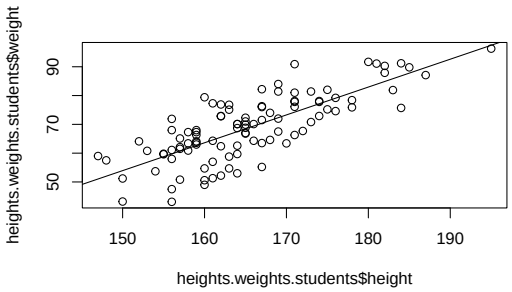
```
Call:
lm(formula = weight ~ height, data = heights.weights.students)

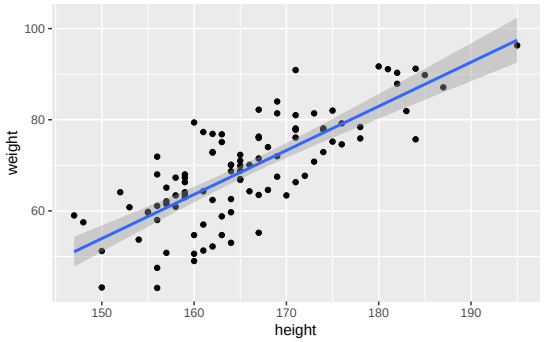
Residuals:
     Min       1Q   Median       3Q      Max
-16.6372  -4.8272   0.9568   4.8008  16.6542

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -91.15252   13.28198  -6.863 6.16e-10 ***
height        0.96724    0.08009  12.077  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.356 on 98 degrees of freedom
Multiple R-squared:  0.5981,    Adjusted R-squared:  0.594
F-statistic: 145.9 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
# plot()
plot(heights.weights.students$height, heights.weights.students$weight)
abline(fit)
# ggplot()
ggplot(heights.weights.students, aes(x=height, y=weight)) +
  geom_point() +
  geom_smooth(method="lm")
```



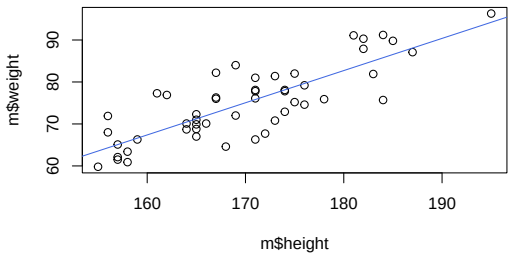


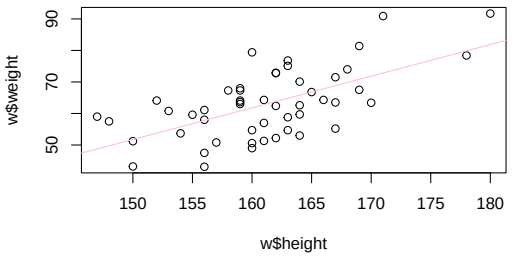
c) Erstellen Sie eine Punktwolke inklusive Regressionsgeraden jeweils für Männer und Frauen getrennt.

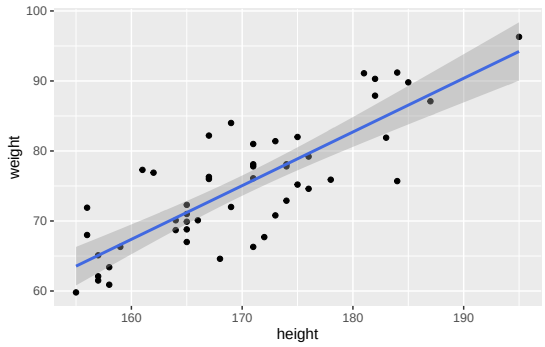
```r
m <- subset(heights.weights.students, sex=="male")
w <- subset(heights.weights.students, sex=="female")

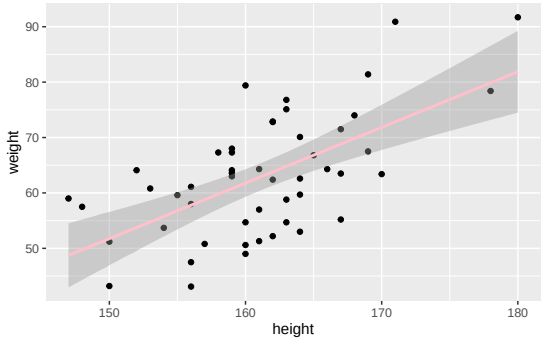
fit1 <- lm(weight ~ height, data=m)
fit2 <- lm(weight ~ height, data=w)

## plot()
# männlich
plot(m$height, m$weight)
abline(fit1, col="royalblue")
# weiblich
plot(w$height, w$weight)
abline(fit2, col="pink")
## ggplot()
# männlich
ggplot(m, aes(x=height, y=weight))  +
  geom_point() +
  geom_smooth(method="lm", color="royalblue")
# weiblich
ggplot(w, aes(x=height, y=weight))  +
  geom_point() +
  geom_smooth(method="lm", color="pink")
```









d) Berechnen Sie die Bestimmtheitskoeffizienten (R^2) für beide Modelle. Welches Modell erklärt besser die Beziehung zwischen Gewicht und Größe, das der Männer oder das der Frauen? Begründen Sie die Antwort.

```
# Männer
summary(fit1)$r.squared
```

```
[1] 0.6699418
```

```
# Frauen
summary(fit2)$r.squared
```

```
[1] 0.3828876
```

Das Modell der Männer erklärt 0.67% der Streuung, und das der Frauen "nur" 0.38%. Somit ist das Modell für Männer *besser* als das der Frauen.

e) Was ist das zu erwartende Gewicht für einen Mann mit 170cm Körpergröße? Und für eine Frau der selben Größe?

```
# Männer
predict(fit1, list(height=170))
```

```
     1
75.048
```

```
# Frauen
predict(fit2, list(height=170))
```

```
      1
71.8338
```

4.2.23 Lösung zur Aufgabe 4.1.3.8

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/neonates.RData"))
```

a) Erstellen Sie eine Kreuztabelle vom APGAR-Wert nach 1 Minute und dem Rauchverhalten der Mütter während der Schwangerschaft. Welche Schlüsse lassen sich ziehen?

```
# entweder
table(neonates$smoke, neonates$apgar1)
```

```
       2  3  4  5  6  7  8  9
  No   1  6 18 50 77 40 23  5
  Yes  3 15 20 31 20  6  5  0
```

```
# oder
xtabs(~ smoke + apgar1, data=neonates)
```

```
      apgar1
smoke  2  3  4  5  6  7  8  9
  No   1  6 18 50 77 40 23  5
  Yes  3 15 20 31 20  6  5  0
```

Kinder von Frauen, die nicht während der Schangerschaft rauchen, haben höhere APGAR1-Werte als Kinder von Raucherinnen.

b) Erstellen Sie eine Kreuztabelle vom APGAR-Wert nach 1 Minute und der Alterskategorie der Mütter. Welche Schlüsse lassen sich ziehen?

```
# entweder
table(neonates$age, neonates$apgar1)
```

```
                  2  3  4  5  6  7  8  9
  greater than 20  2 10 22 53 69 34 24  4
  less than 20     2 11 16 28 28 12  4  1
```

```
# oder
xtabs(~ age + apgar1, data=neonates)
```

```
                apgar1
age                2  3  4  5  6  7  8  9
  greater than 20  2 10 22 53 69 34 24  4
  less than 20     2 11 16 28 28 12  4  1
```

Kinder von Frauen, die älter als 20 Jahre sind, haben höhere APGAR1-Werte als Kinder von jüngeren Müttern.

c) Führen Sie eine lineare Regression für `Geburtsgewicht erklärt durch Anzahl täglich gerauchter Zigaretten` durch. Gibt es einen starken linearen Zusammenhang?

```
# Regression
fit <- lm(weight ~ cigarettes, data=neonates)
summary(fit)
```

```
Call:
lm(formula = weight ~ cigarettes, data = neonates)

Residuals:
     Min       1Q   Median       3Q      Max
-0.98756 -0.16656 -0.00649  0.18769  1.03544

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.146557   0.018604  169.13   <2e-16 ***
cigarettes  -0.031067   0.002512  -12.37   <2e-16 ***
---
```

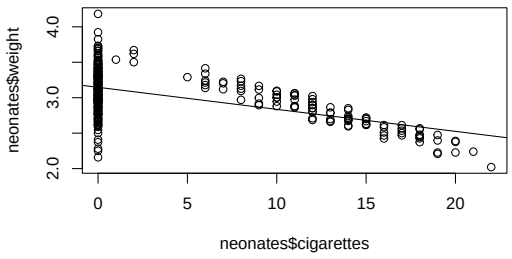
```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

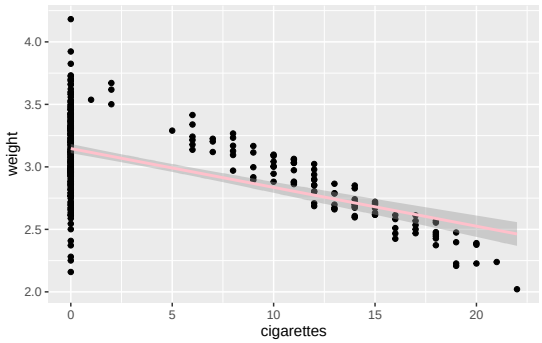
Residual standard error: 0.2832 on 318 degrees of freedom
Multiple R-squared:  0.3248,    Adjusted R-squared:  0.3227
F-statistic:   153 on 1 and 318 DF,  p-value: < 2.2e-16
```

Der Zusammenhang ist eher gering.

d) Plotten Sie Ihre Regression. Passt die Regressionsgerade gut zur Punktwolke?

```
# plot()
plot(neonates$cigarettes, neonates$weight)
abline(fit)
# ggplot()
ggplot(neonates, aes(x=cigarettes, y=weight))  +
  geom_point() +
  geom_smooth(method="lm", color="pink")
```





Der Zusammenhang wird durch die Nichtraucherinnen (0 Zigaretten) verzerrt.

e) Wiederholen Sie die Regression, aber nutzen Sie dieses Mal nur Daten von Raucherinnen. Ist dieses Modell besser oder schlechter als das vorherige? Wieviel Gewicht verliert ein Neugeborenes nach diesem Modell pro täglich gerauchter Zigarette?

```
# Subset erzeugen
smoke <- subset(neonates, smoke=="Yes")
# Regression
fit2 <- lm(weight ~ cigarettes, data=smoke)
summary(fit2)
```

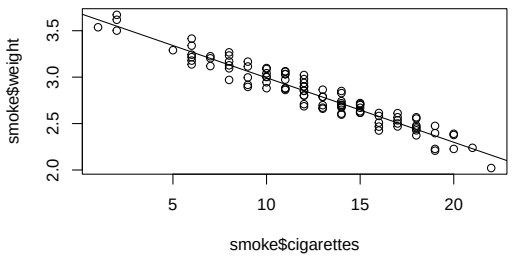
```
Call:
lm(formula = weight ~ cigarettes, data = smoke)
```

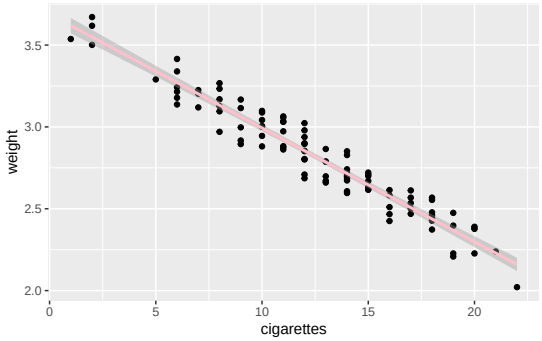
```
Residuals:
      Min        1Q    Median        3Q       Max
-0.168338 -0.057531  0.002855  0.068180  0.168662

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.687879   0.025542  144.38   <2e-16 ***
cigarettes  -0.069462   0.001928  -36.03   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08735 on 98 degrees of freedom
Multiple R-squared:  0.9298,    Adjusted R-squared:  0.9291
F-statistic:  1298 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
# plot()
plot(smoke$cigarettes, smoke$weight)
abline(fit2)
# ggplot()
ggplot(smoke, aes(x=cigarettes, y=weight))  +
  geom_point() +
  geom_smooth(method="lm", color="pink")
```





Der Zusammenhang ist nun sehr stark.

f) Welches Geburtsgewicht sagt dieses Modell für ein Neugeborenes vorher, dessen Mutter 5 Zigaretten täglich während der Schwangerschaft geraucht hat? Wieviel für eine Mutter, die 30 Zigaretten täglich raucht. Wie zuverlässich sind diese Ergebnisse?

```
# Vorhersage
predict(fit2, list(cigarettes=c(5, 30)))
```

```
       1        2
3.340570 1.604026
```

g) Ändert sich der lineare Zusammenhang, wenn die Daten nach Altersgruppen getrennt untersucht werden?

```
# Subset
s1 <- subset(smoke, age=="greater than 20")
s2 <- subset(smoke, age=="less than 20")

# neue Modelle
fit1 <- lm(weight ~ cigarettes, data=s1)
fit1 <- lm(weight ~ cigarettes, data=s2)

# vergleichen
summary(fit1)
```

```

Call:
lm(formula = weight ~ cigarettes, data = s2)

Residuals:
      Min        1Q    Median        3Q       Max
-0.151567 -0.049334 -0.001749  0.062936  0.127103

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.65752    0.05636   64.90  < 2e-16 ***
cigarettes  -0.06833    0.00375  -18.22 6.33e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08286 on 20 degrees of freedom
Multiple R-squared:  0.9432,    Adjusted R-squared:  0.9404
F-statistic: 332.1 on 1 and 20 DF,  p-value: 6.333e-14
```

```r
summary(fit2)
```

```
Call:
lm(formula = weight ~ cigarettes, data = smoke)

Residuals:
      Min        1Q    Median        3Q       Max
-0.168338 -0.057531  0.002855  0.068180  0.168662

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.687879   0.025542  144.38   <2e-16 ***
cigarettes  -0.069462   0.001928  -36.03   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08735 on 98 degrees of freedom
Multiple R-squared:  0.9298,    Adjusted R-squared:  0.9291
F-statistic:  1298 on 1 and 98 DF,  p-value: < 2.2e-16
```

Der Zusammenhang bleibt unabhängig von der Altersgruppe bestehen.

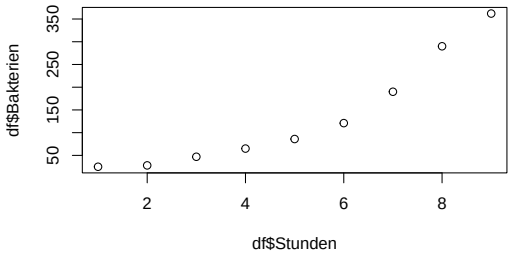
4.2.24 Lösung zur Aufgabe 4.1.4.1

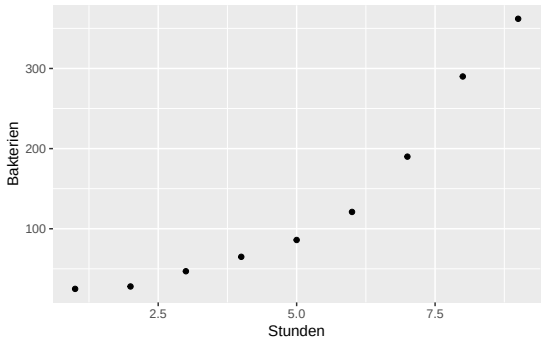
a) Erstellen Sie ein Datenframe mit den Variablen `Stunden` und `Bakterien`.

```r
# Daten erzeugen
df <- data.frame(Stunden=c(1:9),
                 Bakterien=c(25, 28, 47, 65, 86, 121, 190, 290, 362))
```

b) Erzeugen Sie ein Scatterplot. Welche Regression würden Sie auf Grundlage des Plots vorschlagen?

```r
# plot()
plot(df$Stunden, df$Bakterien)
# ggplot()
ggplot(df, aes(x=Stunden, y=Bakterien)) +
  geom_point()
```





Die Punktwolken sprechen für einen exponentiellen Anstieg.

c) Berechnen Sie die quadratischen und exponentiellen Modelle für die Bakterienvermehrung über die Zeit.

```
# quadratisch
q <- lm(Bakterien ~ Stunden + I(Stunden^2), data=df)
summary(q)
```

```
Call:
lm(formula = Bakterien ~ Stunden + I(Stunden^2), data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-16.617  -8.297  -1.430   9.916  15.442

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   53.2381    15.9862   3.330   0.0158 *
Stunden      -26.7420     7.3403  -3.643   0.0108 *
I(Stunden^2)   6.8009     0.7159   9.500 7.75e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.56 on 6 degrees of freedom
Multiple R-squared:  0.9919,    Adjusted R-squared:  0.9892
F-statistic: 368.8 on 2 and 6 DF,  p-value: 5.254e-07
```

```
# exponentiell
e <- lm(log(Bakterien) ~ Stunden, data=df)
summary(e)
```

```
Call:
lm(formula = log(Bakterien) ~ Stunden, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-0.12676 -0.06057  0.01145  0.03920  0.11190

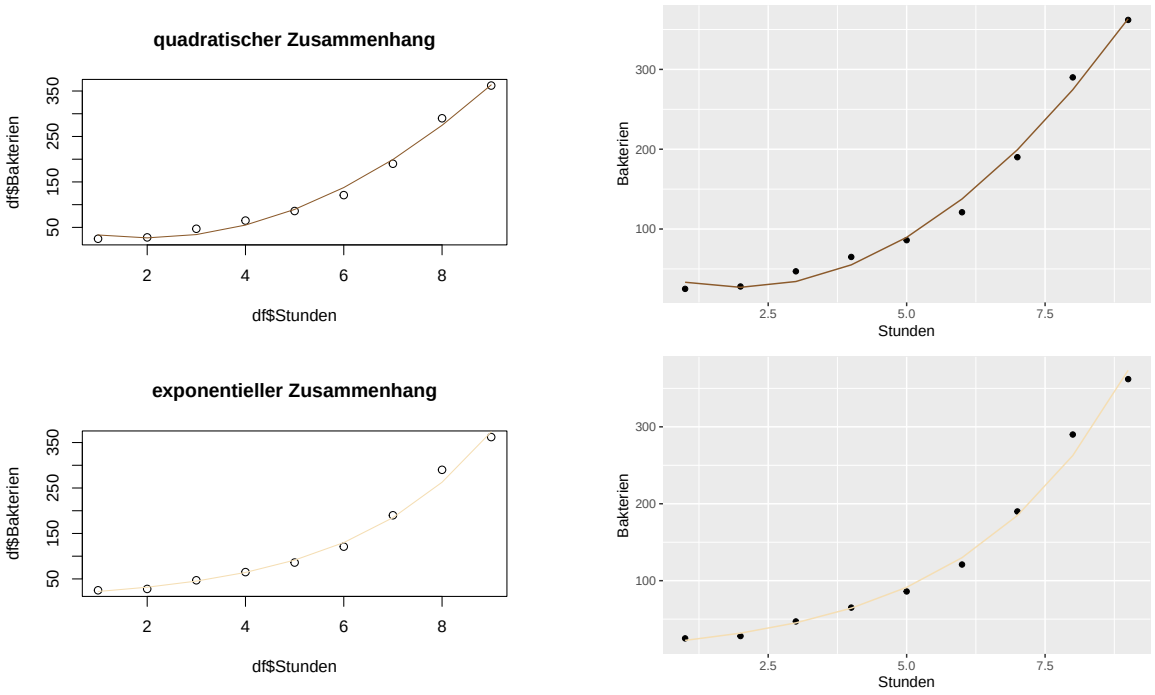
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.75498    0.06174   44.62 7.41e-10 ***
Stunden      0.35199    0.01097   32.08 7.39e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08498 on 7 degrees of freedom
Multiple R-squared:  0.9932,    Adjusted R-squared:  0.9923
F-statistic:  1029 on 1 and 7 DF,  p-value: 7.389e-09
```

d) Plotten Sie das bessere Modell in die Punktwolke.

```r
# Vorbereitung quadratisch
vorhersageQ <- predict(q, list(Stunden=df$Stunden))
# plot() quadratisch
plot(df$Stunden, df$Bakterien,
     main="quadratischer Zusammenhang")
lines(df$Stunden, vorhersageQ, col="tan4")
# ggplot() quadratisch
ggplot(df, aes(x=Stunden, y=Bakterien)) +
  geom_point() +
  geom_line(aes(y=vorhersageQ), col="tan4")
# Vorbereitung exponentiell
vorhersageE <- exp(predict(e, list(Stunden=df$Stunden)))

# plot() exponentiell
plot(df$Stunden, df$Bakterien,
     main="exponentieller Zusammenhang")
lines(df$Stunden, vorhersageE, col="wheat")
# ggplot() exponentiell
ggplot(df, aes(x=Stunden, y=Bakterien)) +
  geom_point() +
  geom_line(aes(y=vorhersageE), col="wheat")
```



e) Wie viele Bakterien werden nach dem besten Modell 3 Stunden nach Anlegen der Kultur vorhanden sein? Und nach 10 Stunden? Sind diese Vorhersagen zuverlässig?

```
# Vorhersage
exp(predict(e, list(Stunden=c(3, 10))))
```

```
       1        2
45.19322 531.05241
```

Nach 3 Stunden können wir 46 Bakterien erwarten, nach 10 Stunden 532.

f) Machen Sie eine möglichst zuverlässige Vorhersage über die Zeit, die benötigt wird, um 100 Bakterien in der Kultur zu haben.

```
# neues Modell
df$BakterienLog <- log(df$Bakterien)
fit <- lm(BakterienLog ~ Stunden, data=df)
a <- exp(fit$coefficients[1])
b <- fit$coefficients[2]
# Vorhersage für 100 Bakterien
(log(100) - log(a)) / b
```

```
(Intercept)
   5.256395
```

Nach ca 5.3 Stunden sind 100 Bakterien zu erwarten.

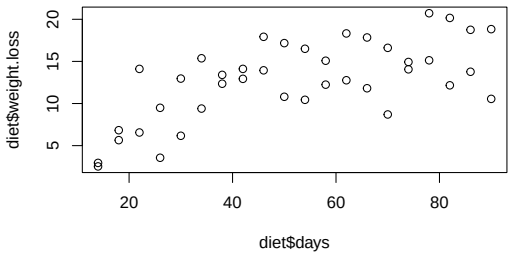
4.2.25 Lösung zur Aufgabe 4.1.4.2

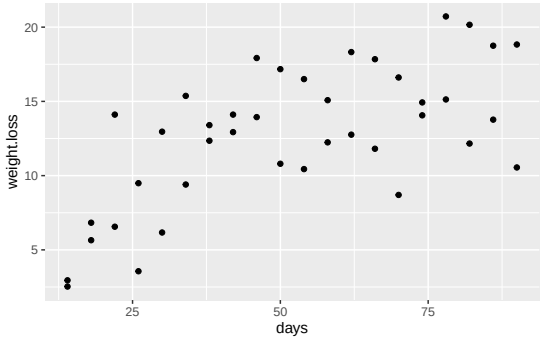
a) Laden Sie den Datensatz `diet` in Ihre R-Session.

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/diet.RData"))
```

b) Erstellen Sie eine Punktwolke. Welche Art von Modell erklärt auf Grundlage der Punktwolke den Gewichtsverlust pro Diättag besser?

```
# plot()
plot(diet$days, diet$weight.loss)
# ggplot()
ggplot(diet, aes(x=days, y=weight.loss)) +
  geom_point()
```





c) Berechnen Sie das Regressionsmodell, welches den Gewichtsverlust mit der Anzahl an Diättagen am besten (im Vergleich zu anderen) erklären kann. Wird das Modell zuverlässige Vorhersagen machen?

```
# Vergleiche Bestimmheitsmaße verschiedener Modelle

# quadratisches Modell
q <- lm(weight.loss ~ days + I(days^2), data=diet)

# exponentielles Modell
e <- lm (log(weight.loss) ~ days, data=diet)

# logarithmisches Modell
l <- lm(weight.loss ~ log(days), data=diet)

# sigmoidales Modell
s <- lm(log(weight.loss) ~ I(1/days), data=diet)

result <- data.frame(Modell = c("quadratisch", "exponentiell",
                                "logarithmisch", "sigmoidal"),
                     R.square = c(summary(q)$r.square,
                                  summary(e)$r.square,
                                  summary(l)$r.square,
                                  summary(s)$r.square))
# Anzeigen
result[order(result$R.square, decreasing = TRUE),]
```

	Modell	R.square
4	sigmoidal	0.6662170
1	quadratisch	0.5397848
3	logarithmisch	0.5254856
2	exponentiell	0.4308936

```
## oder mit der compare.lm()-Funktion
jgsbook::compare.lm(diet$weight.loss, diet$days)
```

	Modell	R.square
6	sigmoidal	0.6662170
7	potenz	0.5684490
3	kubisch	0.5584355
2	quadratisch	0.5397848
5	logarithmisch	0.5254856
1	linear	0.4356390
4	exponentiell	0.4308936

Das sigmoidale Modell kann die Daten am besten erklären, da in diesem Modell das Bestimmtheitsmaß am größten ist.

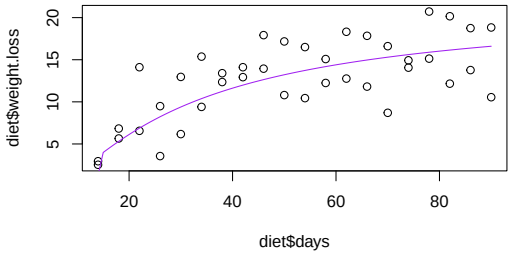
d) Plotten Sie Ihr Modell.

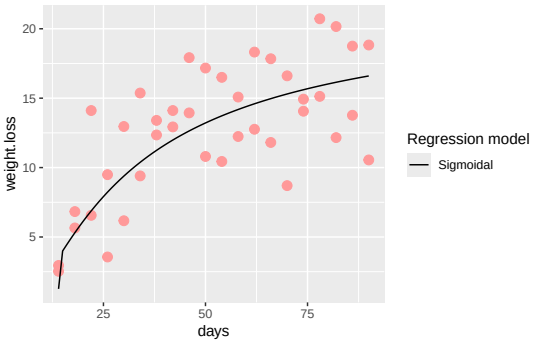
```
# sigmoidales Modell
s <- lm(log(weight.loss) ~ I(1/days), data=diet)

# vorhersage vorbereiten
tage <- seq(min(diet$days), max(diet$days))
# alle "tage" vorhersagen
vorhersage <- predict(s, list(days=tage))
vorhersage[-1]=exp(vorhersage[-1])

# plot()
plot(diet$days, diet$weight.loss)
lines(tage, vorhersage, col="purple")
# ggplot()
# in Datenframe für ggplot speichern
helper <- data.frame(tage, vorhersage)

ggplot(diet, aes(x=days, y=weight.loss)) +
  geom_point(color="#FF9999", size=3) +
    # Legend
    scale_linetype("Regression model") +
    # Sigmoidal model
    geom_line(data=helper, aes(x=tage, y=vorhersage, linetype="Sigmoidal"))
```





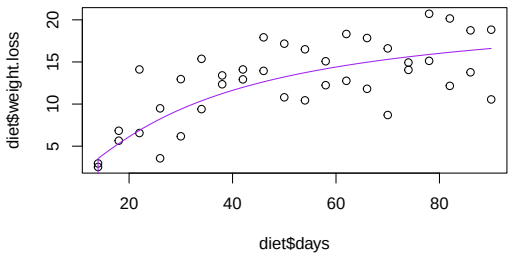
Auch die Vorhersagewerte für die Idealkurve können mittels `compare.lm()` und dem Parameter `predict=TRUE` erzeugt werden.

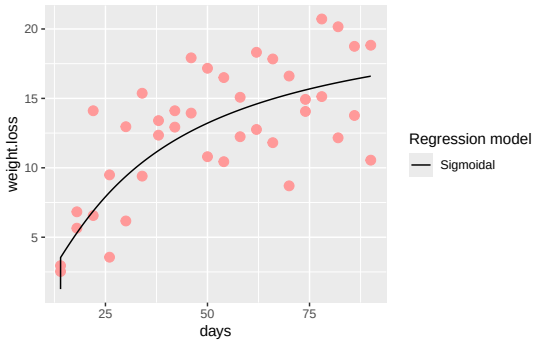
```
help <- jgsbook::compare.lm(diet$weight.loss, diet$days,
                            predict=TRUE)

# anschauen
head(help)
```

pred.x	line	quad	cube	expo	loga	sigm	power	logistic
14.00	7.645786	4.766305	3.526750	6.714088	5.166988	1.262199	4.966500	4.261378
14.01	7.647113	4.770032	3.532726	6.715040	5.171506	3.537806	4.969006	4.264669
14.02	7.648440	4.773757	3.538700	6.715992	5.176020	3.542430	4.971512	4.267962
14.03	7.649767	4.777482	3.544671	6.716944	5.180531	3.547052	4.974018	4.271256
14.04	7.651094	4.781207	3.550641	6.717896	5.185040	3.551674	4.976523	4.274552
14.05	7.652421	4.784931	3.556608	6.718848	5.189544	3.556296	4.979027	4.277850

```
# plot()
# plot()
plot(diet$days, diet$weight.loss)
lines(help$pred.x, help$sigm, col="purple")
# ggplot()
ggplot(diet, aes(x=days, y=weight.loss)) +
  geom_point(color="#FF9999", size=3) +
    # Legend
    scale_linetype("Regression model") +
    # Sigmoidal model
    geom_line(data=help, aes(x=pred.x, y=sigm, linetype="Sigmoidal"))
```





e) Berechnen Sie das Regressionsmodell, das den Gewichtsverlust anhand der Tage der Diät für die Gruppe der Personen, die sich nicht regelmäßig körperlich betätigen, am besten erklärt.

```
# Subset bilden
df1 <- subset(diet, exercise=="no")

# Vergleiche Bestimmheitsmaße verschiedener Modelle

# quadratisches Modell
q1 <- lm(weight.loss ~ days + I(days^2), data=df1)

# exponentielles Modell
e1 <- lm (log(weight.loss) ~ days, data=df1)

# logarithmisches Modell
l1 <- lm(weight.loss ~ log(days), data=df1)

# sigmoidales Modell
s1 <- lm(log(weight.loss) ~ I(1/days), data=df1)

result1 <- data.frame(Modell = c("quadratisch", "exponentiell",
                                 "logarithmisch", "sigmoidal"),
                      R.square = c(summary(q1)$r.square,
                                   summary(e1)$r.square,
                                   summary(l1)$r.square,
                                   summary(s1)$r.square))
result1[order(result1$R.square, decreasing = TRUE),]
```

	Modell	R.square
4	sigmoidal	0.7401212
1	quadratisch	0.7100610
3	logarithmisch	0.6494521
2	exponentiell	0.5222832

```
  # oder mittels compare.lm()
  jgsbook::compare.lm(df1$weight.loss, df1$days)
```

	Modell	R.square
6	sigmoidal	0.7401212
3	kubisch	0.7151929
2	quadratisch	0.7100610
7	potenz	0.6700051
5	logarithmisch	0.6494521
1	linear	0.5286338
4	exponentiell	0.5222832

Das sigmoidale Modell liefert wieder die beste Erklärung der Daten.

f) Wiederholen Sie die Analyse für die Gruppe, die sich regelmäßig körperlich betätigt.

```
# Subset bilden
df2 <- subset(diet, exercise=="yes")

# Vergleiche Bestimmheitsmaße verschiedener Modelle

# quadratisches Modell
q2 <- lm(weight.loss ~ days + I(days^2), data=df2)

# exponentielles Modell
e2 <- lm (log(weight.loss) ~ days, data=df2)

# logarithmisches Modell
l2 <- lm(weight.loss ~ log(days), data=df2)

# sigmoidales Modell
s2 <- lm(log(weight.loss) ~ I(1/days), data=df2)

result2 <- data.frame(Modell = c("quadratisch", "exponentiell",
                                 "logarithmisch", "sigmoidal"),
                      R.square = c(summary(q2)$r.square,
                                   summary(e2)$r.square,
                                   summary(l2)$r.square,
                                   summary(s2)$r.square))
result2[order(result2$R.square, decreasing = TRUE),]
```

	Modell	R.square
4	sigmoidal	0.8305013
3	logarithmisch	0.7885173
1	quadratisch	0.7791671
2	exponentiell	0.4945564

```
# oder mittels compare.lm()
jgsbook::compare.lm(df2$weight.loss, df2$days)
```

	Modell	R.square
3	kubisch	0.8326179
6	sigmoidal	0.8305013
5	logarithmisch	0.7885173
2	quadratisch	0.7791671
7	potenz	0.6704843
1	linear	0.6623502
4	exponentiell	0.4945564

Das kubische Modell liefert hier die beste Erklärung der Daten.

g) Benutzen Sie die erstellen Modelle, um den Gewichtsverlust nach 30 und nach 100 Tagen Diät für Personen, die sich körperlich betätigen, und für solche, die dies nicht tun, vorherzusagen. Sind diese Vorhersagen zuverlässig?

```
# Vorhersage Kein Sport
exp(predict(s1, list(days=c(30,100))))
```

```
        1         2
 7.808926 13.806339
```

```
# Vorhersage Sport
exp(predict(s2, list(days=c(30,100))))
```

```
       1        2
11.28578 21.13143
```

4.2.26 Lösung zur Aufgabe 4.1.4.3

```
df <- data.frame(Stunden = c(2:8),
                 Konzentration = c(25, 36, 48, 64, 86, 114, 168))
```

a) Benutzen Sie ein exponentielles Modell, um die Konzentration nach 10 Stunden vorherzusagen. Ist die Vorhersage zuverlässig?

```
# exponentielles Modell
fit <- lm(log(Konzentration) ~ Stunden, data=df)
summary(fit)
```

```

Call:
lm(formula = log(Konzentration) ~ Stunden, data = df)

Residuals:
       1        2        3        4        5        6        7
-0.023147  0.034218  0.014623 -0.004972 -0.016786 -0.042212  0.038276

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.627468   0.033673   78.03 6.55e-09 ***
Stunden     0.307277   0.006253   49.14 6.59e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.03309 on 5 degrees of freedom
Multiple R-squared:  0.9979,    Adjusted R-squared:  0.9975
F-statistic:  2415 on 1 and 5 DF,  p-value: 6.594e-08
```

```
# Vorhersage 10 Stunden
exp(predict(fit, list(Stunden=10)))
```

```
     1
298.94
```

Nach 10 Stunden beträgt die Konzentration 298,94 mg/dl.
Das Bestimmtheitsmaß R^2 des Modells ist mit 0,9979 sehr groß. Die Daten werden sehr gut durch das Modell erklärt.

b) Benutzen Sie ein logarithmisches Modell um zu bestimmen, nach wie vielen Stunden eine Konzentration von 100 mg/dl erreicht sein wird.

```
# exponentielles Modell
fit <- lm(log(Konzentration) ~ Stunden, data=df)

# coefficienten
a <- fit$coefficient[1]
b <- fit$coefficient[2]

# Vorhersage 100 mg/dl
( log(100) - a ) / b
```

```
(Intercept)
   6.436209
```

Nach 6,436209 Stunden sind 100 mg/dl erreicht.

4.2.27 Lösung zur Aufgabe 4.1.5.1

a) Lassen Sie in R eine beliebige Poker-Spielkarte ziehen.

```
# lade Kartenspiele
load(url("https://www.produnis.de/R/data/cards.RData"))

# ziehe Karte
sample(poker, 1)
```

```
[1] Pik 10
52 Levels: Kreuz 2 < Karo 2 < Herz 2 < Pik 2 < Kreuz 3 < Karo 3 < ... < Pik As
```

```
# alternativ kann das 'probs'-Paket verwendet werden
karten <- probs::cards(makespace=TRUE)
probs::sim(karten, ntrials=1)
```

rank	suit
6	Club

b) Lassen Sie in R 2 Münzen werfen.

```
# lade Datensatz
coin <- c("Kopf", "Zahl")

# wirf 2 Münzen
sample(coin, 2, replace=TRUE)
```

```
[1] "Kopf" "Zahl"
```

```
# alternativ kann das 'probs'-Paket verwendet werden
coin <- probs::tosscoin(2, makespace=TRUE)
probs::sim(coin, ntrials=1)
```

toss1	toss2
H	T

c) Lassen Sie in R 2 Würfel werfen.

```
# lade Datensatz
würfel <- c(1:6)

# wirf 2 Münzen
sample(würfel, 2, replace=TRUE)
```

```
[1] 2 4
```

```
# alternativ kann das 'probs'-Paket verwendet werden
würfel <- probs::rolldie(2, makespace=TRUE)
probs::sim(würfel, ntrials=1)
```

X1	X2
5	4

4.2.28 Lösung zur Aufgabe 4.1.5.2

a) Wiederholen Sie die Zufallsexperimente und lassen Sie R 10 mal, 100 mal 1.000 mal und 1.000.000 mal zwei Münzen werfen. Erstellen Sie je eine relative Häufigkeitstabelle der Ergebnisse. Wie sind die Tabellen zu bewerten?

```
# erzeuge Wahrscheinlichkeitsraum
münzen <- probs::tosscoin(2, makespace=TRUE)

# werfe 10mal 2 Münzen
versuch <- probs::sim(münzen, ntrials=10)
# berechne Wahrscheinlichkeitsverteilung
probs::empirical(versuch)
```

toss1	toss2	probs
T	H	0.3
H	T	0.4
T	T	0.3

```
# werfe 100mal 2 Münzen
versuch <- probs::sim(münzen, ntrials=100)
# berechne Wahrscheinlichkeitsverteilung
probs::empirical(versuch)
```

toss1	toss2	probs
H	H	0.30
T	H	0.20
H	T	0.28
T	T	0.22

```
# werfe 1.000mal 2 Münzen
versuch <- probs::sim(münzen, ntrials=1000)
# berechne Wahrscheinlichkeitsverteilung
probs::empirical(versuch)
```

toss1	toss2	probs
H	H	0.252
T	H	0.248
H	T	0.249
T	T	0.251

```
# werfe 1.000.000mal 2 Münzen
versuch <- probs::sim(münzen, ntrials=1000000)
# berechne Wahrscheinlichkeitsverteilung
probs::empirical(versuch)
```

toss1	toss2	probs
H	H	0.249949
T	H	0.249696
H	T	0.250325
T	T	0.250030

Mit zunehmender Wiederholung nähern sich die Wahrscheinlichkeitsverteilungen den relativen Häufigkeiten an.

b) Welche theoretischen Wahrscheinlichkeiten haben die möglichen Wurfergebnisse? Stimmen diese mit den beobachteten Ergebnissen überein?

Je häufiger das Zufallsexperiment wiederholt wird, desto mehr nähern sich die beobachteten Wahrscheinlichkeiten den theoretischen Wahrscheinlichkeiten an.

4.2.29 Lösung zur Aufgabe 4.1.5.3

a) Ziehen Sie zufällig 3 Boxen, ohne zurücklegen.

```
boxen <- c("A", "A", "A", "B", "B", "C")

# ziehe 3 Boxen ohne Zurücklegen
sample(boxen, 3, replace=FALSE)
```

```
[1] "B" "A" "B"
```

Ziehen Sie zufällig 3 Boxen, diesmal mit zurücklegen.

```
boxen <- c("A", "A", "A", "B", "B", "C")

# ziehe 3 Boxen ohne Zurücklegen
sample(boxen, 3, replace=TRUE)
```

```
[1] "B" "A" "B"
```

4.2.30 Lösung zur Aufgabe 4.1.5.4

a) Erstellen Sie ein Datenframe mit den Variablen `Windpocken`, `Masern`, `Röteln` und `Häufigkeit` und übertragen Sie die Daten.

```
df <- tribble(
  ~Windpocken, ~Masern, ~Röteln, ~Häufigkeit,
  "No",        "No",    "No",    2654,
  "No",        "No",    "Yes",   1436,
  "No",        "Yes",   "No",    1682,
  "No",        "Yes",   "Yes",   668,
  "Yes",       "No",    "No",    1747,
  "Yes",       "No",    "Yes",   476,
  "Yes",       "Yes",   "No",    876,
  "Yes",       "Yes",   "Yes",   265
)
```

b) Erstellen Sie den Wahrscheinlichkeitsraum der Lebenszeitprävalenz.

```
# Wahrscheinlichkeitsraum
wr <- probs::probspace(df[,-4], probs=df$Häufigkeit/sum(df$Häufigkeit))
wr
```

Windpocken	Masern	Röteln	probs
No	No	No	0.2707058
No	No	Yes	0.1464708
No	Yes	No	0.1715626
No	Yes	Yes	0.0681355
Yes	No	No	0.1781926
Yes	No	Yes	0.0485516
Yes	Yes	No	0.0893513
Yes	Yes	Yes	0.0270298

```
# Erstelle daraus die marginale Verteilung
probs::marginal(wr)
```

Windpocken	Masern	Röteln	probs
No	No	No	0.2707058
Yes	No	No	0.1781926
No	Yes	No	0.1715626
Yes	Yes	No	0.0893513
No	No	Yes	0.1464708
Yes	No	Yes	0.0485516
No	Yes	Yes	0.0681355
Yes	Yes	Yes	0.0270298

c) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Windpocken hatte?

```
# Wahrscheinlichkeitsraum
wr <- probs::probspace(df[,-4], probs=df$Häufigkeit/sum(df$Häufigkeit))
# berechne Wahrscheinlichkeit
probs::Prob(wr, event=Windpocken=="Yes")
```

```
[1] 0.3431253
```

Die Wahrscheinlichkeit beträgt 34.31%.

d) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Windpocken oder Masern hatte?

```
# berechne Wahrscheinlichkeit
probs::Prob(wr, event=Windpocken=="Yes" | Röteln=="Yes")
```

```
[1] 0.5577315
```

Die Wahrscheinlichkeit beträgt 55.77%.

e) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person Masern und Röteln hatte?

```
# berechne Wahrscheinlichkeit
probs::Prob(wr, event=Masern=="Yes" & Röteln=="Yes")
```

```
[1] 0.09516524
```

Die Wahrscheinlichkeit beträgt 9.52%.

f) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person, die bereits an Masern erkrankte, nun an Windpocken erkrankt?

```
# berechne Wahrscheinlichkeit
probs::Prob(wr, event=Windpocken=="Yes", given= Masern=="Yes")
```

```
[1] 0.3268404
```

Die Wahrscheinlichkeit beträgt 32.68%.

g) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig gezogene Person, die keine Masern und keine Röteln hatte, an Windpocken erkrankt?

```
# berechne Wahrscheinlichkeit
probs::Prob(wr, event=Masern=="No" & Röteln=="No",
                given= Windpocken=="Yes")
```

```
[1] 0.5193222
```

Die Wahrscheinlichkeit beträgt 51.93%.

4.2.31 Lösung zur Aufgabe 4.1.5.5

a) Erstellen Sie ein Datenframe mit den Variablen `Schwanger`, `Testergebnis` und `Häufigkeit`.

```
df <- tribble(
  ~Schwanger, ~Test, ~Häufigkeit,
    "Nein", "-",   3876,
    "Nein", "+",  47,
    "Ja", "-",  12,
    "Ja", "+", 131
)
```

b) Erstellen Sie den Wahrscheinlichkeitsraum.

```
# Wahrscheinlichkeitsraum
wr <- probs::probspace(df[,-3], probs=df$Häufigkeit/sum(df$Häufigkeit))
wr
```

Schwanger	Test	probs
Nein	-	0.9532710
Nein	+	0.0115593
Ja	-	0.0029513
Ja	+	0.0322184

```
# Erstelle daraus die marginale Verteilung
probs::marginal(wr)
```

Schwanger	Test	probs
Ja	-	0.0029513
Nein	-	0.9532710
Ja	+	0.0322184
Nein	+	0.0115593

c) Berechnen Sie die Prävalenz der Schwangerschaften.

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Schwanger=="Ja")
```

```
[1] 0.0351697
```

Die Prävalenz liegt bei 3.52%.

d) Wie groß ist die Wahrscheinlichkeit, ein positives Testergebnis zu ziehen?

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Test=="+")
```

```
[1] 0.04377767
```

Die Wahrscheinlichkeit liegt bei 4.38%.

e) Bestimmen Sie die Sensitivität des Tests

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Test=="+", given= Schwanger=="Ja")
```

```
[1] 0.9160839
```

Die Sensitivität liegt bei 91.61%.

f) Bestimmen Sie die Spezifität des Tests

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Test=="-", given= Schwanger=="Nein")
```

```
[1] 0.9880194
```

Die Spezifität liegt bei 98.8%.

g) Bestimmen Sie den positiv prädiktiven Wert des Tests

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Schwanger=="Ja", given=Test=="+")
```

```
[1] 0.7359551
```

Der positiv prädiktive Wert liegt bei 73.6%.

h) Bestimmen Sie den negativ prädiktiven Wert des Tests

```
# Wahrscheinlichkeitsraum
probs::Prob(wr, event=Schwanger=="Nein", given=Test=="-")
```

```
[1] 0.9969136
```

Der negative prädiktive Wert liegt bei 99.69%.

Alternativ kann auch die Funktion `sens.spec()` aus dem Paket `jgsbook` verwendet werden:

```
jgsbook::sens.spec(rp=131, fp=12, rn=3876, fn=47)
```

sens	spec	ppw	npw
73.6	99.69	91.61	98.8

4.2.32 Lösung zur Aufgabe 4.1.5.6

Erstelle den Ereignisraum des Zufallsexperiments, das aus dem Werfen einer Münze, dem Werfen eines Würfels und dem Ziehen einer Karte aus einem französischen Kartenspiel besteht.

```
würfel <- 1:6
münze <- c("Kopf", "Zahl")
bild <- c(7:10 ,"B", "D", "K", "A")
farbe <- c("Kreuz", "Pik", "Karo", "Herz")

Ereignisraum <- expand.grid(Münze=münze, Bild=bild,
                            Farbe=farbe, Würfel=würfel)
head(Ereignisraum)
```

Münze	Bild	Farbe	Würfel
Kopf	7	Kreuz	1
Zahl	7	Kreuz	1
Kopf	8	Kreuz	1
Zahl	8	Kreuz	1
Kopf	9	Kreuz	1
Zahl	9	Kreuz	1

4.2.33 Lösung zur Aufgabe 4.1.5.7

```
df <- tribble(
  ~Impfung, ~Grippe, ~Häufigkeit,
   "Nein",   "Nein",   418,
   "Nein",    "Ja",    312,
   "Ja",     "Nein",   233,
   "Ja",     "Ja",    37)
```

a) Erzeugen Sie den Wahrscheinlichkeitsraum

```
wr <- probs::probspace(df[,-3],probs=df$Häufigkeit/sum(df$Häufigkeit))
wr
```

Impfung	Grippe	probs
Nein	Nein	0.418
Nein	Ja	0.312
Ja	Nein	0.233
Ja	Ja	0.037

```
# Erstelle daraus die marginale Verteilung
probs::marginal(wr)
```

Impfung	Grippe	probs
Ja	Ja	0.037
Nein	Ja	0.312
Ja	Nein	0.233
Nein	Nein	0.418

b) Wie groß ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Person geimpft ist?

```
probs::Prob(wr,event=Impfung=="Ja")
```

```
[1] 0.27
```

Die Wahrscheinlichkeit beträgt 27%.

c) Wie hoch ist die Prävalenz der Grippe?

```
probs::Prob(wr,event=Grippe=="Ja")
```

```
[1] 0.349
```

Die Prävalenz beträgt 34.9%.

d) Wie groß ist die Wahrscheinlichkeit, dass geimpfte Personen an Grippe erkranken? Ist die Impfung effektiv?

```
probs::Prob(wr,event=Grippe=="Ja", given=Impfung=="Ja")
```

```
[1] 0.137037
```

Die Wahrscheinlichkeit beträgt 13.7%.

4.2.34 Lösung zur Aufgabe 4.1.5.8

```
df <- tribble(
  ~Ebola, ~Test, ~Häufigkeit,
   "Nein",   "+",   28,
   "Nein",    "-",    97465,
   "Ja",     "+",   147,
   "Ja",     "-",   65)
```

a) Erzeugen Sie den Wahrscheinlichkeitsraum

```
wr <- probs::probspace(df[,-3],probs=df$Häufigkeit/sum(df$Häufigkeit))
wr
```

Ebola	Test	probs
Nein	+	0.0002866
Nein	-	0.9975436
Ja	+	0.0015045
Ja	-	0.0006653

```
# Erstelle daraus die marginale Verteilung
probs::marginal(wr)
```

Ebola	Test	probs
Ja	-	0.0006653
Nein	-	0.9975436
Ja	+	0.0015045
Nein	+	0.0002866

b) Berechnen Sie die Prävalenz von Ebola in der Bevölkerung.

```
probs::Prob(wr,event=Ebola=="Ja")
```

```
[1] 0.002169797
```

Die Prävalenz beträgt 0.22%.

c) Wie hoch ist die Wahrscheinlichkeit, ein negatives Testergebnis zu erhalten?

```
probs::Prob(wr,event=Test=="-")
```

```
[1] 0.9982089
```

Die Prävalenz beträgt 99.82%.

d) Berechnen Sie die Sensitivität und Spezifität des Tests.

```
# entweder
# Sensitivität
probs::Prob(wr,event=Test=="+", given=Ebola=="Ja")
```

```
[1] 0.6933962
```

```
# Spezifität
probs::Prob(wr,event=Test=="-", given=Ebola=="Nein")
```

```
[1] 0.9997128
```

```
# oder
jgsbook::sens.spec(fp=28, rn=97465, rp=147, fn=65)
```

sens	spec	ppw	npw
69.34	99.97	84	99.93

e) Kann der Test besser Erkrankte erkennen, oder Gesunde?

```
jgsbook::sens.spec(fp=28, rn=97465, rp=147, fn=65)
```

sens	spec	ppw	npw
69.34	99.97	84	99.93

Er kann besser Gesunde erkennen.

f) Wenn eine Person einen positiven Test erhält, wie hoch ist dann die Wahrscheinlichkeit, dass er tatsächlich krank ist?

```
# positiv prädiktiv
probs::Prob(wr,event=Ebola=="Ja", given=Test=="+")
```

```
[1] 0.84
```

Die Wahrscheinlichkeit liegt bei 84%.

g) Wenn eine Person einen negativen Test erhält, wie hoch ist dann die Wahrscheinlichkeit, dass er tatsächlich gesund ist?

```
# negativ prädiktiv
probs::Prob(wr,event=Ebola=="Nein", given=Test=="-")
```

```
[1] 0.9993335
```

Die Wahrscheinlichkeit liegt bei 99.93%.

4.2.35 Lösung zur Aufgabe 4.1.6.1

a) Berechnen Sie die Wahrscheinlichkeitsverteilung von `X`

```
# mögliche Ausprägungen von x
x <- 0:10
# Wahrscheinlichkeiten berechnen
w <- dbinom(x, size = 10, prob = 0.50)

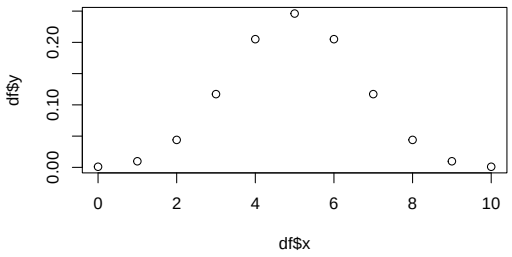
data.frame(Kopf=x, Wahrscheinlichkeit = w)
```

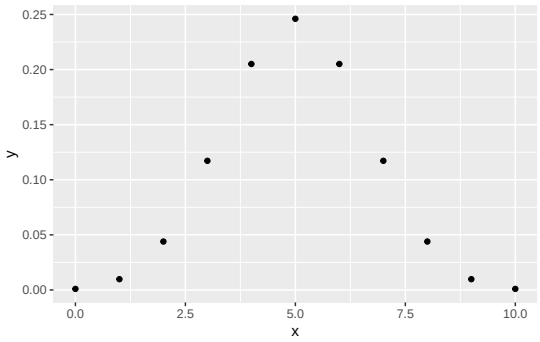
Kopf	Wahrscheinlichkeit
0	0.0009766
1	0.0097656
2	0.0439453
3	0.1171875
4	0.2050781
5	0.2460938
6	0.2050781
7	0.1171875
8	0.0439453
9	0.0097656
10	0.0009766

b) Plotten Sie die Wahrscheinlichkeitsfunktion von `X`

```
# mögliche Ausprägungen von x
x <- 0:10
df = data.frame(x, y=dbinom(x, size = 10, prob = 0.50))

# plot()
plot(df$x, df$y)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point()
```

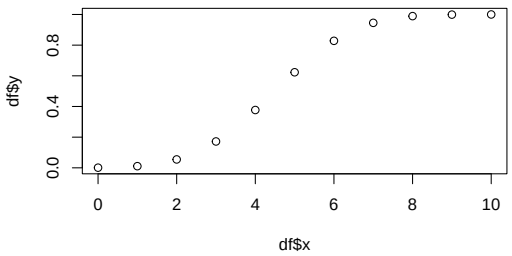


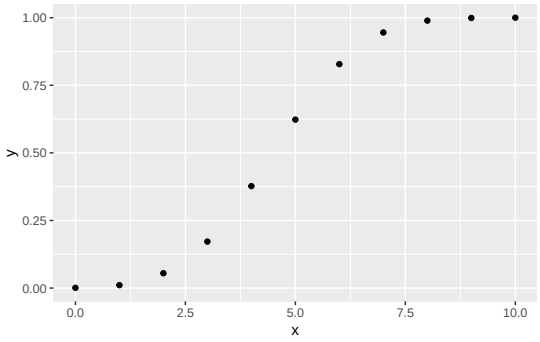


c) Plotten Sie die Dichteverteilung

```r
# mögliche Ausprägungen von x
x <- 0:10
df = data.frame(x, y=pbinom(x, size = 10, prob = 0.50))

# plot()
plot(df$x, df$y)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point()
```





d) Berechnen Sie die Wahrscheinlichkeit, 7 mal `Kopf` zu werfen.

```r
dbinom(7, size = 10, prob = 0.50)
```

```
[1] 0.1171875
```

e) Berechnen Sie die Wahrscheinlichkeit, weniger als als 4 mal `Kopf` zu werfen.

```
pbinom(4, size = 10, prob = 0.50, lower.tail=TRUE)
```

```
[1] 0.3769531
```

f) Berechnen Sie die Wahrscheinlichkeit, mehr als als 5 mal `Kopf` zu werfen.

```
pbinom(5, size = 10, prob = 0.50, lower.tail=FALSE)
```

```
[1] 0.3769531
```

g) Berechnen Sie die Wahrscheinlichkeit, 2 bis 8 mal `Kopf` zu werfen.

```
# weniger als 2mal
w2 <- pbinom(1, size = 10, prob = 0.5)
# weniger als 8mal
w8 <- pbinom(8, size = 10, prob = 0.5)

# Wahrscheinlichkeit 2 bis 8 Köpfe zu werfen
w8 - w2
```

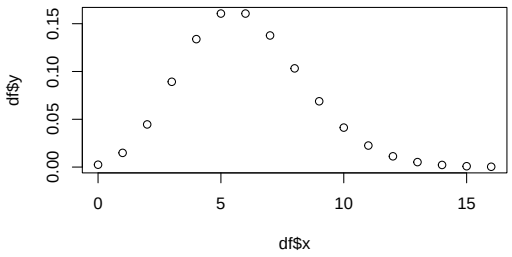
```
[1] 0.9785156
```

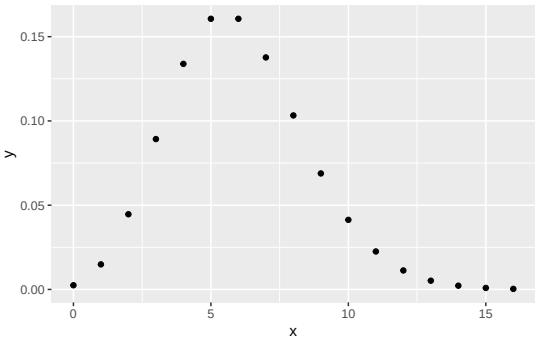
4.2.36 Lösung zur Aufgabe 4.1.6.2

a) Plotten Sie die Wahrscheinlichkeitsfunktion von `X`

```
#  x-Achse
x = 0:16
# Poisson-Werte
df = data.frame(x, y=dpois(x, lambda = 6))

# plot()
plot(df$x, df$y)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point()
```

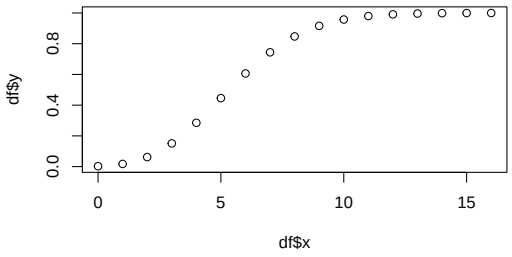


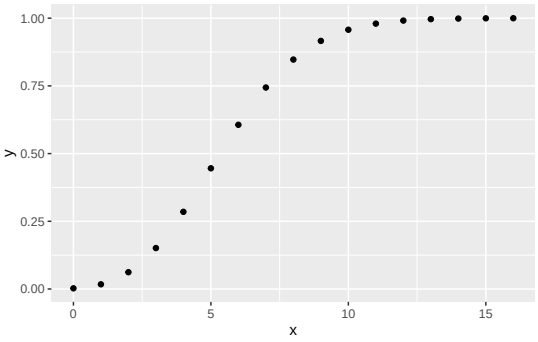


b) Plotten Sie die Verteilungsfunktion von `X`

```
#  x-Achse
x = 0:16
# Poisson-Werte
df = data.frame(x, y=ppois(x, lambda = 6))

# plot()
plot(df$x, df$y)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_point()
```





c) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag (nur) 1 Geburt stattfindet?

```
dpois(1, lambda = 6)
```

```
[1] 0.01487251
```

d) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag weniger als 6 Geburten stattfinden?

```
ppois(5, lambda = 6, lower.tail=TRUE)
```

```
[1] 0.4456796
```

e) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag 4 oder mehr Geburten stattfinden?

```
ppois(3, lambda = 6, lower.tail=FALSE)
```

```
[1] 0.8487961
```

f) Wie groß ist die Wahrscheinlicheit, dass an einem zufälligen Tag 4 bis 8 Geburten stattfinden?

```
# entweder
sum(dpois(4:8, lambda=6))
```

```
[1] 0.6960336
```

```
# oder
ppois(8, lambda=6) - ppois(3, lambda=6)
```

```
[1] 0.6960336
```

g) Wie groß ist die Wahrscheinlicheit, dass in einer Woche zwischen 30 und 40 Geburten stattfinden?

```
# lambda für eine Woche = 7* 6 = 42
ppois(40, lambda=42) - ppois(29, lambda=42)
```

```
[1] 0.3959028
```

4.2.37 Lösung zur Aufgabe 4.1.6.3

a) berechnen Sie die Wahrscheinlichkeitsverteilung des binomialen Modells $B(30, 0.1)$.

```
dbinom(c(0,1,2,3,4,5,6,7,8,9,10), size = 30, prob = 0.1)
```

```
 [1] 0.0423911583 0.1413038609 0.2276562204 0.2360879322 0.1770659492
 [6] 0.1023047706 0.0473633197 0.0180431694 0.0057637902 0.0015654739
[11] 0.0003652772
```

b) berechnen Sie die Wahrscheinlichkeitsverteilung des Poissonmodells $P(3)$ und vergleichen Sie es mit dem binomialen Modell $B(30, 0.1)$.

```
result <- data.frame(binomial= dbinom(c(0,1,2,3,4,5,6,7,8,9,10),
                                      size = 30, prob = 0.1),
                     poisson = dpois(c(0,1,2,3,4,5,6,7,8,9,10),
                                     lambda = 3))
head(result)
```

binomial	poisson
0.0423912	0.0497871
0.1413039	0.1493612
0.2276562	0.2240418
0.2360879	0.2240418
0.1770659	0.1680314
0.1023048	0.1008188

c) berechnen Sie die Wahrscheinlichkeitsverteilung des binomialen Modells $B(100, 0.3)$ und vergleichen Sie es es mit dem Modell $P(3)$. Sind diese Modelle ähnlicher als die vorherigen?

```
result <- data.frame(binomial= dbinom(c(0,1,2,3,4,5,6,7,8,9,10),
                                      size = 100, prob = 0.3),
                     poisson = dpois(c(0,1,2,3,4,5,6,7,8,9,10),
                                     lambda = 3))
head(result)
```

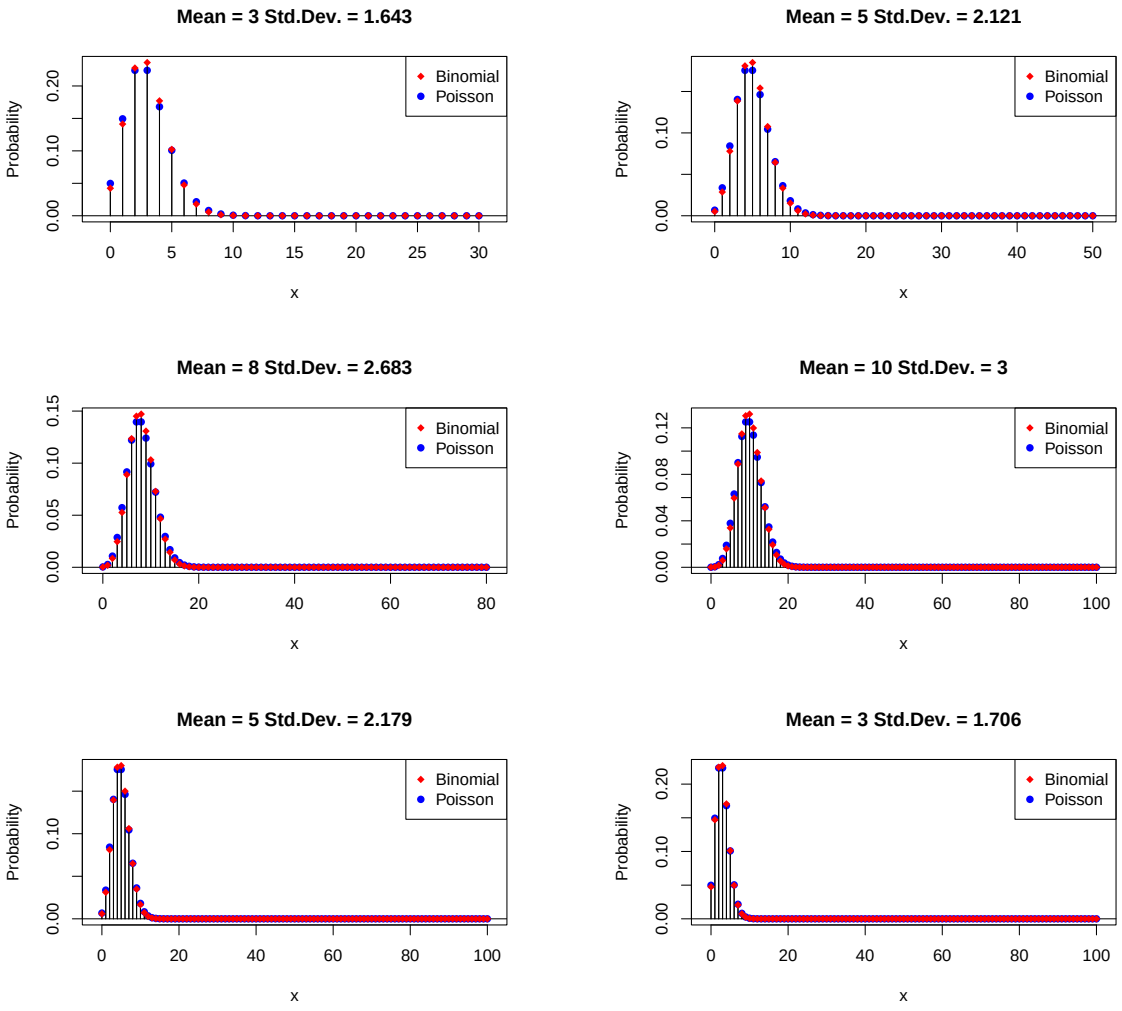
binomial	poisson
0	0.0497871
0	0.1493612
0	0.2240418

0 0.2240418
0 0.1680314
0 0.1008188

d) Plotten Sie die Wahrscheinlichkeitsfunktionen der vorherigen Modelle. Erhöhen Sie die Anzahl der Wiederholungen und verringern Sie die Erfolgswahrscheinlichkeit im Binomialmodell und beobachten Sie, wie sich die Wahrscheinlichkeiten des Binomialmodells und des Poissonmodells annähern.

```
# um nicht immer wieder den selben Plot-Befehl aufzurufen
# erstellen wir eine Hilfsfunktion
#--------------------------------------------------
myplot <- function(n, p){
  # vorberechnen
  mu <- p*n
  sd <- sqrt(n*p*(1-p))
  # plotten
  plot( seq(0,n), dpois( seq(0,n), mu ), type="h",
       xlim=c(-1,n+1), xlab="x", ylab="Probability",
       ylim=range(0,dpois( seq(0,n), mu), dbinom(seq(0,n),n,p)))
  points( seq(0,n), dpois( seq(0,n), mu ), pch=16, col="blue")
  points( seq(0,n), dbinom( seq(0,n), n, p), type="h")
  abline(h=0)
  points( seq(0,n), dbinom( seq(0,n), n, p), pch=18, col="red" )
  title( paste("Mean", "=", round(mu,3), "Std.Dev.", "=", round(sd,3)))
  legend("topright", c("Binomial", "Poisson"),
         col = c("red","blue"), pch = c(18,16))   }
#--------------------------------------------------

# plots vergleichen
myplot(30, 0.1)
myplot(50, 0.1)
myplot(80, 0.1)
myplot(100, 0.1)
myplot(100, 0.05)
myplot(100, 0.03)
```



4.2.38 Lösung zur Aufgabe 4.1.6.4

Wie groß ist die Wahrscheinlichkeit, beim Werfen von 100 Münzen zwischen 40 und 60 Mal `Kopf` zu erhalten (beide Werte eingeschlossen)?

```
sum(dbinom(40:60, size = 100, prob = 0.5))
```

```
[1] 0.9647998
```

4.2.39 Lösung zur Aufgabe 4.1.6.5

a) wie groß ist die Wahrscheinlichkeit, dass die Hälfte der Patienten geheilt wird?

```
# n=6 Patienten
# p =0.85
# k=3 (die Hälfte von 6)
dbinom(3, size = 6, prob = 0.85)
```

```
[1] 0.04145344
```

b) wie groß ist die Wahrscheinlichkeit, dass mindestens 4 Patienten geheilt werden?

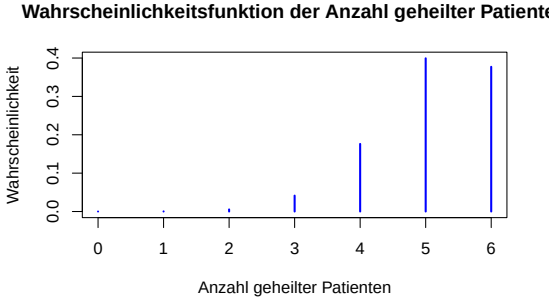
```
pbinom(3, size = 6, prob = 0.85, lower.tail = FALSE)
```

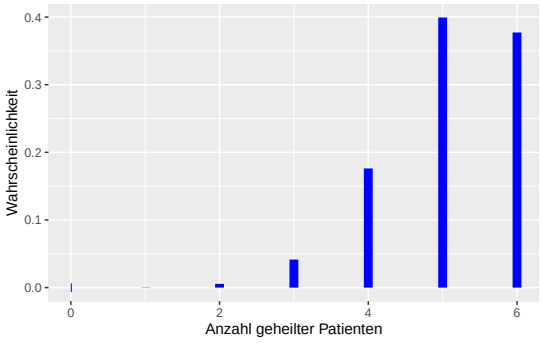
```
[1] 0.9526614
```

c) plotten Sie die Wahrscheinlichkeitsfunktion für die Anzahl geheilter Patienten.

```
df <- data.frame(x=0:6,
                 y=dbinom(0:6, size = 6, prob = 0.85))

# plot()
plot(df$x, df$y, type="h", lwd=2, col="blue",
     xlab = "Anzahl geheilter Patienten",
     ylab = "Wahrscheinlichkeit",
     main = "Wahrscheinlichkeitsfunktion der Anzahl geheilter Patienten")
# ggplot
ggplot(df, aes(x=x, y=0, xend=x, yend=y)) +
  geom_segment(col="blue", lwd=3) +
  xlab("Anzahl geheilter Patienten") +
  ylab("Wahrscheinlichkeit")
```





4.2.40 Lösung zur Aufgabe 4.1.6.6

Die Wahrscheinlichkeit einer starken Impfreaktion beträgt $0,001$. Wenn 2.000 Personen geimpft werden, wie hoch ist die Wahrscheinlichkeit für starke Reaktionen?

```
# n=2000 Patienten
# p =0.001
# k=1
pbinom(1, size = 2000, prob = 0.001, lower.tail=FALSE)
```

```
[1] 0.5941296
```

4.2.41 Lösung zur Aufgabe 4.1.6.7

a) Wie hoch ist die Wahrscheinlichkeit, dass weniger als 4 Anrufe in 2 Sekunden eintreffen?

```
# 120 Anrufe pro Minute sind
# 2 Anrufe pro Sekunde
# lambda für 2 Sekunden ist also 4
ppois(3, lambda=4, lower.tail=TRUE)
```

```
[1] 0.4334701
```

b) Wie hoch ist die Wahrscheinlichkeit, dass mindestens 3 Anrufe in 3 Sekunden eintreffen?

```
ppois(2, lambda=6, lower.tail=FALSE)
```

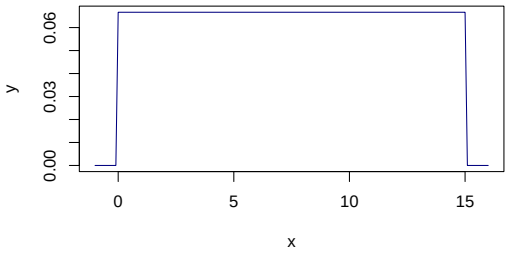
```
[1] 0.9380312
```

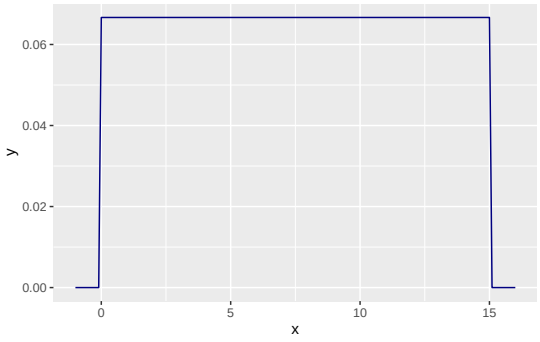
4.2.42 Lösung zur Aufgabe 4.1.7.1

a) Plotten Sie die Dichtefunktion der Wartezeit.

```
# x-Werte
x <- seq(-1, 16, by=0.1)
# Dichtefunktion der Uniformverteilung für alle x
y <- dunif(x, min=0, max=15)
# Datenframe
df <- data.frame(x, y)

# plot()
plot(x,y, type="l", col="navyblue")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="navyblue")
```

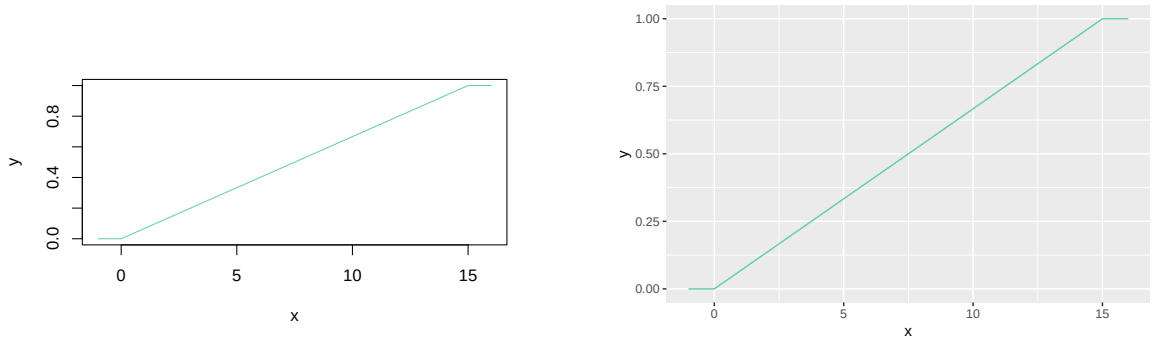




b) Plotten Sie die Verteilungsfunktion der Wartezeit.

```
# x-Werte
x <- seq(-1, 16, by=0.1)
# Verteiluingsfunktion der Uniformverteilung für alle x
y <- punif(x, min=0, max=15)
# Datenframe
df <- data.frame(x, y)

# plot()
plot(x,y, type="l", col="aquamarine3")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="aquamarine3")
```



c) Berechnen Sie die Wahrscheinlichkeit, weniger als 5 Minuten auf den Bus zu warten.

```
punif(5, min = 0, max = 15, lower.tail=TRUE)
```

```
[1] 0.3333333
```

d) Berechnen Sie die Wahrscheinlichkeit, länger als 12 Minuten auf den Bus zu warten.

```
punif(12, min = 0, max = 15, lower.tail=FALSE)
```

```
[1] 0.2
```

Die Wahrscheinlichkeit beträgt 20%.

e) Berechnen Sie die Wahrscheinlichkeit, zwischen 5 und 10 Minuten auf den Bus zu warten.

```
punif(10, min = 0, max = 15) - punif(5, min = 0, max = 15)
```

```
[1] 0.3333333
```

Die Wahrscheinlichkeit beträgt 33,33%.

f) Bei welcher Zeit zwischen 0 und 15 Minuten muss die Hälft der Personen kürzer auf den Bus warten als die angegebene Zeit?

```
qunif(0.5, min = 0, max = 15, lower.tail=TRUE)
```

```
[1] 7.5
```

50% der Personen muss weniger als 7,5 Minuten auf den Bus warten.

g) Bei welcher Zeit zwischen 0 und 15 Minuten müssen 10% der Personen länger auf den Bus warten als die angegebene Zeit?

```
qunif(0.1, min = 0, max = 15, lower.tail=FALSE)
```

```
[1] 13.5
```

10% der Personen müssen länger als 13,5 Minuten auf den Bus warten.

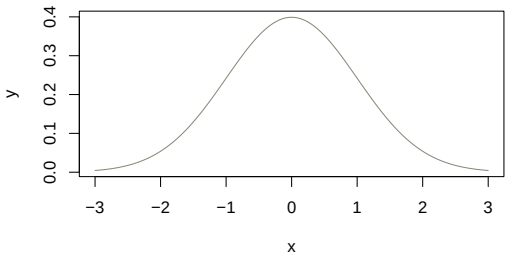
4.2.43 Lösung zur Aufgabe 4.1.7.2

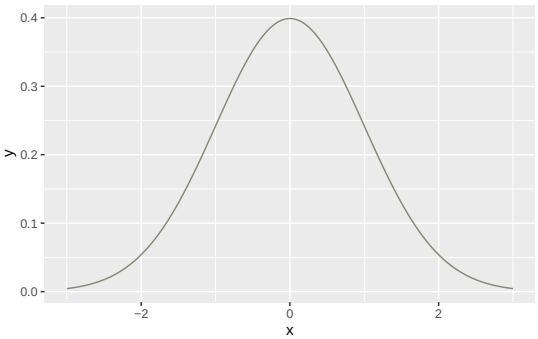
a) Plotten Sie die Dichtefunktion von Z.

```
x <- seq(-3, 3, 0.01)
y <- dnorm(x, mean = 0, sd = 1)

df = data.frame(x = x, y = y)

# plot()
plot(x,y, type="l", col="cornsilk4")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="cornsilk4")
```

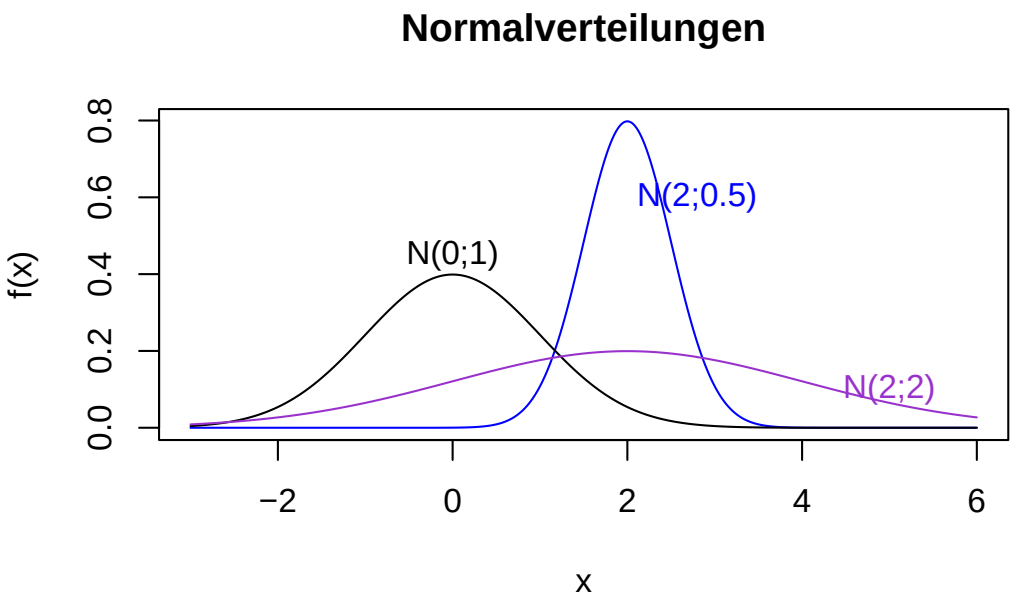




b) Wie beeinflussen Mittelwert und Standardabweichung die Form der Gausschen Glockenkurve?

```
# erzeuge neue Werte von -3 bis 6
x <- seq(-3,6, by=0.005)

# Alles zusammen plotten
plot(x,dnorm(x,mean=2,s=0.5), col="blue", type="l", xlab="x",
     ylab="f(x)",main="Normalverteilungen")
lines(x,dnorm(x,mean=0,s=1), col="black")
lines(x,dnorm(x,mean=2,s=2), col="darkorchid")
text(0,.45,"N(0;1)")
text(2.8, 0.6, "N(2;0.5)", col="blue")
text(5, 0.1, "N(2;2)", col="darkorchid")
```

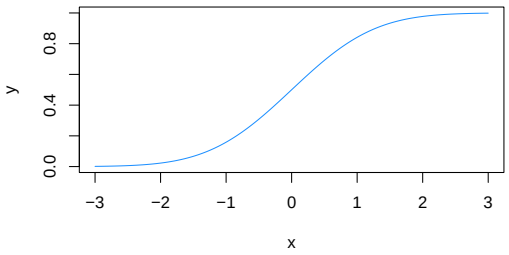


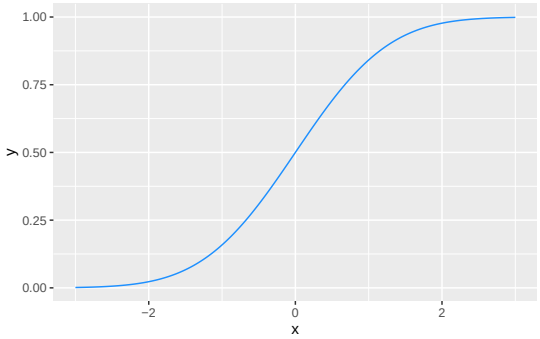
Der MIttelwert verschiebt die Kurve, die Standardabweichung verformt sie. Je größer die Standardabweichung, desto flacher und breiter ist die Kurve.

c) Plotten Sie die Verteilungsfunktion von Z.

```
# erzeuge neue Werte von -3 bis 3
x <- seq(-3, 3, 0.01)
y <- pnorm(x, mean=0, sd=1)
df = data.frame(x, y)

# plot()
plot(x,y, type="l", col="dodgerblue1")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="dodgerblue1")
```





d) Berechnen Sie die Wahrscheinlichkeit $P(Z < -1)$.

```
pnorm(-1, mean=0, sd=1, lower.tail=TRUE)
```

```
[1] 0.1586553
```

e) Berechnen Sie die Wahrscheinlichkeit $P(Z > 1)$.

```
pnorm(1, mean=0, sd=1, lower.tail=FALSE)
```

```
[1] 0.1586553
```

f) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus der Standardabweichung und dem Mittelwert plus der Standardabweichung liegt, d. h. $P(-1 \leq Z \leq 1)$.

```
pnorm(1, mean=0, sd=1) - pnorm(-1, mean=0, sd=1)
```

```
[1] 0.6826895
```

g) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus zwei Standardabweichungen und dem Mittelwert plus zwei Standardabweichungen liegt, d. h. $P(-2 \leq Z \leq 2)$.

```
pnorm(2, mean=0, sd=1) - pnorm(-2, mean=0, sd=1)
```

```
[1] 0.9544997
```

h) Berechnen Sie die Wahrscheinlichkeit, dass Z zwischen dem Mittelwert minus drei Standardabweichungen und dem Mittelwert plus drei Standardabweichungen liegt, d. h. $P(-3 \leq Z \leq 3)$.

```
pnorm(3, mean=0, sd=1) - pnorm(-3, mean=0, sd=1)
```

```
[1] 0.9973002
```

i) Berechnen Sie die Quartile.

```
qnorm(c(0.25, 0.5, 0.75), mean=0, sd=1)
```

```
[1] -0.6744898  0.0000000  0.6744898
```

j) Bei welchem Z-Wert liegen 95% der Fläche unterhalb des Wertes?

```
qnorm(0.95, mean=0, sd=1, lower.tail=TRUE)
```

```
[1] 1.644854
```

k) Bei welchem Z-Wert liegen 2,5% der Fläche oberhalb des Wertes?

```
qnorm(0.025, mean=0, sd=1, lower.tail=FALSE)
```

```
[1] 1.959964
```

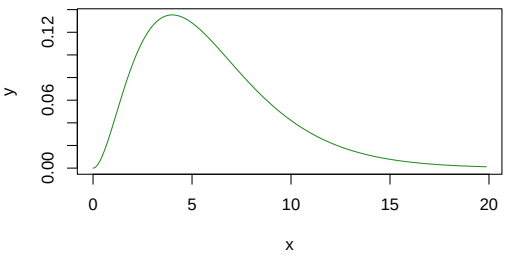
4.2.44 Lösung zur Aufgabe 4.1.7.3

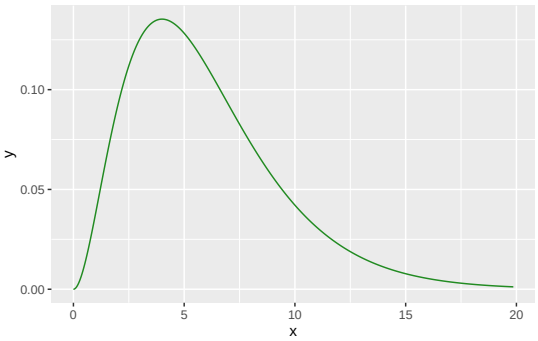
a) Plotten Sie die Dichtefunktion dieser Verteilung.

```
x <- seq(0, 19.86, 0.01)
y <- dchisq(x, df=6)

df = data.frame(x = x, y = y)

# plot()
plot(x,y, type="l", col="forestgreen")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="forestgreen")
```





b) Wie groß ist die Wahrscheinlichkeit für $P(X < 6)$?

```
pchisq(6, df=6, lower.tail=TRUE)
```

```
[1] 0.5768099
```

c) Berechnen Sie das fünfte Perzentil der Verteilung.

```
qchisq(0.05, df=6)
```

```
[1] 1.635383
```

d) Bei welchem Wert liegen 10% der Fläche oberhalb des Wertes?

```
qchisq(0.1, df=6, lower.tail=FALSE)
```

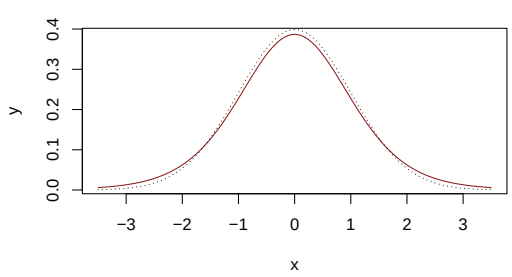
```
[1] 10.64464
```

4.2.45 Lösung zur Aufgabe 4.1.7.4

a) Plotten Sie die Dichtefunktion von X und vergleichen Sie diese mit der Dichtefunktion der Standardnormalverteilung.

```
x <- seq(-3.5, 3.5, 0.01)
y <- dt(x, df=8)
y2 <- dnorm(x)
df = data.frame(x=x, y=y, y2=y2)

# plot()
plot(x,y, type="l", col="firebrick4")
lines(x,y2, lty=3)
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="firebrick4") +
  geom_line(aes(x=x, y=y2), lty=3)
```





b) Berechnen Sie das 8te Perzentil von X.

```
qt(0.08, df=8)
```

```
[1] -1.548892
```

c) Bei welchem Wert von X liegen 5% aller Fälle oberhalb dieses Wertes?

```
qt(0.05, df=8, lower.tail = FALSE)
```

```
[1] 1.859548
```

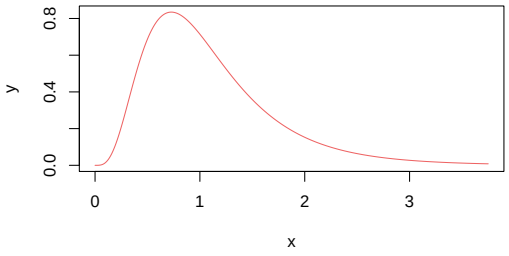
4.2.46 Lösung zur Aufgabe 4.1.7.5

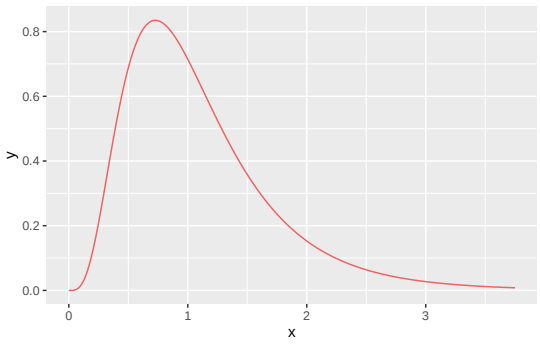
a) Plotten Sie die Dichtefunktion von X.

```
x <- seq(0, 3.75, 0.01)
y <- df(x, df1 = 10, df2 = 20)

df = data.frame(x=x, y=y)

# plot()
plot(x,y, type="l", col="indianred2")
# ggplot()
ggplot(df, aes(x=x, y=y)) +
  geom_line(col="indianred2")
```





b) Berechnen Sie die Wahrscheinlichkeit $P(X > 1)$.

```
pf(1, df1=10, df2=20, lower.tail=FALSE)
```

```
[1] 0.4755005
```

c) Berechnen Sie den Interquartilsabstand.

```
qf(c(0.75), df1=10, df2=20)  - qf(0.25, df1=10, df2=20)
```

```
[1] 0.7430938
```

4.2.47 Lösung zur Aufgabe 4.1.7.6

a) Berechnen Sie die Wahrscheinlichkeit, dass ein zufällig ausgewählter Diabetiker einen Glukosespiegel von weniger als 120 mg/100 ml hat.

```
pnorm(120, mean=106, sd=8)
```

```
[1] 0.9599408
```

b) Wie viel Prozent der Personen haben einen Glukosespiegel zwischen 90 und 120 mg/100 ml?

```
pnorm(120, mean=106, sd=8) - pnorm(90, mean=106, sd=8)
```

```
[1] 0.9371907
```

Etwa 93.72% der Personen.

c) Berechnen und interpretieren Sie das erste Quartil des Glukosespiegels.

```
qnorm(0.25, mean=106, sd=8)
```

```
[1] 100.6041
```

4.2.48 Lösung zur Aufgabe 4.1.7.7

a) Wie viele von ihnen haben einen Cholesterinspiegel zwischen 210 und 240 mg/dl?

```
# Anteile berechnen
pnorm(240, mean=220, sd=30) - pnorm(210, mean=220, sd=30)
```

```
[1] 0.3780661
```

Etwa 37.81% der Personen.

b) Wenn ein Cholesterinspiegel von mehr als 250 mg/dl eine Thrombose auslösen kann, wie viele von ihnen sind thrombosegefährdet?

```
pnorm(250, mean=220, sd=30, lower.tail=FALSE)
```

```
[1] 0.1586553
```

Etwa 15.87% der Personen.

c) Welcher Cholesterinwert wird von mindestens 20% der Männer erreicht?

```
# Anteile berechnen
qnorm(0.2, mean=220, sd=30)
```

```
[1] 194.7514
```

4.2.49 Lösung zur Aufgabe 4.1.8.1

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Konzentration`.

```
Konzentration <- c(17.6, 19.2, 21.3, 15.1, 17.6, 18.9, 16.2, 18.3, 19.0, 16.4)
```

b) Berechnen Sie das Konfidenzintervall für die mittlere Konzentration bei einem Konfidenzniveau von 95% (Signifikanzlevel $\alpha = 0,05$).

```
d <-t.test(Konzentration, mu=0, conf.level=0.95)
d
```

```
    One Sample t-test
```

```
data:  Konzentration
t = 31.78, df = 9, p-value = 1.485e-10
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 16.68158 19.23842
sample estimates:
mean of x
    17.96
```

```
# nur Konfidenzintervall ausgeben
as.numeric(d$conf.int)
```

```
[1] 16.68158 19.23842
```

c) Berechnen Sie das Konfidenzintervall für die mittlere Konzentration bei einem Konfidenzniveau von 99% (Signifikanzlevel $\alpha = 0,01$).

```
d <-t.test(Konzentration, mu=0, conf.level=0.99)
d
```

```
    One Sample t-test

data:  Konzentration
t = 31.78, df = 9, p-value = 1.485e-10
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 16.1234 19.7966
sample estimates:
mean of x
    17.96
```

```
# nur Konfidenzintervall ausgeben
as.numeric(d$conf.int)
```

```
[1] 16.1234 19.7966
```

d) Wenn wir die Genauigkeit des Intervalls als den Kehrwert seiner Breite definieren, wie ändert sich die Genauigkeit eines Intervalls, wenn wir das Konfidenzniveau erhöhen?

Mit höherem Konfidenzniveau sinkt die Genauigkeit der Aussagen.

e) Welche Stichprobengröße wird benötigt, um den mittleren Konzentrationswert mit einem Fehler von $\pm 0.5 \text{mg/mm}^3$ und einem Konfidenzniveau von 95% Sicherheit zu bestimmen?

```
# Berechnung der Standardabweichung der Stichprobe
sigma <- sd(Konzentration)

# Gegebene Werte
# z-Wert für 95% Konfidenzniveau
z <- 1.96

# Fehlermarge
E <- 0.5

# Berechnung der Stichprobengröße
n <- (z * sigma / E)^2

# Aufrunden auf die nächste ganze Zahl
ceiling(n)
```

```
[1] 50
```

f) Wenn die Konzentration des Wirkstoffs mindestens 16 mg/mm^3 betragen muss, um wirksam zu sein, ist dann unsere Medikamentencharge wirksam?

```
t.test(Konzentration, mu = 16, alternative = "greater")
```

```

    One Sample t-test

data:  Konzentration
t = 3.4682, df = 9, p-value = 0.003534
alternative hypothesis: true mean is greater than 16
95 percent confidence interval:
 16.92404      Inf
sample estimates:
mean of x
    17.96
```

Der Test ist signifikant. Wir können also sagen, dass unserer Medikamentencharge wirksam ist.

4.2.50 Lösung zur Aufgabe 4.1.8.2

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Hof` und `Fett`.

```
# Daten übertragen
Hof1 <- data.frame(Fett = c(0.34, 0.34,
          0.32, 0.35,
          0.33, 0.33,
          0.32, 0.32,
          0.33, 0.30,
          0.31, 0.32))
Hof1$Hof <- "Hof 1"

Hof2 <- data.frame(Fett = c(0.28, 0.29,
          0.30, 0.32,
          0.32, 0.31,
          0.29, 0.29,
          0.31, 0.32,
          0.29, 0.31,
          0.33, 0.32,
          0.32, 0.33))
Hof2$Hof <- "Hof 2"

milch <- rbind(Hof1, Hof2)
milch$Hof <- factor(milch$Hof)
```

b) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt.

```
t.test(milch$Fett, conf.level=0.95)
```

```
    One Sample t-test

data:  milch$Fett
t = 96.537, df = 27, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.3090040 0.3224246
sample estimates:
mean of x
0.3157143
```

c) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt, getrennt nach Höfen.

```
t.test(Hof1$Fett, conf.level=0.95)
```

```
    One Sample t-test

data:  Hof1$Fett
t = 81.853, df = 11, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.3170719 0.3345948
sample estimates:
mean of x
0.3258333
```

```
t.test(Hof2$Fett, conf.level=0.95)
```

```
    One Sample t-test

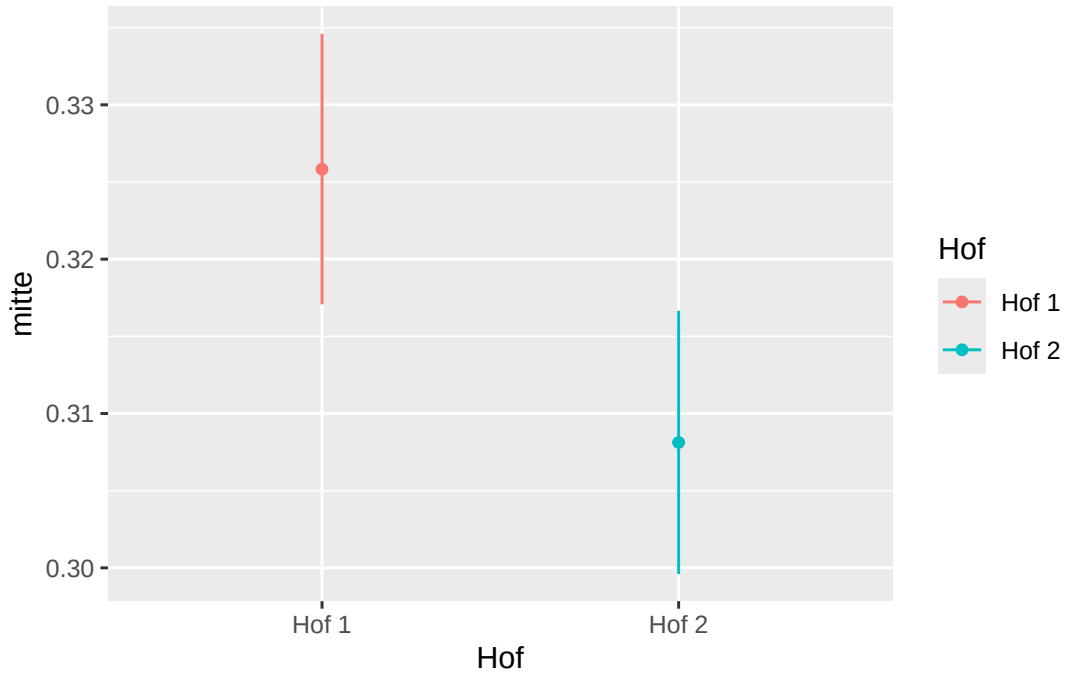
data:  Hof2$Fett
t = 76.994, df = 15, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.299595 0.316655
sample estimates:
mean of x
 0.308125
```

d) Plotten Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettgehalt, getrennt nach Höfen..

```
# Vorbereitung
h1 <- t.test(Hof1$Fett, conf.level=0.95)
h2 <- t.test(Hof2$Fett, conf.level=0.95)

# Als Datenframe für ggplot zusammenbauen
df <- data.frame(unten = c(h1$conf.int[1], h2$conf.int[1]),
                 oben  = c(h1$conf.int[2], h2$conf.int[2]),
                 mitte = c(h1$estimate, h2$estimate ),
                 Hof = c("Hof 1", "Hof 2"))

# ggplot
ggplot(df, aes(x=Hof, color=Hof)) +
  geom_point(aes(y=mitte)) +
  geom_segment(aes(xend=Hof, y=unten, yend=oben))
```



e) Lässt sich aus den Konfidenzintervallen ein signifikanter Untschied zwischen den Höfen feststellen?

```
df
```

unten	oben	mitte	Hof
0.3170719	0.3345948	0.3258333	Hof 1
0.2995950	0.3166550	0.3081250	Hof 2

Die beiden Konfidenzintervalle überschneiden sich nicht. Das heisst, es kann ein signifikanter Unterschied abgeleitet werden.

4.2.51 Lösung zur Aufgabe 4.1.8.3

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Antwort`.

```
# Daten übertragen
Antwort <- c("nein",  "ja",  "nein",  "nein",  "nein",  "ja",  "nein",
             "ja", "ja",  "ja",  "ja",  "nein",  "ja",  "nein", "ja",
             "nein",  "nein", "nein",  "ja",  "ja",  "ja",  "nein",
             "nein",  "ja",  "nein",  "nein", "ja",  "ja",  "nein",
             "nein",  "ja",  "nein",  "ja", "nein")
```

b) Berechnen Sie das Konfidenzintervall für den Anteil an Studierenden, welche die Bibliothek wöchentlich nutzen mit einem Signifikanzlevel von $\alpha = 0,01$.

```
freq <- table(Antwort)
bib <- prop.test(freq[["ja"]], sum(freq),
                 alternative="two.sided",
                 p=0.5, conf.level=0.99)
bib
```

```
    1-sample proportions test with continuity correction

data:  freq[["ja"]] out of sum(freq), null probability 0.5
X-squared = 0.029412, df = 1, p-value = 0.8638
alternative hypothesis: true p is not equal to 0.5
99 percent confidence interval:
 0.2617050 0.6896622
sample estimates:
        p
0.4705882
```

c) Wie präzise ist das Intervall?

```
bib$conf.int[2] - bib$conf.int[1]
```

```
[1] 0.4279572
```

Das Intervall ist sehr breit und daher unpräzise.

d) Welcher Stichprobenumfang ist erforderlich, um eine Schätzung des Anteils der Studenten zu erhalten, die die Bibliothek mindestens einmal pro Woche nutzen, mit einem Fehler von ±1% und einem Konfidenzniveau von 95%?

```
# gemessene Proportionen
prop <- bib$estimate

# Z-Wert für 95% Konfidenz
z <- 1.96

# Fehlerspanne +-1%
e <- 0.01

# Fallzahl berechnen
n <- (z^2 * prop * (1 - prop)) / (e^2)
```

Es werden 9571 Probanden benötigt.

4.2.52 Lösung zur Aufgabe 4.1.8.4

a) Berechnen Sie das 95%-Konfidenzintervall für den Anteil an geimpften Probanden in der Grundgesamtheit.

```
# Daten übertragen
prop.test(154, 200, p=0.5, conf.level=0.95)
```

```
    1-sample proportions test with continuity correction

data:  154 out of 200, null probability 0.5
X-squared = 57.245, df = 1, p-value = 3.848e-14
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.7042503 0.8251428
sample estimates:
   p
0.77
```

b) Wenn das Gesundheitsministerium das Ziel verfolgt, dass mindestens 70% der Menschen über 65 mit Atemwegserkrankungen geimpft sind, können wir dann sagen, dass das Ministerium das Ziel erreicht hat?

```
# Daten übertragen
prop.test(154, 200, p=0.5, conf.level=0.95)$conf.int[1]
```

```
[1] 0.7042503
```

Da die untere Grenze des Konfidenzintervalls größer als 0,7 ist, können wir bestätigen, dass das Ministerium sein Ziel erreicht hat.

4.2.53 Lösung zur Aufgabe 4.1.8.5

a) Berechnen Sie die Konfidenzintervalle für den Mittelwert mit den Signifikanzniveaus 0.1, 0.05 und 0.01.

```
# Daten übertragen
cholesterin <- c(196, 212, 188, 206, 203, 210, 201, 198)

t.test(cholesterin, conf.level=0.90)
```

```

    One Sample t-test

data:  cholesterin
t = 72.849, df = 7, p-value = 2.416e-11
alternative hypothesis: true mean is not equal to 0
90 percent confidence interval:
 196.5031 206.9969
sample estimates:
mean of x
   201.75
```

```
t.test(cholesterin, conf.level=0.95)
```

```

    One Sample t-test

data:  cholesterin
t = 72.849, df = 7, p-value = 2.416e-11
alternative hypothesis: true mean is not equal to 0
```

```
95 percent confidence interval:
 195.2014 208.2986
sample estimates:
mean of x
   201.75
```

```
t.test(cholesterin, conf.level=0.99)
```

```

    One Sample t-test

data:  cholesterin
t = 72.849, df = 7, p-value = 2.416e-11
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 192.0585 211.4415
sample estimates:
mean of x
   201.75
```

b) Kann man schließen, dass der Mittelwert des Cholesterinspiegels der Bevölkerung unter 210 mg/dl liegt?

```
# Daten übertragen
as.numeric(t.test(cholesterin, conf.level=0.90)$conf.int)
```

```
[1] 196.5031 206.9969
```

```
as.numeric(t.test(cholesterin, conf.level=0.95)$conf.int)
```

```
[1] 195.2014 208.2986
```

```
as.numeric(t.test(cholesterin, conf.level=0.99)$conf.int)
```

```
[1] 192.0585 211.4415
```

Die obere Grenze des 99%-Konfidenzintervall ist größer als 210. Wird dies berücksichtigt, lässt dich die Aussage nicht bestätigen.

4.2.54 Lösung zur Aufgabe 4.1.8.6

a) Berechnen Sie für jede Therapie das 95% Konfidenzintervall für den Anteil an Personen, die geheilt wurden.

```
# Daten übertragen
a <- c(rep("geheilt", 18), rep("nicht", 7))
b <- c(rep("geheilt", 21), rep("nicht", 14))

freqA <- table(a)
freqB <- table(b)
A <- prop.test(freqA[["geheilt"]], sum(freqA),
                alternative="two.sided",
                p=0.5, conf.level=0.95)
B <- prop.test(freqB[["geheilt"]], sum(freqB),
                alternative="two.sided",
                p=0.5, conf.level=0.95)
# Konfidenzintervalle
A$conf.int[2] - A$conf.int[1]
```

```
[1] 0.3672662
```

```
B$conf.int[2] - B$conf.int[1]
```

```
[1] 0.3343891
```

Das Konfidenzintervall von Gruppe B ist schmaler, und damit auch präziser.

4.2.55 Lösung zur Aufgabe 4.1.8.7

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/neonates.RData"))
```

a) Berechnen Sie das 99% Konfidenzintervall für den Mittelwert des Gewichts der Neugeborenen.

```
# t-Test
d <- t.test(neonates$weight, conf.level = 0.99)

# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] 2.975844 3.075531
```

b) Berechnen Sie die Konfidenzintervalle für den APGAR-Score nach 1 Minute und für den APGAR-Score nach 5 Minuten und vergleiche sie beide Intervalle. Gibt es auf Grundlage der Konfidenzintervalle einen signifikanten Unterschied zwischen den Mittelwerten der beiden Scores?

```
# t-Test
a1 <- t.test(neonates$apgar1, conf.level = 0.99)
a5 <- t.test(neonates$apgar5, conf.level = 0.99)

# Konfidenzgrenzen
as.numeric(a1$conf.int)
```

```
[1] 5.422182 5.834068
```

```
as.numeric(a5$conf.int)
```

```
[1] 5.998597 6.426403
```

Die Konfidenzgrenzen schneiden sich nicht. Das bedeutet, wir können von einem signifikanten Unterschied ausgehen.

c) Berechnen Sie die Konfidenzintervalle für den Prozentsatz der Neugeborenen mit einem Gewicht von $\leq 2,5$ kg für Raucher- und Nichtrauchermütter und vergleichen Sie die Intervalle.

```
# geringes Gewicht kategorisieren
neonates$GG <- "normal"
neonates$GG[neonates$weight<2.50001] <- "low"

# Subsets bilden
k1 <- subset(neonates, smoke=="Yes")
k2 <- subset(neonates, smoke=="No")
# Häufigkeitstabelle
freq1 <- table(k1$GG)
freq2 <- table(k2$GG)

# Proportion Test
m1 <- prop.test(freq1[["low"]], sum(freq1),
                alternative="two.sided",
                p=0.5, conf.level=0.95)

m2 <- prop.test(freq2[["low"]], sum(freq2),
                alternative="two.sided",
                p=0.5, conf.level=0.95)

# Konfidenzgrenzen anzeigen
as.numeric(m1$conf.int)
```

```
[1] 0.1049334 0.2610870
```

```
as.numeric(m2$conf.int)
```

```
[1] 0.008396857 0.055169043
```

Die Konfidenzgrenzen schneiden sich nicht. Das bedeutet, wir können von einem signifikanten Unterschied ausgehen.

4.2.56 Lösung zur Aufgabe 4.1.9.1

a) Erstellen Sie ein Datenframe mit den Variablen `vorher` und `nachher` und übertragen Sie die Daten.

```
df <- data.frame(
  vorher = c(147, 163, 121, 205, 132, 190, 176, 147),
  nachher = c(150, 171, 132, 208, 141, 184, 182, 145)
)
```

b) Berechnen Sie den Mittelwert der monatlichen Umsätze vor und nach der Kampagne. Sind die Mittelwerte unterschiedlich? Hat die Kampagne den Absatz des Arzneimittels erhöht?

```
mean(df$vorher)
```

```
[1] 160.125
```

```
mean(df$nachher)
```

```
[1] 164.125
```

Der Mittelwert ist nach der Kampagne höher.

c) Berechnen Sie die Konfidenzintervalle für den durchschnittlichen Unterschied mit $\alpha = 0,05$ und $\alpha = 0,01$.

```
# t-Tests durchführen
a5 <- t.test (df$vorher, df$nachher, paired=TRUE, conf.level=0.95)
a1 <- t.test (df$vorher, df$nachher, paired=TRUE, conf.level=0.99)

# nur Konfidenzintervalle anzeigen
as.numeric(a5$conf.int)
```

```
[1] -8.8129585  0.8129585
```

```
as.numeric(a1$conf.int)
```

```
[1] -11.122852   3.122852
```

Beide Intervalle "reißen" die 0. Das bedeutet, dass der wahre Unterschied auch `0` sein könnte.

d) Können wir dieselbe Schlussfolgerung ziehen, wenn wir die Verkäufe nach der Kampagne der beiden letzten Apotheken ändern und 190 statt 182 und 165 statt 145 angeben? Was passiert mit den Konfidenzintervallen?

```
# Daten neu eingeben
df <- data.frame(
  vorher = c(147, 163, 121, 205, 132, 190, 176, 147),
  nachher = c(150, 171, 132, 208, 141, 184, 190, 165)
)
# t-Tests durchführen
a5 <- t.test (df$vorher, df$nachher, paired=TRUE, conf.level=0.95)
a1 <- t.test (df$vorher, df$nachher, paired=TRUE, conf.level=0.99)

# nur Konfidenzintervalle anzeigen
as.numeric(a5$conf.int)
```

```
[1] -13.740228  -1.259772
```

```
as.numeric(a1$conf.int)
```

```
[1] -16.735113   1.735113
```

Das 95%-Intervall lässt einen Unterschied vermuten. Das 99%-Intervall hingegen enthält immer noch die 0.

4.2.57 Lösung zur Aufgabe 4.1.9.2

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Hof1` und `Hof2`.

```
# Daten übertragen
Hof1 <- data.frame(Fett = c(0.34, 0.34,
          0.32, 0.35,
          0.33, 0.33,
          0.32, 0.32,
          0.33, 0.30,
          0.31, 0.32))
Hof1$Hof <- "Hof 1"

Hof2 <- data.frame(Fett = c(0.28, 0.29,
          0.30, 0.32,
          0.32, 0.31,
          0.29, 0.29,
          0.31, 0.32,
          0.29, 0.31,
          0.33, 0.32,
          0.32, 0.33))
Hof2$Hof <- "Hof 2"

milch <- rbind(Hof1, Hof2)
milch$Hof <- factor(milch$Hof)
```

b) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Fettunterschied in der Milch von `Hof1` und `Hof2`.

```
# Daten übertragen
# zuvor auf gleiche Varainz prüfen
var.test(Hof1$Fett, Hof2$Fett)$p.value
```

```
[1] 0.626044
```

Der Test ist nicht signifikant, die Varianz ist in beiden Höfen gleich.

```
# t.Test
d <- t.test(Fett ~ Hof, data=milch, var.equal=TRUE)
# Konfidenzintervall anzeigen
as.numeric(d$conf.int)
```

```
[1] 0.00584816 0.02956851
```

c) Kann man daraus schließen, dass der Unterschied zwischen den Milchfettmittelwerten der Betriebe signifikant ist? Welcher Betrieb hat Milch mit mehr Fett? Wie viel mehr Fett hat die Milch von `Hof1` als die Milch von `Hof2`?

```
d <- t.test(Fett ~ Hof, data=milch, var.equal=TRUE)
d
```

```
    Two Sample t-test

data:  Fett by Hof
t = 3.0691, df = 26, p-value = 0.004973
alternative hypothesis: true difference in means between group Hof 1 and group Hof 2
95 percent confidence interval:
 0.00584816 0.02956851
sample estimates:
mean in group Hof 1 mean in group Hof 2
          0.3258333           0.3081250
```

```
# Unterschied
d$estimate[1] - d$estimate[2]
```

```
mean in group Hof 1
         0.01770833
```

Hof1 hat 0.0177083 mehr Fett in der Milch als Hof2. Da das Intervall die 0 nicht mit einschließt, können wir von einem Signifikanten Unterschied ausgehen.

4.2.58 Lösung zur Aufgabe 4.1.9.3

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Antwort` und `Geschlecht`.

```
# Daten übertragen
df <- data.frame(
  Antwort = c("nein", "ja", "nein", "nein", "nein", "ja", "nein",
              "ja", "ja", "ja", "ja", "nein", "ja", "nein", "ja",
              "nein", "nein",  "nein", "ja", "ja", "ja", "nein",
              "nein", "ja", "nein", "nein", "ja", "ja", "nein",
              "nein", "ja", "nein", "ja", "nein"),
  Geschlecht = c("m", "w", "w", "m", "m", "m", "w", "w", "w", "w",
                 "m", "m", "w", "m", "w", "m", "m", "w", "m", "w",
                 "w", "w", "m", "w", "m", "m", "w", "w", "m", "m",
                 "w", "w", "w", "m")
)
```

b) Berechnen Sie das Konfidenzintervall für den Unterschied zwischen den Anteilen der Frauen und Männern, die die Bibliothek mindestens einmal pro Woche nutzen.

```
freq <- table(df)
prop.test(c(freq[["ja","m"]], freq[["ja","w"]]),
          c(sum(freq[,"m"]), sum(freq[,"w"])),
          alternative="two.sided", conf.level=0.95)
```

```
    2-sample test for equality of proportions with continuity correction

data:  c(freq[["ja", "m"]], freq[["ja", "w"]]) out of c(sum(freq[, "m"]), sum(freq[,
X-squared = 7.6937, df = 1, p-value = 0.005541
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.8755141 -0.1939304
sample estimates:
   prop 1    prop 2
0.1875000 0.7222222
```

Das Konfidenzintervall schließt die 0 nicht mit ein. Wir können von einem signifikanten Unterschied ausgehen.

4.2.59 Lösung zur Aufgabe 4.1.9.4

```
df <- data.frame(course = c(rep("bestanden", 55), rep("durchgefallen", 25),
                            rep("bestanden", 32), rep("durchgefallen", 58)),
                 time = c(rep("morgens", 80), rep("abends", 90))

                 )
```

Gibt es signifikante Unterschiede zwischen den Prozentsätzen der Studierenden, die am Vormittag und am Nachmittag bestanden haben? Kann man daraus schließen, dass der Stundenplan die Ursache für diese Unterschiede ist?

```
freq <- table(df)
prop.test(c(freq[["bestanden","morgens"]], freq[["bestanden","abends"]]),
          c(sum(freq[,"morgens"]), sum(freq[,"abends"])),
          alternative="two.sided", conf.level=0.95)
```

```
    2-sample test for equality of proportions with continuity correction

data:  c(freq[["bestanden", "morgens"]], freq[["bestanden", "abends"]]) out of c(sum
X-squared = 17.372, df = 1, p-value = 3.072e-05
alternative hypothesis: two.sided
95 percent confidence interval:
 0.1783764 0.4855125
sample estimates:
   prop 1    prop 2
0.6875000 0.3555556
```

Das Konfidenzintervall enthält nicht die 0. Wir können also von einem signifikanten Unterschied ausgehen.

4.2.60 Lösung zur Aufgabe 4.1.9.5

```
df <- data.frame(vorher = c(182, 232, 191, 200, 148, 249, 276, 213, 241, 280, 262),
                 nachher = c(198, 210, 194, 220, 138, 220, 219, 161, 210, 213, 226)
                 )
```

a) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Unterschied der Cholesterinwerte vor und nach den körperlichen Übungen.

```
t.test(df$vorher, df$nachher, paired=TRUE, conf.level=0.95)
```

```
    Paired t-test

data:  df$vorher and df$nachher
t = 2.756, df = 10, p-value = 0.02027
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
  4.614342 43.567477
sample estimates:
mean difference
       24.09091
```

b) Berechnen Sie das 99%-Konfidenzintervall für den durchschnittlichen Unterschied der Cholesterinwerte vor und nach den körperlichen Übungen

```
t.test(df$vorher, df$nachher, paired=TRUE, conf.level=0.99)
```

```
    Paired t-test

data:  df$vorher and df$nachher
t = 2.756, df = 10, p-value = 0.02027
alternative hypothesis: true mean difference is not equal to 0
99 percent confidence interval:
 -3.61228 51.79410
sample estimates:
mean difference
       24.09091
```

c) Auf Grundlage der zuvor berechneten Intervalle, welchen Schluss bezüglich des Einflusses von körperlichen Aktivitäten auf den Cholesterinspiegel können Sie ziehen?

```
d <- t.test(df$vorher, df$nachher, paired=TRUE, conf.level=0.99)
as.numeric(d$conf.int)
```

```
[1] -3.61228 51.79410
```

Das 99% Intervall enthält die 0. Auf diesem Niveau können wir den Unterschied nicht bestätigen.

4.2.61 Lösung zur Aufgabe 4.1.9.6

```
df <- data.frame(satisf = c(rep("zufrieden", 140), rep("unzufrieden", 60),
                            rep("zufrieden", 180), rep("unzufrieden", 120)),
                 haus = c(rep("Haus 1", 200), rep("Haus 2", 300))
                 )
```

a) Berechnen Sie das 95%-Konfidenzintervall für den Anteilsunterschied an zufriedenen Patienten in beiden Häusern.

```
freq <- table(df)
prop.test(c(freq[["zufrieden","Haus 1"]], freq[["zufrieden","Haus 2"]]),
          c(sum(freq[,"Haus 1"]), sum(freq[,"Haus 2"])),
          alternative="two.sided", conf.level=0.95)
```

```
    2-sample test for equality of proportions with continuity correction

data:  c(freq[["zufrieden", "Haus 1"]], freq[["zufrieden", "Haus 2"]]) out of c(sum(
X-squared = 4.7833, df = 1, p-value = 0.02874
alternative hypothesis: two.sided
95 percent confidence interval:
 0.01153209 0.18846791
sample estimates:
prop 1 prop 2
   0.7    0.6
```

b) Wenn $\alpha = 0,01$ ist, können dann Rückschlüsse gezogen werden, ob der Unterschied der Anteile zufriedener Patienten signifikant ist?

```
prop.test(c(freq[["zufrieden","Haus 1"]], freq[["zufrieden","Haus 2"]]),
          c(sum(freq[,"Haus 1"]), sum(freq[,"Haus 2"])),
          alternative="two.sided", conf.level=0.99)
```

```
    2-sample test for equality of proportions with continuity correction

data:  c(freq[["zufrieden", "Haus 1"]], freq[["zufrieden", "Haus 2"]]) out of c(sum(
X-squared = 4.7833, df = 1, p-value = 0.02874
alternative hypothesis: two.sided
```

```
99 percent confidence interval:
 -0.01495727  0.21495727
sample estimates:
prop 1 prop 2
   0.7    0.6
```

Wenn $\alpha = 0,01$ verwendet wird, enthält das Intervall die 0. Auf diesem Niveau können wir keinen signifikanten Unterschied zeigen.

4.2.62 Lösung zur Aufgabe 4.1.9.7

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/neonates.RData"))
```

a) Berechnen Sie das 95%-Konfidenzintervall für den durchscnnittlichen Unterschied des Geburtsgewichts zwischen Kindern von Raucherinnen und Nichtraucherinnen. Wie groß ist der durchschnittliche Gewichtsunterschied?

```
# subsets
k1 <- subset(neonates, smoke=="Yes")
k2 <- subset(neonates, smoke=="No")

# t-Test
d <- t.test(k1$weight, k2$weight, conf.level = 0.95)

# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] -0.3714943 -0.2179094
```

```
# Unterschied
d$estimate[2] - d$estimate[1]
```

```
mean of y
0.2947018
```

Der Unterschied beträg 0.29kg.

b) Berücksichtigen Sie nur die Daten der Mütter, die während der Schwangerschaft nicht geraucht haben. Berechnen Sie das 95%-Konfidenzintervall für den durchscnnittlichen Unterschied des Geburtsgewichts zwischen Kindern von Müttern, die vor der Schwangerschaft geraucht haben, und den Nichtraucherinnen.

```
# subsets
k  <- subset(neonates, smoke=="No")
k1 <- subset(k, smoke.before=="Yes")
k2 <- subset(k, smoke.before=="No")

# t-Test
d <- t.test(k1$weight, k2$weight, conf.level = 0.95)

# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] -0.23226491  0.03252437
```

```
# Unterschied
d$estimate[2] - d$estimate[1]
```

```
 mean of y
0.09987027
```

Das Konfidenzintervall enthält die 0, das heisst, wir können nicht von einem signifikanten Unterschied ausgehen.

c) Berechnen Sie das 95%-Konfidenzintervall für den durchschnittlichen Unterschied von APGAR-1-Werten und APGAR-5-Werten. Wie entwickeln sich Neugeborene in den ersten 5 Minuten nach der Geburt?

```
# t-Test
d <- t.test(neonates$apgar1, neonates$apgar5, conf.level = 0.95)

# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] -0.8093862 -0.3593638
```

```
# Unterschied
d$estimate[2] - d$estimate[1]
```

```
mean of y
 0.584375
```

Nach 5 Minuten haben die Neugeborenen einen im Schnitt 0,58 Punkte höheren APGAR-Wert.

d) Wenn Neugeborene mit einem APGAR-1-Wert ≤ 3 in einem kritischen Zustand sind, berechnen Sie das 90%-Konfidenzintervall für den Unterschied der Anteile von Neugeborenen in kritischem Zustand zwischen Müttern, die während der Schwangerschaft geraucht haben und den Nichtraucherinnen.

```
# neue Variable "kritisch"
neonates$kritisch <- "normal"
# nur solche mit APGAR<4 sind kritisch
neonates$kritisch[neonates$apgar1<4] <- "kritisch"

freq <- table(neonates$kritisch, neonates$smoke)

d <- prop.test(c(freq[["kritisch","No"]], freq[["kritisch","Yes"]]),
               c(sum(freq[,"No"]), sum(freq[,"Yes"])),
               alternative="two.sided", conf.level=0.90)



# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] -0.22157738 -0.07478626
```

```
# Unterschied
d$estimate[2] - d$estimate[1]
```

```
   prop 2
0.1481818
```

Bei Raucherinnen sind durchschnittlich 15% mehr Neugeborene im kritischen Zustand zu finden als bei Nichtraucherinnen.

e) Hat das Alter der Mutter einen signifikanten Einfluss auf den Anteil an Neugeborenen in kritischem Zustand?

```
# neue Variable "kritisch"
neonates$kritisch <- "normal"
# nur solche mit APGAR<4 sind kritisch
neonates$kritisch[neonates$apgar1<4] <- "kritisch"

freq <- table(neonates$kritisch, neonates$age)

d <- prop.test(c(freq[["kritisch","greater than 20"]],
                 freq[["kritisch","less than 20"]]),
               c(sum(freq[,"greater than 20"]),
                 sum(freq[,"less than 20"])),
               alternative="two.sided", conf.level=0.95)



# Konfidenzgrenzen
as.numeric(d$conf.int)
```

```
[1] -0.151048570  0.006238353
```

```
# Unterschied
d$estimate[2] - d$estimate[1]
```

```
    prop 2
0.07240511
```

Das 95% Intervall enthält die 0. Wir können keinen signifikanten Unterschied feststellen.

4.2.63 Lösung zur Aufgabe 4.1.10.1

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `Konzentration`.

```
Konzentration <- c(17.6, 19.2, 21.3, 15.1, 17.6, 18.9, 16.2, 18.3, 19.0, 16.4)
```

b) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 18$ versus $H_1 : \mu \neq 18$ mit einem Signifikanzniveau von $\alpha = 0,05$.

```
t.test (Konzentration, alternative="two.sided", mu=18, conf.level=0.95)
```

```
    One Sample t-test

data:  Konzentration
t = -0.07078, df = 9, p-value = 0.9451
alternative hypothesis: true mean is not equal to 18
95 percent confidence interval:
 16.68158 19.23842
sample estimates:
mean of x
    17.96
```

Das Ergebnis ist nicht signifikant.

c) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 19,5$ versus $H_1 : \mu \neq 19,5$ mit den Signifikanzniveaus von $\alpha = 0,05$ und $0,01$. Wie beeinflusst das Signifikanzniveau das Testergebnis?

```
t.test (Konzentration, alternative="two.sided", mu=19.5, conf.level=0.95)
```

```
    One Sample t-test

data:  Konzentration
t = -2.725, df = 9, p-value = 0.02341
alternative hypothesis: true mean is not equal to 19.5
95 percent confidence interval:
 16.68158 19.23842
sample estimates:
mean of x
    17.96
```

```
t.test (Konzentration, alternative="two.sided", mu=19.5, conf.level=0.99)
```

```
    One Sample t-test

data:  Konzentration
```

```
t = -2.725, df = 9, p-value = 0.02341
alternative hypothesis: true mean is not equal to 19.5
99 percent confidence interval:
 16.1234 19.7966
sample estimates:
mean of x
    17.96
```

Da der p-Wert bei $0,02341$ liegt, ist das Ergebnis für $\alpha = 0,05$ signifikant, für $\alpha = 0,01$ jedoch nicht.

d) Testen Sie die zweiseitige Hypothese $H_0 : \mu = 17$ versus $H_1 : \mu \neq 17$ mit einem Signifikanzniveau von $\alpha = 0,05$. Testen Sie ebenfalls die Hypothesen $H_0 : \mu = 17$ versus $H_1 : \mu > 17$ mit $\alpha = 0,05$. Was ist der Unterschied zwischen den p-Werten des zweiseitigen und des einseitigen Tests?

```
t.test (Konzentration, alternative="two.sided", mu=17, conf.level=0.95)
```

```
    One Sample t-test

data:  Konzentration
t = 1.6987, df = 9, p-value = 0.1236
alternative hypothesis: true mean is not equal to 17
95 percent confidence interval:
 16.68158 19.23842
sample estimates:
mean of x
    17.96
```

```
t.test (Konzentration, alternative="greater", mu=17, conf.level=0.95)
```

```
    One Sample t-test

data:  Konzentration
t = 1.6987, df = 9, p-value = 0.0618
alternative hypothesis: true mean is greater than 17
95 percent confidence interval:
 16.92404      Inf
sample estimates:
mean of x
    17.96
```

Der p-Wert ist beim einseitigen Test kleiner. Beide Werte sind jedoch größer als $0,05$.

e) Wenn der Hersteller angibt, die Konzentration des Wirkstoffs erhöht zu haben (im Vergleich zu früheren Chargen, bei denen der Mittelwert der Konzentration 17 mg/mm^3 war), können wir ihm glauben?

```
t.test (Konzentration, alternative="greater", mu=17, conf.level=0.95)
```

```
    One Sample t-test

data:  Konzentration
t = 1.6987, df = 9, p-value = 0.0618
alternative hypothesis: true mean is greater than 17
95 percent confidence interval:
 16.92404      Inf
sample estimates:
mean of x
    17.96
```

Der p-Wert ist nicht signifikant. Wir können dem Hersteller also nicht glauben.

f) Welche Fallzahl würde benötigt, um einen Konzentrationsanstieg von 0,5 mg/mm^3 zu erkennen (mit $\alpha = 0,05$ und einer Power von $1 - \beta = 0,8$)?

```
# Power-t-test
power.t.test(delta=0.5, sd=sd(Konzentration),
             sig.level=0.05, power=0.8, type = "one.sample")
```

```
     One-sample t test power calculation

              n = 102.2077
          delta = 0.5
             sd = 1.787114
      sig.level = 0.05
          power = 0.8
    alternative = two.sided
```

Es wird eine Fallzahl von 103 benötigt.

4.2.64 Lösung zur Aufgabe 4.1.10.2

a) Übertragen Sie die Daten in ein Datenframe mit der Variable `bib`.

```
# Daten übertragen
bib <- c("nein",  "ja",  "nein",  "nein",  "nein",  "ja",  "nein",
         "ja", "ja",  "ja",  "ja",  "nein",  "ja",  "nein", "ja",
         "nein",  "nein", "nein",  "ja",  "ja",  "ja",  "nein",
         "nein",  "ja",  "nein",  "nein", "ja",  "ja",  "nein",
         "nein",  "ja",  "nein",  "ja", "nein")
```

b) Testen Sie die Hypothese, dass der Anteil an Studierenden, die wöchentlich die Bibliothek nutzen, größer als 40% ist.

```
freq <- table(bib)
# testen
prop.test(freq[["ja"]], sum(freq), alternative="greater", p=0.4, conf.level=0.95)
```

```
    1-sample proportions test with continuity correction

data:  freq[["ja"]] out of sum(freq), null probability 0.4
X-squared = 0.4424, df = 1, p-value = 0.253
alternative hypothesis: true p is greater than 0.4
95 percent confidence interval:
 0.3238772 1.0000000
sample estimates:
        p
0.4705882
```

Der Test ist nicht signifikant.

4.2.65 Lösung zur Aufgabe 4.1.10.3

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Alter` und `Population`.

```
# Daten übertragen
df <- data.frame(Alter = c(9.5, 10.5, 9.0, 9.8, 10.0, 13.0,
                           10.0, 13.5, 10.0, 9.8, 12.5, 9.5,
                           13.5, 13.8, 12.0, 13.8, 12.5, 9.5,
                           12.0, 13.5, 12.0, 12.0),
                 Population = c(rep("A", 10), rep("B", 12)))
```

b) Testen Sie die Hypothese, dass das durchschnittliche Alter in den Populationen unterschiedlich ist, mit $\alpha = 0,05$.

```
# teste, ob Varianzhomogenität vorliegt
var.test(Alter ~ Population, data=df)$p.value
```

```
[1] 0.9164489
```

```
# liegt vor
t.test(Alter ~ Population, data=df, var.equal=TRUE)
```

```
    Two Sample t-test

data:  Alter by Population
t = -2.6982, df = 20, p-value = 0.01383
alternative hypothesis: true difference in means between group A and group B is not
95 percent confidence interval:
 -3.0260864 -0.3872469
sample estimates:
mean in group A mean in group B
       10.51000        12.21667
```

Das Ergebnis ist signifikant, p ist kleiner als 0,05. Es liegt also ein Unterschied vor.

4.2.66 Lösung zur Aufgabe 4.1.10.4

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `vorher` und `nachher`.

```
# Daten übertragen
df <- data.frame(vorher = c(60.6, 12.0, 56.0, 75.2, 12.5, 29.7,
                            57.2, 62.7, 28.7, 66.0, 25.2, 40.1),
                 nachher = c(47.5, 13.3, 33.0, 55.2, 21.9, 27.9,
                             54.3, 13.9, 8.90, 46.1, 29.8, 36.2))
```

b) Testen Sie, ob sich die Bronchialretention nach dem Rauchstopp verringert.

```
# Daten übertragen
t.test (df$vorher, df$nachher, alternative="greater", paired=TRUE, conf.level=0.95)
```

```
    Paired t-test
```

```
data:  df$vorher and df$nachher
t = 2.4847, df = 11, p-value = 0.01516
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 3.185837      Inf
sample estimates:
mean difference
       11.49167
```

Das Ergebnis ist signifikant, p ist kleiner als 0,05. Es liegt also ein Unterschied vor, die Retention hat sich verringert.

4.2.67 Lösung zur Aufgabe 4.1.10.5

Gibt es signifikante Unterschiede zwischen den Prozentsätzen der Studiereden, die am Vormittag und am Nachmittag bestanden haben? Kann man daraus schließen, dass der Stundenplan die Ursache für diese Unterschiede ist?

```r
# Daten übertragen
df <- data.frame(course = c(rep("bestanden", 55), rep("durchgefallen", 25),
                            rep("bestanden", 32), rep("durchgefallen", 58)),
                 time = c(rep("morgens", 80), rep("abends", 90))

                 )

freq <- table(df)
prop.test(c(freq[["bestanden","morgens"]], freq[["bestanden","abends"]]),
          c(sum(freq[,"morgens"]), sum(freq[,"abends"])),
          alternative="two.sided", conf.level=0.95)
```

```
    2-sample test for equality of proportions with continuity correction

data:  c(freq[["bestanden", "morgens"]], freq[["bestanden", "abends"]]) out of c(sum
X-squared = 17.372, df = 1, p-value = 3.072e-05
alternative hypothesis: two.sided
95 percent confidence interval:
 0.1783764 0.4855125
sample estimates:
   prop 1    prop 2
0.6875000 0.3555556
```

Das Ergebnis ist signifikant, p ist kleiner als 0,05. Es liegt also ein Unterschied zwischen

morgens und abends vor.

4.2.68 Lösung zur Aufgabe 4.1.10.6

```
# lade Datensatz
load(url("https://www.produnis.de/R/data/pulse.RData"))
```

a) Testen Sie, ob der Ruhepuls weniger als 75 Schläge pro Minute beträgt.

```
t.test(pulse$pulse1, mu=75, alternative = "less")
```

```

    One Sample t-test

data:  pulse$pulse1
t = -1.8562, df = 91, p-value = 0.03333
alternative hypothesis: true mean is less than 75
95 percent confidence interval:
     -Inf 74.77684
sample estimates:
mean of x
 72.86957
```

Das Ergebnis ist signifikant.

b) Welcher Stichprobenumfang ist erforderlich, um einen Anstieg des Ruhepulses um 2 Schläge pro Minute mit einem Signifikanzniveau von 0,05 und einer Power von 0,9 festzustellen?

```
power.t.test(delta=2, sd=sd(pulse$pulse1),
             sig.level=0.05, power=0.9)
```

```

     Two-sample t test power calculation

              n = 637.6676
          delta = 2
             sd = 11.00871
      sig.level = 0.05
          power = 0.9
    alternative = two.sided

NOTE: n is number in *each* group
```

Es werden 638 Probanden benötigt.

c) Testen Sie, ob der Puls nach dem Laufen größer als 85 Schläge pro Minute ist.

```
t.test(pulse$pulse2, mu=85, alternative="greater")
```

```
    One Sample t-test

data:  pulse$pulse2
t = -2.8056, df = 91, p-value = 0.9969
alternative hypothesis: true mean is greater than 85
95 percent confidence interval:
 77.03847      Inf
sample estimates:
mean of x
       80
```

Das Ergebnis ist nicht signifikant

d) Eine Person hat eine leichte Tachykardie, wenn der Ruhepuls größer als 90 Schläge pro Minute ist. Prüfen Sie, ob der Prozentsatz der Personen mit leichter Tachykardie größer als 5% ist.

```
pulse$tachy <- "nein"
pulse$tachy[pulse$pulse1 > 90] <- "ja"

freq <- table(pulse$tachy)
prop.test(freq[["ja"]], sum(freq), alternative="greater",
          p=0.05, conf.level=0.95)
```

```
    1-sample proportions test with continuity correction

data:  freq[["ja"]] out of sum(freq), null probability 0.05
X-squared = 0.18535, df = 1, p-value = 0.3334
alternative hypothesis: true p is greater than 0.05
95 percent confidence interval:
 0.03035962 1.00000000
sample estimates:
         p
0.06521739
```

Das Ergebnis ist nicht signifikant.

e) Kann man mit 95%iger Sicherheit schließen, dass Bewegung den Puls erhöht? Und bei einem Signifikanzniveau von $\alpha = 0,01$?

```
# test ob pulse1 kleiner ist als pulse2
t.test(pulse$pulse1, pulse$pulse2, alternative="less", conf.level = 0.95)
```

```
    Welch Two Sample t-test

data:  pulse$pulse1 and pulse$pulse2
t = -3.3638, df = 155.41, p-value = 0.0004841
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf -3.622838
sample estimates:
mean of x mean of y
 72.86957  80.00000
```

```
t.test(pulse$pulse1, pulse$pulse2, alternative="less", conf.level = 0.99)
```

```
    Welch Two Sample t-test

data:  pulse$pulse1 and pulse$pulse2
t = -3.3638, df = 155.41, p-value = 0.0004841
alternative hypothesis: true difference in means is less than 0
99 percent confidence interval:
      -Inf -2.147776
sample estimates:
mean of x mean of y
 72.86957  80.00000
```

Das Ergebnis ist in beiden Fällen signifikant. Bewegung erhöht also den Puls.

f) Gibt es einen Unterschied zwischen den durchschnittlichen Pulsschlägen nach dem Gehen und dem Laufen?

```
# test ob pulse1 kleiner ist als pulse2
t.test(pulse2 ~ type, data=pulse, conf.level = 0.95)
```

```
    Welch Two Sample t-test

data:  pulse2 by type
t = 5.8335, df = 45.695, p-value = 5.251e-07
alternative hypothesis: true difference in means between group running and group wal
95 percent confidence interval:
 13.22755 27.16944
sample estimates:
mean in group running mean in group walking
             92.51429              72.31579
```

Es gibt einen signifikanten Unterschied.

g) Gibt es einen Unterschied zwischen den Mittelwerten des Ruhepulses von Männern und Frauen? Und nach dem Laufen?

```
# test ob pulse1 kleiner ist als pulse2
t.test(pulse1 ~ sex, data=pulse, conf.level = 0.95)
```

```
    Welch Two Sample t-test

data:  pulse1 by sex
t = -2.7217, df = 63.675, p-value = 0.008367
alternative hypothesis: true difference in means between group male and group female
95 percent confidence interval:
 -11.160619  -1.711561
sample estimates:
  mean in group male mean in group female
            70.42105              76.85714
```

```
t.test(pulse2 ~ sex, data=pulse, conf.level = 0.95)
```

```
    Welch Two Sample t-test
```

```
data:  pulse2 by sex
t = -2.7849, df = 51.047, p-value = 0.007494
alternative hypothesis: true difference in means between group male and group female
95 percent confidence interval:
 -18.64912  -3.02507
sample estimates:
  mean in group male mean in group female
            75.87719             86.71429
```

Für beide Pulse gibt es signifikante Unterschiede zwischen Männern und Frauen.

4.2.69 Lösung zur Aufgabe 4.1.11.1

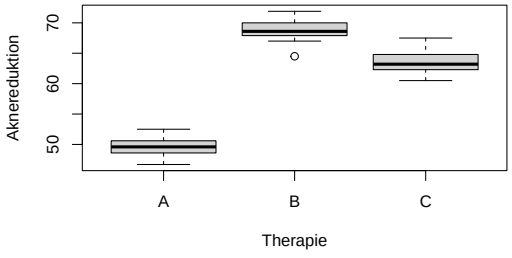
a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Therapie` und `Aknereduktion`.

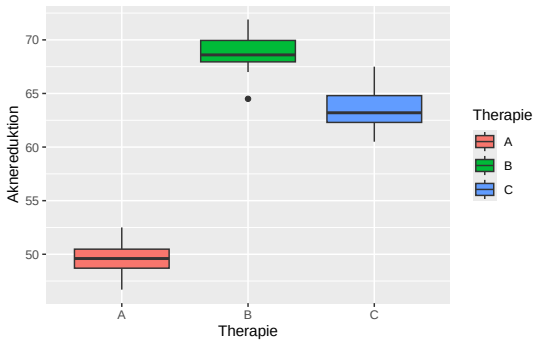
```r
A <- c(48.6, 50.8, 49.4, 47.1, 50.1, 52.5, 49.8, 49, 50.6, 46.7)
B <- c(68, 71.9, 67, 71.5, 70.1, 69.9, 64.5, 68.9, 68, 67.8, 68.3, 68.9)
C <- c(67.5, 61.4, 62.5, 67.4, 64.2, 65.4, 62.5, 63.2, 63.9, 61.2, 64.8, 60.5, 62.3)

df <- data.frame(Therapie = c(rep("A", length(A)),
                              rep("B", length(B)),
                              rep("C", length(C))),
                 Aknereduktion = c(A,B,C)
)
```

b) Plotten Sie die Aknereduktion für jede Therapie. Sind Unterschiede erkennbar?

```r
# plot()
boxplot(Aknereduktion ~ Therapie, data=df)
# ggplot()
ggplot(df, aes(x=Therapie, y=Aknereduktion)) +
  geom_boxplot(aes(fill=Therapie))
```





Therapie A unterscheidet sich deutlich von B und C.

c) Führen Sie eine ANOVA durch. Gibt es signifikante Unterschiede zwischen den Therapien?

```
fit <- aov(Aknereduktion ~ Therapie, data=df)
summary(fit)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
Therapie      2 2133.7  1066.8     262 <2e-16 ***
Residuals    32  130.3     4.1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Es gibt Unterschiede. Schauen wir genauer hin.

```
pairwise.t.test(df$Aknereduktion, df$Therapie, p.adjust="bonferroni")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  df$Aknereduktion and df$Therapie

  A       B
B < 2e-16 -
C < 2e-16 1.2e-06

P value adjustment method: bonferroni
```

Alle Gruppen unterscheiden sich jeweils voneinander.

d) Berechnen Sie die Konfidenzintervalle für die paarweisen Unterschiede zwischen den drei Behandlungen. Bei welchen Behandlungen gibt es signifikante Unterschiede?

```
ab <- t.test(A,B)
ac <- t.test(A,C)
cb <- t.test(C,B)

# Konfidenzintervalle
as.numeric(ab$conf.int)
```

```
[1] -20.93395 -17.61271
```

```
as.numeric(ac$conf.int)
```

```
[1] -15.85486 -12.42514
```

```
as.numeric(cb$conf.int)
```

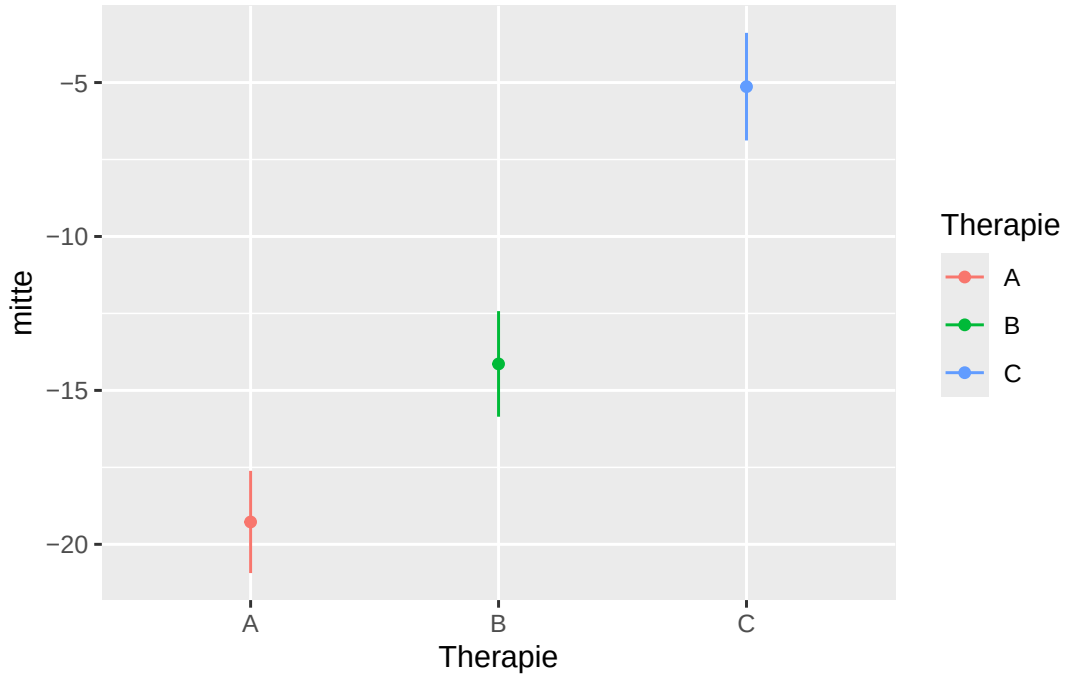
```
[1] -6.880637 -3.386029
```

Keines der Intervalle schließt die 0 ein. Alle Therapien unterscheiden sich jeweils voneinander.

e) Plotten Sie diese Konfidenzintervalle.

```
# Als Datenframe für ggplot zusammenbauen
plotdf <- data.frame(unten = c(ab$conf.int[1], ac$conf.int[1], cb$conf.int[1]),
                     oben  = c(ab$conf.int[2], ac$conf.int[2], cb$conf.int[2]),
                     mitte = c(ab$estimate[1]-ab$estimate[2],
                               ac$estimate[1]-ac$estimate[2],
                               cb$estimate[1]-cb$estimate[2] ),
                     Therapie = c("A", "B", "C"))

# ggplot
ggplot(plotdf, aes(x=Therapie, color=Therapie)) +
  geom_point(aes(y=mitte)) +
  geom_segment(aes(xend=Therapie, y=unten, yend=oben))
```



Keines der Intervalle schließt die 0 ein. Alle Therapien unterscheiden sich jeweils voneinander.

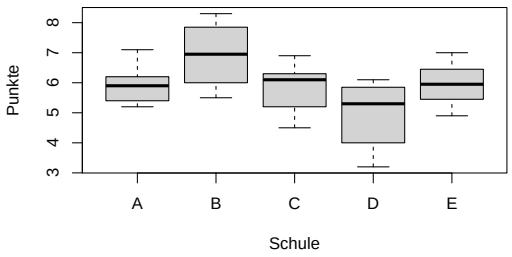
4.2.70 Lösung zur Aufgabe 4.1.11.2

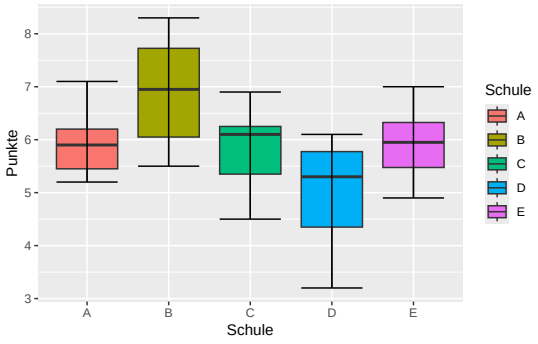
a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Schule` und `Punkte`.

```
# Erstellen des tibbles mit tribble()
df <- tribble(
  ~A,   ~B,   ~C,   ~D,   ~E,
  5.5,  6.1,  4.9,  3.2,  6.7,
  5.2,  7.2,  5.5,  3.3,  5.8,
  5.9,  5.5,  6.1,  5.5,  5.4,
  7.1,  6.7,  6.1,  5.7,  5.5,
  6.2,  7.6,  6.2,  6.0,  4.9,
  5.9,  5.9,  6.4,  6.1,  6.2,
  5.3,  8.1,  6.9,  4.7,  6.1,
  6.2,  8.3,  4.5,  5.1,  7.0
) %>% # und pivot_longer
  pivot_longer(A:E, names_to="Schule", values_to="Punkte")
```

b) Plotten Sie die durchschnittlich erreichten Punkte pro Schule. Sind Unterschiede erkennbar?

```
# plot()
boxplot(Punkte ~ Schule, data=df)
# ggplot()
ggplot(df, aes(x=Schule, fill=Schule, y=Punkte)) +
  geom_boxplot() +
  # whiskers
  stat_boxplot(geom="errorbar")
```





c) Führen Sie eine ANOVA durch. Gibt es signifikante Unterschiede zwischen den Schulen?

```
fit <- aov(Punkte ~ Schule, data=df)
summary(fit)
```

```
             Df Sum Sq Mean Sq F value  Pr(>F)
Schule        4  15.69   3.921   5.031 0.00261 **
Residuals    35  27.28   0.779
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

d) In welcher Schule sind die sportlichen Leistungen am besten?

```
pairwise.t.test(df$Punkte, df$Schule, p.adjust="bonferroni")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  df$Punkte and df$Schule

  A       B       C       D
```

```
B 0.27923 -       -       -
C 1.00000 0.17589 -       -
D 0.36032 0.00078 0.55345 -
E 1.00000 0.33829 1.00000 0.29780

P value adjustment method: bonferroni
```

Es gibt einen Unterschied zwischen Schule B und Schule D. In Schule B sind die Leistungen besser als in Schule D. Die Leistungen in B sind aber nicht "die besten", da kein Unterschied zu den anderen Schulen gezeigt werden kann. Die Mittelwerte sind in B aber höher.

4.2.71 Lösung zur Aufgabe 4.1.11.3

Gibt es laut den Daten signifikante Unterschiede zwischen den vier Gruppen?

```r
# Erstellen des tibbles mit tribble()
df <- tribble(
  ~Kontrolle,  ~AnginaP,  ~Arrhytmia,  ~Herzinfarkt,
          83,  81,  75,  61,
          61,  65,  68,  75,
          80,  77,  80,  78,
          63,  87,  80,  80,
          67,  95,  74,  68,
          89,  89,  78,  65,
          71,  103, 69,  68,
          73,  89,  72,  69,
          70,  78,  76,  70,
          66,  83,  75,  79,
          57,  91,  69,  61
) %>% # und pivot_longer
  pivot_longer(1:4, names_to="Gruppe", values_to="Puls")

# ANOVa
fit <- aov(Puls ~ Gruppe, data=df)
summary(fit)
```

```
             Df Sum Sq Mean Sq F value  Pr(>F)
Gruppe        3   1587   529.1   8.043 0.00026 ***
Residuals    40   2631    65.8
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# paarweise
pairwise.t.test(df$Puls, df$Gruppe, p.adjust="bonferroni")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  df$Puls and df$Gruppe

            AnginaP Arrhytmia Herzinfarkt
Arrhytmia   0.01584 -         -
Herzinfarkt 0.00062 1.00000   -
Kontrolle   0.00100 1.00000   1.00000

P value adjustment method: bonferroni
```

Patienten mit Angina Pectoris unterscheiden sich von allen anderen Patientengruppen.

4.2.72 Lösung zur Aufgabe 4.1.11.4

Gibt es laut den Daten signifikante Unterschiede zwischen den drei Gruppen?

```r
# Erstellen des tibbles mit tribble()
df <- tribble(
  ~Low, ~Medium, ~High,
  36,    43,     45,
  33,    38,     39,
  35,    41,     33,
  39,    34,     39,
  41,    28,     33,
  41,    44,     26,
  44,    30,     39,
  45,    31,     29
) %>% # und pivot_longer
  pivot_longer(1:3, names_to="Konzentration", values_to="Atemfrequenz")

# ANOVa
fit <- aov(Atemfrequenz ~ Konzentration, data=df)
summary(fit)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
Konzentration  2   67.6   33.79   1.056  0.366
Residuals     21  672.2   32.01
```

```r
# paarweise
pairwise.t.test(df$Atemfrequenz, df$Konzentration, p.adjust="bonferroni")
```

```
    Pairwise comparisons using t tests with pooled SD

data:  df$Atemfrequenz and df$Konzentration

       High Low
Low    0.56 -
Medium 1.00 0.85

P value adjustment method: bonferroni
```

Es kann kein signifikanter Unterschied festgestellt werden.

4.2.73 Lösung zur Aufgabe 4.1.12.1

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Geschwuer` und `Blutgruppe`.

```r
df <- data.frame(Geschwuer=c(rep("Geschwür", 1655),
                             rep("gesund", 10000)),
                 Blutgruppe=c(rep("0", 911), rep("A", 579),
                              rep("B", 124), rep("AB", 41),
                              rep("0", 4578), rep("A", 4219),
                              rep("B", 890), rep("AB", 313))
)
```

b) Führen Sie einen Chiquadrattest auf die Hypothese durch, dass die Geschwüre von der Blutgruppe abhängig sind.

```r
chisq.test(df$Geschwuer, df$Blutgruppe)
```

```
    Pearson's Chi-squared test

data:  df$Geschwuer and df$Blutgruppe
X-squared = 49.016, df = 3, p-value = 1.295e-10
```

Es gibt unterschiede, p ist kleiner als 0,05.

c) Gibt es in Anbetracht der Ergebnisse des Vergleichs einen Zusammenhang zwischen dem Magengeschwür und der Blutgruppe? Können wir behaupten, dass der Anteil der Ulkuspatienten je nach Blutgruppe unterschiedlich ist?

```
reporttools::pairwise.fisher.test(df$Geschwuer, df$Blutgruppe,
                                  p.adjust.method = "bonferroni")
```

```
    Pairwise comparisons using

data:  df$Geschwuer and df$Blutgruppe

   0       A      AB
A  4.1e-10 -      -
AB 0.0695  1.0000 -
B  0.0023  1.0000 1.0000

P value adjustment method: bonferroni
```

Blutgruppe 0 unterscheidet sich von allen anderen.

4.2.74 Lösung zur Aufgabe 4.1.12.2

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Region` und `Blutgruppe`.

```
df <- data.frame(
          Blutgruppe=c(rep("A", 185), rep("B", 55),
                       rep("0", 223), rep("AB", 15)),
          Region=c(rep("Eskdale", 33), rep("Annadale", 54), rep("Nithsdale", 98),
                   rep("Eskdale", 6), rep("Annadale", 14), rep("Nithsdale", 35),
                   rep("Eskdale", 56), rep("Annadale", 52), rep("Nithsdale", 115),
                   rep("Eskdale", 5), rep("Annadale", 5), rep("Nithsdale", 5))
)
```

b) Führen Sie einen Chiquadrattest auf die Hypothese durch, dass die Blutgruppe von der Region abhängig sind.

```
chisq.test(df$Blutgruppe, df$Region)
```

```
Warning in chisq.test(df$Blutgruppe, df$Region): Chi-Quadrat-Approximation kann
inkorrekt sein
```

```
    Pearson's Chi-squared test

data:  df$Blutgruppe and df$Region
X-squared = 10.454, df = 6, p-value = 0.1068
```

Der Test ist nicht signifikant.

c) Gibt es in Anbetracht der Ergebnisse einen Zusammenhang zwischen der Blutgruppe und der Region? Können wir behaupten, dass die Region keinen Einfluss auf die Blutgruppe hat?

```
reporttools::pairwise.fisher.test(df$Blutgruppe, df$Region,
                                  p.adjust.method = "bonferroni")
```

```
    Pairwise comparisons using

data:  df$Blutgruppe and df$Region

          Annadale Eskdale
Eskdale   0.39     -
Nithsdale 1.00     0.10

P value adjustment method: bonferroni
```

Es sind keine Unterschiede zu finden.

4.2.75 Lösung zur Aufgabe 4.1.12.3

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `Rauchen` und `Geschlecht`.

```
df <- data.frame(
          Geschlecht=c(rep("m", 9), rep("w", 17)),
          Rauchen=c(rep("ja", 2), rep("nein", 7),
                    rep("ja", 6), rep("nein", 11))
)
```

b) Führen Sie einen Chi-Quadrat-Test durch, um festzustellen, ob das Rauchen mit dem Geschlecht zusammenhängt.

```
chisq.test(df$Rauchen, df$Geschlecht)
```

```
Warning in chisq.test(df$Rauchen, df$Geschlecht): Chi-Quadrat-Approximation
kann inkorrekt sein
```

```

    Pearson's Chi-squared test with Yates' continuity correction

data:  df$Rauchen and df$Geschlecht
X-squared = 0.057825, df = 1, p-value = 0.81
```

```
# kleines sample, besser exakten Fisher Test
fisher.test(df$Rauchen, df$Geschlecht)
```

```

    Fisher's Exact Test for Count Data

data:  df$Rauchen and df$Geschlecht
p-value = 0.6673
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.04160546 4.26600654
sample estimates:
odds ratio
  0.536523
```

c) Ist die Verteilung der Raucher bei beiden Geschlechtern gleich?

```
# kleines sample, besser exakten Fisher Test
fisher.test(df$Rauchen, df$Geschlecht)
```

```

    Fisher's Exact Test for Count Data

data:  df$Rauchen and df$Geschlecht
p-value = 0.6673
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.04160546 4.26600654
```

```
sample estimates:
odds ratio
  0.536523
```

Beide Tests sind nicht signifikant, es besteht kein Unterschied in den Gruppen.

4.2.76 Lösung zur Aufgabe 4.1.12.4

a) Übertragen Sie die Daten in ein Datenframe mit den Variablen `drug1` und `drug2`.

```
df <- data.frame(
      drug1 = c( "Ja", "Ja", "Ja", "Ja", "Ja", "Nein", "Ja", "Nein",
                 "Ja", "Ja", "Ja", "Nein", "Ja", "Nein", "Ja", "Ja",
                 "Ja", "Nein", "Ja", "Ja"),
      drug2 = c("Nein", "Nein", "Ja", "Nein", "Ja", "Ja", "Nein", "Nein",
                "Nein", "Nein", "Ja", "Nein", "Ja", "Nein", "Nein", "Ja",
                "Nein", "Ja", "Nein", "Nein")
)
```

b) Führen Sie einen McNemar-Test durch, um festzustellen, ob die Linderung mit dem Medikament zusammenhängt.

```
mcnemar.test(df$drug1, df$drug2)
```

```
    McNemar's Chi-squared test with continuity correction

data:  df$drug1 and df$drug2
McNemar's chi-squared = 4.0833, df = 1, p-value = 0.04331
```

Das Ergebnis ist signifikant. Es gibt einen Unterschied.

c) Können wir nach dem Ergebnis des Tests behaupten, dass die Linderung der Migräne vom Medikament abhängt? Wenn ja, welches Medikament bewirkt eine signifikant höhere Linderung?

```
prop.table(table(df$drug1))
```

```
  Ja Nein
0.75 0.25
```

```r
prop.table(table(df$drug2))
```

```
  Ja Nein
0.35 0.65
```

Medikament `drug1` wirkt besser.

4.2.77 Lösung zur Aufgabe 4.1.12.5

Ist ein komatöser Zustand bei der Ankunft im Krankenhaus ein Risikofaktor zu versterben?

```r
df <- data.frame(
      Ergebnis = c(rep("überlebt", 521), rep("verstorben", 207)),
      Komatös = c(rep("nein", 484), rep("ja", 37),
                  rep("nein", 118), rep("ja", 89)
  )
)

chisq.test(df$Ergebnis, df$Komatös)
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  df$Ergebnis and df$Komatös
X-squared = 130.86, df = 1, p-value < 2.2e-16
```

Ja, es gibt einen Unterschied zwischen komatösen und nicht-komatösen Patienten.

4.2.78 Lösung zur Aufgabe 4.1.12.6

Ist die Wirksamkeit der beiden Behandlungen die gleiche?

```
df <- data.frame(
      Therapie = c(rep("A", 32), rep("B", 28)),
      Wirkung = c(rep("Sehr gut", 10), rep("gut", 14), rep("schlecht", 8),
                  rep("Sehr gut", 12), rep("gut", 10), rep("schlecht", 6)
  )
)

chisq.test(df$Wirkung, df$Therapie)
```

```
    Pearson's Chi-squared test

data:  df$Wirkung and df$Therapie
X-squared = 0.87141, df = 2, p-value = 0.6468
```

Es kann kein signifikanter Unterschied gezeigt werden.

4.2.79 Lösung zur Aufgabe 4.1.12.7

Können wir dann behaupten, dass Frauen in diesem Fach erfolgreicher sind als Männer?

```
df <- data.frame(
      Geschlecht = c(rep("m", 10), rep("w", 10)),
      bestanden = c(rep("ja", 2), rep("nein", 8),
                    rep("ja", 4), rep("nein", 6)
  )
)

table(df)
```

```
          bestanden
Geschlecht ja nein
         m  2    8
         w  4    6
```

```
chisq.test(df$bestanden, df$Geschlecht)
```

```
Warning in chisq.test(df$bestanden, df$Geschlecht): Chi-Quadrat-Approximation
kann inkorrekt sein
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  df$bestanden and df$Geschlecht
X-squared = 0.2381, df = 1, p-value = 0.6256
```

Es kann kein signifikanter Unterschied gezeigt werden.

4.2.80 Lösung zur Aufgabe 4.1.12.8

Können wir bestätigen, dass es unterschiedliche Meinungen über Hans und Erna gibt?

```
df <- data.frame(
      Erna = c(rep("ja", 85), rep("nein", 65)),
      Hans = c(rep("ja", 37), rep("nein", 48),
               rep("ja", 44), rep("nein", 21)
  )
)

table(df)
```

```
      Hans
Erna   ja nein
  ja   37   48
  nein 44   21
```

```
chisq.test(df$Erna, df$Hans)
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  df$Erna and df$Hans
X-squared = 7.712, df = 1, p-value = 0.005486
```

Es gibt einen Signifikanten Unterschied zwischen den Dozenten.

```
prop.table(table(df$Erna))
```

```
       ja      nein
0.5666667 0.4333333
```

```
prop.table(table(df$Hans))
```

```
  ja nein
0.54 0.46
```

Die Studierenden mögen Erna ein bisschen mehr als Hans.

Literaturverzeichnis

[1] Grabinger B: Fit fürs Studium – Statistik. Bonn: Rheinwerk Computing, 3 edition 2024, [https://www.rheinwerk-verlag.de/fit-fuers-studium-statistik/].

[2] Isfort M, Rottländer R, Weidner F, Gehlen D, Hylla J, Tucman D: Pflege-Thermometer 2018 - Eine bundesweite Befragung von Leitungskräften zur Situation der Pflege und Patientenversorgung in der stationären Langzeitpflege in Deutschland. Köln: Deutsches Institut für angewandte Pflegeforschung e.V. (DIP) 2018.

[3] Wickham H: **Tidy Data**. Journal of Statistical Software 2014, **59**(10), [https://www.jstatsoft.org/article/view/v059i10].

[4] Kuckartz U, Rädiker S, Ebert T, Schehl J: Statistik - Eine verständliche Einführung. Wiesbaden: Springer VS, 2 edition 2013.

[5] Xie Y, Allaire J, Grolemund G: R Markdown: The Definitive Guide. Chapman and Hall 2020, [https://bookdown.org/yihui/rmarkdown/].

[6] Wickham H, Grolemund G: R for Data Science. Sebastopol, CA: O'Reilly Media 2017, [https://r4ds.had.co.nz/].

[7] große Schlarmann J: table traineR. Workouts für {data.table}. Krefeld: Hochschule Niederrhein 2024, [https://www.produnis.de/tabletrainer/].

[8] große Schlarmann J, Galatsch M: **Regressionsmodelle für ordinale Zielvariablen**. GMS Medizinische Informatik, Biometrie und Epidemiologie 2014, **10**:1–9, [https://www.egms.de/static/en/journals/mibe/2014-10/mibe000154.shtml].

[9] Cohen J: **A Power Primer**. Psychological Bulletin 1992, **112**:155–159.

[10] Wickham H: **A Layered Grammar of Graphics**. Journal of Computational and Graphical Statistics 2012, **19**:3–28.

[11] Wilkinson L, Wills D, Rope D, Norton A, Dubbs R: The Grammar of Graphics. New York: Springer, 2nd edition edition 2005.

[12] Wickham H, Navarro D, Pedersen TL: ggplot2: Elegant Graphics for Data Analysis. Springer, 3rd edition 2022, [https://ggplot2-book.org/].

[13] Gimeno EA, Garro JC, Alberca AS, Zaragoza de Lorite A: **Applied Biostatistics with R and rk.Teaching** 2022, [https://github.com/asalber/statistics_practice_rkteaching].

[14] Mitchell R, Izatt MM, Sunderland E, Cartwright R: **Blood groups antigens, plasma protein and red cell isoenzyme polymorphisms in South-west Scotland**. Annals of Human Biology 1976, **3**(2):157–171.

Stichwortverzeichnis

`[1]`, 16

Anfängerfehler, 16, 21, 32, 38, 43, 48, 68, 122, 123, 131, 135, 168, 199
ANOVA, 285, 296
arithmetisches Mittel, 215

Balkendiagramme, 384
Binomial-Verteilung, 97
Bonferroni, 296
Boxplots, 386

Chiquadrat-Verteilung, 93
    Plots erstellen, 439
    Referenztabellen, 472
Chunk, 123, 135
Cohen's `D`, 276, 277
CSV, 71, 79

data.table, 192
Datenframe, 42
Datensatz, 42
    Datensatz in Zusatzpaket, 84, 148
dbl, 147
Diagramme, 320
    Balkendiagramm (`R` base), 340
    Boxplot (`R` base), 342
    Farben, 327
    Histogramm (`R` base), 331
    Kreisdiagramm (`R` base), 335
    Legendenbox (`R` base), 347
    Liniendiagramm (`R` base), 329
    Punktwolke (`R` base), 322
    QQ-Plots (`R` base), 346
    speichern, 350
    Säulendiagramm (`R` base), 338
Dichteverteilung, 371
Dispersion, 243

Effektgröße, 276, 277

Fallzahlen
    Referenztabellen, 463
Fallzahlkalkulation, 315
    klinische Studien, 315
    Umfragen, 316
Farben, 327
fct, 147
Friedman ANOVA, 291
Funktionen programmieren, 99

ggplot, 352, 429
    Cheatsheet, 353

Histogramm, 369
Häufigkeitstabelle, 212, 416

Interquartilsabstand, 218

jgsbook, 84

Kaplan-Meier-Analysen, 298
Klassierung, 61, 188
Konfidenzintervalle, 259
    Binomialverteilung, 266
    Normalverteilung, 259
Korrelation, 220
    Pearson, 220
    Spearman, 221
Kreisdiagramme, 391
Kreuztabelle, 416
Kreuztabellen, 218

LaTeX, 139, 460, 465
lgl, 147
Liniendiagramm, 365

Markdown, 116
    quarto, 127
    RMarkdown, 117
Median, 215
Modalwert, 216

NA, 56
Normalverteilung
    Dichtefunktion, 87
    Plots erstellen, 426
    Quantile, 88
    Referenztabellen, 466
    Verteilungsfunktion, 88
    Zufallswerte, 88

Pipe, 64, 144, 191
Poisson-Verteilung, 94
    Plots erstellen, 442
Post-Hoc-Tests, 283
    Paarweise Chiquadrat, 283
    Paarweise Fisher-Test, 283
    Paarweise t-Test, 284
    Paarweise Wilcoxon-Test (Mann-Whitney-U), 284
Power, 315
    Referenztabellen, 463

Punktwolke, 363

Quartile, 216
Quartilsabstand, 218
quarto, 127, 460
    Bots, 462
    Chunk, 135
    Chunk-Optionen, 136
    Dokumentenaufbau, 132
    Editoransicht, 131
    Installation, 127
    Intext-Codeintegration, 138
    Markdown, 116
    R-Code integrieren, 135
    RStudio, 127
    Syntax, 132

R
    Anfängerfehler, 16, 21, 32, 38, 43, 48, 68, 122, 123, 131, 135, 168, 199
    Arbeitsverzeichnis, 67
    Ausgabe in Datei, 85
    CRAN, 80
    Daten laden, 66
    Daten speichern, 65
    Datenexport, 79
    Datenframes, 42
    Datenimport, 68
    Datenklassen, 29
    Datenobjete, 27
    Datentyp, 27
    Faktoren, 37
    Farben, 327
    Installation, 2
    Listen, 54
    Matrizen, 31
    NA, 56
    Nachkommastellen, 17
    Pipe, 64
    Rechenoperationen, 15
    RMarkdown, 117
    Runden, 17
    Umgang mit Datensätzen (klassich), 57
    Umgang mit Datensätzen (tidy), 160
    Variablen und Objekte, 19
    Vektor, 30
    Youtube Anleitungen, 210
    Zufallszahlen, 25
    Zusatzpakete installieren, 80
Referenztabellen, 463
Regression, 223
    Exponentialfunktion, 233
    Funktionen höherer Ordnung, 230
    generalisiert, 237
    linear, 223
    logistisch, 247
    multiple, 226
    nicht-linear, 227
    ordinal, 250
    Poisson, 239
    quadratische Funktion, 228
    Sigmoidalfunktion, 235
rename(), 160
RMarkdown
    Chunk, 123
    Syntax, 121
RStudio
    Arbeitsverzeichnis, 5
    Cheatsheets, 209, 353
    Daten importieren, 75
    Datensätze, 51
    Erscheinungsbild, 6
    Installation, 2
    Markdown, 116
    Projekte, 11
    quarto, 127
    R-Zusatzpakete installieren, 81
    Tastaturkürzel, 5
    Youtube Anleitungen, 210

Scatterplot, 363
Screeplot, 307
Signifikanztest, 275, 420
    χ^2-Test, 275
    F-Test, 278
    Kolmogorov-Smirnov-Test, 282
    Levene-Test, 279
    Mann-Whitney-U-Test, 279
    Normalverteilung, 281, 282
    Post-Hoc-Tests, 283
    t-Test, 275
    Wilcoxon-Test, 280
Skalenniveau
    nominal, 37, 227, 247
    ordinal, 37, 40, 250
Sonderzeichen, 411
Spaltensummen, 418
SPSS
    exportiere aus R, 79
    gelabelte Daten, 149
    importiere nach R, 72
    Labels, 72
Standardabweichung, 217
Säulendiagramme, 376

t-Verteilung, 91
    Plots erstellen, 436
    Referenztabellen, 470
Tibble, 145
Tidy Data, 42, 52, 141
    long table, 52, 142, 152
    wide table, 52, 141, 152, 465

Tidyverse, 141
Daten *tidy* machen, 151
Daten importieren, 149
Pipe, 144, 191
tibble, 145
Umgang mit Datensätzen, 160

Varianz, 218
Varianzanalyse, 285
ANOVA, 296
Friedman, 291
Verteilungsfunktionen, 87

Youtube Anleitungen, 210

Z-Transformation, 219
z-Transformation, 100
Zeilensummen, 418
Zufallszahlen, 25
Zusatzpakete installieren, 80
Buchpaket, 84

Überlebenszeitanalysen, 298
Cox-Regression, 305
Kaplan-Meier-Analysen, 298

Verzeichnis der verwendeten R-Befehle

.N(), 203
:=, 197, 201
?, 14

abline(), 225, 307, 323, 543, 563, 571
abs(), 17
across(), 154, 156, 416
add_count(), 179
aes(), 354
after_stat(), 450, 525, 534, 543
after_stat(count), 374
AIC(), 252, 258
annotate(), 410
anova(), 244
anova_test(), 289
aov(), 288, 296, 672, 675
apply(), 108, 422, 463
arrange(), 162
as.data.frame(), 51, 289, 422, 536
as.data.table(), 193, 204
as.Date(), 474
as.numeric(), 77, 78, 220, 244, 645
as.vector(), 53
as_factor(), 150
as_factor()(), 73, 149
as_tibble(), 151
attr(), 72
autoplot(), 300

barplot(), 338, 340, 520, 534
bind_cols(), 167
bind_rows(), 168
bmp(), 351
boxplot(), 342, 522, 529

c(), 21, 30
case_when(), 189, 423
cat(), 86
cbind(), 33, 45, 53, 151
ceiling(), 18, 244, 638
chisq.test(), 275, 420, 678, 679, 684
class(), 28, 35, 40, 43, 47, 50
coef(), 237, 251, 254, 258
cohens_d(), 290
cohensD(), 276, 277
colnames(), 35, 49, 317
colors(), 321, 327, 407, 424
compare.lm(), 595
confint(), 237, 242
coord_flip(), 385
coord_polar(), 391
coord_sf(), 456
copy(), 199
cor(), 220, 221, 307, 311, 570, 580
cor.test(), 221, 222, 569
corrplot(), 311
cortest.bartlett(), 306
cos(), 17
count(), 178
coxph(), 305
cumsum(), 212, 334, 374, 521, 543
cut(), 62, 190, 197, 523, 539, 574

data(), 82, 84
data.frame(), 42
data.table(), 194
dbinom(), 97, 616, 617, 621
dcast(), 206
dchisq(), 93, 439
describe(), 213, 306
detach(), 80
deviance(), 252, 258
df.residual(), 253
dir(), 67
dispersiontest(), 243
dnorm(), 87, 373, 628, 629
dpois(), 94, 442, 618, 619, 621
drop_na(), 177, 357
dt(), 91, 633
dunif(), 626

eigen(), 307
element_blank(), 402
everything(), 175
exp(), 17, 234, 236, 242, 248, 251, 591, 592
expand.grid(), 52, 448, 463, 611
expression(), 411
ezPlot(), 286, 292

fa.parallel(), 308
facet_grid(), 359
facet_wrap(), 359, 390, 533
factor(), 37
fct_collapse(), 188
fct_explicit_na(), 432
fct_infreq(), 182, 377
fct_lump_min(), 187
fct_lump_n(), 187

fct_recode(), 180
fct_relevel(), 186
fct_reorder(), 185
fct_rev(), 41, 183, 377
fill(), 158
filter(), 160, 474, 547
fisher.test(), 275, 681
floor(), 18
format(), 19
fread(), 193
freq(), 212
freqTable(), 213, 521, 538
friedman_effsize(), 295
friedman_test(), 294
full_join(), 175, 474
function(), 99, 111, 255, 463, 622
fwrite(), 194

gather(), 474
generate_dictionary(), 150
geom_bar(), 376, 520, 534
geom_boxplot(), 386, 522, 531
geom_col(), 376, 382, 392
geom_contour_filled(), 448
geom_freqpoly(), 525
geom_histogram(), 369, 450, 523
geom_hline(), 409, 543
geom_line(), 365, 409, 591
geom_point(), 355, 363, 561, 571
geom_rect(), 424
geom_segment(), 624, 641, 673
geom_sf(), 453, 454
geom_smooth(), 356, 563, 571
geom_text(), 383, 392, 424
geom_vline(), 409
get_anova_table(), 289
getwd(), 67
ggforest(), 305
ggplot(), 352, 429
ggqqplot(), 287, 293, 394
ggsave(), 398, 425
ggsurvplot(), 300, 303
glimpse(), 147
glm(), 237, 238, 241, 247
group_by(), 166, 177, 474, 556
gsub(), 78
guides(), 456

head(), 57, 67, 250, 574
help(), 14
hist(), 331, 523, 548

I(), 228, 590, 594
if(), 101
ifelse(), 188
inner_join(), 172
install.packages(), 80, 212, 250
installed.packages(), 102
IQR(), 218, 553
is.character(), 29
is.data.frame(), 50
is.factor(), 40
is.logical(), 29
is.matrix(), 36
is.numeric(), 29
is.vector(), 31

jpeg(), 350

KIbinomial_a(), 270
KIbinomial_u(), 274
KInormal_a(), 262
KInormal_u(), 266
KMO(), 307
ks.test(), 282
kurtosi(), 217, 555

label_to_colnames(), 150
labs(), 399, 403
lapply(), 114, 417
left_join(), 172, 455
legend(), 318, 347, 622
length(), 22, 31, 39, 212
let(), 198
levels(), 39
library(), 80
line(), 463
lines(), 318, 324, 330, 524
list(), 54, 196
lm(), 200, 223, 226, 238, 562, 566, 574, 590, 592, 599
load(), 66, 204, 239, 250, 537, 572
log(), 17, 234, 236, 590, 592, 594
log10(), 17
log2(), 17
logLik(), 252
lrtest(), 253, 257
ls(), 22, 23

matrix(), 31
mcnemar.test(), 682
mean(), 215, 553
median(), 215
melt(), 205
merge(), 46
mfv(), 216
min(), 214, 215
mutate(), 73, 149, 154, 176, 177, 188, 423, 474
mutate_if(), 150, 156

n(), 179
names(), 310
nls(), 104

nrow(), 203

order(), 59, 196, 200, 599
ordered(), 40

pairwise.fisher.test(), 283, 678
pairwise.t.test(), 284, 288, 296, 672, 675
pairwise.wilcox.test(), 284
pairwise_t_test(), 289
par(), 224, 226, 227
paste(), 64, 622
pbinom(), 97, 617, 618
pchisq(), 93, 253, 632
pf(), 635
pie(), 335
pivot_longer(), 152, 285, 437, 552
pivot_wider(), 152
plot(), 318, 320, 322, 426
png(), 350
pnorm(), 88, 467, 469, 630, 636
points(), 622
polygon(), 344
position_dodge(), 381
position_fill(), 381
position_stack(), 380
power.t.test(), 663, 667
ppois(), 94, 245, 619, 620
predict(), 226, 229, 236, 237, 244, 248, 569, 576
principal(), 313
print(), 85
prop.table(), 419, 682
prop.test(), 267, 270, 642, 643, 646, 656
pt(), 91
pull(), 176
punif(), 626, 627
pwr.t.test(), 315, 463

q(), 15
qbinom(), 97
qchisq(), 93, 473, 633
qf(), 635
qnorm(), 88, 268, 272, 631, 636
qpois(), 94
qqline(), 346
qqnorm(), 346
qqPlot(), 346
qt(), 91, 260, 261, 470, 634
quantile(), 216, 553
quit(), 15
qunif(), 627

rainbow(), 329, 339, 341
range(), 215, 553
rbind(), 33, 43, 139, 416, 639
rbinom(), 97
rchisq(), 93
read.csv(), 71, 474
read.table(), 69, 71, 76, 422
read_excel(), 73
read_ods(), 74
read_sav(), 72
read_sf(), 452
reframe(), 556
relevel(), 240
rep(), 24, 646
replace_na(), 158
reshape(), 464
residual(), 237
return(), 99
rev(), 41, 445
revalue(), 39
right_join(), 173
rm(), 23
rnorm(), 26, 88
round(), 17, 463
rownames(), 35, 50
rpois(), 94
rt(), 27, 91
runif(), 26

sample(), 25, 61, 188, 602, 603
sample.size.mean(), 316
sample.size.prop(), 316
sapply(), 113, 317
save(), 66
save.image(), 65
scale(), 219
scale_color_manual(), 407
scale_fill_discrete(), 404, 408
scale_fill_gradient2(), 456
scale_fill_hue(), 406
scale_fill_identity(), 424
scale_fill_manual(), 407, 531
scale_linetype(), 595
scale_x_continuous(), 401, 542
scale_y_continuous(), 401
sd(), 217, 553
select(), 175
sens.spec(), 611, 615
separate(), 153
seq(), 24, 467, 469, 470, 473, 629
setwd(), 68
shapiro.test(), 281, 294
sin(), 17
sink(), 85
skew(), 217, 555
slice(), 163
slice_head(), 165
slice_min(), 163
slice_sample(), 165
slice_tail(), 165
sort(), 58, 216
source(), 107

sqrt(), 15, 16
stat_bin(), 370
stat_boxplot(), 386, 522, 531
stat_count(), 376, 380
stat_density(), 372, 450
stat_function(), 92, 373, 436, 450
stat_identity(), 376, 382, 384
stat_qq(), 394
stat_qq_band(), 395
stat_qq_line(), 395
stat_qq_point(), 395
stat_summary(), 389
sticker(), 451
str(), 57
sub(), 474
subset(), 60, 464, 522, 528, 574
sum(), 49, 56, 374, 543
summ(), 239
summarise(), 177, 474
summary(), 200, 213, 537
Surv(), 298
survdiff(), 304
survfit(), 298

t(), 34, 53
t.test(), 276, 636, 649, 651
tab_model(), 242
table(), 40, 49, 212, 216, 218, 266, 417, 646, 656
tail(), 58
tan(), 17
tapply(), 115
text(), 629
theme(), 399, 402, 551
theme_classic(), 475
theme_void(), 450
tibble(), 146, 147
tiff(), 351
title(), 622
transmute(), 416
tribble(), 146, 170, 606, 609

uniqueN(), 203
unlist(), 114
url(), 67, 69, 71, 76, 239, 422, 572

var(), 218, 553
var.test(), 278, 279, 651, 665
vglm(), 250, 255

which(), 56
wilcox.test(), 279, 280
wilcox_effsize(), 295
wilcox_test(), 294
with(), 49, 61
write.csv(), 79
write.table(), 79
write_sav(), 79

xlab(), 400
xlim(), 400
xtable(), 465
xtabs(), 219, 270, 275, 417, 584

ylab(), 400
ylim(), 400



Prof. Dr. rer. medic. MScN, BScN, RN
Jörg große Schlarmann
Professor für Pflegewissenschaft
Fachbereich 10 - Gesundheitswesen
Campus Krefeld Süd
Hochschule Niederrhein
joerg.grosseschlarmann@hs-niederrhein.de
https://mastodon.social/@rbuch