

Praxis der Datenanalyse ENTWURF

Skript zum Modul

*Sebastian Sauer. Mit Beiträgen von Oliver Gansser, Matthias Gehrke und
Karsten Lübke*

12 June, 2017

Inhaltsverzeichnis

Vorwort	xvii
Organisatorisches	xix
0.1 Modulziele	xix
0.2 Themen pro Termin	xix
0.3 Vorerfahrung	xx
0.4 Prüfung	xx
0.4.1 Prüfungshinweise	xx
0.4.2 Klausur	xxi
0.4.3 Datenanalyse	xxi
0.4.4 Gliederungsvorschlag zur Datenanalyse	xxii
0.5 Literatur	xxiii
I Grundlagen	1
1 Rahmen	3
1.1 Software installieren	4
1.1.1 R und RStudio installieren	5
1.1.2 Sonstiges Material für dieses Skript	6
1.1.3 Hilfe! R startet nicht!	6
1.1.4 Pakete	8
1.1.5 R-Pakete für dieses Buch	9
1.1.6 Vertiefung: Zuordnung von Paketen zu Befehlen	10
1.1.7 Datensätze	11
1.2 ERRRstkontakte	12
1.2.1 R-Skript-Dateien	12
1.2.2 Datentypen in R	12
1.2.3 Hinweise	13
1.2.4 R als Taschenrechner	13
1.2.5 Text und Variablen zuweisen	14
1.2.6 Funktionen aufrufen	15
1.2.7 Das Arbeitsverzeichnis	15
1.2.8 Hier werden Sie geholfen	16

1.2.9	Aufgaben	17
1.3	Was ist Statistik? Wozu ist sie gut?	17
1.4	Befehlsübersicht	19
1.5	Verweise	20
2	Daten einlesen	21
2.1	Daten in R importieren	22
2.1.1	Excel-Dateien importieren	22
2.1.2	CSV-Dateien importieren	23
2.2	Normalform einer Tabelle	24
2.3	Tabelle in Normalform bringen	26
2.4	Textkodierung	28
2.5	Befehlsübersicht	28
2.6	Aufgaben	29
2.7	Verweise	29
3	Datenjudo	31
3.1	Typische Probleme der Datenaufbereitung	33
3.2	Daten aufbereiten mit <code>dplyr</code>	33
3.2.1	Die zwei Prinzipien von <code>dplyr</code>	33
3.3	Zentrale Bausteine von <code>dplyr</code>	34
3.3.1	Zeilen filtern mit <code>filter</code>	34
3.3.2	Spalten wählen mit <code>select</code>	37
3.3.3	Zeilen sortieren mit <code>arrange</code>	38
3.3.4	Datensatz gruppieren mit <code>group_by</code>	41
3.3.5	Eine Spalte zusammenfassen mit <code>summarise</code>	44
3.3.6	Zeilen zählen mit <code>n</code> und <code>count</code>	46
3.4	Die Pfeife	51
3.4.1	Spalten berechnen mit <code>mutate</code>	53
3.4.2	Aufgaben	54
3.5	Deskriptive Statistik	56
3.6	Befehlsübersicht	59
3.7	Verweise	59
4	Praxisprobleme der Datenaufbereitung	61
4.1	Datenaufbereitung	61
4.1.1	Auf fehlende Werte prüfen	61
4.1.2	Fälle mit fehlenden Werte löschen	62
4.1.3	Fehlende Werte ggf. ersetzen	63
4.1.4	Nach Fehlern suchen	64
4.1.5	Ausreißer identifizieren	64
4.1.6	Hochkorrelierte Variablen finden	64
4.1.7	z-Standardisieren	66
4.1.8	Quasi-Konstante finden	67
4.1.9	Auf Normalverteilung prüfen	68

4.1.10 Werte umkodieren und partionieren (“binnen”)	68
4.2 Deskriptive Statistiken berechnen	74
4.2.1 Mittelwerte pro Zeile berechnen	74
4.2.2 Mittelwerte pro Spalte berechnen	74
4.2.3 Korrelationstabellen berechnen	76
4.3 Befehlsübersicht	78
5 Fallstudie ‘movies’	79
5.1 Wie viele Filme gibt es pro Genre?	80
5.2 Welches Genre ist am häufigsten?	81
5.3 Zusammenhang zwischen Budget und Beurteilung	81
5.4 Wurden die Filme im Lauf der Jahre teurer und/oder “besser”?	82
6 Daten visualisieren	83
6.1 Ein Bild sagt mehr als 1000 Worte	84
6.2 Die Anatomie eines Diagramms	85
6.3 Einstieg in ggplot2 - qplot	86
6.4 Häufige Arten von Diagrammen	88
6.4.1 Eine kontinuierliche Variable	88
6.4.2 Zwei kontinuierliche Variablen	93
6.4.3 Eine nominale Variable	97
6.4.4 Zwei nominale Variablen	100
6.4.5 Zusammenfassungen zeigen	101
6.4.6 Überblick zu häufigen Diagrammtypen	105
6.5 Die Gefühlswelt von ggplot2	105
6.6 Aufgaben	106
6.7 Lösungen	106
6.8 Richtig oder Falsch	109
6.9 Befehlsübersicht	109
6.10 Vertiefung: Geome bei ggplot2	109
6.11 Verweise	111
7 Fallstudie zur Visualisierung	113
7.1 Daten einlesen	113
7.2 Daten umstellen	114
7.3 Diagramme für Anteile	115
7.4 Um 90° drehen	116
7.5 Text-Labels für die Items	117
7.6 Diagramm mit Häufigkeiten	119
7.7 Farbschema	120
II Modellieren	121
8 Grundlagen des Modellierens	123

8.1	Was ist ein Modell? Was ist Modellieren?	124
8.2	Ein Beispiel zum Modellieren in der Datenanalyse	126
8.3	Taxonomie der Ziele des Modellierens	129
8.4	Die vier Schritte des statistischen Modellierens	130
8.5	Einfache vs. komplexe Modelle: Unter- vs. Überanpassung	131
8.6	Bias-Varianz-Abwägung	133
8.7	Training- vs. Test-Stichprobe	134
8.8	Wann welches Modell?	135
8.9	Modellgüte	135
8.10	Auswahl von Prädiktoren	136
8.11	Aufgaben	136
8.12	Befehlsübersicht	138
8.13	Verweise	138
9	Der p-Wert, Inferenzstatistik und Alternativen	139
9.1	Der p-Wert sagt nicht das, was viele denken	140
9.1.1	Von Männern und Päpsten	141
9.2	Der p-Wert ist eine Funktion der Stichprobengröße	142
9.3	Mythen zum p-Wert	143
9.3.1	Wann welcher Inferenztest?	143
9.4	Zur Philosophie des p-Werts: Frequentismus	144
9.5	Alternativen zum p-Wert	145
9.5.1	Konfidenzintervalle	145
9.5.2	Effektstärke	146
9.5.3	Bayes-Statistik	147
9.6	Fazit	149
9.7	Verweise	149
III	Geleitetes Modellieren	151
10	Lineare Regression	153
10.1	Die Idee der klassischen Regression	154
10.2	Vorhersagegüte	156
10.2.1	Mittlere Quadratfehler	157
10.2.2	R-Quadrat (R^2)	158
10.3	Die Regression an einem Beispiel erläutert	159
10.4	Überprüfung der Annahmen der linearen Regression	162
10.5	Regression mit kategorialen Prädiktoren	164
10.6	Multiple Regression	166
10.7	Interaktionen	168
10.8	Fallstudie zu Overfitting	169
10.9	Befehlsübersicht	170
11	Klassifizierende Regression	173

11.1 Vorbereitung	174
11.2 Problemstellung	174
11.3 Die Idee der logistischen Regression	177
11.4 Kein R^2 , dafür AIC	179
11.5 Interpretation der Koeffizienten	179
11.5.1 y-Achsenabschnitt (Intercept) β_0	179
11.5.2 Steigung β_i mit $i = 1, 2, \dots, K$	180
11.5.3 Aufgabe	180
11.6 Kategoriale Prädiktoren	181
11.7 Multiple logistische Regression	183
11.8 Modellgüte	184
11.9 Klassifikationskennzahlen	184
11.9.1 Vier Arten von Ergebnissen einer Klassifikation	184
11.9.2 Klassifikationsgütekennzahlen	186
11.9.3 ROC-Kurven	187
11.10 Befehlsübersicht	189
12 Fallstudien zum geleiteten Modellieren	191
12.1 Überleben auf der Titanic	191
12.1.1 Daten und Pakete laden	191
12.1.2 Erster Blick	192
12.1.3 Welche Variablen sind interessant?	192
12.1.4 Univariate Häufigkeiten	192
12.1.5 Bivariate Häufigkeiten	194
12.1.6 Signifikanztest	196
12.1.7 Effektstärke	197
12.1.8 Logististische Regression	199
12.1.9 Effektstärken visualisieren	201
12.1.10 Fazit	205
12.2 Außereheliche Affären	205
12.2.1 Zentrale Statistiken	206
12.2.2 Visualisieren	207
12.2.3 Wer ist zufriedener mit der Partnerschaft: Personen mit Kindern oder ohne?	208
12.2.4 Wie viele fehlende Werte gibt es?	208
12.2.5 Wer ist glücklicher: Männer oder Frauen?	209
12.2.6 Effektstärken	210
12.2.7 Korrelationen	211
12.2.8 Ehejahre und Affären	212
12.2.9 Ehezufriedenheit als Prädiktor	213
12.2.10 Weitere Prädiktoren der Affärenhäufigkeit	215
12.2.11 Unterschied zwischen den Geschlechtern	216
12.2.12 Kinderlose Ehe vs. Ehen mit Kindern	218
12.2.13 Halodries	219
12.2.14 logistische Regression	220

12.2.15 Zum Abschluss	222
12.3 Befehlsübersicht	223
IV Ungeleitetes Modellieren	225
13 Vertiefung: Clusteranalyse	227
13.1 Einführung	227
13.2 Intuitive Darstellung der Clusteranalyse	228
13.3 Euklidische Distanz	230
13.4 Daten	233
13.5 Distanzmaße mit R berechnen	234
13.6 k-Means Clusteranalyse	234
13.7 Aufgaben	237
13.8 Befehlsübersicht	237
13.9 Verweise	238
14 Vertiefung: Dimensionsreduktion	239
14.1 Einführung	240
14.2 Gründe für die Notwendigkeit der Datenreduktion	241
14.3 Intuition zur Dimensionsreduktion	241
14.4 Daten	243
14.5 Neuskalierung der Daten	243
14.6 Zusammenhänge in den Daten	244
14.7 Daten mit fehlende Werten	245
14.8 Hauptkomponentenanalyse (PCA)	245
14.8.1 Bestimmung der Anzahl der Hauptkomponenten	246
14.8.2 Scree-Plot	246
14.8.3 Ellbogen-Kriterium	247
14.8.4 Eigenwert-Kriterium	247
14.8.5 Biplot	248
14.8.6 Aufgaben	249
14.8.7 Interpretation der Ergebnisse der PCA	250
14.9 Exploratorische Faktorenanalyse (EFA)	252
14.9.1 Finden einer EFA Lösung	253
14.9.2 Schätzung der EFA	253
14.9.3 Vertiefung: Heatmap mit Ladungen	254
14.9.4 Berechnung der Faktor-Scores	255
14.10 Interne Konsistenz der Skalen	256
14.11 Befehlsübersicht	257
15 Vertiefung: Textmining	259
15.1 Zentrale Begriffe	260
15.2 Grundlegende Analyse	261
15.2.1 Tidy Text Dataframes	261

15.2.2 Text-Daten einlesen	262
15.2.3 Worthäufigkeiten auszählen	264
15.2.4 Visualisierung	266
15.3 Aufgaben	267
15.4 Befehlsübersicht	268
15.5 Verweise	268
A Probeklausur	269
B Lösungen	273
C Hinweise	275
D Icons	277
E Voraussetzungen	279
F Zitationen	281
G Lizenz	283
H Autoren	285
I Danke	287
J Zitation dieses Skripts	289
K Kontakt	291
L Technische Details	293
M Sonstiges	295
N Literaturverzeichnis	297

Tabellenverzeichnis

1	Zuordnung von Themen zu Terminen	xx
1.1	Wichtige Datentypen in R	12
1.2	Befehle des Kapitels 'Rahmen'	20
2.1	Befehle des Kapitels 'Daten einlesen'	28
3.1	Befehle des Kapitels 'Datenjudo'	59
4.1	Befehle des Kapitels 'Praxisprobleme'	78
6.1	Häufige Diagrammtypen	105
6.2	Befehle des Kapitels 'Daten visualisieren'	109
8.1	Befehle des Kapitels 'Modellieren'	138
9.1	Überblick über gängige Effektstärkemaße (continued below)	146
10.1	Befehle des Kapitels 'Regression'	171
11.1	Vier Arten von Ergebnisse von Klassifikationen	185
11.2	Geläufige Kennwerte der Klassifikation	186
11.3	Befehle des Kapitels 'Logistische Regression'	189
12.1	Befehle des Kapitels 'Fallstudien'	223
13.1	Befehle des Kapitels 'Clusteranalyse'	238
14.3	Befehle des Kapitels 'Dimensionsreduktion'	258
15.1	Die häufigsten Wörter im AfD-Parteiprogramm	265
15.2	Die häufigsten Wörter im AfD-Parteiprogramm mit 'stemming'	265
15.3	Befehle des Kapitels 'Textmining'	268

Abbildungsverzeichnis

1.1	Der Prozess der Datenanalyse	4
1.2	So installiert man Pakete in RStudio	8
1.3	Hier werden Sie geholfen: Die Dokumentation der R-Pakete	10
1.4	Das Arbeitsverzeichnis mit RStudio auswählen	16
1.5	Sinnbild für die Deskriptiv- und die Inferenzstatistik	18
2.1	Daten sauber einlesen	22
2.2	Daten einlesen (importieren) mit RStudio	22
2.3	Trennzeichen einer CSV-Datei in RStudio einstellen	24
2.4	Schematische Darstellung eines Dataframes in Normalform	25
2.5	Dieselben Daten - einmal breit, einmal lang	25
2.6	Illustration eines Datensatzes in Normalform	26
2.7	Mit 'gather' und 'spread' wechselt man von der breiten Form zur langen Form	27
2.8	Ein Beispiel für eine Abbildung zu einer Normalform-Tabelle	27
3.1	Daten aufbereiten	32
3.2	Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine	34
3.3	Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergereicht	34
3.4	Zeilen filtern	35
3.5	Spalten auswählen	37
3.6	Spalten sortieren	40
3.7	Datensätze nach Subgruppen aufteilen	42
3.8	Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'	43
3.9	Spalten zu einer Zahl zusammenfassen	44
3.10	Sinnbild für 'count'	48
3.11	Das ist keine Pfeife	51
3.12	Das 'Durchpfeifen'	51
3.13	Sinnbild für mutate	54
4.1	Ausreißer identifizieren	65
4.2	Ein Korrelationsplot	66
4.3	Visuelles Prüfen der Normalverteilung	68
4.4	Sinnbild für Umkodieren	69
4.5	Sinnbild zum 'Binnen'	69

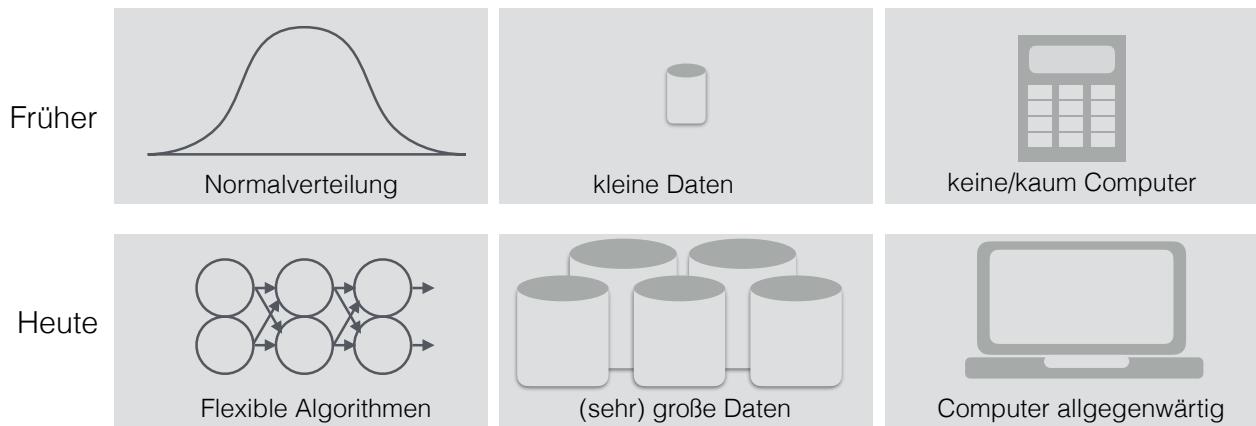
6.1	Das Anscombe-Quartett	84
6.2	Anatomie eines Diagramms	85
6.3	Mittleres Budget pro Jahr	86
6.4	Film-Budgets über die die Jahrzehnte	88
6.5	Verteilung des Budgets von Filmen	89
6.6	Film-Budgets mit Histogrammen	107
8.1	Modell eines VW-Käfers	124
8.2	Modellieren	125
8.3	Formaleres Modell des Modellierens	126
8.4	Ein Beispiel für Modellieren	127
8.5	Ein Beispiel für ein Pfadmodell	127
8.6	Ein etwas aufwändigeres Modell	127
8.7	Modelle mit schwarzer Kiste	128
8.8	Die zwei Arten des ungeleiteten Modellierens	130
8.9	Welches Modell (Teil B-D; rot, grün, blau) passt am besten zu den Daten (Teil A) ?	131
8.10	'Mittlere' Komplexität hat die beste Vorhersagegenauigkeit (am wenigsten Fehler) in der Test-Stichprobe	132
8.11	Der Spagat zwischen Verzerrung und Varianz	133
8.12	Bias-Varianz-Abwägung. Links: Wenig Bias, viel Varianz. Rechts: Viel Bias, wenig Varianz.	137
9.1	Der größte Statistiker des 20. Jahrhunderts ($p < .05$)	140
9.2	Der p-Wert wird oft als wichtig erachtet	140
9.3	Moslem und Terrorist zu sein, ist nicht das gleiche.	141
9.4	Zwei Haupteinflüsse auf den p-Wert	142
9.5	Anteil von 'Kopf' bei wiederholtem Münzwurf	144
9.6	Die zwei Stufen der Bayes-Statistik in einem einfachen Beispieli	148
10.1	Beispiel für eine Regression	154
10.2	Zwei weitere Beispiele für Regressionen	155
10.3	Geringer (links) vs. hoher (rechts) Vorhersagefehler	157
10.4	Streudiagramm von Lernzeit und Klausurerfolg	160
10.5	Die Residuen verteilen sich hinreichend normal.	162
10.6	Vorhergesagte Werte vs. Residualwerte im Datensatz tips	163
10.7	Eine multivariate Analyse fördert Einsichten zu Tage, die bei einfacheren Analysen verborgen bleiben	167
10.8	Eine Regressionsanalyse mit Interaktionseffekten	169
11.1	Streudiagramm von Risikobereitschaft und Aktienkauf	175
11.2	Regressionsgerade für Aktien-Modell	176
11.3	Die logistische Regression beschreibt eine 's-förmige' Kurve	176
11.4	Modelldiagramm für den Aktien-Datensatz	177
11.5	Verwackeltes Streudiagramm ('Jitter')	181
11.6	Eine ROC-Kurve	188

11.7 Beispiel für eine sehr gute (A), gute (B) und schlechte (C) Klassifikation	188
13.1 Ein Streudiagramm - sehen Sie Gruppen (Cluster) ?	228
13.2 Ein Streudiagramm - mit drei Clustern	229
13.3 Unterschiedliche Anzahlen von Clustern im Vergleich	229
13.4 Die Summe der Varianz within in Abhängigkeit von der Anzahl von Clustern. Ein Screeplot.	230
13.5 Distanz zwischen zwei Punkten in der Ebene	231
13.6 Pythagoras in 3D	231
13.7 Pythagoras in Reihe geschaltet	232
14.1 Der Pfeil ist eindimensional; reduziert also die drei Dimensionen auf eine	242
14.2 Screeplot	247
14.3 VSS-Screepplot	248
14.4 Ein Biplot für den Werte-Datensatz	249
14.5 Beispiel für eine rechtwinklige Rotation	254
14.6 Heatmap einer EFA	255

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 such 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 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.

Dieser Kurs 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.

Dieses Skript ist publiziert unter CC-BY-NC-SA 3.0 DE².



Sebastian Sauer

Herausgeber: FOM Hochschule für Oekonomie & Management gemeinnützige GmbH

Dieses Skript dient als Begleitmaterial zum Modul “Praxis der Datenanalyse” des Masterstudiengangs “Wirtschaftspsychologie & Consulting” der FOM Hochschule für Oekonomie & Management.

FOM. Die Hochschule. Für Berufstätige. Die mit bundesweit über 42.000 Studierenden größte private Hochschule Deutschlands führt seit 1993 Studiengang für Berufstätige durch, die einen staatlich und international anerkannten Hochschulabschluss (Bachelor/Master) erlangen wollen. Weitere Informationen finden Sie unter <www.fom.de>

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>

²<https://creativecommons.org/licenses/by-nc-sa/3.0/de/>

Organisatorisches



0.1 Modulziele



Die Studierenden können nach erfolgreichem Abschluss des Moduls:

- 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.

0.2 Themen pro Termin

Für dieses Modul sind 44 UE für Lehre eingeplant, aufgeteilt in 11 Termine (vgl. 1).

Folgende Abfolge von Themen sind pro Termin vorgeschlagen:

Tabelle 1: Zuordnung von Themen zu Terminen

Termin	Thema / Kapitel
1	Organisatorisches
1	Einführung
1	Rahmen
1	Daten einlesen
2	Datenjudo
3	Daten visualisieren
4	Fallstudie
5	Der p-Wert
5	Daten modellieren
6	Lineare Regression - metrisch
7	Lineare Regression - kategorial
8	Fallstudie
9	Vertiefung 1: Textmining oder Clusteranalyse
10	Vertiefung 2: Baumbasierte Verfahren oder Dimensionsreduktion
11	Wiederholung

Tabelle 1 ordnet die Themen des Moduls den Therminen (1-11) zu.

0.3 Vorerfahrung

Bei den Studierenden werden folgende Themen als bekannt vorausgesetzt:

- Deskriptive Statistik
- Inferenzstatistik
- Grundlagen R
- Grundlagen der Datenvisualisierung

0.4 Prüfung

0.4.1 Prüfungshinweise

- Die Prüfung besteht aus zwei Teilen
 - einer Klausur (50% der Teilnote)
 - einer Datenanalyse (50% der Teilnote).

Prüfungsrelevant ist der gesamte Stoff aus dem Skript und dem Unterricht mit folgenden Ausnahmen:

- Inhalte/Abschnitte, die als “nicht klausurrelevant” gekennzeichnet sind,

- Inhalte/Abschnitte, die als “Vertiefung” gekennzeichnet sind,
- Fallstudien (nur für Klausuren nicht prüfungsrelevant),
- die Inhalte von Links,
- die Inhalte von Fußnoten,
- die Kapitel *Vorwort*, *Organisatorisches* und *Anhang*.

Alle Hinweise zur Prüfung gelten nur insoweit nicht anders vom Dozenten festgelegt.

0.4.2 Klausur

- Die Klausur besteht fast oder komplett aus Multiple-Choice (MC)-Aufgaben mit mehreren Antwortoptionen (sofern nicht anders vom Dozenten vorgegeben).
- Die (maximale) Anzahl der richtigen Aussagen ist pro Aufgabe angegeben. Werden mehr Aussagen als “richtig” angekreuzt als angegeben, so wird die Aufgabe mit 0 Punkten beurteilt. Ansonsten werden Teipunkte für jede Aufgabe vergeben.
- Jede Aussage gilt *ceteris paribus* (unter sonst gleichen Umständen). Aussagen der Art “A ist B” (z.B. “Menschen sind sterblich”) sind *nur* dann als richtig auszuwählen, wenn die Aussage *immer* richtig ist.
- Im Zweifel ist eine Aussage auf den Stoff, so wie im Unterricht behandelt, zu beziehen. Werden in Aussagen Zahlen abgefragt, so sind Antworten auch dann richtig, wenn die vorgeschlagene Antwort ab der 1. Dezimale von der wahren Antwort abweicht (einigermaßen genaue Aussagen werden als richtig akzeptiert). Bei Fragen zu R-Syntax spielen Aspekte wie Enter-Taste o.ä. bei der Beantwortung der Frage keine Rolle; diese Aspekte dürfen zu ignorieren.
- Jede Aussage einer MC-Aufgabe ist entweder richtig oder falsch (aber nicht beides oder keines).
- Die MC-Aufgaben sind nur mit Kreuzen zu beantworten; Text wird bei der Korrektur nicht berücksichtigt.
- Bei Nachholklausuren gelten die selben Inhalte (inkl. Schwerpunkte) wie bei der Standard-Klausur, sofern nicht anderweitig angegeben.
- I.d.R. sind nur Klausurpapier und ein nicht-programmierbarer Taschenrechner als Hilfsmittel zulässig.
- Die Musterlösungen zu offenen Fragen sind elektronisch hinterlegt.

0.4.3 Datenanalyse

- Wenden Sie die passenden, im Modul eingeführten statistischen Verfahren an.
- Werten Sie die Daten mit R aus; R-Syntax soll verwendet und im Hauptteil dokumentiert werden.

- In der Wahl des Datensatzes sind Sie frei, mit folgender Ausnahme: Im Unterricht besprochene Datensätze dürfen nicht als Prüfungsleistung eingereicht werden (vgl. Abschnitt 13.4).
- Der (Original-)Name des Datensatzes (sowie ggf. Link) ist bei der Anmeldung anzugeben.
- Gruppenarbeiten sind nicht zulässig.
- Hat sich jemand schon für einen Datensatz angemeldet, so darf dieser Datensatz nicht mehr gewählt werden (“first come, first serve”).
- Fundorte für Datensätze sind z.B. hier³, hier⁴ und hier⁵; im Internet finden sich viele Datensätze⁶.
- Schreiben Sie Ihre Ergebnisse in einer Ausarbeitung zusammen; der Umfang der Ausarbeitung umfasst ca. 1000-1500 Wörter (nur Hauptteil; d.h. exklusive Deckblatt, Verzeichnisse, Anhang etc.).
- Untersuchen Sie 2-3 Hypothesen.
- Denken Sie daran, Name, Matrikelnummer, Modulname etc. anzugeben (Deckblatt). Bei der Gestaltung des Layout entscheiden Sie selbstständig bitte nach Zweckmäßigkeit (und Ästhetik).
- Fügen Sie keine Erklärungen oder Definitionen von statistischen Verfahren an.

0.4.4 Gliederungsvorschlag zur Datenanalyse

1. Datensatz

1. Beschreibung

- Name
- Hintergrund (Themengebiet, Theorien, Relevanz), ca. 100 Wörter
- Dimension (Zeilen*Spalten)
- Zitation (wenn vorhanden)
- sonstige Hinweise (z.B. Datenqualität, Entstehung des Datensatzes)

2. Variablenbeschreibung (nur für Variablen der Hypothese)

- Skalenniveaus
- Kontinuität (nur bei metrischen Variablen)
- R-Datentyp
- Anzahl Fälle und fehlende Werte
- Erläuterung der Variablen

³<http://www.stat.ufl.edu/~winner/datasets.html>

⁴<http://archive.ics.uci.edu/ml/datasets.html>

⁵<http://vincentarelbundock.github.io/Rdatasets/datasets.html>

⁶Googeln Sie mal nach “open datasets” o.ä.

2. Deduktive Analyse

1. Hypothese(n) Beschreiben Sie die Vermutung(en), die Sie prüfen möchten, möglichst exakt.

2. Deskriptive Statistiken

- Berichten Sie deskriptive Statistiken für alle Variablen der Hypothesen.
- Berichten Sie aber nur univariante Statistiken sowie Subgruppenanalysen dazu.
- Berichten Sie ggf. Effektstärken.

3. Diagramme

- Visualisieren Sie Ihre Hypothese(n) bzw. die Daten dazu, gerne aus mehreren Blickwinkeln.

4. Signifikanztest**3. Explorative Analyse**

- Erörtern Sie interessante Einblicke, die über Ihre vorab getroffenen Hypothesen hinausgehen.
- Diagramme können hier eine zentrale Rolle spielen.

4. Diskussion

1. Zentrale Ergebnisse Fassen Sie das zentrale Ergebnisse zusammen.
2. Interpretation Interpretieren Sie die Ergebnisse: Was bedeuten die Zahlen/Fakten, die die Rechnungen ergeben haben?
3. Grenzen der Analyse
 - Schildern Sie etwaige Schwachpunkte oder Einschränkungen der Analyse.
 - Geben Sie Anregungen für weiterführende Analysen dieses Datensatzes.

0.5 Literatur

Zum Bestehen der Prüfung ist keine weitere Literatur formal notwendig; allerdings ist es hilfreich, den Stoff aus unterschiedlichen Blickwinkeln aufzuarbeiten. Dazu ist am ehesten das Buch von Wickham und Grolemund (Wickham und Grolemund 2016) hilfreich, obwohl es deutlich tiefer geht als dieses Skript.

Teil I

Grundlagen

Kapitel 1

Rahmen



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. 1.1):

Datenanalyse, praktisch betrachtet, kann man in fünf Schritte einteilen (Wickham und Grolemund 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 machen; man nennt dies auch *tidy data* (hört sich cooler an). Sobald die Daten in geeigneter

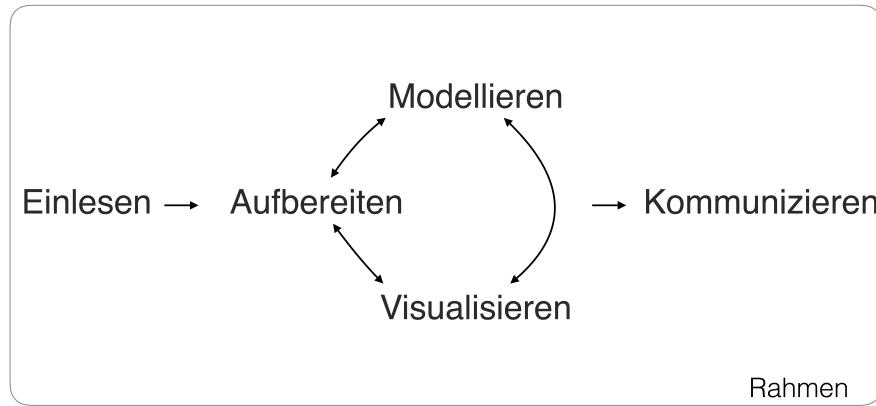


Abbildung 1.1: Der Prozess der Datenanalyse

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.

1.1 Software installieren

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/>.

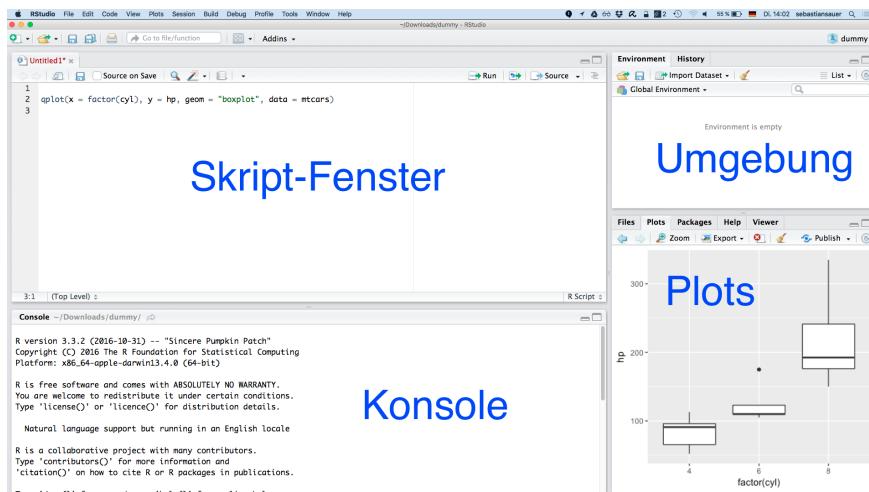
1.1.1 R und RStudio installieren



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.

Die Oberfläche von R, die “Console”, sieht so aus:

Die Oberfläche von RStudio sieht (unter allen Betriebssystemen etwa gleich) so aus:



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.

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.

Wer Shortcuts mag, wird in RStudio überschwänglich beschenkt; der Shortcut für die Shortcuts ist **Shift-Alt-K**.

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.

1.1.2 Sonstiges Material für dieses Skript

Bitte laden Sie sich auch das sonstige Material aus diesem Github-Repositorium¹ herunter:

- Daten (Ordner `data`)
- Liste der benötigten R-Pakete (Datei `Pakete_fuer_PraDa.txt`)

Praktischerweise lädt man im Standard ganze Ordner von Github herunter, so dass man nicht alle Dateien einzeln anpacken muss. Um Download-Zeit zu sparen, sind diese Ordner gezippt. Bevor Sie mit den Dateien arbeiten können, müssen Sie diese erst entzippen².

1.1.3 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")`

¹https://github.com/sebastiansauer/Praxis_der_Datenanalyse/tree/gh-pages

²auf vielen Computern zu bewerkstelligen mit rechter Maustaste/Kontextmenü und dann "Dateien extrahieren" o.ä., sonst Googeln.

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

- `install.packages("derpyler")`
- `install.packages("dplyr") # dependencies vergessen`
- Keine Internet-Verbindung
- `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 Pakete mit `install.packages("name_des_pakets" dependencies = TRUE)` oder mit dem entsprechenden Klick in RStudio.
 - 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.

1.1.3.1 I am an outdated model

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

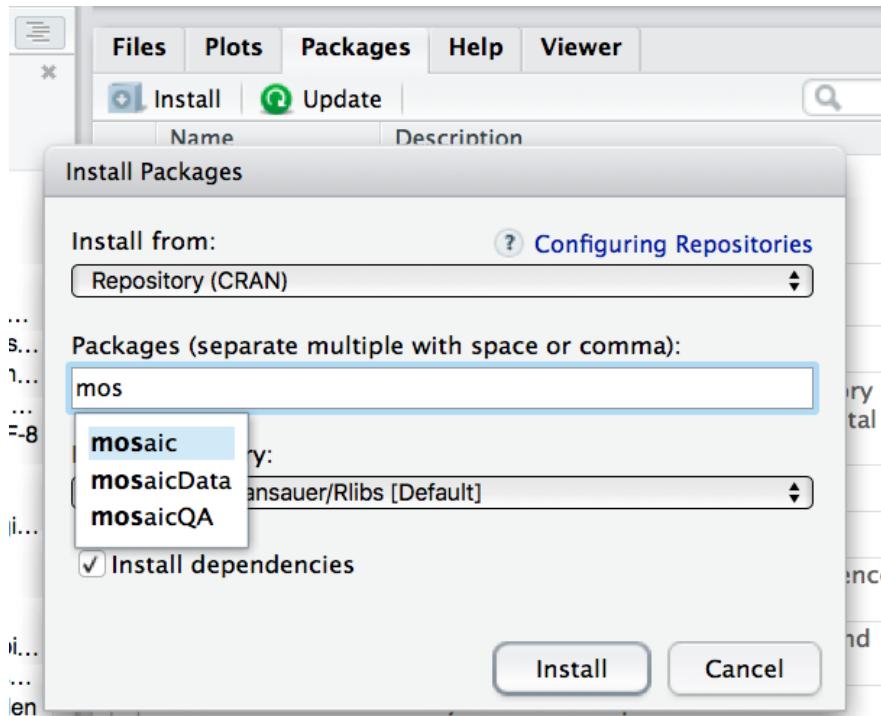


Abbildung 1.2: So installiert man Pakete in RStudio

1.1.4 Pakete

Ein Großteil der Neuentwicklungen bei R passiert in sog. ‘Paketen’ (engl. *packages*), das sind Erweiterungen für R. Jeder, der sich berufen fühlt, kann ein R-Paket schreiben und es zum ‘R-Appstore’ (CRAN⁴) hochladen. Von dort kann es dann frei (frei wie in Bier) heruntergeladen werden.

Am einfachsten installiert man R-Pakete in RStudio über den Button “Install” im Reiter “Packages” (s. Abb. 1.2).

Ein R-Paket, welches für die praktische Datenanalyse praktisch ist, heißt `tidyverse`. Wir werden viel mit diesem Paket arbeiten. Bitte installieren Sie es schon einmal, sofern noch nicht geschehen. Sie können auch folgenden Befehl verwenden, um Pakete zu installieren.

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

Sofern Sie online sind, sollte das Paket `tidyverse` jetzt installiert sein.



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

⁴<https://cran.r-project.org/>

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.

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).

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

```
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. 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')`; geht beides.

Das Installieren und Starten anderer Pakete läuft genauso ab. Am besten installieren Sie alle Pakete, die wir in diesem Buch benötigen auf einmal, dann haben Sie Ruhe (eine schnelle Internetverbindung vorausgesetzt).

1.1.5 R-Pakete für dieses Buch

In diesem Skript verwenden wir die folgenden R-Pakete; diese müssen installiert sein und geladen. Ggf. benötigen Sie Administrator-Rechte, um Pakete zu installieren. Virenscanner müssen evtl. ausgestaltet sein.

```
#> [1] "BaylorEdPsych"   "broom"          "car"            "caret"
#> [5] "cluster"        "corrplot"       "corrr"          "downloader"
#> [9] "dplyr"           "GGally"         "ggplot2"        "grid"
#> [13] "knitr"          "lmtest"         "lsa"            "MBESS"
#> [17] "modelr"         "nycflights13"  "okcupiddata"   "pdftools"
#> [21] "png"             "psych"          "ROCR"           "SDMTools"
#> [25] "SnowballC"      "stringr"        "tidyverse"      "tidytext"
```

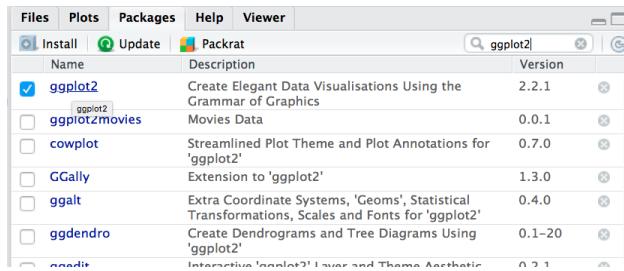


Abbildung 1.3: Hier werden Sie geholfen: Die Dokumentation der R-Pakete

```
#> [29] "tidyverse"      "wordcloud"
```

Mit folgenden Befehlen installieren Sie alle Pakete für diesen Kurs auf einmal. Das ist ganz praktisch, weil Sie ggf. Aktualisierungen bereits installierter Pakete bekommen.

```
load("div/Pakete_fuer_PraDa.Rda")
install.packages(packages)
```

Denken Sie daran, dass dieser Befehl - und alle anderen, die auf Dateien zu diesem Skript zugrifen, davon ausgehen, dass das Arbeitsverzeichnis passend gesetzt ist (vgl. Kapitel 1.2.7). Für jedes Kapitel ist angegeben, welches Kapitels jeweils benötigt d.h. zu laden sind.

1.1.6 Vertiefung: 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 Befehle und Daten aufgeführt (s. Abbildung 1.3). 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 diesen selbstgestrickten Befehl nutzen, den Sie zuerst laden müssen:

```
source("https://sebastiansauer.github.io/Rcode/find_funcs.R")
find_funcs("select")
#> # A tibble: 5 x 3
```

```
#>   package_name builtin_package loaded
#>      <chr>          <lgl>  <lgl>
#> 1    dplyr           FALSE   TRUE
#> 2    MASS            TRUE   FALSE
#> 3    plotly          FALSE   FALSE
#> 4    raster          FALSE   FALSE
#> 5    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.



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`.

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 1.3), dann ist das Paket geladen. Sonst nicht.

1.1.7 Datensätze

Die folgenden Datensätze liegen hier⁸. Bitte laden Sie den Ordner Ordner herunter.

- Datensatz `profiles` aus dem R-Paket `{okcupiddata}` (Kim und Escobedo-Land 2015); es handelt sich um Daten von einer Online-Singlebörsse
- Datensatz `Wage` aus dem R-Paket `{ISLR}` (James, Witten, Hastie, und Tibshirani 2013b); es handelt sich um Gehaltsdaten von US-amerikanischen Männern
- Datensatz `inf_test_short`, URL: <https://osf.io/sjhu> (Sauer 2017a); es handelt sich um Ergebnisse einer Statistikklausur
- Datensatz `flights` aus dem R-Paket `{nycflights13}` (RITA 2013); es handelt sich um Abflüge von den New Yorker Flughäfen

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

⁷es sei denn, sie tun es nicht

⁸https://github.com/sebastiansauer/Praxis_der_Datenanalyse/tree/gh-pages/data

Tabelle 1.1: Wichtige Datentypen in R

Name	Synonyme	Beschreibung
numeric	num, double, dbl	Reelle Zahl (mit Nachkommastellen)
integer	int, L	Ganze Zahl
character	chr, string	Text
logical	lgl, logi	logisch; gibt an, ob ein Ausdruck wahr (TRUE; T) oder falsch (FALSE, F)
factor	fctr	Nominal skalierte Variable mit vorab definierten Ausprägungen; z.B. um
vector	-	Mehrere Elemente eines grundlegenden Typs
dataframe	tibble, df	Tabelle aus einem oder mehr Vektoren, die jeweils einen Namen haben
list	-	Kombination beliebiger Vektoren

- Datensatz 'wo_men', URL: <https://osf.io/ja9dw> (Sauer 2017b); es handelt sich um Körper- und Schuhgröße von Studierenden
- Datensatz **tips** aus dem R-Paket {reshape2} (Bryant und Smith 1995); es handelt sich um Trinkgelder in einem Restaurant
- Datensatz **extra**, URL: <https://osf.io/4kgzh> (Sauer 2016); es handelt sich die Ergebnisse einer Umfrage zu Extraversion
- Datensatz **Werte** URL: <https://osf.io/4kgzh/> (Gansser 2017); es handelt sich um Werte von Studierenden
- Datensatz **Segment** URL: <https://goo.gl/eUm8PI> (Chapman und Feit 2015); es handelt sich um soziodemographische Daten (simuliert)

Wie man Daten in R ‘einlädt’ (Studierende sagen gerne ‘ins R hochladen’), besprechen wir im Kapitel 7.1.

1.2 ERRRstkontakt

1.2.1 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”.

1.2.2 Datentypen in R

Die (für diesen Kurs) wichtigsten Datentypen von R sind in Tabelle 1.1 aufgeführt.

Für die praktische Datenanalyse ist der `dataframe` (Dataframe) am wichtigsten. Grob gesagt handelt es sich dabei um eine Tabelle, wie man sie aus Excel kennt. Etwas genauer ist eine Kombination von Vektoren mit gleicher Länge, so dass eine ‘rechteckige’ Datenstruktur entsteht. Alle Spalten (d.h. Vektoren) haben einen Namen, so dass es ‘Spaltenköpfe’ gibt. Eine neuere Variante von Dataframes sind ‘tibbles’, die *auch* Dataframes sind, aber ein paar praktische Zusatzeigenschaften aufweisen.

1.2.3 Hinweise

Unser erster Kontakt mit R! Ein paar Anmerkungen vorweg:

- 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 von R dann ignoriert.
- *Variablennamen* in R müssen mit Buchstaben beginnen; ansonsten dürfen nur Zahlen, Unterstriche – und Minuszeichen – enthalten sein. Leerzeichen sollte man meiden.
- Variablen einen Namen zu geben, ist nicht leicht, aber wichtig. Namen sollten knapp, aber aussagekräftig sein.

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

# gut:
tips_mw
lm1
```

Um den Inhalt einer Variablen auszulesen, geben wir einfach den Namen des Objekts ein (und schicken den Befehl ab).

1.2.4 R als Taschenrechner

Auch wenn Statistik nicht Mathe ist, so kann man mit R auch rechnen. Geben Sie zum Üben die Befehle in der R Konsole hinter der Eingabeaufforderung `>` ein und beenden Sie die Eingabe mit `Return` bzw. `Enter`.

```
4+2
#> [1] 6
```

Das Ergebnis wird direkt angezeigt. Bei

```
x <- 4+2
```

erscheint zunächst kein Ergebnis. Über `<-` wird der Variable `x` der Wert `4+2` zugewiesen. Wenn Sie jetzt

```
x
```

eingeben, wird das Ergebnis

```
#> [1] 6
```

angezeigt. Sie können jetzt auch mit `x` weiterrechnen, z.B.:

```
x/4  
#> [1] 1.5
```

Vielleicht fragen Sie sich was die `[1]` vor dem Ergebnis bedeutet. R arbeitet vektororientiert, und die `[1]` zeigt an, dass es sich um das erste (und hier auch letzte) Element des Vektors handelt.

1.2.5 Text und Variablen zuweisen

Man kann einer Variablen auch Text zuweisen (im Gegensatz zu Zahlen):

```
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.

Man kann auch einer Variablen *mehr als* einen Wert zuweisen:

```
x <- c(1, 2, 3)
```

Dieser Befehl erzeugt eine “Spalte” (einen Vektor). Will man einer Variablen *mehr als* einen Wert zuweisen, muss man die Werte erst in einen Vektor “zusammen binden”; das geht mit dem Befehl `c` (vom engl. “*combine*”).

1.2.6 Funktionen aufrufen

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

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

Allgemein gesprochen:

```
funktionsname(parametername1 = wert1, parametername2 = wert2, ...)
```

Die drei Punkte ... sollen andeuten, dass evtl. weitere Parameter zu übergeben wären. Die Reihenfolge der Parameter ist *egal* - wenn man die Parameternamen anführt. Ansonsten muss man sich an die Standard-Reihenfolge, die eine Funktion vorgibt halten:

```
#ok:
wo_men <- read.csv(file = "data/wo_men.csv", header = TRUE, sep = ",")
wo_men <- read.csv("data/wo_men.csv", TRUE, ",")
wo_men <- read.csv(header = TRUE, sep = ",", file = "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 Parametern. 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.

1.2.7 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. 1.4. 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("daten_deutsch.csv", sep = ";", dec = ".")
```

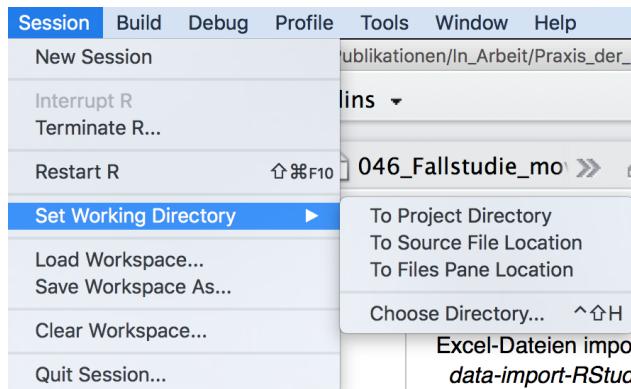


Abbildung 1.4: Das Arbeitsverzeichnis mit RStudio auswählen

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

Für diesen Kurs ist es sinnvoll, das Arbeitsverzeichnis in einen “Hauptordner” zu legen (z.B. “Praxis_der_Datenanalyse”), 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.

1.2.8 Hier werden Sie geholfen

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 (d.i. Funktion) können Sie mit ? Hilfe erhalten; probieren Sie z.B. `?mean`.
- 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>.

1.2.9 Aufgaben

1. Öffnen Sie das Cheatsheet für RStudio und machen Sie sich mit dem Cheatsheet vertraut.
2. Sichten Sie kurz die übrigen Cheatsheets; später werden die Ihnen vielleicht von Nutzen sein.
3. Führen Sie diese Syntax aus:

```
meine_coole_variable <- 10
meine_coole_var1able
```

Woher röhrt der Fehler?

4. Korrigieren Sie die Syntax:

```
install.packages(dplyr)
```

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

```
Hallo R <- 1
```

```
Hallo_R <- 1
```

1.3 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. 1.5). 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 der Inferenzstatistik ist es, zu prüfen, ob Daten einer Stichprobe auf eine Grundgesamtheit verallgemeinert werden können.

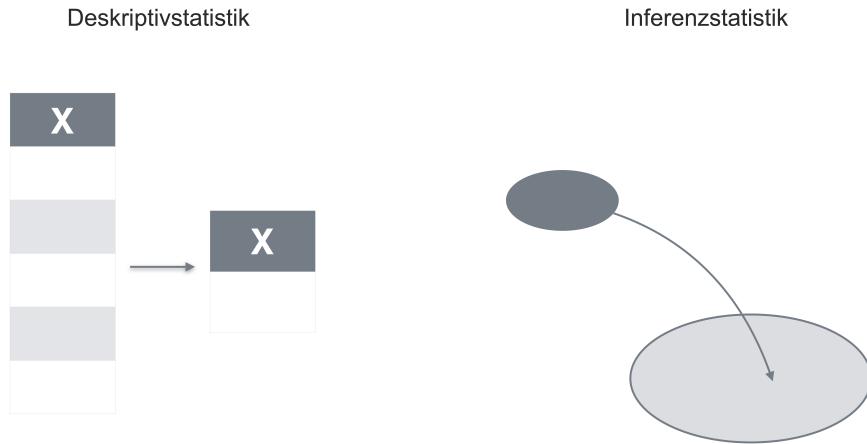


Abbildung 1.5: Sinnbild für die Deskriptiv- und die Inferenzstatistik

Dabei lässt sich der Begriff “Statistik” als Überbegriff von “Datenanalyse” verstehen, wenn diese Sicht auch nicht von allen geteilt wird (Grolmünd 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 Grolmünd 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; Grolmünd 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. 8.

Statistisches Modellieren läuft gewöhnlich nach folgendem Muster ab (Grolmünd 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. Darüber 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 sein⁹. 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.

1.4 Befehlsübersicht

Tabelle 1.2 stellt die Befehle dieses Kapitels dar.

Diese Befehle "wohnen" alle im Standard-R; es ist für diese Befehle nicht nötig, zusätzliche Pakete zu installieren/ laden.

⁹Sind z.B. die Prüfungsergebnisse von Schülern unabhängig voneinander? Möglicherweise haben sie von einem "Superschüler" abgeschrieben. Wenn der Superschüler viel weiß, dann zeigen die Abschreiber auch gute Leistung.

Tabelle 1.2: Befehle des Kapitels 'Rahmen'

Paket::Funktion	Beschreibung
install.packages("x")	Installiert Paket "x" (nicht: Paket "X")
library	lädt ein Paket
<-	Weist einer Variablen einen Wert zu
c	erstellt eine Spalte/ einen Vektor

1.5 Verweise

- Chester Ismay erläutert einige Grundlagen von R und RStudio, die für Datenanalyse hilfreich sind: <https://bookdown.org/chesterismay/rbasics/>.
- Roger Peng und Kollegen bieten hier einen Einstieg in Data Science mit R: <https://bookdown.org/rdpeng/artofdatascience/>
- Wickam und Golemund (2016) geben einen hervorragenden Überblick in das Thema dieses Buches; ihr Buch ist sehr zu empfehlen.
- Wer einen stärker an der Statistik orientierten Zugang sucht, aber “mathematisch sanft” behandelt werden möchte, wird bei James et al. (2013b) glücklich oder zumindest fündig werden.

Kapitel 2

Daten einlesen



Lernziele:

- 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(readr) # Daten einlesen
```

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

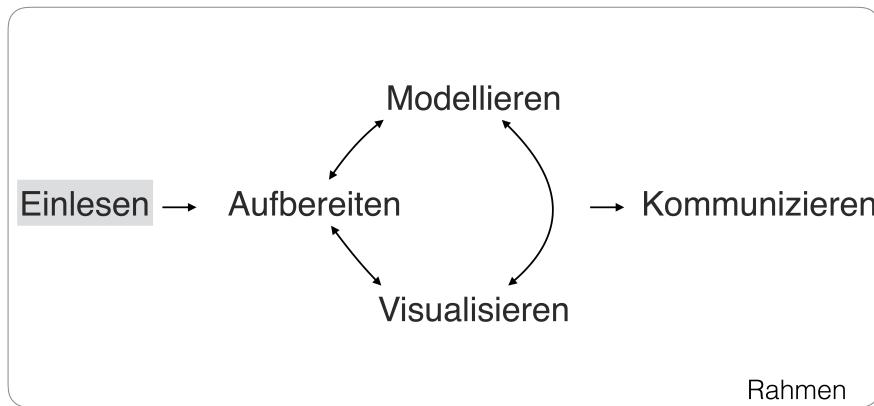


Abbildung 2.1: Daten sauber einlesen

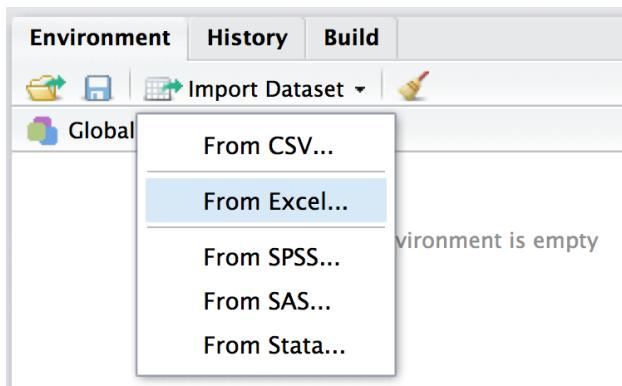


Abbildung 2.2: Daten einlesen (importieren) mit RStudio

2.1 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.

2.1.1 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.

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

“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.

2.1.2 CSV-Dateien importieren

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:

```
df <- read.csv("data/test_inf_short.csv")
```

2.1.2.1 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 exportie-

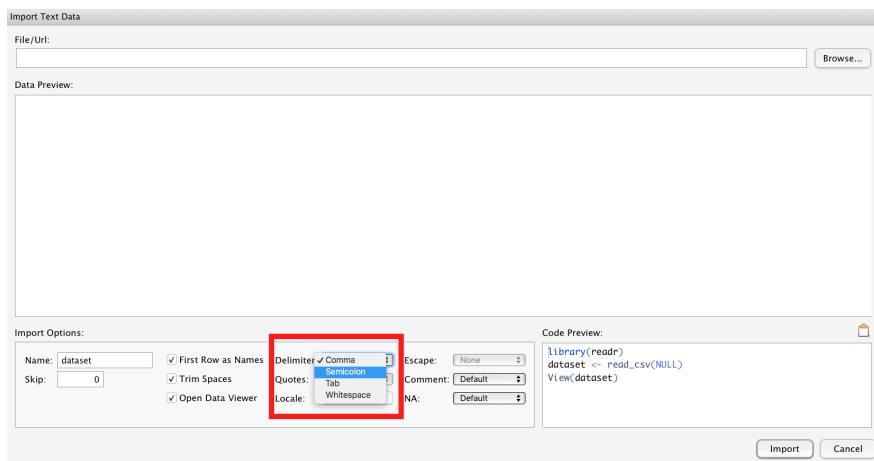


Abbildung 2.3: Trennzeichen einer CSV-Datei in RStudio einstellen

ren, wird diese CSV-Datei als Spaltentrennung ; (Strichpunkt) und als Dezimaltrennzeichen , verwenden. Da der Befehl `read.csv` laut amerikanischen Standard mit Komma als Spalten trennung und Punkt als Dezimaltrennzeichen arbeitet, müssen wir die deutschen Sonderzeichen explizit angeben, z.B. so:

```
# nicht ausführen:
daten_deutsch <- read.csv("daten_deutsch.csv", sep = ";", dec = ".")
```

Dabei steht `sep` (separator) für das Trennzeichen zwischen den Spalten und `dec` 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 Auswahlbutton “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.

2.2 Normalform einer Tabelle

Tabellen in R werden als `data frames` (“Dataframe” auf Englisch; 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.

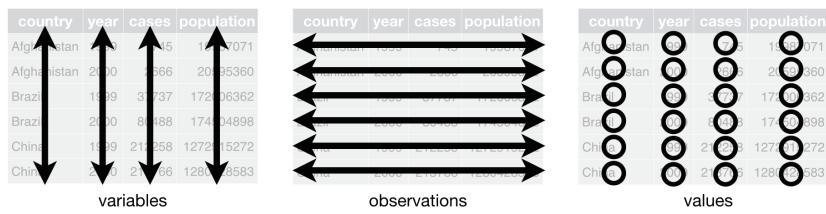


Abbildung 2.4: Schematische Darstellung eines Dataframes in Normalform

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

Breit

ID	Quartal	Umsatz
1	Q1	342
2	Q2	342
3	...	124
...	Q1	342
	Q2	342
	Q3	124
	...	

Lang

Abbildung 2.5: Dieselben Daten - einmal breit, einmal lang

- 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 2.4 visualisiert die Bestimmungsstücke eines Dataframes (Wickham und Grolemund 2016):

Der Punkt *Jede Zeile eine Beobachtung, jede Spalte eine Variable, jede Zelle ein Wert* verdient besondere Beachtung. Betrachten Sie dieses Beispiel:

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.

Datensatz (Normalform)

in Zeilen: Fall/ Beobachtung
(häufig Personen)

in Spalten:
Wert/ Ausprägung
Merkmal/ Variable

<i>ID</i>	<i>age</i>	<i>sex</i>	<i>n_FB_friends</i>
Anna	21	female	212
Berta	24	female	235
Carla	20	male	312
Dora	20	female	21435

Abbildung 2.6: Illustration eines Datensatzes in Normalform

2.3 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. 2.7 zeigt ein Beispiel dazu.

Warum ist es wichtig, von der “breiten” (links in Abb. 2.7) zur “langen” oder “Normalform” (rechts in Abb. 2.7) 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. 2.8).

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")

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

Dabei baut `gather` den Dataframe so um, dass nur zwei Spalten übrig bleiben (s. Abb. 2.7). Eine Spalte nur *Spaltennamen* (“Q1”, “Q2”, ...) enthält; diese Spalte nennt `gather` im

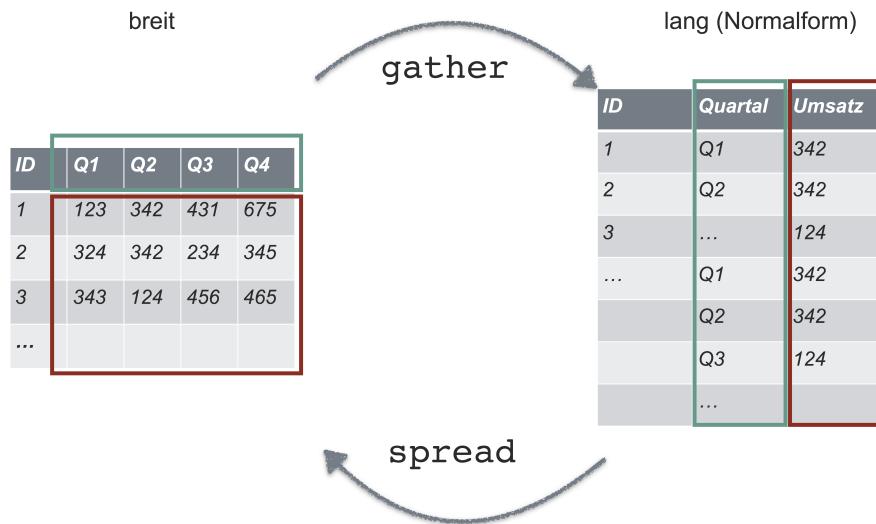


Abbildung 2.7: Mit 'gather' und 'spread' wechselt man von der breiten Form zur langen Form

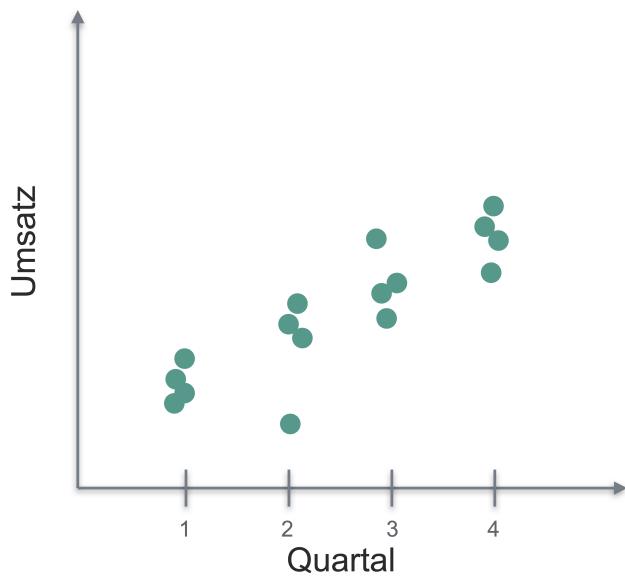


Abbildung 2.8: Ein Beispiel für eine Abbildung zu einer Normalform-Tabelle

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)
```

In Kapitel 5 werden wir dazu ein Fallstudie einüben.

Tabelle 2.1: Befehle des Kapitels 'Daten einlesen'

Paket::Funktion	Beschreibung
read.csv	Liest eine CSV-Datei ein.
write.csv	Schreibt einen Dateframe in eine CSV-Datei.
tidy::gather	Macht aus einem "breiten" Dataframe einen "langen".
tidy::separate	"Zieht" Spalten auseinander.

2.4 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. Ein bestimmtes Zeichen wie "A" bekommt einen bestimmten Code wie "41". Mit etwas Glück weiß der Computer jetzt, dass er das Zeichen "41" auf den Bildschirm ausgeben soll. Es stellt sich jetzt die Frage, welche Code-Tabelle der Computer nutzt? Welchem Code wird "A" (bzw. ein beliebiges Zeichen) zugeordnet? 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.

2.5 Befehlsübersicht

Tabelle 2.1 stellt die Befehle dieses Kapitels dar.

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

2.6 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.

2.7 Verweise

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

⁵F, R, F, R, R, R, F, F

Kapitel 3

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`
```

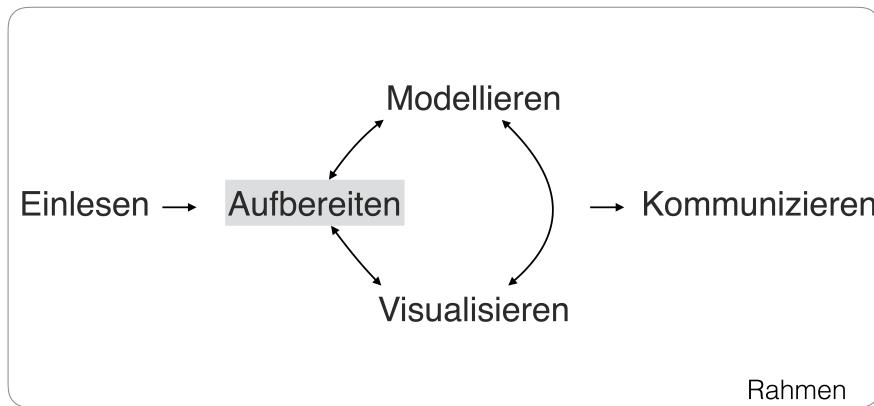


Abbildung 3.1: Daten aufbereiten

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 Quellenangebe, 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.

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

3.1 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:

- *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.

3.2 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).

3.2.1 Die zwei Prinzipien von `dplyr`

Es gibt viele Möglichkeiten, Daten mit R aufzubereiten; `dplyr`² ist ein populäres Paket dafür. `dplyr` basiert auf zwei Ideen:

1. *Lego-Prinzip* Komplexe Datenanalysen in Bausteine zerlegen (vgl. Abb. 3.2).
2. *Durchpfeifen*: Alle Operationen werden nur auf Dataframes angewendet; jede Operation erwartet einen Dataframe als Eingabe und gibt wieder einen Dataframe aus (vgl. Abb. 3.3).

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

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

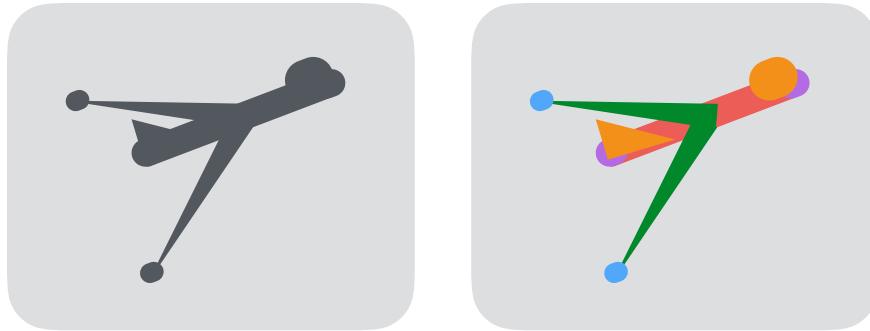


Abbildung 3.2: Lego-Prinzip: Zerlege eine komplexe Struktur in einfache Bausteine

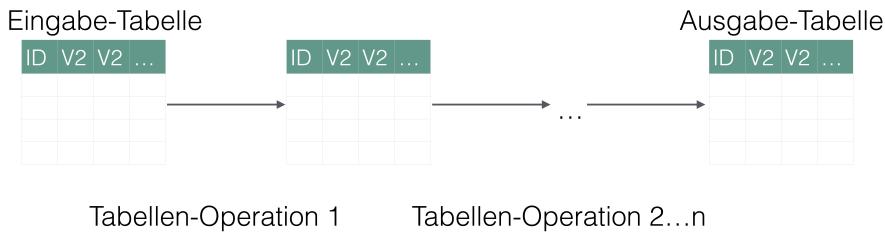


Abbildung 3.3: Durchpfeifen: Ein Dataframe wird von Operation zu Operation weitergereicht

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 *durchzureichen*. `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`.

3.3 Zentrale Bausteine von `dplyr`

3.3.1 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.

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 3.4: Zeilen filtern

Abb. 3.4 zeigt ein Sinnbild für `filter`.

Merke:

Die Funktion `filter` filtert Zeilen aus einem Dataframe.

Schauen wir uns einige Beispiel 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
```

```
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_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.



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(...)`.

3.3.1.1 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")`.

3.3.1.2 Vertiefung: 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’.

³F, R, F, F, R

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

Abbildung 3.5: Spalten auswählen

3.3.2 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. 3.5 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/test_inf_short.csv")
```

Dieser Datensatz beinhaltet Daten zu einer Statistikklausur.

Beachten Sie, dass diese Syntax davon ausgeht, dass sich die Daten in einem Unterordner mit dem Namen `data` befinden, welcher sich im Arbeitsverzeichnis befindet⁴.

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

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

⁴der angegebene Pfad ist also *relativ* zum aktuellen Verzeichnis.

```
select(stats_test, score:study_time) # dito
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`-Cheatsheet findet sich ein guter Überblick dazu.

3.3.2.1 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.

3.3.3 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:

⁵F, F, R, R, F

```

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
#> 2       4 06.01.2017 09:58:05     nein        2        3        2
#> 3      131 19.01.2017 18:03:45     nein        2        3        4
#> 4      142 19.01.2017 19:02:12     nein        3        4        1
#> 5       35 12.01.2017 19:04:43     nein        1        2        3
#> 6       71 15.01.2017 15:03:29     nein        3        3        3
#>   score
#> 1    17
#> 2    18
#> 3    18
#> 4    18
#> 5    19
#> 6    20
#>   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
#> 3      29 12.01.2017 09:48:16     ja         4       10        3
#> 4      41 13.01.2017 12:07:29     ja         4       10        3
#> 5      58 14.01.2017 15:43:01     ja         3        8        2
#> 6      83 16.01.2017 10:16:52     ja        NA       NA       NA
#>   score
#> 1    40
#> 2    40
#> 3    40
#> 4    40
#> 5    40
#> 6    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

The diagram shows two tables side-by-side. The left table has columns ID, Name, and Note1. The right table also 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 between them.

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 3.6: Spalten sortieren

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 3.6) erläutert die Arbeitsweise von `arrange`.

Merke:

Die Funktion `arrange` sortiert die Zeilen eines Datafames.

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
#> 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”.

3.3.3.1 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.

3.3.4 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).

⁶F, F, F, F, F

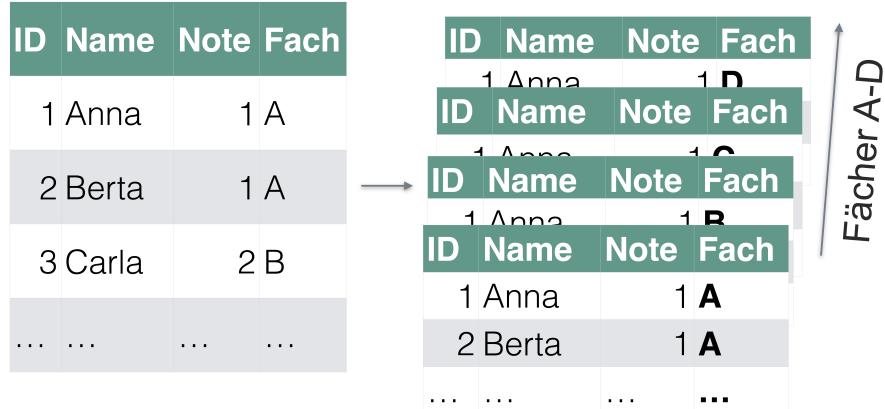


Abbildung 3.7: Datensätze nach Subgruppen aufteilen

In Abbildung 3.7 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).

```
test_gruppiert <- group_by(stats_test, interest)
test_gruppiert
#> Source: local data frame [306 x 7]
#> Groups: interest [7]
#>
#> # A tibble: 306 x 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

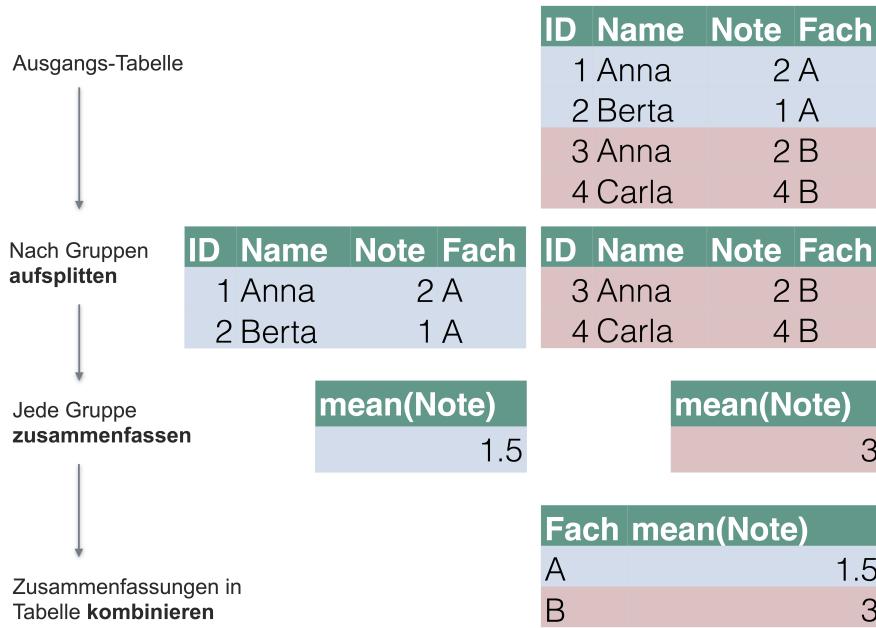


Abbildung 3.8: Schematische Darstellung des 'Gruppieren - Zusammenfassen - Kombinieren'

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. 3.8).

3.3.4.1 Aufgaben⁸



Richtig oder Falsch!?

1. Mit `group_by` gruppiert 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.

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

⁸R, F, R, R



Abbildung 3.9: Spalten zu einer Zahl zusammenfassen

Merke:

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

3.3.5 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. 3.9). 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 heißt 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.

3.3.5.1 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.
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
```

3.3.6 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.

⁹R, R, R, R, R

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

```

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
#> 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
dplyr::count(stats_test, study_time)
#> # A tibble: 6 x 2
#>   study_time     n
#>   <int> <int>
#> 1      1    31
#> 2      2    49
#> 3      3    85
#> 4      4    56
```

Gruppe A Gruppe B Gruppe C

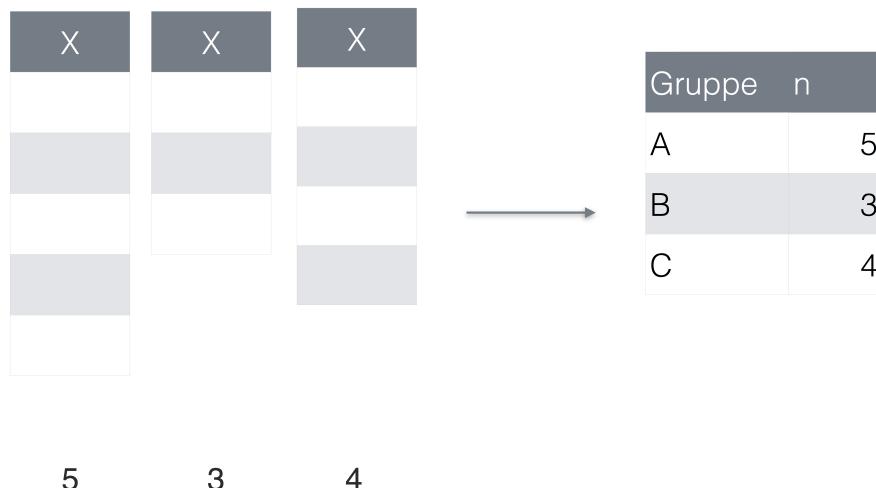


Abbildung 3.10: Sinnbild für 'count'

```
#> 5      5    17
#> 6     NA   68
dplyr::count(stats_test, interest, study_time)
#> Source: local data frame [29 x 3]
#> Groups: interest [?]
#>
#> # A tibble: 29 x 3
#>   interest study_time     n
#>   <int>     <int> <int>
#> 1 1         1       12
#> 2 1         2       7
#> 3 1         3       8
#> 4 1         4       2
#> 5 1         5       1
#> 6 2         1       9
#> 7 2         2      15
#> 8 2         3      16
#> 9 2         4       6
#> 10 2        5       1
#> # ... with 19 more rows
```

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. 3.10).

Merke:

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

3.3.6.1 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: Wieviel Prozent der Fälle sind Frauen, wie viele sind Männer?

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))
#> Source: local data frame [14 x 4]
#> Groups: interest [7]
#>
#> # A tibble: 14 x 4
#>   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))
```

3.3.6.2 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!

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

¹¹R, R, F, F



Abbildung 3.11: Das ist keine Pfeife

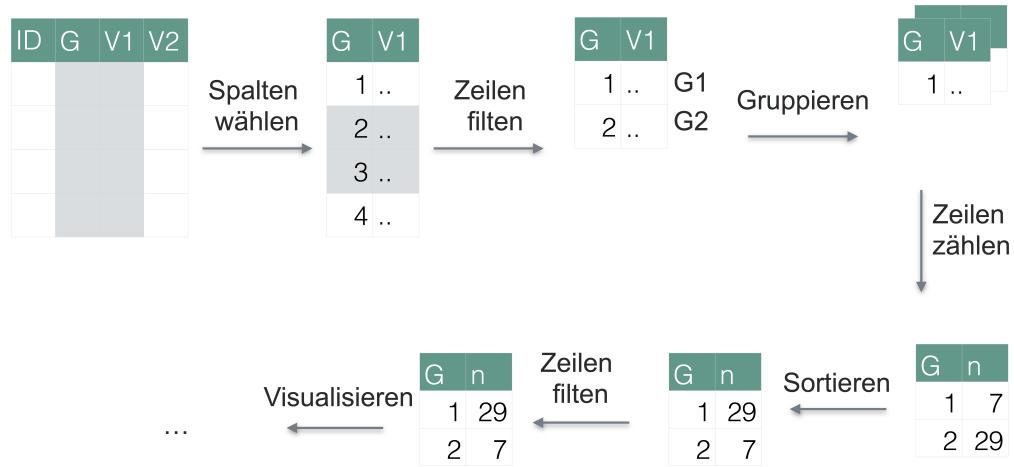


Abbildung 3.12: Das 'Durchpeifen'

Ah! Der Score 34 ist der häufigste!

3.4 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. Das berühmte Bild von René Magritte stand dabei Pate (s. Abb. 3.11; (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. 3.12).

Die sog. “Pfeife” (pipe: `%>%`) in Anspielung an das berühmte Bild von René Magritte, verkettet

Befehle hintereinander. Das ist praktisch, da es die Syntax vereinfacht.



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.

3.4.1 Spalten berechnen mit `mutate`

Wenn man die Pfeife benutzt, ist der Befehl `mutate` ganz praktisch: Er berechnet eine Spalte. Normalerweise kann man einfach eine Spalte berechnen mit dem Zuweisungsoperator:

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:



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
```

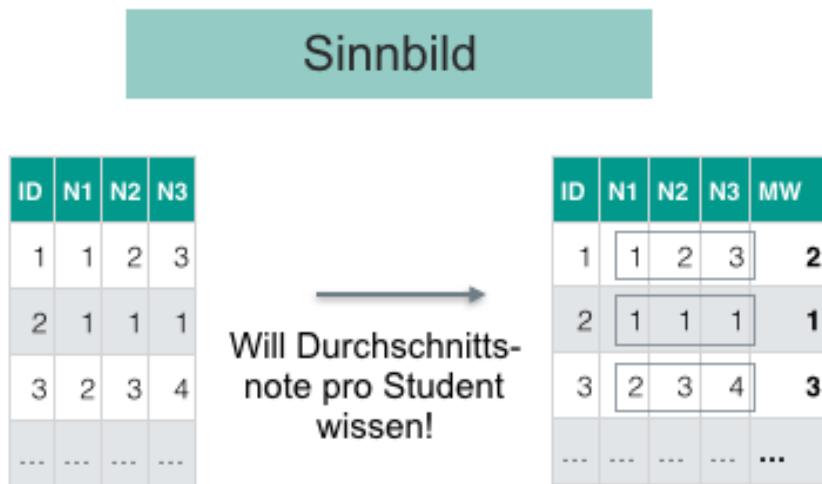


Abbildung 3.13: Sinnbild für mutate

```
#> 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. 3.13 zeigt Sinnbild für `mutate`:



`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*.

3.4.2 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"))
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())
#> # A tibble: 0 x 3
#> # ... with 3 variables: interest <int>, Punkte <dbl>, n <int>
```

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))
#>   sqrt(score_var)
#> 1          5.73
```

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, durchgepfiffenen 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

3.5 Deskriptive Statistik

`dplyr` kann man gut gebrauchen, um deskriptive Statistik zu berechnen. `summarise` charakterisiert 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* (`Md`; `median`). 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.

¹²`sd(stats_test$score)`

¹³die sd von `score`

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(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.

`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.

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

```
stats_test2 <- 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 <- 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.

3.5.0.1 Vertiefung - eigene Statistikauswahl bei `desctable`

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

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

desctable(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:

¹⁶Die Idee kommt von Norman Markgraf

Tabelle 3.1: Befehle des Kapitels 'Datenjudo'

Paket::Funktion	Beschreibung
dplyr::arrange	Sortiert Spalten
dplyr::filter	Filtert Zeilen
dplyr::select	Wählt Spalten
dplyr::group_by	gruppiert einen Dataframe
dplyr::n	zählt Zeilen
dplyr::count	zählt Zeilen nach Untergruppen
%>% (dplyr)	verkettet Befehle
dplyr::mutate	erzeugt/berechnet Spalten
desctable::desctable	Liefert Tabelle mit deskriptiver Statistik zurück

```
stats_test %>%
  select(-c(row_number, date_time)) %>%
  group_by(bestanden) %>%
  desctable
```

3.6 Befehlsübersicht

Tabelle 3.1 fasst die R-Funktionen dieses Kapitels zusammen.

3.7 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/2kX9lvC>.
- 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 4

Praxisprobleme der Datenaufbereitung



Lernziele:

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

Laden wir zuerst die benötigten Pakete; v.a. ist das `dplyr` and friends. Das geht mit dem Paket `tidyverse`.

```
library(tidyverse)
library(corr)
library(gridExtra)
library(car)
```

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.

4.1 Datenaufbereitung

4.1.1 Auf fehlende Werte prüfen

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

```
stats_test <- read.csv("data/test_inf_short.csv")
summarise(stats_test)
#> data frame with 0 columns and 0 rows
```

4.1.2 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] 306

# Nach Umwandlung in neuen Dataframe
stats_test %>%
  na.omit -> stats_test_na OMIT
nrow(stats_test_na OMIT)
#> [1] 238

# Nur die Anzahl der bereinigten Daten
stats_test %>%
  na.omit %>%
  nrow
#> [1] 238
```



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 68 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
#>   row_number      date_time study_time self_eval interest score
```

#> 1	6 06.01.2017 14:21:18	NA	NA	NA	39
#> 2	7 06.01.2017 14:25:49	NA	NA	NA	40
#> 3	15 09.01.2017 15:23:15	NA	NA	NA	30
#> 4	19 10.01.2017 17:16:48	NA	NA	NA	22
#> 5	42 13.01.2017 14:08:08	NA	NA	NA	38
#> 6	49 14.01.2017 07:02:39	NA	NA	NA	39

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

4.1.3 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$interest <- replace(stats_test$interest, is.na(stats_test$interest),
                                mean(stats_test$interest, na.rm = TRUE))

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

replace² ersetzt Werte aus dem Vektor stats_test\$interest alle Werte, für die is.na(stats_test\$interest) wahr ist, bei Zeilen mit fehlenden Werten in dieser Spalte also. Diese Werte werden durch den Mittelwert der Spalte ersetzt³.

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

4.1.4 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 %>%
  count(interest)
#> # A tibble: 7 x 2
#>   interest     n
#>   <dbl> <int>
#> 1     1.00    30
#> 2     2.00    47
#> 3     3.00    66
#> 4     3.21    68
#> 5     4.00    41
#> 6     5.00    45
#> 7     6.00     9
```

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.

4.1.5 Ausreißer identifizieren

Ähnlich zu Fehlern, steht man Ausreißer häufig skeptisch gegenüber. Allerdings kann man nicht pauschal sagen, dass Extremwerte entfernt werden sollen: Vielleicht war jemand in der 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. 4.1).

```
qplot(x = score, data = stats_test, binwidth = 1)
```

Mit `binwidth = 1` sagen wir, dass jeder Balken (bin) eine Breite (width) von 1 haben soll.

4.1.6 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.

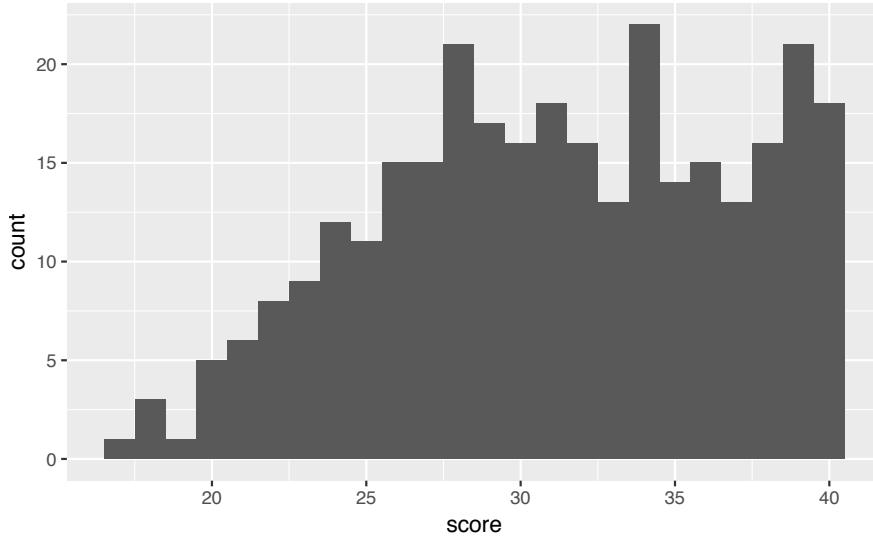


Abbildung 4.1: Ausreißer identifizieren

Nehmen wir dazu den Datensatz `extra` her.

```
extra <- read.csv("data/extradata.csv")
```

```
extra %>%
  select(i01:i10) %>% # Wähle die Variablen von i01 bis i10 aus
  correlate() -> km    # Korrelationsmatrix berechnen
km
#> # A tibble: 10 x 11
#>   rowname     i01    i02r    i03    i04    i05    i06r    i07    i08    i09
#>   <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#> 1 i01        NA 0.4895 0.0805 0.4528 0.4481 0.450 0.309 0.387 0.3795
#> 2 i02r      0.4895      NA 0.0849 0.3603 0.3897 0.520 0.240 0.323 0.2730
#> 3 i03        0.0805 0.0849      NA 0.0323 0.0492 0.155 0.156 0.101 0.0211
#> 4 i04        0.4528 0.3603 0.0323      NA 0.6478 0.316 0.446 0.219 0.2472
#> 5 i05        0.4481 0.3897 0.0492 0.6478      NA 0.348 0.395 0.287 0.2983
#> 6 i06r      0.4504 0.5197 0.1554 0.3163 0.3482      NA 0.163 0.294 0.2937
#> 7 i07        0.3090 0.2396 0.1557 0.4459 0.3949 0.163      NA 0.317 0.2803
#> 8 i08        0.3873 0.3232 0.1006 0.2190 0.2875 0.294 0.317      NA 0.4095
#> 9 i09        0.3795 0.2730 0.0211 0.2472 0.2983 0.294 0.280 0.409      NA
#> 10 i10       0.1850 0.0789 0.0939 0.3520 0.2929 0.136 0.380 0.220 0.1552
#> # ... with 1 more variables: i10 <dbl>
```

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

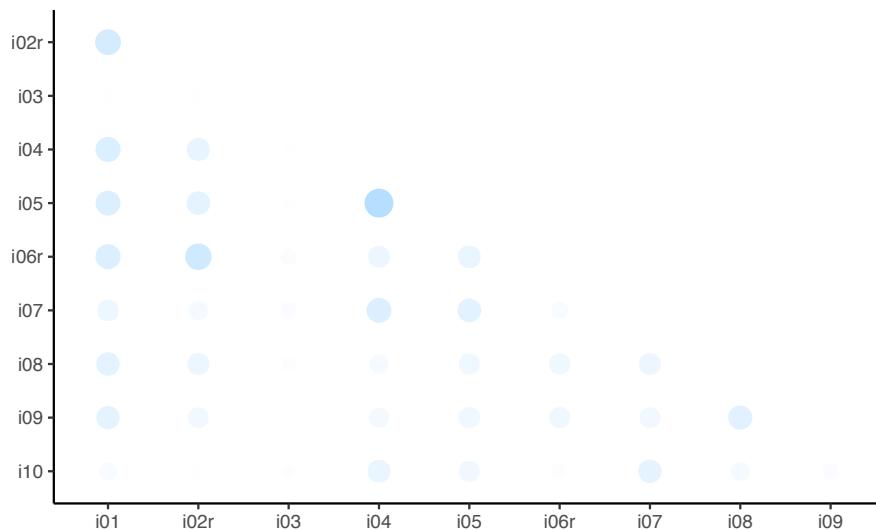


Abbildung 4.2: Ein Korrelationsplot

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

Die Funktion `correlate` stammt aus dem Paket `corr4`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.

4.1.7 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 %>%
  select(i01, i02r) %>%
  scale() %>% # z-standardisieren
  head() # nur die ersten paar Zeilen abdrucken
#>      i01  i02r
```

⁴<https://github.com/drsimonj/corr4>

```
#> [1,] -0.499 -0.15
#> [2,] -1.964 -1.39
#> [3,] -0.499  1.09
#> [4,] -0.499 -0.15
#> [5,]  0.966 -0.15
#> [6,] -0.499 -1.39
```

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)) %>%
  head
```

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.

4.1.8 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. 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$n_facebook_friends, na.rm = TRUE)  # keine Konstante
#> [1] 288
n_distinct(extra$sex)  # es scheint 3 Geschlechter zu geben...
#> [1] 3
```

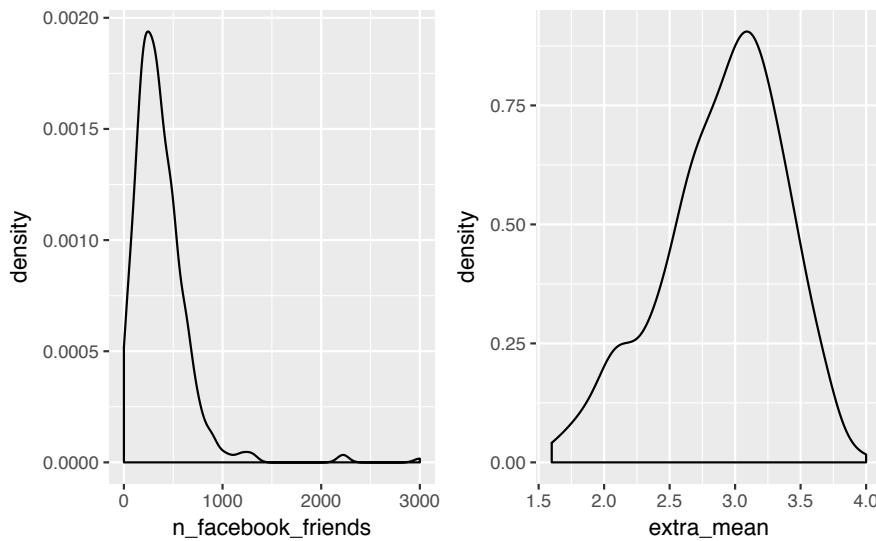


Abbildung 4.3: Visuelles Prüfen der Normalverteilung

4.1.9 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. 4.3).

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.

4.1.10 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 echt ziemlich sicher, dass ich 1 für Männer kodiert habe, wahrscheinlich...”). Besser wäre es, die Ausprägungen `male` und `female` (“Mann”, “Frau”) o.ä. zu verwenden (vgl. Abb. 4.4).

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. 4.5).

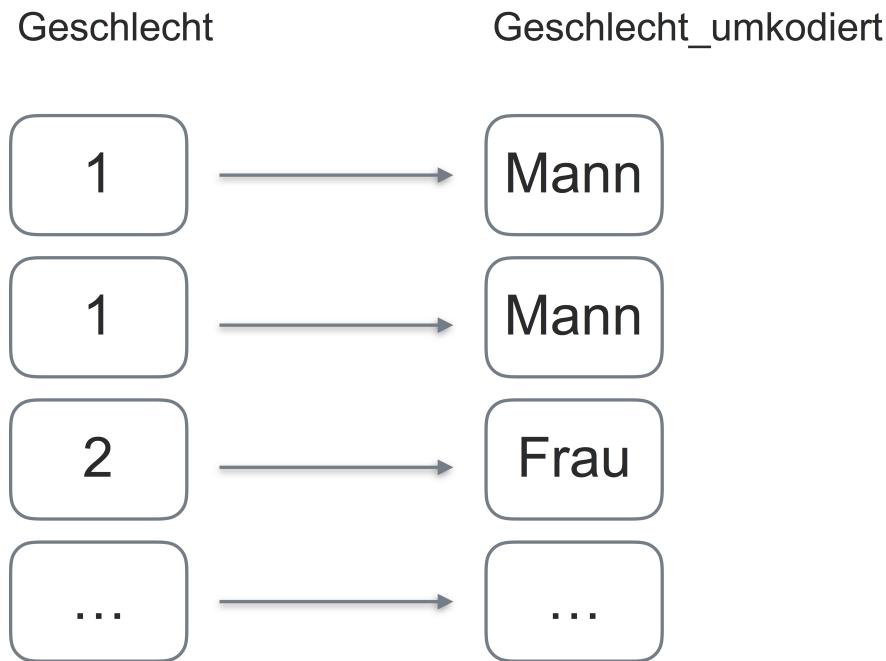


Abbildung 4.4: Sinnbild für Umkodieren

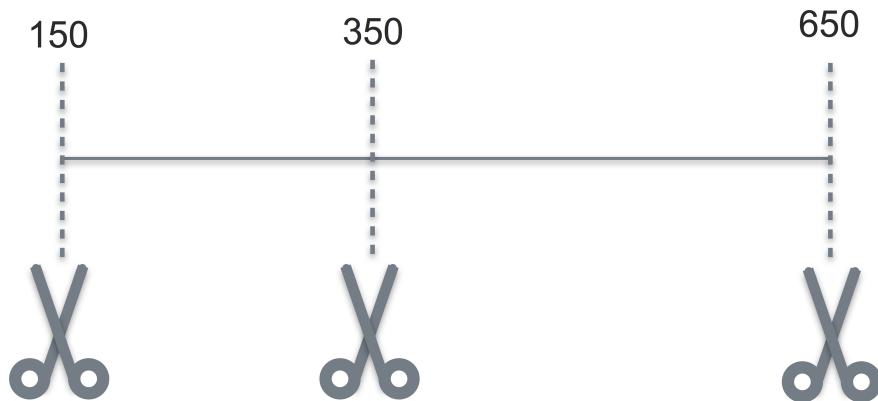


Abbildung 4.5: Sinnbild zum 'Binnen'

4.1.10.1 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 <- read.csv("data/test_inf_short.csv")

stats_test$score_fac <- car::recode(stats_test$study_time,
                                     "5 = 'sehr viel'; 2:4 = 'mittel'; 1 = 'wenig'",
```

```

            as.factor.result = TRUE)
stats_test$score_fac <- car::recode(stats_test$study_time,
                                      "5 = 'sehr viel'; 2:4 = 'mittel'; 1 = 'wenig'",
                                      as.factor.result = FALSE)

stats_test$study_time_2 <- car::recode(stats_test$study_time,
                                         "5 = 'sehr viel'; 4 = 'wenig';
                                         else = 'Hilfe'", 
                                         as.factor.result = TRUE)

head(stats_test$study_time_2)
#> [1] sehr viel Hilfe      sehr viel Hilfe      wenig       Hilfe
#> Levels: Hilfe sehr viel wenig

```

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$no_interest <- car::recode(stats_test$interest,
                                         "1 = 6; 2 = 5; 3 = 4; 4 = 3;
                                         5 = 2; 6 = 1; else = NA")
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)
#> # 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
dplyr::count(stats_test, no_interest)
#> # A tibble: 7 x 2
#>   no_interest     n
#>   <dbl> <int>
#> 1             1     9
#> 2             2    45
#> 3             3    41
#> 4             4    66
#> 5             5    47
#> 6             6    30
#> 7             NA   68
```

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

```
stats_test$Ergebnis <- car::recode(stats_test$score,
                                      "1:38 = 'durchgefallen'";
                                      else = 'bestanden'")
```

Natürlich gibt es auch eine Pfeifen kompatible Version, um Variablen umzukodieren bzw. zu binnern: `dplyr::recode`⁵. Die Syntax ist allerdings etwas weniger komfortabel (da strenger), so dass wir an dieser Stelle bei `car::recode` bleiben.

4.1.10.2 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$bestanden <- stats_test$score > 24

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

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

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

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

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

Natürlich könnte man auch hier “Durchpfeifen”:

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

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

4.1.10.3 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

4.1.10.3.1 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` nehmen⁶.

⁶d.h. `ggplot2` muss geladen sein; wenn man `tidyverse` lädt, wird `ggplot2` automatisch auch geladen

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

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).

4.1.10.3.2 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).

4.1.10.3.3 In beliebige Klassen gruppieren

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

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. 4.5). Wenn wir 6 Schnitte (`breaks`) tun, haben wir 5 Teile, wie Abb. 4.5 zeigt. Darum müssen wir auch nur 5 (6-1) `labels` für die Teile vergeben.

4.2 Deskriptive Statistiken berechnen

4.2.1 Mittelwerte pro Zeile berechnen

4.2.1.1 `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 Extraversions-Wert pro Person (d.h. pro Zeile) berechnen.

```
extra <- read.csv("data/extradata.csv")

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

# oder:
# select(extra_items, i01:i10)

extra$extra_mw <- rowMeans(extra_items)
```

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”).

4.2.2 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      30.6     5.72        31          9
```

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(median(score, na.rm = TRUE))
#> # A tibble: 6 x 2
#>   interest `median(score, na.rm = TRUE)`
#>   <int>                <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>      <dbl>
#> 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 `interessiert` zu erstellen:

```
stats_test %>%
  na.omit %>%
  filter(study_time > 1) %>%
  mutate(interessiert = interest > 3) %>%
  group_by(interessiert) %>%
  summarise(md_gruppe = median(score),
            mw_gruppe = mean(score))
#> # A tibble: 2 x 3
#>   interessiert md_gruppe mw_gruppe
#>   <lgl>        <dbl>      <dbl>
#> 1 FALSE         30        30.6
#> 2 TRUE          32        31.5
```

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.

4.2.3 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     NA
#> score             NA       NA     1
```

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:

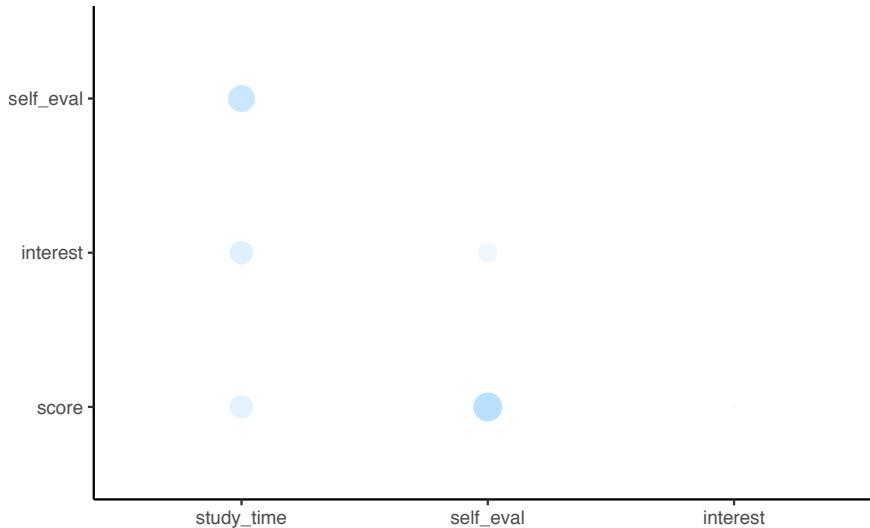
```
stats_test %>%
  select(study_time:score) %>%
  correlate
#> # A tibble: 4 x 5
#>   rowname study_time self_eval interest score
#>   <chr>     <dbl>     <dbl>     <dbl> <dbl>
#> 1 study_time     NA     0.559    0.461 0.441
#> 2 self_eval      0.559     NA     0.360 0.628
#> 3 interest       0.461    0.360     NA 0.223
#> 4 score          0.441    0.628    0.223  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 redundanten obere Dreieck ab (da Korrelationstabellen ja symmetrisch sind):

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

Tabelle 4.1: Befehle des Kapitels 'Praxisprobleme'

Paket::Funktion	Beschreibung
na.omit	Löscht Zeilen, die fehlende Werte enthalten
nrow	Liefert die Anzahl der Zeilen des Dataframes zurück
complete.cases	Gibt die Zeilen ohne fehlenden Werte eines Dataframes zurück
car::recode	Kodiert Werte um
cut	Schneidet eine kontinuierliche Variable in Wertebereiche
rowMeans	Berechnet Zeilen-Mittelwerte
dplyr::rowwise	Gruppert nach Zeilen
ggplot2::cut_number	Schneidet eine kontinuierliche Variable in n gleich große Bereiche
ggplot2::cut_interval	Schneidet eine kontinuierliche Variable in Intervalle der Größe k
head	Zeigt nur die ersten Zeilen/Werte eines Dataframes/Vektors an.
scale	z-skaliert eine Variable
dplyr::select_if	Wählt eine Spalte aus, wenn ein Kriterium erfüllt ist
dplyr::glimpse	Gibt einen Überblick über einen Dataframe
dplyr::mutate_if	definiert eine Spalte, wenn eine Kriterium erfüllt ist
:	Definiert einen Bereich von ... bis ...
corrr::correlate	Berechnet Korrelationstabelle, liefert einen Dataframe zurück
cor	Berechnet Korrelationstabelle
corrr::rplot	Plottet Korrelationsmatrix von correlate
corrr::shave	“Rasiert” redundantes Dreieck in Korrelationsmatrix ab



4.3 Befehlsübersicht

Tabelle 4.1 stellt die Befehle dieses Kapitels dar.

Kapitel 5

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(corr)
```

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

```

movies <- read.csv("data/movies.csv")
glimpse(movies)
#> Observations: 58,788
#> Variables: 24
#> $ title      <fctr> $, $1000 a Touchdown, $21 a Day Once a Month, $40...
#> $ year       <int> 1971, 1939, 1941, 1996, 1975, 2000, 2002, 2002, 19...
#> $ length     <int> 121, 71, 7, 70, 71, 91, 93, 25, 97, 61, 99, 96, 10...
#> $ budget     <int> NA, NA...
#> $ rating     <dbl> 6.4, 6.0, 8.2, 8.2, 3.4, 4.3, 5.3, 6.7, 6.6, 6.0, ...
#> $ votes      <int> 348, 20, 5, 6, 17, 45, 200, 24, 18, 51, 23, 53, 44...
#> $ r1          <dbl> 4.5, 0.0, 0.0, 14.5, 24.5, 4.5, 4.5, 4.5, 4.5...
#> $ r2          <dbl> 4.5, 14.5, 0.0, 0.0, 4.5, 4.5, 0.0, 4.5, 4.5, 0.0, ...
#> $ r3          <dbl> 4.5, 4.5, 0.0, 0.0, 4.5, 4.5, 4.5, 4.5, 4.5, ...
#> $ r4          <dbl> 4.5, 24.5, 0.0, 0.0, 14.5, 14.5, 4.5, 4.5, 0.0, 4...
#> $ r5          <dbl> 14.5, 14.5, 0.0, 0.0, 14.5, 14.5, 24.5, 4.5, 0.0, ...
#> $ r6          <dbl> 24.5, 14.5, 24.5, 0.0, 4.5, 14.5, 24.5, 14.5, 0.0, ...
#> $ r7          <dbl> 24.5, 14.5, 0.0, 0.0, 0.0, 4.5, 14.5, 14.5, 34.5, ...
#> $ r8          <dbl> 14.5, 4.5, 44.5, 0.0, 0.0, 4.5, 4.5, 14.5, 14.5, 4...
#> $ r9          <dbl> 4.5, 4.5, 24.5, 34.5, 0.0, 14.5, 4.5, 4.5, 4.5, 4...
#> $ r10         <dbl> 4.5, 14.5, 24.5, 45.5, 24.5, 14.5, 14.5, 14.5, 24...
#> $ mpaa        <fctr> , , , , , R, , , , , PG-13, PG-13, , , ...
#> $ Action       <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, ...
#> $ Animation    <int> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ Comedy        <int> 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, ...
#> $ Drama         <int> 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, ...
#> $ Documentary   <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ Romance       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ Short         <int> 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, ...

```

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

5.1 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))
#>   Action Animation Comedy Drama Documentary Romance Short

```

```
#> 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 `funs()` definiert.

5.2 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()

#>      key value
#> 1    Action 4688
#> 2 Animation 3690
#> 3    Comedy 17271
#> 4    Drama 21811
#> 5 Documentary 3472
#> 6    Romance 4744
#> 7    Short 9458
```

Der Befehl `gather` baut einen Dataframe von “breit” nach “lang” um (vgl. Kapitel 2.3). Ah, Schmunzettlen Dramen sind also am häufigsten (wie der Befehl `arrange` dann zeigt). Welcome to Hollywood. :tada:

5.3 Zusammenhang zwischen Budget und Beurteilung

Werden teuerere 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! Teuere 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 im Kino - auch wenn diese Filme nicht besser sind. Teuerere Filme sind also bekannter, wenn auch nicht besser (beurteilt); so könnte man die Daten lesen.

5.4 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 6

Daten visualisieren



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
```

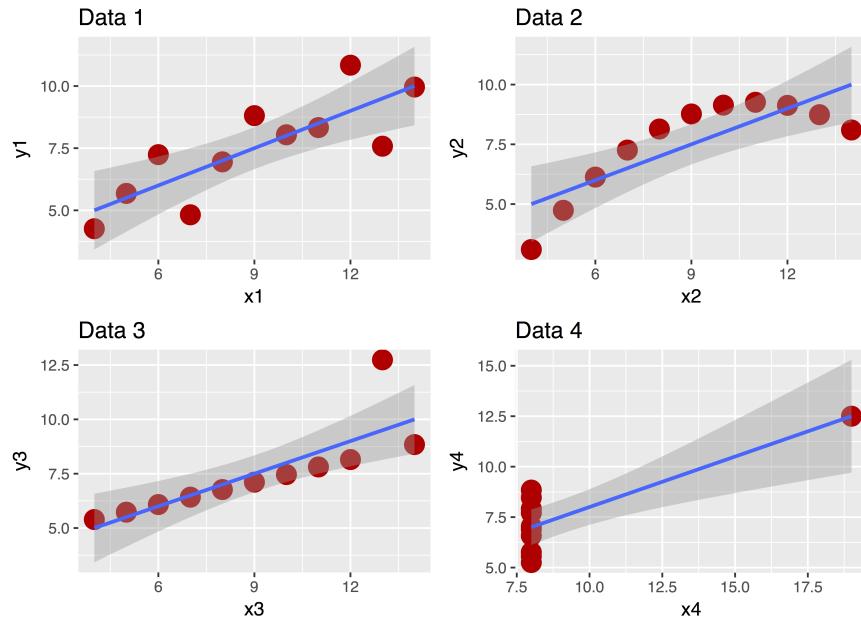
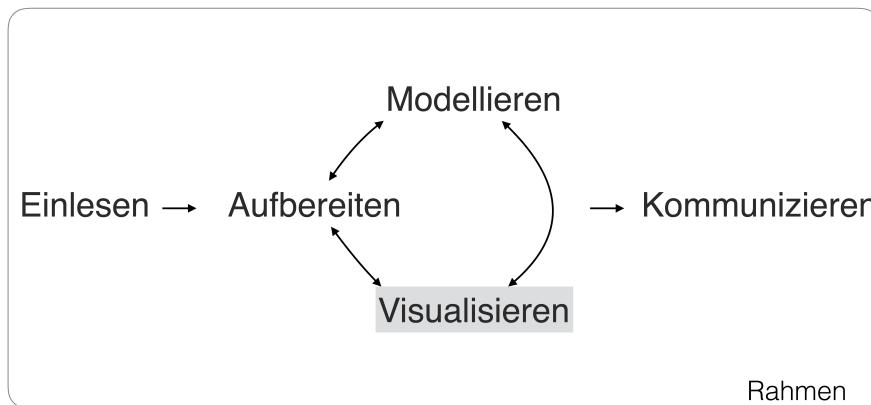


Abbildung 6.1: Das Anscombe-Quartett



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

6.1 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

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

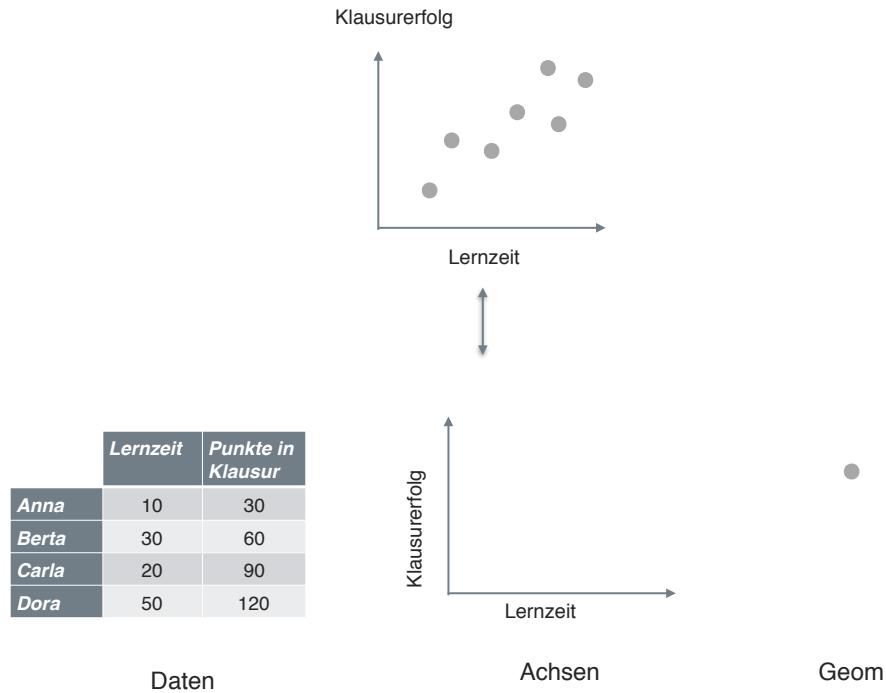


Abbildung 6.2: Anatomie eines Diagramms

Statistiken nicht aufgedeckt; erst die Visualisierung erhellte 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 & 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.

6.2 Die Anatomie eines Diagramms

`ggplot2` unterscheidet folgende Bestandteile (“Anatomie”) eines Diagramms (vgl. Abb. 6.2):

- 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`

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>

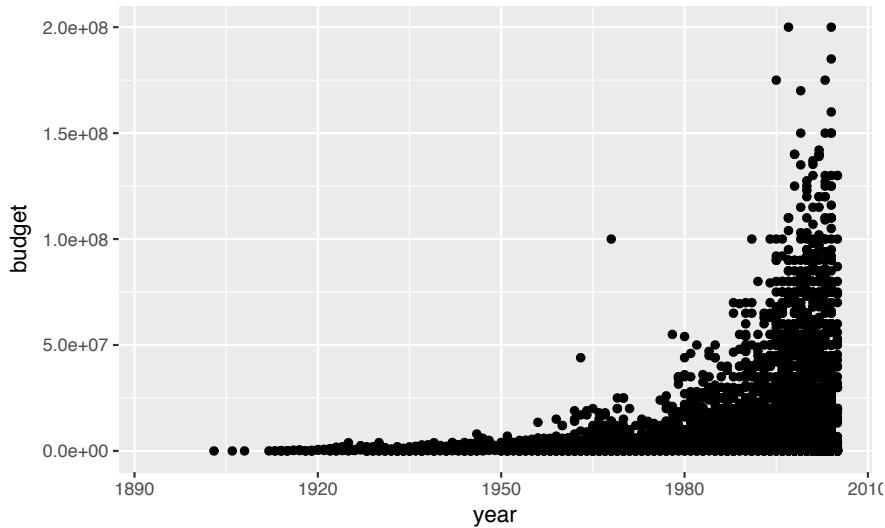


Abbildung 6.3: Mittleres Budget pro Jahr

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 6.10).

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

6.3 Einstieg in `ggplot2` - `qplot`

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

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

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

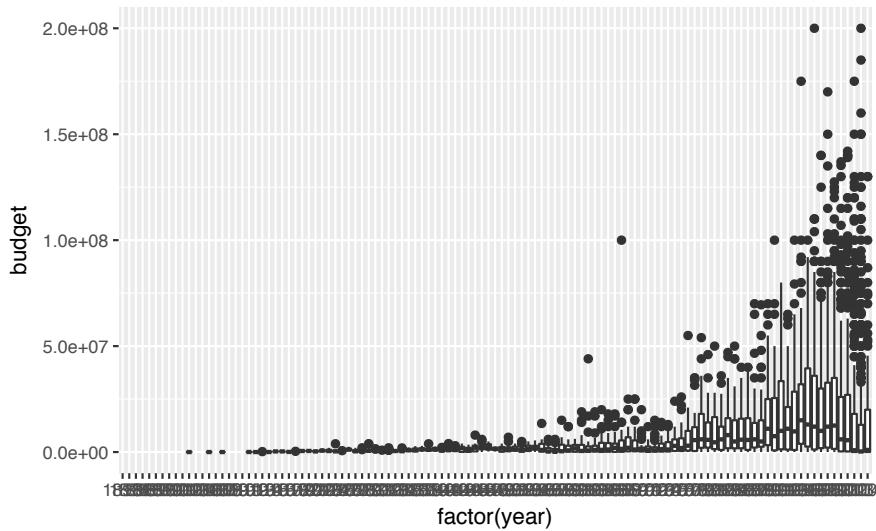
Schauen wir uns den Befehl `qplot` etwas näher an. Wie ist er aufgebaut?



`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. `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.

Es sind zu viele Jahre, das macht das Diagramm unübersichtlich. Besser wäre es, Jahrzehnte dazustellen:

```
movies$Jahrzehnt <- (movies$year / 10) %>% trunc  
movies$Jahrzehnt <- movies$Jahrzehnt * 10
```

Um Jahrzehnte darzustellen müssen wir sozusagen die letzte Jahresziffer “abhacken”: “1978” wird zu “178”. Das erreichen wir in dem wir die Jahreszahl durch 10 teilen und dann den Rest unter den Tisch fallen lassen (trunkieren, `trunc`).

Ok, auf ein Neues (Abb. 6.4):

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

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

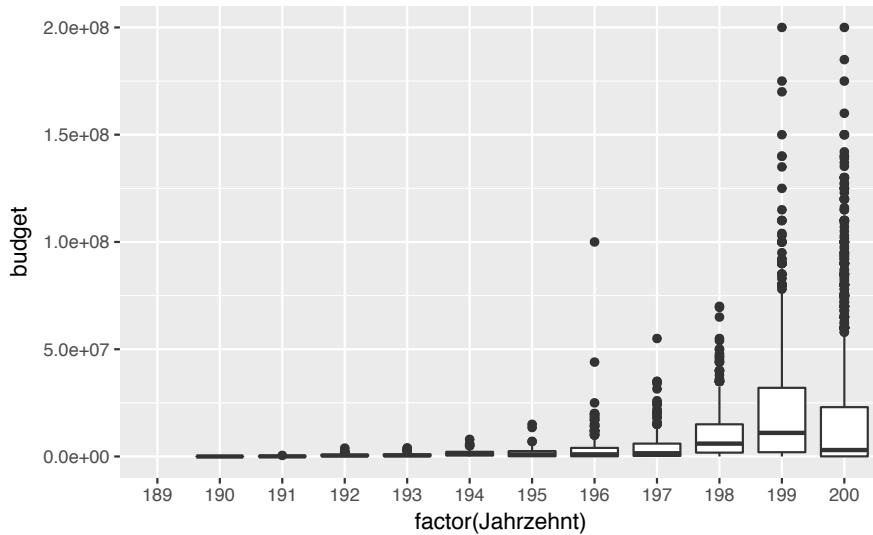


Abbildung 6.4: Film-Budgets über die Jahrzehnte

“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.

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...

6.4 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.

6.4.1 Eine kontinuierliche Variable

Schauen wir uns die Verteilung von Filmbudgets aus `movies` an (s. Abb. 6.5).

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

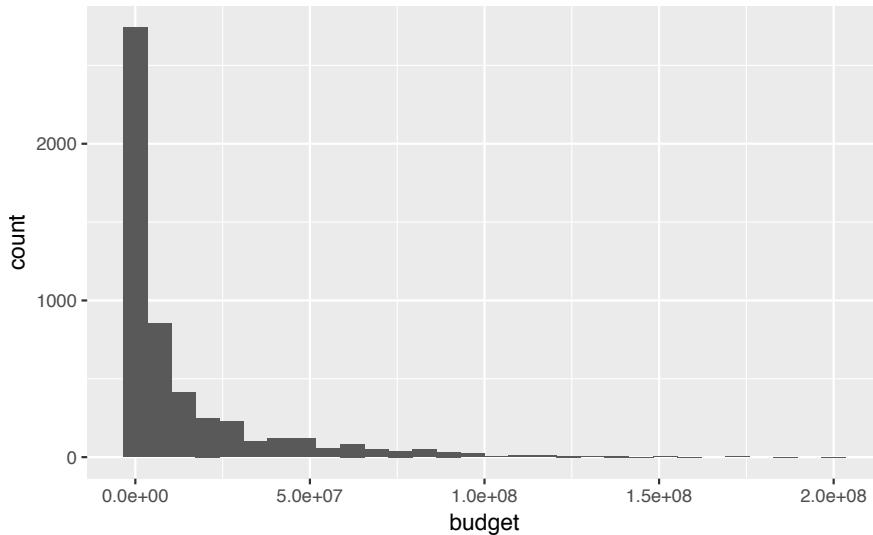


Abbildung 6.5: Verteilung des Budgets von Filmen

```
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).

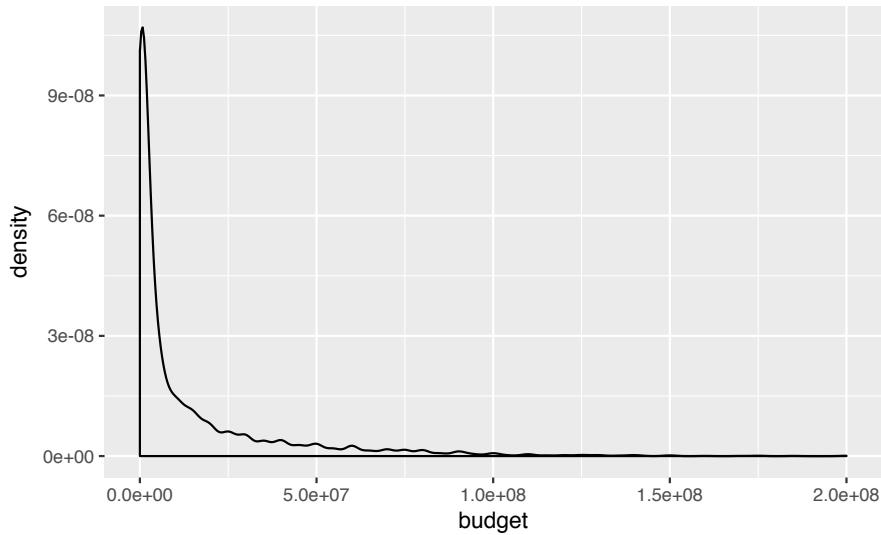


Was heißt das kleine ‘e’, das man bei wissenschaftlichen Zahlen hin und wieder sieht (wie im Diagramm 6.5)?

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* Nullen: 10000000. Die schöne Zahl soll noch mit 5 multipliziert werden: also 50000000. Bei so vielen Nullen 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.

Alternativ wäre ein Dichtediagramm hier von Interesse:

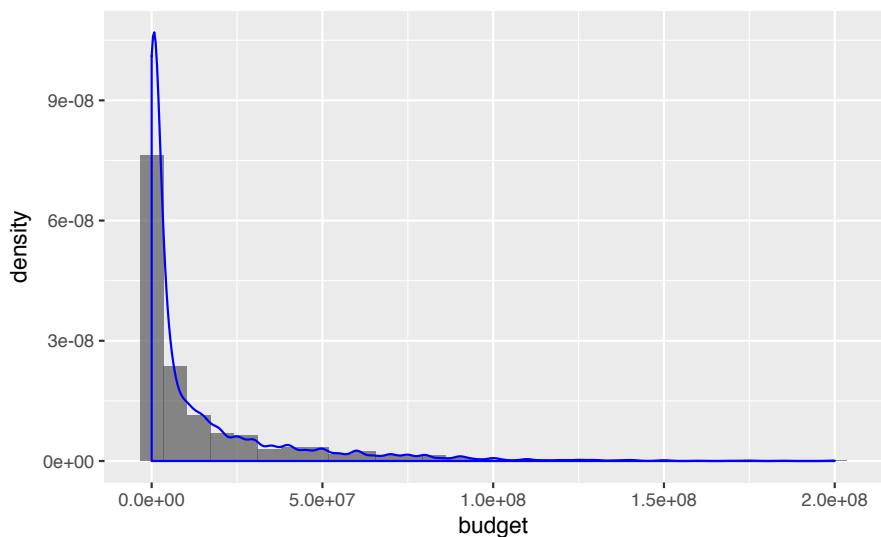
```
qplot(x = budget, data = movies, geom = "density")
```



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

Vielelleicht 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 Histogramm

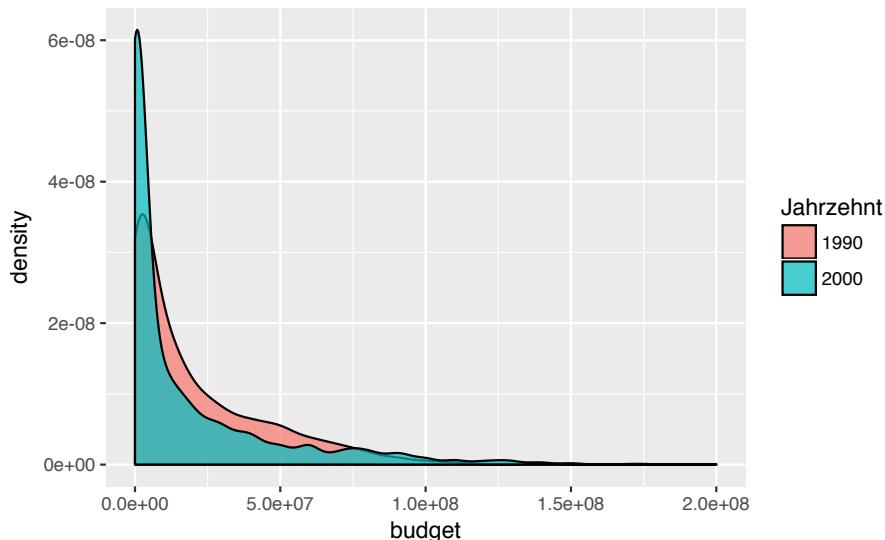
hätten: `geom_histogram`. Dabei soll aber nicht wie gewöhnlich auf der X-Achse die Häufigkeit stehen, sondern die Dichte. `ggplot` berechnet selbstständig die Dichte und nennt diese Variable `..density..`; die vielen Punkte sollen wohl klar machen, dass es sich nicht um eine “normale” Variable aus dem 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$Jahrzehnt <- factor(movies2$Jahrzehnt)

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

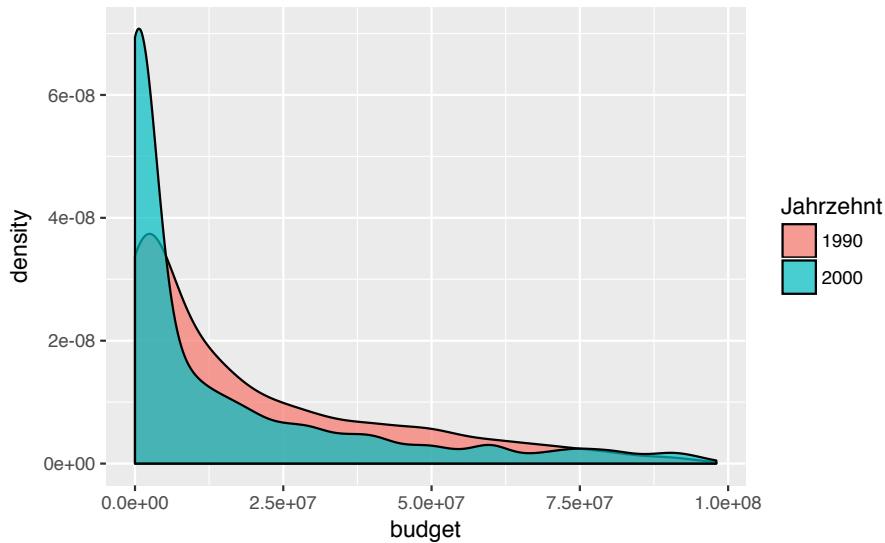


`qplot` erwartet immer *Variablen* als Parameter; wollen wir mal keine Variable, sondern eine fixen Wert, wie 0.7, übergeben, so können wir das mit dem Befehl `I` (wie “identity”) tun.

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

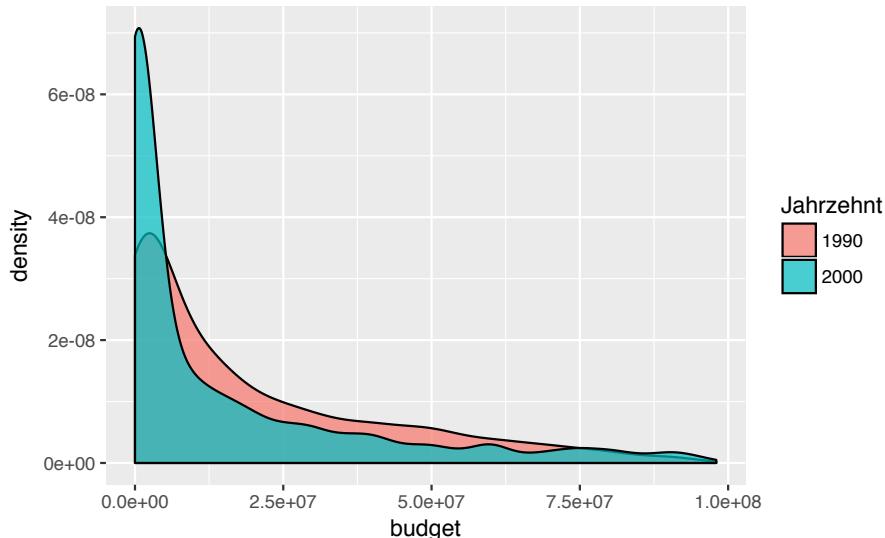
```
movies2 %>%
  filter(budget < 1e08) -> movies2
```

```
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))
```



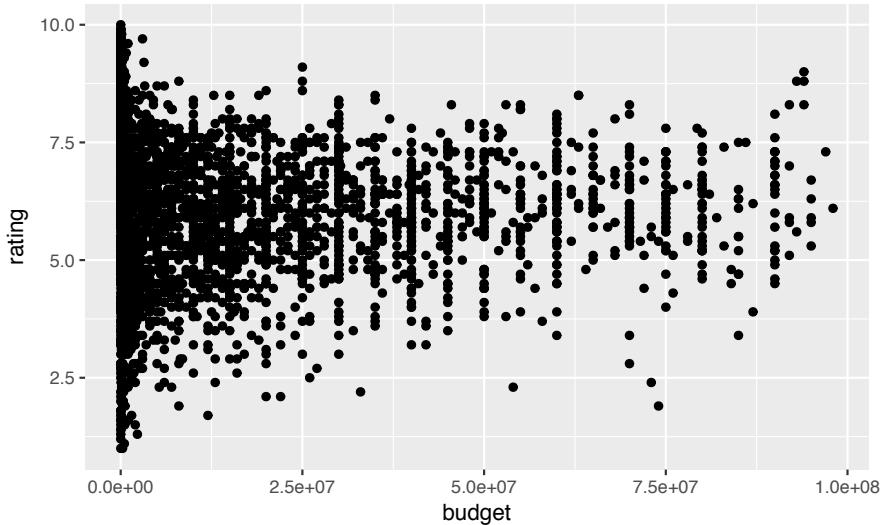
Die Pfeife versucht im Standard, das Endprodukt des letzten Arbeitsschritts an den *ersten* Parameter des nächsten Befehls weiterzugeben. Ein kurzer Blick in die Hilfe von `qplot` zeigt, dass der erste Parameter nicht `data` ist, sondern `x`. Daher müssen wir explizit sagen, an welchen Parameter wir 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.

6.4.2 Zwei kontinuierliche Variablen

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

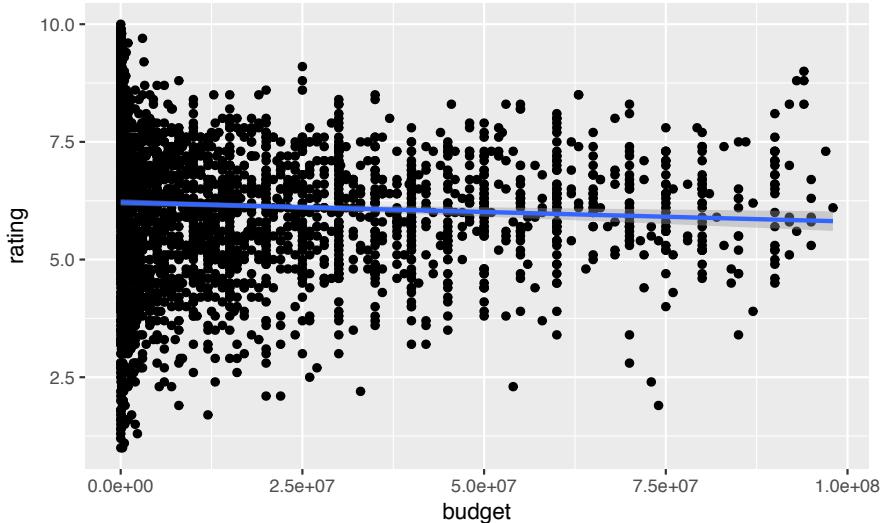
```
p <- qplot(x = budget, y = rating, data = movies2)
p
```



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")
```

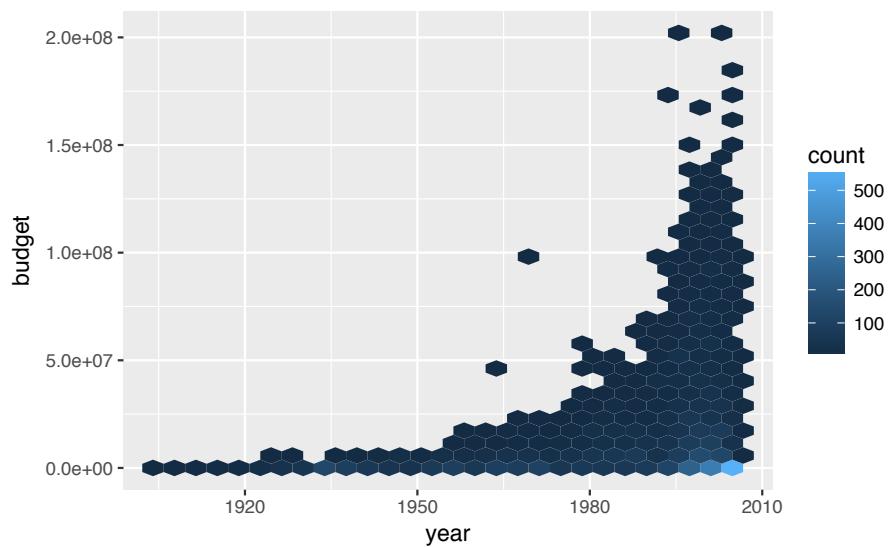
Da `ggplot` als *ersten* Parameter die Daten erwartet, kann die Pfeife hier problemlos durchgebracht werden. Innerhalb eines `ggplot`-Aufrufs werden die einzelnen Teile durch ein Pluszeichen + voneinander getrennt. Nachdem wir den Dataframe benannt haben, definieren wir die Zuweisung der Variablen zu 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.

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

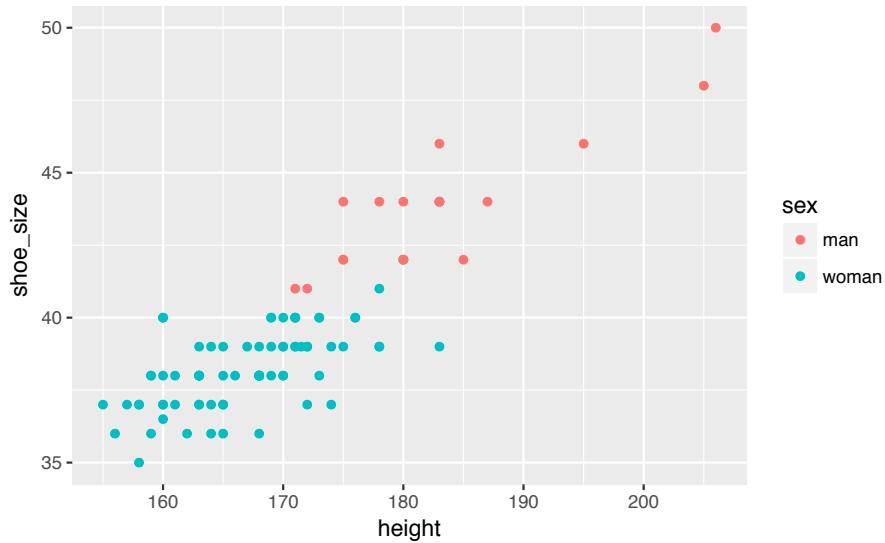
ggplot(movies) +
  aes(x = year, y = budget) +
  geom_hex()
```



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

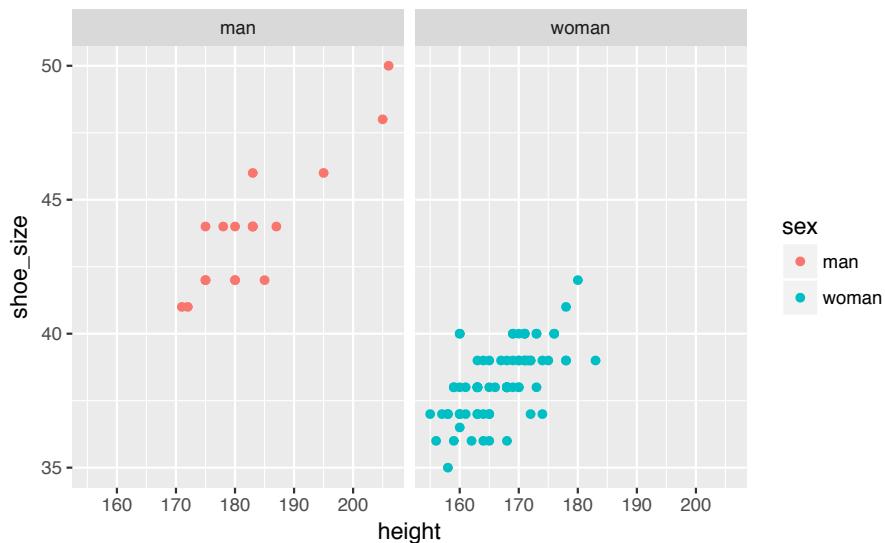
wo_men2 %>%
  qplot(x = height, y = shoe_size, color = sex, data = .)
```



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 = .)
```



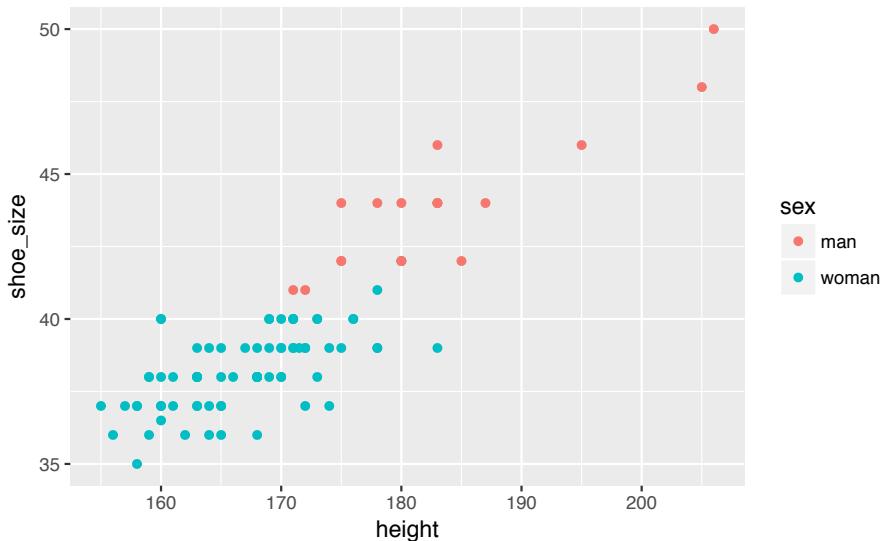
Man beachte die Tilde `~`, die vor die “Gruppierungsvariable” `sex` zu setzen ist.

6.4.2.1 Vertiefung zu Facetten

Ein netter visueller Effekt wird erreicht, wenn in jeder Facette zwar alle Punkte gezeigt werden in einem leichten Grau. Aber farbig betont werden nur die Punkte, die zur jeweiligen Gruppe gehören. Der optische Eindruck erklärt es einfacher als Worte:

```
wo_men %>%
  dplyr::filter(height > 150, height < 210, shoe_size < 55) %>%
  dplyr::select(-sex) -> wo_men4

wo_men4 %>%
  ggplot(ggplot2::aes(x = height, y = shoe_size)) +
  geom_point(color = "grey80") +
  # facet_wrap(~sex) +
  geom_point(data = wo_men2, ggplot2::aes(color = sex))
```



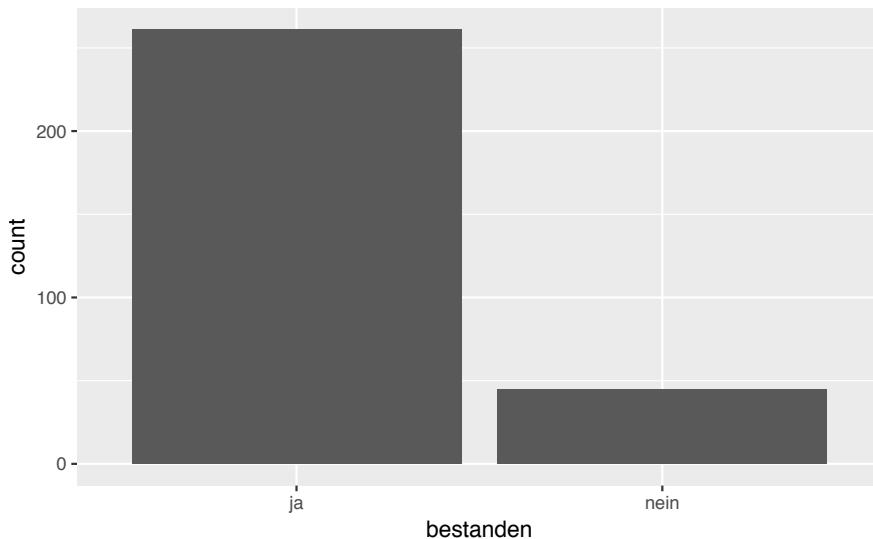
Der “ggplot-Trick” ist, zuerst die Punkte *ohne* Gruppierungsinformation (hier: `sex`) zu plotten. Danach plotten wir die nach Gruppenzugehörigkeit gefärbten Punkte.

6.4.3 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/test_inf_short.csv")
```

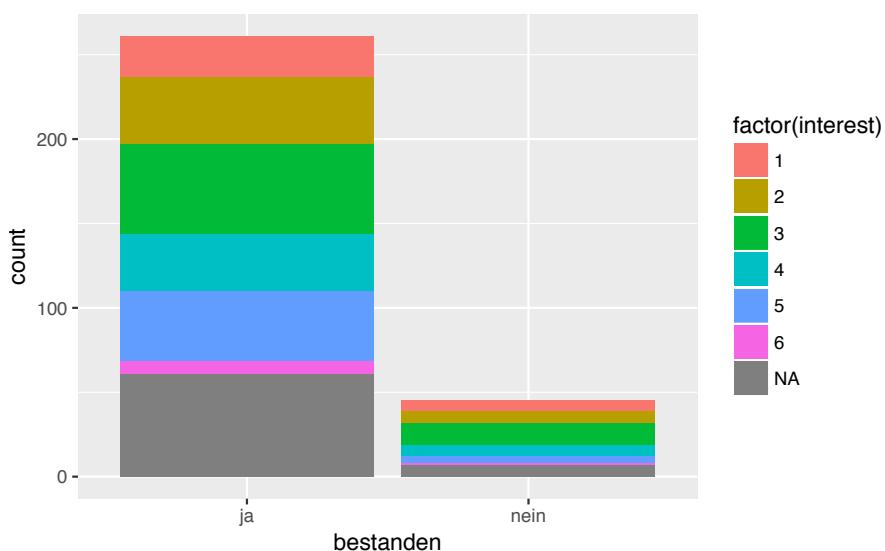
```
)
qplot(x = bestanden, data = stats_test)
```



Falls nur die X-Achse definiert ist und dort eine Faktorvariable oder eine Text-Variable 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.

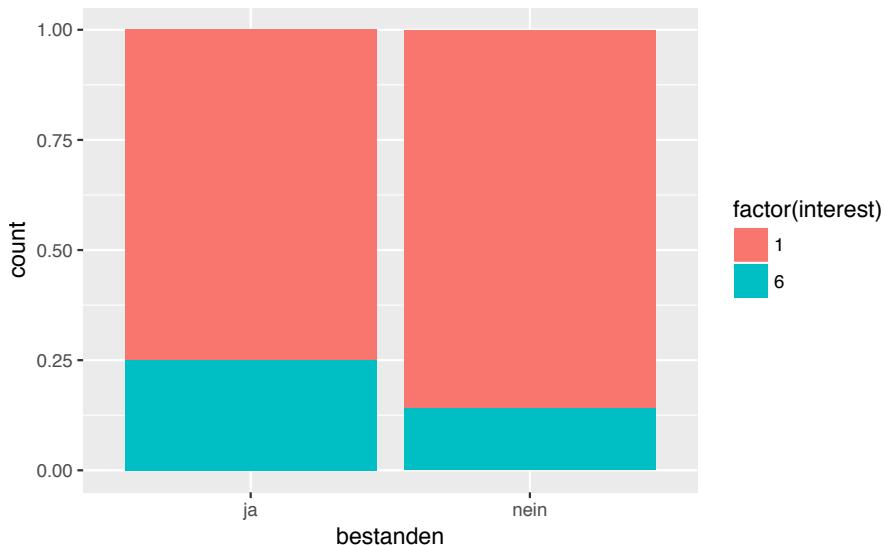
```
qplot(x = bestanden, fill = factor(interest), data = stats_test)
```



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 nominal-skalierte 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_point` etc. gewählt.

position = fill: `position_fill` will 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).

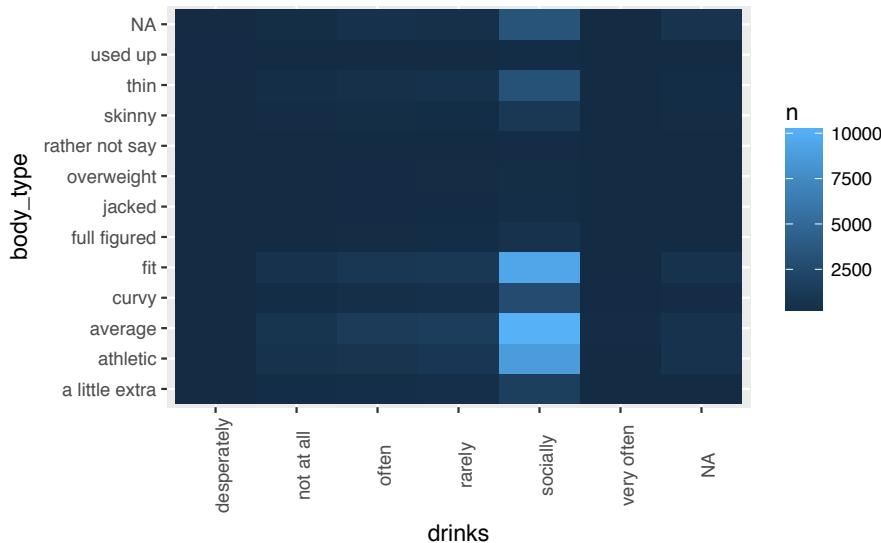
Je unterschiedlicher die “Füllhöhe”, desto stärker sind die Variablen (X-Achse vs. Füllfarbe) voneinander abhängig (bzw. desto stärker der Zusammenhang).

6.4.4 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 Körperform 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.

6.4.5 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.

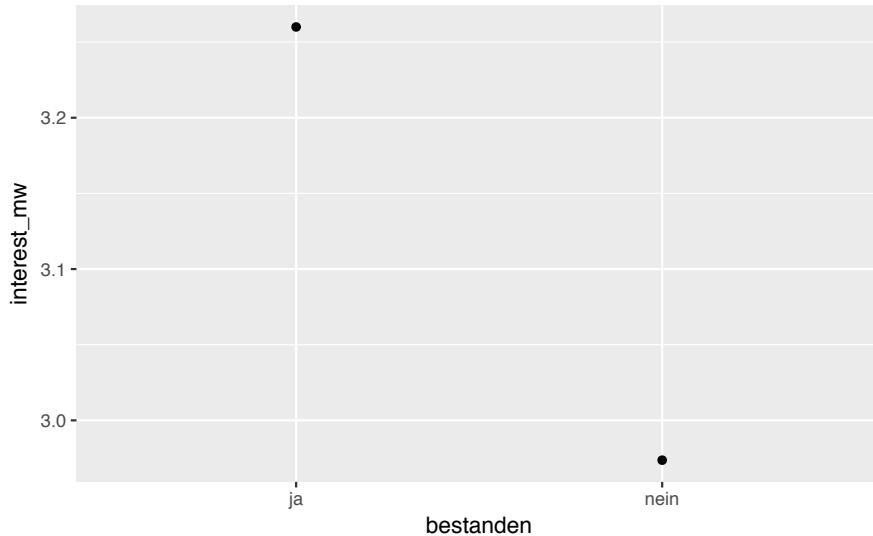
```
stats_test %>%
  group_by(bestanden) %>%
  summarise(interest_mw = mean(interest, na.rm = TRUE)) -> stats_test_summary

stats_test_summary
```

```
#> # A tibble: 2 x 2
#>   bestanden interest_mw
#>   <fctr>      <dbl>
#> 1 ja          3.26
#> 2 nein        2.97
```

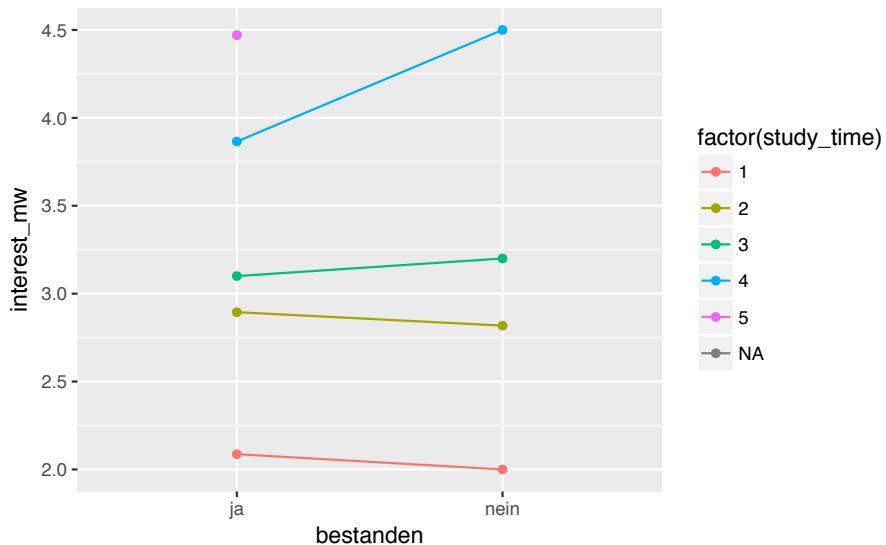
Diese Tabelle schieben wir jetzt in ggplot2; natürlich hätten wir das gleich in einem Rutsch durchpfeifen können.

```
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, wieviel (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:



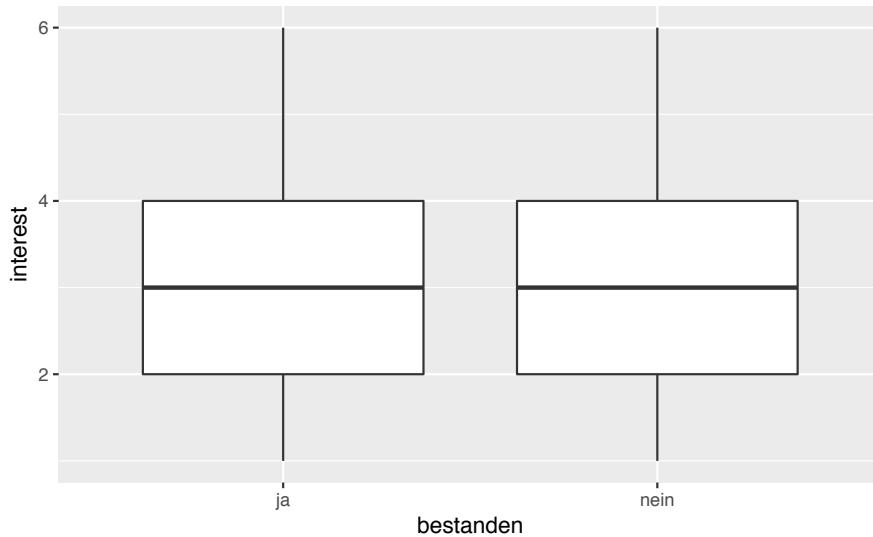
Nehme den Datensatz “stats_test” UND DANN
gruppiere nach den Variablen **bestanden** und **study_time** UND DANN fasse für diese
Gruppen jeweils die Spalte **interest** zum Mittelwert zusammen UND DANN
male einen schnellen Plot mit diesen Daten UND DANN füge ein Liniendiagramm dazu,
wobei jede Stufe 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

Alternativ, und deutlich informationsreicher (besser) sind hier Boxplots.

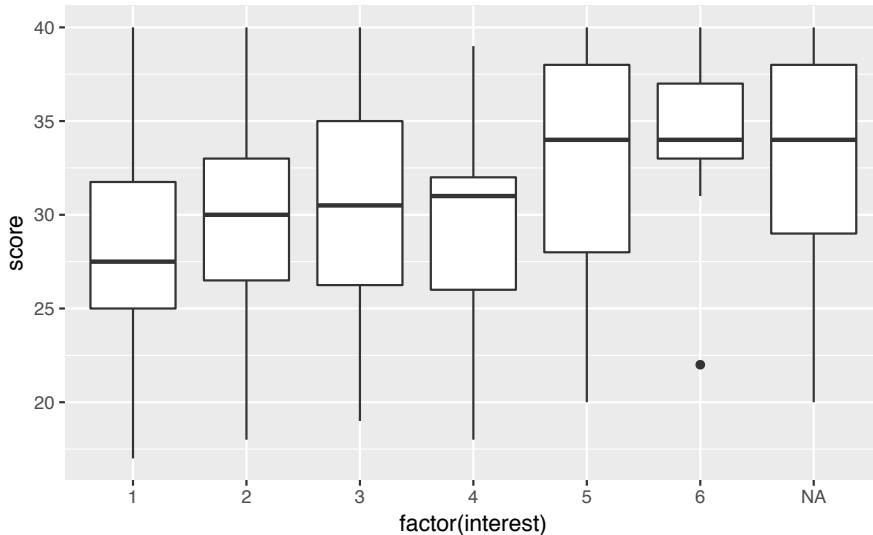
```
qplot(x = bestanden,
      y = interest,
      data = stats_test,
      geom = "boxplot")
```

⁶es gibt kein **study_time == 5** bei den Durchfallen, d.h. bei **bestanden == nein**.



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")
```



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.

Tabelle 6.1: 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)

6.4.6 Überblick zu häufigen Diagrammtypen

Die Tabelle 6.1 fasst die gerade besprochenen Diagrammtypen zusammen.

6.5 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:

```
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)
```

6.6 Aufgaben

1. Erzählen Sie einer vertrauenswürdigen Person jeweils eine “Geschichte”, die das Zustandekommen der vier Plots von Anscombe (Abb. 6.1) erklärt!
2. Abb. 6.4 stellt das mittlerer Budget von Filmmem 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!

6.7 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. 6.6). Durch die gleiche Ausrichtung der Boxplots ist es dem Auge viel einfacher, Vergleiche anzustellen im Vergleich zu den Histogrammen. Einen optisch schönen Effekt könnte man mit `geom_jitter` anstelle von `geom_point` erreichen. Auch die Reihenfolge der beiden Geome könnte man umdrehen. Natürlich ist auch an Form, Größe und Farbe der Geome noch zu feilen.

3. :

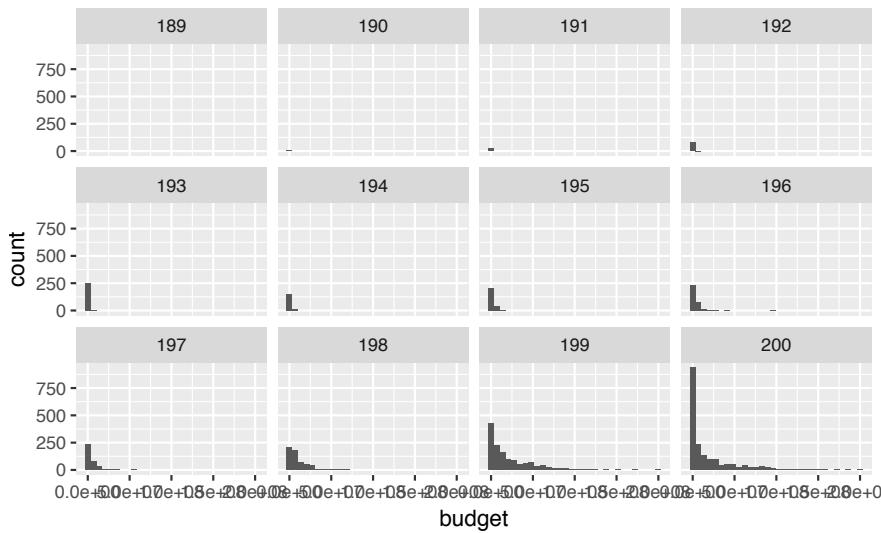
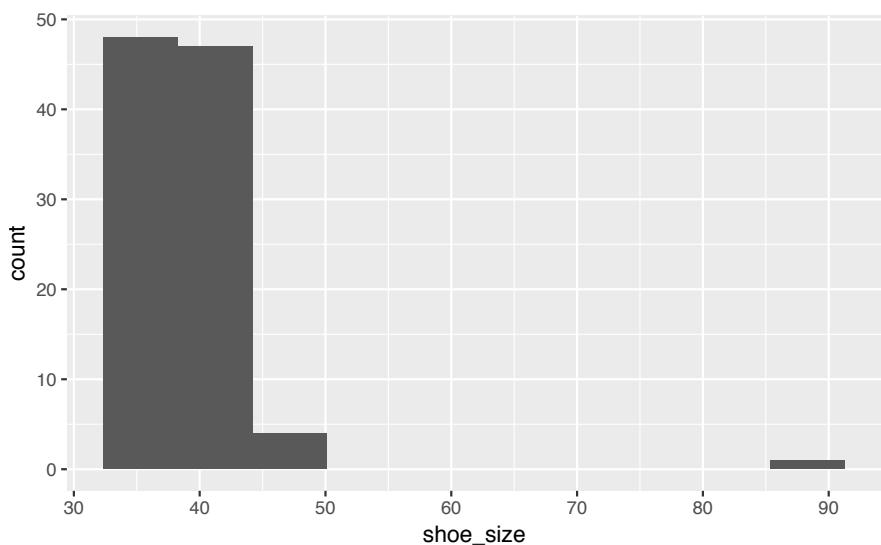
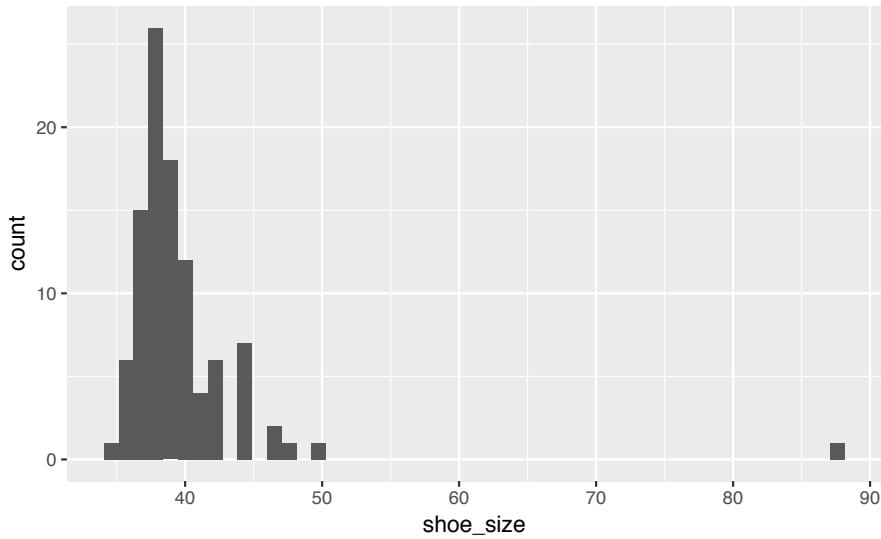


Abbildung 6.6: Film-Budgets mit Histogrammen

```
qplot(x = shoe_size, data = wo_men, bins = 10)
qplot(x = shoe_size, data = wo_men, bins = 50)
```

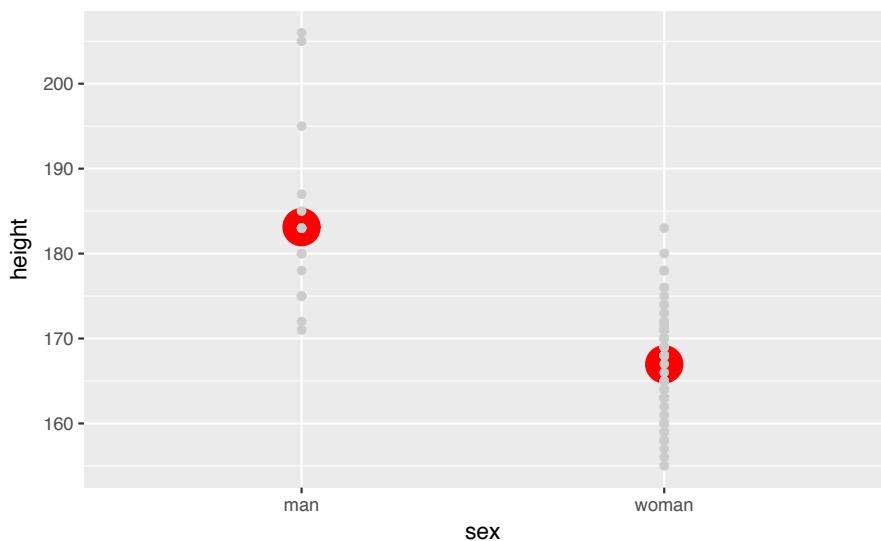




4. :

```
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

Tabelle 6.2: Befehle des Kapitels 'Daten visualisieren'

Paket::Funktion	Beschreibung
ggplot2::qplot	Malt schnell mal einen Plot
ggplot2::ggplot	Malt einen Plot
factor	Wandelt einen Vektor in den Typ factor um

Achsen-Beschriftung bei beiden Geomen identisch sein, sonst gibt es eine Fehlermeldung.

6.8 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)

6.9 Befehlsübersicht

Tabelle 6.2 fasst die R-Funktionen dieses Kapitels zusammen.

6.10 Vertiefung: Geome bei ggplot2

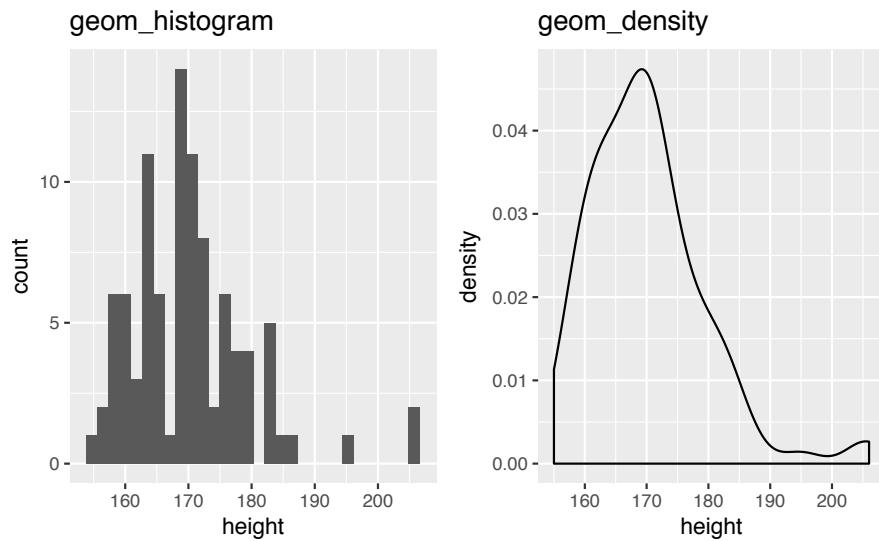
Einen guten Überblick über Geome bietet das Cheatsheet von ggplot2⁸.

Verschiedenen Taxonomien von statistischen "Bildchen" sind denkbar; eine einfache ist die folgende; es wird nur ein Teil der verfügbaren Geome dargestellt.

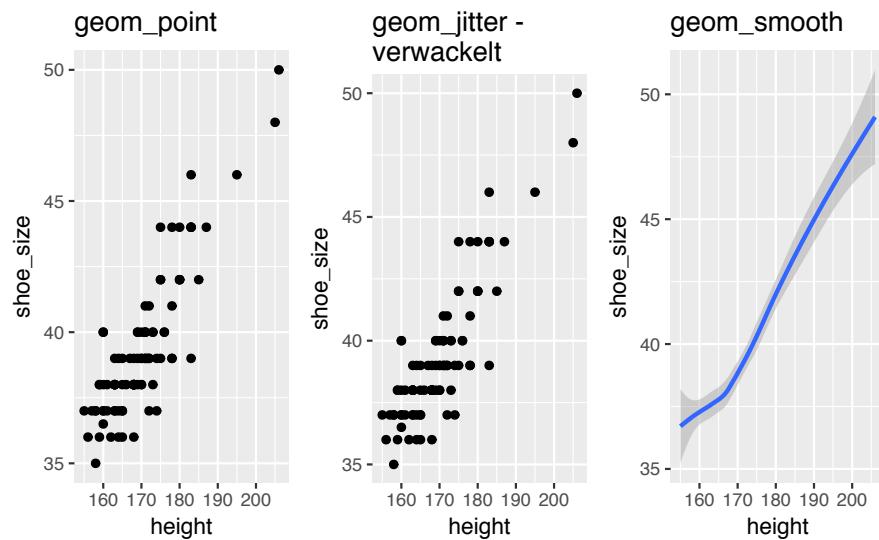
⁷R, R, F, F, R

⁸<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

1. Eine kontinuierliche Variable

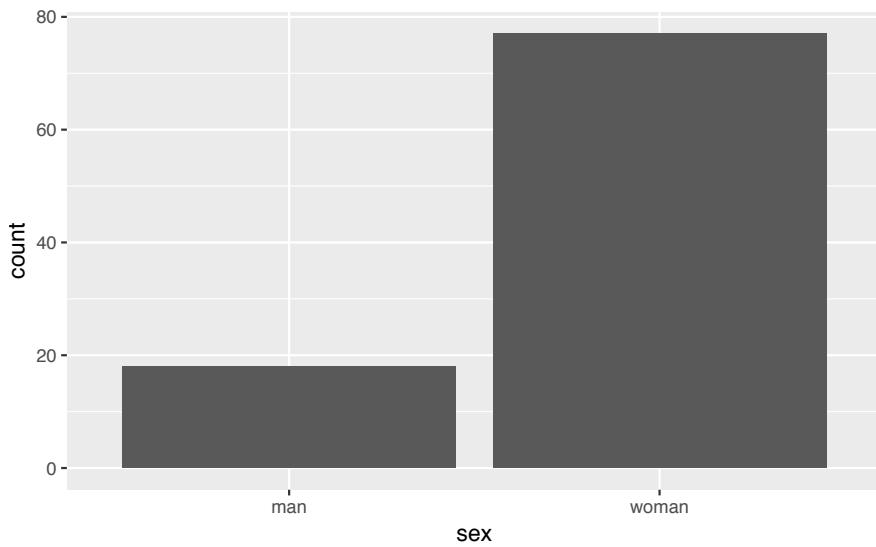


2. Zwei kontinuierliche Variablen



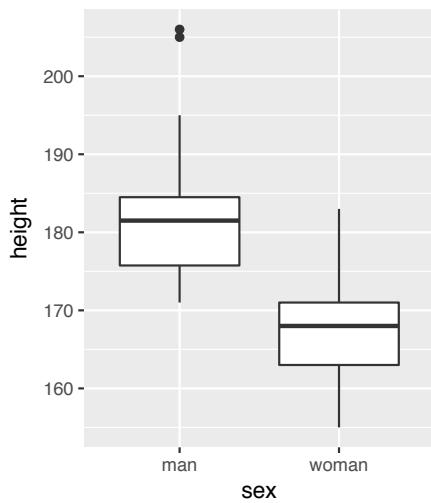
3. Eine diskrete Variable (X-Achse)

```
ggplot(wo_men2) +
  aes(x = sex) +
  geom_bar()
```

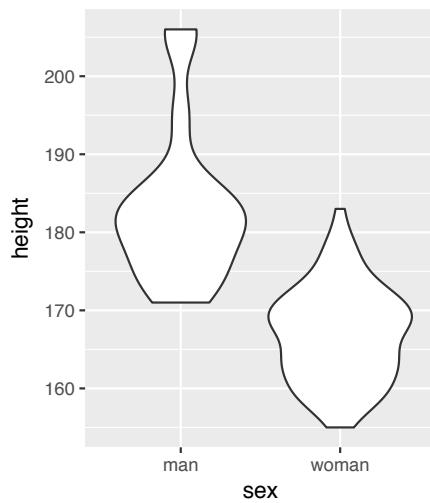


4. Eine diskrete Variable auf der X-Achse und eine kontinuierliche Y-Achse

geom_boxplot



geom_violin



6.11 Verweise

- Einen Befehlsüberblick zu `ggplot2` findet sich hier: <http://ggplot2.tidyverse.org/reference/>.
- 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>.

Kapitel 7

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.

Benötigte Pakete:

```
library(tidyverse)
library(corr)
library(GGally)
```

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.

7.1 Daten einlesen

Hier laden wir einen Datensatz von einer Online-Umfrage:

```
data <- read.csv("data/extrा.сsv")
```

Der Datensatz besteht aus 10 Extraversions-Items (B5T nach Satow¹) sowie einigen Verhaltenskorrelaten (zumindest angenommenen). Uns interessieren also hier nur die 10 Extraversions-

¹https://www.zpid.de/pub/tests/PT_9006357_B5T_Forschungsbericht.pdf

Items, die zusammen Extraversion als Persönlichkeitseigenschaft messen (sollen). Wir werden die Antworten der Befragten darstellen, aber uns hier keine Gedanken über Messqualität u.a. machen.

Die Umfrage kann hier² eingesehen werden. Schauen wir uns die Daten mal an:

```
glimpse(data)
```

7.2 Daten umstellen

Wir haben ein Diagramm vor Augen (s.u.), 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.

Also, los geht’s: Zuerst wählen wir aus der Fülle der Daten, die Spalten, die uns interessieren: Die 10 Extraversions-Items, in diesem Fall.

```
data_items <- select(data, i01:i10)
```

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.

```
data_long <- gather(data_items, key = items, value = Antwort)
data_long$Antwort <- factor(data_long$Antwort)
```

Den Befehl mit `factor` brauchen wir für zum Diagramm erstellen im Folgenden. Dieser Befehl macht aus den Zahlen bei der Variable `Antwort` eine nominale Variable (in R: `factor`) mit Text-Werten “1”, “2” und so weiter. Wozu brauchen wir das? Der Diagrammbefehl unten kann nur mit nominalen Variablen Gruppierungen durchführen. Wir werden in dem Diagramm die Anzahl der Antworten darstellen - die Anzahl der Antworten nach Antwort-Gruppe (Gruppe mit Antwort “1” etc.).

Keine Sorge, wenn sich das reichlich ungewöhnlich anhört. Sie müssen es an dieser Stelle nicht erfinden :-)

²https://docs.google.com/forms/d/e/1FAIpQLSfD4wQuhDV_edx1WBfN3Qos7XqoVbe41VpiKLRKtGLeuUD09Q/viewform

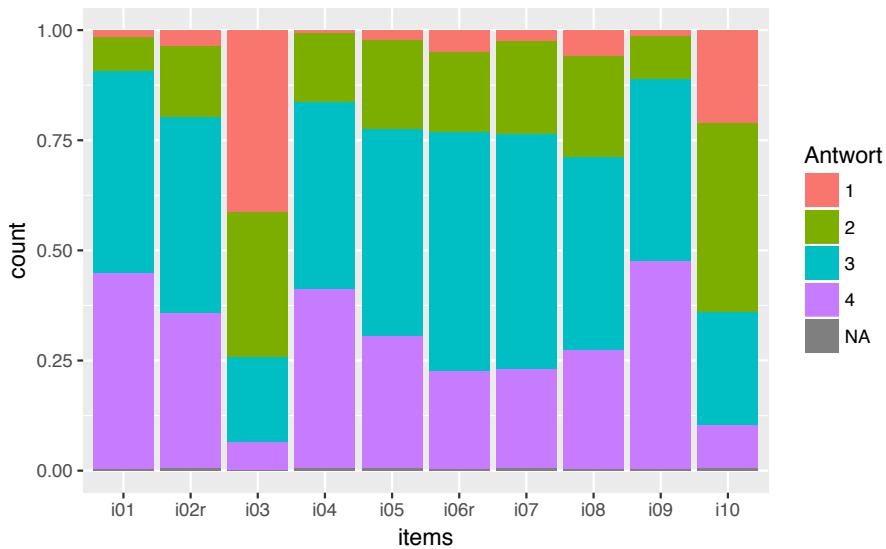
Man gewöhnt sich daran einerseits; und andererseits ist es vielleicht auch so, dass diese Funktionen nicht perfekt sind, oder nicht aus unserer Sicht oder nur aus Sicht des Menschen, der die Funktion geschrieben hat. Jedenfalls brauchen wir hier eine **factor** Variable zur Gruppierung...

Damit haben wir es schon! Jetzt wird gemalt.

7.3 Diagramme für Anteile

Wir nutzen `ggplot2`, wie gesagt, und davon die Funktion `qplot` (q wie quick, nehme ich an.).

```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill")
```



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.).

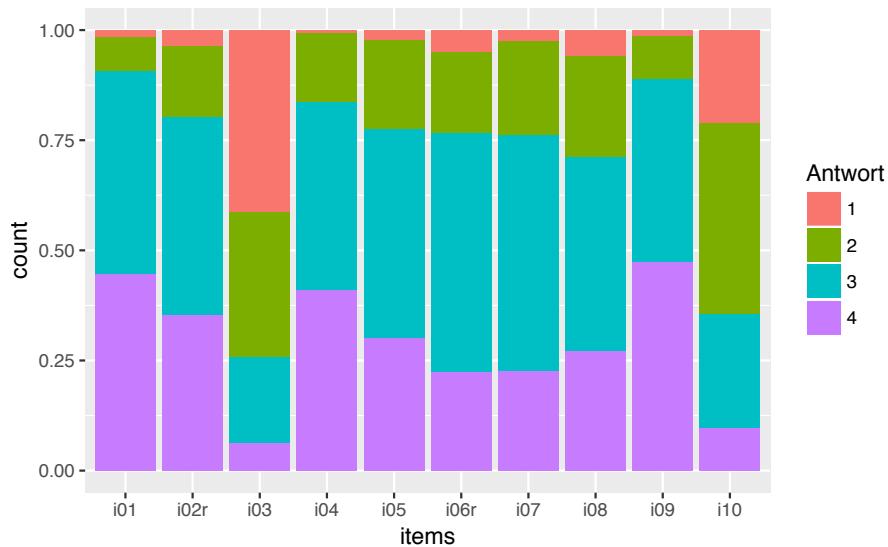
- `position = "fill"` sagt, dass die Gesamt-Höhe des Balken aufgeteilt werden soll mit den “Teil-Höhen” der Gruppen (Antwort-Kategorien 1 bis 4); wir hätten die Teil-Höhen auch nebeneinander stellen können.

Vielleicht ist es schöner, die NAs erst zu entfernen.

```
data_long <- na.omit(data_long)
```

Und dann noch mal plotten:

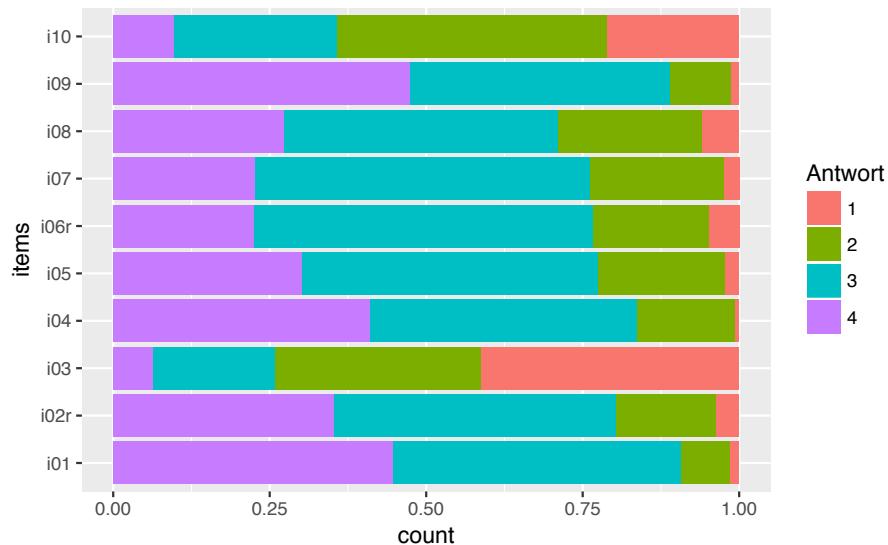
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill")
```



7.4 Um 90° drehen

Dazu nehmen wir `+ coord_flip()`, also “flippe das Koordinatensystem”.

```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip()
```



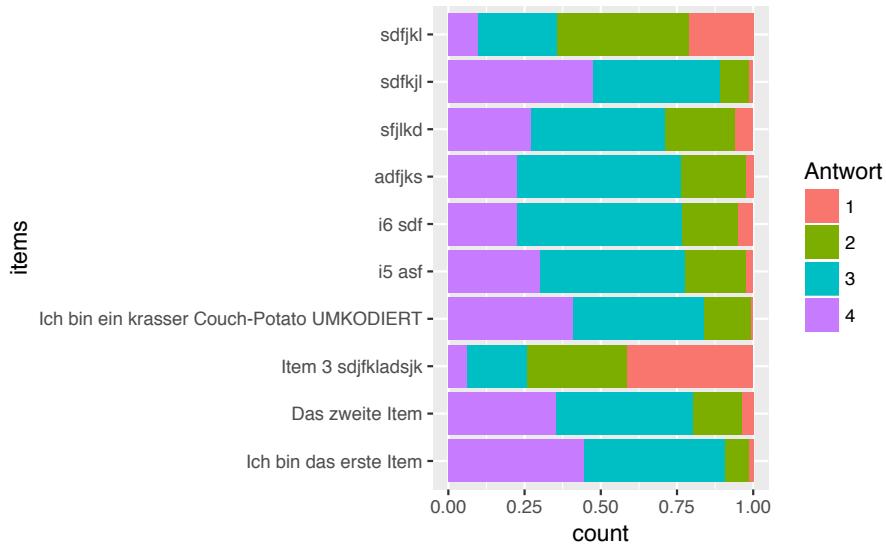
7.5 Text-Labels für die Items

Wir definieren die Texte (“Labels”) für die Items:

```
item_labels <- c("Ich bin das erste Item",
               "Das zweite Item",
               "Item 3 sdjfkladsjk",
               "Ich bin ein krasser Couch-Potato UMKODIERT",
               "i5 asf", "i6 sdf", "adfjks", "sfjlkd", "sdfkjl", "sdfjkl")
```

Jetzt hängen wir die Labels an die Items im Diagramm:

```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip() +
  scale_x_discrete(labels = item_labels)
```

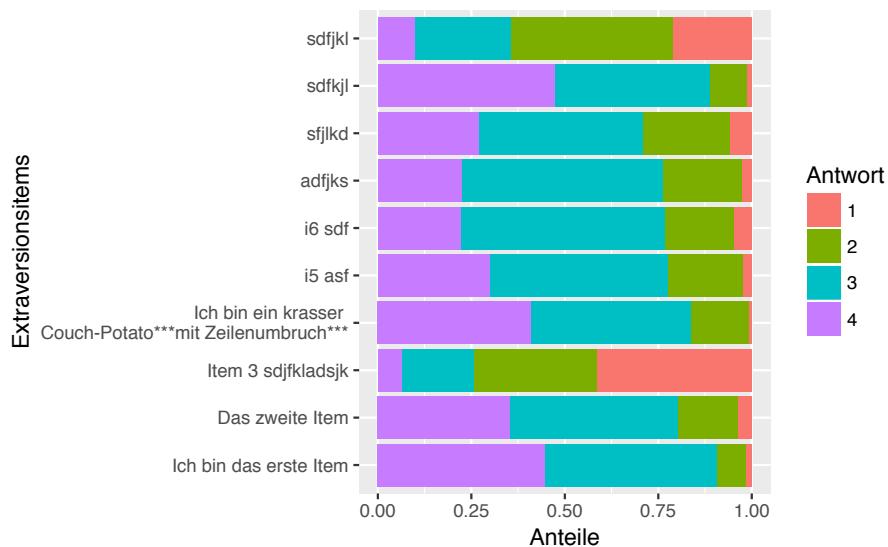


Man kann auch einen Zeilenumbruch in den Item-Labels erzwingen... wobei das führt uns schon recht weit, aber gut, zum Abschluss :-)

```
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")
```

Und wieder plotten:

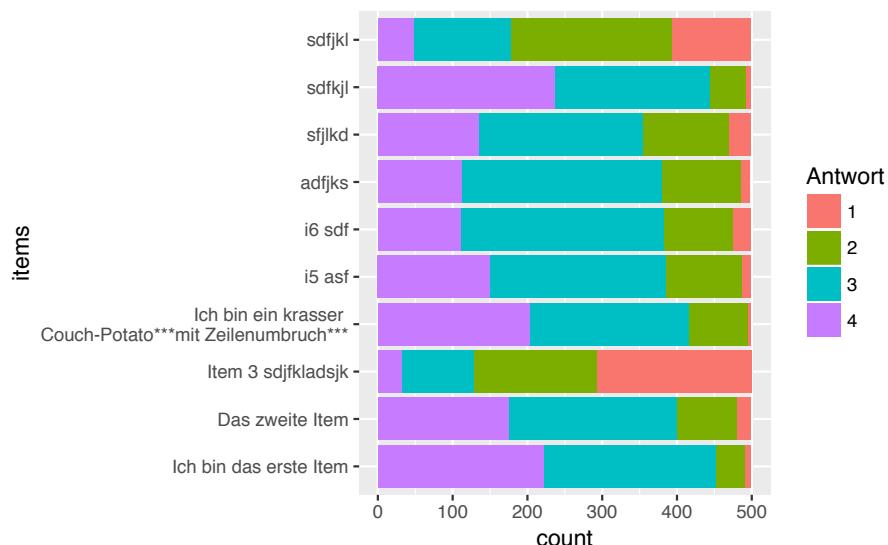
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip() +
  scale_x_discrete(labels = item_labels, name = "Extraversionsitems") +
  scale_y_continuous(name = "Anteile")
```



7.6 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”

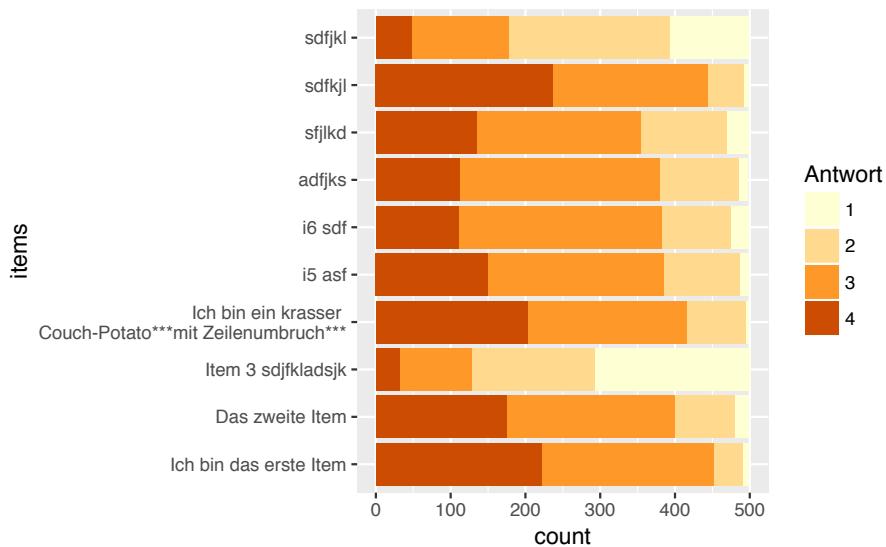
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "stack") +
  coord_flip() +
  scale_x_discrete(labels = item_labels)
```



7.7 Farbschema

Ja, die Wünsche hören nicht auf... Also, noch ein anderes Farbschema:

```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "stack") +
  coord_flip() +
  scale_x_discrete(labels = item_labels) +
  scale_fill_brewer(palette = 17)
```



Teil II

Modellieren

Kapitel 8

Grundlagen des Modellierens



Lernziele:

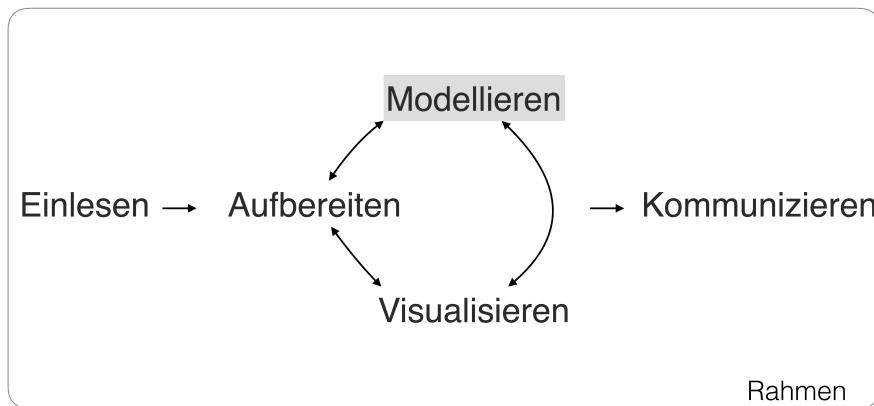
- Erläutern können, was man unter einem Modell versteht.
- Die Ziele des Modellieren aufzählen und erläutern können.
- Die Vor- und Nachteile von einfachen vs. komplexen Modellen vergleichen können.
- Wissen, was man unter “Bias-Varianz-Abwägung” versteht.
- Um die Notwendigkeit von Trainings- und Test-Stichproben wissen.
- Wissen, was man unter Modellgüte versteht.
- Um die Schwierigkeiten der Prädiktorenauswahl wissen.

In diesem Kapitel benötigen wir diese Pakete:



Abbildung 8.1: Modell eines VW-Käfers

```
library(tidyverse)
```



8.1 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. 8.1; (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 berücksichtigt.

Manche Aspekte der Wirklichkeit sind wirklicher als andere: Interessiert man sich für den Zusammenhang von Temperatur und Grundwasserspiegel, so sind diese Dinge direkt beobacht-

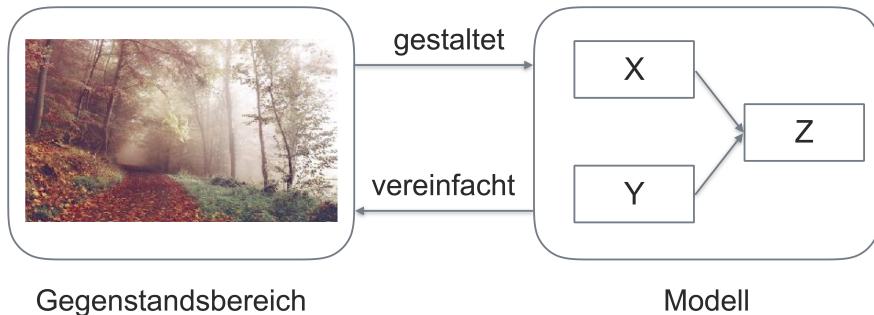


Abbildung 8.2: Modellieren

bar. 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. 8.2, 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 Beispiel 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*. Auch hier gibt es eine Reihe von (mathematischen) Objekten wie Zahlen oder Vektoren. Diese mathematischen Objekten stehen ebenfalls 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 erwiedern, dass Alois einen hohen Klausurerfolg hat (Modelle

¹wieder ein typisches Dozentenbeispiel



Abbildung 8.3: Formaleres Modell des Modellierens

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. 8.3 stellt diese formalere Sichtweise des Modellierens dar; das empirische System E wird dem numerischen System Z zugeordnet. Dabei besteht E aus einer Menge von Objekten O sowie einer Menge von Relationen R_E (Relationen 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.

8.2 Ein Beispiel zum Modellieren in der Datenanalyse

Schauen wir uns ein Beispiel aus der Datenanalyse an; laden Sie dazu zuerst den Datensatz zur Statistikklausur.

Im ersten Plot von Abb. 8.4 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 nächsten Plot (Abb. 8.5) 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. 8.6):

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*.

²Diese Sichtweise des Modellierens basiert auf der Repräsentationstheorie des Messens nach Suppes und Zinnes (1962) zurück; vgl. Gigerenzer (1980)

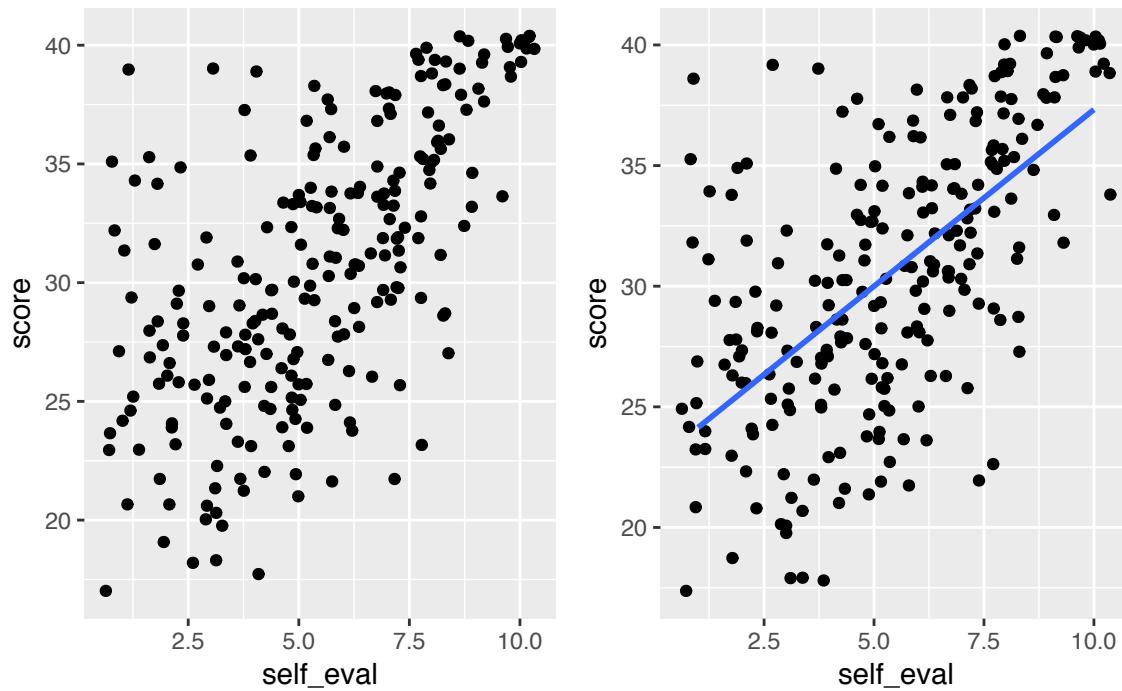


Abbildung 8.4: Ein Beispiel für Modellieren



Abbildung 8.5: Ein Beispiel für ein Pfadmodell

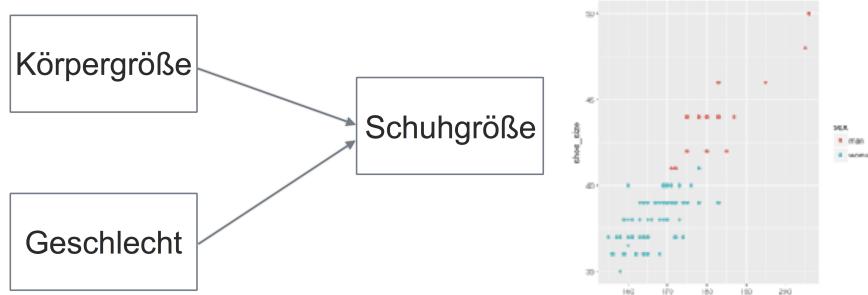


Abbildung 8.6: Ein etwas aufwändigeres Modell

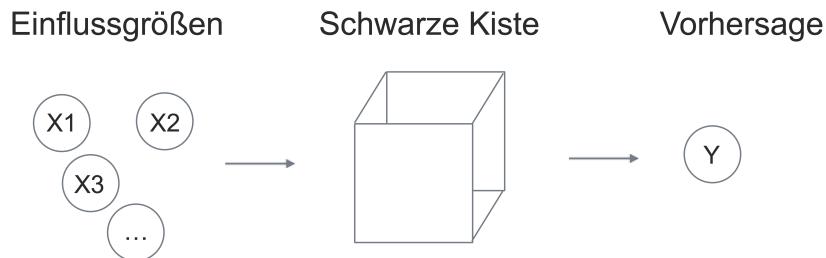


Abbildung 8.7: Modelle mit schwarzer Kiste



Einfluss ist hier *nicht* 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, 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.

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*;

³das ist schließlich entscheidend - frei nach Helmut Kohl

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.

8.3 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. 8.5. 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 Vorhersagefehler möglichst klein ist. Man zielt also darauf ab, möglichst exakte Vorhersagen zu treffen. Sicherlich wird der Vorhersagefehler kaum jemals Null sein; aber je präziser, desto besser. Das Innenleben der “schwarzen Kiste” interessiert uns hier *nicht*.

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.

Vorhersagen und Erklären haben gemein, dass Eingabegrößen genutzt werden, um Aussagen über einen Ausgabegröße zu treffen. 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 8.7 gibt diesen Fall wieder.

Beim *ungeleiteten Modellieren* entfällt die Unterteilung zwischen Prädiktor und Kriterium.

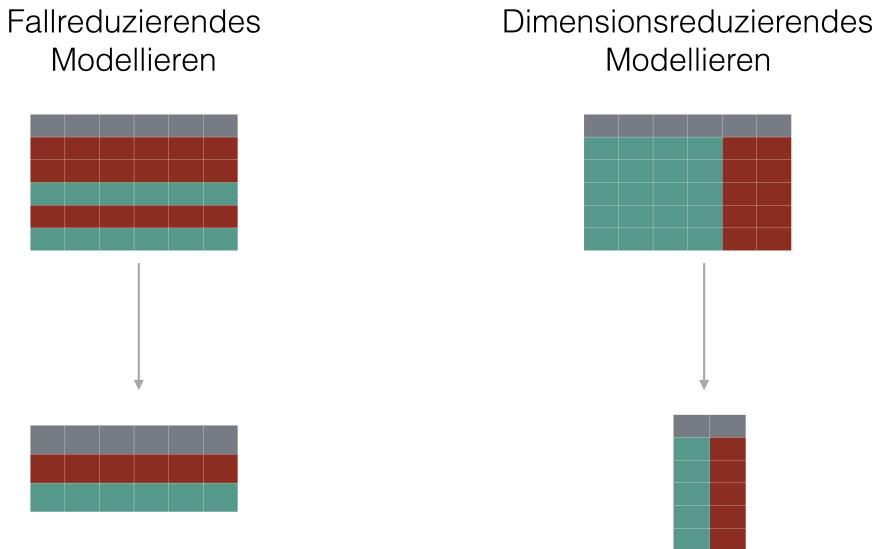


Abbildung 8.8: Die zwei Arten des ungeleiteten Modellierens

Ungeleitetes Modelieren (*Reduzieren*) meint, dass man die Fülle des Datenmaterials verringert, in dem man ähnliche Dinge zusammenfasst (vgl. Abb. 8.8).

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” von Herrn Casper Bussi-Ness ist. Wir sagen eher, “OK, die drei Typen sind sich irgendwie ähnlich, sie werden wohl zum selben Typen von Kunden gehören”. Wir tappen (noch mehr) im Dunkeln, was die “Wahrheit” ist. 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*).

8.4 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).

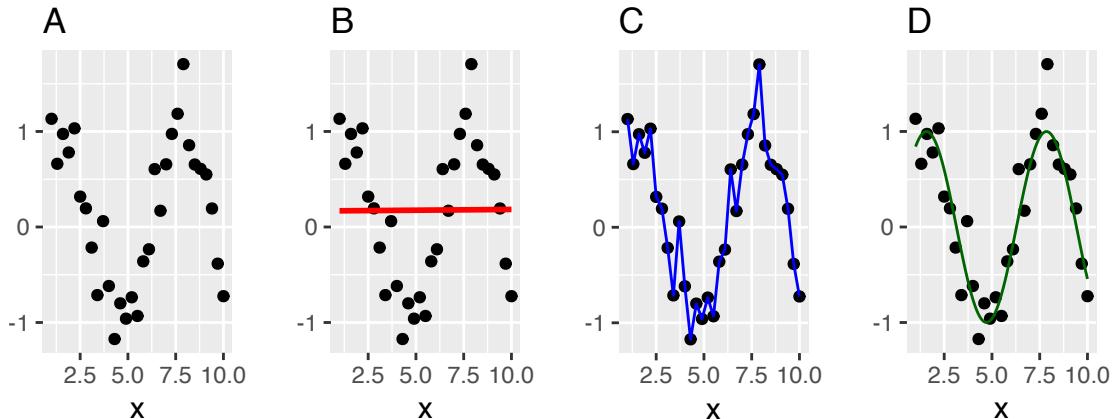


Abbildung 8.9: Welches Modell (Teil B-D; rot, grün, blau) passt am besten zu den Daten (Teil A) ?

2. Man wählt ein Modell aus (genauer: eine Modellfamilie), 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?

8.5 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. 8.9 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 “übergenaus” oder zu komplex. Man spricht von *Überanpassung* (engl. *overfitting*). Das Modell scheint zufälliges, bedeutungsloses Rauschen zu ernst zu nehmen. Das

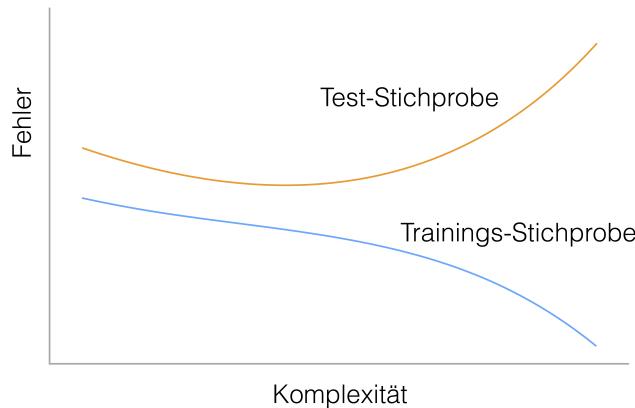


Abbildung 8.10: 'Mittlere' Komplexität hat die beste Vorhersagegenauigkeit (am wenigsten Fehler) in der Test-Stichprobe

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 erklären⁴.

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. 8.10)! Eine 'mittlere' Komplexität ist daher am besten; die Frage ist nur, wieviel 'mittel' ist.

⁴Tatsächlich wurden die Y-Werte als Sinus-Funktion plus etwas normalverteiltes Rauschen simuliert.

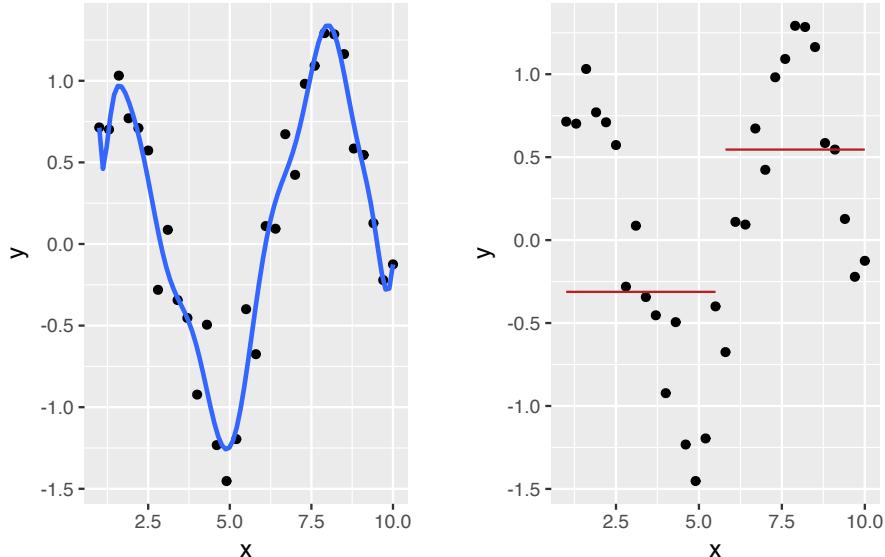


Abbildung 8.11: Der Spagat zwischen Verzerrung und Varianz

8.6 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.

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. 8.11; vgl. Kuhn & Johnson (2013)).

Der linke Plot zeigt ein komplexes Modell⁵; das Modell (blaue Linie) erscheint “zittrig”; kleine 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

⁵Genauer gesagt ein Polynom von Grad 5.

Daten werden vergleichsweise wenig Einfluss auf das Modell (die beiden roten Linien) haben.

8.7 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: 10.8).

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.

```
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 entwickeln können.

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. Wir bräuchten also noch ein Verfahren, welches *Variabilität* in der Modellgüte widerspiegelt.

8.8 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”, 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.

8.9 Modellgüte

Wie “gut” ist mein Modell? Modelle bewerten bzw. vergleichend bewerten ist einer der wichtigsten Aufgaben beim Modellieren. 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 Vorhersagen⁶. 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 (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

⁶Gerade wenn sie die Zukunft betreffen; ein Bonmot, das Yogi Berra nachgesagt wird.

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.

Ein Klassifikationsmodell muss anders beurteilt werden als ein Regressionsmodell. Reduktionsmodelle müssen wiederum anders beurteilt werden. In den entsprechenden Kapiteln werden diese Unterschiede präzisiert.

8.10 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.
- 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.

8.11 Aufgaben

1. Erfolg beim Online-Dating

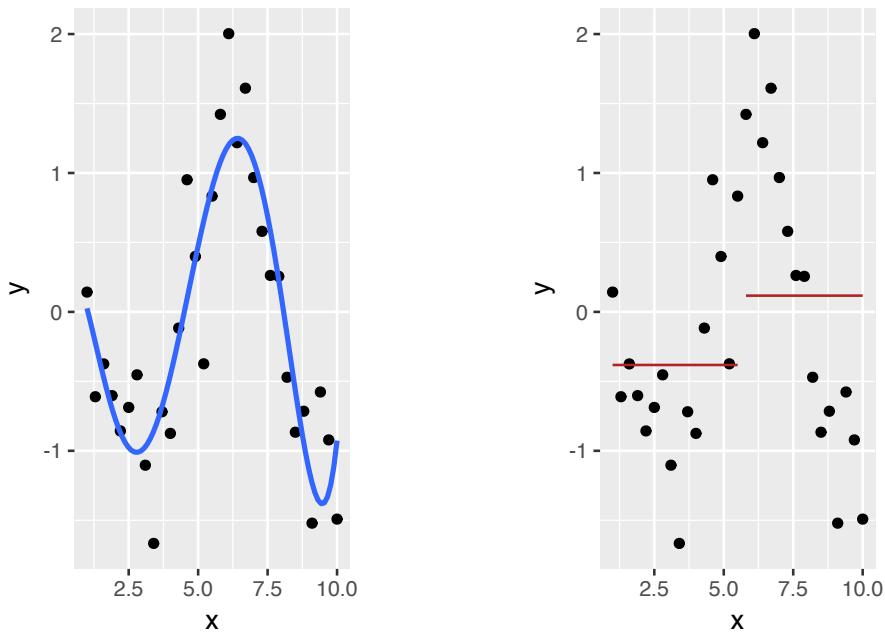


Abbildung 8.12: Bias-Varianz-Abwägung. Links: Wenig Bias, viel Varianz. Rechts: Viel Bias, wenig Varianz.

Lesen Sie diesen⁷ Artikel (Sauer und Wolff 2016). Zeichnen Sie ein Pfaddiagramm zum Modell!⁸.

2. Ziele des Modellierens

Welche drei Ziele des Modellierens kann man unterscheiden?⁹

3. Bias-Varianz-Abwägung

Betrachten Sie Abb. 8.12. Welches der beiden Modelle (visualisiert im jeweiligen Plot) ist wahrscheinlich...

- mehr bzw. weniger robust gegenüber Änderungen im Datensatz?
- mehr oder weniger präzise?

4. Richtig oder falsch?¹⁰



Richtig oder Falsch!?

1. Die Aussage "Pro Kilo Schoki steigt der Hüftumfang um einen Zentimeter" kann als Beispiel für ein deterministisches Modell herhalten.

⁷https://thewinnow.com/papers//5202-the-effect-of-a-status-symbol-on-success-in-online-dating-/an-experimental-study-data-paper?review_it=true

⁸Status → Erfolg beim Online-Dating

⁹8.3

¹⁰R, F, F, F, R

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.

8.12 Befehlsübersicht

Tabelle ?? fasst die R-Funktionen dieses Kapitels zusammen.

Tabelle 8.1: Befehle des Kapitels ‘Modellieren’

Paket	Funktion	Beschreibung
dplyr::sample_frac		Zielt eine Stichprobe von x% aus einem Dataframe
dplyr::anti_join		Behält alle Zeilen von df1, die <i>nicht</i> in df2 vorkommen
dplyr::slice		Schneidet eine ‘Scheibe’ aus einem Datensatz (filter Zeilen nach angegebener Start- und Endposition)

8.13 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 PDF¹¹.

¹¹<http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf>

Kapitel 9

Der p-Wert, Inferenzstatistik und Alternativen



Lernziele:

- 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)
library(compute.es)
library(tidyverse)
```

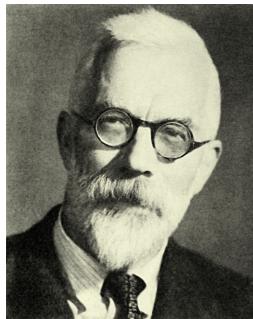


Abbildung 9.1: Der größte Statistiker des 20. Jahrhunderts ($p < .05$)

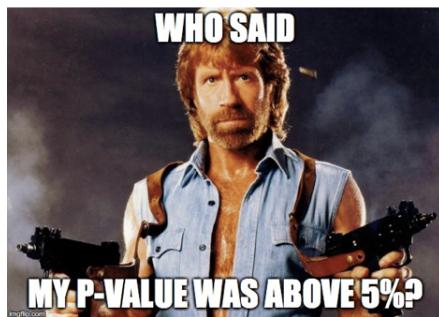


Abbildung 9.2: Der p-Wert wird oft als wichtig erachtet

9.1 Der p-Wert sagt nicht das, was viele denken

Der p-Wert, entwickelt von Sir Ronald Fisher (Abb. 9.1), ist die heilige Kuh der 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. 9.2).

Der p-Wert ist der tragende Ziegelstein in einem Theoriegebäude, das als *Nullhypothesen-Signifikanztesten* (NHST¹) bezeichnet wird. Oder kurz als ‘Inferenzstatistik’ bezeichnet. Was sagt uns der p-Wert? Eine gute intuitive Definition ist:

Der p-Wert sagt, wie gut die Daten zur Nullhypothese passen.

Die (genaue) 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

¹Der Term ‘Signifikanz-Hypothesen-Inferenz-Testen’ hat sich nicht durchgesetzt

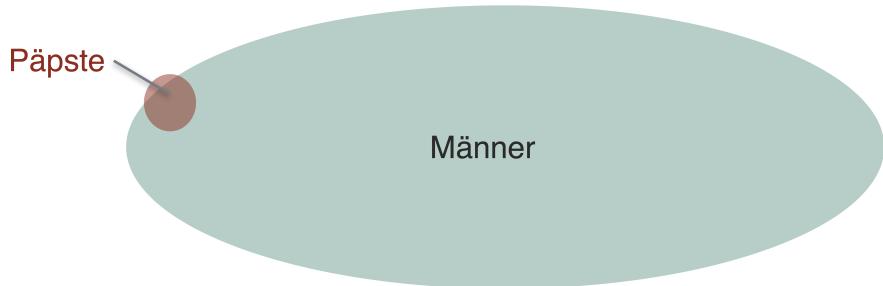


Abbildung 9.3: Moslem und Terrorist zu sein, ist nicht das gleiche.

vom Begriff ‘nullifizieren’ (vewerfen) 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 H₀.

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 - inkl. Professoren und 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:

$$P(D|H) \neq P(H|D)$$

9.1.1 Von Männern und Päpsten

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. 9.3) zeigt den Anteil der Männer an den Päpsten (sehr hoch). Und es zeigt

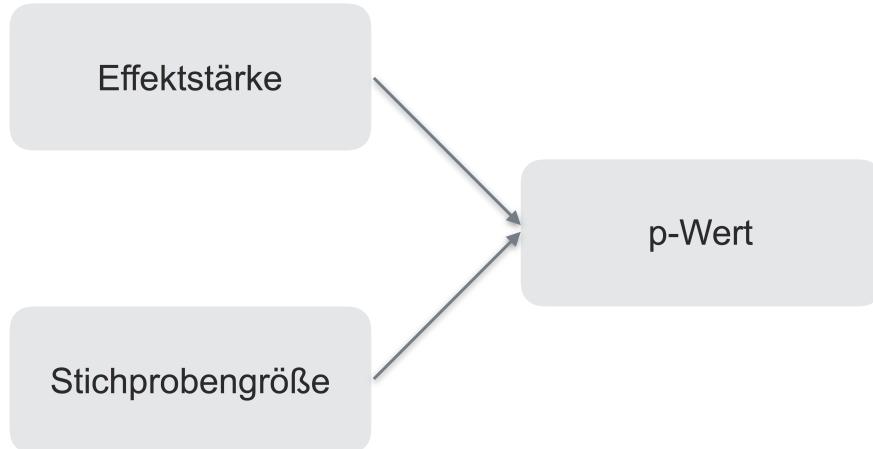


Abbildung 9.4: Zwei Haupeinflüsse auf den p-Wert

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 Mann ist*. Das ist falsch. Der p-Wert ist die *Wahrscheinlichkeit, Papst zu sein, wenn man Mann ist*. Ein großer Unterschied.

9.2 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. “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. 9.4. Die Details der Simulation, die hinter Abb. 9.4 sind etwas umfangreicher und hier nicht so wichtig, daher nicht angegeben².

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.

²s. hier für Details: https://sebastiansauer.github.io/pvalue_sample_size/

9.3 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 (H1) sicher richtig.

Richtig ist: "Wenn der p-Wert kleines ist als 5% (oder allgemeiner: kleiner als α , dann sind die Daten (oder noch extereme) unwahrscheinlich, vorausgesetzt die H0 gilt".

Wenn der p-Wert kleiner als 5% ist, dann ist meine Hypothese (H1) höchstwahrscheinlich richtig.

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 habe ich die Ursache eines Phänomens gefunden.

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?

9.3.1 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. Die folgende Auflistung gibt einen 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`

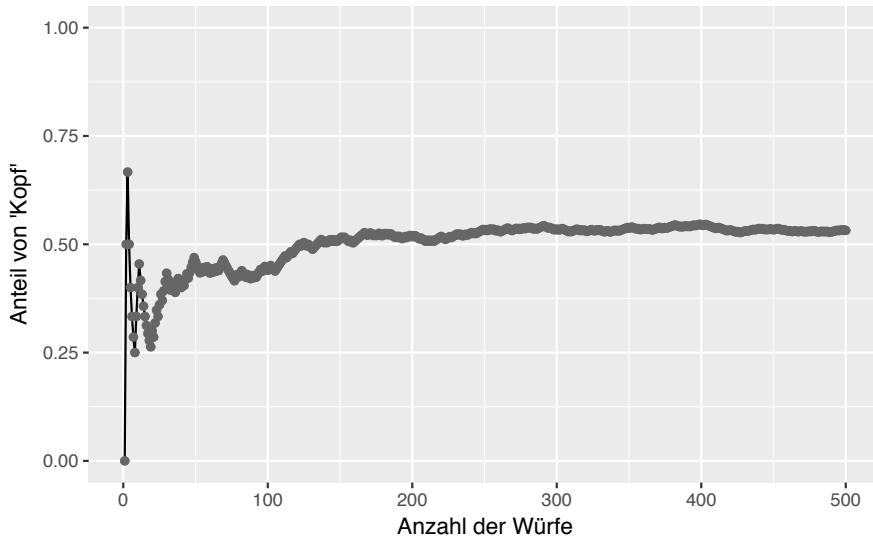


Abbildung 9.5: Anteil von 'Kopf' bei wiederholtem Münzwurf

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”)

9.4 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’. 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. 9.5). 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” (vgl. Briggs 2016). 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.

9.5 Alternativen zum p-Wert

Eine Reihe von Alternativen (oder Ergänzungen zum p-Wert) wurden vorgeschlagen.

9.5.1 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. 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 Konfidenzintervall³.

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 Aussagen⁴.

-
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 0,4.

³<http://rpsychologist.com/d3/CI/>

⁴alle sechs sind falsch

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, genau 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 könnte, 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).

9.5.2 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. So wäre ein Unterschied von 5€ bei Sportwagen gering und bei Eiskugeln groß. Ein unstandardisierter Kennwert berücksichtigt dies nicht. Um zwei Mittelwerte zu vergleichen, ist *Cohens d* gebräuchlich. Es gibt den Unterschied der Mittelwert standardisiert an der Standardabweichung an (Cohen 1988).

Tabelle ?? 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 9.1: Überblick über gängige Effektstärkemaße (continued below)

Name	Test	kleiner.Effekt
Cohens d	Unterschied zwischen zwei Mittelwerten	.2-.5
r	Zusammenhang zweier metrischer Größen	0.1
p	Unterschied in zwei Anteilen	NA

Name	Test	kleiner.Effekt
R^2 , η^2	Anteil aufgeklärter Varianz (Varianzanalyse, Regressionsanalyse)	0.01
f^2	Verhältnis von erklärter zu nicht erklärter Varianz (signal-to-noise ratio)	0.02
ω	Häufigkeitsunterschiede	0.1

mittlerer.Effekt	großer.Effekt
.5-.8	>.8
0.3	0.5
NA	NA
0.06	0.14
0.15	0.35
0.3	0.5

Mit dem Paket `pwr` kann man sich Cohens Konventionen der Effektstärkehöhen in Erinnerung rufen lassen. Er bietet folgene Optionen:

```
cohen.ES(test = c("p", "t", "r", "anov", "chisq", "f2"),
         size = c("small", "medium", "large"))
```

Möchte man sich Effektstärken berechnen lassen, ist das Paket `compute.es` hilfreich. Um beispielsweise Mittelwertsunterschiede in Cohens d umzurechnen, steht der Befehl `mes` (m wie 'mean' und es wie 'effect size') zur Verfügung. Mit `help(mes)` kann man sich die Parameter anzeigen lassen.

Auch die Vorhersagegüte kann man als eine Effektstärke auffassen.

9.5.3 Bayes-Statistik

Bayes' Ansatz verrechnet zwei Komponenten, um die Wahrscheinlichkeit einer Hypothese im Lichte bestimmter Daten zu berechnen. Der Ansatz ist elegant, mathematisch lupenrein 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: $p(D|H)$.

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 konkurrienden Hypothesen. Betrachten wir ein Beispiel. Die Hypothese "Ich bin krank" sei unter Betrachtung (jetzt noch

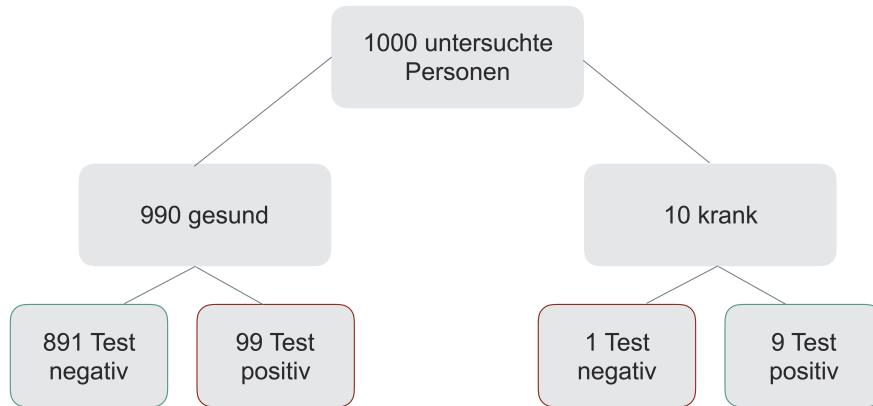


Abbildung 9.6: Die zwei Stufen der Bayes-Statistik in einem einfachen Beispiel

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 einschätzen (*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 9.6 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.

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 nicht⁵ (God 2016). Wir werden also eine Einschätzung treffen müssen, die subjektiv sein kann. Diese Subjektivität ist von Kritikern moniert worden.

⁵<https://twitter.com/TheTweetOfGod/status/688035049187454976>

9.6 Fazit

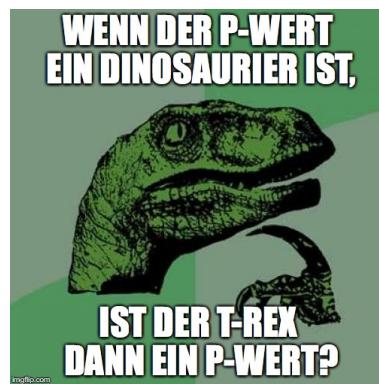
Der p-Wert ist eine häufig verwendete Methode, um datenbasiert zu entscheiden, ob man eine Hypothese annimmt oder nicht. Allerdings hat der p-Wert auch seine Probleme.

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



9.7 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 vielverprechende, noch recht neue Software ist JASP⁶, die nicht nur schöne Diagramme erstellt, sondern auch auf Mausklick eine Reihe von bayesianischer (und frequentistischer) Tests durchrechnet.

Teil III

Geleitetes Modellieren

Kapitel 10

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
```

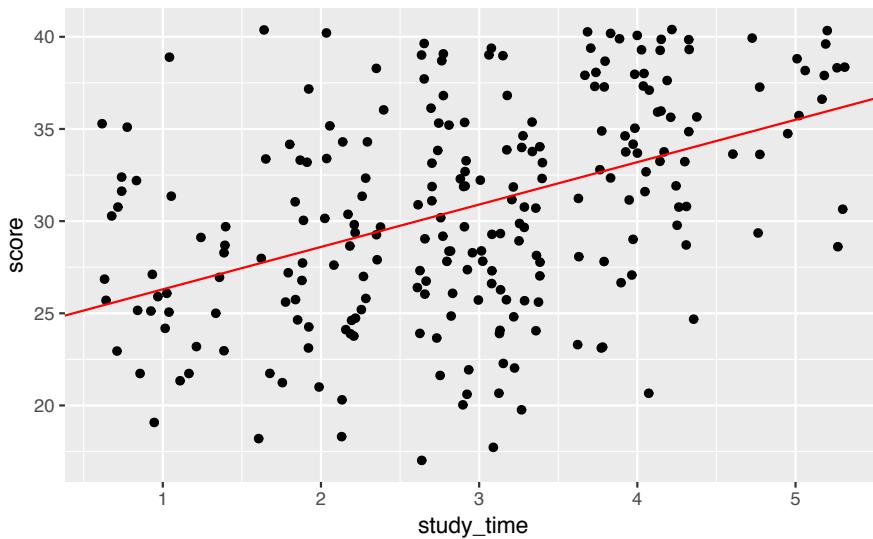


Abbildung 10.1: Beispiel für eine Regression

10.1 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. 10.1). Die Gerade ‘erklärt’ die Daten: Für jeden X-Wert liefert sie einen Y-Wert als Vorhersage zurück.

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

stats_test %>%
  ggplot +
  aes(x = study_time, y = score) +
  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 Punktewolke 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`

definiere das Diagramm (X-Achse, Y-Achse)

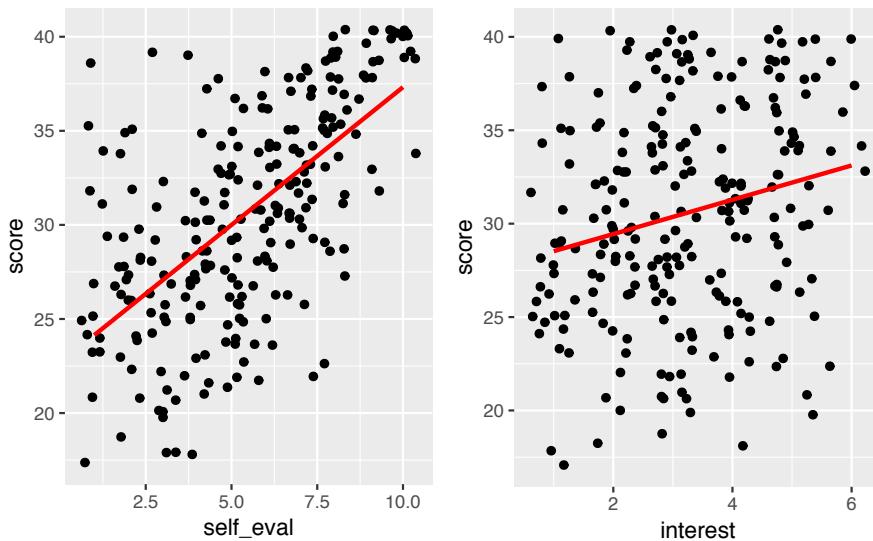


Abbildung 10.2: Zwei weitere Beispiele für Regressionen

zeichne das Geom “Jitter” (verwackeltes Punktediagramm) 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. 10.2).

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 einen anderen Blatt.

Man beachte, dass eine Gerade über ihre *Steigung* und ihren *Achsenabschnitt* festgelegt ist; in Abb. 10.1 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 `summary` nutzen:

```
mein_lm <- lm(kriterium ~ praediktor, data = meine_datentabelle)
summary(mein_lm)
```

Natürlich kann das auch ~~in der Pfeife rauchen~~ mit der Pfeife darstellen:

```
lm(kriterium ~ praediktor, data = meine_datentabelle) %>%
  summary
```

10.2 Vorhersagegüte

Der einfache Grundsatz lautet: Je geringer die Vorhersagefehler, desto besser; Abb. 10.3 zeigt ein Regressionsmodell mit wenig Vorhersagefehler (links) und ein Regressionsmodell mit viel Vorhersagefehler (rechts).

In einem Regressionsmodell lautet die grundlegenden Überlegung zur Modellgüte damit:

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 *Vorhersagefehler*, häufig mit ϵ (griechisches e wie “error”) abgekürzt.

Betrachten Sie die beiden Plots in Abb. 10.3. Die rote Linie gibt die *vorhergesagten* (geschätzten) Werte wieder; die Punkte die *beobachteten* (“echten”) Werte. Je länger die blauen

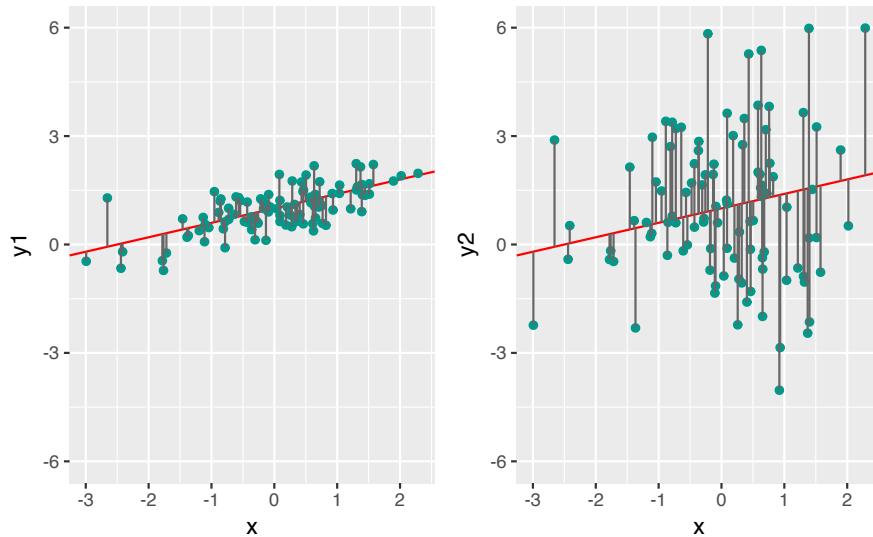


Abbildung 10.3: Geringer (links) vs. hoher (rechts) Vorhersagefehler

Linien, desto größer die Vorhersagefehler. Je größer der Vorhersagefehler, desto schlechter. Und umgekehrt.

Je kürzer die typische “Abweichungslinie”, desto besser die Vorhersage.

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 wieviel besser man als dieses Referenzkriterium ist.

10.2.1 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. 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.

10.2.2 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.



Man sollte in der Regel die Korrelation (r) nicht als Gütekriterium verwenden. Der Grund ist, dass die Korrelation sich nicht verändert, wenn man die Variablen skaliert. Die Korrelation zieht allein auf das Muster der Zusammenhänge - nicht die Größe der Abstände - ab. In der Regel ist die Größe der Abstände zwischen beobachteten und vorhergesagten Werten das, was uns interessiert.

10.3 Die Regression an einem Beispiel erläutert

Schauen wir uns den Datensatz zur Statistikklausur noch einmal an. Welchen Einfluss hat die Lernzeit auf den Klausurerfolg? Wieviel 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.

Versuchen wir im ersten Schritt die Stärke des Einfluss an einem Streudiagramm abzuschätzen (s. Abb. 10.1).

Hey R - berechne uns die "Trendlinie"! Dazu nimmt man den Befehl `lm`:

```
mein_lm <- lm(score ~ study_time, data = stats_test)
summary(mein_lm)
#>
#> Call:
#> lm(formula = score ~ study_time, data = stats_test)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -13.758  -3.693   0.242   3.983  12.760
#>
#> Coefficients:
```

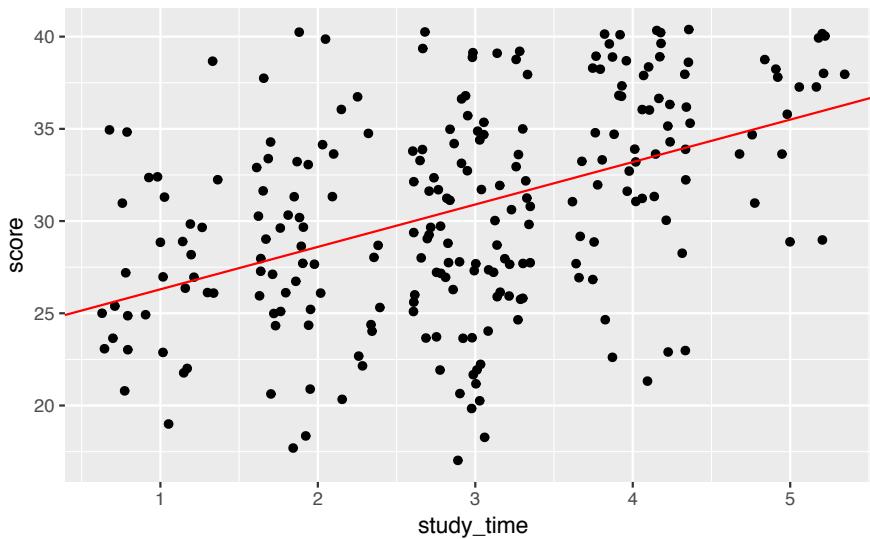


Abbildung 10.4: Streudiagramm von Lernzeit und Klausurerfolg

```
#>           Estimate Std. Error t value Pr(>|t|) 
#> (Intercept) 23.981     0.934   25.67 <2e-16 ***
#> study_time    2.259     0.300    7.54  1e-12 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.15 on 236 degrees of freedom
#> (68 observations deleted due to missingness)
#> Multiple R-squared:  0.194, Adjusted R-squared:  0.191 
#> F-statistic: 56.8 on 1 and 236 DF,  p-value: 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 10.4).

```
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 Punktewolke 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.

```
summary(mein_lm)
#>
#> Call:
#> lm(formula = score ~ study_time, data = stats_test)
#>
#> Residuals:
#>    Min      1Q  Median      3Q     Max
#> -13.758  -3.693   0.242   3.983  12.760
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 23.981     0.934  25.67 <2e-16 ***
#> study_time   2.259     0.300    7.54  1e-12 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.15 on 236 degrees of freedom
#>   (68 observations deleted due to missingness)
#> Multiple R-squared:  0.194,  Adjusted R-squared:  0.191
#> F-statistic: 56.8 on 1 and 236 DF,  p-value: 1.02e-12
```

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.

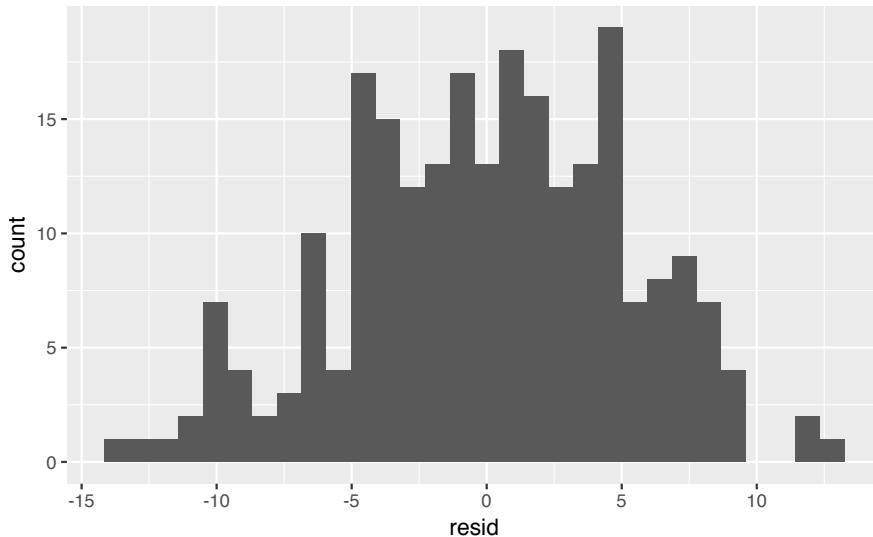


Abbildung 10.5: Die Residuen verteilen sich hinreichend normal.

10.4 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 ??). 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.

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) %>%
```

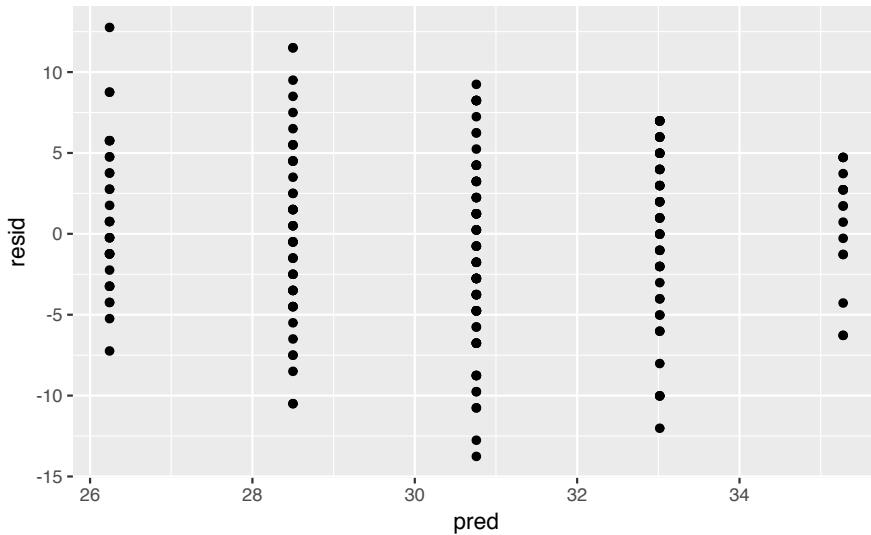


Abbildung 10.6: Vorhergesagte Werte vs. Residualwerte im Datensatz tips

```
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 10.6).

Die geschätzten (angepassten) Werte kann man über den Befehl `add_predictions()` aus 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?

10.5 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:

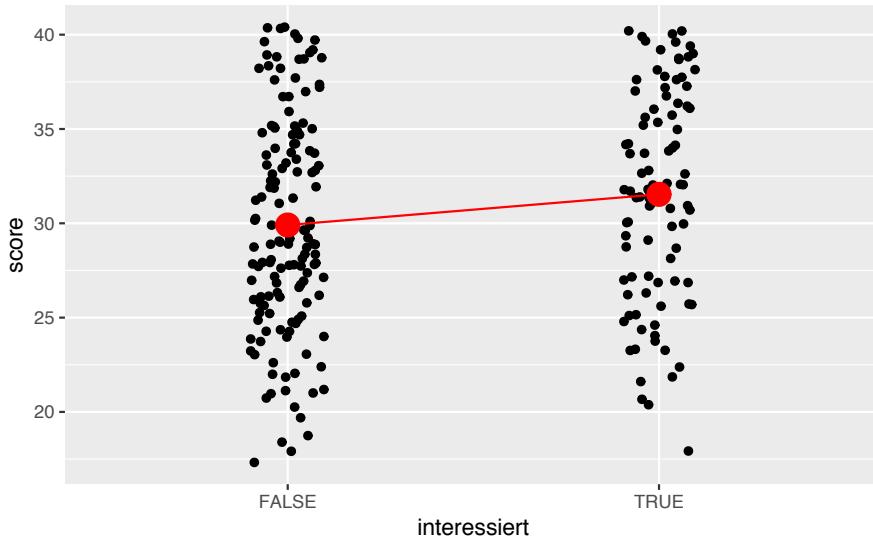
```
stats_test %>%
  group_by(interessiert) %>%
  summarise(score = mean(score)) -> score_interesse

score_interesse
#> # A tibble: 3 x 2
#>   interessiert score
#>   <lgl>    <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. Wir haben in dem Fall nur zwei Punkte, die entsprechend verbunden werden.

Visualisieren Sie den Gruppenunterschied auch mit einem Boxplot.

Und als 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 (zugunsten `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?

10.6 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
#>
#> 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
```

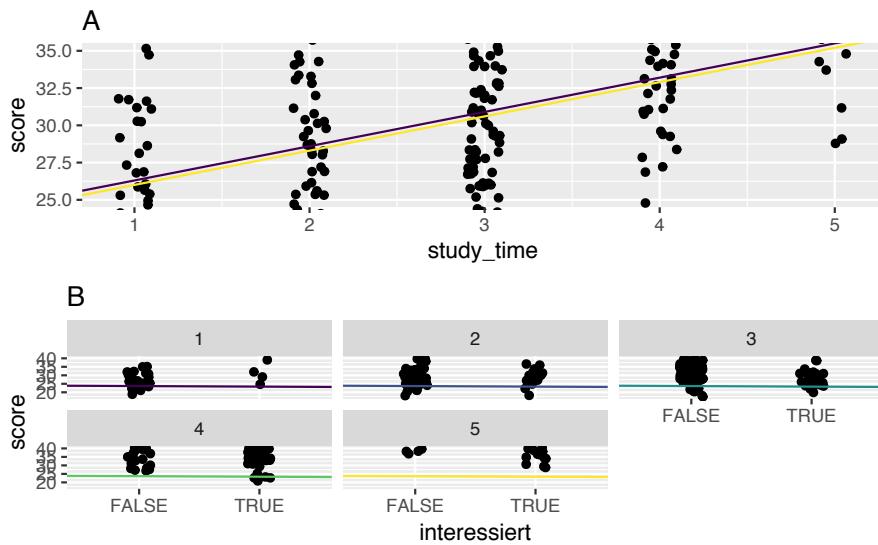


Abbildung 10.7: Eine multivariate Analyse fördert Einsichten zu Tage, die bei einfacheren Analysen verborgen bleiben

```
#> 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 10.7).

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 10.7): 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 Regressiongeraden in Abbildung 10.7 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.

10.7 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 Regressiongeraden in den beiden Gruppen unterschiedlich sein. Schauen wir uns es an. Um R dazu zu bringen, die Regressiongeraden 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)
#> Multiple R-squared:  0.196, Adjusted R-squared:  0.185
#> F-statistic: 19 on 3 and 234 DF,  p-value: 4.81e-11
```

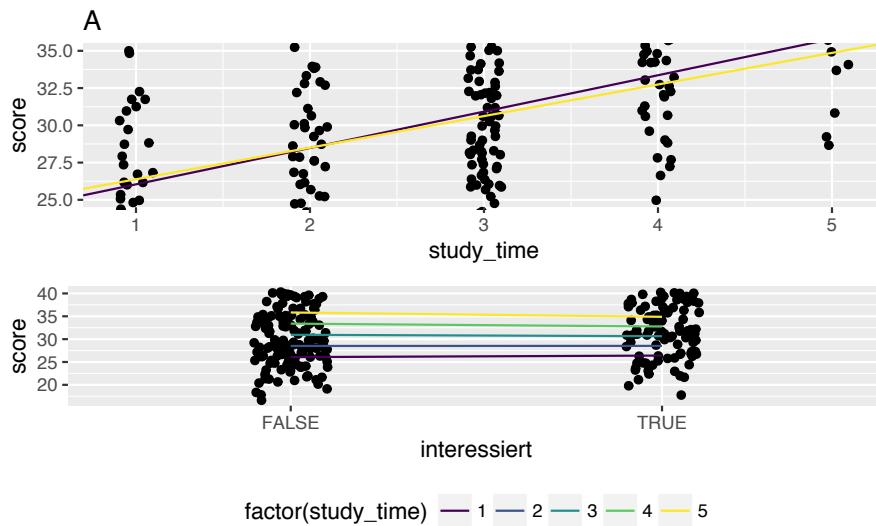


Abbildung 10.8: Eine Regressionsanalyse mit Interaktionseffekten

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 10.8, 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.

10.8 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.

10.9 Befehlsübersicht

Tabelle ?? stellt die Befehle dieses Kapitels dar.

Tabelle 10.1: Befehle des Kapitels ‘Regression’

Paket..Funktion	Beschreibung
lm	Berechnet eine Regression (“lm” steht für “lineares Modell”)
sqrt	Zieht die Quadratwurzel
caret::postResample	Berechnet Gütekriterien für das Testsample
summary	Fasst zentrale Informationen zu einem Objekt zusammen
modelr::add_residuals	Fügt eine Spalte mit den Residuen zu einem Dataframe hinzu
modelr::add_predictions	Fügt eine Spalte mit den vorhergesagten Werten zu einem Dataframe hinzu
levels	Zeigt oder ändert die Stufen eines Faktors
factor	Erstellt einen Faktor (nominalskalierte Variable)
coef	Zeigt die Koeffizienten eines Objekts z.B. vom typ “lm” an.
step	Führt eine Schrittweise-Rückwärtsselektion auf Basis des Akaike-Informationskriteriums für ein Regressionsmodell durch
sample_frac	Sampelt einen Prozentsatz aus einem Datensatz
anti_join	Fügt nicht-matchende Zeilen eines Datensatzes zu einem anderen Datensatz hinzu

Kapitel 11

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
```

11.1 Vorbereitung

Hier werden wir den Datensatz *Aktienkauf* der Universität Zürich (Universität Zürich, Methodenberatung¹) analysieren. Es handelt es sich um eine finanzpsychologische Fragestellung. Es wurde untersucht, welche Variablen mit der Wahrscheinlichkeit, dass jemand Aktien erwirbt, zusammenhängen. Insgesamt wurden 700 Personen befragt. Folgende Daten wurden erhoben:

- Aktienkauf (0 = nein, 1 = ja)
- Jahreseinkommen (in Tausend CHF),
- Risikobereitschaft (Skala von 0 bis 25)
- Interesse an der aktuellen Marktlage (Skala von 0 bis 45).

Importieren Sie zunächst die Daten.

```
Aktien <- readr::read_csv("data/Aktien.csv") %>% na.omit
```

Um uns das Leben leichter zu machen, haben wir fehlende Werte (NAs) mit `na.omit` gelöscht.

11.2 Problemstellung

Können wir anhand der Risikobereitschaft abschätzen, ob die Wahrscheinlichkeit für einen Aktienkauf steigt? Schauen wir uns zunächst ein Streudiagramm an (Abb. 11.1).

```
p1 <- ggplot(aes(y = Aktienkauf, x = Risikobereitschaft), data = Aktien) + geom_point()
```

Berechnen wir dann eine normale Regression.

```
lm1 <- lm(Aktienkauf ~ Risikobereitschaft, data = Aktien)
summary(lm1)
#>
#> Call:
#> lm(formula = Aktienkauf ~ Risikobereitschaft, data = Aktien)
#>
#> Residuals:
#>     Min      1Q Median      3Q     Max
#> -0.684 -0.243 -0.204  0.348  0.814
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
```

¹<http://www.methodenberatung.uzh.ch/de/datenanalyse/zusammenhaenge/lreg.html>

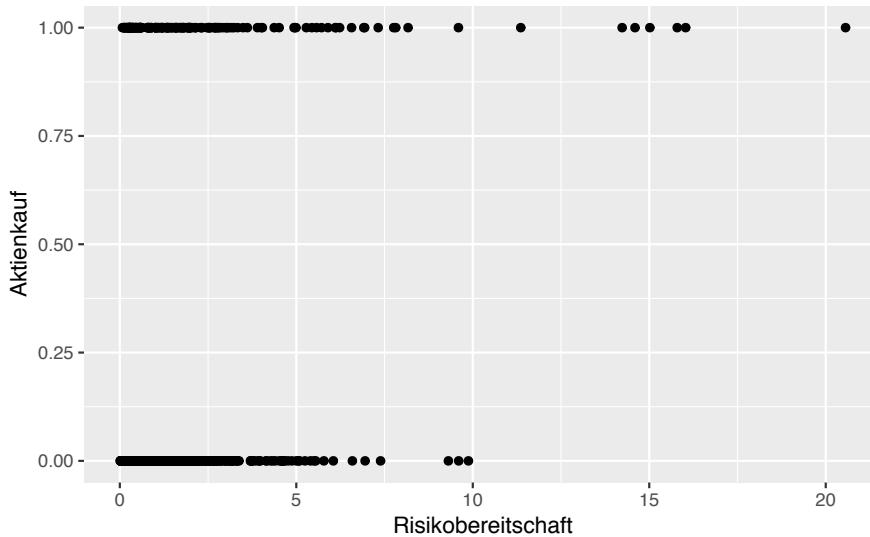


Abbildung 11.1: Streudiagramm von Risikobereitschaft und Aktienkauf

```
#> (Intercept)      0.18246    0.02001    9.12 < 2e-16 ***
#> Risikobereitschaft 0.05083    0.00762    6.67 5.2e-11 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.427 on 698 degrees of freedom
#> Multiple R-squared: 0.0599, Adjusted R-squared: 0.0586
#> F-statistic: 44.5 on 1 and 698 DF, p-value: 5.25e-11
```

Der Zusammenhang scheint nicht sehr ausgeprägt zu sein. Lassen Sie uns dennoch ein lineare Regression durchführen und das Ergebnis auswerten und graphisch darstellen.

```
p1 + geom_abline(intercept = .18, slope = .05, color = "red")
```

Der Schätzer für die Steigung für **Risikobereitschaft** ist signifikant. Das Bestimmtheitsmaß R^2 ist allerdings sehr niedrig, aber wir haben bisher ja auch nur eine unabhängige Variable für die Erklärung der abhängigen Variable herangezogen.

Doch was bedeutet es, dass die Wahrscheinlichkeit ab einer Risikobereitsschaft von ca. 16 über 1 liegt?

Wahrscheinlichkeiten müssen zwischen 0 und 1 liegen. 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:

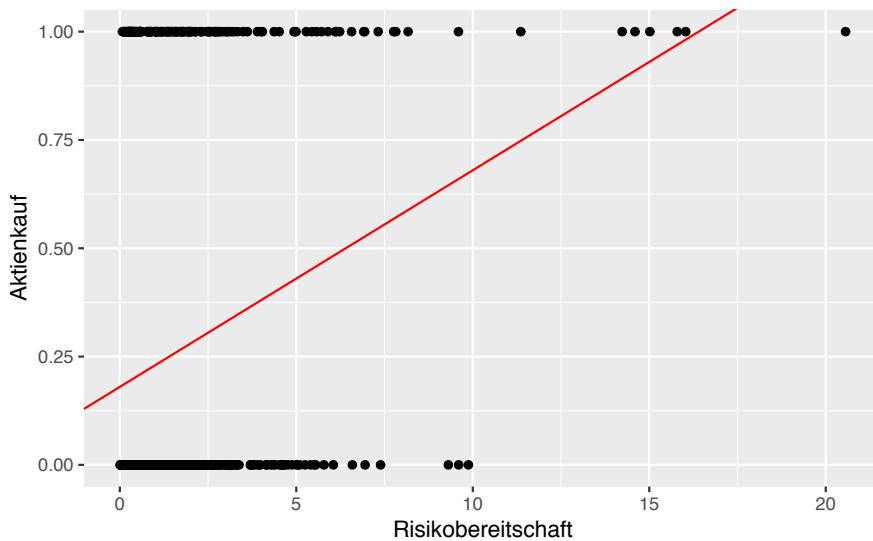


Abbildung 11.2: Regressionsgerade für Aktien-Modell

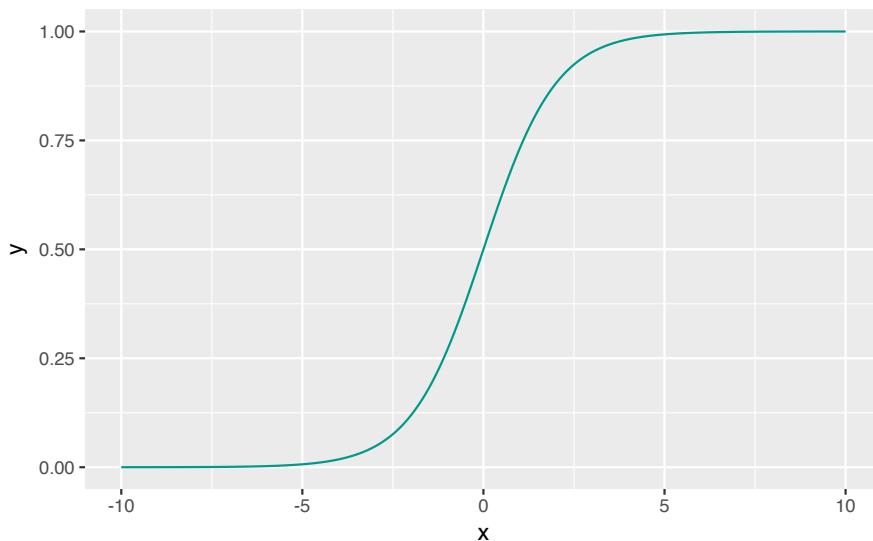


Abbildung 11.3: Die logistische Regression beschreibt eine 's-förmige' Kurve

$$p(y = 1) = \frac{e^x}{1 + e^x}$$

Exemplarisch können wir die logistische Funktion für einen Bereich von $\eta = -10$ bis $+10$ darstellen (vgl. 11.3). Der Graph der logistischen Funktion ähnelt einem langgestreckten S (“Ogive” genannt).

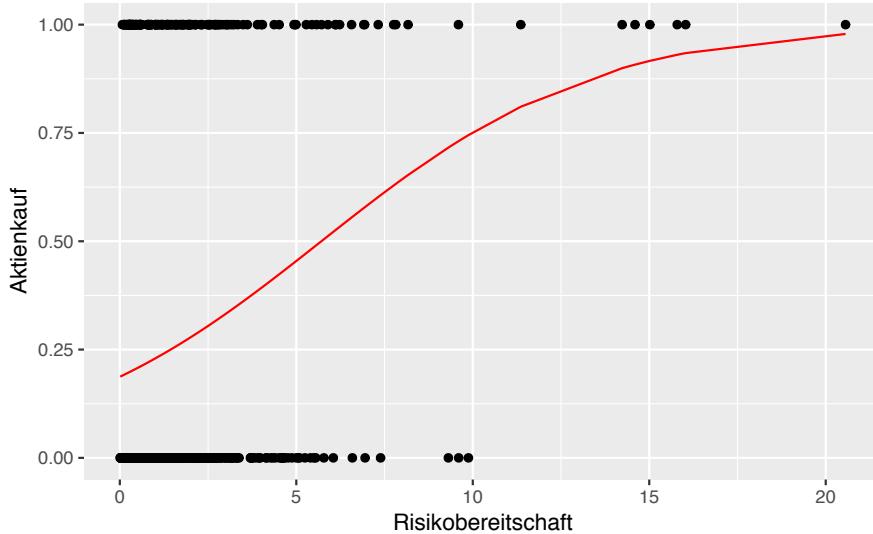


Abbildung 11.4: Modelldiagramm für den Aktien-Datensatz

11.3 Die Idee der logistischen Regression

Die logistische Regression ist eine Anwendung des *Allgemeinen Linearen Modells* (general linear model, GLM). Die Modellgleichung lautet:

$$p(y_i = 1) = L(\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_K \cdot x_{ik}) + \epsilon_i$$

- 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(Aktienkauf ~ Risikobereitschaft,
              family = binomial("logit"),
              data = Aktien)
```

Wir schauen uns zunächst den Plot an (Abb. 11.4).

Es werden ein Streudiagramm der beobachteten Werte sowie die *Regressionslinie* ausgegeben. Wir können so z. B. ablesen, dass ab einer Risikobereitschaft von etwa 7 die Wahrscheinlichkeit für einen Aktienkauf nach unserem Modell bei mehr als 50 % liegt.

Die Zusammenfassung des Modells zeigt folgendes:

```
summary(glm1)
#>
#> Call:
#> glm(formula = Aktienkauf ~ Risikobereitschaft, family = binomial("logit"),
#>       data = Aktien)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -1.653  -0.738  -0.677   0.825   1.823
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -1.4689    0.1184  -12.4 < 2e-16 ***
#> Risikobereitschaft 0.2573    0.0468     5.5 3.8e-08 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 804.36 on 699 degrees of freedom
#> Residual deviance: 765.86 on 698 degrees of freedom
#> AIC: 769.9
#>
#> Number of Fisher Scoring iterations: 4
```

Die p-Werte der Koeffizienten können in der Spalte $\text{Pr}(>|z|)$ abgelesen werden. Der Achsenabschnitt (`intercept`) wird mit -1.47 geschätzt, die Steigung in Richtung `Risikobereitschaft` mit 0.26. Allerdings sind die hier dargestellten Werte sogenannte *Logits L*:

$$\mathcal{L} = \ln\left(\frac{p}{1-p}\right)$$

Zugeben, dass klingt erstmal opaque. Das praktische ist, dass wir die Koeffizienten in gewohnter Manier verrechnen dürfen. Wollen wir zum Beispiel wissen, welche Wahrscheinlichkeit für Aktienkauf eine Person mit einer Risikobereitschaft von 3 hat, können wir einfach rechnen:

`y = intercept + 3*Risikobereitschaft`, also

```
(y <- -1.469 + 3 * 0.257)
#> [1] -0.698
```

Einfach, oder? Aber beachten Sie, dass das Ergebnis in *Logits* angegeben ist. Was ein Logit ist? Naja, der Logarithmus der Chancen (Wahrscheinlichkeit zu Gegenwahrscheinlichkeit, auch *Odds* genannt). Um zur ‘normalen’ Wahrscheinlichkeit zu kommen, muss man also erst ‘deologarithmieren’. Deologarithmieren bedeutet, die e-Funktion anzuwenden, `exp` auf Errisch:

```
exp(y)
#> [1] 0.498
```

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 Durchfallchance ist also 9:1 oder 9. Die Durchfallwahrscheinlichkeit 9/10 oder .9. Also kann man so umrechnen:

```
wskt = 9 / (9+1) = 9/10 = .9.
```

In unserem Fall sind die Chancen 0.322; also lautet die Umrechnung:

```
(wskt <- .498 / (.498+1))
#> [1] 0.332
```

Diesen Ritt kann man sich merklich kommorder bereiten, wenn man diesen Befehl kennt:

```
predict(glm1, newdata = data.frame(Risikobereitschaft = 3), type = "response")
#>      1
#> 0.332
```

11.4 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.

11.5 Interpretation der Koeffizienten

Ist ein Logit \mathcal{L} größer als 0, so ist die zugehörige Wahrscheinlichkeit größer als 50% (und umgekehrt.)

11.5.1 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.

11.5.2 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.

11.5.3 Aufgabe

Berechnen Sie das relative Risiko für unser Beispielmodell, wenn sich die **Risikobereitschaft** um 1 erhöht (Funktion `exp()`). Vergleichen Sie das Ergebnis mit der Punktprognose für **Risikobereitschaft= 7** im Vergleich zu **Risikobereitschaft= 8**.

Lösung:

```
# aus Koeffizient abgeschätzt
exp(coef(glm1)[2])
#> Risikobereitschaft
#> 1.29
```

In Worten: "Mit jedem Punkt mehr Risikobereitschaft steigen die Chancen (das OR) für Aktienkauf um 1.293".

```
# mit dem vollständigen Modell berechnet
predict(glm1, data.frame(Risikobereitschaft = 1),
       type = "response")
#> 1
#> 0.229

predict(glm1, data.frame(Risikobereitschaft = 8),
       type = "response")
#> 1
#> 0.643
```

Bei einer Risikobereitschaft von 1 beträgt die Wahrscheinlichkeit für $y = 1$, d.h. für das Ereignis "Aktienkauf", 0.23. Bei einer Risikobereitschaft von 8 liegt diese Wahrscheinlichkeit bei 0.64.

Sie sehen also, die ungefähr abgeschätzte Änderung der Wahrscheinlichkeit weicht hier doch deutlich von der genau berechneten Änderung ab. Der Anteil der Datensätze mit **Risikobereitschaft= 1** liegt allerdings auch bei 0.26.



Abbildung 11.5: Verwackeltes Streudiagramm ('Jitter')

11.6 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 <- read.csv("data/test_inf_short.csv")

stats_test$interessiert <- stats_test$interest > 3
```

Zunächst ein Plot (Abb. ??).

```
stats_test %>%
  na.omit %>%
  ggplot() +
  aes(x = interessiert, y = bestanden) +
  geom_jitter(width = .1)
```

Eine Sache sollten wir noch ändern: Auf der Y-Achse (`bestanden`) steht unten "ja" und oben "nein". Für unsere logistische Regression macht es aber genau anders herum Sinn: `bestanden=="ja")` soll oben stehen.`bestanden`` ist eine Variable vom Typ 'Faktor':

```
str(stats_test$bestanden)
#> Factor w/ 2 levels "ja","nein": 1 1 1 2 1 1 1 2 1 1 ...
```

Wir sehen, dass `bestanden` offenbar zwei Stufen hat (“ja” und “nein”). Die Reihenfolge der Stufen können wir so ändern:

```
stats_test$bestanden <- factor(stats_test$bestanden, levels = c("nein", "ja"))
```

Die neue Reihenfolge ist nicht nur optisch ansprechender, sondern auch für die logistische Regression sinnvoll: R berechnet die Wahrscheinlichkeit für die zweite Stufe (also die “obere” auf der Y-Achse); für R entspricht die erste Stufe 0 und die zweite Stufe 1. Außerdem sollten Achsen stets von wenig (nichts/nein) zu viel (ja) gehen.

Los geht's, probieren wir die logistische Regression aus:

```
log_stats <- glm(bestanden ~ interessiert,
                  family = binomial("logit"),
                  data = stats_test)
summary(log_stats)
#>
#> Call:
#> glm(formula = bestanden ~ interessiert, family = binomial("logit"),
#>       data = stats_test)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -2.034   0.520   0.520   0.633   0.633
#>
#> Coefficients:
#>              Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  1.504     0.217   6.94   4e-12 ***
#> interessiertTRUE 0.430     0.377   1.14   0.25
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 209.02 on 237 degrees of freedom
#> Residual deviance: 207.68 on 236 degrees of freedom
#> (68 observations deleted due to missingness)
#> AIC: 211.7
#>
#> Number of Fisher Scoring iterations: 4
```

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(log_stats, newdata = data.frame(interessiert = FALSE),
        type = "response")
#>     1
#> 0.818
predict(log_stats, newdata = data.frame(interessiert = TRUE),
        type = "response")
#>     1
#> 0.874
```

Also 82% bzw. 87%; kein gewaltig großer Unterschied.

11.7 Multiple logistische Regression

Wir kehren wieder zurück zu dem Datensatz *Aktienkauf*. Können wir unser Modell `glm1` mit nur einer erklärenden Variable verbessern, indem weitere unabhängige Variablen hinzugefügt werden?

```
glm2 <- glm(Aktienkauf ~ Risikobereitschaft + Einkommen + Interesse,
             family = binomial("logit"),
             data = Aktien)

summary(glm2)
#>
#> Call:
#> glm(formula = Aktienkauf ~ Risikobereitschaft + Einkommen + Interesse,
#>       family = binomial("logit"), data = Aktien)
#>
#> Deviance Residuals:
#>      Min        1Q    Median        3Q       Max
#> -2.130   -0.715   -0.539    0.518    3.214
#>
#> Coefficients:
#>              Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -1.66791   0.27903  -5.98  2.3e-09 ***
#> Risikobereitschaft 0.34781   0.08822   3.94  8.1e-05 ***
#> Einkommen     -0.02157   0.00564  -3.83  0.00013 ***
#> Interesse      0.08520   0.01775   4.80  1.6e-06 ***
#> ---
```

```
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 804.36 on 699 degrees of freedom
#> Residual deviance: 679.01 on 696 degrees of freedom
#> AIC: 687
#>
#> Number of Fisher Scoring iterations: 5
```

Alle Schätzer sind signifikant zum 0.1 %-Niveau (** in der Ausgabe). Zunehmende Risikobereitschaft (der Einfluss ist im Vergleich zum einfachen Modell stärker geworden) und zunehmendes Interesse erhöhen die Wahrscheinlichkeit für einen Aktienkauf. Steigendes Einkommen hingegen senkt die Wahrscheinlichkeit.

11.8 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 Abweichungssquarene 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 Abweichungssquarene ein R^2 berechnen kann; weil es kein ‘echtes’ R^2 ist, nennt man es auch *Pseudo-R²*.

Eine Reihe von R-Paketen bieten die Berechnung an:

```
library(BaylorEdPsych)
PseudoR2(glm1)
PseudoR2(glm2)
```

Die Ausgabe zeigt uns, dass `glm2` die Daten besser erklärt als `glm1`.

11.9 Klassifikationskennzahlen

11.9.1 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.

Tabelle 11.1: 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 (TN)	P
Summe	N*	P*	N+P

- 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.

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 11.1).

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(Aktien$Aktienkauf, glm1$fitted.values))
#>     obs
#> pred   0   1
#>      0 509 163
#>      1   8  20
#> attr(,"class")
#> [1] "confusion.matrix"
```

Dabei stehen **obs** (observed) für die wahren, also tatsächlich beobachteten Werte und **pred** (predicted) für die geschätzten (vorhergesagten) Werte.

Tabelle 11.2: Geläufige Kennwerte der Klassifikation

Name	Definition	Synonyme
Falsch-Positiv-Rate (FP-Rate)	FP/N	Alphafehler, Typ-1-Fehler, 1-Spezifität, Fehlalarm
Richtig-Positiv-Rate (RP-Rate)	RP/N	Power, Sensitivität, 1-Betafehler, Recall
Falsch-Negativ-Rate (FN-Rate)	FN/N	Fehlender Alarm, Befafehler
Richtig-Negativ-Rate (RN-Rate)	RN/N	Spezifität, 1-Alphafehler
Positiver Vorhersagewert	RP/P*	Präzision, Relevanz
Negativer Vorhersagewert	RN/N*	Segreganz
Gesamtgenauigkeitsrate	(RP+RN) / (N+P)	Richtigkeit, Korrektklassifikationsrate

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?

11.9.2 Klassifikationsgütekennzahlen

In der Literatur und Praxis herrscht eine recht wilde Vielfalt an Begriffen dazu, deswegen stellt Tabelle 11.1 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] 0.109
specificity(cm)
#> [1] 0.985
```

Wir haben die Aktienkäufer (1) also nur schlecht identifiziert; die Nichtkäufer (0) haben wir hingegen fast alle gefunden.

Wir könnten jetzt sagen, dass wir im Zweifel lieber eine Person als Käufer einschätzen (um ja keinen Kunden zu verlieren). Dazu würden wir den Schwellenwert (`threshold`) von 50% auf z.B. 30% herabsetzen:

```
(cm <- confusion.matrix(Aktien$Aktienkauf, glm1$fitted.values, threshold = .3))
#>      obs
#> pred   0    1
```

```
#>      0 440 128
#>      1 77 55
#> attr(,"class")
#> [1] "confusion.matrix"
sensitivity(cm)
#> [1] 0.301
specificity(cm)
#> [1] 0.851
```

11.9.3 ROC-Kurven

Siehe da! Die Sensitivität ist gestiegen, wir haben mehr Käufer als solche identifiziert. Unsere liberalere Strategie hat aber mehr falsch-positive Fälle produziert. 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 Daigramm ab, so bekommt man Abbildung 11.6, eine sog. *ROC-Kurve*.

```
lets_roc <- roc(Aktien$Aktienkauf, glm1$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 11.6 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 11.7 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

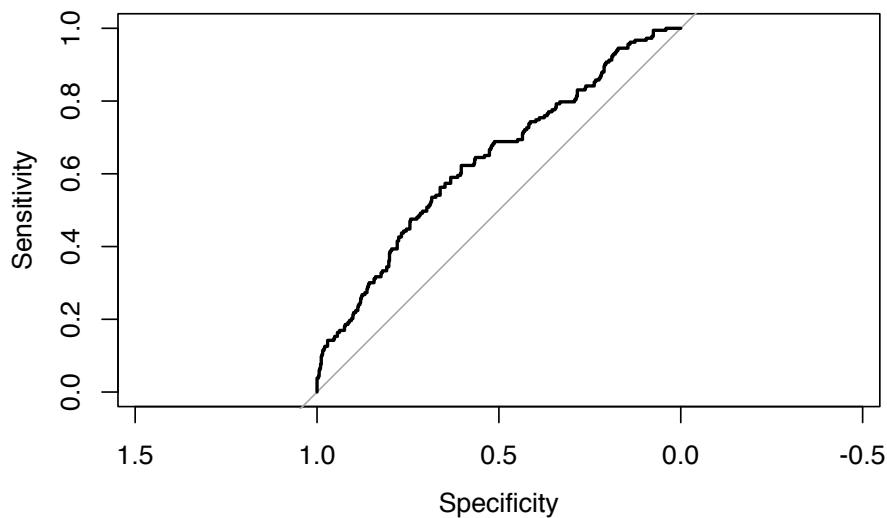


Abbildung 11.6: Eine ROC-Kurve

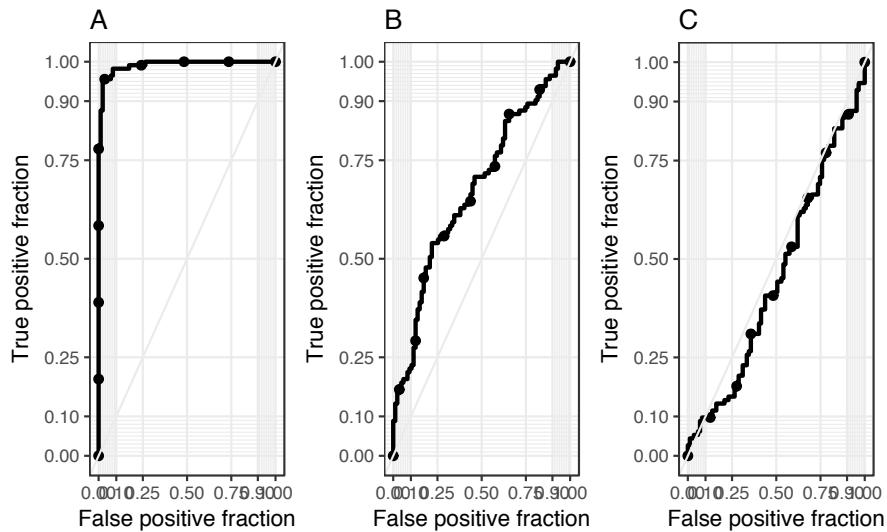


Abbildung 11.7: Beispiel für eine sehr gute (A), gute (B) und schlechte (C) Klassifikation

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
#>      0.229        0.603        0.623
```

11.10 Befehlsübersicht

Tabelle ?? stellt die Befehle dieses Kapitels dar.

Tabelle 11.3: Befehle des Kapitels ‘Logistische Regression’

Paket..Funktion	Beschreibung
ggplot2::geom_abline	Fügt das Geom “abline” (normale Gerade) zu einem ggplot2-Plot hinzu
glm	Berechnet ein “generalisiertes lineares Modell”, z.B. eine logistische Regression
exp	Berechnet die e-Funktion für den angegebenen Ausdruck (synonym: “deologarithmiert” den Ausdruck)
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

Kapitel 12

Fallstudien zum geleiteten Modellieren

12.1 Überleben auf der Titanic

In dieser YACSDA (Yet-another-case-study-on-data-analysis) 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 Passagier reist, ab?

12.1.1 Daten und Pakete laden

```
library("titanic")
data(titanic_train)
```

Man beachte, dass ein Paket nur *einmalig* zu installieren ist (wie jede Software). Dann aber muss das Paket bei jedem Starten von R wieder von neuem gestartet werden. Außerdem ist es wichtig zu wissen, dass das Laden eines Pakets nicht automatisch die Datensätze aus dem Paket lädt. Man muss das oder die gewünschten Pakete selber (mit `data(...)`) laden. Und: Der Name eines Pakets (z.B. `titanic`) muss nicht identisch sein mit dem oder den Datensätzen des Pakets (z.B. `titanic_train`).

```
library(tidyverse)
```

12.1.2 Erster Blick

Werfen wir einen ersten Blick in die Daten:

```
# install.packages("dplyr", dependencies = TRUE) # ggf. vorher installieren
glimpse(titanic_train)
#> Observations: 891
#> Variables: 12
#> $ PassengerId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
#> $ Survived <int> 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, ...
#> $ Pclass <int> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, ...
#> $ Name <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bra...
#> $ Sex <chr> "male", "female", "female", "female", "male", "mal...
#> $ Age <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, ...
#> $ SibSp <int> 1, 1, 0, 1, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, ...
#> $ Parch <int> 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, ...
#> $ Ticket <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "1138...
#> $ Fare <dbl> 7.25, 71.28, 7.92, 53.10, 8.05, 8.46, 51.86, 21.07...
#> $ Cabin <chr> "", "C85", "", "C123", "", "", "E46", "", "", "", ...
#> $ Embarked <chr> "S", "C", "S", "S", "Q", "S", "S", "C", ...
```

12.1.3 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.

12.1.4 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)?

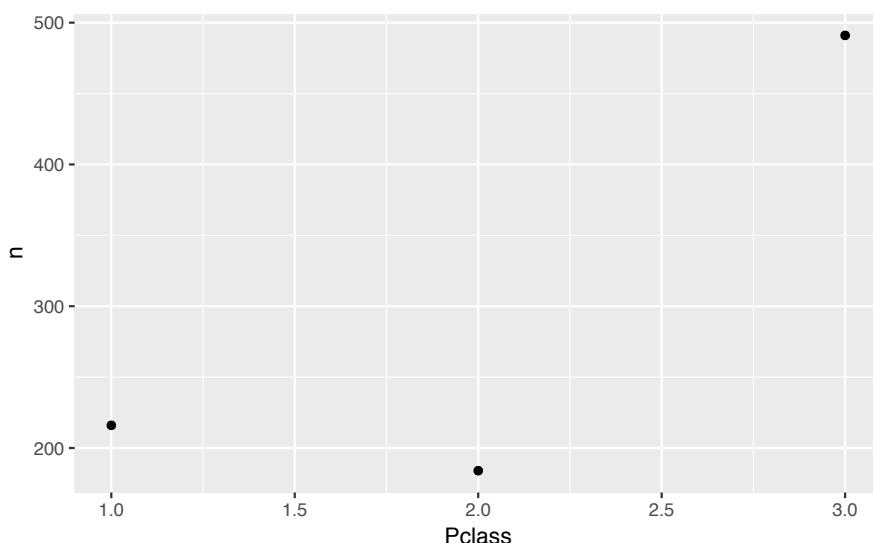
```
c1 <- dplyr::count(titanic_train, Pclass)
c1
#> # A tibble: 3 x 2
#>   Pclass     n
#>   <int> <int>
#> 1     1    216
#> 2     2    184
#> 3     3    491
```



Achtung - Namenskollision! Sowohl im Paket `mosaic` als auch im Paket `dplyr` gibt es einen Befehl `count`. Für `select` gilt das gleiche. 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 `::`.

Aha. Zur besseren Anschaulichkeit können wir das auch plotten (ein Diagramm dazu malen).

```
# install.packages("ggplot2", dependencies = TRUE)
qplot(x = Pclass, y = n, data = c1)
```



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    549
#> 2     1    342
```

Man beachte, dass der Befehl `count` stehts eine Tabelle (`data.frame` bzw. `tibble`) verlangt und zurückliefert.

12.1.5 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?

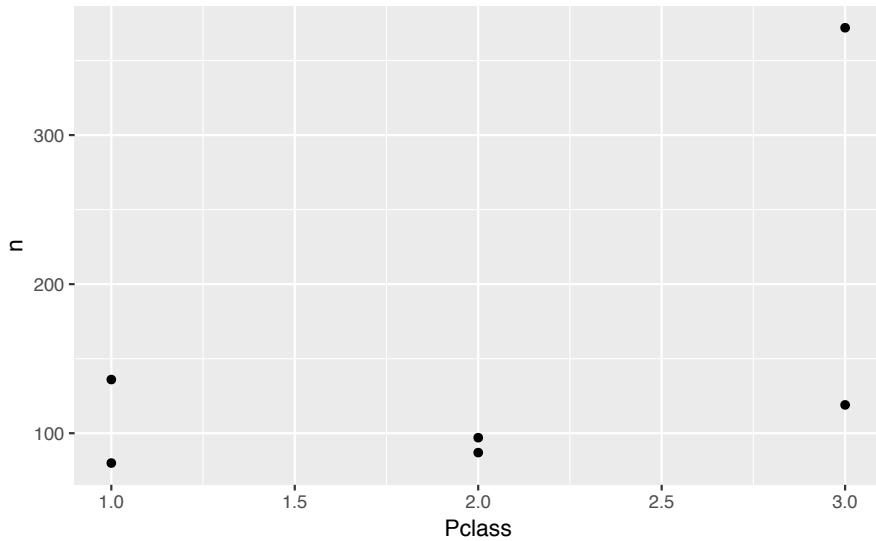
Zählen wir zuerst die Häufigkeiten für alle Kombinationen von `Survived` und `Pclass`:

```
c3 <- dplyr::count(titanic_train, Survived, Pclass)
c3
#> Source: local data frame [6 x 3]
#> Groups: Survived [?]
#>
#> # A tibble: 6 x 3
#>   Survived Pclass     n
#>   <int> <int> <int>
#> 1     0     1     80
#> 2     0     2     97
#> 3     0     3    372
#> 4     1     1    136
#> 5     1     2     87
#> 6     1     3    119
```

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.

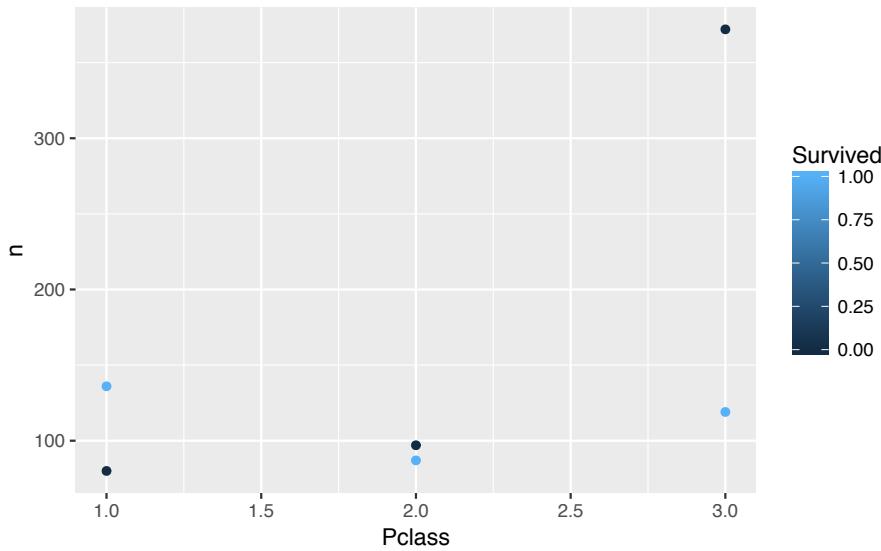
Es ist hilfreich, sich diese Häufigkeiten wiederum zu plotten; wir nehmen den gleichen Befehl wie oben.

```
qplot(x = Pclass, y = n, data = c3)
```



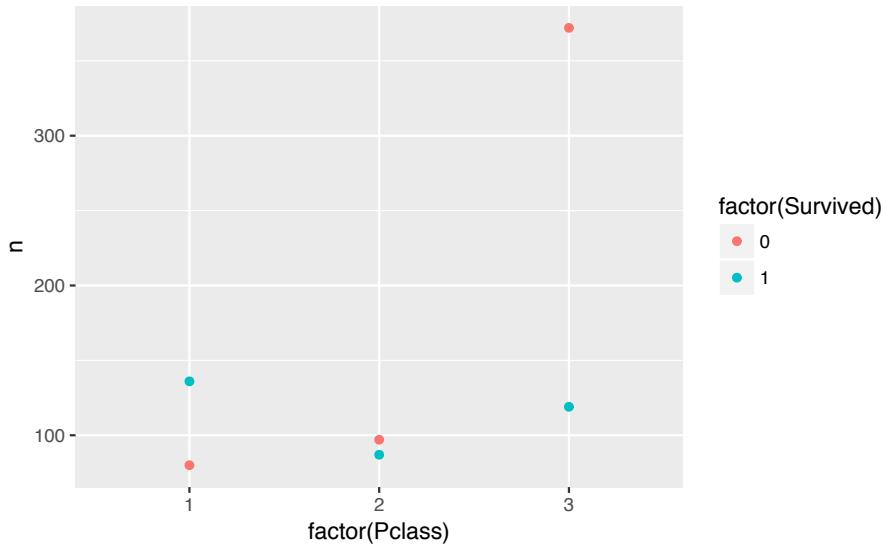
Hm, nicht so hilfreich. Schöner wäre, wenn wir (farblich) erkennen könnten, welcher Punkt für “Überlebt” und welcher Punkt für “Nicht-Überlebt” steht. Mit `qplot` geht das recht einfach: Wir sagen der Funktion `qplot`, dass die Farbe (`color`) der Punkte den Stufen von `Survived` zugeordnet werden sollen:

```
qplot(x = Pclass, y = n, color = Survived, data = c3)
```



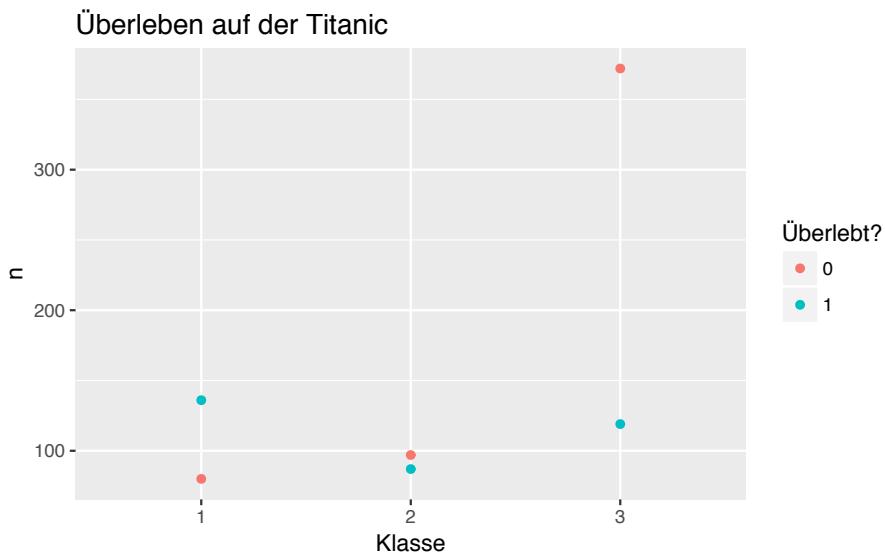
Viel besser. Was noch stört, ist, dass `Survived` als metrische Variable verstanden wird. Das Farbschema lässt Nuancen, feine Farbschattierungen, zu. Für nominale Variablen macht das keinen Sinn; es gibt da keine Zwischentöne. Tot ist tot, lebendig ist lebendig. Wir sollten daher der Funktion sagen, dass es sich um nominale Variablen handelt:

```
qplot(x = factor(Pclass), y = n, color = factor(Survived), data = c3)
```



Viel besser. Jetzt noch ein bisschen Schnickschnack:

```
qplot(x = factor(Pclass), y = n, color = factor(Survived), data = c3) +
  labs(x = "Klasse",
       title = "Überleben auf der Titanic",
       colour = "Überlebt?")
```



12.1.6 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 (er gibt *nicht* die Wahrscheinlichkeit der getesteten Hypothese an, was die Frage aufwirft, warum er mich dann interessieren sollte). Aber sei drum, berechnen wir mal einen p-Wert.

Es gibt mehrere statistische Tests, die sich hier potenziell anbieten (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 = 100, 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”.

12.1.7 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    216
#> 2     3    491
```

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 = 100, 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).

```
library(compute.es)
chies(chi.sq = 96, n = 707)
#> Mean Differences ES:
#>
#> d [ 95 %CI] = 0.79 [ 0.63 , 0.95 ]
#> var(d) = 0.01
#> p-value(d) = 0
#> U3(d) = 78.6 %
#> CLES(d) = 71.2 %
#> Cliff's Delta = 0.42
#>
#> g [ 95 %CI] = 0.79 [ 0.63 , 0.95 ]
#> var(g) = 0.01
#> p-value(g) = 0
#> U3(g) = 78.6 %
#> CLES(g) = 71.2 %
#>
#> Correlation ES:
#>
#> r [ 95 %CI] = 0.37 [ 0.3 , 0.43 ]
#> var(r) = 0
#> p-value(r) = 0
#>
#> z [ 95 %CI] = 0.39 [ 0.31 , 0.46 ]
#> var(z) = 0
#> p-value(z) = 0
#>
#> Odds Ratio ES:
#>
```

```
#> OR [ 95 %CI] = 4.21 [ 3.15 , 5.61 ]
#> p-value(OR) = 0
#>
#> Log OR [ 95 %CI] = 1.44 [ 1.15 , 1.73 ]
#> var(lOR) = 0.02
#> p-value(Log OR) = 0
#>
#> Other:
#>
#> NNT = 3.57
#> Total N = 707
```

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...

12.1.8 Logististische Regression

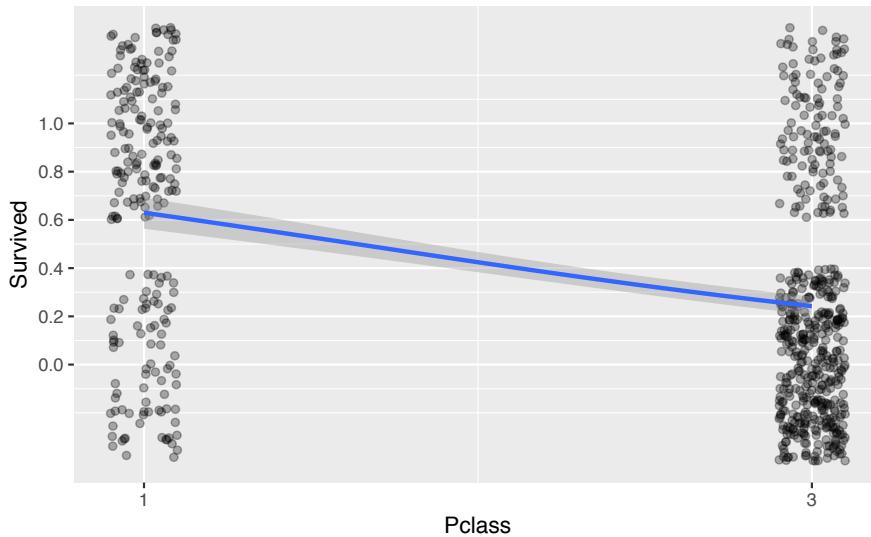
Berechnen wir noch das Odds Ratio mit Hilfe der logistischen Regression. Zum Einstieg: Ignorieren Sie die folgende Syntax und schauen Sie sich das Diagramm an. Hier sehen wir die (geschätzten) Überlebens-Wahrscheinlichkeiten für Passagiere der 1. Klasse vs. Passagiere der 3. Klasse.

```
titanic2 <- titanic_train %>%
  filter(Pclass %in% c(1,3)) %>%
  mutate(Pclass = factor(Pclass))

glm1 <- glm(data = titanic2,
            formula = Survived ~ Pclass,
            family = "binomial")

exp(coef(glm1))
#> (Intercept)      Pclass3
#>           1.700      0.188

titanic2$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 (delogarithmiert, `boa`). Wir sehen, dass die Überlebens-*Chance* (Odds) 1.7 zu 1 betrug - bei der *ersten* Stufe von `Pclass` (1)¹; von 27 Menschen überlebten in dieser Gruppe also 17 ($17/27 = .63$ Überlebens-*Wahrscheinlichkeit*); s. `Intercept`; der Achsenabschnitt gibt den Odds an, wenn die Prädiktor-Variable(n) den Wert “Null” hat/ haben, bzw. die erste Ausprägung, hier 1.

Im Vergleich dazu wird die Überlebens-Chance deutlich schlechter, wenn man die nächste Gruppe von `Pclass` (3) betrachtet. Die Odds verändern sich um den Faktor ~0.2. Da der Faktor *kleiner* als 1 ist, ist das kein gutes Zeichen. Die Überlebens-Chance *sinkt*; etwas genauer auf: $1.7 * 0.2 \approx 0.34$. Das heißt, die Überlebens-Chance ist in der 3. Klasse nur noch ca. 1 zu 3 (Überlebens-*Wahrscheinlichkeit*: ~25%).

Komfortabler können wir uns die Überlebens-*Wahrscheinlichkeiten* mit der Funktion `predict` ausgeben lassen.

```
predict(glm1, newdata = data.frame(Pclass = factor("1")), type = "response")
#>     1
#> 0.63
```

¹Darum haben wir `Pclass` in eine Faktor-Variable umgewandelt. Die “erste Klasse” ist jetzt die Referenzklasse, also sozusagen $x = 0$. Hätten wir `Pclass` als numerische Variable beibehalten, so würde der Achsenabschnitt die Überlebensrat für die “nullte” Klasse geben, was wenig Sinn macht.

```
predict(glm1, newdata = data.frame(Pclass = factor("3")), type = "response")
#>      1
#> 0.242
```

Alternativ kann man die 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))
#> Source: local data frame [4 x 4]
#> Groups: Pclass [2]
#>
#> # A tibble: 4 x 4
#>   Pclass Survived     n Anteil
#>   <int>    <int> <int>  <dbl>
#> 1     1        0     80  0.370
#> 2     1        1    136  0.630
#> 3     3        0    372  0.758
#> 4     3        1    119  0.242
```

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.

12.1.9 Effektstärken visualieren

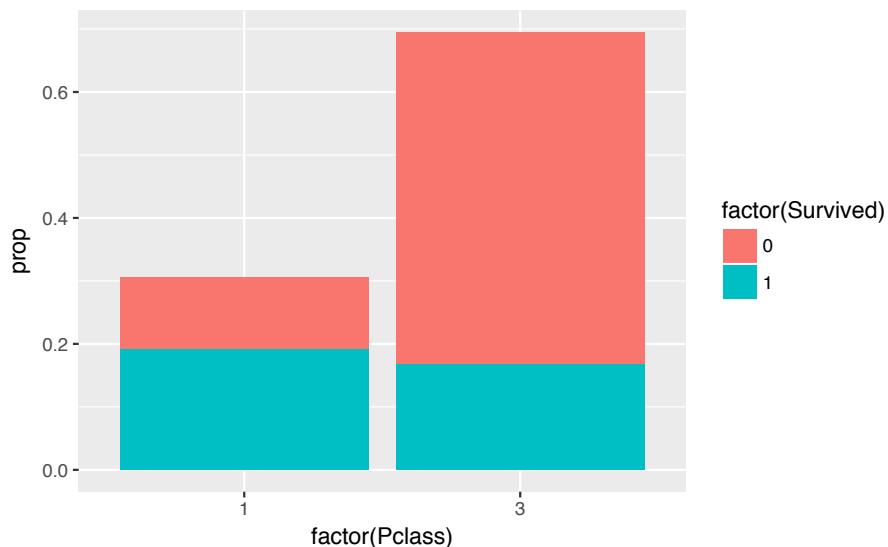
Zum Abschluss schauen wir uns die Stärke des Zusammenhangs noch einmal graphisch an. Wir berechnen dafür die relativen Häufigkeiten pro Gruppe (im Datensatz ohne 2. Klasse, der Einfachheit halber).

```
c5 <- dplyr::count(t2, Pclass, Survived)
c5$prop <- c5$n / 707
c5
#> Source: local data frame [4 x 4]
#> Groups: Pclass [?]
#>
#> # A tibble: 4 x 4
#>   Pclass Survived     n   prop
#>   <int>   <int> <int> <dbl>
#> 1     1       0     80  0.113
#> 2     1       1    136  0.192
#> 3     3       0    372  0.526
#> 4     3       1    119  0.168
```

Genauer gesagt haben die Häufigkeiten pro Gruppe in Bezug auf die Gesamtzahl aller Passagiere berechnet; die vier Anteile addieren sich also zu 1 auf.

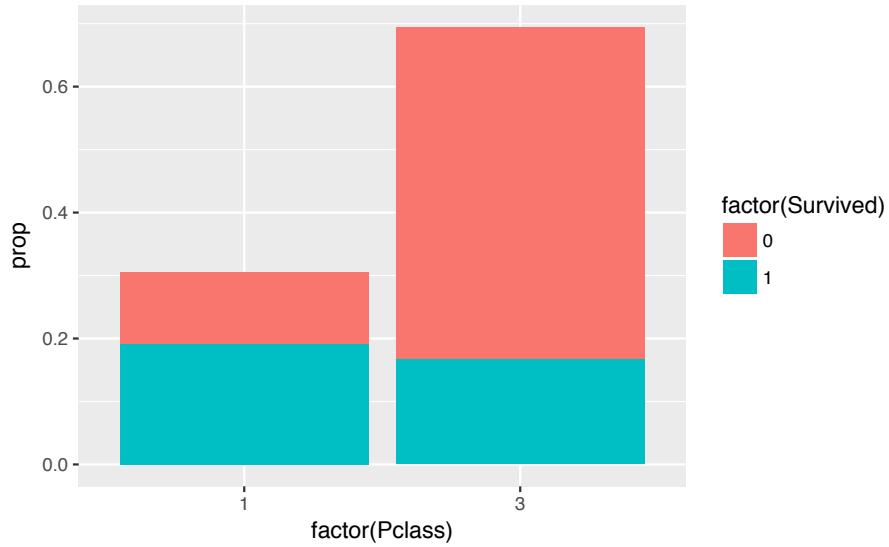
Das visualisieren wir wieder

```
qplot(x = factor(Pclass),
      y = prop,
      fill = factor(Survived),
      data = c5,
      geom = "col")
```



Das `geom = "col"` heißt, dass als “geometrisches Objekt” dieses Mal keine Punkte, sondern Säulen (columns) verwendet werden sollen.

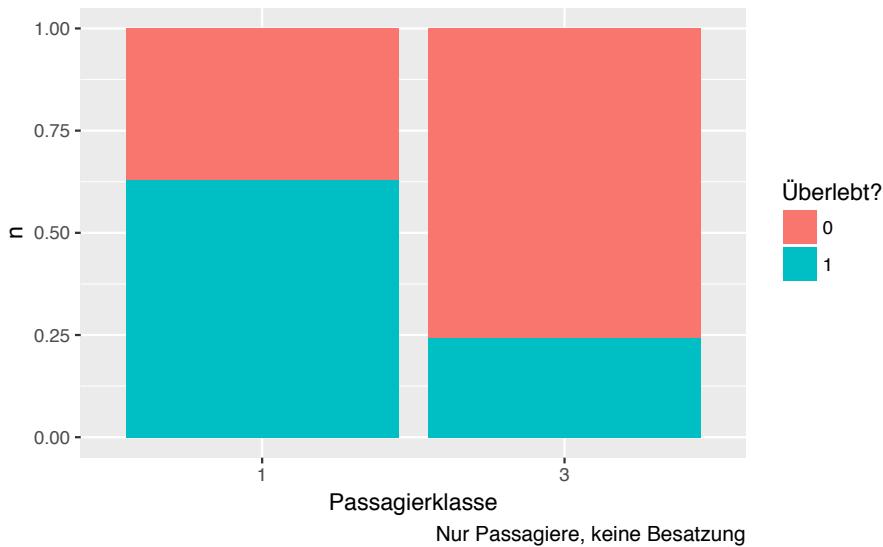
```
qplot(x = factor(Pclass), y = prop, fill = factor(Survived), data = c5, geom = "col")
```



Ganz nett, aber die Häufigkeitsunterscheide 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:

```
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")
```

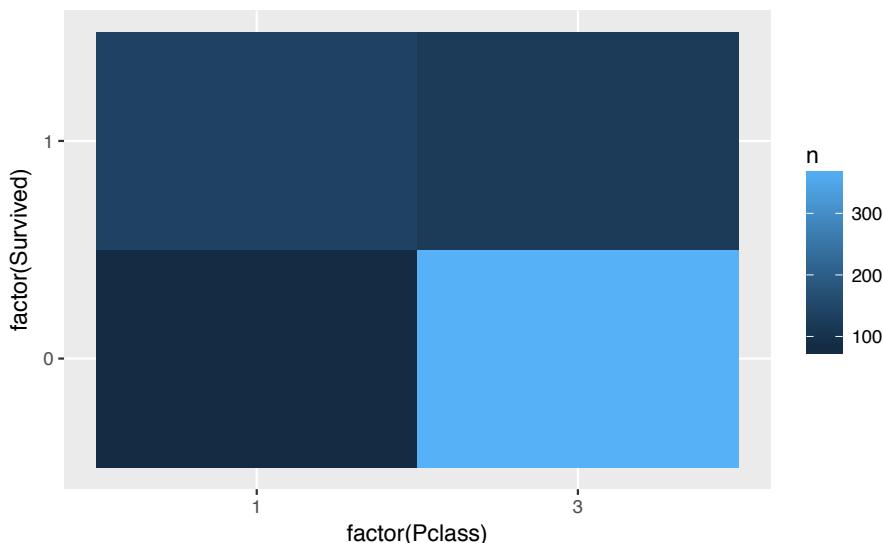


Jeden sehen wir die Häufigkeiten des Überlebens bedingt auf die Passagierklasse besser. Wir sehen auf den ersten Blick, dass sich die Überlebensraten deutlich unterscheiden: Im linken Balken überleben die meisten; im rechten Balken ertrinken die meisten.

Diese letzte Analyse zeigt deutlich 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 ist diese:

```
c5 %>%
  ggplot +
  aes(x = factor(Pclass), y = factor(Survived), fill = n) +
  geom_tile()
```



Hier werden die vier “Fliesen” gleich groß dargestellt; die Fallzahl wird durch die Füllfarbe besorgt.

12.1.10 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.

12.2 Außereheliche Affären

Wovon ist die Häufigkeit von Affären (Seitensprünge) in Ehen abhängig? Diese Frage soll anhand des Datensatzes `Affair` untersucht werden.

Quelle: <http://statsmodels.sourceforge.net/0.5.0/datasets/generated/fair.html>

Der Datensatz findet sich (in ähnlicher Form) auch im R-Paket COUNT (<https://cran.r-project.org/web/packages/COUNT/index.html>).

Laden wir als erstes den Datensatz in R. Wählen Sie zuerst das Verzeichnis als Arbeitsverzeichnis, in dem die Daten liegen. Dann laden Sie z.B. mit dem R-Commander (s. Skript) oder “per Hand” z.B. bei mir so:

```
Affair <- read.csv("data/Affairs.csv")
```

Schauen wir mal, ob es funktioniert hat (“Datenmatrix betrachten”):

```
head(Affair)
#>   X affairs gender age yearsmarried children religiousness education
#> 1 1      0 male  37     10.00      no       3        18
#> 2 2      0 female 27      4.00      no       4        14
#> 3 3      0 female 32     15.00     yes       1        12
#> 4 4      0 male  57     15.00     yes       5        18
#> 5 5      0 male  22      0.75      no       2        17
#> 6 6      0 female 32      1.50      no       2        17
#>   occupation rating
#> 1          7    4
#> 2          6    4
#> 3          1    4
#> 4          6    5
```

```
#> 5      6      3
#> 6      5      5
```

Ok scheint zu passen. Was jetzt?

12.2.1 Zentrale Statistiken

Geben Sie zentrale deskriptive Statistiken an für Affärenhäufigkeit und Ehezufriedenheit!

```
# nicht robust:
mean(Affair$affairs, na.rm = T)
#> [1] 1.46
sd(Affair$affairs, na.rm = T)
#> [1] 3.3
# robust:
median(Affair$affair, na.rm = T)
#> [1] 0
IQR(Affair$affair, na.rm = T)
#> [1] 0
```

Es scheint, die meisten Leute haben keine Affären:

```
table(Affair$affairs)
#>
#> 0   1   2   3   7  12
#> 451 34  17  19  42  38
```

Man kann sich viele Statistiken mit dem Befehl `describe` aus `psych` ausgeben lassen, das ist etwas praktischer:

```
library(psych)

describe(Affair$affairs)
#> vars n mean sd median trimmed mad min max range skew kurtosis se
#> X1 1 601 1.46 3.3 0 0.55 0 0 12 12 2.34 4.19 0.13
describe(Affair$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`).

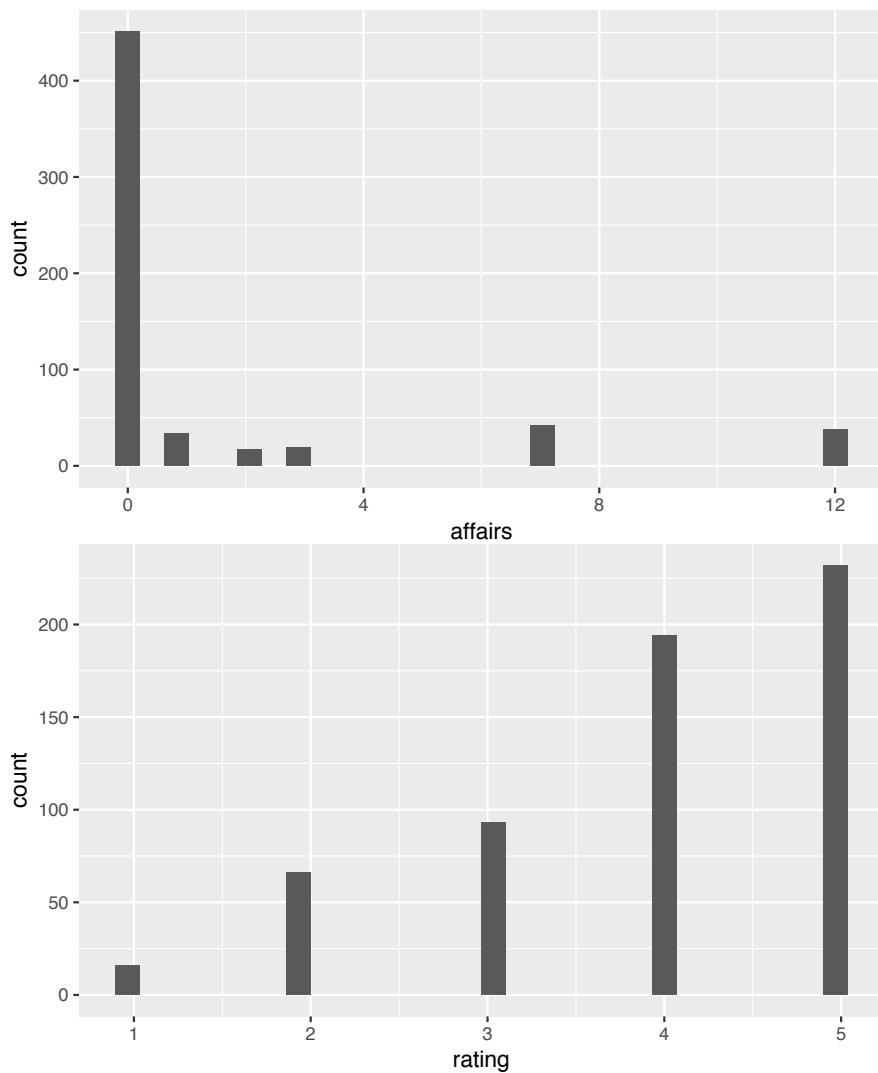
```
install.packages("psych")
```

12.2.2 Visualisieren

Visualisieren Sie zentrale Variablen!

Sicherlich sind Diagramme auch hilfreich. Dies geht wiederum mit dem R-Commander oder z.B. mit folgenden Befehlen:

```
library(ggplot2)
qplot(x = affairs, data = Affair)
qplot(x = rating, data = Affair)
```



Die meisten Menschen (dieser Stichprobe) scheinen mit Ihrer Beziehung sehr zufrieden zu sein.

12.2.3 Wer ist zufriedener mit der Partnerschaft: Personen mit Kindern oder ohne?

Nehmen wir dazu mal ein paar dplyr-Befehle:

```
library(dplyr)
Affair %>%
  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.

12.2.4 Wie viele fehlende Werte gibt es?

Was machen wir am besten damit?

Diesen Befehl könnten wir für jede Spalte aufführen:

```
sum(is.na(Affair$affairs))
#> [1] 0
```

Oder lieber alle auf einmal:

```
Affair %>%
  summarise_each(funs(sum(is.na(.))))
#> # X 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 Sheet² für dplyr.

²<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Ah, gut, keine fehlenden Werte. Das macht uns das Leben leichter.

12.2.5 Wer ist glücklicher: Männer oder Frauen?

```
Affair %>%
  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?

```
Affair %>%
  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...

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. Zumindest fällt mir kein sinnvoller Grund ein; zumal die vorherige eindimensionale Analyse keine Unterschiede zu Tage gefördert hat.

```
Affair %>%
  group_by(gender, children) %>%
  summarise(affairs_mean = mean(affairs),
            rating_mean = mean(rating))
#> Source: local data frame [4 x 4]
#> Groups: gender [?]
#>
#> # A tibble: 4 x 4
#>   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

Affair %>%
  group_by(children, gender) %>%
  summarise(affairs_mean = mean(affairs),
             rating_mean = mean(rating))
#> Source: local data frame [4 x 4]
#> Groups: children [?]

#>
#> # A tibble: 4 x 4
#>   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

```

12.2.6 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:

```

Affair %>%
  group_by(children, gender) %>%
  summarise(rating_mean = mean(rating),
             rating_sd = sd(rating))
#> Source: local data frame [4 x 4]
#> Groups: children [?]

#>
#> # A tibble: 4 x 4
#>   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.

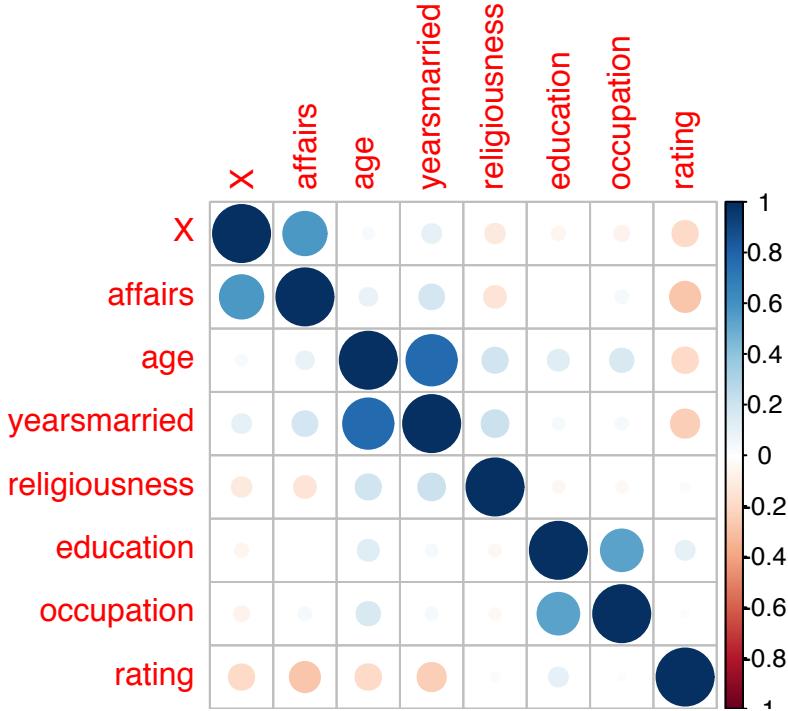
12.2.7 Korrelationen

Berechnen und visualisieren Sie zentrale Korrelationen!

```
Affair %>%
  select_if(is.numeric) %>%
  cor -> cor_tab

cor_tab
#>
#>          X affairs      age yearsmarried religiousness
#> X        1.0000  0.57692  0.0362       0.1078      -0.1164
#> affairs  0.5769  1.00000  0.0952       0.1868      -0.1445
#> age      0.0362  0.09524  1.0000       0.7775      0.1938
#> yearsmarried  0.1078  0.18684  0.7775       1.0000      0.2183
#> religiousness -0.1164 -0.14450  0.1938       0.2183      1.0000
#> education    -0.0537 -0.00244  0.1346       0.0400      -0.0426
#> occupation   -0.0691  0.04961  0.1664       0.0446      -0.0397
#> rating       -0.1951 -0.27951 -0.1990      -0.2431      0.0243
#>
#>          education occupation rating
#> X        -0.05371  -0.0691 -0.1951
#> affairs  -0.00244   0.0496 -0.2795
#> age      0.13460   0.1664 -0.1990
#> yearsmarried  0.04000   0.0446 -0.2431
#> religiousness -0.04257 -0.0397  0.0243
#> education    1.00000   0.5336  0.1093
#> occupation   0.53361   1.0000  0.0174
#> rating       0.10930   0.0174  1.0000

library(corrplot)
corrplot(cor_tab)
```



12.2.8 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 = Affair)
summary(lm1) # Ergebnisse der Regression zeigen
#>
#> Call:
#> lm(formula = affairs ~ yearsmarried, data = Affair)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max 
#> -2.211 -1.658 -0.994 -0.597 11.366 
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)    
#> (Intercept)  0.5512    0.2351   2.34    0.019 *  
#> yearsmarried  0.1106    0.0238   4.65    4e-06 *** 
#> ---
```

```

#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.24 on 599 degrees of freedom
#> Multiple R-squared:  0.0349, Adjusted R-squared:  0.0333
#> F-statistic: 21.7 on 1 and 599 DF, p-value: 4e-06
lm2 <- lm(affairs ~ rating, data = Affair)
summary(lm2)
#>
#> Call:
#> lm(formula = affairs ~ rating, data = Affair)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -3.906 -1.399 -0.563 -0.563 11.437
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)    
#> (Intercept)  4.742     0.479    9.90  <2e-16 ***
#> rating       -0.836     0.117   -7.12  3e-12 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.17 on 599 degrees of freedom
#> Multiple R-squared:  0.0781, Adjusted R-squared:  0.0766
#> F-statistic: 50.8 on 1 and 599 DF, p-value: 3e-12

```

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.

12.2.9 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)?

```

lm3 <- lm(affairs ~ rating + yearsmarried, data = Affair)
lm4 <- lm(affairs ~ yearsmarried + rating, data = Affair)
summary(lm3)
#>
#> Call:
#> lm(formula = affairs ~ rating + yearsmarried, data = Affair)
#>

```

```

#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -4.147 -1.650 -0.837 -0.162 11.894
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  3.7691    0.5671   6.65  6.8e-11 ***
#> rating       -0.7439    0.1200  -6.20  1.1e-09 ***
#> yearsmarried  0.0748    0.0238   3.15   0.0017 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.15 on 598 degrees of freedom
#> Multiple R-squared:  0.0931, Adjusted R-squared:  0.0901
#> F-statistic: 30.7 on 2 and 598 DF,  p-value: 2.01e-13
summary(lm4)
#>
#> Call:
#> lm(formula = affairs ~ yearsmarried + rating, data = Affair)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -4.147 -1.650 -0.837 -0.162 11.894
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  3.7691    0.5671   6.65  6.8e-11 ***
#> yearsmarried  0.0748    0.0238   3.15   0.0017 **
#> rating       -0.7439    0.1200  -6.20  1.1e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.15 on 598 degrees of freedom
#> Multiple R-squared:  0.0931, Adjusted R-squared:  0.0901
#> F-statistic: 30.7 on 2 and 598 DF,  p-value: 2.01e-13

```

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.

12.2.10 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] "X"           "affairs"      "gender"       "age"
#> [5] "yearsmarried" "children"     "religiousness" "education"
#> [9] "occupation"   "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-Comamnder ein Regressionsmodell anfordern oder es mit der Syntax probieren:

```
lm5 <- lm(affairs ~ rating + yearsmarried + children, data = Affair)
summary(lm5)
#>
#> Call:
#> lm(formula = affairs ~ rating + yearsmarried + children, data = Affair)
#>
#> Residuals:
#>    Min      1Q Median      3Q      Max
#> -4.354 -1.732 -0.893 -0.172 12.016
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)    
#> (Intercept) 3.8524    0.5881   6.55  1.2e-10 ***
#> rating      -0.7486    0.1204  -6.22  9.6e-10 ***
#> yearsmarried 0.0833    0.0285   2.92   0.0036 ** 
#> childrenyes -0.1881    0.3482  -0.54   0.5893    
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.15 on 597 degrees of freedom
#> Multiple R-squared:  0.0936, Adjusted R-squared:  0.089
#> F-statistic: 20.5 on 3 and 597 DF,  p-value: 1.11e-12
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 lugert 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 = Affair)
summary(lm6)
#>
#> Call:
#> lm(formula = affairs ~ ., data = Affair)
#>
#> Residuals:
#>    Min      1Q Median      3Q     Max
#> -5.162 -1.644 -0.484  1.016  9.509
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 0.612898  1.008088   0.61  0.54343
#> X            0.010085  0.000634  15.92 < 2e-16 ***
#> gendermale   -0.222695  0.252198  -0.88  0.37759
#> age           -0.029519  0.018987  -1.55  0.12054
#> yearsmarried  0.120077  0.034656   3.46  0.00057 ***
#> childrenyes   -0.357956  0.293529  -1.22  0.22314
#> religiousness -0.272637  0.094432  -2.89  0.00403 **
#> education      0.001544  0.053711   0.03  0.97708
#> occupation     0.182340  0.074579   2.44  0.01478 *
#> rating         -0.456198  0.101757  -4.48  8.8e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.59 on 591 degrees of freedom
#> Multiple R-squared:  0.392, Adjusted R-squared:  0.383
#> F-statistic: 42.4 on 9 and 591 DF, p-value: <2e-16
r2_lm6 <- round(summary(lm6)$r.squared, 2)
```

Der “.” im Befehl `affairs ~ .` oben soll sagen: nimm “alle Variablen, die noch in der Datenmatrix übrig sind”.

Insgesamt bleibt die erklärte Varian in sehr bescheidenem Rahmen: 0.39. Das zeigt uns, dass es immer noch nur schlecht verstanden ist – im Rahmen dieser Analyse – welche Faktoren die Affärenhäufigkeit erklärt.

12.2.11 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:

```
t1 <- t.test(affairs ~ gender, data = Affair)
t1
#>
#> Welch Two Sample t-test
#>
#> data: affairs by gender
#> t = -0.3, df = 600, p-value = 0.8
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -0.607 0.452
#> sample estimates:
#> mean in group female   mean in group male
#>                      1.42                      1.50
```

Der p-Wert ist mit $0.774 > \alpha$. 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.

12.2.12 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:

```
Affair2 <- Affair[Affair$children == "yes", ]
lm7 <- lm(affairs~ rating, data = Affair2)
summary(lm7)

#>
#> Call:
#> lm(formula = affairs ~ rating, data = Affair2)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -4.190 -1.488 -0.587 -0.488 11.413
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  5.091     0.570   8.93 < 2e-16 ***
#> rating      -0.901     0.144  -6.25 9.8e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.34 on 428 degrees of freedom
#> Multiple R-squared:  0.0837, Adjusted R-squared:  0.0815
#> F-statistic: 39.1 on 1 and 428 DF,  p-value: 9.84e-10

Affair3 <- Affair[Affair$children == "no", ]
lm8 <- lm(affairs~ rating, data = Affair3)
summary(lm8)

#>
#> Call:
#> lm(formula = affairs ~ rating, data = Affair3)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -2.55  -1.05  -0.55  -0.55 11.45
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  3.045     0.914   3.33  0.0011 **
```

```
#> rating      -0.499      0.208     -2.40    0.0177 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.68 on 169 degrees of freedom
#> Multiple R-squared:  0.0328, Adjusted R-squared:  0.0271
#> F-statistic: 5.74 on 1 and 169 DF,  p-value: 0.0177
```

Übrigens, einfacher geht das “Subsetten” so:

```
library(dplyr)
Affair4 <- filter(Affair, children == "yes")
head(Affair4)
#>   X affairs gender age yearsmarried children religiousness education
#> 1 3       0 female 32          15 yes           1        12
#> 2 4       0 male   57          15 yes           5        18
#> 3 8       0 male   57          15 yes           2        14
#> 4 9       0 female 32          15 yes           4        16
#> 5 11      0 male   37          15 yes           2        20
#> 6 12      0 male   27             4 yes           4        18
#>   occupation rating
#> 1           1     4
#> 2           6     5
#> 3           4     4
#> 4           1     2
#> 5           7     2
#> 6           6     4
```

12.2.13 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(Affair, affairs != 0)
lm9 <- lm(affairs ~ rating, data = Affair5)
summary(lm9)
#>
#> Call:
```

```
#> lm(formula = affairs ~ rating, data = Affair5)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -6.06 -3.52 -0.06  3.69  7.48
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  8.757     1.023   8.56 1.3e-14 ***
#> rating       -0.848     0.280  -3.03  0.0029 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 4.14 on 148 degrees of freedom
#> Multiple R-squared:  0.0584, Adjusted R-squared:  0.052
#> F-statistic: 9.18 on 1 and 148 DF, p-value: 0.00289
```

12.2.14 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!

```
Affair %>%
  mutate(affairs_dichotom = if_else(affairs == 0, 0, 1)) %>%
  glm(affairs_dichotom ~ gender + children + rating + yearsmarried,
      data = .,
      family = "binomial") -> lm10

summary(lm10)
#>
#> Call:
#> glm(formula = affairs_dichotom ~ gender + children + rating +
#>       yearsmarried, family = "binomial", data = .)
#>
#> Deviance Residuals:
#>    Min     1Q Median     3Q    Max
#> -1.420 -0.764 -0.617 -0.443  2.171
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.0537    0.4299   0.12    0.90
```

```
#> gendermale    0.2416    0.1966    1.23    0.22
#> childrenyes   0.3935    0.2831    1.39    0.16
#> rating        -0.4654    0.0874   -5.33   1e-07 ***
#> yearsmarried   0.0221    0.0212    1.04    0.30
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 675.38 on 600 degrees of freedom
#> Residual deviance: 630.26 on 596 degrees of freedom
#> AIC: 640.3
#>
#> Number of Fisher Scoring iterations: 4
```

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 Funktion `lm` und `glm` spucken leider keine brave Tabelle in Normalform aus. Aber man leicht eine schöne Tabelle (data.frame) bekommen mit dem Befehl `tidy` aus `broom`:

```
library(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
```

Und Tabellen (d.h. brave Dataframes) kann man sich schön ausgeben lassen z.B. mit dem Befehl `knitr::kable`:

```
library(knitr)
tidy(lm10) %>% kable
```

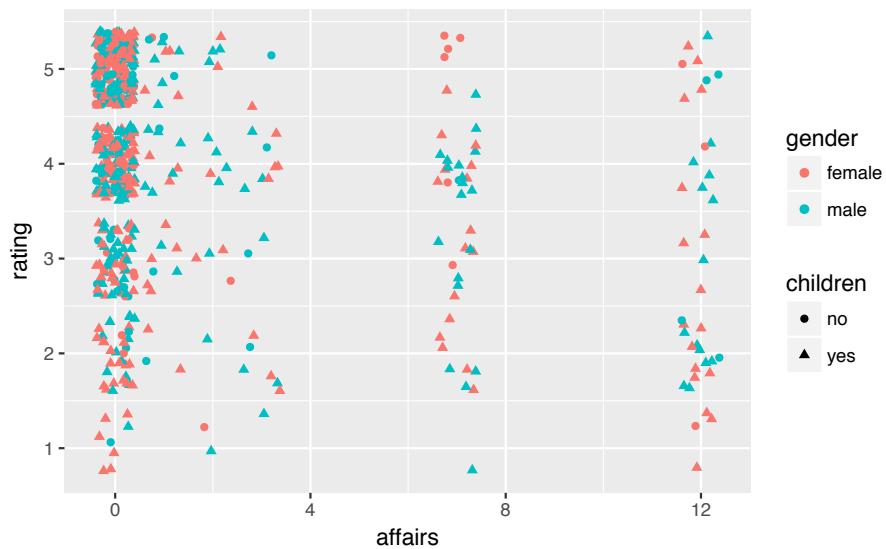
term	estimate	std.error	statistic	p.value
(Intercept)	0.054	0.430	0.125	0.901
gendermale	0.242	0.197	1.229	0.219
childrenyes	0.394	0.283	1.390	0.164
rating	-0.465	0.087	-5.327	0.000
yearsmarried	0.022	0.021	1.040	0.299

12.2.15 Zum Abschluss

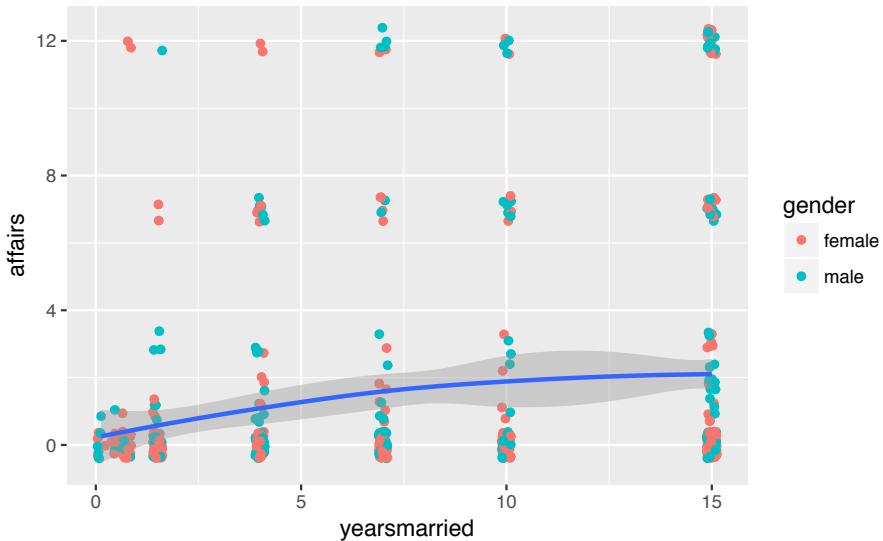
Visualisieren wir mal was!

Ok, wie wäre es damit:

```
Affair %>%
  select(affairs, gender, children, rating) %>%
  ggplot(aes(x = affairs, y = rating)) +
  geom_jitter(aes(color = gender, shape = children))
```



```
Affair %>%
  mutate(rating_dichotom = ntile(rating, 2)) %>%
  ggplot(aes(x = yearsmarried, y = affairs)) +
  geom_jitter(aes(color = gender)) +
  geom_smooth()
```



Puh. Geschafft!

12.3 Befehlsübersicht

Tabelle ?? fasst die R-Funktionen dieses Kapitels zusammen.

Tabelle 12.1: Befehle des Kapitels ‘Fallstudien’

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
glm	Rechnet eine generalisiertes lineares Modell (logistische Regression)
exp	Delogarithmiert einen Ausdruck
coef	Liefert die Koeffizienten von einem Objekt des Typs lm oder glm zurück.
predict	Macht eine Vorhersage für ein Objekt des Typs lm oder glm
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 Wertebereich in n gleich große Teile auf (d.h. mit jeweils gleich vielen Fällen)

Paket..Funktion	Beschreibung
broom::tidy	Wandelt ein Modellobjekt (z.B. von "lm") in einen Dataframe um.

Teil IV

Ungeleitetes Modellieren

Kapitel 13

Vertiefung: Clusteranalyse



Benötigte Pakete:

```
library(tidyverse)
library(cluster)
```



Lernziele:

- Das Ziel einer Clusteranalyse erläutern können.
- Das Konzept der euklidischen Abstände verstehen.
- Eine k-Means-Clusteranalyse berechnen und interpretieren können.

13.1 Einführung

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.

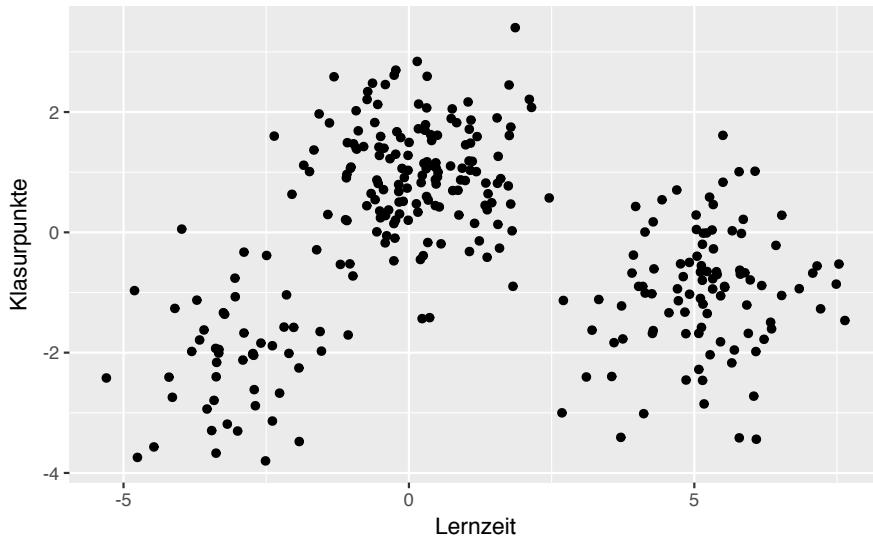


Abbildung 13.1: Ein Streudiagramm - sehen Sie Gruppen (Cluster) ?

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 euklischen Abstand konzentrieren.

13.2 Intuitive Darstellung der Clusteranalyse

Betrachten Sie das folgende Streudiagramm (die Daten sind frei erfunden; "simuliert", sagt der Statistiker). Es stellt den Zusammenhang von Lernzeit (wieviel 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. 13.1)?

Färben wir das Diagramm mal ein (Abb. 13.2).

Nach dieser "Färbung", d.h. nach dieser Aufteilung in drei Gruppen, scheint es folgende "Cluster", "Gruppen" oder "Typen" von Studierenden zu geben:

- "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.

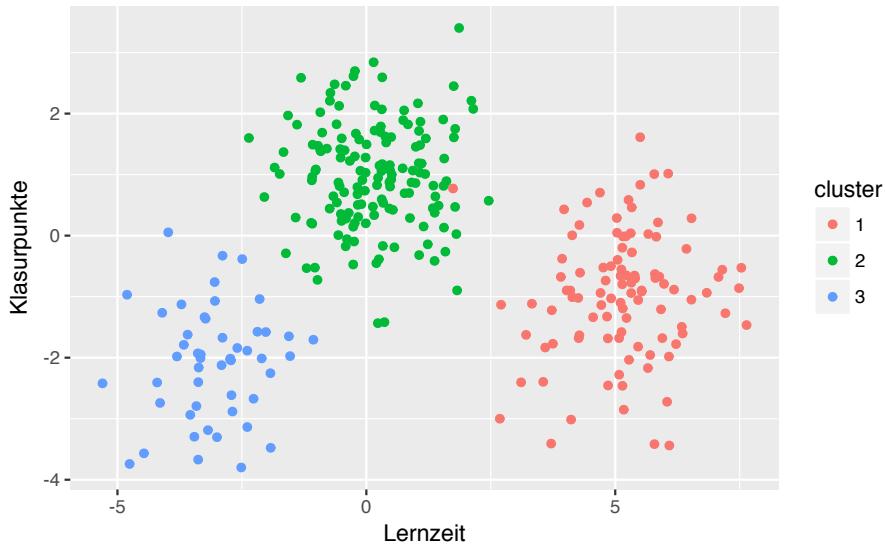


Abbildung 13.2: Ein Streudiagramm - mit drei Clustern

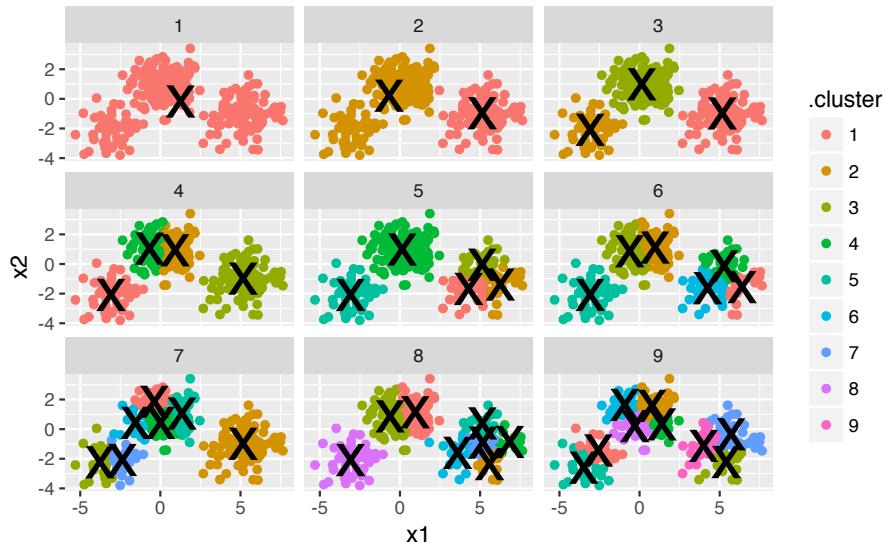


Abbildung 13.3: Unterschiedliche Anzahlen von Clustern im Vergleich

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 13.3).

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"

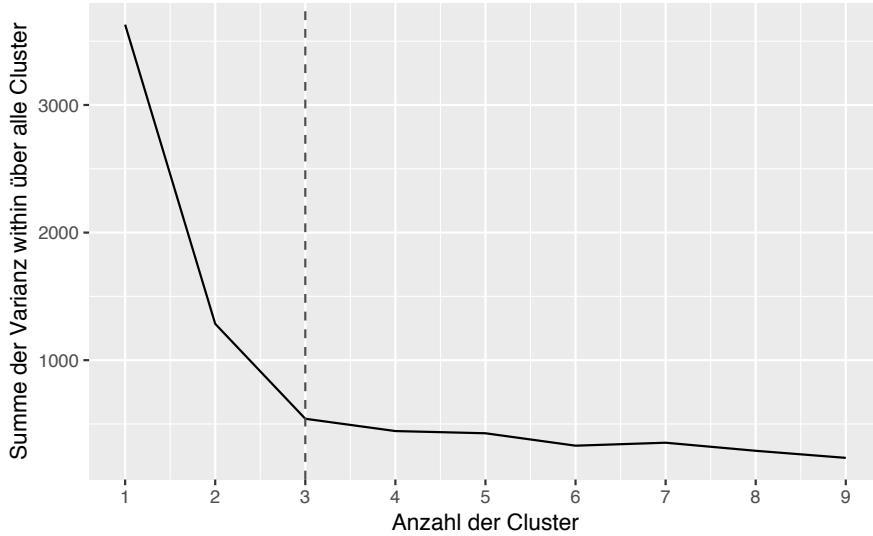


Abbildung 13.4: Die Summe der Varianz within in Abhangigkeit von der Anzahl von Clustern. Ein Screeplot.

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 Losung seien, weil mehr Cluster die Varianz innerhalb der Cluster nur noch wenig verringern. Diese Art von Diagramm wird als "Screeplot" bezeichnet. Fertig!

13.3 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 (bedauernswert). Die beiden unterscheiden sich sowohl in Lernzeit als auch in Klausurerfolg. Aber wie sehr unterscheiden sie sich? Wie gro ist der "Abstand" zwischen Anna und Berta (vgl. Abb. 13.5)?

Eine Moglichkeit, die Distanz zwischen zwei Punkten in der Ebene (2D) zu bestimmen, ist der *Satz des Pythagoras* (leise Trompetenfanfare). Generationen von Schulern haben diese Gleichung ´ahmm... geliebt:

$$c^2 = a^2 + b^2$$

In unserem Beispiel heit 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 betragt 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 Raumen mehr mehr Dimensionen??? Betrachten wir den Versuch, zwei Dreiecke in 3D zu zeichnen. Stellen wir uns vor, zusatzlich zu Lernzeit und Klausurerfolg hatten wir als 3.

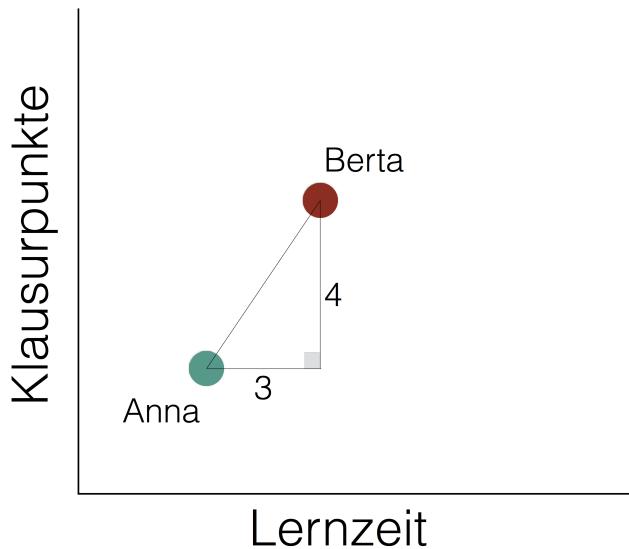


Abbildung 13.5: Distanz zwischen zwei Punkten in der Ebene

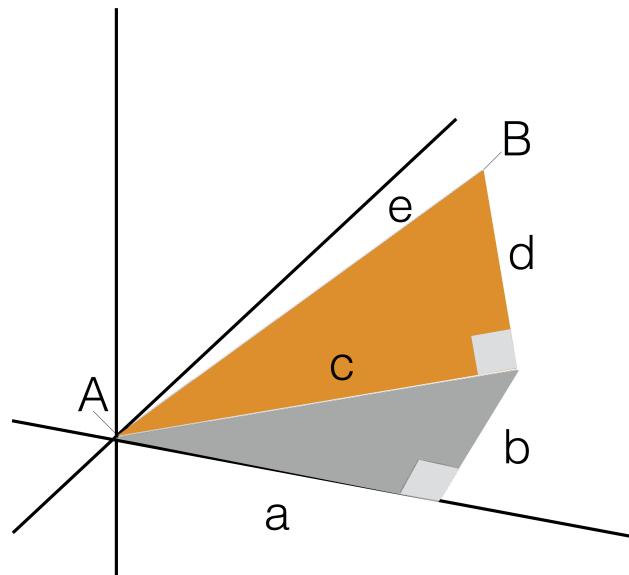


Abbildung 13.6: Pythagoras in 3D

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 orange farbenen 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

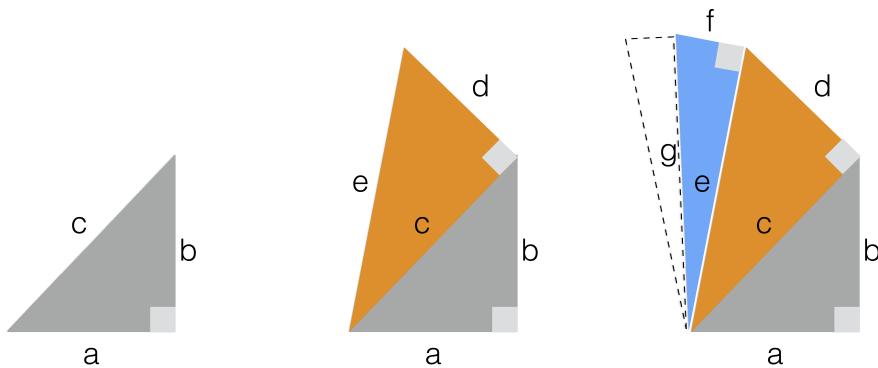


Abbildung 13.7: Pythagoras in Reihe geschaltet

$$e^2 = c^2 + d^2$$

$$e^2 = 5^2 + 2^2$$

$$e^2 = 25 + 4$$

$$e = \sqrt{29} \approx 5.4$$

Ah! Der Unterschied zwischen den beiden Studentinnen beträgt also $\sim 5.4!$

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”.

```
profs <- 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 eingen 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...

13.4 Daten

Schauen wir uns eine Clusteranalyse praktisch an. Wir werden einen *simulierten* Datensatz aus Chapman und Feit (2015) analysieren. Näheres zu den Daten siehe Kapitel 5 dort.

Sie können ihn von <https://goo.gl/eUm8PI> als csv-Datei herunterladen; oder, wenn sich die Datei im Unterordner data/ (relativ zu ihrem Arbeitsverzeichnis) befindet:

```
segment <- read.csv2("data/segment.csv")
```

Wir verwenden die Variante `read.csv2`, da es sich um eine “deutsche” CSV-Datei handelt.

Ein Überblick über die Daten verschafft uns die Funktion `glimpse`.

```
glimpse(segment)
#> Observations: 300
#> Variables: 7
#> $ Alter           <dbl> 50.2, 40.7, 43.0, 40.3, 41.1, 40.2, 39.5, 35.7, ...
#> $ Geschlecht       <fctr> Mann, Mann, Frau, Mann, Frau, Mann, Frau, Mann...
#> $ Einkommen        <dbl> 51356, 64411, 71615, 42728, 71641, 60325, 54746...
#> $ Kinder           <int> 0, 3, 2, 1, 4, 2, 5, 1, 1, 0, 3, 4, 0, 2, 6, 0, ...
#> $ Eigenheim        <fctr> Nein, Nein, Ja, Nein, Nein, Ja, Nein, Nein, Ne...
#> $ Mitgliedschaft   <fctr> Nein, Nein, Nein, Nein, Nein, Nein, Ja, Ja, Ne...
#> $ Segment          <fctr> Gemischte Vorstadt, Gemischte Vorstadt, Gemisc...
```

13.5 Distanzmaße mit R berechnen

Auf Basis der drei metrischen Merkmale (d. h. `Alter`, `Einkommen` und `Kinder`) ergeben sich für die ersten sechs Beobachtungen folgende Abstände:

```
dist(head(segment))
#>      1     2     3     4     5
#> 2 19941.8
#> 3 30946.1 11004.3
#> 4 13179.5 33121.3 44125.6
#> 5 30985.9 11044.0   39.9 44165.3
#> 6 13700.4  6241.5 17245.8 26879.9 17285.5
```

Sie können erkennen, dass die Beobachtungen 5 und 3 den kleinsten Abstand haben, während 5 und 4 den größten haben. Allerdings zeigen die Rohdaten auch, dass die euklidischen Abstände von der Skalierung der Variablen abhängen (`Einkommen` streut stärker als `Kinder`). Daher kann es evt. sinnvoll sein, 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

```
daisy(head(segment))
```

13.6 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.

Hinweis: Die (robuste) Funktion `pam()` aus dem Paket `cluster` kann auch mit allgemeinen Distanzen umgehen. Außerdem für gemischte Variablentypen gut geeignet: Das Paket `clustMixType`¹.

Zur Vorbereitung überführen wir die nominalen Merkmale in logische, d. h. binäre Merkmale und löschen die Segmente sowie das Ergebnis der hierarchischen Clusteranalyse:

```
segment.num <- segment %>%
  mutate(Frau = Geschlecht == "Frau") %>%
  mutate(Eigenheim = Eigenheim == "Ja") %>%
  mutate(Mitgliedschaft = Mitgliedschaft == "Ja") %>%
  dplyr::select(-Geschlecht, -Segment)
```

Über die Funktion `mutate()` werden Variablen im Datensatz erzeugt oder verändert. Über `select()` werden einzene Variablen ausgewählt. Mit der “Pfeife” `%>%` übergeben wir das Ergebnis der vorherigen Funktion an die folgende.

Aufgrund von (1.) hängt das Ergebnis einer k-Means Clusteranalyse vom Zufall ab. Aus Gründen der Reproduzierbarkeit sollte daher der Zufallszahlengenerator gesetzt werden. Außerdem bietet es sich an verschiedene Startkonfigurationen zu versuchen. In der Funktion `kmeans()` erfolgt dies durch die Option `nstart =`. Hier mit `k = 4` Clustern:

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

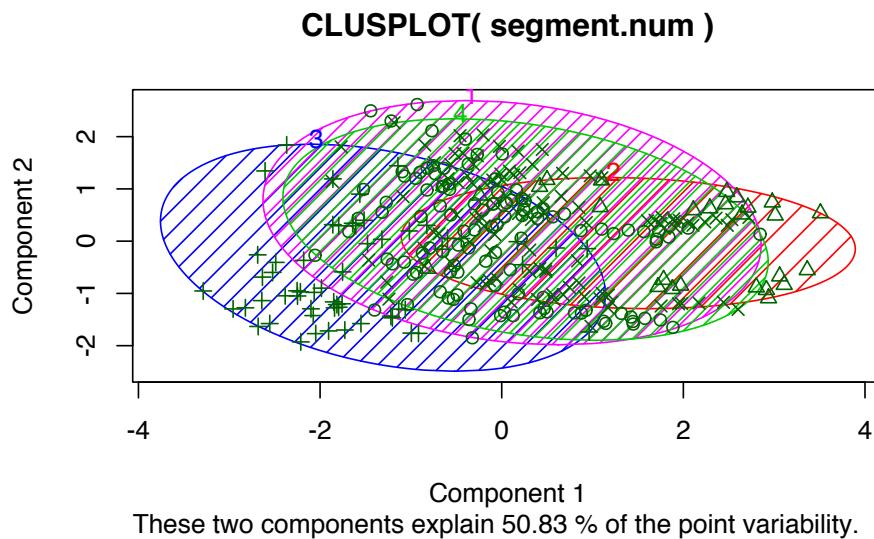
```
#> [141] 3 3 3 3 3 3 3 3 3 1 2 4 2 2 4 1 1 2 2 4 4 1 1 4 2 4 4 1 2 2 3 4 1 2
#> [176] 2 4 2 3 4 4 4 1 1 1 1 1 4 3 1 4 4 4 4 1 1 1 2 4 4 1 2 4 4 1 4 2 1 2
#> [211] 4 3 4 2 2 4 2 1 4 3 1 2 2 4 2 4 4 1 4 4 1 1 1 1 1 3 1 1 4 1 4 3 1 4 1
#> [246] 4 1 4 1 4 4 4 4 1 1 1 4 4 1 1 1 1 1 4 1 1 1 1 2 4 4 1 4 1 1 1 2
#> [281] 4 4 4 4 1 4 1 4 4 4 1 4 1 4 1 1 4 1
#>
#> Within cluster sum of squares by cluster:
#> [1] 3.18e+09 2.22e+09 1.69e+09 2.81e+09
#> (between_SS / total_SS = 90.6 %)
#>
#> Available components:
#>
#> [1] "cluster"      "centers"       "totss"        "withinss"
#> [5] "tot.withinss" "betweenss"    "size"         "iter"
#> [9] "ifault"
```

Neben der Anzahl Beobachtungen im Cluster (z. B. 26 in Cluster 2) werden auch die Clusterzentren ausgegeben. Diese können dann direkt verglichen werden. Sie sehen z. B., dass das Durchschnittsalter in Cluster 3 mit 27 am geringsten ist. Der Anteil der Eigenheimbesitzer ist mit 54 % in Cluster 2 am höchsten.

In zwei Dimensionen kann man Cluster gut visualisieren (Abbildung 13.3).; 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.

Einen Plot der Scores auf den beiden ersten Hauptkomponenten können Sie über die Funktion `clusplot()` aus dem Paket `cluster` erhalten.

```
clusplot(segment.num, seg.k$cluster,
         color = TRUE, shade = TRUE, labels = 4)
```



Wie schon im deskriptiven Ergebnis: Die Cluster 1 und 4 unterscheiden sich (in den ersten beiden Hauptkomponenten) nicht wirklich. Vielleicht sollten dies noch zusammengefasst werden, d. h., mit `centers=3` die Analyse wiederholt werden?²

13.7 Aufgaben

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, 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 oder ähnlichen Datensatzes!

13.8 Befehlsübersicht

Tabelle 13.1 fasst die R-Funktionen dieses Kapitels zusammen.

²Das Paket `NbClust`, siehe Malika Charrad, Nadia Ghazzali, Veronique Boiteau, Azam Niknafs (2014) *NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set*, Journal of Statistical Software, 61(6), 1-36. <http://dx.doi.org/10.18637/jss.v061.i06>, bietet viele Möglichkeiten die Anzahl der Cluster optimal zu bestimmen.

Tabelle 13.1: 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

13.9 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 Unported³. Der Code steht unter der Apache Lizenz 2.0⁴.
- 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/>.

³<http://creativecommons.org/licenses/by-sa/3.0>

⁴<http://www.apache.org/licenses/LICENSE-2.0>

Kapitel 14

Vertiefung: Dimensionsreduktion



Lernziele:

- Den Unterschied zwischen einer Hauptkomponentenanalyse und einer Exploratorische Faktorenanalyse kennen
- Methoden kennen, um die Anzahl von Dimensionen zu bestimmen
- Methoden der Visualisierung anwenden können
- Umsetzungsmethoden in R anwenden können
- Ergebnisse interpretieren können.

In diesem Kapitel werden folgende Pakete benötigt:

```
library(corrplot) # für `corrplot`  
library(gplots) # für `heatmap.2`  
library(nFactors) # PCA und EFA  
library(tidyverse) # Datenjudo  
library(psych) # für z.B. 'alpha'
```

14.1 Einführung

Datensätze in den Sozialwissenschaften, und damit auch in der Wirtschaftspsychologie, haben oft viele Variablen - oder auch Dimensionen - und es ist vorteilhaft, diese auf eine kleinere Anzahl von Variablen (oder Dimensionen) zu reduzieren. Zusammenhänge zwischen verschiedenen Dimensionen oder Unterschiede zwischen verschiedenen Gruppen bezüglich einer oder mehrerer Dimensionen (z. B. bei Experimenten) können so klarer und einfacher identifiziert werden. Dimensionen mit konkreten Sachverhalten werden in der Sprache der Wissenschaft häufig als *Konstrukte* bezeichnet.

Konstrukte stellen in den Sozialwissenschaften gedankliche bzw. theoretische Sachverhalt dar, die nicht direkt beobachtbar und damit nicht direkt messbar sind. Nehmen wir beispielsweise an, es soll das Konstrukt *Anerkennung* im Rahmen einer sozialwissenschaftlichen Studie gemessen werden. Dabei gibt es zunächst zwei Fragestellungen:

1. Was bedeutet Anerkennung?
2. Wie wird Anerkennung gemessen?

Liest man bei Wikipedia diesen Begriff nach, kommt folgende Antwort: "Anerkennung bedeutet die Erlaubnis einer Person oder einer Gruppe gegenüber einer anderen Person, Gruppe oder Institution, sich mit ihren derzeitigen spezifischen Eigenschaften an der Kommunikation, an Entscheidungsprozessen oder anderen gesellschaftlichen Prozessen zu beteiligen. Der Begriff Anerkennung wird auch als Synonym für Akzeptanz, Lob oder Respekt verwendet." Gut, wir kennen nun die Bedeutung von Anerkennung, aber wir wissen immer noch nicht, wie wir Anerkennung messen können. Da die Suche nach Anerkennung in der Psychologie kein neues **Konstrukt** darstellt, sondern schon vielfach gemessen wurde, müssen wir nur in bisherigen Forschungsergebnissen nachlesen. Dies führt unweigerlich dazu, dass wir auf bisherige Forschungen stoßen, die das Konstrukt Anerkennung als ein **multidimensionales Konstrukt** definieren und mit mehr als einem *Item (Indikator)* messen. Mehr zur Messung von Anerkennung weiter unten im Datenbeispiel. D. h. der Sachverhalt **Anerkennung** wird aus anderen, messbaren Sachverhalten (**Indikatoren**) messbar gemacht. Der Prozess des "*Messbar machen*" heißt *Operationalisierung*. Mehr zur Operationalisierung von Anerkennung und anderen Konstrukten betrachten wir weiter untern im Datenbeispiel.

In diesem Kapitel betrachten wir zwei gängige Methoden, um die Komplexität von multivarianten, metrischen Daten zu reduzieren, indem wir die Anzahl der Dimensionen in den Daten reduzieren.

- Die *Hauptkomponentenanalyse* (engl. principal component analysis, PCA) versucht, unkorrelierte Linearkombinationen zu finden, die die maximale Varianz in den Daten erfassen. Die PCA beinhaltet also das Extrahieren von linearen Zusammenhängen der beobachteten Variablen.
- Die *Exploratorische Faktorenanalyse (EFA)* versucht, die Varianz auf Basis einer kleinen Anzahl von Dimensionen zu modellieren, während sie gleichzeitig versucht, die Dimensionen in Bezug auf die ursprünglichen Variablen interpretierbar zu machen. Es wird davon ausgegangen, dass die Daten einem Faktoren Modell entsprechen, bei der die beobachteten Korrelationen auf **latente** Faktoren zurückführen. Mit der EFA wird

nicht die gesamte Varianz erklärt.

In der Psychologie werden diese beiden Methoden oft in der Konstruktion von mehrstufigen Tests angewendet, um festzustellen, welche Items auf welche Konstrukte laden. Sie ergeben in der Regel ähnliche inhaltliche Schlussfolgerungen. Dies erklärt, warum einige Statistik-Software-Programme beide Methoden zusammenpacken. So wird die PCA als Standard-Extraktionsmethode in den SPSS-Faktorenanalyse-Routinen verwendet. Dies führt zweifellos zu einer gewissen Verwirrung über die Unterscheidung zwischen den beiden Methoden. Die EFA wird oft als *Common Factor Analysis* oder *principal axis analysis (Hauptachsenanalyse)* bezeichnet. Die EFA verwendet eine Vielzahl von Optimierungsroutinen und das Ergebnis, im Gegensatz zu PCA, hängt von der verwendeten Optimierungsroutine und Ausgangspunkten für diese Routinen ab. Es gibt also *keine einzigartige* Lösung bei der EFA.

Eine einfache Faustregel für die Entscheidung zwischen diesen beiden Methoden:

- Führe die PCA durch, wenn die korrelierten beobachteten Variablen einfach auf einen kleineren Satz von wichtigen unabhängigen zusammengesetzten Variablen reduziert werden soll.
- Führe die EFA durch, wenn ein theoretisches Modell von latenten Faktoren zugrunde liegt, dass die beobachtete Variablen verursacht.

14.2 Gründe für die Notwendigkeit der Datenreduktion

- *Dimensionen reduzieren*: Im technischen Sinne der Dimensionsreduktion können wir statt Variablen-Sets die Faktor-/ Komponentenwerte verwenden (z. B. für Mittelwertvergleiche zwischen Experimental- und Kontrollgruppe, Regressionsanalyse und Clusteranalyse).
- *Unsicherheit verringern*: Wenn wir glauben, dass ein Konstrukt nicht eindeutig messbar ist, dann kann mit einem Variablen-Set die Unsicherheit reduziert werden.
- *Aufwand verringern*: Wir können den Aufwand bei der Datenerfassung vereinfachen, indem wir uns auf Variablen konzentrieren, von denen bekannt ist, dass sie einen hohen Beitrag zum interessierenden Faktor/ Komponente leisten. Wenn wir feststellen, dass einige Variablen für einen Faktor nicht wichtig sind, können wir sie aus dem Datensatz eliminieren. Außerdem werden die statistischen Modelle einfacher, wenn wir statt vieler Ausgangsvariablen einige wenige Komponenten/Faktoren als Eingabevariablen verwenden.

14.3 Intuition zur Dimensionsreduktion

Betrachten Sie die die Visualisierung eines Datensatzes mit 3 Dimensionen (Spalten) in Abbildung 14.1). Man braucht nicht viel Phantasie, um einen Pfeil (Vektor) in der Punktwolke

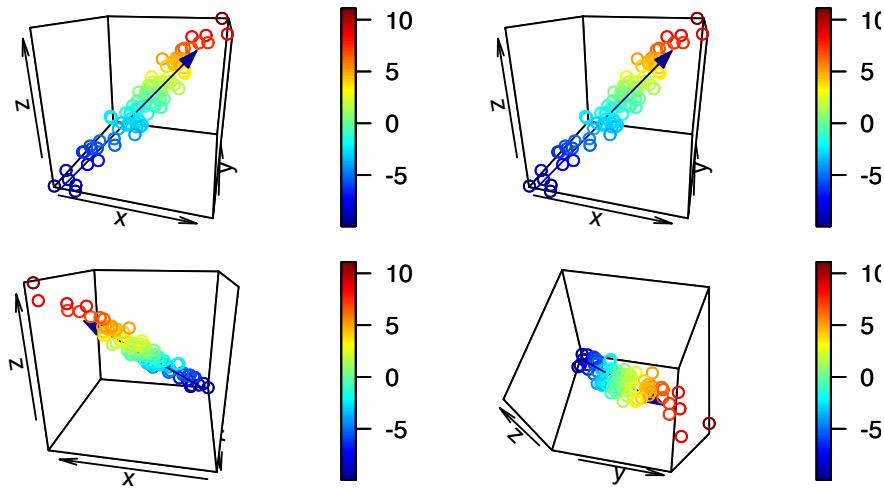


Abbildung 14.1: Der Pfeil ist eindimensional; reduziert also die drei Dimensionen auf eine

zu sehen. Um jeden Punkt einigermaßen genau zu bestimmen, reicht es, seine “Pfeil-Koordinate” zu wissen. Praktischerweise geben in Abbildung 14.1 die Farben (in etwa) die Koordinaten auf dem Pfeil an¹. Damit können wir die Anzahl der Variablen (Dimensionen), die es braucht, um einen Punkt zu beschreiben von 3 auf 1 reduzieren; 2/3 der Komplexität eingespart. Wir verlieren etwas Genauigkeit, aber nicht viel. Dieser Pfeil, der mitten durch den Punkteschwarm geht, nennt man auch die 1. Hauptkomponente.

Beachten Sie, dass hoch korrelierte Variablen eng an der Regressionsgeraden liegen; entsprechend sind in Abbildung ?? die drei Variablen stark korreliert. Sehen Sie auch, dass die Hauptkomponente Varianz erklärt: Jede Variable für sich genommen, hat recht viel Streuung. Die Streuung der Punkte zur Hauptkomponente ist aber relativ gering. Daher sagt man, die Streuung (Varianz) wurde reduziert durch die Hauptkomponente.

Der längste Vektor, den man in die Punktewolke legen kann, bezeichnet man als den 1. Eigenvektor oder die 1. Hauptkomponente.

In Abbildung 14.1 ist dieser als Pfeil eingezeichnet². Weitere Hauptkomponenten kann man nach dem gleichen Muster bestimmen mit der Auflage, dass sie im rechten Winkel zu bestehenden Hauptkomponenten liegen. Damit kann man in einer 3D-Raum nicht mehr als 3 Hauptkomponenten bestehen (in einem n -dimensionalen Raum also maximal n Hauptkomponenten).

```
#>      V1      V2      V3
#> V1  1.000  0.933  0.955
#> V2  0.933  1.000  0.833
#> V3  0.955  0.833  1.000
```

Je stärker die Korrelation zwischen Variablen, desto besser kann man sie zusam-

¹genau genommen ist hier die Regressionsgerade gezeichnet, es müsste aber der größte Eigenvektor sein. Geschenkt.

²die Hauptkomponente ist hier ähnlich zur Regressionslinie, aber nicht identisch

menfassen.

14.4 Daten

Wir untersuchen die Dimensionalität mittels einer auf 1000 Fälle reduzierten Zufallsauswahl von 15 Variablen zur Messung der grundlegenden Wertorientierungen von Menschen. Die Daten wurden im Sommersemester 2017 von FOM Studierenden im ersten Semester an der FOM bundesweit erhoben. Die Variablen zu Wertorientierungen wurden ursprüngliche aus dem 40-Item-Set des Portraits Value Questionnaire» (PVQ) von Schmidt u. a. (2007) adaptiert und durch Studien an der FOM seit 2014 stufenweise bis auf 15 relevante Variablen reduziert. Alle Variablen wurden auf einer Skala von 1 bis 7 (wobei 1 am wenigsten und 7 am meisten zutrifft) abgefragt.

Das Einlesen der Daten erfolgt mit dem Befehl `read.csv2`.

```
Werte <- read.csv2("data/Werte.csv")
```

Wir überprüfen zuerst die Struktur des Datensatzes, die ersten 6 Zeilen und die Zusammenfassung. Probieren Sie die folgenden Befehle aus:

```
glimpse(Werte)
```

Wir sehen mit `glimpse`, dass die Bereiche der Bewertungen für jede Variable 1-7 sind. Außerdem sehen wir, dass die Bewertungen als numerisch (Integer, also ganzzahlig) eingelesen wurden. Die Daten sind somit offenbar richtig formatiert.

14.5 Neuskalierung der Daten

In vielen Fällen ist es sinnvoll, Rohdaten neu zu skalieren - auch bei der Dimensionsreduktion. Warum ist das nötig?

Dies wird üblicherweise als *Standardisierung*, *Normierung*, oder *Z-Transformation* bezeichnet. Als Ergebnis ist der Mittelwert aller Variablen über alle Beobachtungen dann 0 und die Standardabweichung (SD) 1. Da wir hier gleiche Skalenstufen haben, ist ein Skalieren nicht unbedingt notwendig, wir führen es aber trotzdem durch.

Ein einfacher Weg, alle Variablen im Datensatz auf einmal zu skalieren ist der Befehl `scale()`. Da wir die Rohdaten nie ändern wollen, weisen wir die Rohwerte zuerst einem neuen Dataframe `Werte.sc` zu und skalieren anschließend die Daten. Wir skalieren in unserem Datensatz alle Variablen.

```
Werte.sc <- Werte %>% scale %>% as_tibble
summary(Werte.sc)
```



<-: Das Objekt `Werte.sc` soll wie folgt definiert sein
 Nimm das Objekt (ein Dataframe) `Werte` UND DANN z-skaliere das Objekt UND
 DANN
 definiere es als Dataframe (genauer: tibble). FERTIG.

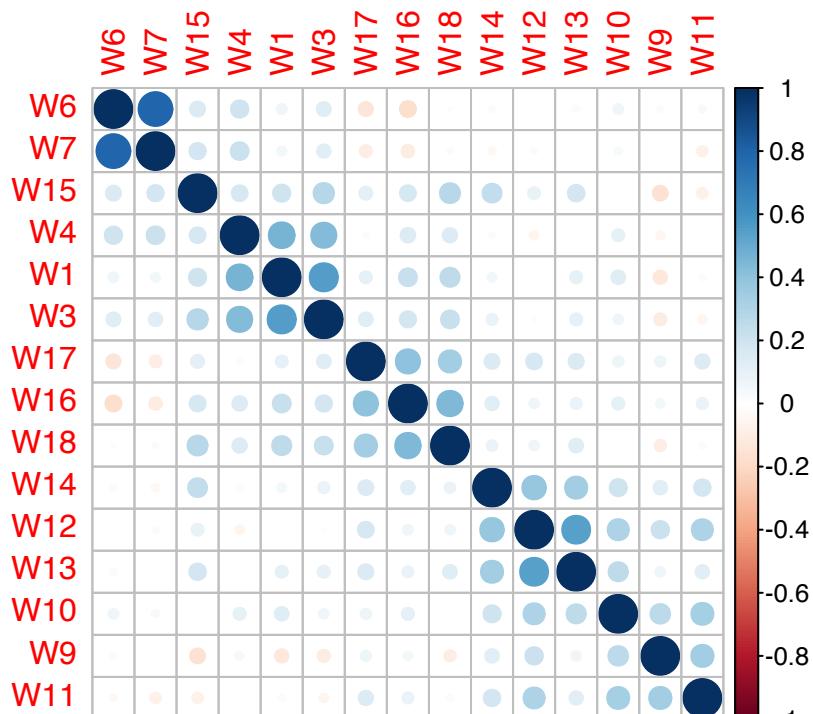
Ach ja, dann zeig noch ein `summary` von diesem Objekt.

Die Daten wurden richtig skaliert, da der Mittelwert aller Variablen über alle Beobachtungen 0 und die sd 1 ist.

14.6 Zusammenhänge in den Daten

Wir verwenden den Befehl `corrplot()` für die Erstinspektion von bivariaten Beziehungen zwischen den Variablen. Das Argument `order = "hclust"` ordnet die Zeilen und Spalten entsprechend der Ähnlichkeit der Variablen in einer hierarchischen Cluster-Lösung der Variablen (mehr dazu im Kapitel 13) neu an.

```
corrplot(cor(Werte.sc), order = "hclust")
```



Die Visualisierung der Korrelation der Variablen scheint fünf Cluster zu zeigen:

- (“W1”, “W3”, “W4”)
- (“W12”, “W13”, “W14”, “W15”)
- (“W16”, “W17”, “W18”)
- (“W6”, “W7”)
- (“W9”, “W10”, “W11”)

14.7 Daten mit fehlende Werten

Wenn in den Daten leere Zellen, also fehlende Werte, vorhanden sind, dann kann es bei bestimmten Rechenoperationen zu Fehlermeldungen kommen. Dies betrifft zum Beispiel Korrelationen, PCA und EFA. Der Ansatz besteht deshalb darin, NA-Werte explizit zu entfernen. Dies kann am einfachsten mit dem Befehl `na.omit()` geschehen:

Beispiel:

```
Werte.sc <- na.omit(Werte.sc)

corrplot(cor(Werte.sc), order = "hclust")
```

Da wir in unserem Datensatz vollständige Daten verwenden, gibt es auch keine Leerzellen.

Mit dem Parameter `order` kann man die Reihenfolge (order) der Variablen, wie sie im Diagramm dargestellt werden ändern (vgl `help(corrplot)`). Hier haben wir die Variablen nach Ähnlichkeit aufgereiht: Ähnliche Variablen stehen näher beieinander. Damit können wir gut erkennen, welche Variablen sich ähnlich sind (hoch korreliert sind) und somit Kandidaten für eine Einsparung (Zusammenfassung zu einer Hauptkomponente bzw. einem Faktor) sind.

14.8 Hauptkomponentenanalyse (PCA)

Die PCA berechnet ein Variablenset (Komponenten) in Form von linearen Gleichungen, die die linearen Beziehungen in den Daten erfassen. Die erste Komponente erfasst so viel Streuung (Varianz) wie möglich von allen Variablen als eine einzige lineare Funktion. Die zweite Komponente erfasst unkorreliert zur ersten Komponente so viel Streuung wie möglich, die nach der ersten Komponente verbleibt. Das geht so lange weiter, bis es so viele Komponenten gibt wie Variablen.

14.8.1 Bestimmung der Anzahl der Hauptkomponenten

Betrachten wir in einem ersten Schritt die wichtigsten Komponenten für die Werte. Wir finden die Komponenten mit `prcomp()`.

```
Werte.pc <- prcomp(Werte.sc) # Principal Components berechnen
summary(Werte.pc)
#> Importance of components%>%
#>                 PC1    PC2    PC3    PC4    PC5    PC6    PC7
#> Standard deviation     1.691  1.542  1.384  1.1428 1.0797 0.8855 0.8298
#> Proportion of Variance 0.191  0.159  0.128  0.0871 0.0777 0.0523 0.0459
#> Cumulative Proportion   0.191  0.349  0.477  0.5639 0.6416 0.6939 0.7398
#>                         PC8    PC9    PC10   PC11   PC12   PC13   PC14
#> Standard deviation      0.8078 0.7882 0.7599 0.7413 0.6884 0.648  0.6449
#> Proportion of Variance 0.0435 0.0414 0.0385 0.0366 0.0316 0.028  0.0277
#> Cumulative Proportion   0.7833 0.8247 0.8632 0.8999 0.9315 0.959  0.9872
#>                         PC15
#> Standard deviation      0.4388
#> Proportion of Variance 0.0128
#> Cumulative Proportion   1.0000
```

```
# Berechnung der Gesamtvarianz
Gesamtvarianz <- sum(Werte.pc$sdev^2)

# Bei sum(Werte.pc$sdev^2) wird die Summe aller 15 Standardabweichungen berechnet.

# Varianzanteil der ersten Hauptkomponente
Werte.pc$sdev[1]^2 / Gesamtvarianz
#> [1] 0.191
```

14.8.2 Scree-Plot

Der Standard-Plot `plot()` für die PCA ist ein *Scree-Plot*³, Dieser zeigt uns in Reihenfolge der Hauptkomponenten jeweils die durch diese Hauptkomponente erfasste Streuung (Varianz). Wir plotten ein Liniendiagramm mit dem Argument `type = "l"` (l für Linie), s. Abb. 14.2).

```
plot(Werte.pc, type="l")
```

Wir sehen in Abb. 14.2, dass bei den Werte-Daten der Anteil der Streuung nach der fünften Komponente nicht mehr wesentlich abnimmt. Es soll die Stelle gefunden werden, ab der

³scree: engl. “Geröll”

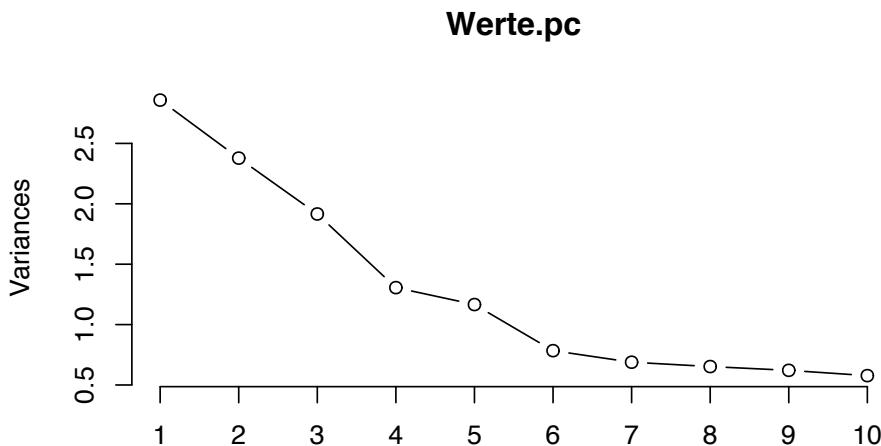


Abbildung 14.2: Screeplot

die Varianzen der Hauptkomponenten deutlich kleiner sind. Je kleiner die Varianzen, desto weniger Streuung erklärt diese Hauptkomponente.

14.8.3 Ellbogen-Kriterium

Nach dem *Ellbogen-Kriterium* werden alle Hauptkomponenten berücksichtigt, die links von der Knickstelle im Scree-Plot liegen. Gibt es mehrere Knicks, dann werden jene Hauptkomponenten ausgewählt, die links vom rechtenen Knick liegen. Gibt es keinen Knick, dann hilft der Scree-Plot nicht weiter. Bei den Werte-Daten tritt der Ellbogen, je nach Betrachtungsweise, entweder bei vier oder sechs Komponenten auf. Dies deutet darauf hin, dass die ersten fünf Komponenten die meiste Streuung in den Werte-Daten erklären.

14.8.4 Eigenwert-Kriterium

Der *Eigenwert* ist eine Metrik für den Anteil der erklärten Varianz pro Hauptkomponente. Die Anzahl Eigenwerte können wir über den Befehl `eigen()` ausgeben.

```
eigen(cor(Werte))
```

Der Eigenwert einer Komponente/ eines Faktors sagt aus, wie viel Varianz dieser Faktor an der Gesamtvarianz aufklärt. Laut dem Eigenwert-Kriterium sollen nur Faktoren mit einem *Eigenwert größer 1* extrahiert werden. Dies sind bei den Werte-Daten fünf Komponenten/ Faktoren, da fünf Eigenwerte größer 1 sind. Der Grund ist, dass Komponenten/ Faktoren mit einem Eigenwert kleiner als 1 weniger Erklärungswert haben als die ursprünglichen Variablen.

Dies kann auch grafisch mit dem `psych::VSS.Scree4` geplottet werden (s. Abb. 14.3).

⁴das Paket `psych` wird automatisch vom Paket `nfactors` gestartet, sie müssen es nicht extra starten

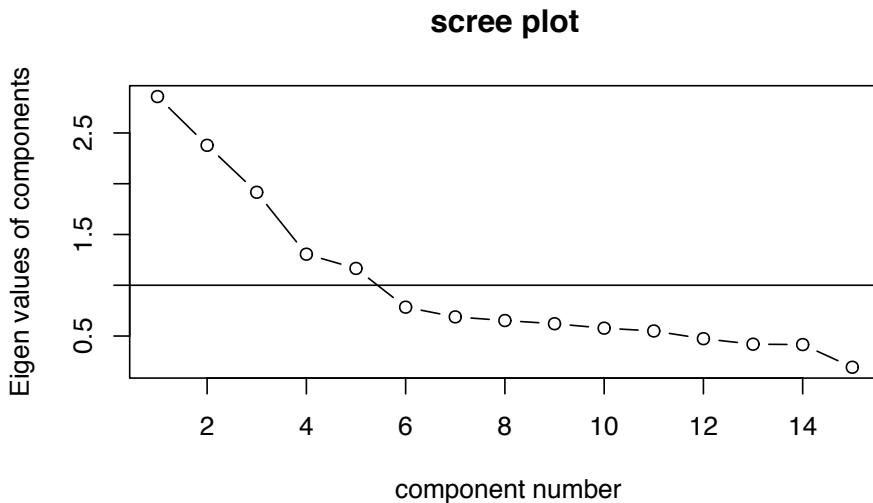


Abbildung 14.3: VSS-Screepplot

```
VSS.scree(Werte)
```

14.8.5 Biplot

Eine gute Möglichkeit die Ergebnisse der PCA zu analysieren, besteht darin, die ersten Komponenten zuzuordnen, die es uns ermöglichen, die Daten in einem niedrigdimensionalen Raum zu visualisieren. Eine gemeinsame Visualisierung ist ein *Biplot*. Ein Biplot zeigt die Ausprägungen der Fälle auf den ersten beiden Hauptkomponenten. Häufig sind die beiden ersten Hauptkomponenten schon recht aussagekräftig, vereinen also einen Gutteil der Streuung auf sich. Dazu verwenden wir `biplot()` (s. Abbildung 14.4)

```
biplot(Werte.pc)
```

Die einzelnen Ausgangsvariablen sind in Abbildung Abbildung 14.4 durch rote Pfeile (Vektoren) gekennzeichnet.

Je paralleler der Vektor einer Ausgangsvariable zur X-Achse (1. Hauptkomponente) ist, umso identischer sind sich die entsprechende Variable und die Hauptkomponente. Das hilft uns, die Hauptkomponente inhaltlich zu interpretieren. Hauptkomponenten (oder Faktoren) sollten stets inhaltlich interpretiert werden - auch wenn eine subjektive Komponente mitschwingt.

Die 1. Hauptkomponente wird offenbar stark geprägt durch die Ausgangsvariablen W6, W7, W9 und W11. Die 2. Hauptkomponente primär durch W18 und W19.

Zusätzlich erhalten wir einen Einblick in die Bewertungscluster (als dichte Bereiche von Beobachtungspunkten): Gruppen von Punkten entsprechen ähnlichen Fällen (ähnlich hinsichtlich

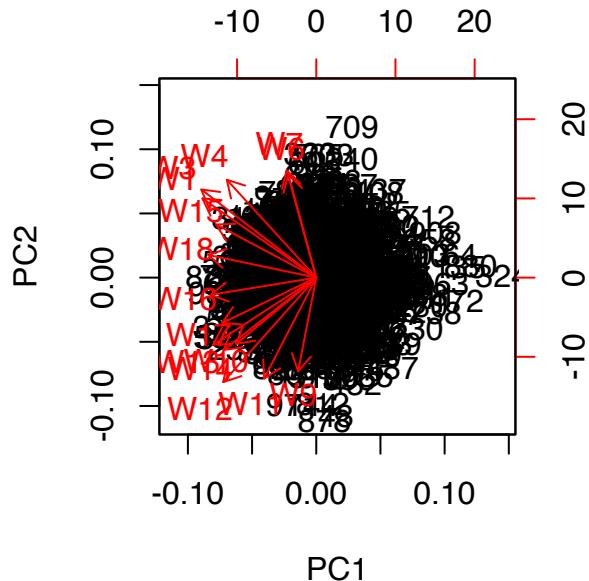


Abbildung 14.4: Ein Biplot für den Werte-Datensatz

ihrer Werte in den ersten zwei Hauptkomponenten). Der Biplot ist hier durch die große Anzahl an Beobachtung allerdings recht unübersichtlich.

14.8.6 Aufgaben



1. Ziehen Sie eine Zufallsstichprobe aus dem Datensatz, berechnen Sie die PCA erneut und betrachten Sie den Biplot. Wie stark ist die Änderung?
2. Erstellen Sie mehrere Streudiagramme und überprüfen Sie die bivariaten Zusammenhänge (die ja zur Dimensionsreduktion führen) visuell.

Am einfachsten lassen sich die Komponenten extrahieren mit dem `principal`-Befehl aus dem Paket `psych`:

```
Werte.pca <- principal(Werte, nfactors = 5, rotate = "none")
print(Werte.pca, cut = 0.5, sort = TRUE, digits = 2)
#> Principal Components Analysis
#> Call: principal(r = Werte, nfactors = 5, rotate = "none")
#> Standardized loadings (pattern matrix) based upon correlation matrix
#>    item   PC1   PC2   PC3   PC4   PC5   h2   u2 com
#> W3     2 0.57               0.65 0.35 2.8
#> W1     1 0.57               0.71 0.29 3.4
#> W18   15 0.54               0.63 0.37 3.0
#> W16   13 0.53               0.66 0.34 3.0
```

```

#> W13   10  0.51          0.64 0.36 3.7
#> W15   12              0.53 0.47 2.5
#> W17   14              0.57 0.43 3.3
#> W14   11              0.50 0.50 3.4
#> W10    7              0.54 0.46 4.1
#> W12    9          0.52          0.67 0.33 3.3
#> W11    8          0.50          0.56 0.44 3.2
#> W4     3              0.63 0.37 3.2
#> W9     6              0.57 0.43 3.3
#> W6     4      -0.51 0.71          0.88 0.12 2.3
#> W7     5      -0.54 0.66          0.89 0.11 2.7
#>
#>                               PC1  PC2  PC3  PC4  PC5
#> SS loadings            2.86 2.38 1.92 1.31 1.17
#> Proportion Var         0.19 0.16 0.13 0.09 0.08
#> Cumulative Var        0.19 0.35 0.48 0.56 0.64
#> Proportion Explained  0.30 0.25 0.20 0.14 0.12
#> Cumulative Proportion 0.30 0.54 0.74 0.88 1.00
#>
#> Mean item complexity = 3.2
#> Test of the hypothesis that 5 components are sufficient.
#>
#> The root mean square of the residuals (RMSR) is 0.07
#> with the empirical chi square 1038 with prob < 1.6e-191
#>
#> Fit based upon off diagonal values = 0.88

```

`cut = 0.5` heißt, dass nur Ladungen ab 0.5 angezeigt werden sollen. Mit `rotate = 'none'` sagen wir, dass wir keine Rotation wünschen. Eine Rotation ist

14.8.7 Interpretation der Ergebnisse der PCA

Das Ergebnis sieht sehr gut aus. Es laden immer mehrere Items (Ausgangsvariablen) (mindestens 2) hoch (> 0.5) auf einer Komponente (die mit RC1 bis RC5 bezeichnet werden, *RC* steht für *Rotated Component*). Mit "laden" ist die Parallelität der Ausgangsvariable zur Hauptkomponente gemeint. Vereinfacht gesprochen ist die Ladung die Korrelation der Items mit der jeweiligen Komponente.

Innerhalb einer PCA kann die Interpretierbarkeit über eine **Rotation** erhöht werden. Wenn die Rotation nicht ausgeschlossen wird (mit dem Argument `rotate="none"`), dann ist die Voreinstellung eine **Varimax-Rotation**.

Mit `h2` (Kommunalität) ist der Anteil eines Items bezeichnet, der durch die Komponenten insgesamt erklärt wird. Hier haben die Anzahl der Komponenten auf 5 beschränkt. Daher

wird nicht die ganze Varianz des Items erklärt.

Es gibt keine Items die auf mehr als einer Komponente hoch laden. Die Ladungen sind Korrelationskoeffizienten zwischen den Items und den Hauptkomponenten. In der Zeile *SS loadings* finden wir die Eigenwerte der fünf Hauptkomponenten (berechnet als Summe der quadrierten Ladungen). Den Anteil an der Gesamtvarianz, den sie erklären, findet man in der Zeile *Proportion Var.* Aufsummiert sind die Anteile in der Zeile *Cumulative Var.* Insgesamt werden durch die fünf Hauptkomponenten 64% der Gesamtvarianz erklärt. Die starke Hauptkomponente hat einen Eigenwert von 2.08 und erklärt 14% der Varianz.

Einzig das Item W15 lädt auf keine der Hauptkomponenten hoch.

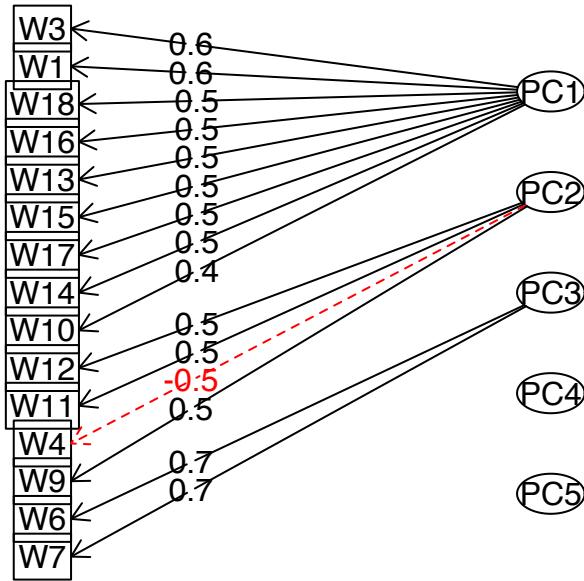
Um die inhaltliche Bedeutung der Komponenten zu interpretieren, schauen wir uns die Inhalte der jeweiligen Items an und versuchen hierfür einen inhaltlichen Gesamtbegriff zu finden. Die Erste Komponenten könnte mit **Genuss**, die zweite mit **Sicherheit**, die dritte mit **Bewusstsein**, die vierte mit **Konformismus** und die fünfte mit **Anerkennung** bezeichnet werden.

Item	Inhalt des Items
W1:	Spaß haben und Dinge tun, die Freude bereiten.
W3:	Freuden des Lebens genießen und sich selbst verwöhnen.
W4:	Überraschungen mögen und aufregendes Leben führen.
W12:	Im sicheren Umfeld leben und Gefahren meiden.
W13:	Interne und externe Sicherheit im Land.
W14:	Wertschätzen von Ordentlichkeit, Sauberkeit und ablehnen von Unordnung.
W16:	Andersartigen Menschen zuhören und Meinung verstehen.
W17:	Für die Natur einsetzen und um die Umwelt kümmern.
W18:	Interessieren, neugierig sein und versuchen Dinge verstehen.
W9:	Religiös sein und auch danach leben.
W10:	Eltern und ältere Menschen respektieren und gehorsam sein.
W11:	Demütig und bescheiden sein, keine Aufmerksamkeit anziehen.
W6:	Führung übernehmen.
W7:	Entscheidungen treffen.

Mit der Funktion `fa.diagram` kann das Ergebnis auch grafisch dargestellt werden.

```
fa.diagram(Werte.pca)
```

Factor Analysis



14.9 Exploratorische Faktorenanalyse (EFA)

Genau genommen ist der Begriff *Faktorenanalyse (FA)* ein Überbegriff für mehrere Arten von ähnlichen Verfahren der Dimensionsreduktion. Ein Beispiel für eine Art von Faktorenanalyse wäre dann die PCA. Aber der Begriff Faktorenanalyse wird auch verwendet, um eine bestimmte Art von Faktorenanalyse - sozusagen eine Faktorenanalyse im engeren Sinne - zu bezeichnen. Wir halten uns hier an letztere Begriffskonvention.

In diesem Sinne ist die *Exploratorische Faktorenanalyse (EFA)* eine Methode, um die Beziehung von Konstrukten (Konzepten), d. h. Faktoren zu Variablen zu beurteilen. Dabei werden die Faktoren als *latente Variablen* betrachtet, die nicht direkt beobachtet werden können. Stattdessen werden sie empirisch durch mehrere Variablen beobachtet, von denen jede ein Indikator der zugrundeliegenden Faktoren ist. Diese beobachteten Werte werden als *manifeste Variablen* bezeichnet und umfassen Indikatoren. Die EFA versucht den Grad zu bestimmen, in dem Faktoren die beobachtete Streuung der manifesten Variablen berücksichtigen.

Das Ergebnis der EFA ist ähnlich zur PCA: eine Matrix von Faktoren (ähnlich zu den PCA-Komponenten) und ihre Beziehung zu den ursprünglichen Variablen (Ladung der Faktoren auf die Variablen). Im Gegensatz zur PCA versucht die EFA, Lösungen zu finden, die in den *manifesten variablen maximal interpretierbar* sind. Im Allgemeinen versucht sie, Lösungen zu finden, bei denen eine kleine Anzahl von Ladungen für jeden Faktor sehr hoch ist, während andere Ladungen für diesen Faktor gering sind. Wenn dies möglich ist, kann dieser Faktor mit diesem Variablen-Set interpretiert werden.

14.9.1 Finden einer EFA Lösung

Als erstes muss die Anzahl der zu schätzenden Faktoren bestimmt werden. Hierzu verwenden wir wieder das Ellbow-Kriterium und das Eigenwert-Kriterium. Beide Kriterien haben wir schon bei der PCA verwendet, dabei kommen wir auf 5 Faktoren.

Durch das Paket `nFactors` bekommen wir eine ausgedehntere Berechnung der Scree-Plot Lösung mit dem Befehl `nScree()` - es werden noch weitere, sophistiziertere Methoden zur Berechnung der ‘richtigen’ Anzahl von Faktoren eingesetzt. Wir sparen uns hier die Details.

```
nScree(Werte)
#>   noc naf nparallel nkaiser
#> 1   5   3       5      5
```

`nScree` gibt vier methodische Schätzungen für die Anzahl an Faktoren durch den Scree-Plot aus. Wir sehen, dass drei von vier Methoden fünf Faktoren vorschlagen. Nach kurzer Überlegung und Blick aus dem Fenster entscheiden wir uns für 5 Faktoren.

14.9.2 Schätzung der EFA

Eine EFA wird geschätzt mit dem Befehl `factanal(x, factors = k)`, wobei `k` die Anzahl Faktoren angibt und `x` den Datensatz.

```
Werte.fa<-factanal(Werte, factors = 5)
Werte.fa
```

Eine übersichtlichere Ausgabe bekommen wir mit dem `print` Befehl, in dem wir zusätzlich noch die Dezimalstellen kürzen mit `digits = 2`, alle Ladungen kleiner als 0,5 ausblenden mit `cutoff = .4` und die Ladungen mit `sort = TRUE` so sortieren, dass die Ladungen, die auf einen Faktor laden, untereinander stehen.

```
print(Werte.fa, digits = 2, cutoff = .4, sort = TRUE)
```

Standardmäßig wird bei `factanal()` eine *Varimax-Rotation* durchgeführt (das Koordinatensystem der Faktoren wird so rotiert, dass eine optimale Zuordnung zu den Variablen erfolgt). Bei Varimax gibt es keine Korrelationen zwischen den Faktoren. Sollen Korrelationen zwischen den Faktoren zugelassen werden, empfiehlt sich die Oblimin-Rotation mit dem Argument `rotation="oblimin"` aus dem Paket `GPArotation`.

Das eine Rotation sinnvoll ist, kann man sich am einfachsten an einem Diagramm verdeutlichen (s. Abbildung 14.5, (Fjalnes 2014)).

Das Rotieren kann man sich als Drehen des Koordinatensystems vorstellen. Durch die Rotation sind die Items ‘näher’ an den Faktoren: Die Faktorladung zu einem Faktor wurde größer,

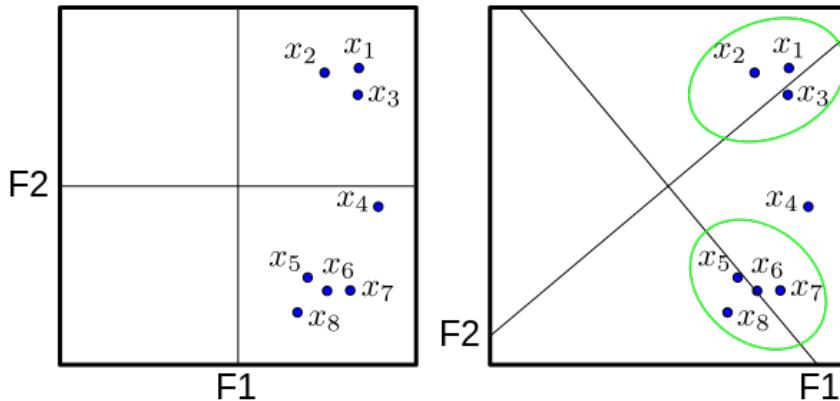


Abbildung 14.5: Beispiel für eine rechtwinklige Rotation

zum anderen Faktor hingegen geringer. Damit wurde die Ladung, also die Zuordnung der Items zu den Faktoren, insgesamt klarer, besser. Das wollen wir. Übrigens: Der Winkel der Achsen ist beim Rotieren gleich (rechtwinklig, orthogonaloal) geblieben. Daher spricht man von einer rechtwinkligen oder orthogonalen Rotation. Man kann auch die Achsen unterschiedlich rotieren, so dass sie nicht mehr rechtwinklig sind. Das könnte die Ladung noch klarer machen, führt aber dazu, dass die Faktoren dann korreliert sind. Korrelierte Faktoren sind oft nicht wünschenswert, weil ähnlich.

14.9.3 Vertiefung: Heatmap mit Ladungen

In der obigen Ausgabe werden die Item-to-Faktor-Ladungen angezeigt. Im zurückgegebenen Objekt `Werte.fa` sind diese als `$loadings` vorhanden. Wir können die Item-Faktor-Beziehungen mit einer Heatmap von `$loadings` visualisieren aus dem Paket `gplots`⁵, s. Abb. 14.6:

```
heatmap.2(Werte.fa$loadings,
           dendrogram = "both",
           labRow = NULL,
           labCol = NULL,
           cexRow=1,
           cexCol=1,
           margins = c(7,7),
           trace = "none",
           #lmat = rbind(c(0,0),c(0,1)),
           lhei = c(1,4),
           keyszie=0.75,
           key.par = list(cex=0.5)
```

⁵bereits automatisch geladen

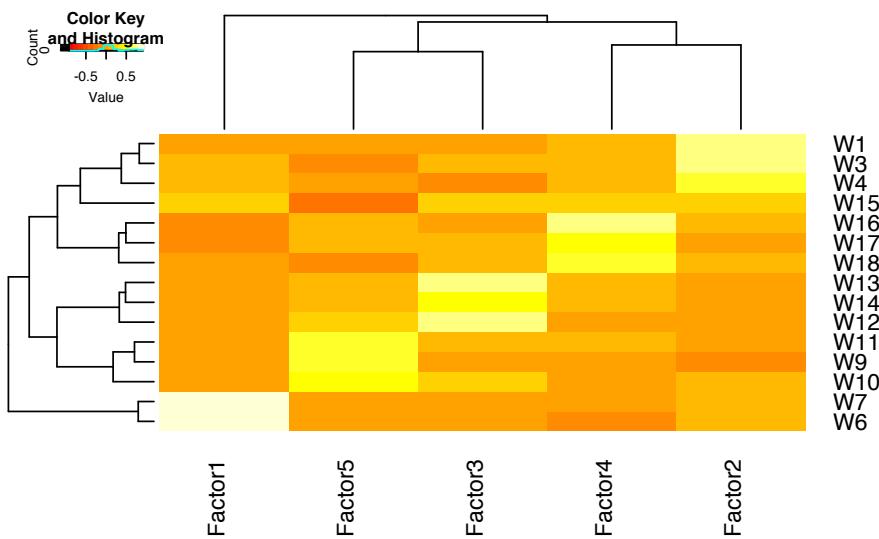


Abbildung 14.6: Heatmap einer EFA

)

Das Ergebnis aus der Heatmap zeigt eine deutliche Trennung der Items in 5 Faktoren, die interpretierbar sind als *Anerkennung*, *Genuss*, *Sicherheit*, *Bewusstsein* und *Konformismus*.

14.9.4 Berechnung der Faktor-Scores

Zusätzlich zur Schätzung der Faktorstruktur kann die EFA auch die latenten Faktorwerte für jede Beobachtung schätzen. Die gängige Extraktionsmethode ist die Bartlett-Methode, worauf wir hier nicht weiter eingehen. Kurz gesagt: Jeder Fall (jede Zeile im Datensatz, jede Person) bekommt einen Wert pro Komponente bzw. Faktor, man spricht von Faktor-Scores oder Faktorwerten der Beobachtungen.

```
Werte.ob <- factanal(Werte, factors = 5, scores = "Bartlett")
Werte.scores <- data.frame(Werte.ob$scores)
names(Werte.scores) <- c("Anerkennung", "Genuss", "Sicherheit", "Bewusstsein", "Konformismus")
head(Werte.scores)
#>   Anerkennung Genuss Sicherheit Bewusstsein Konformismus
#> 1      1.380  0.985     0.563     0.173    -0.106
#> 2     -1.404 -0.721     1.597     1.166     0.695
#> 3      1.532 -0.657     1.672    -2.003     0.239
#> 4     -0.579  2.344    -1.056    -1.120    -0.118
#> 5      0.234 -1.652     1.189    -1.701     0.437
#> 6     -0.130  0.111    -1.053     0.791    -1.003
```

Wir haben nun anstatt der 15 Variablen 5 Faktoren mit Scores. Die Dimensionen wurden um

ein Drittel reduziert.

14.10 Interne Konsistenz der Skalen

Das einfachste Maß für die *interne Konsistenz* ist die *Split-Half-Reliabilität*. Die Items werden in zwei Hälften unterteilt und die resultierenden Scores sollten in ihren Kenngrößen ähnlich sein. Hohe Korrelationen zwischen den Hälften deuten auf eine hohe interne Konsistenz hin. Das Problem ist, dass die Ergebnisse davon abhängen, wie die Items aufgeteilt werden. Ein üblicher Ansatz zur Lösung dieses Problems besteht darin, den Koeffizienten *Alpha* (*Cronbachs Alpha*) zu verwenden.

Der Koeffizient *Alpha* ist der Mittelwert aller möglichen Split-Half-Koeffizienten, die sich aus verschiedenen Arten der Aufteilung der Items ergeben. Dieser Koeffizient variiert von 0 bis 1. Inhaltlich ist Alpha eine Art mittlere Korrelation, die sich ergibt wenn man alle Items (paarweise) miteinander korriert: I1-I2, I1-I3,...

Faustregeln für die Bewertung von Cronbachs Alpha:

Alpha	Bedeutung
größer 0,9	exzellent
größer 0,8	gut
größer 0,7	akzeptabel
größer 0,6	fragwürdig
größer 0,5	schlecht

Wir bewerten nun die interne Konsistenz der Items Beispielhaft für das Konstrukt **Sicherheit** und nehmen zur Demonstration das Item W15 mit in die Analyse auf.

```
Werte %>%
  select(W12, W13, W14, W15) -> df

psych::alpha(df, check.keys = TRUE)
#>
#> Reliability analysis
#> Call: psych::alpha(x = df, check.keys = TRUE)
#>
#>   raw_alpha std.alpha G6(smc) average_r S/N    ase mean sd
#>       0.64      0.63      0.6       0.3 1.7 0.018  5.4   1
#>
#>   lower alpha upper      95% confidence boundaries
#>   0.6 0.64 0.67
#>
```

```

#> Reliability if an item is dropped:
#>   raw_alpha std.alpha G6(smc) average_r S/N alpha se
#> W12      0.51      0.51    0.42     0.26 1.05  0.026
#> W13      0.50      0.49    0.42     0.24 0.97  0.027
#> W14      0.54      0.53    0.49     0.28 1.15  0.025
#> W15      0.68      0.69    0.61     0.42 2.19  0.017
#>
#> Item statistics
#>   n raw.r std.r r.cor r.drop mean  sd
#> W12 1000  0.75  0.73  0.64   0.49  5.0 1.6
#> W13 1000  0.75  0.75  0.65   0.52  5.7 1.4
#> W14 1000  0.73  0.72  0.55   0.45  5.2 1.6
#> W15 1000  0.52  0.56  0.29   0.23  5.6 1.3
#>
#> Non missing response frequency for each item
#>   1   2   3   4   5   6   7 miss
#> W12 0.02 0.06 0.12 0.15 0.21 0.25 0.19   0
#> W13 0.01 0.02 0.05 0.11 0.18 0.27 0.35   0
#> W14 0.02 0.05 0.10 0.14 0.19 0.25 0.25   0
#> W15 0.01 0.02 0.06 0.11 0.22 0.31 0.28   0

```

Bei dem Konstrukt **Sicherheit** können wir durch Elimination von W15 das Cronbachs Alpha von 0,64 auf einen fast akzeptablen Wert von 0,68 erhöhen.

Das Argument `check.keys=TRUE` gibt uns eine Warnung aus, sollte die Ladung eines oder mehrerer Items negativ sein. Dies ist hier nicht der Fall, somit müssen auch keine Items recodiert werden.

14.11 Befehlsübersicht

Tabelle 14.3 fasst die R-Funktionen dieses Kapitels zusammen.

Tabelle 14.3: Befehle des Kapitels 'Dimensionsreduktion'

Paket::Funktion	"Beschreibung"
cor	"Berechnet eine Korrelationsmatrix."
read.csv2	"Liest eine 'deutsche' CSV-Datei ein."
glimpse	"Wirft einen Blick (to glimpse) in den Datensatz."
scale	"führt eine z-Transformation durch"
corrplot::corrplot	"Plottet einen Korrelationsplot."
na.omit	"Schließt Zeilen mit fehlenden Werten von Datensatz aus."
pr.comp	"Berechnet Hauptkomponentenanalyse."
eigen	"Berechnet Eigenwerte."
psych::VSS.scree	"Plottet einen Screeplot."
biplot	"Plottet einen Biplot."
psych::principal	"Berechnet die Statistiken für eine Hauptkomponentenanalyse"
psych::fa.diagram	"Plottet ein Pfaddiagramm für eine Faktorenanalyse"
nFactors::nscreen	"Gibt verschiedenen Vorschläge für die Anzahl der 'richtigen' Faktoren"
factanal	"Berechnet eine Faktorenanalyse"
gplots::heatmap.2	"Plottet ein Heatmap"
factanal	"Berechnet Faktor-Scores"
psych::alpha	"Berechnet Cronbachs Alpha und weitere Statistiken"

Kapitel 15

Vertiefung: 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(pdftools) # PDF einlesen
library(downloader) # Daten herunterladen
library(lsa) # Stopwörter
library(SnowballC) # Wörter trunkieren
library(wordcloud) # Wordcloud anzeigen
```

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 antagonierenden 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.

15.1 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 Term steht (Silge und Robinson 2016).

15.2 Grundlegende Analyse

15.2.1 Tidy Text Dataframes

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.

```
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.

Und “dehnen” diesen Dataframe zu einem *tidy text* Dataframe.

```
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’ Spalte heißen soll; `input` sagt der Funktion, welche Spalte sie als ihr Futter betrachten soll (welche Spalte in *tidy text* umgewandelt

¹Nach dem Gedicht “Jahrgang 1899” von Erich Kästner

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.

```
text_df %>%
  unnest_tokens(wort, text) %>%
  filter(str_detect(wort, "[a-z]"))
#> # A tibble: 24 x 2
#>   Zeile  wort
#>   <int> <chr>
#> 1     1  wir
#> 2     1  haben
#> 3     1  die
#> # ... with 21 more rows
```

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.

15.2.2 Text-Daten einlesen

Nun lesen wir Text-Daten ein; das können beliebige Daten sein². Eine gewisse Reichhaltigkeit ist von Vorteil. Nehmen wir das Parteidokument der Partei AfD³. Vor dem Hintergrund des

²Ggf. benötigen Sie Administrator-Rechte, um Dateien auf Ihre Festplatte zu speichern.

³https://www.alternativefuer.de/wp-content/uploads/sites/7/2016/05/2016-06-27_afd-grundsatzprogramm_web-version.pdf

Erstarkens des Populismus weltweit und der großen Gefahr, die davon ausgeht - man blicke auf die Geschichte Europas in der ersten Hälfte des 20. Jahrhunderts - verdient erfordert der politische Prozess und speziell Neuentwicklungen darin unsere besondere Beachtung.

```
afd_url <- paste0("https://www.alternativefuer.de",
"/wp-content/uploads/sites/7/2016/05/",
"2016-06-27_afd-grundsatzprogramm_web-version.pdf")

afd_pfad <- "data/afd_programm.pdf"

download(afd_url, afd_pfad)

afd_raw <- pdf_text(afd_pfad)
```

Mit `head(afd_raw)` können Sie sich den Beginn dieses Textvektor anzeigen lassen.

Mit `download` haben wir die Datei mit der Url `afd_url` heruntergeladen und als `afd_pfad` gespeichert. Für uns ist `pdf_text` sehr praktisch, da diese Funktion Text aus einer beliebigen PDF-Datei in einen Text-Vektor einliest. `head(afd_raw, 1)` liest das 1. Element (und nur das erste) aus `afd_raw` aus.

Der Vektor `afd_raw` hat 96 Elemente (entsprechend der Seitenzahl des Dokuments); zählen wir die Gesamtzahl an Wörtern. Dazu wandeln wir den Vektor in einen tidy text Dataframe um. Auch die Stopwörter entfernen wir wieder wie gehabt.

```
afd_df <- data_frame(Zeile = 1:96,
                      afd_raw)
afd_df %>%
  unnest_tokens(output = token, input = afd_raw) %>%
  dplyr::filter(str_detect(token, "[a-z"])) -> afd_df

count(afd_df)
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 26396
```

Eine substanzielle Menge von Text. Was wohl die häufigsten Wörter sind?

15.2.3 Worthäufigkeiten auszählen

```
afd_df %>%
  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
#> # ... with 7,084 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)

stopwords_de <- data_frame(word = stopwords_de)

stopwords_de <- stopwords_de %>%
  rename(token = word)
# Für das Joinen werden gleiche Spaltennamen benötigt

afd_df %>%
  anti_join(stopwords_de) -> afd_df
```

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_df %>%
  count(token, sort = TRUE) -> afd_count
```

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”.

Tabelle 15.1: Die häufigsten Wörter im AfD-Parteiprogramm

token	n
deutschland	190
afd	171
programm	80
wollen	67
bürger	57
euro	55
dafür	53
eu	53
deutsche	47
deutschen	47

Tabelle 15.2: Die häufigsten Wörter im AfD-Parteiprogramm 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

```
afd_df %>%
  mutate(token_stem = wordStem(.$token, language = "german")) %>%
  count(token_stem, sort = TRUE) -> afd_count

afd_count %>%
  top_n(10) %>%
  knitr::kable(caption = "Die häufigsten Wörter im AfD-Parteiprogramm 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 `.` bei `dplyr` nur den Datensatz meint, wie er im letzten Schritt definiert war. Mit `.$token` wählen wir also die Variable `token` aus `afd_raw` aus.

15.2.4 Visualisierung

Zum Abschluss noch eine Visualisierung mit einer “Wordcloud” dazu.

```
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 beeinflussen⁴.

Weniger verspielt ist eine schlichte visualisierte Häufigkeitsauszählung dieser Art, z.B. mit Balkendiagrammen (gedreht).

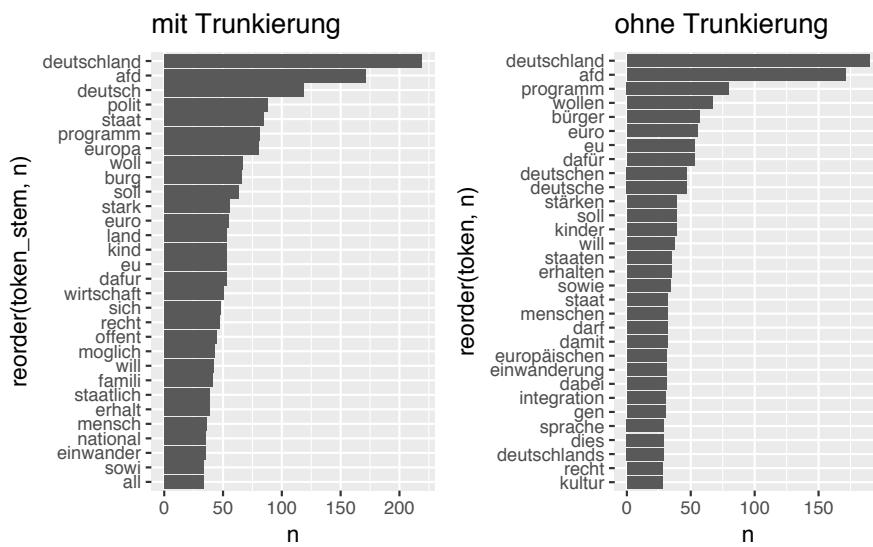
```
afd_count %>%
  top_n(30) %>%
  ggplot() +
  aes(x = reorder(token_stem, n), y = n) +
  geom_col() +
  labs(title = "mit Trunkierung") +
  coord_flip() -> p1

afd_df %>%
  count(token, sort = TRUE) %>%
  top_n(30) %>%
  ggplot() +
  aes(x = reorder(token, n), y = n) +
```

⁴<https://cran.r-project.org/web/packages/wordcloud/index.html>

```
geom_col() +
  labs(title = "ohne Trunkierung") +
  coord_flip() -> p2
```

```
library(gridExtra)
grid.arrange(p1, p2, ncol = 2)
```



Die beiden Diagramme vergleichen die trunkierten Wörter mit den nicht trunktierten 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. `grid.arrange` packt beide Plots in eine Abbildung, welche 2 Spalten (`ncol`) hat.

15.3 Aufgaben⁵



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.

⁵F, R, F, F, R, R

6. Balkendiagramme sind sinnvolle und auch häufige Diagrammtypen, um die häufigsten Wörter (oder auch Tokens) in einem Corpus darzustellen.

15.4 Befehlsübersicht

Tabelle ?? fasst die R-Funktionen dieses Kapitels zusammen.

Tabelle 15.3: Befehle des Kapitels ‘Textmining’

Paket..Befehl	Beschreibung
tidytext::unnest_tokens	Jedes Token (Wort) einer Spalte bekommt eine eigene Zeile in einem Dataframe
stringr::str_detect downloader:: download	Sucht nach einem String (Text) lädt eine Datei aus dem Internet herunter
dplyr::rename anti_join	Benennt Spalten um Führt Dataframes zusammen, so dass nicht matchende Einträge übernommen werden
wordcloud::wordcloud ggplot2::labs	Erstellt eine Wordcloud Fügt Titel oder andere Hinweise einem ggplot2-Objekt hinzu
ggplot2::coord_flip	Dreht die Achsen um 90 Grad

15.5 Verweise

- Das Buch *Tidy Text Mining* (Julia und David 2017) ist eine hervorragende Quelle vertieftem Wissens zum Textmining mit R.

Anhang A

Probeklausur



Aussagen sind entweder als “richtig” oder als “falsch” zu beantworten. Offene Fragen verlangen einen “Text” als Antwort.

1. Bei `install.packages` spielt der Parameter `dependencies = TRUE` in der Praxis keine Rolle.
2. Dateien mit der Endung `.R` sind keine Textdateien.
3. Der Befehl `read.csv` kann auch Dateien einlesen, die nicht lokal, sondern auf einem Server im Internet gespeichert sind.
4. Fehlende Werte werden in R durch `NA` kodiert.
5. Um Variablen einen Wert zuzuweisen, kann man in R den Zuweisungspfeil `<-` verwenden.
6. Die deutsche Version von R verwendet im Standard das Komma als Dezimaltrennzeichen.
7. Statistisches Modellieren verwendet die Abduktion als zentrale Denkfigur.
8. Eine Abduktion führt zu sicheren Schlüssen.
9. Das CSV-Format ist identisch zum Excel-Format, was sich auch darin zeigt, dass Excel CSV-Dateien oft problemlos öffnet.

10. Das Arbeitsverzeichnis (engl. *working directory*) ist der Ort, in dem R eine Datei, die Sie aufrufen, vermutet - sofern kein anderer Pfad angegeben ist.
11. In einer Tabelle in Normalform steht in jeder Zeile eine Variable und in jeder Spalte eine Beobachtung.
12. Die Funktion `filter` filtert Spalten aus einer Tabelle.
13. Die Funktion `select` lässt Spalten sowohl anhand ihres Namens als auch ihrer Nummer (Position in der Tabelle) auswählen.
14. Die Funktion `group_by` gruppert eine Tabelle anhand der Werte einer diskreten Variablen.
15. Die Funktion `group_by` akzeptiert nur Faktorvariablen als Gruppierungsvariablen.
16. Die Funktion `summarise` darf nur für Funktionen verwendet werden, welche genau *einen* Wert zurückliefern.
17. Was sind drei häufige Operationen der Datenaufbereitung?
18. Um Korrelationen mit R zu berechnen, kann man die Funktion `corr::correlate` verwenden.
19. `corr::correlate` liefert stets einen Dataframe zurück.
20. Tibbles sind eine spezielle Art von Dataframes.
21. Was zeigt uns “Anscombes Quartett”?
22. `ggplot` unterscheidet drei Bestandteile eines Diagramms: Daten, Geome und Transformationen.
23. Um eine kontinuierliche Variable zu plotten, wird häufig ein Histogramm verwendet.
24. Das Geom `tile` zeigt drei Variablen.
25. Geleitetes Modellieren kann unterteilt werden in prädiktives und explikatives Modellieren.
26. Der Befehl `scale` verschiebt den Mittelwert einer Verteilung auf 0 und skaliert die sd auf 1.
27. Mit “binnen” im Sinne der Datenanalyse ist gemeint, eine kategoriale Variable in eine kontinuierliche zu überführen.
28. Die Gleichung $y = ax + b$ lässt sich in R darstellen als $y \sim ax + b$.
29. R^2 , auch Bestimmtheitsmaß oder Determinationskoeffizient genannt, gibt die Vorhersagegüte im Verhältnis zu einem “Nullmodell” an.
30. Bei der logistischen Regression gilt: Bei $\beta_0 > 0$ ist die Wahrscheinlichkeit *kleiner* als 50% gibt, dass das modellierte Ereignis eintritt, wenn alle anderen Prädiktoren Null sind.

31. Die logistische Regression sollte *nicht* verwendet werden, wenn die abhängige Variable dichotom ist.
32. Die logistische Regression stellt den Zusammenhang zwischen Prädiktor und Kriterium nicht mit einer Geraden, sondern mit einer “s-förmigen” Kurve dar.
33. Bevor die Koeffizienten der logistischen Regression als Odds Ration interpretiert werden können, müssen sie “deologarithmiert” werden.
34. Unter “deologarithmieren” versteht man, die Umkehrfunktion der e-Funktion auf eine Gleichung anzuwenden.
35. Wendet man die “normale” Regression an, um eine dichotome Variable als Kriterium zu modellieren, so kann man Wahrscheinlichkeiten größer als 1 und kleiner als 0 bekommen.
36. Eine typische Idee der Clusteranalyse lautet, die Varianz innerhalb der Cluster jeweils zu maximieren.
37. Bei einer k-means-Clusteranalyse darf man nicht die Anzahl der Cluster vorab festlegen; vielmehr ermittelt der Algorithmus die richtige Anzahl der Cluster.
38. Für die Wahl der “richtigen” Anzahl der Cluster kann das “Ellbogen-Kriterium” als Entscheidungsgrundlage herangezogen werden.
39. Ein “Screeplot” stellt die Varianz innerhalb der Cluster als Funktion der Anzahl der Cluster dar (im Rahmen der Clusteranalyse).
40. Die euklidische Distanz zwischen zwei Objekten in der Ebene lässt sich mit dem Satz des Pythagoras berechnen.

Anhang B

Lösungen

1. Falsch
2. Falsch
3. Richtig
4. Richtig
5. Richtig
6. Falsch
7. Richtig
8. Falsch
9. Falsch
10. Richtig
11. Falsch
12. Falsch
13. Falsch
14. Richtig
15. Richtig
16. Falsch
17. Richtig
18. Auf fehlende Werte prüfen, Fälle mit fehlenden Werte löschen, Fehlende Werte ggf. ersetzen, Nach Fehlern suche, Ausreiser identifizieren, Hochkorrelierte Variablen finden, z-Standardisieren, Quasi-Konstante finden, Auf Normalverteilung prüfen, Werte umkodieren und “binnen”.
19. Richtig
20. Richtig
21. Richtig
22. 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.
23. Falsch

- 24. Richtig
- 25. Richtig
- 26. Falsch
- 27. Richtig
- 28. Falsch
- 29. Richtig
- 30. Richtig
- 31. Falsch
- 32. Falsch
- 33. Richtig
- 34. Richtig
- 35. Falsch. Richtig wäre: Die Umkehrfunktion des Logarithmus, also die e-Funktion, auf eine Gleichung anzuwenden.
- 36. Richtig
- 37. Falsch
- 38. Falsch. Richtig wäre: Man gibt die Anzahl der Cluster vor. Dann vergleicht man die Varianz within der verschiedenen Lösungen.
- 39. Richtig
- 40. Richtig
- 41. Richtig

Anhang C

Hinweise

Anhang D

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. Zugebenermaßen braucht es etwas Zeit, bis man ein paar Brocken "Errisch" spricht. Um den Einstieg zu erleichern, 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 E

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 F

Zitationen

Kunstwerke (Bilder) sind genau wie Standard-Literatur im Text zitiert. Alle Werke (auch Daten und Software) finden sich im Literaturverzeichnis.

Anhang G

Lizenz

Dieses Skript ist publiziert unter CC-BY-NC-SA 3.0 DE¹.



¹<https://creativecommons.org/licenses/by-nc-sa/3.0/de/>

Anhang H

Autoren

Sebastian Sauer schrieb den Hauptteil dieses Skripts. *Oliver Gansser* schrieb das Kapitel zur Dimensionsreduktion. *Karsten Lübke* schrieb den Großteil des Kapitels zur Regression und zur Clusteranalyse sowie Teile des Kapitels ‘Rahmen’. *Matthias Gehrke* schrieb den Großteil des Kapitels zur logistischen Regression.

Anhang I

Danke

Norman Markgraf hat umfangreich Fehler gejagt und Verbesserungen ~~angemahnt~~ vorgenommen. Der Austausch mit den ifes-Mitgliedern hielt die Flamme am Köcheln. Eine Reihe weiterer Kollegen standen mit Rat und Tat zur Seite. Die Hochschulleitung sowie das Dekanat für Wirtschaftspsychologie hat dieses Projekt unterstützt. Die Abteilung Medienentwicklung der FOM hat bei Fragen rund um die Veröffentlichung geholfen. Last but not least: Viele Studierenden wiesen auf Inkonsistenzen, Fehler und Unklarheiten hin. Ihnen allen: Vielen Dank!

Anhang J

Zitation dieses Skripts

Bitte zitieren Sie das Skript so:

Sauer, S. (2017). *Praxis der Datenanalyse*. Skript zum Modul im MSc.-Studiengang “Wirtschaftspsychologie & Consulting” an der FOM. FOM Nürnberg. DOI: 10.5281/zenodo.580649.

Mehr Infos zum DOI hier: <https://zenodo.org/badge/latestdoi/81811975>

Ein Bib-File um dieses Skript zu zitieren finden Sie hier: https://raw.githubusercontent.com/sebastiansauer/Praxis_der_Datenanalyse/master/Praxis_der_Datenanalyse.bib.

Anhang K

Kontakt

Wenn Sie einen Fehler oder Verbesserungshinweise berichten möchten, können Sie unter https://github.com/sebastiansauer/Praxis_der_Datenanalyse/issues einen “Issue” einreichen (Button “New Issue”). Alternativ können Sie Sebastian Sauer und die anderen Autoren über den Online Campus der FOM kontaktieren (eine Nachricht schreiben). Sebastian Sauer können Sie via Twitter folgen (https://twitter.com/sauer_sebastian) oder seinen Blog lesen (<https://sebastiansauer.github.io>).

Anhang L

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 M

Sonstiges

Aus Gründen der Lesbarkeit wird das männliche Generikum verwendet, welches Frauen und Männer in gleichen Maßen ansprechen soll.

Anhang N

Literaturverzeichnis

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>.

Auguie, Baptiste. 2016. *gridExtra: Miscellaneous Functions for „Grid“ Graphics*. <https://CRAN.R-project.org/package=gridExtra>.

Beaujean, A. Alexander. 2012. *BaylorEdPsych: R Package for Baylor University Educational Psychology Quantitative Courses*. <https://CRAN.R-project.org/package=BaylorEdPsych>.

Benoit, Kenneth, und Paul Nulty. 2016. *quanteda: Quantitative Analysis of Textual Data*. <https://CRAN.R-project.org/package=quanteda>.

Bouchet-Valat, Milan. 2014. *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. <https://CRAN.R-project.org/package=SnowballC>.

Briggs, William M. 2008. *Breaking the Law of Averages: Real-Life Probability and Statistics in Plain English*. Lulu.com.

———. 2016. *Uncertainty: The Soul of Modeling, Probability & Statistics*. Springer.

Bryant, PG, und MA Smith. 1995. „Practical Data Analysis: Case Studies in Business Statistics, Homewood, IL: Richard D“. Irwin Publishing.

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:10.1007/978-3-319-14436-8¹.
- Cleveland, William S. 1993. *Visualizing Data*. Hobart Press.
- 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.
- 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.
- 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“. 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:

¹<https://doi.org/10.1007/978-3-319-14436-8>

587–606. doi:10.1016/j.socec.2004.09.033².

God. 2016. „I don't care about you. Please share this with friends.“ Twitter Tweet. *TheTweetOfGod*. <https://twitter.com/TheTweetOfGod/status/688035049187454976>.

Grolemund, 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.

Hatzinger, Reinhold, 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.1002106³.

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. <https://books.google.de/books?id=gDuRBAAQBAJ>.

Icon-Pond. 2016. „Education. 35 Icons.“ Flaticon. <http://www.flaticon.com/authors/popcorns-arts>.

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>.

²<https://doi.org/10.1016/j.socec.2004.09.033>

³<https://doi.org/10.1371/journal.pbio.1002106>

`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.

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 Science* 1 (5). Burlington, MA: Academic Press: 658–76.

Kuhn, Max, und Kjell Johnson. 2013. *Applied predictive modeling*. Bd. 26. Springer.

Ligges, Uwe, Martin Maechler, und Sarah Schnackenberg. 2017. *scatterplot3d: 3D Scatter Plot*. <https://CRAN.R-project.org/package=scatterplot3d>.

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.

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>.

Ooms, Jeroen. 2016. *pdftools: Text Extraction and Rendering of PDF Documents*. <https://CRAN.R-project.org/package=pdftools>.

//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.

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>.

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 Resources and Evaluation (LREC'10)*, 1168–71.

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>.

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>.

Robinson, David, und Julia Silge. 2016. *tidytext: Text Mining using 'dplyr', 'ggplot2', and Other Tidy Tools*. <https://CRAN.R-project.org/package=tidytext>.

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/4KGZH⁴.

———. 2017a. „Dataset 'predictors of performance in stats test'“. Open Science Framework.

⁴<https://doi.org/10.17605/OSF.IO/4KGZH>

doi:10.17605/OSF.IO/SJHUY⁵.

———. 2017b. „Dataset 'Height and shoe size'“. Open Science Framework. doi:10.17605/OSF.IO/JA9DW⁶.

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.147241.13309⁷.

Schloerke, Barret, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, 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:10.1024/0044-3514.38.4.261⁸.

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.56714⁹.

Silge, Julia, und David Robinson. 2016. „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.00037¹⁰.

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).

Suppes, Patrick, und Joseph L Zinnes. 1962. *Basic measurement theory*. Institute for mathematical studies in the social sciences.

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 Januchowski, Luke Shoo, und Collin Storlie.

⁵<https://doi.org/10.17605/OSF.IO/SJHUY>

⁶<https://doi.org/10.17605/OSF.IO/JA9DW>

⁷<https://doi.org/10.15200/winn.147241.13309>

⁸<https://doi.org/10.1024/0044-3514.38.4.261>

⁹<https://doi.org/10.5281/zenodo.56714>

¹⁰<https://doi.org/10.21105/joss.00037>

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/bf03194105¹¹.

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>.

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.2016.01832¹².

Wickham, Hadley. 2009. *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. <http://ggplot2.org>.

———. 2014. „Tidy Data“. *Journal of Statistical Software* 59 (1): 1–23. doi:10.18637/jss.v059.i10¹³.

———. 2016a. *reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package.* <https://CRAN.R-project.org/package=reshape2>.

———. 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,*

¹¹<https://doi.org/10.3758/bf03194105>

¹²<https://doi.org/10.3389/fpsyg.2016.01832>

¹³<https://doi.org/10.18637/jss.v059.i10>

- 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>.
- Zumel, Nina, John Mount, und Jim Porzak. 2014. *Practical data science with R.* Manning.

Index

- 68
Überanpassung, 131
- Allgemeines Lineares Modells, 177
angeleitetes Lernen, 129
- Befehl, Funktion, 15
Bestimmtheitsmaß, 158
Bias, 133
Binnen, 68
Biplot, 248
- Cohens d, 146
Cronbachs Alpha, 256
- datengenerierende Maschine, 128
Datenjudo, 32
Determinationskoeffizient, 158
deterministisch, 128
Dimensionsreduzierendes Modellieren, 130
dplyr::arrange, 38
dplyr::count, 47
dplyr::filter, 34
dplyr::mutate, 53
dplyr::n, 46
dplyr::select, 37
dplyr::summarise, 44
Durchpfeifen, 51
- Effektstärke, 146
Eigenwert, 247
Einflussgrößen, 126
Ellbogen-Kriterium, 247
Erklären, 129
euklidischen Abstand, 230
Explikatives Modellieren, 129
Exploratorische Faktorenanalyse, 240, 252
Faktorenanalyse, 252
- Fallreduzierendes Modellieren, 130
Funktion, 5
- Geleitetes Modellieren, 129
- Hauptachsenanalyse, 241
Hauptkomponentenanalyse, 240
- Interaktionseffekts, 169
interne Konsistenz, 256
Item, 240
- Klassifikation, 128, 129, 184
Konfidenzintervalle, 145
Konfusionsmatrix, 185
Konsole, 5
Konstrukte, 240
Kriterium, 126
Kriterium der Kleinsten Quadrate, 161
- Lagemaße, 56
Lernen ohne Anleitung, 130
logistische Funktion, 175
Logit, 178
- multivariat, 167
- Nullhypothese, 140
Nullhypotesen-Signifikanztesten, 140
- Operationalisierung, 240
Ordinary Least Squares, 161
overfitting, 131
- p-Wert, 140
Parameter eines R-Befehls, 15
PCA, 240
Pfeife, 51
Prädiktives Modellieren, 129
Prädiktoren, 126

R-Skript, 12
Reduzieren, 130
Regression, 154
Residuen, 162
robust, 133
ROC, 187

Signifikanz, 141
Skript-Fenster, 5
Split-Half-Reliabilität, 256
Standardnormalverteilung, 66
Streuungsmaße, 56

Umgebung, 5
Umkodieren, 68
underfitting, 132
Ungeleiteten Modellieren, 129
unsupervised learning, 130
Unteranpassung, 132

Variablen, 13
Vorhersagefehler, 158
Vorhersagen, 129

Youden-Index, 186