

Statistik 21 – Eine praktische Einführung in moderne Datenanalyse mit R

Sebastian Sauer

20 November, 2017

Inhaltsverzeichnis

Vorwort	xix
I Rahmen	1
1 Statistik heute	3
1.1 Was ist Statistik? Wozu ist sie gut?	3
1.2 Was ist Data Science? Was ist der Unterschied zu Statistik?	5
2 Hallo, R	7
2.1 Eine kurze Geschichte von R	7
2.2 Warum R? Warum, R?	8
2.2.1 Warum R?	8
2.2.2 Warum, R?	10
2.3 Zum Weiterlesen	12
3 Arbeiten mit R	13
3.1 R und RStudio installieren	13
3.2 Pakete	15
3.2.1 Pakete von CRAN installieren	15
3.2.2 Pakete installieren vs. Pakete starten	16
3.2.3 Pakete von Github installieren	17
3.3 Hilfe! R startet nicht!	17
3.4 Zuordnung von Paketen zu Befehlen	18
3.5 R-Skript-Dateien	20
3.6 Daten	20
3.7 Projektmanagement mit RStudio	21
3.7.1 Das Arbeitsverzeichnis	22
3.7.2 RStudio-Projekte	23
3.7.3 Versionisierung	23
3.8 Hier werden Sie geholfen	23
3.8.1 Wo finde ich Hilfe?	23
3.8.2 Einfache reproduzierbare Beispiele (ERBies)	23
4 ERRRstkontakt	25

4.1	R ist pingelig	26
4.1.1	Variablen zuweisen und auslesen	27
4.1.2	Funktionen aufrufen	28
4.2	Logische Prüfungen	29
4.3	Vektorielle Funktionen	30
4.4	Literaturhinweise	31
5	Datenstrukturen	33
5.1	Zentrale Objektarten	33
5.1.1	Reine Vektoren	35
5.1.2	Faktoren	36
5.1.3	Listen	37
5.1.4	Matrizen und Arrays	37
5.1.5	Dataframes	38
5.2	Datenstruktur auslesen	39
5.2.1	Reine Vektoren	39
5.3	Matrizen und Arrays	41
5.4	Listen	41
5.5	Dataframes	43
II	Daten einlesen	45
6	Daten einlesen	47
6.1	Daten in R importieren	48
6.1.1	Excel-Dateien importieren	48
6.1.2	Daten aus R-Paketen importieren	48
6.1.3	Daten im R-Format laden	49
6.1.4	CSV-Dateien importieren	49
6.2	Normalform einer Tabelle	52
6.3	Tabelle in Normalform bringen	53
6.4	Textkodierung	55
6.5	Befehlsübersicht	55
6.6	Aufgaben	56
6.7	Verweise	56
III	Daten aufbereiten	57
7	Datenjudo	59
7.1	Typische Probleme der Datenaufbereitung	60
7.2	Daten aufbereiten mit <code>dplyr</code>	61
7.3	Zentrale Bausteine von <code>dplyr</code>	62
7.3.1	Zeilen filtern mit <code>filter</code>	62
7.3.2	Spalten wählen mit <code>select</code>	65

7.3.3	Zeilen sortieren mit <code>arrange</code>	66
7.3.4	Datensatz gruppieren mit <code>group_by</code>	69
7.3.5	Eine Spalte zusammenfassen mit <code>summarise</code>	71
7.3.6	Zeilen zählen mit <code>n</code> und <code>count</code>	74
7.4	Die Pfeife	78
7.4.1	Spalten berechnen mit <code>mutate</code>	80
7.4.2	Aufgaben	82
7.5	Deskriptive Statistik	83
7.5.1	Deskriptive Statistik mit <code>dplyr</code>	83
7.5.2	Viele Statistiken auf einmal mit <code>desctable</code>	85
7.6	Bedingte Analysen mit den Suffixen von <code>dplyr</code>	87
7.6.1	Suffix <code>_if</code>	87
7.6.2	Suffix <code>_all</code>	87
7.6.3	Suffix <code>_at</code>	88
7.7	Tabellen zusammenführen	89
7.8	Verweise	91
8	Praxisprobleme der Datenaufbereitung	93
8.1	Datenaufbereitung	93
8.1.1	Auf fehlende Werte prüfen	93
8.1.2	Fälle mit fehlenden Werte löschen	94
8.1.3	Fehlende Werte zählen	95
8.1.4	Fehlende Werte ggf. ersetzen	97
8.1.5	“-99” in NA umwandeln	98
8.1.6	Doppelte Fälle löschen	98
8.1.7	Spaltennamen ändern	99
8.1.8	Nach Fehlern suchen	99
8.1.9	Ausreißer identifizieren	100
8.1.10	Hochkorrelierte Variablen finden	101
8.1.11	z-Standardisieren	103
8.1.12	Quasi-Konstante finden	103
8.1.13	Auf Normalverteilung prüfen	104
8.1.14	Variablentypen ändern	105
8.1.15	Werte umkodieren und partionieren (“binnen”)	106
8.2	Deskriptive Statistiken berechnen	112
8.2.1	Mittelwerte pro Zeile berechnen	112
8.2.2	Mittelwerte pro Spalte berechnen	112
8.2.3	Korrelationstabellen berechnen	114
IV	Daten visualisieren	117
9	Fallstudie ‘movies’	119
9.1	Wie viele Filme gibt es pro Genre?	119
9.2	Welches Genre ist am häufigsten?	120

9.3 Zusammenhang zwischen Budget und Beurteilung	121
9.4 Wurden die Filme im Lauf der Jahre teurer und/oder “besser”?	121
10 Grundlagen der Datenvisualisierung mit ggplot2	123
10.1 Ein Bild sagt mehr als 1000 Worte	124
10.2 Die Anatomie eines Diagramms	125
10.3 Einstieg in ggplot2 - qplot	125
10.4 Häufige Arten von Diagrammen	129
10.4.1 Eine kontinuierliche Variable	129
10.4.2 Zwei kontinuierliche Variablen	133
10.4.3 Eine nominale Variable	137
10.4.4 Zwei nominale Variablen	140
10.4.5 Zusammenfassungen zeigen	141
10.4.6 Überblick zu häufigen Diagrammtypen	145
10.5 Die Gefühlswelt von ggplot2	145
10.6 ggplot() der große Bruder von qplot()	147
10.7 Aufgaben	147
10.8 Lösungen	148
10.9 Richtig oder Falsch	150
10.10 Sonstiges	152
10.11 Zum Weiterlesen	152
11 Farben wählen	153
11.1 Die Farben von Cynthia Brewer	154
11.2 Die Farben von Wes Anderson	156
11.3 Themen ändern	158
12 Fallstudie zur Visualisierung	161
12.1 Umfragedaten visualisieren mit likert	162
12.2 Umfragedaten visualisieren mit ggplot	163
12.2.1 Daten umstellen	164
12.2.2 Diagramme für Anteile	164
12.2.3 Rotierte Balkendiagramme	165
12.2.4 Text-Labels für die Items	167
12.2.5 Diagramm mit Häufigkeiten	169
12.3 Farbschemata	169
12.3.1 Diagramm beschriften	170
12.3.2 Balken mit Häufigkeitswerten	171
12.3.3 Sortieren der Balken	172
12.4 Zum Weiterlesen	173
13 Karten zeichnen	175
13.1 Kartendaten	175
13.1.1 Geo-Daten der deutschen Verwaltungsgebiete	176
13.1.2 Daten der Wahlkreise	177

13.2 Fallbeispiel: Unterschiede in den Wahlkreisen visualisieren	178
13.2.1 Karte der Wahlkreise gefärbt nach Arbeitslosigkeit	178
13.2.2 Wahlergebnisse nach Wahlkreisen	179
13.2.3 Zusammenhang von Arbeitslosigkeit und AfD-Wahlergebnis	180
13.2.4 Ein komplexeres Modell	182
13.3 Weltkarten	183
13.3.1 rworldmap	183
13.3.2 rworldmap mit geom_sf	184
13.4 Fallbeispiel: Konkordanz von Kulturwerten und Wohlbefinden	187
13.4.1 Standardisierung der Maße	187
13.5 Aufgaben	192
13.6 Weiterführende Literatur	192
V Modellieren	195
14 Grundlagen des Modellierens	197
VI Daten modellieren	199
14.1 Was ist ein Modell? Was ist Modellieren?	201
14.2 Ein Beispiel zum Modellieren in der Datenanalyse	203
14.3 Taxonomie der Ziele des Modellierens	206
14.4 Die vier Schritte des statistischen Modellierens	209
14.5 Einfache vs. komplexe Modelle: Unter- vs. Überanpassung	209
14.6 Bias-Varianz-Abwägung	210
14.7 Training- vs. Test-Stichprobe	212
14.8 Kreuzvalidierung	213
14.9 Wann welches Modell?	214
14.10 Modellgüte	214
14.10.1 Modellgüte in Regressionsmodellen	215
14.10.2 Modellgüte bei Klassifikationsmodellen	216
14.11 Auswahl von Prädiktoren	216
14.12 Aufgaben	218
14.13 Verweise	219
15 Inferenzstatistik	221
15.1 Stichprobe und Grundgesamtheit	221
15.2 Die Stichprobenverteilung	224
15.3 Der Bootstrap	228
15.4 Nyphypothesen auf Signifikanz testen	229
15.5 Der p-Wert - Nutzen und Grenzen	231
15.5.1 Was sagt der p-Wert?	231
15.5.2 Der p-Wert ist eine Funktion der Stichprobengröße	234
15.5.3 Mythen zum p-Wert	235

15.6 Wann welcher Inferenztest?	236
15.7 Vertiefung: Beispiele für häufige Inferenztests	237
15.7.1 χ^2 -Test	237
15.7.2 t-Test	238
15.7.3 Varianzanalyse	239
15.7.4 Korrelationen auf Signifikanz prüfen	240
15.7.5 Regression	240
15.7.6 Wilcoxon-Test	241
15.7.7 Kruskal-Wallis-Test	241
15.7.8 Shapiro-Test	242
15.7.9 Logistische Regression	242
15.7.10 Spearmans Korrelation	242
15.8 Zur Philosophie des p-Werts: Frequentismus	243
15.9 Alternativen zum p-Wert	244
15.9.1 Konfidenzintervalle	244
15.9.2 Effektstärke	245
15.9.3 Bayes-Statistik	248
15.10 Aufgaben	250
15.11 Fazit	251
15.12 Verweise	251
VII Geleitetes Modellieren	253
16 Lineare Regression	255
16.1 Die Idee der klassischen Regression	255
16.2 Modellgüte	258
16.2.1 Mittlere Quadratfehler	259
16.2.2 R-Quadrat (R^2)	259
16.3 Die Regression an einem Beispiel erläutert	260
16.4 Überprüfung der Annahmen der linearen Regression	262
16.5 Regression mit kategorialen Prädiktoren	265
16.5.1 Aufgaben	266
16.6 Multiple Regression	267
16.7 Interaktionen	269
16.8 Fallstudie zu Overfitting	270
16.9 Aufgaben	272
16.10 Befehlsübersicht	272
17 Klassifizierende Regression	273
17.1 Normale Regression für ein binäres Kriterium	274
17.2 Die logistische Funktion	275
17.3 Die Idee der logistischen Regression	275
17.4 Kein R^2 , dafür AIC	278
17.5 Interpretation der Koeffizienten	278

17.5.1 y-Achsenabschnitt (Intercept) β_0	278
17.5.2 Steigung β_i mit $i = 1, 2, \dots, K$	278
17.5.3 Aufgabe	279
17.6 Kategoriale Prädiktoren	279
17.7 Multiple logistische Regression	280
17.8 Modellgüte	281
17.8.1 Vier Arten von Ergebnissen einer Klassifikation	281
17.8.2 Klassifikationsgütekennzahlen	283
17.8.3 ROC-Kurven	284
17.9 Aufgaben	286
17.10 Befehlsübersicht	286
18 Fallstudien zum geleiteten Modellieren	287
18.1 Überleben auf der Titanic	287
18.1.1 Daten laden	287
18.1.2 Erster Blick	288
18.1.3 Welche Variablen sind interessant?	288
18.1.4 Univariate Häufigkeiten	288
18.1.5 Bivariate Häufigkeiten	289
18.1.6 Signifikanztest	291
18.1.7 Effektstärke	291
18.1.8 Logististische Regression	292
18.1.9 Effektstärken visualisieren	295
18.1.10 Fazit	297
18.2 Außereheliche Affären	297
18.2.1 Zentrale Statistiken	298
18.2.2 Visualisieren	299
18.2.3 Wer ist zufriedener mit der Partnerschaft: Personen mit Kindern oder ohne?	300
18.2.4 Vertiefung: Wie viele fehlende Werte gibt es?	300
18.2.5 Wer ist glücklicher: Männer oder Frauen?	301
18.2.6 Effektstärken	302
18.2.7 Korrelationen	303
18.2.8 Ehejahre und Affären	304
18.2.9 Ehezufriedenheit als Prädiktor	305
18.2.10 Weitere Prädiktoren der Affärenhäufigkeit	305
18.2.11 Unterschied zwischen den Geschlechtern	306
18.2.12 Kinderlose Ehe vs. Ehen mit Kindern	307
18.2.13 Halodries	307
18.2.14 logistische Regression	308
18.2.15 Zum Abschluss	308
18.3 Befehlsübersicht	309
19 Baumbasierte Verfahren	311
19.1 Entscheidungsbäume	312

19.1.1 Einführendes Beispiel	312
19.1.2 Tuningparameter	318
19.1.3 Entscheidungsbäume mit <code>caret</code>	318
19.1.4 Vohersagegüte	320
19.1.5 Der Algorithmus der Entscheidungsbäume	323
19.1.6 Maße der Homogenität	323
19.1.7 Regressionsbäume	325
19.1.8 Stärken und Schwächen von Bäumen	326
19.2 Bagging	327
19.3 Eine Bootstrapping-Simulation	328
19.4 Random Forests	329
19.4.1 Grundlagen	329
19.4.2 Variablenrelevanz	332
VIII Ungeleitetes Modellieren	335
20 Clusteranalyse	337
20.1 Grundlagen der Clusteranalyse	337
20.1.1 Intuitive Darstellung der Clusteranalyse	338
20.1.2 Euklidische Distanz	339
20.1.3 k-Means Clusteranalyse	343
20.2 Beispiel für eine einfache Clusteranalyse	343
20.2.1 Distanzmaße berechnen	344
20.2.2 kmeans für den Extraversionsdatensatz	344
20.3 Aufgaben	347
20.4 Befehlsübersicht	348
20.5 Verweise	348
21 Einführung in statisches Lernen mit <code>caret</code>	349
21.1 Daten aufbereiten	350
21.2 Trainings- und Test-Sample aufteilen	350
21.3 Das Modell berechnen	351
22 Grundlagen des Textmining	353
22.1 Zentrale Begriffe	354
22.2 Grundlegende Analyse	354
22.2.1 Tidy Text Dataframes	354
22.2.2 Text-Daten einlesen	357
22.2.3 Worthäufigkeiten auszählen	358
22.2.4 Visualisierung	360
22.3 Aufgaben	361
22.4 Sentiment-Analyse	362
22.4.1 Ungewichtete Sentiment-Analyse	362
22.5 Verweise	364

IX Rahmen 2	365
23 Programmieren mit R	367
23.1 Funktionen schreiben	368
23.2 Wiederholungen	371
23.2.1 Wiederholungen für Elemente eines Vektors	372
23.2.2 Wiederholungen für Spalten eines Dataframes	372
23.2.3 Dateien wiederholt einlesen	375
23.2.4 Fallbeispiele für <code>map</code>	377
23.2.5 Einige Rechtschreibregeln für <code>map()</code>	380
24 Programmieren mit dplyr and friends	381
24.1 Wie man mit dplyr nicht sprechen darf	381
24.2 Wie man Funktionen mit dplyr-Verben schreibt	382
24.3 Non-standard evaluation	385
24.4 Beispiele für NSE-Funktionen	387
24.4.1 Funktion, die einen Skalar zurückliefert	387
24.4.2 Funktion mit beliebig vielen Parametern	387
24.4.3 Funktionen für ggplot	389
A Was ist RMarkdown?	391
B Forderungen	393
C Bevor es losgeht	397
D RMarkdown in Action	399
E Die Arbeitsschritte mit RMarkdown	401
F Aufbau einer Markdown-Datei	403
G Syntax-Grundlagen von Markdown	405
H Tabellen	407
I Zitieren	409
J Kollaboration und Versionierung	411
K Verweise	413
L Hinweise	415
M Icons	417
N Voraussetzungen	419

O Zitationen	421
P Technische Details	423
Q Sonstiges	425
R Literaturverzeichnis	427

Tabellenverzeichnis

4.1 Logische Operatoren in R	30
6.1 Befehle des Kapitels 'Daten einlesen'	55
10.1 Häufige Diagrammtypen	145
14.1 Veranschaulichung der Klassifikationsgüte eines Klassifikationsmodells	217
15.1 Überblick über gängige Effektstärkemaße	247
15.2 R-Befehle für gängige Effektstärkemaße	248
16.1 Befehle des Kapitels 'Regression'	272
17.1 Vier Arten von Ergebnisse von Klassifikationen	282
17.2 Geläufige Kennwerte der Klassifikation. Anmerkungen: F: Falsch. R: Richtig. P: Positiv. N: Negativ.	283
17.3 Befehle des Kapitels 'Logistische Regression'	286
18.1 Befehle des Kapitels 'Fallstudien titanic und affairs'	310
20.1 Befehle des Kapitels 'Clusteranalyse'	348
22.1 Die häufigsten Wörter	359
22.2 Die häufigsten Wörter - mit 'stemming'	359
H.1 Eine Tabelle mit Kable	407

Abbildungsverzeichnis

1.1	Sinnbild für die Deskriptiv- und die Inferenzstatistik	3
2.1	Anzahl (neuer) R-Pakete auf CRAN	9
2.2	Beispiele für Datenvisualisierung mit R	11
2.3	Schwierigkeiten mit R	11
3.1	RStudio	14
3.2	So installiert man Pakete in RStudio	15
3.3	Hier werden Sie geholfen: Die Dokumentation der R-Pakete	19
3.4	Das Arbeitsverzeichnis mit RStudio auswählen	22
4.1	Der Prozess der Datenanalyse	25
4.2	Eine Variable definieren	27
5.1	Datenstrukturen in R; der Vektor steht im Mittelpunkt	34
6.1	Daten sauber einlesen	47
6.2	Daten einlesen (importieren) mit RStudio	48
6.3	Trennzeichen einer CSV-Datei in RStudio einstellen	52
6.4	Schematische Darstellung eines Dataframes in Normalform	52
6.5	Dieselben Daten - einmal breit, einmal lang	53
6.6	Mit 'gather' und 'spread' wechselt man von der breiten Form zur langen Form	54
6.7	Ein Beispiel für eine Abbildung zu einer Normalform-Tabelle	54
7.1	Daten aufbereiten	59
7.2	Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine	62
7.3	Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergereicht	62
7.4	Zeilen filtern	63
7.5	Spalten auswählen	65
7.6	Spalten sortieren	68
7.7	Datensätze nach Subgruppen aufteilen	69
7.8	Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'	71
7.9	Spalten zu einer Zahl zusammenfassen	72
7.10	Sinnbild für 'count'	76
7.11	Das ist keine Pfeife	78
7.12	Das 'Durchpfeifen'	79

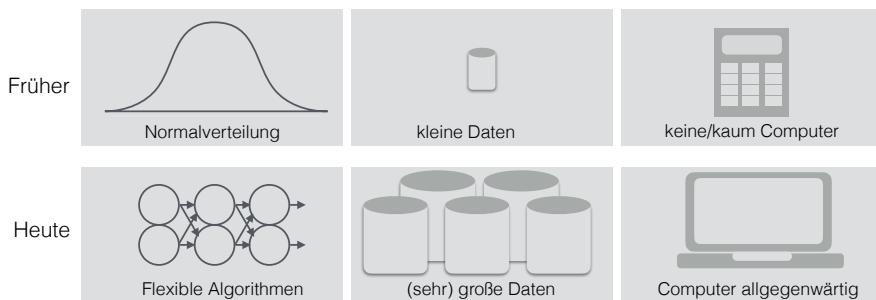
7.13 Sinnbild für mutate	81
8.1 Ausreißer identifizieren	100
8.2 Ausreißer identifizieren	101
8.3 Ein Korrelationsplot	102
8.4 Visuelles Prüfen der Normalverteilung	104
8.5 Sinnbild für Umkodieren	107
8.6 Sinnbild zum 'Binnen'	107
10.1 Das Anscombe-Quartett	124
10.2 Anatomie eines Diagramms	125
10.3 Mittleres Budget pro Jahr	126
10.4 Film-Budgets über die die Jahrzehnte	128
10.5 Verteilung des Budgets von Filmen	130
10.6 Überblick zu häufigen Diagrammtypen	146
10.7 Film-Budgets mit Histogrammen	148
11.1 GGplot mit dem Thema 'cowplot'	160
12.1 Umfrageergebnisse visualisieren mit 'likert'	163
12.2 Balkendiagramm, unrotiert und rotiert	166
12.3 Itemnummern von oben nach unten aufsteigend	167
12.4 Rotiertes Balkendiagramm mit Item-Label	168
12.5 Alternative Farbschemata	170
12.6 Balkendiagramme mit Zahlen und sortiert	173
13.1 Arbeitslosigkeit nach Wahlkreis	179
13.2 AfD-Wahlergebnisse nach Wahlkreisen	181
13.3 Ein komplexeres Erklärungsmodell zum AfD-Erfolg	182
13.4 Konkordanz von Intellektueller Autonomie und Lebenszufriedenheit	192
14.1 Ein Modell eines VW-Käfers als Prototyp eines Modells	202
14.2 Modellieren	202
14.3 Formaleres Modell des Modellierens	203
14.4 Ein Beispiel für Modellieren	204
14.5 Ein Beispiel für ein Pfadmodell	204
14.6 Ein etwas aufwändigeres Modell	205
14.7 Modelle mit schwarzer Kiste	205
14.8 Die zwei Arten des ungeleiteten Modellierens	208
14.9 Welches Modell (Teil B-D; rot, grün, blau) passt am besten zu den Daten (Teil A) ?	209
14.10 'Mittlere' Komplexität hat die beste Vorhersagegenauigkeit (am wenigsten Fehler) in der Test-Stichprobe	211
14.11 Der Spagat zwischen Verzerrung und Varianz	211
14.12 Beispiel für eine k=4 Kreuzvalidierung	213
14.13 Geringer (links) vs. hoher (rechts) Vorhersagefehler	215

14.14 Sinnbild für die Trefferquote eines Klassifikationsmodells	216
14.15 Bias-Varianz-Abwägung. Links: Wenig Bias, viel Varianz. Rechts: Viel Bias, wenig Varianz.	218
15.1 Stichproben vs. Vollerhebung	222
15.2 Die grobe Stichprobenverteilung von arrdelays	226
15.3 Nullhypthesen auf Signifikanz testen	230
15.4 Simulation von Verteilungen	232
15.5 Der größte Statistiker des 20. Jahrhunderts ($p < .05$)	232
15.6 Der p-Wert wird oft als wichtig erachtet	232
15.7 Mann und Papst zu sein ist nicht das gleiche.	234
15.8 Zwei Haupteinflüsse auf den p-Wert	235
15.9 Anteil von 'Kopf' bei wiederholtem Münzwurf	243
15.10 Die zwei Stufen der Bayes-Statistik in einem einfachen Beispieli	249
16.1 Beispiel für eine Regression	256
16.2 Zwei weitere Beispiele für Regressionen	257
16.3 Streudiagramm von Lernzeit und Klausurerfolg	261
16.4 Die Residuen verteilen sich hinreichend normal.	263
16.5 Vorhergesagte Werte vs. Residualwerte im Datensatz tips	264
16.6 Eine multivariate Analyse fördert Einsichten zu Tage, die bei einfacheren Analysen verborgen bleiben	268
16.7 Eine Regressionsanalyse mit Interaktionseffekten	270
17.1 Regressionsgerade für das Bestehen-Modell	275
17.2 Die logistische Regression beschreibt eine 's-förmige' Kurve	276
17.3 Modelldiagramm mit logistischer Regression	277
17.4 Eine ROC-Kurve	285
17.5 Beispiel für eine sehr gute (A), gute (B) und schlechte (C) Klassifikation	285
18.1 Überlebensraten auf der Titanic, in Abhängigkeit von der Passagierklasse	290
18.2 Logistische Regression zur Überlebensrate nach Passagierklasse	293
18.3 Absolute Überlebenshäufigkeiten	295
18.4 Relative Überlebenshäufigkeiten	296
18.5 Überlebenshäufigkeiten anhand eines Fliesen-Diagramms dargestellt	297
18.6 Affären, mit Jitter	309
18.7 Affären, mit Smooth	310
19.1 Ein einfacher Entscheidungsbaum, der die Stichprobe in zwei Gruppen aufteilt	313
19.2 Alternative Visualisierungen für Entscheidungsbäume	315
19.3 Ein komplexerer Entscheidungsbaum	315
19.4 Vergleich der Kreuzvalidierungsergebnisse für das Affären-Modell	320
19.5 Klassifikationsfehler und Gini-Koeffizient kommen manchmal zu unterschiedlichen Entscheidungen	325
19.6 Vergleich von linearen Modellen und Entscheidungsbäumen	326
19.7 Klassifikationsgüte in Abhängigkeit von der Anzahl der Knoten im Baum	327

20.1 Ein Streudiagramm - sehen Sie Gruppen (Cluster) ?	338
20.2 Ein Streudiagramm - mit drei Clustern	338
20.3 Unterschiedliche Anzahlen von Clustern im Vergleich	339
20.4 Die Summe der Varianz within in Abhangigkeit von der Anzahl von Clustern. Ein Screeplot.	340
20.5 Distanz zwischen zwei Punkten in der Ebene	340
20.6 Pythagoras in 3D	341
20.7 Pythagoras in Reihe geschaltet	342
20.8 Schematische Darstellung zweier einfacher Clusterlosungen; links: geringe Varianz innerhalb der Cluster; rechts: hohe Varianz innerhalb der Cluster . .	346
22.1 Illustration eines Tidy Text Dataframe	355
22.2 Eine Wordwolke zum AfD-Parteiprogramm	360
22.3 Worthaufigkeiten im AfD-Parteiprogramm	361
23.1 Daten aufbereiten	367
23.2 Funktionen definieren	368
23.3 Die Funktion f wird auf jedes Element i von E angewendet	371
23.4 Multiple t-Tests und deren p-Werte	379
24.1 Der Unterschied zwischen Ausfuhren und Zitieren	386
C.1 Die zwei Arbeitsschritte bei der Konvertierung von RMarkdown-Dokumenten	398
D.1 Neue Rmd-Datei erstellen	399
D.2 Wir stricken	400
D.3 HTML-Datei aus Rmd-Datei erstellt	400
E.1 Die Arbeitsschritte mit RMarkdown	401

Vorwort

Statistik heute; was ist das? Sicherlich haben sich die Schwerpunkte von “gestern” zu “heute” verschoben. Wenig überraschend spielt der Computer eine immer größere Rolle; die Daten werden vielseitiger und massiger. Entsprechend sind neue Verfahren nötig - und vorhanden, in Teilen - um auf diese neue Situation einzugehen. Einige Verfahren werden daher weniger wichtig, z.B. der p-Wert oder der t-Test. Allerdings wird vielfach, zumeist, noch die Verfahren gelehrt und verwendet, die für die erste Hälfte des 20. Jahrhunderts entwickelt wurden. Eine Zeit, in der kleine Daten, ohne Hilfe von Computern und basierend auf einer kleinen Theoriefamilie im Rampenlicht standen (Cobb 2007). Die Zeiten haben sich geändert!



Zu Themen, die heute zu den dynamischsten Gebieten der Datenanalyse gehören, die aber früher keine große Rolle spielten, gehören (Hardin u. a. 2015):

- Nutzung von Datenbanken und anderen Data Warehouses
- Daten aus dem Internet automatisch einlesen (“scraping”)
- Genanalysen mit Tausenden von Variablen
- Gesichtserkennung

Sie werden in diesem Kurs einige praktische Aspekte der modernen Datenanalyse lernen. Ziel ist es, Sie - in Grundzügen - mit der Art und Weise vertraut zu machen, wie angewandte Statistik bei führenden Organisationen und Praktikern verwendet wird¹.

Es ist ein Grundlagenkurs; das didaktische Konzept beruht auf einem induktiven, intuitiven Lehr-Lern-Ansatz. Formeln und mathematische Hintergründe sucht man meist vergebens (tja).

Im Gegensatz zu anderen Statistik-Büchern steht hier die Umsetzung mit R stark im Vordergrund. Dies hat pragmatische Gründe: Möchte man Daten einer statistischen Analyse

¹Statistiker, die dabei als Vorbild Pate standen sind: Roger D. Peng: <http://www.biostat.jhsph.edu/~rpeng/>, Hadley Wickham: <http://hadley.nz>, Jennifer Bryan: <https://github.com/jennybc>

unterziehen, so muss man sie zumeist erst aufbereiten; oft mühselig aufbereiten. Selten kann man den Luxus genießen, einfach “nur”, nach Herzenslust sozusagen, ein Feuerwerk an multivariater Statistik abzubrennen. Zuvor gilt es, die Daten aufzubereiten, umzuformen, zu prüfen und zusammenzufassen. Diesem Teil ist hier recht ausführlich Rechnung getragen.

“Statistical thinking” sollte, so eine verbreitete Idee, im Zentrum oder als Ziel einer Statistik-Ausbildung stehen (Wild und Pfannkuch 1999). Es ist die Hoffnung der Autoren dieses Skripts, dass das praktische Arbeiten (im Gegensatz zu einer theoretischen Fokus) zur Entwicklung einer Kompetenz im statistischen Denken beiträgt.

Außerdem spielt in diesem Kurs die Visualisierung von Daten eine große Rolle. Zum einen könnte der Grund einfach sein, dass Diagramme ansprechen und gefallen (einigen Menschen). Zum anderen bieten Diagramme bei umfangreichen Daten Einsichten, die sonst leicht wortwörtlich überersehen würden.



Nach der Lektüre dieses Buches können Sie:

- den Ablauf eines Projekts aus der Datenanalyse in wesentlichen Schritten nachvollziehen,
- Daten aufbereiten und ansprechend visualisieren,
- Inferenzstatistik anwenden und kritisch hinterfragen,
- klassische Vorhersagemethoden (Regression) anwenden,
- moderne Methoden der angewandten Datenanalyse anwenden (z.B. Textmining),
- betriebswirtschaftliche Fragestellungen mittels datengetriebener Vorhersagemodelle beantworten.

Besondere Vorerfahrung wird nicht vorausgesetzt.

Dieses Buch zielt auf die praktischen Aspekte der Analyse von Daten ab: “wie mache ich es?”; mathematische und philosophische Hintergründe werden vernachlässigt bzw. auf einschlägige Literatur verwiesen.

Sebastian Sauer

Teil I

Rahmen

Kapitel 1

Statistik heute

1.1 Was ist Statistik? Wozu ist sie gut?

Zwei Fragen bieten sich am Anfang der Beschäftigung mit jedem Thema an: Was ist die Essenz des Themas? Warum ist das Thema (oder die Beschäftigung damit) wichtig?

Was ist Statistik? Eine Antwort dazu ist, dass Statistik die Wissenschaft von Sammlung, Analyse, Interpretation und Kommunikation von Daten ist mithilfe mathematischer Verfahren ist und zur Entscheidungshilfe beitragen solle (*The Oxford Dictionary of Statistical Terms 2006*; Romeijn 2016). Damit hätten wir auch den Unterschied zur schnöden Datenanalyse (ein Teil der Statistik) herausgemeißelt. Statistik wird häufig in die zwei Gebiete *deskriptive* und *inferierende* Statistik eingeteilt (vgl. Abb. 1.1). Erstere fasst viele Zahlen zusammen, so dass wir den Wald statt vieler Bäume sehen. Letztere verallgemeinert von den vorliegenden (sog. “Stichproben-”)Daten auf eine zugrunde liegende Grundmenge (Population). Dabei spielt die Wahrscheinlichkeitsrechnung (Stochastik) eine große Rolle.

Aufgabe der deskriptiven Statistik ist es primär, Daten prägnant zusammenzufassen. Aufgabe

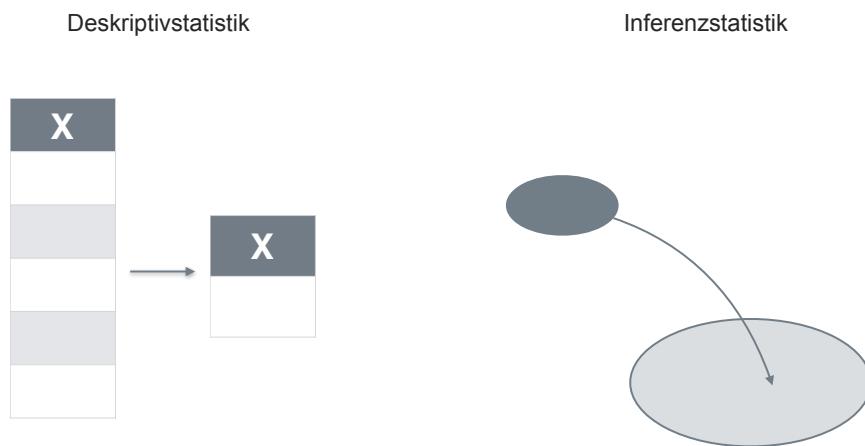


Abbildung 1.1: Sinnbild für die Deskriktiv- und die Inferenzstatistik

der Inferenzstatistik ist es, zu prüfen, ob Daten einer Stichprobe auf eine Grundgesamtheit verallgemeinert werden können.

Dabei lässt sich der Begriff “Statistik” als Überbegriff von “Datenanalyse” verstehen, wenn diese Sicht auch nicht von allen geteilt wird (Gromelund und Wickham 2014). In diesem Buch steht die Aufbereitung, Analyse, Interpretation und Kommunikation von Daten im Vordergrund. Liegt der Schwerpunkt dieser Aktivitäten bei computerintensiven Methoden, so wird auch von *Data Science* gesprochen, wobei der Begriff nicht einheitlich verwendet wird (Wickham und Gromelund 2016; Hardin u. a. 2015)

Daten kann man definieren als *Informationen, die in einem Kontext stehen* (Moore 1990), wobei eine numerische Konnotation mitschwingt.

Modellieren kann man als *zentrale Aufgabe von Statistik* begreifen (Cobb 2007; Gromelund und Wickham 2014). Einfach gesprochen, bedeutet Modellieren in diesem Sinne, ein mathematisches Narrativ (“Geschichte”) zu finden, welches als Erklärung für gewisse Muster in den Daten fungiert; vgl. Kap. 14.

Statistisches Modellieren läuft gewöhnlich nach folgendem Muster ab (Gromelund und Wickham 2014):

Prämissen 1: Wenn Modell M wahr ist, dann sollten die Daten das Muster D aufweisen.

Prämissen 2: Die Daten weisen das Muster D auf.

Konklusion: Daher muss das Modell M wahr sein.

Die Konklusion ist *nicht* zwangsläufig richtig. Es ist falsch zu sagen, dass dieses Argumentationsmuster - Abduktion (Peirce 1955) genannt - wahre, sichere Schlüsse (Konklusionen) liefert. Die Konklusion *kann, muss aber nicht*, zutreffen.

Ein Beispiel: Auf dem Nachhauseweg eines langen Arbeitstags wartet, in einer dunklen Ecke, ein Mann, der sich als Statistik-Professor vorstellt und Sie zu einem Glücksspiel einlädt. Sofort sagen Sie zu. Der Statistiker will 10 Mal eine Münze werfen, er setzt auf Zahl (versteht sich). Wenn er gewinnt, bekommt er 10€ von Ihnen; gewinnen Sie, bekommen Sie 11€ von ihm. Hört sich gut an, oder? Nun wirft er die Münze zehn Mal. Was passiert? Er gewinnt 10 Mal, natürlich (so will es die Geschichte). Sollten wir glauben, dass er ein Betrüger ist?

Ein Modell, welches wir hier verwenden könnten, lautet: Wenn die Münze gezinkt ist (Modell M zutrifft), dann wäre diese Datenlage D (10 von 10 Treffern) wahrscheinlich - Prämisse 1. Datenlage D ist tatsächlich der Fall; der Statistiker hat 10 von 10 Treffer erzielt - Prämisse 2. Die Daten D “passen” also zum Modell M; man entscheidet sich, dass der Professor ein Falschspieler ist.

Wichtig zu erkennen ist, dass Abduktion mit dem Wörtchen *wenn* beginnt. Also davon *ausgeht*, dass ein Modell M der Fall ist (der Professor also tatsächlich ein Betrüger ist). Das, worüber wir entscheiden wollen, wird bereits vorausgesetzt. Falls M gilt, gehen wir mal davon aus, wie gut passen dann die Daten dazu?

Wie gut passen die Daten D zum Modell M?

Das ist die Frage, die hier tatsächlich gestellt bzw. beantwortet wird.

Natürlich ist es keineswegs sicher, *dass* das Modell gilt. Darauf macht die Abduktion auch keine Aussage. Es könnte also sein, dass ein anderes Modell zutrifft: Der Professor könnte ein Heiliger sein, der uns auf etwas merkwürdige Art versucht, Geld zuzuschanzen... Oder er hat einfach Glück gehabt.

Statistische Modelle beantworten i.d.R. nicht, wie wahrscheinlich es ist, dass ein Modell gilt. Statistische Modelle beurteilen, wie gut Daten zu einem Modell passen.

Häufig trifft ein Modell eine Reihe von Annahmen, die nicht immer explizit gemacht werden, aber die klar sein sollten. Z.B. sind die Münzwürfe unabhängig voneinander? Oder kann es sein, dass sich die Münze "einschießt" auf eine Seite? Dann wären die Münzwürfe nicht unabhängig voneinander. In diesem Fall klingt das reichlich unplausibel; in anderen Fällen kann dies eher der Fall sein[^447]. Auch wenn die Münzwürfe unabhängig voneinander sind, ist die Wahrscheinlichkeit für Zahl jedes Mal gleich? Hier ist es wiederum unwahrscheinlich, dass sich die Münze verändert, ihre Masse verlagert, so dass eine Seite Unwucht bekommt. In anderen Situationen können sich Untersuchungsobjekte verändern (Menschen lernen manchmal etwas, sagt man), so dass die Wahrscheinlichkeiten für ein Ereignis unterschiedlich sein können, man dies aber nicht berücksichtigt.

1.2 Was ist Data Science? Was ist der Unterschied zu Statistik?

Kapitel 2

Hallo, R

2.1 Eine kurze Geschichte von R

Was ist R? Wo kommt es (er?sie?) her? Und warum dieser komische Name? Was R ist, ist einfach zu beantworten: Ein Dialekt der Programmiersprache S. S wiederum wurde von John Chambers und anderen bei der Firma AT&T entwickelt, beginnend im Jahre 1976, schon etwas her (Peng 2014). Diese Software wurde einige Male hin und her verkauft und erweitert, schließlich unter dem Namen S-Plus. Das Besondere an S-Plus ist, dass es seine Wurzeln nicht in der typischen Programmierung, sondern in der Datenanalyse hat. S-Plus sollte eine *interaktive Umgebung* bieten, so John Chambers, in der der Nutzer schnell und unkompliziert einige Analysen ausführen kann. Das ist sicher der Grund, warum man in R genauso wie in S-Plus einzeln Zeilen direkt ausführen kann (ein “Interpreter”), anstatt ein komplettes Programm schreiben zu müssen, was nur als Ganzes in Computersprache übersetzt (“kompiliert”) und ausgeführt werden kann (ein “Compiler”). Allerdings soll S-Plus gleichzeitig die Möglichkeit bieten, größere Programme zu schreiben und alle wichtigen Eigenschaften einer vollwertigen Programmiersprache aufweisen, so dass fortgeschrittene Nutzer anspruchsvollen Code schreiben können. Man kann R also sowohl als *Programmiersprache* oder (auch) als *interaktive Analyseumgebung* bezeichnen.

Ein großer Nachteil von S ist, dass es kommerziell vertrieben ist, was eine Hemmschwelle für Nutzer darstellt. 1991 entwickelten Ross Ihaka und Robert Gentleman von der Uni Auckland in Neuseeland R (Ihaka und Gentleman 1996); R wurde explizit für Datenanalyse und Datenvizualisierung entwickelt. R ist frei¹! Frei wie in Bier und frei wie in Freiheit². Die Syntax von R ist immer noch sehr ähnlich zu S-Plus, allerdings hat sich R stark entwickelt seit seiner Geburt in den 1990er Jahren. Die Freiheit von R ist eine große Stärke: Alle, die sich berufen fühlen, können die Funktionalität von R erweitern. Wenn Sie eine geniale Idee für eine neue Methode der Statistik haben, nurzu, Sie können Sie einfach für die ganze Welt verfügbar machen. Und die ganze Welt kann Ihre Idee einfach nutzen. Das führt daszu, dass

¹hier finden Sie Details, was *freie Software* bedeutet: <http://www.fsf.org/>

²GNU-Lizenz; [https://en.wikipedia.org/wiki/R_\(programming_language\)](https://en.wikipedia.org/wiki/R_(programming_language))

es eine unglaublich reichhaltige Fülle an Erweiterungen für R gibt (sog. *Pakete*), um eben jene neuen Ideen, wie genial auch immer, für Ihre Analysen nutzbar zu machen.

R ist für alle gängigen Betriebssysteme verfügbar (Windows, Mac, Linux) und steht an einem zentralen Ort, dem *Comprehensive R Archive Network*³, kurz CRAN, zum Download bereit. Dort findet man auch weitere Informationen, wie häufige Fragen oder Anleitungen und eine große Zahl an Erweiterungen für R. Genauer gesagt, beinhaltet das “Standard-R”, wie man es sich von CRAN herunterlädt, schon ein paar Pakete, die sozusagen von Haus aus mitgeliefert werden. Dazu zählt das Paket **base**, das grundlegende Funktionen enthält und noch gut zwei Dutzend weitere⁴.

Wie jedes Ding, oder wie jede Software, hat R auch Schwächen. Der R-Kern ist vergleichsweise alt und einige Erblasten werden aus Kompatibilitätsgründen mitgeschleppt. Aber grundlegende Funktionsweisen von R zu ändern könnte dazu führen, dass einfacher, Standard-R-Code auf einmal nicht mehr funktioniert. Wer weiß, was dann passiert? Studierende lernen dann vielleicht Falsches zum p-Wert, mein Download bricht vorzeitig ab und Ihr Aktien-Forecast macht Sie doch nicht zum Millionär... Nicht auszudenken. Das ist der Grund, warum Neuerungen bei R nicht im “Kern”, in Standard-R, sondern in den Paketen von sich gehen. R wurde (und wird) eher von Statistik-Freunden denn von waschechten Programmieren entwickelt und genutzt; das hat R den Vorwurf eingebracht, nicht so effizient und breit nutzbar zu sein, wie andere Sprachen. Tatsächlich ist in Kreisen, die eher aus der Informatik stammen, die Programmiersprache *Python*⁵ verbreiteter. In einigen Foren tobt die Diskussion, welche von beiden Sprachen den Größten Nutzerkreis habe, am coolsten sei und so weiter. Python hat auch einige Vorteile, die hier aus Gründen der Befangenheit begrenzten Seitenzahl nicht ausgeführt. Es gibt auch einige Neuentwicklungen, zum Beispiel die Programmiersprache *Julia*⁶, die schneller rechnet als R. Schwer zu sagen, welche Programmiersprache in ein paar Jahren der Platzhirsch. Vorhersagen, sind bekanntlich schwierig, gerade wenn sie die Zukunft betreffen. Aber gut möglich, dass ein so dynamisches Feld wie *Data Science* genug Platz für mehrere Ökosysteme bietet.

2.2 Warum R? Warum, R?

2.2.1 Warum R?

Warum sollte man Daten mit *R* analysieren und nicht mit ... Excel, zum Beispiel? Vielleicht ist die schärfere Unterscheidung zu fragen, ob man eine *Programmiersprache* (wie R) verwendet oder eine “klickbare Oberfläche” (wie R). Für beide Kategorien gibt es einige Vertreter und die im Folgenden aufgeführten Vor- und Nachteile gelten für Programmiersprache (wie R aber auch Python oder Julia) bzw. klickbare Programme (wie Excel oder SPSS). Wenn es auch hier vertreten wird, dass eine Programmiersprache wie R für ernsthafte Datenanalyse besser

³<https://cran.r-project.org/>

⁴<https://stat.ethz.ch/R-manual/R-devel/doc/html/packages.html>

⁵<https://www.python.org/>

⁶<https://julialang.org/>

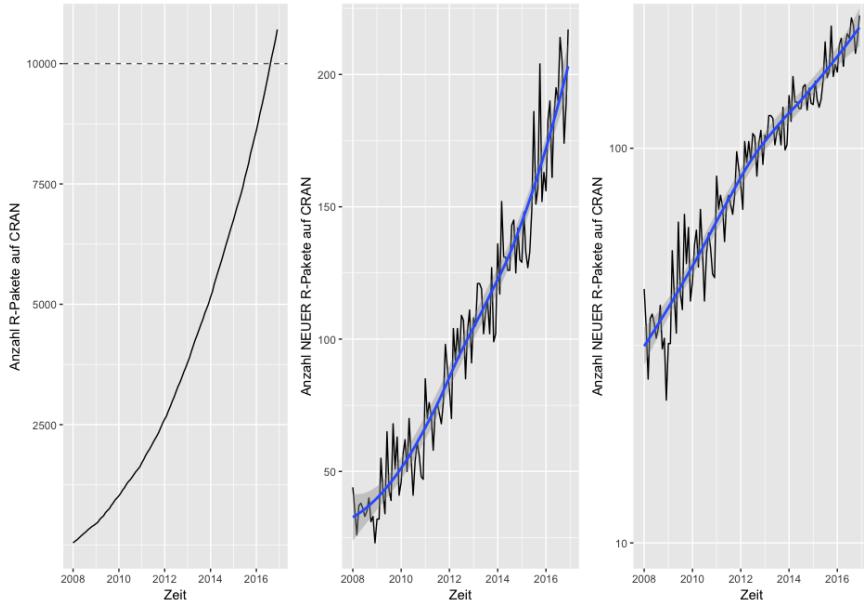


Abbildung 2.1: Anzahl (neuer) R-Pakete auf CRAN

geeignet ist, so ist die Wahl der Programmiersprache (für Datenanalyse) zweitrangig. Neben R ist Python auch sehr stark verbreitet, allerdings ist R für einen Einstieg sehr gut geeignet: Eine aktuelle Analyse basierend auf Textmining von Posts bei www.stackoverflow.com zeigt, dass R die am meisten geschätzte Programmiersprache ist (Robinson 2017).

R ist beliebt und bleibt, was die Anzahl von “Paketen” angeht, also Erweiterungen im Sinne von neuen Funktionen von R. Mittlerweile gibt es mehr als 10000 R-Pakete; Tendenz steigend. Abbildung 2.1 zeigt im linken Teil die Gesamtanzahl von R-Paketen auf CRAN, dem zentralen Ablageort (Repository; eine Art “Webstore”) für R-Pakete (Rickert 2017a). Der mittlere und rechte Teil zeigt die Anzahl der neuen Pakete pro Monat; der rechte Teil des Diagrammes zeigt den Wachstum mit einer logarithmischen Achse. Da die Steigung recht linear aussieht bei dieser logarithmischen Achse, deutet es auf eine exponentielle Steigung hin.

1. R kennt die modernen Verfahren - Excel nicht Moderne Verfahren der Datenanalyse, speziell aus der *prädiktiven Modellierung* sind in R sehr gut vertreten. In Excel nicht oder viel schwächer. So beinhaltet das R-Paket **caret** aktuell 238 verschiedene prädiktive Modelle - Regression ist (nur) eines davon. Diese Vielfalt gibt es in Excel nicht. Außerdem sind die Verfahren, die es in Excel gibt, alt. In R können neue Verfahren ohne Zeitverlust der Allgemein zur Verfügung gestellt werden. Daher ist man am Puls der Zeit, wenn man mit R arbeitet. In Excel nicht.
2. R ist reproduzierbar - Excel nicht In R ist der “Rohstoff” (Daten) von den “Verarbeitungsschritten” (Analyseverfahren wie Mittelwert bilden) *getrennt*; in Excel *vermengt*. In Excel sind Daten und Analysebefehle in einer Tabelle vermengt; teilweise sogar in einer Zelle. Das macht das Vorgehen intransparent; in welcher Zelle hatte ich noch mal den 78. Arbeitsschritt geschrieben? Und war es eigentlich der 78. oder doch der

79. Schritt? In dieser vermengten Form ist es schwierig, die Analyse zu dokumentieren. Genauso schwierig ist es, Fehler zu finden. Komplexe Exceltabellen an Andere (inklusive Ihr zukünftiges Ich) ist daher problematisch. In jüngerer Zeit sind einige Fehler in Datenanalysen bekannt gewordne, die auf dier Verwendung von Excel beruhen. So berichten Ziemann, Eren, und El-Osta (2016), dass etwa ein Fünftel von Fachartikeln in führenden Zeitschriften Fehler enthalten, die von Excel erzeugt wurden. So wurden z.B. Gennamen wie “Septin 2” von Excel unaufgefordert in ein Datum (September) umformatiert; damit wurde die eigentliche Information zerstört. Solche Fehler sind schwer zu finden, wenn Daten und Analyseverfahren vermenkt sind, wie in Excel. Aktuell wird verstärkt diskutiert über Forschungsergebnisse, die sich nicht bestätigen lassen (2015; Begley und Ioannidis 2015); alarmierende und schockierende⁷ Neuigkeiten. Wenn schon in der Wissenschaft, wo die strengsten Maßstäbe an Sorgfalt und Strenge angelegt werden, so viele Fehler auftauchen, wie mag es an in der “Praxis” aussehen, wo häufig “pragmatischer” vorgegangen wird? Die Möglichkeit, einen Analyse effizient nachzuvollziehen, ist daher zentral. *Reproduzierbarkeit* ist daher eine zentrale Forderung zur Sicherung der Qualität einer Datenanalyse. Mit R ist das ohne Weiteres möglich; mit Excel ist es schwierig.

3. R ist automatisierbar - Excel nicht oder weniger Hat man eine Analyse “verschriftlicht”, also ein Skript geschrieben, in dem alle Befehle niedergeschrieben sind, kann man dieses Skript starten und die Analyse wird automatisch ausgeführt. Ähnliches ist mit Excel deutlich schwieriger. Problemlos kann man in R die Analyse parametrisieren, also z.B. sagen “Hey, führe die Analyse mit den Geschäftszahlen letzter Woche durch und nimm das Stylesheet ‘mightyDonald’ ”. In Excel? Schwierig. Da R-Skripte Textdateien sind, kann man sie mit Versionierungswerkzeugen wit *Git*
4. R ist frei: quelloffen und kostenlos - Excel nicht möchten wissen, wie eine bestimmte Berechnung genau durchgeführt wird? In R kein Problem. Sie möchten diese Berechnung ändern? In R kein Problem. Außerdem möchten Sie dafür nichts zahlen und es an interessiertes Fachpublikum weitergeben? In R kein Problem. R ist ein Computerprogramm, dass von Freiwilligen gepflegt und weiterentwickelt wird. Jeder kann mitmachen. Kostet nix. Manch kommerzielles Programme für Datenanalyse hingegen ist *sehr* teuer.
5. R erstellt elegante Diagramme - Excel nicht

Abbildung 2.2 zeigt Beispiele für hochwertige Abbildungen, die mit R erstellt wurden (links: Ein Circlize-Plot mit dem R-Paket `circlize`; Zuguang (2017); rechts: Kartenmaterial visualisiert; Kashnitsky (2017a))

2.2.2 Warum, R?

Die Kehrseite von R ist die aufwändiger Einarbeitung. Es dauert länger mit R als mit Excel, bis man die Grundlagen beherrscht, also einfache Arbeitsschritte selber volbringen kann. Die Lernkurve ist zu Beginn steiler als bei der Arbeit mit Excel (s. Abbildung 2.3, linke

⁷überraschend vielleicht nur für die, die nicht in der Forschung arbeiten

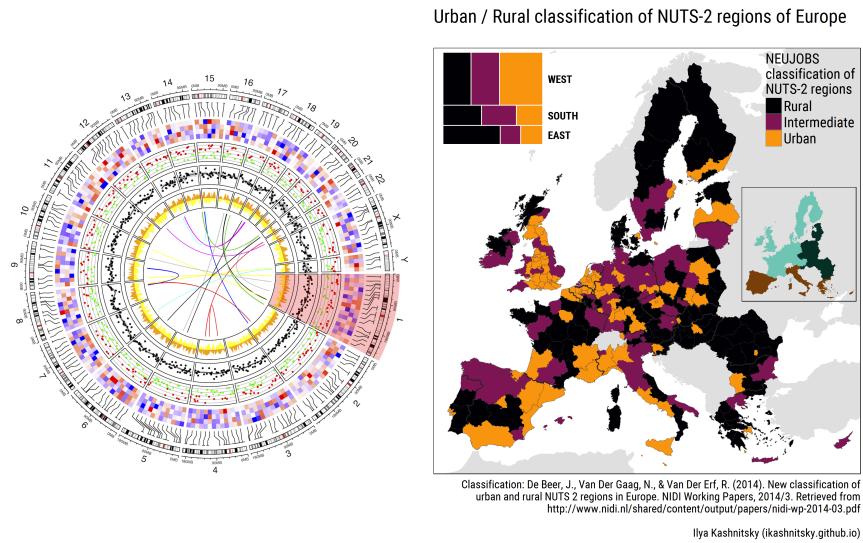


Abbildung 2.2: Beispiele für Datenvisualisierung mit R

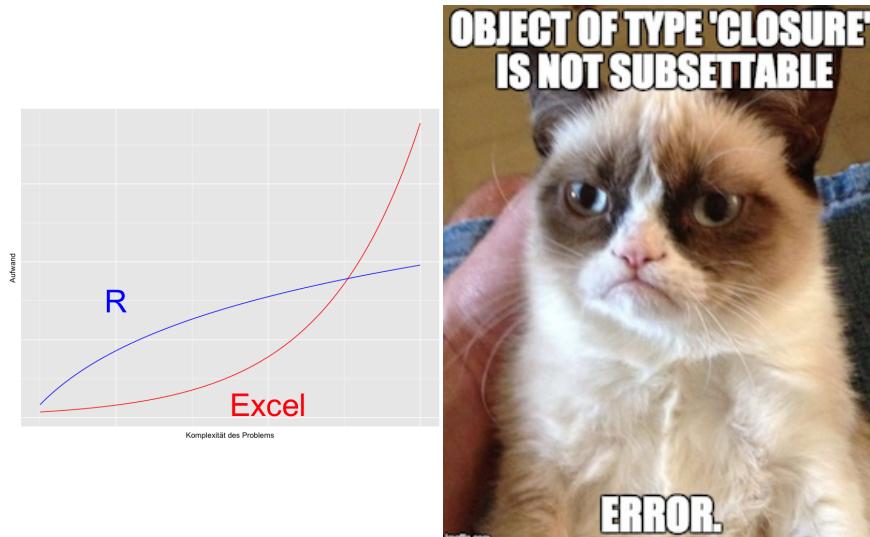


Abbildung 2.3: Schwierigkeiten mit R

Seite). Bei zunehmender Komplexität wird der Aufwand, ein Problem zu lösen mit R aber geringer, geringer schließlich als mit Excel. Das liegt daran, dass, wie oben gesagt, komplexe Analyseverfahren in Excel nicht oder nur umständlich verfügbar sein.

Eine andere Eigenart einer Programmiersprache kann bisweilen für Frust sorgen: Man hackt ein paar geniale Befehle ein, freut sich, erwartet Wunderbares und es passiert ... nichts. Oder es kommt eine Fehlermeldung, zumeist präzise, verständlich und konstruktiv. Ein Klassiker ist “Object of type closure is not subsettable” (s. Abbildung 2.3, rechte Seite⁸). Diese Fehlermeldung tritt dann auf, wenn man versucht, eine Funktion (closure) zu indizieren (s. Abschnitt 5.2).

⁸<https://imgflip.com/i/1z7avz>

2.3 Zum Weiterlesen

Eine Apologie von Schwierigkeiten beim Lernens von R, angelehnt an Dantes Göttliche Komödie findet sich bei Burns (2012). Wer es formal mag, kann sich mit der R-Language-Definition vertraut machen, in der zentrale Punkte von GNU-R erläutert werden (??); dieser Text ist aber für fortgeschrittene Nutzer gedacht. Die vielleicht beste und tiefste Erläuterung der R-Sprache findet sich bei Wickham (??), dort wird alles was man braucht um R tief zu verstehen. Es ist allerdings auch ein Text für hoch motivierte Einsteiger oder fortgeschrittene Nutzer, die ein tiefes Verständnis erwerben wollen. Ein guter Einstieg in die Datenanalyse mit R findet sich bei Ligges (2008). Alternativ bietet Luhmann (2015) einen Einstieg, der Leser aus den Sozialwissenschaften im Blick hat.

Kapitel 3

Arbeiten mit R

Als Haupt-Analysewerkzeug nutzen wir R; daneben wird uns die sog. “Entwicklungsumgebung” RStudio einiges an komfortabler Funktionalität bescheren. Eine Reihe von R-Paketen (“Packages”; d.h. Erweiterungen) werden wir auch nutzen. R ist eine recht alte Sprache; viele Neuerungen finden in Paketen Niederschlag, da der “harte Kern” von R lieber nicht so stark geändert wird. Stellen Sie sich vor: Seit 29 Jahren nutzen Sie eine Befehl, der Ihnen einen Mittelwert ausrechnet, sagen wir die mittlere Anzahl von Tassen Kaffee am Tag. Und auf einmal wird der Mittelwert anders berechnet?! Eine Welt stürzt ein! Naja, vielleicht nicht ganz so tragisch in dem Beispiel, aber grundsätzlich sind Änderungen in viel benutzen Befehlen potenziell problematisch. Das ist wohl ein Grund, warum sich am “R-Kern” nicht so viel ändert. Die Innovationen in R passieren in den Paketen. Und es gibt viele davon; als ich diese Zeilen schreibe, sind es fast schon 10.000! Genauer: 9937 nach dieser Quelle: <https://cran.r-project.org/web/packages/>. Übrigens können R-Pakete auch Daten enthalten.

3.1 R und RStudio installieren



Sie können R unter <https://cran.r-project.org> herunterladen und installieren (für Windows, Mac oder Linux). RStudio finden Sie auf der gleichnamigen Homepage: <https://www.rstudio.com>; laden Sie die “Desktop-Version” für Ihr Betriebssystem herunter.

RStudio ist, vereinfacht gesagt, “nur” eine Oberfläche (“GUI”) für R, mit einer R von praktischen Zusatzfunktionen¹. Die eigentlich Arbeit verrichtet das “normale” R, welches automatisch gestartet wird, wenn Sie RStudio starten (sofern R installiert ist). Die Oberfläche

¹ehrlicherweise ist das schamlos untertrieben; RStudio kann viel. Aber für den Anfang ist es eine nützliche Vorstellung.

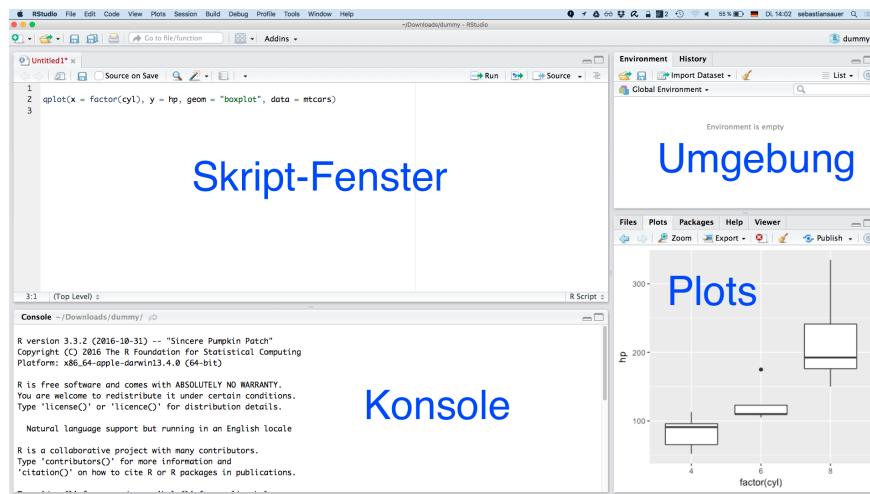


Abbildung 3.1: RStudio

von RStudio sieht (unter allen Betriebssystemen etwa gleich) so aus wie in Abbildung 3.1 dargestellt.

Das *Skript-Fenster* ähnelt einem normalem Text-Editor; praktischerweise finden Sie aber einen Button “run”, der die aktuelle Zeile oder die Auswahl “abschickt”, d.h. in die Konsole gibt, wo die Syntax ausgeführt wird. Wenn Sie ein Skript-Fenster öffnen möchten, so können Sie das Icon klicken (Alternativ: Ctrl-Shift-N oder File > New File > R Script).

Aus dem Fenster der *Konsole* spricht R zu uns bzw. wir mit R. Wird ein Befehl (synonym: *Funktion*) hier eingegeben, so führt R ihn aus. Es ist aber viel praktischer, Befehle in das Skript-Fenster einzugeben, als in die Konsole. Behalten Sie dieses Fenster im Blick, wenn Sie Antwort von R erwarten.

Im Fenster *Umgebung* (engl. environment) zeigt R, welche Variablen (Objekte) vorhanden sind. Stellen Sie sich die Umgebung wie einen Karpfenteich vor, in dem die Datensätze und andere Objekte herumschwimmen. Was nicht in der Umgebung angezeigt wird, existiert nicht für R. Kurz gesagt: Die Daten, die Ihr R kennt, werden in der Umgebung angezeigt.

Im Fenster rechts unten werden mehrere Informationen bereit gestellt, z.B. werden Diagramme (Plots) dort ausgegeben. Klicken Sie mal die anderen Reiter im Fenster rechts unten durch.



- Wenn Sie RStudio starten, startet R automatisch auch. Starten Sie daher, wenn Sie RStudio gestartet haben, *nicht* noch extra R. Damit hätten Sie sonst zwei Instanzen von R laufen, was zu Verwirrungen (bei R und beim Nutzer) führen kann.
- Wer Shortcuts mag, wird in RStudio überschwänglich beschenkt; der Shortcut für die Shortcuts ist **Shift-Alt-K**.

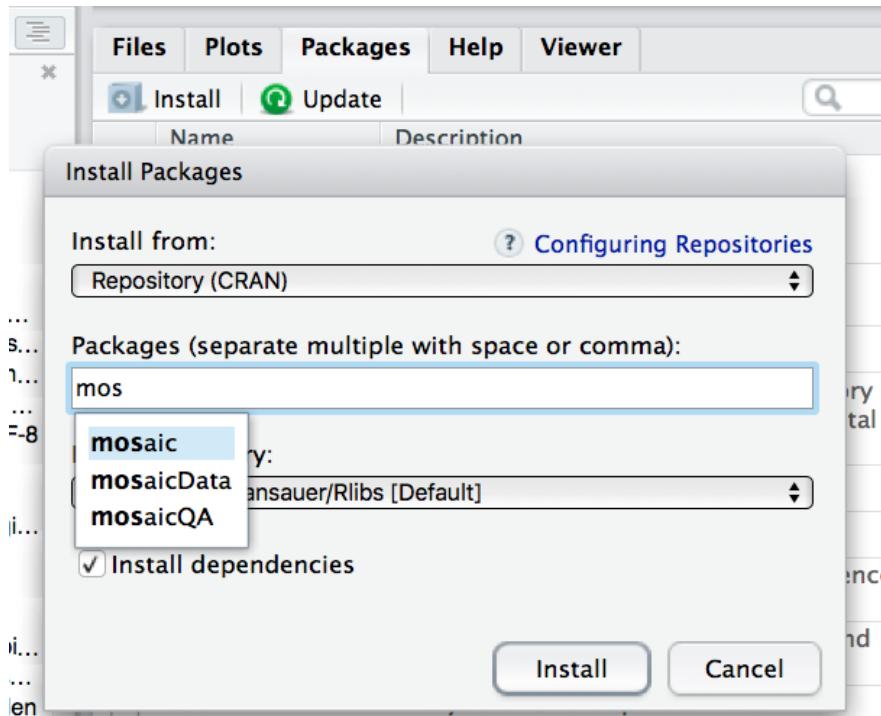


Abbildung 3.2: So installiert man Pakete in RStudio

3.2 Pakete

Ein Großteil der Neuentwicklungen bei R passiert in sog. ‘Paketen’ (packages), das sind Erweiterungen für R. Diese Erweiterungen beherbigen zusätzliche Funktionen und manchmal auch Daten und mehr. Jeder, der sich berufen fühlt, kann ein R-Paket schreiben und es zum ‘R-Appstore’ (CRAN²) hochladen. Von dort kann es dann frei (das heißt auch kostenlos) heruntergeladen werden.

3.2.1 Pakete von CRAN installieren

Am einfachsten installiert man R-Pakete in RStudio über den Button *Install* im Reiter *Packages* (s. Abb. 3.2).



Beim Installieren von R-Paketen könnten Sie gefragt werden, welchen “Mirror” Sie verwenden möchten. Das hat folgenden Hintergrund: R-Pakete sind in einer Art “App-Store”, mit Namen CRAN (Comprehensive R Archive Network) gespeichert. Damit nicht ein armer, kleiner Server überlastet wird, wenn alle Studis dieser Welt just gerade beschließen, ein Paket herunterzuladen, gibt es viele Kopien dieses Servers - seine Spiegelbilder (engl. “mirrors”). Suchen Sie sich einfach einen aus, der in der Nähe ist.

²<https://cran.r-project.org/>

Bei der Installation von Paketen mit `install.packages("name_des_pakets")` sollte stets der Parameter `dependencies = TRUE` angefügt werden. Also `install.packages("name_des_pakets", dependencies = TRUE)`. Hintergrund ist: Falls das zu installierende Paket seinerseits Pakete benötigt, die noch nicht installiert sind (gut möglich), dann werden diese sog. "dependencies" gleich mitinstalliert (wenn Sie `dependencies = TRUE` setzen).

Der Klick auf den Menüpunkt "Install" löst letztlich diesen R-Befehl aus:

```
install.packages("name_des_pakets", dependencies = TRUE)
```

Das Argument `dependencies = TRUE` bedeutet, dass etwaige andere Pakete, von denen unser Paket abhängt, auch installiert werden sollen. Das ist zumeist nötig; daher empfehle ich Ihnen diese Option immer hinzufügen, wenn Sie den Befehl eingeben. RStudio führt dieses Argument in der Voreinstellung auch so aus.

3.2.2 Pakete installieren vs. Pakete starten

Nicht vergessen: Installieren muss man eine Software *nur einmal; starten* (laden) muss man die R-Pakete jedes Mal, wenn man sie vorher geschlossen hat und wieder nutzen möchte.

Wenn Sie R bzw. RStudio schließen, werden alle Pakete ebenfalls geschlossen. Sie müssen die benötigten Pakete beim erneuten Öffnen von RStudio wieder starten.

```
library(dplyr)
```

Der Befehl bedeutet sinngemäß: "Hey R, geh in die Bücherei (library) und hole das Buch (package) dplyr!".



Wann benutzt man bei R Anführungszeichen? Das ist etwas verwirrend im Detail, aber die Grundregel lautet: wenn man Text anspricht. Im Beispiel oben "library(dplyr)" ist "dplyr" hier erst mal für R nichts Bekanntes, weil noch nicht geladen. Demnach müssten *eigentlich* Anführungsstriche stehen. Allerdings meinte ein Programmierer, dass es doch so bequemer ist, so ohne Anführungsstriche - schon wieder zwei Anschläge auf der Tastatur gespart. Hat er Recht. Aber bedenken Sie, dass es sich um die Ausnahme einer Regel handelt. Sie können also auch schreiben: `library("dplyr")` oder `library('dplyr')`; beides geht. Und folgt strenger der R-Logik.

Zu Beginn jedes Kapitels dieses Buchs stehen die R-Pakete, die in dem jeweiligen Kapitel verwendet werden. Bitte installieren Sie diese (falls noch nicht geschehen) und starten Sie diese. Am Anfang vergisst man manchmal ein Paket zu starten und wundert sich dann, warum eine Funktion R unbekannt ist.

3.2.3 Pakete von Github installieren

Github ist ein weiterer Ort, wo man viele R-Pakete finden kann. Im Unterschied zu CRAN, wo der Autor eines Paket zu einigen Qualitätschecks gezwungen wird, ist der Autor bei Github sein eigener Herr. Daher finden sich häufig Pakete, die noch in einer früheren Entwicklungsphase sind, auf Github. Um Pakete von Github zu installieren, kann man den Befehl `install_github()` verwenden, der aus dem Paket `devtools` kommt.

Die Daten dieses Buches finden sich in einem Paket auf Github (`pradadata`). Daher installieren Sie bitte zuerst `devtools` und dann das Paket `pradadata`:

```
install.packages("devtools", dependencies = TRUE)
library(devtools)
install_github("sebastiansauer/pradadata")
```

3.3 Hilfe! R startet nicht!

Manntje, Manntje, Timpe Te,
 Buttje, Buttje inne See,
 myne Fru de Ilsebill
 will nich so, as ik wol will.

*Gebrüder Grimm, Märchen vom Fischer und seiner Frau*³

Ihr R startet nicht oder nicht richtig? Die drei wichtigsten Heilmittel sind:

1. Schließen Sie die Augen für eine Minute. Denken Sie an etwas Schönes und was Rs Problem sein könnte.
2. Schalten Sie den Rechner aus und probieren Sie es morgen noch einmal.
3. Googeln.

Sorry für die schnoddrigen Tipps. Aber: Es passiert allzu leicht, dass man *Fehler* wie diese macht:



OH NO:

- `install.packages(dplyr)`
- `install.packages("dliar")`
- `install.packages("derpyler")`
- `install.packages("dplyr") # dependencies vergessen`
- Keine Internet-Verbindung

³https://de.wikipedia.org/wiki/Vom_Fischer_und_seiner_Frau

- `library(dplyr) # ohne vorher zu installieren`

Wenn R oder RStudio dann immer noch nicht starten oder nicht richtig laufen, probieren Sie dieses:

- Sehen Sie eine Fehlermeldung, die von einem fehlenden Paket spricht (z.B. “Package ‘Rcpp’ not available”) oder davon spricht, dass ein Paket nicht installiert werden konnte (z.B. “Package ‘Rcpp’ could not be installed” oder “es gibt kein Paket namens ‘Rcpp’ ” oder “unable to move temporary installation XXX to YYY”), dann tun Sie folgendes:
- Schließen Sie R und starten Sie es neu.
- Installieren Sie das oder die angesprochenen Pakete⁴.
- Starten Sie das entsprechende Paket mit `library(name_des_pakets)`.
- Gerade bei Windows 10 scheinen die Schreibrechte für R (und damit RStudio oder RCommander) eingeschränkt zu sein. Ohne Schreibrechte kann R aber nicht die Pakete (“packages”) installieren, die Sie für bestimmte R-Funktionen benötigen. Daher schließen Sie R bzw. RStudio und suchen Sie das Icon von R oder wenn Sie RStudio verwenden von RStudio. Rechtsklicken Sie das Icon und wählen Sie “als Administrator ausführen”. Damit geben Sie dem Programm Schreibrechte. Jetzt können Sie etwaige fehlende Pakete installieren.
- Ein weiterer Grund, warum R bzw. RStudio die Schreibrechte verwehrt werden könnten (und damit die Installation von Paketen), ist ein VirensScanner. Der VirensScanner sagt, nicht ganz zu Unrecht: “Moment, einfach hier Software zu installieren, das geht nicht, zu gefährlich”. Grundsätzlich gut, in diesem Fall unnötig. Schließen Sie R/RStudio und schalten Sie dann den VirensScanner *komplett* (!) aus. Öffnen Sie dann R/RStudio wieder und versuchen Sie fehlende Pakete zu installieren.



Verwenden Sie möglichst die neueste Version von R, RStudio und Ihres Betriebssystems. Ältere Versionen führen u.U. zu Problemen; je älter, desto Problem... Update Sie Ihre Packages regelmäßig z.B. mit `update.packages()` oder dem Button “Update” bei RStudio (Reiter Packages).

3.4 Zuordnung von Paketen zu Befehlen

Woher weiß man, welche Befehle (oder auch Daten) in einem Paket enthalten sind?

Eine einfache Möglichkeit ist es, beim Reiter ‘Pakete’ auf den Namen eines der installierten Pakete zu klicken. Daraufhin öffnet sich die Dokumentation des Pakets und man sieht dort alle

⁴`install.packages("name_des_pakets", dependencies = TRUE)` oder mit dem entsprechenden Klick in RStudio

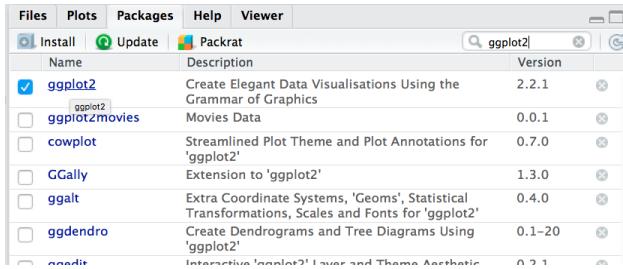


Abbildung 3.3: Hier werden Sie geholfen: Die Dokumentation der R-Pakete

Befehle und Daten aufgeführt (s. Abbildung 3.3). Übrigens sehen Sie dort auch die Version eines Pakets (vielleicht sagt jemand mal zu Ihnen, “Sie sind ja outdated”, dann schauen Sie mal auf die die Paket-Versionen).

Für geladenen Pakete kann man auch den Befehl `help` nutzen, z.B. `help(ggplot2)`.

Und umgekehrt, woher weiß ich, in welchem Paket ein Befehl ‘wohnt’?

Probieren Sie den Befehl `help.search("qplot")`, wenn Sie wissen möchten, in welchem Paket `qplot` zuhause ist. `help.search` sucht alle Hilfeseiten von *installierten* Paketen, in der der Suchbegriff irgendwie vorkommt. Um das Paket eines *geladenen* Befehl zu finden, hilft der Befehl `find: find("qplot")`. Sie können auch den Befehl `find_funs` aus dem Paket `prada` nutzen; dann müssen Sie aber zuerst dieses Paket von Github installieren (s. Abschnitt `??install-github`).

```
prada::find_funs("select")
#> # A tibble: 7 x 3
#>   package_name builtin_package loaded
#>   <chr>           <lgl>  <lgl>
#> 1 BDgraph        FALSE   FALSE
#> 2 dplyr          FALSE   FALSE
#> 3 jmvcore        FALSE   FALSE
#> 4 MASS            TRUE    FALSE
#> 5 plotly          FALSE   FALSE
#> 6 raster          FALSE   FALSE
#> 7 VGAM            FALSE   FALSE
```

In diesem Skript sind am Ende jedes Kapitels die jeweils besprochenen (neuen) Befehle aufgeführt - inklusive ihres Paketes. Falls bei einem Befehl kein Paket angegeben ist, heißt das, dass der Befehl im ‘Standard-R’ wohnt - Sie müssen kein weiteres Paket laden⁵. Also zum Beispiel `ggplot2::qplot`: Der Befehl `qplot` ist im Paket `ggplot2` enthalten. Das Zeichen `::` trennt also Paket von Befehl.

⁵Eine Liste der Pakete, die beim Standard-R enthalten sind (also bereits installiert sind) finden Sie hier⁶



Manche Befehle haben Allerweltsnamen (z.B. ‘filter’). Manchmal gibt es Befehle mit gleichem Namen in verschiedenen Paketen; besonders Befehle mit Allerweltsnamen (wie ‘filter’) sind betroffen (‘mosaic::filter’ vs. ‘dplyr::filter’). Falls Sie von wirre Ausgaben bekommen oder diffuse Fehlermeldung kann es sein, kann es sein, dass R einen Befehl mit dem richtigen Namen aber aus dem ‘falschen’ Paket zieht. Geben Sie im Zweifel lieber den Namen des Pakets vor dem Paketnamen an, z.B. so `dplyr::filter`. Der ‘doppelte Doppelpunkt’ trennt den Paketnamen vom Namen der Funktion.

Außerdem sind zu Beginn jedes Kapitels die in diesem Kapitel benötigten Pakete angegeben. Wenn sie diese Pakete laden, werden alle Befehle dieses Kapitels funktionieren⁷.

Wie weiß ich, ob ein Paket geladen ist?

Wenn der Haken im Reiter ‘Packages’ gesetzt ist (s. Abbildung 3.3), dann ist das Paket geladen. Sonst nicht. Alternativ können Sie den Befehl `search()` ausführen.

3.5 R-Skript-Dateien

Ein neues *R-Skript* im RStudio können Sie z.B. öffnen mit **File-New File-R Script**. Schreiben Sie dort Ihre R-Befehle; Sie können die Skriptdatei speichern, öffnen, ausdrucken, übers Bett hängen... R-Skripte können Sie speichern (unter **File-Save**) und öffnen. R-Skripte sind einfache Textdateien, die jeder Texteditor verarbeiten kann. Nur statt der Endung `.txt`, sind R-Skripte stolzer Träger der Endung `.R`. Es bleibt aber eine schnöde Textdatei. Geben Sie Ihren R-Skript-Dateien die Endung “.R”, damit erkennt RStudio, dass es sich um ein R-Skript handelt und bietet ein paar praktische Funktionen wie den “Run-Button”.

3.6 Daten

Die folgenden Datensätze sind entweder im Paket `prada` enthalten oder können aus anderen Paketen geladen werden. Um Daten aus einem (intallierten) Paket zu laden, gibt es den Befehl `data: data("name_datenobjekt", package = "Paketname")`. Also zum Beispiel:

```
data("stats_test", package = "pradadata")
```

Wenn ein bestimmtes Paket geladen ist, können Sie auch auf den Parameter `package = ...` verzichten, wenn ihr Datensatz in jedem Paket wohnt: Geladene Pakete werden vom Befehl `data` automatisch durchsucht. Die folgenden Datensätze nutzen wir im Rahmen dieses Kurses:

- Datensatz `profiles` aus dem R-Paket `{okcupiddata}` (Kim und Escobedo-Land 2015); es handelt sich um Daten von einer Online-Singlebörsen

⁷es sei denn, sie tun es nicht

- Datensatz `stats_test` aus dem R-Paket `{pradadata}` (Sauer 2017a); es handelt sich um Ergebnisse einer Statistikklausur (einer Probeklausur)
- Datensatz `flights` aus dem R-Paket `{nycflights13}` (RITA 2013); es handelt sich um Abflüge von den New Yorker Flughäfen
- Datensatz `wo_men` aus dem Paket `{pradadata}` oder per URL: <https://osf.io/ja9dw> (Sauer 2017b); es handelt sich um Körper- und Schuhgröße von Studierenden
- Datensatz `extra` aus dem R-Paket `{pradadata}` (Sauer 2016); es handelt sich die Ergebnisse einer Umfrage zu Extraversion
- Datensatz `titanic_train` aus dem Paket `{titanic}` von kaggle⁸; es handelt sich um Überlebensraten vom Titanic-Unglück.
- Datensatz `Affairs` aus dem Paket `{AER}` (Fair 1978); es handelt sich um eine Umfrage zu außerehelichen Affären.
- Datensatz `Werte` von Gansser (2017) (auch im Paket `{pradadata}`); es handelt sich um Daten einer Umfrage zu Werte hinsichtlich Kaufentscheidungen.

Wie man Daten in R ‘einlädt’ (Studierende sagen gerne ‘ins R hochladen’), besprechen wir im Kapitel 6.



Alternativ können Sie die Daten auch im Ordner `data` im Github-Repositorium herunterladen. Gehen Sie auf die Github-Seite dieses Kurses⁹, klicken Sie auf den großen grünen Button “Clone or download”, wählen Sie dann “Download ZIP”, um alle Dateien herunterzuladen. Nach dem Entzippen können Sie dann auf alle Dateien, inklusive Daten, zugreifen.

3.7 Projektmanagement mit RStudio

Wenn man anfängt, sich mit R intensiver zu beschäftigen, kommt der Tag, an dem man professioneller arbeiten möchte: Alle Dateien für ein Projekt sollten zusammen an einem Ort liegen, das Arbeitsverzeichnis an diesen Ort zeigt und es eine vernünftige Ordnerstruktur gibt. RStudio bietet für diesen Zweck, den ich hier als *Projektmanagement* bezeichne, eine Reihe von Hilfen. Dazu zählen das Spezifizieren des Arbeitsverzeichnis, die Arbeit mit Projekten, das Mischen von “Prosa” und R-Code, einfache Publikationsmöglichkeiten und Versionierungswerkzeuge.

Es ist sinnvoll, wenn alle Dateien, die zu einem “Projekt” im weiteren Sinne gehören, in einem gemeinsamen Ordner liegen. Dateien kann man entlang der Arbeitsschritte in vier Arten aufteilen:

- Eingabe: Rohdaten
- Skripte: (R-)Syntax zur Verarbeitung der Rohdaten
- Aufbereitete Daten: Als Zwischenstufe oder Ergebnis nach Durchlauf von Skripten
- Ausgaben: Zumeist Diagramme und Dokumente

⁸<https://www.kaggle.com/c/titanic/data>

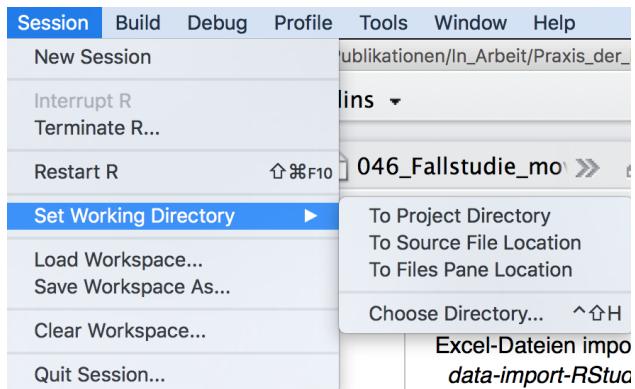


Abbildung 3.4: Das Arbeitsverzeichnis mit RStudio auswählen

3.7.1 Das Arbeitsverzeichnis

Das aktuelle Verzeichnis (Arbeitsverzeichnis; “working directory”) kann man mit `getwd()` erfragen und mit `setwd()` einstellen. Komfortabler ist es aber, das aktuelle Verzeichnis per Menü zu ändern (vgl. Abb. 3.4). In RStudio: Session > Set Working Directory > Choose Directory ... (oder per Shortcut, der dort angezeigt wird).

Es ist praktisch, das Arbeitsverzeichnis festzulegen, denn dann kann man z.B. eine Datendatei einlesen, ohne den Pfad eingeben zu müssen:

```
# nicht ausführen:
daten_deutsch <- read_csv("meine_daten.csv")
```

R geht dann davon aus, dass sich die Datei `meine_daten.csv` im Arbeitsverzeichnis befindet.

Für diesen Kurs ist es sinnvoll, das Arbeitsverzeichnis in einen “Hauptordner” zu legen (z.B. “Statistik_21”), in dem Daten und sonstiges Material als Unterordner abgelegt sind. Übrigens: Wenn Sie keinen Pfad angeben, so geht R davon aus, dass die Daten im aktuellen Verzeichnis (dem *working directory*) liegen.



Spezifizieren Sie ein Arbeitsverzeichnis, wird dort der Inhalt Ihrer Umgebung (d.h. alle geladenen Daten/ Objekte) gespeichert, wenn Sie RStudio schließen und diese Option aktiviert haben (Tools > Global Options > Save Workspace on Exit). Die Umgebung wird auch als *Workspace* bezeichnet. Allerdings sucht RStudio beim Starten nicht zwangsläufig in diesem Ordner. Per Doppelklick auf die Datei (`.Rdata`) oder über File > Open File... können Sie Ihren Workspace wieder laden. Das kann praktisch sein. Ist man aber darauf aus, “schönen” R-Code zu schreiben, ist es teilweise sinnvoller, eine Skript-Datei erneut ablaufen zu lassen, um sich die Datenobjekte neu berechnen zu lassen. Wenn Sie sich in einem RStudio-Projekt befinden,

3.7.2 RStudio-Projekte

Unter dem Menüpunkt Datei > New Projekt kann man ein neues Projekt anlegen. Dabei wird man gefragt, ob man einen neuen Ordner anlegen möchte oder auf einen vorhandenen Ordner zurückgreifen möchte. So oder so, in diesem Ordner sollen dann alle relevanten Dateien für das Projekt abgelegt werden. Man kann auch auf ein Projekt öffnen, welches Versionierung mit Git unterstützt; s. [??](#). Erstellt man ein neues Projekt, so fragt RStudio, um welche Art von Projekt es sich handelt. Für spezielle Arten von Projekten - z.B. Pakete, Bücher, Blogs oder Web-Applikationen - bietet RStudio spezielle Hilfen an. Wir bleiben hier beim allgemeinen Feld-Wald-Wiesen-Projekt (“Empty Project”). Das Arbeitsverzeichnis von R wird dabei automatisch auf diesen Ordner gesetzt. RStudio-Projekte sind eine praktische Angelegenheit, deren Komfort man nicht missen möchte, wenn man mehr macht, als etwas herumzuspielen.

3.7.3 Versionisierung

3.8 Hier werden Sie geholfen

3.8.1 Wo finde ich Hilfe?

Es ist keine Schande, nicht alle Befehle der ca. 10,000 R-Pakete auswendig zu wissen. Schlauer ist, zu wissen, wo man Antworten findet. Hier eine Auswahl:

- Zu diesen Paketen gibt es gute “Spickzettel” (cheatsheets): ggplot2, RMarkdown, dplyr, tidyr. Klicken Sie dazu in RStudio auf *Help > Cheatsheets > ...* oder gehen Sie auf <https://www.rstudio.com/resources/cheatsheets/>.
- In RStudio gibt es eine Reihe (viele) von Tastaturkürzeln (Shortcuts), die Sie hier finden: *Tools > Keyboard Shortcuts Help*.
- Für jeden Befehl aus einem *geladenen* Paket können Sie mit `help()` die Hilfe-Dokumentation anschauen, also z.B. `help("qplot")`.
- Im Internet finden sich zuhauf Tutorials.
- Der Reiter “Help” bei RStudio verweist auf die Hilfe-Seite des jeweiligen Pakets bzw. Befehls.
- Die bekannteste Seite um Fragen rund um R zu diskutieren ist <http://stackoverflow.com>.

3.8.2 Einfache reproduzierbare Beispiele (ERBies)

Sagen wir, Sie haben ein Problem. Mit R. Bevor Sie jemanden bitten, Ihr Problem zu lösen, haben Sie schon drei dreizehn dreißig Minuten recherchiert, ohne Erfolg. Sie entschließen

sich, bei Stackoverflow¹⁰ Ihr Problem zu posten. Außerdem kann sicher eine Mail zu einem Bekannten, einem Dozenten oder sonstwem, der sich auskennen sollte, nicht schaden. Sie formulieren also Ihr Problem: “Hallo, mein R startet nicht, und wenn es startet, dann macht es nicht, was ich soll, außerdem funktioniert der Befehl ‘mean’ bei mir nicht. Bitte lös mein Problem!”. Seltsamerweise reagieren die Empfänger Ihrer Nachricht nicht alle begeistert. Stattdessen verlangt jemand (dreist) nach einer genauen Beschreibung Ihres Problems, mit dem Hinweis, dass “Ferndiagnosen” schwierig sein. Genauer gesagt möchte ihr potenzieller Helfer ein ‘minimal reproducible example’ (MRE) oder, Deutsch, ein *einfaches reproduzierbares Beispiel* (ERBie).

Wenn Sie jemanden um R-Hilfe bitten, dann sollten Sie Ihr Problem prägnant beschreiben.

Was sollte alles in einem ERBie enthalten sein?

Ein ERBie besteht aus vier Teilen: Syntax, Daten, Paketen und Infos zum laufenden System (R Version etc.)

Wie sollte so ein ERBie aussehen? Ich empfehle, folgende Eckpunkte zu beachten¹¹:

- Syntax: Stellen Sie die R-Syntax bereit, die ein Problem bereit (d.h. die einen Fehler liefert).
- Als Text, kein Screenshot: Wenn Sie einen Screenshot schicken, zwingen Sie Ihre Helfer, Ihre Syntax unnötigerweise abzutippen. Ihre Syntax sollte immer als Text (“copy-pastebar”) zur Verfügung stehen.
- Einfach: Geben Sie soweit Syntax wie möglich an. Es bereitet Ihrem Helfer nur wenig Spaß, sich durch 2000 Zeilen Code zu wühlen, wenn es 10 Zeilen auch getan hätten.
- Reproduzierbar Geben Sie soviel Syntax wie nötig, um den Fehler zu erzeugen (aber nicht mehr).
- Schreiben Sie Ihre Syntax übersichtlich, verständlich und kommentiert; z.B. sollten die Variablennamen informativ sein.
- Beschreiben Sie den Fehler genau (“läuft nicht” reicht nicht); z.B. ist es hilfreich, den Wortlaut einer Fehlermeldung bereitzustellen.
- Zu Beginn der Syntax sollten die benötigten Pakete geladen werden.
- Zu Ende des ERBie sollte der Output von `sessionInfo()` einkopiert werden; damit werden Informationen zum laufenden System (wie Version von R, Betriebssystem etc.) bereitgestellt.
- Beziehen Sie sich möglichst auf Daten, die in R schon “eingebaut sind” wie die Datensätze `iris` oder `mtcars`.

Natürlich sollte man immer erst selbst nach einer Lösung recherchieren, bevor man jemanden um Hilfe bittet. Viele Fragen wurden schon einmal diskutiert und oft auch gelöst.

¹⁰www.stackoverflow.com

¹¹Hier finden Sie weitere Hinweise zu ERBies: <https://stackoverflow.com/help/mcve> oder <https://gist.github.com/hadley/270442>

Kapitel 4

ERRRstkontakt



Lernziele:

- Einen Überblick über die fünf wesentliche Schritte der Datenanalyse gewinnen.
- R und RStudio installieren können.
- Einige häufige technische Probleme zu lösen wissen.
- R-Pakete installieren können.
- Einige grundlegende R-Funktionalitäten verstehen.
- Auf die Frage “Was ist Statistik?” eine Antwort geben können.

In diesem Skript geht es um die Praxis der Datenanalyse. Mit Rahmen ist das “Drumherum” oder der Kontext der eigentlichen Datenanalyse gemeint. Dazu gehören einige praktische Vorbereitungen und ein paar Überlegungen. Zum Beispiel brauchen wir einen Überblick über das Thema. Voilà (Abb. 4.1):

Datenanalyse, praktisch betrachtet, kann man in fünf Schritte einteilen (Wickham und Gromelund 2016). Zuerst muss man die Daten *einlesen*, die Daten also in R (oder einer anderen Software) verfügbar machen (laden). Fügen wir hinzu: In *schöner Form* verfügbar

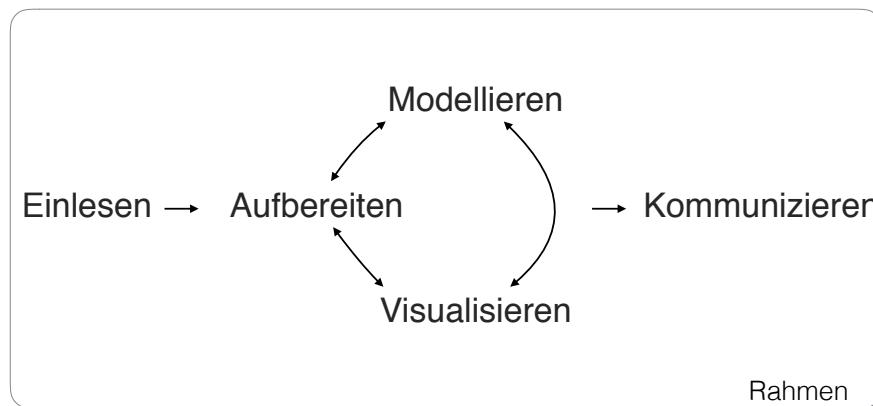


Abbildung 4.1: Der Prozess der Datenanalyse

machen; man nennt dies auch *tidy data* (hört sich cooler an). Sobald die Daten in geeigneter Form in R geladen sind, folgt das *Aufbereiten*. Das beinhaltet Zusammenfassen, Umformen oder Anreichern je nach Bedarf. Ein nächster wesentlicher Schritt ist das *Visualisieren* der Daten. Ein Bild sagt bekanntlich mehr als viele Worte. Schließlich folgt das *Modellieren* oder das Hypothesen prüfen: Man überlegt sich, wie sich die Daten erklären lassen könnten. Zu beachten ist, dass diese drei Schritte - Aufbereiten, Visualisieren, Modellieren - keine starre Abfolge sind, sondern eher ein munteres Hin-und-Her-Springen, ein aufbauendes Abwechseln. Der letzte Schritt ist das *Kommunizieren* der Ergebnisse der Analyse - nicht der Daten. Niemand ist an Zahlenwüsten interessiert; es gilt, spannende Einblicke zu vermitteln.

Der Prozess der Datenanalyse vollzieht sich nicht im luftleeren Raum, sondern ist in einem *Rahmen* eingebettet. Dieser beinhaltet praktische Aspekte - wie Software, Datensätze - und grundsätzliche Überlegungen - wie Ziele und Grundannahmen.

4.1 R ist pingelig

R nimmt einige Dinge recht genau

- R unterscheidet zwischen Groß- und Kleinbuchstaben, d.h. `Oma` und `oma` sind zwei verschiedene Dinge für R!
- R verwendet den Punkt `.` als Dezimaltrennzeichen.
- Fehlende Werte werden in R durch `NA` kodiert.
- Kommentare werden mit dem Rautezeichen `#` eingeleitet; der Rest der Zeile von R dann ignoriert.
- *Variablennamen* in R sollten mit Buchstaben beginnen; ansonsten dürfen nur Zahlen und Unterstriche `(_)` enthalten sein. Leerzeichen sollte man meiden. Das gilt auch für Spaltennamen.
- Um den Inhalt einer Variablen auszulesen, geben wir einfach den Namen des Objekts ein (und schicken den Befehl ab).
- Bleiben Sie konsistent, in der Art und Weise, wie Sie Ihre Syntax schreiben. Ein Vorschlag zum ‘Syntax-Stil’ finden Sie hier¹.
- Variablen einen treffenden Namen zu geben, ist nicht immer leicht, aber wichtig. Namen sollten knapp, aber aussagekräftig sein.

```
# so nicht:
var
x
dummy
objekt
dieser_name_ist_etwas_lang_vielleicht

# gut:
tips_mw
```

¹<http://adv-r.had.co.nz/Style.html>

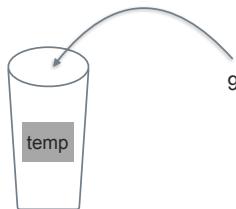


Abbildung 4.2: Eine Variable definieren

lm1

4.1.1 Variablen zuweisen und auslesen

Eine *Variable* ist ein Platzhalter für einen oder mehrere Werte. Man kann sich eine Variable vorstellen, wie ein Behälter, in den man einen Wert (oder mehrere) „einfüllt“ (zuweist). Außerdem hat der Behälter einen Namen (es klebt ein Zettel auf dem Behälter), wie könnte man ihn sonst benennen?. Voilà: eine Variable (s. Abbildung 4.2). Eine Variable wird auch als *Objekt* bezeichnet; mehr dazu in Abschnitt 5.

Gerade wenn man viele Werte hat, ist es praktisch, diese einer Variable zuzuweisen und dann (zu R) zu sagen, “Hey R, berechne den Mittelwert aller Werte in diesem Behälter”. Dann müssen Sie nicht jedes Mal alle Werte gebetsmühlenartig aufzagen.

In R definiert man Variablen mit dem Zuweisungspfeil:

```
temperatur <- 9
```

Mit dem Operator `<-` weisen wir dem Behälter (der Variablen) mit dem Namen `temperatur` den Wert 9 zu; wir “binden” den Wert 9 an das Symbol `temperatur`, sagt man.

Um den Inhalt der Variablen `temperatur` auszulesen (wir haben ihn vergessen), geben wir einfach den Namen der Variablen ein:

```
temperatur
#> [1] 9
```



An vielen, den meisten Stellen, darf man Leerzeichen setzen, wenn man mit R spricht (Befehle erteilt). Der Zuweisungspfeil ist eine Ausnahme: es ist “verboten” < - zu schreiben, also ein Leerzeichen zwischen dem Kleinerzeichen und dem Minuszeichen einzufügen. R würde diesen Versuch mit einer Fehlermeldung quittieren, da R dann etwas von einem Kleinerzeichen und einem Minuszeichen liest und nicht ahnt, dass Sie einen Zuweisungszeichen schreiben wollten.

Wenn man keine Zahlen, sondern *Text* zuweisen will, so muss man den Text in Anführungszeichen setzen (doppelte oder einfache sind erlaubt):

```
y <- "Hallo R!"
```

Man kann auch einer Variablen eine andere zuweisen:

```
y <- x
```

Wird jetzt y mit dem Inhalt von x überschrieben oder umgekehrt? Der Zuweisungspfeil <- macht die Richtung der Zuweisung ganz klar. Zwar ist in R das Gleichheitszeichen (=) synonym zum Zuweisungspfeil erlaubt, aber der Zuweisungspfeil macht die Sache glasklar und sollte daher bevorzugt werden.

Übrigens ist R manchmal etwas pingelig. Dieser Aufruf wird zu einem Fehler führen, da es das Objekt **temperatur** nicht gibt; wir haben nur das Objekt **temperatur** definiert. Gerade wenn man einem Objekt einen kreativen Namen wie x gegeben hat, kann es schnell vorkommen, dass man sich vertut und z.B. X eingibt.

```
Temperatur
```

4.1.2 Funktionen aufrufen

Um einen *Befehl* (präziser aber hier synonym gebraucht: eine Funktion) aufzurufen, geben wir ihren Namen an und definieren sog. *Argumente* in einer runden Klammer, z.B. so:

```
wo_men <- read.csv("data/wo_men.csv")
```

Allgemein gesprochen:

```
funktionsname(argumentname1 = wert1, argumentname2 = wert2, ...)
```

Die drei Punkte ... sollen andeuten, dass evtl. weitere Argumente² übergeben werden könnten. Die Reihenfolge der Argumente ist *egal* - wenn man die Argumentnamen anführt. Ansonsten muss man sich an die Standard-Reihenfolge, die eine Funktion vorgibt, halten. Die Hilfeseite einer Funktion gibt Aufschluss über die Namen und Standard-Reihenfolge der Argumente (z.B. ?read_csv).

```
#ok:
wo_men <- read_csv(path = "data/wo_men.csv", col_names = TRUE)
wo_men <- read_csv("data/wo_men.csv", TRUE, )
wo_men <- read_csv(header = TRUE, path = "data/wo_men.csv")

# ohno:
wo_men <- read.csv(TRUE, "data/wo_men.csv", ",")
```

In der Hilfe zu einem Befehl findet man die Standard-Syntax inklusive der möglichen Parameter, ihrer Reihenfolge und Standardwerten (default values) von Argumenten. Zum Beispiel ist beim Befehl `read.csv` der Standardwert für `sep` mit ; voreingestellt (schauen Sie mal in der Hilfe nach). Gibt man einen Parameter nicht an, für den ein Standardwert eingestellt ist, ‘befüllt’ R den Parameter mit diesem Standardwert.

4.2 Logische Prüfungen

Eine häufige Tätigkeit in der Datenanalyse (und beim Programmieren allgemein) sind logische Prüfungen (Vergleiche). Damit ist gemeint, zu prüfen, ob eine Aussage wahr ist oder falsch. Zum Beispiel ist die Aussage wahr:

```
2 < 3
#> [1] TRUE
```

Dafür hätten Sie jetzt nicht R gebraucht, OK. Anstelle von Zahlen können auch Objekte Teil der Prüfung sein:

```
temperatur < 10
#> [1] TRUE
temperatur == 10
#> [1] FALSE
```

Um Gleichheit zu prüfen, sieht die Rechtschreibung von R das *doppelte* Gleichheitszeichen vor. Ein häufiger Fehler ist es, Gleichheitsprüfung mit *einem* Gleichheitszeichen zu überprüfen, da

²auch als Parameter bezeichnet

Tabelle 4.1: Logische Operatoren in R

Logik-Operator	Beschreibung
<code>==</code>	Prüfung auf Gleichheit
<code>!=</code>	Prüfung auf Ungleichheit
<code><</code>	kleiner als
<code><=</code>	kleiner als oder gleich groß
<code>></code>	größer als
<code>>=</code>	größer als oder gleich groß
<code>!</code>	nicht (Negation)
<code>&</code>	logisches Und
<code> </code>	logisches Oder

das einfache Gleichheitszeichen für Zuweisungen vorgesehen ist (aber nicht empfehlenswert) und für Funktionsargumente gebraucht wird. Ist eine logische Aussage wahr, so quittiert R das mit `TRUE`; falsche Aussagen mit `FALSE`. R unterstützt die klassischen Logik-Operationen wie das logische Und sowie das logische Oder:

```
# Temperatur zwischen 0 und 10 Grad?
temperatur < 10 & temperatur > 0
#> [1] TRUE
```

In Tabelle 4.1 sind gängige logische Operatoren aufgeführt.

4.3 Vektorielle Funktionen

Angenommen, Sie haben eine Liste der Temperaturen jedes Tages Ihres Wohnorts; und zwar von den letzten 100 Jahren. Aus irgendwelchen Gründen sind die Temperaturen in Fahrenheit angegeben; Celcius wäre Ihnen lieber. Aufgrund der großen Menge der Zahlen käme es Ihnen ungelegen, jeden einzelnen Wert umzurechnen. Praktischerweise sind viele (nicht alle) R-Befehle *vektoriell* aufgebaut. Das bedeutet: Sie geben eine Variable mit vielen Werten als Eingabe und R führt den Befehl *auf jedes einzelne Element* der Eingabe aus. Damit müssen Sie nur einmal den Befehl eingeben, das ist ziemlich praktisch.

Sagen wir, das sei unsere Variable, mit einigen Temperaturwerten (wir geben uns hier mit derer drei zufrieden):

```
temperaturen <- c(32, 54, 0) # Temperaturen in Fahrenheit
```

Der Befehl `c()` erstellt eine Variable, mit *mehr als einem Element*; mehr dazu in Abschnitt XXX. Jetzt rechnen wir die Werte in Celcius um:

```
temperaturen_celcius <- (temperaturen - 32) * 5/9
temperaturen_celcius
#> [1] 0.0 12.2 -17.8
```

Da Daten häufig als Spalte und damit als eine Variable dargestellt sind, kommt uns diese Fähigkeit von R sehr gelegen.

4.4 Literaturhinweise

Kapitel 5

Datenstrukturen



Lernziele:

- Die wesentlichen Datenstrukturen von R kennen
- Verstehen, warum bzw. inwiefern Vektoren im Zentrum der Datenstrukturen stehen
- Einige Unterschiede zwischen den Datenstrukturen wissen
- Datenstrukturen erzeugen können

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse)
```

5.1 Zentrale Objektarten

R ist eine Programmiersprache, um *Daten* zu analysieren. Da liegt es nahe, dass man diese Daten irgendwie ansprechen können muss. Außerdem sollte R verstehen, ob es sich z.B. um Zahlen oder um Text handelt. Wir lehnen uns nicht zu weit aus dem Fenster, dass Die Art, wie Daten in R strukturiert (repräsentiert, dargestellt) sind, eine zentrale Angelegenheit bei der computergestützten Analyse von Daten ist. *Datenstrukturen* oder synonym *Objekte* meint also die Art, wie Daten in R dargestellt werden¹.

Die wichtigsten Datenstrukturen von R kann man anhand zweier Merkmale gliedern: Dimension und Homogenität. Unter *Dimension* wird hier verstanden, ob das Objekt eine, zwei oder mehrere Dimensionen besitzt. Mit *Homogenität* ist gemeint, ob das Objekt “sorterein” ist, also nur eine Art von Daten verträgt (z.B. Zahlen oder keinen Text). Daraus lassen sich fünf zentrale Objektarten ableiten (Wickham 2014a):

¹übrigens ist in R alles ein Objekt, auch Funktionen oder Argumente von Funktionen

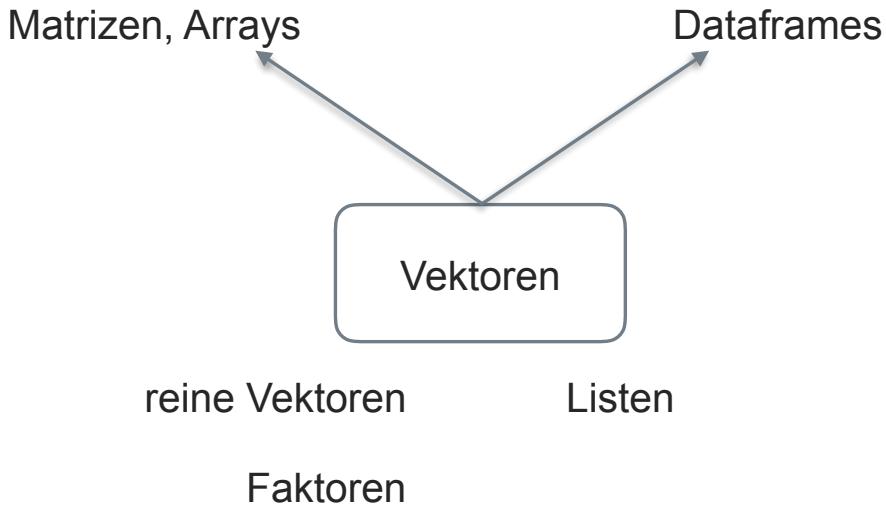


Abbildung 5.1: Datenstrukturen in R; der Vektor steht im Mittelpunkt

Dimensionen	Homogen	Heterogen
1	reiner Vektor, Faktor	Liste
2	Matrix	Dataframe
beliebig	Array	

Es lohnt sich, sich mit diesen Objektarten vertraut zu machen, denn fast alle Objekte in R basieren auf diesen Typen (Wickham 2014a). Praktisch ist hier der Befehl `str(Objektname)`, der den Aufbau (die Struktur) des Objekts `Objektname` recht übersichtlich wiedergibt.

Vielleicht die zentrale Datenstruktur in R ist der *Vektor*, der einem Vektor aus der Algebra gleich: $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

Man erstellt einen Vektor mit der Funktion `c()` (combine)²:

```
ein_vektor <- c(1, 2, 3)
ein_kurzer_vektor <- c(1)
```

Interessanterweise gibt es in R keine Datenstruktur für Skalaren (einzelne Zahlen); diese kurzen Geschöpfe werden als Vektoren der Länge 1 dargestellt und ebenfalls mit `c()` erstellt. Die anderen genannten Datenstrukturen kann man als Spezialfälle von Vektoren verstehen (s. Abbildung 5.1); fast alle Datenstrukturen in R werden intern als Vektor gespeichert (repräsentiert).

Man kann den Elementen eines Vektors einen Namen geben:

²Achtung: `c()`, nicht `C()`

```
ein_vektor <- c(a = 1, b = 2, c = 3)
str(ein_vektor)
#> Named num [1:3] 1 2 3
#> - attr(*, "names")= chr [1:3] "a" "b" "c"
```

Hinter den Kulissen bekommt der Vektor ein “Attribut”. Man kann Vektoren nach Belieben Attribute zuweisen, um Metadaten zu speichern:

```
attr(ein_vektor, "Autor") <- "student"
attr(ein_vektor, "Datum") <- 2017
attr(ein_vektor, "Datum")
#> [1] 2017
```

5.1.1 Reine Vektoren

Vektoren gibt es in zwei Geschmacksrichtungen: reine (atomare) Vektoren und Listen (s. Abbildung 5.1). Ein Vektor ist eindimensional mit einem oder mehr Elementen eines Datentyps. Die wichtigsten Datentypen bzw. Arten reiner Vektoren sind:

- Vektoren für logische Ausdrücke (`logical`)
- Vektoren für ganze Zahlen (`integer`)
- Vektoren für reelle Zahlen (`numeric`, auch `double` genannt)
- Vektoren für Text (`character`)

Alle Typen von Vektoren können mit `c()` erstellt und modifiziert werden:

```
lgl_vector <- c(TRUE, FALSE, T, F) # TRUE und T sind das gleiche, F/FALSE auch
int_vector <- c(1L, 2L, 3L) # Das L sorgt für ganze Zahlen (nicht reell)
num_vector <- c(1, 2.71, 3.14)
chr_vector <- c("Hallo ", "R")
```



1. Liefert folgende zwei Zeilen das gleiche Ergebnis?

```
c(1, c(2, 3)) c(1, 2, 3)
```

2. Wie erwähnt, müssen Vektoren sortenrein sein. Aber für welche “Geschmacksrichtung” entscheidet sich R, wenn dazu gezwungen (man spricht übrigens von *coercion* wenn ein Objekttyp in ein anderer automatisch überführt wird)? Tatsächlich gibt es eine Hierarchie von Datentypen, in die R überführt. Finden Sie sie heraus.
3. Wie hießen noch mal die vier Arten von atomaren Vektoren?
4. Welche anderen Datenstrukturen außer Vektoren gibt es?

5. In welche zwei Arten lassen sich Vektoren aufteilen?

5.1.2 Faktoren

Faktoren sind Vektoren, die nur vorab festgelegte Elemente speichern; z.B. könnte man das Geschlecht von Studienteilnehmern oder Kunden in einem Vektor speichern, der nur `mann` und `frau` als Werte zulässt. Faktoren sind technisch gesehen reine Vektoren vom Typ `integer`, also ganzzahlig. Man nutzt sie, um nominal skalierte Daten zu speichern. Man kann Faktoren mit `c()` anlegen, und sie dann mit `factor()` zu einen Faktor konvertieren:

```
sex <- factor(c("mann", "frau", "frau", "frau"))
str(sex)
#> Factor w/ 2 levels "frau", "mann": 2 1 1 1
```

Wie `str()` zeigt, werden die Elemente intern mit ganzen Zahlen abgespeichert; das erst genannte Element bekommt die Zahl 1, kommt ein anderes Element hinzu, bekommt dieses die Zahl 2 usw. Die *unterschiedlichen* Elemente werden als *Stufen* (Faktorstufen, levels) oder Ausprägungen bezeichnet. Meist werden Faktoren wie Textvariablen verarbeitet; da sie aber technisch Ganzzahl-Vektoren gleichen, kann es zu Problemen kommen, wenn man versucht, sie mit `as.numeric()` in einen numerischen Vektor zu konvertieren

```
fieber <- factor(c(41, 40, 40.5, 41))
feier
#> [1] 41 40 40.5 41
#> Levels: 40 40.5 41
feier_nichtnum <- as.numeric(feier)
feier_nichtnum
#> [1] 3 1 2 3
```

Das Ergebnis 1 2 3 war sicher nicht das gewünschte! Woran liegt das? Der Grund ist, dass "40" intern als erste Ausprägung (level), also als 1 gespeichert ist. Möchte man 40 40.5 41 als Zahlen zurückgeliefert bekommen, hilft dieser Weg:

```
fieber_chr <- as.character(feier)
feier_num <- as.numeric(fieber_chr)
feier_num
#> [1] 41.0 40.0 40.5 41.0
feier_num <- as.numeric(as.character(feier)) # kompakter und identisch
```

Manchmal ist es praktisch, mit Faktoren zu arbeiten, z.B. wenn man den Faktorstufen eine Reihenfolge geben möchte. Sagen wir, uns liegt dieser Vektor vor:

```
tage <- factor(c("Mittwoch", "Montag", "Mittwoch", "Dienstag"))
str(tage)
#> Factor w/ 3 levels "Dienstag", "Mittwoch", ... : 2 3 2 1
```

Eine typische Anwendung ist die Datenvisualisierung; viele Befehle zur Visualisierung nutzen Faktoren bzw. die Reihenfolge ihrer Stufen, um die Elemente einer Achse anzugeordnen (s. Abschnitt XXX). Wie ist eigentlich die Reihenfolge der Stufen von `tage`? `levels` gibt uns Antwort:

```
levels(tage)
#> [1] "Dienstag" "Mittwoch" "Montag"
```

Das ist nicht die übliche Reihenfolge... So lässt sich die Reihenfolge der Faktorstufen ändern:

```
tage <- factor(tage, levels = c("Montag", "Dienstag", "Mittwoch"))
levels(tage)
#> [1] "Montag"    "Dienstag"   "Mittwoch"
```

Technisch sind Faktoren nichts anderes als reine Vektoren plus den zwei Attributten `class` “Faktor” und `levels`, die die Faktorstufen definieren. Ein “reiner Vektor”, wie wir ihn oben definiert haben, hat also nicht die beiden Attribute `class` und `levels`.

5.1.3 Listen

Listen (lists) unterscheiden sich von reinen Vektoren in zwei Hinsichten: zum einen müssen sie nicht homogen sein und zweitens können sie aus Elementen unterschiedlicher Länge bestehen. Listen werden mit dem Befehl `list()` erstellt.

```
eine_liste <- list(1, c(1, 2), c(TRUE, FALSE), c("Hallo ", "Liste"))
str(eine_liste)
#> List of 4
#> $ : num 1
#> $ : num [1:2] 1 2
#> $ : logi [1:2] TRUE FALSE
#> $ : chr [1:2] "Hallo " "Liste"
```

5.1.4 Matrizen und Arrays

Matrizen sind Vektoren mit zwei Dimensionen; wie Tabellen sind es “rechteckige” Datenstrukturen. Arrays sind die Verallgemeinerung von Matrizen auf n Dimensionen; wir werden uns

nicht weiter um Arrays kümmern. Weist man einem Vektor mit `dim()` zwei Dimensionen zu, so entsteht eine Matrix:

```
eine_matrix <- 1:6
dim(eine_matrix) <- c(3, 2)
is.matrix(eine_matrix)
#> [1] TRUE
```

5.1.5 Dataframes

Dataframes ist die zentrale Datenstruktur in R. Unter der Motorhaube ist ein Dataframe eine Liste, deren Elemente gleichlang sein müssen und die Spalten einer Tabelle repräsentieren. Außerdem haben die Elemente (Spalten) Namen. Damit ist eine Dataframe zweidimensional und eine Art Kreuzung von Liste und Matrix. Ein Dataframe lässt sich sowohl als Liste als auch als Matrix ansprechen. Dataframes kann man sich gut als Tabelle vorstellen. Mit `data.frame` oder `tibble::data_frame` lassen sich Dataframes erstellen:

```
ein_df <- data.frame(essen = c("Suppe", "Suppe", "Pizza"), geschmack = c(2, 2, 5))
str(ein_df)
#> 'data.frame': 3 obs. of 2 variables:
#> $ essen    : Factor w/ 2 levels "Pizza", "Suppe": 2 2 1
#> $ geschmack: num 2 2 5
```

Wie man sieht, wird `essen` als Faktor angelegt. Das ist oft nicht praktisch; Textvariablen sind vielfach einfacher. Mit `data_frame` aus `tibble` werden in der Voreinstellung Textvariablen in diesem Fall erzeugt:

```
ein_df2 <- data_frame(essen = c("Suppe", "Suppe", "Pizza"), geschmack = c(2, 2, 5))
str(ein_df2)
#> Classes 'tbl_df', 'tbl' and 'data.frame': 3 obs. of 2 variables:
#> $ essen    : chr "Suppe" "Suppe" "Pizza"
#> $ geschmack: num 2 2 5
```

Tibbles (`tbl` bzw. `tbl_df`) sind auch Dataframes, verhalten sich aber etwas anders. Vor allem sind sie *typstabil*, bleiben also Dataframes, wenn man aus ihnen nur einen Teil der Daten ausliest (s. Abschnitt XXX). Normale Dataframes könnten unerwartet zu reinen Vektoren konvertieren, wenn man nur einen Spalte ausliest. Meist führt das zu Problemen, so dass Tibbles zu bevorzugen sind. Mit `as_tibble()` kann man ein Objekt zwingen, ein Tibble zu werden:

```
ein_df <- as_tibble(ein_df)
ein_tbl <- as_tibble(x)
ein_tbl2 <- as_tibble(eine_liste)
```

Wenn wir im Folgenden von Dataframes sprechen, ist der Oberbegriff gemeint; Tibbles dürfen sich auch angesprochen fühlen. Laxerweise werde ich auch manchmal von “Tabellen” sprechen, damit ist etwas gemeint, was ähnlich oder identisch zu Dataframes ist.

5.2 Datenstruktur auslesen

Möchte man den Inhalt eines Objekts wissen, so gibt man einfach den Namen des Objekts an R und R gibt seinen Inhalt aus:

```
ein_vektor
#> a b c
#> 1 2 3
#> attr(,"Autor")
#> [1] "student"
#> attr(,"Datum")
#> [1] 2017
```

Man kann auch nur einen Teil eines Objekts auslesen; das nennt man auch *Indizieren* (subsetting). Wir werden in Kapitel ?? komfortable Wege für das Indizieren von Dataframes anschauen; die für den Alltag wichtiger sind. Hier geht es um grundlegende Prinzipien von R, die vor allem für den Umgang mit Vektoren sinnvoll sind.

5.2.1 Reine Vektoren

Reine Vektoren können auf mehreren Wege indiziert werden (Wickham 2014a). Betrachten wir diesen einfachen Vektor, `x` als Beispiel:

```
x <- c(2.2, 3.3, 4.4)
```

1. *Positive Ganze Zahlen* geben das Element mit der jeweiligen Position zurück:

```
x[1]
#> [1] 2.2
```

Man kann mehrere Positionen abfragen, wenn man diese mit `c()` kombiniert:

```
x[c(1,2)]
#> [1] 2.2 3.3
```

Der Fantasie sind dabei keine Grenzen gesetzt; ob es nützlich ist, muss man selber entscheiden:

```
x[c(1, 1, 2, 3, 1)]
#> [1] 2.2 2.2 3.3 4.4 2.2
```

2. *Negative Ganze Zahlen* lassen das Element an der angegebenen Stelle aus:

```
x[-1]
#> [1] 3.3 4.4
x[-c(1,2)]
#> [1] 4.4
```

3. *Logische Vektoren* geben die Elemente wieder, deren korrespondieren logischer Wert **TRUE** ist:

```
x[c(TRUE, FALSE, TRUE)]
#> [1] 2.2 4.4
```

Anstelle von **TRUE** oder **FALSE** kann man auch Ausdrücke verwenden, die in einen logischen Wert münden. Dieser Ausdruck prüft für jedes Element, ob es größer als 3 ist. Falls ja, wird **TRUE** zurückgegeben, ansonsten **FALSE**:

```
x > 3
#> [1] FALSE  TRUE  TRUE
```

Diesen Ausdruck können wir zum Indizieren nutzen:

```
x[x > 3]
#> [1] 3.3 4.4
```

Synonym:

```
pruefung_x <- x > 3
x[pruefung_x]
#> [1] 3.3 4.4
```

4. *Namen*: Haben die Elemente einen Namen, so können sie anhand des Namens indiziert werden:

```
names(x) <- c("a", "b", "c")
x["a"]
#> a
#> 2.2
x[c("a", "a", "c")]
#> a a c
#> 2.2 2.2 4.4
```

5.3 Matrizen und Arrays

Matrizen (und Arrays) können analog zu reinen Listen ausgelesen werden. Der Unterschied ist, dass für beide Dimensionen ein Wert angegeben werden, getrennt durch ein Komma:

```
eine_matrix[1,1]
#> [1] 1
eine_matrix[c(1,2), 1]
#> [1] 1 2
```

Der erste Wert spricht die *Zeile* an, der zweite die *Spalte*. Möchte man alle Zeilen auslesen, so gibt man vor dem Komma *nichts* an:

```
eine_matrix[ ,1]
#> [1] 1 2 3
eine_matrix[1, ]
#> [1] 1 4
```

Für Spalte gilt das gleiche, nur *nach* dem Komma.

Daher Matrizen auch eine Art Vektoren sind, kann man sie als Vektoren ansprechen, also eindimensional indizieren.

```
eine_matrix[5]
#> [1] 5
```

So wird das 5 Element der Matrix wiedergeben (spaltenweise gezählt).

5.4 Listen

Listen können auch wie reine Vektoren oder indiziert werden:

```
eine_liste[1] # erstes Element
#> [[1]]
#> [1] 1
eine_liste[c(2,3)] # zweites und drittes Element
#> [[1]]
#> [1] 1 2
#>
#> [[2]]
#> [1] TRUE FALSE
```

Indiziert man mit [, so wird eine Liste zurückgegeben. Möchte man nicht eine Liste zurückbekommen, sondern einen Vektor, so verwendet man die Operatoren [[oder \$:

```
x <- eine_liste[[2]]
str(x)
#> num [1:2] 1 2
```

Der Dollar-Operator (\$) setzt voraus, dass das Element einen Namen hat. Also geben wir Namen:

```
eine_liste <- list(l1 = 1,
                    l2 = c(l2_1 = 1, l2_2 = 2),
                    l3 = c(l3_1 = TRUE, l3_2 = FALSE),
                    l4 = c(l4_1 = "Hallo ", l4_2 = "Liste"))
eine_liste$l2
#> l2_1 l2_2
#> 1 2
```

Auch der Operator [[kann Elemente bei ihrem Namen ansprechen - genau wie bei reinen Listen. Und genau wie bei reinen Listen wird dann ein reiner Vektor zurückgegeben:

```
x <- eine_liste[["l2"]]
str(x)
#> Named num [1:2] 1 2
#> - attr(*, "names")= chr [1:2] "l2_1" "l2_2"
```

Moment mal: x ist ein reiner Vektor, stimmt's? Ja. Und auf reine Vektoren kann man mit [indizieren, stimmt's Ja. Und `eine_liste[["l2"]]` liefert einen reinen Vektor zurück, stimmt's? ... Ja³. Dann kann man die einzelnen Ausdrücke gleichsetzen. Dieser Ausdruck

³nicht glauben, prüfen

```
x[1]
#> l2_1
#> 1
```

liefert also das gleiche wie folgender, weil `eine_liste[["l2"]]` nichts anderes als `x` zurückgibt.

```
eine_liste[["l2"]][1]
#> l2_1
#> 1
```

Interessanterweise liefert die folgende Indizierung wieder das gleiche Ergebnis, ist also synonym:

```
eine_liste[[c(2, 1)]]
#> [1] 1
```

5.5 Dataframes

Dataframes können wie Listen oder wie Matrizen angesprochen werden. Als Liste sieht es so aus:

```
# gibt Dataframe zurück:
ein_df[1]
#> essen
#> 1 Suppe
#> 2 Suppe
#> 3 Pizza
ein_df["essen"]
#> essen
#> 1 Suppe
#> 2 Suppe
#> 3 Pizza

# gibt Vektor zurück:
ein_df[[1]]
#> [1] Suppe Suppe Pizza
#> Levels: Pizza Suppe
ein_df[[["essen"]]]
#> [1] Suppe Suppe Pizza
#> Levels: Pizza Suppe
```

Wie man sieht, gibt es einen Unterschied bei der Ansprache mit [im Gegensatz zu [[: Die einfache Klammer gibt wieder den Dataframe zurück; die Doppelklammer vereinfacht zu einen reinen Vektor. Dann gibt es noch den Dollar-Operator \$, der etwas Tipparbeit spart und daher beliebt ist. Er spricht den Dataframe auch als Liste an, gibt also einen reinen Vektor zurück:

```
ein_df$essen
#> [1] Suppe Suppe Pizza
#> Levels: Pizza Suppe
ein_df$essen[1]
#> [1] Suppe
#> Levels: Pizza Suppe
```

Spricht man als Matrix an, so indiziert man beide Dimensionen:

```
ein_df[1, 1] # erste Zeile, erste Spalte
#> [1] Suppe
#> Levels: Pizza Suppe
ein_df[2, c(1,2)]
#>   essen geschmack
#> 2 Suppe          2
ein_df[1, ]
#>   essen geschmack
#> 1 Suppe          2
```

Teil II

Daten einlesen

Kapitel 6

Daten einlesen



Lernziele:

- Wissen, auf welchen Wegen man Daten in R hineinbekommt.
- Wissen, was eine CSV-Datei ist.
- Wissen, was UTF-8 bedeutet.
- Erläutern können, was R unter dem “working directory” versteht.
- Erkennen können, ob eine Tabelle in Normalform vorliegt.
- Daten aus R hinauskriegen (exportieren).

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Datenjudo und Visualisierung
```

Dieses Kapitel beantwortet eine Frage: “Wie kriege ich Daten in vernünftiger Form in R hinein?“.

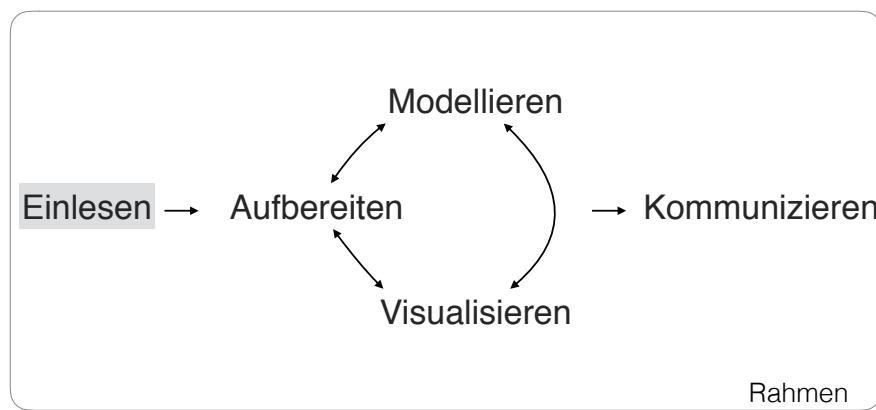


Abbildung 6.1: Daten sauber einlesen

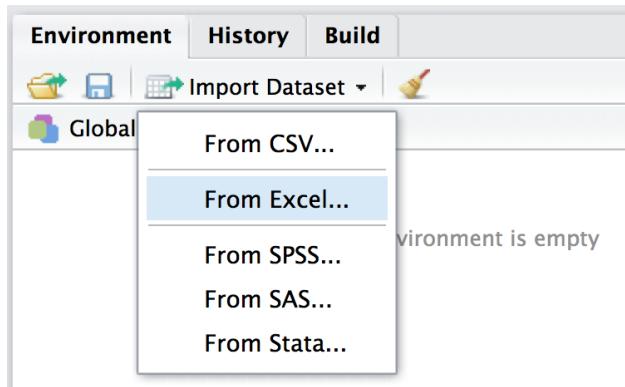


Abbildung 6.2: Daten einlesen (importieren) mit RStudio

6.1 Daten in R importieren

In R kann man ohne Weiteres verschiedene, gebräuchliche (Excel oder CSV) oder weniger gebräuchliche (Feather¹) Datenformate einlesen. In RStudio lässt sich dies z.B. durch einen schnellen Klick auf `Import Dataset` im Reiter `Environment` erledigen².

6.1.1 Excel-Dateien importieren

Am einfachsten ist es, eine Excel-Datei (.xls oder .xlsx) über die RStudio-Oberfläche zu importieren; das ist mit ein paar Klicks geschehen³:

Es ist für bestimmte Zwecke sinnvoll, nicht zu klicken, sondern die Syntax einzutippen. Zum Beispiel: Wenn Sie die komplette Analyse als Syntax in einer Datei haben (eine sog. "Skriptdatei"), dann brauchen Sie (in RStudio) nur alles auszuwählen und auf `Run` zu klicken, und die komplette Analyse läuft durch! Die Erfahrung zeigt, dass das ein praktisches Vorgehen ist.



Daten (CSV, Excel,...) können Sie *nicht* öffnen über `File > Open File ...`. Dieser Weg ist Skript-Dateien und R-Daten-Objekten vorbehalten.

6.1.2 Daten aus R-Paketen importieren

In R-Paketen wohnen nicht nur Funktionen, sondern auch Daten. Diese Daten kann man mit dem Befehl `data` laden, dem man den Namen des zu ladenen Datensatzes `dataset` und

¹<https://cran.r-project.org/web/packages/feather/index.html>

²Um CSV-Dateien zu laden wird durch den Klick im Hintergrund das Paket `readr` verwendet (Wickham, Hester, und Francois 2016a); die entsprechende Syntax wird in der Konsole ausgegeben, so dass man sie sich anschauen und weiterverwenden kann

³im Hintergrund wird das Paket `readxl` verwendet

seines Heimatpakets `paket` übergibt: `data(dataset, package = "paket")`. Natürlich muss das Paket installiert sein. Zum Beispiel:

```
data(movies, package = "ggplot2movies")
data(extra, package = "prada")
```

6.1.3 Daten im R-Format laden

Das R-Datenformat erkennt man an der R-Endung `.rda` oder `RData`. Dateien mit diesem Format kann man in RStudio über `File > Open File...` öffnen. Oder mit dem Befehl `load(file)`, wobei `file` der Dateiname ist, also z.B. `extra.RData`. Mit dem Schwesterbefehl `save` können Sie ein Objekt im R-Datenformat speichern, z.B. `save(stats_test, file = "stats_test.RData")`.

6.1.4 CSV-Dateien importieren

6.1.4.1 Aufbau von CSV-Dateien

Die gebräuchlichste Form von Daten für statistische Analysen ist wahrscheinlich das CSV-Format. Das ist ein einfaches Format, basierend auf einer Textdatei. Schauen Sie sich mal diesen Auszug aus einer CSV-Datei an.

```
row_number,date_time,study_time,self_eval,interest,score
1,05.01.2017 13:57:01,5,8,5,29
2,05.01.2017 21:07:56,3,7,3,29
3,05.01.2017 23:33:47,5,10,6,40
4,06.01.2017 09:58:05,2,3,2,18
5,06.01.2017 14:13:08,4,8,6,34
6,06.01.2017 14:21:18,NA,NA,NA,39
```

Erkennen Sie das Muster? Die erste Zeile gibt die “Spaltenköpfe” wieder, also die Namen der Variablen. Hier sind es 6 Spalten; die fünft heißt “score” und gibt die Punkte eines Studierenden in einer Statistikklausur wieder. Die Spalten sind offenbar durch Komma , voneinander getrennt. Dezimalstellen sind in amerikanischer Manier mit einem Punkt . dargestellt. Die Daten sind “rechteckig”; alle Spalten haben gleich viele Zeilen und umgekehrt alle Spalten gleich viele Zeilen. Man kann sich diese Tabelle gut als Excel-Tabelle mit Zellen vorstellen, in denen z.B. “row_number” (Zelle oben links) oder “39” (Zelle unten rechts) steht.

An einigen Stelle steht `NA`. Das ist Errisch für “fehlender Wert”. Häufig wird die Zelle auch leer gelassen, um auszudrücken, dass ein Wert hier fehlt (hört sich nicht ganz doof an). Aber man findet alle möglichen Ideen, um fehlende Werte darzustellen. Ich rate von allen anderen ab; führt nur zu Verwirrung.

Lesen wir diese Daten jetzt ein:

```
stats_test <- read_csv("data/stats_test.csv")
```

6.1.4.1.1 Daten aus Github einlesen

Die Funktion `read.csv` ist im Standard-R enthalten; `read_csv` kommt aus `readr`, welches über `tidyverse` geladen wird. Übrigens kann man mit `read_csv()` auch CSV-Dateien von einer *URL* (Webseite) aus einlesen. *Github* ist ein zentraler Ort, wo R-Code und auch Daten abgelegt werden.

Um von Github Daten einzulesen - oder allgemeiner: Dateien zu öffnen, muss man Links so aufbauen:

```
https://raw.githubusercontent.com/nutzername/name\_des\_projekts/name\_des\_zweigs/ordner/dateiname.end
```

Mit “Zweig” (branch) ist die Variante des Projekts gemeint: Github baut auf der Versionierungssoftware `git` auf, welche es erlaubt, von einer Datei mehrere (aktuelle, nicht überholte) Varianten gleichzeitig vorzuhalten. Der Standardzweig heißt `master`. Sie könnten z.B. den Datensatz `extra` so einlesen:

```
prada_stats_test_url <-
  paste0("https://raw.github.com/", # Webseite
           "sebastiansauer/", # Nutzer
           "Praxis_der_Datenanalyse/", # Projekt/Repositorium
           "master/", # Variante
           "data/stats_test.csv") # Ordner und Dateinamen

stats_test <- read_csv(prada_stats_test_url)
```



Der Befehl `paste0` “klebt” (to paste) mehrere Textabschnitte zusammen; die Null dabei heißt, zwischen den Textabschnitten steht nichts; die Textabschnitte werden nahtlos zusammengeklebt. Wir hätten die ganze URL auch in einem Textschnipsel packen können. So ist es nur ein bisschen übersichtlicher.

6.1.4.2 Besonderheiten von Tibbles

Importiert man Daten mit `read_csv`, so werden einige Metadaten als Attribute zum Dataframe hinzugespeichert. `str(stats_test)` (von *structure*) offenbart diese Attribute. Möchte man diese Attribute wieder loswerden, so kann man diese Attribute mit `attr` auf `NULL` setzen: `attr(stats_test, "spec") <- NULL`. Außerdem liefert `read_csv` keinen normalen Dataframe zurück, sondern einen *Tibble*. Tibbles sind Dataframes mit einer Reihe von zusätzlichen Eigenschaften, dazu gehören:

Wählt man nur eine Spalte aus einem Tibble, so bleibt es ein Tibble; normale Dataframes verwandeln sich in diesem Fall manchmal zu einem Vektor. Da ist Verwirrung vorprogrammiert. Probieren Sie mal folgende Syntax aus; Zuerst machen wir aus `stats_test` einen normalen Dataframe. Dann wählen mir mit dem [-Operator zuerst zwei Spalten aus, dann eine Spalte aus. Im ersten Fall wird ein Dataframe zurückgeliefert; im zweiten Fall aber ein Vektor, also keine Tabelle! Unschön. Tibbles haben dieses Überraschungsmerkmal nicht, was vor unerwarteten Ärgernissen schützt. Probieren Sie mal den folgenden Code. Ein normaler Dataframe ist auf einmal keine Tabelle mehr, wenn man nur eine Spalte auswählt, sondern er verwandelt sich in einen Vektor. Das kann zu unerwarteten Problemen führen. Tibbles sind "typstabil", bleiben also immer Tibbles, egal wie viele Spalten man auswählt.

```
stats_test %>% as.data.frame -> stats_test_normaler_df
stats_test_normaler_df[, c("self_eval", "score")] %>% str
stats_test_normaler_df[, "self_eval"] %>% str
```

Tibbles haben eine schönere Print-Funktion als normale Dataframes. In R werden Objekte am Bildschirm ausgegeben (gedruckt), wenn man ihren Namen eingibt. Vergleichen Sie mal:

```
extra <- read_csv("data/extrac.csv")
extra
extra %>% as.data.frame
```

6.1.4.3 Vorsicht bei nicht-amerikanisch kodierten Textdateien

Der Befehl `read_csv` liest also eine CSV-Datei, was uns jetzt nicht übermäßig überrascht. Aber Achtung: Wenn Sie aus einem Excel mit *deutscher* Einstellung eine CSV-Datei exportieren, wird diese CSV-Datei als Spaltentrennung ; (Strichpunkt) und als Dezimaltrennzeichen , verwenden. Da der Befehl `read_csv` laut amerikanischen Standard mit Komma als Spaltentrennung und Punkt als Dezimaltrennzeichen arbeitet, müssen wir die deutschen Sonderzeichen explizit angeben, z.B. so:

```
daten_deutsch <- read_delim("daten_deutsch.csv", delim = ";", locale = locale(decimal_ma
```

Dabei steht `delim` (delimiter) für das Trennzeichen zwischen den Spalten und `decimal_mark` für das Dezimaltrennzeichen. R bietet eine Kurzfassung für `read_csv` mit diesen Parametern: `read_csv2("daten_deutsch.csv")`.

Man kommt hier auch mit "Klicken statt Tippen" zum Ziel; in der Maske von "Import Dataset" (für CSV-Dateien) gibt es den Auswahlpunkt "Delimiter" (Trennzeichen). Dort kann man das Komma durch einen Strichpunkt (oder was auch immer) ersetzen. Es hilft, im Zweifel, die Textdatei vorab mit einem Texteditor zu öffnen.

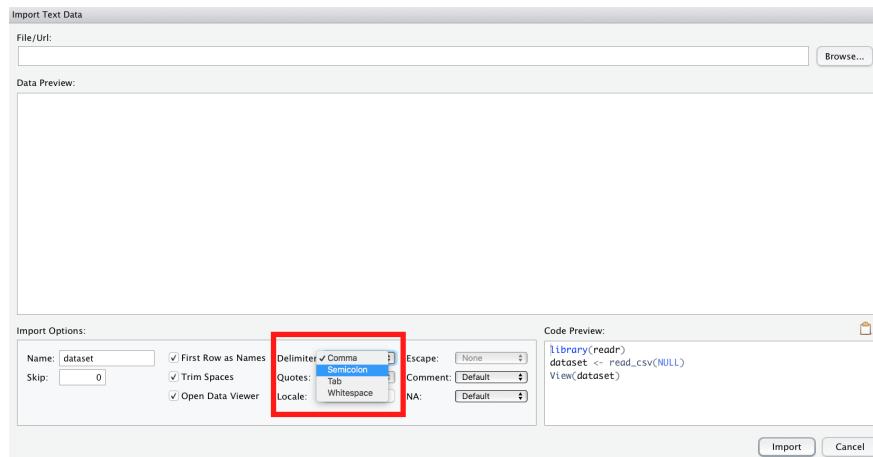


Abbildung 6.3: Trennzeichen einer CSV-Datei in RStudio einstellen

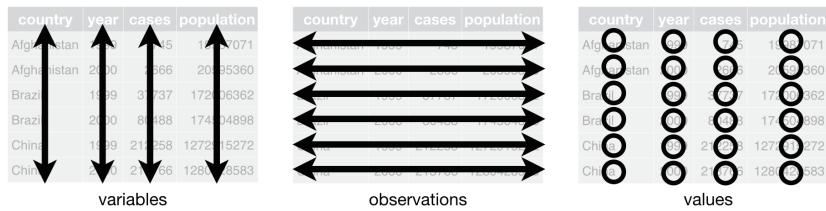


Abbildung 6.4: Schematische Darstellung eines Dataframes in Normalform

6.2 Normalform einer Tabelle

Tabellen in R werden als `data frames` (“Dataframe” auf Denglisch; moderner: als `tibble`, Tibble kurz für “Table-df”) bezeichnet. Tabellen sollten in “Normalform” vorliegen (“tidy”), bevor wir weitere Analysen starten. Unter Normalform verstehen sich folgende Punkte:

- Es handelt sich um einen Dataframe, also um eine Tabelle mit Spalten mit Namen und gleicher Länge; eine Datentabelle in rechteckiger Form und die Spalten haben einen Namen.
- In jeder Zeile steht eine Beobachtung, in jeder Spalte eine Variable.
- Fehlende Werte sollten sich in *leeren* Zellen niederschlagen.
- Daten sollten nicht mit Farbmarkierungen o.ä. kodiert werden.
- Es gibt keine Leerzeilen und keine Leerspalten.
- In jeder Zelle steht ein Wert.
- Am besten verwendet man keine Sonderzeichen verwenden und keine Leerzeichen in Variablennamen und -werten, sondern nur Ziffern und Buchstaben und Unterstriche.
- Variablennamen dürfen nicht mit einer Zahl beginnen.

Abbildung 6.4 visualisiert die Bestimmungsstücke eines Dataframes (Wickham und Grolemund 2016):

The diagram illustrates the transformation of data from a wide format to a long format. On the left, a 'Breit' (wide) table has five columns: ID, Q1, Q2, Q3, and Q4. It contains five rows of data. On the right, a 'Lang' (long) table has three columns: ID, Quartal, and Umsatz. It contains seven rows of data, where the first four rows correspond to the data in the wide table, and the remaining three rows show the continuation of the series.

Breit					Lang		
ID	Q1	Q2	Q3	Q4	ID	Quartal	Umsatz
1	123	342	431	675	1	Q1	342
2	324	342	234	345	2	Q2	342
3	343	124	456	465	3	...	124
...					...	Q1	342
						Q2	342
						Q3	124
						...	

Abbildung 6.5: Dieselben Daten - einmal breit, einmal lang

Der Punkt *Jede Zeile eine Beobachtung, jede Spalte eine Variable, jede Zelle ein Wert* verdient besondere Beachtung. Betrachten Sie das Beispiel in Abbildung 6.5.

In der rechten Tabelle sind die Variablen **Quartal** und **Umsatz** klar getrennt; jede hat ihre eigene Spalte. In der linken Tabelle hingegen sind die beiden Variablen vermischt. Sie haben nicht mehr ihre eigene Spalte, sondern sind über vier Spalten verteilt. Die rechte Tabelle ist ein Beispiel für eine Tabelle in Normalform, die linke nicht.

6.3 Tabelle in Normalform bringen

Eine der ersten Aktionen einer Datenanalyse sollte also die “Normalisierung” Ihrer Tabelle sein. In R bietet sich dazu das Paket **tidyverse** an, mit dem die Tabelle von *Breit- auf Langformat* (und wieder zurück) geschoben werden kann.

Abb. 6.6 zeigt ein Beispiel dazu.

Warum ist es wichtig, von der “breiten” (links in Abb. 6.6) zur “langen” oder “Normalform” (rechts in Abb. 6.6) zu wechseln. Ganz einfach: viele Befehle (allgemeiner: Tätigkeiten) verlangen die Normalform; hin und wieder sind aber die Tabellen von ihrem Schöpfer in breiter Form geschaffen worden. Zum Beispiel erwartet **ggplot2** - und viele andere Diagrammbefehle - dass man *einer* Achse *eine* Spalte (Variable) zuweist, z.B. die Variable “Umsatz” auf die Y-Achse. Der X-Achse könnten wir dann z.B. die Variable “Quartal” packen (s. Abb. 6.7).

Um von der breiten Form zur langen Form zu kommen, kann man den Befehl **tidyverse::gather** nehmen. Von der langen Form zur breiten Form gibt es **tidyverse::spread**. Also etwa:

```
library(tidyverse)
df_lang <- gather(df_breit, key = "Quartal", value = "Umsatz")
```

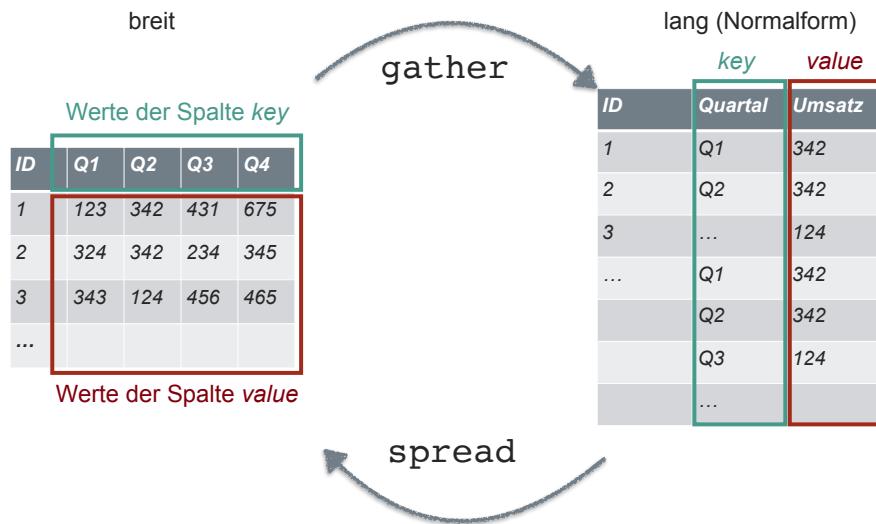


Abbildung 6.6: Mit 'gather' und 'spread' wechselt man von der breiten Form zur langen Form

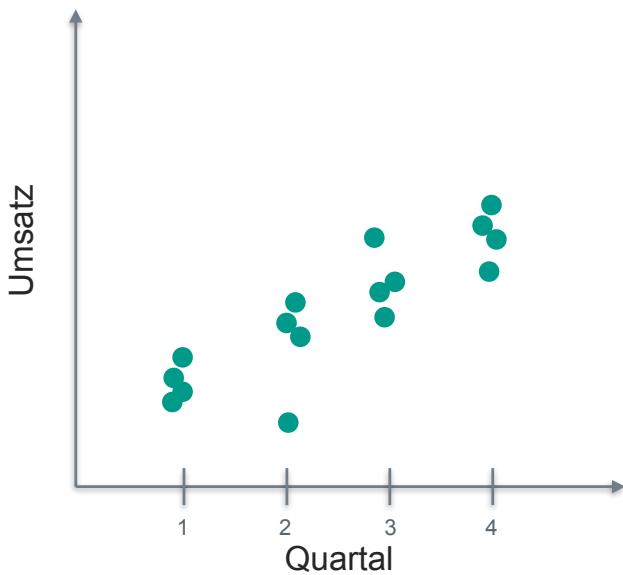


Abbildung 6.7: Ein Beispiel für eine Abbildung zu einer Normalform-Tabelle

```
df_breit <- spread(df_lang, Quartal, Umsatz)
```

Dabei baut `gather` den Dataframe so um, dass nur zwei Spalten übrig bleiben (s. Abb. 6.6). Eine Spalte nur *Spaltennamen* ("Q1", "Q2", ...) enthält; diese Spalte nennt `gather` im Standard `key`. Die zweite Spalte enthält die Werte (z.B. Umsätze), die vormals über mehrere Spalten verstreut waren. Diese Spalte heißt per Default `value`. Im Beispiel oben macht die Spalte `ID` bei dem Spiel "Aus vielen Spalten werden zwei" nicht mit. Möchte man eine Spalte aussparen, so schreibt man das bei `gather` so:

```
df_lang <- gather(df_breit, key = "Quartal", value = "Umsatz", -ID)
```

Tabelle 6.1: Befehle des Kapitels 'Daten einlesen'

Paket::Funktion	Beschreibung
read.csv	Liest eine CSV-Datei ein.
write.csv	Schreibt einen Dateframe in eine CSV-Datei.
tidyr::gather	Macht aus einem "breiten" Dataframe einen "langen".
tidyr::separate	"Zieht" Spalten auseinander.
paste0	Fügt Textschnipsel ohne Trennzeichen zusammen.

In Kapitel 9 werden wir dazu ein Fallstudie einüben.

6.4 Textkodierung

Öffnet man eine Textdatei mit einem Texteditor seiner Wahl, so sieht man... Text und sonst nichts, also keine Formatierung etc. Eine Textdatei besteht aus Text und sonst nichts (daher der Name...). Auch eine R-Skript-Datei (`Coole_Syntax.R`) ist eine Textdatei. Technisch gesprochen werden nur die Textzeichen gespeichert, sonst nichts; im Gegensatz dazu speichert eine Word-Datei noch mehr, z.B. Formatierung. Jetzt steht in der Textdatei der Code "42" für den nächsten Buchstaben. Ja, ist das jetzt ein "A", oder ein "Ä" oder vielleicht ein griechischer Buchstabe? Woher weiß der Computer das eigentlich? Die Antwort ist: Er braucht eine Art Übersetzungstabelle oder Kodierungstafel. Mehrere solcher Kodierungstafeln existieren. Die gebräuchlichste im Internet heißt *UTF-8*⁴. Leider benutzen unterschiedliche Betriebssysteme unterschiedliche Kodierungstafeln, was zu Verwirrung führt. Ich empfehle, Ihre Textdateien als UTF-8 zu kodieren. RStudio fragt Sie, wie eine Textdatei kodiert werden soll. Sie können auch unter `File > Save with Encoding...` die Kodierung einer Textdatei festlegen.

Speichern Sie R-Textdateien wie Skripte stets mit UTF-8-Kodierung ab.

Wie bekommt man seine Daten wieder aus R raus ("ich will zu Excel zurück")?

Eine Möglichkeit bietet die Funktion `write.csv`; sie schreibt eine CSV-Datei:

```
write.csv(name_der_tabelle, "Dateiname.csv")
```

Mit `help(write.csv)` bekommt man mehr Hinweise dazu. Beachten Sie, dass immer in das aktuelle Arbeitsverzeichnis geschrieben wird.

6.5 Befehlsübersicht

Tabelle 6.1 stellt die Befehle dieses Kapitels dar.

⁴<https://de.wikipedia.org/wiki/UTF-8>

6.6 Aufgaben⁵



Richtig oder Falsch!?

1. In CSV-Dateien dürfen Spalten *nie* durch Komma getrennt sein.
2. RStudio bietet die Möglichkeit, CSV-Dateien per Klick zu importieren.
3. RStudio bietet *nicht* die Möglichkeit, CSV-Dateien per Klick zu importieren.
4. “Deutsche” CSV-Dateien verwenden als Spalten-Trennzeichen einen Strichpunkt.
5. In einer Tabelle in Normalform stehen in jeder Zeile eine Beobachtung.
6. In einer Tabelle in Normalform stehen in jeder Spalte eine Variable.
7. R stellt fehlende Werte mit einem Fragezeichen ? dar.
8. Um Excel-Dateien zu importieren, kann man den Befehl `read.csv` verwenden.

6.7 Verweise

- *R for Data Science* bietet umfangreiche Unterstützung zu diesem Thema (Wickham und Grolemund 2016).

⁵F, R, F, R, R, R, F, F

Teil III

Daten aufbereiten

Kapitel 7

Datenjudo



Lernziele:

- Die zentralen Ideen der Datenanalyse mit dplyr verstehen.
- Typische Probleme der Datenanalyse schildern können.
- Zentrale dplyr-Befehle anwenden können.
- dplyr-Befehle kombinieren können.
- Die Pfeife anwenden können.
- Werte umkodieren und “binnen” können.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Datenjudo
library(stringr) # Texte bearbeiten
library(car) # für 'recode'
library(desctable) # Statistiken auf einen Streich
library(lsr) # für Befehl `aad`
```

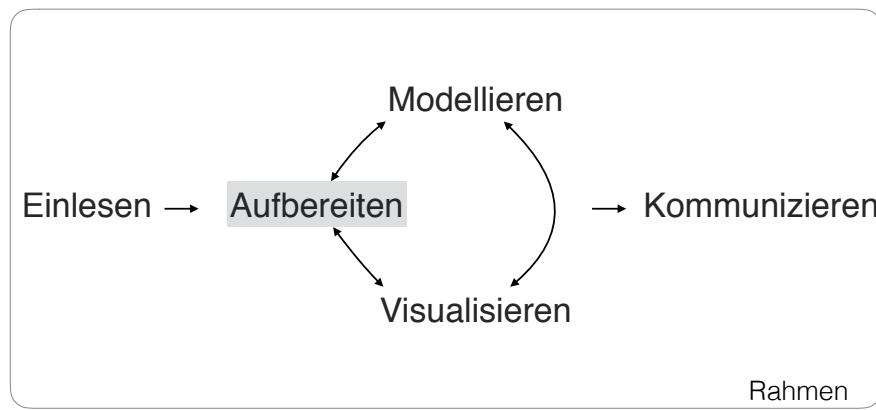


Abbildung 7.1: Daten aufbereiten

```
library(nycflights13) # für `join`
```

Das Paket `tidyverse` lädt `dplyr`, `ggplot2` und weitere Pakete¹. Daher ist es komfortabler, `tidyverse` zu laden, damit spart man sich Tipparbeit. Die eigentliche Funktionalität, die wir in diesem Kapitel nutzen, kommt aus dem Paket `dplyr`.

Mit *Datenjudo* ist gemeint, die Daten für die eigentliche Analyse “aufzubereiten”. Unter *Aufbereiten* ist hier das Umformen, Prüfen, Bereinigen, Gruppieren und Zusammenfassen von Daten gemeint. Die deskriptive Statistik fällt unter die Rubrik Aufbereiten. Kurz gesagt: Alles, was tut, nachdem die Daten “da” sind und bevor man mit anspruchsvoller(er) Modellierung beginnt.

Ist das Aufbereiten von Daten auch nicht statistisch anspruchsvoll, so ist es trotzdem von großer Bedeutung und häufig recht zeitintensiv. Eine Anekdote zur Relevanz der Datenaufbereitung, die (so will es die Geschichte) mir an einer Bar nach einer einschlägigen Konferenz erzählt wurde (daher keine Quellenangabe, Sie verstehen...). Eine Computerwissenschaftlerin aus den USA (deutschen Ursprungs) hatte einen beeindruckenden “Track Record” an Siegen in Wettkämpfen der Datenanalyse. Tatsächlich hatte sie keine besonderen, raffinierten Modellierungstechniken eingesetzt; klassische Regression war ihre Methode der Wahl. Bei einem Wettkampf, bei dem es darum ging, Krebsfälle aus Krankendaten vorherzusagen (z.B. von Röntgenbildern) fand sie nach langem Datenjudo heraus, dass in die “ID-Variablen” Information gesickert war, die dort nicht hingehörte und die sie nutzen konnte für überraschend (aus Sicht der Mitstreiter) gute Vorhersagen zu Krebsfällen. Wie war das möglich? Die Daten stammten aus mehreren Kliniken, jede Klinik verwendete ein anderes System, um IDs für Patienten zu erstellen. Überall waren die IDs stark genug, um die Anonymität der Patienten sicherzustellen, aber gleich wohl konnte man (nach einigem Judo) unterscheiden, welche ID von welcher Klinik stammte. Was das bringt? Einige Kliniken waren reine Screening-Zentren, die die Normalbevölkerung versorgte. Dort sind wenig Krebsfälle zu erwarten. Andere Kliniken jedoch waren Onkologie-Zentren für bereits bekannte Patienten oder für Patienten mit besonderer Risikolage. Wenig überraschen, dass man dann höhere Krebsraten vorhersagen kann. Eigentlich ganz einfach; besondere Mathe steht hier (zumindest in dieser Geschichte) nicht dahinter. Und, wenn man den Trick kennt, ganz einfach. Aber wie so oft ist es nicht leicht, den Trick zu finden. Sorgfältiges Datenjudo hat hier den Schlüssel zum Erfolg gebracht.

7.1 Typische Probleme der Datenaufbereitung

Bevor man seine Statistik-Trickkiste so richtig schön aufmachen kann, muss man die Daten häufig erst noch in Form bringen. Das ist nicht schwierig in dem Sinne, dass es um komplizierte Mathe ginge. Allerdings braucht es mitunter recht viel Zeit und ein paar (oder viele) handwerkliche Tricks sind hilfreich. Hier soll das folgende Kapitel helfen.

Typische Probleme, die immer wieder auftreten, sind:

¹für eine Liste s. `tidyverse_packages(include_self = TRUE)`

- *Fehlende Werte*: Irgend jemand hat auf eine meiner schönen Fragen in der Umfrage nicht geantwortet!
- *Unerwartete Daten*: Auf die Frage, wie viele Facebook-Freunde er oder sie habe, schrieb die Person “I like you a lot”. Was tun???
- *Daten müssen umgeformt werden*: Für jede der beiden Gruppen seiner Studie hat Joachim einen Google-Forms-Fragebogen aufgesetzt. Jetzt hat er zwei Tabellen, die er “verheiraten” möchte. Geht das?
- *Neue Variablen (Spalten) berechnen*: Ein Student fragt nach der Anzahl der richtigen Aufgaben in der Statistik-Probelektionsur. Wir wollen helfen und im entsprechenden Datensatz eine Spalte erzeugen, in der pro Person die Anzahl der richtig beantworteten Fragen steht.

7.2 Daten aufbereiten mit dplyr

Willkommen in der Welt von `dplyr`! `dplyr` hat seinen Namen, weil es sich ausschließlich um Dataframes bemüht; es erwartet einen Dataframe als Eingabe und gibt einen Dataframe zurück (zumindest bei den meisten Befehlen).

Es gibt viele Möglichkeiten, Daten mit R aufzubereiten; `dplyr`² ist ein populäres Paket dafür (wenn Sie `tidyverse` laden, wird `dplyr` auch geladen). `dplyr` basiert auf zwei Ideen:

1. *Lego-Prinzip* Komplexe Datenanalysen in Bausteine zerlegen (vgl. Abb. 7.2).
2. *Durchpfeifen*: Alle Operationen werden nur auf Dataframes angewendet; jede Operation erwartet einen Dataframe als Eingabe und gibt wieder einen Dataframe aus (vgl. Abb. 7.3).

Das *erste Prinzip* von `dplyr` ist, dass es nur ein paar *wenige Grundbausteine* geben sollte, die sich gut kombinieren lassen. Sprich: Wenige grundlegende Funktionen mit eng umgrenzter Funktionalität. Der Autor, Hadley Wickham, sprach einmal in einem Forum (citation needed...), dass diese Befehle wenig können, das Wenige aber gut. Ein Nachteil dieser Konzeption kann sein, dass man recht viele dieser Bausteine kombinieren muss, um zum gewünschten Ergebnis zu kommen. Außerdem muss man die Logik des Baukastens gut verstanden haben - die Lernkurve ist also erstmal steiler. Dafür ist man dann nicht darauf angewiesen, dass es irgendwo “Mrs Right” gibt, die genau das kann, was ich will. Außerdem braucht man sich auch nicht viele Funktionen merken. Es reicht einen kleinen Satz an Funktionen zu kennen (die praktischerweise konsistent in Syntax und Methodik sind). Diese Bausteine sind typische Tätigkeiten im Umgang mit Daten; nichts Überraschendes. Wir schauen wir uns diese Bausteine gleich näher an.

Das *zweite Prinzip* von `dplyr` ist es, einen Dataframe von Operation zu Operation *durchzurichten*. `dplyr` arbeitet also *nur* mit Dataframes. Jeder Arbeitsschritt bei `dplyr` erwartet einen Dataframe als Eingabe und gibt im Gegenzug wieder einen Dataframe aus.

Werfen wir einen Blick auf ein paar typische Bausteine von `dplyr`.

²<https://cran.r-project.org/web/packages/dplyr/index.html>

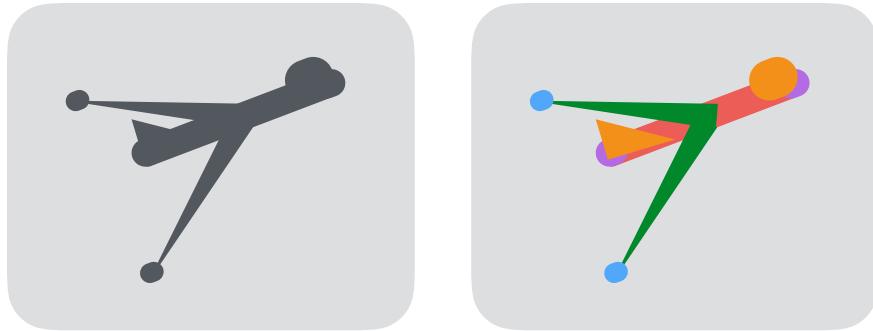


Abbildung 7.2: Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine

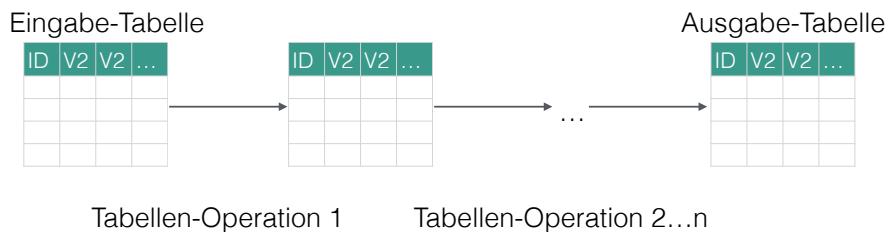


Abbildung 7.3: Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergeleitet

7.3 Zentrale Bausteine von dplyr

7.3.1 Zeilen filtern mit filter

Häufig will man bestimmte Zeilen aus einer Tabelle filtern; `filter`. Zum Beispiel man arbeitet für die Zigarettenindustrie und ist nur an den Rauchern interessiert (die im Übrigen unser Gesundheitssystem retten (Krämer 2011)), nicht an Nicht-Rauchern; es sollen die nur Umsatzzahlen des letzten Quartals untersucht werden, nicht die vorherigen Quartale; es sollen nur die Daten aus Labor X (nicht Labor Y) ausgewertet werden etc.

Abb. 7.4 zeigt ein Sinnbild für `filter`.

Merke:

Die Funktion `filter` filtert Zeilen aus einem Dataframe.

Schauen wir uns einige Beispiele an; zuerst die Daten laden nicht vergessen. Achtung: “Wohnen” die Daten in einem Paket, muss dieses Paket installiert sein, damit man auf die Daten zugreifen kann.

```
data(profiles, package = "okcupiddata") # Das Paket muss installiert sein
```

ID	Name	Note1
1	Anna	1
2	Anna	1
3	Berta	2
4	Carla	2
5	Carla	2

ID	Name	Note1
1	Anna	1
2	Anna	1

Abbildung 7.4: Zeilen filtern

```
df_frauen <- filter(profiles, sex == "f") # nur die Frauen
df_alt <- filter(profiles, age > 70) # nur die alten Menschen
df_alte_frauen <- filter(profiles, age > 70, sex == "f")
# nur die alten Frauen, d.h. UND-Verknüpfung
df_mittelalt <- filter(profiles, between(age, 35, 60))
# zwischen 35 und 60

df_nosmoke_nodrinks <- filter(profiles, smokes == "no" | drinks == "not at all")
# liefert alle Personen, die Nicht-Raucher *oder* Nicht-Trinker sind
```

Gar nicht so schwer, oder? Allgemeiner gesprochen werden diejenigen Zeilen gefiltert (also behalten bzw. zurückgeliefert), für die das Filterkriterium TRUE ist.

`filter` ist deutlich einfacher (und klarer) als Standard-R. Vergleichen Sie mal:

```
filter(profiles, age > 70, sex == "f", drugs == "sometimes")

# base-R:

profiles[df$age > 70 & df$sex == "f" & df$drugs == "sometimes", ]
```



Manche Befehle wie `filter` haben einen Allerweltsnamen; gut möglich, dass ein Befehl mit gleichem Namen in einem anderen (geladenen) Paket existiert. Das kann dann zu Verwirrungen führen - und kryptischen Fehlern. Im Zweifel den Namen des richtigen Pakets ergänzen, und zwar zum Beispiel so: `dplyr::filter(...)`.

7.3.1.1 Fortgeschrittene Beispiele für filter

Einige fortgeschrittene Beispiele für filter:

Man kann alle Elemente (Zeilen) filtern, die zu einer Menge gehören und zwar mit diesem Operator: %in%:

```
filter(profiles, body_type %in% c("a little extra", "average"))
```

Besonders Textdaten laden zu einigen Extra-Überlegungen ein; sagen wir, wir wollen alle Personen filtern, die Katzen bei den Haustieren erwähnen. Es soll reichen, wenn cat ein Teil des Textes ist; also likes dogs and likes cats wäre OK (soll gefiltert werden). Dazu nutzen wir ein Paket zur Bearbeitung von Strings (Textdaten):

```
filter(profiles, str_detect(pets, "cats"))
```

Ein häufiger Fall ist, Zeilen ohne fehlende Werte (NAs) zu filtern. Das geht einfach:

```
profiles_keine_nas <- na.omit(profiles)
```

Aber was ist, wenn wir nur bei bestimmten Spalten wegen fehlender Werte besorgt sind? Sagen wir bei income und bei sex:

```
filter(profiles, !is.na(income) | !is.na(sex))
```

Der horizontale Strich | steht bei R für logisches ‘oder’.

7.3.1.2 Aufgaben³



Richtig oder Falsch!?

1. filter filtert Spalten.
2. filter ist eine Funktion aus dem Paket dplyr.
3. filter erwartet als ersten Parameter das Filterkriterium.
4. filter lässt nur ein Filterkriterium zu.
5. Möchte man aus dem Datensatz profiles (okcupiddata) die Frauen filtern, so ist folgende Syntax korrekt: filter(profiles, sex == "f").
6. filter(profiles, age > 35 | age > 60) filtert die mittelalten Frauen (zwischen 35 und 60)

³F, R, F, F, R, R

The diagram illustrates a data frame transformation. On the left, under 'vorher', there is a data frame with columns: ID, Name, N1, N2, and N3. The rows contain data for three individuals: Anna (ID 1), Berta (ID 2), and Carla (ID 3). The values for N1, N2, and N3 are 1, 2, 3 for Anna; 1, 1, 1 for Berta; and 2, 3, 4 for Carla respectively. Ellipses indicate more rows below. An arrow points from this state to the right, labeled 'nachher'. On the right, under 'nachher', there is a smaller data frame with columns: ID, Name, and N1. It contains the same three rows as before, but only the N1 column is explicitly shown with values 1, 1, and 2 respectively.

ID	Name	N1	N2	N3
1	Anna	1	2	3
2	Berta	1	1	1
3	Carla	2	3	4
...

ID	Name	N1
1	Anna	1
2	Berta	1
3	Carla	2
...

Abbildung 7.5: Spalten auswählen

7.3.2 Spalten wählen mit `select`

Das Gegenstück zu `filter` ist `select`; dieser Befehl liefert die gewählten Spalten zurück. Das ist häufig praktisch, wenn der Datensatz sehr “breit” ist, also viele Spalten enthält. Dann kann es übersichtlicher sein, sich nur die relevanten auszuwählen. Abb. 7.5 zeigt Sinnbild für diesen Befehl:

Merke:

Die Funktion `select` wählt Spalten aus einem Dataframe aus.

Laden wir als ersten einen Datensatz.

```
stats_test <- read.csv("data/stats_test.csv")
```

Dieser Datensatz beinhaltet Daten zu einer Statistikklausur.

```
select(stats_test, score) # Spalte `score` auswählen
select(stats_test, score, study_time)
# Spalten `score` und `study_time` auswählen

select(stats_test, score:study_time) # ditto
select(stats_test, 5:6) # Spalten 5 bis 6 auswählen
```

Tatsächlich ist der Befehl `select` sehr flexibel; es gibt viele Möglichkeiten, Spalten auszuwählen. Im `dplyr`-Cheatsheet⁴ findet sich ein guter Überblick dazu.

⁴<https://www.rstudio.com/resources/cheatsheets/>

7.3.2.1 Aufgaben⁵



Richtig oder Falsch!?

1. `select` wählt *Zeilen* aus.
2. `select` ist eine Funktion aus dem Paket `knitr`.
3. Möchte man zwei Spalten auswählen, so ist folgende Syntax prinzipiell korrekt:
`select(df, spalte1, spalte2)`.
4. Möchte man Spalten 1 bis 10 auswählen, so ist folgende Syntax prinzipiell korrekt:
`'select(df, spalte1:spalte10)`
5. Mit `select` können Spalten nur bei ihrem Namen, aber nicht bei ihrer Nummer aufgerufen werden.

7.3.3 Zeilen sortieren mit `arrange`

Man kann zwei Arten des Umgangs mit R unterscheiden: Zum einen der “interaktive Gebrauch” und zum anderen “richtiges Programmieren”. Im interaktiven Gebrauch geht es uns darum, die Fragen zum aktuell vorliegenden Datensatz (schnell) zu beantworten. Es geht nicht darum, eine allgemeine Lösung zu entwickeln, die wir in die Welt verschicken können und die dort ein bestimmtes Problem löst, ohne dass der Entwickler (wir) dabei Hilfestellung geben muss. “Richtige” Software, wie ein R-Paket oder Microsoft PowerPoint, muss diese Erwartung erfüllen; “richtiges Programmieren” ist dazu vonnöten. Natürlich sind in diesem Fall die Ansprüche an die Syntax (der “Code”, hört sich cooler an) viel höher. In dem Fall muss man alle Eventualitäten voraussehen und sicherstellen, dass das Programm auch beim merkwürdigsten Nutzer brav seinen Dienst tut. Wir haben hier, beim interaktiven Gebrauch, niedrigere Ansprüche bzw. andere Ziele.

Beim interaktiven Gebrauch von R (oder beliebigen Analyseprogrammen) ist das Sortieren von Zeilen eine recht häufige Tätigkeit. Typisches Beispiel wäre der Lehrer, der eine Tabelle mit Noten hat und wissen will, welche Schüler die schlechtesten oder die besten sind in einem bestimmten Fach. Oder bei der Prüfung der Umsätze nach Filialen möchten wir die umsatzstärksten sowie -schwächsten Niederlassungen kennen.

Ein R-Befehl hierzu ist `arrange`; einige Beispiele zeigen die Funktionsweise am besten:

```
arrange(stats_test, score) # liefert die *schlechtesten* Noten zuerst zurück
arrange(stats_test, -score) # liefert die *besten* Noten zuerst zurück
arrange(stats_test, interest, score)
```

```
#>   row_number      date_time bestanden study_time self_eval interest
#> 1       234 23.01.2017 18:13:15     nein          3        1        1
```

⁵F, F, R, R, F

```
#> 2          4 06.01.2017 09:58:05      nein      2      3      2
#>   score
#> 1    17
#> 2    18
#>   row_number      date_time bestanden study_time self_eval interest
#> 1      3 05.01.2017 23:33:47      ja      5      10      6
#> 2      7 06.01.2017 14:25:49      ja     NA     NA     NA
#>   score
#> 1    40
#> 2    40
#>   row_number      date_time bestanden study_time self_eval interest
#> 1      234 23.01.2017 18:13:15     nein      3      1      1
#> 2      142 19.01.2017 19:02:12     nein      3      4      1
#>   score
#> 1    17
#> 2    18
```

Einige Anmerkungen. Die generelle Syntax lautet `arrange(df, Spalte1, ...)`, wobei `df` den Dataframe bezeichnet und `Spalte1` die erste zu sortierende Spalte; die Punkte `...` geben an, dass man weitere Parameter übergeben kann. Man kann sowohl numerische Spalten als auch Textspalten sortieren. Am wichtigsten ist hier, dass man weitere Spalten übergeben kann. Dazu gleich mehr.

Standardmäßig sortiert `arrange` *aufsteigend* (weil kleine Zahlen im Zahlenstrahl vor den großen Zahlen kommen). Möchte man diese Reihenfolge umdrehen (große Werte zuerst, d.h. *absteigend*), so kann man ein Minuszeichen vor den Namen der Spalte setzen.

Gibt man *zwei oder mehr* Spalten an, so werden pro Wert von `Spalte1` die Werte von `Spalte2` sortiert etc; man betrachte den Output des Beispiels oben dazu. Abbildung 7.6) erläutert die Arbeitsweise von `arrange`.

Merke:

Die Funktion `arrange` sortiert die Zeilen eines Dataframes.

Ein ähnliches Ergebnis erhält mit man `top_n()`, welches die *n größten Ränge* wiedergibt:

```
top_n(stats_test, 3, interest)
#>   row_number      date_time bestanden study_time self_eval interest
#> 1      3 05.01.2017 23:33:47      ja      5      10      6
#> 2      5 06.01.2017 14:13:08      ja      4       8      6
#> 3     43 13.01.2017 14:14:16      ja      4       8      6
#> 4     65 15.01.2017 12:41:27     nein      3       6      6
#> 5    110 18.01.2017 18:53:02      ja      5       8      6
#> 6    136 19.01.2017 18:22:57      ja      3       1      6
```

The diagram shows two tables side-by-side. The left table has columns 'ID', 'Name', and 'Note1'. The right table has columns 'ID', 'Name', and 'Note1'. An arrow points from the left table to the right table, with the text 'Gute Noten zuerst!' written vertically next to the arrow.

ID	Name	Note1
1	Anna	1
2	Anna	5
3	Berta	2
4	Carla	4
5	Carla	3

ID	Name	Note1
1	Anna	1
3	Berta	2
5	Carla	3
4	Carla	4
2	Anna	5

Abbildung 7.6: Spalten sortieren

```
#> 7      172 20.01.2017 20:42:46    ja      5      10     6
#> 8      214 22.01.2017 21:57:36    ja      2      6      6
#> 9      301 27.01.2017 08:17:59    ja      4      8      6
#>   score
#> 1      40
#> 2      34
#> 3      36
#> 4      22
#> 5      37
#> 6      39
#> 7      34
#> 8      31
#> 9      33
```

Gibt man *keine* Spalte an (also nur `top_n(stats_test)`), so bezieht sich `top_n` auf die letzte Spalte im Datensatz.

Wenn sich aber, wie hier, mehrere Objekte, den größten Rang (Wert 6) teilen, bekommen wir *nicht* 3 Zeilen zurückgeliefert, sondern entsprechend mehr. dplyr “denkt” sich: “Ok, er will die drei besten Ränge; aber 9 Studenten teilen sich den ersten Rang (Interesse 6), wen sollte ich da ausschließen? Am besten ich liefere alle 9 zurück, sonst wäre es ja ungerecht, weil alle 9 sind ja gleich vom Interesse her”.

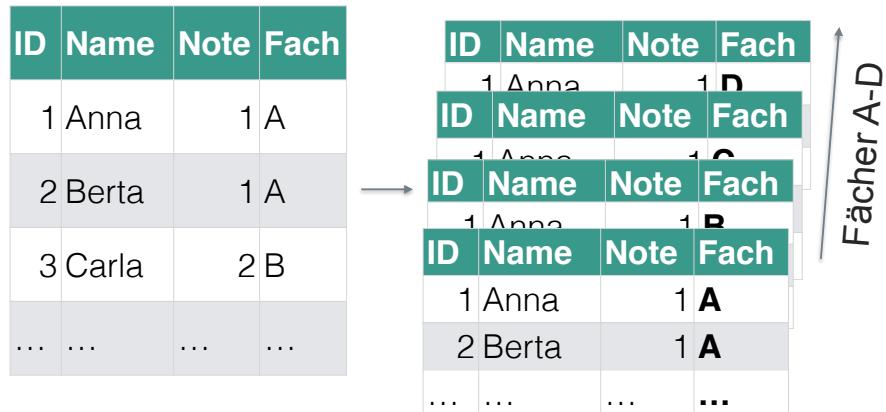


Abbildung 7.7: Datensätze nach Subgruppen aufteilen

7.3.3.1 Aufgaben⁶



Richtig oder Falsch!?

1. `arrange` arrangiert Spalten.
2. `arrange` sortiert im Standard absteigend.
3. `arrange` lässt nur ein Sortierkriterium zu.
4. `arrange` kann numerische Werte, aber nicht Zeichenketten sortieren.
5. `top_n(5)` liefert immer fünf Werte zurück.

7.3.4 Datensatz gruppieren mit `group_by`

Einen Datensatz zu gruppieren ist eine häufige Angelegenheit: Was ist der mittlere Umsatz in Region X im Vergleich zu Region Y? Ist die Reaktionszeit in der Experimentalgruppe kleiner als in der Kontrollgruppe? Können Männer schneller ausparken als Frauen? Man sieht, dass das Gruppieren v.a. in Verbindung mit Mittelwerten oder anderen Zusammenfassungen sinnvoll ist; dazu im nächsten Abschnitt mehr.

Gruppieren meint, einen Datensatz anhand einer diskreten Variablen (z.B. Geschlecht) so aufzuteilen, dass Teil-Datensätze entstehen - pro Gruppe ein Teil-Datensatz (z.B. ein Datensatz, in dem nur Männer enthalten sind und einer, in dem nur Frauen enthalten sind).

In Abbildung 7.7 wurde der Datensatz anhand der Spalte (d.h. Variable) `Fach` in mehrere Gruppen geteilt (Fach A, Fach B...). Wir könnten uns als nächstes z.B. Mittelwerte pro Fach - d.h. pro Gruppe (pro Ausprägung von `Fach`) - ausgeben lassen; in diesem Fall vier Gruppen (Fach A bis D).

⁶F, F, F, F, F

```
test_gruppiert <- group_by(stats_test, interest)
test_gruppiert
#> # A tibble: 306 x 7
#> # Groups:   interest [7]
#>   row_number     date_time bestanden study_time self_eval interest
#>   <int>           <fctr>    <fctr>      <int>      <int>      <int>
#> 1 1 05.01.2017 13:57:01      ja        5         8         5
#> 2 2 05.01.2017 21:07:56      ja        3         7         3
#> 3 3 05.01.2017 23:33:47      ja        5        10         6
#> 4 4 06.01.2017 09:58:05     nein       2         3         2
#> 5 5 06.01.2017 14:13:08      ja        4         8         6
#> 6 6 06.01.2017 14:21:18      ja       NA       NA       NA
#> 7 7 06.01.2017 14:25:49      ja       NA       NA       NA
#> 8 8 06.01.2017 17:24:53     nein       2         5         3
#> 9 9 07.01.2017 10:11:17      ja        2         3         5
#> 10 10 07.01.2017 18:10:05     ja        4         5         5
#> # ... with 296 more rows, and 1 more variables: score <int>
```

Schaut man sich nun den Datensatz an, sieht man erstmal wenig Effekt der Gruppierung. R teilt uns lediglich mit `Groups: interest [7]`, dass es 7 Gruppen gibt, aber es gibt keine extra Spalte oder sonstige Anzeichen der Gruppierung. Aber keine Sorge, wenn wir gleich einen Mittelwert ausrechnen, bekommen wir den Mittelwert pro Gruppe!

Ein paar Hinweise: `Source: local data frame [306 x 6]` will sagen, dass die Ausgabe sich auf einen `tibble` bezieht⁷, also eine bestimmte Art von Dataframe. `Groups: interest [7]` zeigt, dass der Tibble in 7 Gruppen - entsprechend der Werte von `interest` aufgeteilt ist.

`group_by` an sich ist nicht wirklich nützlich. Nützlich wird es erst, wenn man weitere Funktionen auf den gruppierten Datensatz anwendet - z.B. Mittelwerte ausrechnet (z.B mit `summarise`, s. unten). Die nachfolgenden Funktionen (wenn sie aus `dplyr` kommen), berücksichtigen nämlich die Gruppierung. So kann man einfach Mittelwerte pro Gruppe ausrechnen. `dplyr` kombiniert dann die Zusammenfassungen (z.B. Mittelwerte) der einzelnen Gruppen in einen Dataframe und gibt diesen dann aus.

Die Idee des “Gruppieren - Zusammenfassen - Kombinieren” ist flexibel; man kann sie häufig brauchen. Es lohnt sich, diese Idee zu lernen (vgl. Abb. 7.8).

⁷<http://stackoverflow.com/questions/29084380/what-is-the-meaning-of-the-local-data-frame-message-from-dplyr>

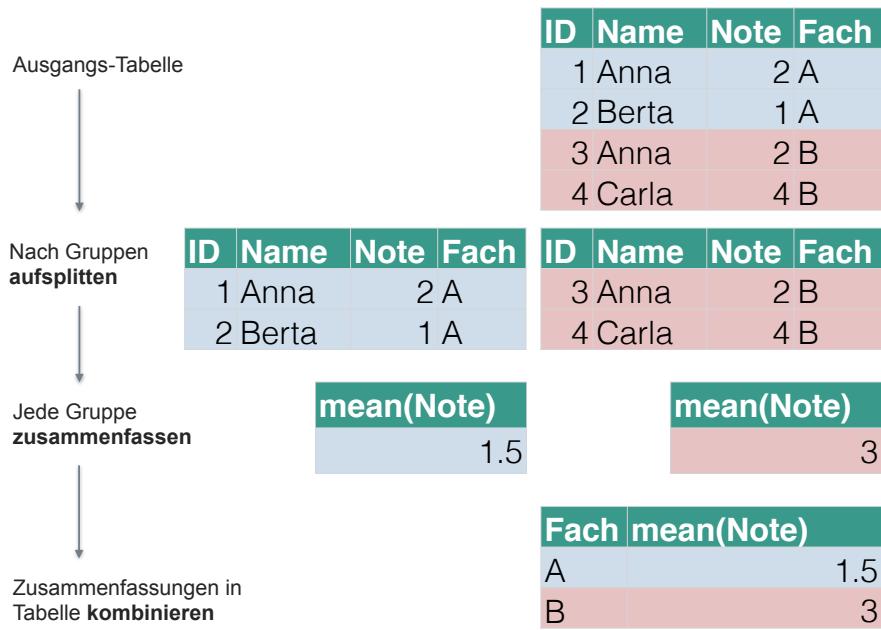


Abbildung 7.8: Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'

7.3.4.1 Aufgaben⁸



Richtig oder Falsch!?

1. Mit `group_by` gruppert man einen Datensatz.
2. `group_by` lässt nur ein Gruppierungskriterium zu.
3. Die Gruppierung durch `group_by` wird nur von Funktionen aus `dplyr` erkannt.
4. `group_by` ist sinnvoll mit `summarise` zu kombinieren.

Merke:

Mit `group_by` teilt man einen Datensatz in Gruppen ein, entsprechend der Werte einer mehrerer Spalten.

7.3.5 Eine Spalte zusammenfassen mit `summarise`

Vielleicht die wichtigste oder häufigste Tätigkeit in der Analyse von Daten ist es, eine Spalte zu *einem* Wert zusammenzufassen; `summarise` leistet dies. Anders gesagt: Einen Mittelwert berechnen, den größten (kleinsten) Wert heraussuchen, die Korrelation berechnen oder eine beliebige andere Statistik ausgeben lassen. Die Gemeinsamkeit dieser Operationen ist, dass sie eine Spalte zu einem Wert zusammenfassen, "aus Spalte mach Zahl", sozusagen. Daher ist der Name des Befehls `summarise` ganz passend. Genauer gesagt fasst dieser Befehl eine Spalte zu einer Zahl zusammen *anhand* einer Funktion wie `mean` oder `max` (vgl. Abb. 7.9).

⁸R, F, R, R

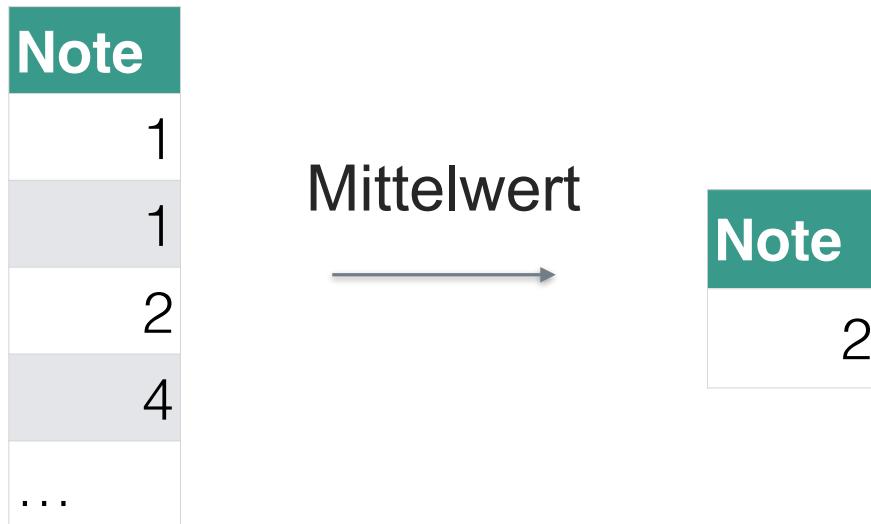


Abbildung 7.9: Spalten zu einer Zahl zusammenfassen

Hierbei ist jede Funktion erlaubt, die eine Spalte als Input verlangt und eine Zahl zurückgibt; andere Funktionen sind bei `summarise` nicht erlaubt.

```
summarise(stats_test, mean(score))
#>   mean(score)
#> 1      31.1
```

Man könnte diesen Befehl so ins Deutsche übersetzen: Fasse aus Tabelle `stats_test` die Spalte `score` anhand des Mittelwerts zusammen. Nicht vergessen, wenn die Spalte `score` fehlende Werte hat, wird der Befehl `mean` standardmäßig dies mit `NA` quittieren. Ergänzt man den Parameter `na.rm = TRUE`, so ignoriert R fehlende Werte und der Befehl `mean` liefert ein Ergebnis zurück.

Jetzt können wir auch die Gruppierung nutzen:

```
test_gruppiert <- group_by(stats_test, interest)
summarise(test_gruppiert, mean(score, na.rm = TRUE))
#> # A tibble: 7 x 2
#>   interest `mean(score, na.rm = TRUE)`
#>   <int>          <dbl>
#> 1     1          28.3
#> 2     2          29.7
#> 3     3          30.8
#> 4     4          29.9
#> 5     5          32.5
#> 6     6          34.0
#> 7    NA          33.1
```

Der Befehl `summarise` erkennt also, wenn eine (mit `group_by`) gruppierte Tabelle vorliegt. Jegliche Zusammenfassung, die wir anfordern, wird anhand der Gruppierungsinformation aufgeteilt werden. In dem Beispiel bekommen wir einen Mittelwert für jeden Wert von `interest`. Interessanterweise sehen wir, dass der Mittelwert tendenziell größer wird, je größer `interest` wird.

Alle diese `dplyr`-Befehle geben einen Dataframe zurück, was praktisch ist für weitere Verarbeitung. In diesem Fall heißen die Spalten `interest` und `mean(score)`. Zweiter Name ist nicht so schön, daher ändern wir den wie folgt:

Jetzt können wir auch die Gruppierung nutzen:

```
test_gruppiert <- group_by(stats_test, interest)
summarise(test_gruppiert, mw_pro_gruppe = mean(score, na.rm = TRUE))
#> # A tibble: 7 x 2
#>   interest mw_pro_gruppe
#>   <int>        <dbl>
#> 1     1        28.3
#> 2     2        29.7
#> 3     3        30.8
#> 4     4        29.9
#> 5     5        32.5
#> 6     6        34.0
#> 7     NA       33.1
```

Nun heißtt die zweite Spalte `mw_pro_Gruppe`. `na.rm = TRUE` veranlasst, bei fehlenden Werten trotzdem einen Mittelwert zurückzuliefern (die Zeilen mit fehlenden Werten werden in dem Fall ignoriert).

Grundsätzlich ist die Philosophie der `dplyr`-Befehle: “Mach nur eine Sache, aber die dafür gut”. Entsprechend kann `summarise` nur *Spalten* zusammenfassen, aber keine *Zeilen*.

Merke:

Mit `summarise` kann man eine Spalte eines Dataframes zu einem Wert zusammenfassen.

7.3.5.1 Aufgaben⁹



Richtig oder Falsch!?

1. Möchte man aus der Tabelle `stats_test` den Mittelwert für die Spalte `score` berechnen, so ist folgende Syntax korrekt: `summarise(stats_test, mean(score))`.
2. `summarise` liefert eine Tabelle, genauer: einen Tibble, zurück.

⁹R, R, R, R, R

3. Die Tabelle, die diese Funktion zurückliefert: `summarise(stats_test, mean(score))`, hat eine Spalte mit dem Namen `mean(score)`.
 4. `summarise` lässt zu, dass die zu berechnende Spalte einen Namen vom Nutzer zugewiesen bekommt.
 5. `summarise` darf nur verwendet werden, wenn eine Spalte zu einem Wert zusammengefasst werden soll.
-
1. (Fortgeschritten) Bauen Sie einen eigenen Weg, um den mittleren Absolutabstand auszurechnen! Gehen Sie der Einfachheit halber (zuerst) von einem Vektor mit den Werten (1,2,3) aus!

Lösung:

```
x <- c(1, 2, 3)
x_mw <- mean(x)
x_delta <- x - x_mw
x_delta <- abs(x_delta)
mad <- mean(x_delta)
mad
#> [1] 0.667
```

7.3.6 Zeilen zählen mit `n` und `count`

Ebenfalls nützlich ist es, Zeilen zu zählen, also Häufigkeiten zu bestimmen. Im Gegensatz zum Standardbefehl¹⁰ `nrow` versteht der `dplyr`-Befehl `n` auch Gruppierungen. `n` darf im Pfeifen-Workflow nur im Rahmen von `summarise` oder ähnlichen `dplyr`-Befehlen verwendet werden.

```
summarise(stats_test, n())
#> n()
#> 1 306
summarise(test_gruppiert, n())
#> # A tibble: 7 x 2
#>   interest `n()`
#>   <int> <int>
#> 1       1     30
#> 2       2     47
#> 3       3     66
#> 4       4     41
```

¹⁰Standardbefehl meint, dass die Funktion zum Standardrepertoire von R gehört, also nicht über ein Paket extra geladen werden muss

```
#> 5      5    45
#> 6      6     9
#> 7      NA   68
nrow(stats_test)
#> [1] 306
```

Außerhalb von gruppierten Datensätzen ist `nrow` meist praktischer.

Praktischer ist der Befehl `count`, der nichts anderes ist als die Hintereinanderschaltung von `group_by` und `n`. Mit `count` zählen wir die Häufigkeiten nach Gruppen; Gruppen sind hier zumeist die Werte einer auszuzählenden Variablen (oder mehrerer auszuzählender Variablen). Das macht `count` zu einem wichtigen Helfer bei der Analyse von Häufigkeitsdaten.

```
dplyr::count(stats_test, interest)
#> # A tibble: 7 x 2
#>   interest     n
#>   <int> <int>
#> 1     1     30
#> 2     2     47
#> 3     3     66
#> 4     4     41
#> 5     5     45
#> 6     6      9
#> 7     NA    68
```

Probieren Sie auch diese Varianten:

```
dplyr::count(stats_test, study_time, sort = TRUE)
dplyr::count(stats_test, interest, study_time)
```

Allgemeiner formuliert lautet die Syntax: `count(df, Spalte1, ...)`, wobei `df` der Dataframe ist und `Spalte1` die erste (es können mehrere sein) auszuzählende Spalte. Gibt man z.B. zwei Spalten an, so wird pro Wert der 1. Spalte die Häufigkeiten der 2. Spalte ausgegeben (vgl. Abb. 7.10).

Merke:

`n` und `count` zählen die Anzahl der Zeilen, d.h. die Anzahl der Fälle.

7.3.6.1 Vertiefung zum Zählen von Zeilen: Relative Häufigkeiten

Manchmal ist es praktisch, nicht nur die (absolute) Häufigkeiten von Zeilen zu zählen, sondern ihren Anteil nach (relative Häufigkeit). Klassisches Beispiel: Wie viel Prozent der Fälle sind Frauen, wie viele sind Männer?

Gruppe A Gruppe B Gruppe C

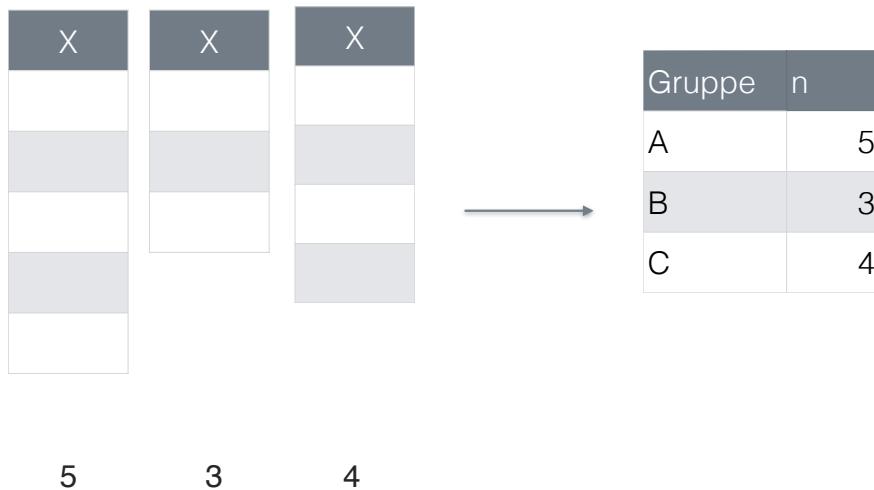


Abbildung 7.10: Sinnbild für 'count'

In dplyr kann man das so umsetzen:

```
stats_test %>%
  count(interest) %>%
  mutate(prop_interest = n / sum(n))
#> # A tibble: 7 x 3
#>   interest     n prop_interest
#>   <int> <int>          <dbl>
#> 1      1    30      0.0980
#> 2      2    47      0.1536
#> 3      3    66      0.2157
#> 4      4    41      0.1340
#> 5      5    45      0.1471
#> 6      6     9      0.0294
#> 7     NA    68      0.2222
```

`prop` steht hier für "Proportion", also Anteil. `sum(n)` liefert die Summe der Fälle zurück, also 306 in diesem Fall.

Etwas komplexer ist es, wenn man zwei Gruppierungsvariablen hat und dann Anteile auszählen möchte:

```
stats_test$bestanden <- stats_test$score > 25

stats_test %>%
  group_by(interest, bestanden) %>%
```

```

summarise(n = n()) %>%
  mutate(prop_interest = n / sum(n))
#> # A tibble: 14 x 4
#> # Groups:   interest [7]
#>   interest bestanden     n prop_interest
#>   <int>      <lgl> <int>      <dbl>
#> 1 1          FALSE    10      0.333
#> 2 1          TRUE     20      0.667
#> 3 2          FALSE    9       0.191
#> 4 2          TRUE     38      0.809
#> 5 3          FALSE    14      0.212
#> 6 3          TRUE     52      0.788
#> 7 4          FALSE    9       0.220
#> 8 4          TRUE     32      0.780
#> 9 5          FALSE    6       0.133
#> 10 5         TRUE     39      0.867
#> 11 6          FALSE    1       0.111
#> 12 6          TRUE     8       0.889
#> 13 NA         FALSE    7       0.103
#> 14 NA         TRUE     61      0.897

```

Synonym zur letzten Syntax könnte man auch schreiben:

```

stats_test %>%
  count(interest, bestanden) %>%
  mutate(prop_interest = n / sum(n))

```

7.3.6.2 Aufgaben¹¹



Richtig oder Falsch!?

1. Mit `count` kann man Zeilen zählen.
2. `count` ist ähnlich (oder identisch) zu einer Kombination von `group_by` und `n()`.
3. Mit `count` kann man nur eine Gruppe beim Zählen berücksichtigen.
4. `count` darf nicht bei nominalskalierten Variablen verwendet werden.

1. Bauen Sie sich einen Weg, um den Modus mithilfe von `count` und `arrange` zu bekommen!

¹¹R, R, F, F



Abbildung 7.11: Das ist keine Pfeife

```
stats_count <- count(stats_test, score)
stats_count_sortiert <- arrange(stats_count, -n)
head(stats_count_sortiert, 1)
#> # A tibble: 1 × 2
#>   score     n
#>   <int> <int>
#> 1     34     22
```

Ah! Der Score 34 ist der häufigste!

7.4 Die Pfeife

Die zweite zentrale Idee von `dplyr` kann man salopp als “Durchpfeifen” oder die “Idee der Pfeife” (Durchpfeifen) bezeichnen; ikonographisch mit einem Pfeifen ähnlichen Symbol dargestellt `%>%`. Der Begriff “Durchpfeifen” ist frei vom Englischen “to pipe” übernommen (und aus dem R-Paket `magrittr`). Das berühmte Bild von René Magritte stand dabei Pate (s. Abb. 7.11; (M7 2004)).

Hierbei ist gemeint, einen Datensatz sozusagen auf ein Fließband zu legen und an jedem Arbeitsplatz einen Arbeitsschritt auszuführen. Der springende Punkt ist, dass ein Dataframe als “Rohstoff” eingegeben wird und jeder Arbeitsschritt seinerseits wieder einen Dataframe ausgibt. Damit kann man sehr schön, einen “Flow” an Verarbeitung erreichen, außerdem spart man sich Tipparbeit und die Syntax wird lesbarer. Damit das Durchpfeifen funktioniert, benötigt man Befehle, die als Eingabe einen Dataframe erwarten und wieder einen Dataframe zurückliefern. Das Schaubild verdeutlicht beispielhaft eine Abfolge des Durchpfeifens (s. Abb. 7.12).

Die sog. “Pfeife” (`pipe: %>%`) in Anspielung an das berühmte Bild von René Magritte, verkettet Befehle hintereinander. Das ist praktisch, da es die Syntax vereinfacht.

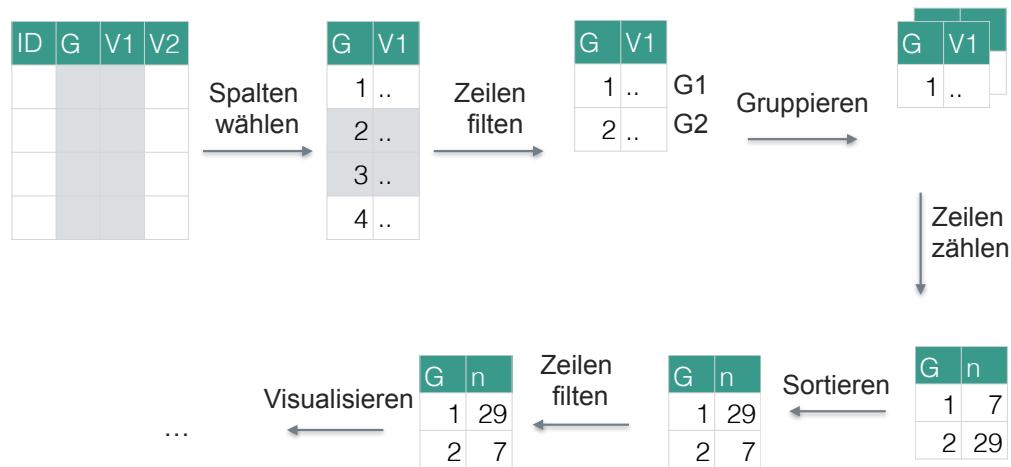


Abbildung 7.12: Das 'Durchpeifen'



Tipp: In RStudio gibt es einen Shortcut für die Pfeife: Strg-Shift-M (auf allen Betriebssystemen).

Vergleichen Sie mal diese Syntax

```
filter(summarise(group_by(filter(stats_test,
    !is.na(score)), interest), mw = mean(score)), mw > 30)
```

mit dieser

```
stats_test %>%
  filter(!is.na(score)) %>%
  group_by(interest) %>%
  summarise(mw = mean(score)) %>%
  filter(mw > 30)
#> # A tibble: 4 x 2
#>   interest     mw
#>   <int> <dbl>
#> 1      3 30.8
#> 2      5 32.5
#> 3      6 34.0
#> 4     NA 33.1
```

Die zweite ist viel einfacher! Lassen Sie uns die "Pfeifen-Syntax" in deutschen Pseudo-Code zu übersetzen.



Nimm die Tabelle “stats_test” UND DANN
 filtere alle nicht-fehlenden Werte UND DANN
 gruppiere die verbleibenden Werte nach “interest” UND DANN
 bilde den Mittelwert (pro Gruppe) für “score” UND DANN
 liefere nur die Werte größer als 30 zurück.

Die zweite Syntax, in “Pfeifenform” ist viel einfacher zu verstehen als die erste! Die erste Syntax ist verschachtelt, man muss sie von innen nach außen lesen. Das ist kompliziert. Die Pfeife in der 2. Syntax macht es viel einfacher, die Syntax zu verstehen, da die Befehle “hintereinander” gestellt (sequenziell organisiert) sind.

Die Pfeife zerlegt die “russische Puppe”, also ineinander verschachtelten Code, in sequenzielle Schritte und zwar in der richtigen Reihenfolge (entsprechend der Abarbeitung). Wir müssen den Code nicht mehr von innen nach außen lesen (wie das bei einer mathematischen Formel der Fall ist), sondern können wie bei einem Kochrezept “erstens …, zweitens …, drittens …” lesen. Die Pfeife macht die Syntax einfacher. Natürlich hätten wir die verschachtelte Syntax in viele einzelne Befehle zerlegen können und jeweils eine Zwischenergebnis speichern mit dem Zuweisungspfeil <- und das Zwischenergebnis dann explizit an den nächsten Befehl weitergeben. Eigentlich macht die Pfeife genau das - nur mit weniger Tipparbeit. Und auch einfacher zu lesen. Flow!



Wenn Sie Befehle verketten mit der Pfeife, sind nur Befehle erlaubt, die einen Datensatz als Eingabe verlangen und einen Datensatz ausgeben. Das ist bei den hier vorgestellten Funktionen der Fall. Viele andere Funktionen erfüllen dieses Kriterium aber nicht; in dem Fall liefert `dplyr` eine Fehlermeldung.

7.4.1 Spalten berechnen mit `mutate`

Wenn man die Pfeife benutzt, ist der Befehl `mutate` ganz praktisch: Er berechnet eine Spalte. Im ‘Standard-R’ kann man eine Spalte berechnen mit dem Zuweisungsoperator und dem \$-Operator:

Zum Beispiel so:

```
df$neue_spalte <- df$spalte1 + df$spalte2
```

Innerhalb einer Pfeifen-Syntax geht das aber nicht (so gut). Da ist man mit der Funktion `mutate` besser beraten; `mutate` leistet just dasselbe wie die Pseudo-Syntax oben:

```
df %>%
  mutate(neue_spalte = spalte1 + spalte2)
```

In Worten:

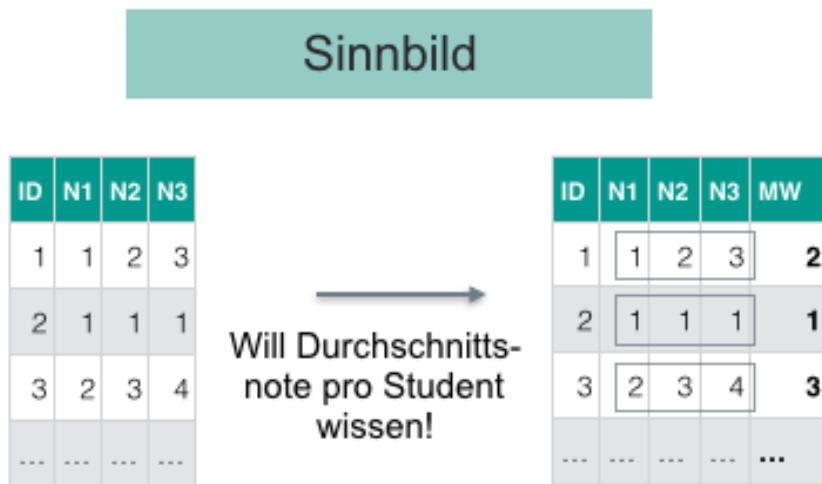


Abbildung 7.13: Sinnbild für mutate



Nimm die Tabelle “df” UND DANN
bilde eine neue Spalte mit dem Namen `neue_spalte`, die sich berechnet als Summe
von `spalte1` und `spalte2`.

Allerdings berücksichtigt `mutate` auch Gruppierungen, das ist praktisch. Der Hauptvorteil ist die bessere Lesbarkeit durch Auflösen der Verschachtelungen.

Ein konkretes Beispiel:

```
stats_test %>%
  select(bestanden, interest, score) %>%
  mutate(Streber = score > 38) %>%
  head()
#>   bestanden interest score Streber
#> 1     TRUE      5    29 FALSE
#> 2     TRUE      3    29 FALSE
#> 3     TRUE      6    40  TRUE
#> 4    FALSE      2    18 FALSE
#> 5     TRUE      6    34 FALSE
#> 6     TRUE     NA    39  TRUE
```

Diese Syntax erzeugt eine neue Spalte innerhalb von `stats_test`; diese Spalte prüft pro Person, ob `score > 38` ist. Falls ja (TRUE), dann ist `Streber` TRUE, ansonsten ist `Streber` FALSE (tja). `head` zeigt die ersten 6 Zeilen des resultierenden Dataframes an.

Abb. 7.13 zeigt Sinnbild für `mutate`; beachten Sie, dass aus einer Spalte eine neue Spalte erzeugt wird. Man könnte also sagen, die alten Werte werden zu neuen *transformiert*.



`mutate` erwartet als Input *keinen* Dateframe, sondern eine Spalte. Betrachten Sie das Sinnbild von `mutate`. Die Idee ist, eine Spalte umzuwandeln nach dem Motto: "Nimm eine Spalte, mach was damit und liefere die neue Spalte zurück". Die Spalte (und damit jeder einzelne Wert in der Spalte) wird *verändert* ('mutiert', daher 'mutate'). Man kann auch sagen, die Spalte wird *transformiert*.

7.4.2 Aufgaben

1. Entschlüsseln Sie dieses Ungetüm! Übersetzen Sie diese Syntax auf Deutsch:

```
bestanden_gruppen <-
  filter(
    summarise(
      group_by(filter(select(stats_test,
                        -c(row_number, date_time)),
                  bestanden == "ja"), interest),
      Punkte = mean(score), n = n())))

```

2. Entschlüsseln Sie jetzt diese Syntax bzw. übersetzen Sie sie ins Deutsche:

```
stats_test %>%
  select(-row_number, -date_time) %>%
  filter(bestanden == "ja") %>%
  group_by(interest) %>%
  summarise(Punkte = mean(score),
            n = n())

```

3. Die Pfeife bei im Klausur-Datensatz

- (Übersetzen Sie die folgende Pseudo-Syntax ins ERRRische!



Nimm den Datensatz `stats_test` UND DANN...
 Wähle daraus die Spalte `score` UND DANN...
 Berechne den Mittelwert der Spalte UND DANN...
 ziehe vom Mittelwert die Spalte ab UND DANN... quadriere die einzelnen Differenzen
 UND DANN... bilde davon den Mittelwert.

Lösung:

```
stats_test %>%
  select(score) %>%
  mutate(score_delta = score - mean(.score)) %>%
  mutate(score_delta_squared = score_delta^2) %>%
  summarise(score_var = mean(score_delta_squared)) %>%
  summarise(sqrt(score_var))
```

Was sagt uns der Punkt . in der Syntax oben? Der Punkt steht für die Tabelle, wie sie gerade aufbereitet ist (also laut letzter Zeile in der Syntax). Warum müssen wir dem Befehl `mean` sagen, welche Spalte/Variable `score` wir meinen? Ist doch logo, wir meinen natürlich die Spalte `score` im aktuellen, durchgefiffenen Datensatz! Leider weiß das der Befehl `mean` nicht. `mean` hat keinerlei Idee von Pfeifen, unseren Wünschen und Sorgen. `mean` denkt sich: "Not my job! Sag mir gefälligst *wie immer*, in welchem Dataframe ich die Spalte finde!". Also sagen wir `mean`, wo er die Spalte findet...

- Berechnen Sie die `sd` von `score` in `stats_test!` Vergleichen Sie sie mit dem Ergebnis der vorherigen Aufgabe!¹²
- Was hat die Pfeifen-Syntax oben berechnet?¹³

7.5 Deskriptive Statistik

7.5.1 Deskriptive Statistik mit `dplyr`

`dplyr` kann man gut gebrauchen, um deskriptive Statistik zu berechnen. Dabei charaktersiert `summarise` eine Hauptidee der Deskriptivstatistik: Einen Vektor zu einer Zahl zusammenzufassen. `group_by` steht für die Idee, ‘Zahlensäcke’ (Verteilungen) in Subgruppen aufzuteilen. `mutate` transformiert Daten. `n` zählt Häufigkeiten.

Ein weiterer zentraler Gedanken der Deskriptivstatistik ist es, dass es beim Zusammenfassen von Daten nicht reicht, sich auf den Mittelwert oder eine (hoffentlich) ‘repräsentative’ Zahl zu verlassen. Man braucht auch einen Hinweis, wie unterschiedlich die Daten sind. Entsprechend spricht man von zwei Hauptbereichen der deskriptiven Statistik.

Die deskriptive Statistik hat zwei Hauptbereiche: Lagemaße und Streuungsmaße.

Lagemaße geben den “typischen”, “mittleren” oder “repräsentativen” Vertreter der Verteilung an. Bei den Lagemaßen denkt man sofort an das *arithmetische Mittel* (synonym: Mittelwert, arithmetisches Mittel; häufig als \bar{X} abgekürzt; `mean`). Ein Nachteil von Mittelwerten ist, dass sie *nicht robust* gegenüber Extremwerten sind: Schon ein vergleichsweise großer Einzelwert kann den Mittelwert stark verändern und damit die Repräsentativität des Mittelwerts für die Gesamtmenge der Daten in Frage stellen. Eine robuste Variante ist der *Median* (M_d ; `median`).

¹²`sd(stats_test$score)`

¹³die `sd` von `score`

Ist die Anzahl der (unterschiedlichen) Ausprägungen nicht zu groß im Verhältnis zur Fallzahl, so ist der *Modus* eine sinnvolle Statistik; er gibt die häufigste Ausprägung an¹⁴.

Streuungsmaße geben die Unterschiedlichkeit in den Daten wieder; mit anderen Worten: sind die Daten sich ähnlich oder unterscheiden sich die Werte deutlich? Zentrale Statistiken sind der *mittlere Absolutabstand* (MAA; engl. mean absolute deviation, MAD),¹⁵ die *Standardabweichung* (sd; **sd**), die *Varianz* (Var; **var**) und der *Interquartilsabstand* (IQR; **IQR**). Da nur der IQR *nicht* auf dem Mittelwert basiert, ist er robuster als Statistiken, die sich aus dem Mittelwert ergeben. Beliebige Quantile bekommt man mit dem R-Befehl **quantile**. Möchte man z.B. Q1, Median und Q3, so kann man das so sagen: **quantile(x, probs = c(.25, .50, .75))**, wobei **x** eine Spalte (ein Vektor) ist.

Der Befehl **summarise** eignet sich, um deskriptive Statistiken auszurechnen.

```
summarise(stats_test, mean(score))
#>   mean(score)
#> 1      31.1
summarise(stats_test, sd(score))
#>   sd(score)
#> 1      5.74
summarise(stats_test, aad(score))  # aus Paket 'lsr'
#>   aad(score)
#> 1      4.84
```

Natürlich könnte man auch einfacher schreiben:

```
mean(stats_test$score)
#> [1] 31.1
median(stats_test$score)
#> [1] 31
aad(stats_test$score)
#> [1] 4.84
```



Viele R-Befehle der deskriptiven Statistik sind im Standard so eingestellt, dass sie **NA** zurückliefern, falls es in den Daten fehlende Werte gibt. Das ist einerseits informativ, aber oft unnötig. Mit dem Parameter **na.rm = TRUE** kann man dieses Verhalten abstellen.

Tipp: Mit dem Befehl **df <- na.omit(df)** entfernen Sie alle fehlenden Werte aus **df**.

¹⁴Der *Modus* ist im Standard-R nicht mit einem eigenen Befehl vertreten. Man kann ihn aber leicht von Hand bestimmen; s.u. Es gibt auch einige Pakete, die diese Funktion anbieten: z.B. <https://cran.r-project.org/web/packages/modes/index.html>

¹⁵Der *MAD* ist im Standard-R nicht mit einem eigenen Befehl vertreten. Es gibt einige Pakete, die diese Funktion anbieten: z.B. **lsr::aad** (absolute average deviation from the mean) <https://artax.karlin.mff.cuni.cz/r-help/library/lsr/html/aad.html>

`summarise` liefert aber im Unterschied zu `mean` etc. immer einen Dataframe zurück. Da der Dataframe die typische Datenstruktur ist, ist es häufig praktisch, wenn man einen Dataframe zurückbekommt, mit dem man weiterarbeiten kann. Außerdem lassen `mean` etc. keine Gruppierungsoperationen zu; über `group_by` kann man dies aber bei `dplyr` erreichen. Alternativ ist die Funktion `mosaic::inspect` hilfreich; sie gibt einen Überblick über alle Variablen eines Dataframes und unterscheidet dabei quantitative von qualitativen Variablen. Allerdings lässt `inspect` keine Subgruppenvergleiche zu; insbesondere versteht `inspect` nicht `group_by`.

7.5.2 Viele Statistiken auf einmal mit `desctable`

Möchte man die “üblichen Verdächtigen” an deskriptiven Statistiken mit einem Befehl bekommen, so ist der Befehl `desctable::desctable` hilfreich:

```
stats_test2 <- dplyr::select(stats_test, -date_time)
desctable(stats_test2)
#>           N Med IQR
#> 1 row_number 306 154 152
#> 2 bestanden 306   1   0
#> 3 study_time 238   3   2
#> 4 self_eval 238   5   3
#> 5 interest 238   3   2
#> 6 score 306   31   9
```

Die Variable `date_time` wurde deswegen entfernt, weil sie vom Typ `factor` ist. Wenn es Faktorvariablen gibt, werden die metrischen Werte von `desctable` für jede Faktorstufe getrennt ausgewiesen. Das wäre hier aber nicht sinnvoll. `desctable` wählt die passenden Statistiken selber aus. Bei metrischen Variablen wird zum Beispiel nur dann der Mittelwert und die SD angezeigt, wenn die Variablen normalverteilt sind. Man kann die Auswahl der Statistiken mit dem Parameter `stats` steuern; folgende Möglichkeiten stehen zur Verfügung: `stats_auto`, `stats_normal`, `stats_nonnorm`, `stats_default`.

```
stats_test2 <- dplyr::select(stats_test, -date_time)
desctable(stats_test2, stats = stats_normal)
#>           N Mean/%      sd
#> 1 row_number 306 153.500 88.479
#> 2 bestanden 306   0.817  0.387
#> 3 study_time 238   2.912  1.116
#> 4 self_eval 238   5.382  2.455
#> 5 interest 238   3.214  1.390
#> 6 score 306   31.121  5.744
```

`stats_normal` gibt Statistiken unter der Annahme von Normalverteilung an.

Möchte man Statistiken nach eigenem Gusto präsentiert bekommen bei `descstable`, so kann man dies so einstellen¹⁶:

```
stats_yeah = function(data) {
  list(N=length, 'Mean/%' = is.factor ~ percent | mean, sd = is.factor ~ NA | sd, Med =
}

descitable(stats_test2, stats = stats_yeah)
#>           N  Mean/%      sd Med IQR
#> 1 row_number 306 153.500 88.479 154 152
#> 2 bestanden 306   0.817  0.387   1   0
#> 3 study_time 238   2.912  1.116   3   2
#> 4 self_eval 238   5.382  2.455   5   3
#> 5 interest 238   3.214  1.390   3   2
#> 6 score 306 31.121  5.744  31   9
```

Natürlich kann man auch Subgruppen so vergleichen:

```
stats_test %>%
  select(-c(row_number, date_time)) %>%
  group_by(bestanden) %>%
  descitable
#>           bestanden: FALSE (n=56) / N bestanden: FALSE (n=56) / Med
#> 1 study_time                               49                         2
#> 2 self_eval                                49                         3
#> 3 interest                                 49                         3
#> 4 score                                    56                         23
#>           bestanden: FALSE (n=56) / IQR bestanden: TRUE (n=250) / N1
#> 1                                         1                           189
#> 2                                         3                           189
#> 3                                         2                           189
#> 4                                         3                           250
#>           bestanden: TRUE (n=250) / Med1 bestanden: TRUE (n=250) / IQR1 tests / p
#> 1                                         3                           2  2.04e-05
#> 2                                         6                           4  1.26e-09
#> 3                                         3                           2  8.88e-02
#> 4                                         33                          8  1.11e-31
#>           tests / test
#> 1 wilcox.test
#> 2 wilcox.test
#> 3 wilcox.test
#> 4 wilcox.test
```

¹⁶Die Idee kommt von Norman Markgraf

7.6 Bedingte Analysen mit den Suffixen von dplyr

Seit der Version 0.7 haben alle Hauptverben von dplyr eine Reihe von Suffixen:

- `_if`
- `_all`
- `_at`

Einige Beispiele machen den Nutzen klar: `summarise_all` fasst alle Spalten zusammen (daher ‘all’). `mutate_at` formt nur bestimmte Variablen um. `summarise_if` fasst nur Spalten zusammen, für die eine bestimmte Bedingung zutrifft. In einigen Fällen sind diese Suffixe praktisch:

7.6.1 Suffix `_if`

```
data(stats_test, package = "pradadata")
```

```
stats_test %>%
  summarise_if(is.numeric, mean)
#>   row_number study_time self_eval interest score
#> 1           154        NA         NA       NA  31.1
```

Diese Syntax lässt sich fast wortwörtlich in Pseudocode übersetzen:



Nimm die Tabelle “stats_test” UND DANN
fasse eine jede Spalte zusammen WENN
die Spalte numerisch ist ACH JA
fasse diese Spalten mithilfe des Mittelwerts zusammen.

7.6.2 Suffix `_all`

Recht elegant lassen sich mit dem Suffix `_all` die Anzahl der fehlenden Werte pro Spalte zusammenfassen.

```
stats_test %>%
  summarise_all(funs(is.na(.) %>% sum))
#>   row_number date_time bestanden study_time self_eval interest score
#> 1           0         0         0        68        68        68       0
```

Wir übersetzen wieder:



Nimm die Tabelle “stats_test” UND DANN
fasse JEDE Spalte zusammen UND ZWAR
mit dieser Funktion (‘funs’) UND ZWAR
mit dem Ergebnis dieser zwei Schritte: (Finde alle NAs und summiere sie).

7.6.3 Suffix `_at`

Beim Suffix `_at` gibt man Spaltennamen an. Wie immer bei dplyr kann man die Spaltennummer oder ihren Namen angeben.

Sagen wir, wir möchten eine Spalte erzeugen für jede Selbsteinschätzungsvariable; in diesen neuen Spalte soll jeweils der maximale Wert der zugrundeliegenden Selbsteinschätzungsvariablen stehen. Das könnte so aussehen:

```
stats_test %>%
  mutate_at(.vars = vars(study_time, self_eval), max, na.rm = TRUE) %>% head
#> # A tibble: 6 x 7
#>   date_time study_time self_eval interest score row_number
#>   <chr>       <dbl>      <dbl>    <int>  <dbl>      <int>
#> 1 19.06.2015 11:14:02     15        80      NA  0.579       1
#> 2 19.06.2015 11:17:06     15        80      NA  0.700       2
#> 3 19.06.2015 11:18:08     15        80      NA  0.400       3
#> 4 19.06.2015 11:18:42     15        80      NA  0.500       4
#> 5 19.06.2015 11:19:39     15        80      NA  0.725       5
#> 6 19.06.2015 11:19:43     15        80      NA  0.850       6
#> # ... with 1 more variables: bestanden <chr>
```

Als ersten Parameter (nach dem Dataframe, der implizit über die Pfeife übergeben wird) erwarten dplyr-Verben mit dem Suffix `_at` den Parameter `.vars`, also den Hinweis, welche Variablen wir einbeziehen wollen. Wie immer, wenn wir die Standard-Reihenfolge beachten, brauchen wir die Parameter nicht zu benennen (wir könnten als `.vars = weglassen`). Hingegen brauchen wir `vars()`, um dplyr zu sagen, von wo bis wo die Liste der relevanten Spalten geht. Der nächste Parameter ist dann der oder die Befehle, die wir nutzen möchten, um die Spalte zu transformieren, z.B. `max`. Etwaige Parameter, wie `na.rm = TRUE`, werden einfach ganz hinten angefügt.

Schauen wir uns ein realistischeres Beispiel an. Sagen wir, wir möchten die Umfrage-Items auf eine Skala von 0 bis 1 standardisieren. Dazu können wir jeden Wert durch den (theoretischen) Maximalwert der Skala teilen. Mit dplyr kann das so aussehen:

```
stats_test %>%
  drop_na %>%
  mutate_at(.vars = vars(study_time, self_eval, interest), .funs = funs(prop = ./max(.)))
#>   row_number      date_time bestanden study_time self_eval interest
#> 1           1 05.01.2017 13:57:01      ja        5       8       5
#> 2           2 05.01.2017 21:07:56      ja        3       7       3
#> 3           3 05.01.2017 23:33:47      ja        5      10       6
#> 4           4 06.01.2017 09:58:05     nein        2       3       2
#> 5           5 06.01.2017 14:13:08      ja        4       8       6
#> 6           8 06.01.2017 17:24:53     nein        2       5       3
#>   score study_time_prop self_eval_prop interest_prop
#> 1    29            1.0            0.8        0.833
#> 2    29            0.6            0.7        0.500
#> 3    40            1.0            1.0        1.000
#> 4    18            0.4            0.3        0.333
#> 5    34            0.8            0.8        1.000
#> 6    24            0.4            0.5        0.500
```

Die Funktion `funs` erzeugt benannte Funktionen; dieser Namen werden dann den zu erstellenden Spalten als Suffix angefügt. `drop_na()` löscht Zeilen mit fehlenden Werten. Man kann `drop_na()` die Spalten einzugrenzen, die beim Suchen fehlender Werten berücksichtigt werden. Gibt man keine Spalten an, so werden alle Spalten durchsucht.

7.7 Tabellen zusammenführen

Angenommen, Sie betreiben einen Webshop, in dem Sie R-Devotionalien verkaufen. Alle Verkäufe halten Sie in einer Tabelle fest (Ihre “Verkaufstabelle”), in der u.a. auch die Kundennummer des Käufers auftaucht. Die Beobachtungseinheit in dieser Tabelle sei eine Artikel; besteht ein Einkauf aus mehreren Artikeln, so würden entsprechend viele Zeilen erscheinen. Natürlich könnten Sie in dieser Tabelle auch die Informationen des jweileigen Käufers (Name, Adresse, Lieblings-Devotionalien) festhalten. Aber effizienter ist es, diese Käufer-Informationen in einer eigenen Tabelle, in der nur die Käufer-Daten aufgeführt sind, festzuhalten. Andernfalls würde Ihre Verkaufstabelle viele redundante Informationen aufweisen. Da R nicht darauf ausgelegt ist, mehrere unterschiedliche Tabellen gleichzeitig im Blick zu halten, ist es häufig sinnvoll, für Ihre Analyse die Verkaufstabelle mit der Kundentabelle zusammenzuführen. So könnte es sinnvoll sein, an die Verkaufstabelle einige Spalten aus der Kundentabelle “anzuhängen”, um zu wissen, welcher Kunde das Produkt gekauft hat bzw. welches Kundenprofil hinter dem Kauf steht. Dieses “Zusammenfügen” von Tabellen (die hier wie immer als Dataframes repräsentiert sind), nennt man auch *merge* oder *join*.

Betrachten wir als Beispiel den Datensatz `flights` aus dem Paket `nycflights13`. Dort sind

Flüge aufgeführt, aber der Name der Airline nur mit einem Kürzel versehen.

```
data(flights)
flights %>%
  select(carrier) %>%
  head(3)
#> # A tibble: 3 x 1
#>   carrier
#>   <chr>
#> 1 UA
#> 2 UA
#> 3 AA
```

Den kompletten Namen der Airline finden wir Datensatz `airlines`.

```
data(airlines)
head(airlines, 3)
#> # A tibble: 3 x 2
#>   carrier           name
#>   <chr>            <chr>
#> 1 9E    Endeavor Air Inc.
#> 2 AA    American Airlines Inc.
#> 3 AS    Alaska Airlines Inc.
```

Unser Ziel ist es jetzt, dem Datensatz `flights` die Spalte `name` aus dem Datensatz `airlines` hinzuzufügen und natürlich so, dass jedes Kürzel dem richtigen Namen zugeordnet wird. Nach diesem *Join* wird `flights` also um eine Spalte breiter sein; man spricht daher auch von einem *mutating join*. Um diese beiden Datensätze sozusagen zu verheiraten, müssen wir angeben, anhand welchem Kriterium, d.h. Spalte, wir die Tabellen aufeinander beziehen. In diesem Fall ist das Vereinigungskriterium (Matching-Kriterium) das Kürzel der Airline (`carrier`). Man beachte, dass diese Spalte in beiden Tabellen vorkommen muss.

```
flights %>%
  inner_join(airlines, by = "carrier") -> flights_joined

head(flights_joined$name)
#> [1] "United Air Lines Inc." "United Air Lines Inc."
#> [3] "American Airlines Inc." "JetBlue Airways"
#> [5] "Delta Air Lines Inc."   "United Air Lines Inc."
```

Wichtig ist noch hinzuzufügen, dass bei `inner_join` Zeilen, die keinen (Heirats-)partner in der anderen Tabellen fanden, *nicht* übernommen, also fallengelassen werden. Das macht `inner_join` häufig ungeeignet, da man zu leicht Daten verliert.



Bei einem *inner join* werden nur Zeilen behalten, die in beiden Tabellen vorkommen. Die resultierende Tabelle ist also nie länger, höchstens kürzer als eine oder beide Ausgangstabellen.

Allgemein kann man die Syntax von `inner_join` so schreiben:



```
x %>% inner_join(y, by = "key")
```

Dabei ist `x` die “linke” Tabelle, `y` die “rechte” Tabelle und mit `key` wird die Spalte benannt, die das Matching-Kriterium enthält. Würde die Matching-Spalte in X `key_X` heißen und in Y `key_Y`, so würde man schreiben: `inner_join(y, by = ("key_X", "key_Y"))`. Für andere Join-Arten ist die Syntax analog.

Ein anderer Join, der häufig verwendet wird, ist `left_join`. Ein `left_join` behält alle Beobachtungen von X (die “linke” Tabelle), auch wenn es kein Match mit Y gibt. Hingegen werden Beobachtungen aus Y entfernt, falls Sie keine Entsprechung in X haben. Bei `right_join` gilt Ähnliches für Y (die “rechte” Tabelle). Bei `full_join` hingegen werden alle Beobachtungen sowohl aus X als auch Y behalten, auch wenn sie keine Entsprechung in der jeweils anderen Tabelle haben.

7.8 Verweise

- Die offizielle Dokumentation von `dplyr` findet sich hier: <https://cran.r-project.org/web/packages/dplyr/dplyr.pdf>.
- Eine schöne Demonstration wie mächtig `dplyr` ist findet sich hier: <http://bit.ly/2kX91vC>.
- Die GUI “exploratory” ist ein “klickbare” Umsetzung von `dplyr` and friends; mächtig, modern und sieht cool aus: <https://exploratory.io>.
- *R for Data Science* bietet umfangreiche Unterstützung zu diesem Thema (Wickham und Grolemund 2016).

Kapitel 8

Praxisprobleme der Datenaufbereitung



Lernziele:

- Typische Probleme der Datenaufbereitung kennen.
- Typische Probleme der Datenaufbereitung bearbeiten können.

Laden wir zuerst die benötigten Pakete und Daten:

```
library(corr) # Korrelationsmatrix
library(car) # Umkodieren
#library(pradadata) # optional: Daten 'extra' und 'stats_test'
library(tidyverse) # Datenjudo
```

Stellen wir einige typische Probleme des Datenjudo (genauer: der Datenaufbereitung) zusammen. Probleme heißt hier nicht, dass es etwas Schlimmes passiert ist, sondern es ist gemeint, wir schauen uns ein paar typische Aufgabenstellungen an, die im Rahmen der Datenaufbereitung häufig anfallen.

8.1 Datenaufbereitung

8.1.1 Auf fehlende Werte prüfen

Das geht recht einfach mit `summary(mein_dataframe)`. Der Befehl liefert für jede Spalte des Dataframe `mein_dataframe` die Anzahl der fehlenden Werte zurück.

```
data(stats_test, package = "pradadata")
summary(stats_test)
```

8.1.2 Fälle mit fehlenden Werte löschen

Weist eine Variable (Spalte) “wenig” fehlende Werte auf, so kann es schlau sein, nichts zu tun. Eine andere Möglichkeit besteht darin, alle entsprechenden Zeilen zu löschen. Man sollte aber schauen, wie viele Zeilen dadurch verloren gehen.

```
# Ursprünglich Anzahl an Fällen (Zeilen)
nrow(stats_test)
#> [1] 2263

# Nach Umwandlung in neuen Dataframe
stats_test %>%
  na.omit -> stats_test_na OMIT
nrow(stats_test_na OMIT)
#> [1] 1627

# Nur die Anzahl der bereinigten Daten
stats_test %>%
  na.omit %>%
  nrow()
#> [1] 1627
```



Bei mit der Pfeife verketteten Befehlen darf man für Funktionen die runden Klammern weglassen, wenn man keinen Parameter schreibt. Also ist `nrow` (ohne Klammern) erlaubt bei `dplyr`, wo es eigentlich `nrow()` heißen müsste. Sie dürfen die Klammern natürlich schreiben, aber sie müssen nicht.

Hier verlieren wir 636 Zeilen, das verschmerzen wir.

Welche Zeilen verlieren wir eigentlich? Lassen wir uns nur die *nicht*-kompletten Fälle anzeigen (und davon nur die ersten paar):

```
stats_test %>%
  filter(!complete.cases(.)) %>%
  head()
```



Man beachte, dass der Punkt . für den Datensatz steht, wie er vom letzten Schritt weitergegeben wurde. Innerhalb einer dplyr-Befehls-Kette können wir den Datensatz, wie er im letzten Schritt beschaffen war, stets mit . ansprechen; ganz praktisch, weil schnell zu tippen. Natürlich könnten wir diesen Datensatz jetzt als neues Objekt speichern und damit weiter arbeiten. Das Ausrufezeichen ! steht für logisches „Nicht“. Mit head bekommt man nur die ersten paar Fälle (6 im Standard) angezeigt, was oft reicht für einen Überblick.

In Pseudo-Syntax liest es sich so:



Nehme den Datensatz `stats_test` UND DANN...
filtere die nicht-kompletten Fälle

8.1.3 Fehlende Werte zählen

Wie viele fehlende Wert weist eine Spalte auf?

```
stats_test %>%
  mutate(self_eval_NA = is.na(self_eval)) %>%
  summarise(self_eval_NA_sum = sum(self_eval_NA))
#>   self_eval_NA_sum
#> 1                 68
```



Nimm die Tabelle `stats_test` UND DANN
berechne eine Spalte `self_eval_NA` in der steht, ob bei `self_eval` ein NA steht UND
DANN
zähle die NAs zusammen. FERTIG.

Praktischer ist es natürlich, mehrere Spalten auf einmal nach fehlenden Werten zu durchkämmen. Betrachten wir dazu den Datensatz extra:

```
data(extra, package = "pradadata")
```

```
extra %>%
  as_tibble %>%
  rownames_to_column %>%
  select(i01:i10, rowname) %>%
  gather(key = item, value = response, -rowname) %>%
  filter(is.na(response)) %>%
```

```
count()
#> # A tibble: 1 x 1
#>      n
#>   <int>
#> 1    72
```



Eine “lange Pfeife”, also eine lange Sequenz verbundener R-Befehle kann manchmal kompliziert sein. Wenn Sie unsicher sind, was ein bestimmter Befehl vor hat, dann markieren Sie nur eine oder eine Auswahl an Zeilen. Führen Sie diesen Teil aus und arbeiten Sie sich auf diese Weise vor.

In 72 Zellen fehlen Werte; dabei wurden die Spalten `i01` bis `i10` berücksichtigt. Schauen wir etwas genauer hin, so dass wir genau wissen, bei welchem Item (Spalte) und welcher Zeile (Teilnehmer) ein Wert fehlt.

```
extra %>%
  filter(!complete.cases(.)) %>% head
#> # A tibble: 6 x 34
#>   timestamp code   i01   i02r   i03   i04   i05   i06r   i07
#>   <chr>     <chr> <int> <int> <int> <int> <int> <int> <int>
#> 1 11.03.2015 19:17:48 HSC     3     3     3     3     4     4     3
#> 2 11.03.2015 19:18:05 ERB     2     2     1     2     3     2     2
#> 3 11.03.2015 19:18:09 ADP     3     4     1     4     4     1     3
#> 4 11.03.2015 19:18:19 KHB     3     3     2     4     3     3     3
#> 5 11.03.2015 19:18:19 PTG     4     3     1     4     4     3     4
#> 6 11.03.2015 19:18:23 ABL     3     2     1     4     2     3     4
#> # ... with 25 more variables: i08 <int>, i09 <int>, i10 <int>,
#> #   n_facebook_friends <dbl>, n_hangover <dbl>, age <int>, sex <chr>,
#> #   extra_single_item <int>, time_conversation <dbl>, presentation <chr>,
#> #   n_party <dbl>, clients <chr>, extra_vignette <chr>, i21 <chr>,
#> #   extra_vignette2 <int>, major <chr>, smoker <chr>, sleep_week <dbl>,
#> #   sleep_wend <int>, clients_freq <dbl>, extra_mean <dbl>,
#> #   extra_md <dbl>, extra_aad <dbl>, extra_mode <dbl>, extra_iqr <dbl>
```

Wie viele Fälle bleiben uns eigentlich übrig, wenn wir die alle Zeile mit fehlenden Werten entfernen?

```
extra %>%
  select(i01:i10) %>%
  filter(complete.cases(.)) %>% nrow
#> [1] 802
```

OK, das ist ja kaum weniger als die ursprüngliche Zeilenzahl. Hier kann man recht beruhigt die Zeilen mit fehlenden Werten löschen. Dazu ziehen wir uns einen Vektor mit allen zu löschen Zeilen und filtern dann entsprechend:

```
extra %>%
  filter(complete.cases(..)) %>%
  pull(code) %>%
  unique -> I_am_complete

extra %>%
  filter(code %in% I_am_complete) -> extra_no_nas
```

8.1.4 Fehlende Werte ggf. ersetzen

Ist die Anzahl der fehlenden Werte zu groß, als dass wir es verkraften könnten, die Zeilen zu löschen, so können wir die fehlenden Werte ersetzen. Allein, das ist ein weites Feld und übersteigt den Anspruch dieses Kurses¹. Eine einfache, aber nicht die beste Möglichkeit, besteht darin, die fehlenden Werte durch einen repräsentativen Wert, z.B. den Mittelwert der Spalte, zu ersetzen.

```
stats_test %>%
  mutate(interest = replace($.interest,
                           is.na($.interest),
                           mean($.interest,
                                na.rm = TRUE))) -> stats_test

sum(is.na(stats_test$interest))
#> [1] 0
```

`replace`² ersetzt Werte aus dem Vektor `$.interest` alle Werte, für die `is.na($.interest)` wahr ist, bei Zeilen mit fehlenden Werten in dieser Spalte also. Der Punkt `.` steht für den Datensatz (so wie er aus dem letzten Pfeifenschritt hervorkam). Diese Werte werden durch den Mittelwert der Spalte ersetzt³. Der Punkt `.` ersetzt den Daten der Tabelle (wir hätten aber auch den Namen der Tabelle ausschreiben können).

¹Das sagen Autoren, wenn sie nicht genau wissen, wie etwas funktioniert.

²aus dem “Standard-R”, d.h. Paket “base”.

³Hier findet sich eine ausführlichere Darstellung: https://sebastiansauer.github.io/checklist_data_cleansing/index.html

8.1.5 “-99” in NA umwandeln

Manchmal kommt es vor, dass fehlende Werte nicht durch leere Zellen bzw. Rs Symbol dafür (NA) kodiert sind, sondern durch ein anderes Symbol. ‘-99’ ist ein Kandidat, den man immer mal wieder trifft. Wie erklären wir R, dass ‘-99’ zu NA umkodiert werden soll. So:

```
stats_test$study_time[1] <- -99
head(stats_test)

stats_test %>%
  mutate_if(is_numeric, na_if, -99) %>% head
```

Der Befehl sagt in etwa “Transformiere jede Spalte, wenn sie numerisch ist. Und die Transformation, die du anwenden sollst, R, ist ‘Erzeuge NA, wenn...’. Und dieses ‘wenn’ ist ‘-99’ ”.

8.1.6 Doppelte Fälle löschen

Es kann passieren, dass sich doppelte Fälle (*Dubletten*) im Datensatz finden; z.B. durch einen Fehler beim Erstellen oder Einlesen der Daten. In dem Fall wären *komplette Zeilen* gleich. Auf ähnliche Weise kann es passieren, dass der Nutzer Alois zwei Mal auf unsere Umfrage antwortet. Wenn sein Gedächtnis nicht so gut ist, ist es gut möglich, dass nicht jede Antwort beim zweiten Mal gleich wie beim ersten Mal beantwortet wurde. In dem Fall wären nur *Teile von Zeilen* gleich. In beiden Fällen werden Sie wahrscheinlich die Dubletten wieder loswerden wollen. Schauen wir uns letzteren Fall zuerst an:

```
stats_test %>%
  mutate(is_duplicate = duplicated(date_time)) %>%
  filter(!is_duplicate) %>%
  nrow
#> [1] 2262
```

Die Funktion `duplicated` gibt für jedes Element eines Vektors zurück, ob das Objekt eine Dublette ist, d.h. ob der betreffende Wert in vorherigen Elementen schon einmal aufgetaucht ist. In dem Datensatz gibt es offenbar keine Dubletten: Alle Werte von `date_time` sind verschieden. Schauen wir uns jetzt den Fall an, wenn zwei Fälle (Zeilen) komplett identisch sind. Schneiden wir zuerst eine Scheibe von 2 Zeilen (1:2) heraus und fügen diesen Schnipsel hinten wieder an, damit wir 2 Dubletten im Datensatz haben. Die Gesamtlänge des Datensatzes sollte dann um 2 steigen. `distinct` filtert die nicht-gleichen Zeilen zurück (die Dubletten werden gelöscht). Damit sollten der resultierende Datensatz wieder auf die ursprüngliche Länge schrumpfen:

```
stats_test %>%
  slice(1:2) -> stats_test_ausschnitt

stats_test %>%
  bind_rows(stats_test_ausschnitt) -> temp
```

8.1.7 Spaltennamen ändern

Es gibt, wie für fast alles, mehrere Wege in R zu diesem Ziel. Möchte man alle Spaltennamen ändern, so ist der Befehl `names` ganz praktisch: `names(mein_df) <- c("Var1", "Var2")`. Möchte man nur einzelne Spaltennamen ändern, so kann man den Befehl `rename` aus dplyr verwenden; die Syntax lautet: `rename(mein_df, neuer_name = alter_name)`.

```
stats_test %>%
  rename(Punkte = score) -> stats_test
```



Übrigens kann man den gleichen Effekt mit `dplyr::select` auch erreichen; genauer den fast gleichen. Probieren Sie mal `rename(stats_test, Punkte = score)` und vergleichen Sie das Ergebnis.

Es gibt auch weniger schöne Wege, um Spalten umzubenennen: `names(stats_test)[names(stats_test) == "bestanden"] <- "jippiejey"`.

8.1.8 Nach Fehlern suchen

Leicht schleichen sich Tippfehler oder andere Fehler ein. Man sollte darauf prüfen; so könnte man sich ein Histogramm ausgeben lassen pro Variable, um “ungewöhnliche” Werte gut zu erkennen. Meist geht das besser als durch das reine Betrachten von Zahlen. Gibt es wenig unterschiedliche Werte, so kann man sich auch die unterschiedlichen Werte ausgeben lassen.

```
stats_test %>%
  dplyr::count(interest, sort = TRUE) %>% head
#> # A tibble: 6 x 2
#>   interest     n
#>   <dbl> <int>
#> 1     3.09    619
#> 2     3.00    432
#> 3     4.00    319
#> 4     2.00    300
```

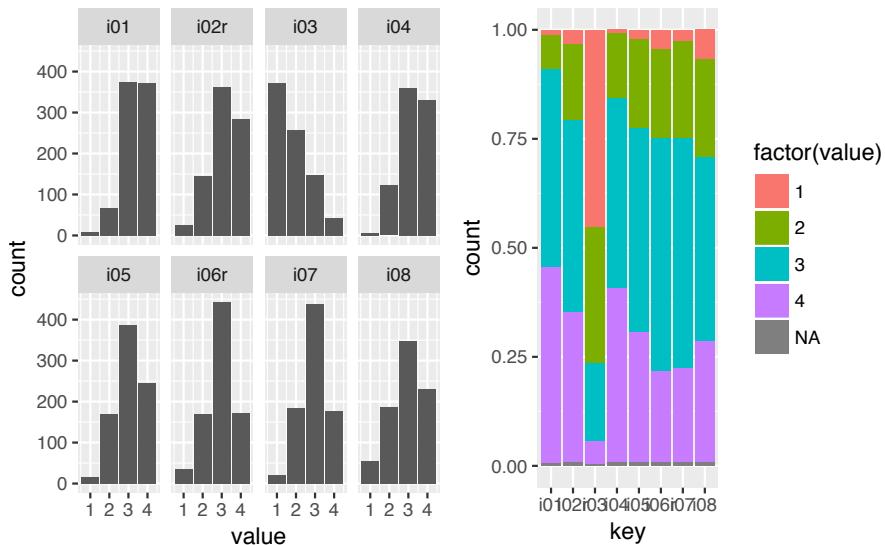


Abbildung 8.1: Ausreißer identifizieren

```
#> 5      1.00    280
#> 6      5.00    250
```

Da in der Umfrage nur ganze Zahlen von 1 bis 5 abgefragt wurden, ist die 3.21... auf den ersten Blick suspekt. In diesem Fall ist aber alles ok, da wir diesen Wert selber erzeugt haben.

Schauen wir uns noch die Verteilungen der Item-Antworten an; sehen die plausibel aus? Einige Verteilungen sind ordentlich schief. Man könnte räsonnieren, dass man sich damit in guter Gesellschaft befindet - mit Blick auf andere Umfragen. Aber die ungleiche Fallzahl pro Itemantwort und die Abweichung von der Normalverteilung könnten später Probleme bereiten. Es würde sich anbieten, die schiefen Items zu überarbeiten (s. Abbildung 8.1).

Findet man ‘merkwürdige’ (unplausible) Werte, so kann es sinnvoll sein, diese Werte herauszunehmen (im Detail eine schwierige Entscheidung). Besser als die Zeilen zu löschen, ist es oft, diese Werte in NA umzuwandeln. Sagen wir, wir möchten alle Fälle mit `score < 21` entfernen bzw. in NA umwandeln.

```
stats_test %>%
  mutate(score_bereinigt = replace(. $score,
                                    . $score < 21,
                                    NA)) -> stats_test
```

8.1.9 Ausreißer identifizieren

Ähnlich zu Fehlern, steht man Ausreißer häufig skeptisch gegenüber. Allerdings kann man nicht pauschal sagen, das Extremwerte entfernt werden sollen: Vielleicht war jemand in der

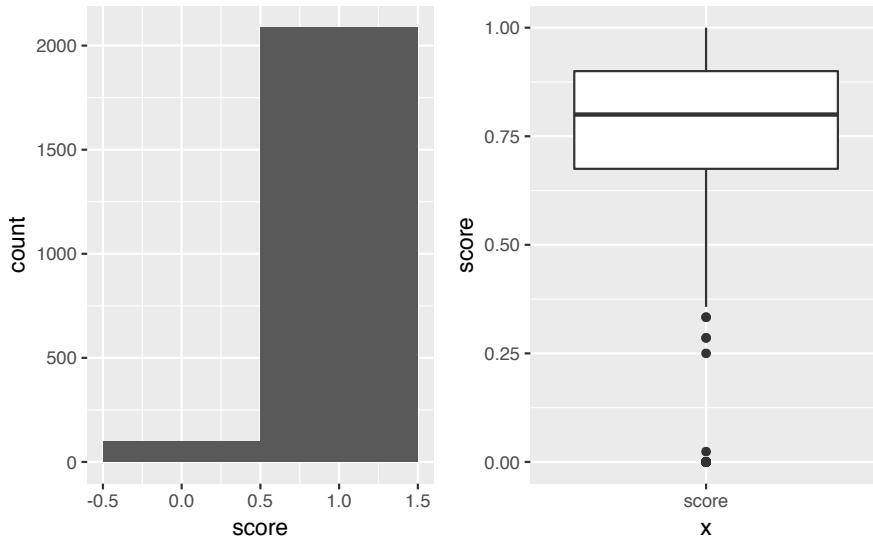


Abbildung 8.2: Ausreißer identifizieren

Stichprobe wirklich nur 1.20m groß? Hier gilt es, begründet und nachvollziehbar im Einzelfall zu entscheiden. Histogramme und Boxplots sind wieder ein geeignetes Mittel, um Ausreißer zu finden (vgl. Abb. 8.2).

Definieren wir Fälle, die mehr als 3 SD vom Mittelwert entfernt (d.h. z-Werte größer als 3 und kleiner als -3) sind als Extremfälle:

```
stats_test %>%
  mutate(score_z = (score - mean(score, na.rm = TRUE)) / sd(score, na.rm = TRUE)) %>%
  mutate(is_extreme = if_else(abs(score_z) > 3, TRUE, FALSE)) %>%
  count(is_extreme)
#> # A tibble: 3 x 2
#>   is_extreme     n
#>   <lgcl> <int>
#> 1 FALSE      2165
#> 2 TRUE       20
#> 3 NA        78
```

Eine (kleine) Zahl an Extremwerten haben wir so identifiziert; diese könnten wir jetzt näher betrachten.

8.1.10 Hochkorrelierte Variablen finden

Haben zwei Leute die gleiche Meinung, so ist einer von beiden überflüssig - wird behauptet. Ähnlich bei Variablen; sind zwei Variablen sehr hoch korreliert ($>.9$, als grober (!) Richtwert), so bringt die zweite kaum Informationszuwachs zur ersten. Und kann z.B. ausgeschlossen werden.

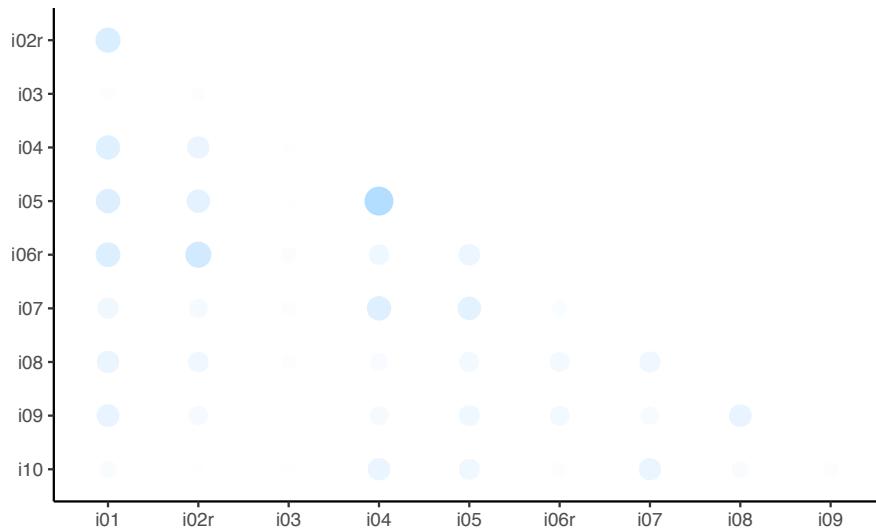


Abbildung 8.3: Ein Korrelationsplot

Nehmen wir dazu den Datensatz `extra` her und berechnen wir eine Korrelationsmatrix, d.h. wir korrelieren alle Variablen paarweise miteinander:

```
data(extra, package = "pradadata")
extra %>%
  dplyr::select(i01:i10) %>% # Wähle die Variablen von i1 bis i10 aus
  corrr::correlate() -> km    # Korrelationsmatrix berechnen
```

Betrachten Sie die Korrelationstabelle, was fällt Ihnen auf? [Sie ist quadratisch, auf der Hauptdiagonalen steht `NA`, und gespiegelt an der Hauptdiagonalen, d.h. das obere Dreieck der Matrix ist redundant.]

In diesem Beispiel sind keine Variablen sehr hoch korreliert. Wir leiten keine weiteren Schritte ein, abgesehen von einer Visualisierung (s. Abb. 8.3).

```
km %>%
  shave() %>% # Oberes Dreieck ist redundant, wird "abrasiert"
  rplot() # Korrelationsplot
```

Die Funktion `correlate` stammt aus dem Paket `corrr`⁴, welches vorher installiert und geladen sein muss. Hier ist die Korrelation nicht zu groß, so dass wir keine weiteren Schritte unternehmen. Hätten wir eine sehr hohe Korrelation gefunden, so hätten wir eine der beiden beteiligten Variablen aus dem Datensatz löschen können.

⁴<https://github.com/drsimonj/corrr>

8.1.11 z-Standardisieren

Für eine Reihe von Analysen ist es wichtig, die Skalierung der Variablen zur vereinheitlichen. Die z-Standardisierung ist ein übliches Vorgehen. Dabei wird der Mittelwert auf 0 transformiert und die SD auf 1; man spricht - im Falle von (hinreichend) normalverteilten Variablen - jetzt von der *Standardnormalverteilung*. Unterscheiden sich zwei Objekte A und B in einer standardnormalverteilten Variablen, so sagt dies nur etwas zur relativen Position von A zu B innerhalb ihrer Verteilung aus - im Gegensatz zu den Rohwerten.

```
extra %>%
  dplyr::select(i01, i02r) %>%
  scale() %>% # z-standardisieren
  head() # nur die ersten paar Zeilen abdrucken
#>      i01    i02r
#> [1,] -0.519 -0.134
#> [2,] -2.002 -1.383
#> [3,] -0.519  1.115
#> [4,] -0.519 -0.134
#> [5,]  0.964 -0.134
#> [6,] -0.519 -1.383
```

Dieser Befehl liefert z-standardisierte Spalten zurück. Kommoder ist es aber, alle Spalten des Datensatzes zurück zu bekommen, wobei zusätzlich die z-Werte aller numerischen Variablen hinzugekommen sind:

```
extra %>%
  mutate_if(is.numeric, funs("z" = scale)) %>%
  glimpse
```

Der Befehl `mutate` berechnet eine neue Spalte; `mutate_if` tut dies nur, wenn die Spalte numerisch ist. Die neue Spalte wird berechnet als z-Transformierung der alten Spalte; zum Spaltenname wird ein “_z” hinzugefügt. Natürlich hätten wir auch mit `select` “händisch” die relevanten Spalten auswählen können. Betrachten Sie die ersten paar Zeilen der neu erstellte Tabelle!⁵

8.1.12 Quasi-Konstante finden

Hier suchen wir nach Variablen (Spalten), die nur einen Wert oder zumindest nur sehr wenige verschiedene Werte aufweisen. Oder, ähnlich: Wenn 99.9% der Fälle nur von einem Wert bestritten wird. In diesen Fällen kann man die Variable als “Quasi-Konstante” bezeichnen.

⁵`head(extra)`

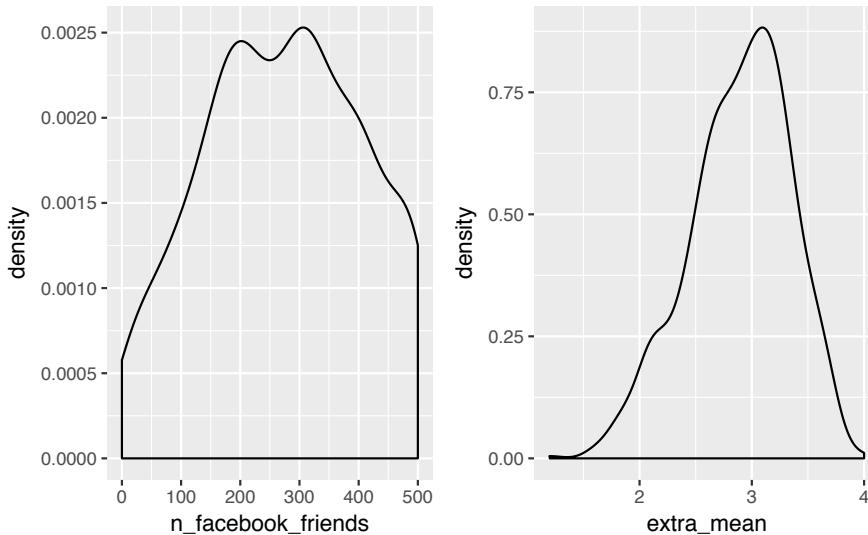


Abbildung 8.4: Visuelles Prüfen der Normalverteilung

Quasi-Konstanten sind für die Modellierung von keiner oder nur geringer Bedeutung; sie können in der Regel für weitere Analysen ausgeschlossen werden.

Haben wir z.B. nur Männer im Datensatz, so kann das Geschlecht nicht für Unterschiede im Einkommen verantwortlich sein. Besser ist es, die Variable Geschlecht zu entfernen. Auch hier sind Histogramme oder Boxplots von Nutzen zur Identifikation von (Quasi-)Konstanten. Alternativ kann man sich auch pro die Streuung (numerische Variablen) oder die Anzahl unterschiedlicher Werte (qualitative Variablen) ausgeben lassen:

```
IQR(extra$n_facebook_friends, na.rm = TRUE) # keine Konstante
#> [1] 300
n_distinct(extra$sex) # es scheint 3 Geschlechter zu geben...
#> [1] 3
```

8.1.13 Auf Normalverteilung prüfen

Einige statistische Verfahren gehen von normalverteilten Variablen aus, daher macht es Sinn, Normalverteilung zu prüfen. *Perfekte* Normalverteilung ist genau so häufig wie *perfekte* Kreise in der Natur. Entsprechend werden Signifikanztests, die ja auf perfekte Normalverteilung prüfen, *immer signifikant* sein, sofern die *Stichprobe groß* genug ist. Daher ist meist zweckmäßiger, einen graphischen “Test” durchzuführen: ein Histogramm, ein QQ-Plot oder ein Dichte-Diagramm als “glatt geschmigelte” Variante des Histogramms bieten sich an (s. Abb. 8.4).

Während die der mittlere Extraversionswert recht gut normalverteilt ist, ist die Anzahl der Facebookfreunde ordentlich (rechts-)schiefl. Bei schießen Verteilung können Transformationen Abhilfe schaffen; ein Thema, auf das wir hier nicht weiter eingehen.

8.1.14 Variablentypen ändern

Mitunter kommt es vor, dass man eine numerische Variable in eine Faktor- oder in eine Textvariable ändern will oder umgekehrt. Solange keine Faktorvariablen involviert sind, ist es einfach. Sagen wir, Sie wollen die Variable `interest` aus `stats_test` in eine Textvariable (`character`) umwandeln. `str(stats_test$interest)` zeigt Ihnen, dass die Variable metrisch ist, genauer, vom ganzzahlig (Typ `integer`). So geht's:

```
stats_test %>%
  mutate(interest = as.character(interest)) -> stats_test
```

Mit `as.numeric` geht es wieder retour. Warum würde man überhaupt den Variablentyp ändern wollen? Weil manche Befehle bestimmte Erwartungen an den Variablentyp haben, den Sie verarbeiten möchten. Ein Beispiel ist die Farbgebung bei `ggplot2`:

```
stats_test %>%
  ggplot(aes(x = interest, fill = as.numeric(bestanden))) +
  geom_bar()
```

Wird als Füllfarbe für `geom_bar` eine numerische Variable angegeben, so verweigert `ggplot` den Dienst. Wird aber eine nominale Variable (`character` oder `factor`) angegeben, so weiß `ggplot`, was zu tun ist.

Möchte man eine Faktor-Variable in eine numerische Variable umwandeln, muss man folgendes Wissen. Die erste Stufe eines Faktors wird intern mit 1 gespeichert, die zweite mit 2 und so weiter; etwa so:

```
bestanden = 1
durchgefallen = 2
nicht abgegeben = 3
```

Warum macht R das so? Diese Art von Kodierung spart Speicher (ja, auch heute kann das nützlich sein). Außerdem ist es manchmal praktisch, wenn vorab definiert ist, welche Werte zulässig sein. Eine Faktorvariable kann nur Werte speichern, für zu einer ihrer Stufen passen.

Möchten wir für 5 Studierende angeben, ob Sie eine Prüfung bestanden haben, durchgefallen sind oder nicht abgegeben haben, könnte das so aussehen:

```
Ergebnis <- c(1, 1, 1, 2, 3)
Ergebnis <- factor(Ergebnis, labels = c("bestanden", "durchgefallen", "nicht abgegeben"))
Ergebnis
#> [1] bestanden      bestanden      bestanden      durchgefallen
#> [5] nicht abgegeben
#> Levels: bestanden durchgefallen nicht abgegeben
```

Moment, bei Prof. Feistersack ist noch ein Student angetreten; tja, durchgefallen: `Ergebnis <- c(Ergebnis, "durchgefallen")`. Das ist eine gültige R-Syntax. Diese Syntax hingegen erzeugt eine Warnung: `Ergebnis[4] <- "fast geschafft"`.

Natürlich hindert uns niemand, als Stufen Zahlen zu definieren:

```
Ergebnis <- c(1, 1, 1, 2, 3)
Ergebnis <- factor(Ergebnis, labels = c("10", "20", "30"))
Ergebnis
#> [1] 10 10 10 20 30
#> Levels: 10 20 30
```

Möchte man jetzt diese Faktorvariable in eine numerische Variable umwandeln, kann es zu Verwirrung kommen. Denn R gibt die Zahlen zurück, die intern verwendet werden, um die Faktorstufen zu definieren:

```
as.numeric(Ergebnis)
#> [1] 1 1 1 2 3
```

Wahrscheinlich wollte man eher “10, 10, 10, 20, 30” als Ergebnis zurückgeliefert bekommen. Um das zu erreichen, wandelt man eine Faktorvariable zuerst einen Textvariable um, und dann in eine numerische Variable:

```
Ergebnis %>% as.character %>% as.numeric
#> [1] 10 10 10 20 30
```

8.1.15 Werte umkodieren und partionieren (“binnen”)

Umkodieren meint, die Werte zu ändern. Man sieht immer mal wieder, dass die Variable “gender” (Geschlecht) mit 1 und 2 kodiert ist. Verwechslungen sind da vorprogrammiert (“Ich bin mir 100% sicher, dass ich wahrscheinlich”1“ für Männer kodiert habe“). Besser wäre es, die Ausprägungen `male` und `female` (“Mann”, “Frau”) o.ä. zu verwenden (vgl. Abb. 8.5). Eine Ausnahme sind Variablen wie `is_female`; hier sind Werte wie 1 (“ja”) und 0 (“nein”) plausibel.

Partitionieren oder “*Binnen*” meint, eine kontinuierliche Variablen in einige Bereiche (mindestens 2) zu zerschneiden. Damit macht man aus einer kontinuierlichen Variablen eine diskrete. Ein Bild erläutert das am einfachsten (vgl. Abb. 8.6).

Möchten man Häufigkeiten bei einer *metrischen* Variablen auszählen (z.B. was ist die häufigste Körpergröße) bietet es sich an, die Werte erstmal zu partionieren (z.B. klein vs. mittel vs. groß). Allgemeiner gesprochen machen Häufigkeitsauswertungen nur dann Sinn, wenn die Gruppen nicht zu dünn besiedelt sind.

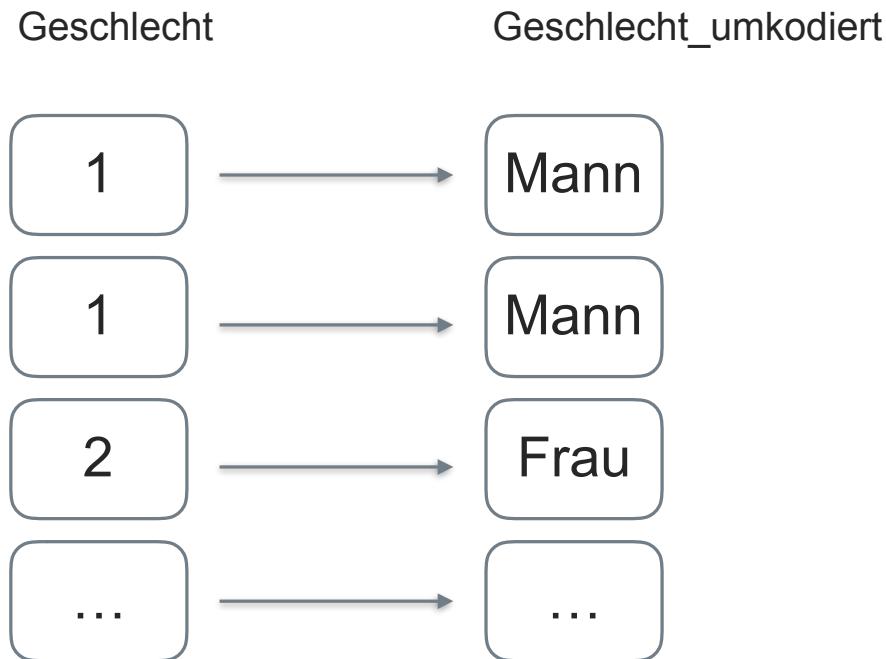


Abbildung 8.5: Sinnbild für Umkodieren

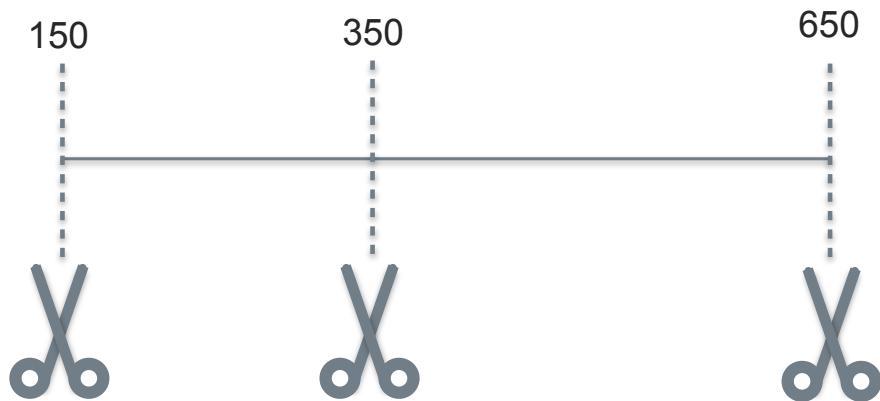


Abbildung 8.6: Sinnbild zum 'Binnen'

8.1.15.1 Umkodieren und partionieren mit `car::recode`

Manchmal möchte man z.B. negativ gepolte Items umdrehen oder bei kategorialen Variablen kryptische Bezeichnungen in sprechendere umwandeln. Hier gibt es eine Reihe praktischer Befehle, z.B. `recode` aus dem Paket `car`. Schauen wir uns ein paar Beispiele zum Umkodieren an.

```
stats_test %>%
  mutate(study_binned = car::recode(.study_time,
"5 = 'sehr viel'; 2:4 = 'mittel'; 1 = 'wenig'",  
as.factor.result = TRUE)) -> stats_test
```

```
stats_test %>%
  mutate(study_binned = car::recode(.study_time,
"5 = 'sehr viel'; 2:4 = 'mittel'; 1 = 'wenig'",
as.factor.result = FALSE)) -> stats_test

stats_test %>%
  mutate(score_binned = car::recode(.score,
"40:38 = 'sehr gut'; 37:35 = 'gut'; else = 'Weiterlernen'",
as.factor.result = TRUE)) -> stats_test

stats_test %>% select(score_binned, study_binned) %>% head
```

Der Befehle `recode` ist praktisch; mit `:` kann man “von bis” ansprechen (das ginge mit `c()` übrigens auch); `else` für “ansonsten” ist möglich und mit `as.factor.result` kann man entweder einen Faktor oder eine Text-Variable zurückgeliefert bekommen. Der ganze “Wechselterm” steht in Anführungsstrichen (“”). Einzelne Teile des Wechselterms sind mit einem Strichpunkt (`;`) voneinander getrennt.

Das klassische Umkodieren von Items aus Fragebögen kann man so anstellen; sagen wir `interest` soll umkodiert werden:

```
stats_test %>%
  mutate(no_interest = car::recode(.interest,
"1 = 6; 2 = 5; 3 = 4; 4 = 3; 5 = 2; 6 = 1; else = NA")) -> stats_test

glimpse(stats_test$no_interest)
#> num [1:306] 2 4 1 5 1 NA NA 4 2 2 ...
```

Bei dem Wechselterm muss man aufpassen, nichts zu verwechseln; die Zahlen sehen alle ähnlich aus...

Testen kann man den Erfolg des Umpolens mit

```
dplyr::count(stats_test, interest)
dplyr::count(stats_test, no_interest)
```

Scheint zu passen. Noch praktischer ist, dass man so auch numerische Variablen in Bereiche aufteilen kann (“binnen”). Hier das Binnen von Prof. Schnaggeldi:

```
stats_test %>%
  mutate(bestanden =
    car::recode(.score,
```

```
"1:38 = 'durchgefallen';
else = 'bestanden'") -> stats_test
```

Natürlich gibt es auch eine Pfeifen kompatible Version, um Variablen umzukodieren bzw. zu binnnen: `dplyr::recode`⁶. Die Syntax ist allerdings etwas weniger komfortabel (da strenger), so dass wir an dieser Stelle bei `car::recode` bleiben.

8.1.15.2 Einfaches Umkodieren mit einer Logik-Prüfung

Nehmen wir an, wir möchten die Anzahl der Punkte in einer Statistikklausur (`score`) umkodieren in eine Variable “bestanden” mit den zwei Ausprägungen “ja” und “nein”; der griesgrämige Professor beschließt, dass die Klausur ab 25 Punkten (von 40) bestanden sei. Die Umkodierung ist also von der Art “viele Ausprägungen in zwei Ausprägungen umkodieren”. Das kann man z.B. so erledigen:

```
stats_test %>%
  mutate(bestanden = score > 24) -> stats_test

head(stats_test$bestanden)
#> [1] TRUE TRUE TRUE FALSE TRUE TRUE
```



Wann braucht man einen Punkt . in einer dplyr-Pfeifenkette? Man braucht ihn nur dann, wenn man innerhalb der Pfeifenkette Nicht-dplyr-Befehle verwendet wie `mean` oder `recode`. Diese Befehle wissen nicht, dass sie gerade in einer Pfeifenkette stehen, daher müssen wir ihnen den Namen der Tabelle explizit erzählen. Genau genommen steht der Punkt für die Tabelle, so wie sie aus dem letzten Pfeifenschritt herausgekommen ist (z.B. schon mit einigen Zeilen weggefiltert).

Genauso könnte man sich die “Grenzfälle” - die Bemitleidenswerten mit 24 Punkten - anschauen (knapp daneben ist auch vorbei, so der griesgrämige Professor weiter):

```
stats_test$Grenzfall <- stats_test$score == 24

dplyr::count(stats_test, Grenzfall)
#> # A tibble: 2 x 2
#>   Grenzfall     n
#>   <lgcl> <int>
#> 1 FALSE     294
#> 2 TRUE      12
```

⁶<https://blog.rstudio.org/2016/06/27/dplyr-0-5-0/>

Natürlich könnte man auch hier “Durchpfeifen”:

```
stats_test <-
stats_test %>%
  mutate(Grenzfall = score == 24)

dplyr::count(stats_test, Grenzfall)
#> # A tibble: 2 x 2
#>   Grenzfall     n
#>   <lgl> <int>
#> 1 FALSE    294
#> 2 TRUE      12
```

8.1.15.3 Binnen mit cut

Numerische Werte in Klassen zu gruppieren (“to bin”, denglisch: “binnen”) kann mit dem Befehl `cut` (and friends) besorgt werden.

Es lassen sich drei typische Anwendungsformen unterscheiden:

Eine numerische Variable . . .

1. in k gleich große Klassen gruppieren (gleichgroße Intervalle)
2. so in Klassen gruppieren, dass in jeder Klasse n Beobachtungen sind (gleiche Gruppengrößen)
3. in beliebige Klassen gruppieren

8.1.15.3.1 Gleichgroße Intervalle

Nehmen wir an, wir möchten die numerische Variable “Körpergröße” in drei Gruppen einteilen: “klein”, “mittel” und “groß”. Der Range von Körpergröße soll gleichmäßig auf die drei Gruppen aufgeteilt werden, d.h. der Range (Intervall) der drei Gruppen soll gleich groß sein. Dazu kann man `cut_interval` aus `ggplot2` nehmen⁷.

```
temp <- cut_interval(x = stats_test$score, n = 3)

levels(temp)
#> [1] "[17,24.7]" "(24.7,32.3]" "(32.3,40]"
```

`cut_interval` liefert eine Variable vom Typ `factor` zurück. Hier haben wir das Punktespektrum in drei gleich große Bereiche unterteilt (d.h. mit jeweils gleichem Punkte-Range).

⁷d.h. `ggplot2` muss geladen sein; wenn man `tidyverse` lädt, wird `ggplot2` automatisch auch geladen

8.1.15.3.2 Gleiche Gruppengrößen

```
temp <- cut_number(stats_test$score, n = 2)
str(temp)
#> Factor w/ 2 levels "[17,31]", "(31,40]": 1 1 2 1 2 2 2 1 1 2 ...
median(stats_test$score)
#> [1] 31
```

Mit `cut_number` (aus `ggplot2`) kann man einen Vektor in `n` Gruppen mit (etwa) gleich viel Observationen einteilen. Hier haben wir `score` am Median geteilt.

Teilt man einen Vektor in zwei gleich große Gruppen, so entspricht das einer Aufteilung am Median (Median-Split).

8.1.15.3.3 In beliebige Klassen gruppieren

```
stats_test %>%
  mutate(punkte_gruppe = cut(stats_test$score,
    breaks = c(-Inf, 25, 29, 33, 37, 40),
    labels = c("5", "4", "3", "2", "1"))) -> stats_test

dplyr::count(stats_test, punkte_gruppe)
#> # A tibble: 5 x 2
#>   punkte_gruppe     n
#>   <fctr>     <int>
#> 1 5             56
#> 2 4             68
#> 3 3             63
#> 4 2             64
#> 5 1             55
```

`cut` ist im Standard-R (Paket “base”) enthalten. Mit `breaks` gibt man die Intervallgrenzen an. Zu beachten ist, dass man eine Unter- bzw. Obergrenze angeben muss. D.h. der kleinste Wert in der Stichprobe wird nicht automatisch als unterste Intervallgrenze herangezogen. Anschaulich gesprochen ist `cut` ein Messer, das ein Seil (die kontinuierliche Variable) mit einem oder mehreren Schnitten zerschneidet (vgl. Abb. 8.6). Wenn wir 6 Schnitte (`breaks`) tun, haben wir 5 Teile, wie Abb. 8.6 zeigt. Darum müssen wir auch nur 5 (6-1) `labels` für die Teile vergeben.

8.2 Deskriptive Statistiken berechnen

8.2.1 Mittelwerte pro Zeile berechnen

8.2.1.1 `rowMeans`

Um Umfragedaten auszuwerten, will man häufig einen Mittelwert *pro Zeile* berechnen. Normalerweise fasst man eine *Spalte* zu einer Zahl zusammen; aber jetzt, fassen wir eine *Zeile* zu einer Zahl zusammen. Der häufigste Fall ist, wie gesagt, einen Mittelwert zu bilden für jede Person. Nehmen wir an, wir haben eine Befragung zur Extraversion durchgeführt und möchten jetzt den mittleren Extraversionswert pro Person (d.h. *pro Zeile*) berechnen.

```
extra_items <- extra %>%
  dplyr::select(i01:i10)

extra %>%
  mutate(extra_mean = rowMeans(extra_items)) -> extra
```

Da der Datensatz über 28 Spalten verfügt, wir aber nur 10 Spalten heranziehen möchten, um Zeilen auf eine Zahl zusammenzufassen, bilden wir als Zwischenschritt einen “schmäleren” Datensatz, `extra_items`. Im Anschluss berechnen wir mit `rowMeans` die Mittelwerte pro Zeile (engl. “row”).

8.2.2 Mittelwerte pro Spalte berechnen

Eine Möglichkeit ist der Befehl `summary` aus `dplyr`.

```
stats_test %>%
  na.omit %>%
  summarise(mean(score),
            sd(score),
            median(score),
            IQR(score))
#>   mean(score) sd(score) median(score) IQR(score)
#> 1      31     5.37      31        8
```

Die Logik von `dplyr` lässt auch einfach Subgruppenanalysen zu. Z.B. können wir eine Teilmenge des Datensatzes mit `filter` erstellen und dann mit `group_by` Gruppen vergleichen:

```
stats_test %>%
  filter(study_time > 1) %>%
```

```
group_by(interest) %>%
  summarise(score, na.rm = TRUE))
#> # A tibble: 6 x 2
#>   interest `median(score, na.rm = TRUE)` 
#>   <dbl>                <dbl>
#> 1 1                  28
#> 2 2                  30
#> 3 3                  33
#> 4 4                  31
#> 5 5                  34
#> 6 6                  34
```

Wir können auch Gruppierungskriterien unterwegs erstellen:

```
stats_test %>%
  na.omit %>%
  filter(study_time > 1) %>%
  group_by(intessiert = interest > 3) %>%
  summarise(md_gruppe = median(score))
#> # A tibble: 2 x 2
#>   intessiert md_gruppe
#>   <lgl>      <int>
#> 1 FALSE       30
#> 2 TRUE        32
```

Die beiden Gruppen von `intessiert` sind “ja, interessiert” (`interest > 3` ist `TRUE`) und “nein, nicht interessiert” (`interest > 3` ist `FALSE`). Außerdem haben wir der Spalte, die die Mediane zurückliefert einen ansprechenderen Namen gegeben (`md_gruppe`).

Etwas expliziter wäre es, `mutate` zu verwenden, um die Variable `intessiert` zu erstellen:

```
stats_test %>%
  na.omit %>%
  filter(study_time > 1) %>%
  mutate(intessiert = interest > 3) %>%
  group_by(intessiert) %>%
  summarise(md_gruppe = median(score),
            mw_gruppe = mean(score))
#> # A tibble: 2 x 3
#>   intessiert md_gruppe mw_gruppe
#>   <lgl>      <int>     <dbl>
#> 1 FALSE       30        31.2
#> 2 TRUE        32        31.8
```

Dieses Mal haben wir nicht nur eine Spalte mit den Medianwerten, sondern zusätzlich noch mit Mittelwerten berechnet.



Statistiken, die auf dem Mittelwert (arithmetisches Mittel) beruhen, sind nicht robust gegenüber Ausreißer: Schon wenige Extremwerte können diese Statistiken so verzerren, dass sie erheblich an Aussagekraft verlieren.

Daher: besser robuste Statistiken verwenden. Der Median, der Modus und der IQR bieten sich an.

8.2.3 Korrelationstabellen berechnen

Korrelationen bzw. Korrelationstabellen lassen sich mit dem R-Standardbefehl `cor` berechnen:

```
stats_test %>%
  select(study_time, interest, score) %>%
  cor()
#>           study_time interest score
#> study_time      1       NA     NA
#> interest        NA     1.000  0.196
#> score           NA     0.196  1.000
```

Oh! Lauter NAs! Besser wir löschen Zeilen mit fehlenden Werten bevor wir die Korrelation ausrechnen:

```
stats_test %>%
  select(study_time:score) %>%
  na.omit %>%
  cor()
#>           study_time self_eval interest score
#> study_time      1.000    0.559   0.461  0.441
#> self_eval        0.559    1.000   0.360  0.628
#> interest         0.461    0.360   1.000  0.223
#> score            0.441    0.628   0.223  1.000
```

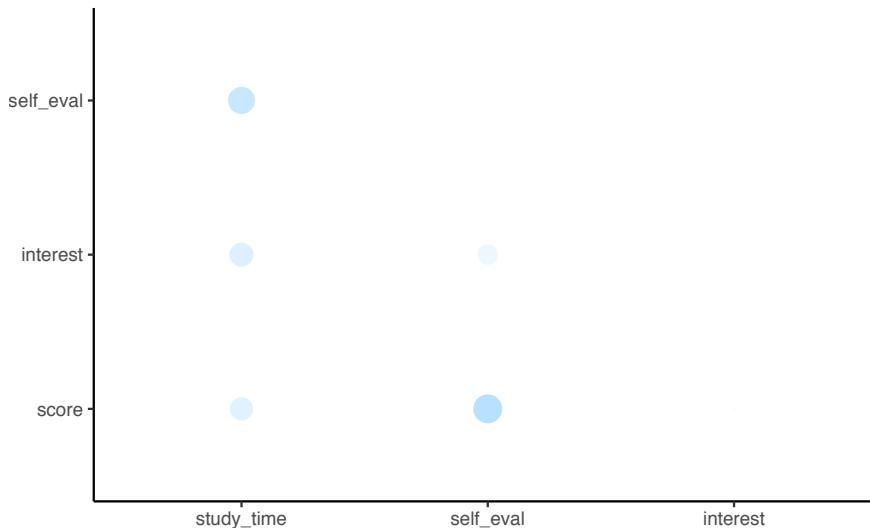
Alternativ zu `cor` kann man auch `corrr::correlate` verwenden:

```
stats_test %>%
  select(study_time:score) %>%
  correlate
#> # A tibble: 4 x 5
```

```
#>      rowname study_time self_eval interest score
#>      <chr>     <dbl>     <dbl>    <dbl> <dbl>
#> 1 study_time        NA     0.559   0.461 0.441
#> 2 self_eval       0.559        NA    0.360 0.628
#> 3 interest        0.461     0.360        NA 0.196
#> 4 score           0.441     0.628     0.196        NA
```

`correlate` hat den Vorteil, dass es bei fehlenden Werten einen Wert ausgibt; die Korrelation wird paarweise mit den verfügbaren (nicht-fehlenden) Werten berechnet. Außerdem wird eine Dataframe (genauer: tibble) zurückgeliefert, was häufig praktischer ist zur Weiterverarbeitung. Wir könnten jetzt die resultierende Korrelationstabelle plotten, vorher “rasieren” wir noch das redundante obere Dreieck ab (da Korrelationstabellen ja symmetrisch sind):

```
stats_test %>%
  select(study_time:score) %>%
  correlate %>%
  shave %>%
  rplot
```



Teil IV

Daten visualisieren

Kapitel 9

Fallstudie ‘movies’



Lernziele:

- Grundlegende Funktionen von `dplyr` anwenden können.
- Das Konzept der Pfeife in einem echten Datensatz anwenden können.
- Auch mit relativ großen Daten sicher hantieren können.

Der Datensatz `movies` enthält Bewertungen von Filmen, zusammen mit einigen zusätzlichen Informationen wie Genre, Erscheinungsjahr und Budgethöhe. Wir nutzen diesen Datensatz um uns einige Übung mit Aufbereiten und Zusammenfassen von Daten zu verschaffen.

Für dieses Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Datenjudo und Visualisierung  
library(corrr) # Korrelation  
library(ggplot2movies) # Daten
```

Zunächst laden wir die Daten und werfen einen Blick in den Datensatz:

```
data(movies, package = "ggplot2movies")  
glimpse(movies)
```

Hier findet man einige Erklärungen zu diesem Datensatz: <http://had.co.nz/data/movies/>.

9.1 Wie viele Filme gibt es pro Genre?

Normalerweise würde man für diese Frage eine Spalte wie “Genre” nehmen und die verschiedenen Werte dieser Spalte auszählen. Das geht sehr bequem mit `dplyr::count`. Hier gibt es allerdings so eine Spalte nicht. Wir müssen uns anders behelfen.

```
movies %>%
  select(Action:Short) %>%
  summarise_all(funs(sum))
#> # A tibble: 1 x 7
#>   Action Animation Comedy Drama Documentary Romance Short
#>   <int>     <int> <int> <int>       <int>    <int> <int>
#> 1     4688      3690  17271 21811       3472     4744  9458
```

Auf Deutsch heißt diese Syntax



Nimm die Tabelle “movies” UND DANN
 nimm alle Spalten von “Action” bis “Short” UND DANN
 fasse alle Spalten (die wir genommen haben) zusammen und zwar... mit der oder den
 Funktionen “Summe” (sum).

Genau wie der Befehl `summarise` fasst auch `summarise_all` Spalten zu einer Zahl zusammen - nur eben nicht *eine*, sondern *alle* Spalten eines Dataframe. Die Funktion(en), die beim Zusammenfassen verwendet werden sollen, werden mit `fun(s)` definiert.

9.2 Welches Genre ist am häufigsten?

Bzw. in welchem Genre wurden am meisten Filme gedreht (in unserem Datensatz)?

```
movies %>%
  select(Action:Short) %>%
  summarise_all(funs(sum)) %>%
  gather() %>%
  arrange(-value)
#> # A tibble: 7 x 2
#>   key     value
#>   <chr> <int>
#> 1 Drama  21811
#> 2 Comedy 17271
#> 3 Short  9458
#> 4 Romance 4744
#> 5 Action  4688
#> 6 Animation 3690
#> 7 Documentary 3472
```

Der Befehl `gather` baut einen Dataframe von “breit” nach “lang” um (vgl. Kapitel 6.3). Ah, Schmunzettlen Dramen sind also am häufigsten (wie der Befehl `arrange` dann zeigt).

Welcome to Hollywood. :tada:

9.3 Zusammenhang zwischen Budget und Beurteilung

Werden teurere Filme (also Filme mit mehr Budget) besser beurteilt im Schnitt? Das würde man erwarten, denn zum Spaß werden die Investoren wohl nicht ihr Geld raus. Schauen wir es uns an.

```
movies %>%
  select(budget, rating, votes) %>%
  correlate
#> # A tibble: 3 x 4
#>   rowname   budget   rating   votes
#>   <chr>     <dbl>    <dbl>    <dbl>
#> 1 budget      NA -0.0142  0.441
#> 2 rating     -0.0142      NA  0.104
#> 3 votes      0.4413  0.1037     NA
```

Wir haben gerade die drei Spalten `budget`, `rating` und `votes` ausgewählt, dann in der nächsten Zeile die fehlenden Werte eliminiert und schließlich die Korrelation zwischen allen Paaren gebildet. Interessanterweise gibt es keine Korrelation zwischen dem Budget und dem Rating! Teurere Filme sind also mitnichten besser bewertet. Allerdings haben Filme mit mehr Budget eine größere Anzahl an Bewertungen, sind also offenbar bekannter. Vielleicht gehen dann auch entsprechend mehr Leute ins Kino - auch wenn diese Filme nicht besser sind. Teurere Filme sind also bekannter, wenn auch nicht besser (beurteilt); so könnte man die Daten lesen.

9.4 Wurden die Filme im Lauf der Jahre teurer und/oder “besser”?

```
movies %>%
  select(year, rating, budget) %>%
  correlate
#> # A tibble: 3 x 4
#>   rowname     year   rating   budget
#>   <chr>       <dbl>    <dbl>    <dbl>
#> 1 year        NA -0.0699  0.2907
#> 2 rating     -0.0699      NA -0.0142
#> 3 budget     0.2907 -0.0142     NA
```

Offenbar wurden die Filme im Lauf der Zeit nicht besser beurteilt: Die Korrelation von `year` und `rating` ist praktisch Null. Wohl wurden sie aber teurer: Die Korrelation von `year` und `budget` ist substanzial.

Kapitel 10

Grundlagen der Datenvisualisierung mit `ggplot2`

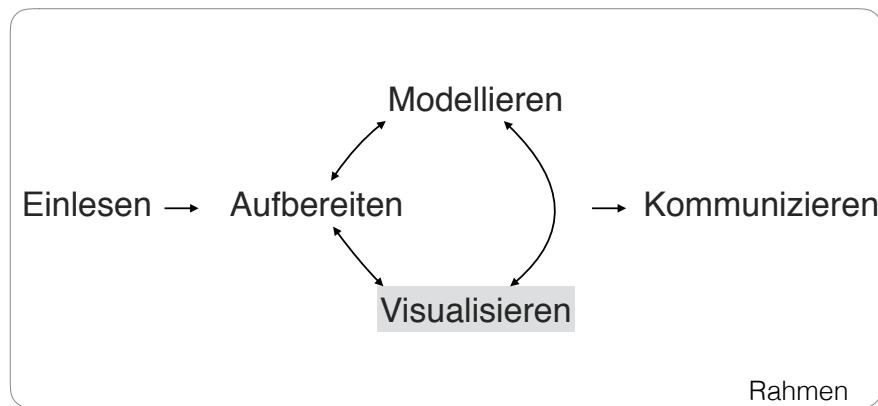


Lernziele:

- An einem Beispiel erläutern können, warum/ wann ein Bild mehr sagt, als 1000 Worte.
- Häufige Arten von Diagrammen erstellen können.
- Diagramme bestimmten Zwecken zuordnen können.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Zum Plotten
library(ggplot2movies) # Daten 'movies'
# library(prada) # optional: Daten 'wo_men', 'stats'test'
# library(AER) # optional: Daten 'Affairs'
# library(okcupiddata) # optional: Daten 'profiles'
```



Dieses Kapitel erläutert das Daten visualisieren anhand des R-Pakets `ggplot2`.

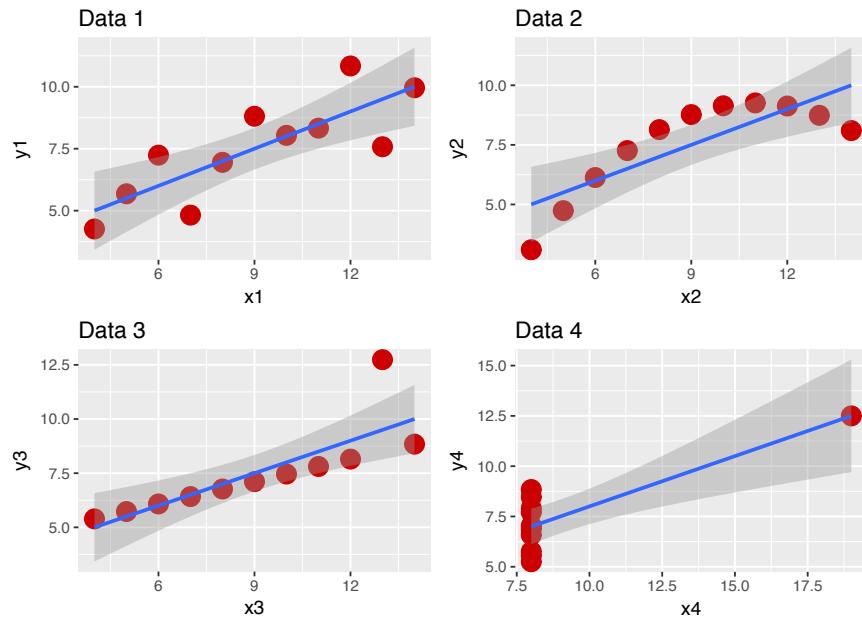


Abbildung 10.1: Das Anscombe-Quartett

10.1 Ein Bild sagt mehr als 1000 Worte

Ein Bild sagt bekanntlich mehr als 1000 Worte. Schauen wir uns zur Verdeutlichung das berühmte Beispiel von Anscombe¹ an. Es geht hier um vier Datensätze mit zwei Variablen (Spalten; X und Y). Offenbar sind die Datensätze praktisch identisch: Alle X haben den gleichen Mittelwert und die gleiche Varianz; dasselbe gilt für die Y. Die Korrelation zwischen X und Y ist in allen vier Datensätzen gleich. Allerdings erzählt eine Visualisierung der vier Datensätze eine ganz andere Geschichte.

Offenbar “passieren” in den vier Datensätzen gänzlich unterschiedliche Dinge. Dies haben die Statistiken nicht aufgedeckt; erst die Visualisierung erhellt uns... Kurz: Die Visualisierung ist ein unverzichtbares Werkzeug, um zu verstehen, was in einem Datensatz (und damit in der zugrunde liegenden “Natur”) passiert.

Eine coole Variante mit der gleichen Botschaft findet sich hier² bzw. mit einer Animation hier³; vgl. Matejka und Fitzmaurice (2017).

Es gibt viele Möglichkeiten, Daten zu visualisieren (in R). Wir werden uns hier auf einen Weg bzw. ein Paket konzentrieren, der komfortabel, aber mächtig ist und gut zum Prinzip des Durchpfeifens passt: `ggplot2`⁴.

¹<https://de.wikipedia.org/wiki/Anscombe-Quartett>

²<https://www.autodeskresearch.com/publications/samestats>

³<https://d2f99xq7vri1nk.cloudfront.net/DinoSequentialSmaller.gif>

⁴“gg” steht für “grammar of graphics” nach einem Buch von Wilkinson(2006); “plot” steht für “to plot”, also ein Diagramm erstellen (“plotten”); vgl. <https://en.wikipedia.org/wiki/Ggplot2>

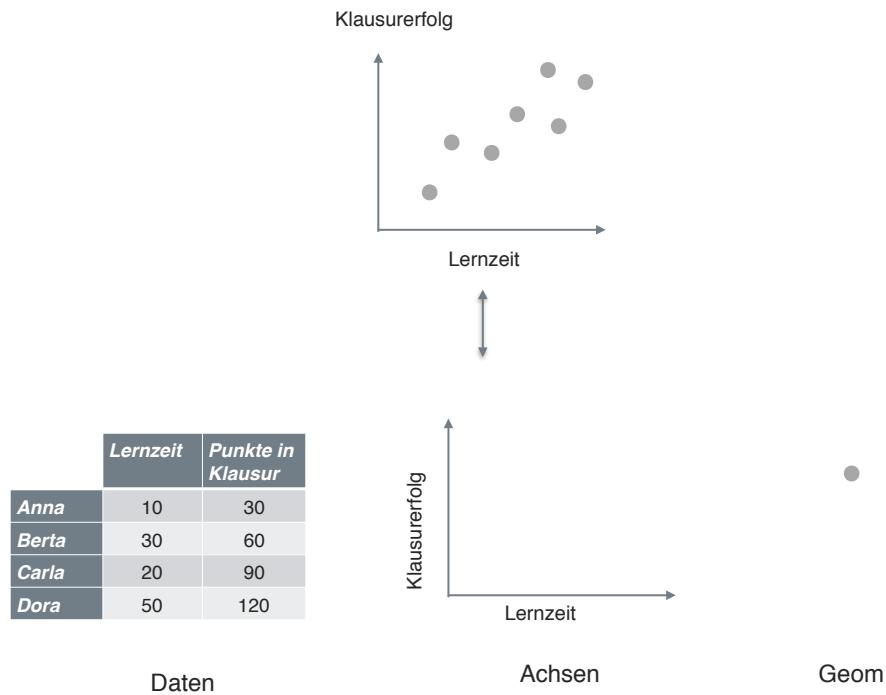


Abbildung 10.2: Anatomie eines Diagramms

10.2 Die Anatomie eines Diagramms

ggplot2 unterscheidet folgende Bestandteile (“Anatomie”) eines Diagramms (vgl. Abb. 10.2):

- Daten
- Abbildende Aspekte (Achsen, Farben, …)
- Geome (statistische Bilder wie Punkte, Linien, Boxplots, …)

Bei *Daten* muss ein Dataframe angegeben werden. Zu den *abbildenden Aspekten* (in ggplot2 als “aesthetics” bzw. `aes` bezeichnet) zählen vor allem die Achsen, aber auch Farben u.a. Was ist mit *abbildend* gemeint? Weist man einer Achse einen Variable zu, so wird jede Ausprägung der Variablen einer Ausprägung der Achse zugeordnet (welcher Wert genau entscheidet ggplot2 für uns, wenn wir es nicht explizieren). Mit *Geom* ist das eigentlich Art von “Bild” gemeint, wie Punkt, Linie oder Boxplot (vgl. Abschnitt ??).

Erstellt ggplot2 ein Diagramm, so ordnet es Spalten den Bestandteilen des zu erzeugenden Diagramms zu (auch “mapping” genannt).

10.3 Einstieg in ggplot2 - qplot

Los geht's! Laden wir zuerst den Datensatz `movies`.

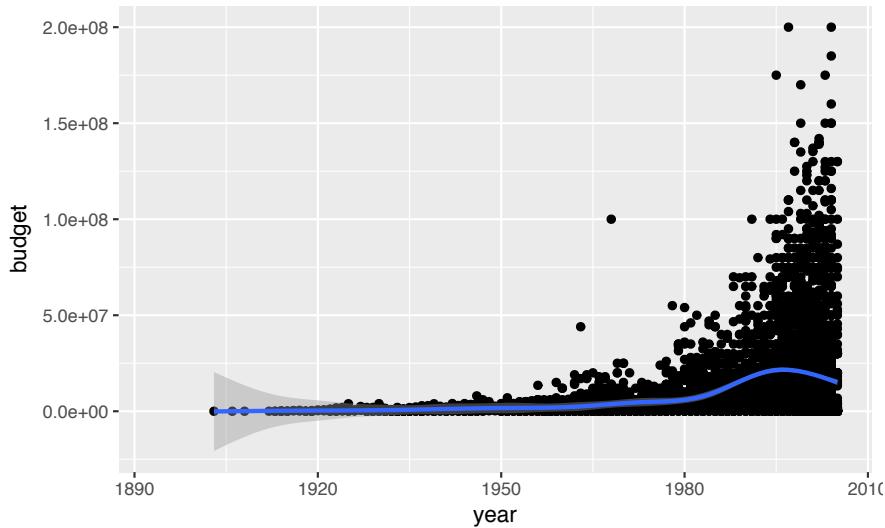


Abbildung 10.3: Mittleres Budget pro Jahr

```
data(movies, package = "ggplot2movies")
```

Betrachten Sie zum Einstieg das Diagramm 10.3:

1. Welche Variable steht auf der X-Achse?
2. Welche Variable steht auf der Y-Achse?
3. Was wird gemalt? Linien, Boxplots, Punkte?
4. Wie heißt der Datensatz, aus dem die Daten gezogen werden?

Der Befehl, der dieses Diagramm erzeugte, heißt `qplot`. Es ist ziemlich genau die Antwort auf die Übungsfragen von gerade eben:

```
qplot(x = year,
      y = budget,
      geom = c("point", "smooth"),
      data = movies)
```

Schauen wir uns den Befehl `qplot` etwas näher an. Wie ist er aufgebaut?



`qplot`: Erstelle schnell (q wie quick in `qplot`) mal einen Plot (engl. “plot”: Diagramm).

`x`: Der X-Achse soll die Variable “year” zugeordnet werden.

`y`: Der Y-Achse soll die Variable “budget” zugeordnet werden.

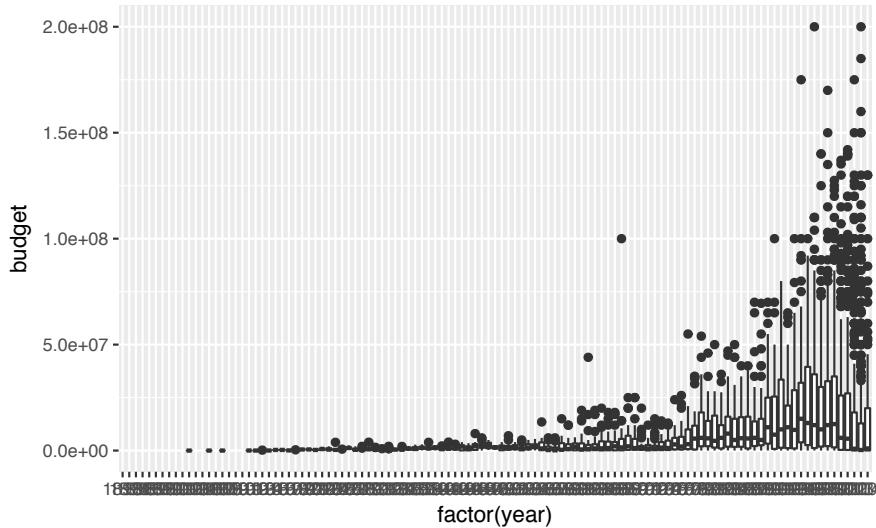
`geom`: (“geometrisches Objekt”) Gemalt werden sollen Punkte und zwar pro Beobachtung (hier: Film) ein Punkt; nicht etwa Linien oder Boxplots.

Außerdem hätten wir gern noch eine “Glättungslinie” (`smooth`), das den “Trend”, also den Y-Mittelwert für jeden Wert der X-Achse angibt.

`data`: Als Datensatz bitte `movies` verwenden.

Offenbar geht die Schwere in den Budgets auseinander; außerdem scheint das Budget größer zu werden. Genau kann man es aber schlecht erkennen in diesem Diagramm. Besser ist es vielleicht die Daten pro Jahr zusammenzufassen in einem Geom und dann diese Geome zu vergleichen:

```
qplot(x = factor(year),
      y = budget,
      geom = "boxplot",
      data = movies)
```



Übrigens: `factor(year)` wird benötigt, um aus `year` eine nominalskalierte Variable zu machen. Nur bei nominalskalierten Variablen auf der X-Achse zeichnet `qplot` mehrere Boxplots nebeneinander. `qplot` bzw. `ggplot2` denkt sich: “Hey, nur wenn es mehrere Gruppen gibt, macht es Sinn, die Gruppen anhand von Boxplots zu vergleichen. Also brauchst du eine Gruppierungsvariable - Faktor oder Text - auf der X-Achse!”.

Es sind zu viele Jahre, das macht das Diagramm unübersichtlich. Besser wäre es, Jahrzehnte dazustellen. Ein Jahrzehnt ist so etwas wie eine Jahreszahl, von der die letzte Ziffer abgeschnitten (d.h. durch 10 teilen und runden) und dann durch eine Null ersetzt wurde (d.h. mit 10 multiplizieren):

```
movies %>%
  mutate(Jahrzehnt = year / 10) %>%
  mutate(Jahrzehnt = trunc(Jahrzehnt)) %>% # trunkieren, abrunden
  mutate(Jahrzehnt = Jahrzehnt * 10) -> movies
```

Schauen Sie sich die ersten Werte von `Jahrzehnt` mal an: `movies %>% select(Jahrzehnt) %>% head`.

Ok, auf ein neues Bild (Abb. 10.4):

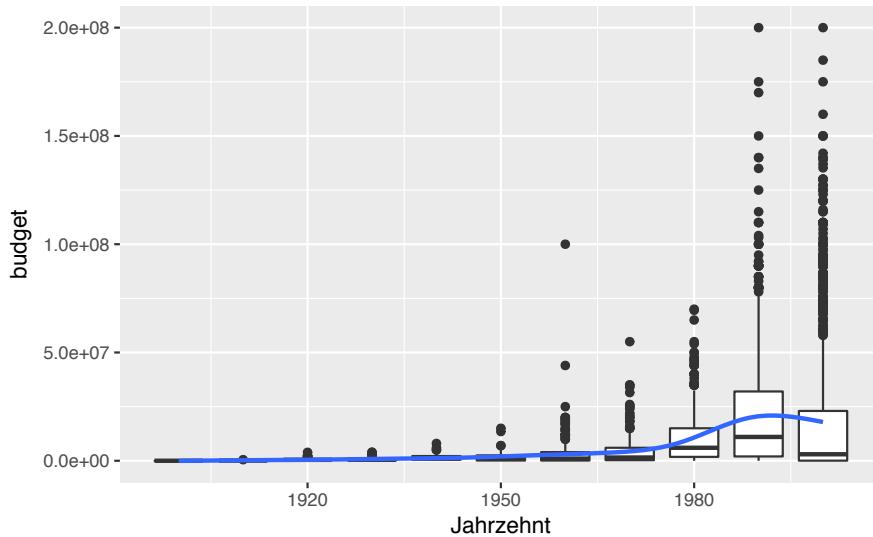


Abbildung 10.4: Film-Budgets über die Jahrzehnte

```
qplot(x = Jahrzehnt,
      y = budget,
      geom = "boxplot",
      group = Jahrzehnt,
      data = movies) + geom_smooth(aes(group = 1), se = FALSE)
```

Weitere Geome kann man auch mit `geom_XXX()` hinzufügen. Warum haben wir das hier so gemacht und nicht die kürzere Methoden wie oben verwendet? Der Grund ist, dass wir mit `geom_XXX()` die Argumente des Geoms, seine Feinheiten also, festlegen können. Mit dem Weg oben können wir nur die voreingestellten Werte verwenden. Die Voreingestellten Werte würden hier dazu führen, dass für jedes Jahrzehnt eine Glättungslinie erzeugt würde. Die hätte aber Länge null; wäre also nicht sichtbar. Warum würde hier in der Voreinstellung für jede Auspräfung von Jahrzehnt eine Gruppe angenommen werden, für die eine eigene Glättungslinie gezeichnet wird? Der Grund ist, dass der Boxplot die Werte bereits gruppiert hat. Der Boxplot hat alle Filme (Werte) eines Jahrzehnts in eine Gruppe eingeteilt. Wir müssen die Gruppierung wieder ändern und R klar machen, dass wir keine Gruppen bzw. nur noch eine Gruppe, die alle Werte enthält, haben möchten, wenn das Geom `smooth` gezeichnet wird. Mit `se = FALSE` schalten wir den grauen "Heiligenschein" der Glättungslinie ab (den Bereich von 1 SE um den Glättungswert herum).

Aha, gut. Interessanterweise sanken die Budgets gegen Ende unserer Datenreihe; das ist aber vielleicht nur ein Zufallsrauschen.

“q” in `qplot` steht für “quick”. Tatsächlich hat `qplot` einen großen Bruder, `ggplot5`, der deutlich mehr Funktionen aufweist - und daher auch die umfangreichere (komplexere) Syntax. Fangen wir mit `qplot` an.

⁵ Achtung: Nicht `qqplot`, nicht `ggplot2`, nicht `gplot`...

Diese Syntax des letzten Beispiels ist recht einfach, nämlich:

```
qplot (x = X_Achse,
       y = Y_Achse,
       data = mein_dataframe,
       geom = "ein_geom")
```

Wir definieren mit `x`, welche Variable der X-Achse des Diagramms zugewiesen werden soll, z.B. `month`; analog mit Y-Achse. Mit `data` sagen wir, in welchem Dataframe die Spalten “wohnen” und als “geom” ist die Art des statistischen “geometrischen Objects” gemeint, also Punkte, Linien, Boxplots, Balken...

10.4 Häufige Arten von Diagrammen

Unter den vielen Arten von Diagrammen und vielen Arten, diese zu klassifizieren greifen wir uns ein paar häufige Diagramme heraus und schauen uns diese der Reihe nach an. Die Arten von Diagrammen richten wir im Folgenden an den beiden Achsen eines typischen Diagramms aus.

10.4.1 Eine kontinuierliche Variable

Schauen wir uns die Verteilung von Filmbudgets aus `movies` an (s. Abb. 10.5).

```
qplot(x = budget, data = movies)
```

Weisen wir nur der X-Achse (aber nicht der Y-Achse) eine kontinuierliche Variable zu, so wählt `ggplot2` automatisch als Geom automatisch ein Histogramm; wir müssen daher nicht explizieren, dass wir ein Histogramm als Geom wünschen (aber wir könnten es hinzufügen).



Was heißt das kleine ‘e’, das man bei wissenschaftlichen Zahlen hin und wieder sieht (wie im Diagramm 10.5)?

Zum Beispiel: $5.0\text{e}+07$. Das e sagt, wie viele Stellen im Exponenten (zur Basis 10) stehen: hier 10^7 . Eine große Zahl - eine 1 gefolgt von *sieben* Nullern: 10000000. Die schöne Zahl soll noch mit 5 multipliziert werden: also 50000000. Bei so vielen Nullern kann man schon mal ein Flimmern vor den Augen bekommen... Daher ist die “wissenschaftliche” Notation ganz praktisch, wenn die Zahlen sehr groß (oder sehr klein) werden. Sehr kleine Zahlen werden mit dieser Notation so dargestellt: $5.0\text{e}-07$ heißt $\frac{1}{10^7}$. Eine Zahl sehr nahe bei Null. Das Minuszeichen zeigt hier, dass wir den Kehrwert der großen Zahl nehmen sollen.

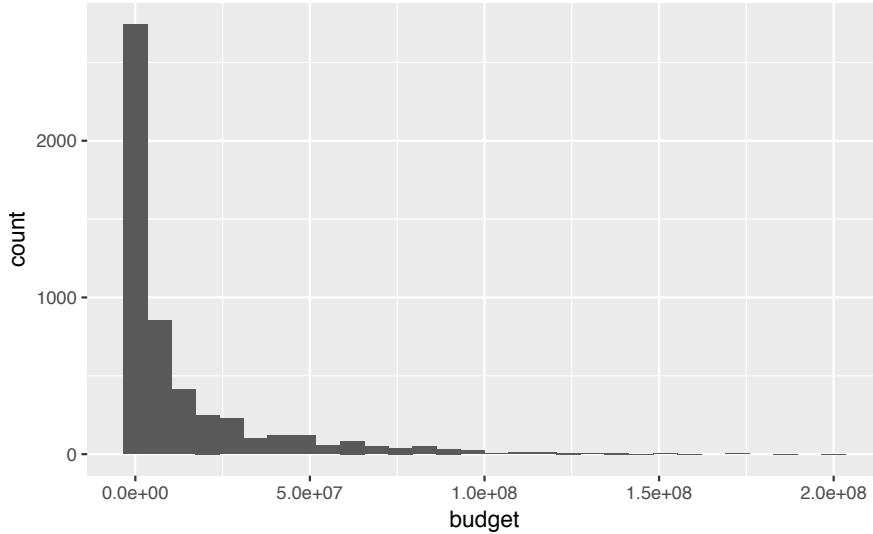
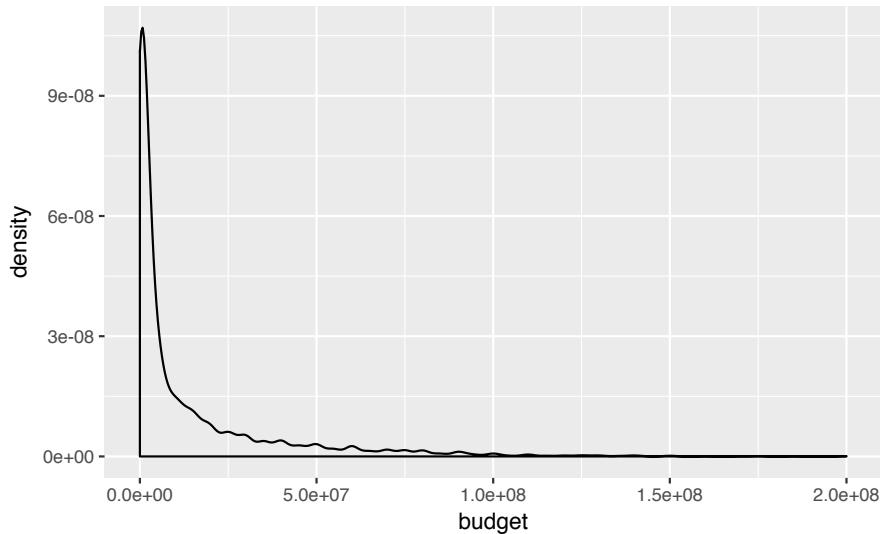


Abbildung 10.5: Verteilung des Budgets von Filmen

Alternativ wäre ein Dichtediagramm hier von Interesse:

```
qplot(x = budget, data = movies, geom = "density")
```

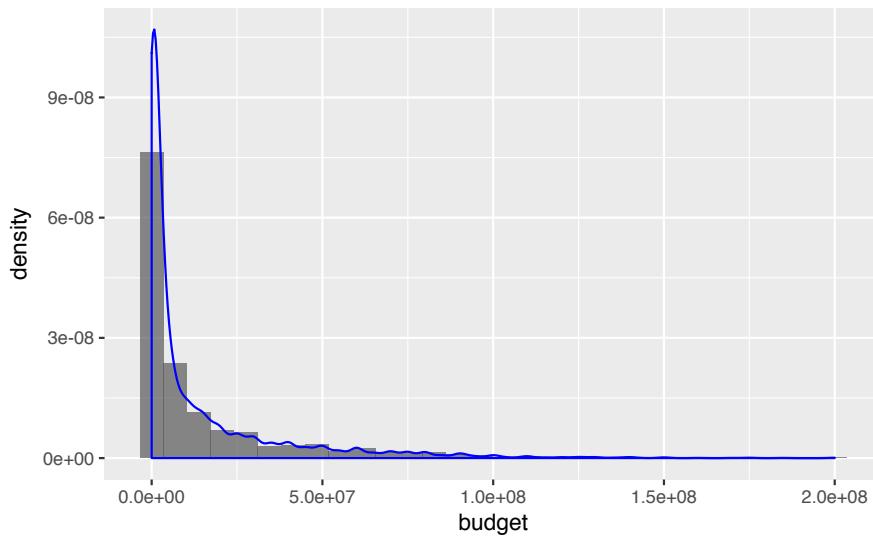


Was man sich merken muss, ist, dass hier nur das Geom mit Anführungsstrichen zu benennen ist, die übrigen Parameter *ohne*.

Vielleicht wäre es noch schön, beide Geome zu kombinieren in einem Diagramm. Das ist etwas komplizierter; wir müssen zum großen Bruder `ggplot` umsteigen, da `qplot` nicht diese Funktionen anbietet.

```
ggplot(data = movies) +
  aes(x = budget) +
```

```
geom_histogram(aes(y = ..density..), alpha = .7) +
  geom_density(color = "blue")
```



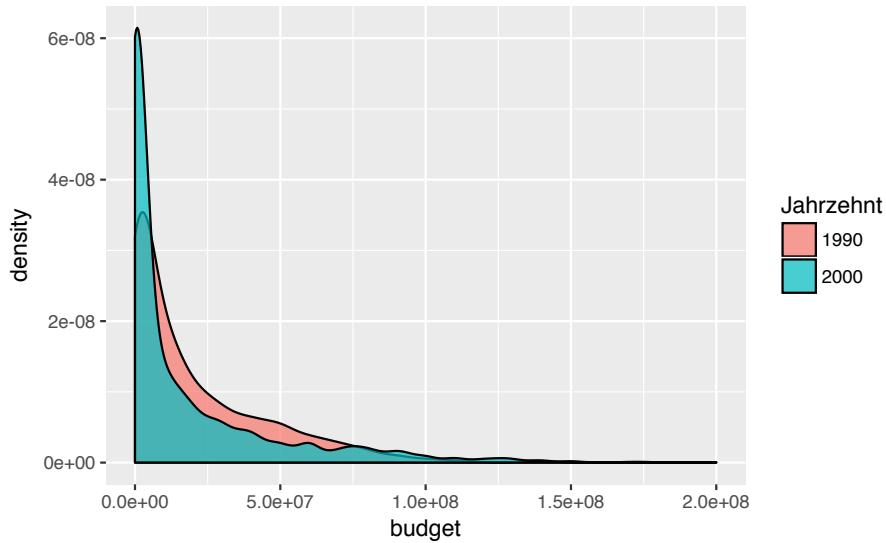
Zuerst haben wir mit dem Parameter `data` den Dataframe benannt. `aes` definiert, welche Variablen welchen Achsen (oder auch z.B. Füllfarben) zugewiesen werden. Hier sagen wir, dass die Schuhgröße auf X-Achse stehen soll. Das `+`-Zeichen trennt die einzelnen Bestandteile des `ggplot`-Aufrufs voneinander. Als nächstes sagen wir, dass wir gerne ein Histogram hätten: `geom_histogram`. Dabei soll aber nicht wie gewöhnlich auf der X-Achse die Häufigkeit stehen, sondern die Dichte. `ggplot` berechnet selbstständig die Dichte und nennt diese Variable `..density..`; die vielen Punkte sollen wohl klar machen, dass es sich nicht um eine “normale” Variable aus dem eigenen Datenframe handelt, sondern um eine “interne” Variable von `ggplot` - die wir aber nichtsdestotrotz verwenden können. `alpha` bestimmt die “Durchsichtigkeit” eines Geoms; spielen Sie mal etwas damit herum. Schließlich malen wir noch ein blaues Dichtediagramm *über* das Histogramm.

Wünsche sind ein Fass ohne Boden... Wäre es nicht interessant, einzelne Zeiträume (Jahrzehnte) zu vergleichen? Schauen wir uns die letzten Jahrzehnte im Vergleich an.

```
movies2 <- filter(movies, Jahrzehnt > 1980)

movies2 %>%
  mutate(Jahrzehnt = factor(.\$Jahrzehnt)) -> movies2

qplot(x = budget,
       data = movies2,
       geom = "density",
       fill = Jahrzehnt,
       alpha = I(.7))
```

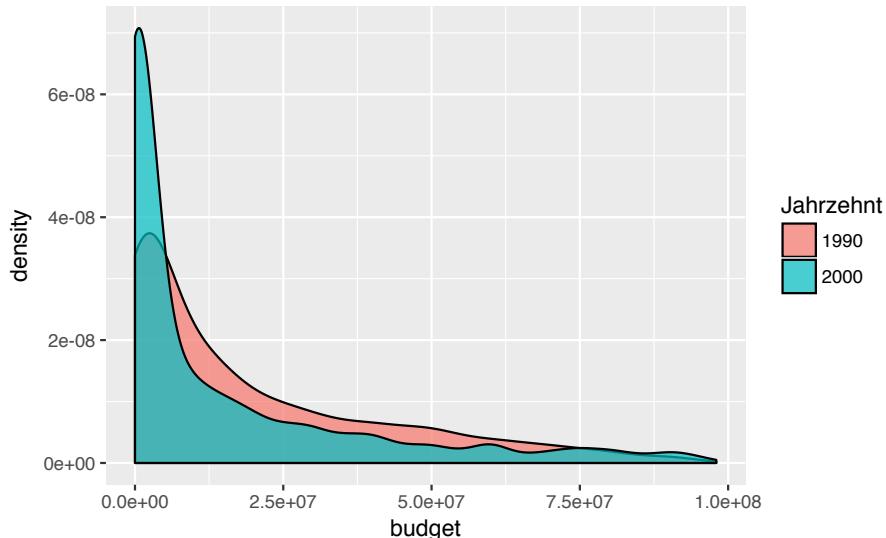


`qplot` erwartet immer *Variablen* als Parameter; wollen wir mal keine Variable, sondern eine fixen Wert, wie 0.7, übergeben, so können wir das mit dem Befehl `I` (wie “identity”) tun.

Hier sollten vielleicht noch die Extremwerte entfernt werden, um den Blick auf das Gros der Werte nicht zu verstauen:

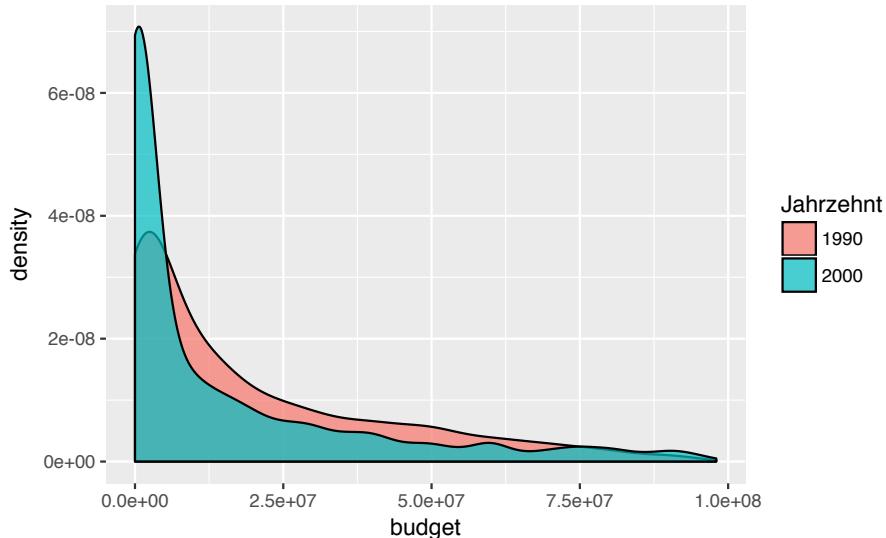
```
movies2 %>%
  filter(budget < 1e08) -> movies2

qplot(x = budget,
      data = movies2,
      geom = "density",
      fill = Jahrzehnt,
      alpha = I(.7))
```



Besser. Man kann das Durchpfeifen auch bis zu `qplot` weiterführen:

```
movies %>%
  filter(budget < 1e+08, Jahrzehnt >= 1990) %>%
  mutate(Jahrzehnt = factor(Jahrzehnt)) %>%
  qplot(x = budget, data = ., geom = "density",
        fill = Jahrzehnt, alpha = I(.7))
```



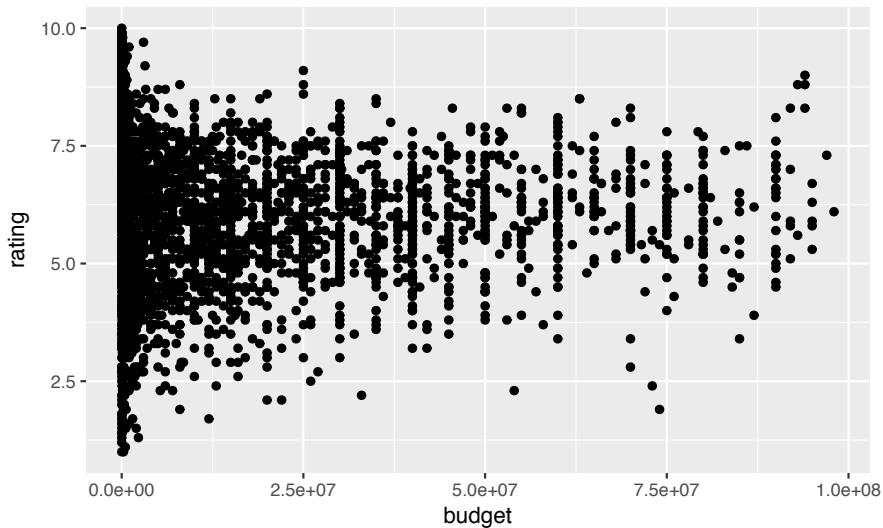
Die Pfeife versucht im Standard, das Endprodukt des letzten Arbeitsschritts an den *ersten* Parameter des nächsten Befehls weiterzugeben. Ein kurzer Blick in die Hilfe von `qplot` zeigt, dass der erste Parameter nicht `data` ist, sondern `x`. Daher müssen wir explizit sagen, an welchen Parameter wir das Endprodukt des letzten Arbeitsschritts geben wollen. Netterweise müssen wir dafür nicht viel tippen: Mit einem schlichten Punkt `.` können wir sagen “nimm den Dataframe, so wie er vom letzten Arbeitsschritt ausgegeben wurde”.

Mit `fill = Jahrzehnt` sagen wir `qplot`, dass er für jedes Jahrzehnt jeweils ein Dichtediagramm erzeugen soll; jedem Dichtediagramm wird dabei eine Farbe zugewiesen (die uns `ggplot2` im Standard voraussucht). Mit anderen Worten: Die Werte von `Jahrzehnt` werden der *Füllfarbe* der Histogramme zugeordnet. Anstelle der Füllfarbe hätten wir auch die Linienfarbe verwenden können; die Syntax wäre dann: `color = sex`. Man beachte, dass die Variable für `fill` oder `color` eine nominale Variable (`factor` oder `character`) sein muss, damit `ggplot2` tut, was will wollen.

10.4.2 Zwei kontinuierliche Variablen

Ein Streudiagramm ist die klassische Art, zwei metrische Variablen darzustellen. Das ist mit `qplot` einfach:

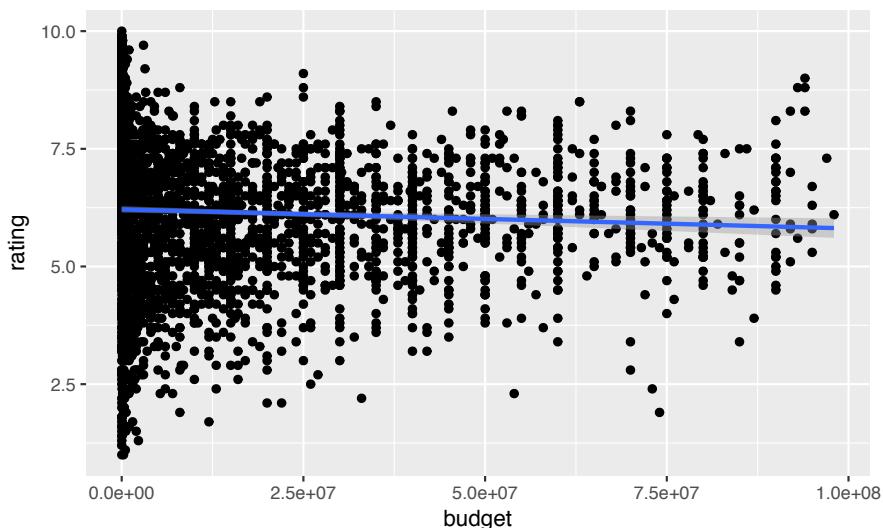
```
p <- qplot(x = budget, y = rating, data = movies2)
p
```



Wir weisen wieder der X-Achse und der Y-Achse eine Variable zu; handelt es sich in beiden Fällen um Zahlen, so wählt ggplot2 automatisch ein Streudiagramm - d.h. Punkte als Geom (geom = "point").

Es ist nicht wirklich ein Trend erkennbar: Teuere Filme sind nicht unbedingt beliebter bzw. besser bewertet. Zeichnen wir eine Trendgerade ein.

```
p + geom_smooth(method = "lm")
```



Synonym könnten wir auch schreiben:

```
wo_men %>%
  filter(height > 150, height < 210, shoe_size < 55) %>%
  ggplot() +
  aes(x = height, y = shoe_size) +
  geom_point() +
  geom_smooth(method = "lm")
```

Da `ggplot` als *ersten* Parameter die Daten erwartet, kann die Pfeife hier problemlos durchgereicht werden. Innerhalb eines `ggplot`-Aufrufs werden die einzelne Teile durch ein Pluszeichen + voneinander getrennt. Nachdem wir den Dataframe benannt haben, definieren wir die Zuweisung der Variablen zu den Achsen mit `aes` (“aes” wie “aesthetics”, also das “Sichtbare” eines Diagramms, die Achsen etc., werden definiert). Ein “Smooth-Geom” ist eine Linie, die sich schön an die Punkte anschmiegt, in diesem Falls als Gerade (lineares Modell, `lm`).

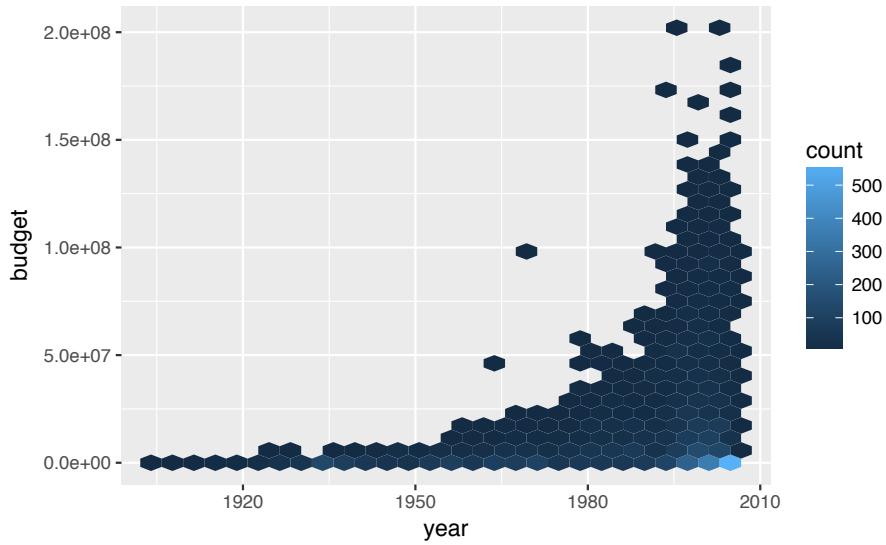
Bei sehr großen Datensätzen, sind Punkte unpraktisch, da sie sich überdecken (“overplotting”). Ein Abhilfe ist es, die Punkte nur “schwach” zu färben. Dazu stellt man die “Füllstärke” der Punkte über `alpha` ein: `geom_point(alpha = 1/100)`. Um einen passablen Alpha-Wert zu finden, bedarf es häufig etwas Probierens. Zu beachten ist, dass es mitunter recht lange dauert, wenn `ggplot` viele (>100.000) Punkte malen soll.

Probieren Sie auch Folgendes aus: Fügen Sie bei `aes` den Parameter `color = sex` hinzu.

Bei noch größeren Datenmengen bietet sich an, den Scatterplot als “Schachbrett” aufzufassen, und das Raster einzufärben, je nach Anzahl der Punkte pro Schachfeld; zwei Geome dafür sind `geom_hex()` und `geom_bin2d()`.

```
nrow(movies) # groß, ein bisschen wenigstens
#> [1] 58788

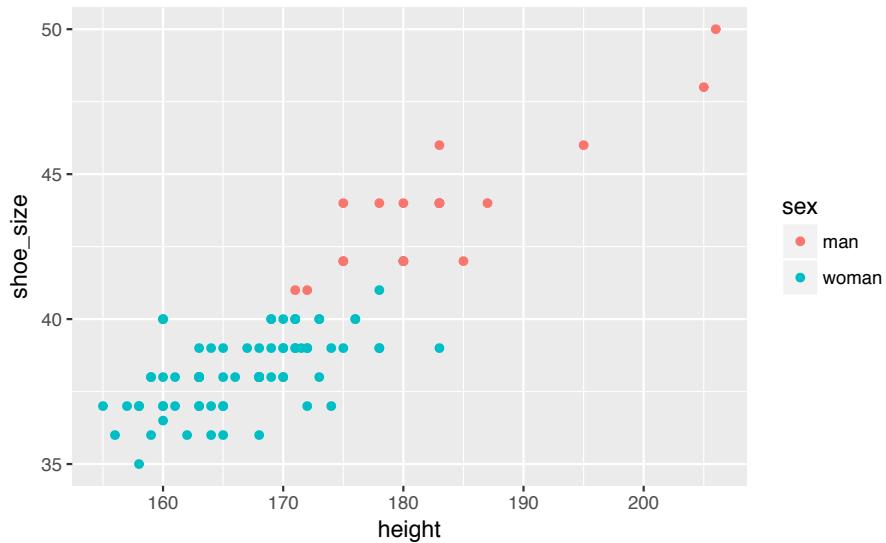
ggplot(movies) +
  aes(x = year, y = budget) +
  geom_hex()
```



Wenn man dies verdaut hat, wächst der Hunger nach einer Aufteilung in Gruppen.

```
wo_men %>%
  dplyr::filter(height > 150, height < 210, shoe_size < 55) -> wo_men2

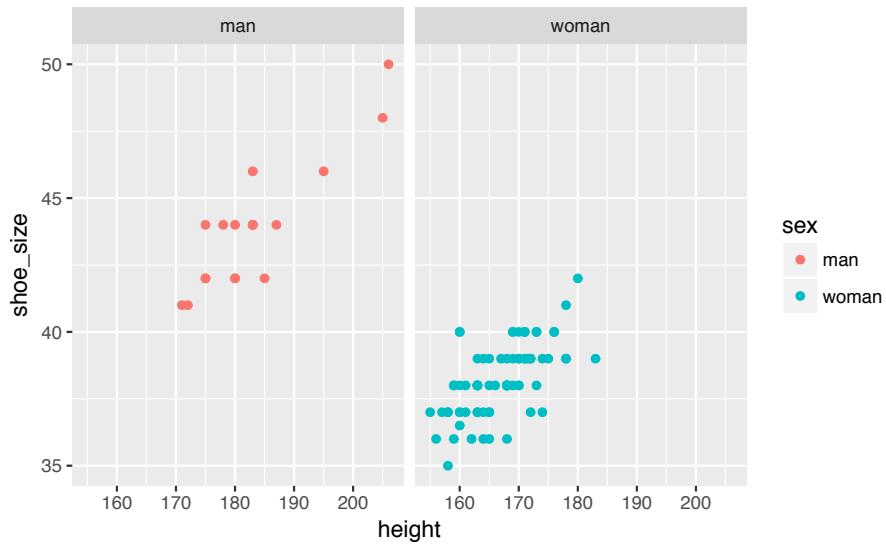
wo_men2 %>%
  qplot(x = height, y = shoe_size, color = sex, data = .)
```



Mit `color = sex` sagen wir, dass die Linienfarbe (der Punkte) entsprechend der Stufen von `sex` eingefärbt werden sollen. Die genaue Farbwahl übernimmt `ggplot2` für uns.

Alternativ kann man auch zwei “Teil-Bildchen” (“facets”) erstellen, eines für Frauen und eines für Männer:

```
wo_men %>%
  dplyr::filter(height > 150, height < 210, shoe_size < 55) %>%
  qplot(x = height, y = shoe_size, facets = "~sex", color = sex, data = .)
```

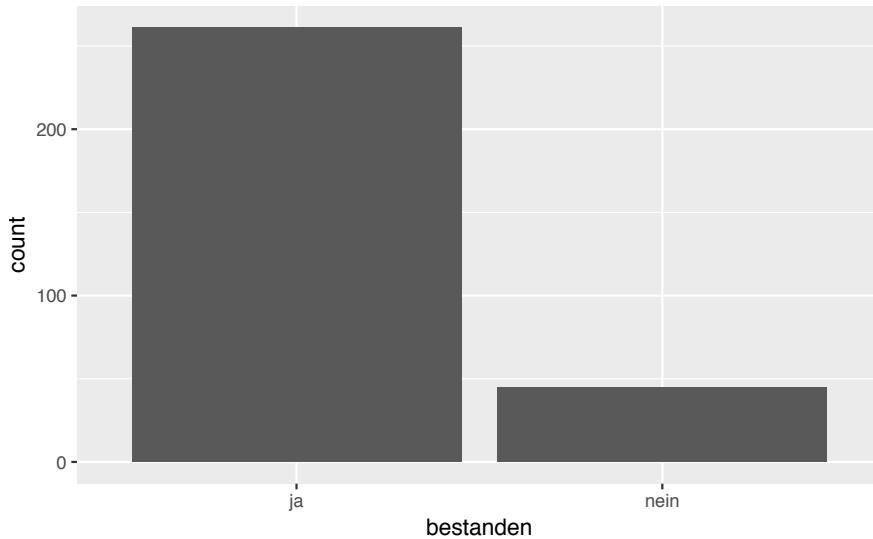


Man beachte die Tilde `~`, die vor die “Gruppierungsvariable” `sex` zu setzen ist.

10.4.3 Eine nominale Variable

Bei nominalen Variablen, geht es in der Regel darum, Häufigkeiten auszuzählen. Ein Klassiker: Wie viele Männer und Frauen finden sich in dem Datensatz? Wie viele Studenten haben (nicht) bestanden?

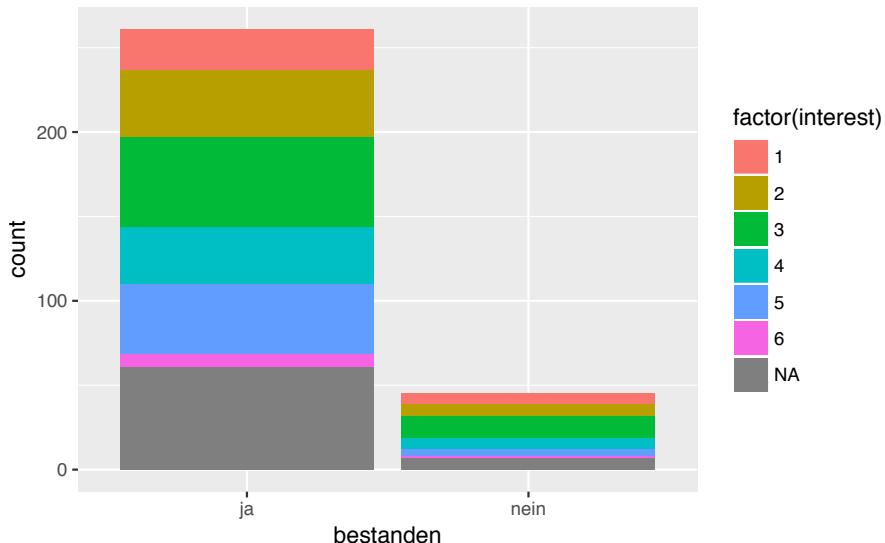
```
stats_test <- read.csv("data/stats_test.csv")
qplot(x = bestanden, data = stats_test)
```



Falls nur die X-Achse definiert ist und dort eine Faktorvariable oder eine Textvariable steht, dann nimmt `qplot` automatisch ein Balkendiagramm als Geom (es steht uns frei, trotzdem `geom = bar` anzugeben).

Wir könnten uns jetzt die Frage stellen, wie viele Nicht-Interessierte und Hoch-Interessierte es in der Gruppe, die bestanden hat (`bestanden == "yes"`) gibt; entsprechend für die Gruppe, die nicht bestanden hat.

```
qplot(x = bestanden, fill = factor(interest), data = stats_test)
```

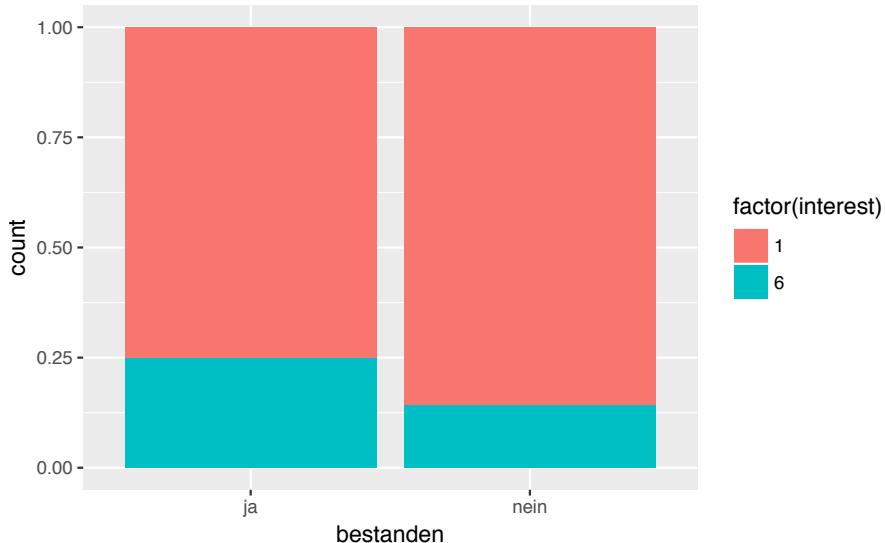


Hier haben wir `qplot` gesagt, dass der die Balken entsprechend der Häufigkeit von `interest` füllen soll. Damit `qplot` (und `ggplot`) sich bequemt, die Füllung umzusetzen, müssen wir aus `interest` eine nominalskalierte Variablen machen - `factor` macht das für uns.

Schön wäre noch, wenn die Balken Anteile (Prozentwerte) angeben würden. Das geht mit

qplot (so) nicht; wir schwenken auf ggplot um. Und, um die Story zuzuspitzen, schauen wir uns nur die Extremwerte von `interest` an.

```
stats_test %>%
  filter(interest == 1 | interest == 6) %>%
  ggplot() +
  aes(x = bestanden, fill = factor(interest)) +
  geom_bar(position = "fill")
```



Der Lehrer freut sich: In der Gruppe, die bestanden hat, ist der Anteil der `freaks` Hoch-Interessierten größer als bei den Durchfallern.

Schauen wir uns die Struktur des Befehls `ggplot` näher an.



stats_test: Hey R, nimm den Datensatz `stats_test` UND DANN... `ggplot()`: Hey R, male ein Diagramm von Typ `ggplot` (mit dem Datensatz aus dem vorherigen Pfeifen-Schritt, d.h. aus der vorherigen Zeile, also `stats_test`)!

filter: wir wollen nur Zeilen (Studenten), für die gilt `interest == 1` oder `interest == 6`. Der horizontale Strich heißt ‘oder’.

+: Das Pluszeichen grenzt die Teile eines ggplot-Befehls voneinander ab.

aes: von “aethetics”, also welche Variablen des Datensatzes den sichtbaren Aspekten (v.a. Achsen, Farben) zugeordnet werden.

x: Der X-Achse (Achtung, `x` wird klein geschrieben hier) wird die Variable `bestanden` zugeordnet.

y: gibt es nicht??? Wenn in einem ggplot-Diagramm *keine* Y-Achse definiert wird, wird ggplot automatisch ein Histogramm bzw. ein Balkendiagramm erstellen. Bei diesen Arten von Diagrammen steht auf der Y-Achse keine eigene Variable, sondern meist die Häufigkeit des entsprechenden X-Werts (oder eine Funktion der Häufigkeit, wie relative Häufigkeit).

fill Das Diagramm (die Balken) sollen so gefüllt werden, dass sich die Häufigkeit der Werte von `interest` darin widerspiegelt. `geom_XYZ`: Als “Geom” soll ein Balken (“bar”) gezeichnet werden. Ein Geom ist in ggplot2 das zu zeichnende Objekt, also ein Boxplot, ein Balken, Punkte, Linien etc. Entsprechend wird gewünschte Geom mit `geom_bar`, `geom_boxplot`, `geom_pointetc.` gewählt. `position = fill`: `position_fill` sagen, dass die Balken alle eine Höhe von 100% (1) haben, d.h. gleich hoch sind. Die Balken zeigen also nur die Anteile der Werte der `fill`-Variablen.

Die einzige Änderung in den Parametern ist `position = "fill"`. Dieser Parameter weist ggplot an, die Positionierung der Balken auf die Darstellung von Anteilen auszulegen. Damit haben alle Balken die gleiche Höhe, nämlich 100% (1). Aber die “Füllung” der Balken schwankt je nach der Häufigkeit der Werte von `groesse_gruppe` pro Balken (d.h. pro Wert von `sex`).

Wir sehen, dass die Anteile von großen bzw. kleinen Menschen bei den beiden Gruppen (Frauen vs. Männer) *unterschiedlich hoch* ist. Dies spricht für einen *Zusammenhang* der beiden Variablen; man sagt, die Variablen sind *abhängig* (im statistischen Sinne).

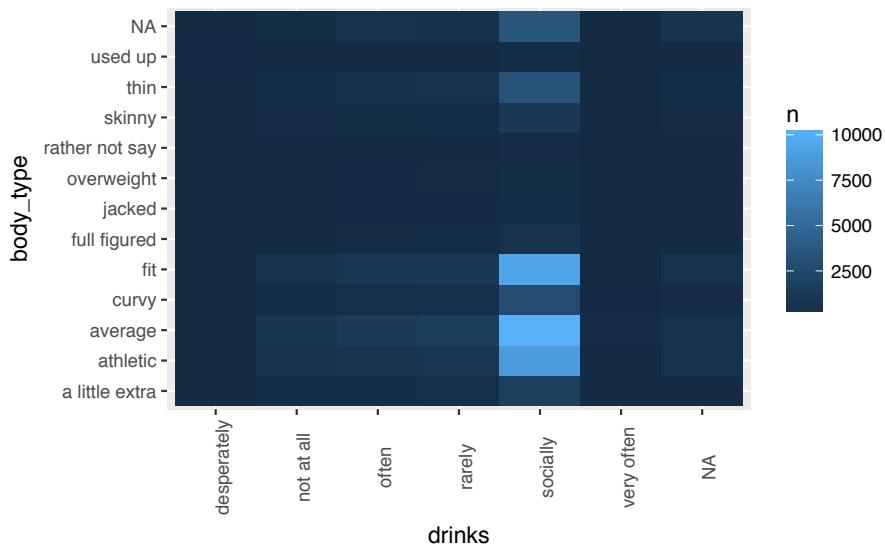
Je unterschiedlicher die “Füllhöhe”, desto stärker sind die Variablen (X-Achse vs. Füllfarbe) voneinander abhängig (bzw. desto stärker der Zusammenhang).

10.4.4 Zwei nominale Variablen

Arbeitet man mit nominalen Variablen, so sind Kontingenztabellen Täglich Brot. Z.B.: Welche Produkte wurden wie häufig an welchem Standort verkauft? Wie viele Narzissen gibt es in welcher Management-Stufe? Wie ist die Verteilung von Alkoholkonsum und Körperperfom bei Menschen einer Single-Börse?. Bleiben wir bei letztem Beispiel.

```
data(profiles, package = "okcupiddata")

profiles %>%
  dplyr::count(drinks, body_type) %>%
  ggplot +
  aes(x = drinks, y = body_type, fill = n) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90))
```



Was haben wir gemacht? Also:



Nehme den Datensatz “profiles” UND DANN
 Zähle die Kombinationen von “drinks” und “body_type” UND DANN
 Erstelle ein ggplot-Plot UND DANN
 Weise der X-Achse “drinks” zu, der Y-Achse “body_type” und der Füllfarbe “n” UND DANN
 Male Fliesen UND DANN
 Passe das Thema so an, dass der Winkel für Text der X-Achse auf 90 Grad steht.

Diese Art von Diagramm nennt man auch ‘Mosaicplot’, weil es an ein Mosaic erinnert (wer hätt's gedacht).

Was sofort ins Auge sticht, ist dass “soziales Trinken”, nennen wir es mal so, am häufigsten ist, unabhängig von der Körperform. Ansonsten scheinen die Zusammenhänge nicht sehr stark zu sein.

10.4.5 Zusammenfassungen zeigen

Manchmal möchten wir *nicht* die Rohwerte einer Variablen darstellen, sondern z.B. die Mittelwerte pro Gruppe. Mittelwerte sind eine bestimmte *Zusammenfassung* einer Spalte; also fassen wir zuerst die Körpergröße zum Mittelwert zusammen - gruppiert nach Geschlecht.

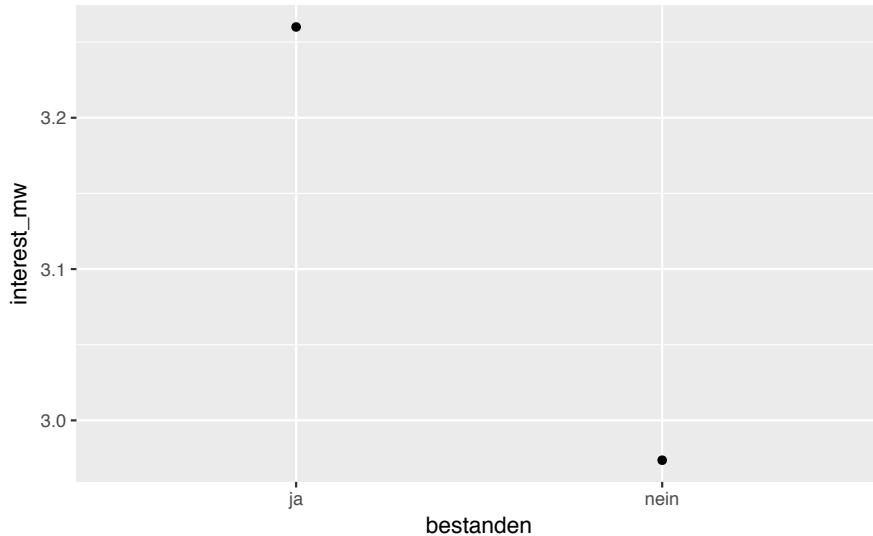
```
stats_test %>%
  group_by(bestanden) %>%
  summarise(interest_mw = mean(interest, na.rm = TRUE)) -> stats_test_summary

stats_test_summary
```

```
#> # A tibble: 2 x 2
#>   bestanden interest_mw
#>   <fctr>      <dbl>
#> 1 ja          3.26
#> 2 nein        2.97
```

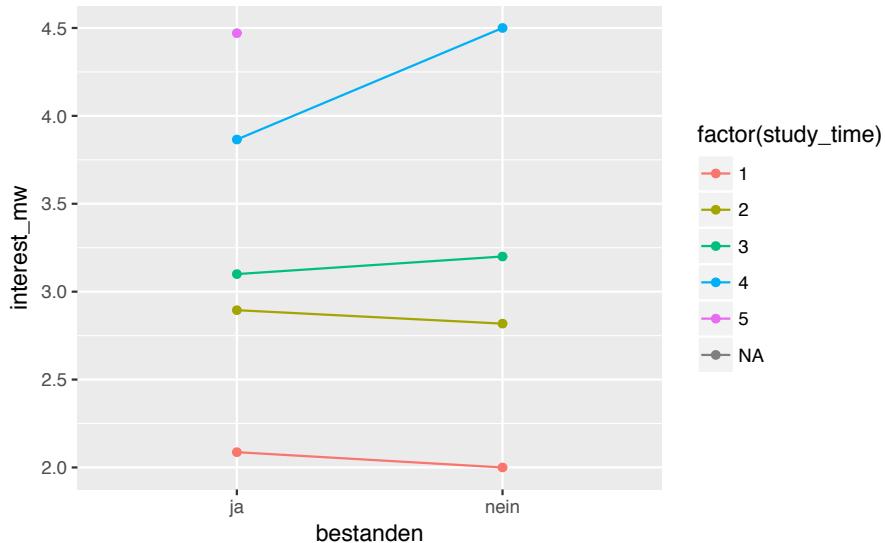
Diese Tabelle schieben wir jetzt in ggplot2; natürlich hätten wir das gleich in einem Rutsch durchpfeifen können.

```
stats_test_summary %>%
  qplot(x = bestanden, y = interest_mw, data = .)
```



Das Diagramm besticht nicht durch die Tiefe und Detaillierung. Bereichern wir das Diagramm um die Frage, wie viel (jeder Student gelernt hat (`study_time`)). Schauen wir uns aber der Einfachheit halber nur die Studenten an, die ganz viel oder ganz wenig gelernt haben.

```
stats_test %>%
  group_by(bestanden, study_time) %>%
  summarise(interest_mw = mean(interest, na.rm = TRUE)) %>%
  qplot(x = bestanden, y = interest_mw, data = ., color = factor(study_time)) +
  geom_line(aes(group = factor(study_time)))
```



In Pseudosyntax:



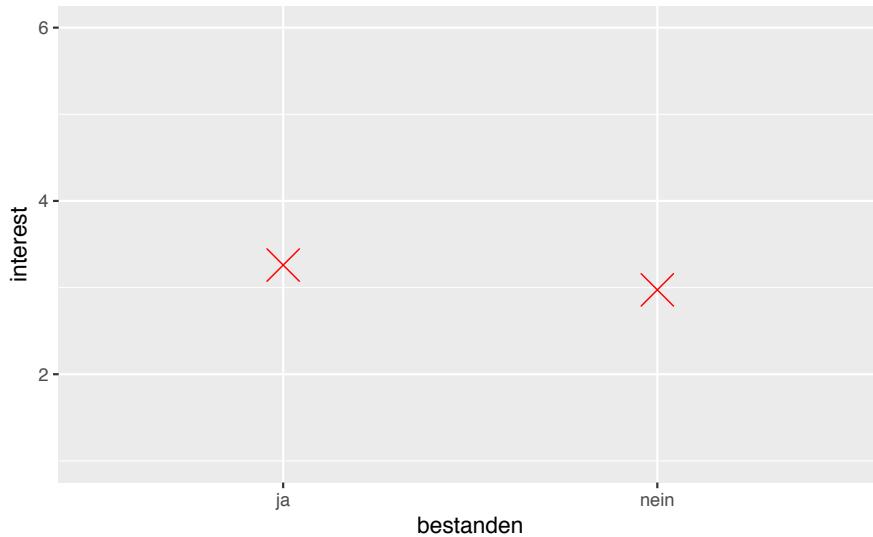
Nehme den Datensatz “stats_test” UND DANN
gruppiere nach den Variablen **bestanden** und **study_time** UND DANN fasse für diese
Gruppen jeweils die Spalte **interest** zum Mittelwert zusammen UND DANN
male einen schnellen Plot mit diesen Daten UND DANN füge ein Liniendiagramm dazu,
wobei jede Stufe von **study_time** eine Gruppe ist. Und Punkte einer Gruppe sollen
verbunden werden.

Warum steht der arme pinkfarbene Punkt bei ‘ja’ und ~4.5 so für sich alleine oder Linie?⁶

Wir haben also die Werte selber aggregiert und `qplot()` hat diese aggregierten Werte geplottet. Wir können die Arbeit auch ganz `qplot()` überlassen. Da `qplot()` immer ein Geom zeichnen möchte, müssen wir es überreden, zumindest das `geom_blank` zu zeichnen, das “blank”, leer ist. Dann können wir mit `+ stat_summary()` eine aggregierte (summary) Statistik hinzufügen.

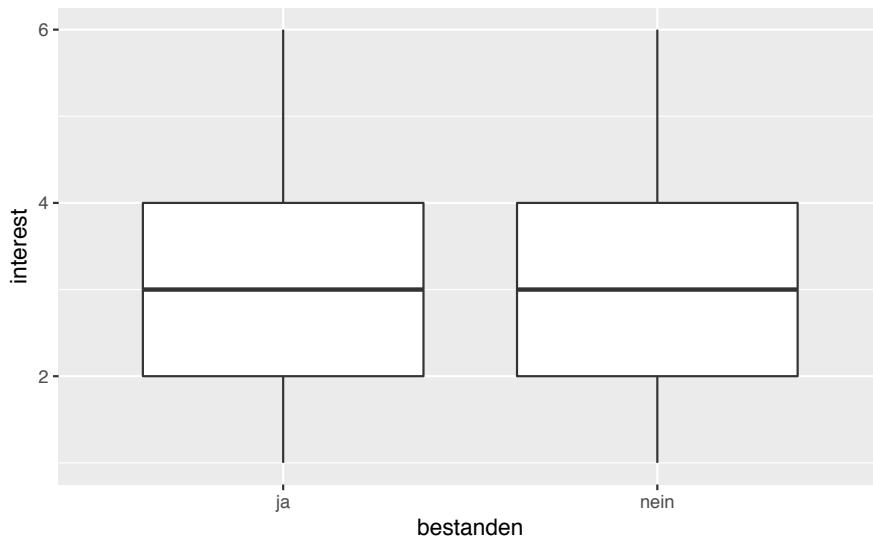
```
qplot(data = stats_test,
      x = bestanden,
      y = interest, geom = "blank") + stat_summary(fun.y = "mean", geom = "point", shape =
```

⁶es gibt kein `study_time == 5` bei den Durchfallen, d.h. bei `bestanden == nein`.



Alternativ kann man hier Boxplots verwenden (deutlich informationsreicher). Grundsätzlich gilt: ein Diagramm, das wenig Information vermittelt, ist überflüssig.

```
qplot(x = bestanden,
      y = interest,
      data = stats_test,
      geom = "boxplot")
```

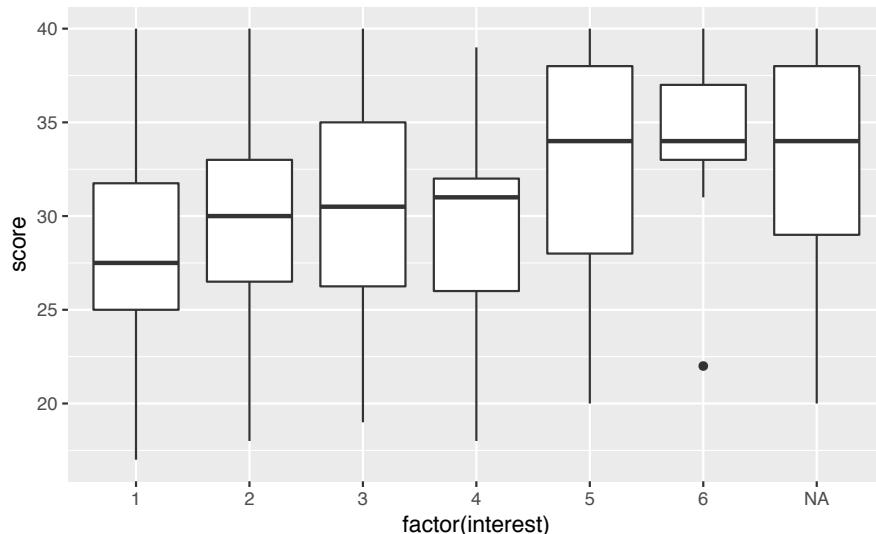


Hm, wie Sie sehen, sehen Sie nix. Kein Unterschied im Median zwischen den Gruppen. Vergleichen wir mal die Punkte zwischen den einzelnen Interessenstufen.

```
qplot(x = factor(interest),
      y = score,
      data = stats_test,
      geom = "boxplot")
```

Tabelle 10.1: Häufige Diagrammtypen

X-Achse	Y-Achse	Diagrammtyp
kontinuierliche Variable	-	Histogramm, Dichtediagramm
kontinuierliche Variable	kontinuierliche Variable	Punkte, Schachbrett-Diagramme
nominale Variable	-	Balkendiagramm
nominale Variable	nominale Variable	Mosaicplot (Fliesen-Diagramm)
nominale Variable	metrische Variable	Punktendiagramm für Zusammenfassungen
nominale Variable	metrische Variable	Boxplots (besser)



Das `factor(interest)` brauchen wir, weil ggplot2 nur dann mehrere Boxplots malt, wenn es Gruppen zum Vergleichen (auf der X-Achse) gibt - sprich wenn auf der X-Achse eine Faktor- oder Textvariable steht.

10.4.6 Überblick zu häufigen Diagrammtypen

Die Tabelle 10.1 und Abbildung 10.6 fassen die gerade besprochenen Diagrammtypen zusammen.

10.5 Die Gefühlswelt von ggplot2

- Geben Sie eine *diskrete X-Achse* an und *keine Y-Achse*, so greift `qplot` im Standard auf das Geom `bar` zurück (Balkendiagramm), falls Sie *kein* Geom angeben:

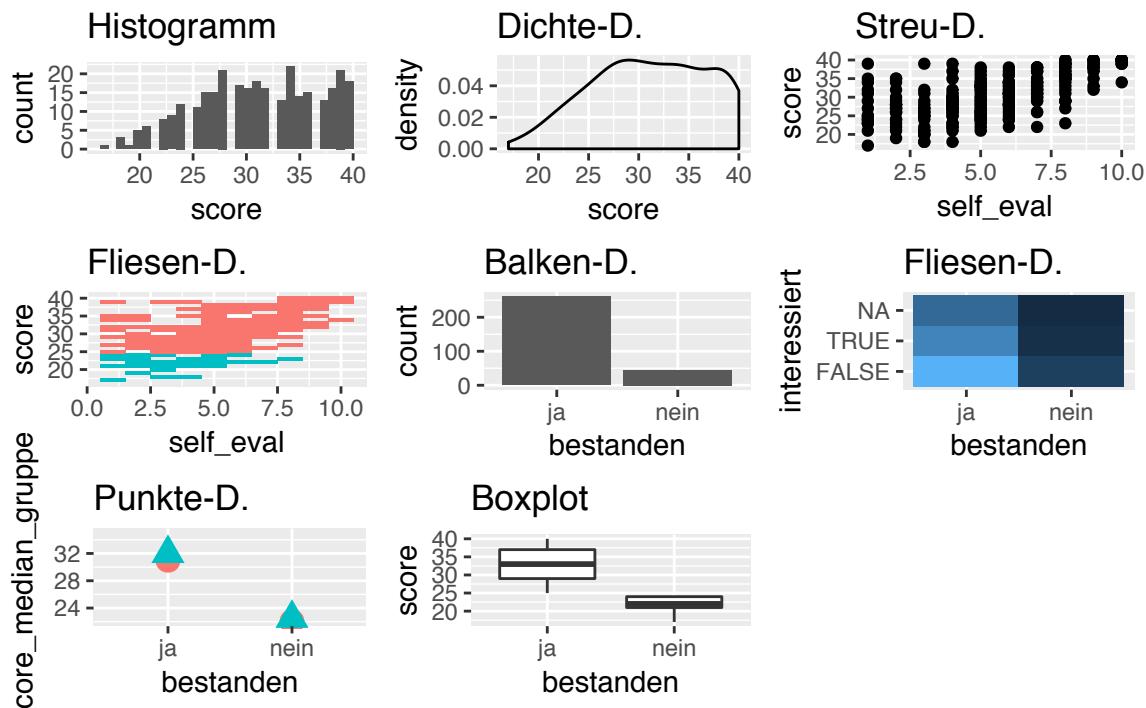


Abbildung 10.6: Überblick zu häufigen Diagrammtypen

```
qplot(x = score, data = stats_test) # identisch zu
qplot(x = score, data = stats_test, geom = "bar")
```

- Geben Sie eine *kontinuierliche X-Achse* an und *keine Y-Achse*, so greift qplot im Standard auf das Geom `histogram` zurück (Histogramm).

```
qplot(x = score, data = stats_test) # identisch zu
qplot(x = score, data = stats_test, geom = "histogram")
```

- Geben Sie eine *kontinuierliche X-Achse* an und eine *kontinuierliche Y-Achse* an, so greift qplot im Standard auf das Geom `point` zurück (Streudiagramm).

```
qplot(x = score, y = self_eval, data = stats_test) # identisch zu
qplot(x = score, y= self_eval, data = stats_test, geom = "point")
```

- Möchten Sie mehrere Geome für eine Variable darstellen, so muss die Gruppierungs-Variable diskret sein:

```
#oh no:
qplot(x = rating, y = affairs, geom = "boxplot", data = Affairs)
```

```
#oh yes:
qplot(x = factor(rating), y = affairs, geom = "boxplot", data = Affairs)

#oh yes:
qplot(x = gender, y = affairs, geom = "boxplot", data = Affairs)
```

10.6 ggplot() der große Bruder von qplot()

qplot() ist dafür gedacht, schnell eine Diagramm zu plotten. Die Syntax ist einfach gehalten, so dass man nicht an allen möglichen Details herumspielen kann. Dafür gibt es ggplot(); mit dieser Funktion hat man Zugriff auf ein großes Spektrum von Einstellungsmöglichkeiten. Die grundlegende Syntax ist ähnlich. Die beiden folgenden Zeilen erstellen das gleiche Diagramm

```
qplot(data = stats_test, x = bestanden, y = interest, geom = "boxplot")
# ist identisch zu
ggplot(data = stats_test) +
  aes(x = bestanden, y = interest) +
  geom_boxplot()
```

Mit aes() wird die “Ästhetik” definiert: Die Spalten des Dataframes werden den visuellen Merkmalen des Diagramms - Achsen und Farben - zugeordnet. Wir werden im Folgenden hauptsächlich mit ggplot() arbeiten, da der Funktionsumfang viel größer ist als bei qplot().

10.7 Aufgaben

1. Erzählen Sie einer vertrauenswürdigen Person jeweils eine “Geschichte”, die das Zustandekommen der vier Plots von Anscombe (Abb. 10.1) erklärt!
2. Abb. 10.4 stellt das mittlerer Budget von Filmen dar; als “Geom” wird ein Boxplot verwendet. Andere Geome wären auch möglich - aber wie sinnvoll wären sie?
3. Erstellen Sie ein Diagramm, welches Histogramme der Verspätung verwendet anstelle von Boxplots! Damit das Diagramm nicht so groß wird, nehmen Sie zur Gruppierung nicht `carrier` sondern `origin`.
4. Ist das Histogramm genauso erfolgreich wie der Boxplot, wenn es darum geht, viele Verteilungen vergleichend zu präsentieren? Warum?
5. Erstellen Sie ein sehr grobes und ein sehr feines Histogramm für die Schuhgröße!
6. Vertiefung: Erstellen Sie ein Diagramm, das sowohl eine Zusammenfassung (Mittelwert) der Körpergrößen nach Geschlecht darstellt als auch die einzelnen Werte darstellt!



Abbildung 10.7: Film-Budgets mit Histogrammen

10.8 Lösungen

1. :-)

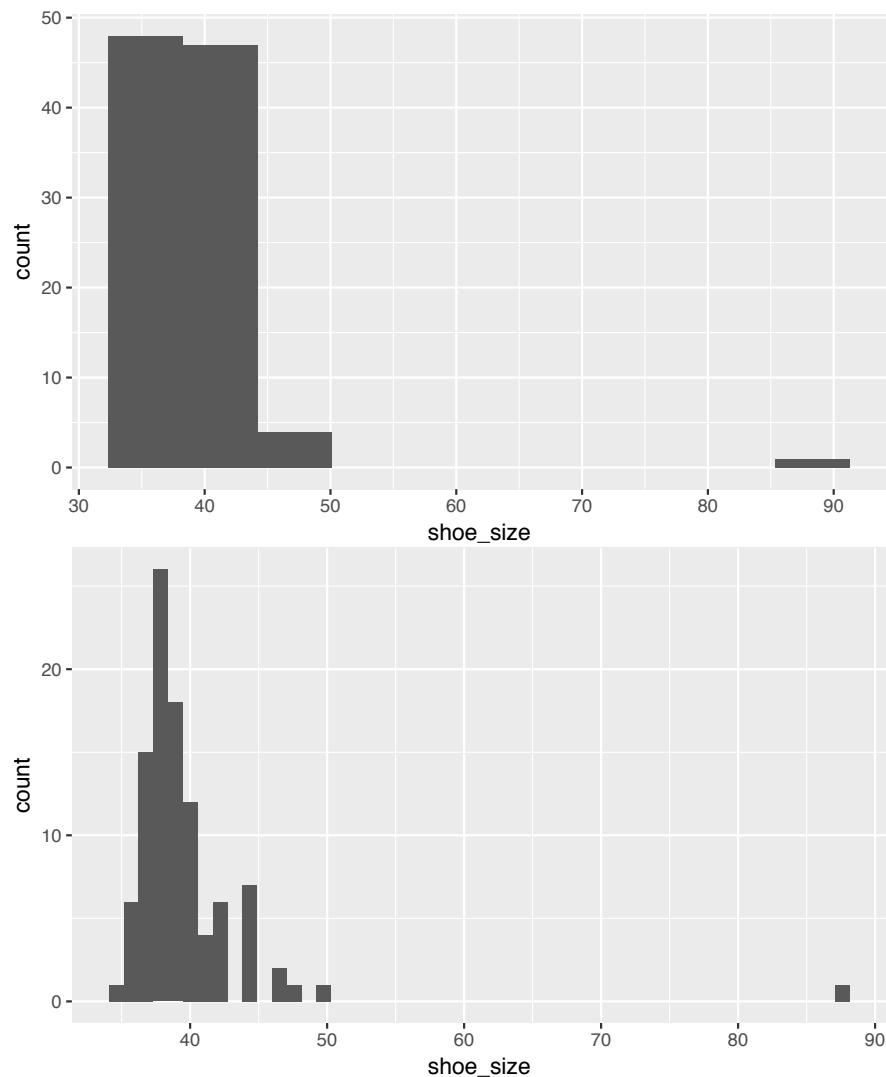
2. :

```
qplot(x = budget, geom = "histogram", data = movies, facets = ~factor(Jahrzehnt))
```

Der Boxplot ist besser geeignet als das Histogramm, um mehrere Verteilungen vergleichend zu präsentieren (vgl. Abb. 10.7). Durch die gleiche Ausrichtung der Boxplots ist es dem Auge viel einfacher, Vergleiche anzustellen im Vergleich zu den Histogrammen. Einen optisch schöneren Effekt könnte man mit `geom_jitter` anstelle von `geom_pointer` erreichen. Auch die Reihenfolge der beiden Geome könnte man umdrehen. Natürlich ist auch an Form, Größe und Farbe der Geome noch zu feilen.

3. :

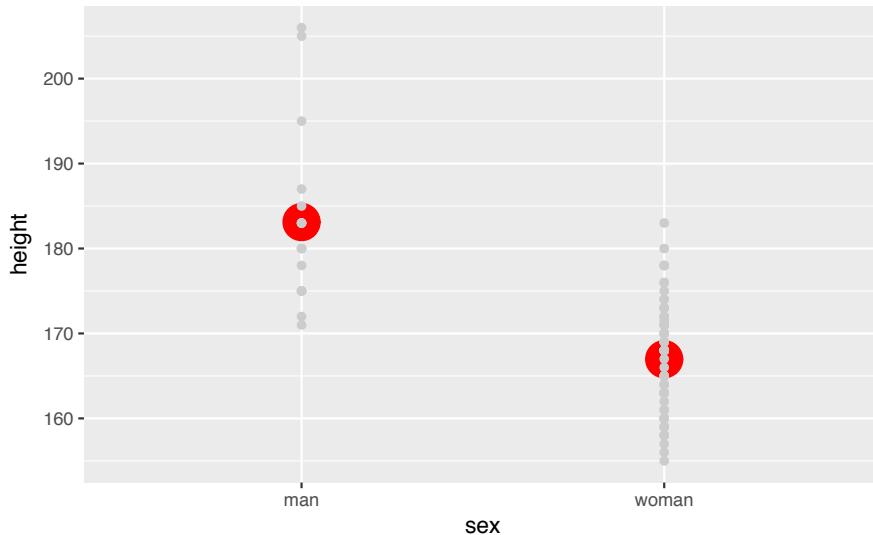
```
qplot(x = shoe_size, data = wo_men, bins = 10)
qplot(x = shoe_size, data = wo_men, bins = 50)
```



4. :

```
wo_men2 %>%
  group_by(sex) %>%
  summarise(height = mean(height)) -> wo_men3
```

```
wo_men3 %>%
  ggplot() +
  aes(x = sex, y = height) +
  geom_point(color = "red", size = 8) +
  geom_point(data = wo_men2, color = "grey80")
```



Der “Trick” ist hier, erst die zusammengefassten Daten in ein Geom zu stecken (`wo_men3`). Dann werden die Rohdaten (`wo_men2`) ebenfalls in ein Geom gepackt. Allerdings muss die Achsen-Beschriftung bei beiden Geomen identisch sein, sonst gibt es eine Fehlermeldung.

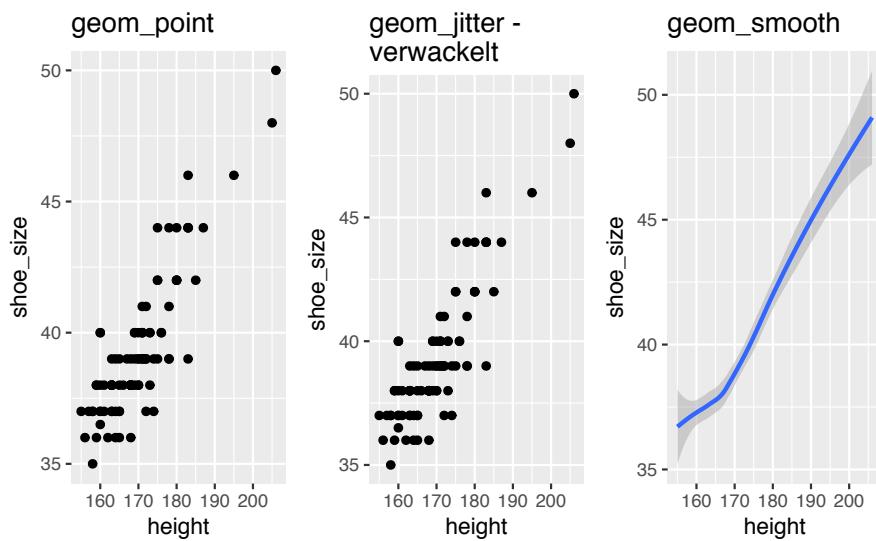
10.9 Richtig oder Falsch⁷



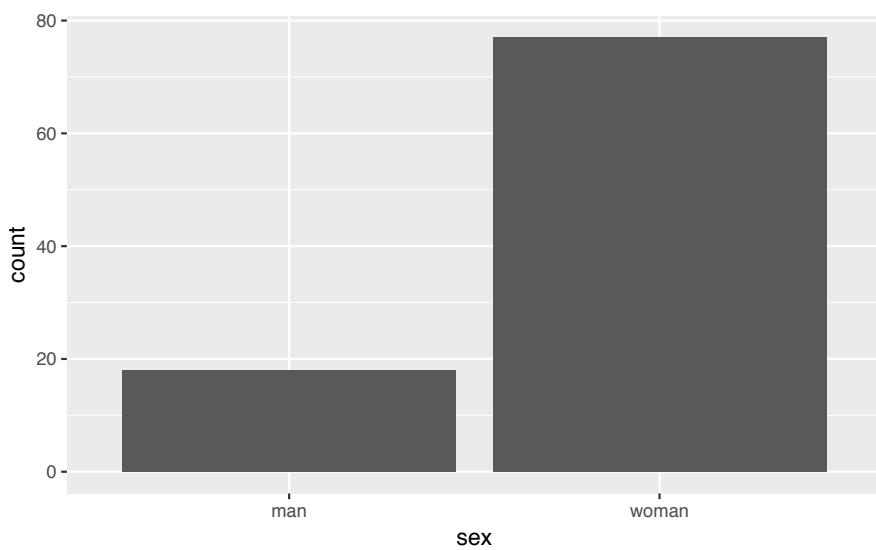
Richtig oder Falsch!

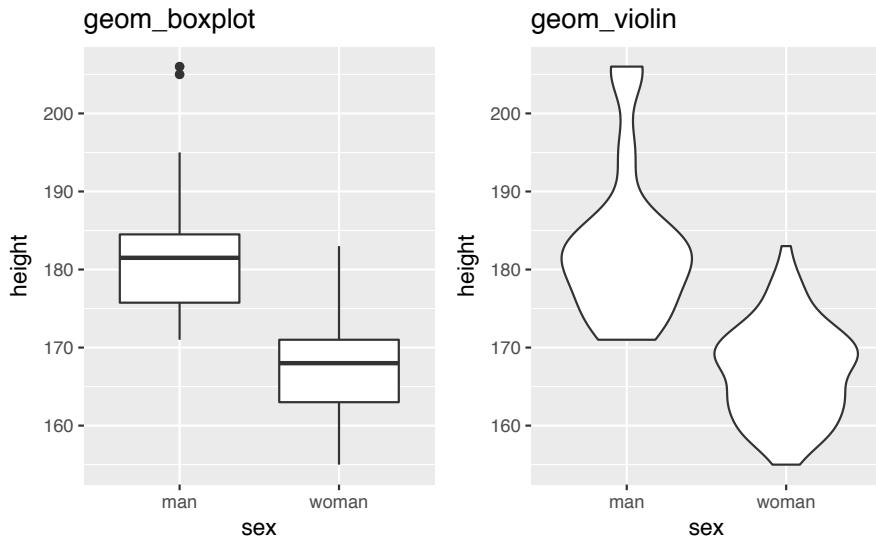
1. Diese Geome gehören zum (Standard-) ggplot2: bar, histogram, point, density, jitter, boxplot.
2. `qplot` ist eine Funktion im Paket `ggplot2`.
3. Mit `aes` definiert man, wie “ästethisch” das Diagramm sein soll (z.B. grauer Hintergrund vs. weißer Hintergrund, Farbe der Achsen etc.).
4. Diese Geome gehören zum (Standard-) ggplot2: smooth, line, boxwhisker, mosaicplot.
5. Möchte man ein Diagramm erstellen, welches auf der X-Achse `total_bill`, auf der Y-Achse `tip` darstellt, als Geom Punkte verwendet und die Daten aus der Tabelle `tips` bezieht, so ist folgende Syntax korrekt: ‘`qplot(x = total_bill, y = tip, geom = “point”, data = tips)`’

⁷R, R, F, F, R



```
ggplot(wo_men2) +  
  aes(x = sex) +  
  geom_bar()
```





10.10 Sonstiges

<https://ikashnitsky.github.io/2017/dd-journals-frequency/>

10.11 Zum Weiterlesen

Einen guten Überblick über Geome bietet das Cheatsheet von ggplot2⁸. Einen Befehlsüberblick zu ggplot2 findet sich hier: <http://ggplot2.tidyverse.org/reference/>. Hadley Wickham, Autor von ggplot2 hat auch ein Buch über sein Paket geschrieben (2016a). Edward Tufte gilt als Grand Seigneur der Datenvisualisierung; er hat mehrere lesenswerte Bücher zu dem Thema geschrieben (Tufte 2001; Tufte 2006; Tufte 1990). William Cleveland, ein amerikanischer Statistiker ist bekannt für seine grundlegenden, und weithin akzeptierten Ansätze für Diagramme, die die wesentliche Aussage schnörkellos transportieren (Cleveland 1993). Die (graphische) Auswertung von Umfragedaten basiert häufig auf Likert-Skalen. Ob diese metrisches Niveau aufweisen, darf bezweifelt werden. Hier findet sich einige vertiefenden Überlegungen dazu und zur Frage, wie Likert-Daten ausgewertet werden könnten: <https://bookdown.org/Rmadillo/likert/>. Es finden sich viele Tutorials online zu ggplot2; ein deutschsprachiger Tutorial findet sich hier: <http://md.psych.bio.uni-goettingen.de/mv/unit/ggplot2/ggplot2.html>.

⁸<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

Kapitel 11

Farben wählen



Lernziele:

- Mindestens eine bekannte Farbauswahl (Palette) nennen können.
- Farbpaletten bei ggplot2 anwenden können.
- Themen bei ggplot2 ändern können.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Zum Plotten
library(wesanderson) # Farb-Palette von Wes Anderson
library(RColorBrewer) # Farb-Palette von Cynthia Brewer
library(knitr) # für HTML-Tabellen
library(gridExtra) # für kombinierte Plots
library(ggthemes) # für zusätzliche ggplot2-Themen (Layouts)
library(nycflights13)
library(cowplot) # ein weiteres Theme für ggplot2
```

Erstens, nicht schaden - so könnte hier die Maßregel der Wahl von Farben sein. Es ist leicht, zu grelle oder wenig kontrastierende Farben auszuwählen. Eine gute Farbauswahl (Palette) ist nicht so leicht und hängt vom Zweck der Darstellung ab.

Meine Erfahrung ist, dass viele die Standard-Farbwahl von ggplot2 nicht besonders lieben. Wenn es Ihnen so geht: kein Problem, es gibt viele Alternativen.

Bevor Sie sich an die Zusammenstellung Ihrer Lieblingsfarben stürzen, ein Cave:

Ihre Farbpalette sollte wiederspiegeln...

ob es sich um eine diskrete oder um eine kontinuierliche Variable handelt je nach Anzahl der darzustellenden unterschiedlichen Werte sollte die Farbwahl sich ändern (v.a. bei diskreten Variablen). Wenn Sie nur wenige Farben haben, sollten die Farben im Farbkreis recht weit

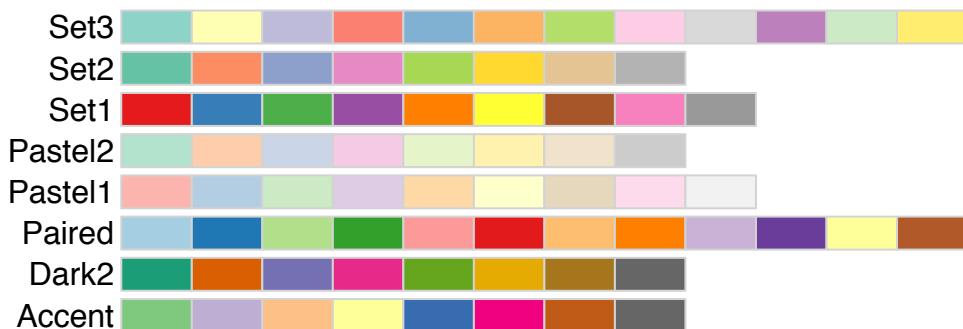
auseinander liegen, um unterschiedlich genug zu sein. Bei mehr Farben ändert sich damit die Wahl der Farben. Man benötigt also am besten für jede Anzahl von Kategorien eine Palette die Farben sollten so unterschiedlich wie nötig, aber die Unterschiede so dezent wie möglich sein Rot-Grün-Blindheit kann (könnte) berücksichtigt werden, da recht verbreitet S/W-Drucker können nur Helligkeitsunterschiede wiedergeben. Kurz: Die Entwicklung eigener Farbpaletten kann umfangreich sein, wenn man es ordentlich machen will. Glücklicherweise gibt es eine Reihe guter Paletten, die frei verfügbar sind, und viele Szenarien abdecken. In R (und für ggplot2) sind viel davon verfügbar. Ein Beispiel für gute, handgewählte Paletten ist die Auswahl von Frau Brewer.

11.1 Die Farben von Cynthia Brewer

Cynthia Brewer¹ hat einige schöne Farbpaletten zusammengestellt; diese sind in R und in ggplot2 über das Paket **RcolorBrewer** verfügbar. Mit **brewer.pal.info()** bekommen Sie einen Überblick.

- Kontrastierende Darstellung (nominale/ qualitative Variablen) - z.B. Männer vs. Frauen

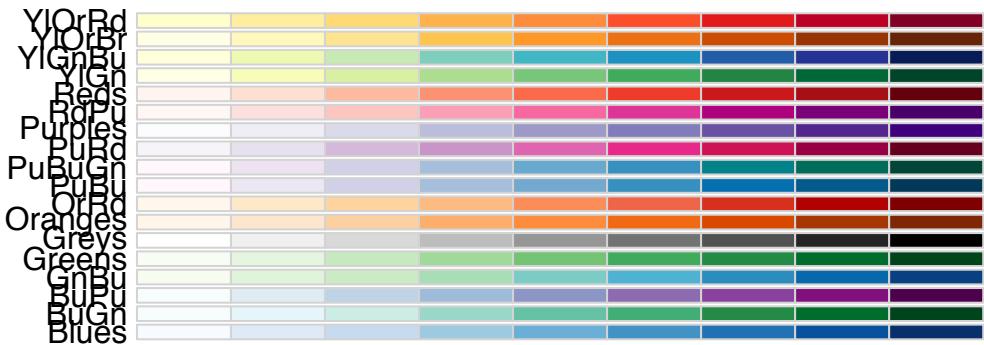
```
display.brewer.all(type="qual")
```



- Sequenzielle Darstellung (unipolare numerische Variablen) - z.B. Preis oder Häufigkeit

```
display.brewer.all(type="seq")
```

¹<http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3>



- Divergierende Darstellung (bipolare numerische Variablen) - z.B. semantische Potenziale oder Abstufung von "stimme überhaupt nicht zu" über "neutral" bis "stimme voll und ganz zu"

```
display.brewer.all(type="div")
```



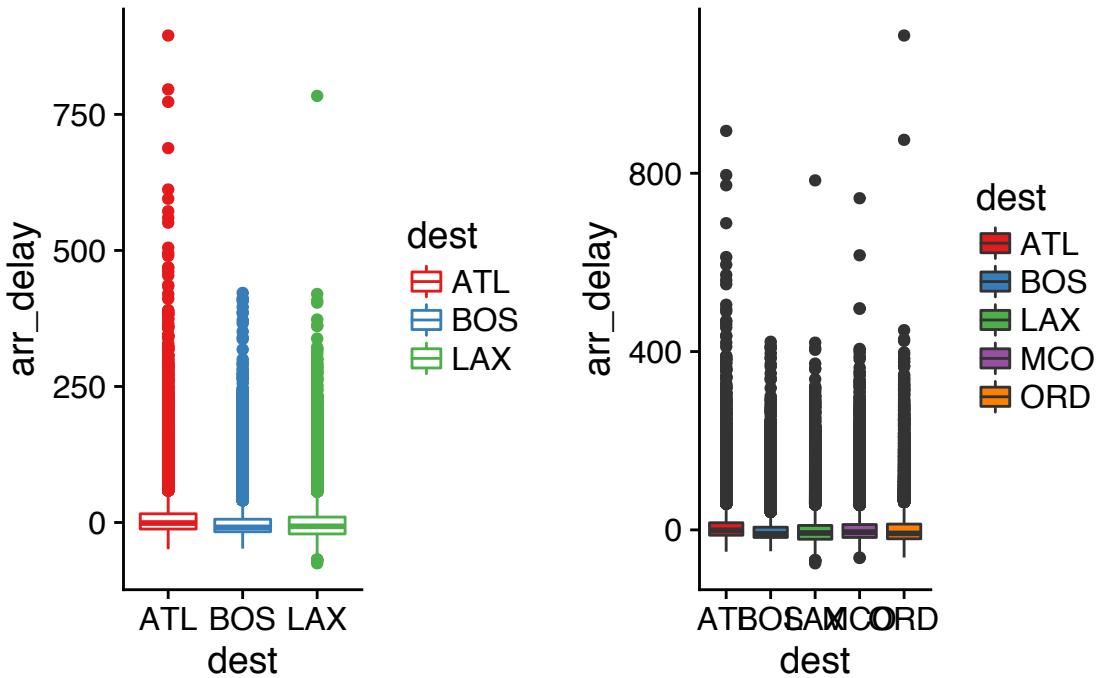
In ggplot2 können wir folgendermaßen Paletten ändern (dazu laden wir den Datensatz flights noch einmal, falls Sie ihn nicht mehr geladen haben), s. Abbildung ??:

```
data(flights)

p1 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = arr_delay, color = dest) +
  geom_boxplot() +
  scale_color_brewer(palette = "Set1")

p2 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX", "MCO", "ORD")) %>%
  ggplot() +
  aes(x = dest, y = arr_delay, fill = dest) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set1")
```

```
grid.arrange(p1, p2, ncol = 2)
```



`scale_color_brewer` meint hier: “Ordne der Variablen, die für ‘color’ zuständig ist, hier `sex`, eine Farbe aus der Brewer-Palette ‘Set1’ zu”. Die Funktion wählt *automatisch* die richtige Anzahl von Farben.

Man beachte, dass die Linienfarbe über `color` und die Füllfarbe über `fill` zugewiesen wird. Punkte haben nur eine Linienfarbe, keine Füllfarbe.

11.2 Die Farben von Wes Anderson

Auch die Farbpaletten von Wes Anderson sind erbaulich². Diese sind nicht “hart verdrahtet” in ggplot2, sondern werden über `scale_XXX_manual` zugewiesen (wobei XXX z.B. `color` oder `fill` sein kann).

```
data(tips, package = "reshape2")

p1 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = wes_palette("GrandBudapest")) +
```

²<https://github.com/karthik/wesanderson>

```

theme(legend.position = "bottom")

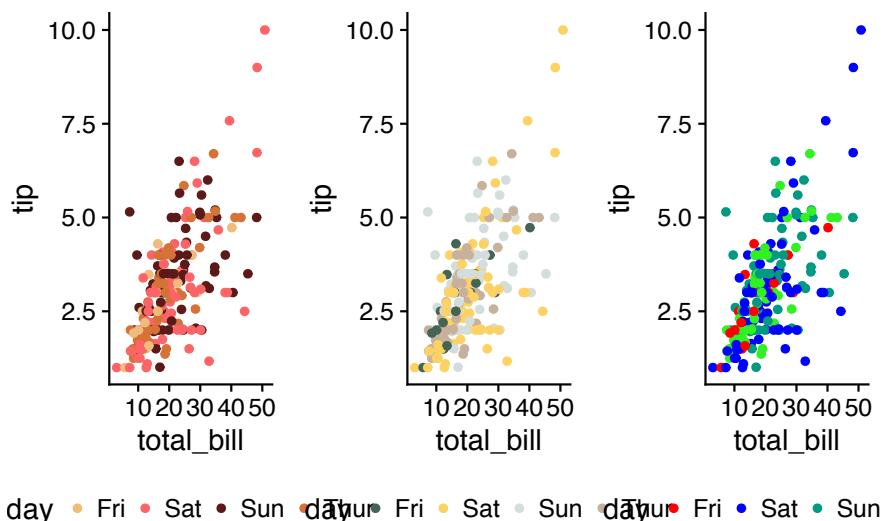
p2 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = wes_palette("Chevalier")) +
  theme(legend.position = "bottom")

meine_farben <- c("red", "blue", "#009981", "#32F122")

p3 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = meine_farben) +
  theme(legend.position = "bottom")

grid.arrange(p1, p2, p3, ncol = 3)

```



Wer sich berufen fühlt, eigene Farben (oder die seiner Organisation zu verwenden), kommt auf ähnlichem Weg zu Ziel. Man definiere sich seine Palette, wobei ausreichend Farben definiert sein müssen. Diese weist man dann über `scale_XXX_manual` dann zu. Man kann einerseits aus den in R definierten Farben auswählen³ oder sich selber die RGB-Nummern (in Hexadezimal-Nummern) heraussuchen.

³<http://sape.inf.usi.ch/quick-reference/ggplot2/colour>

11.3 Themen ändern

Ein “Thema” bei ggplot2 umfasst die Gestaltung aller Aspekte eines Diagramms, welche nicht die Daten betreffen, also Axen, Hintergrund, Titel etc⁴. ggplot2 kommt mit einer Auswahl an “eingebauten” Themen, aber es gibt noch einige weitere Themen in anderen Paketen.

Betrachten wir zuerst die “Standard-Themen”:

```
p1 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = air_time) +
  geom_boxplot() +
  theme_classic() +
  ggtitle("theme_classic") +
  theme(legend.position = "none")

p2 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = air_time) +
  geom_boxplot() +
  theme_bw() +
  theme(legend.position = "none") +
  ggtitle("theme_bw")

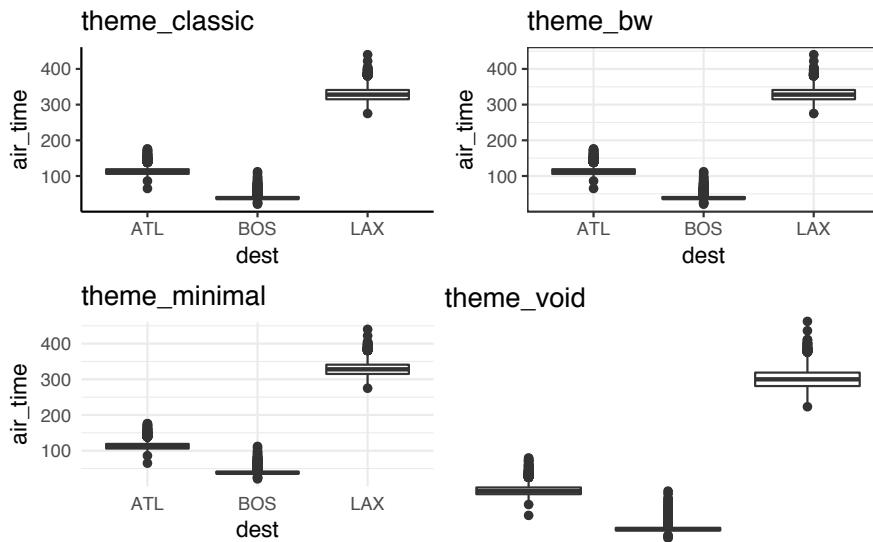
p3 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = air_time) +
  geom_boxplot() +
  theme_minimal() +
  theme(legend.position = "none") +
  ggtitle("theme_minimal")

p4 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = air_time) +
  geom_boxplot() +
```

⁴<http://docs.ggplot2.org/dev/vignettes/themes.html>

```
theme_void() +
  theme(legend.position = "none") +
  ggtitle("theme_void")

grid.arrange(p1, p2, p3, p4, ncol = 2)
```



Mit `theme(legend.position = "none")` kann man noch die Legende abschalten.

Über das Paket `ggthemes`⁵ kann man ein gutes Dutzend Themen anfordern. Probieren Sie mal `theme_excel()`, `theme_tufte()` und `theme_base()`.

Weitere Themes sind verfügbar⁶. Ein recht schönes, weil klares Design (Theme) für `ggplot2` bietet `cowplot`, s. Abbildung 11.1.

```
flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = air_time, color = dest) +
  geom_boxplot()
```

⁵<https://cran.r-project.org/web/packages/ggthemes/vignettes/ggthemes.html>

⁶<https://github.com/ricardo-bion/ggtech>

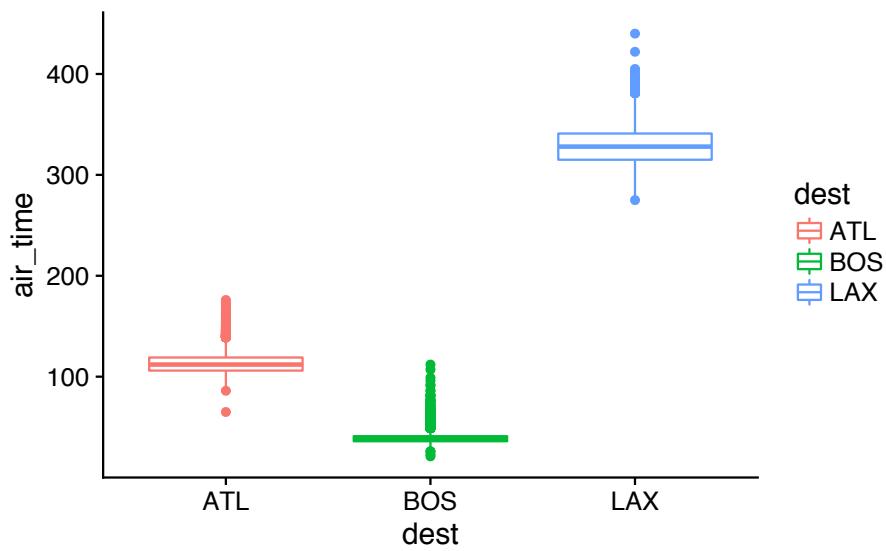


Abbildung 11.1: GGplot mit dem Thema 'cowplot'

Kapitel 12

Fallstudie zur Visualisierung



Lernziele:

- Diagramme für nominale Variablen erstellen können.
- Balkendiagramme mit Prozentpunkten auf der Y-Achse erstellen können.
- Balkendiagramme drehen können.
- Text-Labels an Balkendiagramme anfügen können.
- Farbschemata von Balkendiagrammen ändern können.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse)  # Plotten
library(likert)
library(viridis)   # alternatives Farbschema
library(scales)
```

Eine recht häufige Art von Daten in der Wirtschaft kommen von Umfragen in der Belegschaft. Diese Daten gilt es dann aufzubereiten und graphisch wiederzugeben. Das ist der Gegenstand dieser Fallstudie. Wir nutzen dazu den Datensatz `extra`:

```
data(extra, package = "pradadata")
```

Neben Extraversion wurden noch ein paar weitere Variablen erhoben (`?extra`); uns interessieren hier aber nur die 10 Extraversionsitems, die zusammen Extraversion als Persönlichkeitseigenschaft messen (sollen). Wir werden die Antworten der Befragten darstellen, aber uns hier keine Gedanken über Messqualität u. ä. machen. Schauen sie sich die Daten mal an:

```
glimpse(extra)
```

12.1 Umfragedaten visualisieren mit likert

Das Paket `likert` erlaubt komfortables Arbeiten mit Likert-skalierten Items.

Im ersten Schritt erstellen wir den Dataframe, der die darzustellenden Items enthält. Allerdings verdaut `likert` nur klassische Dataframes, keine neumodischen Tibbles, daher müssen wir `extra` noch in einen `data.frame` umwandeln mit `as.data.frame()`. Außerdem zieht `likert` den Variablenotyp `factor` vor:

```
extra %>%
  select(i01:i10) %>%
  mutate_all(factor) %>%
  as.data.frame -> extra_items
```

Die Faktorstufen kann man sich mit `map(extra_items, levels)` anschauen. Im zweiten Schritt führen die wir Funktion `likert()` aus, die die Daten aufbereitet, so dass sie dann gut geplottet werden können. Betrachten Sie das ausgegebene Objekt, z.B. mit `str(extra_items_likert)`:

```
extra_items %>%
  likert -> extra_items_likert
```

Das Objekt `extra_items_likert` hat das Attribut “likert” (wie uns z.B. `str()`) verrät. Übergeben wir ein Objekt mit diesem Typ an `plot`, so wird automatisch die “richtige” Plot-Methode ausgeführt, mit ansehnlichem Ergebnis (s. Abbildung 12.1).

```
plot(extra_items_likert)
```

Die Hauptarbeit haben wir geschafft; man könnte natürlich noch etwas an der Schönheit feilen. So könnte man die Namen der Items und die Namen der Antwortskala aussgekräftiger benennen, einen Titel hinzufügen und die Farben ändern. Die Webseite zu `likert`¹ und die Hilfeseiten (z.B. `?likert.options`) dokumentieren die Einstellungen für diese Funktion. Schauen wir uns ein paar Optionen an. Zuerst suchen wir die Itemtexte heraus und benennen die Variablen entsprechend um:

```
data(extra_names, package = "pradadata")
item_labels <- extra_names$extra_names[3:12]
names(extra_items) <- item_labels
```

Dann benennen wir die Antwortstufen um; `likert::recode` erwartet allerdings `character` als Eingabe; daher müssen wir etwas umständlich die Spalten erst in den Typ `character` umwandeln; zum Schluss dann wieder zurück in den Typ `factor` für `likert()`:

¹<http://jason.bryer.org/likert/>

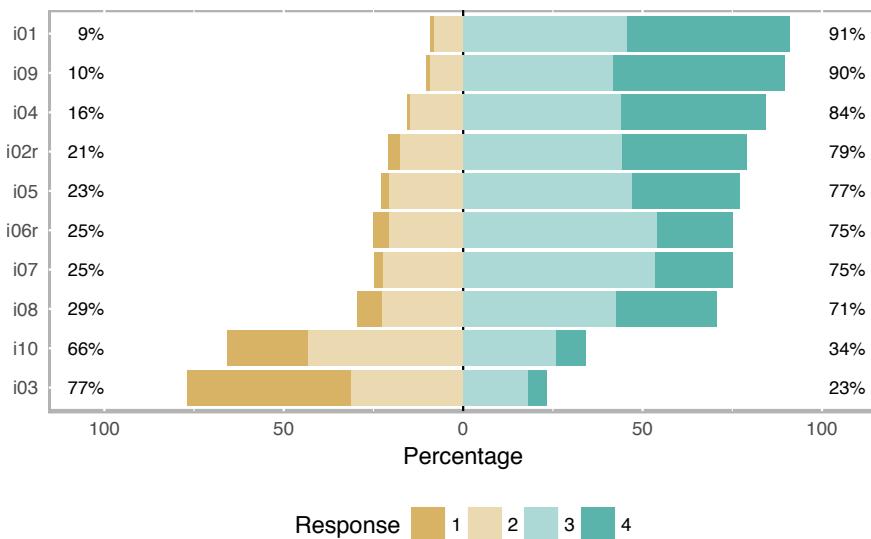


Abbildung 12.1: Umfrageergebnisse visualisieren mit ‘likert’

```
extra_items %>%
  mutate_all(recode_factor,
    "1" = "stimme nicht zu",
    "2" = "stimme eher nicht zu",
    "3" = "stimme eher zu",
    "4" = "stimme voll und ganz zu") %>%
as.data.frame -> extra_items_r
```

Die Funktion `recode_factor()` aus `dplyr` kann man zum Umkodieren von Faktorvariablen verwenden. Mit `mutate_all` werden die Funktionen die mit `fun_()` angegeben sind - hier nur `recode()` auf alle Spalten von `extra` angewendet. Etwaige Parameter von der anzuwendenden Funktion werden mit Komma angefügt. Dann plotten wir wieder (der Übersichtlichkeit halber nur drei Items):

```
extra_items_r_likert <- likert(items = extra_items_r[1:3])
plot(extra_items_r_likert)
```

Mit `summary(extra_items_r_likert)` kann man sich eine Zusammenfassung der Item-Antworten geben lassen.

12.2 Umfragedaten visualisieren mit ggplot

`likert` bietet starke Vorverarbeitung der Daten an, so dass man als Nutzer nur noch wenig selber machen muss. Will man allerdings hohe Kontrolle über das Diagramm und mit vielfach getesteten Werkzeugen arbeiten (`likert` ist noch sehr neu), so bietet sich `ggplot` an.

12.2.1 Daten umstellen

Führen wir uns ein Diagramm vor Augen (s. Abbildung XXX), bei dem auf der X-Achse die Items stehen (1, 2, ..., n) und auf der Y-Achse die Anzahl der Kreuze nach Kategorien. Viele Grafik-Funktionen sind nun so aufgebaut, dass auf der X-Achsen nur *eine* Variable steht. `ggplot2`, das wir hier verwenden, ist da keine Ausnahme. Wir müssen also die “breite” Tabelle (10 Spalten, pro Item eine) in eine “lange Spalte” umbauen: Eine Spalte heißt dann “Itemnummer” und die zweite “Wert des Items” oder so ähnlich. Wie oben wählen wir aus der Fülle der Daten, die Spalten, die uns interessieren: Die 10 Extraversionsitems (Spalten 3 bis 12).

```
extra_items <- dplyr::select(extra, 3:12)
```

Dann stellen wir die Daten von “breit” nach “lang” um, so dass die Items eine Variable bilden und damit für `ggplot2` gut zu verarbeiten sind.

```
extra_items %>%
  gather(key = items, value = Antwort) %>%
  mutate(items = factor(items),
        Antwort = factor(Antwort)) -> extra_items_long
```

Warum wandeln wir die Item-Antworten in eine Textvariable um? Es waren doch so schöne Zahlen! Betrachten Sie zur Erläuterung Abbildung 12.2; wie man sieht, sind die Füllfarben der Balken den Stufen der Variablen `Antwort` zugeordnet. `ggplot` wird aber diese diskrete Farbzuzuordnung nur vornehmen, wenn die Gruppierungsvariable (`Antwort`) nominalskaliert ist (vom Typ `factor` oder `character`). Damit haben wir es schon! Jetzt wird gemalt.

12.2.2 Diagramme für Anteile

Stellen wir die Anteile der Antworten anhand von farbig gefüllten Balken dar (s. Abbildung 12.2). Beachten Sie, dass die Balken alle auf 1 (100%) skaliert sind; es werden also *relative* Häufigkeiten dargestellt. Absolute Häufigkeiten bleiben hier außen vor (s. Abb. 12.2).

```
p1 <- ggplot(data = extra_items_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  theme(legend.position = "bottom") +
  theme(axis.text = element_text(size=rel(0.5)))
```



Was macht dieser `ggplot` Befehl? Schauen wir es uns in Einzelnen an:

- `ggplot(data = ...)`: Wir sagen “Ich möchte gern die Funktion ggplot nutzen, um den Datensatz ... zu plotten”.
- `aes(...)`: Hier definieren wir die “aesthetics” des Diagramms, d.h. alles “Sichtbare”. Wir ordnen in diesem Fall der X-Achse die Variable `items` zu. Per Standardeinstellung geht `ggplot` davon aus, dass sie die Häufigkeiten der X-Werte auf der Y-Achse haben wollen, wenn Sie nichts über die Y-Achse sagen. Jetzt haben wir ein Koordinatensystem definiert (das noch leer ist).
- `geom_bar()`: “Hey R oder ggplot, jetzt male mal einen barplot in den ansonsten noch leeren plot”.
- `aes(fill = Antwort)`: Genauer gesagt nutzen wir `aes` um einen sichtbaren Aspekt des Diagramms (wie die X-Achse) eine Variable des Datensatzes zuzuordnen. Jetzt sagen wir, dass die Füllung (im Balkendiagramm) durch die Werte von `Antwort` definiert sein sollen (also “1”, “2” etc.).
- `position = "fill"` sagt, dass die Gesamt-Höhe des Balken aufgeteilt werden soll mit den “Teil-Höhen” der Gruppen (Antwort-Kategorien 1 bis 4); wir hätten die Teil-Höhen auch nebeneinander stellen können.

Vielelleicht ist es schöner, die NAs erst zu entfernen:

```
extra_items_long <- na.omit(extra_items_long)
```

Plotten Sie das Diagramm dann noch mal:

```
ggplot(data = extra_items_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill")
```

12.2.3 Rotierte Balkendiagramme

Dazu ergänzen wir die Zeile `+ coord_flip()`; das heißt so viel wie “drehe das Koordinaten- system um 90 Grad” (s. Abbildung 12.2).

```
p1_flip <- p1 + coord_flip()
```

Übrigens: Auch bei “geflippten Koordinaten” bleibt für `ggplot2` die X-Achse die X-Achse. In unserem Beispiel ist die Variable `items` der X-Achse zugeordnet, darauf hat die Achsendrehung keinen Einfluss.

Halt, ein Problem haben wir noch. Im Diagramm “liegen” die größeren Kategorien (z.B. 4) “links” von den kleineren Kategorien (z.B. 1). Die Reihenfolge der Kategorien sollte aber

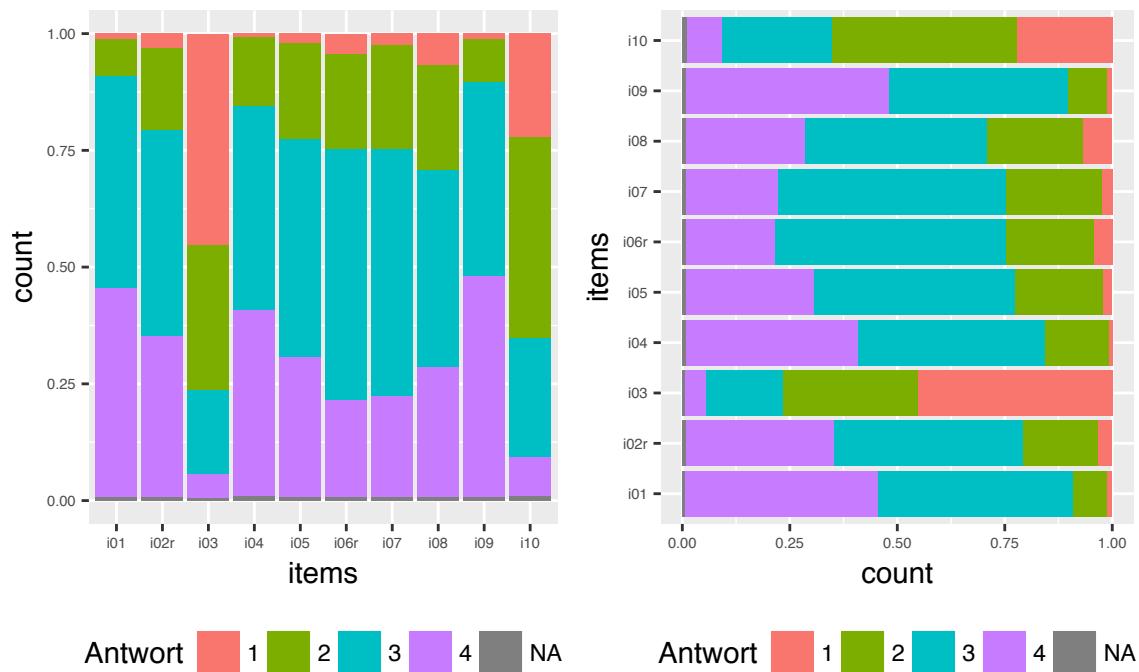


Abbildung 12.2: Balkendiagramm, unrotiert und rotiert

lieber aufsteigend von links nach rechts sein. Daher lieber die Reihenfolge umkehren. Konkret “drehen” wir die Reihenfolge der Faktorstufen von Antwort mit rev (rev wie reverse).

```
extra_items_long %>%
  mutate(Antwort = factor(Antwort,
                          levels = rev(levels(Antwort)))) ->
  extra_items_long_rev
```

Um diese Syntax besser zu verstehen, vergleichen Sie einmal die Ausgaben dieser beiden Zeilen:

```
levels(extra_items_long_rev$Antwort)
levels(extra_items_long_rev$Antwort) %>% rev
```

Jetzt plotten Sie die Daten wie gewohnt²:

```
extra_items_long_rev %>%
  ggplot(aes(x = items)) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip()
```

²Hier: http://docs.ggplot2.org/current/position_stack.html findet man einige Hinweise dazu aus der Dokumentation von ggplot2

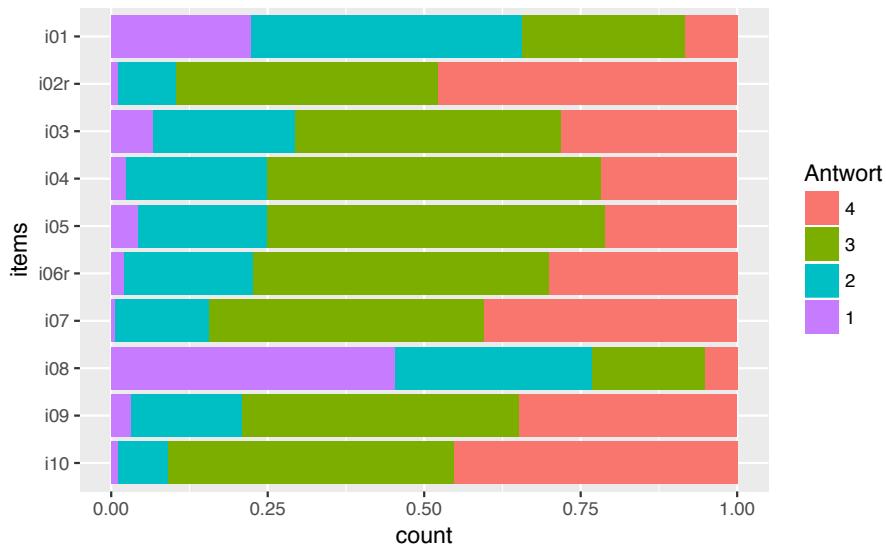


Abbildung 12.3: Itemnummern von oben nach unten aufsteigend

Standardmäßig platziert `ggplot2` geringe Werte auf einem kleineren Skalenwert der Abszisse, was Sinn macht. Allerdings passt es dann nicht mehr zur Leserichtung von oben nach unten. Drehen wir also die Reihenfolge der Kategorien um, so dass die Reihenfolge der Leserichtung entspricht (s. Abb. ??).

```
ggplot(extra_items_long_rev, aes(x = items)) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip() +
  scale_x_discrete(labels = rev(levels(extra_items_long_rev$items))) -> p2
p2
```

Kommentieren Sie mal die Zeile mit `coord_flip()` aus, und betrachten Sie den Unterschied.

Die Funktion `rev(levels(data2$items))` dreht die Reihenfolge der Faktorstufen (`levels`) um (`rev` wie “reverse”). Dann sagen wir `ggplot2`, dass wir die X-Achse (auf der eine diskrete Variable dargestellt ist), anpassen möchten. Genauer gesagt, übergeben wir einen Vektor mit den Bezeichnungen der Kategorien der Achse.

12.2.4 Text-Labels für die Items

Wir definieren die Texte (“Labels”) für die Items:

```
item_labels <- c("Ich bin das erste Item",
                 "Das zweite Item",
                 "Item 3 sdjfkladsjk",
```

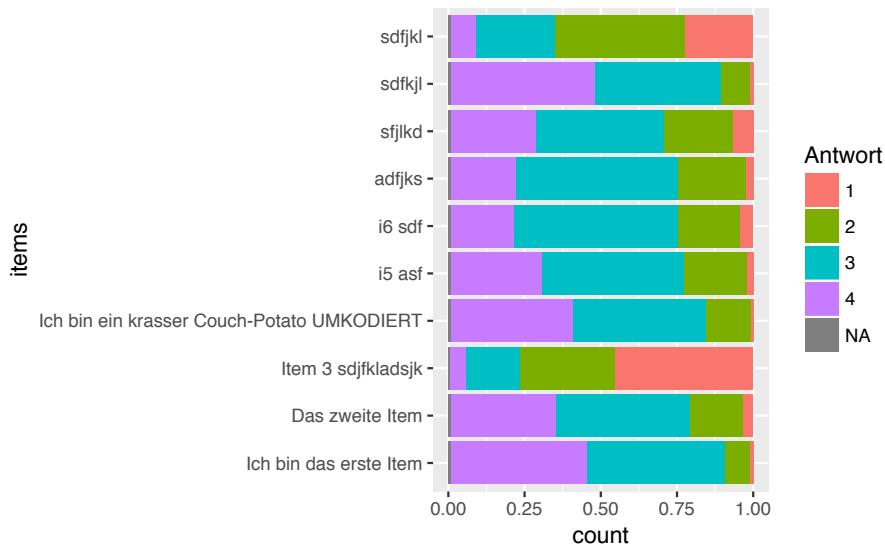


Abbildung 12.4: Rotiertes Balkendiagramm mit Item-Label

```
"Ich bin ein echter Couch-Potato UMKODIERT",
 "i5 asf", "i6 sdf", "adfjks", "sfjlk", "sdfkjl", "sdfjkl")
```

Jetzt hängen wir die Labels an die Items im Diagramm (s. Abbildung 12.4).

```
p1 +
  coord_flip() +
  scale_x_discrete(labels = item_labels)
```

Man kann auch einen Zeilenumbruch in den Item-Labels erzwingen... wobei uns das führt schon recht weit, aber für die Schönheit...

```
item_labels <- c("Ich bin das erste Item",
                "Das zweite Item",
                "Item 3 sdjfkladsjk",
                "Ich bin ein krasser \nCouch-Potato***mit Zeilenumbruch***",
                "i5 asf", "i6 sdf", "adfjks", "sfjlk", "sdfkjl", "sdfjkl")
```

Plotten Sie das dann wieder:

```
ggplot(data = extra_items_long_rev) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip() +
  scale_x_discrete(labels = item_labels, name = "Extraversionsitems") +
```

```
scale_y_continuous(name = "Anteile")
```

12.2.5 Diagramm mit Häufigkeiten

Ach so, schön wäre noch die echten Zahlen an der Y-Achse, nicht Anteile. Dafür müssen wir unseren Diagrammtyp ändern, bzw. die Art der Anordnung ändern. Mit `position = "fill"` wird der Anteil (also mit einer Summe von 100%) dargestellt. Wir können auch einfach die Zahlen/Häufigkeiten anzeigen, in dem wir die Kategorien “aufeinander stapeln”. Probieren Sie dazu die folgende Syntax.

```
ggplot(data = extra_items_long_rev) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "stack") +
  coord_flip() +
  scale_x_discrete(labels = item_labels)
```

12.3 Farbschemata

Ja, die Wünsche hören nicht auf... Also, sehen wir uns noch andere Farbschemata an. In Abbildung 12.5) ist links die Brewer-Palette 17 dargestellt und rechts die Viridis-Farbpalette.

```
p3 <- p1 +
  scale_fill_brewer(palette = 17) +
  theme(legend.position = "bottom")
```

Das Paket `viridis` hat ein gutes Farbschema. Probieren Sie es mal aus:

```
p4 <- p1 +
  scale_fill_viridis(discrete = TRUE) +
  theme(legend.position = "bottom")
```

Natürlich kann man auch eigene Farbschemata definieren:

```
meine_palette <- c("red", "green", "blue", "yellow")

p2 + scale_fill_manual(values = meine_palette)
```

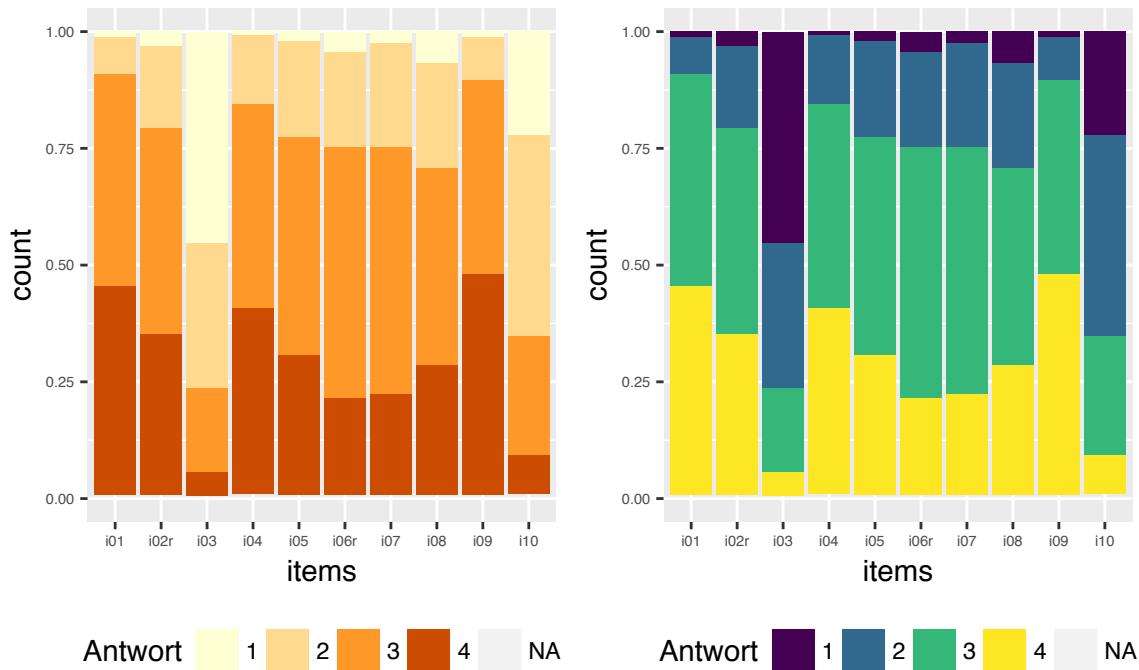


Abbildung 12.5: Alternative Farbschemata

Über Geschmack lässt sich da streiten... Oder auch nicht: Im Zweifelfall ist davon abzuraten, sich eine eigene Palette zusammenzustellen. Es ist gar nicht so leicht, eine gute Auswahl zu treffen (z.B. gute Kontraste zwischen allen Farben der Paletten) Jedenfalls lohnt sich ein Blick auf die Auswahl der 657 Farben in R mit `colors()`. Wer sich die Brewer-Farbpaleetten anschauen will, kann das mit `display.brewer.all()` aus dem Paket `rcolorbrewer` tun.

12.3.1 Diagramm beschriften

Überschrift, Unter-Überschrift oder ‘Figure Caption’ - all diese Bezeichnungen eines Diagramms kann man in `ggplot2` recht komfortabel ändern:

```
p2 + labs(title = "Häufigkeiten der Antworten",
          subtitle = "N = viel",
          caption = "Die Daten wurden 2016 erhoben")
```

Mit `theme` kann man alle “Nicht-Daten-Elemente” des Diagramms einstellen, aber nicht Text zuweisen z.B. Überschriften. Für Letzteres verwendet man `labs` oder `scale_XXX`. Mit `?theme` bekommt man weitere Informationen:

```
p2 + labs(title = "Häufigkeiten der Antworten",
          subtitle = "N = viel",
          caption = "Die Daten wurden 2016 erhoben") +
```

```
theme(axis.ticks = element_blank()) +
scale_fill_discrete(name = "Antwortoptionen")
```

Wo wir schon dabei sind, haben wir noch die Position und den Titel der Legende geändert. Wenn man das Seitenverhältnis (Y:X) des Diagramms einstellen möchte, geht das mit `aspect_ratio: p3 + theme(aspect.ratio = .61)`.

12.3.2 Balken mit Häufigkeitswerten

Persönlich denke ich, dass der generelle Eindruck, den das Diagramm gut transportiert, wichtiger ist als die Details, aber manchmal sind die exakten Zahlen von Interesse. Darüber hinaus würde Tufte vielleicht argumentieren, dass man diese Zahlen ruhig (eher klein) auf das Diagramm aufbringen darf. Schließlich soll, so Tufte, ein gutes Diagramm mehr Details offerieren, wenn man gleichsam näher an das Diagramm herangeht. Um die Häufigkeitswerte pro Kategorie zu bekommen, müssen wir sie selber berechnen:

```
extra_items_long_rev %>%
  filter(items == "i01") %>%
  count(Antwort) %>%
  mutate(n_prop = n / sum(n)) -> extra_items_count
```

Im Folgenden übergeben wir keine Rohwerte, sondern aggregierte Werte (Häufigkeiten). Damit ändert sich die Syntax. `ggplot` muss jetzt nicht mehr selber die relevanten Zeilen zählen, um die Höhe der Balken zu bestimmen. Vielmehr übergeben wir die richtige Zeilenanzahl an `ggplot`; diesem Wert soll die Höhe des Balkens jetzt zugordnet werden. Das kann man mit `geom_col` erreichen (oder mit `geom_bar(stat = "identity", ...)`). mit `vjust` kann man die vertikale Verankerung adjustieren. Außerdem verkleinern wir noch die Schriftgröße (s. Abbildung 12.6, links).

```
extra_items_count %>%
  ggplot +
  aes(x = Antwort, y = n) +
  geom_col(aes(fill = Antwort)) +
  geom_text(aes(label = n), vjust = 1.5, size = 6) +
  theme(legend.position = c(.8, .8),
        axis.text = element_text(size=rel(0.7))) -> p5
```

Bei der Gelegenheit haben wir die Legende in das eigentliche Diagramm gepackt; das Argument `legend.position` bekommt dabei einen Vektor mit zwei Elementen übergeben, die der X- und der Y-Koordinate entsprechen. `c(0, 0)` korrespondiert zur linken unteren Ecke des Diagramms und `c(1, 1)` zur rechten oberen. Das eigentliche Darstellen der numerischen

Werte geht mit `geom_text`, welches Daten einen Text (z.B. eine gedruckte Zahl im Diagramm) zuordnet.

Statt absolute Häufigkeiten wären Prozentzahlen genehm (s. Abbildung 12.6, Mitte):

```
extra_items_count %>%
  ggplot +
  aes(x = Antwort, y = n_prop) +
  geom_col(aes(fill = Antwort)) +
  geom_text(aes(label = percent(n_prop)), vjust = 1.5, size = 6) +
  theme(legend.position = c(.8, .8),
        axis.text = element_text(size=rel(0.7))) +
  scale_y_continuous(labels = percent) -> p6
```

Das Paket `scales` besorgt dabei die Umrechnung in die Prozentzahlen und die korrekte Darstellung. Das eigentliche Berechnen des Anteils `n_prop` haben wir oben erledigt.

12.3.3 Sortieren der Balken

Es ist sinnvoll, die Balken nach ihrer Höhe zu sortieren. Allerdings ist die Standardreihenfolge der Balken bei `ggplot` die, die von der Reihenfolge der Faktorstufen vorgegeben ist:

```
levels(extra_items_count$Antwort)
#> [1] "4" "3" "2" "1"
```

Möchten wir die Balken der Höhe nach sortieren, so bietet es sich an, die Faktorstufen neu zu ordnen: `reorder(zu_sortierender_Vektor, neue_Reihenfolge)`. Dabei wird der ersten Faktorstufen von `zu_sortierender_Vektor` der kleinste Wert von `neue_Reihenfolge` zugeordnet. Möchte man stattdessen der ersten Faktorstufen den größten Wert zuordnen, so hilft ein Minuszeichen vor `neue_Reihenfolge` (s. Abbildung 12.6, rechts):

```
extra_items_count %>%
  ggplot +
  aes(x = reorder(Antwort, -n_prop), y = n_prop) +
  geom_col(aes(fill = Antwort)) +
  geom_text(aes(label = percent(n_prop)), vjust = 1.5, size = 6) +
  theme(legend.position = c(.8, .8),
        axis.text = element_text(size=rel(0.7))) +
  scale_y_continuous(labels = percent) -> p7
```

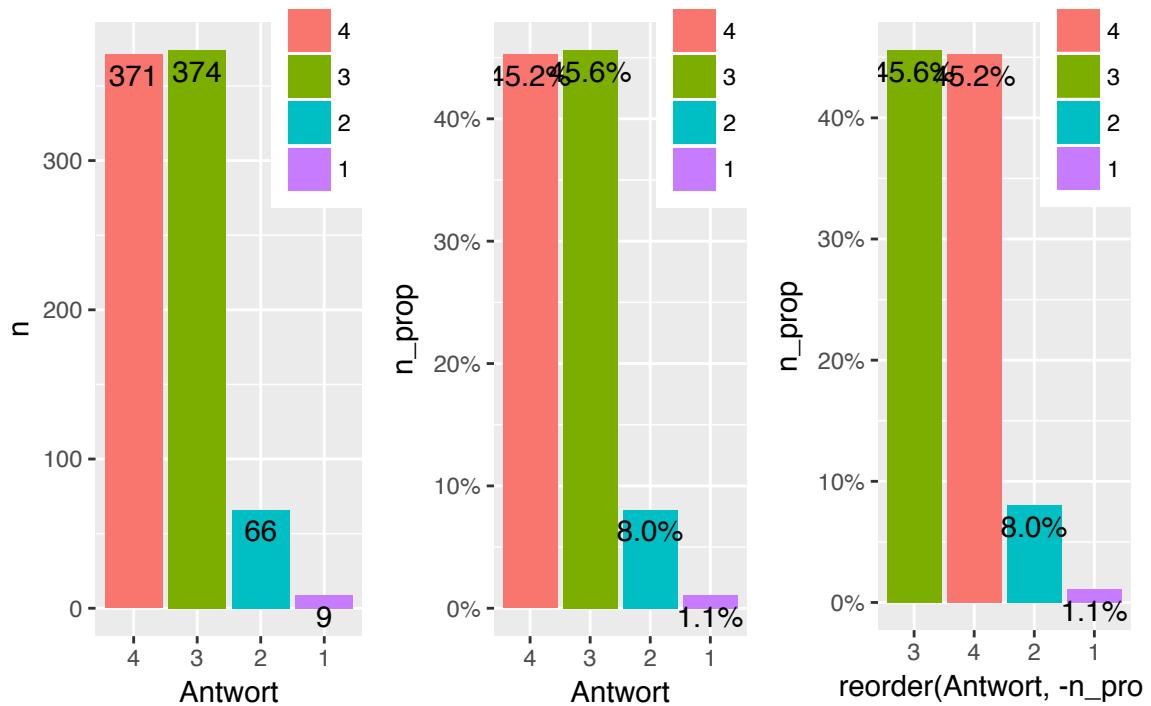


Abbildung 12.6: Balkendiagramme mit Zahlen und sortiert

12.4 Zum Weiterlesen

Ein Diagramm zu erzeugen, dass keine Wünsche offen lässt, ist (auch) mit `ggplot2` kein Selbstläufer³. Viele Details können (und sollten oft) angepasst werden. Die einfache Lehre daraus ist, dass man zwischen Diagrammen für “privaten” Gebrauch und “öffentlichen” Gebrauch unterscheiden muss. Man kann auch diskutieren, ob es die Zahlen (Anteile der Kategorien) im Diagramm wirklich braucht. Aber die Erfahrung zeigt, dass diese Zahlen verlangt werden.

³<https://xkcd.com/1319/>

Kapitel 13

Karten zeichnen

Karten zeichnen, um statistische Auswertungen mit geographischen Daten zu verknüpfen, eine schöne Sache, ist Gegenstand dieses Kapitels.



Lernziele:

- Grundlegende Aspekte der Visualisierung von Kartenmaterial kennen
- Kartenmaterial mit anderen Daten assoziieren
- Erste Erfahrung mit Listenspalten sammeln

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse)
library(viridis)
library(stringr)
library(pradadata)
library(gridExtra)
library(sf)
library(rworldmap)
```

13.1 Kartendaten

Wenig überraschend ist Kartenmaterial die Grundlage für dieses Thema. Genauer gesagt benötigen wir drei Arten von Informationen:

1. Zum einen die reinen geographischen Informationen (Geo-Daten¹), vor allem Grenzen von Verwaltungseinheiten wie Staaten oder Landkreisen aber auch andere Geo-Daten wie Flüsse oder Seen. Diese Daten sind als Vektorarten gespeichert (Linien, Polygone)

¹man spricht auch von Geo-Info-Systemen, GIS

2. Zum anderen semantische Informationen, die mit rein geographischen Informationen verknüpft sind, also z.B. die Hauptstadt eines Landes.
 3. Dazu kommt dann noch drittens Strukturdaten und sonstige statistische Informationen, die wiederum mit den semantischen Informationen verknüpft sind.

13.1.1 Geo-Daten der deutschen Verwaltungsgebiete

Die Geo-Daten werden wir im *Shape-Format* verarbeiten - eines der verbreitesten Formate für Geo-Daten. Das Shape-Format besteht aus zumindest drei Dateien (.shp, .dbf, .shx); achten Sie darauf, dass diese drei Arten von Dateien verfügbar sind (und im gleichen Ordner). Eine Shape-Datei für Deutschland kann vom Bundesamt für Kartografie und Geodäsie (BKG) heruntergeladen werden: <http://www.bkg.bund.de>; dort sind auch die Nutzungsbedingungen² geregelt. Das Paket `pradadata` stellt diese Daten zur Verfügung:

Die Daten sind komplex; ein Blick in die Dokumentation (im Download enthalten) ist sinnvoll (s. Ordner *Dokumentation* bei den herunter geladenen Daten). In der “Linien-Datei” steht die Spalte AGZ für *Art der Grenze*, wobei 1 für die Staatsgrenze steht (vgl. vg250.pdf). RDG steht für *rechtliche Definition der Grenze* mit 1 = *festgelegt*. GM5 steht für die Grenzmerkmale von Verwaltungsgemeinschaftsgrenzen (für uns nicht weiter von Belang). Diese Variablen sind relevant für die Datei, die die *Linien* (L) kodiert; für Punkte (P) oder Flächen (F) gibt es weitere Variablen, man konsultiere die Dokumentation dafür. In der “Flächen-Datei” steht GF für *Geofaktor*, wobei 1 und 2 Gewässer markieren. BSG steht für *besondere Gebiete*; für uns ohne Belang. Interessanter ist RS, der *Regionalschlüssel*, der die einzelnen Ebenen der Verwaltungseinheiten kodiert.

Hat man Shape-Daten als eigene Dateien vorliegen, so ist die Funktion `st_read` komfortabel; sie lädt sowohl Geo-Daten als auch sonstige Daten in einen Dataframe. Die ganze Geo-Info findet sich *einer* Spalte (`geometry`) des Dataframes wieder. Interessant ist, dass in der Spalte `geometry` (pro Zeile) mehrere Informationen gespeichert sind. Man spricht von einer *Listenspalte*. Praktischerweise kümmert sich `st_read` und seine Freunde um diesen speziellen Datentyp. Das Argument `quiet = TRUE` unterdrückt die Ausgabe einer Infos, die die Funktion sonst redseligerweise von sich geben würde.

²<http://www.geodatenzentrum.de/auftrag/pdf/geonutz.pdf>; (GeoNutzV); die Daten wurden am 9. Okt. 2017 heruntergeladen

Wohlan! Eine Karte von Deutschland, genauer gesagt, aller Verwaltungseinheiten ist leicht zu erhalten:

```
ggplot(data = de_L) +
  geom_sf(size = .1, color = "lightgrey") -> p_de1
```

Allein, es dauert lang, denn es gibt doch einiges zu sehen (bzw. einige Grenzen zu ziehen) in Deutschland. Begrenzen wir uns auf die Bundesländer. Es gibt mehere Wege dorthin, z.B. ist der Flächen-Datei die Staatsgrenze mit AGZ = 1 definiert, s. Abbildung ??:

```
de_L %>%
  filter(AGZ == 1) %>%
  ggplot +
  geom_sf() -> p_de2
```

13.1.2 Daten der Wahlkreise

Alternativ bietet der Bundeswahlleiter eine Karte mit den Wahlkreisen an³. Das ist ganz praktisch, weil dort auch sozioökonomische Daten zur Verfügung stehen, die den Wahlkreisen zugeordnet sind.

Die Wahlkreise-Geo-Daten für die Bundestagswahl 2017 sind im Paket `pradadata` enthalten:

```
data("wahlkreise_shp")
glimpse(wahlkreise_shp)
#> Observations: 299
#> Variables: 5
#> $ WKR_NR    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
#> $ LAND_NR   <fctr> 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 01, 13, 13, ...
#> $ LAND_NAME  <fctr> Schleswig-Holstein, Schleswig-Holstein, Schleswig-H...
#> $ WKR_NAME   <fctr> Flensburg - Schleswig, Nordfriesland - Dithmarschen...
#> $ geometry   <simple_feature> MULTIPOLYGON (((543474.9057..., MULTIPOLY...
```

In diesem Dataframe gibt es vier Spalten mit Daten zu den Verwaltungsgebieten: `WKR_NR`, die Ordnungsnummer des Wahlkreises; `LAND_NR`, die Ordnungsnummer des Bundeslandes; `LAND_NAME`, der Name des Bundeslandes; `WKR_NAME`, der Name des Wahlkreises (vgl. `wahlkreise_shp`). In der Spalte `geometry` sind wiederum die Geo-Daten gespeichert; auch hier handelt es sich um eine Listenspalte, d.h. eine Spalte, die aus mehreren “Spalten” besteht.

Laden wir dazu noch die vom Bundeswahlleiter bereitgestellten sozioökonomischen Daten, und bringen beides in eine Karte:

³(c) Der Bundeswahlleiter, Wiesbaden 2017 <https://www.bundeswahlleiter.de/info/impressum.html>

```
data(socec)
#glimpse(socec)
#data(socec_dict)
#socec_dict
```

In `socec` sind die Spaltennamen nur mit einer laufenden Nummer kodiert; ihr Inhalt kann über `?socec` oder über den Dataframe `socec_dict` ausgelesen werden.

13.2 Fallbeispiel: Unterschiede in den Wahlkreisen visualisieren

13.2.1 Karte der Wahlkreise gefärbt nach Arbeitslosigkeit

Färben wir nun die Wahlkreise nach ihrer Arbeitslosigkeit ein; die Arbeitslosenquote findet sich als V47 (Stand März 2017). Vorher müssen wir noch die Geo-Daten mit den Strukturdaten vereinen. Dafür benötigen wir aus beiden Dataframes eine ID-Spalte, die die Wahlkreisnamen (oder -nummern) kodiert. Mit `left_join` führen wir dann die Vereinigung durch:

```
socec_short <- socec %>%
  select(WKR_NAME = V3,
         WKR_NR = V2,
         wegzug = V11, # Wegzugssalde
         migration = V19, # Migrationsanteil
         auslaender = V8, # Ausländeranteil
         alq = V47, # Arbeitslosenquote
         migration = V19) # Anteil mit Migrationshintergrund

socec_short %>%
  left_join(wahlkreise_shp) %>%
  ggplot +
  aes(fill = alq) +
  geom_sf() -> p1

socec %>%
  select(WKR_NAME = V3,
         alq = V47) %>% # Arbeitslosenquote
  left_join(wahlkreise_shp) %>%
  ggplot +
  aes(fill = alq) +
  scale_fill_viridis()
```

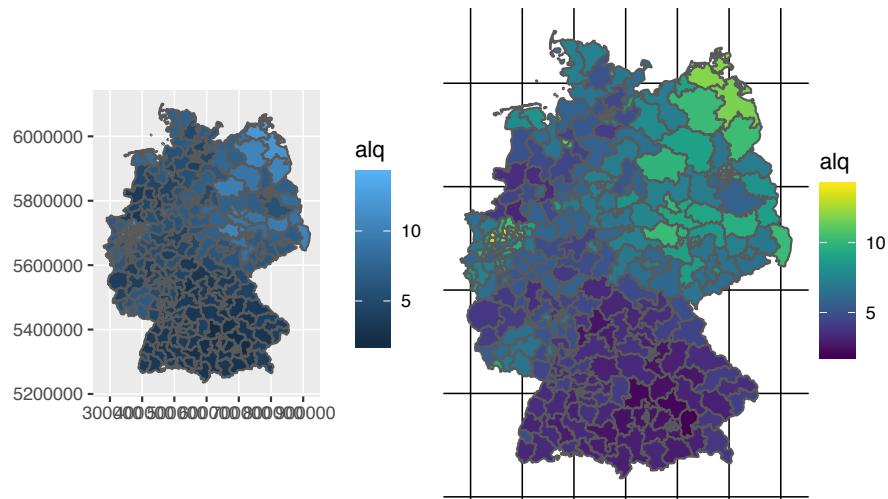


Abbildung 13.1: Arbeitslosigkeit nach Wahlkreis

```
theme_void() +
geom_sf() -> p2

grid.arrange(p1, p2, nrow = 1)
```

Wir haben nur Geo-Daten für 299 Wahlkreise, aber 316 Sätze mit Strukturdaten; auf diese Inkonsistenz macht uns die Funktion aufmerksam (hier nicht wiedergegeben). Außerdem ist die ID-Spalte einmal als Faktor, einmal als Textvariable definiert. Beide Warnungen schlagen wir für einen Moment in den Wind und betrachten lieber das Ergebnis in Abbildung 13.1. Das Farbschema im rechten Teilbild greift die Unterschiede “dramatischer” ab als das linke Farbschema. Außerdem haben wir im rechten Teilbild noch das Thema “void” gewählt, welches transparenten Hintergrund aufweist und auf Axen verzichtet. Ähnliche Analysen lassen sich analog für die anderen sozioökonomischen Indikatoren durchführen, die sich in `socec` finden.

13.2.2 Wahlergebnisse nach Wahlkreisen

Nach gleichem Prinzip können wir die Wahlergebnisse nach Wahlkreisen visualisieren; nehmen wir den Anteil gültiger Zweitstimmen für die AfD. Diese Daten finden sich beim Bundeswahlleiter; das Paket `prada` stellt diese Daten in etwas aufgeräumter Form zur Verfügung.

```
data("elec_results")
```

Pro Partei finden sich 4 Spalten:

1. Anteil gültiger Erststimmen dieser Wahl
2. Anteil gültiger Erststimmen in der letzten Wahl

3. Anteil gültiger Zweitstimmer dieser Wahl
4. Anteil gültiger Zweitstimmen in der letzten Wahl

Zurück zur Frage, warum es mehr Wahlergebnisse gibt als Wahlkreise. Ein Blick in `wahlkreise_shp` zeigt, dass nicht nur die Wahlkreise, sondern auch zusätzlich die Bundesländer aufgeführt sind. Deren übergeordneter Wahlkreis (`parent`) ist die Bundesrepublik, mit 99 kodiert. Probieren Sie mal aus:

```
elec_results %>%
  select(parent_district_nr, district_name, district_nr) %>%
  filter(parent_district_nr == 99)
```

```
elec_results %>%
  filter(parent_district_nr != 99) -> elec_results
```

```
elec_results %>%
  select(WKR_NR = district_nr, AfD_3, votes_3) %>%
  mutate(afd_prop = AfD_3 / votes_3) %>%
  left_join(wahlkreise_shp) %>%
  ggplot +
  aes(fill = afd_prop) +
  geom_sf() +
  scale_fill_viridis() +
  theme_void() +
  labs(caption = "Anteil gültiger Zweitstimmen für die AfD") -> p3
```

Das Farbschema stellt schön die Unterschiede zwischen den Anteilen heraus (vgl. Abbildung 13.2, linke Seite). Interessanterweise ist ein ziemlich klares Muster erkennbar; Ostdeutschland ist deutlich “heller”, also weist höhere AfD-Quoten auf. Der Westen und der Norden zeigen die geringsten AfD-Ergebnisse.

13.2.3 Zusammenhang von Arbeitslosigkeit und AfD-Wahlergebnis

Man könnte die Hypothese vertreten, dass das AfD-Wahlergebnis mit der Arbeitslosigkeit ansteige. Überprüfen wir diese Hypothese und stellen das Ergebnis wiederum auf der Karte dar.

Dazu sagen wir die AfD-Quote anhand der Arbeitslosigkeit vorher mit einem linearen Modell (einfache Regression). Die Wahlkreise färben wir ein, je nach dem wie gut sie zur Vorhersage passen (also nach der Größe des Vorhersagefehlers). Mehr zum linearen Modell findet sich im Kapitel XXX. Zuerst bereiten wir uns einen passenden Datensatz (`afd_df`), dann plotten wir das Ergebnis (vgl. Abbildung 13.2, rechte Seite).

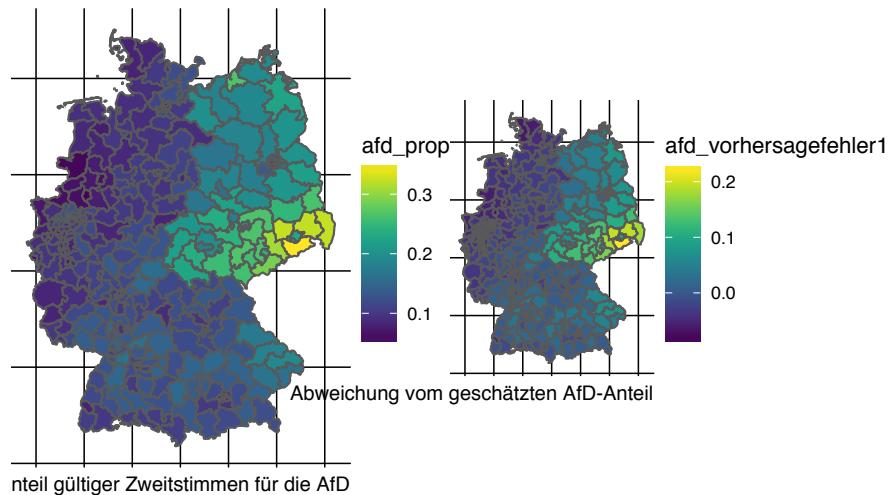


Abbildung 13.2: AfD-Wahlergebnisse nach Wahlkreisen

```
elec_results %>%
  select(WKR_NR = district_nr, AfD_3, votes_3) %>%
  mutate(afd_prop = AfD_3 / votes_3) %>%
  inner_join(wahlkreise_shp) %>%
  inner_join(socec_short) %>%
  mutate(afd_vorhersagefehler1 = lm(afd_prop ~ alq, data = .)$residuals) -> afd_df
```

Der AfD-Vorhersagefehler zeigt also, um wieviel Prozentpunkte das AfD-Wahlergebnis eines Wahlkreises von der Prognose anhand der lokalen Arbeitslosigkeit abweicht. Die Funktion `lm()` liefert als Ergebnis eine Liste mit mehreren Elementen zurück; einer davon heißt `residuals`. Mit diesem Wert füllen wir die Wahlkreise nach bekannter Manier:

```
afdf %>%
  ggplot +
  aes(fill = afd_vorhersagefehler1) +
  geom_sf() +
  labs(caption = "Abweichung vom geschätzten AfD-Anteil") +
  theme_void() -> p4

grid.arrange(p3, p4 + scale_fill_viridis(), nrow = 1)
```

Auch hier wieder zeigt sich ein klares Muster: Der Osten ist farblich erkennbar abgesetzt vom Rest der Republik. Hier ist der AfD-Anteil höher als es die Arbeitslosigkeit erwarten ließe. Umgekehrtes gilt für den westlichen Rand der Republik; dort ist der AfD-Anteil geringer als eine Vorhersage durch die Arbeitslosigkeit erwarten lassen würde. Die Analyse zeigt, dass mittels Visualisierung einige Überraschungen und erhellende Momente möglich sind.

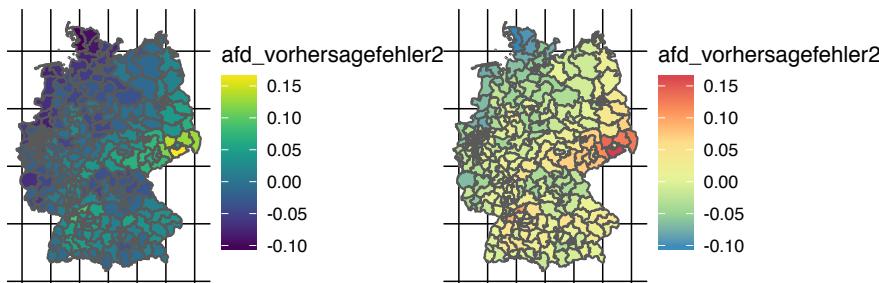


Abbildung 13.3: Ein komplexeres Erklärungsmodell zum AfD-Erfolg

13.2.4 Ein komplexeres Modell

Gerade eben habe wir versucht, den AfD-Erfolg anhand einer Variablen (der Arbeitslosigkeit) zu verstehen. Sicher wird unser Verständnis besser, wenn wir mehrere Variablen zur Vorhersage verwenden? Natürlich basiert die Auswahl an Variablen, die wir gleich treffen werden, auf ausdifferenzierten und bewährten Theorien... Nur ist der Seitenrand hier zu schmal, um diese aufzuführen... Sei's drum. Nehmen wir folgende Prädiktoren, um den AfD-Anteil vorherzusagen: Anteil der Menschen mit Migrationshintergrund (V19), Wegzugs-Saldo (V11), Arbeitslosigkeit (V47) und Ausländeranteil (V8).

```
afdf %>%
  mutate(afd_vorhersagefehler2 = lm(afd_prop ~ alq + wegzug + migration + auslaender, da
```

Hier haben wir `inner_join` mitgeteilt, welche Spalte als Vereinigungsschlüssel dienen soll (`by`). Ansonsten sind wir genau so vorgegangen wie beim vorherigen Modell. Das Ergebnis ist in Abbildung 13.3 zu sehen. Auf dem ersten Blick sieht das Ergebnis ähnlich aus wie beim vorherigen Modell. Allerdings sollten wir uns überlegen, ob das Farbschema für Vorhersagefehler gut passt. Vorhersagefehler können entweder etwa null sein - oder positiv oder negativ. Wir brauchen also eine Palette, die einen farblichen Mittelpunkt hat und Farben die Abweichungen nach unten bzw. nach oben markieren. Man spricht hier von einer *divergierenden* Farbpalette.

```
afdf %>%
  ggplot +
  aes(fill = afd_vorhersagefehler2) +
  geom_sf() +
  theme_void() -> p5

grid.arrange(p5 + scale_fill_viridis(),
             p5 + scale_fill_distiller(palette = "Spectral"),
             nrow = 1)
```

Übrigens: `distiller` erweitert ein Farbspektrum mit einer begrenzten Anzahl von Farben, so dass es für kontinuierliche Variablen verwenbar ist.

13.3 Weltkarten

13.3.1 rworldmap

Das R-Paket `rworldmap` bietet Karten und Strukturdaten für 244 Länder der Erde; der Datensatz `countriesLow` beinhaltet diese Informationen. Das Paket hat eine eigene Plot-Funktion, mit `plot(countriesLow)` bekommt man schon eine Weltkarte. Daneben gibt es eine Funktion, mit der man eigene Daten zu den Ländern des Datensatzes hinzufügen möchte, z.B. die Information `will_ich_hin`. Ländernamen kann man z.B. nach der Isonorm ISO 3166-1 alpha-3⁴ eingeben. Sagen wir, unser Vektor `will_ich_hin` sei bestückt mit folgenden Ländern:

```
will_ich <- data_frame(
  hin = c("TTO", "COK", "MNG", "CAN"),
  ja_wirklich = c("will ich hin", "will ich hin", "will ich hin", "will ich hin")
)
```

Diesen Datensatz mappen wir jetzt zu `countriesLow`:

```
will_ich_hin_map <- joinCountryData2Map(will_ich, joinCode = "ISO3", nameJoinColumn = "name")
#> 4 codes from your data successfully matched countries in the map
#> 0 codes from your data failed to match with a country code in the map
#>      failedCodes failedCountries
#> 239 codes from the map weren't represented in your data
mapCountryData(will_ich_hin_map, nameColumnToPlot="ja_wirklich", catMethod = "categorical")
```

ja_wirklich



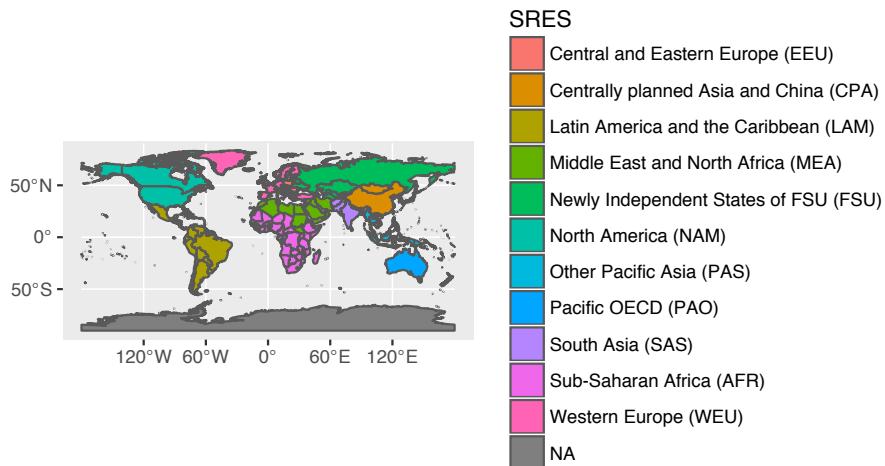
⁴https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3

13.3.2 rworldmap mit geom_sf

Das Paket `sf` und `geom_sf` lassen jedoch ein komfortableres Arbeiten zu; außerdem ist die Darstellung ästhetischer. Was ist der Vorteil, wenn wir das Kartenobjekt in ein Objekt vom Typ `sf` umwandeln? Das Objekt `countriesLow` ist ein kompliziertes, mehrfach verschachteltes Objekt⁵. Wandeln wir in ein `sf`-Objekt um, so können wir mit dem gewohnten Schritten arbeiten:

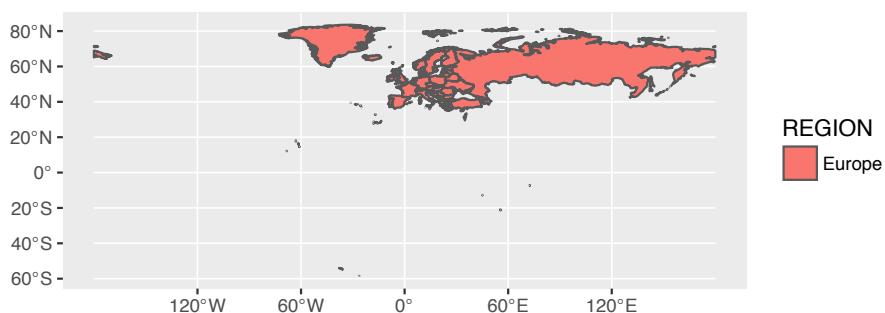
```
world_sf <- st_as_sf(countriesLow)

ggplot() +
  geom_sf(data = world_sf) +
  aes(fill = SRES)
```



Z.B. können wir einfach nach Europa filtern:

```
world_sf %>%
  filter(REGION == "Europe") %>%
  ggplot() +
  aes(fill = REGION) +
  geom_sf()
```



⁵des Typ S4

Interessieren uns die EU-Länder, so trage man zuerst eine Liste dieser Länder aus dem Gedächtnis zusammen... oder man lese nach. Hier eine Liste der EU-Länder im ISO 3166 alpha-2:

```
eu <- c("BE", "BG", "CZ", "DK", "DE", "EE", "IE", "GR", "ES", "FR", "HR", "IT", "CY", "L
```

Prüfen wir, ob diese Länder im Datensatz `countries` enthalten sind:

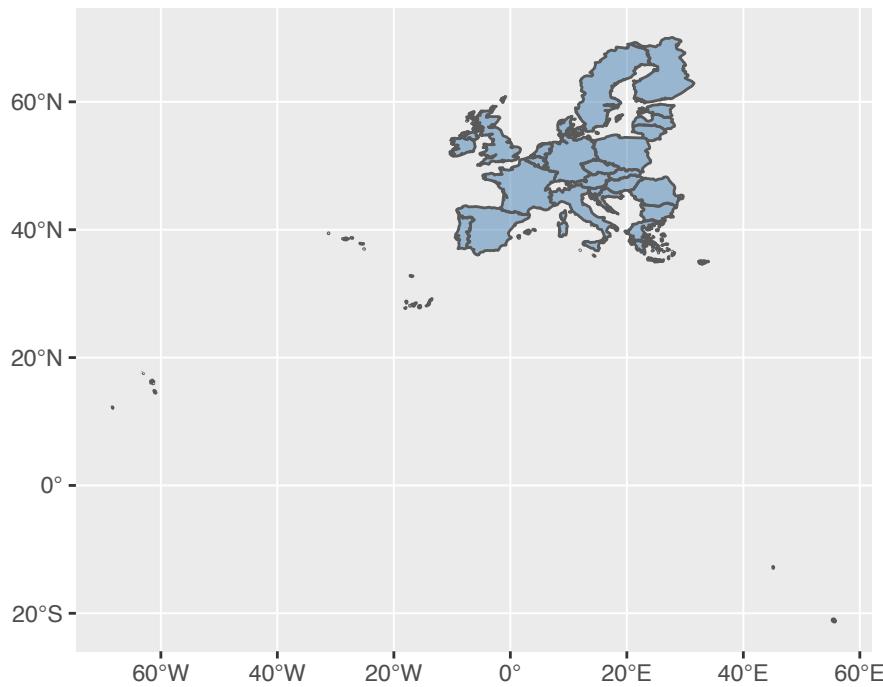
```
data(countries)
countries %>%
  filter(cca2 %in% eu) %>% nrow
#> [1] 28
```

Das passt; jeweils 28 EU-Länder wurden identifiziert. Ergänzen wir zu `countries` noch eine Spalte `is_EU`, für etwaige spätere Verwendung:

```
countries %>%
  mutate(is_EU = ifelse(cca2 %in% eu, TRUE, FALSE)) -> countries
```

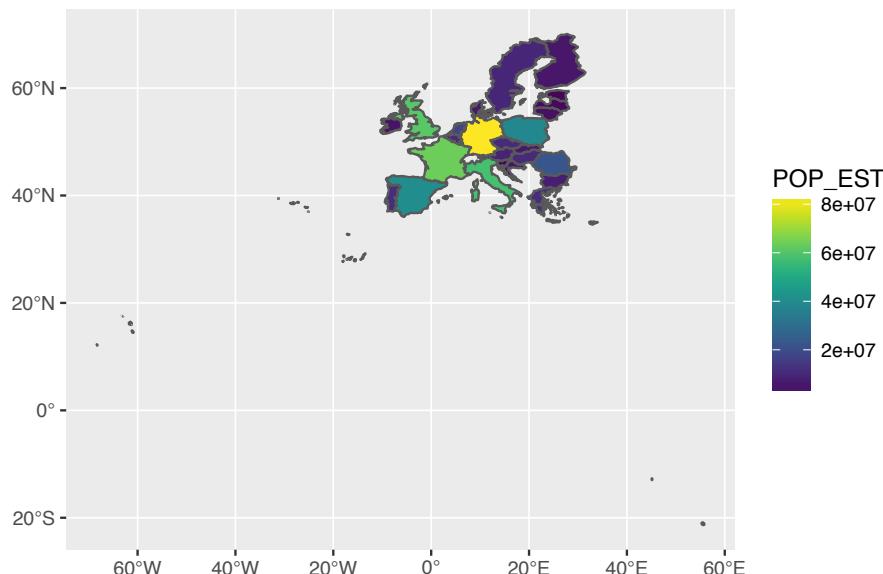
Dann filtern wir `world_sf` so, dass nur die Namen aus der EU übrig bleiben:

```
world_sf %>%
  filter(ISO_A2 %in% eu) %>%
  ggplot() +
  aes() +
  geom_sf(fill = "steelblue", alpha = .5)
```



Natürlich können wir den Farben der Länder eine beliebige Variable zuordnen; `aes()` sorgt dann dafür, dass die Werte dieser Variablen jeweils einer Farbe zugeordnet (gemapt) werden.

```
world_sf %>%
  filter(ISO_A2 %in% eu) %>%
  ggplot() +
  aes() +
  geom_sf(aes(fill = POP_EST)) +
  scale_fill_viridis()
```



Das Farbspektrum arbeitet die Bevölkerungszahlen schön heraus; Deutschland sticht mit

einer großen Bevölkerung hervor.

13.4 Fallbeispiel: Konkordanz von Kulturwerten und Wohlbefinden

13.4.1 Standardisierung der Maße

Betrachten wir ein weiteres Fallbeispiel, basierend auf Daten zu den *Cultural Values* nach Schwartz (Schwartz 1999) und den *Wellbeing Indicators* der OECD (Durand 2015). Untersuchen wir die Hypothese, ob “freies Denken” - intellektuelle Autonomie in Schwartzens Begriffen - mit Lebenszufriedenheit einher geht. Anders gesagt: Haben Länder mit hohen Werten in intellektueller Autonomie tendenziell höhere Werte in der Lebenszufriedenheit? Die Frage ist sicherlich interessant; allerdings darf dabei nicht vergessen werden, dass wir hier ohne tieferes theoretisches Fundament an die Frage herangehen. In der Art, wie wir die Frage stellen, ist sie sicherlich hochgradig naiv; außerdem haben wir uns nicht die Mühe gemacht, den Stand der Forschung zu verstehen. All das sind Punkte, die es in erstgemeinter Datenanalyse als Teil ernstgemeinter Forschung zu beachten gilt. Sei’s drum - mit welchem Maß können wir fassen, ob ein bestimmtes Land (Absurdistan) ähnlich “gut” in beiden Aspekten abschneidet?

Eine Möglichkeit wäre, für beide Maße eine *Rangliste* zu erstellen; dann könnte man für Absurdistan den Rangplatz in intellektueler Autonomie mit dem Rangplatz für Lebenszufriedenheit vergleichen. Eine einfache Differenz würde vielleicht reichen. Besser noch erstellt man eine *relative Rangliste* von Listenplatz 0% (der höchste Wert) bis zum Listenplatz 100% (der geringste) Wert, um mögliche Unterschiede in der Länge der beiden Listen auszugleichen. Man könnte argumentieren, dass es für unsere Fragestellung nur darauf ankäme, wie ähnlich die beiden Rangplätze, wie gering also die Differenz ist. Dann würde das Vorzeichen keine Rolle spielen; nur der Absolutbetrag der Differenz spielte eine Rolle. Dieser Ansatz impliziert, dass wir an den Unterschieden zwischen den Rangplätzen nicht interessiert sind. Anders gesagt nehmen wir an, dass die Unterschiede zwischen den Rangplätzen gleich groß ist; wir messen auf ordinalem Niveau, wie man sagt.

Eine relative Rangliste kann man so vereinbaren:

$$R = \frac{r_i - 1}{n - 1}$$

wobei r_i die Position einer Beobachtung ist, wenn die Beobachtungen (aufsteigend) sortiert sind; n ist die Anzahl der Beobachtungen. Dabei stellt sich noch die Frage, wie man gleichen Werten (ties) umgeht (s. (Agresti 2013) für Details). Eine Umsetzung in R für die relative Rangliste bietet `dplyr::percent_rank()`:

```
x <- c(1, 2, 3)
percent_rank(x)
#> [1] 0.0 0.5 1.0
```

Ein anderer Ansatz wäre, den Wertebereich (Range) beider Maße auf 1 zu standardisieren; so dass der kleinste Wert 0 und der größte Wert 1 ist (*Min-Max-Standardisierung* oder 0-1-Standardisierung). Dann könnte man, ohne den Umweg über Rangplätze zu gehen, direkt (Absolutbeträge von) Differenzen berechnen. Hätten beide Maße die gleiche Skalierung, so könnte man sich diese Standardisierung sparen. Dieser Ansatz setzt voraus, dass die Grenzen der Skalierung (0 und 1) sinnvolle, nicht-willkürliche Werte darstellen, an denen man sich orientieren kann.

Möchte man X min-max-Normalisieren, so kann man dies so tun:

```
S <- (X - min(X)) / (max(X) - min(X))
```

Als Formel geschrieben:

$$s_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

wobei $X = (x_1, x_2, \dots, x_n)$ der ursprüngliche Vektor und s_i der normalisierte Wert der Beobachtung i ist.

Ein dritter Weg wäre die *z-Skalierung* beider Maße; dabei wird der relative Abstand eines Landes vom Mittelwert aller Länder betrachtet. Diese Skalierung nimmt die Skalengrenzen nicht als Orientierungspunkt, sondern den Mittelwert der Länder. Dieser Überlegung liegt zugrunde, dass es bei vielen Maßen der Sozialwissenschaften keinen fixen Nullpunkt gibt.

z-Werte sind so definiert:

$$z_i = \frac{x_i - \bar{x}}{sd(x)}$$

mit x_i der Wert des zu standardisierenden Maßes der Beobachtung i ; sd ist die Standardabweichung.

In R kann man sich z-Werte so ausgeben lassen:

```
x <- c(1, 2, 3)

scale(x) %>% as.numeric
```

Welche Standardisierung gewählt wird, hängt also von theoretisch-methodischen Annahmen ab. Schließlich wäre noch die Frage zu erörtern, wie es um die Messgüte der Maße bestellt ist.

Gehen wir von vielen Unschärfen aus, denken pessimistisch, dass es mehr Rauschen als Signal in den Daten gebe, dann sind wir besser beraten, auf Scheingenaugkeit zu verzichten, und metrisches Niveau aufzugeben. Dann sollte man sich für Analyse der Rangplätze entscheiden. Ist man optimistischer und sieht den Messfehler in strenge Ketten gelegt, so darf man sich Aussagen mit höheren Aussagekraft genehmigen.

Führt man sich vor Augen, dass die Daten aus unterschiedlichen Jahren stammen, wohl von vielen verschiedenen Versuchsleitern erhoben, über Sprach- und Kulturgrenzen hinweg gemessen, mit Instrumenten unbekannter Beschaffeten, und weitere mögliche, hier nicht bekannte Schwächen aufweisen, sollten wir uns mit geringerer Aussagekraft begnügen oder besser gesagt, keine Scheingenaugkeit produzieren. Es sollte die Maxime gelten, lieber nichts zu sagen, als etwas Falsches. Wir unterwerfen uns also dem Regime der Rangplatzanalyse; ein Gebiet von respektablen wissenschaftlichem Renommee (Agresti 2013).

Laden wir also die Daten (`cult_values` für die Kulturwerte und `well_being` für die OECD-Daten des Wohlbefindens) und standardisieren zu relativen Rangplätzen. Die intellektuelle Autonomität findet sich in der Spalte `cult_values$intel_auton` und die Lebenszufriedenheit in `$wellbeing$Life_satisfaction` (der Datensatz stellt zwei ähnliche Maße bereit; bleiben wir beim ersten).

```
data(cult_values)
data(wellbeing)
```

```
names(wellbeing)
names(cult_values)
```

```
cult_values %>%
  mutate(intel_auton_rank = percent_rank(intel_auton)) -> cult_values

wellbeing %>%
  mutate(Life_satisfaction_rank = percent_rank(Life_satisfaction),
        Country = tolower(Country)) -> wellbeing
```

Nun müssen wir die beiden Datensätze zusammenführen, um die Daten aufeinander zu beziehen (so dass in der Zeile von Absurdistan beide Werte, intellektuelle Autonomie und Lebenszufriedenheit, stehen). Dabei ist zu beachten, dass die Ländernamen einmal mit großem Anfangsbuchstaben und einmal mit kleinem Anfangsbuchstaben geschrieben waren; das haben wir mit `tolower()` gerade angeglichen. Außerdem ist die USA einmal als `unitedstates` und einmal als `United States` bezeichnet; für `unitedkingdom` gilt ähnliches. Das sollten wir noch angleichen⁶.

⁶das ist mir erst aufgefallen, nach dem es mir auf die Füße gefallen war

```
cult_values %>%
  mutate(country_long = recode(country_long,
                               "unitedstates" = "united states", # erst alter Wert,
                               "unitedkingdom" = "united kingdom")) -> cult_values

c("united states", "united kingdom") %in% cult_values$country_long
#> [1] TRUE TRUE
```

`x %in% y` prüft, ob `x` in `y` enthalten ist; es empfiehlt sich, stets zu prüfen, ob das, was man denkt, dass es funktionioert, auch tatsächlich funktioniert... Mit `inner_join` werden nur Zeilen behalten, die sich in beiden Dataframes wiederfinden:

```
wellbeing %>%
  filter(region_type == "country_whole") %>%
  inner_join(cult_values, by = c("Country" = "country_long")) -> df_joined
```

Das Zusammenführen von Datensätzen unterschiedlicher Herkunft zeigt auf, wie viel Wissen so erzeugt bzw. abgeschöpft werden kann. Hätten wir Daten auf Personenebene, so würde die Gefahr von De-Anonymisierung beträchtlich steigen; hier ist das kein Problem. Wir können uns erfreuen am Reichtum möglicher Forschungsfragen, die die Daten zulassen. Betrachten wir die Konkordanz unserer beider Maße:

```
df_joined %>%
  select(Country, Life_satisfaction_rank, intel_auton_rank) %>%
  gather(key = indicator, value = perc_rank, -Country) %>%
  ggplot +
  aes(x = indicator, y = perc_rank) +
  geom_point(size = 5, alpha = .5) +
  scale_x_discrete(labels = c("Autonomie", "Zufried")) +
  geom_line(aes(group = Country)) -> p_rank_comp
```

Wie Abbildung XXX zeigt, scheint die Konkordanz beider Maße nicht sonderlich ausgeprägt zu sein. Wäre sie stark ausgeprägt, so wären die Linien parallel zur X-Achse; je extremer die Steigung der Linien, desto *geringer* die Konkordanz. Überprüfen wir diesen optischen Eindruck an der Rang-Korrelation:

```
df_joined %>%
  select(Life_satisfaction_rank, intel_auton_rank) %>%
  cor(method = "spearman")
#>           Life_satisfaction_rank    intel_auton_rank
#> Life_satisfaction_rank                1.000               0.313
```

#> intel_auton_rank

0.313

1.000

Spearmans Rangkorrelation zeigt einen Wert von .41 an; gar nicht so wenig.

Berechnen wir als nächstes für jedes Land die absolute Differenz der beiden Indikatoren und färben eine Karte entsprechend ein.

```
df_joined %>%
  mutate(konkordanz = abs(Life_satisfaction_rank - intel_auton_rank)) %>%
  select(Country, konkordanz, Life_satisfaction_rank, intel_auton_rank) -> df_joined_small
```

Falls Sie die Kartendaten noch nicht geladen haben sollten, tun Sie es bitte jetzt (vgl. Abschnitt ??).

```
data(countriesLow)
world_sf <- st_as_sf(countriesLow)
```

Dann “joinen” wir die Kartendaten zu unseren Umfragedaten und erstellen eine *Choroplethen-karte*, eine Karte also, die Flächen Werte zuweist, welche durch Farbwerte o.ä. dargestellt sind:

```
world_sf %>%
  mutate(NAME = tolower(NAME)) %>%
  inner_join(df_joined_small, by = c("NAME" = "Country")) -> world_sf_joined
```

```
world_sf_joined %>%
  ggplot +
  aes(fill = konkordanz) +
  geom_sf() +
  scale_fill_viridis() +
  theme(text = element_text(size = 6, family = "Arial"),
        legend.position = "bottom") -> p_concordance

grid.arrange(p_rank_comp, p_concordance,
             layout_matrix = rbind(c(1, 2, 2), c(1, 2, 2)))
```

Wie man sieht, scheint diese Konkordanz für Zentraleuropa und USA zu gelten, aber weniger oder kaum für andere OECD-Länder⁷.

⁷ohne `text = element_text(size = 6, family = "Arial")` kann es aufgrund eines Bugs in RStudio zu einer Fehlermeldung kommen; hier gilt der Grundatz: Googeln kann helfen

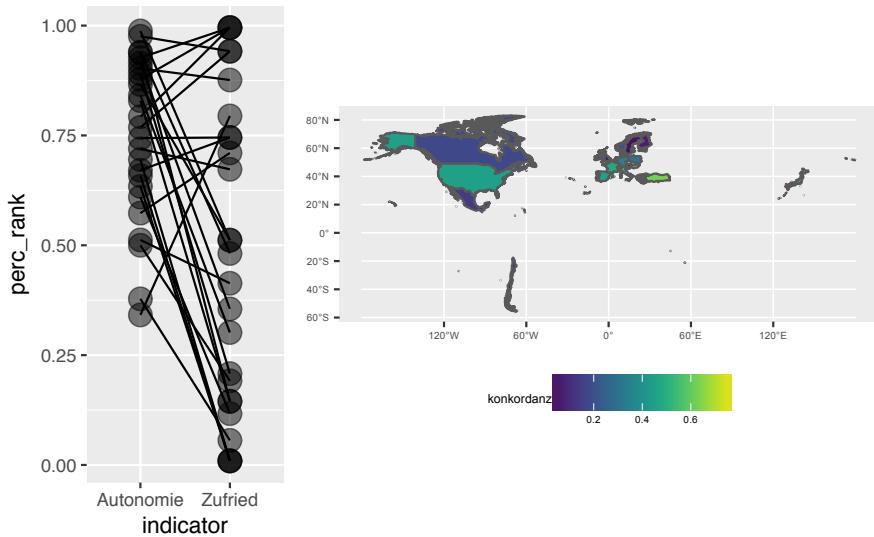


Abbildung 13.4: Konkordanz von Intellektueller Autonomie und Lebenszufriedenheit

13.5 Aufgaben



1. Probieren Sie verschiedenen Farbskalen durch, z.B. RdBu.
2. Welches Farbschema passt für welche Situationen?
3. Welche weiteren Farbschemata sind noch implementiert? Recherchieren Sie im Paket **RColorBrewer**.
4. Bestimmen Sie R^2 und den RMSE für das einfache und das komplexere Erklärungsmodell zum AfD-Erfolg.
5. Welche weiteren Prädiktoren könnten für ein Erklärungsmodell genutzt werden?
6. Erstellen Sie eine Karte Europas nur mit Ländern nördlich des 20. Breitengrades.

13.6 Weiterführende Literatur

Im Buch von Rahlf (2014) gibt es eine Reihe von weiteren Ansätzen zur Geo-Visualisierung von Daten - überhaupt von vielen Möglichkeiten der Datenvizualisierung in R. Rahlf baut nicht auf ggplot auf, sondern nutzt die herkömmlichen Grafik-Funktionen von R, die einen sehr sauberen Eindruck machen. (Baumer, Kaplan, und Horton 2017) widmen ein Kapitel dem Thema Geo-Mapping; man findet dort auch eine Einführung in die Geodäsie; wer immer schon mal wissen wollte, was eine Mercator-Projektion ist, und was ihre Schwächen sind, der wird hier eine Idee bekommen. Außerdem führen die Autoren in interaktive Kartendienste ein, wie etwa von Google. Neben der Funktion `geom_sf` gibt es weitere R-Pakete, die die Darstellung von Geo-Daten dienen. Mit dem Paket `rgdal` (Bivand, Keitt, und Rowlingson 2017) ist Ähnliches möglich. Mit `rworldmap` (South 2011) kann man sich Kartenmaterial der ganzen Welt (in gewisser Auflösung und mit begrenzter Aufteilung in Verwaltungseinheiten) vor-

Augen führen. Hat man die Längen- und Breitengrade von Objekten, so kann man sie auf der Karte einzeichnen. Will man keine Karten auf Papier erzeugen, so lässt sich mit interaktiven Karten à la Google Maps viel erreichen; das Paket `leaflet` (Cheng, Karambelkar, und Xie 2017) ist eine schöne Umsetzung dafür. Einen umfassenderen Einblick in die Thematik bietet das Lehrbuch von (Bivand, Pebesma, und Gomez-Rubio 2013) oder von (Brunsdon und Comber 2015).

Teil V

Modellieren

Kapitel 14

Grundlagen des Modellierens

Teil VI

Daten modellieren

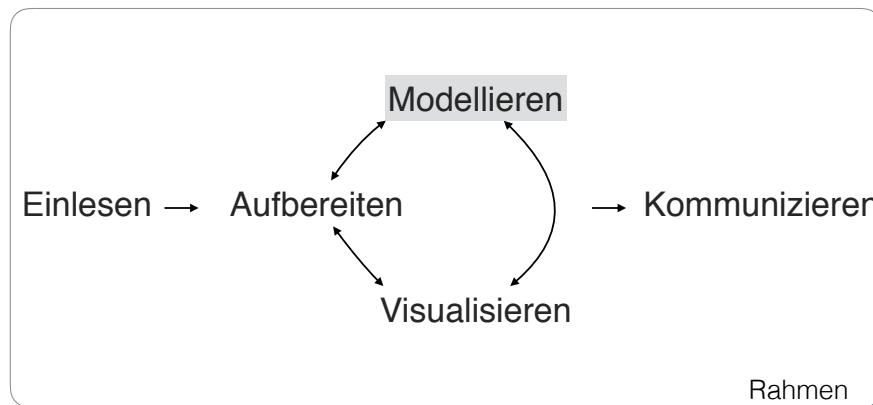


Lernziele:

- Erläutern können, was man unter einem Modell versteht.
- Die Ziele des Modellieren aufzählen und erläutern können.
- Die Vor- und Nachteile von einfachen vs. komplexen Modellen vergleichen können.
- Wissen, was man unter “Bias-Varianz-Abwägung” versteht.
- Um die Notwendigkeit von Trainings- und Test-Stichproben wissen.
- Wissen, was man unter Modellgüte versteht.
- Um die Schwierigkeiten der Prädiktorenauswahl wissen.

In diesem Kapitel benötigen wir diese Pakete:

```
library(tidyverse)
```



14.1 Was ist ein Modell? Was ist Modellieren?

In diesem Kapitel geht es um *Modelle* und *Modellieren*; aber was ist das eigentlich? Seit dem 16. Jahrhundert wird mit dem italienischen Begriff *modelle* ein *verkleinertes Muster, Abbild* oder Vorbild für ein Handwerksstück benannt (Gigerenzer 1980). Prototypisch für ein Modell ist - wer hätt's gedacht - ein Modellauto (s. Abb. 14.1; (Spurzem 2017)).

In die Wissenschaft kam der Begriff in der Zeit nach Kant, als man sich klar wurde, dass (physikalische) Theorien nicht die Wirklichkeit als solche zeigen, sondern ein *Modell* davon. Modellieren ist eine grundlegenden Tätigkeit, derer sich Menschen fortlaufend bedienen, um die Welt zu *verstehen*. Denn das Leben ist schwer... oder sagen wir: komplex. Um einen Ausschnitt der Wirklichkeit zu verstehen, erscheint es daher sinnvoll, sich einige als wesentlich erachteten Aspekte “herauszugreifen” bzw. auszusuchen und sich nur noch deren Zusammenspiel näher anzuschauen. Modelle sind häufig vereinfachend: es wird nur ein Ausschnitt der Wirklichkeit in einfacher Form berücksichtigt.

Da wir die Natur bzw. die Wirklichkeit oft nicht komplett erfassen, erschaffen wir uns ein Abbild von der Wirklichkeit, ein Modell.



Abbildung 14.1: Ein Modell eines VW-Käfers als Prototyp eines Modells

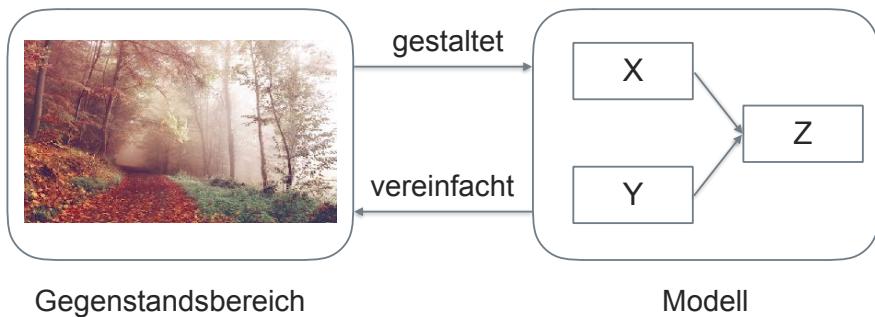


Abbildung 14.2: Modellieren

Manche Aspekte der Wirklichkeit sind wirklicher als andere: Interessiert man sich für den Zusammenhang von Temperatur und Grundwasserspiegel, so sind diese Dinge direkt beobachtbar. Interessiert man sich hingegen für Lebensqualität und Zufriedenheit, so muss man diese Untersuchungsgegenstände erst konstruieren, da Lebensqualität nicht direkt beobachtbar ist. Sprechen wir daher von Wirklichkeit lieber vorsichtiger vom *Gegenstandsbereich*, also den *konstruierten Auszug der Wirklichkeit* für den sich die forschende Person interessiert. Beste-nfalls (er)findet man eine *Annäherung* an die Wirklichkeit, schlechterenfalls eine *verzerrte*, gar *groß falsche* Darstellung. Da keine Wiedergabe der Wirklichkeit perfekt ist, sind streng genommen alle Modelle “falsch” in diesem Sinne.

Gegenstandsbereich und Modelle stehen in einer Beziehung miteinander (vgl. Abb. 14.2, das Foto stammt von Unrau (2017)).

Damit verstehen wir *Modellieren* als eine typische Aktivität von Menschen (Gigerenzer 1980), genauer *eines Menschen* mit einem *bestimmten Ziel*. Wir können gar nicht anders, als uns ein Modell unserer Umwelt zu machen; entsprechend kann (und muss man) von *mentalnen Modellen* sprechen. Vielfältige Medien kommen dazu in Frage: Bilder, Geschichten, Logik, Gleichungen. Wir werden uns hier auf eine bestimmte Art formalisierter Modelle, *numerische Modelle*, konzentrieren, weil es dort am einfachsten ist, die Informationen auf präzise Art und Weise herauszuziehen. Allgemein gesprochen ist hier unter Modellieren der Vorgang gemeint, ein Stück Wirklichkeit (“Empirie”) in eine *mathematische Struktur* zu übersetzen.

Wirklichkeit kann dabei als *empirisches System* bezeichnet werden, welches aus einer oder mehr Mengen von Objekten besteht, die zu einander in bestimmten Beziehungen stehen. Ein



Abbildung 14.3: Formaleres Modell des Modellierens

Bespiel wäre eine Reihe von Personen, die in bestimmten Größe-Relationen zueinander stehen oder eine Reihe von Menschen, bei denen die Füße tendenziell größer werden, je größer die Körpergröße ist.

Mit mathematische Struktur ist ein formalisiertes Pendant zum empirischen System gemeint, daher spricht man von einem *numerischen System*. Im Gegensatz zur empirischen System ist das numerische System rein theoretisch, also ausgedacht, nicht empirisch. Auch hier gibt es eine Reihe von Objekten, aber mathematischer Art, also z.B. Zahlen oder Vektoren. Diese mathematischen Objekten stehen wiederum in gewissen Relationen zueinander. Der springende Punkt ist: Im Modell sollen die Beziehungen zwischen den mathematischen Objekten die Beziehungen zwischen den empirischen Objekten widerspiegeln. Was heißt das? Stellen wir uns vor, der Klausurerfolg steigt mit der Lernzeit¹. Fragen wir das Modell, welchen Klausurerfolg Alois hat (er hat sehr viel gelernt), so sollte das Modell erwidern, dass Alois einen hohen Klausurerfolg hat (Modelle geben in diesem Fall gerne eine im Verhältnis große Zahl von sich). Damit würde das Modell korrekt die Empirie widerspiegeln.

Modellieren bedeutet ein Verfahren zu erstellen, welches empirische Sachverhalte adäquat in numerische Sachverhalte umsetzt.

Etwas spitzfindig könnte man behaupten, es gibt keine Modelle - es gibt nur Modelle *von* etwas; dieser Satz soll zeigen, dass zwar ein empirisches System für sich alleine stehen kann, aber ein Modell nicht. Ein Modell verweist immer auf ein empirisches System.

Abb. 14.3 stellt diese formalere Sichtweise des Modellierens dar; das empirische System E wird dem numerische System Z zugeordnet. Dabei besteht E aus einer Menge von Objekten O sowie einer Menge von *Relationen* R_E (Relationen meint hier nichts mehr als irgendwelche Beziehungen zwischen den Elementen von O). Analog besteht Z aus einer Menge von numerischen Objekten Z sowie einer Menge von Relationen R_Z (Relationen zwischen den Elementen von Z)².

14.2 Ein Beispiel zum Modellieren in der Datenanalyse

Schauen wir uns ein Beispiel aus der Datenanalyse an; laden Sie dazu zuerst den Datensatz zur Statistikklausur.

¹wieder ein typisches Dozentenbeispiel

²Diese Sichtweise des Modellierens basiert auf der Repräsentationstheorie des Messens nach Suppes und Zinnes (1962) zurück; vgl. Gigerenzer (1980)

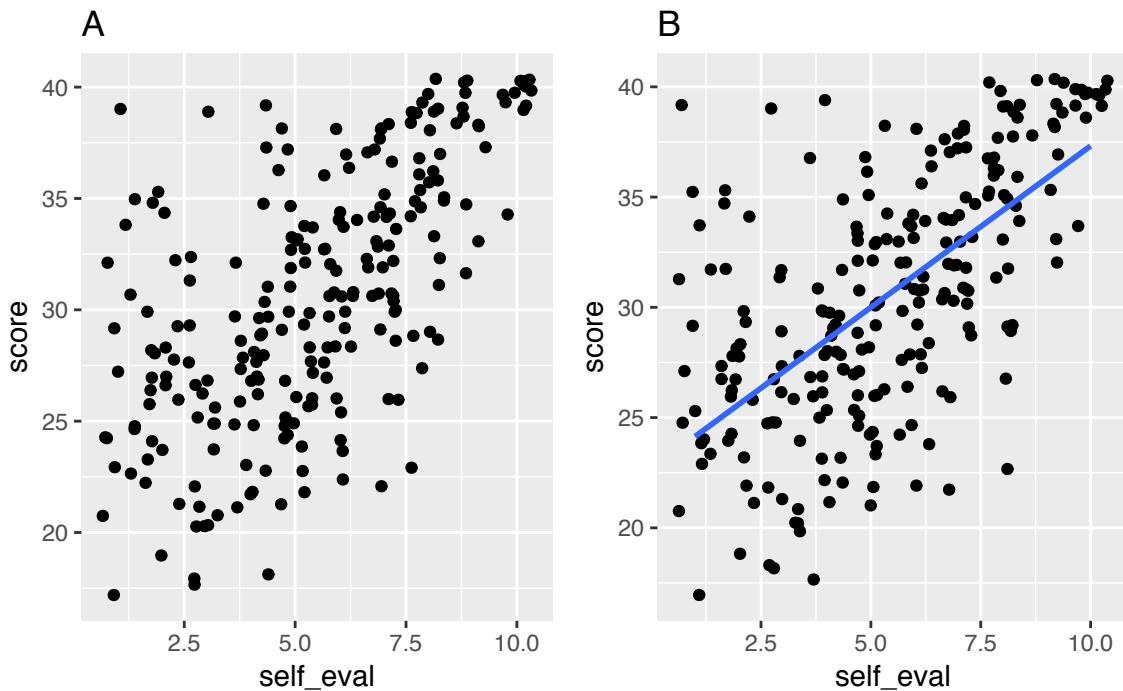


Abbildung 14.4: Ein Beispiel für Modellieren



Abbildung 14.5: Ein Beispiel für ein Pfadmodell

Im linken Plot (A; Abb. 14.4) sehen wir - schon übersetzt in eine Datenvisualisierung - den Gegenstandsbereich. Dort sind einige Objekte zusammen mit ihren Relationen abgebildet (Körpergröße und Schuhgröße). Der rechte Plot spezifiziert nun diesen Einfluss: Es wird ein *linearer Zusammenhang* (eine Gerade) zwischen Körpergröße und Schuhgröße unterstellt.

Im rechten Plot (B; Abb. 14.5) sehen wir ein Schema dazu, ein sog. *Pfadmodell*. Noch ist das Modell recht unspezifisch; es wird nur postuliert, dass Körpergröße auf Schuhgröße einen linearen Einfluss habe. Linear heißt hier, dass der Einfluss von Körpergröße auf Schuhgröße immer gleich groß ist, also unabhängig vom Wert der Körpergröße.

Ein etwas aufwändigeres Modell könnte so aussehen (Abb. 14.6):

Allgemeiner formuliert, haben wir einen oder mehrere *Eingabegrößen* bzw. *Prädiktoren*, von denen wir annehmen, dass sie einen Einfluss haben auf genau eine *Zielgröße* (*Ausgabegröße*) bzw. *Kriterium*.



Einfluss ist hier nicht (notwendig) kausal gemeint, auch wenn es das Wort so vermuten lässt. Stattdessen ist nur ein statistischer Einfluss gemeint; letztlich nichts anderes als ein Zusammenhang. In diesem Sinne könnte man postulieren, dass die Größe des Autos,

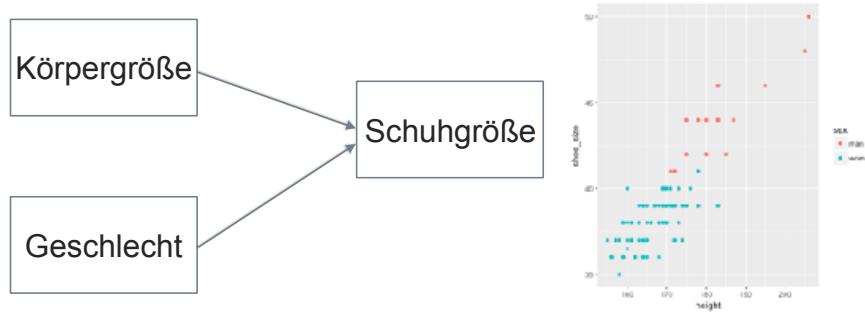


Abbildung 14.6: Ein etwas aufwändigeres Modell

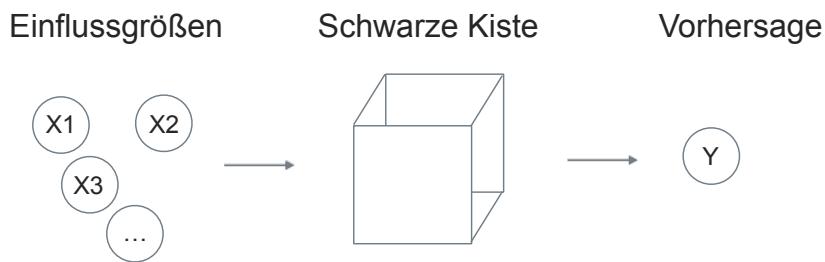


Abbildung 14.7: Modelle mit schwarzer Kiste

das man fährt einen “Einfluss” auf das Vermögen des Fahrers habe. Empirisch ist es gut möglich, dass man Belege für dieses Modell findet. Jedoch wird dieser Einfluss nicht kausal sein (man informiere mich, wenn es anders sein sollte).

Modelle, wie wir sie betrachten werden, berechnen eine quantitativen Zusammenhang zwischen diesen beiden Arten von Größen - Prädiktoren und Kriterien. Damit lassen sich unsere Modelle in drei Aspekte gliedern.

Die Einflussgrößen werden in einer “schwarzen Kiste”, die wir hier noch nicht näher benennen, irgendwie verwurstet, will sagen, verrechnet, so dass ein *geschätzter* Wert für das Kriterium, eine *Vorhersage* “hinten bei rauskommt”³. Wir gehen dabei nicht davon aus, dass unsere Modelle perfekt sind, sondern dass Fehler passieren. Mathematischer ausgedrückt:

$$Y = f(X) + \epsilon$$

Hier stehen Y für das Kriterium, X für den oder die Prädiktoren, f für die “schwarze Kiste” und ϵ für den Fehler, den wir bei unserer Vorhersage begehen. Durch den Fehlerterm in der Gleichung ist das Modell *nicht deterministisch*, sondern beinhaltet erstens einen funktionalen Term ($Y = f(x)$) und zweitens einen *stochastischen* Term (ϵ). Die schwarze Kiste könnte man auch als eine *datengenerierende Maschine* oder datengenerierenden Prozess bezeichnen.

³das ist schließlich entscheidend - frei nach Helmut Kohl

Übrigens: Auf das Skalenniveau der Eingabe- bzw. Ausgabegrößen (qualitativ vs. quantitativ) kommt es hier nicht grundsätzlich an; es gibt Modelle für verschiedene Skalenniveaus bzw. Modelle, die recht anspruchslos sind hinsichtlich des Skalenniveaus (sowohl für Eingabe- als auch Ausgabegrößen). Was die Ausgabegröße (das Kriterium) betrifft, so “fühlen” qualitative Variablen von quantitativen Variablen anders an. Ein Beispiel zur Verdeutlichung: “Gehört Herr Bussi-Ness zur Gruppe der Verweigerer oder der Wichtigmacher?” (qualitatives Kriterium); “Wie hoch ist der Wichtigmacher-Score von Herrn Bussi-Ness?” (quantitatives Kriterium). Ein Modell mit qualitativem Kriterium bezeichnet man auch als *Klassifikation*; ein Modell mit quantitativem Kriterium bezeichnet man auch als *Regression*. Bei letzterem Begriff ist zu beachten, dass er *doppelt* verwendet wird. Neben der gerade genannten Bedeutung steht er auch für ein häufig verwendetes Modell - eigentlich das prototypische Modell - für quantitative Kriterien.

14.3 Taxonomie der Ziele des Modellierens

Modelle kann man auf vielerlei Arten gliedern; für unsere Zwecke ist folgende Taxonomie der Ziele von Modellieren nützlich.

- Geleitetes Modellieren
 - Prädiktives Modellieren
 - Explikatives Modellieren
 - Ungeleitetes Modellieren
 - Dimensionsreduzierendes Modellieren
 - Fallreduzierendes Modellieren
-

Betrachten wir diese vier Ziele des Modellierens genauer.

Geleitetes Modellieren ist jede Art des Modellierens, wo die Variablen in Prädiktoren und Kriterien unterteilt werden, z.B. Abb. 14.5. Man könnte diese Modelle einfach darstellen als “X führt zu Y”.

Prädiktives Modellieren könnte man kurz als *Vorhersagen* bezeichnen. Hier ist das Ziel, eine Black Box geschickt zu wählen, so dass der Vohersagefehler möglichst klein ist. Man zielt also darauf ab, möglichst exakte Vorhersagen zu treffen. Sicherlich wird der Vohersagefehler kaum jemals Null sein; aber je präziser, desto besser. Das Innenleben der “schwarzen Kiste” interessiert uns hier *nicht*. Wir machen keine Aussagen über Ursache-Wirkungs-Beziehungen. Ein Beispiel für ein prädiktives Modell: “Je größer das Auto, desto höher das Gehalt”. Dabei werden wir wohl nicht annehmen, dass die Größe des Auto die Ursache für die Höhe des Gehalts ist⁴. Wir sind - in diesem Beispiel - lediglich daran interessiert, das Gehalt möglichst präzise zu schätzen; die Größe des Autos dient uns dabei als Prädiktor, wir verstehen sie

⁴bitte mir Bescheid geben, falls ich hier etwas übersehen haben sollte

nicht als Ursache. Ein altbekanntes Lamento der Statistiklehrer lautet “Korrelation heißt noch nicht Kausation!”. OK.

Explikatives Modellieren oder kurz *Erklären* bedeutet, verstehen zu wollen, *wie* oder *warum* sich ein Kriteriumswert so verändert, wie er es tut. Auf *welche Art* werden die Prädiktoren verrechnet, so dass eine bestimmter Kriteriumswert resultiert? Welche Prädikatoren sind dabei (besonders) wichtig? Ist die Art der Verrechnung abhängig von den Werten der Prädiktoren? Hierbei interessiert uns vor allem die *Beschaffenheit* der schwarzen Kiste; die Güte der Vorhersage ist zweitrangig. Oft, aber nicht immer, steht ein Interesse an den Ursache hinter dieser Art der Modellierung. Ursache-Wirkungs-Beziehungen gehören sicherlich zu den interessantesten und wichtigsten Dinge, die man untersuchen kann. Die Wissenschaft ist (bzw. viele Wissenschaftler sind) primär an Fragestellungen zur kausalen Beschaffenheit interessiert (Shmueli 2010). Ein Beispiel für diese Art von Modellierung wäre, ob Achtsamkeit zu weniger intensiven emotionalen Reaktionen führt (Sauer, Walach, und Kohls 2010). Übrigens: Es ist erlaubt, eine kausale Theorie zu vertreten, auch wenn eine Studie solche Schlussfolgerungen nur *eingeschränkt* oder gar *nicht* erlaubt (Shmueli 2010). Häufig werden Beobachtungsstudien auf Korrelationsbasis angeführt, um kausale Theorien zu testen. Natürlich ist der Preis für eine einfachere Studie, dass man weniger Evidenz für eine Theorie mit Kausalanspruch einstreichen kann. Aber irgendwo muss man ja anfangen (aber man sollte nicht bei einfachen Studien stehen bleiben).



Findet man, dass zwei Ereignisse (Variablen) zusammen hängen, so heißt das nicht, dass das eine Ereignis die Ursache des anderen sein muss. Man denke an Störche und Babies (wo es viele Störche gibt, gibt es viele Babies; so will es die Geschichte).

Auf der anderen Seite: Impliziert *Abwesenheit* von Korrelation, *Abwesenheit von kausalen Zusammenhängen*? Nein.

Warum ist das so? Zum einen können (nicht-lineare) Zusammenhänge vorliegen, die ein bestimmter Zusammenhangskoeffizient nicht aufdeckt. Oder Drittvariablen können einen Zusammenhang überdecken (sog. Suppressorvariablen). Außerdem mag die Stichprobe die Grundgesamtheit nicht korrekt widerspiegeln. Die Metapher von Milton Friedmans Thermostat⁵ zeigt das gut: Stellen Sie sich vor, Sie fahren in Ihrem Auto. Je stärker Sie auf das Gaspedal drücken, umso schneller fährt das Auto (unter sonst gleichem Umständen), richtig? Richtig. Stellen Sie sich vor, das Auto fährt bergauf. Dann wird das Auto langsamer werden (unter sonst gleichen Umständen), richtig? Richtig. Jetzt stellen Sie sich vor, Sie sitzen auf dem Beifahrersitz und haben vom Autofahren wenig Ahnung. Die Fahrt geht durch die Berge; das Auto fährt mit konstanter Geschwindigkeit. Sie könnten denken: “Ah, wieviel Gas man gibt, hat keinen Einfluss auf die Geschwindigkeit!” und “Ah, die Steigung der Straße hat keinen Einfluss auf die Geschwindigkeit”. Obwohl es tatsächlich einen kausalen Einfluss beider Größen auf die Geschwindigkeit gibt, sieht man keinen Zusammenhang in den Daten. Da führt kein Weg raus. Auf einer tieferen Ebene: Datenanalyse allein kann die “Essenz” eines Phänomens grundsätzlich nicht aufdecken.

⁵<http://themonkeycage.org/2012/07/milton-friedmans-thermostat/>

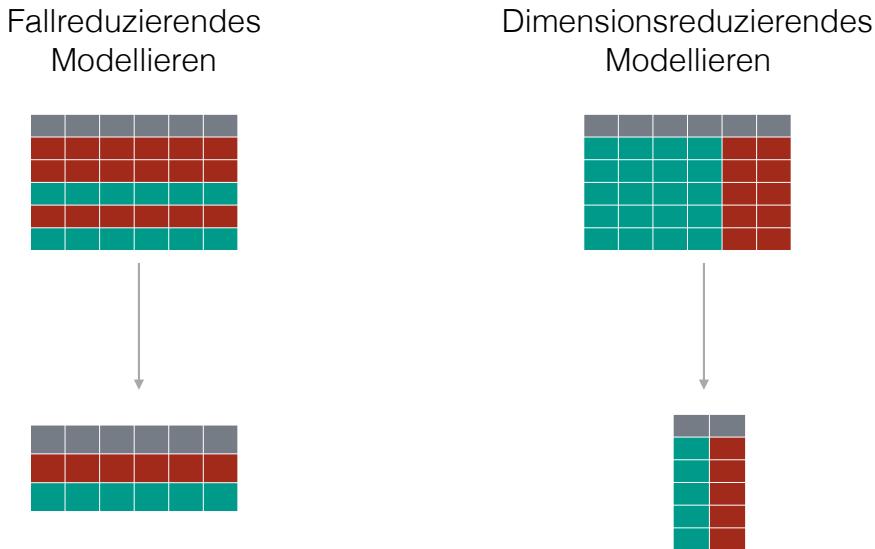


Abbildung 14.8: Die zwei Arten des ungeleiteten Modellierens

Vorhersagen und Erklären haben gemein, dass Eingabegrößen genutzt werden, um Aussagen über einen Ausgabegröße zu treffen. Anders gesagt: Es liegt eine Zielgröße mit bekannten Ausprägungen vor, zumindest für eine Reihe von Fällen. Hat man einen Datensatz, so kann man prüfen, *wie gut* das Modell funktioniert, also wie genau man die Ausgabewerte vorhergesagt hat. Das ist also eine Art “Lernen mit Anleitung” oder *angeleitetes Lernen* oder *geleitetes Modellieren* (engl. *supervised learning*). Abbildung 14.7 gibt diesen Fall wieder.

Beim *ungeleiteten Modellieren* entfällt die Unterteilung zwischen Prädiktor und Kriterium. *Ungeleitetes Modellieren (Reduzieren)* meint, dass man die Fülle des Datenmaterials verringert, in dem man ähnliche Dinge zusammenfasst (vgl. Abb. 14.8).

Fasst man *Fälle* zusammen, so spricht man von *Fallreduzierendem Modellieren*. Zum Beispiel könnte man spektakulärerweise “Britta”, “Carla” und “Dina” zu “Frau” und “Joachim”, “Alois” und “Casper” zu “Mann” zusammen fassen.

Analog spricht man von *Dimensionsreduzierendes Modellieren* wenn Variablen zusammengefasst werden. Hat man z.B. einen Fragebogen zur Mitarbeiterzufriedenheit mit den Items “Mein Chef ist fair”, “Mein Chef ist kompetent”, “Meinem Chef ist meine Karriere wichtig”, so könnte man - wenn die Daten dies unterstützen - die Items zu einer Variable “Zufriedenheit mit Chef” zusammenfassen.

Wenn also das Ziel des Modellieren lautet, die Daten zu reduzieren, also z.B. Kunden nach Persönlichkeit zu gruppieren, so ist die Lage anders als beim geleiteten Modellieren: Es gibt keine Zielgröße. Wir wissen nicht, was die “wahre Kundengruppe” ist, zu der Herrn Casper Bussi-Ness gehört. Wir sagen eher, “OK, die drei Typen sind sich irgendwie ähnlich, sie werden wohl zum selben Typen von Kunden gehören”. Wir tappen (noch mehr) in Dunkeln, was die “Wahrheit” ist im Vergleich zum angeleiteten Modellieren. Unser Modell muss ohne Hinweise darauf, was richtig ist auskommen. Man spricht daher in diesem Fall von *Lernen ohne Anleitung* oder *ungeleitetes Modellieren* (engl. *unsupervised learning*).

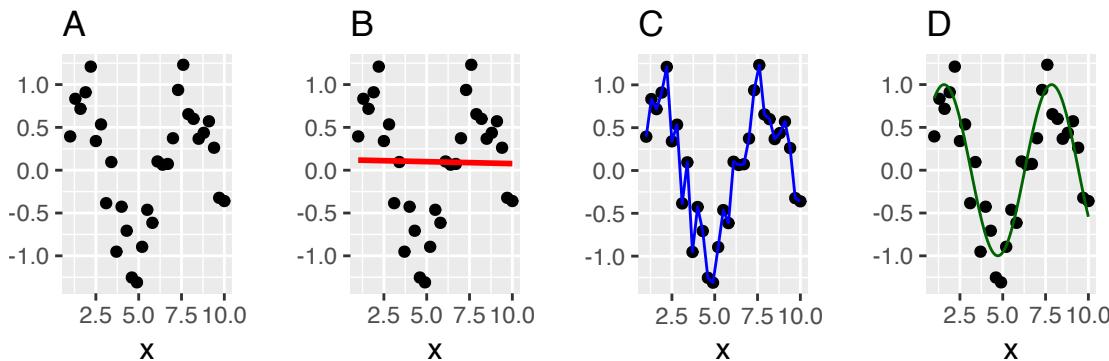


Abbildung 14.9: Welches Modell (Teil B-D; rot, grün, blau) passt am besten zu den Daten (Teil A) ?

14.4 Die vier Schritte des statistischen Modellierens

Modellieren ist in der Datenanalyse bzw. in der Statistik eine zentrale Tätigkeit. Modelliert man in der Statistik, so führt man die zwei folgenden Schritte aus:

1. Man wählt eines der vier Ziele des Modellierens (z.B. ein prädiktives Modell).
2. Man wählt ein Modell aus (genauer: eine Modelfamilie), z.B. postuliert man, dass die Körpergröße einen linearen Einfluss auf die Schuhgröße habe.
3. Man bestimmt (berechnet) die Details des Modells anhand der Daten: Wie groß ist die Steigung der Geraden und wo ist der Achsenabschnitt? Man sagt auch, dass man die *Modellparameter* anhand der Daten schätzt (“Modellinstantiierung” oder “Modellanpassung”, engl. “model fitting”).
4. Dann prüft man, wie gut das Modell zu den Daten passt (Modellgüte, engl. “model fit”); wie gut lässt sich die Schuhgröße anhand der Körpergröße vorhersagen bzw. wie groß ist der Vorhersagefehler?

14.5 Einfache vs. komplexe Modelle: Unter- vs. Überanpassung

Je komplexer ein Modell, desto besser passt sie meistens auf den Gegenstandsbereich. Eine grobe, Holzschnitt artige Theorie ist doch schlechter als eine, die feine Nuancen berücksichtigt, oder nicht? Einiges spricht dafür; aber auch einiges dagegen. Schauen wir uns ein Problem mit komplexen Modellen an.

Der Plot A (links) von Abb. 14.9 zeigt den Datensatz ohne Modell; Plot B legt ein lineares Modell (rote Gerade) in die Daten. Plot C zeigt ein Modell, welches die Daten exakt erklärt - die (blaue) Linie geht durch alle Punkte. Der 4. Plot zeigt ein Modell (grüne Linie), welches die Punkte gut beschreibt, aber nicht exakt trifft.

Welchem Modell würden Sie (am meisten) vertrauen? Das “blaue Modell” beschreibt die

Daten sehr gut, aber hat das Modell überhaupt eine “Idee” vom Gegenstandsbereich, eine “Ahnung”, wie Y und X zusammenhängen, bzw. wie X einen Einfluss auf Y ausübt? Offenbar nicht. Das Modell ist “übergenau” oder zu komplex. Man spricht von *Überanpassung* (engl. *overfitting*). Das Modell scheint zufälliges, bedeutungsloses Rauschen zu ernst zu nehmen. Das Resultat ist eine zu wackelige Linie - ein schlechtes Modell, da wir wenig Anleitung haben, auf welche Y-Werte wir tippen müssten, wenn wir neue, unbekannte X-Werte bekämen.

Beschreibt ein Modell (wie das blaue Modell hier) eine Stichprobe sehr gut, heißt das noch *nicht*, dass es auch zukünftige (und vergleichbare) Stichproben gut beschreiben wird. Die Güte (Vorhersagegenauigkeit) eines Modells sollte sich daher stets auf eine neue Stichprobe beziehen (Test-Stichprobe), die nicht in der Stichprobe beim Anpassen des Modells (Trainings-Stichprobe) enthalten war.

Was das “blaue Modell” zu detailverliebt ist, ist das “rote Modell” zu simpel. Die Gerade beschreibt die Y-Werte nur sehr schlecht. Man hätte gleich den Mittelwert von Y als Schätzwert für jedes einzelne Y_i hernehmen können. Dieses lineare Modell ist *unterangepasst*, könnte man sagen (engl. *underfitting*). Auch dieses Modell wird uns wenig helfen können, wenn es darum geht, zukünftige Y-Werte vorherzusagen (gegeben jeweils einen bestimmten X-Wert).

Ah! Das *grüne Modell* scheint das Wesentliche, die “Essenz” der “Punktebewegung” zu erfassen. Nicht die Details, die kleinen Abweichungen, aber die “große Linie” scheint gut getroffen. Dieses Modell erscheint geeignet, zukünftige Werte gut zu beschreiben. Das grüne Modell ist damit ein Kompromiss aus Einfachheit und Komplexität und würde besonders passen, wenn es darum gehen sollte, zyklische Veränderungen zu erklären⁶.

Je komplexer ein Modell ist, desto besser beschreibt es einen bekannten Datensatz (Trainings-Stichprobe). Allerdings ist das Modell, welches den Trainings-Datensatz am besten beschreibt, nicht zwangsläufig das Modell, welches neue, unbekannte Daten am besten beschreibt. Oft im Gegenteil!

Je komplexer das Modell, desto kleiner der Fehler im *Trainings*-Datensatz. Allerdings: Die Fehler-Kurve im *Test*-Datensatz ist *U-förmig*: Mit steigender Komplexität wird der Fehler einige Zeit lang kleiner; ab einer gewissen Komplexität steigt der Fehler im Test-Datensatz wieder (vgl. Abb. 14.10)! Eine ‘mittlere’ Komplexität ist daher am besten; die Frage ist nur, wie viel ‘mittel’ ist.

14.6 Bias-Varianz-Abwägung

Einfache Modelle bilden (oft) verfehlten oft wesentliche Aspekte des Gegenstandsbereich; die Wirklichkeit ist häufig zu komplex für einfache Modelle. Die resultierende *Verzerrung* in den vorhergesagten Werten nennt man auch *Bias*. Mit anderen Worten: ist ein Modell zu einfach, passt es zu wenig zu den Daten (engl. *underfitting*). Auf der anderen Seite ist das Modell aber *robust* in dem Sinne, dass sich die vorhergesagten Werte kaum ändern, falls sich der Trainings-Datensatz etwas ändert.

⁶Tatsächlich wurden die Y-Werte als Sinus-Funktion plus etwas normalverteiltes Rauschen simuliert.

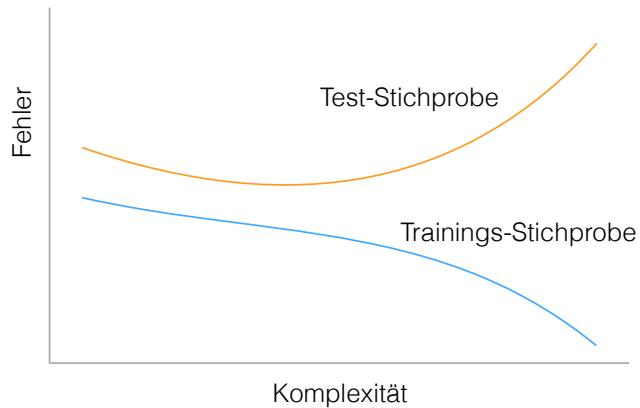


Abbildung 14.10: 'Mittlere' Komplexität hat die beste Vorhersagegenauigkeit (am wenigsten Fehler) in der Test-Stichprobe

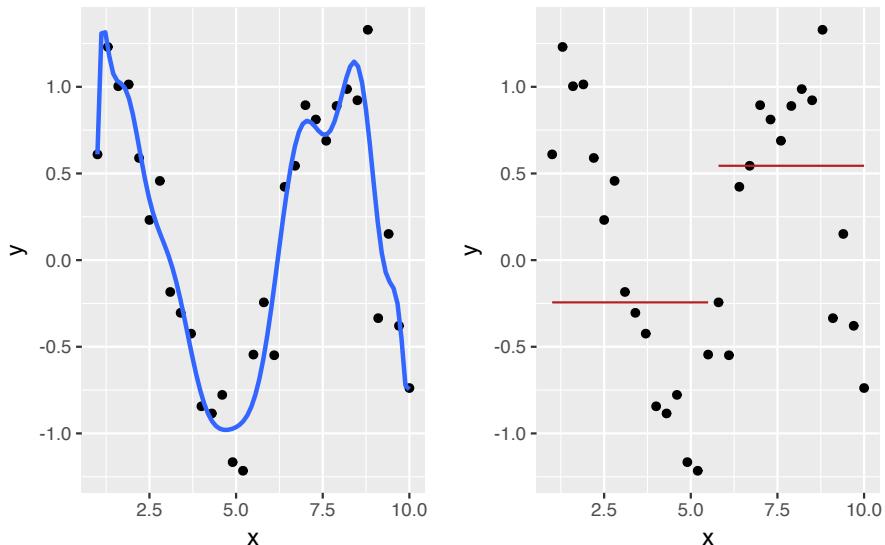


Abbildung 14.11: Der Spagat zwischen Verzerrung und Varianz

Ist das Modell aber zu reichhaltig ("komplex"), bildet es alle Details des Trainings-Datensatzes ab, wird es auch zufällige Variation des Datensatzes vorhersagen; Variation, die nicht relevant ist, der nichts Eigentliches abbildet. Das Modell ist "überangepasst" (engl. *overfitting*); geringfügige Änderungen im Datensatz können das Modell stark verändern. Das Modell ist nicht robust. Auf der positiven Seite werden die Nuancen der Daten gut abgebildet; der Bias ist gering bzw. tendenziell geringer als bei einfachen Modellen.

Einfache Modelle: Viel Bias, wenig Varianz. Komplexe Modelle: Wenig Bias, viel Varianz.

Dieser Sachverhalt ist in folgendem Diagramm dargestellt (vgl. Abb. 14.11; vgl. Kuhn & Johnson (2013)).

Der linke Plot zeigt ein komplexes Modell⁷; das Modell (blaue Linie) erscheint "zittrig"; kleine

⁷Genauer gesagt ein Polynom von Grad 5.

Änderungen in den Daten können große Auswirkungen auf das Modell (Verlauf der blauen Linie) haben. Darüber hinaus sind einige Details des Modells unplausibel: es gibt viele kleine „Hügel“, die nicht augenscheinlich plausibel sind.

Der Plot auf der rechten Seiten hingegen ist sehr einfach und robust. Änderungen in den Daten werden vergleichsweise wenig Einfluss auf das Modell (die beiden roten Linien) haben.

14.7 Training- vs. Test-Stichprobe

Wie wir gerade gesehen haben, kann man *immer* ein Modell finden, welches die *vorhandenen* Daten sehr gut beschreibt. Das gleicht der Tatsache, dass man im Nachhinein (also bei vorhandenen Daten) leicht eine Erklärung findet. Ob diese Erklärung sich in der Zukunft, bei unbekannten Daten bewahrheitet, steht auf einem ganz anderen Blatt.

Daher sollte man *immer* sein Modell an einer Stichprobe *entwickeln* (“trainieren” oder “üben”) und an einer zweiten Stichprobe *testen*. Die erste Stichprobe nennt man auch *training sample* (Trainings-Stichprobe) und die zweite *test sample* (Test-Stichprobe). Entscheidend ist, dass das Test-Sample beim Entwickeln des Modells unbekannt war bzw. nicht verwendet wurde.

Die Güte des Modells sollte nur anhand eines - bislang nicht verwendeten - Test-Samples überprüft werden. Das Test-Sample darf bis zur Modellüberprüfung nicht analysiert werden.

Die Modellgüte ist im Trainings-Sample meist deutlich besser als im Test-Sample (vgl. die Fallstudie dazu: 16.8).

Zum Aufteilen verfügbarer Daten in eine Trainings- und eine Test-Stichprobe gibt es mehrere Wege. Einer sieht so aus:

```
train <- slice(stats_test, 1:200)
test <- slice(stats_test, 201:306)
```

`dplyr::slice` schneidet eine ‘Scheibe’ aus einem Datensatz. Mit `anti_join` kann man einen generellen Ansatz wählen: ‘`anti_join`’ vereinigt die *nicht-gleichen Zeilen* zweier Datensätze. Die Gleichheit wird geprüft anhand aller Spalten mit identischem Namen in beiden Datensätzen.

```
train <- stats_test %>%
  sample_frac(.8, replace = FALSE) # Stichprobe von 80%, ohne Zurücklegen

test <- stats_test %>%
  anti_join(train) # Alle Zeilen von "stats_test", die nicht in "train" vorkommen
```

Damit haben wir ein Trainings-Sample (`train`), in dem wir ein oder besser mehrere Modelle berechnen können. Die Modellgüte beurteilen wir dann im Test-Sample.

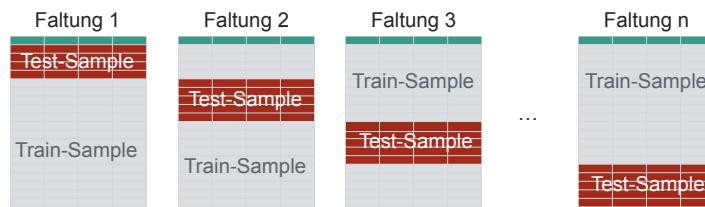


Abbildung 14.12: Beispiel für eine $k=4$ Kreuzvalidierung

14.8 Kreuzvalidierung

So schön wie dieses Vorgehen auch ist, es ist nicht perfekt. Ein Nachteil ist, dass unsere Modellgüte wohl *anders* wäre, hätten wir andere Fälle im Test-Sample erwischt. Würden wir also ein neues Trainings-Sample und ein neues Test-Sample aus diesen Datensatz ziehen, so hätten wir wohl andere Ergebnisse. Was wenn diese Ergebnisse nun deutlich von den ersten abweichen? Dann würde unser Vertrauen in die die Modellgüte sinken. Es wäre also hilfreich, wenn wir das Aufteilen der gesamten Stichprobe in Trainings- und Test-Sample k mal (z.B. $k = 10^8$) wiederholen, um stabilere Schätzungen der Vorhersagegüte im Test-Sample zu bekommen; vgl. Abbildung 14.12. Ein solches Verfahren nennt man *k-fache Kreuzvalidierung* (engl. *k-fold cross-validation*). Jeder Fall kommt also 1 Mal in das Test-Sample und $k-1$ Mal in das Trainings-Sample; die Trainings-Sample sind in jedem Durchlauf gleich groß. Jeder Durchgang wird auch als Faltung (fold) bezeichnet. Der Vorteil dieses Verfahren ist, dass unsere Vorhersagen stabiler werden - und damit im Schnitt besser sind. Ein Nachteil ist, dass ein Modell dann k mal berechnet werden muss, was Zeitverlust bedeutet.

Die Kreuzvalidierung ist Methode zur Beurteilung der Modellgüte. Der Prozess der Aufteilung in Train- und Test-Sample wird dabei (mehrfach) vorgenommen. Die Modellgüte wird anschließend summarisch beurteilt.

⁸10 Mal scheint sich gut zu bewähren: <http://appliedpredictivemodeling.com/blog/2014/11/27/vpuig01pqbk1mi72b81c13ij5hj2qm>

14.9 Wann welches Modell?

Tja, mit dieser Frage lässt sich ein Gutteil des Kopfzerbrechens in diesem Metier erfassen. Die einfache Antwort lautet: Es gibt kein “bestes Modell”⁹, aber es mag für *einen bestimmten Gegenstandsbereich, in einem bestimmten (historisch-kulturellen) Kontext, für ein bestimmtes Ziel und mit einer bestimmten Stichprobe* ein best mögliches Modell geben. Dazu einige Eckpfeiler:

- Unter sonst gleichen Umständen sind einfachere Modelle den komplexeren vorzuziehen. Gott sei Dank.
- Je nach Ziel der Modellierung ist ein erklärendes Modell oder ein Modell mit reinem Vorhersage-Charakter vorzuziehen.
- Man sollte stets mehrere Modelle vergleichen, um abzuschätzen, welches Modell in der aktuellen Situation geeigneter ist.

14.10 Modellgüte

Wie “gut” ist mein Modell? Modelle bewerten bzw. vergleichend bewerten ist einer der wichtigsten Aufgaben beim Modellieren. Ein Klassifikationsmodell muss anders beurteilt werden als ein Regressionsmodell. Die Frage der Modellgüte hat viele feine technisch-statistische Verästelungen, aber einige wesentlichen Aspekte kann man einfach zusammenfassen.

Kriterium der theoretischen Plausibilität: Ein statistisches Modell sollte theoretisch plausibel sein.

Anstelle “alles mit allem” durchzuprobieren, sollte man sich auf Modelle konzentrieren, die theoretisch plausibel sind. Die Modellwahl ist theoretisch zu begründen.

Kriterium der guten Vorhersage: Die Vorhersagen eines Modells sollen präzise und überraschend sein.

Dass ein Modell die Wirklichkeit präzise vorhersagen soll, liegt auf der Hand. Hier verdient nur der Term *vorhersagen* Beachtung. Es ist einfach, im Nachhinein Fakten (Daten) zu erklären. Jede Nachbesprechung eines Bundesliga-Spiels liefert reichlich Gelegenheit, *posthoc* Erklärungen zu hören. Schwieriger sind Vorhersagen¹⁰. Die Modellgüte ist also idealerweise an *in der Zukunft liegende* Ereignisse bzw. deren Vorhersage zu messen. Zur Not kann man auch schon in der Vergangenheit angefallene Daten hernehmen. Dann müssen diese Daten aber *für das Modell* neu sein.

Was ist mit überraschend gemeint? Eine Vorhersage, dass die Temperatur morgen in Nürnberg zwischen -30 und +40°C liegen wird, ist sicherlich sehr treffend, aber nicht unbedingt präzise und nicht wirklich überraschend. Die Vorhersage, dass der nächste Chef der Maurer-Innung

⁹das sog. “No Free Lunch Theorem”

¹⁰Gerade wenn sie die Zukunft betreffen; ein Bonmot, das Yogi Berra nachgesagt wird.

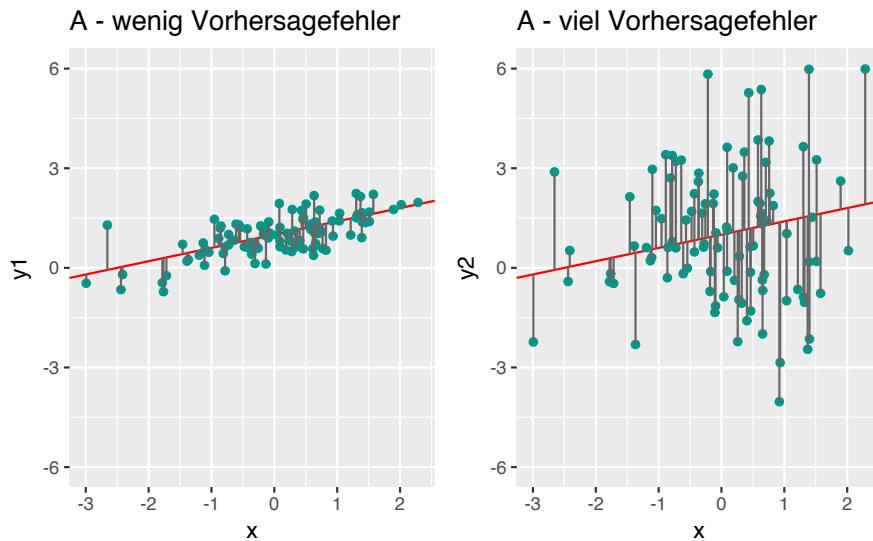


Abbildung 14.13: Geringer (links) vs. hoher (rechts) Vorhersagefehler

(wenn es diese geben sollte) ein Mann sein wird, und nicht eine Frau, kann zwar präzise sein, ist aber nicht überraschend. Wir werden also in dem Maße unseren Hut vor dem Modell ziehen, wenn die Vorhersagen sowohl präzise als auch überraschen sind. Dazu später mehr Details.

Kriterium der Situationsangemessenheit: Die Güte des Modells ist auf die konkrete Situation abzustellen.

14.10.1 Modellgüte in Regressionsmodellen

Einfach gesagt ist das Ziel ein einem Regressionsmodell mit seiner Vorhersage möglichst nahe an der Realität zu sein. Sage ich für morgen eine Höchsttemperatur von 30 Grad voraus, und es werden 20 Grad, so ist mein Vorhersagefehler 10 Grad. Die Vorhersagefehler sollen also möglichst gering sein. Dabei muss beachtet werden, dass komplexere Modelle bessere Vorhersagen treffen. Daher “bestrafen” manche Gütekoeffizienten komplexe Modelle, um diesen Vorteil auszugleichen.

In Regressionsmodellen beruht der Vorhersagefehler oft auf der Differenz zwischen tatsächlichen und vorhergesagten Werten.

Der einfache Grundsatz lautet: Je geringer die Vorhersagefehler, desto besser; Abb. 14.13 zeigt ein Regressionsmodell mit wenig Vorhersagefehler (A; links) und ein Regressionsmodell mit viel Vorhersagefehler B; rechts).

Außerdem kann (und sollte) die Güte eines Regressionsmodells gegen ein *Nullmodell* abgeglichen sein. Ein Nullmodell stellt ein sehr einfaches oder ‘dummies’ Modell dar - etwa in dem es für morgen die Jahresschnittstemperatur vorhersagt (oder den Mittelwert aller Tage, die dem Modell bekannt sind). Ist die Vorhersage unseres Modells nicht (wesentlich) besser als die des Nullmodells, werden sich andere wenig beeindruckt von unserem Modell zeigen.

ID	beobachtet	vorhergesagt
A	bestanden	bestanden
B	bestanden	durchgefallen
C	durchgefallen	durchgefallen
D	bestanden	bestanden
E	durchgefallen	durchgefallen

Abbildung 14.14: Sinnbild für die Trefferquote eines Klassifikationsmodells

14.10.2 Modellgüte bei Klassifikationsmodellen

Bei Klassifikationsmodellen lautet die Schlüsselfrage: Wie oft war die Vorhersage des Modells richtig? Wenn mein Modell für die letzten 10 Tage ‘Regen’ vorhergesagt hat, es aber aber 9 mal nicht geregnet, so läge die Trefferquote (Richtigkeit, Korrektklassifikationsrate) meines Modells bei 90%; Abbildung ?? und Tabelle ?? stellt diesen Zusammenhang sinnbildlich dar.

Zu beachten ist, dass auch bei Klassifikationsmodellen ein Nullmodell zum Vergleich herangezogen werden sollte. Wenn ich weiß, dass es in Nürnberg im Schnitt

14.11 Auswahl von Prädiktoren

Wie oben diskutiert, stellen wir ein (geleitetes) Modell gerne als ein Pfaddiagramm des Typs $X \rightarrow Y$ dar (wobei X ein Vektor sein kann). Nehmen wir an das Kriterium Y als gesetzt an; bleibt die Frage: Welche Prädiktoren (X) wählen wir, um das Kriterium möglichst gut vorherzusagen?

Eine einfache Frage. Keine leichte Antwort. Es gibt zumindest drei Möglichkeiten, die Prädiktoren zu bestimmen: theoriegeleitet, datengetrieben oder auf gut Glück.

- theoriegeleitet: Eine starke Theorie macht präzise Aussagen, welche Faktoren eine Rolle spielen und welche nicht. Auf dieser Basis wählen wir die Prädiktoren. Diese Situation ist wünschenswert; nicht nur, weil Sie Ihnen das Leben leicht macht, sondern weil es nicht die Gefahr gibt, die Daten zu “overfitten”, “Rauschen als Muster” zu bewerten - kurz: zu optimistisch bei der Interpretation von Statistiken zu sein.

Tabelle 14.1: Veranschulichung der Klassifikationsgüte eines Klassifikationsmodells

x	y	binned	y_group_md
1.00	0.61	(-Inf,5.5]	-0.24
1.30	1.23	(-Inf,5.5]	-0.24
1.60	1.00	(-Inf,5.5]	-0.24
1.90	1.01	(-Inf,5.5]	-0.24
2.20	0.59	(-Inf,5.5]	-0.24
2.50	0.23	(-Inf,5.5]	-0.24
2.80	0.46	(-Inf,5.5]	-0.24
3.10	-0.18	(-Inf,5.5]	-0.24
3.40	-0.30	(-Inf,5.5]	-0.24
3.70	-0.42	(-Inf,5.5]	-0.24
4.00	-0.84	(-Inf,5.5]	-0.24
4.30	-0.88	(-Inf,5.5]	-0.24
4.60	-0.78	(-Inf,5.5]	-0.24
4.90	-1.17	(-Inf,5.5]	-0.24
5.20	-1.22	(-Inf,5.5]	-0.24
5.50	-0.55	(-Inf,5.5]	-0.24
5.80	-0.24	(5.5, Inf]	0.54
6.10	-0.55	(5.5, Inf]	0.54
6.40	0.42	(5.5, Inf]	0.54
6.70	0.54	(5.5, Inf]	0.54
7.00	0.89	(5.5, Inf]	0.54
7.30	0.81	(5.5, Inf]	0.54
7.60	0.69	(5.5, Inf]	0.54
7.90	0.89	(5.5, Inf]	0.54
8.20	0.99	(5.5, Inf]	0.54
8.50	0.92	(5.5, Inf]	0.54
8.80	1.33	(5.5, Inf]	0.54
9.10	-0.33	(5.5, Inf]	0.54
9.40	0.15	(5.5, Inf]	0.54
9.70	-0.38	(5.5, Inf]	0.54
10.00	-0.74	(5.5, Inf]	0.54

- datengetrieben: Kurz gesagt werden die Prädiktoren ausgewählt, welche das Kriterium am besten vorhersagen. Das ist einerseits stimmig, andererseits birgt es die Gefahr, dass Zufälligkeiten in den Daten für echte Strukturen, die sich auch in zukünftigen Stichproben finden würden, missverstanden werden.
- auf gut Glück: tja, kann man keine Theorie zu Rate ziehen und sind die Daten wenig aussagekräftig oder man nicht willens ist, sie nicht genug zu quälen analysieren, so neigen Menschen dazu, zuerst sich selbst und dann andere von der Plausibilität der Entscheidung zu überzeugen. Keine sehr gute Strategie.

In späteren Kapiteln betrachten wir Wege, um Prädiktoren für bestimmte Modelle auszuwählen.

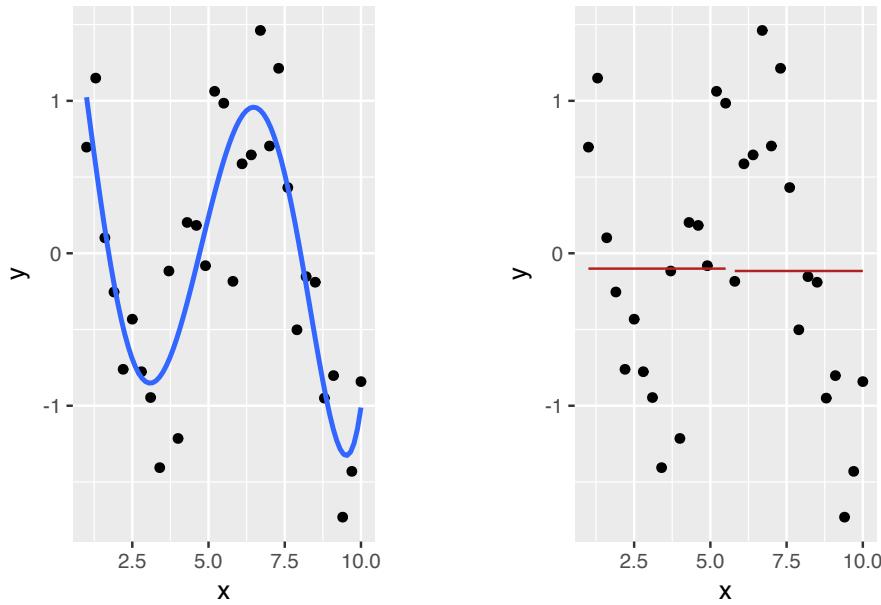


Abbildung 14.15: Bias-Varianz-Abwägung. Links: Wenig Bias, viel Varianz. Rechts: Viel Bias, wenig Varianz.

14.12 Aufgaben

- Erfolg beim Online-Dating

Lesen Sie diesen¹¹ Artikel (Sauer und Wolff 2016). Zeichnen Sie ein Pfaddiagramm zum Modell!¹².

- Ziele des Modellierens

Welche drei Ziele des Modellierens kann man unterscheiden?¹³

- Bias-Varianz-Abwägung

Betrachten Sie Abb. 14.15. Welches der beiden Modelle (visualisiert im jeweiligen Plot) ist wahrscheinlich...

- mehr bzw. weniger robust gegenüber Änderungen im Datensatz?
- mehr oder weniger präzise?

- Richtig oder falsch?¹⁴



Richtig oder Falsch!?

¹¹https://thewinnow.com/papers///5202-the-effect-of-a-status-symbol-on-success-in-online-dating-/an-experimental-study-data-paper?review_it=true

¹²Status → Erfolg beim Online-Dating

¹³14.3

¹⁴R, F, F, F, R

1. Die Aussage “Pro Kilo Schoki steigt der Hüftumfang um einen Zentimeter” kann als Beispiel für ein deterministisches Modell herhalten.
2. Gruppiert man Kunden nach ähnlichen Kaufprofilen, so ist man insofern an “Reduzieren” der Datenmenge interessiert.
3. Grundsätzlich gilt: Je komplexer ein Modell, desto besser.
4. Mit “Bias” ist gemeint, dass ein Modell “zittrig” oder “wackelig” ist - sich also bei geringer Änderung der Stichprobendaten massiv in den Vorhersagen ändert.
5. In der Gleichung $Y = f(x) + \epsilon$ steht ϵ für den Teil der Kriteriums, der nicht durch das Modell erklärt wird.

14.13 Verweise

- Einige Ansatzpunkte zu moderner Statistik (“Data Science”) finden sich bei Peng und Matsui (2015).
- Chester Ismay erläutert einige Grundlagen von R und RStudio, die für Modellierung hilfreich sind: <https://bookdown.org/chesterismay/rbasics/>.
- Eine klassische und sehr gute Einführung findet sich bei James, Witten, Hastie & Tibshirani (James, Witten, Hastie, und Tibshirani 2013b). Dieses Buch bietet ein frei verfügbares PDF¹⁵.

¹⁵ <http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf>

Kapitel 15

Inferenzstatistik



Lernziele:

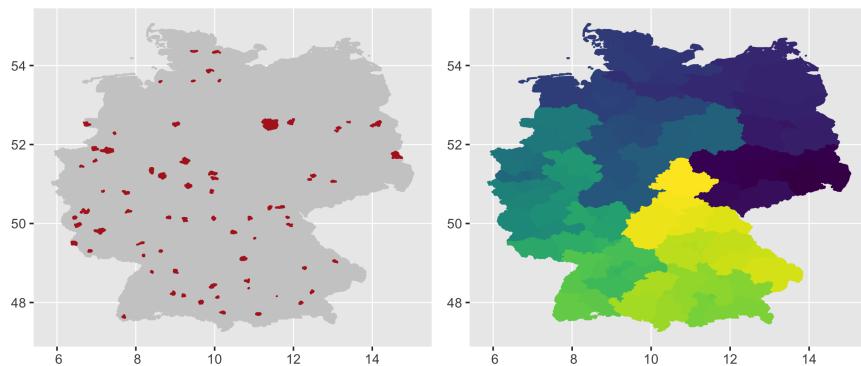
- Den Unterschied zwischen Stichprobe und Grundgesamtheit verstehen.
- Die Idee des statistischen Inferierens erläutern können.
- Simulationskonzepte anwenden können, um inferenzstatistische Schlüsse zu ziehen.
- Den p-Wert erläutern können.
- Den p-Wert kritisieren können.
- Alternativen zum p-Wert kennen.
- Inferenzstatistische Verfahren für häufige Fragestellungen kennen.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(pwr)  # Powerberechnung
library(compute.es)  # Effektstärken
library(tidyverse)  # Datenjudo
library(broom)  # Anova-Ergebnisse aufräumen
library(BayesFactor)  # Bayes-Faktor berechnen
library(mosaic)  # Komfort
```

15.1 Stichprobe und Grundgesamtheit

In bisherigen Kapiteln haben wir einen Datensatz betrachtet und Aussagen über diesen getroffen. Das ist zwar schön, aber häufig ist man an mehr interessiert. Sie haben zehn Ihrer Freunde zu politischen Präferenzen gefragt? Interessant, aber für eine Bild-Schlagzeile reicht es nicht. Hätten Sie *alle Bundesbürger* befragt, würden die Reporter (vielleicht) bei Ihnen Schlange stehen. Leider ist es aufwändig, alle Deutschen zu befragen. Angenehm ist es dann, wenn die Statistik es einem erlaubt, von der Stichprobe auf eine Gesamtmenge zu



Links: Stichproben; Vollerhebung

Abbildung 15.1: Stichproben vs. Vollerhebung

schließen - genauer von einer Stichprobenzkennzahl auf den korrespondierenden Wert einer Grundgesamtheit (synonym: Population). Die Stichprobenzkennzahl bezeichnet man auch als eine *Statistik* und den korrespondierenden Wert in der Population auch als *Parameter*. Diese Art von Schlüsseleien bezeichnet man als *Inferenzstatistik*. Betrachten Sie als Beispiel Abbildung 15.1: Im linken Teil sehen Sie einige Stichproben der PLZ-Gebiete, und rechts sehen Sie das gesamte Bundesgebiet, auf das Sie in der Regel Ihre Aussage verallgemeinern möchten.

Hat man also “nur” eine Stichprobe, keine Vollerhebung, so stellt sich die Frage, wie aussagekräftig eine Stichprobe ist. Oder anders gesagt: Wie ähnlich sich die Statistiken vieler Stichproben wären und wie ähnlich diese Statistiken zum Parameter sind. Untersuchen wir diese Fragen an einem Beispiel.

Nehmen wir an, Sie beraten einen großen deutschen Flughafen in der Entstehung. Leider gab es in der Vergangenheit einige Probleme beim Bau und so sind Zeit und Kosten massiv etwas über den veranschlagten Rahmen hinausgeschossen... Jetzt wurde eine findige Unternehmensberatung beauftragt, den Karren aus dem Dreck zu ziehen, und Sie sind das Masterbrain jener Beratung. Eine erste Aufgabe für Sie ist es, Flugaufkommen anderer Flughäfen zu analysieren, um Anhantspunkte für das Mengengerüst für “Ihren” Flughafen zu bekommen. Auch die zu erwartenden Verspätungen interessieren die Projektleitung. Dazu analysieren Sie die Abflüge der New Yorker Flughäfen im Jahre 2013. Komfortablerweise haben wir die Grundgesamtheit aller Flüge von New York, eine Vollerhebung also. Um das Verhalten von Stichproben zu untersuchen, ziehen wir eine bzw. einige Stichproben der Größe $n = 30$ und schauen, wie zuverlässig verschiedene Statistiken sind.

```
library(nycflights13)
data(flights)

set.seed(1234567)
```

```
flights_sample <- flights %>%
  na.omit %>%
  sample_n(size = 30)
```

Zur Erinnerung: In der Praxis haben wir (fast) immer nur eine Stichprobe, nicht den Luxus einer Vollerhebung. Wir könnten jetzt also relevante Verspätungswerte berechnen und dem Kunden mitteilen:

```
flights_sample %>%
  select(arr_delay) %>%
  summarise_all(funs(min, max, median, mean, sd, IQR))
#> # A tibble: 1 x 6
#>   min   max median  mean    sd    IQR
#>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1 -28   275    6.5  28.1   65   57.2
```

Laut unseren Daten liegt der Mittelwert der Verspätung bei einer knappen halben Stunde; der Median aber nur bei 6.5 Minuten. Die maximale Verspätung beläuft sich auf etwa 4.5 (275/6) Stunden. Möchte der Flughafen also konservativ planen, sollte er sich auf 4.5 Stunden Verspätung im schlimmsten Fall einstellen (?). Schauen wir jetzt also, wie gut die Verspätungen der Flüge (`arr_delay`) in der Stichprobe - genauer einige Statistiken dieser Stichprobe - der Grundgesamtheit aller Flüge (aus 2013, von New York) entsprechen.

```
flights %>%
  select(arr_delay) %>%
  na.omit %>%
  summarise_all(funs(min, max, median, mean, sd, IQR))
#> # A tibble: 1 x 6
#>   min   max median  mean    sd    IQR
#>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1 -86   1272    -5   6.9  44.6   31
```

Der Median der Stiprobe entspricht dem Populationswert recht gut; beim Mittelwert sieht es schon schlechter aus und beim Maximalwert der Verspätung liegen viele Stunden dazwischen!

Eine andere Idee wäre, sich die die 5% Flüge mit der größten Verspätung anzuschauen und diesen Wert als zu erwartende Verspätung anzunehmen. Mit `percent_rank` können wir die Werte von `arr_delay` in Prozentränge umwandeln:

```
flights_sample %>%
  select(arr_delay) %>%
  mutate(delay_ranking = percent_rank(arr_delay)) %>%
```

```

arrange(-delay_ranking)
#> # A tibble: 30 x 2
#>   arr_delay delay_ranking
#>   <dbl>        <dbl>
#> 1     275      1.000
#> 2     167      0.966
#> 3     115      0.931
#> 4      97      0.897
#> 5      66      0.862
#> 6      62      0.828
#> 7      50      0.793
#> 8      49      0.759
#> 9      34      0.724
#> 10     27      0.690
#> # ... with 20 more rows

```

Demnach kommen wir auf ca. 160 Minuten Verspätung; man könnte also argumentieren, dass wir uns auf “normale” Verspätung vorbereiten, aber die 5% extremsten Verspätung als höhere Macht hinnehmen und demnach Vorkehrungen für höchstens 160 Minuten Verspätung treffen. Vergleichen wir wieder die Statistik mit dem Wert der Population. `qdata` gibt das Quantil für ein Perzentil eines Datensatzes (hier 95) zurück.

```

qdata(~arr_delay, p = .95, data = flights)
#>       p quantile
#> 0.95    91.00

```

Das ist eine Stunde *weniger* als in unserer Stichprobe. Ist eine Stunde Abweichung “schlimm”? Das ist keine Frage der Statistik, sondern eine Frage des Sachgegenstands (Sie könnten sagen, dass muss der Kunde entscheiden). Eine andere Frage, und zwar eine der Statistik ist: Wie groß sollte die Stichprobe sein, um eine vom Kunden gewünschte Genauigkeit zu erreichen? Das ist der Punkt, wo die Inferenzstatistik ins Spiel kommt.

15.2 Die Stichprobenverteilung

Wie wir oben gesehen haben, ist es keineswegs sicher, dass eine Statistik dem Parameter (d.h. dem analogen Kennwert in der Population) ähnlich ist. Wir brauchen also einen Kennwert, der uns sagt *wie ähnlich* eine Statistik einem Parameter wohl ist. Der *Standardfehler* und die *Stichprobenverteilung* sind zwei Konzepte dazu.

Leider wissen wir in der Praxis nicht den “echten” Wert (z.B. die wirkliche Verspätung über alle Flüge), wir haben nur eine einsame Stichprobe. Würden wir uns die Mühe machen, viele Stichproben zu ziehen (Sie hören schon den Kunden schreien wegen der Kosten, aber

mittlerweile würde es darauf auch nicht mehr ankommen), dann könnten wir schauen, wie *stabil* die Stichprobenkennwerte sind: Sind sich die Stichproben im relevanten Kennwert ähnlich, so ist das ein Indiz dafür, dass sie den “wahren Wert” gut schätzen. Hier haben wir den Luxus, die Population zu kennen, deswegen ziehen wir zur Demonstration einmal viele (z.B. 500; `n_samples = 500`) Stichproben. Dazu fassen wir das Stichprobenziehen, das Berechnen der mittleren Verspätung und dem Herausziehen (`pull`) dieses Vektors aus dem Dataframe in einer Funktion zusammen (`sample_fun`)¹; diese Funktion wiederholen wir dann 500 Mal mit `replicate`. Mit `favstats` aus `mosaic` kann man sich bequem einige zentrale Statistiken anzeigen lassen. Die Verteilung lässt sich auf üblichem Wege plotten, nur dass wir noch aus dem Vektor `viele_stipros` einen Dataframe machen müssen, das `ggplot` nur mit Dataframes redet (vgl. Abbildung 15.2).

```
n_samples <- 50

sample_fun <- function(sample_size){
  flights %>%
    select(arr_delay) %>%
    na.omit %>%
    sample_n(size = sample_size) %>%
    summarise(delay_mean = mean(arr_delay)) %>%
    pull(delay_mean)
}

viele_stipros <- replicate(n = n_samples,
                           expr = sample_fun(sample_size = 30))
head(viele_stipros)
#> [1] -6.17 12.33  7.63 -6.13  3.20 10.03
favstats(viele_stipros)
#>   min   Q1 median   Q3 max mean   sd n missing
#> -10.2  2.08   6.37  9.74 25.5 6.34 7.79 50       0
```

```
data_frame(n_30 = viele_stipros,
           n_100 = viele_stipros100) %>%
  gather(key = sample_size, value = arr_delay_mean) %>%
  mutate(sample_size = factor(sample_size,
                               levels = c("n_30", "n_100"))) %>%
  ggplot +
  aes(x = arr_delay_mean) +
  geom_histogram() +
  facet_wrap(~sample_size)
```

¹diese Funktion ist nicht sehr effizient. Z.B. löschen wir in jedem Durchgang die fehlenden Werte; einmal würde reichen

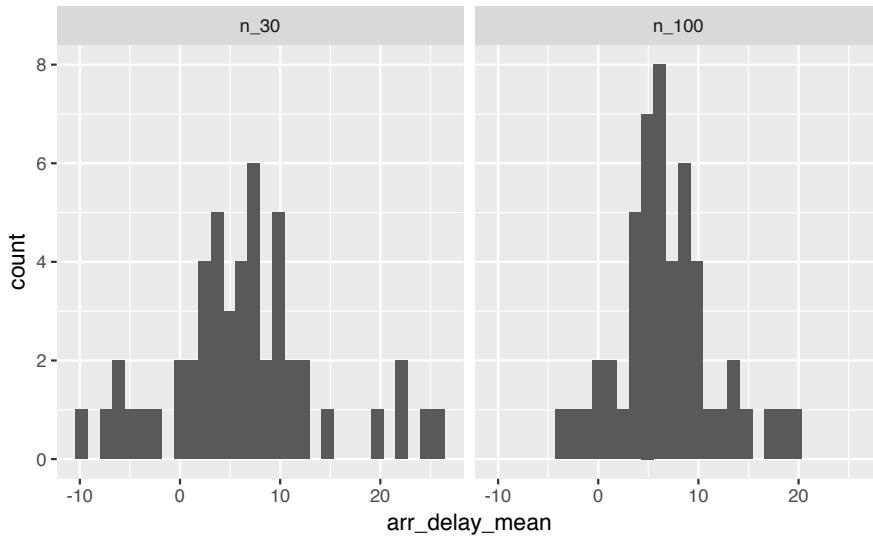


Abbildung 15.2: Die grobe Stichprobenverteilung von arrdelays

Die *Stichprobenverteilung* ist also die *Menge aller möglichen Stichproben aus einer Population*; mit 500 Stichproben haben wir nur eine Näherung, aber das ist meist ausreichend². Eine Stichprobenverteilung ändert sich, wenn man die Stichprobengröße ändert (hier $n = 30$); probieren Sie das mal aus (z.B. $n = 10$ oder $n = 100$).

Die *Streuung (SD) der Stichprobenverteilung* nennt man den *Standardfehler* (abgekürzt *SE*). Der Standardfehler ist ein Maß für die Zuverlässigkeit (Schätzgenauigkeit, Stabilität, Robustheit) unseres Kennwerts; hier lag die SD - d.h. die SE - bei gut 8 Minuten.

Der Standardfehler zeigt, wie sehr sich Stichprobenkennwerte ähneln, wenn man viele Stichproben zieht. Er wird oft als Maß für die Schätzgenauigkeit des Mittelwerts verstanden.

Um die Genauigkeit anzugeben, mit der ein Stichprobenkennwert (z.B. der Mittelwert \bar{X} ; ein “Schätzer”) einen Populationswert schätzt, wird häufig der Bereich von $\bar{X} - SE$ bis $\bar{X} + SE$ angegeben (kürzer: $\bar{X} \pm SE$). Diesen Bereich nennt man auch ein *Konfidenzintervall*. Allerdings wird das Konfidenzintervall meist für $\bar{X} \pm 2SE$ angegeben; der Bereich, der bei einer Normalverteilung ca. 95% der Werte abdeckt.

Vergleichen Sie den Mittelwert der Stichprobenverteilung mit dem Mittelwert der Population. Wie man sieht, liegen die Zahlen eng beieinander. Der Mittelwert der Stichprobenverteilung schätzt den Mittelwert der Population ohne Verzerrung³ und umso genauer, je größer die Stichprobe ist. Der Median schneidet hier ähnlich gut ab.

Halten wir fest: Mittelwert und Median der Population wurden von der Stichprobenverteilung

²theoretisch spitzfindig sind wir natürlich von unendlich vielen Stichproben noch unendlich weit entfernt

³wenn einige Voraussetzungen erfüllt sind

gut geschätzt, die Extremwerte hingegen nicht. Schauen wir uns mal an, wie sehr das Quantil für 95% streut.

```
sample_fun_quantil <- function(sample_size){
  flights %>%
    select(arr_delay) %>%
    na.omit %>%
    sample_n(size = sample_size) %>%
    summarise(delay_95q = qdata(~arr_delay,
                                 p = .95, data = .)[2]) %>%
    pull(delay_95q)
}

viele_stipros3 <- replicate(n = n_samples,
                           expr = sample_fun_quantil(sample_size = 30))
head(viele_stipros3)
#> quantile
#>   44.9      50.1      81.5      93.0      81.1      62.7
favstats(viele_stipros3)
#>   min   Q1 median   Q3 max mean   sd n missing
#> 23.5 47.4  69.8 111 209 84.3 48.6 50       0
qdata(~arr_delay, p = .95, data = flights)
#>   p quantile
#> 0.95     91.00
```

Da die Funktion `qdata` einen Vektor der Länge 2 zurückliefert, aber `summarise` nur einen Vektor der Länge 1 (also eine einzelne Zahl) verkraftet, wählen wir mit `[2]` nur den zweiten Wert aus; am ersten Platz steht der gewählte Prozentrang, im zweiten das Quantil.

In der Population liegt das 95%-Quantil für Verspätung bei 91 Minuten. In unserer Simulation liegt der Mittelwert bei 80 Minuten und die SD bei 43 Minuten - das ist sehr hoch. Probieren Sie mal aus, um wieviel sich die die SD verringert, wenn Sie die Stichprobe vergrößern ($n = 100$)⁴. Zwar verringert sich die Variabilität (die Werte für das 95%-Quantil gleichen sich an), aber die SD bleibt noch vergleichsweise hoch.

Natürlich kennt man die Population nicht in der Praxis; der Zweck der Übung war das Verhalten von Stichprobenkennwerten zu studieren. Aber was man in der Praxis noch tun kann (und häufig tut), ist eine Annahme über die Population zu treffen (z.B. normalverteilt mit Mittelwert Null), und auf dieser Basis den Standardfehler zu berechnen. Dann erhält man also den Standardfehler für die angenommene Population, das wird zumeist als H_0 bezeichnet. Dazu später mehr.

⁴viele_stipros4 <- replicate(n = 500, expr = sample_fun_quantil(sample_size = 100))

15.3 Der Bootstrap

Wie gerade erläutert, haben wir in der Praxis meist nur eine einzige, einsame Stichproben und keine (sicheren) Werte zu Populationsparametern. Ohne Stichprobenverteilung kein Standardfehler. Der Standardfehler ist, wie oben erörtert, eine gebräuchliche Methode um die Genauigkeit der Schätzung des Populationswertes. Mit einem Trick, bekannt als *Bootstrap* kann man sich dennoch eine Stichprobenverteilung basteln - und damit einen Standardfehler bekommen. Die Idee ist, die Stichprobe als Population zu betrachten: Wir ziehen viele Stichproben aus der Stichprobe (mit Ziehen mit Zurücklegen: nach Entnahme eines Fluges legen wir ihn wieder zurück in die Stichprobe, so bleibt unsere Original-Stichprobe immer gleich groß). Dieses Verfahren nennt man auch *Resampling*. Ein kleines Beispiel: Ziehen wir eine Stichprobe mit $n = 3$ aus der Population; eine normale Stichprobe *ohne* Zurücklegen (`replace = FALSE`). Übrigens: Bei der Funktion `sample_n` ist `replace = TRUE` die Voreinstellung (default); führt man das Argument `replace` nicht an, so geht die Funktion von der Voreinstellung aus.

```
echte_kleine_stichprobe <- sample_n(flights,
                                      size = 3, replace = FALSE)
```

```
#> # A tibble: 3 x 5
#>   year month   day dep_time sched_dep_time
#>   <int> <int> <int>    <int>        <int>
#> 1  2013     8     1      756        800
#> 2  2013     4     1     1307       1230
#> 3  2013     7     7     1510       1515
```

Jetzt ziehen wir aus dieser kleinen Stichprobe eine *neue* Stichprobe der gleichen Größe, dieses Mal *mit* Zurücklegen. Voilà: eine Bootstrap-Stichprobe.

```
bootstrap_sample <- sample_n(flights, size = 3, replace = TRUE)
```

```
#> # A tibble: 3 x 5
#>   year month   day dep_time sched_dep_time
#>   <int> <int> <int>    <int>        <int>
#> 1  2013     7     7      1510        1515
#> 2  2013     7     7      1510        1515
#> 3  2013     4     1     1307       1230
```

In dieser Stichprobe wurde der Flug mit Abflugzeit 855 zwei Mal gezogen. Es hätte auch anders ausgehen können. Probieren Sie das ein paar Mal aus. Ahnen Sie, was kommt? Wir wiederholen diese Stichprobenziehung, genauso wie wir das oben mit der Population getan haben:

```
viele_bootstrap_samples <- replicate(n_samples, sample_fun(sample_size = 30))
favstats(viele_bootstrap_samples)
#>   min   Q1 median   Q3 max mean   sd n missing
#> -0.667 2.06   7.38 11.3 30.8 8.12 7.28 50      0
```

Die Variabilität (Unterschiedlichkeit, Streuung) des Stichprobenkennwerts ist ein Maß für die Robustheit des Kennwerts bzw. für die Genauigkeit der Schätzung des Populationswertes. Aber wissen wir, wie gut unser Bootstrap funktioniert hat? Vergleichen wir unsere Bootstrap-Werte (vor allem die SD der “gebootstrapten” Stichprobenverteilung) mit einer normalen Stichprobenverteilung von “echten” Stichproben.

```
favstats(viele_stipros)
#>   min   Q1 median   Q3 max mean   sd n missing
#> -10.2 2.08   6.37 9.74 25.5 6.34 7.79 50      0
```

Erstaunlich! Der Standardfehler der Bootstrap-Verteilung - das Maß der Schätzgenauigkeit von `arr_delay` ist mit 9.98 nah am Standardfehler aus der Verteilung der Stichproben (10.29), die wir aus der Population gezogen haben; beide liegen etwa bei 10 Sekunden.

Für einigermaßen große Stichproben ist der Bootstrap - bzw. der Bootstrap-Standardfehler - ein guter Schätzer für den “echten” Standardfehler (Efron und Tibshirani 1994).

Jetzt können wir eine Aussage zur Genauigkeit unseres Schätzers (des Stichprobenkennwerts) treffen. Dazu nehmen wir das 94 95%-Konfidenzintervall, weil man das so gerne tut:

```
qdata(viele_bootstrap_samples, p = .025)
#>   p quantile
#> 0.025 -0.237
qdata(viele_bootstrap_samples, p = .975)
#>   p quantile
#> 0.975 27.732
```

Wir schneiden also von der Verteilung links und rechts jeweils 2.5% ab, die verbleibenden, inneren 95% markieren das 95%-Konfidenzintervall (s. Abbildung ??).

15.4 Nyllhypotesen auf Signifikanz testen

Anstelle eines Konfidenzintervalls, wird häufig eine Hypothese getestet, wie “Die mittlere Verspätung \bar{X} beträgt null Minuten”. Der zentrale Gedanke dabei ist “wenn ich meine Stichprobenverteilung betrachte, wie häufig kommt mein Stichprobenergebnis \bar{X} darin vor?”. Etwas ausführlicher, ist der Ablauf in etwa so (vgl. Abbildung 15.3).

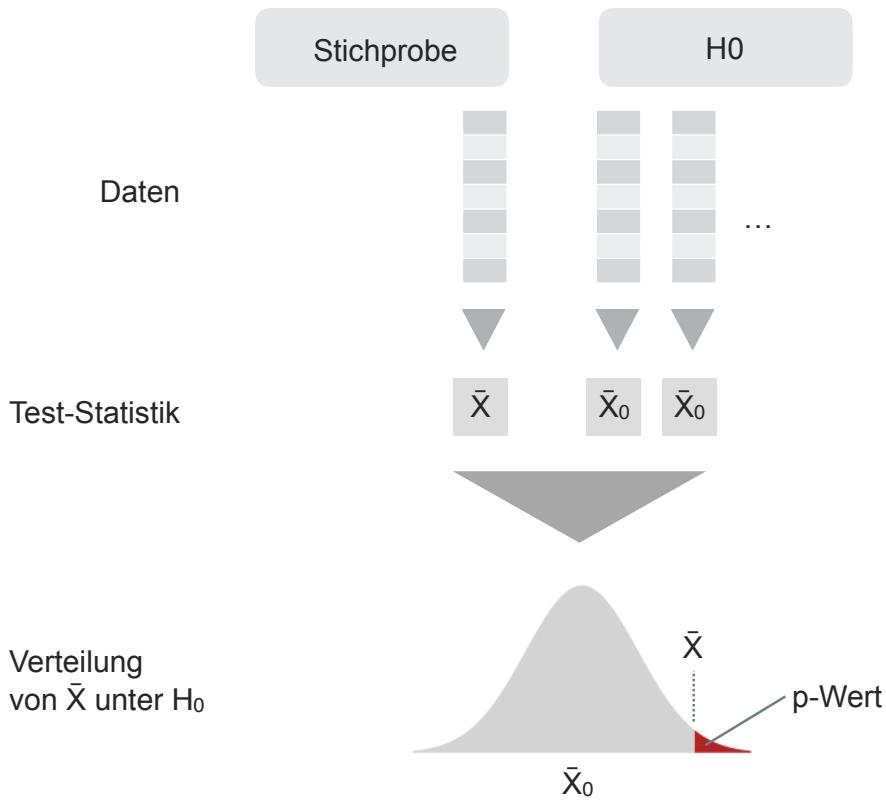


Abbildung 15.3: Nullhypotesen auf Signifikanz testen

1. Berechne einen Stichprobenkennwert, der getestet werden soll (z.B. die mittlere Verspätung); diesen Kennwert nennt man auch die *Test-Statistik*, häufig ist es ein Mittelwert, \bar{X} .
2. Definiere eine Nullhypothese H_0 ; die H_0 nimmt an, dass es keinen Effekt gibt, z.B. dass es keine Verspätung (im Mittel) gibt.
3. Ziehe viele Stichproben aus der Verteilung, die die H_0 vorgibt und berechne jeweils die Test-Statistik, z.B. \bar{X}_0 .
4. Zähle den Anteil der H_0 -Stichproben, deren Test-Statistik (\bar{X}_0) *extremer* als die echte, empirische Teststatistik \bar{X} . Diesen Wert nennt man p-Wert.

Soviel zur Theorie. Probieren wir das einmal aus:

1. Den Verspätungswert unserer echten, empirischen Stichprobe haben wir oben schon einmal bestimmt:

```
flights_sample %>%
  summarise(mean(arr_delay),
            sd(arr_delay))
#> # A tibble: 1 x 2
#>   `mean(arr_delay)` `sd(arr_delay)`
```

```
#>          <dbl>        <dbl>
#> 1       28.1         65
nrow(flights_sample)
#> [1] 30
```

2. H_0 : "Die mittlere Verspätung \bar{X} beträgt null Minuten".
3. Wir ziehen viele Stichproben aus einer Normalverteilung mit Mittelwert = 0. Das leistet die Funktion `rnorm`. Außerdem brauchen wir noch eine Annahme über die Streuung. Der Einfachheit halber nehmen wir die Streuung der Stichprobe als Schätzwert für die Streuung in der Population⁵. Das Ergebnis ist in Abb. 15.4 dargestellt.

```
sample_fun_rnorm <- function(sample_size){
  rnorm(n = sample_size,
        sd = sd(flights_sample$arr_delay),
        mean = 0) %>% mean -> delay_mean_H0
  return(delay_mean_H0)
}

viele_stipros_H0 <- replicate(n = n_samples, sample_fun_rnorm(sample_size = 30))
head(viele_stipros_H0)
#> [1] -3.07 -15.36 -5.47  5.75  1.78 -9.75
favstats(viele_stipros_H0)
#>   min    Q1 median    Q3 max   mean    sd  n missing
#> -28 -9.38  0.458 7.38 20.8 -1.66 11.9  50       0
```

15.5 Der p-Wert - Nutzen und Grenzen

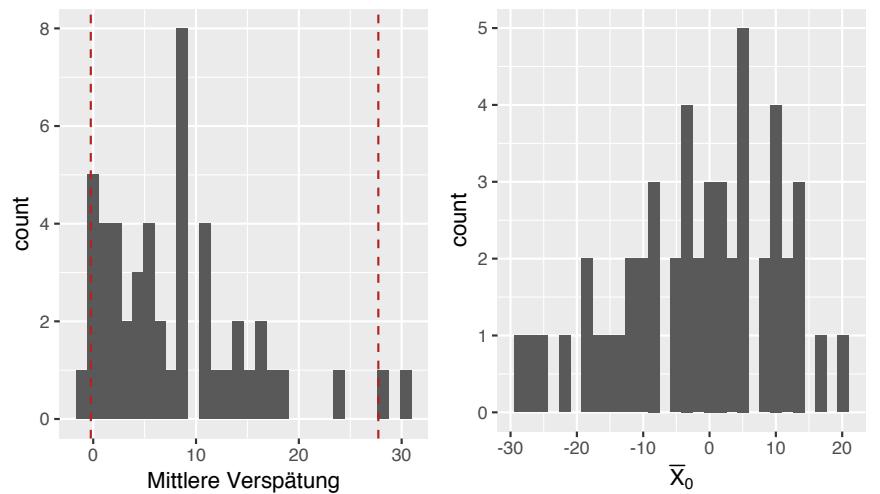
15.5.1 Was sagt der p-Wert?

Der p-Wert, entwickelt von Sir Ronald Fisher (Abb. 15.5), ist die heilige Kuh vieler Forschenden. Das ist nicht normativ, sondern deskriptiv gemeint. Der p-Wert entscheidet (häufig) darüber, was publiziert wird, und damit, was als Wissenschaft sichtbar ist - und damit, was Wissenschaft ist (wiederum deskriptiv, nicht normativ gemeint). Kurz: Dem p-Wert kommt viel Bedeutung zu bzw. ihm wird viel Bedeutung zugemessen (vgl. Abb. 15.6).

Der p-Wert ist der tragende Ziegelstein in einem Theoriegebäude, das als *Nullhypothesen-Signifikanztesten* (NHST⁶) bezeichnet wird. Oder kurz als 'Inferenzstatistik' bezeichnet. Was sagt uns der p-Wert? Eine gute intuitive Definition ist:

⁵das ist nicht der beste Weg, aber ein einfacher

⁶Der Term 'Signifikanz-Hypothesen-Inferenz-Testen' hat sich nicht durchgesetzt



Stichprobenverteilung aus der H_0 , der Anteil der rot markierten Fläche zur Gesam

Abbildung 15.4: Simulation von Verteilungen

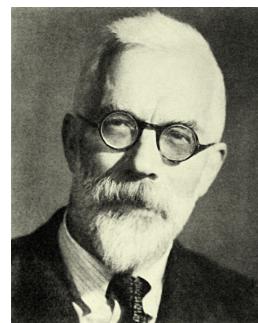


Abbildung 15.5: Der größte Statistiker des 20. Jahrhunderts ($p < .05$)

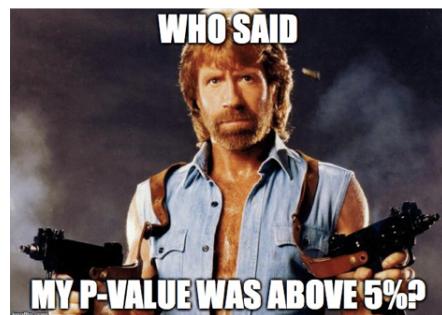


Abbildung 15.6: Der p-Wert wird oft als wichtig erachtet

Der p-Wert sagt, wie gut die Daten zur Nullhypothese passen; wie “plausibel” die Daten für die Hypothese sind.

Die (genauere) Definition des p-Werts ist kompliziert; man kann sie leicht missverstehen:

Der p-Wert - $P(D|H)$ - gibt die Wahrscheinlichkeit P unserer Daten D an (und noch extremerer), unter der Annahme, dass die getestete Hypothese H wahr ist (und wenn wir den Versuch unendlich oft wiederholen würden, unter identischen Bedingungen und ansonsten zufällig).

Mit anderen Worten: Je *größer* p , desto *besser* passen die Daten zur *Nullhypothese*. Mit Nullhypothese (H_0) bezeichnet man die getestete Hypothese. Der Name Nullhypothese röhrt vom Begriff ‘nullifizieren’ (verwerfen) her, da (nach dem Falsifikationismus) eine These immer nur verworfen, nie bestätigt werden kann. Da viele die eigene Hypothese nur ungern verwerfen wollen, wird die ‘gegnerische Hypothese’, die man loswerden will, getestet. Fällt p unter die magische Zahl von 5%, so proklamiert man Erfolg (*Signifikanz*) und verwirft die H_0 .

Der p-Wert ist weit verbreitet. Er bietet die Möglichkeit, relativ objektiv zu quantifizieren, wie gut ein Kennwert, mindestens so extrem wie der aktuell vorliegende, zu einer Hypothese passt. Allerdings hat der p-Wert seine Probleme. Vor allem: Er wird missverstanden. Jetzt kann man sagen, dass es dem p-Wert (dem armen) nicht anzulasten, dass andere/ einige ihn missverstehen. Auf der anderen Seite finde ich, dass sich Technologien dem Nutzer anpassen sollten (soweit als möglich) und nicht umgekehrt.

Viele Menschen - Statistik-Dozenten - haben Probleme mit dieser Definition (Gigerenzer 2004). Das ist nicht deren Schuld: Die Definition ist kompliziert. Vielleicht denken viele, der p-Wert sage das, was tatsächlich interessant ist: die Wahrscheinlichkeit der (getesteten) Hypothese H , gegeben der Tatsache, dass bestimmte Daten D vorliegen. Leider ist das *nicht* die Definition des p-Werts. Also:

$$P(D|H) \neq P(H|D)$$

Formeln haben die merkwürdige Angewohnheit vor dem inneren Auge zu verschwinden; Bilder sind für viele Menschen klarer, scheint's. Übersetzen wir die obige Formel in folgenden Satz:

Wahrscheinlichkeit, Mann zu sein, wenn man Papst ist UNGLEICH zur Wahrscheinlichkeit, Papst zu sein, wenn man Mann ist.

Oder kürzer:

$$P(M|P) \neq P(P|M)$$

Das Bild (Abb. 15.7) zeigt den Anteil der Männer an den Päpsten (sehr hoch). Und es zeigt den Anteil der Päpsten von allen Männern (sehr gering). Dabei können wir uns Anteil mit Wahrscheinlichkeit übersetzen. Kurz: Die beiden Anteile (Wahrscheinlichkeiten) sind nicht gleich. Man denkt leicht, der p-Wert sei die *Wahrscheinlichkeit, Papst zu sein, wenn man*

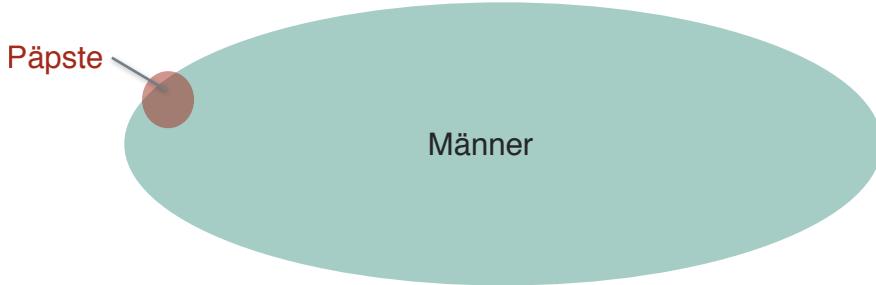


Abbildung 15.7: Mann und Papst zu sein ist nicht das gleiche.

Mann ist. Das ist falsch. Der p-Wert ist die *Wahrscheinlichkeit, Papst zu sein, wenn man Mann ist.* Das ist ein Unterschied.

Die amerikanische Gesellschaft für Statistik⁷ hat vor einiger Zeit einige Richtlinien zur Interpretation von p-Werten voröffentlich, vor dem Hintergrund der regen oder erregten Diskussion (Wasserstein und Lazar 2016):

1. P-Werte zeigen an, wie kompatibel Daten mit einer getesteten statistischen Hypothese sind.
2. P-Werte messen nicht die Wahrscheinlichkeit einer getesteten Hypothese und auch nicht die Wahrscheinlichkeit, dass die Daten durch puren Zufall zu erklären sind.
3. Wissenschaftliche Schlüsse oder Entscheidungen in der Praxis (Geschäftswelt, Politik) sollten nicht ausschließlich auf der Frage abzielen, ob der p-Wert unter einer bestimmten Schwelle liegt.
4. Angemessene statistische Inferenz verlangt stets lückenlose Berichte und Transparenz in der Analyse.
5. Ein p-Wert bzw. das Vorliegen von statistischer Signifikanz ist kein Maß für die Stärke eines Effekts oder die Bedeutung (Relevanz) eines Ergebnisses.
6. Der p-Wert für sich genommen ist kein gutes Maß für die Evidenz eines Modells oder einer Hypothese.

15.5.2 Der p-Wert ist eine Funktion der Stichprobengröße

Der p-Wert ist für weitere Dinge kritisiert worden (Wagenmakers 2007, Briggs (2016)); z.B. dass die “5%-Hürde” einen zu schwachen Test für die getestete Hypothese bedeutet. Letzterer Kritikpunkt ist aber nicht dem p-Wert anzulasten, denn dieses Kriterium ist beliebig, könnte konservativer gesetzt werden und jegliche mechanisierte Entscheidungsmethode kann ausgenutzt werden. Ähnliches kann man zum Thema “P-Hacking” argumentieren (Head u. a. 2015, Wicherts u. a. (2016)): andere statistische Verfahren können auch gehackt werden.

⁷ASA: <http://www.amstat.org/>

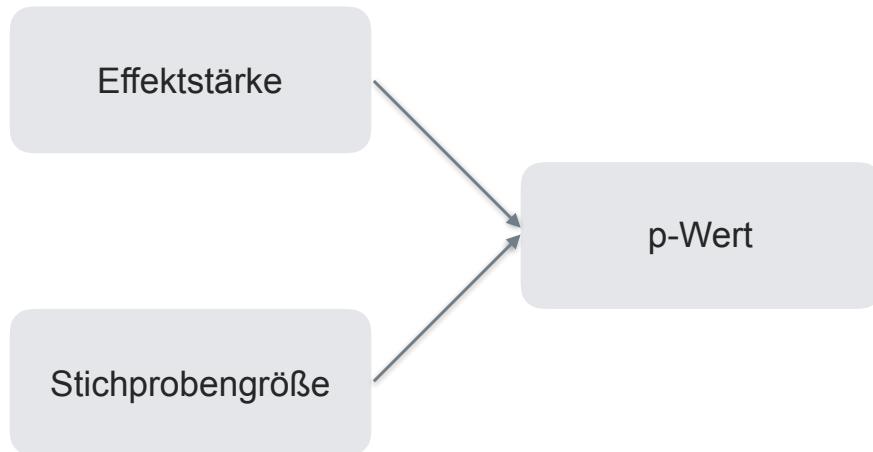


Abbildung 15.8: Zwei Haupteinflüsse auf den p-Wert

“Hacken” soll hier sagen, dass man - Kreativität und Wille vorausgesetzt - immer Wege finden kann, um einen Kennwert in die gewünschte Richtung zu drängen.

Ein anderer Anklagepunkt lautet, dass der p-Wert nicht nur eine Funktion der Effektgröße sei, sondern auch der Stichprobengröße. Sprich: Bei großen Stichproben wird jede Hypothese signifikant. Das ist richtig. Das schränkt die praktische Nützlichkeit ein (vgl. Abb. 15.8. Die Details der Simulation, die hinter Abb. 15.8 sind etwas umfangreicher und hier nicht so wichtig, daher nicht angegeben⁸.

Egal wie klein die Effektstärke ist, es existiert eine Stichprobengröße, die diesen Effekt beliebig signifikant werden lässt.

Die Verteidigung argumentiert hier, dass das “kein Bug, sondern ein Feature” sei: Wenn man z.B. die Hypothese prüfe, dass der Gewichtsunterschied zwischen Männern und Frauen 0,00000000kg sei und man findet 0,000000123kg Unterschied, ist die getestete Hypothese falsch. Punkt. Der p-Wert gibt demnach das korrekte Ergebnis. Meiner Ansicht nach ist die Antwort zwar richtig, geht aber an den Anforderungen der Praxis vorbei.

15.5.3 Mythen zum p-Wert

Falsche Lehrmeinungen sterben erst aus, wenn die beteiligten Professoren in Rente gehen, heißt es. Jedenfalls halten sich eine Reihe von Mythen hartnäckig; sie sind alle falsch.

Wenn der p-Wert kleiner als 5% ist, dann ist meine Hypothese (H_1) sicher richtig.

Falsch. Richtig ist: “Wenn der p-Wert kleiner ist als 5% (oder allgemeiner: kleiner als α , dann sind die Daten (oder noch extremer) unwahrscheinlich, vorausgesetzt die H_0 gilt”.

Wenn der p-Wert kleiner als 5% ist, dann ist meine Hypothese (H_1) höchstwahrscheinlich richtig.

⁸s. hier für Details: https://sebastiansauer.github.io/pvalue_sample_size/

Falsch. Richtig ist: Wenn der p-Wert kleiner ist als *alpha*, dann sind die Daten unwahrscheinlich, *falls* die H₀ gilt. Ansonsten (wenn H₀ nicht gilt) können die Daten sehr wahrscheinlich sein.

Wenn der p-Wert kleiner als 5% ist, dann ist die Wahrscheinlichkeit der H₀ kleiner als 5%.

Falsch. Der p-Wert gibt *nicht* die Wahrscheinlichkeit einer Hypothese an. Richtig ist: Ist der p-Wert kleiner als 5%, dann sind meine Daten (oder noch extremere) unwahrscheinlich (<5%), *wenn* die H₀ gilt.

Wenn der p-Wert kleiner als 5% ist, dann habe ich die Ursache eines Phänomens gefunden.

Falsch. Richtig ist: Keine Statistik kann für sich genommen eine Ursache erkennen. Bestenfalls kann man sagen: hat man alle konkurrierenden Ursachen ausgeschlossen *und* sprechen die Daten für die Ursache *und* sind die Daten eine plausible Erklärung, so erscheint es der beste Schluss, anzunehmen, dass man *eine* Ursache gefunden hat - im Rahmen des Geltungsbereichs einer Studie.

Wenn der p-Wert kleiner als 5% ist, dann kann ich meine Studie veröffentlichen.

Richtig. Leider entscheidet zu oft (nur) der p-Wert über das Wohl und Wehe einer Studie. Wichtiger wäre zu prüfen, wie "gut" das Modell ist - wie präzise sind die Vorhersagen? Wie theoretisch befriedigend ist das Modell?

Wenn der p-Wert *größer* als 5% ist, dann ist das ein Beleg *für* die H₀.

Falsch. Richtig ist: Ein großer p-Wert ist ein Beleg, dass die Daten plausibel unter der H₀ sind. Wenn es draußen regnet, ist es plausibel, dass es Herbst ist. Das heißt aber nicht, dass andere Hypothesen nicht auch plausibel sind. Ein großer p-Wert ist also Abwesenheit von klarer Evidenz – *nicht* Anwesenheit von klarer Evidenz zugunsten der H₀. Schöner ausgedrückt: "No evidence of effect is not the same as evidence of no effect". Für die Wissenschaft ist das insofern ein großes Problem, als dass sich Zeitschriften weigern, nicht-signifikante Studien aufzunehmen: "Das ist eine unklare Befundlage. Kein Mehrwert." so die Haltung. Das führt dazu, dass die wissenschaftliche Literatur einer großen Verzerrung unterworfen ist.

15.6 Wann welcher Inferenztest?

In der Praxis ist es eine häufige Frage, wann man welchen statistischen Test verwenden soll. Bei Eid, Gollwitzer, und Schmitt (2010) findet man eine umfangreiche Tabelle dazu; auch online wird man schnell fündig (z.B. bei der Methodenberatung der Uni Zürich⁹ oder beim Ärzteblatt¹⁰, Prel u. a. (2010)).

⁹<http://www.methodenberatung.uzh.ch/de/datenanalyse.html>

¹⁰<https://www.aerzteblatt.de/archiv/74880/Auswahl-statistischer-Testverfahren>

Die folgende Auflistung gibt einen *kurzen* Überblick zu gebräuchlichen Verfahren. Entscheidungskriterium ist hier (etwas vereinfacht) das Skalenniveau der Variablen (unterschieden in Input- und Outputvariablen).

1. 2 nominale Variablen: χ^2 -Test - `chisq.test`
2. Output: 1 metrisch, Input: 1 dichotom: t-Test - `t.test`
3. Output: 1 oder mehr metrisch, 1 nominal: Varianzanalyse - `aov`
4. 2 metrische Variablen: Korrelation - `cor.test`
5. Output: 1 metrisch, Input: 1 oder mehr nominal oder metrisch: Regression - `lm`
6. Output: 1 ordinal, Input: 1 dichotom: Wilcoxon (Mann-Whitney-U-Test) - `wilcox.test`
7. Output: 1 ordinal, Input: 1 nominal: Kruskal-Wallis-Test - `kruskal.test`
8. 1 metrisch (Test auf Normalverteilung): Shapiro-Wilk-Test - `shapiro.test`
9. Output: 1 dichotom, Input 1 oder mehr nominal oder metrisch: logistische (klassifikatorische) Regression: `glm(..., family = "binomial")`
10. 2 ordinal: Spearmans Rangkorrelation - `cor.test(x, y, method = "spearman")`

15.7 Vertiefung: Beispiele für häufige Inferenztests

Schauen wir uns für jeden Test aus Kapitel 15.6 ein Anwendungsbeispiel an.

15.7.1 χ^2 -Test

Laden wir den Datensatz `extra`. Ob es wohl einen Zusammenhang gibt zwischen (der Anzahl von) Geschlecht und extremen Alkoholgenuss? Definieren wir ‘extrem’ durch mehr als 10 Kater.

Forschungsfrage: Gibt es einen Zusammenhang zwischen Geschlecht und extremen Alkoholgenuss?

Synonym wäre zu fragen, ob sich die Stufen von Geschlecht (die Geschlechter, also Mann und Frau) hinsichtlich Saufenextremen Alkoholgenuss unterscheiden.

```
extra <- read.csv("data/extra.csv")

extra$viel_saeufer <- extra$n_hangover > 10

chisq.test(x = extra$sex, extra$viel_saeufer)
#>
#> Pearson's Chi-squared test with Yates' continuity correction
#>
#> data: extra$sex and extra$viel_saeufer
#> X-squared = 30, df = 1, p-value = 4e-08
```

Achtung, falls Ihre Daten in aggregierter Form vorliegen, müssen Sie sie folgendermaßen übergeben werden:

```
table(x = extra$sex, extra$viel_saeufer) %>% chisq.test
#>
#> Pearson's Chi-squared test with Yates' continuity correction
#>
#> data: .
#> X-squared = 30, df = 1, p-value = 4e-08
```

15.7.2 t-Test

Forschungsfrage: Sind Männer im Schnitt extrovertierter als Frauen?

```
extra %>%
  filter(sex %in% c("Frau", "Mann")) %>%
  mutate(sex = factor(. $sex)) %>%
  t.test(extra_mean ~ sex, data = ., alternative = "less")
#> extra_mean ~ sex
#> <environment: 0x7fb0fd7cbf80>
#>
#> Welch Two Sample t-test
#>
#> data: extra_mean by sex
#> t = 1, df = 500, p-value = 0.9
#> alternative hypothesis: true difference in means is less than 0
#> 95 percent confidence interval:
#> -Inf 0.104
#> sample estimates:
#> mean in group Frau mean in group Mann
#> 2.91 2.86
```

Auf Deutsch liest sich der letzte Befehlsblock so:



Nimm den Datensatz **extra** UND DANN
 filtere nur Zeilen heraus, in denen bei Geschlecht ‘Mann’ oder ‘Frau’ steht (es gibt Zeilen mit ‘’’’ als Wert) UND DANN
 definiere **sex** als Faktor und zwar so, dass es nur Faktorstufen gibt, die es auch in den Daten gibt (Frau oder Mann) UND DANN führe einen gerichteten t-Test durch mit ‘extra_meanals Output-Variable undsex‘ als Gruppierungsvariable.

Der Punkt . meint hier den Datensatz in aktueller Form, so also, wie er aus der letzten (vorherigen) Zeile herausgekommen ist.

Hinweise:

- Der t-Test testet im Standard *ungerichtet*.
- Wird eine Gruppierungsvariable (wie Geschlecht) vom Typ **factor** angegeben, so muss diese 2 Faktorstufen haben. Allein durch filtern wird man zusätzliche Faktorstufen nicht los (im Gegensatz zu Variablen vom Typ **character**, Text). Man muss die Faktorvariable neu als Faktorvariable definieren. Dann werden nur die existierenden Werte als Faktorstufen herangezogen.
- Bei gerichteten Hypothesen sieht **t.test** zwei Möglichkeiten vor: **less** und **greater**. Woher weiß man, welches von beiden man nehmen muss? Die Antwort lautet: Bei Textvariablen sind die Stufen alphabetisch geordnet. R sagt also sozusagen: Frau ? Mann. Und für ? müssen wir das richtige Ungleichheitszeichen einsetzen (< oder >), so dass es unserer Hypothese entspricht. In diesem Fall glauben wir, dass Frauen weniger (bzw. Männer mehr) trinken, also haben wir **less** gewählt.
- Liegt der Datensatz nicht tidy vor, also gibt es z.B. eine Spalte mit Extraversionswerten für Männer und eine für Frauen, so darf man *nicht* die Formelsyntax (Kringel, Tilde “~”) nehmen, sondern benennt die Spalten mit X und Y: **t.text(x = df\$extra_maenner, y = df\$extra_frauen)**.

15.7.3 Varianzanalyse

Forschungsfrage: Unterscheiden sich Menschen mit unterschiedlich viel Kundenkontakt in ihrer Extraversion?

Der Kundenkontakt wurde mit einer Likertskaala gemessen, die mehrere Stufen von “weniger als einmal pro Quartal” bis “im Schnitt mehrfach pro Tag” reichte. Wir gehen nicht davon aus, dass diese Skala Intervallniveau aufweist. Obwohl Ordinalskalenniveau plausibel ist, bleiben wir bei der ANOVA (Varianzanalyse, AOV), die nur nominales Niveau ausschöpft. Beachten Sie, dass die entsprechende Variable **clients_freq** als Ganzzahl in R definiert ist, obwohl die Abstände nicht sicher gleich sind. Es ist zwar erlaubt, den Stufen einer nominalen Variablen Zahlen zuzuordnen, aber wir sollten nicht vergessen, dass die Zahlen “keine echten Zahlen” sind, also nicht metrisches Niveau aufweisen (zumindest ist das nicht sicher).

Die verschiedenen Stufen einer Variablen kann man sich so ausgeben lassen:

```
extra %>% distinct(clients_freq)
```

Jetzt die ANOVA:

```
aov(extra_mean ~ sex, data = extra) %>% glance
#>   r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC
```

```
#> 1 0.00247      0.00125 0.45      2.02  0.156 2  -505 1017 1031
#>   deviance df.residual
#> 1       165          813
```

`glance` räumt das Ergebnis der ANOVA etwas auf, so dass die Ausgabe ein Dataframe ist und die “Überblick-Koeffizienten” (daher ‘`glance`’, engl. ‘Blick’) ausgegeben werden. Ganz interessantes Ergebnis: statistisch signifikant ($p < .05$), aber R^2 ist sehr klein. Der F-Wert ist als `statistic` bezeichnet.

15.7.4 Korrelationen auf Signifikanz prüfen

Forschungsfrage: Ist der Extraversion-Mittelwert und die Anzahl der Facebook-Freunde korreliert?

Der Test prüft, ob diese Korrelation 0 ist¹¹.

```
cor.test(extra$extra_mean, extra$n_facebook_friends)
#>
#> Pearson's product-moment correlation
#>
#> data: x and y
#> t = 1, df = 700, p-value = 0.2
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#> -0.0303 0.1208
#> sample estimates:
#> cor
#> 0.0455
```

Man kann auch - wie beim t-Test - gerichtet testen mit der gleichen Syntax, vgl. `?cor.test`.

15.7.5 Regression

Forschungsfrage: Wie groß ist der Einfluss von der Anzahl von Parties auf die Anzahl der Kater?

```
lm(n_hangover ~ n_party, data = extra) %>% tidy
#>   term estimate std.error statistic p.value
#> 1 (Intercept) 0.159     1.401    0.113 9.10e-01
```

¹¹genauer gesagt prüft so ein Inferenztest, wie wahrscheinlich eine Datenlage ist, die mindestens so extrem ist wie unsere – unter der Annahme, dass die H₀ (zumeist: $r=0$) gilt

```
#> 2    n_party      0.539      0.054     9.983 3.62e-22
lm(n_hangover ~ n_party, data = extra) %>% glance
#>   r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC
#> 1     0.113          0.112   29.2      99.7 3.62e-22  2  -3752 7510 7524
#>   deviance df.residual
#> 1    665751         781
```

`statistic` ist bei dieser Ausgabe übrigens der F-Wert und `sigma` die SD der Residualstreuung.
`estimate` ist die Steigung der Regressionsgeraden.

15.7.6 Wilcoxon-Test

Forschungsfrage: Unterscheiden sich die Geschlechter in ihrer mittleren Extraversion?

Hier nehmen wir nicht an, dass Extraversion metrisch ist, sondern begnügen uns mit der schwächeren Annahme eines ordinalen Niveaus.

```
extra %>%
  filter(sex %in% c("Frau", "Mann")) %>%
  mutate(sex = factor(.sex)) %>%
  wilcox.test(extra_mean ~ sex, data = .)
#>
#> Wilcoxon rank sum test with continuity correction
#>
#> data: extra_mean by sex
#> W = 80000, p-value = 0.4
#> alternative hypothesis: true location shift is not equal to 0
```

15.7.7 Kruskal-Wallis-Test

Forschungsfrage: Unterscheiden sich Menschen mit unterschiedlich viel Kundenkontakt in ihrer Extraversion?

Genau wie beim Wilcoxon-Test gehen wir wieder nur von ordinalem Niveau bei Extraversion aus.

```
extra %>%
  filter(sex %in% c("Frau", "Mann")) %>%
  mutate(sex = factor(.sex)) %>%
  kruskal.test(extra_mean ~ sex, data = .)
#>
```

```
#> Kruskal-Wallis rank sum test
#>
#> data: extra_mean by sex
#> Kruskal-Wallis chi-squared = 0.6, df = 1, p-value = 0.4
```

15.7.8 Shapiro-Test

Forschungsfrage: Ist Extraversion normalverteilt?

Wahrscheinlich ist es sinnvoller, diese Frage mit einem Histogramm (oder QQ-Plot) zu beantworten, weil der Test bei großen Stichproben (zu) schnell signifikant wird. Aber machen wir es mal:

```
shapiro.test(extra$extra_mean)
#>
#> Shapiro-Wilk normality test
#>
#> data: extra$extra_mean
#> W = 1, p-value = 9e-09
```

Signifikant. Die Variable ist also *nicht* (exakt) normalverteilt. Böse Zungen behaupten, die Normalverteilung sei ungefähr so häufig wie Einhörner (Micceri 1989). Trotzdem setzen viele Verfahren sie voraus. Glücklicherweise reicht es häufig, wenn eine Variable *einigermaßen* normalverteilt ist (wobei es hier keine klaren Grenzen gibt).

15.7.9 Logistische Regression

Forschungsfrage: Kann man anhand der Extraversion vorhersagen, ob eine Person Extremtrinker ist?

```
glm(viel_saeufer ~ extra_mean, data = extra, family = "binomial") %>% tidy
#>           term estimate std.error statistic p.value
#> 1 (Intercept) -4.207     0.661      -6.37 1.93e-10
#> 2 extra_mean   0.942     0.218       4.33 1.51e-05
```

15.7.10 Spearmans Korrelation

Forschungsfrage: Ist die Extraversion assoziiert mit der Anzahl der Kundenbesuche?

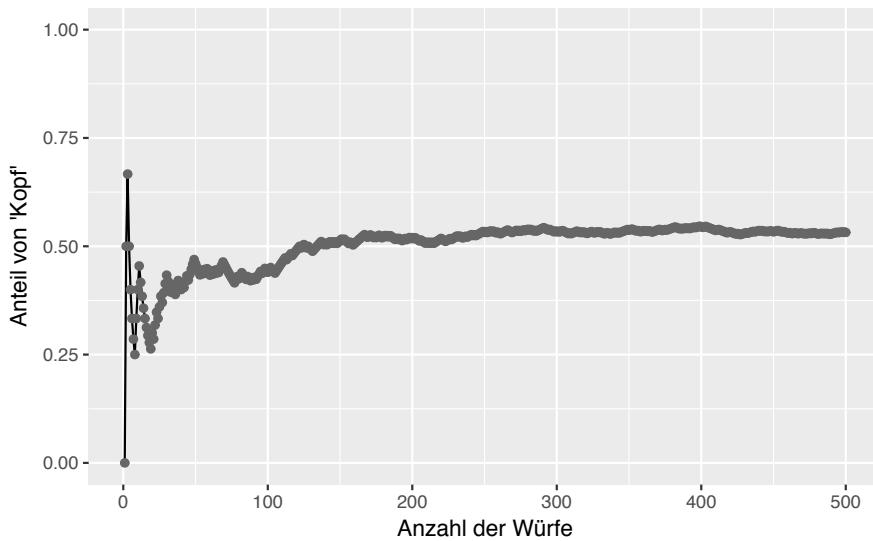


Abbildung 15.9: Anteil von 'Kopf' bei wiederholtem Münzwurf

```
cor.test(extra$extra_single_item, extra$clients_freq, method = "spearman")
```

15.8 Zur Philosophie des p-Werts: Frequentismus

Der p-Wert basiert auf der Idee, dass man ein Experiment *unendlich* oft wiederholen könnte (wer die Zeit hat, nicht wahr); und das unter *zufälligen* aber *ansonsten komplett gleichen* Bedingungen; das ist eine Kernidee des sog. ‘Frequentismus’ (Neyman und Pearson 1933). Diese Philosophie betrachtet Wahrscheinlichkeit als der Anteil, der sich bei unendlich häufiger Wiederholung eines Experiments ergibt. Ein Münzwurf hingegen ist das klassische Modell der frequentistischen Idee der Wahrscheinlichkeit (vgl. Abb. 15.9). Wirft man eine faire Münze oft, so nähert sich der relative Anteil von ‘Kopf’ an 50% an.

Ob es im Universum irgendetwas gibt, das unendlich ist, ist streitbar (Rucker 2004, Briggs (2016)). Jedenfalls ist die Vorstellung, das Experiment unendlich oft zu wiederholen, unrealistisch. Inwieweit Zufälligkeit und Vergleichbarkeit hergestellt werden kann, ist auch fragwürdig (Briggs 2016).

Die frequentistische Idee der Wahrscheinlichkeit darf Aussagen wie dieser keine Wahrscheinlichkeit zuweisen: “5 von 10 Marsianer trinken gerne Bier und Schorsch ist Marsianer” (Briggs 2016; Neyman und Pearson 1992; Neyman und Pearson 1933). Häufigkeitsaussagen a la Frequentismus machen hier offenbar wenig Sinn. Trotzdem fühlen sich manche unter uns geneigt, die Wahrscheinlichkeit, dass Schorsch der Marsianer gern Bier trinkt, auf 50% zu bemessen. Ein anderes, weniger fernes Beispiel: Ich werfe eine Münze hoch, fange sie auf, verdeckt. Wie hoch ist die Wahrscheinlichkeit, dass die Münze mit Kopf nach oben liegt? 50%? Moment, einzelne Ereignisse haben keine Wahrscheinlichkeit, sagt der Frequentismus. Wer sich geneigt fühlt (wie ich), hier doch eine Wahrscheinlichkeit zuzuordnen (50%), der tut

dies offenbar nicht auf Basis des Frequentismus. Eine theoretische Position, die Wahrscheinlichkeiten erlaubt, kann man als *epistemologische Wahrscheinlichkeit* bezeichnen (Briggs 2016): Alle möglichen von n Ergebnissen erscheinen uns gleich plausibel. Daher schließen wir, dass die Wahrscheinlichkeit des Ereignisses k 1 durch n ($1/n$) beträgt.

15.9 Alternativen zum p-Wert

Eine Reihe von Alternativen (oder Ergänzungen zum p-Wert) wurden vorgeschlagen.

15.9.1 Konfidenzintervalle

Konfidenzintervalle (Zu) einfach gesagt, gibt ein 95%-Konfidenzintervall an, wie groß der Bereich ist, mit dem der gesuchte Parameter zu 95% Wahrscheinlichkeit liegt (oder allgemeiner das $1 - \alpha$ -Konfidenzintervall. Das kennt man aus dem Wetterbericht, wenn es heißt, dass die Höchsttemperatur morgen zwischen 20 und 24 Grad liegen werde.

Etwas genauer gesagt ist es nach den Urhebern des Konfidenzintervalls, Neyman und Pearson, gar nicht möglich, für ein einzelnes Ereignis eine Wahrscheinlichkeit anzugeben (Clopper und Pearson 1934; Neyman 1935). Wenn ich eine Münze hochwerfe und sie auffange, wie groß ist die Wahrscheinlichkeit, dass sie auf Kopf gelandet ist? 50%? Falsch, sagen ‘Frequentisten’ a la Neyman und Pearson, entweder ist die Münze auf Kopf gelandet, dann kann man höchstens sagen, $p(K) = 1$ oder auf Zahl, dann entsprechend $p(Z) = 1$. Eine Wahrscheinlichkeit macht nur Sinn nach diesem Verständnis, wenn man den Versuch *oft* (unendlich) wiederholt. Daher lautet eine genauere Definition:

Das 95%-Konfidenzintervall ist der Bereich, in dem der Parameter in 95% der Fälle fallen würde bei sehr häufiger Wiederholung des Versuchs.

Mit Parameter ist hier der Mittelwert der Population gemeint (auch bezeichnet als ‘wahrer Mittelwert’). Das Konfidenzintervall macht also Aussagen zur *über ein Verfahren* (einen Bereich berechnen auf Basis von Stichprobendaten), *nicht über den wahren Mittelwert*.

Hier findet sich eine schöne Visualisierung zum Konfidenzintervall¹².

Genau wie der p-Wert werden Konfidenzintervalle häufig missverstanden (sie sind Blutsbrüder im Geiste). Die Studie von Hoekstra, Morey, Rouder und Wagenmakers (2014) zeigt das auf amüsante Weise. In der Studie legten die Autoren einigen Studenten und Wissenschaftlern sechs Fragen zum Wissens-Konfidenzintervall vor, die beantwortet werden sollten. Es wurde ein Kontext vorgestellt, etwa so “Professor Bumbledorf führt ein Experiment durch. Das Ergebnis fasst er in einem 95%-Konfidenzintervall für den Mittelwert zusammen, welches von 0,1 bis 0,4 reicht”. Dann folgten sechs Aussagen, die mit *stimmt* oder *stimmt nicht* zu beantworten waren. Beurteilen auch Sie diese Aussagen¹³.

¹²<http://rpsychologist.com/d3/CI/>

¹³alle sechs sind falsch

-
1. Die Wahrscheinlichkeit, dass der wahre Mittelwert größer als 0 ist, liegt bei mindestens 95%.
 2. Die Wahrscheinlichkeit, dass der wahre Mittelwert gleich 0 ist, ist kleiner als 5%.
 3. Die Nullhypothese, dass der wahre Mittelwert 0 ist, ist wahrscheinlich falsch.
 4. Die Wahrscheinlichkeit, dass der wahre Mittelwert zwischen 0,1 und 0,4 liegt, beträgt 95%.
 5. Wir können zu 95% sicher sein, dass der wahre Mittelwert zwischen 0,1 und 0,4 liegt.
 6. Wenn wir das Experiment immer wieder wiederholen würden, dann würde der wahre Mittelwert in 95% der Fälle zwischen 0,1 und 0,4 fallen.
-

Aussagen 1, 2, 3 und 4 behaupten, der Hypothese bzw. dem Parameter eine Wahrscheinlichkeit zuweisen zu können. Innerhalb des NHST ist das nicht erlaubt, für da Konfidenzintervall soweinig wie für den p-Wert. Aussagen 5 trifft eine Aussage über den wahren Wert, aber Konfidenzintervalle treffen Aussagen über ein Verfahren. Aussage 6 behauptet, dass der wahre Wert variieren könne, tut der aber nicht. Die richtige Aussage, die nicht dabei stand, ist: "Wenn man den Versuch immer wiederholen würden, würden 95% der Intervalle den wahren Mittelwert enthalten". Im Schnitt wurden etwa 3,5 Antworten mit *stimmt* angekreuzt (die Wissenschaftler waren nicht besser als die Studenten).

15.9.2 Effektstärke

Eine weitere Alternative sind Maße der *Effektstärke* (Cohen 1992). Effektstärkemaße geben an, wie sehr sich zwei Parameter unterscheiden: "Deutsche Männer sind im Schnitt 13cm größer als Frauen" (Wikipedia 2017). Oder: "In Deutschland ist die Korrelation von Gewicht und Größe um 0,12 Punkte höher als in den USA" (frei erfunden). Im Gegensatz zu p-Werten wird keine Art von Wahrscheinlichkeitsaussage angestrebt, sondern die Größe von Parameter(unterschieden) quantifiziert. Effektstärken sind, im Gegensatz zum p-Wert, auch nicht abhängig von der Stichprobengröße. Man kann Effektstärken in nicht-standardisierte (wie Unterschiede in der Größe) oder standardisierte (wie Unterschiede in der Korrelation) einteilen.

Nicht-standardisierte Effektstärken haben den Vorteil der Anschaulichkeit. Standardisierte Effektgrößen sind präziser, aber unanschaulicher. Bei Variablen mit unanschaulichen Metriken (wie psychologische Variablen und Umfragen) ist ein standardisiertes Maß häufig nützlicher.

Anschauliche Variablen sind oft mit unstandardisiertes Effektstärken adäquat dargestellt. Variablen mit wenig anschaulichen Metriken profitieren von standardisierten Effektstärkemaßen.

Um zwei Mittelwerte zu vergleichen, ist *Cohens d* gebräuchlich. Es gibt den Unterschied der Mittelwert standardisiert an der Standardabweichung an (Cohen 1988). Das ist oft sinnvoll, denn 5\$ Preisunterschied können viel oder weniger sein: Bei Eiskugeln wäre der Unterschied

enorm (die Streuung ist viel weniger als 5€), bei Sportwagen wäre der Unterschied gering (die Streuung ist viel höher als 5€).

15.9.2.1 Typische Effektstärkemaße

Einige typischen Effektstärkemaße sind im Folgenden aufgelistet (vgl. Eid, Gollwitzer, und Schmitt (2010)). Eine einfache Faustregel ist, dass man für einen bestimmten Test meist ein “zugehöriges” Effektstärkemaß angibt.

- d (Cohens d) wird zur Bemessung des Unterschieds der Überlappung zweier Verteilungen verwendet, z.B. um die Effektstärke eines t-Werts zu quantifizieren. d berechnet sich im einfachsten Fall als: $d = \frac{\mu_1 - \mu_2}{sd}$.
- r Pearsons R ist ein Klassiker, um die Stärke des linearen Zusammenhangs zweier metrischen Größen zu quantifizieren. r berechnet sich als: $r = mw(\sum z_x z_y)$, wobei mw für den Mittelwert steht und z für einen z-Wert.
- R^2 , η^2 sind Maße für den Anteil aufgeklärter Varianz; sie finden in der Varianzanalyse oder der Regressionsanalyse Verwendung. R^2 wird u.a. so berechnet: $R^2 = \frac{QS_F}{QS_T}$, wobei QS für die Quadratsummen stehen und QS_F für die Varianz, die auf den Faktor (unabhängige Variable) zurückgeht und QS_T für die Gesamtvarianz.
- f^2 ist ein Maß, dass aus der erklärten Varianz abgeleitet ist. Es gibt das Verhältnis von erklärter zu nicht-erklärter Varianz wieder (auch ‘signal-noise-ratio’ genannt). Es berechnet sich als $f^2 = \frac{R^2}{1-R^2}$.
- ω (Cohens Omega) ist ein Maß für die Stärke des Zusammenhangs zweier nominaler Variablen, abgeleitet vom χ^2 -Test. Es berechnet sich als $\omega = \sqrt{\chi^2}$.
- OR (Odds Ratio) ist ebenfalls ein Maß für die Stärke des Zusammenhangs zweier nominaler Variablen, allerdings *binärer* (zweistufige) Variablen. Es berechnet sich als $OR = \frac{c}{1-c}$, wobei c die Chancen für ein Ereignis E angeben (z.B. 9:1). OR kann aus ω abgeleitet werden.
- *Simple Difference* ist eine einfachere Variante zum OR . Ein Beispiel: Bei Prof. Feistersack fallen 90% der Kandidaten durch die Prüfung. Bei Prof. Schnaggeldi hingegen nur 50%. Der Wert für *Simple Difference* liegt dann bei $90\%-50\% = 40\%$. Doch, so einfach ist das (Kerby 2014).
- *Anteil konformer Paarvergleiche* ist ein Effektstärkemaß für den Wilcoxon- bzw. den Mann-Whitney-U-Test. Ein Beispiel: Studierende zweier Gruppen müssen wollen sich zu ihrem Statistikwissen testen lassen (Gruppe Prof. F. und Prof. S.). Für jeden Studenten aus Gruppe F wird nun geprüft, gegen wie viele Studenten aus Gruppe S er oder sie ‘gewinnt’. Dann wird ausgezählt, wie viele solcher Paarvergleiche insgesamt von Gruppe F gewonnen wurden. Der Anteil der hypothesenkonformen Paarvergleiche ist dann das Effektstärkemaß (Kerby 2014).

Tabelle 15.1 gibt einen groben Überblick über Effektstärken (nach Cohen (1988) und Eid, Schmitt und Gollwitzer (2010)). Zu beachten ist, dass die Einschätzung was ein ‘großer’ oder ‘kleiner’ Effekt ist, nicht pauschal übers Knie gebrochen werden sollte. Besser ist es, die Höhe der Effektstärke im eigenen Datensatz mit relevanten anderen Datensätzen zu vergleichen.

Tabelle 15.1: Überblick über gängige Effektstärkemaße

Name	kleiner Effekt	mittlerer Effekt	großer Effekt
Cohens d	.2-.5	.5-.8	>.8
r	0.1	0.3	0.5
$\hat{R^2}$, $\hat{\eta^2}$	0.01	0.06	0.14
$\hat{f^2}$	0.02	0.15	0.35
$\hat{\omega^2}$	0.1	0.3	0.5
OR	1.5	3	9

Was ein “kleiner” oder “großer” Effekt ist, sollte im Einzelfall entschieden werden.

Mit dem Paket `pwr` kann man sich Cohens Konventionen der Effektstärkehöhen in Erinnerung rufen lassen. Er bietet folgende Optionen:

```
cohen.ES(test = c("p", "t", "r", "anov", "chisq", "f2"),
         size = c("small", "medium", "large"))
```

15.9.2.2 Effektstärken berechnen

Möchte man sich Effektstärken berechnen lassen, ist das Paket `compute.es` hilfreich. Im Folgenden sind Effektstärkeberechnungen für gängige Inferenztests vorgestellt, in Fortsetzung zu den Beispielen oben.

- χ^2 -Test: `compute.es::chies(30, n = 826)`
- t-Test: `compute.es::tes(t = 1, n.1 = 529, n.2 = 286)`
- Varianzanalyse (F-Test): `broom::glance` gibt R^2 aus; mit `lsr::etaSquared(mein_aov)` ebenfalls. Möchte man f^2 berechnen, so tut man das am besten per Hand, z.B. $0.002 / (1-0.002)$.
- Korrelation: Der Korrelationswert r ist schon ein Maß der Effekstärke. Yeah.
- Regression: Die Steigung der Regressionsgeraden (b) ist ein (unstandardisiertes) Maß für die Stärke des (“Netto”-)Einflusses eines Prädiktors. R^2 hingegen ein Maß für die relative Varianzaufklärung aller Prädiktoren gemeinsam.
- Logistische Regression: Mit `BaylorEdPsych::PseudoR2(mein_glm_objekt)` kann man eine Art R^2 bekommen (s. Kapitel 17.8).
- Wilcoxon-Test/ Mann-Whitney-U-Test: Anteil der paarweisen Vergleiche, die hypothesenkonform sind (vgl. Kerby 2014). Dazu kann man z.B. die Funktion `prop_fav` aus dem Paket `prada` nutzen (vgl. `help(prop_fav)`).

Die R-Befehle für gängige Effektstärkemaße sind in Tabelle 15.2 im Überblick dargestellt.

Tabelle 15.2: R-Befehle für gängige Effektstärkemaße

Test	Effektstärkemaß	R.Befehl
‘t.Test’	Cohens d	‘compute.es::tes‘
‘aov’	\$R^2\$	‘lsr::etaSquared‘
‘lm’	\$R^2\$	‘broom::glance‘ oder ‘summary‘
‘glm’	Pseudo-\$R^2\$	‘BaylorEdPsych::PseudoR2‘
‘chisq.test’	\$\backslash\omega\$	‘sqrt(chisq_wert)‘
‘chisq.test’	OR	‘computes.es::chies‘
‘wilcox.test’	Simple Difference	‘prada::prop_fav‘

15.9.3 Bayes-Statistik

Bayes' Ansatz verrechnet zwei Komponenten, um die Wahrscheinlichkeit einer Hypothese im Lichte bestimmter Daten zu berechnen. Der Ansatz ist elegant, mathematisch luppenrein und ist überhaupt eine tolle Sache. Bayes' Theorem gibt uns das, was uns eigentlich interessiert: Die Wahrscheinlichkeit der getesteten Hypothese, im Lichte der vorliegenden Daten: $p(H|D)$. Diesen Wert nennt man auch den *Vorhersagewert*. Zur Erinnerung: Der p-Wert gibt die Wahrscheinlichkeit der Daten an, unter Annahme der getesteten Hypothese: $p(D|H)$. Offenbar sind beide Terme nicht identisch.

Die Bayes-Statistik zieht zwei Komponenten zur Berechnung von $p(H|D)$ heran. Zum einen die Grundrate einer Hypothese $p(H)$ zum anderen die relative Plausibilität der Daten unter meiner Hypothese im Vergleich zur Plausibilität der Daten unter konkurrierenden Hypothesen. Betrachten wir ein Beispiel. Die Hypothese "Ich bin krank" sei unter Betrachtung (jetzt noch keine vorschnellen Einschätzungen). Die Grundrate der fraglichen Krankheit sei 10 von 1000 (1%). Der Test, der zur Diagnose der Krankheit verwendet wird, habe eine Sicherheit von 90%. Von 100 Kranken wird der Test demnach 90 identifizieren (auch *Sensitivität* genannt) und 10 werden übersehen (ein Überseh- oder *Betafehler* von 10%). Umgekehrt wird der Test von 100 Gesunden wiederum 90 als Gesund, und demnach korrekt diagnostizieren (*Spezifität*); 10 werden fälschlich als krank einschätzt (*Fehlalarm* oder *Alpha-Fehler*).

Jetzt Achtung: Der Test sagt, ich sei krank. Die Gretchen-Frage lautet, wie hoch ist die Wahrscheinlichkeit, dass diese Hypothese, basierend auf den vorliegenden Daten, korrekt ist?

Abbildung 15.10 stellt das Beispiel in Form eines Baumdiagrammes dar.

In der Medizin ist 'positiv' zumeist eine schlechte Nachricht, es soll sagen, dass der Test der Meinung ist, die getestete Person ist krank (das getestete Kriterium trifft zu).

Wie man leicht nachrechnen kann, beträgt die Wahrscheinlichkeit, *in Wirklichkeit krank* zu sein, wenn der positiv ist, $\sim 8\%$: $9/(99 + 9) = \frac{9}{108} \approx 8\%$. Das überrascht auf den ersten Blick, ist doch der Test so überaus zufällig (jedenfalls zu 90%)! Aber die Wahrscheinlichkeit, dass die Hypothese 'krank' zutrifft, ist eben nicht nur abhängig von der Sicherheit des Tests, sondern auch von der Grundrate. Beide Komponenten sind nötig, um den Vorhersagewert zu berechnen. Der p-Wert begnügt sich mit der Aussage, ob der Test positiv oder negativ ist. Die Grundrate wird nicht berücksichtigt.

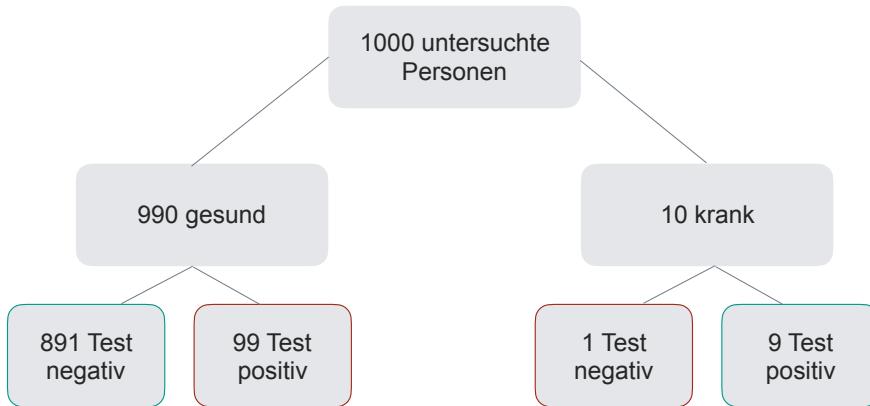


Abbildung 15.10: Die zwei Stufen der Bayes-Statistik in einem einfachen Beispiel

Die Bayes-Statistik liefert die Wahrscheinlichkeit einer Hypothese H , wenn wir die Daten D (d.h. ein gewisses Stichprobenergebnis) gefunden haben: $p(H|D)$. Damit gibt die Bayes-Statistik die Antwort, die sich die meisten Anwender wünschen.

Fairerweise muss man hinzufügen, dass die Grundrate für die Wissenschaft oft nicht einfach zu bestimmen ist. Wer kennt schon die Grundrate der ‘guten Ideen’? Vielleicht der liebe Gott, aber der hilft uns nicht¹⁴ (God 2016). Wir werden also eine Einschätzung treffen müssen, die subjektiv sein kann. Diese Subjektivität ist von Kritikern moniert worden.

Auf der anderen Seite kann man diese Subjektivität umgehen, indem man nur angibt, um welchen Faktor die H_1 wahrscheinlicher ist als die H_0 , durch die Daten der Studie. Das wird durch den sog. *Bayes-Faktor BF* ausgedrückt. Liegt BF bei 10, so eine gängige Konvention, so ist dies “starke” Evidenz für H_1 (da H_1 dann 10 mal wahrscheinlicher als die H_0); entsprechend stark ist ein BF von 0.1 (1/10) - zugunsten H_0 . Gängige Software (s. Abschnitt 15.12) geben den Bayes-Faktor aus.

Ein t-Test a la Bayes kann z.B. so berechnet werden:

```

extra %>%
  group_by(sex) %>%
  summarise(mean(extra_mean, na.rm = TRUE))
#> # A tibble: 3 x 2
#>   sex `mean(extra_mean, na.rm = TRUE)` 
#>   <fctr>                <dbl>
#> 1 Frau                  2.91
#> 2 Mann                  2.86
#> 3 <NA>                  2.73

extra %>%
  filter(sex %in% c("Mann", "Frau")) %>%
  
```

¹⁴<https://twitter.com/TheTweetOfGod/status/688035049187454976>

```

mutate(sex = factor(sex)) %>%
as.data.frame %>% # 'ttestBF' verkraftet nur althergebrachte data.frames!
ttestBF(formula = extra_mean ~ sex,
       data = .) # 'formula' musst hingeschrieben sein, sonst droht Fehlermeldung
#> Bayes factor analysis
#> -----
#> [1] Alt., r=0.707 : 0.22 ±0%
#>
#> Against denominator:
#>   Null, mu1-mu2 = 0
#> ---
#> Bayes factor type: BFindepSample, JZS

```

Hey, Sie haben gerade einen Bayes-Test gerechnet! Wow! Das Ergebnis zeigt einen *BF* von 0.24; Evidenz *zugunsten* der H₀. Nicht stark; sondern schwach. Keine überzeugende Evidenz für H₁. Man beachte, dass der Befehl hier “indifferent” gegenüber der H₀ und der H₁ war. A priori wurden hier beide Hypothesen als gleich wahrscheinlich angesehen. Jetzt ist unsere Überzeugung für die H₁ gesunken bzw. für die H₀ gestiegen und zwar etwa um den Faktor 1/4 auf 0.24.

15.10 Aufgaben¹⁵



Richtig oder Falsch!?

1. Der p-Wert gibt die Wahrscheinlichkeit der H₀ an unter der Annahme der Daten.
2. $p(D|H) = p(H|D)$
3. Der p-Wert sagt, wie gut die Daten zur Nullhypothese passen.
4. Bei sehr großen Stichproben werden nur sehr große Effekte signifikant.
5. Egal wie klein die Effektstärke ist, es existiert eine Stichprobengröße, die diesen Effekt beliebig signifikant werden lässt.
6. Wenn der p-Wert kleiner als 5% ist, dann ist meine Hypothese (H₁) höchstwahrscheinlich richtig.
7. Wenn der p-Wert größer als 5% ist, dann ist das ein Beleg für die H₀.
8. Der p-Wert basiert auf der Idee, dass man ein Experiment unendlich oft wiederholt; und das unter zufälligen aber ansonsten komplett gleichen Bedingungen.

¹⁵F, F, R, F, F, F, F, R, R, R

9. Das 95%-Konfidenzintervall ist der Bereich, in dem der Parameter in 95% der Fälle fallen würde bei sehr häufiger Wiederholung des Versuchs.
10. Der Vorhersagewert ist definiert als $p(H|D)$.

15.11 Fazit

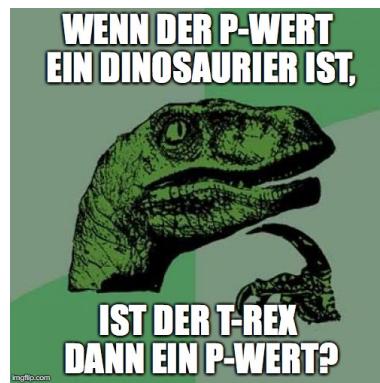
Der p-Wert ist eine häufig verwendete Methode, um datenbasiert zu entscheiden, ob man eine Hypothese annimmt oder nicht. Allerdings hat der p-Wert auch seine Probleme.

Der p-Wert sollte nicht als einziges Kriterium verwendet werden, um eine Hypothese bzw. ein Modell zu beurteilen.

Da der p-Wert aber immer noch der Platzhirsch auf vielen Forschungsauen ist, führt kein Weg um ihn herum. Er muss genau verstanden werden: Was er sagt und - wichtiger noch - was er nicht sagt.

Alternativen zum p-Wert sind

- Konfidenzintervalle
- Effektstärkemaße inkl. Maße der Vorhersagegenauigkeit
- Bayes-Theorem



15.12 Verweise

- Eine Einführung zur Bayes-Statistik findet man z.B. bei Kruschke (2010) oder bei Etz u. a. (2016).
- Eine ausführliche Darstellung der Inferenzstatistik und des p-Werts findet sich z.B. bei Lübke und Vogt (2014) oder Eid, Gollwitzer, und Schmitt (2010).

- Eine vielversprechende, noch recht neue Software ist JASP¹⁶, die nicht nur schöne Diagramme erstellt, sondern auch auf Mausklick eine Reihe von bayesianischer (und frequentistischer) Tests durchrechnet.

Teil VII

Geleitetes Modellieren

Kapitel 16

Lineare Regression



Lernziele:

- Wissen, was man unter Regression versteht.
- Die Annahmen der Regression überprüfen können.
- Regression mit kategorialen Prädiktoren durchführen können.
- Die Modellgüte bei der Regression bestimmen können.
- Interaktionen erkennen und ihre Stärke einschätzen können.

Für dieses Kapitel benötigen Sie folgende Pakete:

```
library(caret) # Modellieren
library(tidyverse) # Datenjudo, Visualisierung, ...
library(gridExtra) # Mehrere Plots kombinieren
library(modelr) # Residuen und Schätzwerte zum Datensatz hinzufügen
library(broom) # Regressionswerte geordnet ausgeben lassen
```

16.1 Die Idee der klassischen Regression

Regression ist eine bestimmte Art der *Modellierung* von Daten. Wir legen eine Gerade ‘schön mittig’ in die Daten; damit haben wir ein einfaches Modell der Daten (vgl. Abb. 16.1). Die Gerade ‘erklärt’ die Daten: Für jeden X-Wert liefert sie einen Y-Wert als Vorhersage zurück.

```
stats_test <- read.csv("data/stats_test.csv")

stats_test %>%
  ggplot +
  aes(x = study_time, y = score) +
```

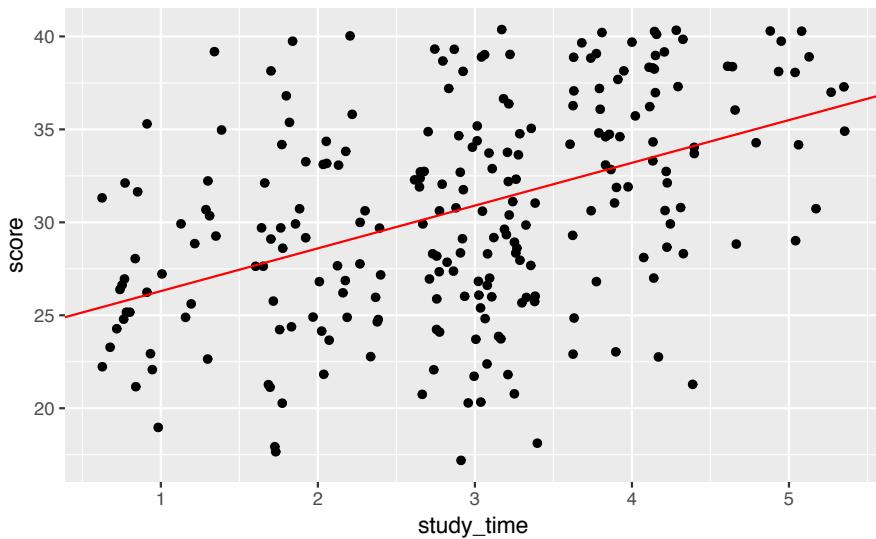


Abbildung 16.1: Beispiel für eine Regression

```
geom_jitter() +
  geom_abline(intercept = 24,
              slope = 2.3,
              color = "red")
```

Wie wir genau die Regressionsgerade berechnet haben, dazu gleich mehr. Fürs Erste begnügen wir uns mit der etwas groberen Beobachtung, dass die Gerade ‘schön mittig’ in der Punktwolke liegt.

Schauen wir uns zunächst die Syntax genauer an.



Lade die CSV-Datei mit den Daten als `stats_test`.

Nehme `stats_test` UND DANN...

starte ein neues Diagramm mit `ggplot` UND

definiere das Diagramm (X-Achse, Y-Achse) UND DANN

zeichne das Geom “Jitter” (verwackeltes Punktediagramm) UND DANN

und zeichne danach eine Gerade (“abline” in rot).

Eine Regression zeigt anhand einer Regressionsgeraden einen “Trend” in den Daten an (s. weitere Beispiele in Abb. 16.2).

Eine Regression lädt förmlich dazu ein, Vorhersagen zu treffen: Hat man erstmal eine Gerade, so kann man für jeden X-Wert (“Prädiktor”) eine Vorhersage für den Y-Wert (“Kriterium”) treffen. Anhand des Diagramms kann man also für jede Person (d.h. jeden Wert innerhalb des Wertebereichs von `study_time` oder einem anderen Prädiktor) einen Wert für `score` vorhersagen. Wie gut die Vorhersage ist, steht erstmal auf einem anderen Blatt.

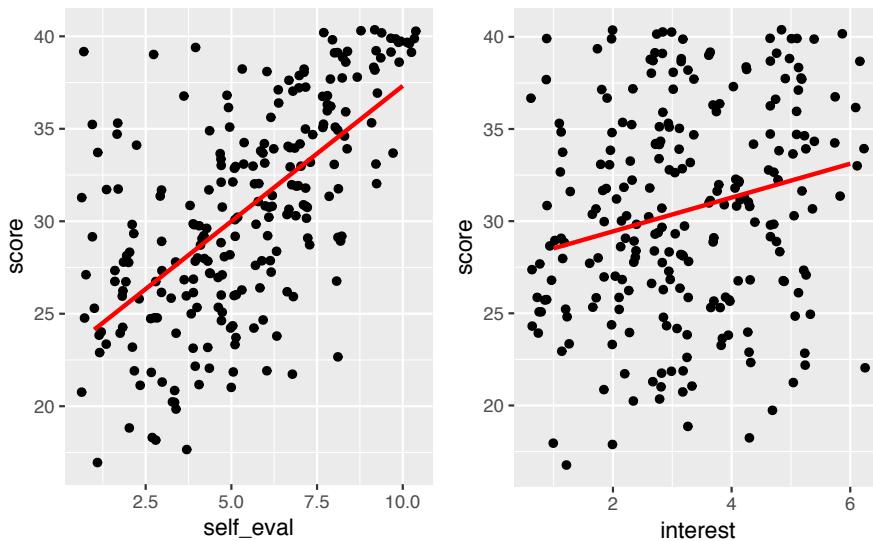


Abbildung 16.2: Zwei weitere Beispiele für Regressionen

Man beachte, dass eine Gerade über ihre *Steigung* und ihren *Achsenabschnitt* festgelegt ist; in Abb. 16.1 ist die Steigung 2.3 und der Achsenabschnitt 24. Der Achsenabschnitt zeigt also an, wie viele Klausurpunkte man “bekommt”, wenn man gar nicht lernt (Gott bewahre); die Steigung gibt eine Art “Wechselkurs” an: Wie viele Klausurpunkte bekomme ich pro Stunde, die ich lerne.

Unser Modell ist übrigens einfach gehalten: Man könnte argumentieren, dass der Zusatznutzen der 393. Stunde lernen geringer ist als der Zusatznutzen der ersten paar Stunden. Aber dann müssten wir anstelle der Gerade eine andere Funktion nutzen, um die Daten zu modellieren. Lassen wir es erst einmal einfach hier.

Als “Pseudo-R-Formel” ausgedrückt:

```
score = achsenabschnitt + steigung*study_time
```

Die Vorhersage für die Klausurpunkte (`score`) einer Person sind der Wert des Achsenabschnitts plus das Produkt aus der Anzahl der gelernten Stunden mal den Zusatznutzen pro gelernter Stunde.

Aber wie erkannt man, ob eine Regression “gut” ist - die Vorhersagen also präzise?

In R kann man eine Regression so berechnen:

```
lm(score ~ study_time, data = stats_test)
#>
#> Call:
#> lm(formula = score ~ study_time, data = stats_test)
#>
#> Coefficients:
#> (Intercept)   study_time
```

#>	23.98	2.26
----	-------	------

`lm` steht dabei für “lineares Modell”; allgemeiner gesprochen lautet die Rechtschreibung für diesen Befehl:

```
lm(kriterium ~ praediktor, data = meine_datentabelle)
```

Um ausführlichere Informationen über das Regressionsmodell zu bekommen, kann man die Funktion `broom::tidy` nutzen:

```
mein_lm <- lm(kriterium ~ praediktor, data = meine_datentabelle)
tidy(mein_lm)
```

Natürlich kann das auch ~~in der Pfeife rauchen~~ mit der Pfeife darstellen:

```
lm(kriterium ~ praediktor, data = meine_datentabelle) %>%
  summary
```

16.2 Modellgüte

In einem Regressionsmodell lautet die grundlegenden Überlegung zur Modellgüte:

Wie groß ist der Unterschied zwischen Vorhersage und Wirklichkeit?

Die Größe des Unterschieds (Differenz, “Delta”) zwischen vorhergesagten (geschätzten) Wert und Wirklichkeit, bezeichnet man als *Fehler*, *Residuum* oder Vohersagefehler, häufig mit ϵ (griechisches e wie “error”) abgekürzt.

Betrachten Sie die beiden Plots in Abb. 14.13. Die rote Linie gibt die *vorhergesagten* (geschätzten) Werte wieder; die Punkte die *beobachteten* (“echten”) Werte. Je länger die blauen Linien, desto größer die Vorhersagefehler. Je größer der Vorhersagefehler, desto schlechter. Und umgekehrt.

Je kürzer die typische “Abweichungslinie”, desto besser die Vohersage.

Sagt mein Modell voraus, dass Ihre Schuhgröße 49 ist, aber in Wahrheit liegt sie bei 39, so werden Sie dieses Modell als schlecht beurteilen, wahrscheinlich.

Leider ist es nicht immer einfach zu sagen, wie groß der Fehler sein muss, damit das Modell als “gut” bzw. “schlecht” gilt. Man kann argumentieren, dass es keine wissenschaftliche Frage sei, wie viel “viel” oder “genug” ist (Briggs 2016). Das ist zwar plausibel, hilft aber nicht, wenn ich eine Entscheidung treffen muss. Stellen Sie sich vor: Ich zwinge Sie mit der Pistole auf der Brust, meine Schuhgröße zu schätzen.

Eine einfache Lösung ist, das beste Modell unter mehreren Kandidaten zu wählen.

Ein anderer Ansatz ist, die Vorhersage in Bezug zu einem Kriterium zu setzen. Dieses “andere Kriterium” könnte sein “einfach die Schuhgröße raten”. Oder, etwas intelligenter, Sie schätzen meine Schuhgröße auf einen Wert, der eine gewisse Plausibilität hat, also z.B. die durchschnittliche Schuhgröße des deutschen Mannes. Auf dieser Basis kann man dann quantifizieren, ob und wie viel besser man als dieses Referenzkriterium ist.

16.2.1 Mittlere Quadratfehler

Eine der häufigsten Gütekennzahlen ist der *mittlere quadrierte Fehler* (engl. “mean squared error”, MSE), wobei Fehler wieder als Differenz zwischen Vorhersage (`pred`) und beobachtete Wirklichkeit (`obs`, `y`) definiert ist. Dieser berechnet für jede Beobachtung den Fehler, quadriert diesen Fehler und bilden dann den Mittelwert dieser “Quadratfehler”, also einen *mittleren Quadratfehler*. Die englische Abkürzung *MSE* ist auch im Deutschen gebräuchlich.

$$MSE = \frac{1}{n} \sum (pred - obs)^2$$

Konzeptionell ist dieses Maß an die Varianz angelehnt. Zieht man aus diesem Maß die Wurzel, so erhält man den sog. *root mean square error* (RMSE), welchen man sich als die Standardabweichung der Vorhersagefehler vorstellen kann. Dieses Maß (RMSE) ist vielleicht das gebräuchlichste. In Pseudo-R-Syntax:

```
RMSE <- sqrt(mean((df$pred - df$obs)^2))
```

Der RMSE hat die selben Einheiten wie die zu schätzende Variable, also z.B. Schuhgrößen-Nummern.

16.2.2 R-Quadrat (R^2)

R^2 , auch *Bestimmtheitsmaß* oder *Determinationskoeffizient* genannt, setzt die Höhe unseres Vorhersagefehlers im Verhältnis zum Vorhersagefehler eines “Nullmodell”. Das Nullmodell hier würde sagen, wenn es sprechen könnte: “Keine Ahnung, was ich schätzen soll, mich interessieren auch keine Prädiktoren, ich schätzen einfach immer den Mittelwert der Grundgesamtheit!”.

Analog zum Nullmodell-Fehler spricht auch von der Gesamtvarianz oder SS_T (sum of squares total); beim Vorhersagefehler des eigentlichen Modells spricht man auch von SS_M (sum of squares model).

Damit gibt R^2 an, wie gut unsere Vorhersagen im Verhältnis zu den Vorhersagen des Nullmodells sind. Ein R^2 von 25% (0.25) hieße, dass unser Vorhersagefehler 25% kleiner ist als der der Nullmodells. Ein R^2 von 100% (1) heißt also, dass wir den kompletten Fehler reduziert haben (Null Fehler übrig) - eine perfekte Vorhersage. Etwas formaler, kann man R^2 so definieren:

$$R^2 = 1 - \left(\frac{SS_T - SS_M}{SS_T} \right)$$

Präziser, in R-Syntax:

```
R2 <- 1 - sum((df$pred - df$obs)^2) / sum((mean(df$obs) - df$obs)^2)
```

Praktischerweise gibt es einige R-Pakete, z.B. `caret`, die diese Berechnung für uns besorgen:

```
postResample(obs = obs, pred = pred)
```

Hier steht `obs` für beobachtete Werte und `pred` für die vorhergesagten Werte (beides numerische Vektoren). Dieser Befehl gibt sowohl RMSE als auch R^2 wieder. Wir betrachten gleich ein Beispiel an echten Daten.



Verwendet man die Korrelation (`r`) oder R^2 als Gütekriterium, so sollte man sich über folgenden Punkt klar sein. Bei Skalierung der Variablen ändert sich die Korrelation nicht; das gilt auch für R^2 . Beide Koeffizienten ziehen allein auf das *Muster* der Zusammenhänge ab - nicht die Größe der Abstände. Aber häufig ist die Größe der Abstände zwischen beobachteten und vorhergesagten Werten das, was uns interessiert. In dem Fall wäre der (R)MSE vorzuziehen.

16.3 Die Regression an einem Beispiel erläutert

Schauen wir uns den Datensatz zur Statistikklausur noch einmal an. Welchen Einfluss hat die Lernzeit auf den Klausurerfolg? Wie viel bringt es also zu lernen? Wenn das Lernen keinen Einfluss auf den Klausurerfolg hat, dann kann man es ja gleich sein lassen... Aber umgekehrt, wenn es viel bringt, ok gut, dann könnte man sich die Sache (vielleicht) noch mal überlegen. Aber was heißt "viel bringen" eigentlich?

Wenn für jede Stunde Lernen viele zusätzliche Punkte herausspringen, dann bringt Lernen viel. Allgemeiner: Je größer der Zuwachs im Kriterium ist pro zusätzliche Einheit des Prädiktors, desto größer ist der Einfluss des Prädiktors.

Natürlich könnte jetzt jemand argumentieren, dass die ersten paar Stunden lernen viel bringen, aber dann flacht der Nutzen ab, weil es ja schnell einfach und trivial wird. Aber wir argumentieren (erstmal) so nicht. Wir gehen davon aus, dass jede Stunde Lernen gleich viel (oder wenig) Nutzen bringt.

Geht man davon aus, dass jede Einheit des Prädiktors gleich viel Zuwachs bringt, unabhängig von dem Wert des Prädiktors, so geht man von einem linearen Einfluss aus.

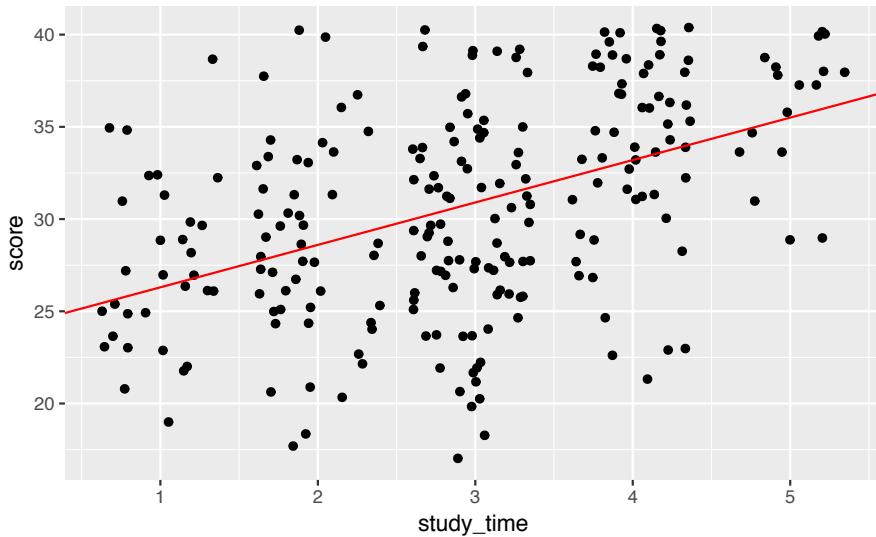


Abbildung 16.3: Streudiagramm von Lernzeit und Klausurerfolg

Versuchen wir im ersten Schritt die Stärke des Einfluss an einem Streudiagramm abzuschätzen (s. Abb. 16.1).

Hey R - berechne uns die “Trendlinie”! Dazu nimmt man den Befehl `lm`:

```
mein_lm <- lm(score ~ study_time, data = stats_test)
tidy(mein_lm)
#>   term estimate std.error statistic p.value
#> 1 (Intercept)  23.98     0.934    25.67 2.94e-70
#> 2 study_time    2.26     0.300     7.54 1.02e-12
```

`lm` steht für ‘lineares Modell’, eben weil eine *Linie* als Modell in die Daten gelegt wird. Aha. Die Steigung der Geraden beträgt 2.3 - das ist der Einfluss des Prädiktors Lernzeit auf das Kriterium Klausurerfolg! Man könnte sagen: Der “Wechselkurs” von Lernzeit auf Klausurpunkte. Für jede Stunde Lernzeit bekommt man offenbar 2.3 Klausurpunkte (natürlich viel zu leicht). Wenn man nichts lernt (`study_time == 0`) hat man 24 Punkte.

Der Einfluss des Prädiktors steht unter ‘estimate’. Der Kriteriumswert wenn der Prädiktor Null ist steht unter ‘(Intercept)’.

Malen wir diese Gerade in unser Streudiagramm (Abbildung 16.3).

```
ggplot(data = stats_test) +
  aes(y = score, x = study_time) +
  geom_jitter() +
  geom_abline(slope = 2.3, intercept = 24, color = "red")
```

Jetzt kennen wir die Stärke (und Richtung) des Einflusses der Lernzeit. Ob das viel oder

wenig ist, ist am besten im Verhältnis zu einem Referenzwert zu sagen.

Die Gerade wird übrigens so in die Punktwolke gelegt, dass die (quadrierten) Abstände der Punkte zur Geraden minimal sind. Dies wird auch als *Kriterium der Kleinsten Quadrate (Ordinary Least Squares, OLS)* bezeichnet.

Jetzt können wir auch einfach Vorhersagen machen. Sagt uns jemand, ich habe “viel” gelernt (Lernzeit = 4), so können wir den Klausurerfolg grob im Diagramm ablesen.

Genauer geht es natürlich mit dieser Rechnung:

$$y = 4 * 2.3 + 24$$

Oder mit diesem R-Befehl:

```
predict(mein_lm, data.frame(study_time = 4))
#> 1
#> 33
```

Berechnen wir noch die Vorversagegüte des Modells. Dazu kann man den Befehl `summary` nehmen, oder auch `broom::glance`. `glance` gibt Informationen zur Modellgüte zurück und das in Form eines Dateframes. Summary liefert eine Menge Informationen mit einem infomrationen Ausdruck, aber nicht in Form eines Dataframes (sondern in Form einer Liste).

```
glance(mein_lm)
#>   r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC
#> 1    0.194        0.191  5.15     56.8 1.02e-12  2   -727 1459 1470
#>   deviance df.residual
#> 1      6255       236
```

Das Bestimmtheitsmaß R^2 ist mit 0.19 “ok”: 19-% der Varianz des Klausurerfolg wird im Modell ‘erklärt’. ‘Erklärt’ meint hier, dass wenn die Lernzeit konstant wäre, würde die Varianz von Klausurerfolg um diesen Prozentwert sinken.

16.4 Überprüfung der Annahmen der linearen Regression

Aber wie sieht es mit den Annahmen aus?

- Die *Linearität des Zusammenhangs* haben wir zu Beginn mit Hilfe des Scatterplots überprüft. Es schien einigermaßen zu passen.
- Zur Überprüfung der *Normalverteilung der Residuen* zeichnen wir ein Histogramm (s. Abbildung 16.4). Die *Residuen* können über den Befehl `add_residuals` (Paket `modelr`) zum Datensatz hinzugefügt werden. Dann wird eine Spalte mit dem Namen `resid` zum Datensatz hinzugefügt.

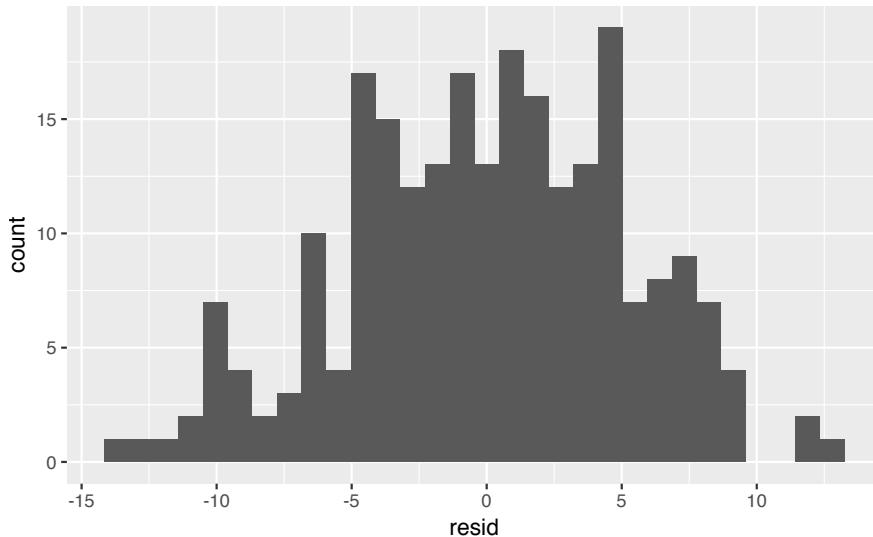


Abbildung 16.4: Die Residuen verteilen sich hinreichend normal.

Hier scheint es zu passen:

```
stats_test %>%
  add_residuals(mein_lm) %>%
  ggplot +
  aes(x = resid) +
  geom_histogram()
```

Sieht passabel aus. Übrigens kann man das Paket `modelr` auch nutzen, um sich komfortabel die vorhergesagten Werte zum Datensatz hinzufügen zu lassen (Spalte `pred`):

```
stats_test %>%
  add_predictions(mein_lm) %>%
  select(pred) %>%
  head
#>   pred
#> 1 35.3
#> 2 30.8
#> 3 35.3
#> 4 28.5
#> 5 33.0
#> 6    NA
```

- *Konstante Varianz*: Dies kann z. B. mit einem Scatterplot der Residuen auf der Y-Achse und den vorhergesagten Werten auf der X-Achse überprüft werden. Bei jedem X-Wert sollte die Varianz der Y-Werte (etwa) gleich sein (s. Abbildung 16.5).

Die geschätzten (angepassten) Werte kann man über den Befehl `add_predictions()` aus

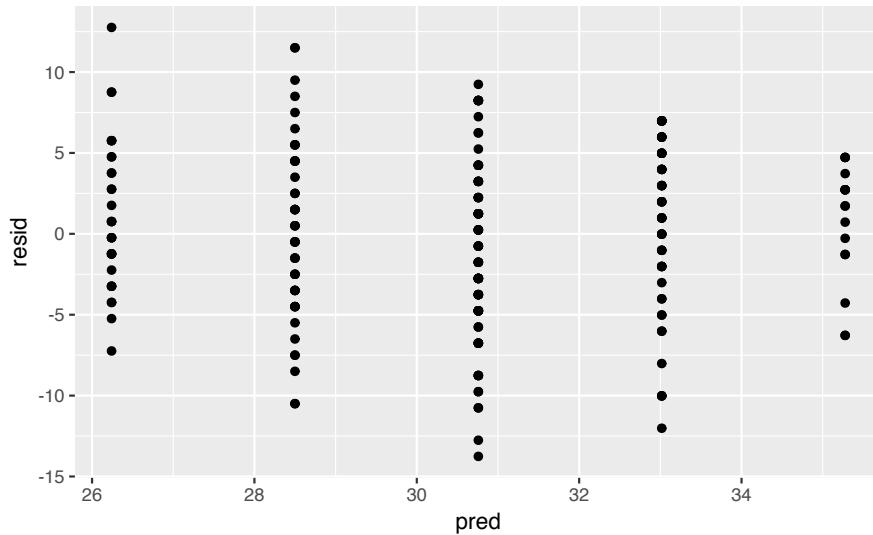


Abbildung 16.5: Vorhergesagte Werte vs. Residualwerte im Datensatz tips

dem Paket `modelr` bekommen. Die Fehlerwerte entsprechend mit dem Befehl `add_residuals`.

```
stats_test %>%
  add_predictions(mein_lm) %>%
  add_residuals(mein_lm) %>%
  ggplot() +
  aes(y = resid, x = pred) +
  geom_point()
```

Die Annahme der konstanten Varianz scheint verletzt zu sein: Die sehr großen vorhersagten Werte können recht genau geschätzt werden; aber die mittleren Werte nur ungenau. Die Verletzung dieser Annahme beeinflusst *nicht* die Schätzung der Steigung, sondern die Schätzung des Standardfehlers, also des p-Wertes der Einflusswerte.

- *Extreme Ausreißer*: Extreme Ausreißer scheint es nicht zu geben.
- *Unabhängigkeit der Beobachtungen*: Wenn die Studenten in Lerngruppen lernen, kann es sein, dass die Beobachtungen nicht unabhängig voneinander sind: Wenn ein Mitglied der Lerngruppe gute Noten hat, ist die Wahrscheinlichkeit für ebenfalls gute Noten bei den anderen Mitgliedern der Lerngruppe erhöht. Böse Zungen behaupten, dass ‘Abschreiben’ eine Gefahr für die Unabhängigkeit der Beobachtungen sei.



1. Wie groß ist der Einfluss des Interesss?
2. Für wie aussagekräftig halten Sie Ihr Ergebnis aus 1.?
3. Welcher Einflussfaktor (in unseren Daten) ist am stärksten?

16.5 Regression mit kategorialen Prädiktoren

Vergleichen wir interessierte und nicht interessierte Studenten. Dazu teilen wir die Variable `interest` in zwei Gruppen (1-3 vs. 4-6) auf:

```
stats_test$interessiert <- stats_test$interest > 3
```

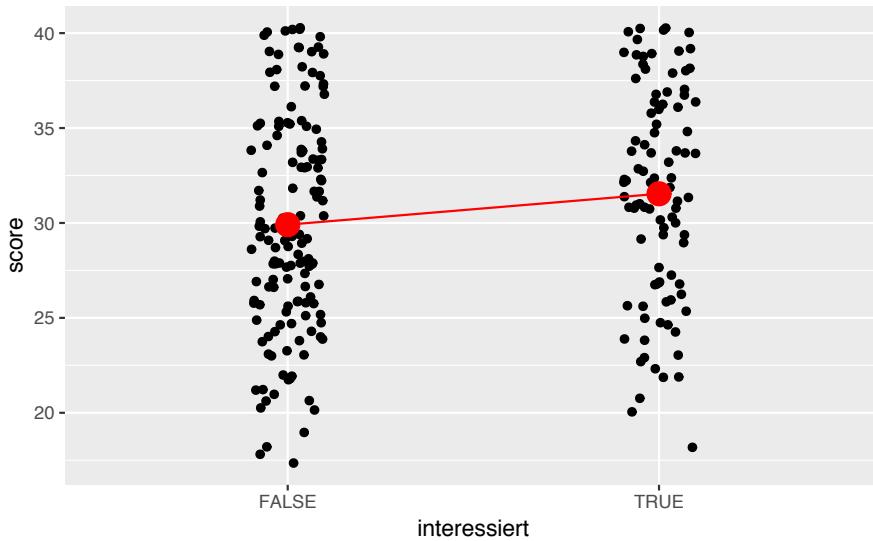
Vergleichen wir die Mittelwerte des Klausurerfolgs zwischen den Interessierten und Nicht-Interessierten:

```
stats_test %>%
  group_by(interessiert) %>%
  summarise(score = mean(score)) -> score_interesse

score_interesse
#> # A tibble: 3 x 2
#>   interessiert score
#>   <lgcl>     <dbl>
#> 1 FALSE      29.9
#> 2 TRUE       31.5
#> 3 NA        33.1
```

Aha, die Interessierten haben im Schnitt mehr Punkte; aber nicht viel.

```
stats_test %>%
  na.omit %>%
  ggplot() +
  aes(x = interessiert, y = score) +
  geom_jitter(width = .1) +
  geom_point(data = score_interesse, color = "red", size = 5) +
  geom_line(data = score_interesse, group = 1, color = "red")
```



Mit `group=1` bekommt man eine Linie, die alle Punkte verbindet (im Datensatz `score_interesse` sind es dieser zwei). Wir haben in dem Fall nur zwei Punkte, die entsprechend verbunden werden.

16.5.1 Aufgaben

1. Visualisieren Sie den Gruppenunterschied auch mit einem Boxplot!
2. Berechnen Sie ein lineares Modell dazu!

Lösung:

1. Boxplot: `qplot(x = interessiert, y = score, data = stats_test, geom = "boxplot")`
2. Lineares Modell:

```
lm2 <- lm(score ~ interessiert, data = stats_test)
summary(lm2)
#>
#> Call:
#> lm(formula = score ~ interessiert, data = stats_test)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -13.537  -4.380  -0.537   4.463  10.091
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 29.909     0.475   62.99  <2e-16 ***
#>
```

```
#> interessiertTRUE    1.628      0.752     2.17     0.031 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.68 on 236 degrees of freedom
#>   (68 observations deleted due to missingness)
#> Multiple R-squared:  0.0195, Adjusted R-squared:  0.0153
#> F-statistic: 4.69 on 1 and 236 DF,  p-value: 0.0313
```

Der Einfluss von `interessiert` ist statistisch signifikant ($p = .03$). Der Stärke des Einflusses ist im Schnitt 1.6 Klausurpunkte (zum Vorteil von `interessiertTRUE`). Das ist genau, was wir oben herausgefunden haben.



3. Wie ist der Einfluss von `study_time`, auch in zwei Gruppen geteilt?
4. Wie viel % der Variation des Klausurerfolgs können Sie durch das Interesse modellieren?

16.6 Multiple Regression

Aber wie wirken sich mehrere Einflussgrößen *zusammen* auf den Klausurerfolg aus?

```
lm3 <- lm(score ~ study_time + interessiert, data = stats_test)
summary(lm3)
#>
#> Call:
#> lm(formula = score ~ study_time + interessiert, data = stats_test)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -13.896 -3.577  0.418  3.805 13.065
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  23.955    0.938   25.55 < 2e-16 ***
#> study_time    2.314    0.323    7.15 1.1e-11 ***
#> interessiertTRUE -0.333    0.736   -0.45    0.65
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

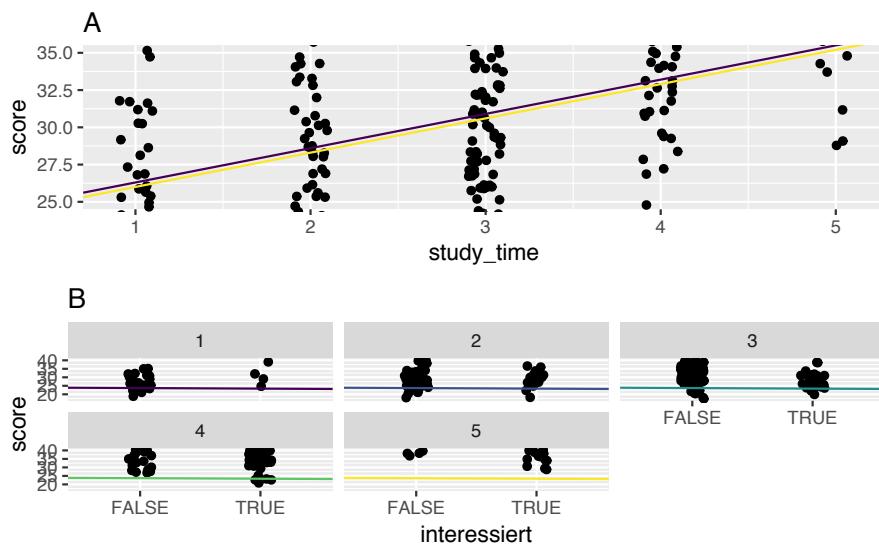


Abbildung 16.6: Eine multivariate Analyse fördert Einsichten zu Tage, die bei einfacheren Analysen verborgen bleiben

```
#> Residual standard error: 5.16 on 235 degrees of freedom
#>   (68 observations deleted due to missingness)
#> Multiple R-squared:  0.195, Adjusted R-squared:  0.188
#> F-statistic: 28.4 on 2 and 235 DF, p-value: 8.81e-12
```

Interessant ist das *negative* Vorzeichen vor dem Einfluss von `interessiertTRUE!` Die multiple Regression untersucht den ‘Nettoeinfluss’ jedes Prädiktors. Den Einfluss also, wenn der andere Prädiktor *konstant* gehalten wird. Anders gesagt: Betrachten wir jeden Wert von `study_time` separat, so haben die Interessierten jeweils im Schnitt etwas *weniger* Punkte (jesses). Allerdings ist dieser Unterschied nicht statistisch signifikant.

Die multiple Regression zeigt den ‘Nettoeinfluss’ jedes Prädiktor: Den Einfluss dieses Prädiktor, wenn der andere Prädiktor oder die anderen Prädiktoren konstant gehalten werden.

Hier haben wir übrigens dem Modell aufgezwungen, dass der Einfluss von Lernzeit auf Klausurerfolg bei den beiden Gruppen gleich groß sein soll (d.h. bei Interessierten und Nicht-Interessierten ist die Steigung der Regressionsgeraden gleich). Das illustriert sich am einfachsten in einem Diagramm (s. Abbildung 16.6).

Diese *multivariate* Analyse (mehr als 2 Variablen sind beteiligt) zeigt uns, dass die Regressionsgerade nicht gleich ist in den beiden Gruppen (Interessierte vs. Nicht-Interessierte; s. Abbildung 16.6): Im Teildiagramm A sind die Geraden (leicht) versetzt. Analog zeigt Teildiagramm B, dass die Interessierten (`interessiert == TRUE`) geringe Punktewerte haben als die Nicht-Interessierten, wenn man die Werte von `study_time` getrennt betrachtet.

Die multivariate Analyse zeigt ein anderes Bild, ein genaueres Bild als die einfache Analyse. Ein Sachverhalt, der für den ganzen Datensatz gilt, kann in

Subgruppen anders sein.

Ohne multivariate Analyse hätten wir dies nicht entdeckt. Daher sind multivariate Analysen sinnvoll und sollten gegenüber einfacheren Analysen bevorzugt werden.

Man könnte sich jetzt noch fragen, ob die Regressionssgerade in Abbildung 16.6 parallel sein müssen. Gerade hat unser R-Befehl sie noch gezwungen, parallel zu sein. Gleich lassen wir hier die Zügel locker. Wenn die Regressionsgerade nicht mehr parallel sind, spricht man von *Interaktionseffekten*.

Das Ergebnis des zugrunde-liegenden F-Tests (vgl. Varianzanalyse) wird in der letzten Zeile angegeben (*F-Statistic*). Hier wird H_0 also verworfen.

16.7 Interaktionen

Es könnte ja sein, dass die Stärke des Einflusses von Lernzeit auf Klausurerfolg in der Gruppe der Interessierten anders ist als in der Gruppe der Nicht-Interessierten. Wenn man nicht interessiert ist, so könnte man argumentieren, dann bringt eine Stunden Lernen weniger als wenn man interessiert ist. Darum müssten die Steigungen der Regressionsgeraden in den beiden Gruppen unterschiedlich sein. Schauen wir uns es an. Um R dazu zu bringen, die Regressionsgeraden frei variieren zu lassen, so dass sie nicht mehr parallel sind, nutzen wir das Symbol *, dass wir zwischen die betreffenden Prädiktoren schreiben:

```
lm4 <- lm(score ~ interessiert*study_time, data = stats_test)
summary(lm4)
#>
#> Call:
#> lm(formula = score ~ interessiert * study_time, data = stats_test)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -13.950  -3.614   0.356   4.020  12.598
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  23.627    1.158   20.40 < 2e-16 ***
#> interessiertTRUE 0.655    2.170   0.30    0.76
#> study_time   2.441    0.418   5.85  1.7e-08 ***
#> interessiertTRUE:study_time -0.321    0.662   -0.48    0.63
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.17 on 234 degrees of freedom
#> (68 observations deleted due to missingness)
```

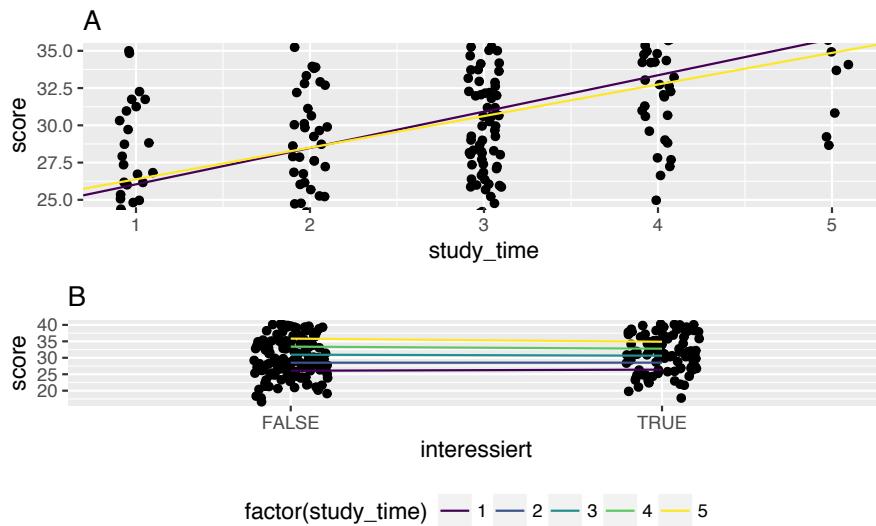


Abbildung 16.7: Eine Regressionsanalyse mit Interaktionseffekten

```
#> Multiple R-squared:  0.196,  Adjusted R-squared:  0.185
#> F-statistic:    19 on 3 and 234 DF,  p-value: 4.81e-11
```

Interessanterweise zeigen die Interessierten nun wiederum - betrachtet man jede Stufe von `study_time` einzeln - bessere Klausurergebnisse als die Nicht-Interessierten. Ansonsten ist noch die Zeile `interessiertTRUE:study_time` neu. Diese Zeile zeigt die Höhe des *Interaktionseffekts*. Bei den Interessierten ist die Steigung der Geraden um 0.32 Punkte geringer als bei den Nicht-Interessierten. Der Effekt ist klein und nicht statistisch signifikant, so dass wir wahrscheinlich Zufallsrauschen überinterpretieren. Aber die reine Zahl sagt, dass bei den Interessierten jede Lernstunde weniger Klausurerfolg bringt als bei den Nicht-Interessierten. Auch hier ist eine Visualisierung wieder hilfreich.

Wir sehen in Abbildung 16.7, dass der Einfluss von `study_time` je nach Gruppe (Wert von `interessiert`) unterschiedlich (Teildiagramm A). Analog ist der Einfluss des Interesses (leicht) unterschiedlich, wenn man die fünf Stufen von `study_time` getrennt betrachtet.

Sind die Regressionsgerade nicht parallel, so liegt ein Interaktionseffekt vor. Andernfalls nicht.

16.8 Fallstudie zu Overfitting

Vergleichen wir im ersten Schritt eine Regression, die die Modellgüte anhand der *Trainingsstichprobe* schätzt mit einer Regression, bei der die Modellgüte in einer *Test-Stichprobe* überprüft wird.

Betrachten wir nochmal die einfache Regression von oben. Wie lautet das R^2 ?

```
lm1 <- lm(score ~ study_time, data = stats_test)
```

Es lautet `round(summary(lm1)$r.squared, 2)`.

Im zweiten Schritt teilen wir die Stichprobe in eine Trainings- und eine Test-Stichprobe auf. Wir “trainieren” das Modell anhand der Daten aus der Trainings-Stichprobe:

```
train <- stats_test %>%
  sample_frac(.8, replace = FALSE) # Stichprobe von 80%, ohne Zurücklegen

test <- stats_test %>%
  anti_join(train) # Alle Zeilen von "df", die nicht in "train" vorkommen

lm_train <- lm(score ~ study_time, data = train)
```

Dann testen wir (die Modellgüte) anhand der *Test*-Stichprobe. Also los, `lm_train`, mach Deine Vorhersage:

```
lm2_predict <- predict(lm_train, newdata = test)
```

Diese Syntax sagt:



Speichere unter dem Namen “`lm2_predict`” das Ergebnis folgender Berechnung:
Mache eine Vorhersage (“to predict”) anhand des Modells “`lm2`”,
wobei frische Daten (“`newdata = test`”) verwendet werden sollen.

Als Ergebnis bekommen wir einen Vektor, der für jede Beobachtung des Test-Samples den geschätzten (vorhergesagten) Klausurpunktewert speichert.

```
caret::postResample(pred = lm2_predict, obs = test$score)
#>      RMSE Rsquared
#>    4.331    0.345
```

Die Funktion `postResample` aus dem Paket `caret` liefert uns zentrale Gütekennzahlen unser Modell. Wir sehen, dass die Modellgüte im Test-Sample deutlich *schlechter* ist als im Trainings-Sample. Ein typischer Fall, der uns warnt, nicht vorschnell optimistisch zu sein!

Die Modellgüte im in der Test-Stichprobe ist meist schlechter als in der Trainings-Stichprobe. Das warnt uns vor Befunden, die naiv nur die Werte aus der Trainings-Stichprobe berichten.

Tabelle 16.1: Befehle des Kapitels 'Regression'

Paket..Funktion	Beschreibung
lm	Berechnet eine Regression
sqrt	Zieht die Quadratwurzel
caret::postResample	Berechnet Gütekriterien für das Testsample
summary	Fasst zentrale Informationen zu einem Objekt zusammen
modelr::add_residuals	Erstellt eine Spalte mit Residuen
modelr::add_predictions	Erstellt eine Spalte mit den vorhergesagten Werten
levels	Zeigt oder ändert die Stufen eines Faktors
factor	Erstellt einen Faktor (nominalskalierte Variable)
coef	Zeigt die Koeffizienten eines Objekts an.
step	Führt eine Schrittweise-Rückwärtsselektion durch
sample_frac	Sampelt einen Prozentsatz aus einem Datensatz
anti_join	Fügt Vereint nicht-gleiche Zeilen zweier Datensätze

16.9 Aufgaben¹



Richtig oder Falsch!?

1. X-Wert: Kriterium; Y-Wert: Prädiktor.
2. Der Y-Wert in der einfachen Regression wird berechnet als Achsenabschnitt plus x mal die Geradensteigung.
3. R^2 liefert einen *relativen* Vorhersagefehler und MSE einen *absoluten* (relativ im Sinne eines Anteils).
4. Unter 'Ordinary Least Squares' versteht man eine abschätzige Haltung gegenüber Statistik.
5. Zu den Annahmen der Regression gehört Normalverteilung der *Kriteriumswerte*.
6. Die Regression darf nicht bei kategorialen Prädiktoren verwendet werden.
7. Mehrere bivariate Regressionsanalysen (1 Prädiktor, 1 Kriterium) sind einer multivariaten Regression i.d.R. vorzuziehen.
8. Interaktionen erkennt man daran, dass die Regressionsgeraden *nicht* parallel sind.

16.10 Befehlsübersicht

Tabelle 16.1 stellt die Befehle dieses Kapitels dar.

¹F, R, R, F, F, F, F, F, R

Kapitel 17

Klassifizierende Regression



Lernziele:

- Die Idee der logistischen Regression verstehen.
- Die Koeffizienten der logistischen Regression interpretieren können.
- Die Modellgüte einer logistischen Regression einschätzen können.
- Klassifikatorische Kennzahlen kennen und beurteilen können.

Für dieses Kapitel benötigen Sie folgende Pakete:

```
library(SDMTools) # Güte von Klassifikationsmodellen
library(pROC) # für ROC- und AUC-Berechnung
library(tidyverse) # Datenjudo
library(BaylorEdPsych) # Pseudo-R-Quadrat
library(broom) # lm-Ergebnisse aufräumen
```

Hilft Lernen, eine Statistikklausur zu bestehen? Kommt es auf Interesse an? Versuchen wir vorherzusagen, wer eine Statistikklausur besteht. Etwas genauer gesagt, sagen wir ein *binäres* (dichotomes) Ereignis - Bestehen der Klausur - vorher anhand von einer mehr Variablen mit beliebigen Skalenniveau.

Laden wir die Klausurdaten.

```
stats_test <- read.csv("data/stats_test.csv")
stats_test <- na.omit(stats_test)
```

Um uns das Leben leichter zu machen, haben wir fehlende Werte (NAs) mit `na.omit` gelöscht.

17.1 Normale Regression für ein binäres Kriterium

In gewohnter Manier nutzen wir die normale Regression um das Kriterium ‘Bestehen’ anhand der Vorbreitungszeit vorherzusagen. Mit einem kleinen Trick können wir die binäre Variable `bestanden` in eine Art metrische Variable umwandeln, damit sie wieder in unser Regressions-Handwerk passt: Wenn `bestanden=="ja"` dann sei `bestanden_num = 1`; ansonsten `bestanden_num = 0`. Dieses ‘wenn-dann’ leistet der Befehl `if_else: if_else(bedingung, wenn_erfüllt, ansonsten)`. In unserem Fall sieht das so aus:

```
stats_test %>%
  mutate(bestanden_num = if_else(bestanden == "ja", 1, 0),
        bestanden_log = bestanden == "ja") -> stats_test
```

Die logistische Regression in R wünscht, dass das Kriterium eine binäre Variable (Stufen 0 und 1) oder eine logische Variable (Stufen TRUE und FALSE) sei; eine dichotome Variable (Stufen ja und nein) wird nicht goutiert.

Rechnen wir jetzt unsere Regression:

```
lm1 <- lm(bestanden_num ~ study_time, data = stats_test)
tidy(lm1)
#>           term estimate std.error statistic p.value
#> 1 (Intercept)  0.6465    0.0654     9.89 1.61e-19
#> 2 study_time   0.0666    0.0210     3.17 1.70e-03
```

OK¹. Stellen wir das Ergebnis grafisch dar (vgl. Abbildung 17.1).

```
ggplot(stats_test) +
  aes(x = study_time, y = bestanden_num) +
  geom_jitter() +
  geom_abline(slope = .07, intercept = .65, color = "red")
```

Ah! Mehr Lernen hilft offenbar: Die Regressionsgerade steigt.

Betrachten Sie diese praktische Eigenschaft der Regression: Obwohl die Kriteriumsvariable (Y-Achse) nur zwei Ausprägungen aufweist (0 und 1), sagt sich die Regression: “Hey, 0 und 1 sind normale reelle Zahlen und zwischen jedem solcher Zahlenpaare gibt es Zahlen dazwischen. Also kann ich meine Regressionserade ohne abzusetzen durchmalen”. Damit können wir die Werte zwischen 0 und 1 wie Wahrscheinlichkeiten interpretieren: Sagt die Regressionsgerade für bestimmte Prädiktorwerte hohe Kriteriumswerte voraus, so können wir sagen, die Wahrscheinlichkeit, die Klausur zu bestehen, ist hoch.

¹wir hätten auch `bestanden_log` verwenden können als Kriterium.

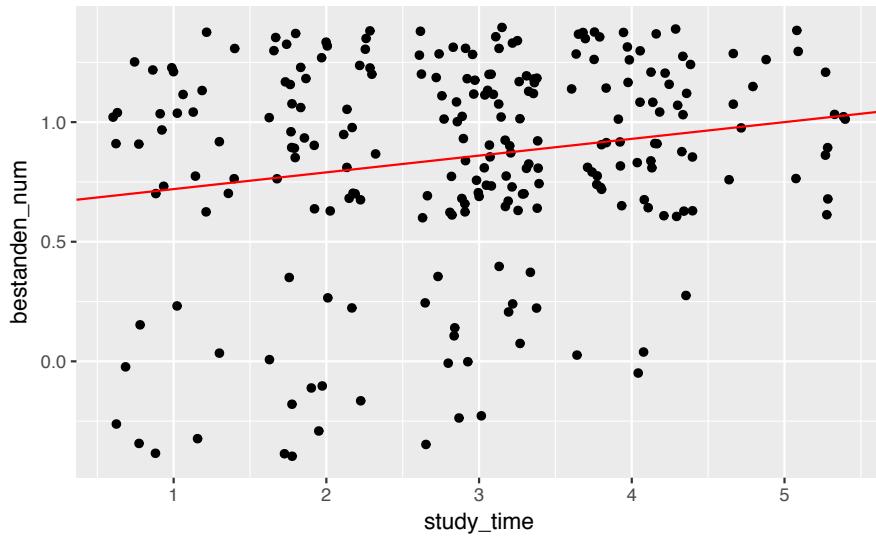


Abbildung 17.1: Regressionsgerade für das Bestehen-Modell

Soweit, so gut. Aber Moment. Was bedeutet es, wenn die Wahrscheinlichkeit größer 1 ist? Dass der Professor vorher einen eidesstattliche Erklärung für Bestehen geschickt hat? Von so etwas hat man noch nicht gehört... Kurz gesagt: Wahrscheinlichkeiten größer 1 und kleiner 0 sind Quatsch. Wahrscheinlichkeiten müssen zwischen 0 und 1 liegen.

17.2 Die logistische Funktion

Daher brauchen wir eine Funktion, die das Ergebnis einer linearen Regression in einen Bereich von 0 bis 1 „umbiegt“ (die sogenannte *Linkfunktion*). Eine häufig dafür verwendete Funktion ist die *logistische Funktion*. Im einfachsten Fall:

$$p(y = 1) = \frac{e^x}{1 + e^x} = \frac{e^x}{e^x(\frac{1}{e^x} + 1)} = \frac{1}{\frac{1}{e^x} + 1} = \frac{1}{e^{-x} + 1}$$

Exemplarisch können wir die logistische Funktion für einen Bereich von $x = -10$ bis $x = +10$ darstellen (vgl. 17.2). Der Graph der logistischen Funktion ähnelt einem langgestreckten S (“Ogive” genannt).

17.3 Die Idee der logistischen Regression

Die logistische Regression ist eine Anwendung des *Allgemeinen Linearen Modells* (general linear model, GLM). Die Modellgleichung lautet:

$$p(y_i = 1) = L(\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_K \cdot x_{ik}) + \epsilon_i$$

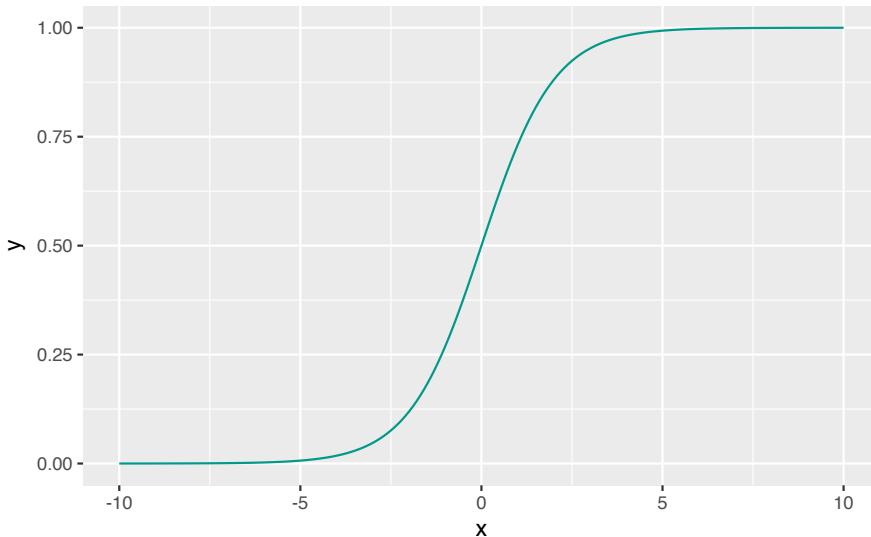


Abbildung 17.2: Die logistische Regression beschreibt eine 's-förmige' Kurve

- L ist die Linkfunktion, in unserer Anwendung die logistische Funktion.
- x_{ik} sind die beobachteten Werte der unabhängigen Variablen X_k .
- k sind die unabhängigen Variablen 1 bis K .

Die Funktion `glm` führt die logistische Regression durch.

```
glm1 <- glm(bestanden_num ~ study_time,
              family = "binomial",
              data = stats_test)
```

Wir schauen uns zunächst den Plot an (Abb. 17.3).

Es werden ein Streudiagramm der beobachteten Werte sowie die *Regressionslinie* ausgegeben. Wir können so z. B. ablesen, dass mit einer Lernzeit von 5 die Wahrscheinlichkeit für Bestehen bei knapp 100% liegt; viel zu einfach. . .

Die Zusammenfassung des Modells zeigt folgendes:

```
tidy(glm1)
#>   term estimate std.error statistic p.value
#> 1 (Intercept)  0.276    0.458     0.602  0.54698
#> 2 study_time   0.513    0.168     3.049  0.00229
```

Die p-Werte der Koeffizienten können in der Spalte $\text{Pr}(>|z|)$ abgelesen werden. Der Achsenabschnitt (`intercept`) wird mit 0.28 geschätzt, die Steigung in Richtung `study_time` mit

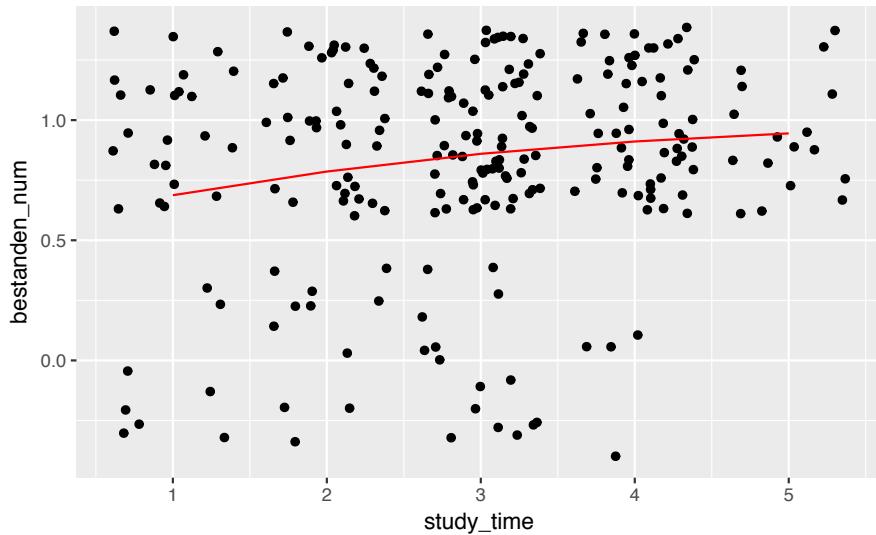


Abbildung 17.3: Modelldiagramm mit logistischer Regression

0.51. Allerdings sind die hier dargestellten Werte sogenannte *Logits* \mathfrak{L} ²:

$$\mathfrak{L} = \ln\left(\frac{p}{1-p}\right)$$

Zugeben, dass klingt erstmal opaque. Das Praktische ist, dass wir die Koeffizienten in Logitform in gewohnter Manier verrechnen dürfen. Wollen wir zum Beispiel wissen, wie wahrscheinlich das Ereignis ‘Bestehen’ für eine Person mit einer Lernzeit von 3 ist, können wir einfach rechnen:

`y = intercept + 3*study_time`, also

```
(y <- .27 + 3 * 0.51)
#> [1] 1.8
```

Einfach, oder? Genau wie bei der normalen Regression. Aber beachten Sie, dass das Ergebnis in *Logits* angegeben ist. Was ein *Logit* ist? Naja, das ist der *Logarithmus der Chancen*; unter ‘Chancen’ versteht man den Quotienten von Wahrscheinlichkeit p zur Gegenwahrscheinlichkeit, $1 - p$; die Chancen werden auch *Odds* oder *Wettquotient* genannt.

Um zur ‘normalen’ Wahrscheinlichkeit zu kommen, muss man also erst ‘deologarithmieren’. Deologarithmieren bedeutet, die e-Funktion anzuwenden, `exp` auf Errisch:

```
exp(y)
#> [1] 6.05
```

Jetzt haben wir wir also Chancen. Wie rechnet man Chancen in Wahrscheinlichkeiten um? Ein Beispiel zur Illustration. Bei Prof. Schnaggeldi fallen von 10 Studenten 9 durch. Die

²ein schnödes L wie in Ludwig

Durchfall *chance* ist also 9:1 oder 9. Die Durchfall *wahrscheinlichkeit* 9/10 oder .9. Also kann man so umrechnen:

$$\text{wskt} = 9 / (9+1) = 9/10 = .9.$$

In unserem Fall sind die Chancen etwa 6:1; also lautet die Umrechnung:

```
(wskt <- 6 / (6+1))
#> [1] 0.857
```

Diesen Ritt kann man sich merklich kommoder bereiten, wenn man diesen Befehl kennt:

```
predict(glm1, newdata = data.frame(study_time = 3), type = "response")
#>      1
#> 0.86
```

17.4 Kein R^2 , dafür AIC

Es gibt kein R^2 im Sinne einer erklärten Streuung der y -Werte, da die beobachteten y -Werte nur 0 oder 1 annehmen können. Das Gütemaß bei der logistischen Regression ist das *Akaike Information Criterion (AIC)*. Hier gilt allerdings: je *kleiner*, desto *besser*. (Anmerkung: es kann ein Pseudo- R^2 berechnet werden – kommt später.) Richtlinien, was ein “guter” AIC-Wert ist, gibt es nicht. Diese Werte helfen nur beim Vergleichen von Modellen.

17.5 Interpretation der Koeffizienten

Ist ein Logit \mathfrak{L} größer als 0, so ist die zugehörige Wahrscheinlichkeit größer als 50% (und umgekehrt.)

17.5.1 y-Achsenabschnitt (Intercept) β_0

Für $\beta_0 > 0$ gilt, dass selbst wenn alle anderen unabhängigen Variablen 0 sind, es eine Wahrscheinlichkeit von mehr als 50% gibt, dass das modellierte Ereignis eintritt. Für $\beta_0 < 0$ gilt entsprechend das Umgekehrte.

17.5.2 Steigung β_i mit $i = 1, 2, \dots, K$

Für $\beta_i > 0$ gilt, dass mit zunehmenden x_i die Wahrscheinlichkeit für das modellierte Ereignis steigt. Bei $\beta_i < 0$ nimmt die Wahrscheinlichkeit entsprechend ab.

17.5.3 Aufgabe

Berechnen Sie den Zuwachs an Wahrscheinlichkeit für unser Beispielmodell, wenn sich die `study_time` von 1 auf 2 erhöht. Vergleichen Sie das Ergebnis mit der Punktprognose für `study_time= 7` im Vergleich zu `study_time= 8`.

Lösung:

```
# aus Koeffizient abgeschätzt
wskt1 <- predict(glm1, data.frame(study_time = 1), type = "response")

wskt2 <- predict(glm1, data.frame(study_time = 2), type = "response")

wskt2 - wsbt1
#>      1
#> 0.0985
```

Anders gesagt: “Mit jedem Punkt mehr ‘study_time’ steigt der Logit (die logarithmierten Chancen) für Bestehen um 0.513”.

```
# mit dem vollständigen Modell berechnet
predict(glm1, data.frame(study_time = 1),
        type = "response")
#>      1
#> 0.688

predict(glm1, data.frame(study_time = 8),
        type = "response")
#>      1
#> 0.988
```

Bei einer `study_time` von 4 beträgt die Wahrscheinlichkeit für $y = 1$, d.h. für das Ereignis ‘Bestehen’, 0.91. Bei einer `study_time` von 58 liegt diese Wahrscheinlichkeit bei 0.94.

17.6 Kategoriale Prädiktoren

Wie in der linearen Regression können auch in der logistischen Regression kategoriale Variablen als unabhängige Variablen genutzt werden.

Betrachten wir als Beispiel die Frage, ob die kategoriale Variable “Interessiert” (genauer: dichotome Variable) einen Einfluss auf das Bestehen in der Klausur hat, also die Wahrscheinlichkeit für Bestehen erhöht.

```
stats_test$interessiert <- stats_test$interest > 3
```

Erstellen Sie zum Aufwärmen ein passendes Diagramm!

Los geht's, probieren wir die logistische Regression aus:

```
glm2 <- glm(bestanden_num ~ interessiert,
             family = "binomial",
             data = stats_test)
tidy(glm2)
#>   term estimate std.error statistic p.value
#> 1 (Intercept)    1.50     0.217      6.94 4.00e-12
#> 2 interessiertTRUE    0.43     0.377      1.14 2.55e-01
```

Der Einflusswert (die Steigung) von `interessiert` ist positiv: Wenn man interessiert ist, steigt die Wahrscheinlichkeit zu bestehen. Gut. Aber wie groß ist die Wahrscheinlichkeit für jede Gruppe? Am einfachsten lässt man sich das von R ausrechnen:

```
predict(glm2, newdata = data.frame(interessiert = FALSE),
       type = "response")
#> 1
#> 0.818
predict(glm2, newdata = data.frame(interessiert = TRUE),
       type = "response")
#> 1
#> 0.874
```

Also 82% bzw. 87%; kein gewaltig großer Unterschied, aber immerhin...

17.7 Multiple logistische Regression

Können wir unser Model `glm1` mit nur einer erklärenden Variable verbessern, indem weiterer Prädiktoren hinzugefügt werden? Verbessern heißt hier: Können wir die Präzision der Vorhersage verbessern durch Hinzunahme weiterer Prädiktoren?

```
glm3 <- glm(bestanden_num ~ study_time + interest + self_eval,
             family = binomial,
             data = stats_test)
tidy(glm3)
#>   term estimate std.error statistic p.value
```

```
#> 1 (Intercept) -0.202      0.550     -0.367 0.71359
#> 2 study_time   0.115      0.218      0.529 0.59665
#> 3 interest    -0.155      0.155     -0.998 0.31820
#> 4 self_eval    0.447      0.107      4.173 0.00003
```

Hm, die Interessierten schneiden jetzt - unter Konstanthalten anderer Einflussfaktoren - *schlechter* ab als die Nicht-Interessierten. Als Statistik-Dozent bin ich der Meinung, dieses Ergebnis sollte in der Schubladen verschwinden (wie es geläufige Praxis ist in vielen Laboren...).

17.8 Modellgüte

Aber wie gut ist das Modell? Und welches Modell von beiden ist besser? R hat uns kein R^2 ausgegeben. R hat uns deswegen kein R^2 ausgegeben, weil die Regressionsfunktion nicht über Abweichungsquadrate bestimmt wird. Stattdessen wird das Maximum Likelihood-Verfahren eingesetzt. Man kann also kein R^2 ausrechnen, zumindest nicht ohne Tricks. Einige findige Statistiker haben sich aber Umrechnungswege einfallen lassen, wie man auch ohne Abweichungsquadrate ein R^2 berechnen kann; weil es kein ‘echtes’ R^2 ist, nennt man es auch *Pseudo-R²*. Es gibt ein paar Varianten, wir bleiben bei der Variante von Herrn McFadden (s. Ausgabe).

Eine Reihe von R-Paketen bieten die Berechnung a:

```
library(BaylorEdPsych)
PseudoR2(glm1)
PseudoR2(glm2)
PseudoR2(glm3)
```

Die Ausgabe zeigt uns, dass das erste Modell schon schlecht ist, dass zweite praktisch keinen Erklärungswert und das dritte einen zumindest kleinen bis mittleren Erklärungswert bietet: $f^2 = \frac{R^2}{1-R^2} \approx \frac{.1}{.9} = .11$.

17.8.1 Vier Arten von Ergebnissen einer Klassifikation

Logistische Regressionsmodelle werden häufig zur *Klassifikation* verwendet. Das heißt man versucht, Beobachtungen richtig zu zu Klassen zuzuordnen:

- Ein medizinischer Test soll Kranke als krank und Gesunde als gesund klassifizieren.
- Ein statistischer Test sollte wahre Hypothesen als wahr und falsche Hypothesen als falsch klassifizieren.
- Ein Personaler sollte geeignete Bewerber als geeignet und nicht geeignete Bewerber als nicht geeignet einstufen.

Tabelle 17.1: Vier Arten von Ergebnisse von Klassifikationen

Wahrheit	Als negativ (-) vorhergesagt	Als positiv (+) vorhergesagt	Summe
In Wahrheit negativ (-)	Richtig negativ (RN)	Falsch positiv (FP)	N
In Wahrheit positiv (+)	Falsch negativ (FN)	Richtig positiv (RN)	P
Summe	N*	P*	N+P

Diese beiden Arten von Klassifikationen können unterschiedlich gut sein. Im Extremfall könnte ein Test alle Menschen als krank ('positiv') einstufen. Mit Sicherheit wurden dann alle Kranken korrekt als krank diagnostiziert. Dummerweise würde der Test 'auf der anderen Seite' viele Fehler machen: Gesunde als gesund ('negativ') zu klassifizieren.

Ein Test, der alle positiven Fälle korrekt als positiv klassifiziert muss deshalb noch lange nicht alle negativen Fälle als negativ klassifizieren. Die beiden Werte können unterschiedlich sein.

Etwas genauer kann man folgende vier Arten von Ergebnisse aus einem Test erwarten (s. Tabelle 17.1, vgl. James, Witten, Hastie, und Tibshirani (2013b)).

Die logistische Regression gibt uns für jeden Fall eine Wahrscheinlichkeit zurück, dass der Fall zum Ereignis 1 gehört. Wir müssen dann einen Schwellenwert (threshold) auswählen. Einen Wert also, der bestimmt, ob der Fall zum Ereignis 1 gehört. Häufig nimmt man 0.5. Liegt die Wahrscheinlichkeit unter dem Schwellenwert, so ordnet man den Fall dem Ereignis 0 zu.

Beispiel: Alois' Wahrscheinlichkeit, die Klausur zu bestehen, wird vom Regressionsmodell auf 51% geschätzt. Unser Schwellenwert sei 50%; wir ordnen Alois der Klasse "bestehen" zu. Alois freut sich. Das Modell sagt also "bestehen" (1) für Alois voraus. Man sagt auch, der 'geschätzte Wert' (*fitted value*) von Alois sei 1.

Die aus dem Modell ermittelten Wahrscheinlichkeiten werden dann in einer sogenannten Konfusionsmatrix (*confusion matrix*) mit den beobachteten Häufigkeiten verglichen:

```
(cm <- confusion.matrix(stats_test$bestanden_num,
                         glm3$fitted.values))

#>      obs
#> pred  0   1
#>     0  1   1
#>     1 37 199
#> attr(,"class")
#> [1] "confusion.matrix"
```

Dabei stehen **obs** (observed) für die wahren, also tatsächlich beobachteten Werte und **pred** (predicted) für die geschätzten (vorhergesagten) Werte.

Wie häufig hat unser Modell richtig geschätzt? Genauer: Wie viele echte 1 hat unser Modell als 1 vorausgesagt und wie viele echte 0 hat unser Modell als 0 vorausgesagt?

Tabelle 17.2: Geläufige Kennwerte der Klassifikation. Anmerkungen: F: Falsch. R: Richtig. P: Positiv. N: Negativ.

Name	Definition	Synonyme
FP-Rate	FP/N	Alphafehler, Typ-1-Fehler, 1-Spezifität, Fehlalarm
RP-Rate	RP/N	Power, Sensitivität, 1-Betafehler, Recall
FN-Rate	FN/N	Fehlender Alarm, Befafehler
RN-Rate	RN/N	Spezifität, 1-Alphafehler
Pos. Vorhersagewert	RP/P*	Präzision, Relevanz
Neg. Vorhersagewert	RN/N*	Segreganz
Richtigkeit	(RP+RN) / (N+P)	Korrektklassifikationsrate

17.8.2 Klassifikationsgütekennzahlen

In der Literatur und Praxis herrscht eine recht wilde Vielfalt an Begriffen dazu, deswegen stellt Tabelle 17.1 einen Überblick vor.

Zu beachten ist, dass die Gesamtgenauigkeit einer Klassifikation an sich wenig aussagekräftig ist: Ist eine Krankheit sehr selten, werde ich durch die einfache Strategie “diagnostiziere alle als gesund” insgesamt kaum Fehler machen. Meine Gesamtgenauigkeit wird beeindruckend genau sein - trotzdem lassen Sie sich davon wohl kaum beeindrucken. Besser ist, die Richtig-Positiv- und die Richtig-Negativ-Raten getrennt zu beurteilen. Aus dieser Kombination leitet sich der *Youden-Index* ab.. Er berechnet sich als: RP-Rate + RN-Rate - 1.

Sie können die Konfusionsmatrix mit dem Paket `confusion.matrix()` aus dem Paket `SDMTools` berechnen.

```
sensitivity(cm)
#> [1] 0.995
specificity(cm)
#> [1] 0.0263
```

Unser Modell hat es sich recht leicht gemacht: Es hat immer auf ‘bestanden’ getippt: Damit wurden alle ‘Besteher’ korrekt identifiziert (Sensitivität = 1); allerdings wurden auch alle ‘Nicht-Besteher’ übersehen (Spezifität = 0).

Wir könnten jetzt sagen, dass wir im Zweifel lieber eine Person als Nicht-Besteher einschätzen (um die Lernschwachen noch unterstützen zu können). Dazu würden wir den Schwellenwert (threshold) von 50% auf z.B. 80% heraufsetzen. Erst bei Erreichen des Schwellenwerts klassifizieren wir die Beobachtung als ‘bestanden’ (1):

```
(cm <- confusion.matrix(stats_test$bestanden_num,
                         glm3$fitted.values, threshold = .8))
#>      obs
#> pred  0   1
```

```
#>      0 24 47
#>      1 14 153
#> attr(,"class")
#> [1] "confusion.matrix"
sensitivity(cm)
#> [1] 0.765
specificity(cm)
#> [1] 0.632
```

17.8.3 ROC-Kurven

Siehe da! Die Spezifität ist gestiegen, wir haben mehr Nicht-Lerner als solche identifiziert. Unsere liberalere Strategie hat aber mehr Falsch-Negative Fälle produziert (geringere Sensitivität). So können wir jetzt viele verschiedene Schwellenwerte vergleichen.

Ein Test ist dann gut, wenn wir für alle möglichen Schwellenwert insgesamt wenig Fehler produziert.

Hierzu wird der Cutpoint zwischen 0 und 1 variiert und die Richtig-Positiv-Rate (Sensitivität) gegen die Falsch-Positiv-Rate (1 – Spezifität) abgetragen. Das Paket `pROC` hilft uns hier weiter. Zuerst berechnen wir für viele verschiedene Schwellenwerte jeweils die beiden Fehler (Falsch-Positiv-Rate und Falsch-Negativ-Rate). Trägt man diese in ein Diagramm ab, so bekommt man Abbildung 17.4, eine sog. *ROC-Kurve*.

```
lets_roc <- roc(stats_test$bestanden_num, glm3$fitted.values)
```

Da die Sensitivität determiniert ist, wenn die Falsch-Positiv-Rate bekannt ist ($1 - \text{FP-Rate}$), kann man statt Sensitivität auch die FP-Rate abbilden. Für die Spezifität und die Falsch-Negativ-Rate gilt das gleiche. In Abbildung 17.4 steht auf der X-Achse Spezifität, aber die Achse ist ‘rückwärts’ (absteigend) skaliert, so dass die X-Achse identisch ist mit FP-Rate (normal skaliert; d.h. aufsteigend).

```
plot(lets_roc)
```

Die ‘Fläche unter der Kurve’ (area under curve, AUC) ist damit ein Maß für die Güte des Tests. Abbildung 17.5 stellt drei Beispiele von Klassifikationsgüten dar: sehr gute (A), gute (B) und schlechte (C). Ein hohe Klassifikationsgüte zeigt sich daran, dass eine hohe Richtig-Positiv-Rate mit einer kleinen Fehlalarmquote einher geht: Wir finden alle Kranken, aber nur die Kranken. Die AUC-Kurve “hängt oben links an der Decke”. Ein schlechter Klassifikator trifft so gut wie ein Münzwurf: Ist das Ereignis selten, hat er eine hohe Falsch-Positiv-Rate und eine geringe Falsch-Negativ-Rate. Ist das Ereignis hingegen häufig, liegen die Fehlerhöhen

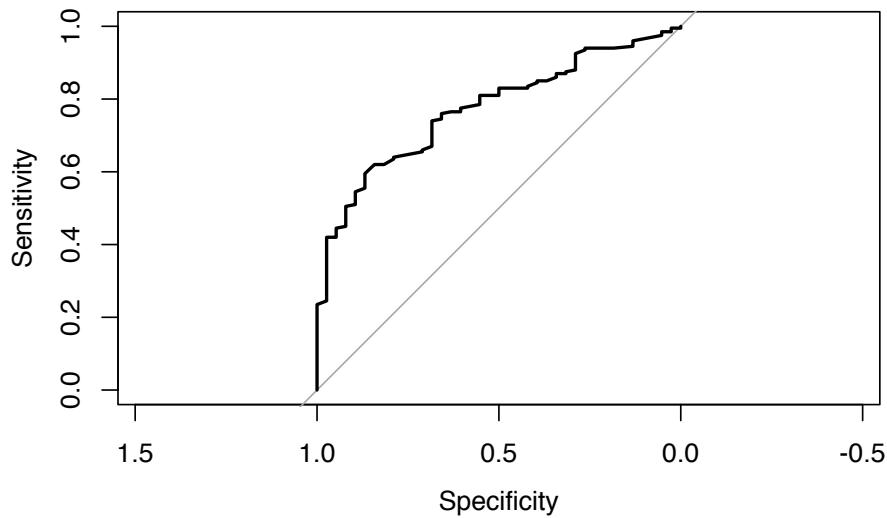


Abbildung 17.4: Eine ROC-Kurve

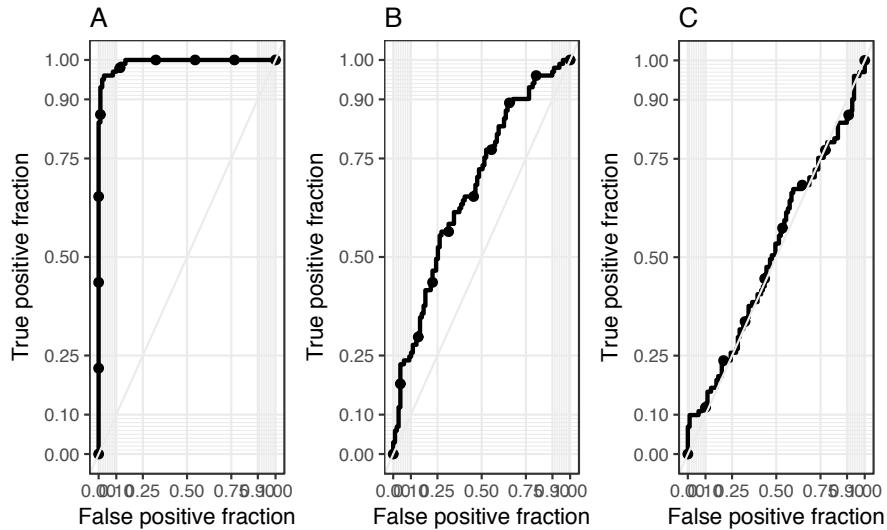


Abbildung 17.5: Beispiel für eine sehr gute (A), gute (B) und schlechte (C) Klassifikation

genau umgekehrt: Eine hohe Richtig-Positiv-Rate wird mit einer hoher Falsch-Positiv-Rate einher.

Fragt sich noch, wie man den besten Schwellenwert herausfindet. Den besten Schwellenwert kann man als besten Youden-Index-Wert verstehen. Im Paket pROC gibt es dafür den Befehl `coords`, der uns im ROC-Diagramm die Koordinaten des besten Schwellenwert und den Wert dieses besten Schwellenwerts liefert:

```
coords(lets_roc, "best")
#> threshold specificity sensitivity
#>      0.874       0.868       0.595
```

Tabelle 17.3: Befehle des Kapitels 'Logistische Regression'

Paket..Funktion	Beschreibung
ggplot2::geom_abline	Fügt das Geom "abline" (normale Gerade) hinzu
glm	Berechnet eine logistische Regression
exp	Berechnet die e-Funktion
SDMTools::confusion.matrix	Berechnet eine Konfusionsmatrix
SDMTools::sensitivity	Berechnet die Sensitivität eines Klassifikationsmodells
SDMTools::specificity	Berechnet die Spezifität eines Klassifikationsmodells
ROCR::performance	Erstellt Objekte mit Gütekennzahlen von Klassifikationsmodellen
BaylorEdPsych::PseudoR2	Berechnet Pseudo-R-Quadrat-Werte

17.9 Aufgaben³



Richtig oder Falsch!?

1. Die logistische Regression ist eine Regression für dichotome Kriterien.
2. Unter einer OliveOgive versteht man eine eine "s-förmige" Kurve.
3. Berechnet man eine "normale" (OLS-)Regression bei einem dichotomen Kriterium, so kann man Wahrscheinlichkeiten < 0 oder > 1 erhalten, was keinen Sinn macht.
4. Ein Logit ist definiert als der Einfluss eines Prädiktors in der logistischen Regression. Der Koeffizient berechnet sich als Logarithmus des Wettquotienten.
5. Das AIC ein Gütemaß, welches man bei der logistischen Regression generell vermeidet.
6. Eine Klassifikation kann 4 Arten von Ergebnissen bringen - gemessen an der Richtigkeit des Ergebnisses.
7. Der 'positive Vorhersagewert' ist definiert als der Anteil aller richtig-positiven Klassifikationen an allen als positiv klassifizierten Objekten.

17.10 Befehlsübersicht

Tabelle 17.3 stellt die Befehle dieses Kapitels dar.

³R, R, R, R, F, R, R

Kapitel 18

Fallstudien zum geleiteten Modellieren

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse)  # Datenjudo
library(psych)    # Befehl 'describe'
library(broom)     # lm-Ergebnisse aufpolieren
library(corrplot)  # Korrelationstabellen visualisieren
library(titanic)   # Für Datensatz 'titanic'
library(compute.es) # Effektstärken berechnen
```

18.1 Überleben auf der Titanic

In dieser YACSDA¹ geht es um die beispielhafte Analyse nominaler Daten anhand des “klassischen” Falls zum Untergang der Titanic. Eine Frage, die sich hier aufdrängt, lautet: Kann (konnte) man sich vom Tod freikaufen, etwas polemisch formuliert. Oder neutraler: Hängt die Überlebensquote von der Klasse, in der der Passagiers reist, ab?

18.1.1 Daten laden

Mit dem Befehl `data` kann man Daten aus Paketen laden; lässt man den Parameter `package` weg, so werden alle geladenen Pakete nach diesem Datensatz durchsucht. Benennt man den Parameter, so kann man auch *nicht* geladene Pakete damit ansteuern.

¹Yet-another-case-study-on-data-analysis

```
data(titanic_train, package = "titanic")
titanic_train <- na.omit(titanic_train)
```

18.1.2 Erster Blick

Werfen Sie einen ersten Blick in die Daten mit `glimpse(titanic_train)`. Lassen Sie sich dann einige deskriptive Statistiken ausgeben²

18.1.3 Welche Variablen sind interessant?

Von 12 Variablen des Datensatzes interessieren uns offenbar `Pclass` und `Survived`; Hilfe zum Datensatz kann man übrigens mit `help(titanic_train)` bekommen. Diese beiden Variablen sind kategorial (nicht-metrisch), wobei sie in der Tabelle mit Zahlen kodiert sind. Natürlich ändert die Art der Codierung (hier als Zahl) nichts am eigentlichen Skalenniveau. Genauso könnte man “Mann” mit 1 und “Frau” mit 2 kodieren; ein Mittelwert bliebe genauso (wenig) aussagekräftig. Zu beachten ist hier nur, dass sich manche R-Befehle verunsichern lassen, wenn nominale Variablen mit Zahlen kodiert sind. Daher ist es oft besser, nominale Variablen mit Text-Werten zu benennen (wie “survived” vs. “drowned” etc.). Wir kommen später auf diesen Punkt zurück.

18.1.4 Univariate Häufigkeiten

Bevor wir uns in kompliziertere Fragestellungen stürzen, halten wir fest: Wir untersuchen zwei nominale Variablen. Sprich: wir werden Häufigkeiten auszählen. Häufigkeiten (und relative Häufigkeiten, also Anteile oder Quoten) sind das, was uns hier beschäftigt.

Zählen wir zuerst die univariaten Häufigkeiten aus: Wie viele Passagiere gab es pro Klasse? Wie viele Passagiere gab es pro Wert von `Survived` (also die überlebten bzw. nicht überlebten)?

```
c1 <- dplyr::count(titanic_train, Pclass)
c1
#> # A tibble: 3 x 2
#>   Pclass     n
#>   <int> <int>
#> 1     1    186
#> 2     2    173
#> 3     3    355
```

²z.B. mit `titanic_train %>% count(Survived)` oder `titanic_train %>% summarise(Ticketpreis = mean(Fare, na.rm = TRUE))`



Achtung - Namenskollision! Sowohl im Paket `mosaic` als auch im Paket `dplyr` gibt es einen Befehl `count`. Für `select` gilt Ähnliches - und für eine Reihe anderer Befehle auch. Das arme R weiß nicht, welchen von beiden wir meinen und entscheidet sich im Zweifel für den falschen. Da hilft, zu sagen, aus welchem Paket wir den Befehl beziehen wollen. Das macht der Operator `:::`. Probieren Sie die Funktion `find_funs` aus Kapitel 3.4, um herauszufinden, welche Pakete z.B. den Befehl `count` beherbergen.

Aha. Zur besseren Anschaulichkeit können Sie das auch plotten (ein Diagramm dazu malen). Wie?³

Der Befehl `qplot` zeichnet automatisch Punkte, wenn auf beiden Achsen “Zahlen-Variablen” stehen (also Variablen, die keinen “Text”, sondern nur Zahlen beinhalten. In R sind das Variablen vom Typ `int` (integer), also Ganze Zahlen oder vom Typ `num` (numeric), also reelle Zahlen).

```
c2 <- dplyr::count(titanic_train, Survived)
c2
#> # A tibble: 2 x 2
#>   Survived     n
#>   <int> <int>
#> 1     0    424
#> 2     1    290
```

Man beachte, dass der Befehl `count` steht eine Tabelle (data.frame bzw. `tibble`) verlangt und zurückliefert.

18.1.5 Bivariate Häufigkeiten

OK, gut. Jetzt wissen wir die Häufigkeiten pro Wert von `Survived` (dasselbe gilt für `Pclass`). Eigentlich interessiert uns aber die Frage, ob sich die relativen Häufigkeiten der Stufen von `Pclass` innerhalb der Stufen von `Survived` unterscheiden. Einfacher gesagt: Ist der Anteil der Überlebenden in der 1. Klasse größer als in der 3. Klasse?

Zählen wir zuerst die Häufigkeiten für alle Kombinationen von `Survived` und `Pclass`:

```
c3 <- dplyr::count(titanic_train, Survived, Pclass)
c3
#> # A tibble: 6 x 3
#>   Survived Pclass     n
#>   <int> <int> <int>
#> 1     0     1     64
```

³`qplot(x = Pclass, y = n, data = c1)`

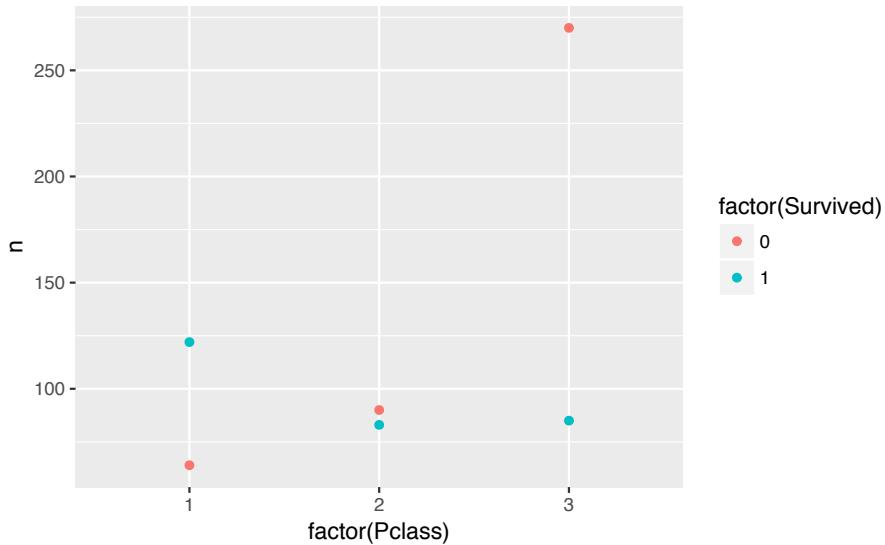


Abbildung 18.1: Überlebensraten auf der Titanic, in Abhängigkeit von der Passagierklasse

```
#> 2      0      2     90
#> 3      0      3    270
#> 4      1      1    122
#> 5      1      2     83
#> 6      1      3     85
```

Da `Pclass` 3 Stufen hat (1., 2. und 3. Klasse) und innerhalb jeder dieser 3 Klassen es die Gruppe der Überlebenden und der Nicht-Überlebenden gibt, haben wir insgesamt $3 \times 2 = 6$ Gruppen.

Es ist hilfreich, sich diese Häufigkeiten wiederum zu plotten; probieren Sie `qplot(x = Pclass, y = n, data = c3)`.

Hm, nicht so hilfreich. Schöner wäre, wenn wir (farblich) erkennen könnten, welcher Punkt für “Überlebt” und welcher Punkt für “Nicht-Überlebt” steht. Mit `qplot` geht das recht einfach: Wir sagen der Funktion `qplot`, dass die Farbe (`color`) der Punkte den Stufen von `Survived` zugeordnet werden sollen: `qplot(x = Pclass, y = n, color = Survived, data = c3)`.

Viel besser. Was noch stört, ist, dass `Survived` als metrische Variable verstanden wird. Das Farbschema lässt Nuancen, feine Farbschattierungen, zu. Für nominale Variablen macht das keinen Sinn; es gibt da keine Zwischentöne. Tot ist tot, lebendig ist lebendig. Wir sollten daher der Funktion sagen, dass es sich um nominale Variablen handelt (s. Abbildung 18.1).

```
qplot(x = factor(Pclass), y = n, color = factor(Survived), data = c3)
```

Viel besser. Jetzt fügen Sie noch ein bisschen Schnickschnack hinzu:

```
qplot(x = factor(Pclass), y = n, color = factor(Survived), data = c3) +
  labs(x = "Klasse",
       title = "Überleben auf der Titanic",
       colour = "Überlebt?")
```

18.1.6 Signifikanztest

Manche Leute mögen Signifikanztests. Ich persönlich stehe ihnen kritisch gegenüber, da ein p-Wert eine Funktion der Stichprobengröße ist und außerdem zumeist missverstanden wird (und er gibt *nicht* die Wahrscheinlichkeit der getesteten Hypothese an, was die Frage aufwirft, warum er mich dann interessieren sollte). Aber seid drüber, berechnen wir mal einen p-Wert. Es gibt mehrere statistische Tests, die sich hier potenziell anbieten und unterschiedliche Ergebnisse liefern können (Briggs 2008a) (was die Frage nach der Objektivität von statistischen Tests in ein ungünstiges Licht rückt). Nehmen wir den χ^2 -Test.

```
chisq.test(titanic_train$Survived, titanic_train$Pclass)
#>
#> Pearson's Chi-squared test
#>
#> data: titanic_train$Survived and titanic_train$Pclass
#> X-squared = 90, df = 2, p-value <2e-16
```

Der p-Wert ist kleiner als 5%, daher entscheiden wir uns, entsprechend der üblichen Gepflogenheit, gegen die H₀ und für die H₁: “Es gibt einen Zusammenhang von Überlebensrate und Passagierklasse”.

18.1.7 Effektstärke

Abgesehen von der Signifikanz, und interessanter, ist die Frage, wie sehr die Variablen zusammenhängen. Für Häufigkeitsanalysen mit 2*2-Feldern bietet sich das “Odds Ratio” (OR), das Chancenverhältnis an. Das Chancen-Verhältnis beantwortet die Frage: “Um welchen Faktor ist die Überlebenschance in der einen Klasse größer als in der anderen Klasse?”. Eine interessante Frage, als schauen wir es uns an.

Das OR ist nur definiert für 2*2-Häufigkeitstabellen, daher müssen wir die Anzahl der Passagierklassen von 3 auf 2 verringern. Nehmen wir nur 1. und 3. Klasse, um den vermuteten Effekt deutlich herauszuschälen:

```
t2 <- filter(titanic_train, Pclass != 2) # "!=" heißt "nicht"
```

Alternativ (synonym) könnten wir auch schreiben:

```
t2 <- filter(titanic_train, Pclass == 1 | Pclass == 3) # "/" heißt "oder"
```

Und dann zählen wir wieder die Häufigkeiten aus pro Gruppe:

```
c4 <- dplyr::count(t2, Pclass)
c4
#> # A tibble: 2 x 2
#>   Pclass     n
#>   <int> <int>
#> 1     1    186
#> 2     3    355
```

Schauen wir nochmal den p-Wert an, da wir jetzt ja mit einer veränderten Datentabelle operieren:

```
chisq.test(t2$Survived, t2$Pclass)
#>
#> Pearson's Chi-squared test with Yates' continuity correction
#>
#> data: t2$Survived and t2$Pclass
#> X-squared = 90, df = 1, p-value <2e-16
```

Ein χ^2 -Wert von ~96 bei einem n von 707.

Dann berechnen wir die Effektstärke (OR) mit dem Paket `compute.es` (muss ebenfalls installiert sein)

```
compute.es::chies(chi.sq = 96, n = 707)
```

Das OR beträgt also etwa 4.21. Die Chance zu überleben ist also in der 1. Klasse mehr als 4 mal so hoch wie in der 3. Klasse. Es scheint: Money buys you life...

18.1.8 Logististische Regression

Berechnen wir noch das Odds Ratio mit Hilfe der logistischen Regression. Zum Einstieg: Ignorieren Sie die folgende Syntax und schauen Sie sich das Diagramm an. Hier sehen wir die (geschätzten) Überlebens-Wahrscheinlichkeiten für Passagiere der 1. Klasse vs. Passagiere der 2. vs. der 3. Klasse.

```
glm1 <- glm(data = titanic_train,
            formula = Survived ~ Pclass,
```

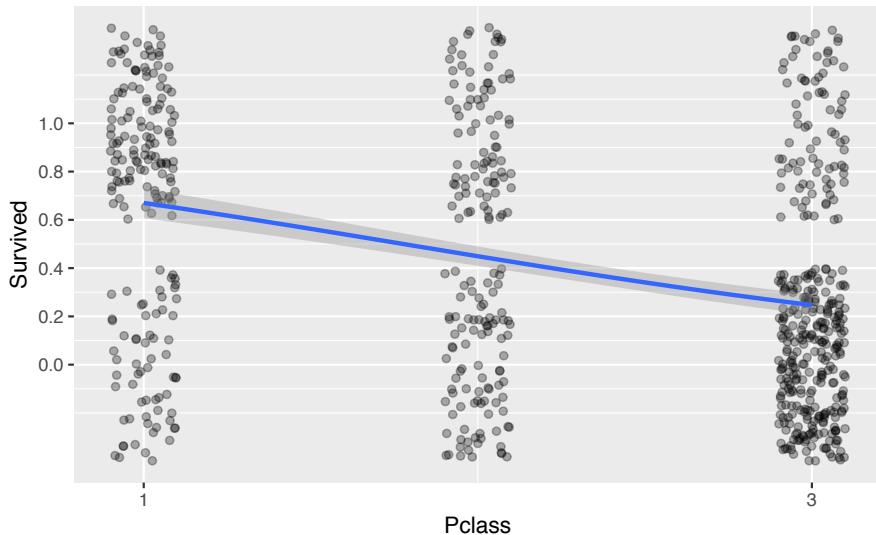


Abbildung 18.2: Logistische Regression zur Überlebensrate nach Passagierklasse

```

family = "binomial")

exp(coef(glm1))
#> (Intercept)      Pclass
#>      5.056       0.402

titanic_train$pred_prob <- predict(glm1, type = "response")

```

Wir sehen, dass die Überlebens-Wahrscheinlichkeit in der 1. Klasse höher ist als in der 3. Klasse. Optisch grob geschätzt, ~60% in der 1. Klasse und ~25% in der 3. Klasse.

Schauen wir uns die logistische Regression an: Zuerst haben wir den Datensatz auf die Zeilen beschränkt, in denen Personen aus der 1. und 3. Klasse vermerkt sind (zwecks Vergleichbarkeit zu oben). Dann haben wir mit `glm` und `family = "binomial"` eine *logistische* Regression angefordert. Man beachte, dass der Befehl sehr ähnlich zur normalen Regression (`lm(...)`) ist.

Da die Koeffizienten in der Logit-Form zurückgegeben werden, haben wir sie mit der Exponential-Funktion in die “normale” Odds-Form gebracht (deologarithmiert, *boa*) mithilfe von `exp(coef)`. Wir sehen, dass sich die Überlebens-*Chance* (Odds; nicht Wahrscheinlichkeit) um den Faktor .4 verringert pro zusätzlicher Stufe der Passagierklasse. Würde jemand in der “nullten” Klasse fahren, wäre seine Überlebenschance ca. 5:1 (5/6, gut 80%). Die Überlebenschance in der 1. Klasse sind demnach etwa: 5 * 0.4, also 2:1, etwa 67%.

Komfortabler können wir uns die Überlebens-*Wahrscheinlichkeiten* mit der Funktion `predict` ausgeben lassen.

```
predict(glm1, newdata = data.frame(Pclass = 1), type = "response")
#>     1
#> 0.67
predict(glm1, newdata = data.frame(Pclass = 2), type = "response")
#>     1
#> 0.449
predict(glm1, newdata = data.frame(Pclass = 3), type = "response")
#>     1
#> 0.247
```

Alternativ kann man die tatsächlichen (beobachteten) Häufigkeiten auch noch “per Hand” bestimmen:

```
titanic_train %>%
  filter(Pclass %in% c(1,3)) %>%
  dplyr::select(Survived, Pclass) %>%
  group_by(Pclass, Survived) %>%
  summarise(n = n()) %>%
  mutate(Anteil = n / sum(n))
#> # A tibble: 4 x 4
#> # Groups:   Pclass [2]
#>   Pclass Survived     n Anteil
#>   <int>    <int> <int>  <dbl>
#> 1     1        1     64  0.344
#> 2     1        0    122  0.656
#> 3     3        1     270  0.761
#> 4     3        0     85  0.239
```

Übersetzen wir dies Syntax auf Deutsch:



Nehme den Datensatz “titanic_train” UND DANN
 Filtere nur die 1. und die 3. Klasse heraus UND DANN
 wähle nur die Spalten “Survived” und “Pclass” UND DANN
 gruppiere nach “Pclass” und “Survived” UND DANN
 zähle die Häufigkeiten für jede dieser Gruppen aus UND DANN
 berechne den Anteil an Überlebenden bzw. Nicht-Überlebenden
 für jede der beiden Passagierklassen. FERTIG.

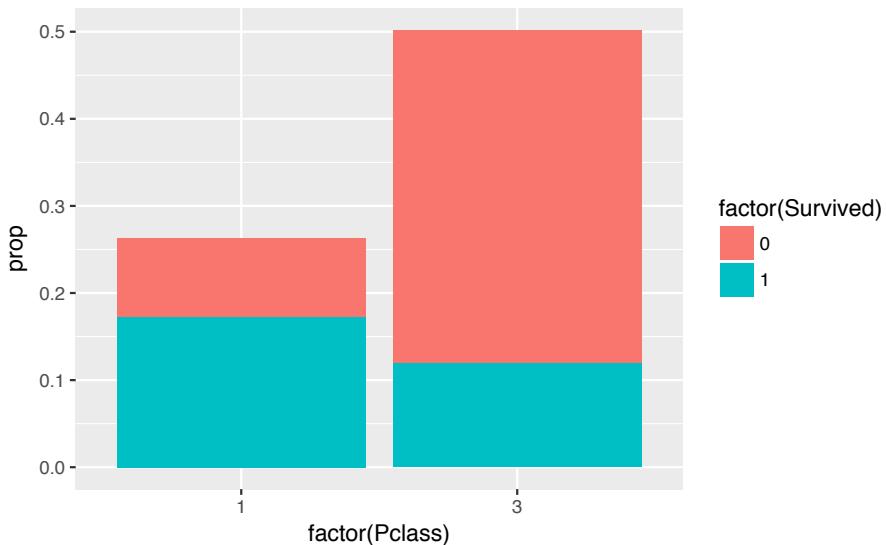


Abbildung 18.3: Absolute Überlebenshäufigkeiten

18.1.9 Effektstärken visualieren

Zum Abschluss schauen wir uns die Stärke des Zusammenhangs noch einmal graphisch an. Wir berechnen dafür die relativen Häufigkeiten pro Gruppe (im Datensatz ohne 2. Klasse, der Einfachheit halber).

```
c5 <- dplyr::count(t2, Pclass, Survived)
c5$prop <- c5$n / 707
c5
#> # A tibble: 4 x 4
#>   Pclass Survived     n    prop
#>   <int>     <int> <int>  <dbl>
#> 1     1         0     64  0.0905
#> 2     1         1    122  0.1726
#> 3     3         0    270  0.3819
#> 4     3         1     85  0.1202
```

Genauer gesagt haben die Häufigkeiten pro Gruppe in Bezug auf die Gesamtzahl aller Passagiere berechnet; die vier Anteile addieren sich also zu 1 auf. Das visualisieren wir wieder, s. Abbildung 18.3.

```
qplot(x = factor(Pclass),
      y = prop,
      fill = factor(Survived),
      data = c5,
      geom = "col")
```

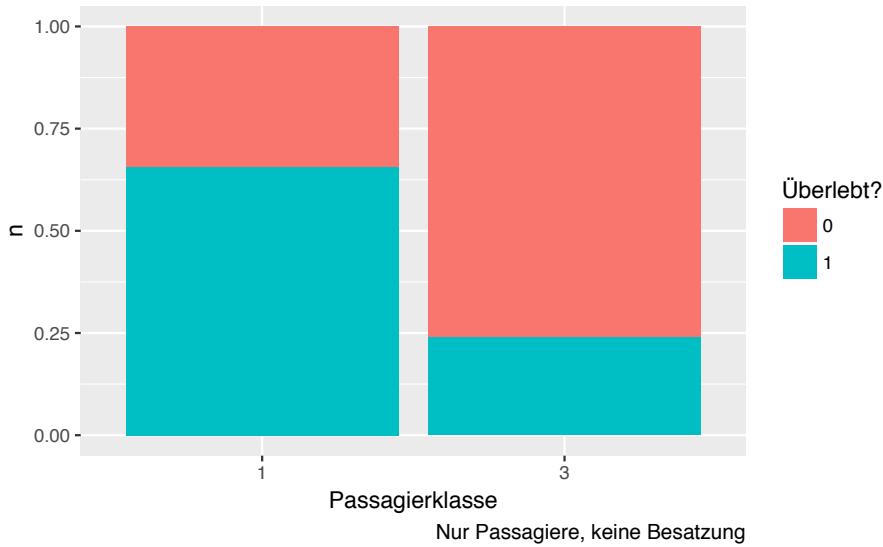


Abbildung 18.4: Relative Überlebenshäufigkeiten

Das `geom = "col"` heißt, dass als “geometrisches Objekt” dieses Mal keine Punkte, sondern Säulen (columns) verwendet werden sollen.

Ganz nett, aber die Häufigkeitsunterschiede von `Survived` zwischen den beiden Werten von `Pclass` stechen noch nicht so ins Auge. Wir sollten es anders darstellen. Hier kommt der Punkt, wo wir von `qplot` auf seinen großen Bruder, `ggplot` wechseln sollten. `qplot` ist in Wirklichkeit nur eine vereinfachte Form von `ggplot`; die Einfachheit wird mit geringeren Möglichkeiten bezahlt. Satteln wir zum Schluss dieser Fallstudie also um, s. Abbildung 18.4.

```
ggplot(data = c5) +
  aes(x = factor(Pclass), y = n, fill = factor(Survived)) +
  geom_col(position = "fill") +
  labs(x = "Passagierklasse",
       fill = "Überlebt?",
       caption = "Nur Passagiere, keine Besatzung")
```

Jeden sehen wir die Häufigkeiten des Überlebens bedingt auf die Passagierklasse besser. Wir sehen auf den ersten Blick, dass sich die Überlebensraten deutlich unterscheiden: Im linken Balken überleben die meisten; im rechten Balken ertrinken die meisten. Mit `labs` haben wir noch die X-Achse (`x`), die Bezeichnung der Füllfarbe (`fill`) sowie die Legende des Diagramms beschrieben. Diese letzte Analyse zeigt schön die Kraft von (Daten-)Visualisierungen auf. Der zu untersuchende Effekt tritt hier am stärken zu Tage; außerdem ist die Analyse relativ einfach.

Eine alternative Darstellung zeigt Abbildung 18.5. Hier werden die vier “Fliesen” gleich groß dargestellt; die Fallzahl wird durch die Füllfarbe besorgt.

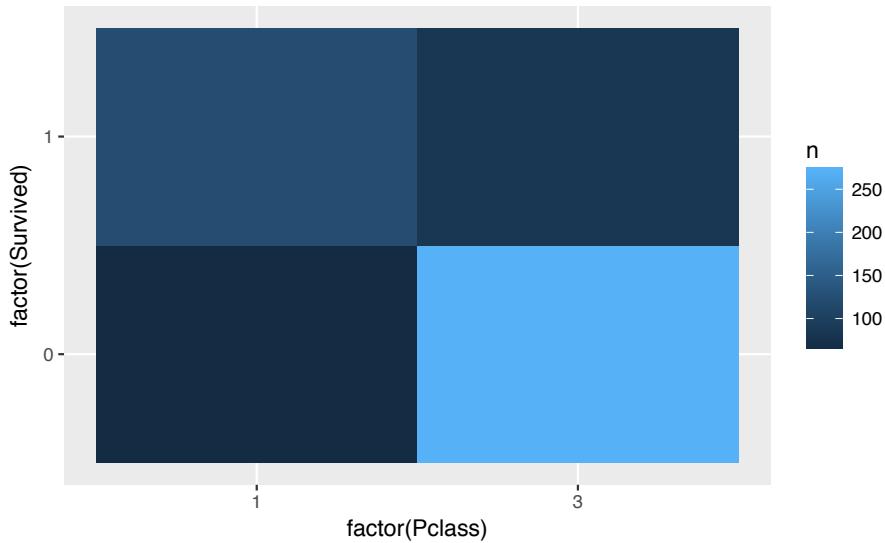


Abbildung 18.5: Überlebenshäufigkeiten anhand eines Fliesenbildes dargestellt

```
c5 %>%
  ggplot +
  aes(x = factor(Pclass), y = factor(Survived), fill = n) +
  geom_tile()
```

18.1.10 Fazit

In der Datenanalyse (mit R) kommt man mit wenigen Befehlen schon sehr weit; `dplyr` und `ggplot2` zählen (zu Recht) zu den am häufigsten verwendeten Paketen. Beide sind flexibel, konsistent und spielen gerne miteinander. Die besten Einblicke haben wir aus deskriptiver bzw. explorativer Analyse (Diagramme) gewonnen. Signifikanztests oder komplizierte Modelle waren nicht zentral. In vielen Studien/Projekten der Datenanalyse gilt ähnliches: Daten umformen und verstehen bzw. „veranschaulichen“ sind zentrale Punkte, die häufig viel Zeit und Wissen fordern. Bei der Analyse von nominalskalierten sind Häufigkeitsauswertungen ideal.

18.2 Außereheliche Affären

Für diese Fallstudie benötigen wir folgende Pakete:

```
library(AER) # Datensatz 'Affairs'
library(psych) # Befehl 'describe'
```

```
library(tidyverse) # Datenjudo
library(broom) # Befehl 'tidy'
```

Wovon ist die Häufigkeit von Affären (Seitensprünge) in Ehen abhängig? Diese Frage soll anhand des Datensatzes `Affairs` untersucht werden. Laden wir als erstes den Datensatz in R.

```
data(Affairs, package = "AER")
```

Verschaffen Sie sich zum Einstieg einen Überblick über die Daten. ... OK, scheint zu passen. Was jetzt?

18.2.1 Zentrale Statistiken

Geben Sie zentrale deskriptive Statistiken an für Affärenhäufigkeit und Ehezufriedenheit!

```
# nicht robust:
mean(Affairs$affairs, na.rm = T)
#> [1] 1.46
sd(Affairs$affairs, na.rm = T)
#> [1] 3.3
# robust:
median(Affairs$Affairs, na.rm = T)
#> NULL
IQR(Affairs$Affairs, na.rm = T)
#> [1] NA
```

Es scheint, die meisten Leute haben keine Affären:

```
count(Affairs, affairs)
#> # A tibble: 6 x 2
#>   affairs     n
#>   <dbl> <int>
#> 1      0    451
#> 2      1     34
#> 3      2     17
#> 4      3     19
#> 5      7     42
#> 6     12     38
```

Man kann sich viele Statistiken mit dem Befehl `describe` aus `psych` ausgeben lassen, das ist etwas praktischer:

```
describe(Affairs$affairs)
#>   vars   n mean   sd median trimmed mad min max range skew kurtosis   se
#> X1     1 601 1.46 3.3      0    0.55   0    0   12    12 2.34      4.19 0.13
describe(Affairs$rating)
#>   vars   n mean   sd median trimmed mad min max range skew kurtosis   se
#> X1     1 601 3.93 1.1      4    4.07 1.48   1    5      4 -0.83    -0.22 0.04
```

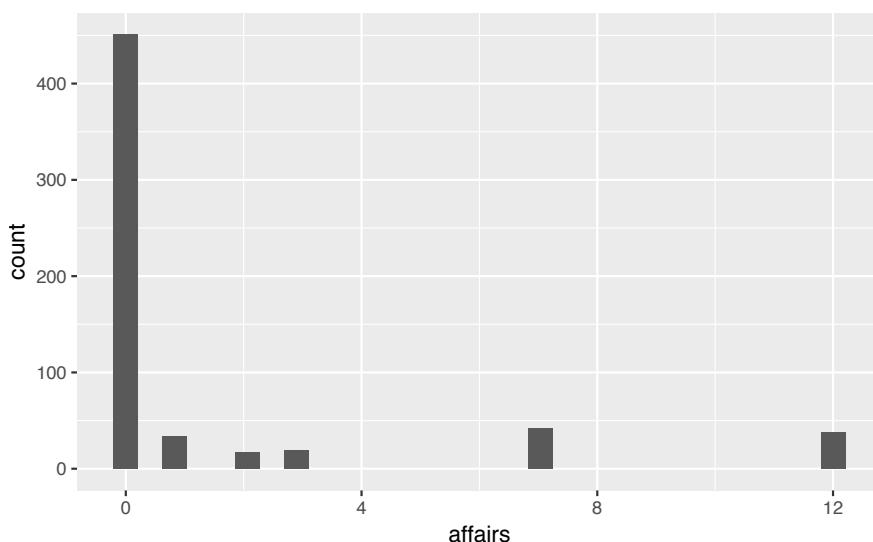
Dazu muss das Paket `psych` natürlich vorher installiert sein. Beachten Sie, dass man ein Paket nur *einmal* installieren muss, aber jedes Mal, wenn Sie R starten, auch starten muss (mit `library`; vgl. Kapitel ??).

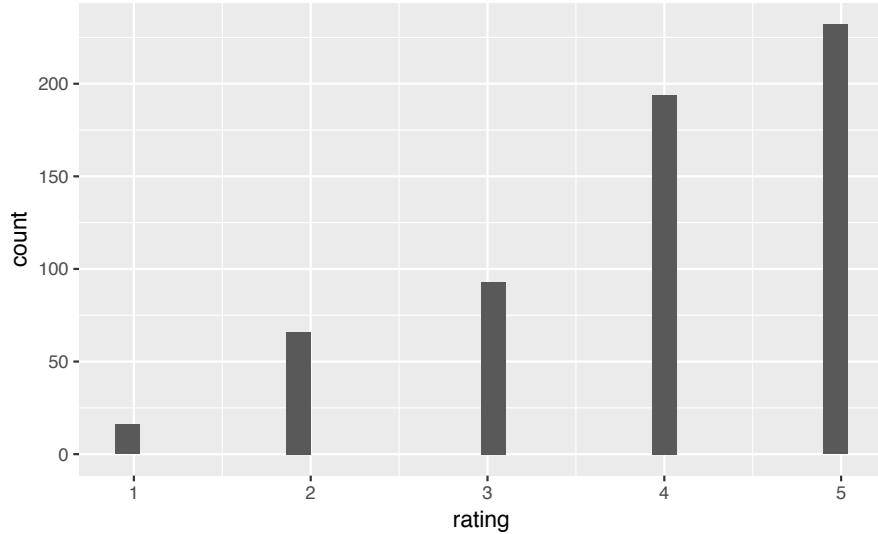
18.2.2 Visualisieren

Visualisieren Sie zentrale Variablen!

Sicherlich sind Diagramme auch hilfreich. Dies geht wiederum mit dem R-Commander oder z.B. mit folgenden Befehlen:

```
qplot(x = affairs, data = Affairs)
qplot(x = rating, data = Affairs)
```





Die meisten Menschen (dieser Stichprobe) scheinen mit Ihrer Beziehung sehr zufrieden zu sein.

18.2.3 Wer ist zufriedener mit der Partnerschaft: Personen mit Kindern oder ohne?

Nehmen wir dazu mal ein paar dplyr-Befehle:

```
library(dplyr)
Affairs %>%
  group_by(children) %>%
  summarise(rating_children =
    mean(rating, na.rm = T))
#> # A tibble: 2 x 2
#>   children rating_children
#>   <fctr>        <dbl>
#> 1 no            4.27
#> 2 yes           3.80
```

Ah! Kinder sind also ein Risikofaktor für eine Partnerschaft! Gut, dass wir das geklärt haben.

18.2.4 Vertiefung: Wie viele fehlende Werte gibt es?

Was machen wir am besten damit?

Diesen Befehl könnten wir für jede Spalte ausführen:

```
sum(is.na(Affairs$affairs))
#> [1] 0
```

Oder lieber alle auf einmal:

```
Affairs %>%
  summarise_all(funs(sum(is.na(.))))
#> affairs gender age yearsmarried children religiousness education
#> 1      0      0    0        0      0      0      0
#> occupation rating
#> 1      0      0
```

Übrigens gibt es ein gutes Cheat Sheet⁴ für dplyr.

Ah, gut, keine fehlenden Werte. Das macht uns das Leben leichter.

18.2.5 Wer ist glücklicher: Männer oder Frauen?

```
Affairs %>%
  group_by(gender) %>%
  summarise(rating_gender = mean(rating))
#> # A tibble: 2 x 2
#>   gender rating_gender
#>   <fctr>     <dbl>
#> 1 female      3.94
#> 2 male        3.92
```

Praktisch kein Unterschied. Heißt das auch, es gibt keinen Unterschied in der Häufigkeit der Affären?

```
Affairs %>%
  group_by(gender) %>%
  summarise(affairs_gender = mean(affairs))
#> # A tibble: 2 x 2
#>   gender affairs_gender
#>   <fctr>     <dbl>
#> 1 female      1.42
#> 2 male        1.50
```

Scheint auch kein Unterschied zu sein...

⁴<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Und zum Abschluss noch mal etwas genauer: Teilen wir mal nach Geschlecht und nach Kinderstatus auf, also in 4 Gruppen. Theoretisch dürfte es hier auch keine Unterschiede/Zusammenhänge geben. Zum mindesten fällt mir kein sinnvoller Grund ein; zumal die vorherige eindimensionale Analyse keine Unterschiede zu Tage gefördert hat.

```
Affairs %>%
  group_by(gender, children) %>%
  summarise(affairs_mean = mean(affairs),
            rating_mean = mean(rating))
#> # A tibble: 4 x 4
#> # Groups:   gender [?]
#>   gender children affairs_mean rating_mean
#>   <fctr>    <fctr>      <dbl>        <dbl>
#> 1 female     no         0.838       4.40
#> 2 female     yes        1.685       3.73
#> 3 male       no         1.014       4.10
#> 4 male       yes        1.659       3.86

Affairs %>%
  group_by(children, gender) %>%
  summarise(affairs_mean = mean(affairs),
            rating_mean = mean(rating))
#> # A tibble: 4 x 4
#> # Groups:   children [?]
#>   children gender affairs_mean rating_mean
#>   <fctr>    <fctr>      <dbl>        <dbl>
#> 1 no        female    0.838       4.40
#> 2 no        male     1.014       4.10
#> 3 yes       female   1.685       3.73
#> 4 yes       male    1.659       3.86
```

18.2.6 Effektstärken

Berichten Sie eine relevante Effektstärke!

Hm, auch keine gewaltigen Unterschiede. Höchstens für die Zufriedenheit mit der Partnerschaft bei kinderlosen Personen scheinen sich Männer und Frauen etwas zu unterscheiden. Hier stellt sich die Frage nach der Größe des Effekts, z.B. anhand Cohen's d. Dafür müssen wir noch die SD pro Gruppe wissen:

```
Affairs %>%
  group_by(children, gender) %>%
```

```
summarise(rating_mean = mean(rating),
           rating_sd = sd(rating))
#> # A tibble: 4 x 4
#> # Groups:   children [?]
#>   children gender rating_mean rating_sd
#>   <fctr> <fctr>     <dbl>      <dbl>
#> 1 no       female      4.40      0.914
#> 2 no       male        4.10      1.064
#> 3 yes      female     3.73      1.183
#> 4 yes      male        3.86      1.046
```

```
d <- (4.4 - 4.1)/(1)
```

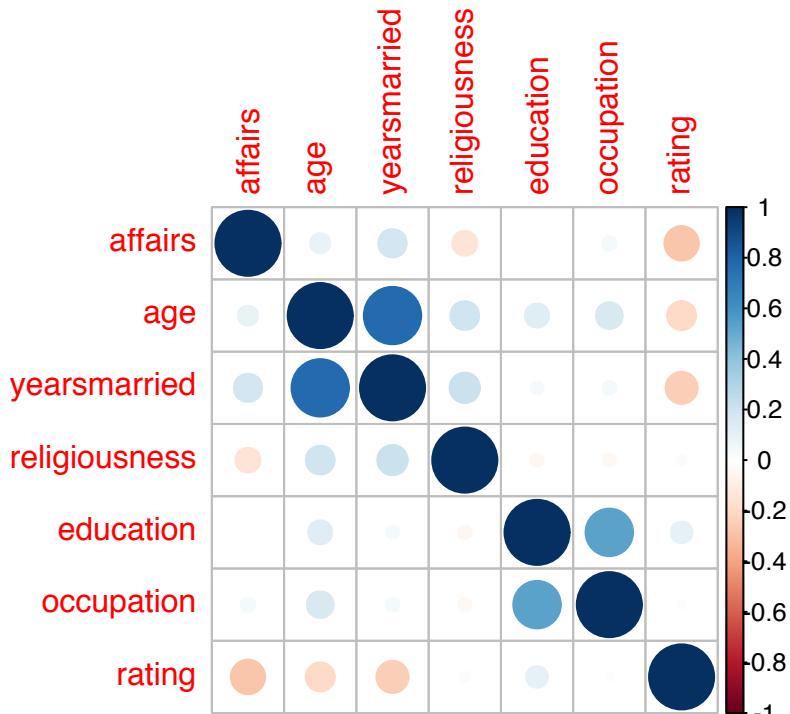
Die Effektstärke beträgt etwa 0.3.

18.2.7 Korrelationen

Berechnen und visualisieren Sie zentrale Korrelationen!

```
Affairs %>%
  select_if(is.numeric) %>%
  cor -> cor_tab

corrplot(cor_tab)
```



18.2.8 Ehejahre und Affären

Wie groß ist der Einfluss (das Einflussgewicht) der Ehejahre bzw. Ehezufriedenheit auf die Anzahl der Affären?

Dazu sagen wir R: "Hey R, rechne mal ein lineares Modell", also eine normale (lineare) Regression. Dazu können wir entweder das entsprechende Menü im R-Commander auswählen, oder folgende R-Befehle ausführen:

```
lm1 <- lm(affairs ~ yearsmarried, data = Affairs)
tidy(lm1) # Ergebnisse der Regression zeigen
#>   term estimate std.error statistic p.value
#> 1 (Intercept)  0.551    0.2351     2.34 0.019378
#> 2 yearsmarried  0.111    0.0238     4.65 0.000004
glance(lm1)
#>   r.squared adj.r.squared sigma statistic p.value df logLik AIC  BIC
#> 1  0.0349      0.0333  3.24      21.7  4e-06  2 -1559 3124 3137
#>   deviance df.residual
#> 1    6301       599
lm2 <- lm(affairs ~ rating, data = Affairs)
tidy(lm2)
#>   term estimate std.error statistic p.value
#> 1 (Intercept)  4.742    0.479     9.90 1.68e-21
#> 2     rating -0.836    0.117    -7.12 3.00e-12
```

```
glance(lm2)
#>   r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC
#> 1    0.0781        0.0766  3.17      50.8   3e-12  2 -1545 3096 3110
#>   deviance df.residual
#> 1      6019         599
```

Also: `yearsmarried` und `rating` sind beide statistisch signifikante Prädiktoren für die Häufigkeit von Affären. Das adjustierte R^2 ist allerdings in beiden Fällen nicht so groß.

18.2.9 Ehezufriedenheit als Prädiktor

Um wie viel erhöht sich die erklärte Varianz (R-Quadrat) von Affärenhäufigkeit wenn man den Prädiktor Ehezufriedenheit zum Prädiktor Ehejahre hinzufügt? (Wie) verändern sich die Einflussgewichte (b)?⁵

```
lm3 <- lm(affairs ~ rating + yearsmarried, data = Affairs)
lm4 <- lm(affairs ~ yearsmarried + rating, data = Affairs)
summary(lm3)
summary(lm4)
```

Ok. Macht eigentlich die Reihenfolge der Prädiktoren in der Regression einen Unterschied? Der Vergleich von Modell 3 vs. Modell 4 beantwortet diese Frage.

Wir sehen, dass beim 1. Regressionsmodell das R^2 0.03 war; beim 2. Modell 0.08 und beim 3. Modell liegt R^2 bei 0.09. Die Differenz zwischen Modell 1 und 3 liegt bei (gerundet) 0.06; wenig.

18.2.10 Weitere Prädiktoren der Affärenhäufigkeit

Welche Prädiktoren würden Sie noch in die Regressionsanalyse aufnehmen?

Hm, diese Frage klingt nicht so, als ob der Dozent die Antwort selber wüsste... Naja, welche Variablen gibt es denn alles:

```
#> [1] "affairs"       "gender"        "age"          "yearsmarried"
#> [5] "children"     "religiousness" "education"    "occupation"
#> [9] "rating"
```

Z.B. wäre doch interessant, ob Ehen mit Kinder mehr oder weniger Seitensprünge aufweisen. Und ob die “Kinderfrage” die anderen Zusammenhänge/Einflussgewichte in der Regression verändert. Probieren wir es auch. Wir können wiederum im R-Commander ein Regressionsmodell anfordern oder es mit der Syntax probieren:

⁵Output im Folgenden nicht abgedruckt.

```
lm5 <- lm(affairs ~ rating + yearsmarried + children, data = Affairs)
summary(lm5)
r2_lm5 <- summary(lm5)$r.squared
```

Das Regressionsgewicht von `childrenyes` ist negativ. Das bedeutet, dass Ehen mit Kindern weniger Affären verbuchen (aber geringe Zufriedenheit, wie wir oben gesehen haben! Hrks!). Allerdings ist der p-Wert nicht signifikant, was wir als Zeichen der Unbedeutsamkeit dieses Prädiktors verstehen können. R^2 lungert immer noch bei mickrigen 0.094 herum. Wir haben bisher kaum verstanden, wie es zu Affären kommt. Oder unsere Daten bergen diese Informationen einfach nicht.

Wir könnten auch einfach mal Prädiktoren, die wir haben, ins Feld schicken. Mal sehen, was dann passiert:

```
lm6 <- lm(affairs ~ ., data = Affairs)
summary(lm6)
```

Der “.” im Befehl `affairs ~ .` oben soll sagen: nimm “alle Variablen, die noch in der Datenmatrix übrig sind”.

Insgesamt bleibt die erklärte Varianz in sehr bescheidenem Rahmen: 0.13. Das zeigt uns, dass es immer noch nur schlecht verstanden ist – im Rahmen dieser Analyse – welche Faktoren die Affärenhäufigkeit erklärt.

18.2.11 Unterschied zwischen den Geschlechtern

Unterscheiden sich die Geschlechter statistisch signifikant? Wie groß ist der Unterschied? Sollte hier lieber das d-Maß oder Rohwerte als Effektmaß angegeben werden?

Hier bietet sich ein t-Test für unabhängige Gruppen an. Die Frage lässt auf eine ungerichtete Hypothese schließen (α sei .05). Mit dem entsprechenden Menüpunkt im R-Commander oder mit folgender Syntax lässt sich diese Analyse angehen:

```
t.test(affairs ~ gender, data = Affairs) -> t1

t1 %>% tidy
#>   estimate estimate1 estimate2 statistic p.value parameter conf.low
#> 1  -0.0775      1.42       1.5     -0.287   0.774      594    -0.607
#>   conf.high               method alternative
#> 1      0.452 Welch Two Sample t-test   two.sided
```

Der p-Wert ist größer als α . Daher wird die H_0 beibehalten. Auf Basis der Stichprobendaten entscheiden wir uns für die H_0 . Entsprechend umschließt das 95%-KI die Null.

Da die Differenz nicht signifikant ist, kann argumentiert werden, dass wir d auf 0 schätzen müssen. Man kann sich den d -Wert auch z.B. von {MBESS} schätzen lassen.

Dafür brauchen wir die Anzahl an Männer und Frauen: 315, 286.

```
library(MBESS)
ci.smd(ncp = t1$statistic,
       n.1 = 315,
       n.2 = 286)
#> $Lower.Conf.Limit.smd
#> [1] -0.184
#>
#> $smd
#>      t
#> -0.0235
#>
#> $Upper.Conf.Limit.smd
#> [1] 0.137
```

Das Konfidenzintervall ist zwar relativ klein (die Schätzung also aufgrund der recht großen Stichprobe relativ präzise), aber der Schätzwert für d `smd` liegt sehr nahe bei Null. Das stärkt unsere Entscheidung, von einer Gleichheit der Populationen (Männer vs. Frauen) auszugehen.

18.2.12 Kinderlose Ehe vs. Ehen mit Kindern

Rechnen Sie die Regressionsanalyse getrennt für kinderlose Ehe und Ehen mit Kindern!

Hier geht es im ersten Schritt darum, die entsprechenden Teil-Mengen der Datenmatrix zu erstellen. Das kann man natürlich mit Excel o.ä. tun. Alternativ könnte man es in R z.B. so machen:

```
Affair4 <- filter(Affairs, children == "yes")
head(Affair4)
```

18.2.13 Halodries

Rechnen Sie die Regression nur für “Halodries”; d.h. für Menschen mit Seitensprüngen. Dafür müssen Sie alle Menschen ohne Affären aus den Datensatz entfernen.

Also, rechnen wir nochmal die Standardregression (`lm1`). Probieren wir den Befehl `filter` dazu nochmal aus:

```
Affair5 <- filter(Affairs, affairs != 0)
lm9 <- lm(affairs ~ rating, data = Affair5)
summary(lm9)
```

18.2.14 logistische Regression

Berechnen Sie für eine logistische Regression mit “Affäre ja vs. nein” als Kriterium, wie stark der Einfluss von Geschlecht, Kinderstatus, Ehezufriedenheit und Ehedauer ist!

```
Affairs %>%
  mutate(affairs_dichotom = affairs == 0) %>%
  glm(affairs_dichotom ~ gender + children + rating + yearsmarried,
    data = .,
    family = "binomial") -> lm10

tidy(lm10)
```

Wenn `if_else` unbekannt ist, lohnt sich ein Blick in die Hilfe mit `?if_else` (`dplyr` muss vorher geladen sein).

Aha, signifikant ist die Ehezufriedenheit: Je größer `rating` desto geringer die Wahrscheinlichkeit für `affairs_dichotom`. Macht Sinn!

Übrigens, die Funktionen `lm`, `glm` und `summary` spucken leider keine brave Tabelle in Normalform aus, was aber schön wäre. Aber man leicht eine Tabelle (data.frame) bekommen mit dem Befehl `tidy` aus `broom`:

```
tidy(lm10)
#>   term estimate std.error statistic p.value
#> 1 (Intercept) -0.0537   0.4299   -0.125 9.01e-01
#> 2 gendermale   -0.2416   0.1966   -1.229 2.19e-01
#> 3 childrenyes  -0.3935   0.2831   -1.390 1.64e-01
#> 4 rating       0.4654   0.0874   5.327 9.97e-08
#> 5 yearsmarried -0.0221   0.0212   -1.040 2.99e-01
```

18.2.15 Zum Abschluss

Visualisieren wir mal was! Ok, wie wäre es mit einem Jitter-Diagramm (vgl. Abbildungen 18.6 und 18.7).

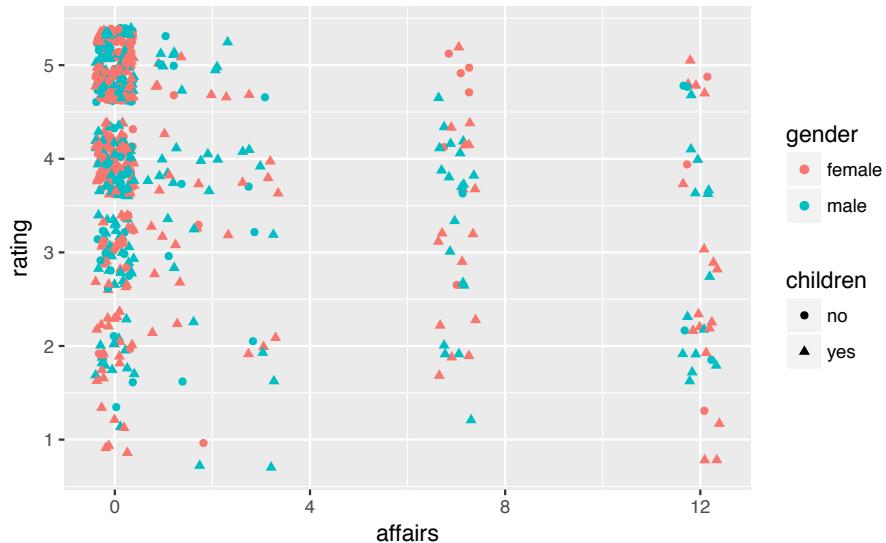


Abbildung 18.6: Affären, mit Jitter

Affairs %>%

```
select(affairs, gender, children, rating) %>%
  ggplot(aes(x = affairs, y = rating)) +
  geom_jitter(aes(color = gender, shape = children))
```

Affairs %>%

```
mutate(rating_dichotom = ntile(rating, 2)) %>%
  ggplot(aes(x = yearsmarried, y = affairs)) +
  geom_jitter(aes(color = gender)) +
  geom_smooth()
```

Puh. Geschafft!

18.3 Befehlsübersicht

Tabelle 18.1 fasst die R-Funktionen dieses Kapitels zusammen.

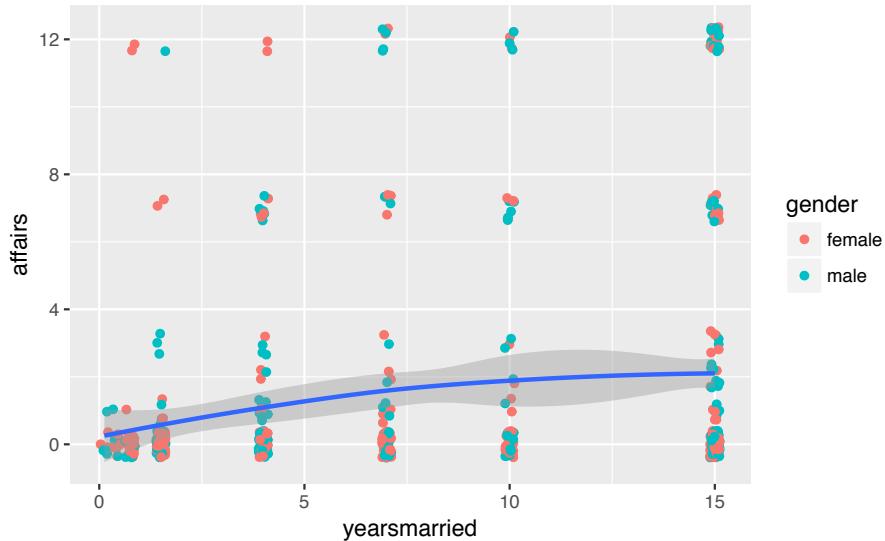


Abbildung 18.7: Affären, mit Smooth

Tabelle 18.1: Befehle des Kapitels 'Fallstudien titanic und affairs'

Paket..Funktion	Beschreibung
data	Lädt Daten aus einem Datensatz
chisq.test	Rechnet einen Chi-Quadrat-Test
compute.es::chies	Liefert Effektstärkemaße für einen Chi-Quadrat-Test
predict	Macht eine Vorhersage
psych::describe	Liefert eine Reihe zentraler Statistiken
is.na	Zeigt an, ob ein Vektor fehlende Werte beinhaltet
dplyr::summarise_each	Führt summarise für jede Spalte aus
t.test	Rechnet einen t-Test
MBESS::ci.smd	Berechnet Cohens d
dplyr::ntile	Teilt einen Vektor in n Teile mit jeweils gleich viel Werten
broom::tidy	Wandelt ein Objekt vom Typ 'lm' in einen Dataframe um.

Kapitel 19

Baumbasierte Verfahren



Lernziele:

- Die wichtigsten Varianten baumbasierter Verfahren kennen und unterscheiden können
- Die grundlegende Funktionsweise (den Algorithmus) von Entscheidungsbäumen erklären können
- Die Erweiterung von Entscheidungsbäumen zu Bagging und Random Forests kennen
- Um die Stärken und Schwächen von baumbasierten Verfahren Bescheid wissen

In diesem Kapitel werden folgende Pakete benötigt:

```
library(tidyverse) # Datenjudo
library(rpart) # Entscheidungsbäume
library(partykit) # Entscheidungsbäume visualisieren
library(caret) # Standard-Syntax für prädiktive Modellierung
library(gridExtra) # mehrere Plots in einem kombinieren
library(mosaic) # Komfort im Datenjudo
library(pradadata)
```

In diesem Kapitel betrachten wir Varianten von *baumbasierten Verfahren*. Dazu zählen *Entscheidungsbäume*, *Bagging* und *Random Forests* (“Zufallswälder”) - neben weiteren Familienmitglieder, die wir außen vor lassen (s. (James, Witten, Hastie, und Tibshirani 2013b, Hastie, Tibshirani, und Friedman (2013)) für eine vertiefte Darstellung).

Baumbasierte Verfahren sind sowohl für Klassifikation (nominale Kriterien) als auch für Regression (quantitative Kriterien) einsetzbar. Technisch bedeutet das zumeist, dass Kriterien vom Typ `factor` oder `character` eine Klassifikation auslösen, wohingegen numerische Kriterien eine Regression nachsichziehen. Betrachten wir zuerst eine Klassifikation. Als Prädiktoren können sowohl numerische als auch nominale Variablen verwendet werden.

19.1 Entscheidungsbäume

Entscheidungsbäume stellen, allgemein gesprochen, Entscheidungen und deren Konsequenzen in einer baumähnlichen Struktur dar. Dabei stellen die Prüfungen die “Astgabeln” oder “Knoten” (nodes) dar, die “Äste” die Entscheidungen der Prüfungen und die Konsequenzen sind die “Blätter” des Baumes; am besten sieht man dies in einem Beispiel (s. Abb. 19.1).

19.1.1 Einführendes Beispiel

Betrachten wir ein einfaches Beispiel; zuerst ist etwas Datenjudo nötig. Wir trennen einen Datensatz in einen Trainings- und eine Test-Stichprobe auf. Aus dem Test-Datensatz löschen wir das vorherzusagende Kriterium (sonst wäre die Aufgabe etwas zu leicht).

Als Datenbeispiel dient uns der Affären-Datensatz; unsere Analysefrage soll sein, vorherzusagen anhand der verfügbaren Variablen, ob jemand Affären begangen hat oder nicht. Dazu erstellen wir eine binäre Variable (nennen wir sie `is_halodrie`). Dabei dürfen wir nicht vergessen, `affairs` zu löschen, nachdem wir `is_halodrie` erstellt haben, weil sonst bliebe ja die zu vorhersagende Information im Datensatz.

```
data(Affairs, package = "AER")

set.seed(42)
index_train <- sample(x = 1:601,
                      size = trunc(.8 * 601))

Affairs <- Affairs %>%
  mutate(is_halodrie = ifelse(affairs > 0, "ja", "nein"),
        is_halodrie = as.factor(is_halodrie))

train_df <- Affairs %>%
  filter(row_number() %in% index_train) %>%
  select(-affairs)

test_df <- Affairs %>%
  filter(!row_number() %in% index_train)

rm(index_train) # Das Objekt wieder löschen, da nicht mehr benötigt
```

Das Kriterium `is_halodrie` haben wir als Faktor definiert, weil die Funktionen, die wir nutzen werden, anhand des Typs der Variable entscheiden, ob eine Klassifikation oder Regression durchgeführt wird. Bei Faktoren kommt eine Klassifikation zum Einsatz und bei numerischen Werten eine Regression. Nach dieser Vorarbeit lassen wir uns den Entscheidungsbaum

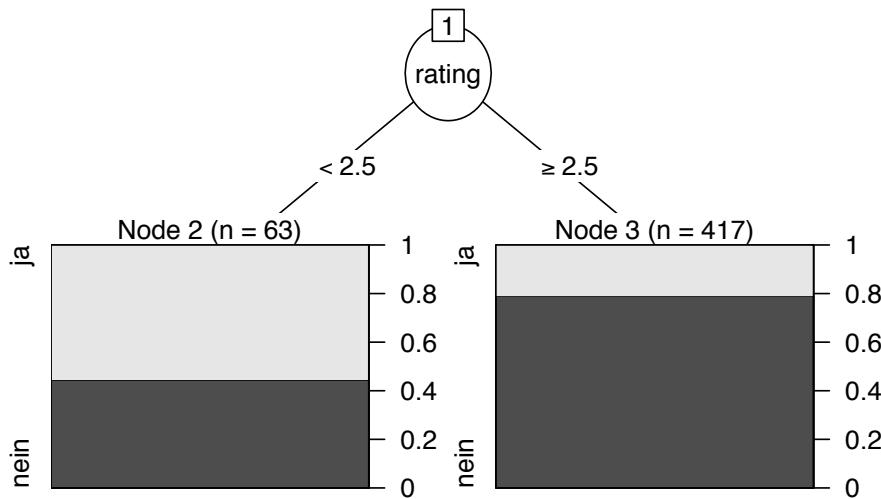


Abbildung 19.1: Ein einfacher Entscheidungsbaum, der die Stichprobe in zwei Gruppen aufteilt

berechnen (mit `rpart`) und plotten (mit `plot` aus `partykit`). Betrachten wir aber erst einmal nur zwei Prädiktoren (Ehezufriedenheit und Geschlecht), der Einfachheit halber.

```
baum1 <- rpart(is_halodrie ~ rating+gender, data = train_df)
plot(as.party(baum1))
```

Der Baum (s. Abb 19.1) ist recht einfach: Er besteh nur aus “Wurzelknoten” (Knoten 1) und zwei Blättern (Endknoten; Knoten 2 und 3). Die einzige Prüfung, die durchgeführt wird, ist, ob die Ehezufriedenheit kleiner als 2.5 ist oder nicht. In der Gruppe der Unzufriedenen (`rating<2.5`) liegt der Halodrie-Anteil bei ca. 60%; in der Gruppe der Zufriedenen bei ca. 20%. Hätten wir keine Unterteilung durchgeführt, hätten wir einen Halodrie-Anteil von 25% festgestellt:

```
train_df %>%
  count(is_halodrie) %>%
  ungroup %>%
  mutate(prop = n / sum(n))
#> # A tibble: 2 x 3
#>   is_halodrie     n   prop
#>   <fctr> <int> <dbl>
#> 1      ja    122 0.254
#> 2     nein    358 0.746
```

Durch diese “Aufsplittung” (*Partitionierung*) anhand einer Prüfung (`rating<2.5`) in zwei Gruppen haben wir “reinere” also homogenere Gruppen bekommen als bei Betrachtung der Halodrie-Häufigkeit in der Stichprobe *ohne* Partitionierung.

Eine homogenere Stichprobe, d.h. sehr wenig oder sehr viel Halodrie-Anteil lässt uns sichere (genauere) Vorhersagen treffen, erhöht also unser Wissen. Homogenere Stichproben sind also für Vorhersagen erstrebenswert. Entscheidungsbäume wählen bei jeder Astgabelung den Prädiktor und den Trennwert, der die Homogenität insgesamt erhöht.

Interessanterweise wurde Prädiktor `gender` gar nicht in den Baum aufgenommen. Der Grund ist, dass dieser Prädiktor die Homogenität der Gruppen nicht (ausreichend) erhöhen würde. Wir lernen daraus, dass `gender` in diesem Fall nicht viel Information birgt. Alternative Visualisierungen für diesen einfachen Entscheidungsbaum sind in im linken Teil der Abbildung 19.2 dargestellt.

```
# baum1
train_df %>%
  mutate(gruppe_zufrieden = rating >= 2.5) %>%
  ggplot +
  aes(x = rating, y = is_halodrie) +
  geom_count(aes(color = gruppe_zufrieden),
             position = "jitter") +
  theme(legend.position = "bottom") +
  geom_vline(xintercept = 2.5, color = "firebrick", linetype = "dashed") -> p1

# baum2
train_df %>%
  ggplot +
  aes(x = rating, y = age) +
  geom_count(aes(shape = is_halodrie,
                 color = is_halodrie),
             position = "jitter") +
  facet_wrap(~gender) +
  theme(legend.position = "bottom") +
  geom_vline(xintercept = 2.5) + # node1
  geom_segment(x = 0, xend = 2.5, # node2
               y = 44.5, yend = 44.5) +
  geom_segment(x = 0, xend = 2.5, # node3
               y = 29.5, yend = 29.5) -> p2

grid.arrange(p1, p2, nrow = 1,
             bottom = "Links: Ein einfacher Baum mit nur einer Verzweigung; \nrechts: Ein
```

Erweitern wir das Beispiel und lassen alle Prädiktoren des (Training-)Datensatzes zu; wie sich wohl der Baum verändert?

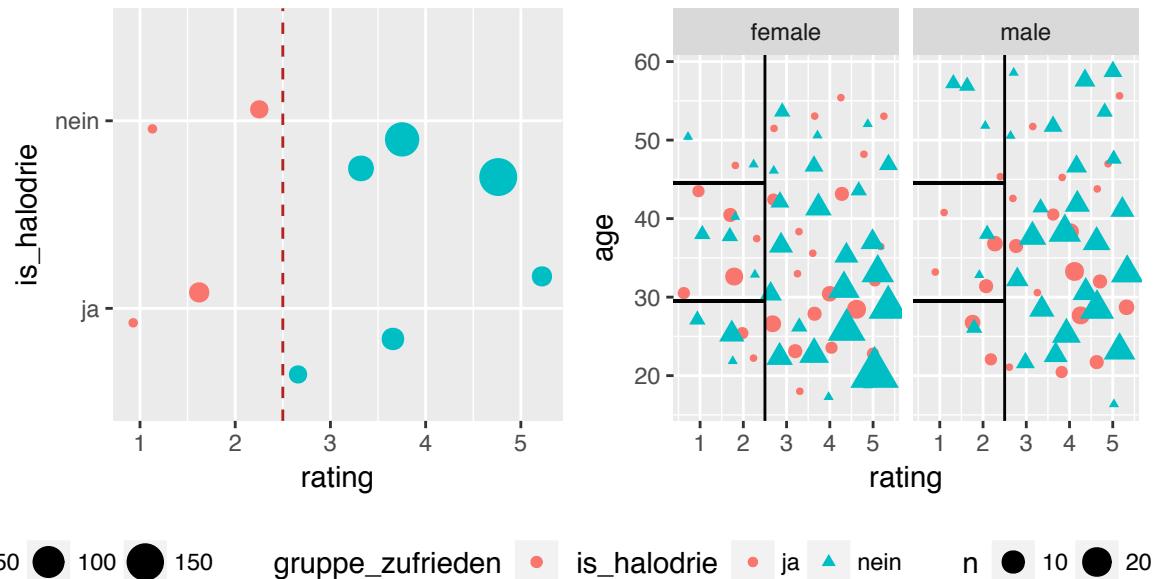


Abbildung 19.2: Alternative Visualisierungen für Entscheidungsbäume

```
baum2 <- rpart(is_halodrie ~ ., data = train_df)
plot(as.party(baum2))
```

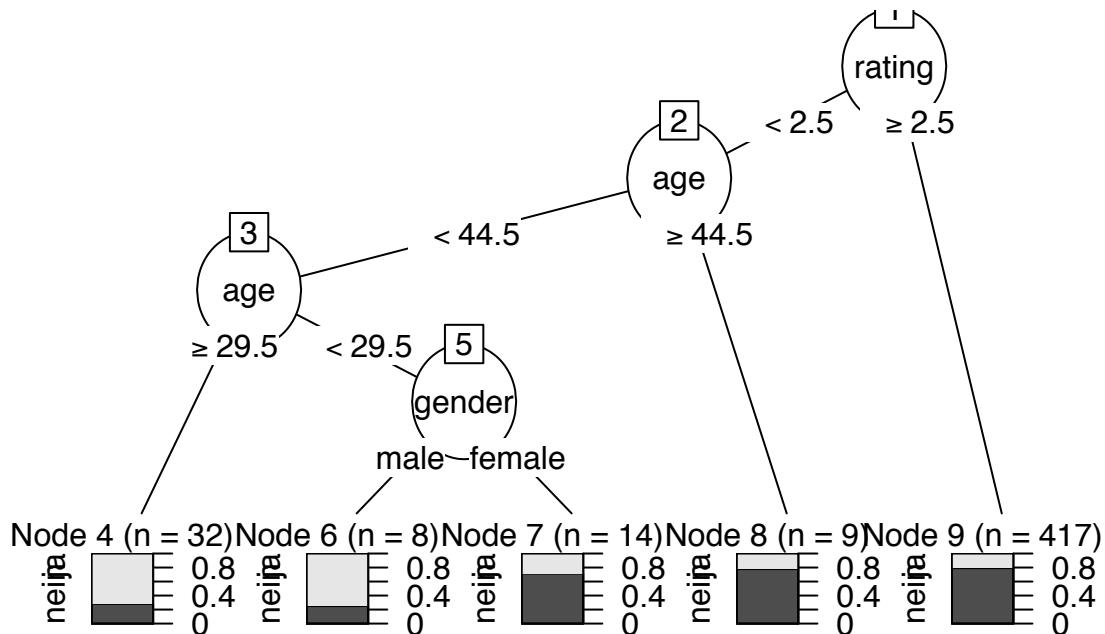


Abbildung 19.3: Ein komplexerer Entscheidungsbau

Betrachten wir unseren Baum in Abbildung 19.3; die “Äste” zeigen nach unten; der Baum steht Kopf. Fantasievolle Mitmenschen sehen statt Ästen Wurzeln, dann stimmt die Ausrichtung des Baumes wieder... Der Baum gibt also folgende Regelwerk (Algorithmus) wieder, zur Entscheidung ob jemand Halodrie ist:

-  – Prüfung: Ist die Ehezufriedenheit größer oder gleich 2.5? (Astgabel 1)
- Wenn die Entscheidung “ja” lautet, ist die Halodrie-Quote (Konsequenz) ca. .20. (Astgabel 2)
- Ansonsten Prüfung: Ist das Alter größer oder gleich 44.5? (Astgabel 3)
- Wenn ja, ist die Halodrie-Quote ca. .20. (Astgabel 4)
- Ansonsten Prüfung: Ist das Alter größer oder gleich 29.5?
- Wenn ja, ist die Halodrie-Quote ca. .75. (Astgabel 9)
- Ansonsten Prüfung: Ist es ein Mann? (Astgabel 6)
- Wenn ja, ist die Halodrie-Quote ca. .75. (Astgabel 8)
- Ansonsten ist die Halodrie-Quote ca. .30. (Astgabel 7)

Es handelt sich also um eine einfaches Entscheidungsregelwerk; man denke an einen Arzt in der Notaufnahme: Ist Puls da? Wenn nein, beginne Wiederbelebung. Ansonsten Prüfung: Ist Atmung erkennbar usw. Eine praktische, gut verständliche und schön visualisierbare Angelegenheit. Die Blätter des Baumes (die Knoten ohne weitere Verzeigungen) geben den Anteil der Halodries wieder. Als Entscheidungskriterium ob jemand, der den Baum bis zu diesem Ast bzw. Blatt (“Endknoten”) durchrutscht, ein Halodrie ist, halten wir an das Mehrheitsprinzip:

-  1. Beobachtung X wird dem Endknoten k zugeordnet.
2. Ordne K der Mehrheit von k zu, d.h. der häufigsten Klasse.

In unserem Beispiel würden wir Personen, die im Blatt 2 “herauskommen” als “Nicht-Halodrie” einordnen (Personen mit hoher Ehezufriedenheit sind offenbar selten Halodries), aber Personen, die dem Blatt 9 zugeordnet werden, bekommen den Halodrie-Stempel.

Anstelle des Graphen können wir uns natürlich auch die numerischen Werte ausgeben lassen:

```
baum2
#> n= 480
#>
#> node), split, n, loss, yval, (yprob)
#>      * denotes terminal node
#>
#> 1) root 480 122 nein (0.254 0.746)
#> 2) rating< 2.5 63 28 ja (0.556 0.444)
#> 4) age< 44.5 54 21 ja (0.611 0.389)
```

```
#>      8) age>=29.5 32   9 ja (0.719 0.281) *
#>      9) age< 29.5 22  10 nein (0.455 0.545)
#>      18) gender=male 8   2 ja (0.750 0.250) *
#>      19) gender=female 14   4 nein (0.286 0.714) *
#>      5) age>=44.5 9   2 nein (0.222 0.778) *
#>      3) rating>=2.5 417  87 nein (0.209 0.791) *
```

So liest sich die Ausgabe: An der “Wurzel” (root) des Baumes (Knoten 1) haben wir 480 Fälle, von denen 122 Halodries sind, was 25% entspricht. Die Mehrheit sind also keine Halodries; unser Tipp bzw. unsere Vorhersage (`yval`) an diesem Punkt ist also “nein” (kein Halodrie). Der Ast mit `rating>=2.5` führt zu einem Blatt mit $n=417$ Observationen; von denen “verlieren” wir 87 Personen (`loss`), wobei aber die Mehrheit (79%) keine Halodries sind; unsere Vorhersage (`yval`) für Individuen, die in diesem Blatt landen ist also “nein, kein Halodrie”. Leider passen die Nummer der Knoten in der Abbildung nicht zur Nummer der Knoten der Textausgabe. Die Richtigkeit der Ergebnisse von `baum1` können wir in gewohnter Manier nachprüfen:

```
count(train_df, is_halodrie) # Halodries insgesamt
#> # A tibble: 2 x 2
#>   is_halodrie     n
#>   <fctr> <int>
#> 1 ja      122
#> 2 nein    358
count(train_df, rating>=2.5) # Prüfung in Knoten 1
#> # A tibble: 2 x 2
#>   `rating >= 2.5`     n
#>   <lgl> <int>
#> 1 FALSE    63
#> 2 TRUE     417

train_df %>% # Prüfung in Knoten 9
  filter(rating>=2.5) %>%
  count(is_halodrie)
#> # A tibble: 2 x 2
#>   is_halodrie     n
#>   <fctr> <int>
#> 1 ja      87
#> 2 nein    330
```

19.1.2 Tuningparameter

Abbildung 19.1 (rechter Teil) zeigt eine alternative Visualisierung für denselben Baum. Auch in diesem Baum fanden wiederum nicht alle Variablen Eingang, da sie die Homogenität nicht (ausreichend) verbesserten. Was “ausreichend” ist wird durch einen sog. *Komplexitätsparameter* gesteuert; in der Voreinstellung liegt der bei 1% Verbesserung zum letzten Split. Ist die Verbesserung der Homogenität geringer als dieser Wert, so wird der Split nicht durchgeführt.

Für Entscheidungsbäume ist der Komplexitätsparameter eine Art Hebel, der die Arbeitsweise der Maschine beeinflusst. Man spricht von *Tuningparametern*. Verschiedene Werte von Tuningparametern dürfen ausprobiert werden, solange dafür nur die Trainings-Stichprobe, nicht die Test-Stichprobe, verwendet wird.

Verändern Sie mal den Wert des Arguments `rpart.control`, z.B. auf 0.001; wie verändert sich der Baum? Mit `printcp(baum2)` kann man sich den Einfluss verschiedener Werte des Komplexitätsparameters ausgeben lassen. Mit `plotcp(baum2)` bekommt man ein Diagramm dazu zurückgeliefert.

19.1.3 Entscheidungsbäume mit caret

```
my_train_control <- trainControl(
  method = "cv",
  number = 10
)

baum3 <- train(is_halodrie ~ .,
  data = train_df,
  method = "rpart",
  trControl = my_train_control,
  cp = 0.002)
```

Soeben haben wir also für Trainings-Sample jeweils einen Baum berechnet; die Vorhersagegüte wurde jedes Mal am Test-Sample berechnet. Dann haben wir uns den Baum mit der besten Vorhersagegüte ausgewählt. Dieser werden wir nehmen, um am *bisher nicht angerührten* Testsample `test_df` die Vorhersagegüte zu berechnen. Interessant ist die Frage, wie unterschiedlich die 10 Bäume sind; werden ihre Vorhersagegüten stark variieren, wird uns das kein großes Vertrauen in das Modell bauen lassen. Die Situation würde einer Waage gleichen, die jedes Mal ein komplett anderes Gewicht anzeigt, wenn Sie sie betreten. Werden die Gewichtswerte sich stets ähneln, fassen wir mehr Vertrauen zur Waage. Betrachten wir also die Variabilität der Vohersagen. Die Objekte, die `train()` zurückliefert, beinhalten das Objekt `resample`; dort sind die Gütekoeffizienten jedes der 10 Durchläufe festgehalten; die Vorhersagen für jede einzelne Beobachtung der 10 Durchgänge finden sich im Modell `pred`; vgl. Abbildung 19.4, linke Seite.

```

baum3$resample %>%
  gather(key = coefficient, value = value, - Resample) -> baum_folds

baum_folds %>%
  ggplot +
  aes(x = coefficient, y = value, color = Resample) +
  geom_point(position = "jitter") -> p1

```

Aber ist die Variation, die wir her sehen, groß oder klein? Schwer zu sagen; klarer wird die Zuverlässigkeit des Modells wenn wir sie in Relation zu einem anderen Modell setzen. Ein gutes Vergleichsmodell, da einfach und bewährt, ist die (logistische) Regression:

```

my_train_control_glm <- trainControl(
  method = "cv",
  number = 10
)

glm1 <- train(is_halodrie ~ .,
               data = train_df,
               method = "glm",
               trControl = my_train_control_glm)

glm1$resample %>%
  gather(key = coefficient, value = value, - Resample) %>%
  mutate(model = "glm") -> glm_folds

```

Bei beiden Dataframes mit den jeweils 10 Faltungen fassen wir zu einen Dataframe zusammen, um die Ergebnisse dann vergleichend zu plotten; vgl. Abbildung 19.4, rechte Seite.

```

baum_folds %>%
  mutate(model = "tree") %>%
  bind_rows(glm_folds) -> folds

folds %>%
  ggplot +
  aes(x= coefficient, y = value, color = model, shape = model) +
  geom_point(position = "jitter") +
  stat_summary(fun.y = "mean", geom = "pointrange", fun.data = "mean_cl_boot", position

```

Mit `stat_summary` haben wir uns noch den Mittelwert der jeweiligen Punkte (Kreuze) angeben lassen, sowie einen Standardfehler, der aus Bootstrap-Werten berechnet wurde (Wickham 2009a). Insgesamt ist der Baum der linearen Modell hier etwas überlegen.

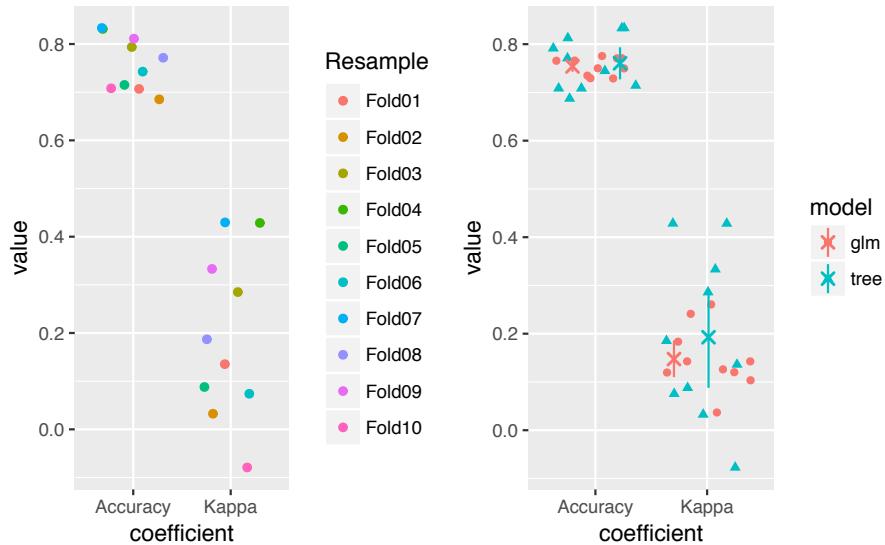


Abbildung 19.4: Vergleich der Kreuzvalidierungsergebnisse für das Affären-Modell

19.1.4 Vorhersagegüte

Wie "gut" ist unser Baum bzw. unser Vorhersagemodell? Genauer gesagt: Wie viele Halodries (positive Fälle) wurden als solche von unserem Modell erkannt und wie viele Nicht-Halodries (Treue Seelen?; negative Fälle) wurden als solche erkannt? Dazu betrachten wir eine Konfusionsmatrix; zuerst für die Trainings-, danach für die Test-Stichprobe. Dabei berechnet sich die Richtigkeit (Korrektklassifikationsrate; accuracy) als die Anteil aller Halodries, die als Halodries erkannt wurden (29) plus aller Nicht-Halodries die als solche erkannt wurden (347). Dieser Anteil bezieht sich auf alle Fälle des Datensatzes.

```
test_df <- test_df %>%
  mutate(halodrie_predict = predict(baum2,
                                    type = "class",
                                    newdata = test_df))

test_df %>%  # Konfusionsmatrix
  count(is_halodrie, halodrie_predict)  -> conf_matrix_2
conf_matrix_2
#> # A tibble: 4 x 3
#>   is_halodrie halodrie_predict     n
#>   <fctr>        <fctr> <int>
#> 1      ja          ja      5
#> 2      ja         nein    23
#> 3     nein          ja      7
#> 4     nein         nein   86

(29+347) / nrow(train_df)
```

```
#> [1] 0.783
```

Mit einer Richtigkeit von .78 ist das Modell nur wenig besser als das Nullmodell, also ein Baum “ohne Äste”. Als Nullmodell definieren wir das Modell, in dem jeder Fall der häufigsten Klasse zuordnen wird (sicher keine ganz doofe Strategie, wenn man sonst keine Informationen hat). In diesem Fall hätten wir eine Richtigkeit von .75 (wie oben berechnet). Vergleichen wir diese Werte jetzt mit Güte in der Test-Stichprobe; dort ist die Richtigkeit zumeist geringer als in der Trainings-Stichprobe.

```
test_df <- test_df %>%
  mutate(halodrie_predict = predict(baum2,
                                    type = "class",
                                    newdata = test_df))

confusionMatrix(
  data = test_df$halodrie_predict,
  reference = test_df$is_halodrie
)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction ja nein
#>      ja     5     7
#>      nein  23   86
#>
#>           Accuracy : 0.752
#>             95% CI : (0.665, 0.826)
#>    No Information Rate : 0.769
#>    P-Value [Acc > NIR] : 0.70965
#>
#>           Kappa : 0.129
#> McNemar's Test P-Value : 0.00617
#>
#>           Sensitivity : 0.1786
#>           Specificity : 0.9247
#>    Pos Pred Value : 0.4167
#>    Neg Pred Value : 0.7890
#>       Prevalence : 0.2314
#>    Detection Rate : 0.0413
#> Detection Prevalence : 0.0992
#>   Balanced Accuracy : 0.5517
#>
#> 'Positive' Class : ja
```

#>

Die Funktion `confusionMatrix` aus `caret` berechnet bequem alle relevanten Statistiken für uns. Unser Baum kommt in diesem Beispiel nicht gut weg; das Nullmodell hat eine Richtigkeit von .77 (`No Information Rate`), was besser ist als unser `baum2` mit .75. Wer vorsichtig ist, der rechne per Hand nach:

```
test_df %>%
  count(is_halodrie, halodrie_predict)
#> # A tibble: 4 x 3
#>   is_halodrie halodrie_predict     n
#>   <fctr>          <fctr> <int>
#> 1 ja              ja      5
#> 2 ja              nein    23
#> 3 nein            ja      7
#> 4 nein            nein    86

(5+86)/nrow(test_df)
#> [1] 0.752

mosaic::tally(~is_halodrie, data = test_df, format = "proportion")
#> is_halodrie
#>   ja  nein
#> 0.231 0.769
```

Wieso ist das Baummodell schlechter als das Nullmodell? Schauen wir uns die Richtigkeit näher an, und zwar für Halodries (positive Fälle; Sensitivität) und für Nicht-Halodries. Das Nullmodell weist alle Fälle der Klasse *Nicht-Halodrie* zu; damit ist die Sensitivität 0 und die Spezifität 1. Das Baummodell hat eine Sensitivität von .18 und eine Spezifität von .92; damit ist die Sensitivität ein ganzes Stück besser als beim Nullmodell, aber die Spezifität nur ein bisschen schlechter. Eigentlich ein guter Tausch, oder? Leider gibt es viel mehr negative Fälle als positive, so dass man besser damit beraten ist, die häufigere Klasse genau zu treffen. Ein extremes Beispiel soll das verdeutlichen: Angenommen, es wären 99% Nicht-Halodries im Datensatz. Dann ist es ein leichtes, 99% Richtigkeit zu erreichen, in dem man das einfache Nullmodell anwendet. Ein überlegteres Modell hat es schwer: Auch hohe Verbesserung der Sensitivität werden leichte Einbußen der Spezifität nicht wett machen. Nehmen wir an, es sind 100 Fälle im Datensatz, davon 99 Nicht-Halodries. Sagen wir, ein Modell finde alle positiven Fälle (hier nur 1 Halodrie), damit läge die Sensitivität bei 100%. Dafür stufen wir aber, nehmen wir an, 10% der Nicht-Halodries falsch ein; hier sinkt die Genauigkeit also um 10 Individuen. Insgesamt ist die Bilanz schlecht: Einen gewonnen, 10 verloren.

Bei der Betrachtung der Gütekriterien sollte man die Randverteilung des Kriteriums im Blick haben.

19.1.5 Der Algorithmus der Entscheidungsbäume

Grob gesagt, ist der Ablauf bzw. der *Algorithmus* der Entscheidungsbäume so:



1. Starte mit der gesamten Stichprobe als “Wurzelknoten”.
2. Partitioniere den aktuellen Knoten so, dass die resultierenden zwei Knoten insgesamt homogener sind als der aktuelle Knoten.
3. Weise den resultierenden zwei Knoten die entsprechende Teilstichprobe zu.
4. Wiederhole Schritte 2 und 3 bis ein Stop-Kriterium erreicht wird.
5. Für jede Observation, die in den gleichen Endknoten fällt, mache die gleiche Vorhersage - und zwar die häufigste Klasse.

Typische *Stop-Kriterien* sind a) dass ein Knoton 100% homogen ist (nur noch Fälle einer Klasse), b) dass die Stichprobengröße einer Klasse unter einen Grenzwert (z.B. $n=5$ fällt) oder c) dass der Homogenitätszuwachs unter einen Grenzwert (z.B. $<1\%$) fällt.

Das Partitionieren (Aufteilen, Splitten) wird so ausgeführt, dass die resultierenden zwei Klassen immer disjunkt (überlappungsfrei) sind. Da dieses Partitionieren wiederholt (rekursiv) ausgeführt wird, spricht man bei baumbasierten Modellen auch von *rekursivem Partitionieren*. Wie geht der Algorithmus vor, um zu wissen, welche Variable und welcher Schnittpunkt die höchste Homogenität verspricht? Es werden einfach alle Variablen und alle relevanten Schnittpunkte ausprobiert. Das hört sich viel an, aber es ist nicht so schlimm, da bei n Beobachtungen nur $n - 1$ Schnittpunkte ausprobiert werden müssen: Gibt es zwei Beobachtungen, so wird die (metrische) Variable in der Mitte zwischen der Werten der beiden Beobachtungen geteilt. Bei nominalen Variablen mit vielen Kategorien kann der Algorithmus ins Schwitzen kommen, da u.U. viele Schnittpunkt möglich sind.

19.1.6 Maße der Homogenität

Wie erläutert, sucht der Algorithmen nach Schnittpunkten, die die insgesamte Homogenität der beiden resultierenden Äste maximiert. Dazu wird das um die Fallzahl der Äste gewichtete Maß der Homogenität berechnet. Eine einfache Möglichkeit bestünde darin, den Klassifikationsfehler E zu minimieren:

$$E = f/n = 1 - k/n$$

wobei f die Anzahl der falsch klassifizierten Fälle, k die Anzahl der korrekt klassifizierten Fälle bezeichnet; n steht für die Anzahl aller zu klassifizierenden Fälle. Da die Vorhersage

gleich der häufigsten Klasse ist, ist der Klassifikationsfehler E gleich dem Anteil aller Fälle, die nicht der größten Klasse angehören:

$$E = 1 - \max(p_i)$$

Dabei bezeichnet p_i die Klasse i ; $\max(p_i)$ gibt also den Anteil der größten (der beiden) Klassen zurück.

Weitere Homogenitätsmaße sind der *Gini-Index* und die *Entropie*; da die Entropie ähnliche Werte liefert wie der Gini-Index, beschränken wir uns hier auf den Gini-Index (s. Tan (2013) für Details). Der Gini-Index G kann als Varianz der Klassenateile verstanden werden:

$$G = 1 - \sum p_i^2$$

Die Homogenitätsmaße sind alle minimal (null), wenn alle Fälle zu einer Klasse gehören (d.h. wenn $p = 0$ oder $p = 1$). Betrachten wir einige Beispiele dazu; dabei beziehen wir uns auf die Anzahl der Halodries H bzw. Nicht-Halodries N .

```
# node1: H = 40, N = 40
E1 <- 1 - max(c(40/80, 40/80)); E1
#> [1] 0.5
G1 <- 1 - sum(c(.5^2, .5^2)); G1
#> [1] 0.5

# node2: H = 10, N = 30
E2 <- 1 - max(c(10/40, 30/40)); E2
#> [1] 0.25
G2 <- 1 - sum(c(.25^2, .75^2)); G2
#> [1] 0.375

# node3: H = 20, N = 40
E3 <- 1 - max(c(20/60, 40/60)); E3
#> [1] 0.333
G3 <- 1 - sum(1/3^2, 2/3^2); G3
#> [1] 0.667

# node4: H = 20, N = 0
E4 <- 1 - max(c(20/20, 0)); E4
#> [1] 0
G4 <- 1 - sum(1^2, 0^2); G4
#> [1] 0
```

Betrachten Sie den Baum, wie in Abbildung 19.5 dargestellt. Der Klassifikationsfehler in den Blättern ist im linken Baum (A) gleich groß wie im rechten Baum (B), wie wir oben

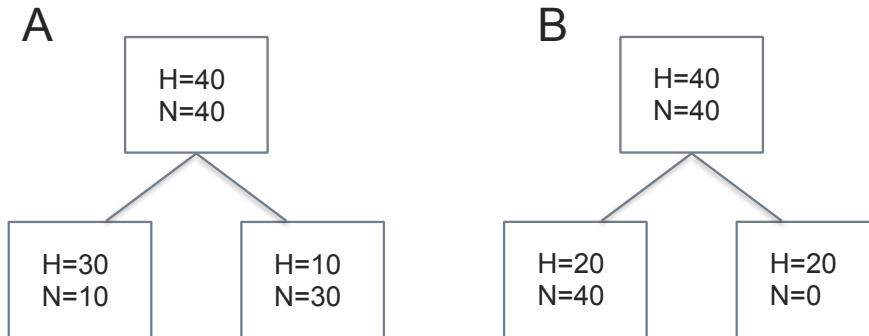


Abbildung 19.5: Klassifikationsfehler und Gini-Koeffizient kommen manchmal zu unterschiedlichen Entscheidungen

ausgerechnet haben (jeweils 20/60). Sind also beide Lösungen (A vs. B) gleich gut? Das wäre die Antwort des Klassifikationsfehlers; der Gini-Koeffizient bevorzugt Lösung B¹:

```
Gini_A <- 1 - .5 * G2 - .5 * G2; Gini_A
#> [1] 0.625
Gini_B <- 1 - 6/8 * G3 - 2/8 * G4; Gini_B
#> [1] 0.5

Klass_A <- 1 - .5 * E2 - .5 * E2; Klass_A
#> [1] 0.75
Klass_B <- 1 - 6/8 * E3 - 2/8 * E4; Klass_B
#> [1] 0.75
```

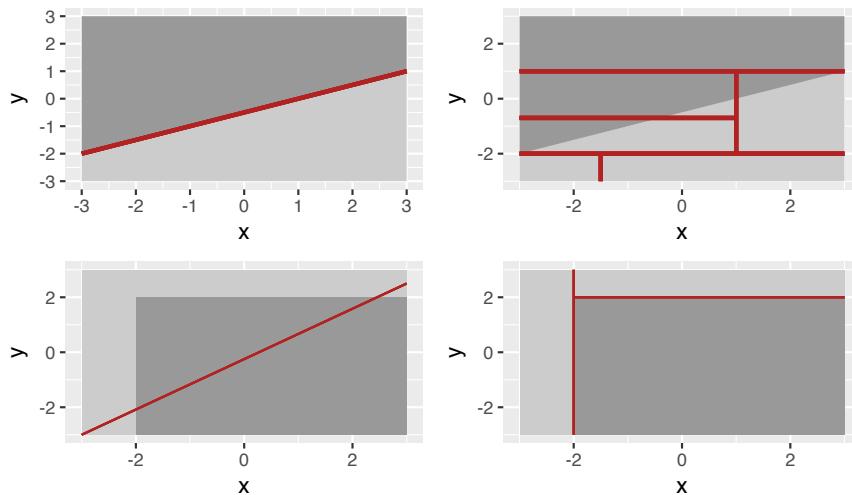
Hier haben wir das Homogenitätsmaß der Partitionierung als gewichtete Summe der beiden resultierenden Blätter berechnet; dabei kam das Gini-Maß (G) bzw. der Klassifikationsfehler (E) zum Einsatz.

Zur Partitionierung (“Wachstum”) eines Baumes ist der Klassifikationsfehler nicht ideal; der Gini-Index oder das Entropie-Maß sind besser geeignet.

19.1.7 Regressionsbäume

Regressionsbäume sind ähnlich zu den Klassifikationsbäumen; allerdings wird hier unter Homogenität nicht ein hoher Anteil der Klasse verstanden, sondern möglichst geringe Abweichung der Vorhersage vom tatsächlichen Wert. Für alle Fälle, die im gleichen Blatt landen, wird wiederum der gleiche Wert vorhergesagt; dieses Mal der Mittelwert der Fälle dieses Blattes.

¹Raschka (o. J.)



besser darauf ausgelegt als ein Entscheidungsbaum. Unten: In diesem Fall sind

Abbildung 19.6: Vergleich von linearen Modellen und Entscheidungsbäumen

19.1.8 Stärken und Schwächen von Bäumen

Bäume sind ein verbreitetes Werkzeug der statistischen Modellierung; man kann sie als nonparametrische Regression verstehen (Strobl und Malley 2009). Im Gegensatz linearen Modellen, wo Prädiktoren linear kombiniert werden, wird hier der Prädiktorenraum in rechteckige Region geteilt. Je nach dem, ob die Entscheidungsgrenze besser mit Rechtecken oder mit einer geraden (bzw. Polynomfunktion) zu modellieren ist, wird entweder ein lineares Modell oder ein Entscheidungsbäum besser passen (vgl. Abbildung 19.6, angelehnt an [James, Witten, Hastie, und Tibshirani (2013b); S. 315]).

Die Stärken von Entscheidungsbäumen sind ihre intuitive Plausibilität, die sich auch in der guten Visualisierbarkeit niederschlägt. Menschliche Entscheidungsprozesse könnten dem von Entscheidungsbäumen in einigen Aspekten ähnlich sein (James, Witten, Hastie, und Tibshirani 2013b). Qualitative Variablen können ohne weitere Vorverarbeitung problemlos eingegeben werden.

Der Nachteil von Entscheidungsbäumen ist ihre Instabilität; nur wenig Änderung in der zugrundeliegenden Stichprobe kann den Baum massiv verändern. Außerdem ist die prädiktive Güte von Entscheidungsbäumen nicht so hoch wie das anderer Verfahren.

Ein weiteres Problem betrifft die Frage der Überanpassung (Overfitting): Zu viele Knoten führen dazu, dass Rauschen, bedingt durch die Zufälligkeit der Stichprobeneinziehung, in das Modell einfließt. Eine Gegenmaßnahme dazu ist das sog. *Pruning*, das „Zurückschneiden“ der Baumäste, um den Baum wieder einfacher (weniger Knoten) zu gestalten; s. (James, Witten, Hastie, und Tibshirani 2013b) für Details dazu.

Den Effekt von Überanpassung ist in Abbildung 19.7 dargestellt: Die Richtigkeit im Trainings-Sample nimmt mit Anzahl der Knoten zu; für das Test-Sample gilt das nicht: Bei drei Knoten erreicht die Richtigkeit ihren Maximalwert, danach steigt sie nicht weiter. Im Gegenteil: häufig sinkt die Test-Richtigkeit mit zunehmender Komplexität des Modells, da Zufallsrauschen im

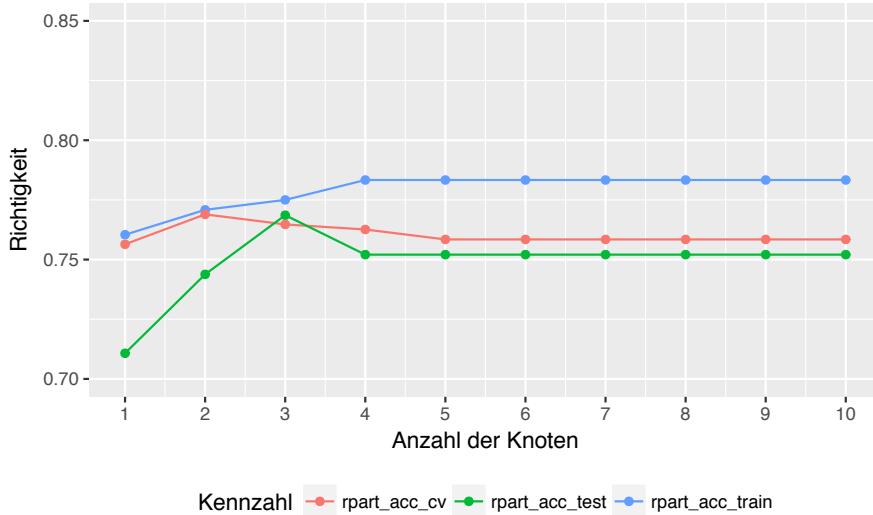


Abbildung 19.7: Klassifikationsgüte in Abhängigkeit von der Anzahl der Knoten im Baum

Modell für bare Münze genommen wird, was schlechte Vorhersagen produzieren muss. Ein ähnliches Bild zeigt die Richtigkeitskurve für die kreuzvalidierten Daten: Ab einer gewissen Komplexität des Modells (4 Knoten) steigt die Richtigkeit nicht weiter.

19.2 Bagging

Ein weiterer Nachteil von Entscheidungsbäumen ist ihre hohe Varianz, in Vergleich zu anderen Modellen: Ändert sich die Beschaffenheit der Stichprobe ein wenig, kann der Entscheidungsbaum deutlich anders aussehen. Eine Möglichkeit, die Varianz zu verringern, ist es, viele Stichproben zu ziehen. Zieht man mehrere Stichproben und berechnet jeweils den Mittelwert, so *sinkt* die Varianz bei steigender Anzahl von Stichproben (James, Witten, Hastie, und Tibshirani 2013b).

Entsprechend wäre es eine gute Strategie, viele Stichproben aus der Population zu ziehen, für jede einen Entscheidungsbaum zu berechnen, und schließlich den Mittelwert der Vorhersagen pro Beobachtungen als Schätzwert zu nehmen. Leider ist es unpraktisch, viele Stichproben zu ziehen. 1000 Mal eine Stichprobe an Halodries zu ziehen... anstrengend. Daher wendet man einen Trick an: Man betrachtet die einzelne Stichprobe als Population. Dann zieht aus dieser “Pseudo-population” einfach viele Stichproben; dabei legt man nach jeder Ziehung die Beobachtungen wieder zurück. Zwangsläufig werden dabei in einer Bootstrap-Stichprobe einige Beobachtungen mehrfach und andere vielleicht gar nicht gezogen werden. Mit diesem Vorgehen wird auch etwas Zufall in das Modell injiziert: Die Stichproben variieren.

Diese Idee heißt *Bootstrapping* (vgl. Abschnitt XXX). Für jede dieser Bootstrap-Samples können wir dann ein Modell (z.B. Entscheidungsbaum) berechnen. Zum Schluss nehmen wir den Mittelwert aller Modelle als Vorhersagewert für jede Beobachtung. Dieses Prinzip heißt *Bagging*. Durch Bagging verringern wir die Varianz der Schätzung, das ist der Vorteil. Ein Nachteil ist, dass das Modell nicht mehr gut zu interpretieren ist: Einen Baum kann

man einfach beschreiben, aber wie soll man den “mittleren Baum” zusammenfassen? Die Interpretierbarkeit leidet.

Durch das Ziehen mit Zurücklegen werden im Schnitt nur etwa zwei Drittel der Beobachtungen gezogen. Das verbleibende Drittel wird also nicht zum Training des Baumes verwendet. Diesen Teil an nicht gezogenen Beobachtungen nennt man die *out of bag* (OOB) Beobachtungen oder das OOB-Sample. Es bietet sich an, die Vorhersagen jedes einzelnen Baumes des Bagging-Modells anhand der OOB-Beobachtungen durchzuführen, um so genauere Einschätzungen der Modellgüte zu bekommen.

In caret sind einige Methoden dazu implementiert²; man übergibt dafür der Funktion `train()` für das Argument `method` den Wert `treebag`.

19.3 Eine Bootstrapping-Simulation

Erkunden wir die Idee, dass ein Mittelwert aus *vielen* (unabhängigen) Stichproben *weniger* schwankt, als ein Mittelwert aus wenigen Stichproben. Ziehen wir zuerst nur wenige Stichproben - eine kleine Stichprobe an Stichproben:

```
data(stats_test)
```

```
samples1 <- vector(mode = "list", length = 5)
# Erstelle einen Vektor des Typs "list" mit Länge 10,
# also eine Liste der Länge 10

set.seed(42)
samples1 %>%
  map(~sample(stats_test$score, size = 10,
              replace = TRUE)) %>%
  map(mean, na.rm = TRUE) %>%
  simplify %>%
  sd -> sd_wenig_stipros
```

Übersetzen wir die Pfeife ins Deutsche:



Nimm den Vektor `samples` UND DANN
 ordne jedem Element eine Stichprobe der Größe 20 zu UND DANN
 ordne jedem Element die Funktion `mean` zu ...
 berechne also für jede Stichprobe den Mittelwert UND DANN

²<https://topepo.github.io/caret/train-models-by-tag.html#boosting>

vereinfache die Liste zu einen reinen Vektor UND DANN berechne die sd dieses Vektors.

Jetzt ziehen wir wieder Stichproben, aber nicht nur 10, sondern viel mehr, 1000; dann vergleichen dann die Variabilität:

```
samples2 <- vector(mode = "list", length = 1000)

set.seed(42)
samples2 %>%
  map(~sample(stats_test$score, size = 1000,
              replace = TRUE)) %>%
  map(mean, na.rm = TRUE) %>%
  simplify %>%
  sd -> sd_viele_stipros

sd_wenig_stipros / sd_viele_stipros
#> [1] 7.89
```

Die Standardabweichungen unterscheiden sich fast um eine Größenordnung (Faktor 8): Die sd ist im zweiten Durchgang deutlich gesunken. Je größer die Stichprobe an Stichproben, desto stabiler wird der Mittelwert³ (weniger Variabilität).

19.4 Random Forests

19.4.1 Grundlagen

Random Forests, “Zufallswälder” an Entscheidungsbäumen führen noch eine zweite Schicht Zufall in das Modell ein. Wie beim Bagging werden viele Bäume aus gebootstrappten Stichproben gezogen. Aber im Unterschied zu Bagging, wird bei jeder Astgabelung eine *kleine Zufallsstichprobe* an m Prädiktoren gezogen, die zur Bestimmung des Gabelung herangezogen wird. Nicht alle Prädiktoren des Datensatzes werden berücksichtigt, sondern nur m zufällige Prädiktoren. Bei jeder Astgabelung wird wieder eine neue Stichprobe an m Prädiktoren gezogen. Das hat natürlich den Effekt, dass Variablen mit viel Einfluss auf das Kriterium weniger häufig zum Zuge kommen. Was sich als ein Nachteil anhört, stellt sich aber als ein Vorteil heraus: Ohne diese zufällige Begrenzung würden stets ähnliche Prädiktoren - die einflussreichsten - ausgewählt werden. Die Bäume würden einander stark ähneln. Anders ausgedrückt, ihre Vorhersagen wären korreliert. Leider reduziert die Aggregation von korrelierten Beobachtungen die Variabilität nicht so stark wie von unkorrelierten Beobachtungen. Korrelierte Bäume bringen nicht viel zusätzliche Information. Man kann Random Forests demnach

³ $n > 1$ vorausgesetzt

als eine Methode zur Dekorrelation der Bäume verstehen. In vielen Fällen funktioniert die Methode erstaunlich gut: Die Vorhersagegüte von Random Forest Modellen ist oft gut.

Der Unterschied zu Bagging ist also, dass nur m Prädiktoren pro Astgabelung berücksichtigt werden, nicht alle p Prädiktoren. Wie viele Prädiktoren werden gewählt, wie groß soll m sein? Häufig nimmt man $m = \sqrt{p}$, wobei es sich anbietet, verschiedene Werte auszuprobieren. m ist damit ein *Tuningparameter* (Hyper-, Metaparameter) des Modells; ein Parameter, der vom Anwender bestimmt wird, um es einfacher auszudrücken. Würde man $m = p$ setzen, so gliche das Random Forest Modell dem Bagging; damit kann Bagging als Spezialfall vom Random Forest Modell gesehen werden. Natürlich muss jedes Ausprobieren im Trainings-Sample stattfinden; erst das finale Modell darf das Test-Sample sehen.

In Software-Umsetzungen von Random Forest wird m meist als `mtry` bezeichnet und die Anzahl der Bäume mit `ntree`, so auch bei `caret` und dem zugrundliegenden Paket `randomForest`. `mtry` ist der Tuningparameter für Random Forests (zumindest für die Umsetzung, die wir verwenden). `ntree` wird nicht `caret` als Tuningparameter gezählt, da es meist so ist, dass es Zahl von Bäumen gibt, bis zu der die Leistung des Modell monoton steigt. Ab einer gewissen Zahl an Bäumen wird ein Plateau erreicht und die Leistung steigt nicht weiter an⁴.

Die `Caret`-Syntax ist wie gesagt für alle Modelle gleich; `caret` probiert auch selbständig ein paar Werte für die Metaparameter durch; im Standard sind das insgesamt 3^k , wobei k die Anzahl der Metaparameter ist (in unserem Fall $k = 1$).

```
my_train_control <- trainControl(
  method = "cv",
  number = 10
)

rf1 <- train(is_halodrie ~ .,
             data = train_df,
             method = "rf",
             trControl = my_train_control,
             importance = TRUE,
             allowParallel=TRUE  # mehrere Kerne dürfen gleichzeitig rechnen
           )

rf1
#> Random Forest
#>
#> 480 samples
#>   8 predictor
```

⁴man könnte `ntree` daher eher als Schokoladenparameter bezeichnen: Eine gewissen Menge wünscht man sich, aber mehr bringt keinen Zusatznutzen; `randomforest` und damit `caret` als Schnittstelle zu `randomforest` nimmt `ntree = 500` als Standard, vgl. `?randomForest`

```
#> 2 classes: 'ja', 'nein'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 432, 432, 432, 432, 432, 431, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy  Kappa
#>   2     0.738     0.161
#>   5     0.725     0.183
#>   8     0.706     0.144
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

Das Modell mit `mtry = 2` schnitt also hinsichtlich Genauigkeit (accuracy) am besten ab. Kappa wäre hier eine alternative Entscheidungsgröße.

Man kann `train` auch die Werte des oder der Tuningparameter vorgeben, dann probiert `caret` brav alle die diese Werte durch. Dazu legt man eine Tabelle an, in der alle Werte der Tuningparameter miteinander kombiniert werden. Hat man z.B. 2 Tuningparameter mit jeweils 3 Werten, so müssten $3^2 = 9$ Werte durchprobiert werden. Das Anlegen dieser Tabelle kann man mit `expand.grid()` machen. Man übergibt die Tuningparameter und ihre Werte, und `expand.grid()` erstellt eine Tabelle mit allen Kombinationen. Da es hier nur einen Tuningparameter gibt, ist die Kombination recht witzlos, aber andere Modelle können mehr Tuningparameter haben.

```
my_tuning_grid <- expand.grid(.mtry = 1:10)
```

Sehen wir nach, ob die Erweiterung des Suchraums die Vorhersagegüte verbessert; wie Sie vielleicht merken, kann es etwas dauern, bis das Modell durchgerechnet ist. Zwar darf `ntree` nicht als Tuningparameter übergeben werden, aber wir können `train()` einen konstanten Wert übergeben:

```
rf2 <- train(is_halodrie ~ .,
              data = train_df,
              method = "rf",
              trControl = my_train_control,
              importance = TRUE,
              ntree = 1000,
              tuneGrid = my_tuning_grid,
              allowParallel = TRUE  # mehrere Kerne dürfen gleichzeitig rechnen
            )
```

```
rf2$results %>%
  top_n(n = 1, wt = Accuracy)
#>   mtry Accuracy    Kappa AccuracySD KappaSD
#> 1     1      0.746 0.00691     0.00801    0.038
```

Mit nur $m = 2$ Prädiktoren wird die beste Vorhersagegüte erzielt. Schauen wir nun, wie gut unsere Vorhersagen im Test-Sample sind:

```
rf2_predict <- predict(rf2, newdata = test_df, type = "raw")
postResample(pred = rf2_predict, obs = test_df$is_halodrie)
#> Accuracy    Kappa
#> 0.769      0.000
```

Die Funktion `postResample` erledigt das sehr bequem. Wie man sieht, die die Vorhersagegüte *geringer* als im Trainings-Sample. Das Modell hat etwas “überfittet”, es hat Rauschen modelliert, was zu Scheingenaugkeit geführt hat. Als das Modell versucht hat, die scheinbar erkannten Regeln auf die unbekannten Daten anzuwenden, hat das nicht komplett geklappt. Die aufschlussreichere Kennzahl ist allerdings Kappa. Kappa setzt die beobachtete Genauigkeit in Bezug zur erwarteten Genauigkeit, zur zufällig erwartbaren Genauigkeit also. Es gibt verschiedene Richtwerte, was ein kleines bzw. ein großes Kappa ist [Banerjee u. a. (1999); Brennan_1981], aber .04 gehört immer zu “schlecht”! Unser Modell ist also wenig besser als blindes Raten. Kann eigentlich nur noch besser werden.

19.4.2 Variablenrelevanz

Ein Vorteil von linearen Modellen ist, dass die Relevanz von Variablen gut ersichtlich ist. Wie ist das bei Random Forests? Kann man dort auch eine Einschätzung zur *Variablenrelevanz* (variable importance) bekommen? Ja. Eine typische Überlegung dazu geht so: Angenommen wir die Werte einer Variablen wild durchmischen und dann mit dieser gemischten (permutierten) Variablen das Modell neu berechnen - dann müsste die Vorhersage schlechter werden, wenn die Variable vorher wichtige Informationen zur Verhersage inne hatte. Durch die Verringerung der Vorhersagegüte können wir die Relevanz einer Variable abschätzen. Gibt man bei `train()` das Argument `importance = TRUE` an (was wir getan haben), so werden die Relevanzwerte mitberechnet. Mit `varImp()` kann man sie sich die Werte ausgeben lassen, wenn man das Modellobjekt an die Funktion übergibt:

```
varImp(rf2)
#> rf variable importance
#>
#>           Importance
#> rating             100.00
```

```
#> religiousness      38.69
#> yearsmarried      33.93
#> gendermale        14.95
#> age                13.70
#> childrenyes       11.17
#> education          6.57
#> occupation         0.00
```

Religiösität wurde als wichtigster Prädiktor identifiziert; die Werte sind so skaliert, dass der beste Prädiktor (mit der größten Verringerung der Vorhersagüte) einen Wert von 100 bekommt. Die übrigen Prädiktoren sind relativ dazu skaliert. Die Absolutwerte bekommt man so:

```
varImp(rf2, scale = FALSE)
#> rf variable importance
#>
#>                               Importance
#> rating                      17.508
#> religiousness                 5.837
#> yearsmarried                  4.931
#> gendermale                    1.319
#> age                           1.080
#> childrenyes                   0.600
#> education                     -0.277
#> occupation                    -1.527
```

Dieser Wert berechnet sich als die Verringerung der Genauigkeit durch das Permutieren der Variable pro Baum. Dieser Wert wird dann gemittelt und durch die Standardabweichung geteilt (Liaw und Wiener 2002). Allerdings raten vorsichtige Gemüter dazu, die Abstände dieser Werte mit Vorsicht zu genießen und nur die Rangfolge zu beachten⁵. Weiterhin haben verschiedenen Modell verschiedene Maße der Variablenrelevanz, so dass ein absoluter Vergleich kaum möglich ist, aber auf Rangebene eher.

⁵zumal die Berechnung der Variablenrelevanz von `caret` offenbar nicht identisch ist mit der von `randomForest`

Teil VIII

Ungeleitetes Modellieren

Kapitel 20

Clusteranalyse

Benötigte Pakete:

```
library(tidyverse)
library(cluster)
```

```
library(broom)
```



Lernziele:

- Das Ziel einer Clusteranalyse erläutern können.
- Das Konzept der euklidischen Abstände verstehen.
- Eine k-Means-Clusteranalyse berechnen und interpretieren können.

20.1 Grundlagen der Clusteranalyse

Das Ziel einer Clusteranalyse ist es, Gruppen von Beobachtungen (d. h. *Cluster*) zu finden, die innerhalb der Cluster möglichst homogen, zwischen den Clustern möglichst heterogen sind. Um die Ähnlichkeit von Beobachtungen zu bestimmen, können verschiedene Distanzmaße herangezogen werden. Für metrische Merkmale wird z. B. häufig die euklidische Metrik verwendet, d. h., Ähnlichkeit und Distanz werden auf Basis des euklidischen Abstands bestimmt. Aber auch andere Abstände wie "Manhattan" oder "Gower" sind möglich. Letztere haben den Vorteil, dass sie nicht nur für metrische Daten sondern auch für gemischte Variablentypen verwendet werden können. Wir werden uns hier auf den euklidischen Abstand konzentrieren.

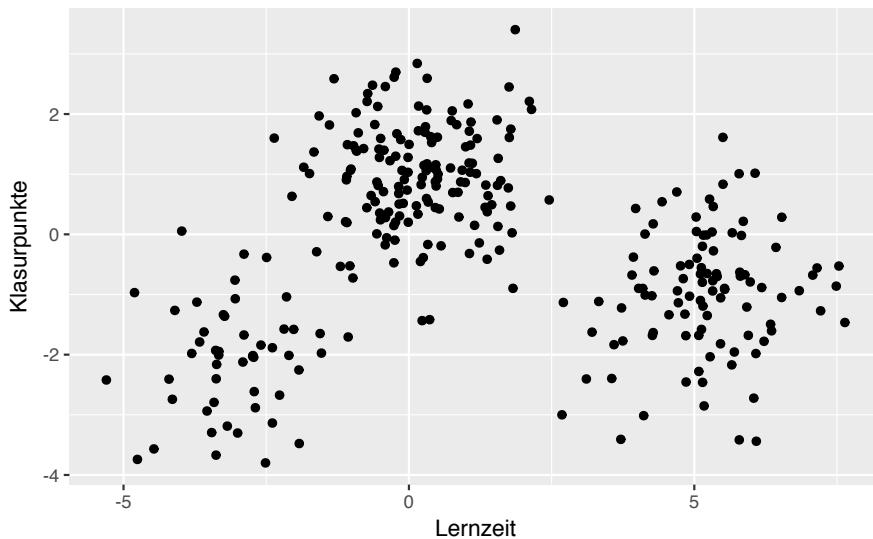


Abbildung 20.1: Ein Streudiagramm - sehen Sie Gruppen (Cluster) ?

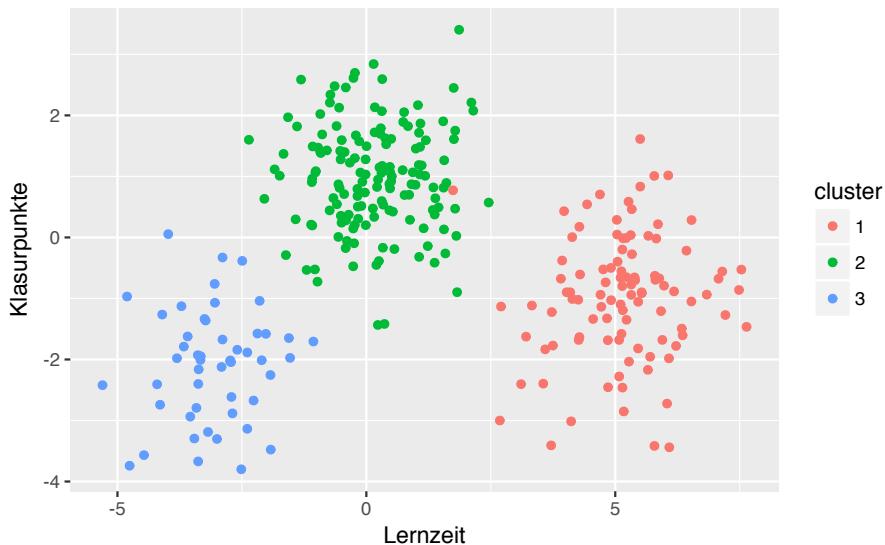


Abbildung 20.2: Ein Streudiagramm - mit drei Clustern

20.1.1 Intuitive Darstellung der Clusteranalyse

Betrachten Sie das folgende Streudiagramm (die Daten sind frei erfunden; “simuliert”, sagt der Statistiker). Es stellt den Zusammenhang von Lernzeit (wie viel ein Student für eine Statistikklausur lernt) und dem Klausurerfolg (wie viele Punkte ein Student in der Klausur erzielt) dar. Sehen Sie Muster? Lassen sich Gruppen von Studierenden mit bloßem Auge abgrenzen (Abb. 20.1)?

Färben wir das Diagramm mal ein (Abb. 20.2).

Nach dieser “Färbung”, d.h. nach dieser Aufteilung in drei Gruppen, scheint es folgende “Cluster”, “Gruppen” oder “Typen” von Studierenden zu geben:

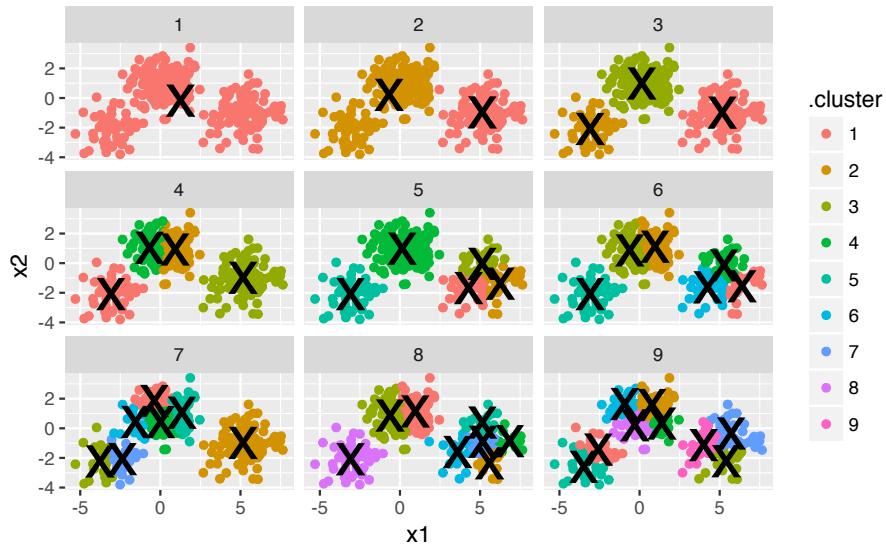


Abbildung 20.3: Unterschiedliche Anzahlen von Clustern im Vergleich

- “Blaue Gruppe”: Fälle dieser Gruppe lernen wenig und haben wenig Erfolg in der Klausur. Tja.
- “Rote Gruppe”: Fälle dieser Gruppe lernen viel; der Erfolg ist recht durchwachsen.
- “Grüne Gruppe”: Fälle dieser Gruppe lernen mittel viel und erreichen einen vergleichsweise großen Erfolg in der Klausur.

Drei Gruppen scheinen ganz gut zu passen. Wir hätten theoretisch auch mehr oder weniger Gruppen unterteilen können. Die Clusteranalyse gibt keine definitive Anzahl an Gruppen vor; vielmehr gilt es, aus theoretischen und statistischen Überlegungen heraus die richtige Anzahl auszuwählen (dazu gleich noch mehr).

Unterteilen wir zur Illustration den Datensatz einmal in bis zu 9 Cluster (Abbildung 20.3).

Das “X” soll den “Mittelpunkt” des Clusters zeigen. Der Mittelpunkt ist so gewählt, dass die Distanz von jedem Punkt zum Mittelpunkt möglichst kurz ist. Dieser Abstand wird auch “Varianz innerhalb des Clusters” oder kurz “Varianz within” bezeichnet. Natürlich wird diese Varianz within immer kleiner, je größer die Anzahl der Cluster wird.

Die vertikale gestrichelte Linie zeigt an, wo die Einsparung an Varianz auf einmal “sprunghaft” weniger wird - just an jedem Knick bei $x=3$; dieser “Knick” wird auch “Ellbogen” genannt (da sage einer, Statistiker haben keine Phantasie). Man kann jetzt sagen, dass 3 Cluster eine gute Lösung seien, weil mehr Cluster die Varianz innerhalb der Cluster nur noch wenig verringern. Diese Art von Diagramm wird als “Screeplot” bezeichnet. Fertig!

20.1.2 Euklidische Distanz

Aber wie weit liegen zwei Punkte entfernt? Betrachten wir ein Beispiel. Anna und Berta sind zwei Studentinnen, die eine Statistikklausur geschrieben haben müssen (bedauerns-

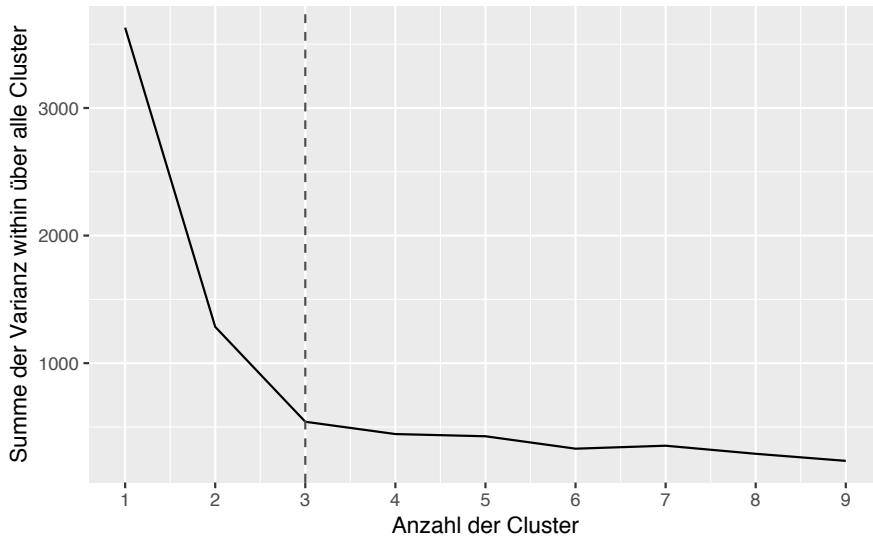


Abbildung 20.4: Die Summe der Varianz within in Abhängigkeit von der Anzahl von Clustern. Ein Screeplot.

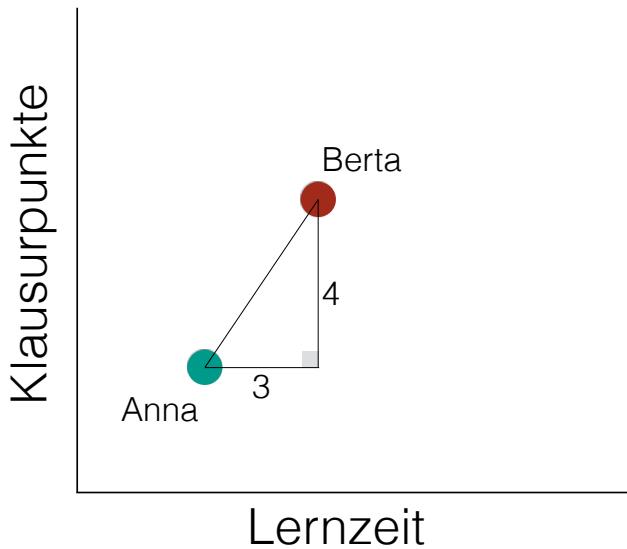


Abbildung 20.5: Distanz zwischen zwei Punkten in der Ebene

wert). Die beiden unterscheiden sich sowohl in Lernzeit als auch in Klausurerfolg. Aber wie sehr unterscheiden sie sich? Wie groß ist der “Abstand” zwischen Anna und Berta (vgl. Abb. 20.5)?

Eine Möglichkeit, die Distanz zwischen zwei Punkten in der Ebene (2D) zu bestimmen, ist der *Satz des Pythagoras* (leise Trompetenfanfare). Generationen von Schülern haben diese Gleichung ähmm... geliebt:

$$c^2 = a^2 + b^2$$

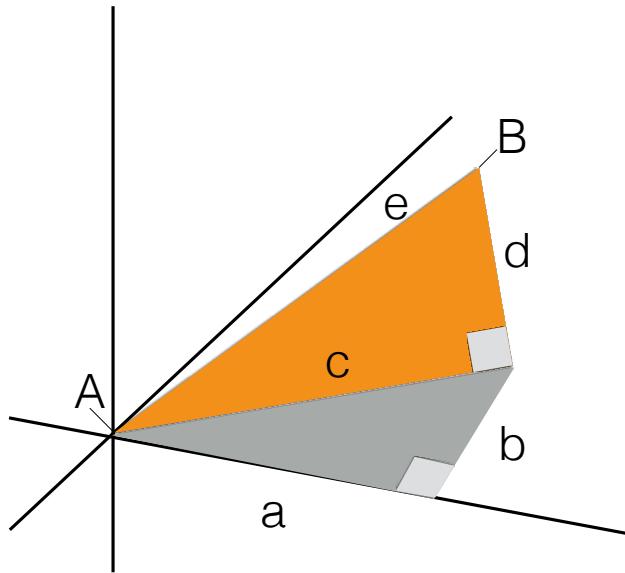


Abbildung 20.6: Pythagoras in 3D

In unserem Beispiel heißt das $c^2 = 3^2 + 4^2 = 25$. Folglich ist $\sqrt{c^2} = \sqrt{25} = 5$. Der Abstand oder der Unterschied zwischen Anna und Berta beträgt also 5 - diese Art von “Abstand” nennt man den *euklidischen Abstand*.

Aber kann man den euklidischen Abstand auch in 3D (Raum) verwenden? Oder gar in Räumen mehr mehr Dimensionen??? Betrachten wir den Versuch, zwei Dreiecke in 3D zu zeichnen. Stellen wir uns vor, zusätzlich zu Lernzeit und Klausurerfolg hätten wir als 3. Merkmal der Studentinnen noch “Statistikliebe” erfasst (Bertas Statistikliebe ist um 2 Punkte höher als Annas).

Sie können sich Punkt *A* als Ecke eines Zimmers vorstellen; Punkt *B* schwebt dann in der Luft, in einiger Entfernung zu *A*.

Wieder suchen wir den Abstand zwischen den Punkten *A* und *B*. Wenn wir die Länge *e* wüssten, dann hätten wir die Lösung; *e* ist der Abstand zwischen *A* und *B*. Im orangenen Dreieck gilt wiederum der Satz von Pythagoras: $c^2 + d^2 = e^2$. Wenn wir also *c* und *d* wüssten, so könnten wir *e* berechnen... *c* haben wir ja gerade berechnet (5) und *d* ist einfach der Unterschied in Statistikliebe zwischen Anna und Berta (2)! Also

$$e^2 = c^2 + d^2$$

$$e^2 = 5^2 + 2^2$$

$$e^2 = 25 + 4$$

$$e = \sqrt{29} \approx 5.4$$

Ah! Der Unterschied zwischen den beiden Studentinnen beträgt also ~5.4!

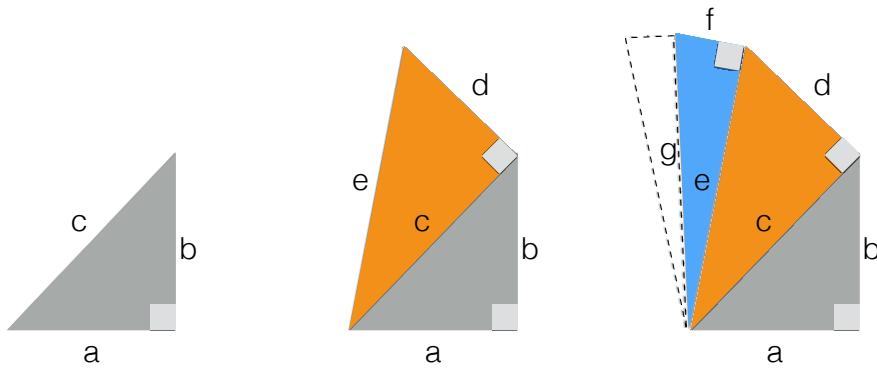


Abbildung 20.7: Pythagoras in Reihe geschaltet

Intuitiv gesprochen, “schalten wir mehrere Pythagoras-Sätze hintereinander”.

Der euklidische Abstand berechnet sich mit Pythagoras’ Satz!

Das geht nicht nur für “zwei Dreiecke hintereinander”, sondern der Algebra ist es wurscht, wie viele Dreiecke das sind.

Um den Abstand zweier Objekte mit k Merkmalen zu bestimmen, kann der euklidische Abstand berechnet werden mit. Bei $k=3$ Merkmalen lautet die Formel dann $e^2 = a^2 + b^2 + d^2$. Bei mehr als 3 Merkmalen erweitert sich die Formel entsprechend.

Dieser Gedanken ist mächtig! Wir können von allen möglichen Objekten den Unterschied bzw. die (euklidische) Distanz ausrechnen! Betrachten wir drei Professoren, die einschätzen sollten, wie sehr sie bestimmte Filme mögen (1: gar nicht; 10: sehr). Die Filme waren: “Die Sendung mit der Maus”, “Bugs Bunny”, “Rambo Teil 1”, “Vom Winde verweht” und “MacGyver”.

```
profss <- data_frame(
  film1 = c(9, 1, 8),
  film2 = c(8, 2, 7),
  film3 = c(1, 8, 3),
  film4 = c(2, 3, 2),
  film5 = c(7, 2, 6)
)
```

Betrachten Sie die Film-Vorlieben der drei Professoren. Gibt es ähnliche Professoren hinsichtlich der Vorlieben? Welche Professoren haben einen größeren “Abstand” in ihren Vorlieben?

Wir könnten einen “fünffachen Pythagoras” zu Rate ziehen. Praktischerweise gibt es aber eine R-Funktion, die uns die Rechnerei abnimmt:

```
dist(profs)
#>      1      2
```

```
#> 2 13.23
#> 3 2.65 10.77
```

Offenbar ist der (euklidische) Abstand zwischen Prof. 1 und 2 groß (13.2); zwischen Prof 2 und 3 auch recht groß (10.8). Aber der Abstand zwischen Prof. 1 und 3 ist relativ klein! Endlich hätten wir diese Frage auch geklärt. Sprechen Sie Ihre Professoren auf deren Filmvorlieben an...

20.1.3 k-Means Clusteranalyse

Beim k-Means Clusterverfahren handelt es sich um eine bestimmte Form von Clusteranalysen; zahlreiche Alternativen existieren, aber die k-Means Clusteranalyse ist recht verbreitet. Im Gegensatz zur z.B. der hierarchischen Clusteranalyse um ein partitionierendes Verfahren. Die Daten werden in k Cluster aufgeteilt – dabei muss die Anzahl der Cluster im vorhinein feststehen. Ziel ist es, dass die Quadratsumme der Abweichungen der Beobachtungen im Cluster zum Clusterzentrum minimiert wird.

Der Ablauf des Verfahrens ist wie folgt:

1. Zufällige Beobachtungen als Clusterzentrum
2. Zuordnung der Beobachtungen zum nächsten Clusterzentrum (Ähnlichkeit, z. B. über die euklidische Distanz)
3. Neuberechnung der Clusterzentren als Mittelwert der dem Cluster zugeordneten Beobachtungen

Dabei werden die Schritte 2. und 3. solange wiederholt, bis sich keine Änderung der Zuordnung mehr ergibt – oder eine maximale Anzahl an Iterationen erreicht wurde. Aufgrund von (1.) hängt das Ergebnis einer k-Means Clusteranalyse vom Zufall ab. Aus Gründen der Reproduzierbarkeit sollte daher der Zufallszahlengenerator gesetzt werden (mit `set.seed`). Außerdem bietet es sich an verschiedene Startkonfigurationen zu versuchen. In der Funktion `kmeans()` erfolgt dies durch die Option `nstart` =.

20.2 Beispiel für eine einfache Clusteranalyse

Nehmen wir uns noch einmal den Extraversionsdatensatz vor. Kann man die Personen clustern anhand von Ähnlichkeiten wie Facebook-Freunde, Partyfrequenz und Katerhäufigkeit? Probieren wir es aus!

```
extra <- read.csv("data/extra.csv")
```

Verschaffen Sie sich einen Überblick mit der Funktion `glimpse`.

20.2.1 Distanzmaße berechnen

Auf Basis der drei metrischen Merkmale (d. h. Alter, Einkommen und Kinder), die wir hier aufs Geratewohl auswählen, ergeben sich für die ersten sechs Beobachtungen folgende Abstände:

```
extra %>%
  dplyr::select(n_facebook_friends, n_hangover, extra_single_item) %>%
  head %>%
  dist(.)
#>      1      2      3      4      5
#> 2 144.01
#> 3 35.01 109.00
#> 4 51.93  95.19 21.24
#> 5 150.00   6.08 115.00 101.12
#> 6 126.00 270.00 161.00 176.56 276.00
```

Sie können erkennen, dass die Beobachtungen 1 und 3 den kleinsten Abstand haben, während 1 und 5 den größten haben.

Allerdings hängen die Abstände von der Skalierung der Variablen ab (`n_facebook_friends` streut stärker als `extra_single_item`). Daher sollten wir die Variablen vor der Analyse zu standardisieren (z. B. über `scale()`).

Mit der Funktion `daisy()` aus dem Paket `cluster` kann man sich auch den Abstand zwischen den Objekten ausgeben lassen. Die Funktion errechnet auch Abstandsmaße, wenn die Objekte aus Variablen mit unterschiedlichen Skalenniveaus bestehen. Allerdings mag `daisy` Variablen vom Typ `chr` nicht, daher sollten wir `sex` zuerst in eine Faktorvariable umwandeln.

```
extra %>%
  dplyr::select(n_facebook_friends, sex, extra_single_item) %>%
  mutate(sex = factor(sex)) %>%
  head %>%
  cluster::daisy(.)
```

20.2.2 kmeans für den Extraversionsdatensatz

Versuchen wir, einige Variablen mit `centers = 4` Clustern mithilfe einer `kmeans`-Clusteranalyse zu clustern.

```
set.seed(1896)
```

```

extra %>%
  mutate(Frau = sex == "Frau") %>%
  dplyr::select(n_facebook_friends, Frau, extra_single_item) %>%
  na.omit %>%
  scale -> extra_cluster

kmeans_extra_4 <- kmeans(extra_cluster, centers = 4, nstart = 10)

```

Lassen Sie sich das Objekt `extra_cluster` ausgeben und betrachten Sie die Ausgabe; auch `str(kmeans_extra_4)` ist interessant. Neben der Anzahl Beobachtungen pro Cluster (z. B. 337 in Cluster 2) werden auch die Clusterzentren ausgegeben. Diese können dann direkt verglichen werden. Schauen wir mal, in welchem Cluster die Anzahl der Facebookfreunde im Schnitt am kleinsten ist:

```

kmeans_extra_4$centers
#>   n_facebook_friends   Frau  extra_single_item
#> 1      -0.0237    0.712        1.3792
#> 2      -0.0445    0.712       -0.3909
#> 3     -0.0368   -1.402       -0.0587
#> 4     25.6707   -1.402        1.3792

```

Betrachten Sie auch die Mittelwerte der anderen Variablen, die in die Clusteranalyse eingegangen sind. Wie ‘gut’ ist diese Clusterlösung? Vielleicht wäre ja eine andere Anzahl von Clustern besser? Eine Antwort darauf liefert die Varianz (Streuung) innerhalb der Cluster: Sind die Summen der quadrierten Abweichungen vom Clusterzentrum gering, so ist die Varianz ‘innerhalb’ der Cluster gering; die Cluster sind homogen und die Clusterlösung ist ‘gut’ (vgl. Abbildung 20.8).

Je größer die Varianz innerhalb der Cluster, um schlechter ist die Clusterlösung.

In zwei Dimensionen kann man Cluster gut visualisieren (Abbildung 20.3); in drei Dimensionen wird es schon unübersichtlich. Mehr Dimensionen sind schwierig. Daher ist es oft sinnvoll, die Anzahl der Dimensionen durch Verfahren der Dimensionsreduktion zu verringern. Die Hauptkomponentenanalyse oder die Faktorenanalyse bieten sich dafür an.

Vergleichen wir ein paar verschiedene Lösungen, um zu sehen, welche Lösung am besten zu sein scheint.

```

kmeans_extra_2 <- kmeans(extra_cluster, centers = 2, nstart = 10)
kmeans_extra_3 <- kmeans(extra_cluster, centers = 3, nstart = 10)
kmeans_extra_5 <- kmeans(extra_cluster, centers = 5, nstart = 10)
kmeans_extra_6 <- kmeans(extra_cluster, centers = 6, nstart = 10)

```

Dann nehmen wir die Gesamtstreuung jeder Lösung und erstellen daraus erst eine Liste und

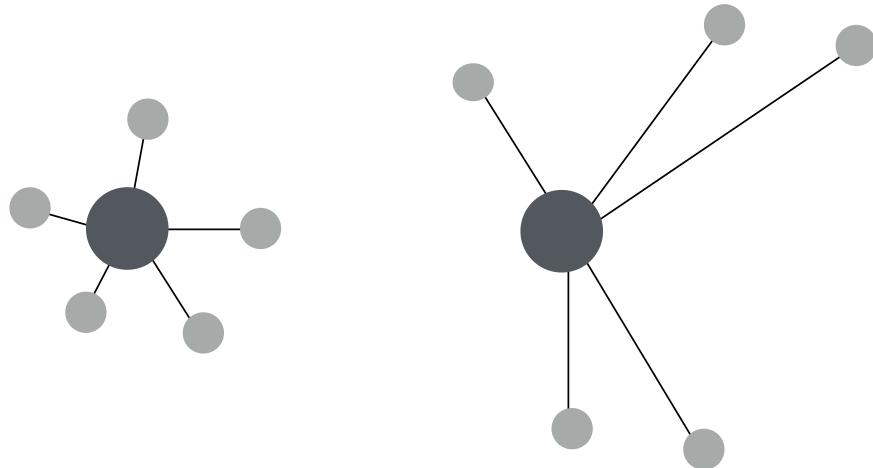


Abbildung 20.8: Schematische Darstellung zweier einfacher Clusterlösungen; links: geringe Varianz innerhalb der Cluster; rechts: hohe Varianz innerhalb der Cluster

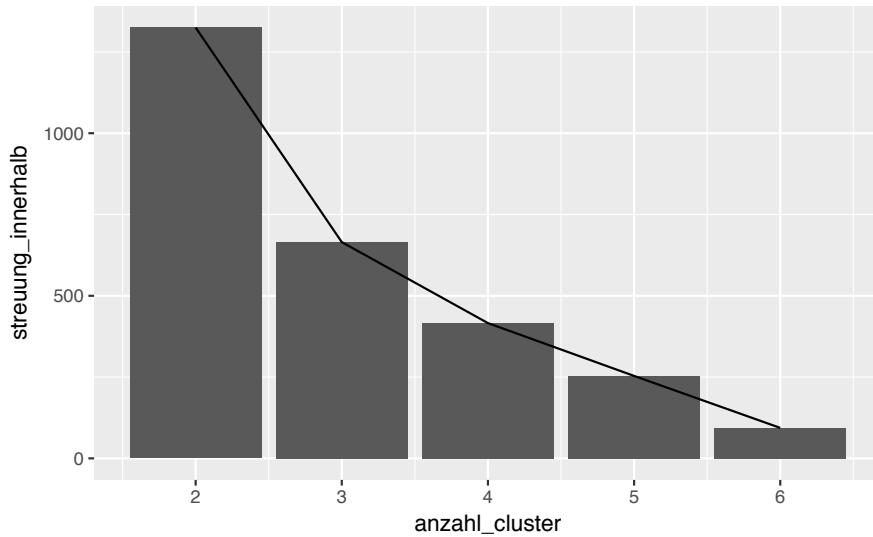
dann einen Dataframe:

```
streuung_innerhalb <- c(kmeans_extra_2$tot.withinss,
                        kmeans_extra_3$tot.withinss,
                        kmeans_extra_4$tot.withinss,
                        kmeans_extra_5$tot.withinss,
                        kmeans_extra_6$tot.withinss)

streuung_df <- data_frame(
  streuung_innerhalb,
  anzahl_cluster = 2:6
)
```

Jetzt plotten wir die Höhe der Streuung pro Clusteranalyse um einen Hinweis zu bekommen, welche Lösung am besten passen könnte.

```
ggplot(streuung_df) +
  aes(x = anzahl_cluster,
      y = streuung_innerhalb) +
  geom_col() +
  geom_line()
```



Nach der Lösung mit 4 Clustern kann man (vage) einen Knick ausmachen: Noch mehr Cluster verbessern die Streuung innerhalb der Cluster (und damit ihre Homogenität) nur noch unwesentlich oder zumindest deutlich weniger. Daher entscheiden wir uns für eine Lösung mit 4 Clustern.

20.3 Aufgaben¹



Richtig oder Falsch!?

1. Die Clusteranalyse wird gemeinhin dazu verwenden, Objekte nach Ähnlichkeit zu Gruppen zusammenzufassen.
2. Die Varianz innerhalb eines Clusters kann als Maß für die Anzahl der zu extrahierenden Cluster herangezogen werden.
3. Unter euklidischer Distanz versteht jedes Maß, welches den Abstand zwischen Punkten in der Ebene misst.
4. Bei der k-means-Clusteranalyse darf man die Anzahl der zu extrahierenden Clustern nicht vorab festlegen.
5. Cluster einer k-means-Clusteranalyse werden so bestimmt, dass die Cluster möglichst homogen sind, d.h. möglichst wenig Streuung aufweisen (m.a.W. möglichst nah am Cluster-Zentrum sind).

Laden Sie den Datensatz **extra** zur Extraversion.

1. Unter Berücksichtigung der 10 Extraversionsitems: Lassen sich die Teilnehmer der Umfrage in eine Gruppe oder in mehrere Gruppen einteilen? Wenn in mehrere Gruppen,

¹R, R, F, F, R

Tabelle 20.1: Befehle des Kapitels 'Clusteranalyse'

Paket::Funktion	Beschreibung
dist	Berechnet den euklidischen Abstand zwischen Vektoren
dplyr::glimpse	Stellt einen Dataframe im Überblick dar
cluster::daisy	Berechnet verschiedene Abstandsmaße
set.seed	Zufallsgenerator auf bestimmte Zahlen festlegen
cluster::clusplot	Visualisiert eine Clusteranalyse

wie viele Gruppen passen am besten?

2. Berücksichtigen Sie den Extraversionsmittelwert und einige andere Variablen aus dem Datensatz (aber nicht die Items). Welche Gruppen ergeben sich? Versuchen Sie die Gruppen zu interpretieren!
3. Suchen Sie sich zwei Variablen aus dem Datensatz und führen Sie auf dieser Basis eine Clusteranalyse durch. Visualisieren Sie das Ergebnis anhand eines Streudiagrammes!

20.4 Befehlsübersicht

Tabelle 20.1 fasst die R-Funktionen dieses Kapitels zusammen.

20.5 Verweise

- Diese Übung orientiert sich am Beispiel aus Kapitel 11.3 aus Chapman und Feit (2015) und steht unter der Lizenz Creative Commons Attribution-ShareAlike 3.0 Unported². Der Code steht unter der Apache Lizenz 2.0³
- Der erste Teil dieser Übung basiert auf diesem Skript: <https://cran.r-project.org/web/packages/broom/vignettes/kmeans.html>
- Eine weiterführende, aber gut verständliche Einführung findet sich bei James, Witten, Hastie, und Tibshirani (2013c).
- Die Intuition zum euklidischen Abstand mit Pythagoras' Satz kann hier im Detail nachgelesen werden: <https://betterexplained.com/articles/measure-any-distance-with-the-pythagorean-theorem/>.

²<http://creativecommons.org/licenses/by-sa/3.0>

³<http://www.apache.org/licenses/LICENSE-2.0>

Kapitel 21

Einführung in statisches Lernen mit caret

In diesem Kapitel werden folgende Pakete und Daten benötigt.

```
library(tidyverse)
library(caret)

stats_test <- read_csv("data/stats_test.csv")

unique(stats_test$bestanden)
#> [1] "ja"    "nein"
x <- factor(stats_test$bestanden, levels = c("nein", "ja"))
levels(x)
#> [1] "nein" "ja"
```

Das R-Paket **caret** (*Classification And Regression Training*) bietet eine komfortable Syntax, um eine Vielzahl – aktueller Stand: 232 – von statistischen Modellen zu berechnen; es gibt sogar ein Lehrbuch, welches eng auf diesem Paket aufbaut (Kuhn und Johnson 2013). Kein Mensch ist scharf drauf, sich die unterschiedlichen Syntax von so vielen Modellen (die in verschiedenen R-Paketen hausen) auswendig zu merken. Das ist der Vorteil von **caret**: Für alle Modelle wird die gleiche Syntax verwendet.

Schauen wir ein einfaches Beispiel einer Analyse mit **caret** an. Sagen wir, wir wollen vorhersagen – möglichst exakt – wer die Klausur besteht (`bestanden == 'ja'` im Datensatz `stats_test`).

21.1 Daten aufbereiten

Bevor wir loslegen, bereiten wir noch etwas den Datensatz auf. Viele Modelle kriegen Bauchschmerzen bei fehlenden Werten (NAs). In Wirklichkeit ist dieser Schritt des Datenaufbereitens häufig recht langwierig.

Bei `caret` werden zwei Arten von prädiktiven Modellen unterschieden: Vorhersage numerische Werte (*Regression*) und Vohersage von nominalen Werten (*Klassifikation*). Möchte man eine Regression durchführen, muss das Kriterium numerisch sein; möchte man eine Klassifikation durchführen, so muss das Kriterium ein Faktor sein. Definiert man eine Faktorvariable, so definiert der *erste* Level das “Nicht-Ereignis” (0) und der *zweite* Level das Ereignis (z.B. “bestanden” oder 1). Daher setzen wir im Folgenden die Reihenfolge der Faktorlevel entsprechend.

```
stats_test %>%
  na.omit %>%
  mutate(bestanden = factor(bestanden, levels = c("nein", "ja"))) %>%
  select(-c(date_time, score)) -> stats_test
```

Zum Umkodieren verwenden wir hier `ifelse`; der Befehl hat folgende Syntaxstruktur: `tue_wenn(Bedingung, Wert wenn ja, Wert ansonsten)`. Hier ist die zu prüfende Bedingung, ob `bestanden == "ja"`, falls das stimmt (für eine bestimmte Person), so soll `bestanden_bin` den Wert 1 erhalten, ansonsten den Wert 0.

21.2 Trainings- und Test-Sample aufteilen

Teilen wir zuerst die Daten in ein Trainings- und ein Test-Sample auf. Eine gängige Aufteilung ist 80:20. Die einfachste Möglichkeit dazu ist, die ersten 80% der Stichprobe dem Trainings-Sample zuzuweisen und den Rest zum Test-Sample:

```
n_train <- round(.8 * nrow(stats_test), digits = 0)

train <- stats_test %>% slice(1:n_train)
test <- stats_test %>% slice(n_train+1:nrow(stats_test))
```

Einfach. Aber schöner wäre natürlich, eine Zufallsstichprobe aus dem gesamten Datensatz zu ziehen; es könnten ja sein, dass die Reihenfolge der Fälle eine Rolle spielt. Zweitens würden wir uns freuen, wenn der Anteil der bestandenen Klausuren im Trainings-Sample gleich groß ist wie im Test-Sample. Praktischerweise stellt `caret` eine Funktion zur Verfügung, die beide Wünsche erfüllt:

```

Trainings_Faelle <- createDataPartition(stats_test$bestanden,
                                         p = .8)
Trainings_Faelle <- unlist(Trainings_Faelle)

train <- stats_test %>% dplyr::filter(row_number %in% Trainings_Faelle)
test <- stats_test %>% dplyr::filter(!(row_number %in% Trainings_Faelle))

train %>%
  select(-row_number) -> train

```

Die Funktion `createDataPartition` liefert ein Objekt vom Typ `list` (Liste) zurück, in dem ein Vektor enthalten ist. Dieser Vektor enthält die Zeilenummern von den 80% (`p = .8`) der Fälle, die in das Trainings-Sample gehören. Da `filter` keine Listen verarbeitet, nur Vektoren, müssen wir den Vektor aus der Liste erst herausschälen, “entlisten” sozusagen, mit `unlist`. Um die Fälle für den Testvektor zu bestimmen, negieren wir die Filteraussage (mit `!`). Abschließend definieren wir noch, ob und welche Kreuzvalidierung wir einsetzen möchten. Dazu gibt es den Befehl `traincontrol`, der das Trainieren kontrolliert.

```

my_crossval <- trainControl(method = "repeatedcv", # Kreuzvalidierung
                           number = 10, # k = 10
                           repeats = 10)

```

Hier sagen wir, dass wir eine $k = 10$ Kreuzvalidierung durchführen wollen. Außerdem soll diese $k = 10$ Kreuzvalidierung 10 Mal wiederholt werden (`repeats = 10`). Diese Wiederholung bringt weitere Stabilität in die Gütekennzahlen¹.

21.3 Das Modell berechnen

Als nächstes berechnen wir ein paar Modelle. Anstelle von “berechnen” spricht man gerne von “anpassen” (engl. to fit), weil man versucht, das Modell so gut es geht den Daten anzupassen (unter sonst gleichen Umständen). Egal welches Modell man anpassen möchte, die Syntax von `train` lautet dabei immer so:

```

glm_fit1 <- train(bestanden ~ ., data = train,
                     method = "glm",
                     trControl = my_crossval)
glm_fit1
#> Generalized Linear Model
#>

```

¹<http://appliedpredictivemodeling.com/blog/2014/11/27/vpuig01pqbk1mi72b8lcl3ij5hj2qm>

```
#> 155 samples
#>   3 predictor
#>   2 classes: 'nein', 'ja'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 140, 139, 139, 140, 139, 139, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.808      -0.00718
```

Je nach Art des zu berechnenden Modells und Datengröße, kann die Rechenzeit einige oder sogar viel Zeit in Anspruch nehmen und viel Speicher benötigen.

Kapitel 22

Grundlagen des Textmining



Lernziele:

- Sie kennen zentrale Ziele und Begriffe des Textminings.
- Sie wissen, was ein ‘tidy text dataframe’ ist.
- Sie können Worthäufigkeiten auszählen.
- Sie können Worthäufigkeiten anhand einer Wordcloud visualisieren.

In diesem Kapitel benötigte R-Pakete:

```
library(tidyverse) # Datenjudo
library(stringr) # Textverarbeitung
library(tidytext) # Textmining
library(lsa) # Stopwörter
library(SnowballC) # Wörter trunkieren
library(wordcloud) # Wordcloud anzeigen
library(skimr)
library(pradadata)
```

Ein großer Teil der zur Verfügung stehenden Daten liegt nicht als braves Zahlenmaterial vor, sondern in “unstrukturierter” Form, z.B. in Form von Texten. Im Gegensatz zur Analyse von numerischen Daten ist die Analyse von Texten weniger verbreitet bisher. In Anbetracht der Menge und der Informationsreichhaltigkeit von Text erscheint die Analyse von Text als vielversprechend.

In gewisser Weise ist das Textmining ein alternative zu klassischen qualitativen Verfahren der Sozialforschung. Geht es in der qualitativen Sozialforschung primär um das Verstehen eines Textes, so kann man für das Textmining ähnliche Ziele formulieren. Allerdings: Das Textmining ist wesentlich schwächer und beschränkter in der Tiefe des Verstehens. Der Computer ist einfach noch (?) wesentlich *dümmer* als ein Mensch, zumindest in dieser Hinsicht. Allerdings ist er auch wesentlich *schneller* als ein Mensch, was das Lesen betrifft.

Daher bietet sich das Textmining für das Lesen großer Textmengen an, in denen eine geringe Informationsdichte vermutet wird. Sozusagen maschinelles Sieben im großen Stil. Da fällt viel durch die Maschen, aber es werden Tonnen von Sand bewegt.

In der Regel wird das Textmining als *gemischte* Methode verwendet: sowohl qualitative als auch quantitative Aspekte spielen eine Rolle. Damit vermittelt das Textmining auf konstruktive Art und Weise zwischen den manchmal antagonistischen Schulen der qualitativ-idiographischen und der quantitativ-nomothetischen Sichtweise auf die Welt. Man könnte es auch als qualitative Forschung mit moderner Technik bezeichnen - mit den skizzierten Einschränkungen wohlgemerkt.

22.1 Zentrale Begriffe

Die computergestützte Analyse von Texten speiste (und speist) sich reichhaltig aus Quellen der Linguistik; entsprechende Fachtermini finden Verwendung:

- Ein *Corpus* bezeichnet die Menge der zu analysierenden Dokumente; das könnten z.B. alle Reden der Bundeskanzlerin Angela Merkel sein oder alle Tweets von “@realDonaldTrump”.
- Ein *Token* (*Term*) ist ein elementarer Baustein eines Texts, die kleinste Analyseeinheit, häufig ein Wort.
- Unter *tidy text* versteht man einen Dataframe, in dem pro Zeile nur *ein* Token (z.B. Wort) steht (Silge und Robinson 2016a). Synonym könnte man von einem “langen” Dataframe sprechen, so wie wir in Kapitel ?? kennen gelernt haben.

22.2 Grundlegende Analyse

22.2.1 Tidy Text Dataframes

Wozu ist es nützlich, einen Text-Dataframe in einen langen Dataframe umzuwandeln? Der Grund ist, dass immer wenn nur ein Wort (allgemeiner: Term) pro Zelle steht, dann können wir die Spalte einfach auszählen. Wir können z.B. `count` nutzen, um zu zählen, wie häufig ein Wort vorkommt. Sprich: Sobald wir einen langen (Text-)Dataframe haben, können wir unsere bekannte Methoden einsetzen.

Basteln wir uns einen *tidy text* Dataframe. Wir gehen dabei von einem Vektor mit mehreren Text-Elementen aus, das ist ein realistischer Startpunkt. Unser Text-Vektor¹ besteht aus 4 Elementen.

¹Nach dem Gedicht “Jahrgang 1899” von Erich Kästner

Breiter Dataframe

Zeile	text
1	Wir haben die Frauen zu Bett gebracht,
2	als die Männer in Frankreich standen.
...	...

Langer Dataframe

Zeile	Wort
1	Wir
1	haben
1	die
1	Frauen
1	zu
...	...

tidy text Dataframe

Abbildung 22.1: Illustration eines Tidy Text Dataframe

```
text <- c("Wir haben die Frauen zu Bett gebracht",
        "als die Männer in Frankreich standen.",
        "Wir hatten uns das viel schöner gedacht.",
        "Wir waren nur Konfirmanden.")
```

Als nächstes machen wir daraus einen Dataframe.

```
text_df <- data_frame(Zeile = 1:4,
                      text = text)
```

Zeile	text
1	Wir haben die Frauen zu Bett gebracht,
2	als die Männer in Frankreich standen.
3	Wir hatten uns das viel schöner gedacht.
4	Wir waren nur Konfirmanden.

Übrigens, falls Sie eine beliebige Textdatei einlesen möchten, können Sie das so tun:

```
text <- read_lines("data/Brecht.txt")
```

Der Befehl `read_lines` (aus `readr`²) liest Zeilen (Zeile für Zeile) aus einer Textdatei.

Dann “dehnen” wir den Dataframe zu einem *tidy text* Dataframe (s. Abb. 22.1); das besorgt die Funktion `unnest_tokens`. ‘unnest’ heißt dabei so viel wie ‘Entschachteln’, also von breit auf lang dehnen. Mit ‘tokens’ sind hier einfach die Wörter gemeint (es könnten aber auch andere Analyseeinheiten sein, Sätze zum Beispiel).

²Teil der Tidyverse-Familie

```
text_df %>%
  unnest_tokens(output = wort, input = text) -> tidytext_df

tidytext_df %>% head
#> # A tibble: 6 x 2
#>   Zeile     wort
#>   <int>    <chr>
#> 1      1     wir
#> 2      1     haben
#> 3      1     die
#> 4      1     frauen
#> 5      1     zu
#> 6      1    bett
```

Der Parameter `output` sagt, wie neue ‘saubere’ (lange) Spalte heißen soll; `input` sagt der Funktion, welche Spalte sie als ihr Futter (Input) betrachten soll (welche Spalte in tidy text umgewandelt werden soll).

In einem ‘tidy text Dataframe’ steht in jeder Zeile ein Wort (token) und die Häufigkeit des Worts im Dokument.

Überprüfen Sie, ob das stimmt: Betrachten Sie den Dataframe `tidytext_df`.

Das `unnest_tokens` kann übersetzt werden als “entschachtele” oder “dehne” die Tokens - so dass in *jeder Zeile* nur noch *ein Wort* (genauer: Token) steht. Die Syntax ist `unnest_tokens(Ausgabespalte, Eingabespalte)`. Nebenbei werden übrigens alle Buchstaben auf Kleinschreibung getrimmt.

Als nächstes filtern wir die Satzzeichen heraus, da die Wörter für die Analyse wichtiger (oder zumindest einfacher) sind.

```
tidytext_df %>%
  filter(str_detect(wort, "[a-z])) -> tidytext_df_lowercase
```

Das “[a-z]” steht für “alle Buchstaben von a-z”. In Pseudo-Code heißt dieser Abschnitt:



Nehme den Datensatz “text_df” UND DANN
dehne die einzelnen Elemente der Spalte “text”, so dass jedes Element seine eigene Spalte bekommt.

Ach ja: Diese “gedehnte” Spalte soll “Wort” heißen (weil nur einzelne Wörter drinnen stehen).

Ach ja 2: Dieses “dehnen” wandelt automatisch Groß- in Kleinbuchstaben um. UND DANN

filtere die Spalte “wort”, so dass nur noch Kleinbuchstaben übrig bleiben. FERTIG.

22.2.2 Text-Daten einlesen

Nun lesen wir Text-Daten ein; das können beliebige Daten sein³. Eine gewisse Reichhaltigkeit ist von Vorteil. Nehmen wir das Parteiprogramm der Partei AfD⁴. Vor dem Hintergrund des Erstarkens des Populismus weltweit und der großen Gefahr, die davon ausgeht - man blicke auf die Geschichte Europas in der ersten Hälfte des 20. Jahrhunderts - verdienterfordert der politische Prozess und speziell Neuentwicklungen darin unsere besondere Beachtung. Das Parteiprogramm ist im Pakete `pradadata` enthalten:

```
data(afd, package = "pradadata")
```

Häufig sind Textdateien in Textdateien gespeichert, dann kann man mit `read_lines` Daten, die keine Tabellenstruktur haben einlesen. Ein anderes häufiges Format sind PDF-Dateien; dafür bietet das Paket `pdfTools` die Funktion `pdf_text`, die den Inhalt einer PDF-Datei einliest. Jede Seite wird dabei als ein Element eines Vektors abgespeichert. Hätten wir die PDF-Datei vorliegen, so könnten wir sie mit `pdf_text` einlesen. Der resultierende Vektor `afd_raw` hat 96 Elemente (entsprechend der Seitenzahl des Dokuments). Wandeln wir als nächstes den Vektor in einen Dataframe um.

```
afd_pfad <- "data/afd_programm.pdf"

afd_raw <- pdf_text(afd_pfad)

afd <- data_frame(Zeile = 1:96,
                   afd_raw)
```

Im Ergebnis hätten wir einen Dataframe, genauso wie er in `pradadata` vorhanden ist. Mit `head(afd_raw)` können Sie sich den Beginn dieses Textvektor anzeigen lassen. Auch die Stopwörter entfernen wir wieder wie gehabt.

```
afd %>%
  unnest_tokens(output = token, input = content) %>%
  dplyr::filter(str_detect(token, "[a-z]")) -> afd_long
```

Insgesamt 96 Wörter⁵; eine substantielle Menge von Text. Was wohl die häufigsten Wörter sind?

³Ggf. benötigen Sie Administrator-Rechte, um Dateien auf Ihre Festplatte zu speichern.

⁴

⁵wie kann man sich die Anzahl der Wörter ausgeben lassen? Z.B. so: `count(afd)`

22.2.3 Worthäufigkeiten auszählen

```
afd_long %>%
  na.omit() %>% # fehlende Werte löschen
  count(token, sort = TRUE)
#> # A tibble: 7,087 x 2
#>   token     n
#>   <chr> <int>
#> 1 die    1151
#> 2 und    1147
#> 3 der     870
#> 4 zu      435
#> 5 für     392
#> 6 in      392
#> 7 den     271
#> 8 von     257
#> 9 ist     251
#> 10 das    225
#> # ... with 7,077 more rows
```

Die häufigsten Wörter sind inhaltsleere Partikel, Präpositionen, Artikel... Solche sogenannten “Stopwörter” sollten wir besser herausfischen, um zu den inhaltlich tragenden Wörtern zu kommen. Praktischerweise gibt es frei verfügbare Listen von Stopwörtern, z.B. im Paket `lsa`.

```
data(stopwords_de, package = "lsa")

stopwords_de <- data_frame(word = stopwords_de)

# Für das Joinen werden gleiche Spaltennamen benötigt
stopwords_de <- stopwords_de %>%
  rename(token = word)

afd_long %>%
  anti_join(stopwords_de) -> afd_no_stop
```

Unser Datensatz hat jetzt viel weniger Zeilen; wir haben also durch `anti_join` Zeilen gelöscht (herausgefiltert). Das ist die Funktion von `anti_join`: Die Zeilen, die in beiden Dataframes vorkommen, werden herausgefiltert. Es verbleiben also nicht “Nicht-Stopwörter” in unserem Dataframe. Damit wird es schon interessanter, welche Wörter häufig sind.

```
afd_no_stop %>%
  count(token, sort = TRUE) -> afd_count
```

Tabelle 22.1: Die häufigsten Wörter

token	n
deutschland	190
afd	171
programm	80
wollen	67
bürger	57
euro	55
dafür	53
eu	53
deutsche	47
deutschen	47

Tabelle 22.2: Die häufigsten Wörter - mit 'stemming'

token_stem	n
deutschland	219
afd	171
deutsch	119
polit	88
staat	85
programm	81
europa	80
woll	67
burg	66
soll	63

Ganz interessant; aber es gibt mehrere Varianten des Themas “deutsch”. Es ist wohl sinnvoller, diese auf den gemeinsamen Wortstamm zurückzuführen und diesen nur einmal zu zählen. Dieses Verfahren nennt man “stemming” oder “trunkieren”.

```
afdfinal %>%
  mutate(token_stem = wordStem(.x$token, language = "de")) %>%
  count(token_stem, sort = TRUE) -> afd_count

afdfinal %>%
  top_n(10) %>%
  knitr::kable(caption = "Die häufigsten Wörter - mit 'stemming'")
```

Das ist schon informativer. Dem Befehl `SnowballC::wordStem` füttert man einen Vektor an Wörtern ein und gibt die Sprache an (Default ist Englisch). Denken Sie daran, dass `.x` bei `dplyr` nur den Datensatz meint, wie er im letzten Schritt definiert war. Mit `.x$token` wählen wir also die Variable `token` aus `afdfinal` aus.



Abbildung 22.2: Eine Wordwolke zum AfD-Parteiprogramm

22.2.4 Visualisierung

Zum Abschluss noch eine Visualisierung mit einer “Wordcloud” dazu (Abbildung 22.2).

```
wordcloud(words = afd_count$token_stem,
          freq = afd_count$n,
          max.words = 100,
          scale = c(2,.5),
          colors=brewer.pal(6, "Dark2"))
```

Man kann die Anzahl der Wörter, Farben und einige weitere Formatierungen der Wortwolke beeinflussen⁶.

Weniger verspielt ist eine schlichte visualisierte Häufigkeitsauszählung dieser Art, z.B. mit Balkendiagrammen (gedreht), s. Abbildung 22.3.

```
afd_count %>%
  top_n(30) %>%
  ggplot() +
  aes(x = reorder(token_stem, n), y = n) +
  geom_col() +
  labs(title = "mit Trunkierung") +
  coord_flip() -> p1

afd_no_stop %>%
  count(token, sort = TRUE) %>%
  top_n(30) %>%
```

⁶<https://cran.r-project.org/web/packages/wordcloud/index.html>

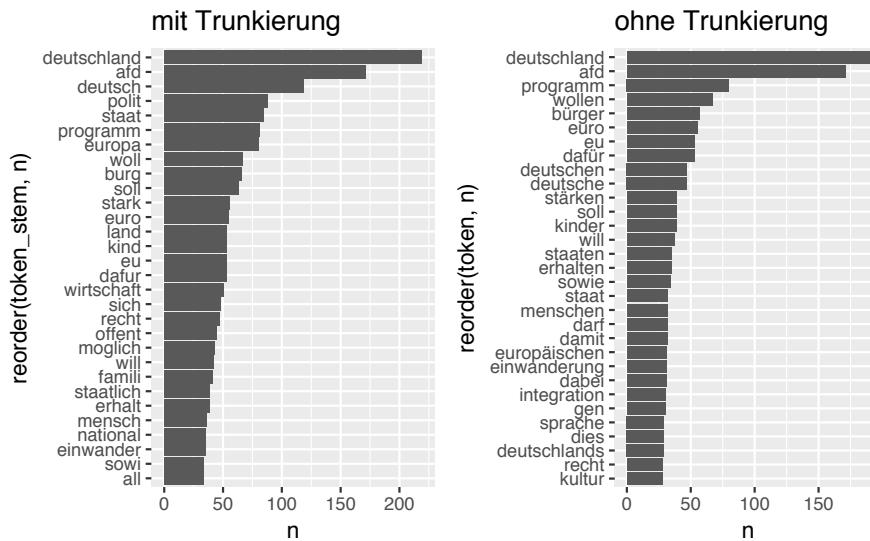


Abbildung 22.3: Worthäufigkeiten im AfD-Parteiprogramm

```
ggplot() +
  aes(x = reorder(token, n), y = n) +
  geom_col() +
  labs(title = "ohne Trunkierung") +
  coord_flip() -> p2
```

Die beiden Diagramme vergleichen die trunkierten Wörter mit den nicht trunkierten Wörtern. Mit `reorder` ordnen wir die Spalte `token` nach der Spalte `n`. `coord_flip` dreht die Abbildung um 90°, d.h. die Achsen sind vertauscht.

22.3 Aufgaben⁷



Richtig oder Falsch!?

1. Unter einem Token versteht man die größte Analyseeinheit in einem Text.
2. In einem tidytext Dataframe steht jedes Wort in einer (eigenen) Zeile.
3. Eine hinreichende Bedingung für einen tidytext Dataframe ist es, dass in jeder Zeile ein Wort steht (beziehen Sie sich auf den tidytext Dataframe wie in diesem Kapitel erörtert).
4. Gibt es ‘Stop-Wörter’ in einem Dataframe, dessen Text analysiert wird, so kommt es - per definitionem - zu einem Stop.
5. Mit dem Befehl `unnest_tokens` kann man einen tidytext Dataframe erstellen.

⁷F, R, F, F, R, R, F, F

6. Balkendiagramme sind sinnvolle und auch häufige Diagrammtypen, um die häufigsten Wörter (oder auch Tokens) in einem Corpus darzustellen.
7. In einem ‘tidy text Dataframe’ steht in jeder Zeile ein Wort (token) *aber nicht* die Häufigkeit des Worts im Dokument.
8. Unter ‘Stemming’ versteht man (bei der Textanalyse), die Etymologie eines Wort (Herkunft) zu erkunden.

22.4 Sentiment-Analyse

Eine weitere interessante Analyse ist, die “Stimmung” oder “Emotionen” (Sentiments) eines Textes auszulesen. Die Anführungszeichen deuten an, dass hier ein Maß an Verständnis suggeriert wird, welches nicht (unbedingt) von der Analyse eingehalten wird. Jedenfalls ist das Prinzip der Sentiment-Analyse im einfachsten Fall so:



Schau dir jeden Token aus dem Text an.
 Prüfe, ob sich das Wort im Lexikon der Sentiments wiederfindet.
 Wenn ja, dann addiere den Sentimentswert dieses Tokens zum bestehenden Sentiments-Wert.
 Wenn nein, dann gehe weiter zum nächsten Wort.
 Liefere zum Schluss die Summenwerte pro Sentiment zurück.

Es gibt Sentiment-Lexika, die lediglich einen Punkt für “positive Konnotation” bzw. “negative Konnotation” geben; andere Lexiko weisen differenzierte Gefühlskonnotationen auf. Wir nutzen hier das Sentimentlexikon von Remus, Quasthoff, und Heyer (2010).

Unser Sentiment-Lexikon sieht so aus:

neg_pos	word	value	inflections
neg	Abbau	-0.058	Abbaus,Abbaues,Abbauen,Abbaue
neg	Abbruch	-0.005	Abbruches,Abbrüche,Abbruchs,Abbrüchen
neg	Abdankung	-0.005	Abdankungen
neg	Abdämpfung	-0.005	Abdämpfungen
neg	Abfall	-0.005	Abfalles,Abfälle,Abfalls,Abfällen
neg	Abfuhr	-0.337	Abfuhen

22.4.1 Ungewichtete Sentiment-Analyse

Nun können wir jedes Token des Textes mit dem Sentiment-Lexikon abgleichen; dabei zählen wir die Treffer für positive bzw. negative Terme. Zuvor müssen wir aber noch die Daten (`afd_long`) mit dem Sentimentlexikon zusammenführen (joinen). Das geht nach bewährter

Manier mit `inner_join`; “inner” sorgt dabei dafür, dass nur Zeilen behalten werden, die in beiden Dataframes vorkommen.

```
afd_long %>%
  inner_join(sentiws, by = c("token" = "word")) %>%
  select(-inflections) -> afd_senti # die Spalte brauchen wir nicht

afd_senti %>%
  group_by(neg_pos) %>%
  summarise(polarity_sum = sum(value),
             polarity_count = n()) %>%
  mutate(polarity_prop = (polarity_count / sum(polarity_count))) %>% round(2)) -> afd_sen
```

neg_pos	polarity_sum	polarity_count	polarity_prop
neg	-52.6	219	0.27
pos	29.6	586	0.73

Die Analyse zeigt, dass die emotionale Bauart des Textes durchaus interessant ist: Es gibt viel mehr positiv getönte Wörter als negativ getönte. Allerdings sind die negativen Wörter offenbar deutlich stärker emotional aufgeladen, dennn die Summe an Emotionswert der negativen Wörter ist überraschenderweise deutlich größer als die der positiven.

Betrachten wir also die intensivsten negativ und positive konnotierten Wörter näher.

```
afd_senti %>%
  distinct(token, .keep_all = TRUE) %>%
  mutate(value_abs = abs(value)) %>%
  top_n(20, value_abs)
#> # A tibble: 20 x 5
#>   page      token neg_pos  value value_abs
#>   <int>     <chr>    <chr>  <dbl>     <dbl>
#> 1 3       ungerecht neg -0.784  0.784
#> 2 9       besonders pos  0.539  0.539
#> 3 10      gefährlich neg -0.637  0.637
#> 4 10      überflüssig neg -0.515  0.515
#> 5 10      behindern neg -0.775  0.775
#> 6 11      gefährden neg -0.501  0.501
#> 7 17      brechen neg -0.799  0.799
#> 8 18      unzureichend neg -0.523  0.523
#> 9 18      gemein neg -0.720  0.720
#> 10 19     verletzt neg -0.520  0.520
#> 11 20     zerstören neg -0.513  0.513
#> 12 32     trennen neg -0.503  0.503
#> 13 41     falsch neg -0.762  0.762
#> 14 43     vermeiden neg -0.526  0.526
```

```
#> 15    54    zerstört    neg -0.503    0.503
#> 16    61    schwach    neg -0.921    0.921
#> 17    74    belasten    neg -0.508    0.508
#> 18    83    schädlich   neg -0.927    0.927
#> 19    87    töten      neg -0.520    0.520
#> 20    88    verbieten   neg -0.629    0.629
```

Diese “Hitliste” wird zumeist (19/20) von negativ polarisierten Begriffen aufgefüllt, wobei “besonders” ein Intensivierwort ist, welches das Bezugswort verstärkt (“besonders gefährlich”).

Nun könnte man noch den erzielten “Netto-Sentimentswert” des Corpus ins Verhältnis setzen Sentimentswert des Lexikons: Wenn es insgesamt im Sentiment-Lexikon sehr negativ zuginge, wäre ein negativer Sentimentwert in einem beliebigen Corpus nicht überraschend. `skimr::skim()` gibt uns einen Überblick der üblichen deskriptiven Statistiken.

```
sentiws %>%
  select(value, neg_pos) %>%
  #group_by(neg_pos) %>%
  skim()
```

Insgesamt ist das Lexikon ziemlich ausgewogen; negative Werte sind leicht in der Überzahl im Lexikon. Unser Corpus hat eine ähnliche mittlere emotionale Konnotation wie das Lexikon:

```
afd_senti %>%
  summarise(senti_sum = mean(value) %>% round(2))
#> # A tibble: 1 x 1
#>   senti_sum
#>   <dbl>
#> 1 -0.03
```

22.5 Verweise

- Das Buch *Tidy Text Minig* (Julia und David 2017) ist eine hervorragende Quelle vertieftem Wissens zum Textmining mit R.

Teil IX

Rahmen 2

Kapitel 23

Programmieren mit R

Dieses Kapitel gibt eine Einführung in einige Aspekte des Programmierens, die für die Datenanalyse wichtig sind.



Lernziele:

- Das Iterationskonzept mit `purrr::map` in den Gründzügen verstehen

In diesem Kapitel werden folgende Pakete benötigt; `tidyverse` lädt auch `purrr`, welcher in diesem Kapitel eine zentrale Rolle spielt.

```
library(tidyverse)
library(magrittr)
```

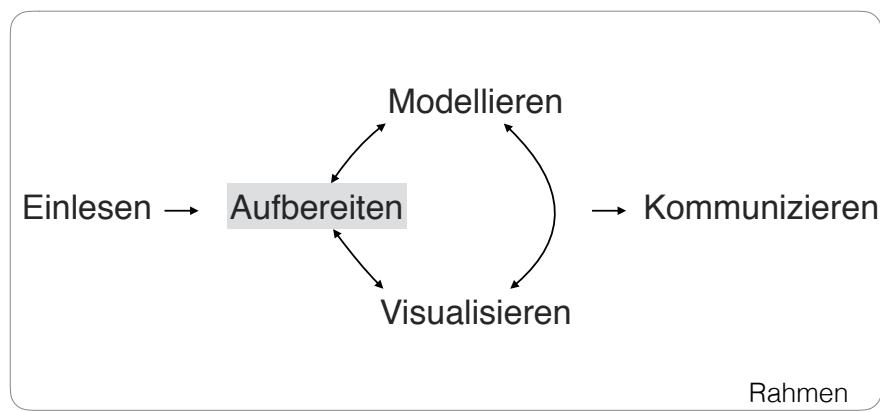


Abbildung 23.1: Daten aufbereiten

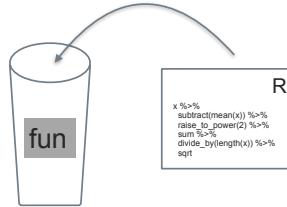


Abbildung 23.2: Funktionen definieren

23.1 Funktionen schreiben

Ähnlich wie Objekte (Variablen) Platzhalter für einen oder mehrere Werte sind, sind *Funktionen* Platzhalter für eine oder mehrere andere Funktionen. Funktionen werden aufgerufen, indem man ihren Namen und einem Paar von runden Klammern eingibt¹, etwa `fun()` oder `sd(temperatur)`. Das haben Sie schon oft getan. Eine Funktion kann, muss aber nicht, Parameter besitzen, mit der die Funktion “gefüttert” wird. Eine Funktion liefert ein Ergebnis, ein Objekt, zurück.

Nehmen wir zur Verdeutlichung an, Sie haben eine Reihe unglaublich schlauer R-Funktionen gefunden, die Sie in unglaublich schlauer Art kombinieren, um ein sagenhaftes Ergebnis zu bekommen. Nehmen wir an, Ihre Abfolge bestehe aus 5-6 Schritten. Jetzt wenden Sie natürlich diese tolle, fünfschrittige Abfolge immer und überall - oder zumindest immer mal wieder - an. Nach einiger Zeit meldet sich eine alte Krankheit, Sehnenscheidenentzündung, wieder - all diese Tipperei! Mit Excel wäre das nicht passiert... Um Sie vor Schlimmerem als Exeel einer Sehnenscheidenentzündung zu bewahren, gibt es Funktionen, die nichts anderes tun, als eine Reihe von Funktionen zu einen Bündel zu schnüren und mit einem Namen, z.B. `fun()` zu versehen, s. Abbildung 23.2.

```
knitr::include_graphics("images/Programmieren/Funs_def.png")
```

Leider wissen wir nicht, wie Ihre tolle Abfolge aussieht; begnügen wir uns mit dieser hier:

¹beim Durchpfeifen darf man auf die runden Klammern bei Funktionsnamen verzichten, was einen zwei Anschläge auf der Tastatur einspart, aber die Syntax verschwommener macht

```
x <- c(1, 2, 3)

x %>%
  subtract(mean(x)) %>%
  raise_to_power(2) %>%
  sum() %>%
  divide_by(length(x)) %>%
  sqrt()
#> [1] 0.816
```

Auf Deutsch heißt diese Syntax etwa:



Nimm den Vektor `x` UND DANN
 ziehe von jedem Element von `x` den Mittelwert von `x` ab UND DANN
 nimm jedes Element hoch 2 (quadriere es) UND DANN
 bilde die Summe der Elemente UND DANN
 teile durch die Anzahl der Elemente von `x` UND DANN
 ziehe von dieser Zahl die Wurzel.

In Lehrbüchern wird diese Abfolge eher so dargestellt:

$$\sqrt{\frac{1}{n} \sum (x - \bar{x})^2}$$

- was auf das Gleiche hinausläuft. Welche Darstellung finden Sie eingängiger? Vermutlich hängt die Antwort von Ihrer Mathe- und Informatik-Vorerfahrung ab. Persönlich glaube ich, dass die R-Syntax-Darstellung einfacher ist, da sie von links nach rechts zu lesen ist, damit nacheinander. Die Formel ist von innen nach außen zu lesen und muss damit gleichsam “auf einen Haps” verstanden werden.

Diese Abfolge wird auch als “Standardabweichung bezeichnet”; übrigens liefert `sd(x)` *nicht* den gleichen Wert, da dort durch $n - 1$ geteilt wird. Wir könnten unsere Abfolge also mit `sd2` bezeichnen (nicht sehr kreativ). Um diese Abfolge in die Form einer Funktion zu giesen, gilt diese Rechtschreibung:

```
sd2 <- function(eingabevektor){
  # hier unsere Rechenschritte
}
```

Wir erklären also `sd2`, dass es eine Funktion ist, mit dem einem Parameter (`eingabevektor`) und dem “Körper” (body), wie er zwischen den beiden geschweiften Klammern definiert ist. Konkret also in unserem Fall:

```
sd2 <- function(eingabevektor){
  eingabevektor %>%
    subtract(mean(eingabevektor)) %>%
    raise_to_power(2) %>%
    sum() %>%
    divide_by(length(eingabevektor)) %>%
    sqrt()
}
```

Natürlich darf der Parameter heißen, wie es uns beliebt; `x` wäre auch in Ordnung gewesen; nützlich ist, wenn die Parameter prägnante Namen haben. Schicken wir unsere Funktion zur Arbeit:

```
sd2(x)
#> [1] 0.816
```

Der Aufruf des Namens einer Funktion f führt eine Reihe von anderen Funktionen aus (nämlich die, die im Körper von f angeführt sind). Die von f aufgerufenen Funktionen können parametrisiert sein: so können die Werte der Argumente von f an die aufgerufenen Funktionen weitergegeben werden. Das was f erledigt, hängt also von ihren Parameter(werten) ab.

Die Sehnenscheidenentzündung (und Excel) konnte noch mal abgewendet werden. Interessanterweise gilt bei R: Alles was, passiert, ist eine Funktion. Auch Operatoren wie `+`, `<-`, `[` und `{` sind Funktionen! Hört sich unglaublich an; ist aber so. Um Funktionen mit “komischen” Namen wie `+` anzusprechen, schreibt man:

```
^+(1, 1)
#> [1] 2
```

Entsprechend kann man unsere Funktion `sd2()` synonym auch so schreiben:

```
sd2 <- function(eingabevektor){
  eingabevektor %>%
    ^-(mean(eingabevektor)) %>%
    ^^(2) %>%
    sum() %>%
    ^/(length(eingabevektor)) %>%
    sqrt() -> ausgabe
    return(ausgabe)

}
sd2_von_x <- sd2(x)
```

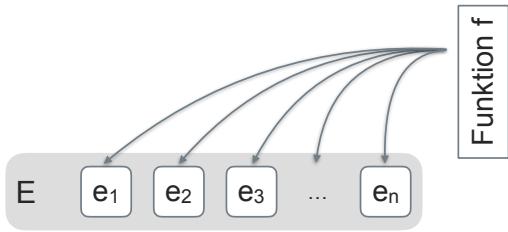


Abbildung 23.3: Die Funktion f wird auf jedes Element i von E angewendet

```
str(sd2_von_x)
#> num 0.816
```

Der letzte Wert, der berechnet wird, ist der, der von der Funktion zurückgegeben wird. Es ist guter Stil, explizit anzugeben (mit `return()`), welches Objekt zurückgeben wird.

Ein Wehrmutstropfen zum Schluss: Funktionen von `dplyr` lassen sich *nicht* so einfach in Funktionen packen. Generell gilt, dass Funktionen, die mit *nonstandard evaluation* (NSE) arbeiten, leider Gottes eine kompliziertere Behandlung verlangen. Dazu später mehr - s. Kapitel XXX.

23.2 Wiederholungen

Häufig möchte man oder muss man Dinge wiederholen; beim Analysieren von Daten ist es genauso. Allgemeiner gesprochen bedeutet “Wiederholen” für uns hier, dass wir Objekt mit mehreren Elementen vor uns haben und wir eine Aktion (Funktion) auf jedes Element anwenden möchten (s. Abbildung ??); man könnte auch sagen, wir ordnen (map) jedem e_i die Funktion f zu oder wir führen die Funktion f für jedes Element von E aus.

Typische Anwendungsfälle von solchen Wiederholungen sind:

1. für jedes Element einer Spalte soll eine Berechnung durchgeführt werden (z.B. von Fahrenheit in Celcius umrechnen; s. Abschnitt XXX)

2. für jede Spalte eines Dataframes soll eine Berechnung durchgeführt werden (z.B. der Mittelwert berechnet werden)
3. Jede Datei in einem Verzeichnis soll eingelesen werden

23.2.1 Wiederholungen für Elemente eines Vektors

Das erste Beispiel bezieht sich auf Wiederholungen für Elemente eines Vektors; das zweite für Elemente eines Dataframes; das dritte wiederum auf Elemente einer Liste. Betrachten wir die Beispiele nacheinander. In Abschnitt 4.3 haben wir festgestellt, dass R *vektoriell* rechnet; viele Funktionen wenden das, was sie tun sollen, auf jedes Element des Vektors an (Peng 2014):

```
x <- 1:3
y <- 4:6
x+y
#> [1] 5 7 9
```

In diesem Fall wurde jedes Element von `x` zu jedem korrespondierenden Element von `y` addiert:

x:	1	2	3
	+	+	+
y:	4	5	6

	5	7	9

Ohne diese Eigenschaft wären wir gezwungen, solche Ungetüme zu schreiben²:

```
z <- vector(mode = "numeric", length = length(x))
for(i in seq_along(x)) {
  z[i] <- x[i] + y[i]
}
z
#> [1] 5 7 9
```

Viele Funktionen für statistische Berechnungen sind “vektoriert”: `mean`, `sum`, `sd`, `sqrt`, so dass man meistens problemlos vorankommt. Weiter unten (Punkt 3) betrachten wir nicht-vektorierte Funktionen.

23.2.2 Wiederholungen für Spalten eines Dataframes

Sagen wir, Sie haben eine Umfrage zur Extraversion junger Erwachsener durchgeführt und möchten nun wissen, wie die deskriptiven Statistiken der Items 1 und 2 aussehen. Eine

²aber das festigt bestimmt den Charakter

komfortable Lösung bietet `dplyr`; laden wir zuerst die Daten der Umfrage `extra`:

```
data(extra, package = "pradadata")

extra %>%
  summarise_at(vars(i01:i02r), funs(mean, median, sd, IQR), na.rm = TRUE)
#> # A tibble: 1 x 8
#>   i01_mean i02r_mean i01_median i02r_median i01_sd i02r_sd i01_IQR
#>   <dbl>     <dbl>      <dbl>       <int>    <dbl>    <dbl>    <dbl>
#> 1     3.35      3.11        3          3  0.674   0.801      1
#> # ... with 1 more variables: i02r_IQR <dbl>
```

`summarise_at` führt eine Zusammenfassung für die mit dem Argument `vars` gewählten Spalten durch; dabei kommen die mit `funs()` gewählten Funktionen zum Zug. Dabei gilt die Einschränkung, dass `summarise` nur Funktionen ausführen kann, die als Ergebnis einen Skalar (eine einzelne Zahl) zurückliefern. Dadurch, dass man mit `funs()` mehrere Funktionen wählen kann, ist man oft ausreichend gerüstet. `summarise_at` führt also eine Wiederholung von `funs()` an (at) den gewünschten Spalten durch.

Möchte man eine Funktion wiederholt ausführen, die *mehr* als ein einzelnes atomares Element zurück liefert, muss man sich anders behelfen. Das Paket `purrr` bietet dazu eine Wiederholungsfunktion, `map(E, f)`, die eine Funktion `f` auf die Elemente von `E` anwendet (zuordnet, daher `map`):

```
extra %>%
  select(i01, i02r) %>%
  map(mosaic::favstats)
#> $i01
#>   min Q1 median Q3 max mean      sd    n missing
#>   1  3     3  4    4 3.35 0.674 820       6
#>
#> $i02r
#>   min Q1 median Q3 max mean      sd    n missing
#>   1  3     3  4    4 3.11 0.801 819       7
```

Warum haben wir nur einen Parameter (`favstats()` aus `mosaic`) angegeben? Durch die Pfeife impliziert, dass wir mit den Daten wie im vorherigen Schritt definiert arbeiten möchten. Genauer: Das Ergebnis der letzten Zeile wird als erster Parameter an `map()` übergeben.

Ein anderes Beispiel für `map()`:

```
extra %>%
  select(i01:i02r) %>%
```

```
map(mean, na.rm = TRUE)
#> $i01
#> [1] 3.35
#>
#> $i02r
#> [1] 3.11
```

Auf Deutsch hießt diese Syntax in etwa:

```
extra %>%
  select(i01:i10) %>%
  map(mean, na.rm = TRUE)
```

Praktischerweise “versteht” `mean` bzw. `map`, dass wir `mean` auf den Dataframe *so wie er aus der letzten Zeile bzw. aus dem letzten Schritt hervorgegangen* ist, anwenden wollen. Alternativ (synonym aus Sicht von `purrr`) hätten wir auch schreiben können:

```
extra %>%
  select(i01:i10) %>%
  map(~mean(., na.rm = TRUE))
```

Der Punkt `.` bezeichnet hier das Datenobjekt, wie es aus der letzten Zeile hervorging, so wie wir es bei `dplyr` kennen gelernt haben. Benutzt man allerdings diese Schreibweise (die sog. Formel-Schreibweise), so muss man die Tilde `~` zu Beginn des Funktionsnamen stellen; dann schreibt man die Funktion wie gehabt. Wie erwähnt, liefert `map` eine Liste zurück. Möchte man einen numerischen Vektor, so kann man `map_dbl()` verwenden:

```
extra %>%
  select(i01:i02r) %>%
  map_dbl(~mean(., na.rm = TRUE))
#> i01 i02r
#> 3.35 3.11
```

Gibt man bei `map()` kein Suffix (wie `map_dbl()`) an, so wird eine Liste zurückgeliefert. Es gibt Varianten von `map()` für z.B. `int`, `lg1`, `chr` (s. `?map`); auch Suffixe wie bei `dplyr` existieren in der Art `_if`, `_at`. Alternativ zu `map()` kann man die Funktion `apply()` und Freunde aus dem Standard-R verwenden; allerdings sind diese Funktionen nicht alle typstabil und haben weniger Komfort-Funktionen im Vergleich zu `map`.

23.2.3 Dateien wiederholt einlesen

Sagen wir, Sie haben von 100 “baugleiche” Datensätze (gleiche Spaltennamen und Datentypen) in einem Verzeichnis liegen; diese sollen eingelesen werden, zu einem Datensatz zusammengeführt werden und schließlich einer geheimen Analyse unterzogen werden.

Um dieses Szenario nachspielen zu können, erstellen wir uns ein paar Datensätze und schreiben diese in ein Verzeichnis auf der Festplatte.

```
1:3 %>%
  map(~filter(extra, row_number() == .)) %>%
  map(~select(., 1:3)) %>%
  imap(~write_csv(., path = paste0("dummy/df", .y, ".csv")))
#> [[1]]
#> # A tibble: 1 x 3
#>       timestamp   code   i01
#>       <chr>     <chr> <int>
#> 1 11.03.2015 19:17:48   HSC     3
#>
#> [[2]]
#> # A tibble: 1 x 3
#>       timestamp   code   i01
#>       <chr>     <chr> <int>
#> 1 11.03.2015 19:18:05   ERB     2
#>
#> [[3]]
#> # A tibble: 1 x 3
#>       timestamp   code   i01
#>       <chr>     <chr> <int>
#> 1 11.03.2015 19:18:09   ADP     3
```

Die Funktion `imap()` gleicht `map()`, nur dass sie über die von ihr erzeugte Variable `.y` den Index, also die Nummer jedes Elements das gemappt wird, verfügt. Damit erstellen wir den Dateinamen:

```
1:3 %>%
  map(~filter(extra, row_number() == .)) %>%
  map(~select(., 1:3)) %>%
  imap(~paste0("dummy/df", .y, ".csv"))
#> [[1]]
#> [1] "dummy/df1.csv"
#>
#> [[2]]
#> [1] "dummy/df2.csv"
```

```
#>
#> [[3]]
#> [1] "dummy/df3.csv"
```



Nimm den Vektor bestehend aus den (ganzen) Zahlen 1 bis 3 UND DANN
 Für jedes Element des Vektors aus der letzten Zeile: wähle Spalten 1:3 aus UND DANN
 Für jedes Element des Vektors aus der letzten Zeile: schreibe ihn (er ist eine Dataframe)
 als CSV-Datei.

Puh, geschafft. In der Praxis ist dieser Schritt nicht nötig, da Sie ja schon die 100 Datensätze in einem Ordner liegen haben. So, jetzt lesen wir diese 100 Datensätze ein. Um einer Sehnenscheiden-Entzündung vorzubeugen, entscheiden wir uns nach kurzem *gegen* dieses - zugegeben einfache - Vorgehen:

```
df1 <- read_csv("dummy/df1.csv")
df2 <- read_csv("dummy/df2.csv")
df3 <- read_csv("dummy/df3.csv")
```

Neben der Gefahr des Viel-Tippens ist dieses Vorgehens fehleranfällig und außerdem schlecht wiederverwendbar für ähnliche Situation. Besser geht es mit `map()`. Zuerst erstellen wir einen Vektor mit den Namen aller CSV-Dateien im betreffenden Ordner, das geht mit ‘`dir()`’:

```
df_filenames <- dir(path = "dummy", pattern = "*.csv")
str(df_filenames)
#> chr [1:3] "df1.csv" "df2.csv" "df3.csv"
```

Diesen Vektor übergeben wir an `map()`, der auf jedes Element die Funktion `read_csv()` anwenden soll. Liegen die Dateien im Arbeitsverzeichnis, so ist die Sache einfach:

```
df_filenames %>%
  map(read_csv)
```

Liegen die Datei in einem anderen Ordner, so muss man `read_csv()` sagen, wo zu suchen ist:

```
df_filenames %>%
  map_df(~read_csv(file = paste0("dummy/", .))) -> df
df
#> # A tibble: 3 x 3
#>       timestamp   code   i01
#>       <chr>     <chr> <int>
```

```
#> 1 11.03.2015 19:17:48 HSC 3
#> 2 11.03.2015 19:18:05 ERB 2
#> 3 11.03.2015 19:18:09 ADP 3
```

23.2.4 Fallbeispiele für `map`

23.2.4.1 Ein lineares Modell für jede Gruppe berechnen

Ein realistisches Exempel für `map()` ist dieses hier:

```
extra %>%
  select(i01, i02r, i03, n_facebook_friends, sex) %>%
  split(. $sex) %>%
  map(~lm(n_facebook_friends ~ i01 + i02r + i03, data = .)) -> extra_models
```

Wieder ist `.` nur eine Art Steno, das das Datenobjekt bezeichnet, so wie es im letzten Schritt berechnet wurde. `split` teilt den Datensatz in *zwei Listen* auf, die jeweils ein Datensatz sind - einer für Männer, einer für Frauen. Das ist übrigens ein Unterschied zu `group_by`, der zwar auch einen Datensatz in Gruppen aufteilt, aber eben nicht in mehrere Listen, sondern in einen speziellen gruppierten Dataframe, was etwas anderes ist. In der nächsten Zeile wird ein lineares Modell mit `lm` ausgeführt, wobei die Anzahl der Facebook-Freunde einer Person das Kriterium ist und die Items 1-3 die Prädiktoren. Da unsere Daten jetzt aus *zwei* Elementen (ein Datensatz für Frauen, einer für Männer) bestehen, wird diese Zeile zwei Mal ausgeführt bzw. für jedes Element einmal ausgeführt. Im nächsten Schritt wäre es interessant, die Ergebnisse des Models zu betrachten. Weist man einem Objekt das Ergebnis von `lm` zu, so kann man mit der Funktion `summary` sich Ergebnisse des Models anzeigen lassen (probieren Sie mal `str(extra_models)`). Führen wir `summary` im Rahmen von `map` aus, so wird mit jedem (der beiden) Elementen von `extra_models` der Befehl `summary` ausgeführt:

```
extra_models %>%
  map(summary) %>%
  map_dbl("r.squared")
#> Frau Mann
#> 0.0421 0.0115
```

Führt man `summary` aus, so wird eine Liste erstellt, die einige Zusammenfassungen für das Modell enthält. Im Rahmen von `map` wird diese “Zusammenfassungsliste” für jedes Element erstellt (also für das Element Frau und das Element Mann). Enthält eine Liste benannte Elemente, so kann man diese beim Namen ansprechen; aus jedem Element der Liste wird das Objekt mit dem entsprechenden Namen dann ausgelesen. Da unter `r.squared` nur ein (reelle)

Zahl gespeichert ist, lassen wir uns anstelle einer Liste eine einfachere Struktur, nämlich einen numerischen Vektor ausgeben.

23.2.4.2 Mehreren Spalten jeweils einen t-Test zuordnen

Sagen wir, wir seienwären brennend an der Frage interessiert, wer häufig fremd geht: Männer oder Frauen. Und diese Frage soll für uns nicht eher entschieden sein, als wir einen p-Wert dazu sehen. Außerdem wollen wir noch für dazugehörige Variablen den Unterschied bzw. den p-Wert wissen. Also los geht's; laden wir zuerst die Daten.

```
data(Affairs, package = "AER")
```

Werfen wir einen Blick in den Datensatz; wir sind dabei nur an numerischen Spalten interessiert, da wir diese einem Test zum Vergleich des Mittelwerts (hinsichtlich Frauen und Männer) unterziehen wollen:

```
Affairs %>%
  select_if(is.numeric) %>% head
#>   affairs age yearsmarried religiousness education occupation rating
#> 4      0    37       10.00          3        18        7      4
#> 5      0    27       4.00           4        14        6      4
#> 11     0    32       15.00          1        12        1      4
#> 16     0    57       15.00          5        18        6      5
#> 23     0    22       0.75           2        17        6      3
#> 29     0    32       1.50           2        17        5      5
```

Wir "mappen" jede dieser Spalten zu einer Funktion, hier einem t-Test, welcher die Spalte hinsichtlich der Geschlechter vergleicht:

```
Affairs %>%
  select_if(is.numeric) %>%
  map(~t.test(. ~ Affairs$gender)) %>%
  map_dbl("p.value")
#>   affairs            age  yearsmarried religiousness   education
#> 7.74e-01  2.85e-06  4.58e-01  8.51e-01  9.77e-24
#>   occupation        rating
#> 8.89e-35  8.53e-01
```

Am Anfang kann die Funktion `map` etwas obskur wirken. Daher schlüsseln wir Sie nochmal auf; der letzte Code-Abschnitt heißt auf Pseudo-Deutsch also:

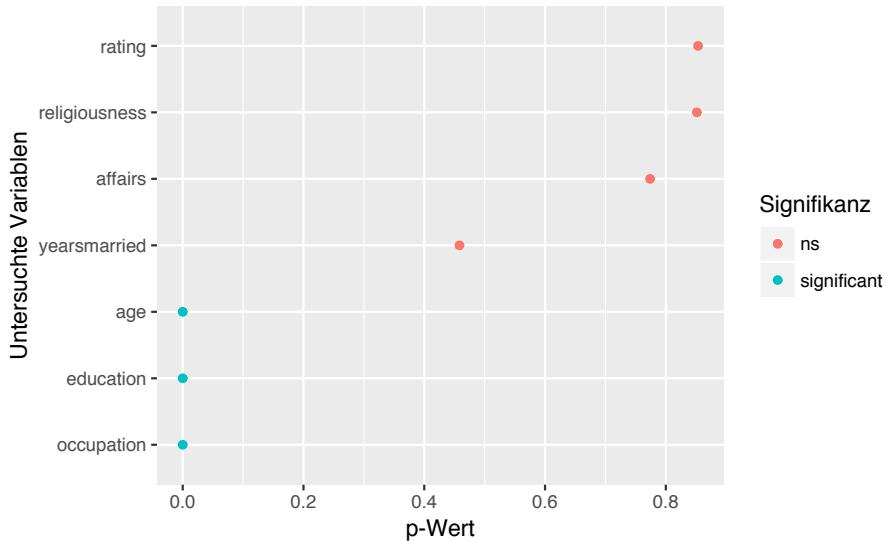


Abbildung 23.4: Multiple t-Tests und deren p-Werte



Nimm die Tabelle **Affairs** UND DANN
 wähle alle numerischen Spalten UND DANN
 wende auf alle Spalten einen t-Test an, wobei die Gruppierungsvariable **Affairs\$gender** ist UND DANN
 wende die Funktion “extrahiere das Element mit dem Namen `p.value`” an; das Ergebnis soll vom Typ ‘reelle Zahl’ (double) sein.

Ein “klassischer” R-Ansatz würde vielleicht so aussehen; wir lassen diese Syntax einfach mal andächtig so stehen.

```
lapply(Affairs[c("affairs", "age", "yearsmarried")], function(x) t.test(x ~ Affairs$gen
```

Schließlich können wir die Ergebnisse noch visualisieren, der Mensch ist halt eine Augentier:

```
Affairs %>%
  select_if(is.numeric) %>%
  map_df(~t.test(. ~ Affairs$gender)$p.value) %>%
  gather %>%
  mutate(signif = ifelse(value < .05, "significant", "ns")) %>%
  ggplot(aes(x = reorder(key, value), y = value)) +
  geom_point(aes(color = signif)) +
  coord_flip() +
  labs(x = "Untersuchte Variablen",
       y = "p-Wert",
       color = "Signifikanz")
```

Und die Daten sagen uns: Die mittlere Anzahl der Seitensprünge sowie die mittlere Ehezufriedenheit ist nicht statistisch signifikant unterschiedlich zwischen Männer und Frauen.

23.2.5 Einige Rechtschreibregeln für `map()`

Wiederholungs-Befehle wie `map()` sind anfangs vielleicht verwirrend, daher hier noch ein Überblick zur Gefühlswelt von `map()`:

1. Die allgemeine Syntax lautet `map(E, f)`: Auf jedes Element des Vektors E wird die Funktion f angewendet
2. Verwendet man `map()` in einer Pfeifensyntax, so braucht man E nicht angeben, dass wird von der Pfeife implizit getan, also `%>% map(f)`.
3. Man kann für f den Namen einer Funktion angeben, etwa: `%>% map(mean, na.rm = TRUE)`; etwaige Parameter werden mit Komma aufgelistet.
4. Man kann auch die Formel-Schreibweise verwenden: `%>% map(~mean(., na.rm = TRUE))`, dann schreiben man die zu wiederholenden Funktion in normaler Schreibweise.
5. Man kann `map()` auch den Namen eines Elements von E übergeben (was voraussetzt, dass die Elemente von E ein Element dieses Namens besitzen); in diesem Fall wird der Wert dieses Elements zurückgeliefert: `%>% map("p.value")`

Kapitel 24

Programmieren mit dplyr and friends

```
library(rlang)
library(pryr)
library(tidyverse)
```

24.1 Wie man mit dplyr nicht sprechen darf

Wie wir in Abschnitt ??writingfun{}) gesehen haben, sind Funktionen eine praktische Sache. Da wir ausgiebig von dplyr Gebrauch gemacht haben, liegt die Idee nahe, unser Wissen über das Schreiben von Funktionen auf dplyr anzuwenden. Probieren wir es aus; schreiben wir eine Funktion, welche von einer Spalte den Median und den IQR berechnet. Erstmal zur Erinnerung, *ohne* eine Funktion zu programmieren würden wir so vorgehen (nehmen wir den Datensatz extra als Beispiel):

```
data(extra, package = "pradadata")
```

```
extra %>%
  summarise(median(i01, na.rm = TRUE),
            IQR(i01, na.rm = TRUE))
#> # A tibble: 1 x 2
#>   `median(i01, na.rm = TRUE)` `IQR(i01, na.rm = TRUE)`
#>           <dbl>          <dbl>
#> 1             3              1
```

Das war einfach; aber schön wäre natürlich, wenn wir eine Funktion hätten, nennen wir sie `stats()`, die das gleiche Ergebnis liefern würde, die also folgenden Befehl erlaubt:

```
extra %>%
  stats(i01)
```

Man könnte meine, so sollte man diese Funktion `stats()` schreiben:

```
stats_ohno <- function(df, spalte){
  df %>%
    summarise(median(spalte, na.rm = TRUE),
              IQR(spalte, na.rm = TRUE))
}
```

Führt man diese Funktion aus, wird man enttäuscht:

```
stats_ohno(extra, i01)
#> Error in summarise_.impl(.data, dots): Evaluation error: object 'i01' not found.
```

Geht aber nicht.

Um den Grund zu verstehen, müssen wir Funktionen noch genauer verstehen. Funktionen bestehen aus drei Komponenten: a) dem “*Körper*” (body), das ist der Teil, der zwischen den geschweiften Klammern bei `stats_ohno()` steht; b) den *Argumenten* (formals), in unsererem Beispiel `df` und `spalte` sowie c) der *Umgebung* (environment), der Bereich, in dem die Variablen einer Funktionen stehen.

24.2 Wie man Funktionen mit dplyr-Verben schreibt

Um den Grund zu verstehen, müssen wir Funktionen noch genauer verstehen. Die Werte von Funktionsargumenten werden auch als *Promise* bezeichnet:

```
f <- function(x){
  is_promise(x)
}

f(x)
#> [1] TRUE
```

Die dplyr-Verben verkrafen aber dieses wundersame Promise-Objekt nicht; sie wollen ein “normales” Objekt, keine (leeren?) Versprechungen... Wir müssen den Promise also in ein normales Objekt umwandeln, um dplyr glücklich zu machen. Das kann man mit `dplyr::enquo()` machen; `enquo()` ersetzt (zitieren oder quoting, sagt man) den Promise durch den Ausdruck, den der Nutzer eingetippt hat, `i01` in unserem Fall:

```
f <- function(x){
  x_subst <- enquo(x)
  print(class(x_subst))
}

var <- c(1, 2, 3)

f(var)
#> [1] "quosure" "formula"
```

`class` zeigt uns den Typ des Objekts `x_subst` - was R als `quosure` (und `formula`) ausgibt. `print()` brauchen wir, weil innerhalb von Funktionen die “inneren” Funktionen ihrer Ergebnisse im Standard *nicht* am Bildschirm ausdrucken, das besorgt `print()`. Wir haben jetzt also ein Objekt vom Typ *quosure*; solche Objekte beinhalten die *Syntax einer Operation*, im einfachsten Fall den Aufruf eines Objekts wie `i01`. Ein Objekt, das Syntax speichert, nennt man auch einen *Ausdruck* (*expression*). Oder man sagt, mit dem Einfangen des Ausdrucks, habe man `i01 zitiert` (*to quote*). So ein Ausdruck-Objekt kann man sich vorstellen, wie einen Zauberspruch auf einem Pergament: Das geheime Wissen ist fein säuberlich niedergeschrieben, aber man muss es *aussprechen* oder *ausführen* (*evaluieren*), damit der Zauberspruch seine Wirkung entfaltet. Damit wir also das “richtige” Objekt `i01` bekommen, müssen wir das Name-Objekt ausführen; in diesem Fall heißt das, `i01` auszulesen. Dafür gibt es die Funktion `UQ()` (wie *unquote*). `UQ()` überführt einen Ausdruck in ein ausgeführtes Objekt; das Ausdruck-Objekt wird evaluiert.

```
f <- function(x){
  x_subst <- enquo(x)
  UQ(x_subst)
}

var <- c(1, 2, 3)

f(var)
#> <quosure: global>
#> ~var
```

Zitieren (quote) bedeutet hier, sagen, was man tut möchte. Evaluieren (unquote) bedeutet, es zu tun.

Soweit sieht es noch unscheinbar aus, aber damit haben wir dplyr glücklich gemacht: Wir können jetzt Funktionen mit dem den pldyr-Verben nach Herzenslust schreiben:

```
stats <- function(df, col){
  col_q = enquo(col)
```

```
df %>%
  summarise(median(UQ(col_q), na.rm = TRUE),
            IQR(UQ(col_q), na.rm = TRUE))
}

stats(extra, i01)
#> # A tibble: 1 x 2
#>   median(i01, na.rm = TRUE) `IQR(i01, na.rm = TRUE)` 
#>   <dbl>          <dbl>
#> 1      3            1
```

Kaum hat man einen Schritt gemeistert, erwachen schon neue Wünsche: Die Spaltennamen, die unsere Funktion von sich gibt, sind nicht sehr ansprechend. So kann die Spaltennamen verschönern:

```
stats <- function(df, col){
  col_q = enquo(col)

  df %>%
    summarise(median := median(UQ(col_q), na.rm = TRUE),
              IQR := IQR(UQ(col_q), na.rm = TRUE))
}

stats(extra, i01)
#> # A tibble: 1 x 2
#>   median  IQR
#>   <dbl> <dbl>
#> 1      3     1
```

Noch schöner, aber auch komplizierter geht es so:

```
stats <- function(df, col){
  col_q = enquo(col)
  col_name_md_q = paste0(quo_name(col_q), "_md")
  col_name_iqr_q = paste0(quo_name(col_q), "_iqr")

  df %>%
    summarise(UQ(col_name_md_q) := median(UQ(col_q), na.rm = TRUE),
              UQ(col_name_iqr_q) := IQR(UQ(col_q), na.rm = TRUE))
}

stats(extra, i01)
```

```
#> # A tibble: 1 x 2
#>   i01_md i01_iqr
#>   <dbl>    <dbl>
#> 1      3        1
```

Mit `quo_name` konvertiert man einen Ausdruck (quosure) in einen String. Die neuen Variablennamen `col_name_md_q` und `col_name_iqr_q` sind zuerst noch Strings; für `summarise` müssen es aber evaluierte Ausdrücke sein, daher noch schon ein `UQ()` hinterhergeschoben. Die Rechtschreibregeln von R verbieten es aber, dass auf der linken Seite des Gleichheitszeichen ein Ausdruck evaluiert wird; daher verwenden wir den `:=` Operator, der über `dplyr` importiert wird.

Ein wichtiger Test für unsere Funktion `stats()` steht noch aus: fügt sie sich klaglos in eine Abfolge anderer `dplyr`-Befehle, mit der Pfeife verbunden, wie üblich?

```
extra %>%
  select(i01:i03, sex) %>%
  group_by(sex) %>%
  stats(i01)
#> # A tibble: 3 x 3
#>   sex i01_md i01_iqr
#>   <chr>    <dbl>    <dbl>
#> 1 Frau      3     1.00
#> 2 Mann      3     1.00
#> 3 <NA>      3     0.75
```

Gut, das funktioniert.

24.3 Non-standard evaluation

Funktionen, die wie die `dplyr`-Verben ihre Argumente nicht einfach ausführen, sondern diese zunächst zitieren, arbeiten mit *non-standard evaluation* (NSE). Damit gewinnt man man einiges an Komfort beim interaktiven Arbeiten: man spart sich die Anführungszeichen ("i01") oder die Nennung der Umgebung eines Objekts (`extra$i01`). Doch der Komfort hat seinen Preis: Das Programmieren von Funktionen ist deutlich komplizierter.

Um NSE zu verstehen, muss man den Unterschied verstehen zwischen einer Aktion (z.B. Variable zuweisen) und seiner Beschreibung (“Variable zuweisen”; man beachte die Anführungszeichen, die das *Beschreiben* kennzeichnen sollen). Das Ausführen einer Aktion (das *Evaluieren*) ist etwas anderes wie Beschreiben (*Zitieren) einer Aktion. Betrachten Sie die folgende Syntax; sie wird einen Fehler erzeugen, wenn es die Variable `geburt` nicht gibt:

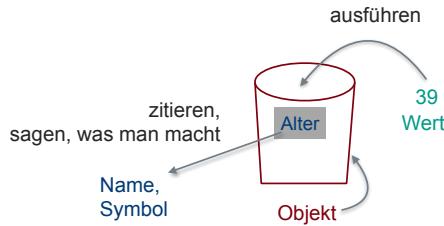


Abbildung 24.1: Der Unterschied zwischen Ausführen und Zitieren

```
alter <- geburt + 39
#> Error in eval(expr, envir, enclos): object 'geburt' not found
```

R bietet die Möglichkeit, die Syntax, also die Beschreibung dieser Aktion des Zuweisens in einem Objekt zu speichern; dafür kann man z.B. die Funktion `quo()` verwenden:

```
quo(alter <- geburt + 39)
#> <quosure: global>
#> ~(~(alter <- geburt + 39))
```

Dieser Befehl liefert *keinen* Fehler, wenn `geburt` nicht existiert; mit `quo()` zitieren die Syntax¹. Das zurückgegebene Objekt ist vom Typ `quosure` und lebt in der Arbeitsumgebung (daher `<quosure: global>`); Objekte dieses Typs sind technisch als Formel umgesetzt; daher die Tilde vor dem zitierten Ausdruck. Abbildung ?? versinnbildlicht den Unterschied zwischen dem Zitieren und dem Ausführen.

In R gibt es verschiedene Wege (Ideen und zugehörige Befehlsgruppen), um NSE zu verwenden. Die NSE-Befehle, die die Pakete des `tidyverse` verwenden, nennen sich *Tidyeval*. Aber auch z.B. im Standard R gibt es NSE.

¹der Unterschied zu `enquo()` ist, kurz gesagt, dass `enquo()` nur in Funktionen aufgerufen werden darf; `quo()` hingegen für den interaktiven Gebrauch bestimmt ist

24.4 Beispiele für NSE-Funktionen

24.4.1 Funktion, die einen Skalar zurückliefert

Eine andere Art von Funktion, die häufig praktisch ist, soll nicht einen Dataframe, sondern eine einzeln Zahl zurückliefern, damit sie mit `summarise()` kompatibel ist; denn `summarise()` kann als Ergebnis nur einzelne Zahlen zurückliefern. Erstellen wir einen Funktion `modus()`, die die häufigste Ausprägung eines Vektors zurückliefert:

```
modus <- function(df, col){
  col_q <- enquo(col)
  df %>%
  count (!!col_q, sort = TRUE) %>%
  slice(1) %>%
  pull(1)
}

extra %>%
  modus(i01)
#> [1] 3
```

Der Operator `!!` ist synonym zu `UQ()` und spricht sich “Bang Bang” (doch, wirklich). Übersetzen wir diese Funktion ins Deutsche:



Definiere `modus()` als Funktion mit den Parametern `df` und `col` Zitiere `col` und weise das Ergebnis `col_q` zu Nimm `df` UND DANN zähle die Ausprägungen von der evaluierten Variablen `col_q` und ... sortiere das Ergebnis UND DANN schneide die 1. Zeile heraus UND DANN ziehe die erste Spalte als Vektor heraus.

24.4.2 Funktion mit beliebig vielen Parametern

Manche Funktionen sind so flexibel, dass sie mit (praktisch) beliebig vielen Argumenten umgehen können. `group_by()` ist so ein Beispiel: Man kann die Funktion mit nur einem Argument bestücken, z.B. `group_by(sex)`, aber auch mit zwei oder mehr Argumenten, etwa `group_by(sex, smoker)`. Wie programmiert man eine Funktion, die diese Flexibilität berücksichtigt? Ein Weg sieht so aus:

```
stats_grouped <- function(df, col, ...){
  col_q <- enquo(col)
```

```

groups_q <- quos(...)

df %>%
  group_by(!!(groups_q)) %>%
  summarise(md = median(UQ(col_q), na.rm = TRUE),
            iqr = IQR(UQ(col_q), na.rm = TRUE))
}

extra %>%
  stats_grouped(i01)
#> # A tibble: 1 x 2
#>       md    iqr
#>     <dbl> <dbl>
#> 1      3     1

```

Eine andere Sache, die mir schon häufiger gefehlt hat, ist, dass `count` keine Anteile ausgibt, nur absolute Häufigkeiten. Schreiben wir eine Funktion, die neben den absoluten Häufigkeiten auch die relativen Häufigkeiten (Anteile) ausgibt:

```

count_prop <- function(df, ...){
  groups_q <- quos(...)

  df %>%
    count(!!(groups_q)) %>%
    mutate(prop = round(n / sum(n), 2))
}

extra %>%
  count_prop(sex, smoker)
#> # A tibble: 7 x 4
#>   sex                      smoker     n   prop
#>   <chr>                    <chr> <int> <dbl>
#> 1 Frau Gelegentlich (z. B. auf Partys)    1  0.00
#> 2 Frau                           Ja     1  0.00
#> 3 Frau                          Nein    8  0.01
#> 4 Frau                         <NA>  519  0.63
#> 5 Mann                           Ja     2  0.00
#> 6 Mann                         <NA>  284  0.34
#> 7 <NA>                         <NA>   11  0.01

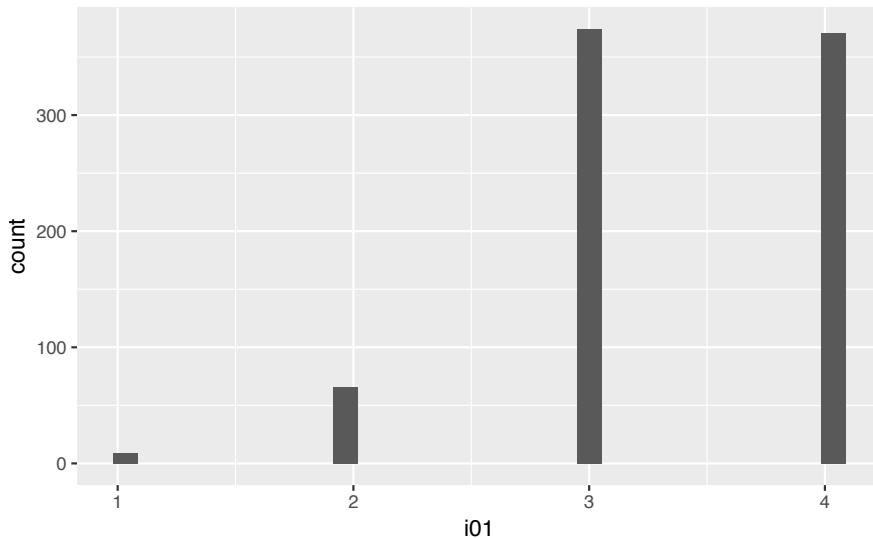
```

24.4.3 Funktionen für ggplot

Auch `ggplot2` nutzt NSE; möchte man eine Funktion mit `ggplot` schreiben, muss man die hier diskutierten Verfahren anwenden. Sagen wir, wir möchten eine Funktion schreiben, die ein Histogramm für eine numerische Variable `col` ausgibt. Das kann so aussehen:

```
gg_fun <- function(data, col)
{
  p <- ggplot(data,
               aes_q(x = UQ(enquo(col))))
  p + geom_histogram()
}

gg_fun(extra, i01)
```



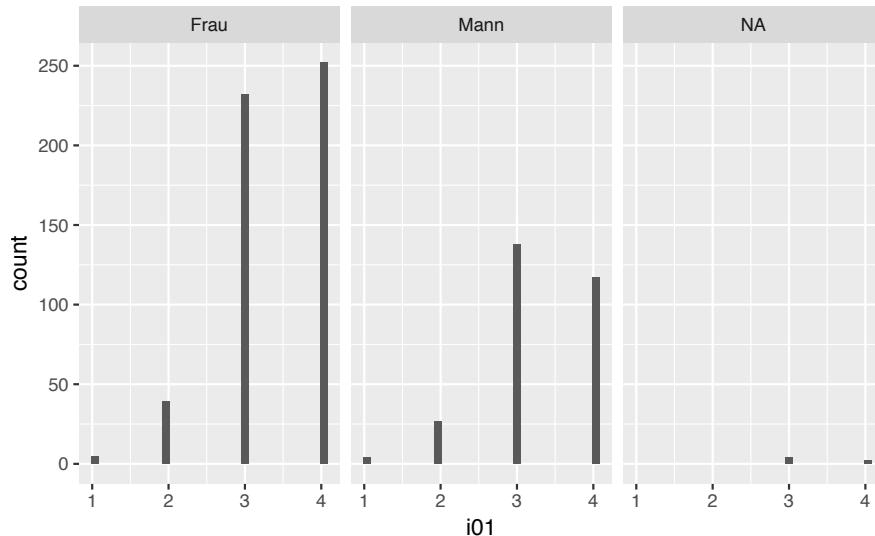
`aes_q` definiert das Mapping (Zuordnung von Spalten zu Achsen etc.) über eine Zitierung (quote, daher `_q`). Wiederum müssen wir die Variable, die der Nutzer (Sie!) tippte, zitieren und diesen Ausdruck dann evaluieren - genau wie wir es oben getan haben.

Möchten wir noch das Histogramm noch in Facetten aufbrechen entsprechend einer Gruppierungsvariablen `group`, kann das so geschehen:

```
gg_fun2 <- function(data, col, group)
{
  p <- ggplot(data,
               aes_q(x = UQ(enquo(col))) +
               facet_wrap(as.formula(UQ(enquo(group))))
```

```
p + geom_histogram()
}
```

```
gg_fun2(extra, i01, sex)
```



Bei `facet_wrap()` haben wir das gleiche Prinzip wie oben angewendet, mit dem einzigen Unterschied, das Objekt noch in eine Formel umzuwandeln, da `facet_wrap()` Objekte vom Formel-Typ versteht. Die Zeile mit `facet_wrap()` arbeitet also folgende Reihenfolge ab:

1. Schau nach, was der Nutzer getippt hat, was also der *Wert* des Arguments `group` ist (`sex`); zitiere dies; anders gesagt: merke dir die Syntax (das ist hier schlicht der Ausdruck `sex`), führe sie aber noch nicht aus.
2. Führe jetzt diesen Ausdruck aus; `sex` wird jetzt also erkannt als Spalte in dem Dataframe `extra`.
3. Das Objekt `sex`, ein Vektor vom Typ String, soll jetzt in ein Formel-Objekt umgewandelt werden, das erledigt `as.formula()`.

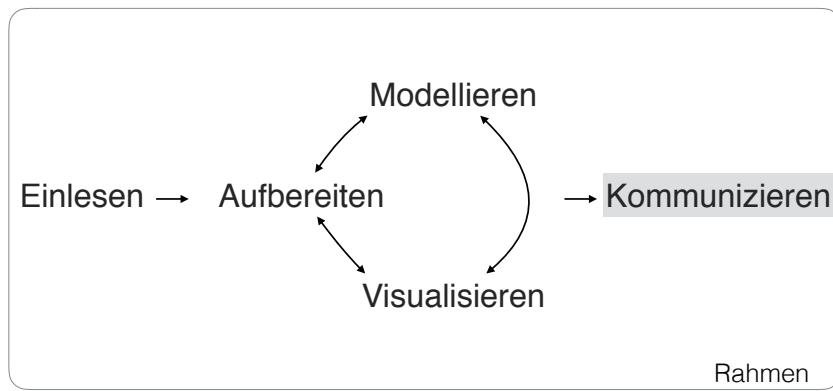
Anhang A

Was ist RMarkdown?



Lernziele:

- Die zentralen Ideen der Datenanalyse mit dplyr verstehen.
- Typische Probleme der Datenanalyse schildern können.
- Zentrale dplyr-Befehle anwenden können.
- dplyr-Befehle kombinieren können.
- Die Pfeife anwenden können.
- Werte umkodieren und “binnen” können.



In diesem Kapitel werden folgende Pakete benötigt:

```
library(knitr)
library(tidyverse)
```

Nehmen wir an, Sie möchten die Ergebnisse Ihrer Datenanalyse niederschreiben. Sei es, dass Sie einen Professor beglücken müssen wollen; sei es, dass Sie einem Kunden eine Bericht der Quartalszahlen inkl. Forecast erstellen oder mit einem Kollegen Ideen austauschen. Welche Anforderungen stellen Sie an ein Werkzeug, mit dem Sie die Ergebnisse niederschreiben?

Anhang B

Forderungen

Einige Anforderungen an Werkzeuge zur Berichterstellung sind im Folgenden aufgelistet:

1. R sollte integrierbar sein; sowohl die Ergebnisse als auch die Syntax.
2. Man sollte sich beim Schreiben auf das Schreiben konzentrieren können - ohne sich auf die Formatierung konzentrieren zu müssen.
3. Das Werkzeug sollte einfach zu erlernen (und zu bedienen) sein.
4. Es sollte verschiedene Ausgabeformate (PDF, HTML,...) unterstützen.
5. Das Dokument sollte optisch ansprechend formatiert sein.
6. Das Werkzeug sollte übergreifend (Betriebssysteme, Zeiten, Versionen) arbeiten.
7. Das Werkzeug sollte mächtig (funktionsreich) sein.
8. Das Werkzeug sollte frei (quelloffen sowie ggf. kostenfrei) sein.
9. Das Werkzeug sollte Kollaboration leicht machen.
10. Das Werkzeug sollte Versionierungen und Änderungsverfolgung unterstützen.

Mit Werkzeug ist hier die Software zur Erstellung des Berichts gemeint; häufig MS Word.

Betrachten wir die einzelnen Forderungen näher.

R sollte integrierbar sein; sowohl die Ergebnisse als auch die Syntax.

Sind das Werkzeug der Datenanalyse und das Werkzeug zur Berichterstellung voneinander getrennt, gibt es eine Schnittstselle, die wie eine Sollbuchstelle wirkt. Dieser Graben ist stets zu überwinden. Zwei Dinge, die dann fast zwangsläufig passieren: Copy-Paste und manuelles Aktualisieren. Copy-Paste ist mühsam und fehleranfällig. Mitunter weicht man vielleicht auf Neu-Eintippen aus: "Ok, der Mittelwert war doch 12,34..."; Fehler sind dann vorprogrammiert. Zum manuellen Aktualisieren: Stellen Sie sich vor, Ihr Bericht beinhaltet sagen wir 50 Diagramme und eine Reihe Tabellen und sonstige Zahlen - pro Kunde, pro Berichtszeitraum. Sie haben 100 Kunden, die einen Bericht pro Woche verlangen... Und jetzt für jedes Diagramm etc. in der Berichts-Software auf "Einfügen..." klicken??? Professionell geht anders. Kurz: Die Analyse sollte sich nahtlos in Ihren Bericht einfügen. Das ist vor allem für die Ergebnisse (Zahlen, Diagramme, Tabellen...) wichtig, aber sekundär auch für die Syntax.

Man sollte sich beim Schreiben auf das Schreiben konzentrieren können - ohne sich

Entscheidend sind die Inhalte, die Formatierung ist zweitrangig und außerdem zeitlich nachgelagert. Bietet das Werkzeug reichhaltige Möglichkeiten zur Formatierung besteht die Gefahr, dass verfrüht der geistige Fokus von den Inhalten zur Dekoration der Inhalte wandert. Besser ist es, sich in gestalterischer Askese zu üben und die mentalen Ressourcen komplett auf die Inhalte konzentrieren zu können. Außerdem ist es wünschenswert, wenn das Werkzeug sich selbstständig um das Formatieren kümmert. Damit soll dem Autor nicht die Möglichkeit verwehrt sein, selber zu gestalten; doch sollte ihm optisch ansprechende Standards automatisch angeboten werden.

Das Werkzeug sollte einfach zu erlernen (und zu bedienen) sein.

Nimmt mir das Werkzeug die Formatierung weitgehend ab, so bleibt (fast nur) das reine Schreiben von Text übrig. Das ist einfach zu bewerkstelligen (oder sollte es sein). Aufwändiges Einarbeiten in neue Werkzeuge sollte entfallen.

Es sollte verschiedene Ausgabeformate (PDF, HTML,...) unterstützen.

Die wichtigsten Formate, um Berichte zu erstellen sind sicherlich PDF, DOC, PPT und neuerdings zunehmend HTML und EPUB-Formate. Diese Formate sollten unterstützt werden. HTML und EPUB gewinnen mit der Verbreitung elektronischer Lesegeräte an Bedeutung. So ist z.B. HTML an einem Tablett oder Handy besser zu lesen als eine PDF-Datei. Ein wesentlicher Grund ist, dass bei HTML Zeilenumbrüche flexibel sind, z.B. je nach Größe des Displays. Bei PDF-Dateien ist dies fix.

Das Dokument sollte optisch ansprechend formatiert sein.

Das Werkzeug sollte - idealerweise als Standard ohne Eingriffe des Nutzers - optisch ansprechende Dokumente erzeugen. Dazu gehört vor allem schöner Schriftsatz: keine hässlichen Löcher zwischen Wörtern oder zwischen Buchstaben, elegante Schriftsätze oder schöne Formeln. Für HTML-Dateien sind "coole" Designs - z.B. mit benutzerfreundlicher Navigationsspalte - sinnvoll.

Das Werkzeug sollte übergreifend (Betriebssysteme, Zeiten, Versionen) arbeiten.

Ärgerlich ist, wenn das Werkzeug nur für bestimmte Betriebssysteme (oder Versionen davon) funktioniert. Nicht so schlimm, aber auch nervig ist, wenn die Varianten eines Werkzeugs zwar übergreifend funktionieren, aber nicht *ganz* gleich sind - wenn z.B. Farben sich ändern oder Textfelder verrutschen. Auch in ein paar Jahren sollten die Dateien noch lesbar sein.

Das Werkzeug sollte mächtig (funktionsreich) sein.

Die Quadratur des Kreises: Ein Werkzeug sollte einfach zu bedienen sein, aber mächtig, also einen großen Funktionsumfang besitzen. Beide Ziele sind nicht leicht unter einen Hut zu bringen und vielleicht das Grundproblem von Benutzerfreundlichkeit von Computersystemen. Eine pragmatische Lösung ist es (oder sollte sein), dem Nutzer vernünftige Standardwerte anzubieten bzw. diese ungefragt anzuwenden aber gleichzeitig dem Nutzer die Möglichkeit geben, in viele Details einzutreten. Letzteres ist häufig kompliziert(er), aber für viele Nutzer nicht nötig, wenn die Standards gut sind.

Das Werkzeug sollte frei (quelloffen sowie ggf. kostenfrei) sein.

Als "Normalnutzer" denken Sie vielleicht: "Ist mir doch egal, ob das Werkzeug von Firma XYZ angeboten wird, kauf ich mir halt!. Das ist in einigen Fällen stimmig. Aber: Die Erfahrung zeigt, dass einige quelloffenen Software tendenziell schneller weiterentwickelt wird (oder Fehler beseitigt werden). Fortgeschrittene Nutzer können so einfach ihre Ideen zur Verfügung stellen. Das geht viel schneller (in der Regel) als wenn es bei einem Konzern durch die Instanzen sickern muss. R ist bestes Beispiel dafür. Außerdem: Manche Werkzeuge der Datenanalyse sind *sehr* teuer; da ist es schön, wenn man (bessere) Leistung kostenfrei bekommt! Nicht jeder ist mit ausreichend Finanzmitteln gesegnet..." Offen" beinhaltet in dem Zusammenhang idealerweise auch, dass die Quelldateien menschenlesbar sind, also reine Text-Dateien. Damit kann jeder, auch ohne ein bestimmtes Werkzeug zu nutzen, die Quelldatei lesen. Reine Textdateien haben wohl von allen Datei-Formaten die beste Aussicht, in 10, 20, 50 oder 100 Jahren noch lesbar zu sein.

Das Werkzeug sollte Kollaboration leicht machen.

Mit Kollaboration ist gemeint, Sie schreiben einen Bericht sagen wir mit 5 Kollegen. Eine beliebte Variante der Zusammenarbeit ist dabei, dass Sie Ihren Entwurf an die Kollegen senden mit der Bitte um Hinweise, Korrekturen oder Überarbeitung. Nettetweise bekommen Sie 6 Versionen zurück (ein Kollege schrieb 3 Mails, sie wusste dann selber nicht mehr warum, dafür meldete sich ein Kollege gar nicht). Ihnen bleibt die glorreiche Aufgabe, diese 6+1 Dokumente zusammenzuführen. Natürlich widersprechen sich die Hinweise der Kollegen, die in Unkenntnis der Hinweise der anderen entstanden... Außerdem nutzten nicht alle die selbe Methode für ihre Hinweise. Kurz: Nach einigen Schreikrämpfen müssen Sie erstmal Spazieren gehen. Schöner wäre ein zentrales Dokument, an dem alle arbeiten. So kann jeder den aktuellen Stand sehen und idealerweise auch die Änderungen der Kollegen (oder die eigenen Änderungen) einsehen. Einfache Werkzeuge dafür sind z.B. Google Docs oder der Dropbox, welche Versionierungsfunktionen bietet. Besser (aber komplexer in der Bedienung) sind spezielle Versionierungs-Werkzeuge. Das bekannteste heißt "Git" (<https://de.wikipedia.org/wiki/Git>).

Das Werkzeug sollte Versionierungen und Änderungsverfolgung unterstützen.

Ähnlich zur Kollaboration ist die Versionierung. Hier geht es darum, die Entwicklungsgeschichte eines Dokuments nachvollziehen zu können. Das ist umso wichtiger, je größer das Dokument bzw. das Projekt zu dem Dokument ist. Eine studentische Abschlussarbeit ist ein gutes Beispiel, aber viele Berichte in Unternehmen taugen auch als Beispiel. Haben Sie schon mal Dateinamen gesehen wie "Bericht_V23_Korrektur_Edit_Willi_Final_2017-02-23_v3_final_Ergänzung.pdf"? Ich gebe zu, das Beispiel ist bewusst auf die Spitze getrieben, aber es schält die Problematik gut heraus. Der Änderungsmodus von MS-Word und ähnlichen Werkzeugen ist hilfreich für eine begrenzte Anzahl von Änderungen. Wurde aber eine bestimmte Passage mehrfach geändert, so kommt dieses Funktionalität schnell an ihre Grenzen. Kurz: Bei wichtigeren oder komplexeren Dokumenten ist eine professionelle Lösung sinnvoll.

RMarkdown erfüllt diese Forderungen. Nicht perfekt, aber besser wohl als jedes andere Werkzeug. Daher ist es ein sehr wertvolles Werkzeug, das Sie kennen sollten, wenn Sie Berichte mit Ergebnissen von Datenanalysen schreiben.

Anhang C

Bevor es losgeht

RMarkdown wird als Teil von RStudio mitgeliefert, kostenlos und großteils quelloffen. Es ist bereits installiert, wenn Sie RStudio installiert haben.

Der Name *RMarkdown* suggeriert, dass R hier eine Rolle spielt. Das ist richtig. Doch was heißt “Markdown”? Der Name röhrt von der Idee von Auszeichnungssprachen (engl. “markup languages”) her. Auszeichnungssprachen sind, einfach gesagt, ein paar Befehle, die man in einen Text hineinschreibt, um den Text zu formatieren. Der Text ist also eine Mischung von Inhalt und Formatierungszeichen. HTML oder *Tex* sind bekannte Auszeichnungssprachen. Beide sind aber recht komplex, bzw. der Text sieht aus wie Kraut und Rüben vor lauter Auszeichnungsbefehlen. Markdown will anders sein, einfacher. Daher keine “Markup-”, sondern eine “Markdown-”Sprache. Vor “Sprache” braucht man sich hier nicht beunruhigen zu lassen. Die “Sprache” Markdown ist in 10 Minuten gelernt! Wie Sie sicher schon geahnt haben, hat das R in RMarkdown die Bedeutung, dass R in das Markdown “eingestrickt” wird. Damit kann man dan einfach Dokumente erzeugen, in denen Daten analysiert werden und zusätzlich Text vorkommt (nennen wir es ein *R-Text-Dokument*).

RMarkdown-Dateien haben sinnvollerweise die Endung `.Rmd`. Rmd-Dateien sind, genau wie normale Markdown-Dateien, schnöde Text-Dateien und können entsprechend von jedem Text-Editor (auch mit RStudio, das hier mit Syntax-Highlighting verwöhnt) geöffnet werden.

Erstellt man eine Rmd-Datei, so wird das Verzeichnis, in dem diese Datei liegt, als Arbeitsverzeichnis betrachtet. Möchte man auf eine Datei verweisen, die in einem Unterverzeichnis liegt (z.B. ein Bild laden), so ist es meist am einfachsten, einen relativen Pfad vom Arbeitsverzeichnis anzugeben: `Bilder/mein_bild.png`.

Doch wie sehen die Arbeitsschritte mit RMarkdown aus? Wo fange ich an, und was “kommt hinten bei raus”? Kurz gesagt, sind zwei Konvertierungen, bzw. zwei technische Schritte beteiligt (s. Abb. C.1).

Man beginnt mit einem R-Text-Dokument im MD-Format (bzw. als RMD bezeichnet). Im ersten Schritt werden die R-Befehle ausgeführt und deren Ergebnis in das Dokument geschrieben. Das resultierende Dokument ist wiederum im MD-Format gehalten. Dieser Arbeitsschritt wird vom R-Paket ‘knitr’ (Xie 2015) geleistet. Im zweiten Schritt wird das

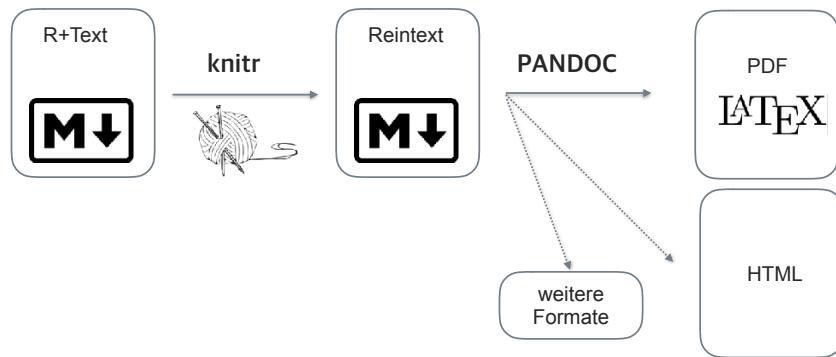


Abbildung C.1: Die zwei Arbeitsschritte bei der Konvertierung von RMarkdown-Dokumenten

MD-Dokument in einen anderen Dokumententyp konvertiert - PDF und HTML sind die häufigsten, aber auch ePub, MS-Word oder Folien sind möglich. Der Fantasie sind dabei kaum Grenzen gesetzt: Man kann Vorlagen einbinden, so dass das resultierende Dokument auf eine gewünschte Art formatiert ist – z.B. so wie die amerikanische Fachgesellschaft für Psychologie (APA) das gerne hätte (vgl. (@ Aust und Barth 2017)). Das alles ist *open*: frei wie in Bier und frei wie in Freiheit. Es kostet nichts, nichts wird geheim oder versteckt gehalten, man darf es nach belieben abändern! Kein Wunder, dass diese Ideen um sich greifen wie die Hand-Fuß-Mund-Krankheit in der Kinderkrippe.

Anhang D

RMarkdown in Action

Werden wir praktisch! Ein Beispiel zur Arbeit mit Markdown.

- Öffnen Sie RStudio.
- Klicken Sie das Icon für “neue Datei” und wählen Sie “R Markdown...”, um eine neue RMarkdown-Datei zu erstellen.
- Dann öffnet sich eine einfache Rmd-Datei, die nicht ganz leer ist, sondern aus didaktischen Gründen ein paar einfache, aber typische, Rmd-Befehle enthält. Schauen Sie sich den Inhalt kurz an; Sie sehen eine Mischung aus “Prosa” und R.
- Jetzt “verstricken” wir R und Prosa zu einer Datei, HTML in dem Fall, welche sowohl Text, R-Ausgaben als auch R-Syntax enthält. Dafür klicken wir auf das Stricknadel-Icon:
- Ach ja, RStudio bittet Sie noch, die Rmd-Datei zu speichern. Tun Sie RStudio den Gefallen.

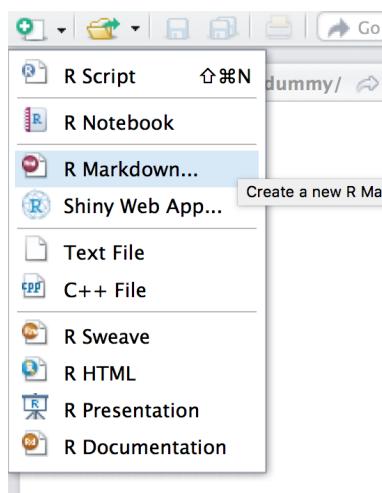
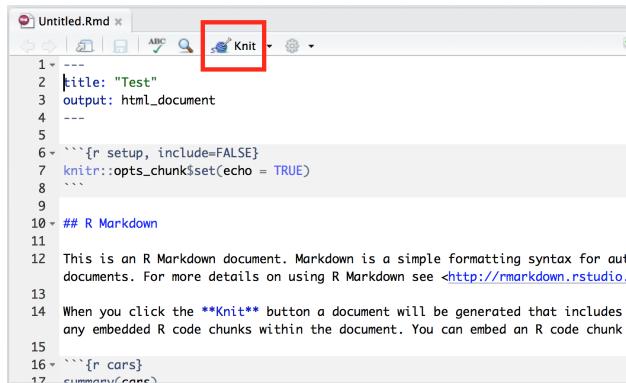


Abbildung D.1: Neue Rmd-Datei erstellen

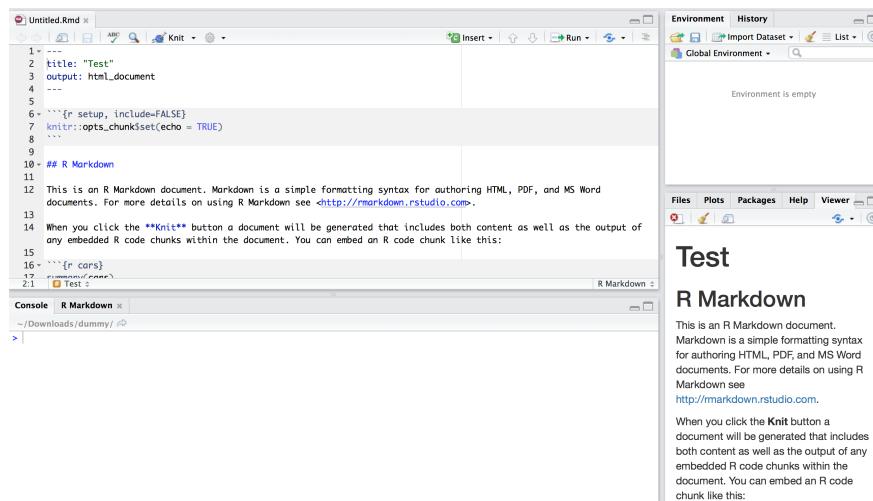


```

1 <!--
2 #title: "Test"
3 #output: html_document
4 ---
5
6 ````{r setup, include=FALSE}
7 knitr::opts_chunk$set(echo = TRUE)
8 ```
9
10 ## R Markdown
11
12 This is an R Markdown document. Markdown is a simple formatting syntax for authoring documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
13
14 When you click the **Knit** button a document will be generated that includes all
any embedded R code chunks within the document. You can embed an R code chunk
15
16 ````{r cars}
17 summary(cars)

```

Abbildung D.2: Wir stricken



The screenshot shows the RStudio interface with the following components:

- Left Panel (Code Editor):** An Rmd file titled "Untitled.Rmd" containing R Markdown code. The "Knit" button in the toolbar above the editor is highlighted.
- Right Panel (Environment):** Shows the global environment, which is currently empty.
- Bottom Panel (Viewer):** Displays the generated HTML output under the heading "Test". The content includes the R Markdown code and a note about the "Knit" button.

Abbildung D.3: HTML-Datei aus Rmd-Datei erstellt

- Als Ergebnis müsste sich im Reiter “Viewer” das Ergebnis zeigen. Sie haben eine, vielleicht Ihre erste HTML-Datei aus einer Rmd-Datei erstellt. Verweilen Sie in Andacht. Sie können sich das Ergebnis, die HTML-Datei im Browser anschauen, in dem Sie betreffende HTML-Datei im Browser öffnen (diese liegt dort, wo auch die zugehörige Rmd-Datei liegt.)

Anhang E

Die Arbeitsschritte mit RMarkdown

Die Arbeitsschritte mit RMarkdown von den ersten Gedanken bis zum fertigen Bericht kann man so zusammenfassen.

Zuerst bietet es sich an - ganz ohne Gedanken an wohl geformte Sprache und roten Faden - seine Gedanken zu Papier (oder in eine Textdatei) zu kriegen (“Ideen sammeln”). Für viele Menschen sind hier Schmier- und Notizzettel oder Schemata, Mindmaps oder andere Visualisierungen hilfreich. Eine Gliederung zu erstellen gehört auch in diesen Schritt. Hat man schließlich die Struktur erarbeitet, so kann den Text - jetzt mit wohlgeformter(er) Sprache und roten Faden - zusammensetzen (“Text schreiben”). Dabei ist es oft so, dass man die Beiträge der Coautoren ~~ertragen muss~~ integrieren möchte. Dies ist mit am besten mit einer zusätzlichen Software mit dem Namen “Git” bzw. “Github” zu erreichen, die ebenfalls quelloffen, kostenlos (und weit verbreitet bei Tekkies) ist (“Coautoren integrieren”). Diese Software erlaubt nicht nur Versionierung lokal an einem eigenen Computer, sondern auch in Zusammenarbeit mit Coautoren (“Änderungen nachverfolgen”). Parallel zum Schreiben des Textes sind die Daten zu analysieren; eine Aufgabe, die prinzipiell unabhängig vom Schreiben des Textes ist (“Daten analysieren”). Aber die Ergebnisse der Analyse sind in den Text zu integrieren. Um Schnittstellen zu vermeiden, ist es sinnvoll, den Inhalt-Text sowie die R-Syntax in einer Datei zu “verstricken”. Dieser Schritt (“R ausführen”) wird von einem R-Paket namens “knitr” (Xie 2015) besorgt (engl. “to knit” - stricken). Knitr führt die R-Syntax im Dokument aus und liefert das Ergebnis im Markdown-Format zurück. Aus R+Markdown wird reines Markdown. Die Markdown-Datei kann nun in fast jedes denkbare andere Markup-Format übersetzt werden, die wichtigsten sind HTML und TeX. Die Übersetzung führt RStudio wiederum, genau wie das “Knittern” für uns komfortabel im Hintergrund aus. Dazu wird auf ein Programm namens “pandoc” (<http://pandoc.org>)

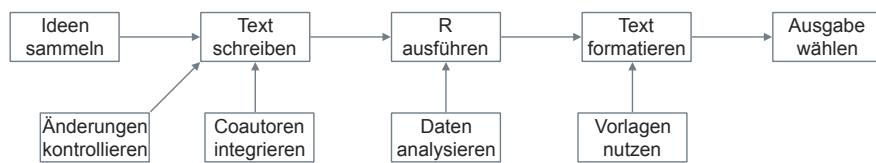


Abbildung E.1: Die Arbeitsschritte mit RMarkdown

zurückgegriffen. Pandoc übersetzt eine Markup-Sprache in eine andere. Haben wir z.B. in TeX übersetzt, so können wir - vorausgesetzt TeX, LaTeX o.ä. ist installiert - von TeX in PDF umwandeln lassen. Ist TeX auf Ihrem Rechner installiert, so wird dieser Übersetzung wiederum automatisch vorgenommen.

Anhang F

Aufbau einer Markdown-Datei

Das wichtigste ist: Eine Markdown-Datei (.md) ist eine reine Text-Datei, genau wie eine Rmd-Datei (die ein Spezialfall einer Markdown-Datei ist). Mit jedem Texteditor können Sie sie öffnen und ändern.

```
---
```

```
title: "Untitled"
author: "Sebastian Sauer"
date: "23 2 2017"
output: html_document
```

```
---
```

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
````
```

```
## R Markdown
```

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML

When you click the **Knit** button a document will be generated that includes both conte

```
```{r cars}
summary(cars)
```
```

Typischerweise besteht eine Markdown-Datei aus drei Teilen:

1. Einem (optionalen) **YAML-Header** abgegrenzt durch ---s.
2. **R-Chunks** (R-Syntax-Abschnitte) abgegrenzt durch ```.
3. Normalem Reintext mit einigen Steuerzeichen wie ***kursiv*** oder **#Überschrift_Ebene_1**.

Der YAML-Header definiert einige Meta-Daten für Ihre Datei; Dinge wie Autor, Titel, Datum

oder Ausgabeformat werden hier festgelegt. Sie müssen keine YAML-Header angeben, dann werden einige Standards verwendet. YAML steht übrigens, falls Sie sich das gerade fragen, für “yet another markup language”, also eine Auszeichnungssprache, die sehr einfach ist und nur für Metadaten zuständig ist.

R-Syntax wird (innerhalb von Rmd-Dateien) nur innerhalb der “Chunks” als R verstanden; außerhalb der eingezäunten R-Abschnitte werden Sie als normaler Text verstanden. Sie können Chunks auch nochmal ausführen, in dem Sie z.B. den Button “Run” klicken. Um das ganze Dokument zu “übersetzen”, reicht ein Klick auf das Stricksymbol.

Der Text bei Markdown sieht im Prinzip so aus, wie man eine Plain-Text-Email früher (oder manchmal heute noch) geschrieben hätte. Auch ohne dass man Markdown kennt, kann man ihn ohne Probleme erfassen.

Anhang G

Syntax-Grundlagen von Markdown

In RStudio wird Pandocs Markdown verwendet; die Übersetzung von Markdown in eine andere Auszeichnungssprache wird komplett von Pandoc abgewickelt.

Man braucht etwas Übung, um sich die Syntax zu merken, aber im Grunde ist es ganz einfach. Schauen Sie selbst:

Text formatieren mit Markdown

```
*kursiv* oder so _kursiv_
**fett** __fett__
`R-Syntax`
hochgestellt^2 und tiefgestellt~2~
```

Überschriften

```
# 1. Ebene
```

```
## 2. Ebene
```

```
### 3. Ebene
```

Aufzählungen

- * Listenpunkt 1
- * Listenpunkt 2
 - * Listenpunkt 2a

* Listenpunkt 2b

1. Element 1 einer nummerierten Liste

1. Element 2. Die korrekte Nummer wird automatisch erstellt.

Links und Bilder

<http://Beispiel.com>

[Bezeichnung des Links] (http://Beispiel.com)

! [Optionale Bildbezeichnung] (path/to/img.png)

Tabellen

| Spaltenkopf1 | Spaltenkopf1 |
|--------------|--------------|
| Zelleninhalt | Zelleninhalt |
| Zelleninhalt | Zelleninhalt |

Wenn man mal etwas vergisst, kann man in RStudio hier nachschauen: *Help > Markdown Quick Reference*.

Anhang H

Tabellen

Praktisch ist, dass man sich Dataframes einfach als Tabellen ausgeben lassen kann. Normalerweise werden in RMarkdown Dataframes in gewohnter Manier ausgegeben:

```
mtcars %>%
  slice(1:3)
#> # A tibble: 3 x 11
#>   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1  21.0     6   160   110  3.90  2.62  16.5     0     1     4     4
#> 2  21.0     6   160   110  3.90  2.88  17.0     0     1     4     4
#> 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
```

`slice` filtert die angegebenen Zeilen; hier 1 bis 3.

Möchte man eine “richtige” Tabelle, so kann man z.B. mit dem Befehl `knitr::kable` arbeiten. Die Tabelle unten (H.1) wurde so erstellt:

```
mtcars %>%
  slice(1:3) %>%
  kable(caption = "Eine Tabelle mit Kable")
```

Tabelle H.1: Eine Tabelle mit Kable

| mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|------|-----|------|-----|------|------|------|----|----|------|------|
| 21.0 | 6 | 160 | 110 | 3.90 | 2.62 | 16.5 | 0 | 1 | 4 | 4 |
| 21.0 | 6 | 160 | 110 | 3.90 | 2.88 | 17.0 | 0 | 1 | 4 | 4 |
| 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.6 | 1 | 1 | 4 | 1 |

Anhang I

Zitieren

Zitate sind bei Rmarkdown, genau wie Bilder und Verweise, nichts anderes als Links, mit einer kleinen Änderungen:

Das ist furchtbar wichtig [@Weis0is2017]. @Feistersack2017 widerspricht dem vehement.

Inerhalb von den eckigen Klammern, die einen Linktext kennzeichnen, wird ein Klammeraffe geschriebne (@); dieser ist die Bezeichnung für eine Zitation. Dann folgt die ID der Zitation; diese findet in der Datei, in der die eigentlichen Zitationen stehen (z.B. ‘bibliography.bib’) bei der jeweiligen Zitation. Lässt man die eckigen Klammern weg, so wird eine In-Text-Zitation ausgegeben. Fügt man ein Minuszeichen vor den Klammeraffen, so wird nur die Jahreszahl ausgegeben (wenn der Zitationsstil sonst den Namen hinzufügen würde).

Mehrere Zitationen werden mit Strichpunkt getrennt: So steht es geschreiben [@Weis2017 ;

Man kann beliebige Kommentare in die Zitation aufnehmen: So war es schon immer [vgl. @Weis2017]

Will man eine Zitation ohne Klammer einfügen, so lässt man die eckigen Klammern weg: @Feistersack2017

Den Namen des Autors/ der Autoren kann man unterdrücken, in dem man ein `^-^` vor die Zitation setzt.

Eine Reihe von bibliographischen Formaten werden unterstützt; darunter BibLaTEX, BibTeX, endnote und medline. Die bibliographischen Einträge alle Zitationen fügt man in eine Textdatei (z.B. mit Namen `bibliography.bib`); im YAML-Header gibt man dann den Dateinamen mit den bibliographischen Informationen an. Anhand der Endung ordnet RMarkdown (d.h. Pandoc) dann zu, um welche Art von Format (z.B. Bibtex) es sich bei der Literaturdatei handelt.

`bibliography: bibliography.bib`
`csl: apa.csl`

Den Zitationsstil kann man, ebenfalls im YAML-Header mit der Variable `cs1` definieren. Dort verweist man auf eine CSL-Stil-Datei; diese Dateien definieren Zitationsstile. Es handelt sich um Text-Dateien mit einer einigermaßen intuitiven Syntax; man könnte daran also rumbasteln, wenn man denn wollte. CSL-Dateien sind quelloffen und man kann sie sich z.B. hier herunterladen: <https://github.com/citation-style-language/styles>. Böse Zungen sagen, dass es mehr Zitationsstile gäbe als wissenschaftliche Zeitschrift, und von letzteren gibt es Tausende.



Geben Sie für die Literatur- und die CSL-Datei keinen Pfad an, so geht R davon aus, dass sich die Dateien im Verzeichnis der jeweiligen Rmd-Datei liegen.

Anhang J

Kollaboration und Versionierung

Kollaboration und Versionierung sind wichtige Aspekte von professionellem Projektmanagement. Das wichtigste Werkzeug hier heißt “Git”. Git ist eine Software, die Versionierungsfunktionen anbietet - auch über mehrere Coautoren hinweg. Auch diese Software ist frei verfügbar. “Github” ist ein Anbieter, der bekannteste, der “Projektordner” online anbietet, so dass sich man komfortabel synchronisieren kann. Alternativ gibt es Plattformen, die ähnliche Dienste anbieten wie <http://authorea.com>. Allerdings ist einiges an Einarbeitung vornötig; wer sich die Mühe macht, wird reich belohnt¹. Wir werden hier aus Platzgründen nicht weiter auf solche Werkzeuge eingehen.

¹Sagen sie alle: <https://xkcd.com/1319/>

Anhang K

Verweise

RMarkdown ist ein junges Ökosystem, das schnell wächst.

- Ein guter Startpunkt, mehr über Markdown zu lernen, ist das Buch von Wickham und Grolemund (2016).
- Größere Texte können mit `bookdown` geschrieben werden, eine Erweiterung zu Markdown (dieses Buch wurde so geschrieben): <https://bookdown.org/yihui/bookdown/>
- Die Cheatsheets von RStudio sind hilfreich: RStudio * Tools > Cheatsheets > ... *.

Anhang L

Hinweise

Anhang M

Icons

R spricht zu Ihnen; sie versucht es jedenfalls, mit einigen Items (Icon-Pond 2016).

R-Pseudo-Syntax: R ist (momentan) die führende Umgebung für Datenanalyse. Entsprechend zentral ist R in diesem Kurs. Zugegebenermaßen braucht es etwas Zeit, bis man ein paar Brocken “Errisch” spricht. Um den Einstieg zu erleichtern, ist Errisch auf Deutsch übersetzt an einigen Stellen, wo mir dies besonders hilfreich erschien. Diese Stellen sind mit diesem

Symbol  gekennzeichnet (für R-Pseudo-Syntax).

Achtung, Falle: Schwierige oder fehlerträchtige Stellen sind mit diesem Symbol  markiert.

Übungsaufgaben: Das Skript beinhaltet in jedem Kapitel Übungsaufgaben oder/und Testfragen.

Auf diese wird mit diesem Icon  verwiesen oder die Übungen sind in einem Abschnitt mit einsichtigem Titel zu finden.

Anhang N

Voraussetzungen

Dieses Skript hat einige *Voraussetzungen*, was das Vorwissen der Leser angeht; folgende Themengebiete werden vorausgesetzt:

- Deskriptive Statistik
- Grundlagen der Inferenzstatistik
- Grundlagen der Regressionsanalyse
- Skalenniveaus
- Grundlagen von R

Anhang O

Zitationen

Kunstwerke (Bilder) sind genau wie Standard-Literatur im Text zitiert. Alle Werke (auch Daten und Software) finden sich im Literaturverzeichnis.

Anhang P

Technische Details

Dieses Skript wurde mit dem Paket `bookdown` (Xie 2015) erstellt, welches wiederum stark auf den Paketen `knitr` (Xie 2015) und `rmarkdown` (Allaire u. a. 2016a) beruht. Diese Pakete stellen verblüffende Funktionalität zur Verfügung als freie Software (frei wie in Bier und frei wie in Freiheit).

Informationen zu den verwendeten Paketen etc. (`sessionInfo()`) finden Sie hier: https://raw.githubusercontent.com/sebastiansauer/Praxis_der_Datenanalyse/master/includes/sessionInfo_PraDa.html.

Anhang Q

Sonstiges

Aus Gründen der Lesbarkeit wird das männliche Generikum verwendet, welches Frauen und Männer in gleichen Maßen ansprechen soll.

Anhang R

Literaturverzeichnis

- Agresti, Alan. 2013. *Categorical data analysis*. Hoboken, N.J: Wiley-Interscience.
- Allaire, JJ, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, und Rob Hyndman. 2016a. *rmarkdown: Dynamic Documents for R*. <https://CRAN.R-project.org/package=rmarkdown>.
- . 2016b. *rmarkdown: Dynamic Documents for R*. <https://CRAN.R-project.org/package=rmarkdown>.
- Aronson, Elliot, Robin M Akert, und Timothy D Wilson. 2010. *Sozialpsychologie*. Muenchen: Pearson Deutschland GmbH.
- Auguie, Baptiste. 2016. *gridExtra: Miscellaneous Functions for „Grid“ Graphics*. <https://CRAN.R-project.org/package=gridExtra>.
- Aust, Frederik, und Marius Barth. 2017. *papaja: Create APA manuscripts with R Markdown*. <https://github.com/crsh/papaja>.
- Banerjee, Mousumi, Michelle Capozzoli, Laura McSweeney, und Debajyoti Sinha. 1999. „Beyond kappa: A review of interrater agreement measures“. *Canadian Journal of Statistics* 27 (1). Wiley-Blackwell: 3–23. doi:10.2307/3315487¹.
- Baumer, Benjamin S., Daniel T. Kaplan, und Nicholas J. Horton. 2017. *Modern Data Science with R (Chapman & Hall/CRC Texts in Statistical Science)*. Boca Raton, Florida: Chapman; Hall/CRC.
- Beaujean, A. Alexander. 2012. *BaylorEdPsych: R Package for Baylor University Educational Psychology Quantitative Courses*. <https://CRAN.R-project.org/package=BaylorEdPsych>.
- Begley, C. Glenn, und John P.A. Ioannidis. 2015. „Reproducibility in Science“. *Circulation Research* 116 (1). American Heart Association, Inc.: 116–26. doi:10.1161/CIRCRESAHA.114.303819².
- Benoit, Kenneth, und Paul Nulty. 2016. *quanteda: Quantitative Analysis of Textual Data*.

¹<https://doi.org/10.2307/3315487>

²<https://doi.org/10.1161/CIRCRESAHA.114.303819>

<https://CRAN.R-project.org/package=quanteda>.

Bivand, Roger S., Edzer Pebesma, und Virgilio Gomez-Rubio. 2013. *Applied Spatial Data Analysis with R (Use R!)*. New York: Springer.

Bivand, Roger, Tim Keitt, und Barry Rowlingson. 2017. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. <https://CRAN.R-project.org/package=rgdal>.

Bouchet-Valat, Milan. 2014. *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. <https://CRAN.R-project.org/package=SnowballC>.

Brennan, Robert L., und Dale J. Prediger. 1981. „Coefficient Kappa: Some Uses, Misuses, and Alternatives“. *Educational and Psychological Measurement* 41 (3). SAGE Publications: 687–99. doi:10.1177/001316448104100307³.

Briggs, William M. 2008a. *Breaking the Law of Averages: Real-Life Probability and Statistics in Plain English*. Lulu.com.

———. 2008b. *Breaking the Law of Averages: Real-Life Probability and Statistics in Plain English*. Lulu.com. <https://www.amazon.com/Breaking-Law-Averages-Probability-Statistics/dp/0557019907?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0557019907>.

———. 2016. *Uncertainty: The Soul of Modeling, Probability & Statistics*. Springer.

Brunsdon, Chris, und Lex Comber. 2015. *An Introduction to R for Spatial Analysis and Mapping*. Thousand Oaks, California: SAGE Publications Ltd. <https://www.amazon.com/Introduction-Spatial-Analysis-Mapping/dp/1446272958?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1446272958>.

Bryant, PG, und MA Smith. 1995. „Practical Data Analysis: Case Studies in Business Statistics, Homewood, IL: Richard D“. Irwin Publishing.

Burns, Patrick. 2012. *The R Inferno*. lulu.com. <https://www.amazon.com/R-Inferno-Patrick-Burns/dp/1471046524?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1471046524>.

Chang, Winston. 2015. *downloader: Download Files over HTTP and HTTPS*. <https://CRAN.R-project.org/package=downloader>.

Chapman, Chris, und Elea McDonnell Feit. 2015. *R for Marketing Research and Analytics*. New York City: Springer. doi:10.1007/978-3-319-14436-8⁴.

Cheng, Joe, Bhaskar Karambelkar, und Yihui Xie. 2017. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*. <https://CRAN.R-project.org/package=leaflet>.

Cleveland, William S. 1993. *Visualizing Data*. Hobart Press.

Clopper, Charles J, und Egon S Pearson. 1934. „The use of confidence or fiducial limits

³<https://doi.org/10.1177/001316448104100307>

⁴<https://doi.org/10.1007/978-3-319-14436-8>

- illustrated in the case of the binomial“. *Biometrika* 26 (4). JSTOR: 404–13.
- Cobb, George W. 2007. „The introductory statistics course: a Ptolemaic curriculum“ *Technology Innovations in Statistics Education* 1 (1).
- Cohen, J. 1992. „A power primer“. *Psychological Bulletin* 112 (1): 155–59.
- Cohen, Jacob. 1988. *Statistical Power Analysis for the Behavioral Sciences*. Routledge. <http://dx.doi.org/10.4324/9780203771587>.
- Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, und José Reis. 2009. „Modeling wine preferences by data mining from physicochemical properties“. *Decision Support Systems* 47 (4). Elsevier: 547–53.
- de Vries, Andrie, und Brian D. Ripley. 2016. *ggdendro: Create Dendograms and Tree Diagrams Using 'ggplot2'*. <https://CRAN.R-project.org/package=ggdendro>.
- Diez, David M, Christopher D Barr, und Mine Cetinkaya-Rundel. 2014. *Introductory Statistics with Randomization and Simulation*. North Charleston, South Carolina: CreateSpace Independent Publishing Platform.
- Durand, Martine. 2015. „The OECD Better Life Initiative: How's Life? and the Measurement of Well-Being“. *Review of Income and Wealth* 61 (1): 4–17. doi:10.1111/roiw.12156⁵.
- Efron, B., und R.J. Tibshirani. 1994. *An Introduction to the Bootstrap*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis. <https://books.google.de/books?id=gLlpIUxRntoC>.
- Eid, Michael, Mario Gollwitzer, und Manfred Schmitt. 2010. *Statistik und Forschungsmethoden*. Göttingen: Hogrefe.
- Etz, Alexander, Quentin Frederik Gronau, Fabian Dablander, Peter Edelsbrunner, und Beth Baribault. 2016. „How to become a Bayesian in eight easy steps: An annotated reading list“. PsyArXiv.
- Fair, Ray C. 1978. „A theory of extramarital affairs“. *Journal of Political Economy* 86 (1). The University of Chicago Press: 45–61.
- Feinerer, Ingo, und Kurt Hornik. 2015. *tm: Text Mining Package*. <https://CRAN.R-project.org/package=tm>.
- Fellows, Ian. 2014. *wordcloud: Word Clouds*. <https://CRAN.R-project.org/package=wordcloud>.
- Fjalnes. 2014. „Orthogonale Faktorrotation“. [https://de.wikipedia.org/wiki/Rotationsverfahren_\(Statistik\)#/media/File:Orthogonale_faktorrotation.svg](https://de.wikipedia.org/wiki/Rotationsverfahren_(Statistik)#/media/File:Orthogonale_faktorrotation.svg).
- Fox, John, und Sanford Weisberg. 2016. *car: Companion to Applied Regression*. <https://CRAN.R-project.org/package=car>.
- Gansser, Oliver. 2017. „Data for Principal Component Analysis and Common Factor Analysis“.

⁵<https://doi.org/10.1111/roiw.12156>

Open Science Framework. osf.io/zg89r.

Gigerenzer, Gerd. 1980. *Messung und Modellbildung in der Psychologie (Uni-Taschenbucher. Psychologie, Padagogik, Soziologie, Psychiatrie) (German Edition)*. E. Reinhardt.

———. 2004. „Mindless statistics“. *The Journal of Socio-Economics* 33 (5). Elsevier BV: 587–606. doi:10.1016/j.soec.2004.09.033⁶.

God. 2016. „I don't care about you. Please share this with friends.“ Twitter Tweet. *TheTweetOfGod*. <https://twitter.com/TheTweetOfGod/status/688035049187454976>.

Grolmund, Garrett, und Hadley Wickham. 2014. „A cognitive interpretation of data analysis“. *International Statistical Review* 82 (2). Wiley Online Library: 184–204.

Hahsler, Michael, Christian Buchta, Bettina Gruen, und Kurt Hornik. 2016. *arules: Mining Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arules>.

Hahsler, Michael, und Sudheer Chelluboina. 2016. *arulesViz: Visualizing Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arulesViz>.

Hamermesh, Daniel S, und Amy Parker. 2005. „Beauty in the classroom: Instructors' pulchritude and putative pedagogical productivity“. *Economics of Education Review* 24 (4). Elsevier: 369–76.

Hardin, Johanna, Roger Hoerl, Nicholas J Horton, Deborah Nolan, Ben Baumer, Olaf Hall-Holt, Paul Murrell, u. a. 2015. „Data science in statistics curricula: Preparing students to ‘Think with Data’“. *The American Statistician* 69 (4). Taylor & Francis: 343–53.

Hastie, Trevor, Robert Tibshirani, und Jerome Friedman. 2013. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Bd. 1. Springer Series in Statistics. New York City: Springer. doi:10.1007/b94608⁷.

Hatzinger, Reinholt, Kurt Hornik, und Herbert Nagel. 2014. *R- Einfuehrung durch angewandte Statistik*. Pearson Studium.

Head, Megan L., Luke Holman, Rob Lanfear, Andrew T. Kahn, und Michael D. Jennions. 2015. „The Extent and Consequences of P-Hacking in Science“. *PLOS Biology* 13 (3). Public Library of Science (PLoS): e1002106. doi:10.1371/journal.pbio.1002106⁸.

Hendricks, Paul. 2015. *titanic: Titanic Passenger Survival Data Set*. <https://CRAN.R-project.org/package=titanic>.

Hoekstra, Rink, Richard D Morey, Jeffrey N Rouder, und Eric-Jan Wagenmakers. 2014. „Robust misinterpretation of confidence intervals“. *Psychonomic bulletin & review* 21 (5). Springer: 1157–64.

Hyndman, R.J., und G. Athanasopoulos. 2014. *Forecasting: principles and practice*: OTexts.

⁶<https://doi.org/10.1016/j.soec.2004.09.033>

⁷<https://doi.org/10.1007/b94608>

⁸<https://doi.org/10.1371/journal.pbio.1002106>

<https://books.google.de/books?id=gDuRBAAAQBAJ>.

Icon-Pond. 2016. „Education. 35 Icons.“ Flaticon. <http://www.flaticon.com/authors/popcorns-arts>.

Ihaka, Ross, und Robert Gentleman. 1996. „R: A Language for Data Analysis and Graphics“. *Journal of Computational and Graphical Statistics* 5 (3): 299–314. doi:10.1080/10618600.1996.10474713⁹.

Ingo Feinerer, Kurt Hornik, und David Meyer. 2008. „Text Mining Infrastructure in R“. *Journal of Statistical Software* 25 (5): 1–54. <http://www.jstatsoft.org/v25/i05/>.

Jackson, Simon. 2016. *corr: Correlations in R*. <https://CRAN.R-project.org/package=corr>.

James, Gareth, Daniela Witten, Trevor Hastie, und Rob Tibshirani. 2013a. *ISLR: Data for An Introduction to Statistical Learning with Applications in R*. <https://CRAN.R-project.org/package=ISLR>.

James, Gareth, Daniela Witten, Trevor Hastie, und Robert Tibshirani. 2013b. *An introduction to statistical learning*. Bd. 6. Springer.

———. 2013c. *An introduction to statistical learning*. Bd. 6. Springer.

Julia, PhD Silge, und PhD Robinson David. 2017. *Text Mining with R: A tidy approach*. O'Reilly Media.

Kashnitsky, Ilya. 2017a. „Subplots in maps with ggplot2“. *ILYA KASHNITSKY*. <https://ikashnitsky.github.io/2017/subplots-in-maps/>.

———. 2017b. *Subplots in maps with ggplot2*. <https://ikashnitsky.github.io/2017/subplots-in-maps/>.

Kerby, Dave S. 2014. „The Simple Difference Formula: An Approach to Teaching Nonparametric Correlation“. *Comprehensive Psychology* 3: 11.IT.3.1. doi:10.2466/11.IT.3.1¹⁰.

Kershaw, Ian. 2015. *To Hell and Back: Europe, 1914-1949 (Alan Lane History)*. London: Allen Lane.

Kim, Albert Y, und Adriana Escobedo-Land. 2015. „OkCupid Data for Introductory Statistics and Data Science Courses“. *Journal of Statistics Education* 23 (2). Citeseer: n2.

Kim, Albert Y., und Adriana Escobedo-Land. 2016. *okcupiddata: OkCupid Profile Data for Introductory Statistics and Data Science Courses*. <https://CRAN.R-project.org/package=okcupiddata>.

Krämer, W. 2011. *Wie wir uns von falschen Theorien täuschen lassen*. Berlin University Press. <https://books.google.de/books?id=HWUKaAEACAAJ>.

Kruschke, John K. 2010. „Bayesian data analysis“. *Wiley Interdisciplinary Reviews: Cognitive*

⁹<https://doi.org/10.1080/10618600.1996.10474713>

¹⁰<https://doi.org/10.2466/11.IT.3.1>

- Science* 1 (5). Burlington, MA: Academic Press: 658–76.
- Kuhn, Max, und Kjell Johnson. 2013. *Applied predictive modeling*. Bd. 26. Springer.
- Liaw, Andy, und Matthew Wiener. 2002. „Classification and Regression by randomForest“. *R News* 2 (3): 18–22. <http://CRAN.R-project.org/doc/Rnews/>.
- Ligges, Uwe. 2008. *Programmieren mit R (Statistik und ihre Anwendungen) (German Edition)*. Springer. <https://www.amazon.com/Programmieren-Statistik-ihre-Anwendungen-German/dp/3540799974?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3540799974>.
- Ligges, Uwe, Martin Maechler, und Sarah Schnackenberg. 2017. *scatterplot3d: 3D Scatter Plot*. <https://CRAN.R-project.org/package=scatterplot3d>.
- Luhmann, Maike. 2015. *R für Einsteiger*. Nordhausen: Beltz.
- Lübke, Karsten, und Martin Vogt. 2014. *Angewandte Wirtschaftsstatistik: Daten und Zufall*. Berlin: Springer.
- M7. 2004. „Savinelli's Italian smoking pipe“. https://commons.wikimedia.org/wiki/File:Pipa_savinelli.jpg.
- Matejka, Justin, und George Fitzmaurice. 2017. „Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing“. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 1290–4. ACM.
- Micceri, Theodore. 1989. „The unicorn, the normal curve, and other improbable creatures.“ *Psychological Bulletin* 105 (1): 156–66. doi:10.1037/0033-2909.105.1.156¹¹.
- Michell, Joel. 2000. „Normal Science, Pathological Science and Psychometrics“. *Theory & Psychology* 10 (5): 639–67.
- Milborrow, Stephen. 2017. *rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'*. <https://CRAN.R-project.org/package=rpart.plot>.
- Moore, David S. 1990. „Uncertainty“. *On the shoulders of giants: New approaches to numeracy*. ERIC, 95–137.
- Mullen, Lincoln. 2016. *tokenizers: A Consistent Interface to Tokenize Natural Language Text*. <https://CRAN.R-project.org/package=tokenizers>.
- Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer Palettes*. <https://CRAN.R-project.org/package=RColorBrewer>.
- Neyman, J., und E. S. Pearson. 1933. „On the Problem of the Most Efficient Tests of Statistical Hypotheses“. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and*

¹¹<https://doi.org/10.1037/0033-2909.105.1.156>

Engineering Sciences 231 (694-706): 289–337. doi:10.1098/rsta.1933.0009¹².

Neyman, Jerzy. 1935. „On the problem of confidence intervals“. *The annals of mathematical statistics* 6 (3). JSTOR: 111–16.

Neyman, Jerzy, und Egon S Pearson. 1992. „On the problem of the most efficient tests of statistical hypotheses“. In *Breakthroughs in statistics*, 73–108. New York City: Springer.

Ooms, Jeroen. 2016. *pdftools: Text Extraction and Rendering of PDF Documents*. <https://CRAN.R-project.org/package=pdftools>.

Peirce, Charles S. 1955. „Abduction and induction“. *Philosophical writings of Peirce* 11. New York.

Peng, Roger D, und Elizabeth Matsui. 2015. „The Art of Data Science“. *A Guide for Anyone Who Works with Data*. Skybrude Consulting 200: 162.

Peng, Roger D. 2014. *R Programming for data science*. Victoria, British Columbia, Canada: Leanpub.

Popper, Karl. 1972. *Die Offene Gesellschaft und ihre Feinde*. Bern: Francke UTB.

Prel, Jean-Baptist du, Bernd Roehrig, Gerhard Hommel, und Maria Blettner. 2010. *Deutsches Aerzteblatt Online*, Mai. Deutscher Aerzte-Verlag. doi:10.3238/arztebl.2010.0343¹³.

Programmieren mit R. 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-540-79998-6¹⁴.

R Core Team. 2016. „R language definition“. *Vienna, Austria: R foundation for statistical computing*.

Rahlf, Thomas. 2014. *Datendesign mit R*. München: Open Source Press.

Raiche, Gilles, und David Magis. 2011. *nFactors: Parallel Analysis and Non Graphical Solutions to the Cattell Scree Test*. <https://CRAN.R-project.org/package=nFactors>.

Ram, Karthik, und Hadley Wickham. 2015. *wesanderson: A Wes Anderson Palette Generator*. <https://CRAN.R-project.org/package=wesanderson>.

Raschka, Sebastian. o. J. „Why are we growing decision trees via entropy instead of the classification error?“ Blog Post. <https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html>.

Re, AC Del. 2014. *compute.es: Compute Effect Sizes*. <https://CRAN.R-project.org/package=compute.es>.

Remus, R., U. Quasthoff, und G. Heyer. 2010. „SentiWS – a Publicly Available German-language Resource for Sentiment Analysis“. In *Proceedings of the 7th International Language*

¹²<https://doi.org/10.1098/rsta.1933.0009>

¹³<https://doi.org/10.3238/arztebl.2010.0343>

¹⁴<https://doi.org/10.1007/978-3-540-79998-6>

Resources and Evaluation (LREC'10), 1168–71.

Rickert, Joseph. 2017a. „10,000 CRAN Packages“. *R-Views*. <https://rviews.rstudio.com/2017/01/06/10000-cran-packages/>.

———. 2017b. *10,000 CRAN Packages*. <https://rviews.rstudio.com/2017/01/06/10000-cran-packages/>.

Ripley, Brian. 2016. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. <https://CRAN.R-project.org/package=MASS>.

rita, Bureau of transportation statistics. 2013. „nycflights13“. http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236.

Robinson, David. 2016. *gutenbergr: Download and Process Public Domain Works from Project Gutenberg*. <https://cran.rstudio.com/package=gutenbergr>.

———. 2017. „What are the Most Disliked Programming Languages?“ *stackoverflow Blog*. <https://stackoverflow.blog/2017/10/31/disliked-programming-languages/>.

Robinson, David, Matthieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, und Hadley Wickham. 2015. *broom: Convert Statistical Analysis Objects into Tidy Data Frames*. <https://CRAN.R-project.org/package=broom>.

Romeijn, Jan-Willem. 2016. „Philosophy of Statistics“. In *The Stanford Encyclopedia of Philosophy*, herausgegeben von Edward N. Zalta, Winter 2016. <http://plato.stanford.edu/archives/win2016/entries/statistics/>.

Rucker, Rudy. 2004. *Infinity and the Mind*. Princeton: Princeton University Press. <https://books.google.de/books?id=MDOUAwAAQBAJ>.

Sauer, Sebastian. 2016. „Extraversion Dataset“. Open Science Framework. doi:10.17605/OSF.IO/4KGZH¹⁵.

———. 2017a. „Dataset 'predictors of performance in stats test'“. Open Science Framework. doi:10.17605/OSF.IO/SJHUY¹⁶.

———. 2017b. „Dataset 'Height and shoe size'“. Open Science Framework. doi:10.17605/OSF.IO/JA9DW¹⁷.

Sauer, Sebastian, und Alexander Wolff. 2016. „The effect of a status symbol on success in online dating: an experimental study (data paper)“. *The Winnower*, August. doi:10.15200/winn.147241.13309¹⁸.

Sauer, Sebastian, Harald Walach, und Niko Kohls. 2010. „Gray's Behavioural Inhibition System as a mediator of mindfulness towards well-being“. *Personality and Individual Differences* 50 (4). Pergamon: 506–51. doi:10.1016/j.paid.2010.11.019¹⁹.

Schloerke, Barret, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen,

¹⁵<https://doi.org/10.17605/OSF.IO/4KGZH>

¹⁶<https://doi.org/10.17605/OSF.IO/SJHUY>

¹⁷<https://doi.org/10.17605/OSF.IO/JA9DW>

¹⁸<https://doi.org/10.15200/winn.147241.13309>

¹⁹<https://doi.org/10.1016/j.paid.2010.11.019>

- Amos Elberg, und Joseph Larmarange. 2016. *GGally: Extension to 'ggplot2'*. <https://CRAN.R-project.org/package=GGally>.
- Schmidt, Peter, Sebastian Bamberg, Eldad Davidov, Johannes Herrmann, und Shalom H. Schwartz. 2007. „Die Messung von Werten mit dem Portraits Value Questionnaire“. *Zeitschrift für Sozialpsychologie* 38 (4). Hogrefe Publishing Group: 261–75. doi:10.1024/0044-3514.38.4.261²⁰.
- Schuler, Heinz. 2015. *Lehrbuch Organisationspsychologie*. Bern: Huber Hans.
- Schwartz, Shalom H. 1999. „A Theory of Cultural Values and Some Implications for Work“. *Applied Psychology* 48 (1). Blackwell Publishing Ltd: 23–47. doi:10.1111/j.1464-0597.1999.tb00047.x²¹.
- Shmueli, Galit. 2010. „To Explain or to Predict?“ *Statistical Science* 25 (3): 289–310. doi:10.1214/10-STS330²².
- Silge, Julia. 2016. *janeaustenr: Jane Austen's Complete Novels*. <https://CRAN.R-project.org/package=janeaustenr>.
- Silge, Julia, David Robinson, und Jim Hester. 2016. „tidytext: Text mining using dplyr, ggplot2, and other tidy tools“. doi:10.5281/zenodo.56714²³.
- Silge, Julia, und David Robinson. 2016a. „tidytext: Text Mining and Analysis Using Tidy Data Principles in R“. *The Journal of Open Source Software* 1 (3). The Open Journal. doi:10.21105/joss.00037²⁴.
- . 2016b. „tidytext: Text Mining and Analysis Using Tidy Data Principles in R“. *JOSS* 1 (3). The Open Journal. doi:10.21105/joss.00037²⁵.
- South, Andy. 2011. „rworldmap: A New R package for Mapping Global Data“. *The R Journal* 3 (1): 35–43. http://journal.r-project.org/archive/2011-1/RJournal_2011-1_South.pdf.
- Spurzem, Lothar. 2017. „VW 1303 von Wiking in 1:87“. [https://de.wikipedia.org/wiki/Modellautomobil#/media/File:Wiking-Modell_VW_1303_\(um_1975\).JPG](https://de.wikipedia.org/wiki/Modellautomobil#/media/File:Wiking-Modell_VW_1303_(um_1975).JPG).
- Stevenson, A. 2010. *Oxford Dictionary of English*. Oxford Dictionary of English. OUP Oxford. <https://books.google.de/books?id=anecAQAAQBAJ>.
- Strobl, Carolin, und James Malley. 2009. „An Introduction to Recursive Partitioning An Introduction to Recursive Partitioning : Rationale , Application and Characteristics Bagging and Random Forests“. *Psychological Methods* 14 (4): 323?348. doi:10.1037/a0016973²⁶.
- Suppes, Patrick, und Joseph L Zinnes. 1962. *Basic measurement theory*. Institute for mathe-

²⁰<https://doi.org/10.1024/0044-3514.38.4.261>

²¹<https://doi.org/10.1111/j.1464-0597.1999.tb00047.x>

²²<https://doi.org/10.1214/10-STS330>

²³<https://doi.org/10.5281/zenodo.56714>

²⁴<https://doi.org/10.21105/joss.00037>

²⁵<https://doi.org/10.21105/joss.00037>

²⁶<https://doi.org/10.1037/a0016973>

matical studies in the social sciences.

Tan, Pang-Ning. 2013. *Introduction to Data Mining*. Addison-Wesley. <https://www.amazon.com/Introduction-to-Data-Mining-Pang-Ning-Tan/dp/0133128903?SubscriptionId=0JYN1NVV651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0133128903>.

The Oxford Dictionary of Statistical Terms. 2006. Oxford University Press.

Therneau, Terry, Beth Atkinson, und Brian Ripley. 2015. *rpart: Recursive Partitioning and Regression Trees*. <https://CRAN.R-project.org/package=rpart>.

Tufte, Edward R. 1990. *Envisioning Information*. Graphics Press.

———. 2001. *The Visual Display of Quantitative Information*. Graphics Press.

———. 2006. *Beautiful Evidence*. Graphics Press.

Unrau, Sebastian. 2017. „No Title“. <https://unsplash.com/photos/CoD2Q92UaEg>.

VanDerWal, Jeremy, Lorena Falconi, Stephanie Jamuchowski, Luke Shoo, und Collin Storlie. 2014. *SDMTools: Species Distribution Modelling Tools: Tools for processing data associated with species distribution modelling exercises*. <https://CRAN.R-project.org/package=SDMTools>.

Wagenmakers, Eric-Jan. 2007. „A practical solution to the pervasive problems of p values“. *Psychonomic Bulletin & Review* 14 (5). Springer Nature: 779–804. doi:10.3758/bf03194105²⁷.

Warnes, Gregory R., Ben Bolker, Lodewijk Bonebakker, Robert Gentleman, Wolfgang Huber, Andy Liaw, Thomas Lumley, Martin Maechler, u. a. 2016. *gplots: Various R Programming Tools for Plotting Data*. <https://CRAN.R-project.org/package=gplots>.

Wasserstein, Ronald L., und Nicole A. Lazar. 2016. „The ASA’s Statement on p-Values: Context, Process, and Purpose“. *The American Statistician* 70 (2). Taylor & Francis: 129–33. doi:10.1080/00031305.2016.1154108²⁸.

Wei, Taiyun, und Viliam Simko. 2016. *corrplot: Visualization of a Correlation Matrix*. <https://CRAN.R-project.org/package=corrplot>.

Wicherts, Jelte M., Coosje L. S. Veldkamp, Hilde E. M. Augusteijn, Marjan Bakker, Robbie C. M. van Aert, und Marcel A. L. M. van Assen. 2016. „Degrees of Freedom in Planning, Running, Analyzing, and Reporting Psychological Studies: A Checklist to Avoid p-Hacking“. *Frontiers in Psychology* 7 (November). Frontiers Media SA. doi:10.3389/fpsyg.2016.01832²⁹.

Wickham, Hadley. 2007. „Reshaping Data with the reshape Package“. *Journal of Statistical*

²⁷<https://doi.org/10.3758/bf03194105>

²⁸<https://doi.org/10.1080/00031305.2016.1154108>

²⁹<https://doi.org/10.3389/fpsyg.2016.01832>

- Software* 21 (12): 1–20. <http://www.jstatsoft.org/v21/i12/>.
- . 2009a. *Ggplot2 : elegant graphics for data analysis*. Dordrecht New York: Springer.
- . 2009b. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <http://ggplot2.org>.
- . 2014a. *Advanced R*. Boca Raton, Florida: CRC Press.
- . 2014b. „Tidy Data“. *Journal of Statistical Software* 59 (1): 1–23. doi:10.18637/jss.v059.i10³⁰.
- . 2016a. *ggplot2: Elegant Graphics for Data Analysis (Use R!)*. New York: Springer.
- . 2016b. *tidyverse: Easily Tidy Data with ‘spread()’ and ‘gather()’ Functions*. <https://CRAN.R-project.org/package=tidyr>.
- . 2017a. *nycflights13: Flights that Departed NYC in 2013*. <https://CRAN.R-project.org/package=nycflights13>.
- . 2017b. *stringr: Simple, Consistent Wrappers for Common String Operations*. <https://CRAN.R-project.org/package=stringr>.
- . 2017c. *tidyverse: Easily Install and Load ‘Tidyverse’ Packages*. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, Hadley, Jim Hester, und Romain Francois. 2016a. *readr: Read Tabular Data*. <https://CRAN.R-project.org/package=readr>.
- . 2016b. *readr: Read Tabular Data*. <https://CRAN.R-project.org/package=readr>.
- Wickham, Hadley, und Romain Francois. 2016. *dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, und Garrett Grolemund. 2016. *R for Data Science: Visualize, Model, Transform, Tidy, and Import Data*. O'Reilly Media.
- Wikipedia. 2017. „Körpergröße — Wikipedia, Die freie Enzyklopädie“ . <https://de.wikipedia.org/w/index.php?title=K%C3%B6rpergr%C3%B6%C3%9Fe&oldid=165047921>.
- Wild, Chris J, und Maxine Pfannkuch. 1999. „Statistical thinking in empirical enquiry“. *International Statistical Review* 67 (3). Wiley Online Library: 223–48.
- Wild, Fridolin. 2015. *lsa: Latent Semantic Analysis*. <https://CRAN.R-project.org/package=lsa>.
- Wilkinson, Leland. 2006. *The grammar of graphics*. Springer Science & Business Media.
- Xie, Yihui. 2015. *Dynamic Documents with R and knitr*. 2nd Aufl. Boca Raton, Florida: Chapman; Hall/CRC. <http://yihui.name/knitr/>.
- . 2016. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://CRAN.R-project.org/package=knitr>.

³⁰<https://doi.org/10.18637/jss.v059.i10>

//CRAN.R-project.org/package=knitr.

Ziemann, Mark, Yotam Eren, und Assam El-Osta. 2016. „Gene name errors are widespread in the scientific literature“. *Genome Biology* 17 (1): 177. doi:10.1186/s13059-016-1044-7³¹.

Zuguang, Gu. 2017. „Add legends to circlize plot“. http://zuguang.de/blog/html/add_legend_to_circlize.html.

Zumel, Nina, John Mount, und Jim Porzak. 2014. *Practical data science with R*. Manning.

2015. „Estimating the reproducibility of psychological science“. *Science* 349 (6251). American Association for the Advancement of Science. doi:10.1126/science.aac4716³².

³¹<https://doi.org/10.1186/s13059-016-1044-7>

³²<https://doi.org/10.1126/science.aac4716>

Index

- 106
Überanpassung, 210
- Algorithmus, 323
Allgemeines Lineares Modells, 275
angeleitetes Lernen, 208
Argument, 28
Ausdruck, 383
- Bagging, 311, 327
baumbasierte Verfahren, 311
Befehl, Funktion, 28
Bestimmtheitsmaß, 259
Bias, 210
Binnen, 106
Bootstrap, 228
Bootstrapping, 327
- Chancen, 277
Cohens d, 245
- Dataframe, 38
datengenerierende Maschine, 205
Datenjudo, 60
Datenstrukturen, 33
Determinationskoeffizient, 259
deterministisch, 205
Dimensionsreduzierendes Modellieren, 208
dplyr::arrange, 66
dplyr::count, 75
dplyr::filter, 62
dplyr::mutate, 80
dplyr::n, 74
dplyr::select, 65
dplyr::summarise, 71
Dublette, 98
Durchpfeifen, 78
- Effektstärke, 245
- einfaches reproduzierbares Beispiel, 24
Einflussgrößen, 204
Entropy, 324
Entscheidungsbäume, 311
environment, 14
Erklären, 207
euklidischen Abstand, 341
Explikatives Modellieren, 207
- Fallreduzierendes Modellieren, 208
Funktion, 14
Funktionen, 368
- Geleitetes Modellieren, 206
Gini-Index, 324
Git, 10
- Indizieren, 39
Interaktionseffekt, 270
- Klassifikation, 206, 281
Konfidenzintervalle, 244
Konfusionsmatrix, 282
Konsole, 14
Kreuzvalidierung, 213
Kriterium, 204
Kriterium der Kleinsten Quadrate, 262
- Lagemaße, 83
Lernen ohne Anleitung, 208
Listen, 37
Listenspalte, 176
logistische Funktion, 275
Logit, 277
- multivariat, 268
- non-standard evaluation, 385
Nullhypothese, 233
Nullhypotesen-Signifikanztesten, 231

- Nullmodell, 215
- Objekt, 27
- Objekte, 33
- Odds, 277
- OOB, 328
- Ordinary Least Squares, 262
- overfitting, 210
- p-Wert, 231
- Pakete, 15
- Partitionierung, 313
- Pfeife, 78
- prädiktive Modellierung, 9
- Prädiktives Modellieren, 206
- Prädiktoren, 204
- R-Skript, 20
- Random Forests, 311
- Reduzieren, 208
- Regression, 255
- rekursives Partitionieren, 323
- Relation, 203
- Reproduzierbarkeit, 10
- Resampling, 228
- Residuen, 262
- robust, 210
- ROC, 284
- Signifikanz, 233
- Skript-Fenster, 14
- Standardfehler, 226
- Standardnormalverteilung, 103
- Stichprobenverteilung, 226
- Stop-Kriterien, 323
- Streuungsmaße, 84
- Transformieren, 81
- Tuningparameter, 318, 330
- Umgebung, 14
- Umkodieren, 106
- underfitting, 210
- Ungeleiteten Modellieren, 208
- unsupervised learning, 208
- Unteranpassung, 210
- Variable, 27
- variable importance, 332
- Variablen, 26
- Variablenrelevanz, 332
- Vektor, 34
- vektoriell, 30
- Vorhersagefehler, 259
- Vorhersagen, 206
- Youden-Index, 283