

# Praxis der Datenanalyse

Sebastian Sauer

2017-03-23



# Contents

<b>Vorwort</b>	<b>xv</b>
<b>Rahmen</b>	<b>5</b>
0.1 Software . . . . .	6
0.1.1 R und RStudio installieren . . . . .	6
0.1.2 Hilfe! R tut nicht so wie ich das will . . . . .	8
0.1.3 Hier werden Sie geholfen . . . . .	9
0.1.4 Die Denk- und Gefühlswelt von R . . . . .	10
0.1.5 Pakete installieren . . . . .	11
0.1.6 R-Pakete für dieses Buch In diesem Buch verwenden wir die folgenden . . . . .	12
0.1.7 Datensätze . . . . .	13
0.2 ERMkontakt . . . . .	14
0.2.1 R als Taschenrechner Auch wenn Statistik nicht Mathe ist, so kann man mit R . . . . .	15
0.3 Was ist Statistik? Wozu ist sie gut? . . . . .	16
0.4 Verweise . . . . .	18
0.5 Versionshinweise . . . . .	19
<b>Tidy Data - Daten sauber einlesen</b>	<b>23</b>
0.6 Daten in R importieren . . . . .	23
0.7 Normalform einer Tabelle . . . . .	26
0.8 Verweise . . . . .	28
<b>Daten aufbereiten</b>	<b>29</b>
0.9 Typische Probleme . . . . .	30
0.10 Daten aufbereiten mit dplyr . . . . .	31

0.10.1	Zeilen filtern mit <code>filter</code>	32
0.10.2	Spalten wählen mit <code>select</code>	34
0.10.3	Zeilen sortieren mit <code>arrange</code>	35
0.10.4	Datensatz gruppieren mit <code>group_by</code>	39
0.10.5	Eine Spalte zusammenfassen mit <code>summarise</code>	40
0.10.6	Zeilen zählen mit <code>n</code> und <code>count</code>	42
0.10.7	Die Pfeife	44
0.10.8	Werte umkodieren und “binnen”	46
0.11	Fallstudie <code>nycflights13</code>	50
0.12	Checkliste zum Datenjudo	55
0.12.1	Auf fehlende Werte prüfen*	55
0.12.2	Fälle mit fehlenden Werte löschen	56
0.12.3	Fehlende Werte ggf. ersetzen	57
0.12.4	Nach Fehlern suchen	57
0.12.5	Ausreiser identifizieren	58
0.12.6	Hochkorrelierte Variablen finden	58
0.12.7	z-Standardisieren	59
0.12.8	Quasi-Konstante finden	60
0.12.9	Auf Normalverteilung prüfen	61
0.12.10	Mittelwerte pro Zeile berechnen	62
0.12.11	U	62
0.13	Verweise	63
<b>Daten visualisieren</b>		<b>65</b>
0.14	Häufige Arten von Diagrammen	68
0.14.1	Eine kontinuierliche Variable	69
0.14.2	Zwei kontinuierliche Variablen	73
0.14.3	Eine diskrete Variable	77
0.14.4	Zwei diskrete Variablen	81
0.14.5	Zusammenfassungen zeigen	82
0.15	Farblehre	84
0.16	Erweiterungen für <code>ggplot</code>	90
0.16.1	<code>ggpairs</code>	90
0.16.2	Correlationsplots	91
0.16.3	Weitere	92
0.17	Fallstudie Extraversion	92
0.17.1	Daten einlesen	92
0.17.2	Daten umstellen	93

0.17.3 Diagramme für Anteile . . . . .	95
0.17.4 Um 90° drehen . . . . .	96
0.17.5 Text-Labels für die Items . . . . .	97
0.17.6 Diagramm mit Häufigkeiten . . . . .	99
0.17.7 Farbschema . . . . .	100
0.18 Verweise . . . . .	101
<b>Grundlagen des Modellierens</b>	<b>105</b>
0.19 Was ist ein Modell? Was ist Modellieren? . . . . .	105
0.20 Ziele des Modellierens . . . . .	109
0.20.1 Modelle zur Erklärung vs. Modelle zur Vorhersage . .	110
0.21 Wann welches Modell? . . . . .	111
0.22 Einfache vs. komplexe Modelle: Unter- vs. Überanpassung .	111
0.23 Bias-Varianz-Abwägung . . . . .	113
0.24 Training- vs. Test-Stichprobe . . . . .	114
0.25 Modellgüte . . . . .	115
0.26 Auswahl von Prädiktoren . . . . .	117
0.27 Verweise . . . . .	117
<b>Inferenzstatistik</b>	<b>119</b>
0.28 Der p-Wert . . . . .	119
<b>Klassische lineare (numerische) Regression</b>	<b>125</b>
0.29 Einfache Regression . . . . .	125
0.30 Überprüfung der Annahmen der linearen Regression . . . . .	128
0.31 Regression mit kategorialen Prädiktoren . . . . .	131
0.32 Multiple Regression . . . . .	135
0.33 Inferenz in der linearen Regression . . . . .	139
0.34 Modellgüte bei Regressionsmodellen . . . . .	140
0.34.1 Mittlere Quadratfehler . . . . .	142
0.34.2 R-Quadrat ( $R^2$ ) . . . . .	142
0.34.3 Likelihood and Friends . . . . .	143
0.35 Vertiefungen zum Regressionmodell . . . . .	144
0.35.1 Modellwahl . . . . .	144
0.35.2 Interaktionen . . . . .	147
0.35.3 Weitere Modellierungsmöglichkeiten . . . . .	148
0.35.4 Prognoseintervalle . . . . .	149
0.36 Übung: Teaching Rating . . . . .	151

0.37 Fallstudie zu Overfitting . . . . .	151
0.38 Literatur . . . . .	152
<b>Klassifizierende Regression</b>	<b>155</b>
0.39 Vorbereitung . . . . .	155
0.40 Problemstellung . . . . .	156
0.41 Die Idee der logistischen Regression . . . . .	158
0.42 Welche Unterschiede zur linearen Regression gibt es in der Ausgabe? . . . . .	160
0.43 Interpretation der Koeffizienten . . . . .	161
0.43.1 y-Achsenabschnitt (Intercept) $\beta_0$ . . . . .	161
0.43.2 Steigung $\beta_i$ mit $i = 1, 2, \dots, K$ . . . . .	161
0.44 Kategoriale Variablen . . . . .	162
0.45 Multiple logistische Regression . . . . .	165
0.46 Modell- bzw. Klassifikationsgüte . . . . .	167
0.47 Erweiterungen . . . . .	170
0.47.1 Modellschätzung . . . . .	170
0.47.2 Likelihood Quotienten Test . . . . .	170
0.47.3 Pseudo- $R^2$ . . . . .	171
0.48 Übung: Rot- oder Weißwein? . . . . .	172
0.49 Literatur . . . . .	173
<b>Baumbasierte Verfahren</b>	<b>175</b>
0.50 Konjunkturanalyse . . . . .	175
0.51 Regressionsbäume . . . . .	177
0.52 Kreuzvalidierung . . . . .	179
0.52.1 Anpassungsgüte . . . . .	179
0.52.2 Prognosegüte . . . . .	181
0.53 Klassifikationsbäume . . . . .	183
0.53.1 Kreuzvalidierung . . . . .	185
0.54 Parameter <code>rpart</code> . . . . .	188
0.55 Fallstudie: Überleben auf der Titanic . . . . .	189
0.55.1 Daten und Pakete laden . . . . .	189
0.55.2 Erster Blick . . . . .	190
0.55.3 Welche Variablen sind interessant? . . . . .	191
0.55.4 Univariate Häufigkeiten . . . . .	191
0.55.5 Bivariate Häufigkeiten . . . . .	193
0.55.6 Signifikanztest . . . . .	196

0.55.7 Effektstärke . . . . .	197
0.55.8 Logististische Regression . . . . .	199
0.55.9 Effektstärken visualieren . . . . .	202
0.55.10 Fazit . . . . .	205
<b>Assoziationsanalyse</b>	<b>209</b>
0.56 Einführung . . . . .	209
0.57 Kennzahlen einer Assoziationsanalyse . . . . .	210
0.58 Beispiele . . . . .	210
0.59 Assoziationsanalyse mit R . . . . .	211
0.60 Fallstudie Lebensmittelwaren . . . . .	211
0.60.1 Regeln finden . . . . .	213
0.60.2 Visualisierung . . . . .	216
0.61 Literatur . . . . .	217
0.62 Versionshinweise: . . . . .	218
<b>Clusteranalyse</b>	<b>219</b>
0.63 Hierarchische Clusteranalyse . . . . .	221
0.64 k-Means Clusteranalyse . . . . .	225
0.65 Übung: B3 Datensatz . . . . .	228
0.65.1 Literatur . . . . .	228
<b>Dimensionsreduktion</b>	<b>229</b>
0.66 Einführung . . . . .	229
0.66.1 Gründe für die Notwendigkeit der Datenreduktion . . . . .	229
0.66.2 Benötigte Pakete . . . . .	230
0.66.3 Daten einlesen . . . . .	230
0.66.4 Neuskalierung der Daten . . . . .	232
0.66.5 Zusammenhänge in den Daten . . . . .	234
0.66.6 Aggregation der durchschnittlichen Bewertungen nach Marke . . . . .	235
0.66.7 Visualisierung der aggregierten Markenbewertungen . . . . .	236
0.67 Hauptkomponentenanalyse (PCA) . . . . .	238
0.67.1 Bestimmung der Anzahl der Hauptkomponenten . . . . .	238
0.67.2 Scree-Plot . . . . .	239
0.67.3 Elbow-Kriterium . . . . .	239
0.67.4 Biplot . . . . .	240
0.67.5 Wahrnehmungsraum . . . . .	241

0.68 Exploratorische Faktorenanalyse (EFA) . . . . .	242
0.68.1 Finden einer EFA Lösung . . . . .	243
0.68.2 Heatmap mit Ladungen . . . . .	247
0.68.3 Berechnung der Faktor-Scores . . . . .	248
0.69 Interne Konsistenz von Skalen . . . . .	251
0.69.1 Übung . . . . .	253
0.69.2 Literatur . . . . .	254
<b>Textmining</b>	<b>255</b>
0.70 Einführung . . . . .	256
0.70.1 Grundbegriffe . . . . .	256
0.71 Grundlegende Analyse . . . . .	257
0.71.1 Tidy Text Dataframes . . . . .	257
0.71.2 Text-Daten einlesen . . . . .	259
0.71.3 Worthäufigkeiten auszählen . . . . .	260
0.71.4 Visualisierung . . . . .	262
0.72 Sentiment-Analyse . . . . .	264
0.72.1 Ungewichtete Sentiment-Analyse . . . . .	265
0.72.2 Anzahl der unterschiedlichen negativen bzw. positiven Wörter . . . . .	268
0.72.3 Gewichtete Sentiment-Analyse . . . . .	268
0.72.4 Tokens mit den extremsten Sentimentwerten . . . . .	269
0.72.5 Relativer Sentiments-Wert . . . . .	271
0.73 Verknüpfung mit anderen Variablen . . . . .	273
0.74 Vertiefung . . . . .	273
0.74.1 Erstellung des Sentiment-Lexikons . . . . .	273
0.75 Verweise . . . . .	275
<b>Was ist RMarkdown?</b>	<b>279</b>
0.76 Forderungen . . . . .	279
0.77 Bevor es losgeht . . . . .	284
0.78 RMarkdown in Action . . . . .	284
0.79 Die Arbeitsschritte mit RMarkdown . . . . .	286
0.80 Aufbau einer Markdown-Datei . . . . .	287
0.81 Syntax-Grundlagen von Markdown . . . . .	289
0.82 Tabellen . . . . .	290
0.83 Zitieren . . . . .	291
0.84 Kollaboration und Versionierung . . . . .	292

**CONTENTS** ix

0.85 Verweise . . . . .	292
<b>Studienpfade</b>	<b>297</b>
0.86 konsequutiver Master of Science in Wirtschaftspsychologie . . .	297
<b>Literaturverzeichnis</b>	<b>299</b>



# List of Tables

2	Eine Tabelle mit Kable . . . . .	291
---	----------------------------------	-----



# List of Figures

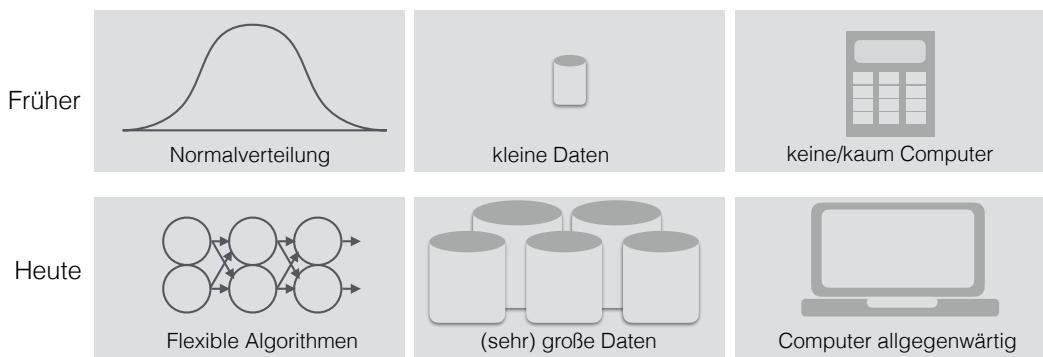
1	Daten sauber einlesen . . . . .	23
2	Daten einlesen (importieren) mit RStudio . . . . .	24
3	Illustration eines Datensatzes in Normalform . . . . .	27
4	Daten aufbereiten . . . . .	29
5	Zeilen filtern . . . . .	32
6	Spalten auswählen . . . . .	34
7	Spalten sortieren . . . . .	37
8	Datensätze nach Subgruppen aufteilen . . . . .	39
9	Spalten zu einer Zahl zusammenfassen . . . . .	40
10	Das 'Durchpeifen' . . . . .	45
11	Korr . . . . .	91
12	Modellieren . . . . .	106
13	Ein Beispiel für Modellieren . . . . .	107
14	Ein etwas aufwändigeres Modell . . . . .	107
15	Modelle mit schwarzer Kiste . . . . .	108
16	Welches Modell passt am besten zu diesen Daten? . . . . .	111
17	Der größte Statistiker des 20. Jahrhunderts ( $p < .05$ ) . . . . .	119
18	Der p-Wert wird oft als wichtig erachtet . . . . .	120
19	Neue Rmd-Datei erstellen . . . . .	285
20	Wir stricken . . . . .	285
21	HTML-Datei aus Rmd-Datei erstellt . . . . .	286
22	Die Arbeitsschritte mit RMarkdown . . . . .	286



# 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 und 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 und 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 et al. 2015):

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

Es gibt bisher noch relativ wenig *deutschsprachige* Bücher für moderne Datenanalyse; es ist zu erwarten, dass sich dies bald ändert. Dieses Buch soll beitragen, diese Lücke zu füllen. Sie werden einige praktische Aspekte der modernen Datenanalyse lernen. Es ist ein Grundlagenkurs; dieses Buch ist nicht für Experten geschrieben und durch das Lesen wird man auch kein Experte. Aber man erwirbt Grundkenntnisse. Das didaktische Konzept beruht auf einem induktiven, intuitiven Lehr-Lern-Ansatz. Auf Formeln und mathematische Hintergründe wurde weitestgehend verzichtet.

Im Gegensatz zu anderen Statistik-Büchern steht hier die Umsetzung mit R deutlich stärker 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 zu entzünden. 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 and Pfannkuch 1999). Es ist die Hoffnung der Autoren dieses Buches, dass das praktische Arbeiten (im Gegensatz zu einer theoretischen Fokus) zur Entwicklung einer Kompetenz im statistischen Denken beiträgt. ↪

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

*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 pro Kapitel Übungsaufgaben oder/und Testfragen. Auf diese wird so  verwiesen.

Ansonsten: Wenn Ihnen R diesen Smiley präsentiert, dann sind Sie am Ziel Ihrer Träume: .

Dieses Skript wurde mit dem Paket `bookdown` (Xie 2015) erstellt, welches wiederum stark auf den Paketen `knitr` (Xie 2015) und `rmarkdown` (Allaire et al. 2016a) beruht. Diese Pakete stellen verblüffende Funktionalität zur Verfügung als freie Software (frei wie in Bier und frei wie in Freiheit).

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

Die Bildnachweise sind in folgenden Muster aufgebaut: Nummer (Verweis) des Bildes, Names des Autors, Titel, Quelle (URL), Lizenz, Abrufdatum.

- @ref(fig:modellieren\_plot), Sebastian Unrau, ohne Titel, <https://unsplash.com/photos/CoD2Q92UaEg>, CC0, 2017-02-12

Alle verwendeten Datensätze und R-Pakete finden sich im Literaturverzeichnis; im Text werden Pakete nicht zitiert.

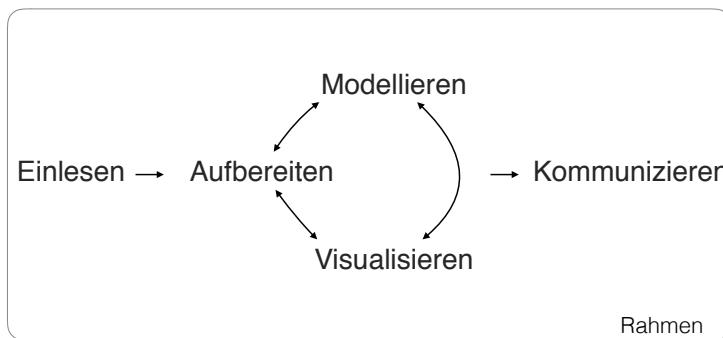


# I GRUNDLAGEN



# Rahmen

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. Voila:



Datenanalyse, praktisch betrachtet, kann man in fünf Schritte einteilen (Wickham and 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 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.

## 0.1 Software

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/>.

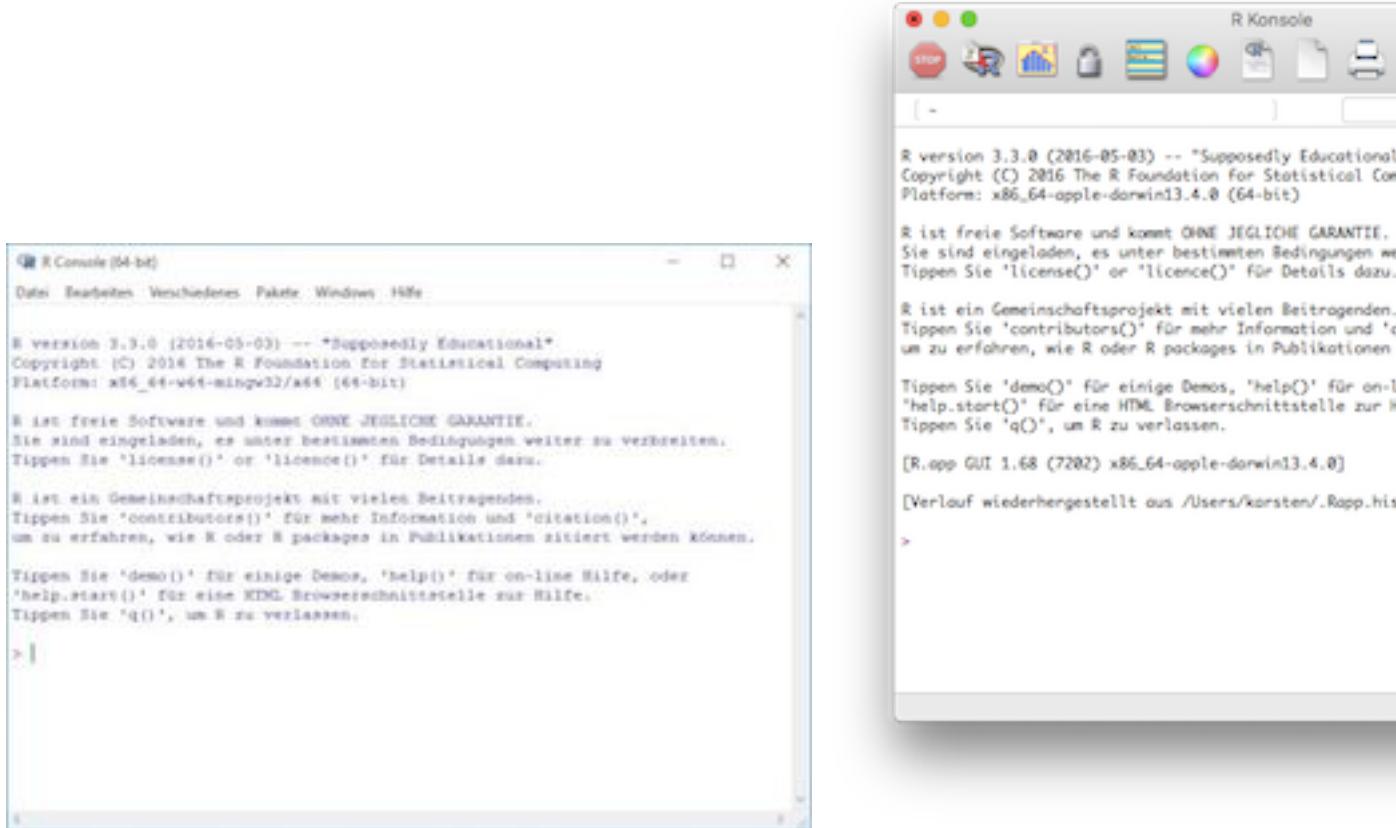
### 0.1.1 R und RStudio installieren

Setzt natürlich voraus, dass R installiert ist. 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.

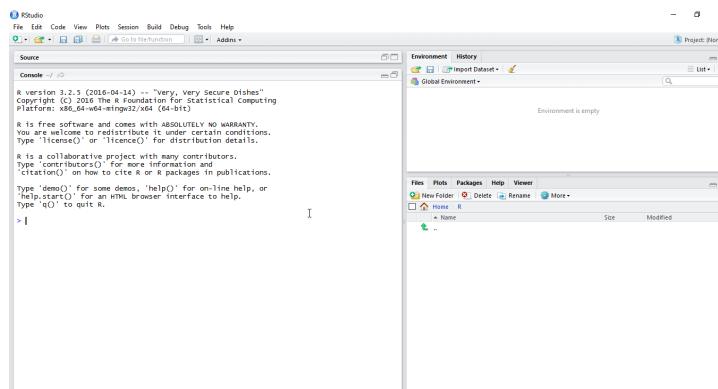
Die Oberfläche von R, die “Console”, sieht so aus:

## 0.1. SOFTWARE

7



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



### 0.1.2 Hilfe! R tut nicht so wie ich das will

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*<sup>1</sup>

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 schnottigen Tipps. Aber: Es passiert allzu leicht, dass man Fehler wie diese macht:

- `install.packages(dplyr)`
- `install.packages("dliar")`
- `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(paket_name)`.

---

<sup>1</sup>[https://de.wikipedia.org/wiki/Vom\\_Fischer\\_und\\_seiner\\_Frau](https://de.wikipedia.org/wiki/Vom_Fischer_und_seiner_Frau)

- Gerade bei Windows 10 scheinen die Schreibrechte für R (und damit RStudio oder RComannder) 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.
- Läuft der RCommander unter Mac nicht, dann prüfen Sie, ob Sie X11 (synonym: XQuartz) installiert haben. X11 muss installiert sein, damit der RCommander unter Mac läuft.
- Die “app nap” Funktion beim Mac kann den RCommander empfindlich ausbremsen. Schalten Sie diese Funktion aus z.B. im RCommander über Tools - Manage Mac OS X app nap for R.app.

### 0.1.3 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 Tastaturkürzlen (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.
- Die bekannteste Seite, um Fragen rund um R zu diskutieren ist: <http://stackoverflow.com>.

#### 0.1.4 Die Denk- und Gefühlswelt von R

- 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.
- Ein neues R-Skript im RStudio können Sie z.B. öffnen mit **File-New File-R Script**.
- R-Skripte können Sie speichern (**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.
- 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).
- Hier finden Sie weitere Hinweise zur Installation des RCommanders: <http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html>.
- Sie müssen online sein, um Packages zu installieren.
- Die “app nap” Funktion beim Mac kann den RCommander empfindlich ausbremsen. Schalten Sie diese Funktion aus z.B. im RCommander über **Tools - Manage Mac OS X app nap for R.app**.

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).

R zu lernen kann hart sein. Ich weiß, wovon ich spreche. Wahrscheinlich eine spirituelle Prüfung in Geduld und Hartnäckigkeit... Tolle Gelegenheit, sich in diesen Tugenden zu trainieren :-)

### 0.1.5 Pakete installieren

Ein R-Paket, welches für die praktische Datenanalyse praktisch ist, heißt `dplyr`. Wir werden viel mit diesem Paket arbeiten. Bitte installieren Sie es schon einmal, sofern noch nicht geschehen:

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

Übrigens, das `dependencies = TRUE` sagt sinngemäß "Wenn das Funktionieren von `dplyr` noch von anderen Paketen abhängig ist (es also Abhängigkeiten (dependencies) gibt), dann installiere die gleich mal mit".



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

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.

### 0.1.6 R-Pakete für dieses Buch In diesem Buch verwenden wir die folgenden

R-Pakete; diese müssen installiert[Ggf. benötigen Sie Administrator-Rechte, um Pakete zu installieren.] sein und geladen:

#### Pakete

```
#> [1] "tidyverse"      "readr"        "knitr"
#> [4] "stringr"       "car"          "nycflights13"
#> [7] "ISLR"          "pdfTools"     "downloader"
#> [10] "ggdendro"      "gridExtra"    "tm"
#> [13] "tidytext"      "lsa"          "SnowballC"
#> [16] "wordcloud"     "RColorBrewer" "okcupiddata"
#> [19] "reshape2"       "wesanderson" "GGally"
#> [22] "titanic"       "compute.es"   "corrr"
#> [25] "rpart"         "rpart.plot"  "MASS"
```

```
#> [28] "titanic"           "arules"          "arulesViz"
#> [31] "SDMTools"          "corrplot"        "gplots"
#> [34] "corrplot"          "scatterplot3d" "BaylorEdPsych"
#> [37] "nFactors"          "rmarkdown"       "methods"
```

Anstelle alle einzeln zu laden (`library` verdaut nur ein Paket auf einmal), können wir mit etwas R-Judo alle auf einen Haps laden:

```
lapply(Pakete, require, character.only = TRUE)
```

Der Befehl heißt auf Deutsch: “Wende auf jedes Element von `Pakete` den Befehl `library` an”<sup>2</sup>.

Hin und wieder ist es sinnvoll, die Pakete auf den neuesten Stand zu bringen; das geht mit `update.packages()`.

### 0.1.7 Datensätze

- Datensatz `profiles` aus dem R-Paket `{okcupiddata}` (Kim and Escobedo-Land 2015); es handelt sich um Daten von einer Online-Singlebörsen
- Datensatz `Wage` aus dem R-Paket `{ISLR}` (James, Witten, Hastie, and Tibshirani 2013b); es handelt sich um Gehaltsdaten von US-amerikanischen Männern
- Datensatz `inf_test_short`, hier herunterzuladen: <[osf.io/sjhu](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
- Datensatz `'wo_men'`, hier herunterzuladen: <[osf.io/ja9dw](https://osf.io/ja9dw/)> (Sauer 2017b); es handelt sich um Körper- und Schuhgröße von Studierenden
- Datensatz `tips` aus dem R-Paket `{reshape2}` (Bryant and Smith 1995); es handelt sich um Trinkgelder in einem Restaurant

---

<sup>2</sup><http://stackoverflow.com/questions/8175912/load-multiple-packages-at-once>

- Datensatz `extra`, hier herunterzuladen: <<https://osf.io/4kgzh/>> (Sauer 2016); es handelt sich die Ergebnisse einer Umfrage zu Extraversion

Wir verwenden zwei Methoden, um Datensätze in R zu laden.

- Zum einen laden wir Datensätze aus R-Paketen, z.B. aus dem Paket `okcupiddata`. Dazu muss das entsprechende Paket installiert und geladen sein. Mit dem Befehl `data(name_des_datensatzes, package = "name_des_paketes")`, kann man dann die Daten laden. Das Laden eines Pakets lädt noch *nicht* die Daten des Paketes; dafür ist der Befehl `data` zuständig.

```
library(okcupiddata) data(profiles, package = "okcupiddata")
```

- Alternativ kann man die Daten als CSV- oder als XLS(X)-Datei importieren. Die Datei darf dabei sowohl auf einer Webseite als auch lokal (Festplatte, Stick...) liegen.

```
Daten <-  
read.csv("https://sebastiansauer.github.io/data/tips.csv")
```

Wir werden mit beiden Methoden arbeiten und “on the job” Details besprechen.

## 0.2 ERFolgskontakt

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 R dann ignoriert.
- R wendet Befehle direkt an.
- R ist objektorientiert, d. h. dieselbe Funktion hat evtl. je nach Funktionsargument unterschiedliche Rückgabewerte.

- Hilfe zu einem Befehl erhält man über ein vorgestelltes Fragezeichen ?.
- Zusätzliche Funktionalität kann über Zusatzpakete hinzugeladen werden. Diese müssen ggf. zunächst installiert werden.
- Mit der Pfeiltaste “nach oben” können Sie in der Konsole einen vorherigen Befehl wieder aufrufen.
- Sofern Sie das Skriptfenster verwenden: einzelne Befehle aus dem Skriptfenster in R Studio können Sie auch mit **Str** und **Enter** an die Console schicken.

### 0.2.1 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.

### 0.3 Was ist Statistik? Wozu ist sie gut?

Zwei Fragen bieten sich 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) herausgemeiselt. Statistik wird häufig in die zwei Gebiete *deskriptive* und *inferierende* Statistik eingeteilt. 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.

Dabei lässt sich der Begriff “Statistik” als Überbegriff von “Datenanalyse” verstehen, wenn diese Sicht auch nicht von allen geteilt wird (Grolmund and 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 and Grolmund 2016; Hardin et al. 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; Grolemund and 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. @ref{mod1}.

Statistisches Modellieren läuft gewöhnlich nach folgendem Muster ab (Grolemund and Wickham 2014):

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

**Prämissse 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 Datnlage D (10 von 10 Treffern) wahrscheinlich - Prämissse 1. Datenlage D ist tatsächlich der Fall; der Statistiker hat 10 von 10 Treffer erzielt - Prämissse 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). Dass, worüber wir entscheiden wollen, wird also bereits vorausgesetzt. Gilt also M, 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 sein<sup>3</sup>. 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.

## 0.4 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/>

---

<sup>3</sup>Sind 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.

- 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.

## 0.5 Versionshinweise

```
sessionInfo()
#> R version 3.3.2 (2016-10-31)
#> Platform: x86_64-apple-darwin13.4.0 (64-bit)
#> Running under: macOS Sierra 10.12.3
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] grid       methods    stats      graphics   grDevices
#> [6] utils      datasets   base
#>
#> other attached packages:
#> [1] rmarkdown_1.3        nFactors_2.3.3
#> [3] lattice_0.20-34      boot_1.3-18
#> [5] psych_1.6.12         BaylorEdPsych_0.5
#> [7] scatterplot3d_0.3-38 gplots_3.0.1
#> [9] corrplot_0.77         SDMTools_1.1-221
#> [11] arulesViz_1.2-0      arules_1.5-0
#> [13] Matrix_1.2-8        MASS_7.3-45
#> [15] rpart.plot_2.1.1     rpart_4.1-10
#> [17] corrr_0.2.1          compute.es_0.2-4
#> [19] titanic_0.1.0         GGally_1.3.0
#> [21] wesanderson_0.3.2     reshape2_1.4.2
#> [23] okcupiddata_0.1.0     wordcloud_2.5
#> [25] RColorBrewer_1.1-2     lsa_0.73.1
```

```

#> [27] SnowballC_0.5.1      tidytext_0.1.2
#> [29] tm_0.6-2            NLP_0.1-10
#> [31] gridExtra_2.2.1     ggdendro_0.1-20
#> [33] downloader_0.4       pdftools_1.0
#> [35] ISLR_1.0             nycflights13_0.2.2
#> [37] car_2.1-4            stringr_1.2.0
#> [39] knitr_1.15.1         dplyr_0.5.0
#> [41] purrr_0.2.2.9000    readr_1.0.0
#> [43] tidyr_0.6.1          tibble_1.2
#> [45] ggplot2_2.2.1        tidyverse_1.1.1
#>
#> loaded via a namespace (and not attached):
#> [1] readxl_0.1.1           backports_1.0.5
#> [3] plyr_1.8.4              lazyeval_0.2.0.9000
#> [5] splines_3.3.2           digest_0.6.12
#> [7] foreach_1.4.3            htmltools_0.3.5
#> [9] viridis_0.3.4           gdata_2.17.0
#> [11] magrittr_1.5            cluster_2.0.5
#> [13] gclus_1.3.1            modelr_0.1.0
#> [15] R.utils_2.5.0           colorspace_1.3-2
#> [17] rvest_0.3.2             haven_1.0.0
#> [19] jsonlite_1.3            lme4_1.1-12
#> [21] zoo_1.7-14              iterators_1.0.8
#> [23] registry_0.3            gtable_0.2.0
#> [25] MatrixModels_0.4-1     kernlab_0.9-25
#> [27] prabclus_2.2-6          DEoptimR_1.0-8
#> [29] SparseM_1.74            scales_0.4.1
#> [31] mvtnorm_1.0-5           DBI_0.5-1
#> [33] Rcpp_0.12.9              viridisLite_0.1.3
#> [35] foreign_0.8-67           mclust_5.2.2
#> [37] stats4_3.3.2            DT_0.2
#> [39] vcd_1.4-3                htmlwidgets_0.8
#> [41] httr_1.2.1                fpc_2.1-10
#> [43] modeltools_0.2-21        reshape_0.8.6
#> [45] R.methodsS3_1.7.1        flexmix_2.3-13
#> [47] nnet_7.3-12              munsell_0.4.3

```

```
#> [49] tools_3.3.2           broom_0.4.2
#> [51] evaluate_0.10         yaml_2.1.14
#> [53] robustbase_0.92-7   caTools_1.17.1
#> [55] dendextend_1.4.0     nlme_3.1-130
#> [57] whisker_0.3-2       quantreg_5.29
#> [59] slam_0.1-40          R.oo_1.21.0
#> [61] xml2_1.1.1           tokenizers_0.1.4
#> [63] pbkrtest_0.4-6       plotly_4.5.6
#> [65] stringi_1.1.2      forcats_0.2.0
#> [67] trimcluster_0.1-2   nloptr_1.0.4
#> [69] lmtest_0.9-35        bitops_1.0-6
#> [71] seriation_1.2-1     R6_2.2.0
#> [73] bookdown_0.3         TSP_1.1-5
#> [75] KernSmooth_2.23-15  janeaustenr_0.1.4
#> [77] codetools_0.2-15    gtools_3.5.0
#> [79] assertthat_0.1        rprojroot_1.2
#> [81] mnormt_1.5-5          diptest_0.75-7
#> [83] mgcv_1.8-16          parallel_3.3.2
#> [85] hms_0.3                class_7.3-14
#> [87] minqa_1.2.4          lubridate_1.6.0
#> [89] base64enc_0.1-3
```



# Tidy Data - Daten sauber einlesen

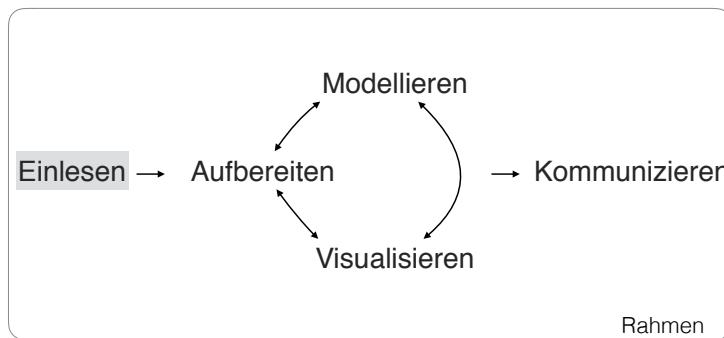


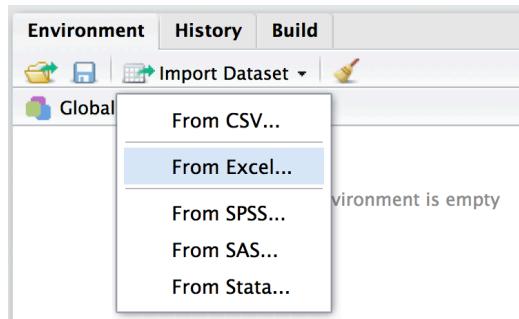
Figure 1: Daten sauber einlesen

## 0.6 Daten in R importieren

In R kann man ohne Weiteres verschiedene, gebräuchliche (Excel) oder weniger gebräuchliche (Feather<sup>4</sup>) Datenformate einlesen. In RStudio lässt sich dies z.B. durch einen schnellen Klick auf **Import Dataset** im Reiter **Environment** erledigen. Dabei wird im Hintergrund das Paket **readr** verwendet (Wickham, Hester, and Francois 2016a) (die entsprechende Syntax wird in der Konsole ausgegeben, so dass man sie sich anschauen und weiterverwenden kann).

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

Am einfachsten ist es, eine Excel-Datei über die RStudio-Oberfläche zu importieren; das ist mit ein paar Klicks geschehen:



**Figure 2:** Daten einlesen (importieren) mit RStudio

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

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.

```
"ID","time","sex","height","shoe_size"
"1","04.10.2016 17:58:51",NA,160.1,40
"2","04.10.2016 17:58:59","woman",171.2,39
"3","04.10.2016 18:00:15","woman",174.2,39
"4","04.10.2016 18:01:17","woman",176.4,40
"5","04.10.2016 18:01:22","man",195.2,46
```

Erkennen Sie das Muster? Die erste Zeile gibt die “Spaltenköpfe” wieder, also die Namen der Variablen. Hier sind es 5 Spalten; die vierte heißt “shoe\_size”. 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. “ID” (Zelle oben links) oder “46” (Zelle unten rechts) steht.

An einer 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:

```
if (!file.exists("./data/wo_men.csv")){
  daten <- read.csv("https://sebastiansauer.github.io/data/wo_men.csv")
} else {
  daten <- read.csv("./data/wo_men.csv")
}
head(daten)
#>   X           time   sex height shoe_size
#> 1 1 04.10.2016 17:58:51 woman    160      40
#> 2 2 04.10.2016 17:58:59 woman    171      39
#> 3 3 04.10.2016 18:00:15 woman    174      39
#> 4 4 04.10.2016 18:01:17 woman    176      40
#> 5 5 04.10.2016 18:01:22 man     195      46
#> 6 6 04.10.2016 18:01:53 woman    157      37
```

Wir haben zuerst geprüft, ob die Datei (`wo_men.csv`) im entsprechenden Ordner existiert oder nicht (das `!`-Zeichen heißt auf Errisch “nicht”). Falls die Datei nicht im Ordner existiert, laden wir sie mit `read.csv` herunter und direkt ins R hinein. Andernfalls (`else`) lesen wir sie direkt ins R hinein.

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

```
# 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")`.

Übrigens: Wenn Sie keinen Pfad angeben, so geht R davon aus, dass die Daten im aktuellen Verzeichnis liegen. Das aktuelle Verzeichnis kann man mit `getwd()` erfragen und mit `setwd()` einstellen. Komfortabler ist es aber, das aktuelle Verzeichnis per Menü zu ändern. In RStudio: **Session > Set Working Directory > Choose Directory ...** (oder per Shortcut, der dort angezeigt wird).

## 0.7 Normalform einer Tabelle

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

- Es handelt sich um einen data frame, also Spalten mit Namen und gleicher Länge; eine Datentabelle in rechteckiger Form
- In jeder Zeile steht eine Beobachtung, in jeder Spalte eine Variable
- Fehlende Werte sollten sich in *leeren* Tabellen niederschlagen
- Daten sollten nicht mit Farbkmarkierungen o.ä. kodiert werden
- keine Leerzeilen, keine Leerspalten
- am besten keine Sonderzeichen verwenden und keine Leerzeichen in Variablenamen und -werten, am besten nur Ziffern und Buchstaben und Unterstriche
- Variablenamen dürfen nicht mit einer Zahl beginnen

Der Punkt “Jede Zeile eine Beobachtung, jede Spalte eine Variable” verdient besondere Beachtung. Betrachten Sie dieses Beispiel:

```
knitr::include_graphics("./images/breit_lang.pdf")
```

Breit

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

Lang

<i>ID</i>	<i>Quartal</i>	<i>Umsatz</i>
1	Q1	342
2	Q2	342
3	...	124
...	Q1	342
	Q2	342
	Q3	124
	...	

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:  
Merkmal/ Variable

Wert/ Ausprägung

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

**Figure 3:** Illustration eines Datensatzes in Normalform

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.

Ein Beispiel dazu:

```
meindf <- read.csv("http://stanford.edu/~ejdemyr/r-tutorials/data/unicef-u5mr.csv")

df_lang <- gather(meindf, year, u5mr, U5MR.1950:U5MR.2015)

df_lang <- separate(df_lang, year, into = c("U5MR", "year"), sep = ".")
```

- Die erste Zeile liest die Daten aus einer CSV-Datei ein; praktischerweise direkt von einer Webseite.
- Die zweite Zeile formt die Daten von breit nach lang um. Die neuen Spalten, nach der Umformung heißen dann `year` und `u5mr` (Sterblichkeit bei Kindern unter fünf Jahren). In die Umformung werden die Spalten `U5MR 1950` bis `U5MR 2015` einbezogen.
- Die dritte Zeile “entzerrt” die Werte der Spalte `year`; hier stehen die ehemaligen Spaltenköpfe. Man nennt sie auch `key` Spalte daher. Steht in einer Zelle von `year` bspw. `U5MR 1950`, so wird `U5MR` in eine Spalte mit Namen `U5MR` und `1950` in eine Spalte mit Namen `year` geschrieben.

## 0.8 Verweise

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

# Daten aufbereiten

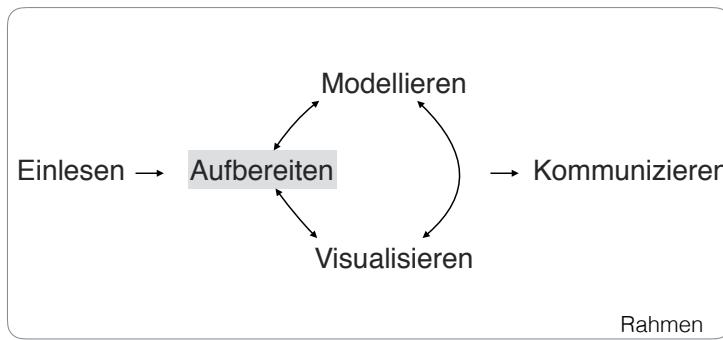


Figure 4: Daten aufbereiten

In diesem Kapitel benötigte Pakete:

```
library(tidyverse) # Datenjudo
library(corr) # Korrelationen berechnen mit der Pfeife
library(stringr) # Texte bearbeiten
library(car) # für 'recode'
library(nycflights13) # Datensatz 'flights'
library(knitr) # für HTML-Tabellen
library(gridExtra) # für Mehrfachplots
```

Unter Daten aufbereiten im engeren Sinne ist gemeint, die Daten einer “Grunderinigung” zu unterziehen, dass sie für weitere Analysen in geeigneter Form sind. Daten zusammenfassen meint die deskriptive Statistik; Daten visualisieren ist das Erstellen von Diagrammen. Im Anschluss kann man die Daten modellieren.

Ist das Explorieren von Daten auch nicht statistisch anspruchsvoll, so ist es trotzdem von großer Bedeutung und häufig recht zeitintensiv, vor allem das Daten aufbereiten. Eine Anekdote zur Relevanz der Exploration, 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. 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.

## 0.9 Typische Probleme

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.

Mit “Datenjudo” (ein Fachbegriff aus der östlichen Zahlentheorie) ist

gemeint, die Daten so “umzuformen”, “aufzubereiten”, oder “reinigen” , dass sie passend für statistische Analysen sind.

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 Spalten berechnen: Ein Student fragt nach der Anzahl der richtigen Aufgaben in der Statistik-Probeklausur. Wir wollen helfen und im entsprechenden Datensatz eine Spalte erzeugen, in der pro Person die Anzahl der richtig beantworteten Fragen steht.

## 0.10 Daten aufbereiten mit dplyr

Es gibt viele Möglichkeiten, Daten mit R aufzubereiten; `dplyr` ist ein populäres Paket dafür. Eine zentrale Idee von `dplyr` ist, dass es nur ein paar wenige Grundbausteine geben sollte, die sich gut kombinieren lassen. Sprich: Wenige grundlegende Funktionen mit eng umgrenzter Funktionalität. Der Autor, Hadley Wickham, sprach einmal in einem Forum (citation needed), dass diese Befehle wenig können, das Wenige aber gut. Ein Nachteil dieser Konzeption kann sein, dass man recht viele dieser Bausteine kombinieren muss, um zum gewünschten Ergebnis zu kommen. Außerdem muss man die Logik des Baukastens gut verstanden haben - die Lernkurve ist also erstmal steiler. Dafür ist man dann nicht darauf angewiesen, dass es irgendwo “Mrs Right” gibt, die genau das kann, so wie ich das 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).

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 Be-

fehlen).

Diese Bausteine sind typische Tätigkeiten im Umgang mit Daten; nichts Überraschendes. Schauen wir uns diese Bausteine näher an.

### 0.10.1 Zeilen filtern mit `filter`

Häufig will man bestimmte Zeilen aus einer Tabelle filtern. 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.

Ein Sinnbild:

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

**Figure 5:** Zeilen filtern

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
df_alte_frauen <- filter(profiles, age > 70, sex == "f") # nur die alten Frauen, d.
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 Allerweltssymbolnamen; 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(...)`.

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))
```

### 0.10.2 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. Das Sinnbild für diesen Befehl:

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
...	...	...	...	...	...	...	...

Figure 6: Spalten auswählen

Merke:

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

```
if (!file.exists("./data/test_inf_short.csv")) {
  stats_test <- read.csv("https://sebastiansauer.github.io/data/test_inf_short.csv")
```

```

} else {
  stats_test <- read.csv("./data/test_inf_short.csv")
}

```

Hier haben wir erst geprüft, ob die Datei `test_inf_short.csv` existiert; falls nein, laden wir sie herunter. Andernfalls lesen wir sie aus dem lokalen Verzeichnis.

```

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

```

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

### 0.10.3 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 vorraussehen und sicherstellen, dass das Programm auch beim merkwürdigsten Nutzer brav seinen Dienst tut. Wir haben hier, beim interaktiven Gebrauch, niedrigere Ansprüche bzw. andere Ziele.

Beim interaktiven Gebrauch von R (oder beliebigen Analyseprogrammen) ist das Sortieren von Zeilen eine recht häufige Tätigkeit. Typisches Beispiel wäre der Lehrer, der eine Tabelle mit Noten hat und wissen will, welche Schüler die

schlechtesten oder die besten sind in einem bestimmten Fach. Oder bei der Prüfung der Umsätze nach Filialen möchten wir die umsatzstärksten sowie -schwächsten Niederlassungen kennen.

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

```
arrange(stats_test, score) %>% head() # liefert die *schlechtesten* Noten zurück
#>   X           V_1 study_time self_eval interest score
#> 1 234 23.01.2017 18:13:15      3       1       1     17
#> 2  4 06.01.2017 09:58:05      2       3       2     18
#> 3 131 19.01.2017 18:03:45      2       3       4     18
#> 4 142 19.01.2017 19:02:12      3       4       1     18
#> 5  35 12.01.2017 19:04:43      1       2       3     19
#> 6  71 15.01.2017 15:03:29      3       3       3     20
arrange(stats_test, -score) %>% head() # liefert die *besten* Noten zurück
#>   X           V_1 study_time self_eval interest score
#> 1  3 05.01.2017 23:33:47      5      10       6     40
#> 2  7 06.01.2017 14:25:49     NA      NA      NA     40
#> 3 29 12.01.2017 09:48:16      4      10       3     40
#> 4 41 13.01.2017 12:07:29      4      10       3     40
#> 5 58 14.01.2017 15:43:01      3       8       2     40
#> 6 83 16.01.2017 10:16:52     NA      NA      NA     40
arrange(stats_test, interest, score) %>% head()
#>   X           V_1 study_time self_eval interest score
#> 1 234 23.01.2017 18:13:15      3       1       1     17
#> 2 142 19.01.2017 19:02:12      3       4       1     18
#> 3 221 23.01.2017 11:40:30      1       1       1     23
#> 4 230 23.01.2017 16:27:49      1       1       1     23
#> 5  92 17.01.2017 17:18:55      1       1       1     24
#> 6 107 18.01.2017 16:01:36      3       2       1     24
```

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. Am wichtigsten ist hier, dass man weitere Spalten übergeben kann. Dazu gleich mehr.

Standardmäßig sortiert `arrange` *aufsteigend* (weil kleine Zahlen im Zahlenstrahl vor den großen Zahlen kommen). Möchte man diese Reihenfolge umdrehen (große Werte zuerst), 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.

Aber was heißt dieses komisch Symbol: `%>%?` Diese sogenannte “Pfeife” lässt sich mit “und dann” ins Deutschce übersetzen. Also:

```
sortiere(diese_Tabelle, nach_dieser_Spalte) UND DANN zeig_die_ersten_Zeilen
```

Der Befehl `head` zeigt die ersten paar Zeilen eines Dataframes.<sup>5</sup>

Merke:

Die Funktion `arrange` sortiert die Zeilen eines Datafames.

Ein Sinnbild zur Verdeutlichung:

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

**Figure 7:** Spalten sortieren

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

```
top_n(stats_test, 3)
```

<sup>5</sup><https://github.com/karthik/wesanderson>

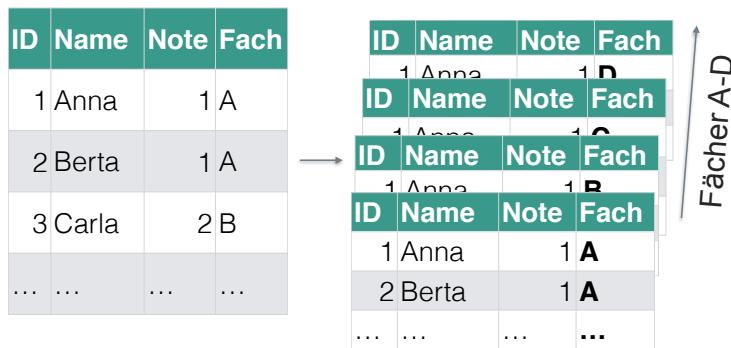
```
#>      X          V_1 study_time self_eval interest score
#> 1 3 05.01.2017 23:33:47      5       10      6     40
#> 2 7 06.01.2017 14:25:49     NA      NA     NA     40
#> 3 29 12.01.2017 09:48:16     4       10      3     40
#> 4 41 13.01.2017 12:07:29     4       10      3     40
#> 5 58 14.01.2017 15:43:01     3       8       2     40
#> 6 83 16.01.2017 10:16:52     NA      NA     NA     40
#> 7 116 18.01.2017 23:07:32     4       8       5     40
#> 8 119 19.01.2017 09:05:01     NA      NA     NA     40
#> 9 132 19.01.2017 18:22:32     NA      NA     NA     40
#> 10 175 20.01.2017 23:03:36     5       10      5     40
#> 11 179 21.01.2017 07:40:05     5       9       1     40
#> 12 185 21.01.2017 15:01:26     4       10      5     40
#> 13 196 22.01.2017 13:38:56     4       10      5     40
#> 14 197 22.01.2017 14:55:17     4       10      5     40
#> 15 248 24.01.2017 16:29:45     2       10      2     40
#> 16 249 24.01.2017 17:19:54     NA      NA     NA     40
#> 17 257 25.01.2017 10:44:34     2       9       3     40
#> 18 306 27.01.2017 11:29:48     4       9       3     40
top_n(stats_test, 3, interest)
#>      X          V_1 study_time self_eval interest score
#> 1 3 05.01.2017 23:33:47      5       10      6     40
#> 2 5 06.01.2017 14:13:08      4       8       6     34
#> 3 43 13.01.2017 14:14:16     4       8       6     36
#> 4 65 15.01.2017 12:41:27     3       6       6     22
#> 5 110 18.01.2017 18:53:02     5       8       6     37
#> 6 136 19.01.2017 18:22:57     3       1       6     39
#> 7 172 20.01.2017 20:42:46     5       10      6     34
#> 8 214 22.01.2017 21:57:36     2       6       6     31
#> 9 301 27.01.2017 08:17:59     4       8       6     33
```

Gibt man *keine* Spalte an, so bezieht sich `top_n` auf die letzte Spalte im Datensatz.

Da sich hier mehrere Personen den größten Rang (Wert 40) teilen, bekommen wir *nicht* 3 Zeilen zurückgeliefert, sondern entsprechend mehr.

### 0.10.4 Datensatz gruppieren mit `group_by`

Einen Datensatz zu gruppieren ist ebenfalls 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.



**Figure 8:** Datensätze nach Subgruppen aufteilen

In der Abbildung wurde der Datensatz anhand der Spalte `Fach` in mehrere Gruppen geteilt. 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 6]
#> Groups: interest [7]
#>
#>       X           V_1 study_time self_eval interest score
#>   <int>      <fctr>     <int>     <int>     <int> <int>
#> 1    1 05.01.2017 13:57:01      5        8      5    29
#> 2    2 05.01.2017 21:07:56      3        7      3    29
#> 3    3 05.01.2017 23:33:47      5       10      6    40
#> # ... with 303 more rows
```

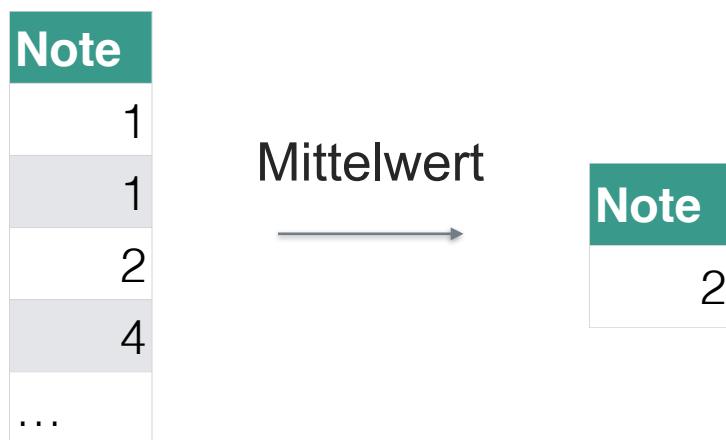
Schaut man sich nun den Datensatz an, sieht man erstmal wenig Effekt der Gruppierung. R teilt uns lediglich mit **Groups: interest [7]**, dass es die Gruppen gibt, aber es gibt keine extra Spalte oder sonstige Anzeichen der Gruppierung. Aber keine Sorge, wenn wir gleich einen Mittelwert ausrechnen, bekommen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

Merke:

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

### 0.10.5 Eine Spalte zusammenfassen mit `summarise`

Vielleicht die wichtigste oder häufigste Tätigkeit in der Analyse von Daten ist es, eine Spalte zu *einem* Wert zusammenzufassen. Anders gesagt: Einen Mittelwert berechnen, den größten (kleinsten) Wert heraussuchen, die Korrelation berechnen oder eine beliebige andere Statistik ausgeben lassen. Die Gemeinsamkeit dieser Operatoren 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`. Hierbei ist jede Funktion erlaubt, die eine Spalte als Input verlangt und eine Zahl zurückgibt; andere Funktionen sind bei `summarise` nicht erlaubt.



**Figure 9:** Spalten zu einer Zahl zusammenfassen

```
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.

Jetzt können wir auch die Gruppierung nutzen:

```
test_gruppiert <- group_by(stats_test, interest)
summarise(test_gruppiert, mean(score))
#> # A tibble: 7 × 2
#>   interest `mean(score)`
#>   <int>     <dbl>
#> 1 1          28.3
#> 2 2          29.7
#> 3 3          30.8
#> # ... with 4 more rows
```

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 × 2
```

```
#>   interest mw_pro_gruppe
#>   <int>      <dbl>
#> 1       1      28.3
#> 2       2      29.7
#> 3       3      30.8
#> # ... with 4 more rows
```

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.

### 0.10.6 Zeilen zählen mit `n` und `count`

Ebenfalls nützlich ist es, Zeilen zu zählen. Im Gegensatz zum Standardbefehle `nrow` versteht der `dplyr`-Befehl `n` auch Gruppierungen. `n` darf nur innerhalb von `summarise` oder ähnlichen `dplyr`-Befehlen verwendet werden.

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

```
#> # ... with 4 more rows
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 × 2
#>   interest     n
#>   <int> <int>
#> 1      1    30
#> 2      2    47
#> 3      3    66
#> # ... with 4 more rows
dplyr::count(stats_test, study_time)
#> # A tibble: 6 × 2
#>   study_time     n
#>   <int> <int>
#> 1      1    31
#> 2      2    49
#> 3      3    85
#> 4      4    56
#> 5      5    17
#> 6     NA    68
dplyr::count(stats_test, interest, study_time)
#> Source: local data frame [29 x 3]
#> Groups: interest [?]
#>
#>   interest study_time     n
#>   <int>       <int> <int>
```

```
#> 1      1      1    12
#> 2      1      2     7
#> 3      1      3     8
#> # ... with 26 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.

Merke:

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

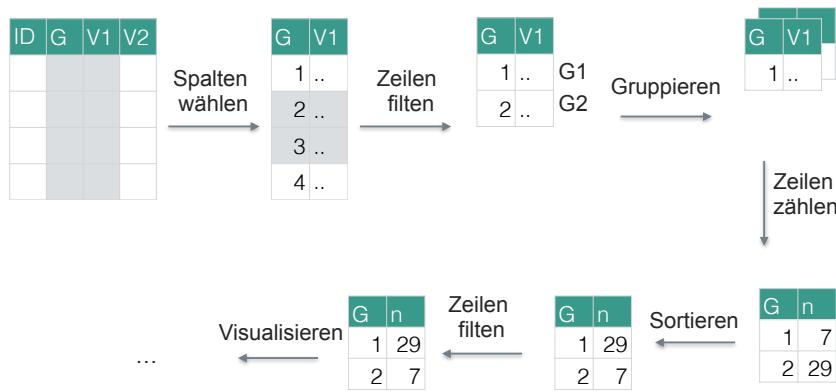
### 0.10.7 Die Pfeife

Die zweite Idee kann man salopp als “Durchpfeifen” bezeichnen; ikonographisch mit diesem Symbol dargestellt `%>%`<sup>6</sup>. Der Begriff “Durchpfeifen” ist frei vom Englischen “to pipe” übernommen. 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.

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

---

<sup>6</sup><http://sape.inf.usi.ch/quick-reference/ggplot2/colour>

**Figure 10:** Das 'Durchpeifen'

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

mit dieser

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

Es ist hilfreich, diese ‘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 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!

## 0.10.8 Werte umkodieren und “binnen”

### 0.10.8.1 car::recode

Manchmal möchte man z.B. negativ gepolte Items umdrehen oder bei kategorialen Variablen kryptische Bezeichnungen in sprechendere umwandeln (ein Klassiker ist 1 in `maennlich` bzw. 2 in `weiblich` oder umgekehrt, kann sich niemand merken). Hier gibt es eine Reihe praktischer Befehle, z.B. `recode` aus dem Paket `car`. Übrigens: Wenn man explizit angeben möchte, aus welchem Paket ein Befehl stammt (z.B. um Verwechslungen zu vermeiden), gibt man `Paketnamen::Befehlnamen` an. Schauen wir uns ein paar Beispiele zum Umkodieren an.

```
stats_test$score_fac <- car::recode(stats_test$study_time, "5 = 'sehr viel'; 2 = 'viel'; 1 = 'wenig'; 0 = 'sehr wenig'")  
stats_test$score_fac <- car::recode(stats_test$study_time, "5 = 'sehr viel'; 2 = 'viel'; 1 = 'wenig'; 0 = 'sehr wenig'")  
  
stats_test$study_time <- car::recode(stats_test$study_time, "5 = 'sehr viel'; 2 = 'viel'; 1 = 'wenig'; 0 = 'sehr wenig'")
```

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

Der Befehle `recode` ist wirklich sehr 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 =
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 × 2
#>   interest     n
#>   <int> <int>
#> 1      1    30
#> 2      2    47
#> 3      3    66
#> # ... with 4 more rows
dplyr::count(stats_test, no_interest)
#> # A tibble: 7 × 2
#>   no_interest     n
#>   <dbl> <int>
#> 1          1     9
```

```
#> 2      2    45
#> 3      3    41
#> # ... with 4 more rows
```

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';
```

Natürlich gibt es auch eine Pfeifen komptatile Version, um Variablen umzukodieren bzw. zu binnnen: `dplyr::recode`<sup>7</sup>. Die Syntax ist allerdings etwas weniger komfortabel (da strenger), so dass wir an dieser Stelle bei `car::recode` bleiben.

### 0.10.8.2 Numerische Werte in Klassen gruppieren 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 gruppiieren (gleichgroße Intervalle)
2. so in Klassen gruppiieren, dass in jeder Klasse  $n$  Beobachtungen sind (gleiche Gruppengrößen)
3. in beliebige Klassen gruppiieren

#### 0.10.8.2.1 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 (Interval) der drei Gruppen soll gleich groß sein. Dazu kann man

---

<sup>7</sup><https://blog.rstudio.org/2016/06/27/dplyr-0-5-0/>.

`cut_interval` aus `ggplot2` nehmen [^d.h. `ggplot2` muss geladen sein; wenn man `tidyverse` lädt, wird `ggplot2` automatisch auch geladen].

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

wo_men %>%
  filter(height > 150, height < 220) -> wo_men2

temp <- cut_interval(x = wo_men2$height, n = 3)

levels(temp)
#> [1] "[155,172]" "(172,189]" "(189,206]"
```

`cut_interval` liefert eine Variabel vom Typ `factor` zurück.

### 0.10.8.2.2 gleiche Gruppengrößen

```
temp <- cut_number(wo_men2$height, n = 2)
str(temp)
#> Factor w/ 2 levels "[155,169]", "(169,206]": 1 2 2 2 2 1 1 2 1 2 ...
```

Mit `cut_number` (aus `ggplot2`) kann man einen Vektor in `n` Gruppen mit (etwa) gleich viel Observationen einteilen.

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

#### 0.10.8.2.2.1 In beliebige Klassen gruppieren

```
wo_men$groesse_gruppe <- cut(wo_men$height,
                                breaks = c(-Inf, 100, 150, 170, 200, 230, Inf))

count(wo_men, groesse_gruppe)
#> # A tibble: 6 × 2
#>   groesse_gruppe     n
#>   <fct>        <dbl>
#> 1 (-Inf,100]      1.00
#> 2 (100,150]      1.00
#> 3 (150,170]      1.00
#> 4 (170,200]      1.00
#> 5 (200,230]      1.00
#> 6 (230,Inf]      1.00
```

```
#>          <fctr> <int>
#> 1      (-Inf,100]     4
#> 2      (150,170]    55
#> 3      (170,200]    38
#> 4      (200,230]     2
#> 5      (230, Inf]     1
#> 6          NA     1
```

`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 wird nicht automatisch als unterste Intervallgrenze herangezogen.

## 0.11 Fallstudie `nycflights13`

Schauen wir uns einige Beispiele der Datenaufbereitung mittels `dplyr` an. Wir verwenden hier den Datensatz `flights` aus dem Package `nycflights13`. Der Datensatz ist recht groß (~300.000 Zeilen und 19 Spalten); wenn man ihn als Excel importiert, kann eine alte Möhre von Computer schon in die Knie gehen. Beim Import als CSV habe ich noch nie von Problemen gehört; beim Import via Package ebenfalls nicht. Werfen wir einen ersten Blick in die Daten:

```
data(flights)
glimpse(flights)
#> Observations: 336,776
#> Variables: 19
#> $ year                  <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, ...
#> $ month                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ day                    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ dep_time               <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 55...
#> $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 60...
#> $ dep_delay              <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2...
#> $ arr_time               <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 7...
```

```
#> $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 7...
#> $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -...
#> $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV",...
#> $ flight          <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79...
#> $ tailnum         <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN...
#> $ origin          <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR"...
#> $ dest            <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL"...
#> $ air_time        <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138...
#> $ distance        <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 94...
#> $ hour            <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, ...
#> $ minute          <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ time_hour       <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013...
```

Der Befehl `data` lädt Daten aus einem zuvor gestarteten Paket.

Achtung, Fallstudie. Sie sind der/die Assistent\_in des Chefs der New Yorker Flughäfen. Ihr Chef kommt gut gelaunt ins Büro und sagt, dass er diesen Schnarchnasen einheizen wolle und sagt, sie sollen ihm mal schnell die Flüge mit der größten Verspätung raussuchen. Nix schickes, aber zacki-zacki...

```
flights %>%
  arrange(arr_delay)
#> # A tibble: 336,776 × 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>    <int>           <int>     <dbl>    <int>
#> 1  2013     5     7     1715             1729      -14     1944
#> 2  2013     5    20      719              735      -16     951
#> 3  2013     5     2     1947             1949      -2     2209
#> # ... with 3.368e+05 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
#> #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Hm, übersichtlicher wäre es wahrscheinlich, wenn wir weniger Spalten anschauen müssten. Am besten neben der Verspätung nur die Information, die wir zur Identifizierung der Schuldigen... will sagen der gesuchten Flüge

benötigen

```
flights %>%
  arrange(arr_delay) %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest)
#> # A tibble: 336,776 × 8
#>   arr_delay carrier month   day dep_time tailnum flight  dest
#>   <dbl>     <chr>  <int> <int>    <int>   <chr>   <int> <chr>
#> 1     -86      VX      5     7     1715  N843VA     193  SFO
#> 2     -79      VX      5    20      719  N840VA      11  SFO
#> 3     -75      UA      5     2     1947  N851UA     612  LAX
#> # ... with 3.368e+05 more rows
```

Da Zahlen in ihrer natürlichen Form von klein nach groß sortiert sind, sortiert `arrange` in ebendieser Richtung. Wir können das umdrehen mit einem Minuszeichen vor der zu sortierenden Spalte:

```
flights %>%
  arrange(-arr_delay) %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest)
#> # A tibble: 336,776 × 8
#>   arr_delay carrier month   day dep_time tailnum flight  dest
#>   <dbl>     <chr>  <int> <int>    <int>   <chr>   <int> <chr>
#> 1     1272     HA      1     9      641  N384HA      51  HNL
#> 2     1127     MQ      6    15     1432  N504MQ     3535  CMH
#> 3     1109     MQ      1    10     1121  N517MQ     3695  ORD
#> # ... with 3.368e+05 more rows
```

Eine kleine Zugabe: Mit dem Befehl `knitr::kable` kann man einen Dataframe automatisch in eine (einigermaßen) schöne Tabelle ausgeben lassen. Oh halt, wir wollen keine Tabelle mit 300.000 Zeilen (der Chef ist kein Freund von Details). Also begrenzen wir die Ausgabe auf die ersten 10 Plätze.

```
flights %>%
  arrange(-arr_delay) %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest) %>%
```

```
filter(row_number() < 11) %>%
kable()
```

arr_delay	carrier	month	day	dep_time	tailnum	flight	dest
1272	HA	1	9	641	N384HA	51	HNL
1127	MQ	6	15	1432	N504MQ	3535	CMH
1109	MQ	1	10	1121	N517MQ	3695	ORD
1007	AA	9	20	1139	N338AA	177	SFO
989	MQ	7	22	845	N665MQ	3075	CVG
931	DL	4	10	1100	N959DL	2391	TPA
915	DL	3	17	2321	N927DA	2119	MSP
895	DL	7	22	2257	N6716C	2047	ATL
878	AA	12	5	756	N5DMAA	172	MIA
875	MQ	5	3	1133	N523MQ	3744	ORD

“Geht doch”, war die Antwort des Chefs, als sie die Tabelle rübergeben (er mag auch keine Emails). “Ach ja”, raunt der Chef, als Sie das Zimmer verlassen wollen, “hatte ich erwähnt, dass ich die gleiche Auswertung für jeden Carrier brauche? Reicht bis in einer halben Stunde”.

Wir gruppieren also den Datensatz nach der Fluggesellschaft (`carrier`) und filtern dann die ersten 3 Zeilen (damit die Tabelle für den Chef nicht zu groß wird). Wie jeder `dplyr`-Befehl wird die vorherige Gruppierung berücksichtigt und daher die Top-3-Zeilen *pro Gruppe*, d.h. pro Fluggesellschaft, ausgegeben.

```
flights %>%
  arrange(-arr_delay) %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest) %>%
  group_by(carrier) %>%
  filter(row_number() < 4)
#> Source: local data frame [48 x 8]
#> Groups: carrier [16]
#>
#>   arr_delay carrier month   day dep_time tailnum flight  dest
#>       <dbl>   <chr> <int> <int>    <int>   <chr> <int> <chr>
#> 1     1272      HA     1     9      641  N384HA      51   HNL
```

```
#> 2      1127      MQ      6     15      1432  N504MQ    3535   CMH
#> 3      1109      MQ      1     10      1121  N517MQ    3695   ORD
#> # ... with 45 more rows
```

Vielleicht gefällt dem Chef diese Darstellung (sortiert nach `carrier`) besser:

```
flights %>%
  arrange(-arr_delay) %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest) %>%
  group_by(carrier) %>%
  filter(row_number() < 4) %>%
  arrange(carrier)
#> Source: local data frame [48 x 8]
#> Groups: carrier [16]
#>
#>   arr_delay carrier month   day dep_time tailnum flight  dest
#>   <dbl>     <chr> <int> <int>   <int>   <chr>   <int> <chr>
#> 1     744     9E     2     16      757  N8940E    3798   CLT
#> 2     458     9E     7     24     1525  N927XJ    3538   MSP
#> 3     421     9E     7     10     2054  N937XJ    3325   DFW
#> # ... with 45 more rows
```

Da Sie den Chef gut kennen, berechnen Sie gleich noch die durchschnittliche Verspätung pro Fluggesellschaft.

```
flights %>%
  select(arr_delay, carrier, month, day, dep_time, tailnum, flight, dest) %>%
  group_by(carrier) %>%
  summarise(delay_mean = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(-delay_mean) %>%
  kable()
```

carrier	delay_mean
F9	21.921
FL	20.116
EV	15.796
YV	15.557
OO	11.931
MQ	10.775
WN	9.649
B6	9.458
9E	7.380
UA	3.558
US	2.130
VX	1.764
DL	1.644
AA	0.364
HA	-6.915
AS	-9.931

Der Chef ist zufrieden. Sie können sich wieder wichtigeren Aufgaben zuwenden...

## 0.12 Checklist zum Datenjudo

Fassen wir einige wesentliche Arbeitsschritte der Datenaufbereitung zusammen<sup>8</sup>.

### 0.12.1 Auf fehlende Werte prüfen\*

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

```
wo_men <- read.csv("https://sebastiansauer.github.io/data/wo_men.csv")
summary(wo_men)
```

---

<sup>8</sup>Achtung: Nicht `qqplot`, nicht `ggplot2`, nicht `gplot...`

```
#>           time      sex     height   shoe_size
#> 11.10.2016 12:31:59: 2 man :18 Min.   : 2 Min.   :35.0
#> 11.10.2016 12:32:10: 2 woman:82 1st Qu.:163 1st Qu.:38.0
#> 11.10.2016 12:32:32: 2 NA's : 1 Median :168 Median :39.0
#> 11.10.2016 12:32:33: 2                   Mean   :165 Mean   :39.8
#> 11.10.2016 12:32:42: 2                   3rd Qu.:174 3rd Qu.:40.0
#> 04.10.2016 17:58:51: 1                   Max.   :364 Max.   :88.0
#> (Other)          :90 NA's   : 1 NA's   : 1
```

## 0.12.2 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.

```
nrow(wo_men)
#> [1] 101
wo_men %>%
  na.omit %>%
  nrow
#> [1] 100
```

Hier verlieren wir nur 1 Zeile, das verschmerzen wir. Welche eigentlich?

```
wo_men %>%
  rownames_to_column %>% # Zeilennummer werden eine eigene Spalte
  filter(!complete.cases(.)) # Nur die nicht-kompletten Fälle filtern
#>   rowname           time  sex height shoe_size
#> 1     86 11.10.2016 12:44:06 <NA>     NA       NA
```

Man beachte, dass der Punkt . für den Datensatz steht, wie er vom letzten Schritt weitergegeben wurde. Natürlich könnten wir diesen Datensatz jetzt als neues Objekt speichern und damit weiter arbeiten.

### 0.12.3 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 Kurses[^Das sagen Autoren, wenn sie nicht genau wissen, wie etwas funktioniert.]. Eine einfache, aber nicht die beste Möglichkeit, besteht darin, die fehlenden Werte durch einen repräsentativen Wert, z.B. den Mittelwert der Spalte, zu ersetzen.

```
wo_men$height <- replace(wo_men$height, is.na(wo_men$height), mean(wo_men$height, na...
```

`replace` ersetzt Werte aus dem Vektor `wo_men$height` alle Werte, für die `is.na(wo_men$height)` wahr ist. Diese Werte werden durch den Mittelwert der Spalte ersetzt<sup>9</sup>.

### 0.12.4 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.

```
wo_men %>%
  count(shoe_size) %>%
  head # nur die ersten paar Zeilen
#> # A tibble: 6 × 2
#>   shoe_size     n
#>   <dbl> <int>
#> 1     35.0     1
#> 2     36.0     6
#> 3     36.5     1
```

---

<sup>9</sup>Hier findet sich eine ausführliche Darstellung: [https://sebastiansauer.github.io/checklist\\_data\\_cleansing/index.html](https://sebastiansauer.github.io/checklist_data_cleansing/index.html)

```
#> 4     37.0    14
#> 5     38.0    26
#> 6     39.0    18
```

### 0.12.5 Ausreiser identifizieren

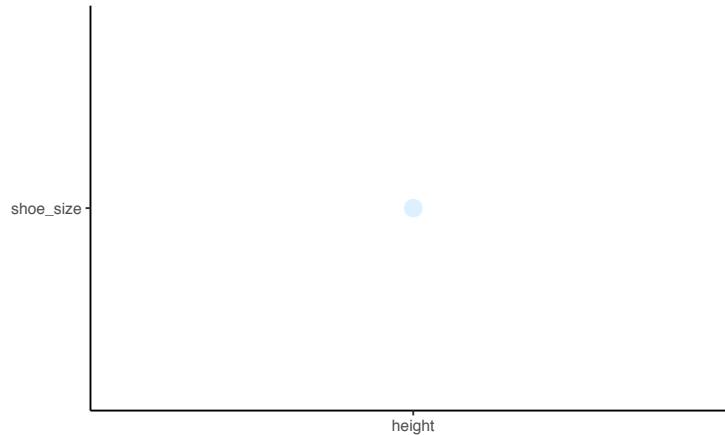
Ähnlich zu Fehlern, steht man Ausreisern häufig skeptisch gegenüber. Allerdings kann man nicht pauschal sagen, das Extremwerte entfernt werden sollen: Vielleicht war jemand in der Stichprobe wirklich nur 1.20m groß? Hier gilt es, begründet und nachvollziehbar im Einzelfall zu entscheiden. Histogramme und Boxplots sind wieder ein geeignetes Mittel, um Ausreiser zu finden.

### 0.12.6 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 ausgeschlossen werden. Oder man fasst ähnliche Variablen zusammen.

```
wo_men %>%
  select(height, shoe_size) %>%
  correlate() -> km    # Korrelationsmatrix berechnen
km
#> # A tibble: 2 × 3
#>   rowname height  shoe_size
#>   <chr>    <dbl>    <dbl>
#> 1 height      NA     0.553
#> 2 shoe_size   0.553      NA

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



Die Funktion `correlate` stammt aus dem Paket `corr10`, welches vorher installiert und geladen sein muss. Hier ist die Korrelation nicht zu groß, so dass wir keine weiteren Schritte unternehmen.

### 0.12.7 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.

```
wo_men %>%
  select_if(is.numeric) %>% # Spalte nur auswählen, wenn numerisch
  scale() %>% # z-standardisieren
  head() # nur die ersten paar Zeilen abdrucken
#>      height shoe_size
#> [1,] -0.132    0.0405
#> [2,]  0.146   -0.1395
#> [3,]  0.221   -0.1395
```

<sup>10</sup><https://github.com/drsimonj/corrr>

```
#> [4,] 0.272   0.0405
#> [5,] 0.751   1.1204
#> [6,] -0.208  -0.4994
```

Dieser Befehl liefert zwei 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:

```
wo_men %>%
  mutate_if(is.numeric, funs("z" = scale)) %>%
  head
#>           time   sex height shoe_size height_z shoe_size_z
#> 1 04.10.2016 17:58:51 woman     160      40   -0.132    0.0405
#> 2 04.10.2016 17:58:59 woman     171      39    0.146   -0.1395
#> 3 04.10.2016 18:00:15 woman     174      39    0.221   -0.1395
#> 4 04.10.2016 18:01:17 woman     176      40    0.272    0.0405
#> 5 04.10.2016 18:01:22 man      195      46    0.751    1.1204
#> 6 04.10.2016 18:01:53 woman     157      37   -0.208   -0.4994
```

Der Befehl `mutate` berechnet eine neue Spalte; `mutate_if` tut dies, 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.

### 0.12.8 Quasi-Konstante finden

Hat eine Variable nur einen Wert, so verdient sie die Ehrenbezeichnung “Variable” nicht wirklich. Haben wir z.B. nur Männer im Datensatz, so kann das Geschlecht nicht für Unterschiede im Einkommen verantwortlich sein. Besser die Variable Geschlecht dann 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.

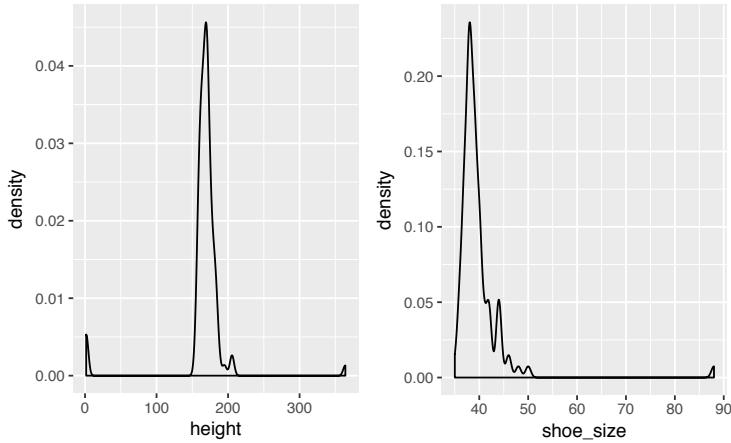
### 0.12.9 Auf Normalverteilung prüfen

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: Histogramm oder eine Dichte-Diagramm als “glatt geschmierelte” Variante des Histogramms bieten sich an.

```
wo_men %>%
  ggplot() +
  aes(x = height) +
  geom_density() -> p1

wo_men %>%
  ggplot() +
  aes(x = shoe_size) +
  geom_density() -> p2

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



Während die Körpergröße sehr deutlich normalverteilt ist, ist die Schuhgröße recht schießt. Bei schießen Verteilung können Transformationen Abhilfe schaffen.

fen. Hier erscheint die Schiefe noch erträglich, so dass wir keine weiteren Maßnahmen einleiten.

### 0.12.10 Mittelwerte pro Zeile berechnen

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/extra.csv")

extra_items <- extra %>%
  select(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”).

### 0.12.11 U

```
extra %>%
  count(n_faceb)
```

## 0.13 Verweise

- Eine schöne Demonstration der Mächtigkeit von `dplyr` findet sich hier<sup>11</sup>.
- Die GUI “exploratory” ist ein “klickbare” Umsetzung von `dplyr`, mächtig, modern und sieht cool aus: <https://exploratory.io>.
- *R for Data Science* bietet umfangreiche Unterstützung zu diesem Thema (Wickham and Grolemund 2016).

---

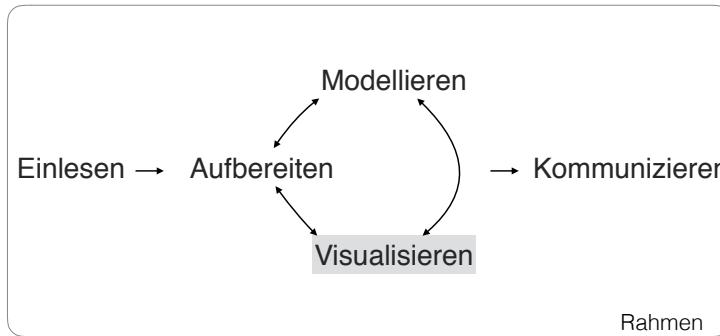
<sup>11</sup><http://bit.ly/2kX91vC>.



# Daten visualisieren

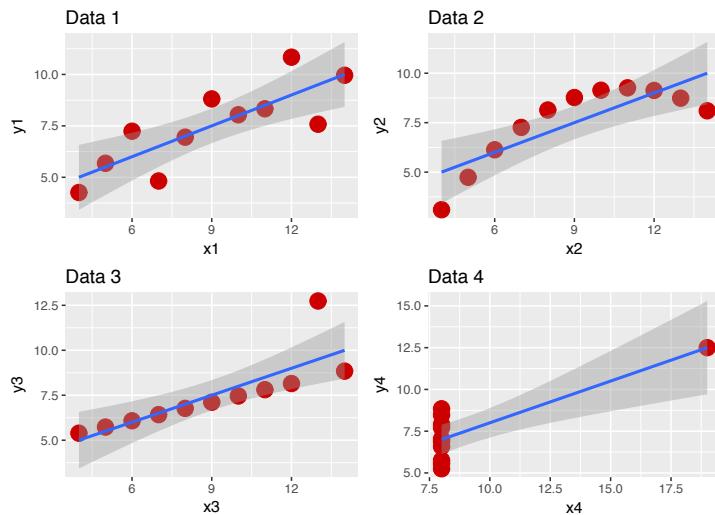
Benötigte Pakete:

```
library(tidyverse) # Zum Plotten
library(wesanderson) # Farb-Palette von Wes Anderson
library(car) # Umkodieren
library(RColorBrewer) # Farb-Palette von Cynthia Breuer
library(GGally) # Scatterplot-Matrizen
library(knitr) # HTML-Tabellen
library(gridExtra) # Mehrere Plots in einem erstellen
library(corr) # Korrelationsplots
```



Ein Bild sagt bekanntlich mehr als 1000 Worte. Schauen wir uns zur Verdeutlichung das berühmte Beispiel von Anscombe<sup>[^https://de.wikipedia.org/wiki/Anscombe-Quartett]</sup> an. Es geht hier um vier Datensätze mit zwei Variablen (Spalten; X und Y). Offenbar sind die Datensätze praktisch identisch: Alle X haben den gleichen Mittelwert und die gleiche Varianz; dasselbe gilt für die Y. Die Korrelation zwischen X und Y ist in allen vier Datensätzen gleich. Allerdings erzählt eine Visualisierung der vier Datensätze eine ganz andere

Geschichte.



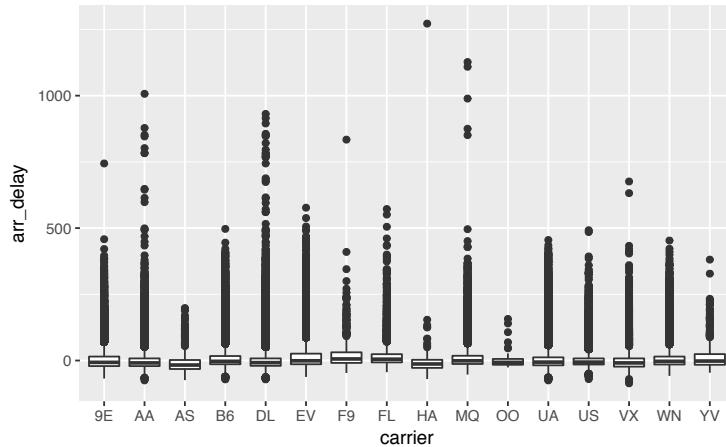
Offenbar “passieren” in den vier Datensätzen gänzlich unterschiedliche Dinge. Dies haben die Statistiken nicht aufgedeckt; erst die Visualisierung erhellt uns... Kurz: Die Visualisierung ist ein unverzichtbares Werkzeug, um zu verstehen, was in einem Datensatz (und damit in der zugrundeliegenden “Natur”)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

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 Durchpfifens passt: `ggplot2`[^“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>].

Laden wir dazu den Datensatz `nycflights::flights`.

```
data(flights, package = "nycflights13")
```

```
qplot(x = carrier, y = arr_delay, geom = "boxplot", data = flights)
```



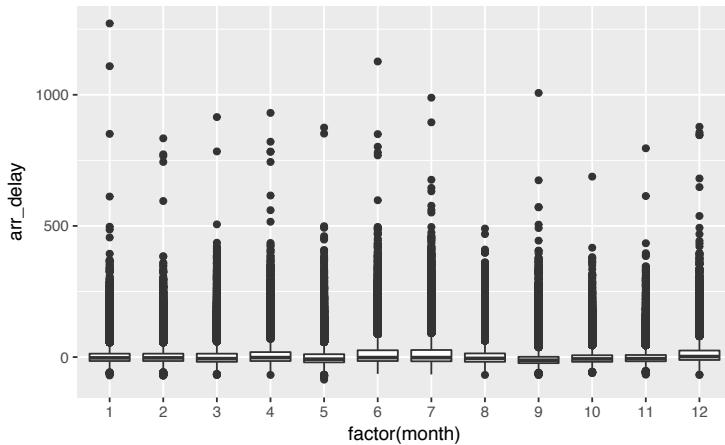
Schauen wir uns den Befehl `qplot` etwas näher an. Wie ist er aufgebaut?



`qplot`: Erstelle schnell (q wie quick in `qplot`) mal einen Plot (engl. “plot”: Diagramm).  
 x: Der X-Achse soll die Variable “carrier” zugeordnet werden.  
 y: Der Y-Achse soll die Variable “arr\_dely” zugeordnet werden.  
 geom: (“geometrisches Objekt”) Gemalt werden soll ein Boxplot, nicht etwa Punkte, Linien oder sonstiges.  
 data: Als Datensatz bitte `flights` verwenden.

Offenbar gibt es viele Extremwerte, was die Verspätung betrifft. Das erscheint mir nicht unplausibel (Schneesturm im Winter, Flugzeug verschwunden...). Vor dem Hintergrund der Extremwerte erscheinen die mittleren Verspätungen (Mediane) in den Boxplots als ähnlich. Vielleicht ist der Unterschied zwischen den Monaten ausgeprägter?

```
qplot(x = factor(month), y = arr_delay, geom = "boxplot", data = flights)
```



Kaum Unterschied; das spricht gegen die Schneesturm-Idee als Grund für Verspätung. Aber schauen wir uns zuerst die Syntax von `qplot` näher an. “q” in `qplot` steht für “quick”. Tatsächlich hat `qplot` einen großen Bruder, `ggplot12`, 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...

## 0.14 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.

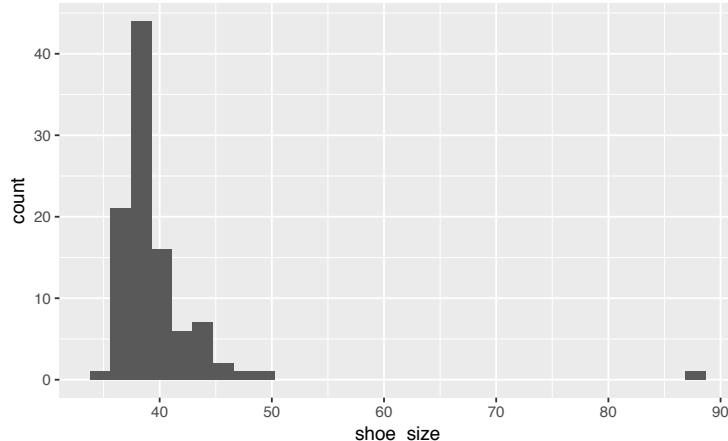
---

<sup>12</sup>Achtung: Nicht `qqplot`, nicht `ggplot2`, nicht `gplot`...

### 0.14.1 Eine kontinuierliche Variable

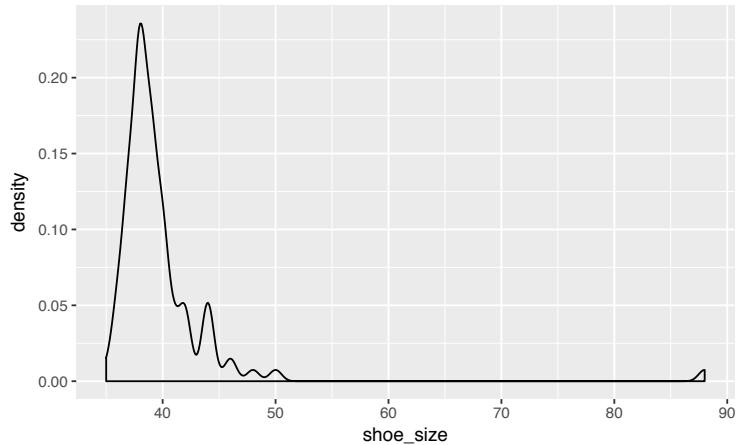
Schauen wir uns die Verteilung der Schuhgrößen von Studierenden an.

```
wo_men <- read.csv("data/wo_men.csv")  
  
qplot(x = shoe_size, data = wo_men)
```



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). Alternativ wäre ein Dichtediagramm hier von Interesse:

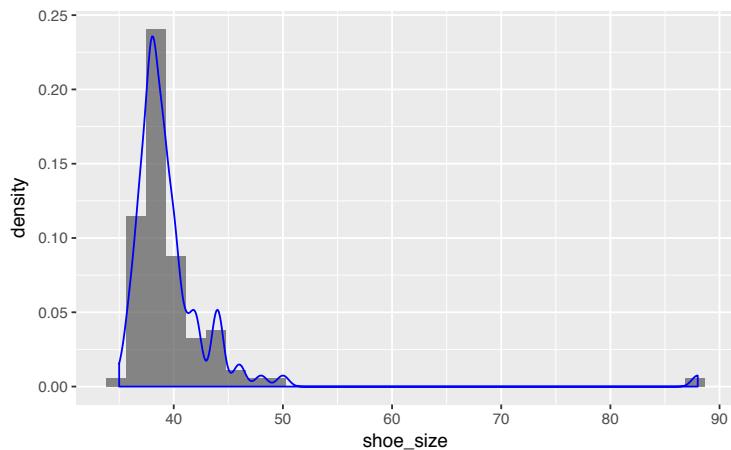
```
# qplot(x = shoe_size, data = wo_men) wie oben  
  
qplot(x = shoe_size, data = wo_men, geom = "density")
```



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

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

```
ggplot(data = wo_men) +
  aes(x = shoe_size) +
  geom_histogram(aes(y = ..density..), alpha = .7) +
  geom_density(color = "blue")
```

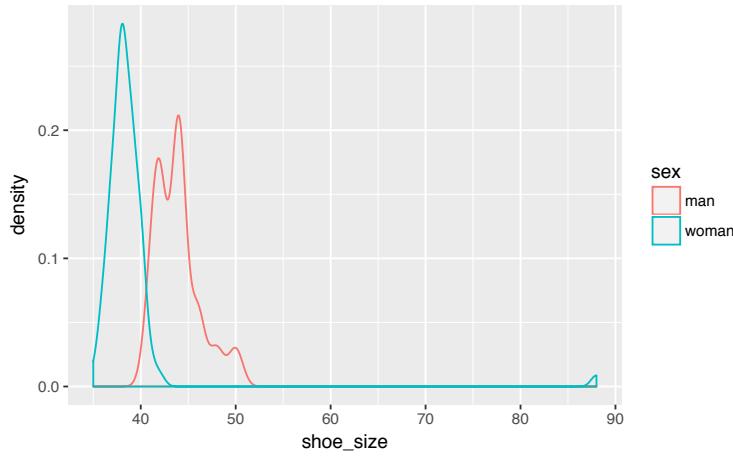


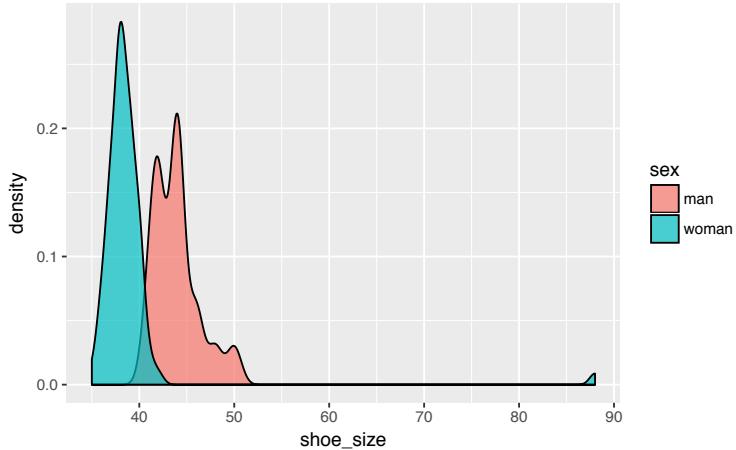
Zuerst haben wir mit dem Parameter `data` den Dateframe benannt. `aes`

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 Dateframe 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 schön, ein Diagramm für Männer und eines für Frauen zu haben, um die Verteilungen vergleichen zu können?

```
qplot(x = shoe_size, data = wo_men, geom = "density", color = sex)
qplot(x = shoe_size, data = wo_men, geom = "density", fill = sex, alpha = I(.7))
```

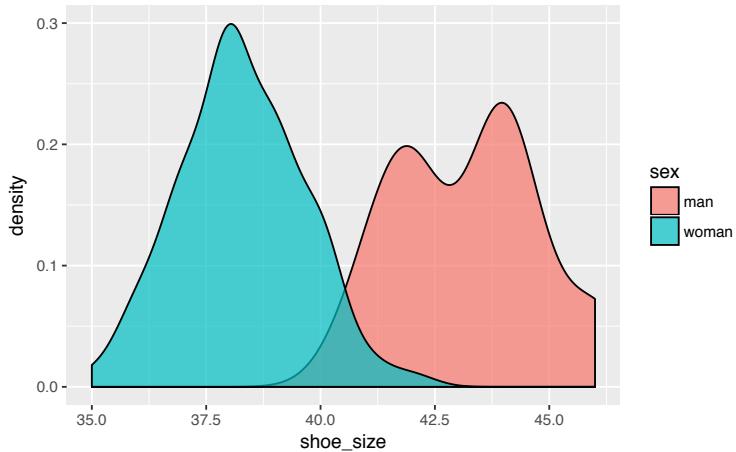




Hier sollten vielleicht noch die Extremwerte entfernt werden, um den Blick auf das Gros der Werte nicht zu verstauen:

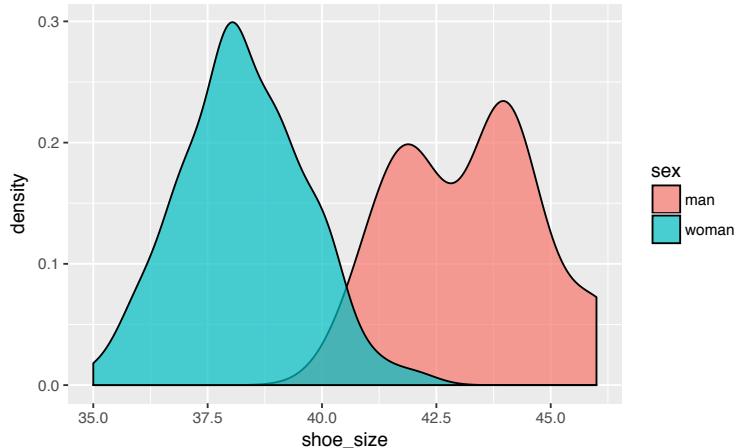
```
wo_men %>%
  filter(shoe_size <= 47) -> wo_men2

qplot(x = shoe_size, data = wo_men2, geom = "density", fill = sex, alpha = I(.2))
```



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

```
wo_men %>%
  filter(shoe_size <= 47) %>%
  qplot(x = shoe_size, data = ., geom = "density", fill = sex, alpha = I(.7))
```



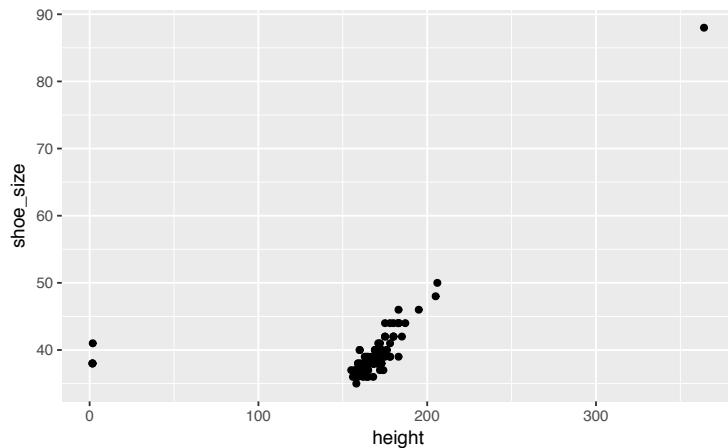
Die Pfeife versucht im Standard, das Endprodukt des letzten Arbeitsschritts an den *ersten* Parameter des nächsten Befehls weiterzugeben. Ein kurzer Blick in die Hilfe von `qplot` zeigt, dass der erste Parameter nicht `data` ist, sondern `x`. Daher müssen wir explizit sagen, an welchen Parameter wir 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 = sex` sagen wir `qplot`, dass er für Männer und Frauen jeweils ein Dichtediagramm erzeugen soll; jedem Dichtediagramm wird dabei eine Farbe zugewiesen (die uns `ggplot2` im Standard voraussucht). Mit anderen Worten: Die Werte von `sex` werden der Füllfarbe der Histogramme zugeordnet. Anstelle der Füllfarbe hätten wir auch die Linienfarbe verwenden können; die Syntax wäre dann: `color = sex`.

## 0.14.2 Zwei kontinuierliche Variablen

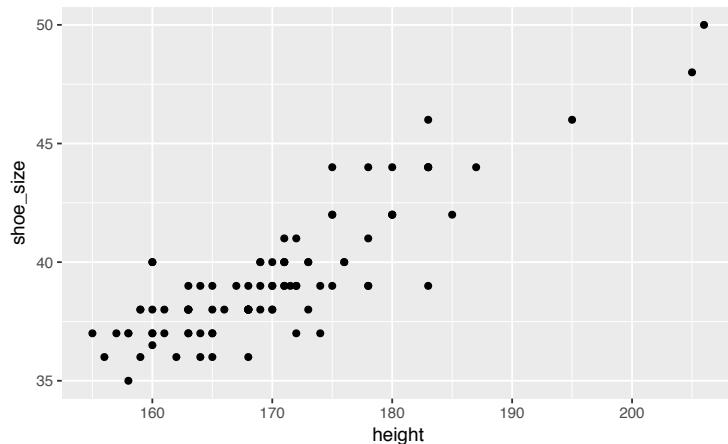
Ein Streudiagramm ist die klassische Art, zwei metrische Variablen darzustellen. Das ist mit `qplot` einfach:

```
qplot(x = height, y = shoe_size, data = wo_men)
```



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"). Wir sollten aber noch die Extremwerte herausnehmen:

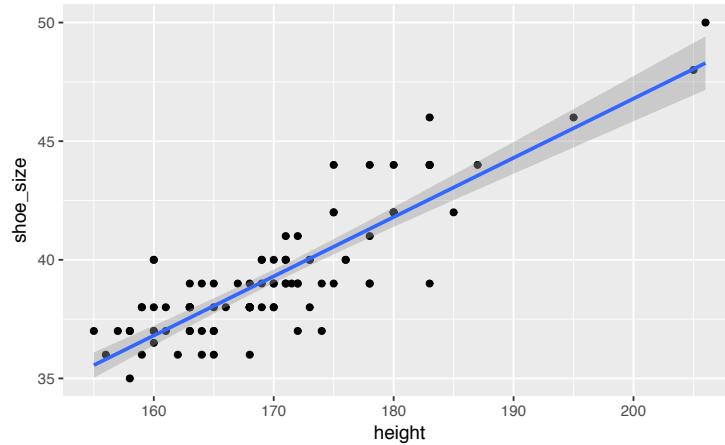
```
wo_men %>%
  filter(height > 150, height < 210, shoe_size < 55) %>%
  qplot(x = height, y = shoe_size, data = .)
```



Der Trend ist deutlich erkennbar: Je größer die Person, desto länger die Fuß'.

Zeichnen wir noch eine Trendgerade ein.

```
wo_men %>%
  filter(height > 150, height < 210, shoe_size < 55) %>%
  qplot(x = height, y = shoe_size, data = .) +
  geom_smooth(method = "lm")
```



Synonym könnten wir auch schreiben:

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

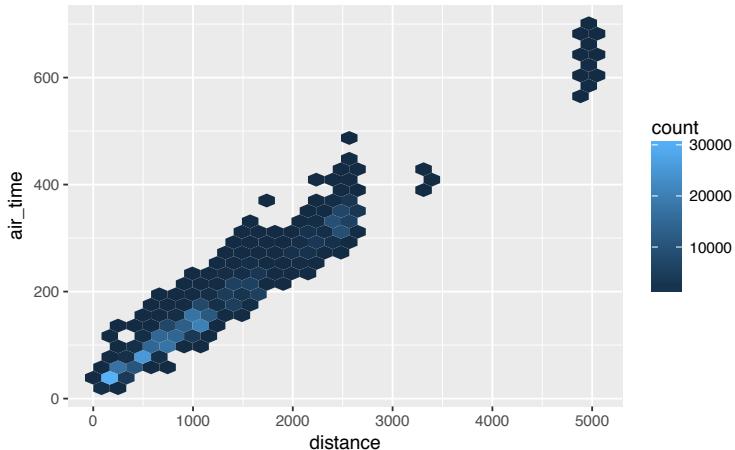
Da `ggplot` als *ersten* Parameter die Daten erwartet, kann die Pfeife hier problemlos durchgereicht werden. Innerhalb eines `ggplot`-Aufrufs werden die einzelnen Teile durch ein Pluszeichen + voneinander getrennt. Nachdem wir den Dataframe benannt haben, definieren wir die Zuweisung der Variablen zu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

```
data(flights, package = "nycflights13")
nrow(flights) # groß!
#> [1] 336776

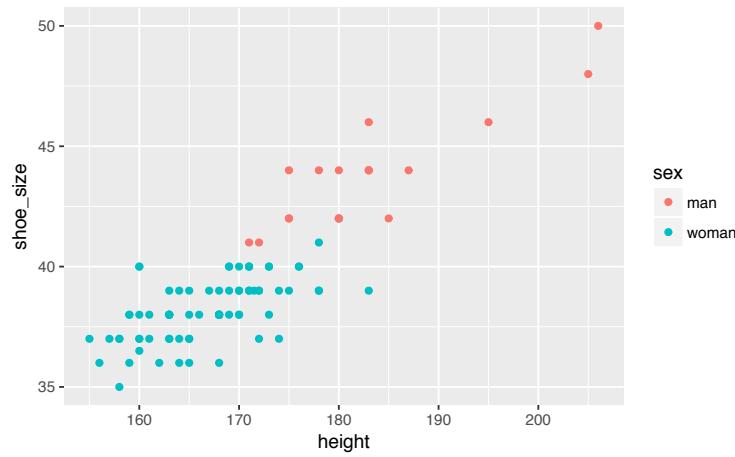
ggplot(flights) +
  aes(x = distance, y = air_time) +
  geom_hex()
```



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

```
wo_men %>%
  filter(height > 150, height < 210, shoe_size < 55) %>%
```

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

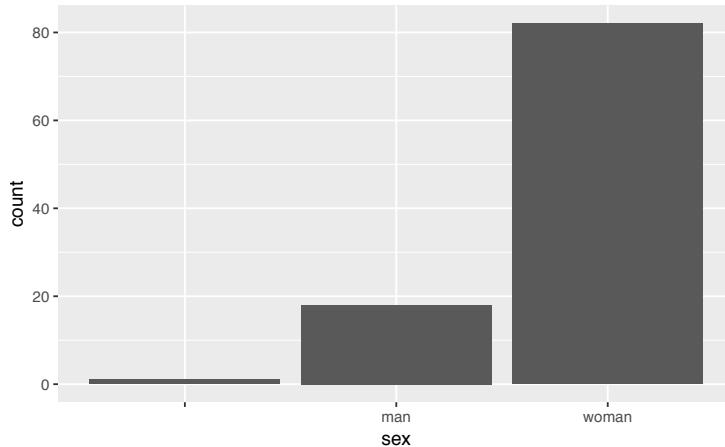


Mit `color = sex` sagen wir, dass die Linienfarbe (der Punkte) entsprechend der Stufen von `sex` eingefärbt werden sollen. Die genaue Farbwahl übernimmt `ggplot2` für uns.

### 0.14.3 Eine diskrete Variable

Bei diskreten Variablen, vor allem nominalen Variablen, geht es in der Regel darum, Häufigkeiten auszuzählen. Wie viele Männer und Frauen sind in dem Datensatz?

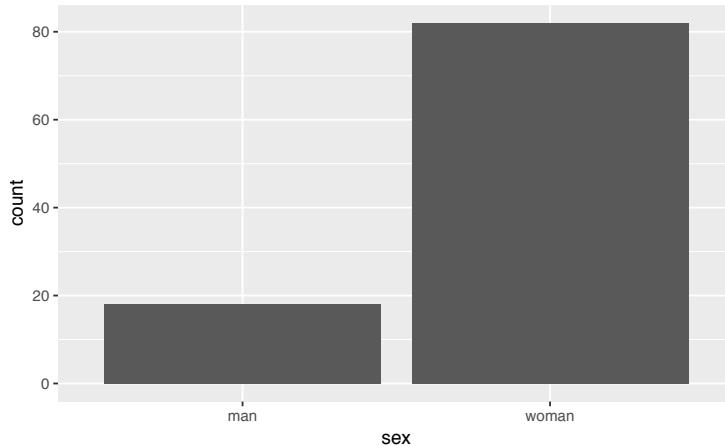
```
qplot(x = sex, data = wo_men)
```



Falls nur die X-Achse definiert ist und dort eine Faktorvariable oder eine Text-Variable steht, dann nimmt `qplot` automatisch ein Balkendiagramm als Geom.

Entfernen wir vorher noch die fehlenden Werte:

```
wo_men %>%
  na.omit() %>%
  qplot(x = sex, data = .)
```



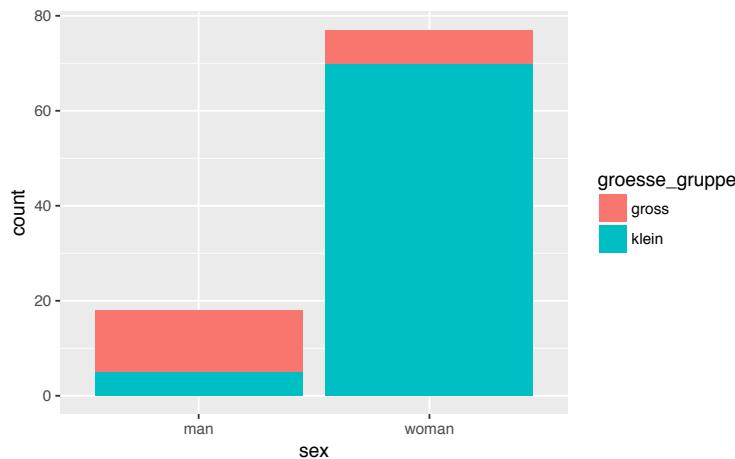
Wir könnten uns jetzt die Frage stellen, wie viele kleine und viele große Menschen es bei Frauen und bei den Männern gibt. Dazu müssen wir zuerst eine Variable wie "Größe gruppiert" erstellen mit zwei Werten: "klein" und

“groß”. Nennen wir sie `groesse_gruppe`

```
wo_men$groesse_gruppe <- car::recode(wo_men$height, "lo:175 = 'klein'; else = 'gross'"
```

```
wo_men %>%
  filter(height > 150, height < 210, shoe_size < 55) %>%
  na.omit -> wo_men2
```

```
qplot(x = sex, fill = groesse_gruppe, data = wo_men2)
```



In Worten sagt der `recode`-Befehl hier in etwa: “Kodiere `wo_men$height` um, und zwar vom kleinsten (`lo`) Wert bis 170 soll den Wert `klein` bekommen, ansonsten bekommt eine Größe den Wert `gross`”.

Hier haben wir `qplot` gesagt, dass die Balken entsprechend der Häufigkeit von `groesse_gruppe` füllen soll. Und bei den Frauen sind bei dieser Variablen die Werte `klein` häufig; bei den Männern hingegen die Werte `gross`.

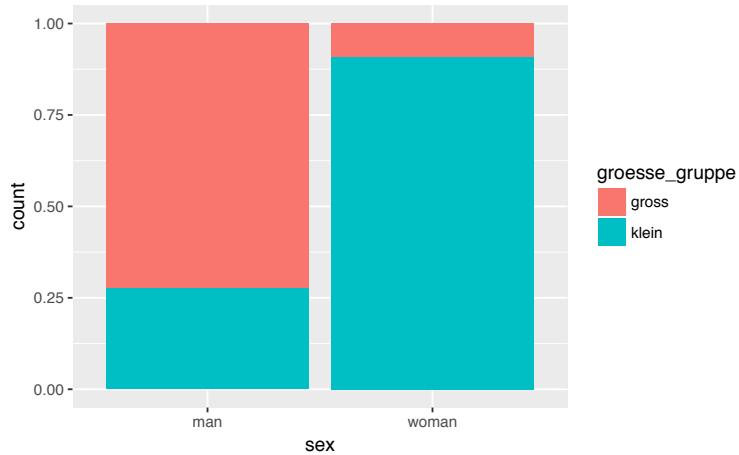
Schön wäre noch, wenn die Balken Prozentwerte angeben würden. Das geht mit `qplot` (so) nicht; wir schwenken auf `ggplot` um<sup>13</sup>.

```
wo_men2 %>%
  ggplot() +
```

---

<sup>13</sup>Cleveland fände diese Idee nicht so gut.

```
aes(x = sex, fill = groesse_gruppe) +
geom_bar(position = "fill")
```



Schauen wir uns die Struktur des Befehls `ggplot` näher an.



`wo_men2`: Hey R, nimm den Datensatz `wo_men2` 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 `wo_men2`)!  
`+`: Das Pluszeichen grenzt die Teile eines ggplot-Befehls voneinander  
 ab.  
`aes`: von “aethetics”, also welche Variablen des Datensatzes den sicht-  
 baren Aspekten (v.a. Achsen, Farben) zugeordnet werden.  
`x`: Der X-Achse (Achtung, x wird klein geschrieben hier) wird die Vari-  
 able `sex` zugeordnet.  
`y`: gibt es nicht??? Wenn in einem ggplot-Diagramm *keine* Y-Achse  
 definiert wird, wird ggplot automatisch ein Histogramm bzw. ein Balken-  
 diagramm 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 rela-  
 tive Häufigkeit).  
`fill` Das Diagramm (die Balken) sollen so gefüllt werden, dass sich die  
 Häufigkeit der Werte von `groesse_gruppe` darin widerspiegelt.

**geom\_XYZ:** Als “Geom” soll ein Balken (“bar”) gezeichnet werden. Ein Geom ist in ggplot2 das zu zeichnende Objekt, also ein Boxplot, ein Balken, Punkte, Linien etc. Entsprechend wird gewünschte Geom mit `geom_bar`, `geom_boxplot`, `geom_pointetc.` gewählt. `position = fill:position_fill` will sagen, dass die Balken alle eine Höhe von 100% (1) haben. 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).

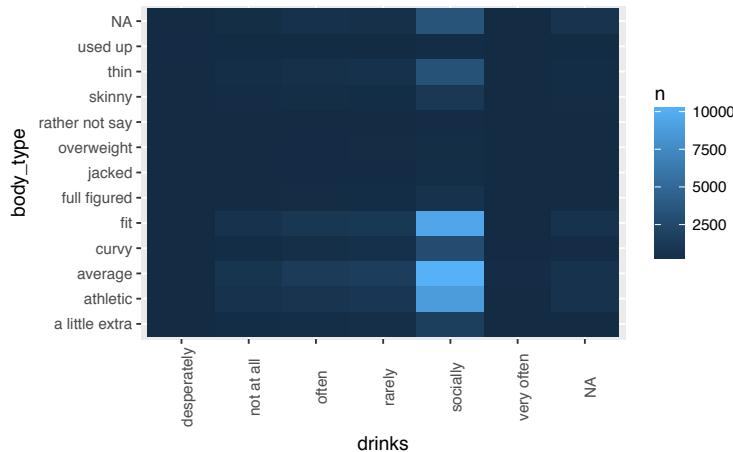
#### 0.14.4 Zwei diskrete Variablen

Arbeitet man mit nominalen Variablen, so sind Kontingenztabellen Täglich Brot. Z.B.: Welche Produkte wurden wie häufig an welchem Standort verkauft? Wie ist die Verteilung von Alkoholkonsum und Körperform bei Menschen einer Single-Börse. Bleiben wir bei letztem Beispiel.

```
data(profiles, package = "okcupiddata")

profiles %>%
  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.

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.

#### 0.14.5 Zusammenfassungen zeigen

Manchmal möchten wir *nicht* die Rohwerte einer Variablen darstellen, sondern z.B. die Mittelwerte pro Gruppe. Mittelwerte sind eine bestimmte

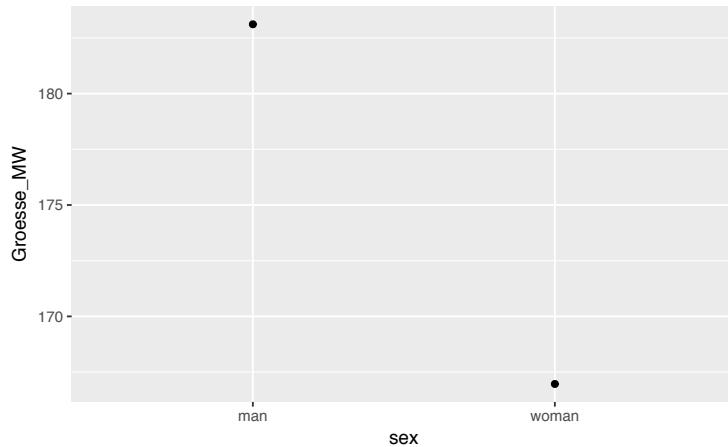
*Zusammenfassung* einer Spalte; also fassen wir zuerst die Körpergröße zum Mittelwert zusammen - gruppiert nach Geschlecht.

```
wo_men2 %>%
  group_by(sex) %>%
  summarise(Groesse_MW = mean(height)) -> wo_men3

wo_men3
#> # A tibble: 2 × 2
#>   sex     Groesse_MW
#>   <fctr>    <dbl>
#> 1 man        183
#> 2 woman      167
```

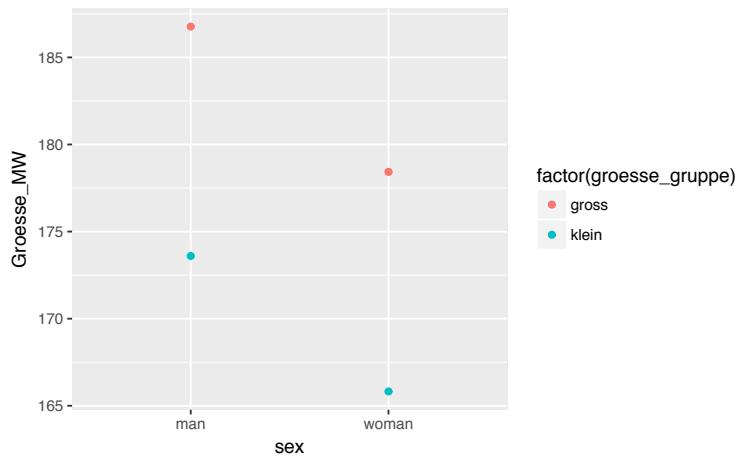
Diese Tabelle schieben wir jetzt in ggplot2; natürlich hätten wir das gleich in einem Rutsch durchpfeifen können.

```
wo_men3 %>%
  qplot(x = sex, y = Groesse_MW, data = .)
```



Das Diagramm besticht nicht durch die Tiefe und Detaillierung. Wenn wir noch zusätzlich die Mittelwerte nach `Groesse_Gruppe` ausweisen, wird das noch überschaubar bleiben.

```
wo_men2 %>%
  group_by(sex, groesse_gruppe) %>%
  summarise(Groesse_MW = mean(height)) %>%
  qplot(x = sex, color = factor(groesse_gruppe), y = Groesse_MW, data = .)
```



## 0.15 Farblehre

Erstens, nicht schaden - so könnte hier die Maßregel sein. Es ist leicht, zu grelle oder wenig kontrastierende Farben auszuwählen. Eine gute Farbauswahl (Palette) ist nicht so leicht und hängt vom Zweck der Darstellung ab.

Cynthia Brewer<sup>14</sup> hat einige schöne Farbpaletten zusammengestellt; diese sind in R und in ggplot2 über das Paket `RcolorBrewer` verfügbar.

```
brewer.pal.info %>% rownames_to_column %>% rename(Name = rowname) %>% kable
```

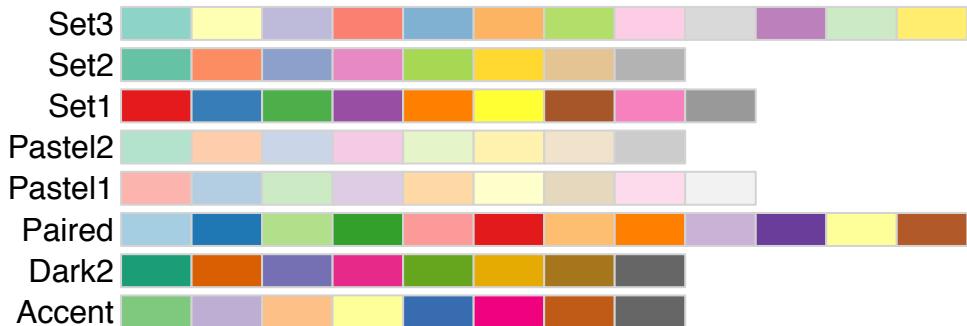
---

<sup>14</sup><http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3>

Name	maxcolors	category	colorblind
BrBG	11	div	TRUE
PiYG	11	div	TRUE
PRGn	11	div	TRUE
PuOr	11	div	TRUE
RdBu	11	div	TRUE
RdGy	11	div	FALSE
RdYlBu	11	div	TRUE
RdYlGn	11	div	FALSE
Spectral	11	div	FALSE
Accent	8	qual	FALSE
Dark2	8	qual	TRUE
Paired	12	qual	TRUE
Pastel1	9	qual	FALSE
Pastel2	8	qual	FALSE
Set1	9	qual	FALSE
Set2	8	qual	TRUE
Set3	12	qual	FALSE
Blues	9	seq	TRUE
BuGn	9	seq	TRUE
BuPu	9	seq	TRUE
GnBu	9	seq	TRUE
Greens	9	seq	TRUE
Greys	9	seq	TRUE
Oranges	9	seq	TRUE
OrRd	9	seq	TRUE
PuBu	9	seq	TRUE
PuBuGn	9	seq	TRUE
PuRd	9	seq	TRUE
Purples	9	seq	TRUE
RdPu	9	seq	TRUE
Reds	9	seq	TRUE
YlGn	9	seq	TRUE
YlGnBu	9	seq	TRUE
YlOrBr	9	seq	TRUE
YlOrRd	9	seq	TRUE

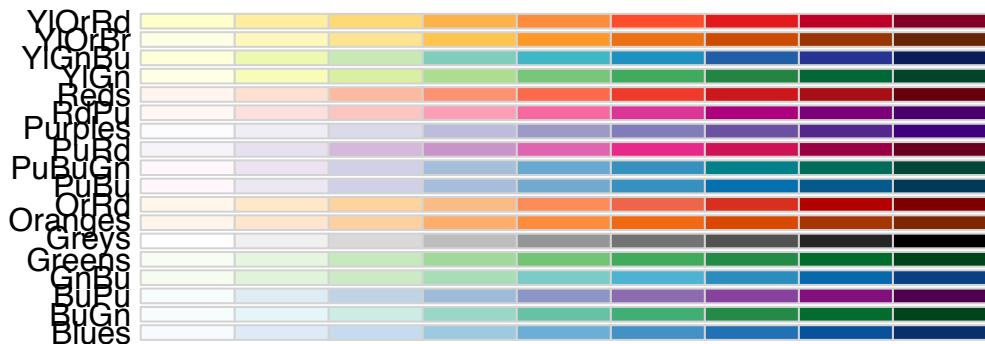
- Kontrastierende Darstellung (nominale/ qualitative Variablen) - z.B. Männer vs. Frauen

```
display.brewer.all(type="qual")
```



- Sequenzielle Darstellung (unipolare numerische Variablen) - z.B. Preis oder Häufigkeit

```
display.brewer.all(type="seq")
```



- Divergierende Darstellung (bipolare numerische Variablen) - z.B. semantische Potenziale oder Abstufung von "stimme überhaupt nicht zu" über "neutral" bis "stimme voll und ganz zu"

```
display.brewer.all(type="div")
```



In `ggplot2` können wir folgendermaßen Paletten ändern.

```

flights %>%
  group_by(dest) %>%
  count(dest) %>%
  top_n(5)
#> # A tibble: 5 × 2
#>   dest     n
#>   <chr> <int>
#> 1 ATL    17215
#> 2 BOS    15508
#> 3 LAX    16174
#> 4 MCO    14082
#> 5 ORD    17283

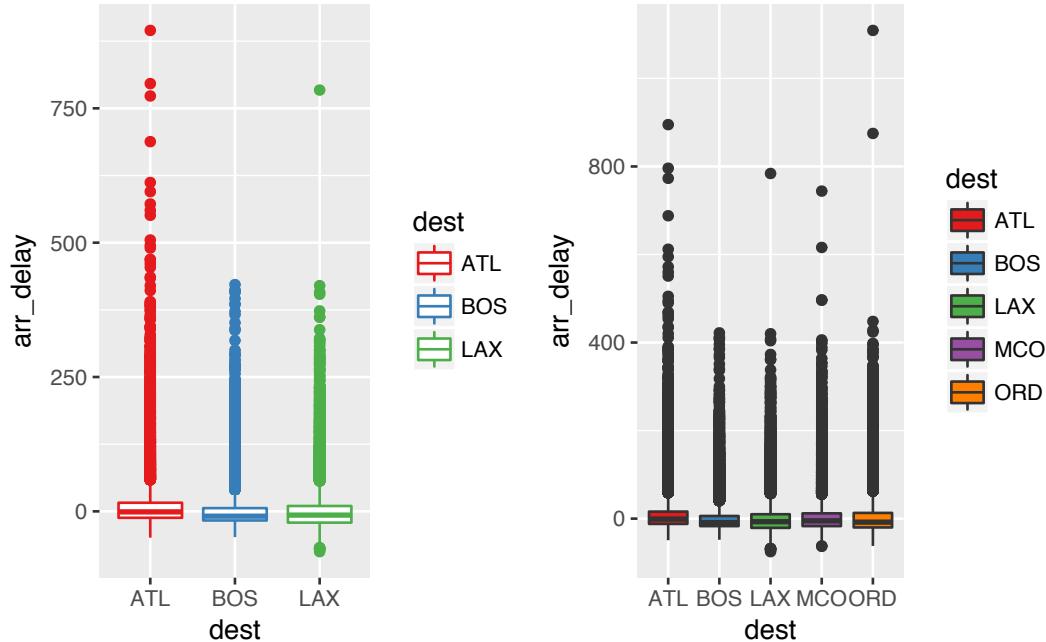
p1 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX")) %>%
  ggplot() +
  aes(x = dest, y = arr_delay, color = dest) +
  geom_boxplot() +
  scale_color_brewer(palette = "Set1")

p2 <- flights %>%
  filter(dest %in% c("BOS", "ATL", "LAX", "MCO", "ORD")) %>%
  ggplot() +
  aes(x = dest, y = arr_delay, fill = dest) +
  geom_boxplot() +

```

```
scale_fill_brewer(palette = "Set1")

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



`scale_color_brewer` meint hier: “Ordne der Variablen, die für ‘color’ zuständig ist, hier `sex`, eine Farbe aus der Brewer-Palette ‘Set1’ zu”. Die Funktion wählt *automatisch* die richtige Anzahl von Farben.

Man beachte, dass die Linienfarbe über `color` und die Füllfarbe über `fill` zugewiesen wird. Punkte haben nur eine Linienfarbe, keine Füllfarbe.

Auch die Farbpaletten von Wes Anderson sind erbaulich<sup>15</sup>. Diese sind nicht “hart verdrahtet” in ggplot2, sondern werden über `scale_XXX_manual` zugewiesen (wobei XXX z.B. `color` oder `fill` sein kann).

```
data(tips, package = "reshape2")
```

<sup>15</sup><https://github.com/karthik/wesanderson>

```

p1 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = wes_palette("GrandBudapest")) +
  theme(legend.position = "bottom")

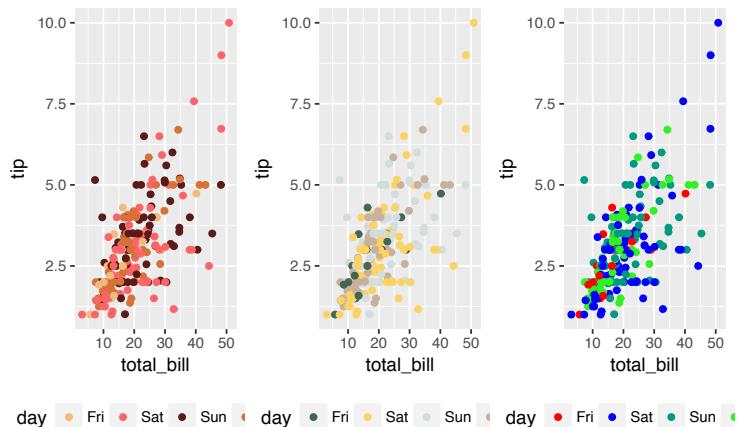
p2 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = wes_palette("Chevalier")) +
  theme(legend.position = "bottom")

meine_farben <- c("red", "blue", "#009981", "#32F122")

p3 <- tips %>%
  ggplot() +
  aes(x = total_bill, y = tip, color = day) +
  geom_point() +
  scale_color_manual(values = meine_farben) +
  theme(legend.position = "bottom")

grid.arrange(p1, p2, p3, ncol = 3)

```



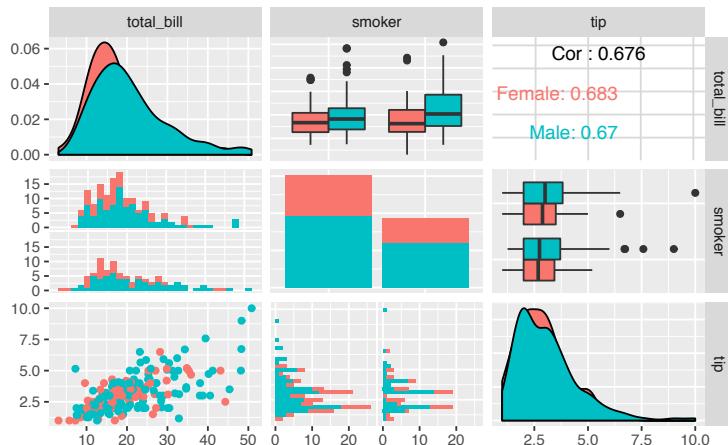
Wer sich berufen fühlt, eigene Farben (oder die seiner Organisation zu verwenden), kommt auf ähnlichem Weg zu Ziel. Man definiere sich seine Palette, wobei ausreichend Farben definiert sein müssen. Diese weist man dann über `scale_XXX_manual` dann zu. Man kann einerseits aus den in R definierten Farben auswählen<sup>16</sup> oder sich selber die RGB-Nummern (in Hexadezimal-Nummern) heraussuchen.

## 0.16 Erweiterungen für ggplot

### 0.16.1 ggpairs

Um eine Streudiagramm-Matrix darzustellen, ist der Befehl `GGally::ggpairs` praktisch:

```
tips %>%
  ggpairs(aes(color = sex), columns = c("total_bill", "smoker", "tip"))
```



Dabei gibt man an, welche Variable (hier `sex`) für die Farben im Diagramm zuständig sein soll (wir ordnen den Werten von `sex` jeweils eine Farbe zu). Mit `columns` sagen wir, welche Spalten des Dataframes wir dargestellt haben

---

<sup>16</sup><http://sape.inf.usi.ch/quick-reference/ggplot2/colour>

möchten. Lassen wir diesen Parameter weg, so werden alle Spalten des Dataframes dargestellt.

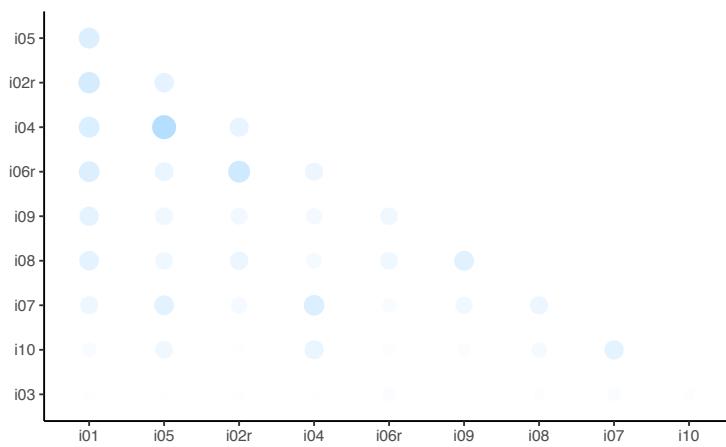
## 0.16.2 Correlationsplots

Mit dem Paket `corr` lassen sich mehrere Korrelationskoeffizienten auf einmal visualisieren.

```
df <- read.csv("https://osf.io/meyhp/?action=download")

library(corr)

df %>%
  select(i01:i10) %>%    # Spalten wählen
  correlate() %>%    # Korrelationsmatrix berechnen
  rearrange() %>%    # Korrelation der Stärke nach ordnen
  shave() %>%    # das obere Dreieck ist redundant, rasieren wir ab
  rplot()    # plotten
```



**Figure 11:** Korr

### 0.16.3 Weitere

Hier finden sich viele weitere Ergänzungen für ggplot2: <https://www.ggplot2-exts.org>

## 0.17 Fallstudie Extraversion

Eine recht häufige Art von Daten in der Wirtschaft kommen von Umfragen in der Belegschaft. Diese Daten gilt es dann aufzubereiten und graphisch wiederzugeben.

### 0.17.1 Daten einlesen

Hier laden wir einen Datensatz zu einer Online-Umfrage:

```
data <- read.csv("https://osf.io/meyhp/?action=download")
```

Der DOI für diesen Datensatz ist 10.17605/OSF.IO/4KGZH.

Der Datensatz besteht aus 10 Extraversions-Items (B5T nach Satow<sup>17</sup>) sowie einigen Verhaltenskorrelaten (zumindest angenommenen). Uns interessieren also hier nur die 10 Extraversions-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 hier<sup>18</sup> eingesehen werden. Schauen wir uns die Daten mal an:

```
glimpse(data)
#> Observations: 501
```

---

<sup>17</sup>[https://www.zpid.de/pub/tests/PT\\_9006357\\_B5T\\_Forschungsbericht.pdf](https://www.zpid.de/pub/tests/PT_9006357_B5T_Forschungsbericht.pdf)

<sup>18</sup>[https://docs.google.com/forms/d/e/1FAIpQLSfD4wQuhDV\\_edx1WBfN3Qos7XqoVbe41VpiKLRKtGLEuUD09Q/viewform](https://docs.google.com/forms/d/e/1FAIpQLSfD4wQuhDV_edx1WBfN3Qos7XqoVbe41VpiKLRKtGLEuUD09Q/viewform)

```
#> Variables: 28
#> $ X <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, ...
#> $ timestamp <fctr> 11.03.2015 19:17:48, 11.03.2015 19:18:05, ...
#> $ code <fctr> HSC, ERB, ADP, KHB, PTG, ABL, ber, hph, IH...
#> $ i01 <int> 3, 2, 3, 3, 4, 3, 4, 3, 4, 4, 3, 3, 4, 4, 3...
#> $ i02r <int> 3, 2, 4, 3, 3, 2, 4, 3, 4, 4, 3, 4, 3, 3, 3...
#> $ i03 <int> 3, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 4, 1...
#> $ i04 <int> 3, 2, 4, 4, 4, 3, 3, 4, 4, 3, 3, 2, 4, 3...
#> $ i05 <int> 4, 3, 4, 3, 4, 2, 3, 2, 3, 3, 2, 3, 3, 3...
#> $ i06r <int> 4, 2, 1, 3, 3, 3, 2, 4, 3, 3, 3, 3, 3, 3...
#> $ i07 <int> 3, 2, 3, 3, 4, 4, 2, 3, 3, 3, 2, 4, 2, 3, 3...
#> $ i08 <int> 2, 3, 2, 3, 2, 3, 3, 2, 3, 3, 2, 3, 3, 4...
#> $ i09 <int> 3, 3, 3, 3, 3, 3, 4, 4, 3, 4, 2, 4, 4, 4...
#> $ i10 <int> 1, 1, 1, 2, 4, 3, 2, 1, 2, 3, 1, 3, 2, 3, 2...
#> $ n_facebook_friends <int> 250, 106, 215, 200, 100, 376, 180, 432, 200...
#> $ n_hangover <int> 1, 0, 0, 15, 0, 1, 1, 2, 5, 0, 1, 2, 20, 2, ...
#> $ age <int> 24, 35, 25, 39, 29, 33, 24, 28, 29, 38, 25, ...
#> $ sex <fctr> Frau, Frau, Frau, Frau, Frau, Mann, Frau, ...
#> $ extra_single_item <int> 4, 3, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4...
#> $ time_conversation <dbl> 10, 15, 15, 5, 5, 20, 2, 15, 10, 10, 1, 5, ...
#> $ presentation <fctr> nein, nein, nein, nein, ja, ja, ja, ...
#> $ n_party <int> 20, 5, 3, 25, 4, 4, 3, 6, 12, 5, 10, 5, 10, ...
#> $ clients <fctr> , , , , , , , , , , , , , , ...
#> $ extra_vignette <fctr> , , , , , , , , , , , , , , ...
#> $ extra_description <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
#> $ prop_na_per_row <dbl> 0.0435, 0.0435, 0.0435, 0.0435, 0.0435, 0.0...
#> $ extra_mean <dbl> 2.9, 2.1, 2.6, 2.9, 3.2, 2.8, 2.8, 2.5, 3.2...
#> $ extra_median <dbl> 3.0, 2.0, 3.0, 3.0, 3.5, 3.0, 3.0, 2.5, 3.5...
#> $ client_freq <int> NA, ...
```

## 0.17.2 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 :-)

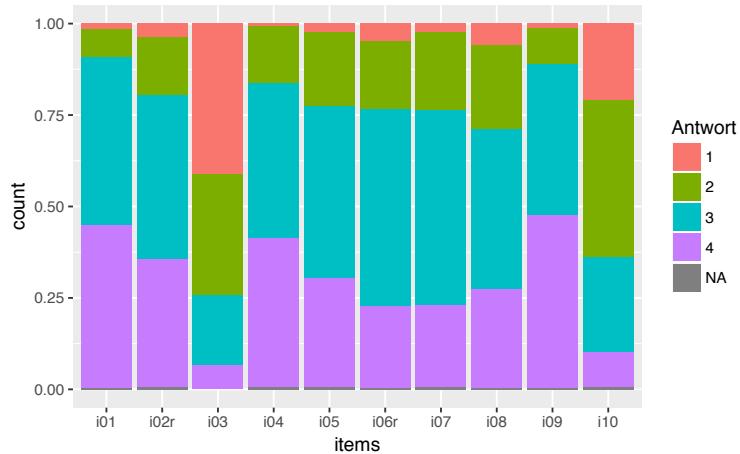
Man gewöhnt sich daran einerseits; und andererseits ist es vielleicht auch so, dass diese Funktionen nicht perfekt sind, oder nicht aus unserer Sicht oder nur aus Sicht des Menschen, der die Funktion geschrieben hat. Jedenfalls brauchen wir hier eine `factor` Variable zur Gruppierung...

Damit haben wir es schon! Jetzt wird gemalt.

### 0.17.3 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 Aspekte 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.).

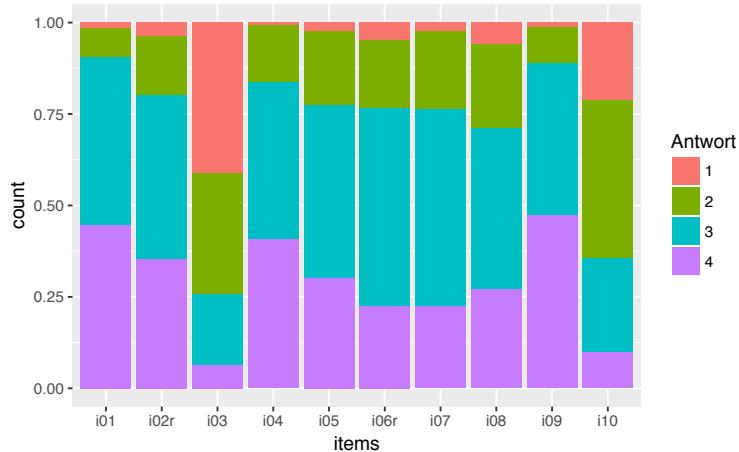
- `position = "fill"` sagt, dass die Gesamt-Höhe des Balken aufgeteilt werden soll mit den “Teil-Höhen” der Gruppen (Antwort-Kategorien 1 bis 4); wir hätten die Teil-Höhen auch nebeneinander stellen können.

Vielleicht ist es schöner, die NAs erst zu entfernen.

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

Und dann noch mal plotten:

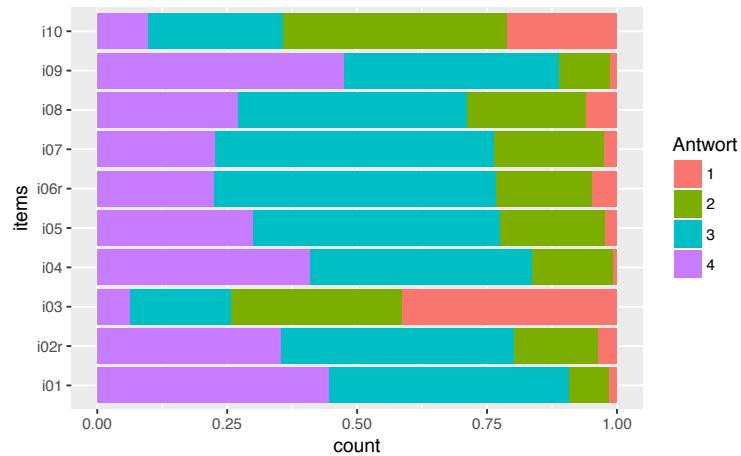
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill")
```



#### 0.17.4 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()
```



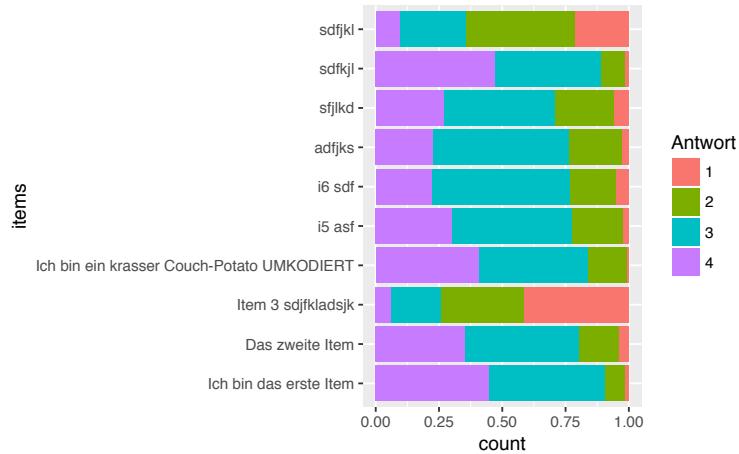
### 0.17.5 Text-Labels für die Items

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", "sfj1kd", "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)
```

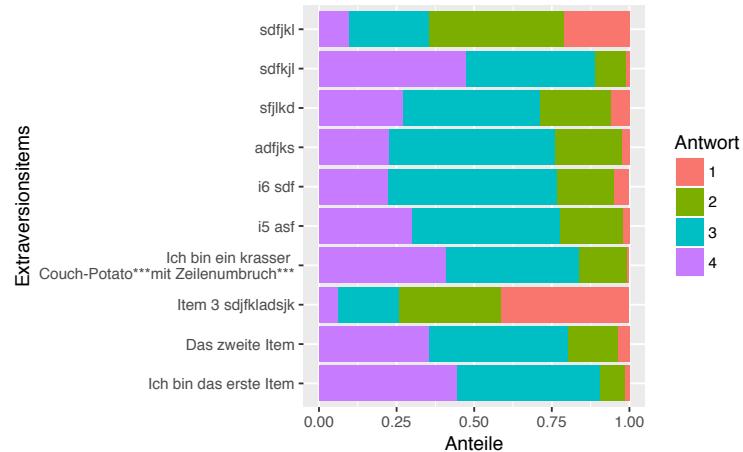


Man kann auch einen Zeilenumbruch in den Item-Labels erzwingen... wobei das führt uns schon recht weit, aber gut, zum Abschluss :-)

```
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", "sfjlkd", "sdfkjl", "sdfjkl")
```

Und wieder plotten:

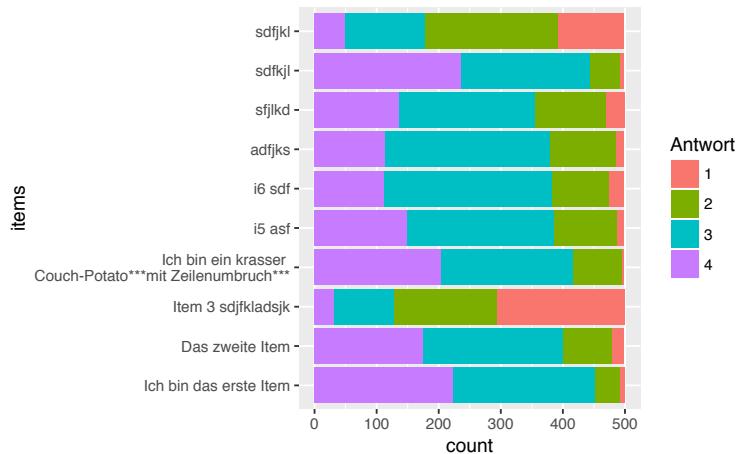
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "fill") +
  coord_flip() +
  scale_x_discrete(labels = item_labels, name = "Extraversionsitems") +
  scale_y_continuous(name = "Anteile")
```



## 0.17.6 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”

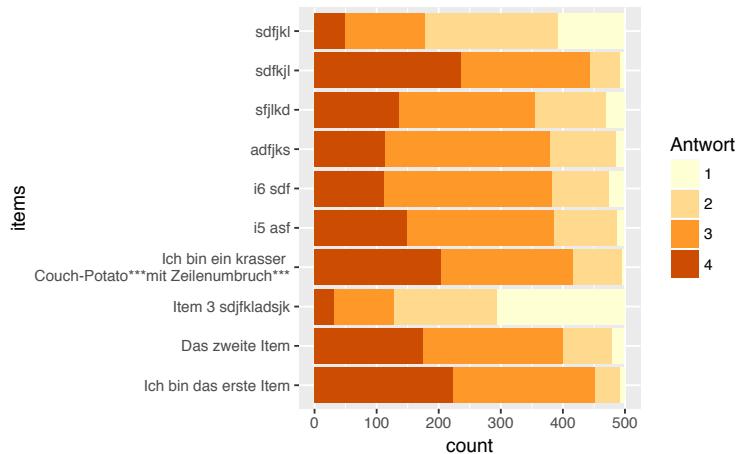
```
ggplot(data = data_long) +
  aes(x = items) +
  geom_bar(aes(fill = Antwort), position = "stack") +
  coord_flip() +
  scale_x_discrete(labels = item_labels)
```



### 0.17.7 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)
```



## 0.18 Verweise

- 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 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/>.



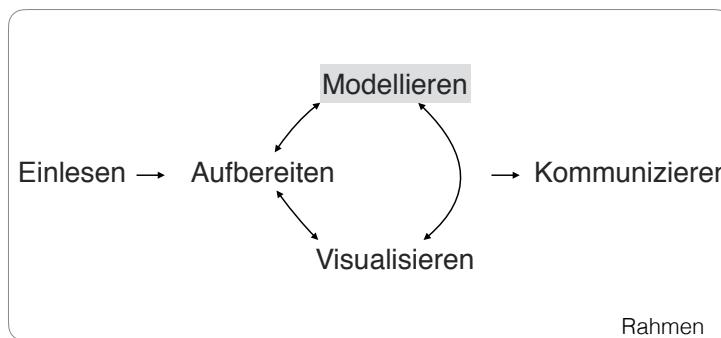
## **II STATISTISCHES MODELLIEREN**



# Grundlagen des Modellierens

In diesem Kapitel benötigte Pakete:

```
library(tidyverse)
require(gridExtra)
```



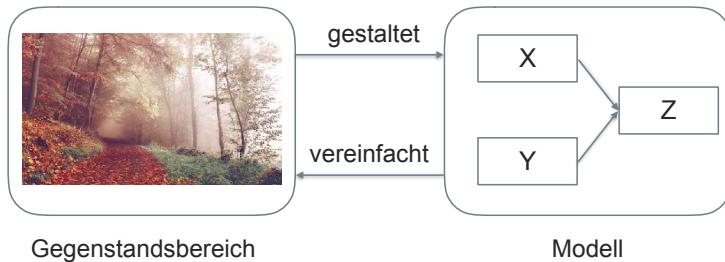
## 0.19 Was ist ein Modell? Was ist Modellieren?

Das Leben ist schwer... oder sagen wir: komplex. Um einen Ausschnitt der Wirklichkeit<sup>19</sup> zu verstehen, erscheint es sinnvoll, sich einige als wesentlich erachteten Aspekte "herauszugreifen" bzw. auszusuchen und sich nur noch deren Zusammenspiel näher anzuschauen.

---

<sup>19</sup>Unter "Wirklichkeit" sei hier ein beliebiges empirisches System vorhanden, mit einer Menge von Objekten  $O$  und einer Menge von Beziehungen (Relationen)  $R$  zwischen den Objekten.

Manche Aspekte der Wirklichkeit sind wirklicher als andere. Interessiert man sich für den Zusammenhang von Temperatur und Grundwasserspiegel, so sind diese Dinge direkt beobachtbar. Interessiert man sich hingegen für Lebensqualität und Zufriedenheit, so muss man diese Untersuchungsgegenstände erst konstruieren. Sprechen wir daher von Wirklichkeit lieber vorsichtiger vom *Gegenstandsbereich*, also den *konstruierten Auszug der Wirklichkeit* für den sich die forschende Person interessiert. Bestenfalls (er)findet man eine *Annäherung* an die Wirklichkeit, schlechterenfalls eine *verzerrte*, gar *grob falsche* Darstellung<sup>20</sup>.



**Figure 12:** Modellieren

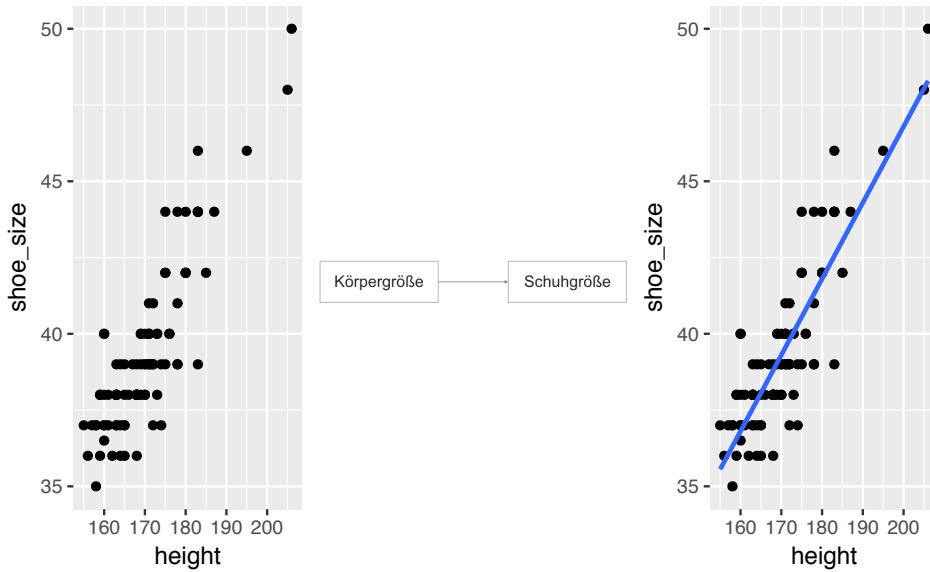
Damit verstehen wir Modellieren als eine typische Aktivität von Menschen (Gigerenzer 1980), genauer eines Menschen mit einem bestimmten Ziel. Wir können gar nicht anders, als nur ein Modell unserer Umwelt zu machen. Vielfältige Medien kommen dazu in Frage: Bilder, Geschichten, Logik, Gleichungen. Wir werden uns hier auf *numerische Modelle* konzentrieren, weil es dort am einfachsten ist, die Informationen herauszuziehen.

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

Im ersten Plot sehen wir - schon übersetzt in eine Datenvisualisierung - den Gegenstandsbereich. Dort sind einige Objekte zusammen mit ihren Relationen abgebildet (Gewicht vs. Körpergröße). Im mittleren Plot sehen wir ein (graphisches) Modell dazu. Noch ist das Modell recht unspezifisch; es wird nur postuliert, dass Körpergröße auf Schuhgröße einen Einfluss hat. Der rechte Plot spezifiziert nun diesen Einfluss: Es wird ein linearer Einfluss (eine Gerade) zwischen Größe und Schuhgröße unterstellt.

---

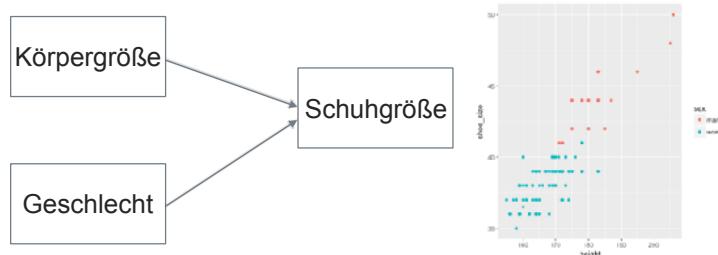
<sup>20</sup>Da keine Wiedergabe der Wirklichkeit perfekt ist, sind streng genommen alle Modelle „falsch“ in diesem Sinne.



**Figure 13:** Ein Beispiel für Modellieren

Bei einem linearen Modell ist der Zuwachs der Ausgabevariablen konstant. Steigt eine Eingabevariable X um k, so steigt die Ausgabevariable ebenfalls um einen  $b^*k$ , unabhängig vom Wert von X.

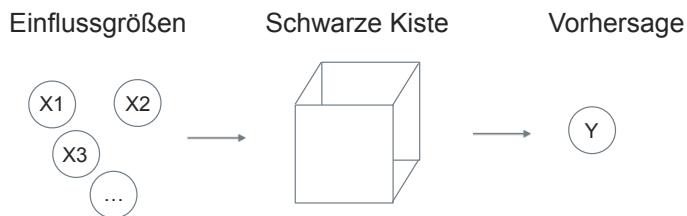
Ein etwas aufwändigeres Modell könnte so aussehen:



**Figure 14:** Ein etwas aufwändigeres Modell

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*. Modelle, wie wir sie betrachten werden, postulieren eine quantifizierbaren Zusammenhang zwis-

chen diesen beiden Arten von Größen. Wir gehen dabei nicht davon aus, dass unsere Modelle perfekt sind, sondern dass Fehler passieren. Damit lassen sich unsere Modelle in drei Aspekte gliedern.



**Figure 15:** Modelle mit schwarzer Kiste

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”. Mathematischer ausgedrückt:

$$Y = f(X) + \epsilon$$

Hier stehen  $Y$  für das Kriterium,  $X$  für den oder die Prädiktoren,  $f$  für die “schwarze Kiste” und  $\epsilon$  für den Fehler, den wir bei unserer Vorhersage begehen. Die schwarze Kiste könnte man auch als die “datengenerierende Maschine” 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*; 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.

## 0.20 Ziele des Modellierens

Man kann drei Arten von Zielen abgrenzen: Vorhersagen, Erklären und Reduzieren.

- *Vorhersagen* hat das Ziel, eine geschickte Black Box zu wählen (oder eine Black Box geschickt zu wählen), so dass der Vohersagefehler möglichst klein ist. Sicherlich wird der Vorhersagefehler nie Null sein. Das Innenleben der “schwarzen Kiste” interessiert uns hier nicht.
- *Erklären* bedeutet, zu verstehen, *wie* oder *warum* sich der Kriteriumswert so verändert, wie er es tut. Auf *welche Art* werden die Prädiktoren verrechnet, so dass eine bestimmter Kriteriumswert resultiert? Welche Prädiktoren 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.
- *Reduzieren* meint, dass man die Fülle des Materials verringert, in dem man ähnliche Dinge zusammenfasst. Dabei kann man sowohl Observationen zusammen fassen (“Britta”, “Carla” und “Dina” zu “Frau” und “Joachim”, “Alois” und “Casper” zu “Mann”) oder auch Variablen zusammen fassen (“Frage 1”, “Frage 2” und “Frage 3” zu “Markentreue” etc.).

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 15 gibt diesen Fall wieder. Soll dem gegenüber das Ziel aber sein, die Daten zu reduzieren, also z.B. Kunden nach Persönlichkeit zu gruppieren, so ist die Lage anders. 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 in Dunkeln,

was die “Warheit” 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*).

### 0.20.1 Modelle zur Erklärung vs. Modelle zur Vorhersage

Man kann statistische Modelle danach unterscheiden, ob sie sich das Innenleben der schwarzen Kiste interessieren oder nicht. Manchmal will man einfach nur (möglichst) gute Vorhersagen treffen; in anderen Situationen will man erklären, warum sich “die Natur” so verhält, wie sie es tut.

Erklären wir das Innenleben der schwarzen Kiste genau, so spricht man von einem *Modell zur Erklärung*; spezifizieren wir das Innenleben der schwarzen Kiste nicht oder weniger genau sondern fokussieren und auf die Vorhersagegüte, so kann man von einem *Modell zur Vorhersage* sprechen.

Ein Beispiel für ein einfaches “erklärendes Modell” ist:

Für je 5kg Körpergewicht steigt die Schuhgröße um 1 Nummer, wobei 50kg bei Schuhgröße 36 aufgehängt ist.

Was ist hier die Erklärung? Die Erklärung liegt darin, dass wir eine mathematische Funktion unterstellen. Diese mathematische Funktion soll “die Natur”, d.h. den Gegenstandsbereich, widerspiegeln, so unsere Hoffnung.

Ein Beispiel für ein nicht-erklärenderes Modell, das sich nur für Vorhersagegüte interessiert, ist:

Schau mal, was die Schuhgröße von Leuten mit ähnlicher Größe ist. Bilde den Mittelwert dieser Schuhgrößen. Gib diesen Mittelwert als Schätzwert an.

Dieses Modell liefert keine “Erklärung”, in dem Sinn, dass eine einfache mathematische Funktion, Eingabe- und Ausgabegröße verbindet. Es werden lediglich Vorhersagen getroffen.

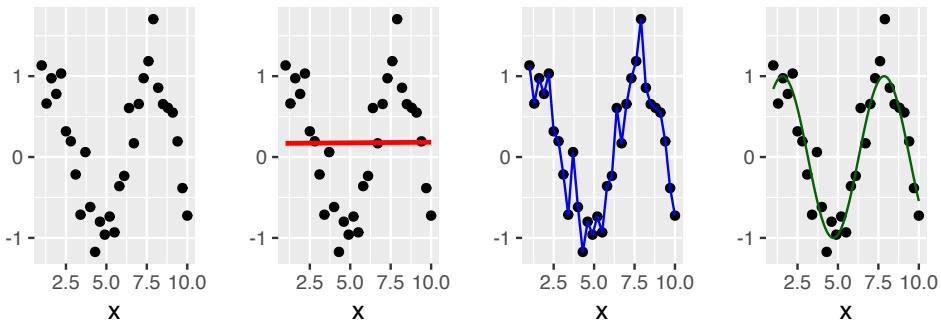
## 0.21 Wann welches Modell?

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.

## 0.22 Einfache vs. komplexe Modelle: Unter- vs. Überanpassung

Je komplexer ein Modell, desto besser passt sie meistens auf den Gegenstandsbereich. Eine grobe, holzschnittsartige 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.



**Figure 16:** Welches Modell passt am besten zu diesen Daten?

Der 1. Plot (links) zeigt den Datensatz ohne Modell; der 2. Plot legt ein

lineares Modell (rote Gerade) in die Daten. Der 3. Plot zeigt ein Modell, welches die Daten exakt erklärt - die (blaue) Linie geht durch alle Punkte. Der 4. Plot zeigt ein Modell (grüne Linie), welches die Punkte gut beschreibt, aber nicht exakt trifft.

Welchem Modell würden Sie (am meisten) vertrauen? Das “blaue” Modell beschreibt die Daten sehr gut, aber hat das Modell überhaupt eine “Idee” vom Gegenstandsbereich, eine “Ahnung”, wie Y und X zusammenhängen, bzw. wie X einen Einfluss auf Y ausübt? Offenbar nicht. Das Modell ist “übergenau” oder zu komplex. Man spricht von *Überanpassung* (engl. *overfitting*). Das Modell scheint zufälliges, bedeutungsloses Rauschen zu ernst zu nehmen. Das Resultat ist eine zu wackelige Linie - ein schlechtes Modell, da wir wenig Anleitung haben, auf welche Y-Werte wir tippen müssten, wenn wir neue, unbekannte X-Werte bekämen.

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 “Punktbewegung” 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 erklären<sup>21</sup>.

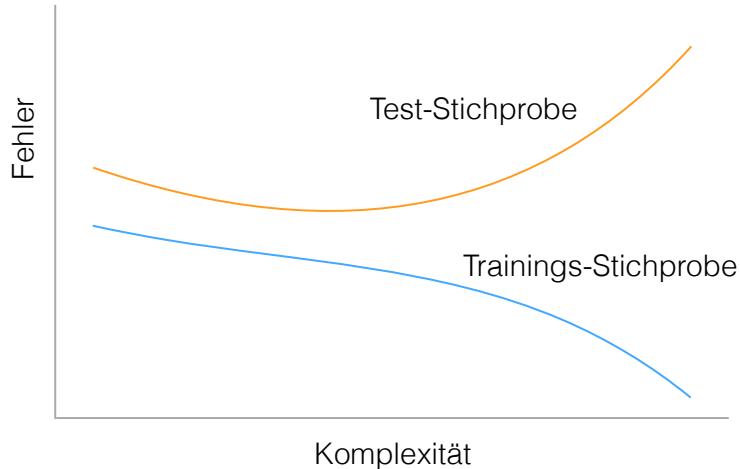
Je komplexer ein Modell ist, desto besser beschreibt es einen bekannten Datensatz (Trainings-Stichprobe). Allerdings ist das Modell, welches den Trainings-Datensatz am besten beschreibt, nicht zwangsläufig das Modell, welches neue, unbekannte Daten am besten beschreibt. Oft im Gegenteil!

Je komplexer das Modell, desto kleiner der Fehler im Trainings-Datensatz. Allerdings: Die Fehler-Kurve im *Test*-Datensatz ist *U-förmig*: Mit steigen-

---

<sup>21</sup>Tatsächlich wurden die Y-Werte als Sinus-Funktion plus etwas normalverteiltes Rauschen simuliert.

der Komplexität wird der Fehler einige Zeit lang kleiner; ab einer gewissen Komplexität steigt der Fehler im Test-Datensatz wieder!



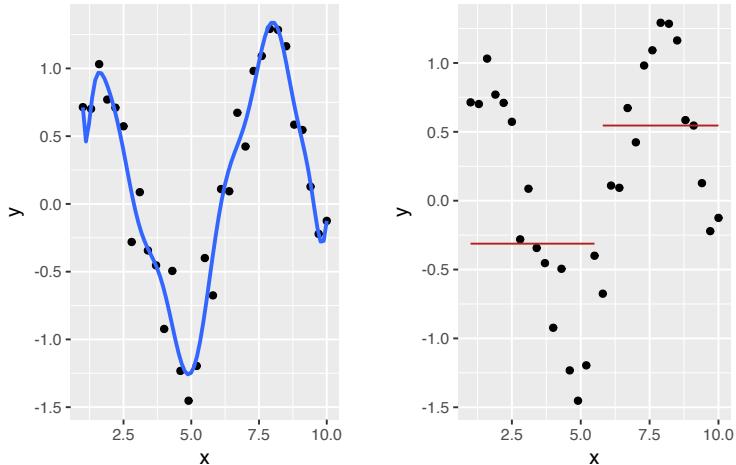
## 0.23 Bias-Varianz-Abwägung

Einfache Modelle bilden (oft) wesentliche Aspekte des Gegenstandsbereich nicht ab; die Wirklichkeit ist komplex. 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<sup>22</sup>.



Der linke Plot zeigt ein komplexes Modell<sup>23</sup>; 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 Daten werden vergleichweise wenig Einfluss auf das Modell (die beiden roten Linien) haben.

## 0.24 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

---

<sup>22</sup>Basierend auf (Kuhn and Johnson 2013)

<sup>23</sup>Genauer gesagt ein Polynom vom Grad 15.

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: @ref(overfitting\_casestudy) **Fallstudie zu Overfitting**).

```
set.seed(42)
train <- wo_men %>%
  sample_frac(.8, replace = FALSE)  # Stichprobe von 80%, ohne Zurücklegen

test <- wo_men %>%
  anti_join(train)  # Alle Zeilen von "wo_men", 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.

## 0.25 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 gut 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 Vorhersagen<sup>24</sup>. 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 angefallende 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 überraschen. 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 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.

---

<sup>24</sup>Gerade wenn sie die Zukunft betreffen; ein Bonmot, das Yogi Berra nachgesagt wird.

## 0.26 Auswahl von Prädiktoren

Es gibt drei Möglichkeiten, die Variablen zu bestimmen: theoriegeleitet, datengetrieben oder auf Gut glück.

## 0.27 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/>.



# Inferenzstatistik

## 0.28 Der p-Wert



**Figure 17:** Der größte Statistiker des 20. Jahrhunderts ( $p < .05$ )

Der p-Wert ist die heilige Kuh der Forschung. 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 wird viel Bedeutung zugemessen.

Was sagt uns der p-Wert? Eine gute intuitive Definition ist:

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

Je größer  $p$ , desto besser passen die Daten zur Nullhypothese.

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 ihm missverstehen. Auf der anderen Seite finde ich, dass sich Technologien dem Nutzer anpassen sollten (soweit als möglich) und nicht umgekehrt. Die (genaue) Definition des p-Werts ist aber auch so kompliziert, man kann sie leicht missverstehen:

Der p-Wert gibt die Wahrscheinlichkeit  $P$  unserer Daten  $D$  an (und noch extremerer), unter der Annahme, dass die getestete

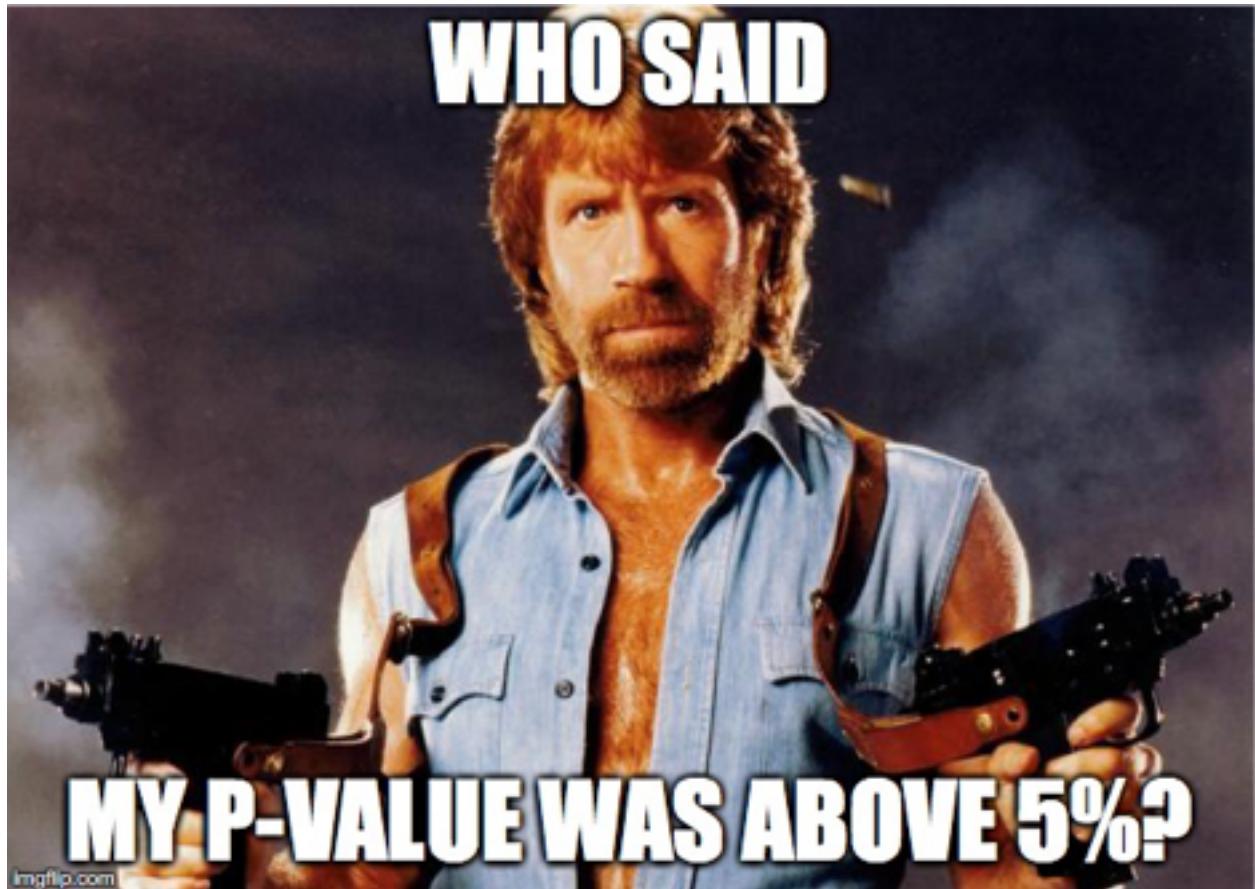


Figure 18: Der p-Wert wird oft als wichtig erachtet

Hypothese H wahr ist (und wenn wir den Versuch unendlich oft wiederholen würden, unter identischen Bedingungen und ansonsten zufällig).  $p = P(D|H)$

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:

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

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 et al. 2015, Wicherts et al. (2016)); andere statistische Verfahren können auch gehackt werden.

Ein wichtiger Anklagepunkt lautet, dass der p-Wert nicht nur eine Funktion der Effektgröße ist, sondern auch der Stichprobengröße. Sprich: Bei großen Stichproben wird jede Hypothese signifikant. Damit verliert der p-Wert an Nützlichkeit. Die Verteitung argumentiert hier, dass das “kein Bug, sondern ein Feature” sei: Wenn man z.B. die Hypothese prüfe, dass der Gewichtsunteschied 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.

Meine Meinung ist, dass der p-Wert ein problematisch ist (und ein Dinosaurier) und nicht oder weniger benutzt werden sollte (das ist eine normative Aussage). 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.

**WENN DER P-WERT  
EIN DINOSAURIER IST,**



**IST DER T-REX  
DANN EIN P-WERT?**

# **III GELEITETES MODELLIEREN**



# Klassische lineare (numerische) Regression

Benötigte Pakete:

```
library(caret) # Modellieren
library(tidyverse) # Datenjudo, Visualisierung, ...
library(gridExtra) # Mehrere Plots kombinieren
```

## 0.29 Einfache Regression

Wir werden weiter den Datensatz *tips* analysieren (Bryant and Smith 1995).

Sofern noch nicht geschehen, können Sie in hier<sup>25</sup> als csv-Datei herunterladen:

```
tips <- read.csv("https://sebastiansauer.github.io/data/tips.csv")
```

Zur Unterstützung der Analyse wird (wieder) das Paket `mosaic` verwendet; außerdem laden wir `ggplot2` für `qplot`:

```
library(mosaic)
library(ggplot2)
```

---

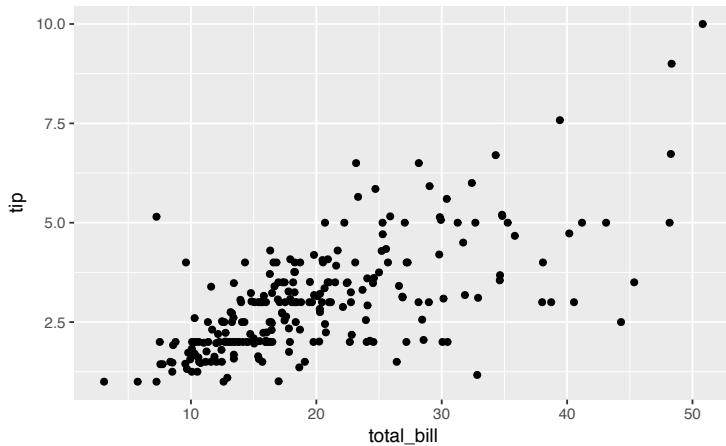
<sup>25</sup><https://goo.gl/whKjn1>

Wie hängen Trinkgeldhöhe `tip` und Rechnungshöhe `total_bill` zusammen?  
 Kann die Höhe des Trinkgeldes als *lineare* Funktion der Rechnungshöhe linear modelliert werden?

$$tip_i = \beta_0 + \beta_1 \cdot total\_bill_i + \epsilon_i$$

Zunächst eine visuelle Analyse mi Hilfe eines Scatterplots.

```
qplot(y = tip, x = total_bill, data = tips)
```



Es scheint einen positiven Zusammenhang zu geben. Modellieren wir die **abhängige** Variable `tip` (inhaltliche Entscheidung!) als lineare Funktion der **unabhängigen** Variable `total_bill`:

```
LinMod.1 <- lm(tip ~ total_bill, data=tips)
summary(LinMod.1)
#>
#> Call:
#> lm(formula = tip ~ total_bill, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -3.198 -0.565 -0.097  0.486  3.743
#>
```

```
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 0.92027   0.15973   5.76  2.5e-08 ***
#> total_bill   0.10502   0.00736  14.26 < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.02 on 242 degrees of freedom
#> Multiple R-squared:  0.457, Adjusted R-squared:  0.454
#> F-statistic: 203 on 1 and 242 DF, p-value: <2e-16
```

Der Achsenabschnitt (`intercept`) wird mit 0.92 geschätzt, die Steigung in Richtung `total_bill` mit 0.11: steigt `total_bill` um einen Dollar, steigt im Durchschnitt `tip` um 0.11.

Die (Punkt-)Prognose für `tip` lautet also

$$\text{tip} = 0.92 + 0.11 * \text{total\_bill}$$

Die Koeffizienten werden dabei so geschätzt, dass  $\sum \epsilon_i^2$  minimiert wird. Dies wird auch als *Kleinste Quadrate (Ordinary Least Squares, OLS)* Kriterium bezeichnet. Eine robuste Regression ist z. B. mit der Funktion `r1m()` aus dem Paket `MASS` möglich.

In mosaic kann ein solches Modell einfach als neue Funktion definiert werden:

```
LinMod.1Fun <- makeFun(LinMod.1)
```

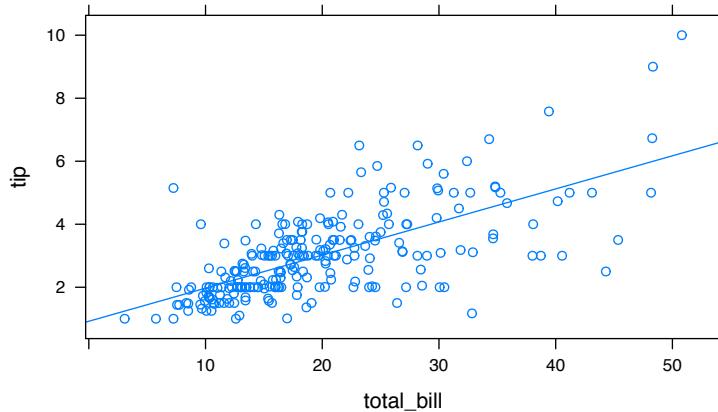
Die (Punkt-)Prognose für die Trinkgeldhöhe, bspw. für eine Rechnung von 30\$ kann dann berechnet werden

```
LinMod.1Fun(total_bill=30)
#>     1
#> 4.07
```

also 4.07\$.

In mosaic kann die Modellgerade über

```
plotModel(LinMod.1)
```



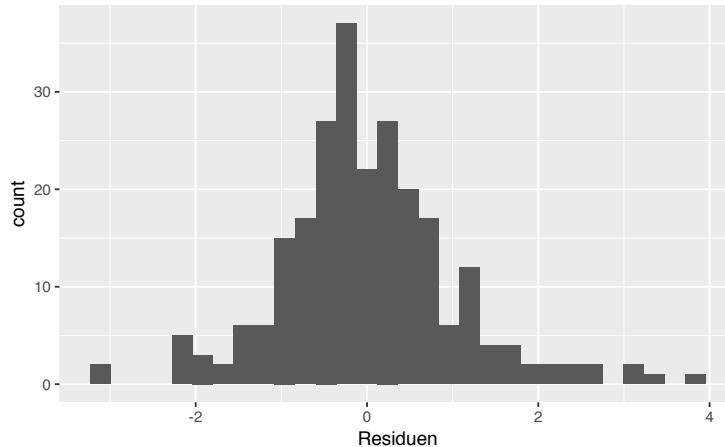
betrachtet werden. Das Bestimmtheitsmaß  $R^2$  ist mit 0.46 “ok”: 46-% der Variation des Trinkgeldes wird im Modell erklärt.

## 0.30 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”.
- Zur Überprüfung der Normalverteilung der Residuen zeichnen wir ein Histogramm. Die *Residuen* können über den Befehl `resid()` aus einem Linearen Modell extrahiert werden. Hier scheint es zu passen:

```
resid_df <- data.frame(Residuen = resid(LinMod.1))
qplot(x = Residuen, data = resid_df)
```

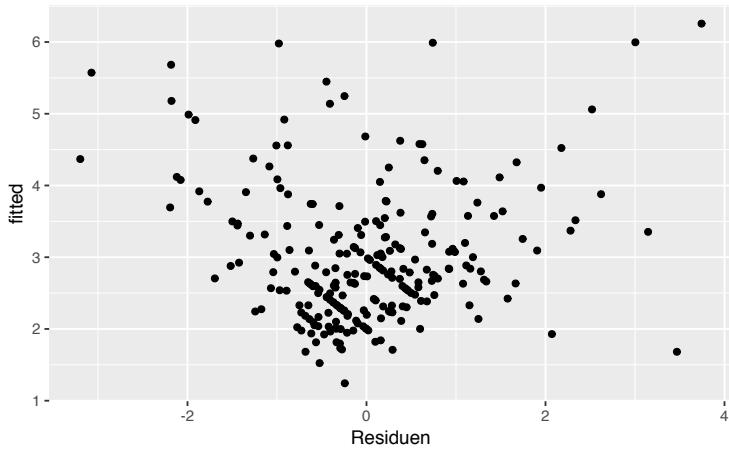


- *Konstante Varianz*: Dies kann z. B. mit einem Scatterplot der Residuen auf der y-Achse und den angepassten Werten auf der x-Achse überprüft werden. Die angepassten (geschätzten) Werte werden über den Befehl `fitted()`<sup>26</sup> extrahiert. Diese Annahme scheint verletzt zu sein (siehe unten): je größer die Prognose des Trinkgeldes, desto größer wirkt die Streuung der Residuen. Dieses Phänomen ließ sich schon aus dem ursprünglichen Scatterplot `qplot(x = tip, y = total_bill, data=tips)` erahnen. Das ist auch inhaltlich plausibel: je höher die Rechnung, desto höher die Varianz beim Trinkgeld. Die Verletzung dieser Annahme beeinflusst *nicht* die Schätzung der Steigung, sondern die Schätzung des Standardfehlers, also des p-Wertes des Hypothesentests, d. h.,  $H_0 : \beta_1 = 0$ .

```
resid_df$fitted <- fitted(LinMod.1)
qplot(x = Residuen, y = fitted, data = resid_df)
```

---

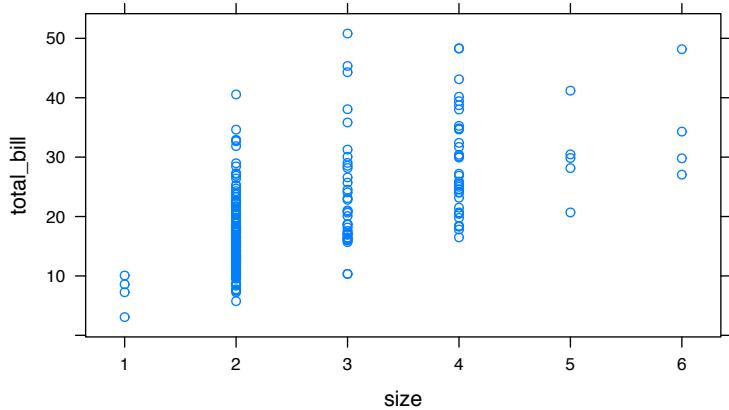
<sup>26</sup>Cleveland fände diese Idee nicht so gut.



- *Extreme Ausreißer:* Wie am Plot der Linearen Regression `plotModel(LinMod.1)` erkennbar, gibt es vereinzelt Ausreißer nach oben, allerdings ohne einen extremen Hebel.

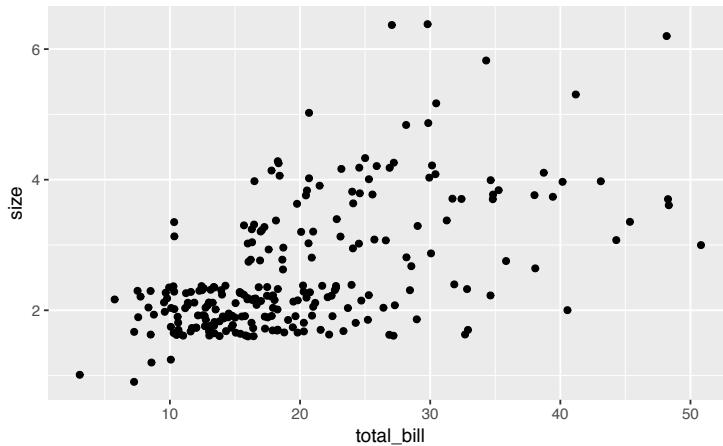
Hängt die Rechnungshöhe von der Anzahl der Personen ab? Bestimmt, aber wie?

```
xyplot(total_bill ~ size, data=tips)
```



Da bei diskreten metrischen Variablen (hier `size`) Punkte übereinander liegen können, sollte man “jittern” (“schütteln”), d. h., eine (kleine) Zufallszahl addieren:

```
qplot(x = total_bill, y = size, data = tips, geom = "jitter")
```



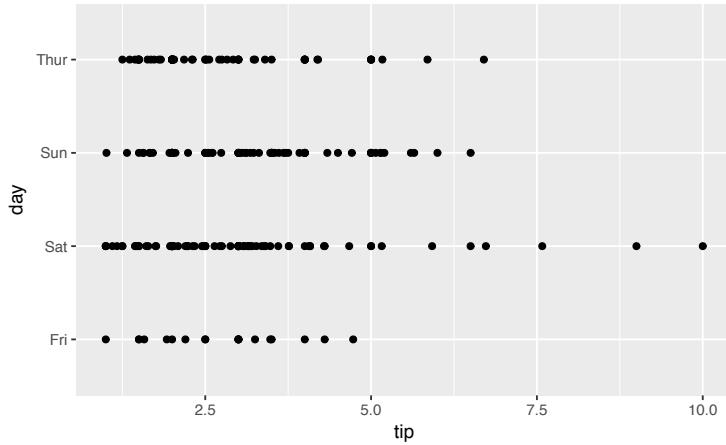
1. Um wie viel Dollar steigt im Durchschnitt das Trinkgeld, wenn eine Person mehr am Tisch sitzt?
2. Für wie aussagekräftig halten Sie Ihr Ergebnis aus 1.?

## 0.31 Regression mit kategorialen Prädiktoren

Der Wochentag `day` ist eine kategoriale Variable. Wie sieht eine Regression des Trinkgeldes darauf aus?

Zunächst grafisch:

```
qplot(x = tip,y = day, data=tips)
```



Und als Lineares Modell:

```

LinMod.2 <- lm(tip ~ day, data=tips)
summary(LinMod.2)

#>
#> Call:
#> lm(formula = tip ~ day, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -2.245 -0.993 -0.235  0.538  7.007
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 2.7347    0.3161   8.65  7.5e-16 ***
#> daySat       0.2584    0.3489   0.74   0.46
#> daySun       0.5204    0.3534   1.47   0.14
#> dayThur      0.0367    0.3613   0.10   0.92
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.38 on 240 degrees of freedom
#> Multiple R-squared:  0.0205, Adjusted R-squared:  0.00823
#> F-statistic: 1.67 on 3 and 240 DF,  p-value: 0.174

```

Die im Modell angegebenen Schätzwerte sind die Änderung der Trinkgeldprognose, wenn z. B. der Tag ein Samstag (`daySat`) im Vergleich zu einer Referenzkategorie. Dies ist in R das erste Element des Vektors der Faktorlevel. Welcher dies ist ist über den Befehl `levels()` zu erfahren

```
levels(tips$day)
#> [1] "Fri"   "Sat"   "Sun"   "Thur"
```

hier also Fri (aufgrund der standardmäßig aufsteigenden alphanumerischen Sortierung). Dies kann über `relevel()` geändert werden. Soll z. B. die Referenz der Donnerstag, Thur sein:

```
tips$day <- relevel(tips$day, ref = "Thur")
levels(tips$day)
#> [1] "Thur"  "Fri"   "Sat"   "Sun"
```

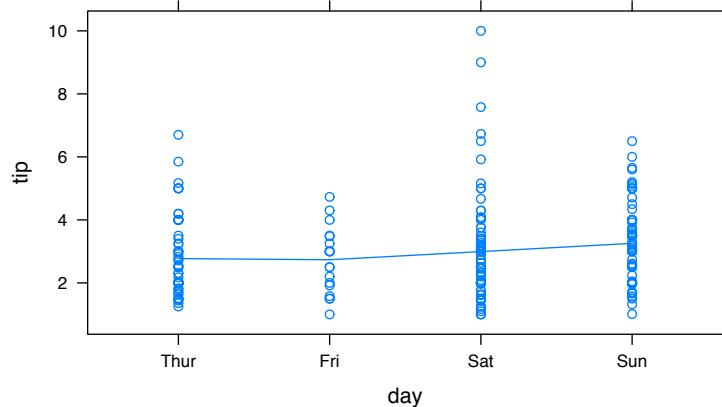
Das Modell ändert sich entsprechend:

```
LinMod.3 <- lm(tip ~ day, data=tips)
summary(LinMod.3)
#>
#> Call:
#> lm(formula = tip ~ day, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -2.245 -0.993 -0.235  0.538  7.007
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  2.7715    0.1750  15.84  <2e-16 ***
#> dayFri      -0.0367    0.3613  -0.10   0.919
#> daySat       0.2217    0.2290   0.97   0.334
#> daySun       0.4837    0.2358   2.05   0.041 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#>
#> Residual standard error: 1.38 on 240 degrees of freedom
#> Multiple R-squared:  0.0205, Adjusted R-squared:  0.00823
#> F-statistic: 1.67 on 3 and 240 DF,  p-value: 0.174
```

sowie als Plot:

```
plotModel(LinMod.3)
```



Eine Alternative zu `relevel()` zur Bestimmung der Referenzkategorie ist es, innerhalb von `factor()` die Option `levels=` direkt in der gewünschten Sortierung zu setzen.

```
day <- factor(tips$day, levels=c("Thur", "Fri", "Sat", "Sun"))
```

Die (Punkt-)Prognose für die Trinkgeldhöhe, bspw. an einen Freitag kann dann berechnet werden

```
LinMod.3Fun <- makeFun(LinMod.3)
LinMod.3Fun(day="Fri")
#>      1
#> 2.73
```



3. Wie verändert sich die Rechnungshöhe im Durchschnitt, wenn die Essenszeit Dinner statt Lunch ist?
4. Wie viel % der Variation der Rechnungshöhe können Sie durch die Essenszeit modellieren?

## 0.32 Multiple Regression

Aber wie wirken sich die Einflussgrößen *zusammen* auf das Trinkgeld aus?

```
LinMod.4 <- lm(tip ~ total_bill + size + sex + smoker + day + time, data=tips)
summary(LinMod.4)

#>
#> Call:
#> lm(formula = tip ~ total_bill + size + sex + smoker + day + time,
#>      data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -2.848 -0.573 -0.103  0.476  4.108
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  0.6416    0.4976   1.29   0.199
#> total_bill   0.0945    0.0096   9.84 <2e-16 ***
#> size         0.1760    0.0895   1.97   0.051 .
#> sexMale     -0.0324    0.1416  -0.23   0.819
#> smokerYes   -0.0864    0.1466  -0.59   0.556
#> dayFri       0.1623    0.3934   0.41   0.680
#> daySat       0.0408    0.4706   0.09   0.931
#> daySun       0.1368    0.4717   0.29   0.772
#> timeLunch    0.0681    0.4446   0.15   0.878
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

```
#> Residual standard error: 1.02 on 235 degrees of freedom
#> Multiple R-squared:  0.47,  Adjusted R-squared:  0.452
#> F-statistic: 26.1 on 8 and 235 DF,  p-value: <2e-16
```

Interessant sind die negativen Vorzeichen vor den Schätzwerten für `sexMale` und `smokerYes` – anscheinend geben Männer und Raucher weniger Trinkgeld, wenn alle anderen Faktoren konstant bleiben. Bei einer rein univariaten Betrachtung wäre etwas anderes herausgekommen.

```
summary(lm(tip ~ sex, data=tips))
#>
#> Call:
#> lm(formula = tip ~ sex, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -2.090 -1.090 -0.090  0.667  6.910
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  2.833     0.148   19.14  <2e-16 ***
#> sexMale      0.256     0.185    1.39    0.17
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.38 on 242 degrees of freedom
#> Multiple R-squared:  0.0079, Adjusted R-squared:  0.0038
#> F-statistic: 1.93 on 1 and 242 DF,  p-value: 0.166
summary(lm(tip ~ smoker, data=tips))
#>
#> Call:
#> lm(formula = tip ~ smoker, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -2.009 -0.994 -0.100  0.558  6.991
```

```
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 2.9919     0.1128   26.52 <2e-16 ***
#> smokerYes   0.0169     0.1828    0.09    0.93
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.39 on 242 degrees of freedom
#> Multiple R-squared:  3.51e-05, Adjusted R-squared: -0.0041
#> F-statistic: 0.00851 on 1 and 242 DF, p-value: 0.927
```

Diese *Umkehrung* des modellierten Effektes liegt daran, dass es auch einen positiven Zusammenhang zur Rechnungshöhe gibt:

```
summary(lm(total_bill ~ sex, data=tips))
#>
#> Call:
#> lm(formula = total_bill ~ sex, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -14.99 -6.02 -1.94  3.99 30.07
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 18.057     0.946   19.08 <2e-16 ***
#> sexMale      2.687     1.180    2.28   0.024 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 8.83 on 242 degrees of freedom
#> Multiple R-squared:  0.021, Adjusted R-squared:  0.0169
#> F-statistic: 5.19 on 1 and 242 DF, p-value: 0.0236
summary(lm(total_bill ~ smoker, data=tips))
#>
```

```

#> Call:
#> lm(formula = total_bill ~ smoker, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -17.69  -6.46  -1.89   4.58  30.05
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 19.188     0.723   26.53 <2e-16 ***
#> smokerYes    1.568     1.172    1.34    0.18
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 8.89 on 242 degrees of freedom
#> Multiple R-squared:  0.00735, Adjusted R-squared:  0.00325
#> F-statistic: 1.79 on 1 and 242 DF, p-value: 0.182

```

Im vollem Modell LinMod.4 sind alle unabhängigen Variablen berücksichtigt, die Koeffizienten beziehen sich dann immer auf: gegeben, die anderen Variablen bleiben konstant, d. h. ceteris paribus.

Vergleichen wir mal zwei Modelle:

```

LinMod.5a <- lm(tip ~ sex, data=tips)
coef(LinMod.5a) # Koeffizienten extrahieren
#> (Intercept)      sexMale
#>           2.833      0.256
LinMod.5b <- lm(tip ~ sex + total_bill, data=tips)
coef(LinMod.5b) # Koeffizienten extrahieren
#> (Intercept)      sexMale  total_bill
#>          0.9333     -0.0266     0.1052

```

Ohne die Berücksichtigung der **Kovariable/Störvariable** Rechnungshöhe geben Male ein um im Durchschnitt 0.26 *höheres* Trinkgeld, bei Kontrolle, d. h. gleicher Rechnungshöhe ein um 0.03 *niedrigeres* Trinkgeld als die Referenzklasse Fe-

male (levels(tips\$sex)[1]).

## 0.33 Inferenz in der linearen Regression

Kehren wir noch einmal zur multivariaten Regression (LinMod.4) zurück.

```
summary(LinMod.4)
#>
#> Call:
#> lm(formula = tip ~ total_bill + size + sex + smoker + day + time,
#>      data = tips)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -2.848 -0.573 -0.103  0.476  4.108
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 0.6416    0.4976   1.29   0.199
#> total_bill   0.0945    0.0096   9.84 <2e-16 ***
#> size         0.1760    0.0895   1.97   0.051 .
#> sexMale     -0.0324    0.1416  -0.23   0.819
#> smokerYes   -0.0864    0.1466  -0.59   0.556
#> dayFri       0.1623    0.3934   0.41   0.680
#> daySat       0.0408    0.4706   0.09   0.931
#> daySun       0.1368    0.4717   0.29   0.772
#> timeLunch    0.0681    0.4446   0.15   0.878
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.02 on 235 degrees of freedom
#> Multiple R-squared:  0.47, Adjusted R-squared:  0.452
#> F-statistic: 26.1 on 8 and 235 DF, p-value: <2e-16
```

In der 4. Spalte der, mit Zeilennamen versehenen Tabelle **Coefficients** stehen die p-Werte der Nullhypothese, die unabhängige Variable hat, gegeben

alle anderen Variablen im Modell, keinen linearen Einfluss auf die abhängige Variable:  $H_0 : \beta_i = 0$ . Zur Bestimmung des p-Wertes wird der Schätzer (**Estimate**) durch den Standardfehler (**Std. Error**) dividiert. Der resultierende t-Wert (**t value**) wird dann, zusammen mit der Anzahl an Freiheitsgraden zur Berechnung des p-Wertes (**Pr(>|t|)**) verwendet. Ein einfacher t-Test!

Zur schnelleren Übersicht finden sich dahinter “Sternchen” und “Punkte”, die die entsprechenden Signifikanzniveaus symbolisieren: \*\*\* bedeutet eine Irrtumswahrscheinlichkeit, Wahrscheinlichkeit für Fehler 1. Art, von unter 0.001, d. h. unter 0,1%. \*\* entsprechend 1%, \* 5% und . 10%.

Zum Signifikanzniveau von 10% sind hier also zwei Faktoren und der Achsenabschnitt ((**Intercept**)) signifikant – nicht notwendigerweise relevant: Rechnungshöhe **total\_bill** sowie Anzahl Personen **size**. Beides wirkt sich linear positiv auf die Trinkgeldhöhe aus: Mit jedem Dollar Rechnungshöhe steigt im Mittelwert die Trinkgeldhöhe um 0.09 Dollar, mit jeder Person um 0.18 Dollar – gegeben alle anderen Faktoren bleiben konstant. Das Bestimmtheitsmaß **R<sup>2</sup>** (**Multiple R-squared**:) liegt bei 0.47, also 47% der Variation des Trinkgeldes wird im Modell erklärt.

Außerdem wird getestet, ob alle Koeffizienten der unabhängigen Variablen gleich Null sind:

$$H_0 : \beta_1 = \beta_2 = \dots = \beta_k = 0$$

Das Ergebnis des zugrundeliegenden F-Tests (vgl. Varianzanalyse) wird in der letzten Zeile angegeben (**F-Statistic**). Hier wird  $H_0$  also verworfen.

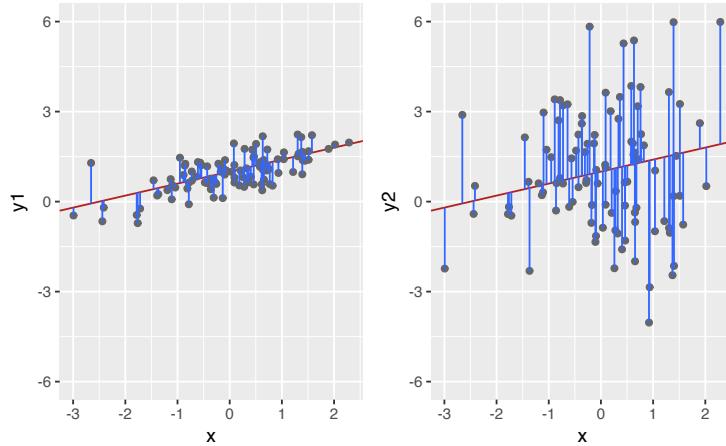
## 0.34 Modellgüte bei Regressionsmodellen

In einem Regressionsmodell lautet die grundlegenden Überlegung zur Modellgüte so:

Wie groß ist der Unterschied zwischen Vorhersage und Wirklichkeit?

Die Größe des Unterschieds (Differenz, “Delta”) zwischen vorhergesagten (geschätzten) Wert und Wirklichkeit, bezeichnet man als *Fehler*, *Residuum* oder Vohersagefehler, häufig mit  $\epsilon$  (griechisch e wie “error”) abgekürzt.

Graphisch kann man das gut veranschaulichen:



Betrachten Sie die beiden Plots. Die rote Linie gibt die vorhergesagten (geschätzten) Werte wieder; die Punkte die beobachteten (“echten”) Werte. Je länger die blauen Linien, desto größer die Vorhersagefehler.

Je 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.

Leider ist es nicht immer einfach zu sagen, wie groß der Fehler sein muss, damit das Modell als “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 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.

### 0.34.1 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.

Übrigens: Der RMSE hat eine Reihe von wünschenswerten statistischen Eigenschaften, über die wir uns hier ausschweigen

### 0.34.2 R-Quadrat ( $R^2$ )

$R^2$ , auch Bestimmtheitsmaß oder Determinationskoeffizient genannt, gibt die Vorhersagegüte im Verhältnis zu einem “Nullmodell” an. 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!”.

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:

$$R^2 = 1 - \left( \frac{\text{Nullmodellfehler} - \text{Vorhersagefehler}}{\text{Nullmodellfehler}} \right)$$

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, die diese Berechnung für uns besorgen:

```
library(caret)
postResample(obs = obs, pred = pred)
```

Hier steht `obs` für beobachtete Werte und `pred` für die vorhergesagten Werte. Dieser Befehl gibt sowohl RMSE als auch  $R^2$  wieder.

### 0.34.3 Likelihood and Friends

Der *Likelihood L* beantwortet folgende Frage:

Angenommen, ein Modell M ist wahr. Wie wahrscheinlich ist es dann, die Daten D zu beobachten?

Zum Beispiel: Eine faire Münze wird 10 Mal geworfen (Modell M: faire Münze). Wie wahrscheinlich ist es, 10 Mal Zahl zu werfen? Die Wahrscheinlichkeit hierfür liegt bei ca. 0.1%. Der Likelihood wäre also hier ~0.1%.

Bei komplexen Modellen kann der Likelihood sehr klein werden. Damit haben Computer Probleme, weil z.B. nur eine begrenzte Anzahl von Dezimalen berücksichtigt werden. Werden zuviele Dezimalstellen gerundet, kann es das Ergebnis verfälschen. Daher wird der Likelihood häufig logarithmiert; man spricht dann vom *log Likelihood*. Der Logarithmus von einer positiven, sehr kleine Zahl ist eine negative Zahl mit großen Absolutwert. Man verwendet meist den natürlichen Logarithmus, wobei das eigentlich keine Rolle spielt. Manchmal dreht man noch das Vorzeichen um, damit der Log Likelihood wieder positiv ist.

Gütekriterien wie AIC, BIC, CAIC oder die Devianz (engl. *deviance*) sind vom Likelihood abgeleitet. Meist wird noch berücksichtigt, wie komplex das Modell ist; komplexe Modelle tun sich leichter als einfache Modelle, die Daten zu erklären. Aber sie könnten die Daten auch "überanpassen". Um die mögliche Scheingenaugkeit komplexerer Modelle auszugleichen, wird der Likelihood vom AIC etc. mit einem Strafwert belegt, der proportional zur Komplexität des Modells ist (Zumel, Mount, and Porzak 2014).



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.

## 0.35 Vertiefungen zum Regressionmodell

### 0.35.1 Modellwahl

Das Modell mit allen Variablen des Datensatzes, d. h., mit 6 unabhängigen (LinMod.4) erklärt 47.01% der Variation, das Modell *nur* mit der Rechnungshöhe als erklärende Variable (LinMod.1) schon 45.66%, der Erklärungszuwachs liegt also gerade einmal bei 1.35 Prozentpunkten. In der Statistik ist die Wahl des *richtigen* Modells eine der größten Herausforderungen, auch deshalb, weil das wahre Modell in der Regel nicht bekannt ist und es schwer ist, die richtige Balance zwischen Einfachheit und Komplexität zu finden. Aufgrund des Zufalls kann es immer passieren, dass das Modell sich zu sehr an die *zufälligen* Daten anpasst (Stichwort: Overfitting). Es gibt unzählige Modellwahlmethoden, und leider garantiert keine, dass immer das beste Modell gefunden wird. Eine Möglichkeit ist die sogenannte Schrittweise-Rückwärtsselektion auf Basis des Akaike-Informationskriteriums (AIC)<sup>27</sup>. Diese ist nicht nur recht weit verbreitet -

---

<sup>27</sup>siehe z. B. Rob J Hyndman & George Athanasopoulos, Forecasting: principles and practice, Kapitel 5.3: Selecting predictors, <https://www.otexts.org/fpp/5/3>

und liefert unter bestimmten Annahmen das “richtige” Modell - sondern in R durch den Befehl `step()` einfach umsetzbar:

```
step(LinMod.4)
#> Start:  AIC=20.5
#> tip ~ total_bill + size + sex + smoker + day + time
#>
#>           Df Sum of Sq RSS   AIC
#> - day        3    0.6 247 15.1
#> - time       1    0.0 247 18.5
#> - sex         1    0.1 247 18.6
#> - smoker      1    0.4 247 18.9
#> <none>          247 20.5
#> - size         1    4.1 251 22.5
#> - total_bill   1 101.6 348 102.7
#>
#> Step:  AIC=15.1
#> tip ~ total_bill + size + sex + smoker + time
#>
#>           Df Sum of Sq RSS   AIC
#> - time       1    0.0 247 13.1
#> - sex         1    0.0 247 13.2
#> - smoker      1    0.4 248 13.5
#> <none>          247 15.1
#> - size         1    4.3 251 17.4
#> - total_bill   1 101.7 349 97.2
#>
#> Step:  AIC=13.1
#> tip ~ total_bill + size + sex + smoker
#>
#>           Df Sum of Sq RSS   AIC
#> - sex         1    0.0 247 11.2
#> - smoker      1    0.4 248 11.5
#> <none>          247 13.1
#> - size         1    4.3 251 15.4
#> - total_bill   1 103.3 350 96.3
#>
```

```

#> Step: AIC=11.2
#> tip ~ total_bill + size + smoker
#>
#>           Df Sum of Sq RSS   AIC
#> - smoker      1     0.4 248  9.5
#> <none>          247 11.2
#> - size        1     4.3 252 13.4
#> - total_bill  1    104.3 351 95.0
#>
#> Step: AIC=9.53
#> tip ~ total_bill + size
#>
#>           Df Sum of Sq RSS   AIC
#> <none>          248  9.5
#> - size        1     5.2 253 12.6
#> - total_bill  1    106.3 354 94.7
#>
#> Call:
#> lm(formula = tip ~ total_bill + size, data = tips)
#>
#> Coefficients:
#> (Intercept)  total_bill       size
#>     0.6689     0.09271    0.1926

```

In den letzten Zeilen der Ausgabe steht das beste Modell, das diese Methode (schrittweise, rückwärts) mit diesem Kriterium (AIC) bei diesen Daten findet (Punktprognose, d. h. ohne Residuum):

```
tip = 0.66894 + 0.09271 * total_bill + 0.19260 * size
```

Der Ausgabe können Sie auch entnehmen, welche Variablen in welcher Reihenfolge *entfernt* wurden: Zunächst `day`, dann `time`, danach `sex` und schließlich `smoker`. Hier sind also dieselben Variablen noch im Modell, die auch in LinMod.4 signifikant zum Niveau 10% waren, eine Auswahl der dort signifikanten Variablen hätte also dasselbe Modell ergeben. Das ist häufig so, aber nicht immer!

## 0.35.2 Interaktionen

Wir haben gesehen, dass es einen Zusammenhang zwischen der Trinkgeldhöhe und der Rechnungshöhe gibt. Vielleicht unterscheidet sich der Zusammenhang je nachdem, ob geraucht wurde, d. h., vielleicht gibt es eine Interaktion (Wechselwirkung). Die kann in `lm` einfach durch ein `*` zwischen den unabhängigen Variablen modelliert werden:

```
LinMod.6 <- lm(tip ~ smoker*total_bill, data = tips)
summary(LinMod.6)

#>
#> Call:
#> lm(formula = tip ~ smoker * total_bill, data = tips)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -2.679 -0.524 -0.120  0.475  4.900
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 0.36007   0.20206   1.78  0.07601 .
#> smokerYes   1.20420   0.31226   3.86  0.00015 ***
#> total_bill   0.13716   0.00968  14.17 < 2e-16 ***
#> smokerYes:total_bill -0.06757   0.01419  -4.76  3.3e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.979 on 240 degrees of freedom
#> Multiple R-squared:  0.506, Adjusted R-squared:  0.5
#> F-statistic: 81.9 on 3 and 240 DF,  p-value: <2e-16
```

Der Schätzwert für die Interaktion steht bei `:`. Hier also: Wenn geraucht wurde, ist die Steigung im Durchschnitt um 6,8 Cent geringer. Aber wenn geraucht wurde, ist die Rechnung im Achsenabschnitt erstmal um 1,20\$ höher (Effekt, *ceteris paribus*). Wer will, kann ausrechnen, ab welcher Rechnungshöhe Rauchertische im Mittelwert lukrativer sind...

Das gleiche Bild (höhere Achsenabschnitt, geringere Steigung) ergibt sich

übrigens bei getrennten Regressionen:

```
lm(tip~total_bill, data=tips, subset = smoker=="Yes")
#>
#> Call:
#> lm(formula = tip ~ total_bill, data = tips, subset = smoker ==
#>      "Yes")
#>
#> Coefficients:
#> (Intercept)  total_bill
#>     1.5643      0.0696
lm(tip~total_bill, data=tips, subset = smoker=="No")
#>
#> Call:
#> lm(formula = tip ~ total_bill, data = tips, subset = smoker ==
#>      "No")
#>
#> Coefficients:
#> (Intercept)  total_bill
#>     0.360      0.137
```

### 0.35.3 Weitere Modellierungsmöglichkeiten

Über das Formelinterface  $y \sim x$  können auch direkt z. B. Polynome modelliert werden. Hier eine quadratische Funktion:

```
summary(lm(tip~I(total_bill^2)+total_bill, data=tips))
#>
#> Call:
#> lm(formula = tip ~ I(total_bill^2) + total_bill, data = tips)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -3.200 -0.559 -0.098  0.484  3.776
#>
```

```
#> Coefficients:
#>                               Estimate Std. Error t value Pr(>|t|)    
#> (Intercept)            8.91e-01   3.47e-01    2.57  0.01078 *  
#> I(total_bill^2)      -5.71e-05   6.02e-04   -0.09  0.92457    
#> total_bill             1.08e-01   3.08e-02    3.51  0.00054 *** 
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.02 on 241 degrees of freedom
#> Multiple R-squared:  0.457, Adjusted R-squared:  0.452 
#> F-statistic: 101 on 2 and 241 DF, p-value: <2e-16
```

D. h., die geschätzte Funktion ist eine “umgedrehte Parabel” (negatives Vorzeichen bei  $I(\text{total\_bill}^2)$ ), bzw. die Funktion ist konkav, die Steigung nimmt ab. Allerdings ist der Effekt nicht signifikant. **Hinweis:** Um zu “rechnen” und nicht beispielsweise Interaktion zu modellieren, geben Sie die Variablen in der Formel in der Funktion  $I()$  (*As Is*) ein.

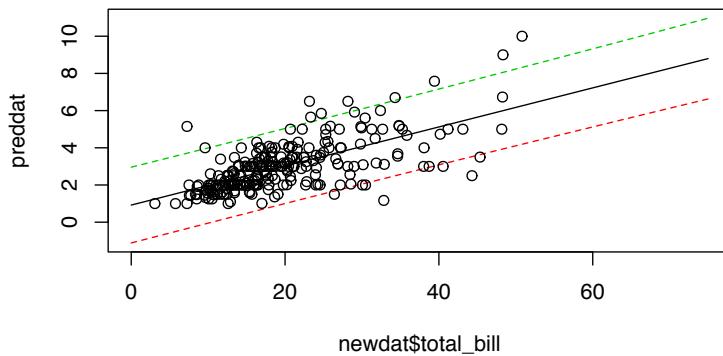
### 0.35.4 Prognoseintervalle

Insgesamt haben wir viel “Unsicherheit” u. a. aufgrund von Variabilität in den Beobachtungen und in den Schätzungen. Wie wirken sich diese auf die Prognose aus?

Dazu können wir über die Funktion `predict.lm` Prognoseintervalle berechnen – hier für das einfache Modell `LinMod.1`:

```
newdat <- data.frame(total_bill = seq(0, 75))
preddat <- predict(LinMod.1, newdata = newdat, interval = "prediction")
head(preddat)
#>   fit   lwr   upr
#> 1 0.92 -1.117 2.96
#> 2 1.03 -1.010 3.06
#> 3 1.13 -0.903 3.16
#> 4 1.24 -0.797 3.27
```

```
#> 5 1.34 -0.690 3.37
#> 6 1.45 -0.583 3.47
tail(predat)
#>    fit lwr upr
#> 71 8.27 6.13 10.4
#> 72 8.38 6.23 10.5
#> 73 8.48 6.33 10.6
#> 74 8.59 6.43 10.7
#> 75 8.69 6.53 10.9
#> 76 8.80 6.63 11.0
matplot(newdat$total_bill, predat, lty = c(1,2,2), type="l")
points(x=tips$total_bill, y=tips$tip)
```



Sie sehen, dass 95% Prognoseintervall ist recht breit: über den gewählten Rechnungsbereich von 0 – 75\$ im Mittelwert bei 4.11\$.

```
favstats((predat[,3]-predat[,2]))
#>   min   Q1 median   Q3 max mean      sd  n missing
#> 4.03 4.04 4.07 4.17 4.34 4.12 0.0904 76          0
```

Zu den Rändern hin wird es breiter. Am schmalsten ist es übrigens beim Mittelwert der unabhängigen Beobachtungen, hier also bei 19.79\$.

## 0.36 Übung: Teaching Rating

Dieser Datensatz analysiert u. a. den Zusammenhang zwischen Schönheit und Evaluierungsergebnis von Dozenten (Hamermesh and Parker 2005). Sie können ihn, sofern noch nicht geschehen, von <https://goo.gl/6Y3KoK> als csv herunterladen.

Versuchen Sie, das Evaluierungsergebnis als abhängige Variable anhand geeigneter Variablen des Datensatzes zu erklären. Wie groß ist der Einfluss der Schönheit? Sind die Modellannahmen erfüllt und wie beurteilen Sie die Modellgüte?

## 0.37 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.

Zuerst führen wir dafür eine simple Regression aus und lassen uns  $R^2$  ausgeben.

```
df <- read_csv("https://sebastiansauer.github.io/data/wo_men.csv")

lm1 <- lm(shoe_size ~ height, data = df)
summary(lm1)$r.squared
#> [1] 0.306
```

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 <- df %>%
  sample_frac(.8, replace = FALSE) # Stichprobe von 80%, ohne Zurücklegen

test <- df %>%
```

```
anti_join(train) # Alle Zeilen von "df", die nicht in "train" vorkommen

lm2 <- lm(shoe_size ~ height, data = train)
```

Dann testen wir (die Modellgüte) anhand der Test-Stichprobe. Also los, `lm2`, mach Deine Vorhersage:

```
lm2_predict <- predict(lm2, 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 (“`data = test`”) verwendet werden sollen.

Als Ergebnis bekommen wir einen Vektor, der für jede Beobachtung des Test-Samples den geschätzten (vorhergesagten) Trinkgeld-Wert speichert.

```
caret::postResample(pred = lm2_predict, obs = test$shoe_size)
#>      RMSE Rsquared
#>    10.540    0.634
```

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!

## 0.38 Literatur

- David M. Diez, Christopher D. Barr, Mine Çetinkaya-Rundel (2014): *Introductory Statistics with Randomization and Simulation*,

- [https://www.openintro.org/stat/textbook.php?stat\\_book=isrs](https://www.openintro.org/stat/textbook.php?stat_book=isrs), Kapitel 5, 6.1-6.3
- Nicholas J. Horton, Randall Pruim, Daniel T. Kaplan (2015): Project MOSAIC Little Books *A Student's Guide to R*, <https://github.com/ProjectMOSAIC/LittleBooks/raw/master/StudentGuide/MOSAIC-StudentGuide.pdf>, Kapitel 5.4, 10.2
  - Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013): *An Introduction to Statistical Learning – with Applications in R*, <http://www-bcf.usc.edu/~gareth/ISL/>, Kapitel 3
  - Maike Luhmann (2015): *R für Einsteiger*, Kapitel 16, 17.1-17.3
  - Andreas Quatember (2010): *Statistik ohne Angst vor Formeln*, Kapitel 3.11
  - Daniel Wollschläger (2014): *Grundlagen der Datenanalyse mit R*, Kapitel 6

---

Diese Übung basiert teilweise auf Übungen zum Buch OpenIntro<sup>28</sup> von Andrew Bray und Mine Çetinkaya-Rundel unter der Lizenz Creative Commons Attribution-ShareAlike 3.0 Unported<sup>29</sup>.

---

<sup>28</sup>[https://www.openintro.org/stat/index.php?stat\\_book=isrs](https://www.openintro.org/stat/index.php?stat_book=isrs)

<sup>29</sup><http://creativecommons.org/licenses/by-sa/3.0>



# Klassifizierende Regression

## 0.39 Vorbereitung

Hier werden wir den Datensatz *Aktienkauf* der Universität Zürich (Universität Zürich, Methodenberatung<sup>30</sup>) analysieren. Es handelt es sich hierbei um eine Befragung einer Bank im Zusammenhang mit den Fakten, die mit der Wahrscheinlichkeit, dass jemand Aktien erwirbt, zusammenhängen. Es wurden 700 Personen befragt. Folgende Daten wurden erhoben: Aktienkauf (0 = nein, 1 = ja), Jahreseinkommen (in Tausend CHF), Risikobereitschaft (Skala von 0 bis 25) und Interesse an der aktuellen Marktlage (Skala von 0 bis 45).

Den Datensatz können Sie in so als csv-Datei herunterladen:

```
Aktien <- read.csv2("https://raw.githubusercontent.com/luebby/Datenanalyse-mit-R/mast
```

Zur Unterstützung der Analyse wird (wieder) `mosaic` und `ggplot2` verwendet.

```
library(mosaic)
library(ggplot2)
```

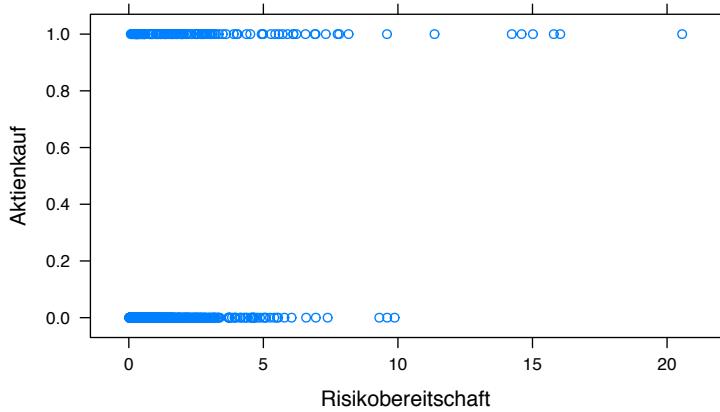
---

<sup>30</sup><http://www.methodenberatung.uzh.ch/de/datenanalyse/zusammenhaenge/lreg.html>

## 0.40 Problemstellung

Können wir anhand der Risikobereitschaft abschätzen, ob die Wahrscheinlichkeit für einen Aktienkauf steigt? Schauen wir uns zunächst ein Streudiagramm an:

```
xyplot(Aktienkauf ~ Risikobereitschaft, data = Aktien)
```



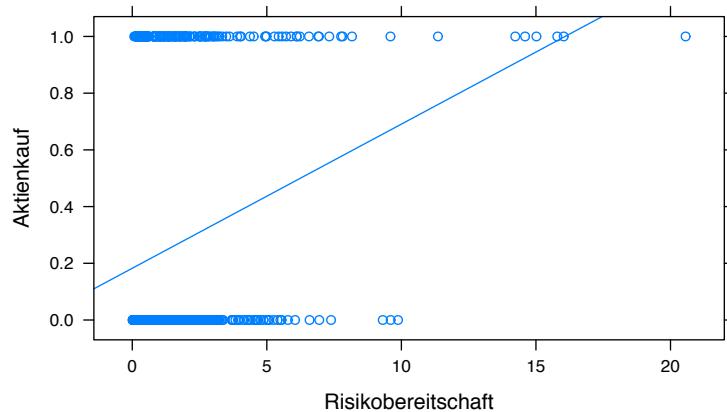
Der Zusammenhang scheint nicht sehr ausgeprgt zu sein. Lassen Sie uns dennoch ein lineare Regression durchfhren und das Ergebnis auswerten und graphisch darstellen.

```

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|)
#> (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
plotModel(lm1)
```



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?

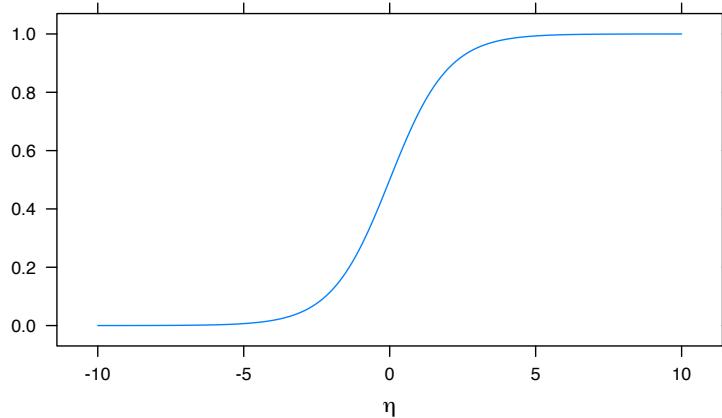
Wahrscheinlichkeiten müssen zwischen 0 und 1 liegen. Daher brauchen wir eine Funktion, die das Ergebnis einer linearen Regression in einen Bereich von 0 bis 1 bringt, die sogenannte *Linkfunktion*. Eine häufig dafür verwendete Funktion ist die logistische Funktion:

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

$\eta$ , das sogenannte *Logit*, ist darin die Linearkombination der Einflussgrößen:

$$\eta = \beta_0 + \beta_1 \cdot x_1 + \dots$$

Exemplarisch können wir die logistische Funktion für einen Bereich von  $\eta = -10$  bis  $+10$  darstellen:



## 0.41 Die Idee der logistischen Regression

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

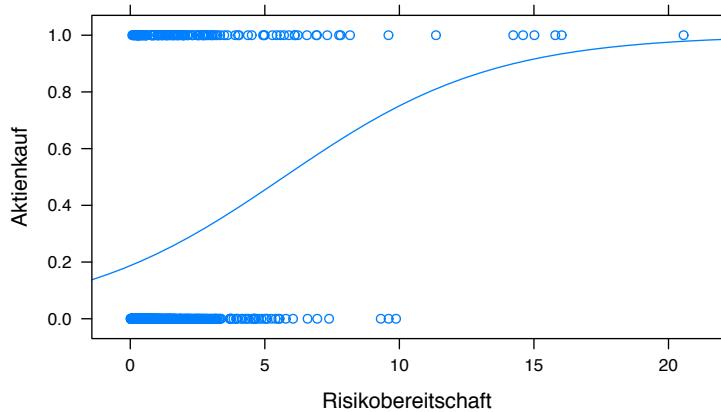
$$p(y_i = 1) = L(\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_K \cdot x_{ik}) + \epsilon_i$$

$L$  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. Wir schauen uns im Anschluss zunächst den Plot an.

```
glm1 <- glm(Aktienkauf ~ Risikobereitschaft, family = binomial("logit"),
              data = Aktien)
plotModel(glm1)
```



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
```

Der Achsenabschnitt (`intercept`) des logits  $\eta$  wird mit -1.47 geschätzt, die Steigung in Richtung **Risikobereitschaft** mit 0.26. Die (Punkt-)Prognose für die Wahrscheinlichkeit eines Aktienkaufs  $p(y = 1)$  benötigt anders als in der linearen Regression noch die Linkfunktion und ergibt sich somit zu:

$$p(\text{Aktienkauf} = 1) = \frac{1}{1 + e^{-( -1.47 + 0.26 \cdot \text{Risikobereitschaft})}}$$

Die p-Werte der Koeffizienten können in der Spalte `Pr(>|z|)` abgelesen werden. Hier wird ein *Wald*-Test durchgeführt, nicht wie bei der linearen Regression ein t-Test, ebenfalls mit der  $H_0 : \beta_i = 0$ . Die Teststatistik (`z value`) wird wie in der linearen Regression durch Divisions des Schätzers (`Estimate`) durch den Standardfehler (`Std. Error`) ermittelt. Im *Wald*-Test ist die Teststatistik allerdings  $\chi^2$ -verteilt mit einem Freiheitsgrad.

## 0.42 Welche Unterschiede zur linearen Regression gibt es in der Ausgabe?

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.)

Es gibt keine F-Statistik (oder ANOVA) mit der Frage, ob das Modell als Ganzes signifikant ist. (Anmerkung: es kann aber ein vergleichbarer Test durchgeführt werden – kommt später.)

## 0.43 Interpretation der Koeffizienten

### 0.43.1 y-Achsenabschnitt (Intercept) $\beta_0$

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.

### 0.43.2 Steigung $\beta_i$ 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.

Eine Abschätzung der Änderung der Wahrscheinlichkeit (*relatives Risiko, relative risk RR*) kann über das Chancenverhältnis (*Odds Ratio OR*) gemacht werden.<sup>31</sup> Es ergibt sich vereinfacht  $e^{\beta_i}$ . Die Wahrscheinlichkeit ändert sich näherungsweise um diesen Faktor, wenn sich  $x_i$  um eine Einheit erhöht. **Hinweis:**  $RR \approx OR$  gilt nur, wenn der Anteil des modellierten Ereignisses in den beobachteten Daten sehr klein (< 5%) oder sehr groß ist (> 95%).

*Übung:* 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**. Zur Erinnerung: Sie können `makeFun(model)` verwenden.

```
# 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”.

---

<sup>31</sup>Wahrscheinlichkeit vs. Chance: Die Wahrscheinlichkeit bei einem fairen Würfel, eine 6 zu würfeln, ist  $1/6$ . Die Chance (*Odd*), eine 6 zu würfeln, ist die Wahrscheinlichkeit dividiert durch die Gegenwahrscheinlichkeit, also  $\frac{1/6}{5/6} = 1/5$ .

```
# mit dem vollständigen Modell berechnet
fun1 <- makeFun(glm1)
fun1(Risikobereitschaft = 1)
#>     1
#> 0.229
fun1(Risikobereitschaft = 8)
#>     1
#> 0.643
# als Faktor ausgeben
fun1(Risikobereitschaft = 8)/fun1(Risikobereitschaft = 1)
#>     1
#> 2.8
```

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.

## 0.44 Kategoriale Variablen

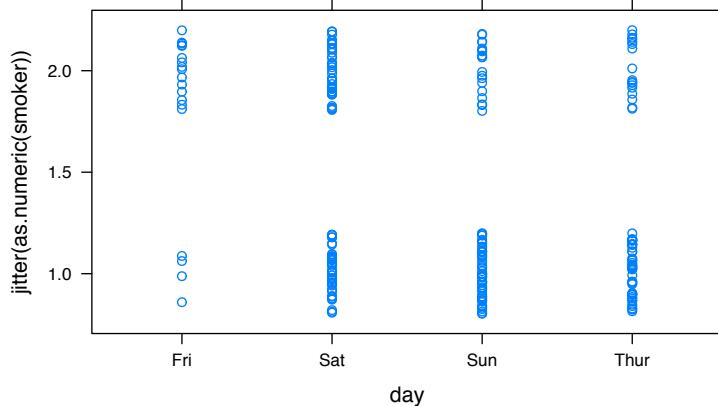
Wie schon in der linearen Regression können auch in der logistischen Regression kategoriale Variablen als unabhängige Variablen genutzt werden. Als Beispiel nehmen wir den Datensatz `tips` und versuchen abzuschätzen, ob sich die Wahrscheinlichkeit dafür, dass ein Raucher bezahlt hat (`smoker == yes`), in Abhängigkeit vom Wochentag ändert.

Sofern noch nicht geschehen, können Sie so als `csv`-Datei herunterladen:

```
tips <- read.csv("https://sebastiansauer.github.io/data/tips.csv")
```

Zunächst ein Plot:

```
xyplot(jitter(as.numeric(smoker)) ~ day, data = tips)
```



**Hinweis:** Um zu sehen, ob es an manchen Tagen mehr Raucher gibt, sollten Sie zumindest eine Variable „verrauschen“ („jittern“). Da die Variable `smoker` eine nominale Variable ist und die Funktion `jitter()` nur mit numerischen Variablen arbeitet, muss sie mit `as.numeric()` in eine numerische Variable umgewandelt werden.

Die relativen Häufigkeiten zeigt folgende Tabelle:

```
(tab_smoke <- tally(smoker ~ day, data = tips, format = "proportion"))
#>      day
#> smoker   Fri   Sat   Sun   Thur
#>   No  0.211 0.517 0.750 0.726
#>   Yes 0.789 0.483 0.250 0.274
```

Hinweis: Durch die Klammerung wird das Objekt `tab_smoke` direkt ausgegeben.

Probieren wir die logistische Regression aus:

```
glmtips <- glm(smoker ~ day, family = binomial("logit"), data = tips)
summary(glmtips)
#>
#> Call:
```

```
#> glm(formula = smoker ~ day, family = binomial("logit"), data = tips)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -1.765  -0.801  -0.758   1.207   1.665
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  1.322    0.563   2.35  0.01883 *
#> daySat       -1.391   0.602  -2.31  0.02093 *
#> daySun       -2.420   0.622  -3.89  1e-04 ***
#> dayThur      -2.295   0.631  -3.64  0.00027 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 324.34 on 243 degrees of freedom
#> Residual deviance: 298.37 on 240 degrees of freedom
#> AIC: 306.4
#>
#> Number of Fisher Scoring iterations: 4
```

Auch hier können wir die Koeffizienten in Relation zur Referenzkategorie (hier: Freitag) interpretieren. Die Wahrscheinlichkeit ist an einem Samstag niedriger, der Wert für daySat ist negativ. Eine Abschätzung erhalten wir wieder mit  $e^{\beta_i}$ :

```
exp(coef(glmtips)[2])
#> daySat
#> 0.249
```

Daher ist das Chancenverhältnis (*Odds Ratio*), dass am Samstag ein Raucher am Tisch sitzt, näherungsweise um den Faktor 0.25 niedriger als am Freitag<sup>32</sup>:

---

<sup>32</sup>Am Freitag liegen die Chancen (das OR) für einen Raucher bei 3.75. Das OR für

$$OR = \frac{\frac{P(Raucher|Samstag)}{1-P(Raucher|Samstag)}}{\frac{P(Raucher|Freitag)}{1-P(Raucher|Freitag)}} = \frac{\frac{0.483}{0.517}}{\frac{0.79}{0.21}} \approx 0.249$$

Die Wahrscheinlichkeit für einen Raucher am Samstag können wir uns wieder komfortabel so ausgeben lassen:

```
fun2 <- makeFun(glmtips)
fun2(day = "Sat")
#>      1
#> 0.483
```

## 0.45 Multiple logistische Regression

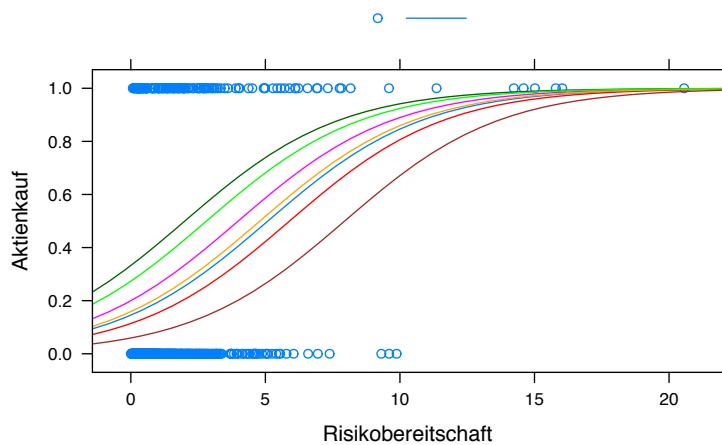
Wir kehren wieder zurück zu dem Datensatz *Aktienkauf*. Können wir unser Model `glm1` mit nur einer erklärenden Variable verbessern, indem weitere unabhängige Variablen hinzugefügt werden?

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

---

Samstag ist das Produkt dieser beiden OR. Um das OR zu einer Wahrscheinlichkeit umzurechnen kann man, möchte vom “von Hand” arbeiten, diese Formel verwenden:  $p = OR/(OR + 1)$ .

```
#> 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 ' '
#>
#> (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.

Ist das Modell besser als das einfache? Ja, da der AIC-Wert von 769.86 auf 687.01 gesunken ist.

Die Graphik zeigt die Verläufe in Abhängigkeit von den verschiedenen Variablen und den Kombinationen der Variablen.

## 0.46 Modell- bzw. Klassifikationsgüte

Logistische Regressionsmodelle werden häufig zur Klassifikation verwendet, z. B. ob der Kredit für einen Neukunden ein “guter” Kredit ist oder nicht. Daraus sind die Klassifikationseigenschaften bei logistischen Modellen wichtige Kriterien.

Hierzu werden die aus dem Modell ermittelten Wahrscheinlichkeiten ab einem Schwellenwert (*cutpoint*), häufig 0.5, einer geschätzten 1 zugeordnet, unterhalb des Schwellenwertes einer 0. Diese aus dem Modell ermittelten Häufigkeiten werden dann in einer sogenannten Konfusionsmatrix (*confusion matrix*) mit den beobachteten Häufigkeiten verglichen.

Daher sind wichtige Kriterien eines Modells, wie gut diese Zuordnung erfolgt. Dazu werden die Sensitivität (*True Positive Rate, TPR*), also der Anteil der mit 1 geschätzten an allen mit 1 beobachteten Werten, und die Spezifität (*True Negative Rate*) berechnet. Ziel ist es, dass beide Werte möglichst hoch sind.

Sie können die Konfusionsmatrix “zu Fuß” berechnen, in dem Sie eine neue Variable einfügen, die ab dem cutpoint 1 und sonst 0 ist und mit dem Befehl `tally()` ausgeben. Alternativ können Sie das Paket `SDMTools` verwenden mit der Funktion `confusion.matrix()`. Ein Parameter ist `cutpoint`, der standardmäßig auf 0.5 steht.

```
# Konfusionsmatrix "zu Fuß" berechnen
# cutpoint = 0.5 setzen
# neue Variable predicted anlegen mit 1, wenn modellierte Wahrscheinlichkeit > 1 ist
cutpoint = 0.5
Aktien$predicted <- ((glm1$fitted.values) > cutpoint)*1
# Kreuztabelle berechnen
(cm <- tally(~predicted+Aktienkauf, data = Aktien))
#>           Aktienkauf
#> predicted  0    1
#>           0 509 163
```

```

#>      1  8 20
# Sensitivität (TPR)
cm[2,2]/sum(cm[,2])
#> [1] 0.109
# Spezifität (TNR)
cm[1,1]/sum(cm[,1])
#> [1] 0.985

# mit Hilfe des Pakets SDMTools
# ggf. install.packages("SDMTools")
library(SDMTools)
# optional noch Parameter cutpoint = 0.5 angeben
(cm <- confusion.matrix(Aktien$Aktienkauf, glm1$fitted.values))
#>    obs
#> pred  0   1
#>     0 509 163
#>     1   8 20
#> attr(,"class")
#> [1] "confusion.matrix"
sensitivity(cm)
#> [1] 0.109
specificity(cm)
#> [1] 0.985

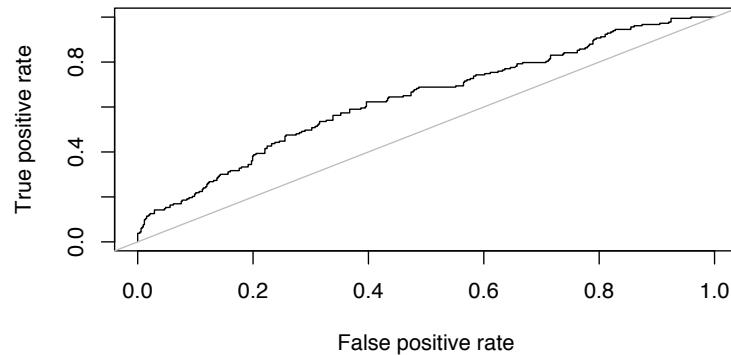
```

Wenn die Anteile der 1 in den beobachteten Daten sehr gering sind (z. B. bei einem medizinischen Test auf eine seltene Krankheit, Klicks auf einen Werbebanner oder Kreditausfall), kommt eine Schwäche der logistischen Regression zum Tragen: Das Modell wird so optimiert, dass die Wahrscheinlichkeiten  $p(y = 1)$  alle unter 0.5 liegen. Das würde zu einer Sensitivität von 0 und einer Spezifität von 1 führen. Daher kann es sinnvoll sein, den Cutpoint zu variierten. Daraus ergibt sich ein verallgemeinertes Gütemaß, die *ROC-Kurve* (*Return Operating Characteristic*) und den daraus abgeleiteten *AUC*-Wert (*Area Under Curve*).

Hierzu wird der Cutpoint zwischen 0 und 1 variiert und die Sensitivität gegen 1-Spezifität (welche Werte sind als 1 modelliert worden unter den beobachteten 0, *False Positive Rate, FPR*). Um diese Werte auszugeben,

benötigen Sie das Paket `ROCR` und die Funktion `performance()`.

```
# ggf. install.packages("ROCR")
library(ROCR)
# Ein für die Auswertung notwendiges prediction Objekt anlegen
pred <- prediction(glm1$fitted.values, Aktien$Aktienkauf)
# ROC Kurve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
abline(0,1, col = "grey")
# Area under curve (ROC-Wert)
performance(pred, "auc")@y.values
#> [[1]]
#> [1] 0.636
```



AUC liegt zwischen 0.5, wenn das Modell gar nichts erklärt (im Plot die graue Linie) und 1. Hier ist der Wert also recht gering. Akzeptable Werte liegen bei 0.7 und größer, gute Werte sind es ab 0.8.<sup>33</sup>

---

<sup>33</sup>Hosmer/Lemeshow, Applied Logistic Regression, 3rd Ed. (2013), S. 164

## 0.47 Erweiterungen

### 0.47.1 Modellschätzung

Das Modell wird nicht wie bei der linearen Regression über die Methode der kleinsten Quadrate (OLS) geschätzt, sondern über die *Maximum Likelihood* Methode. Die Koeffizienten werden so gewählt, dass die beobachteten Daten am wahrscheinlichsten (*Maximum Likelihood*) werden.

Das ist ein iteratives Verfahren (OLS erfolgt rein analytisch), daher wird in der letzten Zeile der Ausgabe auch die Anzahl der Iterationen (**Fisher Scoring Iterations**) ausgegeben.

Die Devianz des Modells (**Residual deviance**) ist  $-2$  mal die logarithmierte Likelihood. Die Nulldevianz (**Null deviance**) ist die Devianz eines Nullmodells, d. h., alle  $\beta$  außer der Konstanten sind 0.

### 0.47.2 Likelihood Quotienten Test

Der Likelihood Quotienten Test (*Likelihood Ratio Test, LR-Test*) vergleicht die Likelihood  $L_0$  des Nullmodells mit der Likelihood  $L_\beta$  des geschätzten Modells. Die Prüfgröße des LR-Tests ergibt sich aus:

$$T = -2 \cdot \ln \left( \frac{L_0}{L_\beta} \right)$$

$T$  ist näherungsweise  $\chi^2$ -verteilt mit  $k$  Freiheitsgraden.

In R können Sie den Test mit `lrtest()` aufrufen. Sie benötigen dazu das Paket `lmtest`.

```
library(lmtest)
lrtest(glm2)
#> Likelihood ratio test
#>
#> Model 1: Aktienkauf ~ Risikobereitschaft + Einkommen + Interesse
#> Model 2: Aktienkauf ~ 1
```

```
#> #Df LogLik Df Chisq Pr(>Chisq)
#> 1 4 -340
#> 2 1 -402 -3 125      <2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Das Modell `glm2` ist als Ganzes signifikant, der p-Wert ist sehr klein.

Den Likelihood Quotienten Test können Sie auch verwenden, um zwei Modelle miteinander zu vergleichen, z. B., wenn Sie eine weitere Variable hinzugenommen haben und wissen wollen, ob die Verbesserung auch signifikant war.

```
lrtest(glm1, glm2)
#> Likelihood ratio test
#>
#> Model 1: Aktienkauf ~ Risikobereitschaft
#> Model 2: Aktienkauf ~ Risikobereitschaft + Einkommen + Interesse
#> #Df LogLik Df Chisq Pr(>Chisq)
#> 1 2 -383
#> 2 4 -340 2 86.9      <2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ja, die Modelle `glm1` (mit einer erklärenden Variable) und `glm2` unterscheiden sich signifikant voneinander.

### 0.47.3 Pseudo- $R^2$

Verschiedene Statistiker haben versucht, aus der Likelihood eine Größe abzuleiten, die dem  $R^2$  der linearen Regression entspricht. Exemplarisch sei hier McFaddens  $R^2$  gezeigt:

$$R^2 = 1 - \frac{\ln(L_\beta)}{\ln(L_0)}$$

Wie bei bei dem  $R^2$  der linearen Regression liegt der Wertebereich zwischen 0 und 1. Ab einem Wert von 0,4 kann die Modellanpassung als gut eingestuft werden. Wo liegen  $R^2$  der beiden Modelle `glm1` und `glm2`? Sie können es direkt berechnen oder das Paket `BaylorEdPsych` verwenden.

```
# direkte Berechnung
1 - glm1$deviance/glm1$null.deviance
#> [1] 0.0479
1 - glm2$deviance/glm2$null.deviance
#> [1] 0.156
# ggf. install.packages("BaylorEdPsych")
library(BaylorEdPsych)
PseudoR2(glm1)
#>      McFadden     Adj. McFadden      Cox.Snell      Nagelkerke
#>      0.0479          0.0404        0.0535       0.0783
#> McKelvey.Zavoina      Effron        Count      Adj.Count
#>      0.0826          0.0584        0.7557       0.0656
#>           AIC    Corrected.AIC
#>      769.8624         769.8796
PseudoR2(glm2)
#>      McFadden     Adj. McFadden      Cox.Snell      Nagelkerke
#>      0.1558          0.1434        0.1640       0.2400
#> McKelvey.Zavoina      Effron        Count      Adj.Count
#>      0.2828          0.1845        0.7614       0.0874
#>           AIC    Corrected.AIC
#>      687.0068         687.0644
```

Insgesamt ist die Modellanpassung, auch mit allen Variablen, als schlecht zu bezeichnen. **Hinweis:** Die Funktion `PseudoR2(model)` zeigt verschiedene Pseudo- $R^2$  Statistiken, die jeweils unter bestimmten Bedingungen vorteilhaft einzusetzen sind. Für weitere Erläuterungen sei auf die Literatur verwiesen.

## 0.48 Übung: Rot- oder Weißwein?

Der Datensatz untersucht den Zusammenhang zwischen der Qualität und physiochemischen Eigenschaften von portugiesischen Rot- und Weißweinen

(Cortez et al. 2009).

Sie können in unter <https://goo.gl/Dkd7nK> herunterladen. Die Originaldaten finden Sie im UCI Machine Learning Repository<sup>34</sup>.

Versuchen Sie anhand geeigneter Variablen, Rot- und Weißweine (richtig) zu klassifizieren<sup>35</sup>.

**Zusatzaufgabe:** Die Originaldaten bestehen aus einem Datensatz für Weißweine und einem für Rotweine. Laden Sie diese, beachten Sie die Fehlermeldung und beheben die damit verbundenen Fehler und fassen beide Datensätze zu einem gemeinsamen Datensatz zusammen, in dem eine zusätzliche Variable `color` aufgenommen wird (Rot = 0, Weiß = 1).

## 0.49 Literatur

- David M. Diez, Christopher D. Barr, Mine Çetinkaya-Rundel (2014): *Introductory Statistics with Randomization and Simulation*, [https://www.openintro.org/stat/textbook.php?stat\\_book=isrs](https://www.openintro.org/stat/textbook.php?stat_book=isrs), Kapitel 6.4
- Nicholas J. Horton, Randall Pruim, Daniel T. Kaplan (2015): Project MOSAIC Little Books *A Student's Guide to R*, <https://github.com/ProjectMOSAIC/LittleBooks/raw/master/StudentGuide/MOSAIC-StudentGuide.pdf>, Kapitel 8
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013): *An Introduction to Statistical Learning – with Applications in R*, <http://www-bcf.usc.edu/~gareth/ISL/>, Kapitel 4.1-4.3
- Maike Luhmann (2015): *R für Einsteiger*, Kapitel 17.5
- Daniel Wollschläger (2014): *Grundlagen der Datenanalyse mit R*, Kapitel 8.1

---

<sup>34</sup><http://archive.ics.uci.edu/ml/datasets/Wine+Quality>

<sup>35</sup>Anregungen zu dieser Übung stammen von INTW Statistics: [https://www.inwt-statistics.de/blog-artikel-lesen/Logistische\\_Regression\\_Beispiel\\_mit\\_R.html](https://www.inwt-statistics.de/blog-artikel-lesen/Logistische_Regression_Beispiel_mit_R.html)



# Baumbasierte Verfahren

## 0.50 Konjunkturanalyse

Der B3 Datensatz *Heilemann, U. and Münch, H.J. (1996): West German Business Cycles 1963-1994: A Multivariate Discriminant Analysis. CIRET-Conference in Singapore, CIRET-Studien 50.* enthält Quartalsweise Konjunkturdaten aus (West-)Deutschland.

```
B3 <- read.csv2("https://goo.gl/0YCEHf")
str(B3) # Datenstruktur
#> 'data.frame': 157 obs. of 14 variables:
#> $ PHASEN : int 2 2 3 3 3 3 3 3 3 ...
#> $ BSP91JW : num 10.53 10.6 9.21 5.17 4.93 ...
#> $ CP91JW : num 9.31 12.66 6.55 7.87 8.6 ...
#> $ DEFRATE : num 0.05 0.06 0.05 0.05 0.04 0.04 0.04 0.03 0.03 0 ...
#> $ EWAJW : num 5.7 5.2 4.8 3.3 2.1 3.2 2.5 2.7 3 0.3 ...
#> $ EXIMRATE: num 3.08 1.96 2.82 3.74 4.16 2.9 3.65 4.57 4.37 2.89 ...
#> $ GM1JW : num 11.15 11.03 10.04 8.33 7.69 ...
#> $ IAU91JW : num 23.56 12.72 11.52 0.85 -2.08 ...
#> $ IB91JW : num 14.69 24.95 14.9 7.55 3.23 ...
#> $ LSTKJW : num 3 2.36 3.39 5.3 6.91 1.03 3.73 6.2 4.12 7.94 ...
#> $ PBSPJW : num 2.89 2.59 3.01 3.03 3.46 1.95 3.18 3.98 3.29 5.63 ...
#> $ PCPJW : num 1.91 2.2 3.09 2.08 1.48 1.65 1.47 3.29 3.59 4.19 ...
#> $ ZINSK : num 6.27 4.6 6.19 6.71 7.1 4.96 5.21 4.83 4.5 3.83 ...
#> $ ZINSLR : num 3.21 3.54 3.22 3.37 3.14 4.95 3.82 3.09 3.91 1.47 ...
head(B3); tail(B3)
#>    PHASEN BSP91JW CP91JW DEFRATE EWAJW EXIMRATE GM1JW IAU91JW IB91JW LSTKJW
```

```

#> 1   2   10.53  9.31   0.05   5.7    3.08 11.15  23.56 14.69  3.0
#> 2   2   10.60  12.66   0.06   5.2    1.96 11.03  12.72 24.95  2.3
#> 3   3   9.21   6.55   0.05   4.8    2.82 10.04  11.52 14.90  3.3
#> 4   3   5.17   7.87   0.05   3.3    3.74 8.33   0.85  7.55  5.3
#> 5   3   4.93   8.60   0.04   2.1    4.16 7.69   -2.08 3.23  6.9
#> 6   3   8.39   5.62   0.04   3.2    2.90 6.62   -3.76 14.58  1.0
#>    PBSPJW PCPJW ZINSK ZINSLR
#> 1   2.89  1.91  6.27   3.21
#> 2   2.59  2.20  4.60   3.54
#> 3   3.01  3.09  6.19   3.22
#> 4   3.03  2.08  6.71   3.37
#> 5   3.46  1.48  7.10   3.14
#> 6   1.95  1.65  4.96   4.95
#>    PHASEN BSP91JW CP91JW DEFRATE EWAJW EXIMRATE GM1JW IAU91JW IB91JW
#> 152   3   -1.27  1.29   -4.87 -1.97   6.03 9.79  -18.29  1.73
#> 153   3   -2.13  -0.57  -2.98 -2.05   7.59 0.72  -15.82 -3.23
#> 154   3   1.39   2.33  -2.86 -1.84   7.49 11.33 -10.59  4.62
#> 155   4   1.63   0.64   1.20 -1.58   7.75 11.38 -4.90  3.62
#> 156   1   1.40   0.57  -3.56 -1.34   5.58 9.53  -0.76  2.19
#> 157   1   1.83  -0.08  -2.22 -0.93   7.50 15.20  2.75  6.12
#>    LSTKJW PBSPJW PCPJW ZINSK ZINSLR
#> 152   1.08  2.73  2.98   6.83   3.55
#> 153   1.67  2.67  3.31   6.35   3.05
#> 154  -0.12  2.66  2.94   5.88   3.17
#> 155  -1.81  1.77  2.58   5.29   4.82
#> 156  -1.54  1.85  2.60   5.01   5.27
#> 157  -0.92  1.79  2.49   5.28   5.62

```

Dabei sind folgende Variablen enthalten:

- Bruttosozialprodukt (real): **BSP91JW**
- Privater Verbrauch (real): **CP91JW**
- Anteil Staatsdefizit am Bruttosozialprodukt (%): **DEFRATE**
- Abhängig Erwerbstätige: **EWAJW**
- Anteil Außenbeitrag am Bruttosozialprodukt (%): **EXIMRATE**
- Geldmenge M1: **GM1JW**
- Investitionen in Ausrüstungsgüter (real): **IAU91JW**

- Investitionen in Bauten (real): IB91JW
- Lohnstückkosten: LSTKJW
- Preisindex des Bruttosozialprodukts: PBSPJW
- Preisindex des privaten Verbrauchs: PCPJW
- Kurzfristiger Zinssatz (nominal): ZINSK
- Langfristiger Zinssatz (real): ZINSLR
- Konjunkturphase: 1. Aufschwung, 2. Oberer Wendepunkt, 3. Abschwung, 4. Unterer Wendepunkt: PHASEN

Variablen mit der Endung *JW* beziehen sich auf die jährliche Veränderung.

## 0.51 Regressionsbäume

Um einen Regressionsbaum zu erzeugen, muss zunächst das Zusatzpaket **rpart** geladen werden:

```
library(rpart)
```

Um z. B. die Veränderung des Bruttosozialprodukt als Funktion von Privater Verbrauch, Investitionen in Ausrüstungsgüter, Investitionen in Bauten und Geldmenge M1 als Regressionsbaum zu modellieren reicht der Befehl

```
regbaum <- rpart(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=B3)
```

Um das Ergebnis auszugeben genügt:

```
regbaum
#> n= 157
#>
#> node), split, n, deviance, yval
#>      * denotes terminal node
#>
#> 1) root 157 1380.0 3.5000
#> 2) CP91JW< 3.71 79 380.0 1.5200
#> 4) IAU91JW< -0.365 38 120.0 -0.0379
```

```

#>      8) IB91JW< -2.22 20   45.1 -0.9020 *
#>      9) IB91JW>=-2.22 18   43.9  0.9220 *
#>      5) IAU91JW>=-0.365 41   81.6  2.9600
#>      10) IB91JW< 2.42 22   29.5  2.2100 *
#>      11) IB91JW>=2.42 19   25.0  3.8400 *
#>      3) CP91JW>=3.71 78   378.0  5.5000
#>      6) IAU91JW< 11.3 50   142.0  4.5700
#>      12) IB91JW< 3.2 22   40.0  3.5500 *
#>      13) IB91JW>=3.2 28   61.3  5.3700 *
#>      7) IAU91JW>=11.3 28   114.0  7.1700
#>      14) IB91JW< 7.55 17   51.2  6.2200 *
#>      15) IB91JW>=7.55 11   23.7  8.6400 *

```

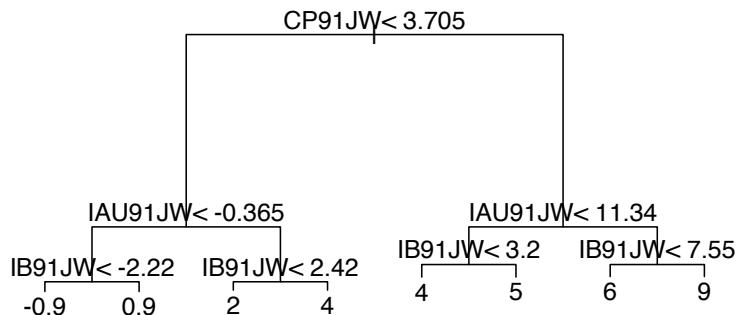
*Lesebeispiel:* Wenn  $CP91JW \geq 3.705$  und  $IAU91JW \geq 11.335$  und  $IB91JW \geq 7.55$  liegt, dann liegt die durchschnittliche Veränderung des BSP91JW bei 8.639. 11 Beobachtungen erfüllen die Kriterien der unabhängigen Variablen

Bzw. um den Baum zu zeichnen

```

par(xpd = TRUE) # Grafikparameter der sicherstellt, dass alles ins Bild passst
plot(regbaum, compress = TRUE) # Baum zeichnen
text(regbaum) # Baum beschriften

```



Eine deutlich schönere Ausgabe erhält man z. B. mit dem Zusatzpaket `rpart.plot`, welches man *einmalig* über

```
install.packages("rpart.plot")
```

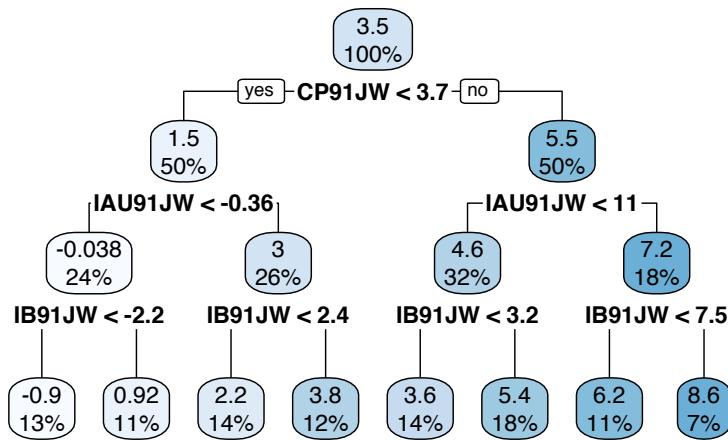
installieren muss und dann benutzen kann.

Zunächst laden

```
library(rpart.plot)
```

und dann zeichnen:

```
rpart.plot(regbaum)
```



## 0.52 Kreuzvalidierung

### 0.52.1 Anpassungsgüte

Wie gut ist das Modell? Über `predict` können die Punktprognosen berechnet werden:

```
head(predict(regbaum))
#>   1    2    3    4    5    6
#> 8.64 8.64 8.64 5.37 5.37 5.37
```

Diese werden mit den beobachteten Werten verglichen:

```
head(B3$BSP91JW)
#> [1] 10.53 10.60 9.21 5.17 4.93 8.39
```

Der **Mean Squared Error** ist dann

```
baummse <- mean( (predict(regbaum) - B3$BSP91JW)^2 )
baummse
#> [1] 2.04
```

Vergleichen wir das Ergebnis mit dem einer linearen Regression

```
reglm <- lm(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=B3)
summary(reglm)
#>
#> Call:
#> lm(formula = BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data = B3)
#>
#> Residuals:
#>   Min     1Q Median     3Q    Max
#> -3.048 -0.880 -0.057  0.801  3.674
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 1.22442   0.26280   4.66  6.9e-06 ***
#> CP91JW      0.38729   0.05846   6.63  5.7e-10 ***
#> IAU91JW     0.12752   0.01634   7.80  9.0e-13 ***
#> IB91JW      0.13880   0.01719   8.08  1.9e-13 ***
#> GM1JW       -0.00996   0.02676  -0.37    0.71
#> ---
```

```
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.37 on 152 degrees of freedom
#> Multiple R-squared: 0.792, Adjusted R-squared: 0.786
#> F-statistic: 145 on 4 and 152 DF, p-value: <2e-16
```

Der MSE der Linearen Regression liegt bei

```
lmmse <- mean( predict(reglm) - B3$BSP91JW)^2 )
lmmse
#> [1] 1.83
```

Der Baum ist einfacher und weniger flexibel, aber auch schlechter im Bezug auf die Anpassungsgüte.

## 0.52.2 Prognosegüte

Für eine k=3 fache Kreuzvalidierung müssen 3 Testdatensätze erzeugt werden.

Zunächst wird dafür ein Aufteilungsvektor gebildet:

```
aufteilung <- rep(1:3, length.out=nrow(B3))
```

und dann wird aufgeteilt:

```
test1 <- B3[aufteilung==1,]
train1 <- B3[aufteilung!=1,]

test2 <- B3[aufteilung==2,]
train2 <- B3[aufteilung!=2,]

test3 <- B3[aufteilung==3,]
train3 <- B3[aufteilung!=3,]
```

Anschließend werden die Modelle auf den Trainingsdaten geschätzt, und auf den Testdaten überprüft:

```

# Runde 1
b1 <- rpart(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train1)
l1 <- lm(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train1)
mseb1 <- mean( (predict(b1, newdata = test1) - test1$BSP91JW)^2 )
msel1 <- mean( (predict(l1, newdata = test1) - test1$BSP91JW)^2 )

# Runde 2
b2 <- rpart(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train2)
l2 <- lm(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train2)
mseb2 <- mean( (predict(b2, newdata = test2) - test2$BSP91JW)^2 )
msel2 <- mean( (predict(l2, newdata = test2) - test2$BSP91JW)^2 )

# Runde 3
b3 <- rpart(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train3)
l3 <- lm(BSP91JW ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train3)
mseb3 <- mean( (predict(b3, newdata = test3) - test3$BSP91JW)^2 )
msel3 <- mean( (predict(l3, newdata = test3) - test3$BSP91JW)^2 )

# Ergebnisse zusammenfassen
msecvb <- c(mseb1, mseb2, mseb3)
msecv1 <- c(msel1, msel2, msel3)

# Mittelwert des Prognose MSE
mean(msecvb)
#> [1] 3.62
mean(msecv1)
#> [1] 1.99

```

Bei den vorliegenden Daten ist also ein *lineares* Modell dem Baummodell im Bezug auf den *MSE* überlegen.

**Hinweis:** In der Praxis führt man die Aufteilung nicht manuell sondern innerhalb von Schleifen durch.

## 0.53 Klassifikationsbäume

Untersuchen wir, ob makroökonomische Kennzahlen geeignet sind, die Konjunkturphasen zu unterscheiden. Zunächst stellen wir fest, dass die eigentlich kategoriale Variable PHASEN hier numerisch kodiert wurde, was aber schnell verwirren würde.

```
typeof(B3$PHASEN)
#> [1] "integer"
```

Typänderung zu `factor` geht einfach:

```
B3$PHASEN <- as.factor(B3$PHASEN)
```

Wenn wir die einzelnen `levels` des Faktors als numerische Werte verwenden wollen würde man den Befehl `as.numeric()` verwenden. Aber sicherheitshalber vorher über `levels()` gucken, ob die Reihenfolge auch stimmt.

Um die Interpretation zu erleichtern können wir hier einfach die Faktorstufe umbenennen.

```
levels(B3$PHASEN) <- c("Aufschwung", "Oberer Wendepunkt",
                        "Abschwung", "Unterer Wendepunkt")
```

Um z. B. die Konjunkturphase als Funktion von Privater Verbrauch, Investitionen in Ausrüstungsgüter, Investitionen in Bauten und Geldmenge M1 als Regressionsbaum zu modellieren reicht jetzt der Befehl

```
klassbaum <- rpart(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=B3)
```

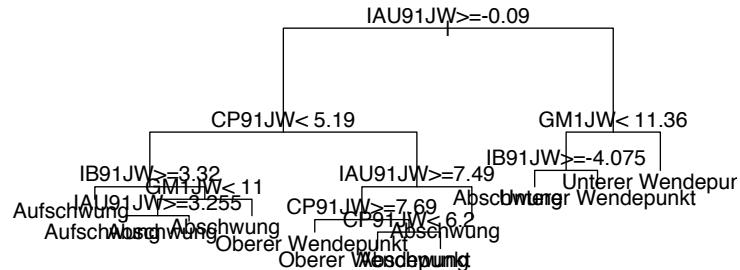
Um das Ergebnis auszugeben genügt:

```
klassbaum
#> n= 157
#>
```

```
#> node), split, n, loss, yval, (yprob)
#>      * denotes terminal node
#>
#> 1) root 157 98 Aufschwung (0.3758 0.1529 0.2994 0.1720)
#> 2) IAU91JW>=-0.09 109 55 Aufschwung (0.4954 0.2110 0.2110 0.0826)
#> 4) CP91JW< 5.19 68 20 Aufschwung (0.7059 0.1029 0.1324 0.0588)
#> 8) IB91JW>=3.32 29 7 Aufschwung (0.7586 0.2069 0.0345 0.0000) *
#> 9) IB91JW< 3.32 39 13 Aufschwung (0.6667 0.0256 0.2051 0.1026)
#> 18) GM1JW< 11 32 7 Aufschwung (0.7813 0.0312 0.1250 0.0625)
#> 36) IAU91JW>=3.25 25 3 Aufschwung (0.8800 0.0400 0.0000 0.0800)
#> 37) IAU91JW< 3.25 7 3 Abschwung (0.4286 0.0000 0.5714 0.0000)
#> 19) GM1JW>=11 7 3 Abschwung (0.1429 0.0000 0.5714 0.2857) *
#> 5) CP91JW>=5.19 41 25 Oberer Wendepunkt (0.1463 0.3902 0.3415 0.1220)
#> 10) IAU91JW>=7.49 31 15 Oberer Wendepunkt (0.1613 0.5161 0.2581 0.0)
#> 20) CP91JW>=7.69 10 2 Oberer Wendepunkt (0.1000 0.8000 0.1000 0.)
#> 21) CP91JW< 7.69 21 13 Oberer Wendepunkt (0.1905 0.3810 0.3333 0.)
#> 42) CP91JW< 6.2 8 3 Oberer Wendepunkt (0.2500 0.6250 0.1250 0.)
#> 43) CP91JW>=6.2 13 7 Abschwung (0.1538 0.2308 0.4615 0.1538) *
#> 11) IAU91JW< 7.49 10 4 Abschwung (0.1000 0.0000 0.6000 0.3000) *
#> 3) IAU91JW< -0.09 48 24 Abschwung (0.1042 0.0208 0.5000 0.3750)
#> 6) GM1JW< 11.4 38 14 Abschwung (0.0789 0.0000 0.6316 0.2895)
#> 12) IB91JW>=-4.08 23 5 Abschwung (0.1304 0.0000 0.7826 0.0870) *
#> 13) IB91JW< -4.08 15 6 Unterer Wendepunkt (0.0000 0.0000 0.4000 0.)
#> 7) GM1JW>=11.4 10 3 Unterer Wendepunkt (0.2000 0.1000 0.0000 0.7000)
```

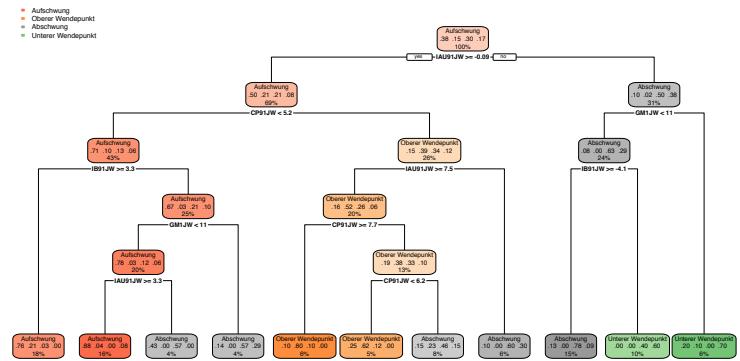
*Lesebeispiel:* Wenn IAU91JW< -0.09 und GM1JW>=11.355 liegt, dann ist der Untere Wendepunkt die häufigste Merkmalsausprägung von PHASEN (relative Häufigkeit von PHASEN=4 hier: 0.7) 10 Beobachtungen erfüllen die Kriterien der unabhängigen Variablen.

```
par(xpd = TRUE) # Grafikparameter der sicherstellt, dass alles ins Bild passst
plot(klassbaum, compress = TRUE) # Baum zeichnen
text(klassbaum) # Baum beschriften
```



Bzw. "schöner":

```
rpart.plot(klassbaum)
```



### 0.53.1 Kreuzvalidierung

Wie gut ist das Modell? Auch hier können über predict die Punktprognosen bestimmt werden:

```
head(predict(klassbaum, type="class"))
#>          1          2          3          4
#> Oberer Wendepunkt Oberer Wendepunkt Abschwung Abschwung
#>          5          6
#>      Abschwung      Abschwung
#> Levels: Aufschwung Oberer Wendepunkt Abschwung Unterer Wendepunkt
```

Diese werden mit den beobachteten Werten verglichen:

```
head(B3$PHASEN)
#> [1] Oberer Wendepunkt Oberer Wendepunkt Abschwung      Abschwung
#> [5] Abschwung          Abschwung
#> Levels: Aufschwung Oberer Wendepunkt Abschwung Unterer Wendepunkt
```

Die Fehlklassifikationsrate ist dann

```
baumer <- mean( predict(klassbaum, type="class") != B3$PHASEN )
baumer
#> [1] 0.293
```

also knapp 30%.

Vergleichen kann man den Klassifikationsbaum z. B. mit der *Linearen Diskriminanzanalyse*. Diese ist im Paket MASS implementiert.

```
library(MASS)

klasslida <- lda(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=B3)
klasslida
#> Call:
#> lda(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data = B3)
#>
#> Prior probabilities of groups:
#>           Aufschwung Oberer Wendepunkt      Abschwung
#>           0.376          0.153            0.299
#> Unterer Wendepunkt
#>           0.172
#>
#> Group means:
#>           CP91JW IAU91JW IB91JW GM1JW
#> Aufschwung      3.55  6.8936   3.40  8.61
#> Oberer Wendepunkt 6.43 11.3604   6.83 11.02
#> Abschwung       3.66 -0.0034   1.67  6.84
#> Unterer Wendepunkt 2.62 -1.9393  -1.49  9.61
```

```

#>
#> Coefficients of linear discriminants:
#>          LD1      LD2      LD3
#> CP91JW  0.1583 -0.52094  0.0758
#> IAU91JW -0.1409  0.06865 -0.0275
#> IB91JW  -0.0478  0.00242 -0.0457
#> GM1JW   -0.0203  0.08559  0.2245
#>
#> Proportion of trace:
#>    LD1     LD2     LD3
#>  0.645  0.230  0.125

ldaer <- mean( predict(klasslda)$class != B3$PHASEN) )
ldaer
#> [1] 0.414

```

Im Bezug auf die *Klassifikation* scheint der Baum in der Anpassungsgüte besser als die Lineare Diskriminanzanalyse zu sein. Aber wie sieht es kreuzvalidiert, d. h. in der Prognose aus?

Zunächst wird wieder dafür ein Aufteilungsvektor gebildet:

```
aufteilung <- rep(1:3, length.out=nrow(B3))
```

und dann wird aufgeteilt:

```

test1 <- B3[aufteilung==1,]
train1 <- B3[aufteilung!=1,]

test2 <- B3[aufteilung==2,]
train2 <- B3[aufteilung!=2,]

test3 <- B3[aufteilung==3,]
train3 <- B3[aufteilung!=3,]

```

```

# Runde 1
b1 <- rpart(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train1)
l1 <- lda(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train1)
erb1 <- mean( (predict(b1, newdata = test1, type = "class") != test1$PHASEN) )
erl1 <- mean( (predict(l1, newdata = test1)$class != test1$PHASEN) )

# Runde 2
b2 <- rpart(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train2)
l2 <- lda(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train2)
erb2 <- mean( (predict(b2, newdata = test2, type = "class") != test2$PHASEN) )
erl2 <- mean( (predict(l2, newdata = test2)$class != test2$PHASEN) )

# Runde 3
b3 <- rpart(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train3)
l3 <- lda(PHASEN ~ CP91JW + IAU91JW + IB91JW + GM1JW, data=train3)
erb3 <- mean( (predict(b3, newdata = test3, type = "class") != test3$PHASEN) )
erl3 <- mean( (predict(l3, newdata = test3)$class != test3$PHASEN) )

# Ergebnisse zusammenfassen
ercvb <- c(erb1, erb2, erb3)
ercvl <- c(erl1, erl2, erl3)

# Mittelwert des Prognose MSE
mean(ercvb)
#> [1] 0.49
mean(ercvl)
#> [1] 0.44

```

In der *Prognosegüte* ist hier – anders als in der Anpassungsgüte – die Lineare Diskriminanzanalyse besser.

## 0.54 Parameter `rpart`

Neben dem Splitkriterium können verschiedene Parameter des Algorithmus eingestellt werden (siehe `?rpart.control`), u. a.:

- minsplit: Minimale Anzahl Beobachtungen im Knoten damit Aufteilung versucht wird
- minbucket: Minimale Anzahl Beobachtungen im Blatt
- cp: Komplexitätspараметer (pruning)
- xval: Anzahl Kreuzvalidierungen (pruning)
- maxdepth: Maximale Tiefe eines Blattes

Diese können mit der Funktion `train` aus dem Paket `caret`<sup>36</sup> automatisch optimiert werden.

Alternativen/ Ergänzungen zu `rpart`:

- `tree`<sup>37</sup>
- `partykit`<sup>38</sup>
- Erweiterung: Viele Bäume: `randomForest`<sup>39</sup>

## 0.55 Fallstudie: 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?

### 0.55.1 Daten und Pakete laden

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

---

<sup>36</sup><https://topepo.github.io/caret/index.html>

<sup>37</sup><https://cran.r-project.org/web/packages/tree/>

<sup>38</sup><http://partykit.r-forge.r-project.org/partykit/>

<sup>39</sup><https://cran.r-project.org/web/packages/randomForest/>

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)
```

### 0.55.2 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, 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", "S", "Q", "S", "S", "C", ...
```

### 0.55.3 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.

### 0.55.4 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)?

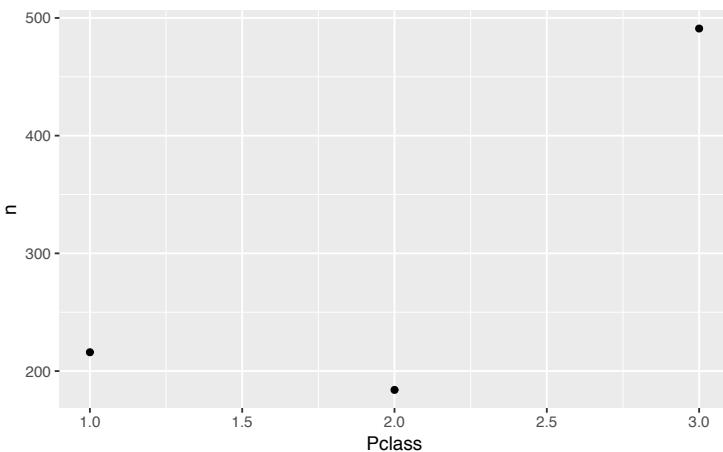
```
c1 <- dplyr::count(titanic_train, Pclass)
c1
#> # A tibble: 3 × 2
#>   Pclass     n
#>   <int> <int>
#> 1     1    216
#> 2     2    184
#> 3     3    491
```



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)
library(ggplot2)
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 × 2
#>   Survived     n
#>   <int> <int>
```

```
#> 1      0  549
#> 2      1  342
```

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

### 0.55.5 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`:

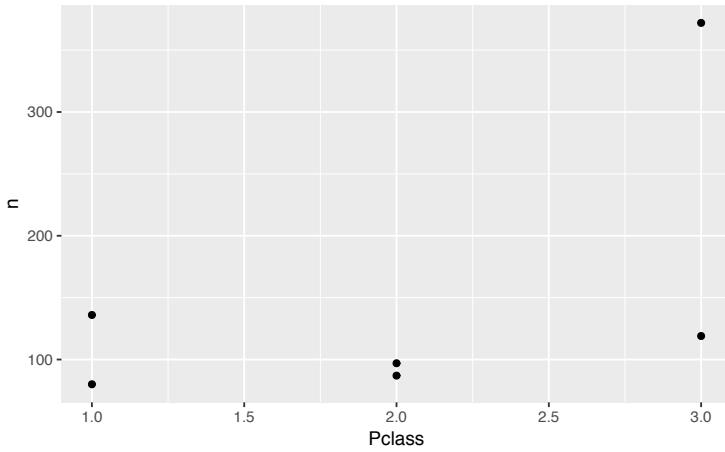
```
c3 <- dplyr::count(titanic_train, Survived, Pclass)
c3
#> Source: local data frame [6 x 3]
#> Groups: Survived [?]
#>
#>   Survived Pclass     n
#>   <int>    <int> <int>
#> 1      0      1    80
#> 2      0      2    97
#> 3      0      3   372
#> 4      1      1   136
#> 5      1      2    87
#>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

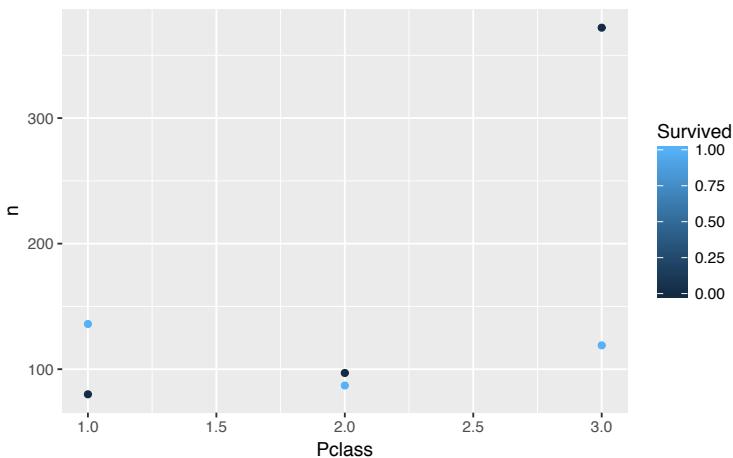
gleichen Befehl wie oben.

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



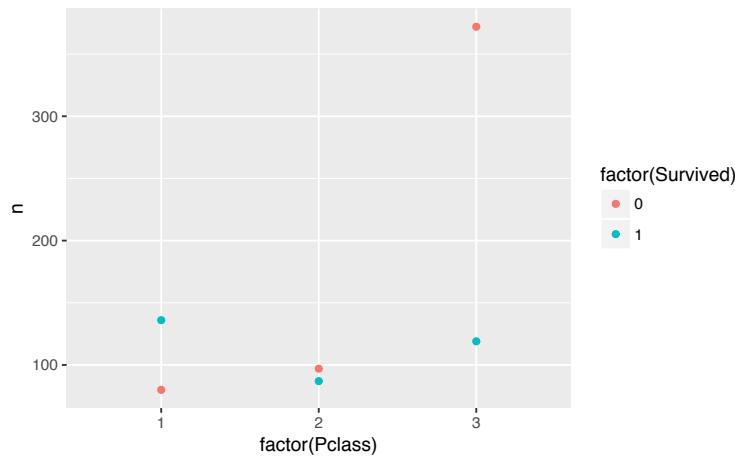
Hm, nicht so hilfreich. Schöner wäre, wenn wir (farblich) erkennen könnten, welcher Punkt für “Überlebt” und welcher Punkt für “Nicht-Überlebt” steht. Mit `qplot` geht das recht einfach: Wir sagen der Funktion `qplot`, dass die Farbe (`color`) der Punkte den Stufen von `Survived` zugeordnet werden sollen:

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



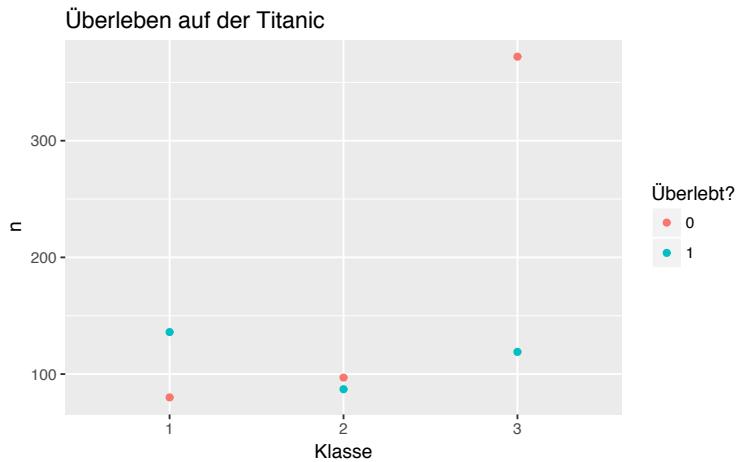
Viel besser. Was noch stört, ist, dass `Survived` als metrische Variable verstanden wird. Das Farbschema lässt Nuancen, feine Farbschattierungen, zu. Für nominale Variablen macht das keinen Sinn; es gibt da keine Zwischenstöne. Tot ist tot, lebendig ist lebendig. Wir sollten daher der Funktion sagen,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



### 0.55.6 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 seisdrum, 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  $\chi^2$ -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 H<sub>0</sub> und für die H<sub>1</sub>: “Es gibt einen Zusammenhang von Überlebensrate und Passagierklasse”.

### 0.55.7 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 × 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  $\chi^2$ -Wert von  $\sim 96$  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 live...

## 0.55.8 Logististische Regression

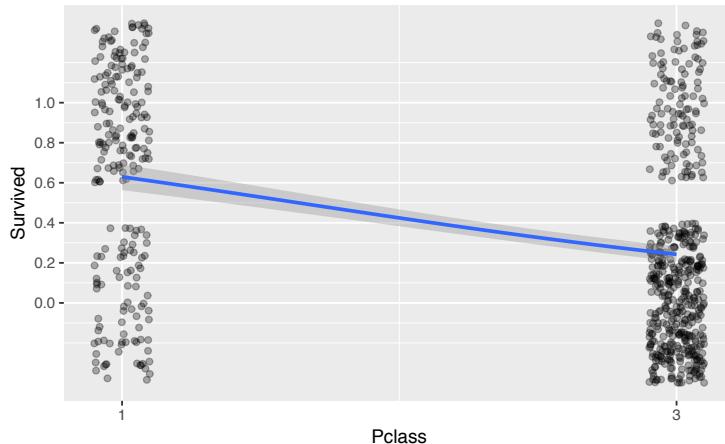
Berechnen wir noch das Odds Ratio mit Hilfe der logistischen Regression. Zum Einstieg: Ignorieren Sie die folgende Syntax und schauen Sie sich das Diagramm an. Hier sehen wir die (geschätzten) Überlebens-Wahrscheinlichkeiten für Passagiere der 1. Klasse vs. Passagiere der 3. Klasse.

```
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 (de-logarithmiert, `boa`). Wir sehen, dass die Überlebens-*Chance* (Odds) 1.7 zu 1 betrug - bei der *ersten* Stufe von `Pclass` (1)<sup>40</sup>; 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.

---

<sup>40</sup>Darum 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.

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  $\sim 0.2$ . Da der Faktor *kleiner* als 1 ist, ist das kein gutes Zeichen. Die Überlebens-Chance *sinkt*; etwas genauer auf:  $1.7 * 0.2 = 0.34$ . Das heißt, die Überlebens-Chance ist in der 3. Klasse nur noch ca. 1 zu 3 (Überlebens-Wahrscheinlichkeit:  $\sim 25\%$ ).

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
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]
#>
#>   Pclass Survived     n Anteil
#>   <int>    <int> <int>  <dbl>
#> 1     1        1     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.

### 0.55.9 Effektstärken visualieren

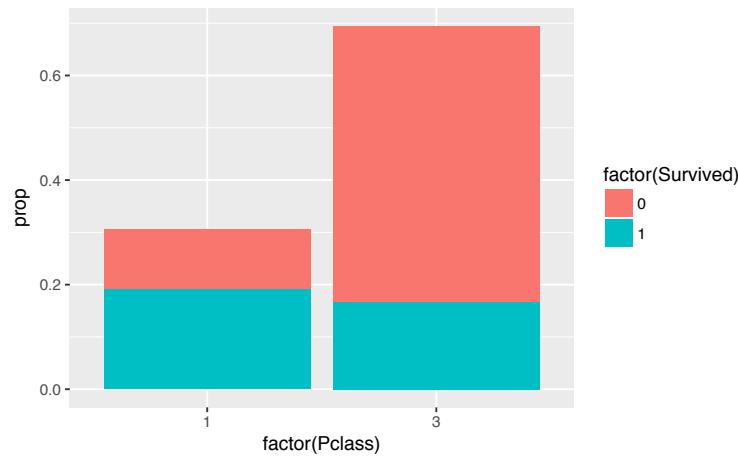
Zum Abschluss schauen wir uns die Stärke des Zusammenhangs noch einmal graphisch an. Wir berechnen dafür die relativen Häufigkeiten pro Gruppe (im Datensatz ohne 2. Klasse, der Einfachheit halber).

```
c5 <- dplyr::count(t2, Pclass, Survived)
c5$prop <- c5$n / 707
c5
#> Source: local data frame [4 x 4]
#> Groups: Pclass [?]
#>
#>   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.

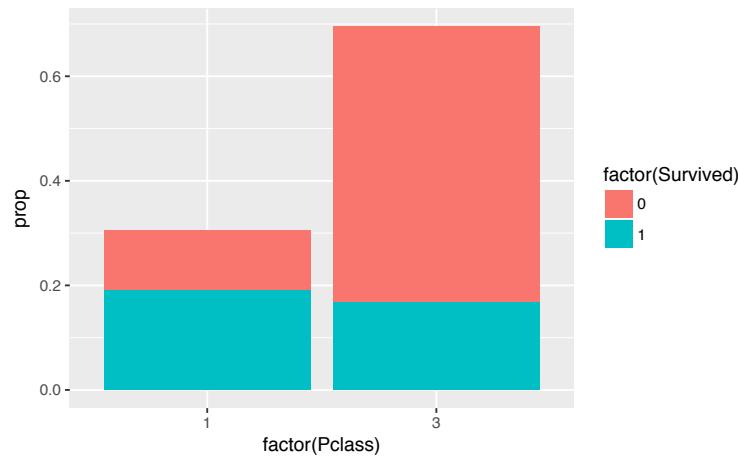
Das visualisieren wir wieder

```
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")
```

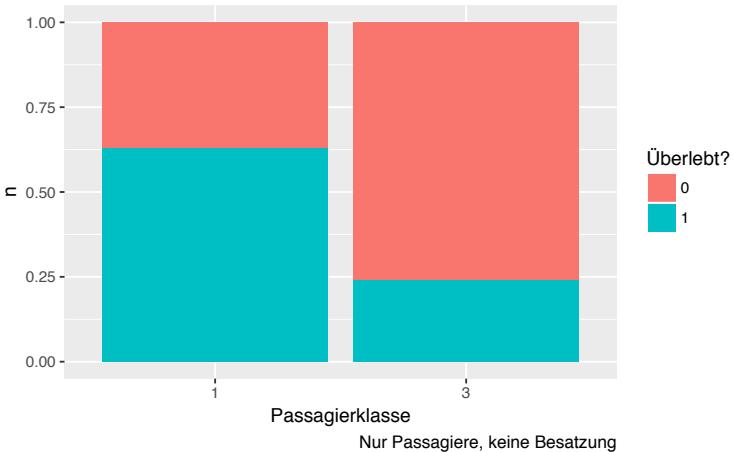


Ganz nett, aber die Häufigkeitsunterscheide von `Survived` zwischen den beiden Werten von `Pclass` stechen noch nicht so ins Auge. Wir sollten es anders darstellen.

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, k")
```

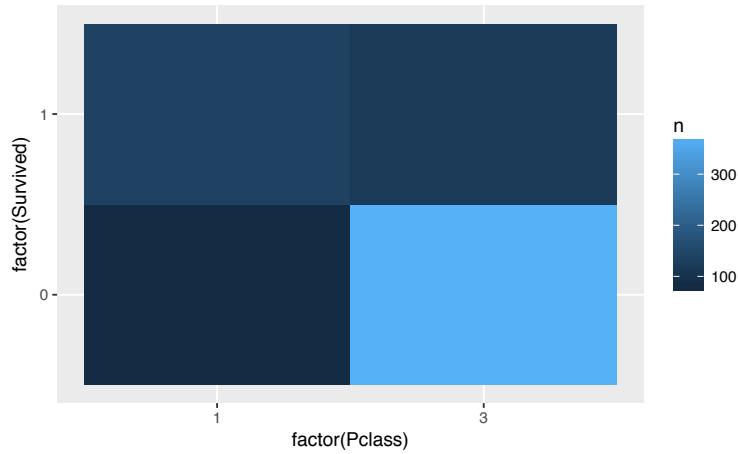


Jeden sehen wir die Häufigkeiten des Überlebens bedingt auf die Passagierklasse besser. Wir sehen auf den ersten Blick, dass sich die Überlebensraten deutlich unterscheiden: Im linken Balken überleben die meisten; im rechten ertrinken die meisten.

Diese letzte Analyse zeigt deutlich die Kraft von (Daten-)Visualisierungen auf. Der zu untersuchende Effekt tritt hier am stärksten 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.

### 0.55.10 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.



# **IV UNGELEITETES MODELLIEREN**



# Assoziationsanalyse

## 0.56 Einführung

Im Rahmen einer Assoziationsanalyse werden Zusammenhänge (Assoziationen) zwischen Ereignissen innerhalb von Datensätzen untersucht, z. B. im Rahmen einer Warenkorbanalyse. Dabei wird eine Menge (*Set*) von *Items* (z. B. gekauften Produkten,  $I = \{i_1, i_2, \dots, i_m\}$ ) innerhalb der *Transaktionen* ( $D = \{t_1, t_2, \dots, t_n\}$ ) betrachtet. Eine *Regel* ( $A \rightarrow B$ ) ist dann ein Zusammenhang zwischen den *Item Sets*.  $A$  und  $B$  sind dabei disjunkte, ggf. leere oder einelementige Teilmengen von Items ( $A, B \subseteq I; A \cap B = \emptyset$ ).

**Achtung:** Eine *Regel* impliziert keinen Kausalzusammenhang!

Die computationale Herausforderung einer Assoziationsanalyse besteht darin, bei vielen *Items* (und damit sehr, sehr vielen Kombinationsmöglichkeiten) innerhalb von vielen *Transaktionen* diejenigen *Sets* zu finden, die häufig vorkommen.

Eine wichtige Anwendung im (e)CRM ist z. B. die Analyse des *Cross Buying* Verhaltens.

Assoziationsregeln sind eine explorative Analysemethode, die keine (statistische) Inferenz oder Kausalität aussagt. D. h. evtl. Schlüsse aus der Analysen sollten vorsichtig schrittweise getestet und verglichen werden.

## 0.57 Kennzahlen einer Assoziationsanalyse

- $Support(A) = \frac{|\{t \in D; A \subseteq t\}|}{|D|}$ : Relative Häufigkeit, d. h. der Anteil der Transaktionen, in denen eine Menge von Items ( $A$ ) vorkommt, bezogen auf alle Transaktionen (vgl. Wahrscheinlichkeit,  $P(A)$ )
- $Support(A \rightarrow B) = Support(A \cup B) = \frac{|\{t \in D; (A \cup B) \subseteq t\}|}{|D|}$ : Relative Häufigkeit, d. h. der Anteil der Transaktionen, in denen die Vereinigung einer Menge von Items  $A$  und  $B$  vorkommt, bezogen auf alle Transaktionen (vgl. gemeinsame Wahrscheinlichkeit,  $P(A \cap B)$ )
- $Confidence(A \rightarrow B) = \frac{support(A \cup B)}{support(A)}$ : Relative Häufigkeit von  $B$  (rechte Seite), bezogen auf alle Transaktionen mit  $A$  (linke Seite; vgl. bedingte Wahrscheinlichkeit,  $P(B|A)$ )
- $Lift(A \rightarrow B) = \frac{support(A \cup B)}{support(A) \cdot support(B)}$ : Steigerung des Supports von  $A$  und  $B$  im Vergleich zur unter Unabhängigkeit von  $A$  und  $B$  erwarteten Häufigkeit (vgl.  $\frac{P(A \cap B)}{P(A) \cdot P(B)} = \frac{P(B|A)}{P(B)}$ )

Hinweis: Support und Lift sind symmetrisch, Confidence nicht. Als Schätzwerte für Wahrscheinlichkeiten etc. können die genannten Kennzahlen ggfs. ungeeignet sein (insbesondere für seltene Items).

## 0.58 Beispiele

- $Support(\text{Chips}) = 0.05$ , d. h. 5% der Transaktionen enthalten Chips
- $Support(\text{Bier}) = 0.01$ , d. h. 1% der Transaktionen enthalten Bier
- $Support(\text{Bier und Chips}) = 0.002$ , d. h. 0,2% der Transaktionen enthalten Bier und Chips, dann:
- $Confidence(\text{Bier} \rightarrow \text{Chips}) = \frac{0.002}{0.01} = 0.2$ , d. h. 20% aller Transaktionen mit Bier enthalten auch Chips
- $Lift(\text{Bier} \rightarrow \text{Chips}) = \frac{0.002}{0.01 \cdot 0.05} = \frac{0.2}{0.05} = 4$ , d. h. die Chance, dass eine Transaktion Chips enthält ist 4x größer als zu erwarten wäre, wenn es keinen Zusammenhang zwischen Bier und Chips gäbe

## 0.59 Assoziationsanalyse mit R

Für eine Assoziationsanalyse kann in R das Zusatzpaket `arules` <https://cran.r-project.org/package=arules> verwendet werden.

Die (einmalige) Installation erfolgt über:

```
install.packages(c("arules", "arulesViz"), dep = TRUE)
```

Geladen wird das Paket dann über

```
library(arules)
```

Eine Einführung erhält man über die Paket-Vignette

```
vignette("arules")
```

## 0.60 Fallstudie Lebensmittelwaren

Im Paket `arules` sind Point-Of-Sale Daten eines Lebensmittelgeschäfts von einem Monat enthalten.<sup>41</sup> Die Lebensmittel wurden zu 169 Kategorien zusammengefasst, und es gibt 9835 Transaktionen:

```
data(Groceries, package = "arules") # Daten laden
# load(file = "data/Groceries.Rda")
summary(Groceries)
#> transactions as itemMatrix in sparse format with
#> 9835 rows (elements/itemsets/transactions) and
```

---

<sup>41</sup> Michael Hahsler, Kurt Hornik, und Thomas Reutterer (2006) *Implications of probabilistic data modeling for mining association rules*. In: M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, und W. Gaul, Editors, From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization, Seiten 598–605. Springer-Verlag.

```

#> 169 columns (items) and a density of 0.0261
#>
#> most frequent items:
#>      whole milk other vegetables      rolls/buns      soda
#>      2513           1903           1809      1715
#>      yogurt          (Other)
#>      1372           34055
#>
#> element (itemset/transaction) length distribution:
#> sizes
#>   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
#> 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55
#>   16  17  18  19  20  21  22  23  24  26  27  28  29  32  33
#>   46  29  14  14   9  11    4    6    1    1    1    1    3    1
#>
#>      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
#>      1.0    2.0    3.0    4.4    6.0    32.0
#>
#> includes extended item information - examples:
#>      labels level2      level1
#> 1 frankfurter sausage meat and sausage
#> 2 sausage sausage meat and sausage
#> 3 liver loaf sausage meat and sausage

```

Hinweis: Um einen Datensatz als Transaktionsdatensatz zu definieren wird der Befehl

```

library(methods)
daten.trans <- methods::as(Groceries, "transactions")

```

verwendet. Siehe auch Hilfeseite zu `transactions-class`.

Über `inspect()` können Transaktionen und Regeln betrachtet werden:

```

arules::inspect(head(daten.trans))
#>      items

```

```
#> [1] {citrus fruit,
#>       semi-finished bread,
#>       margarine,
#>       ready soups}
#> [2] {tropical fruit,
#>       yogurt,
#>       coffee}
#> [3] {whole milk}
#> [4] {pip fruit,
#>       yogurt,
#>       cream cheese ,
#>       meat spreads}
#> [5] {other vegetables,
#>       whole milk,
#>       condensed milk,
#>       long life bakery product}
#> [6] {whole milk,
#>       butter,
#>       yogurt,
#>       rice,
#>       abrasive cleaner}
```

### 0.60.1 Regeln finden

Es existieren verschiedene Algorithmen um Assoziationsregeln zu finden. Hier wird der *Apriori* Algorithmus verwendet, wobei verschiedene Parameter (wie z. B. minimalen Support und Confidence) eingestellt werden können:

```
lebensmittel.regeln <- apriori(Groceries,
                                 parameter=list(supp=0.02, conf=0.1,
                                                target="rules"))
#> Apriori
#>
#> # Parameter specification:
```

```

#> confidence minval smax arem aval originalSupport maxtime support minlen
#>          0.1    0.1     1 none FALSE                      TRUE      5   0.02    1
#> maxlen target ext
#>       10 rules FALSE
#>
#> Algorithmic control:
#> filter tree heap memopt load sort verbose
#>      0.1 TRUE TRUE FALSE TRUE     2     TRUE
#>
#> Absolute minimum support count: 196
#>
#> set item appearances ... [0 item(s)] done [0.00s].
#> set transactions ... [169 item(s), 9835 transaction(s)] done [0.00s].
#> sorting and recoding items ... [59 item(s)] done [0.00s].
#> creating transaction tree ... done [0.00s].
#> checking subsets of size 1 2 3 done [0.00s].
#> writing ... [128 rule(s)] done [0.00s].
#> creating S4 object ... done [0.00s].

```

```

inspect(subset(lebensmittel.regeln, lift>2.5))
#>   lhs                           rhs           support confidence
#> [1] {pip fruit}                 => {tropical fruit} 0.0204  0.270
#> [2] {tropical fruit}           => {pip fruit}      0.0204  0.195
#> [3] {other vegetables,whole milk} => {root vegetables} 0.0232  0.310
#>   lift
#> [1] 2.57
#> [2] 2.57
#> [3] 2.84

```

Das Ergebnis zeigt, dass in ca. 2% der Transaktionen Kern- (“pip”, z. B. Äpfel, Birnen) und Südfrüchte (“tropical”, z. B. Zitronen) enthalten waren (*support*). 27% der Transaktionen mit Kernfrüchten (*lhs*) enthielten auch Südfrüchte (*rhs, confidence*). Wenn also eine Transaktion eine Kernfrucht enthält ist es 2,57x häufiger, dass die Transaktion auch Südfrüchte enthält unter unabhängigen Häufigkeiten (*lift*).

Um die “Top” Regeln zu betrachten müssen die Regeln nach dem gewünschten Kriterium sortiert werden:

```
topregeln <- head(sort(lebensmittel.regeln, by="confidence"), 20)
inspect(topregeln)

#>      lhs                                rhs          support
#> [1] {other vegetables,yogurt}           => {whole milk} 0.0223
#> [2] {butter}                            => {whole milk} 0.0276
#> [3] {curd}                             => {whole milk} 0.0261
#> [4] {root vegetables,other vegetables} => {whole milk} 0.0232
#> [5] {root vegetables,whole milk}         => {other vegetables} 0.0232
#> [6] {domestic eggs}                     => {whole milk} 0.0300
#> [7] {whipped/sour cream}                => {whole milk} 0.0322
#> [8] {root vegetables}                   => {whole milk} 0.0489
#> [9] {root vegetables}                   => {other vegetables} 0.0474
#> [10] {frozen vegetables}                => {whole milk} 0.0204
#> [11] {margarine}                       => {whole milk} 0.0242
#> [12] {beef}                            => {whole milk} 0.0213
#> [13] {tropical fruit}                 => {whole milk} 0.0423
#> [14] {whipped/sour cream}              => {other vegetables} 0.0289
#> [15] {yogurt}                          => {whole milk} 0.0560
#> [16] {pip fruit}                      => {whole milk} 0.0301
#> [17] {whole milk,yogurt}               => {other vegetables} 0.0223
#> [18] {brown bread}                    => {whole milk} 0.0252
#> [19] {other vegetables}              => {whole milk} 0.0748
#> [20] {pork}                           => {whole milk} 0.0222

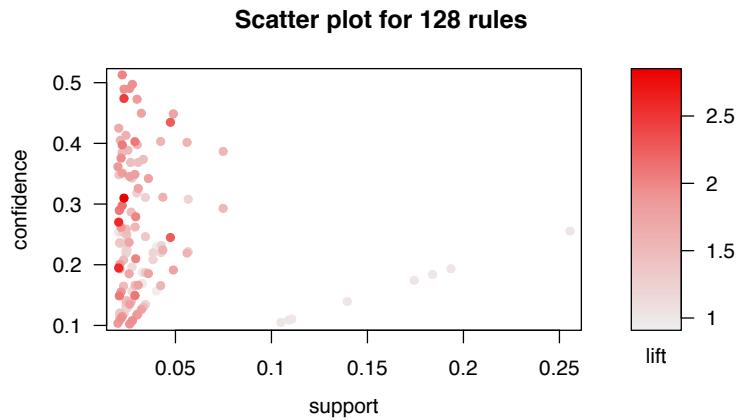
#>      confidence   lift
#> [1] 0.513        2.01
#> [2] 0.497        1.95
#> [3] 0.490        1.92
#> [4] 0.489        1.91
#> [5] 0.474        2.45
#> [6] 0.473        1.85
#> [7] 0.450        1.76
#> [8] 0.449        1.76
#> [9] 0.435        2.25
#> [10] 0.425       1.66
```

```
#> [11] 0.413      1.62
#> [12] 0.405      1.59
#> [13] 0.403      1.58
#> [14] 0.403      2.08
#> [15] 0.402      1.57
#> [16] 0.398      1.56
#> [17] 0.397      2.05
#> [18] 0.389      1.52
#> [19] 0.387      1.51
#> [20] 0.384      1.50
```

## 0.60.2 Visualisierung

Eine mögliche Visualisierung ist ein Streudiagramm von Support und Confidence

```
library(arulesViz)
plot(lebensmittel.regeln)
```

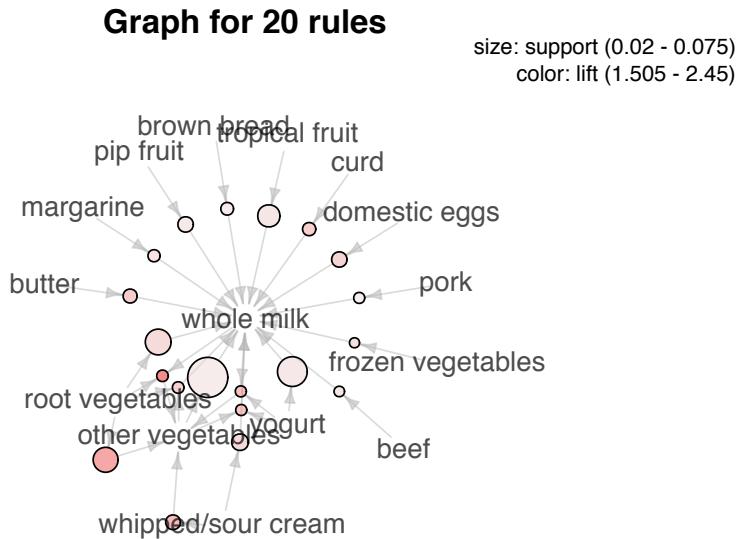


Mit Hilfe der Option `interactive=TRUE` kann in Bereiche gezoomt werden – und Regeln ausgewählt:

```
library(arulesViz)
plot(lebensmittel.regeln, interactive=TRUE)
```

Aber auch z. B. ein Graph eines entsprechenden Netzwerks ist möglich:

```
plot(topregeln, method="graph")
```



## 0.61 Literatur

- Chris Chapman und Elea McDonnell Feit (2015), *R for Marketing Research and Analytics*, Springer<sup>42</sup>
- Michael Hahsler (2015), A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, URL: [http://michael.hahsler.net/research/association\\_rules/measures.html](http://michael.hahsler.net/research/association_rules/measures.html)<sup>43</sup>
- Michael Hahsler, Sudheer Chelluboina, Kurt Hornik, und Christian Buchta (2011), *The arules R-package ecosystem: Analyzing interesting*

<sup>42</sup><http://r-marketing.r-forge.r-project.org>

<sup>43</sup>[http://michael.hahsler.net/research/association\\_rules/measures.html](http://michael.hahsler.net/research/association_rules/measures.html)

*patterns from large transaction datasets.* Journal of Machine Learning Research, 12:1977–1981<sup>44</sup>

- Michael Hahsler, Bettina Gruen und Kurt Hornik (2005), *arules - A Computational Environment for Mining Association Rules and Frequent Item Sets.* Journal of Statistical Software 14/15.<sup>45</sup>
- Michael Hahsler, Kurt Hornik, und Thomas Reutterer (2006), *Implications of probabilistic data modeling for mining association rules.* In: M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, und W. Gaul, Editors, From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization, Seiten 598–605. Springer-Verlag.

## 0.62 Versionshinweise:

Die Darstellung orientiert sich an den Folienunterlagen von Chapman & Feit zum Buch *R for Marketing Research and Analytics*, Springer, 2015, siehe <http://r-marketing.r-forge.r-project.org/Instructor/slides-index.html>

---

<sup>44</sup><http://jmlr.csail.mit.edu/papers/v12/hahsler11a.html>

<sup>45</sup><http://dx.doi.org/10.18637/jss.v014.i15>

# Clusteranalyse

Wir werden einen *simulierten* Datensatz aus *Chapman & Feit (2015): R for Marketing Research and Analytics*. Springer analysieren (<http://r-marketing.r-forge.r-project.org>). Näheres dazu siehe Kapitel 5 dort.

Sie können ihn von hier<sup>46</sup> als csv-Datei herunterladen:

```
#download.file("https://goo.gl/eUm8PI", destfile = "segment.csv")
```

Das Einlesen erfolgt, sofern die Daten im Arbeitsverzeichnis liegen, wieder über:

```
segment <- read.csv2("https://goo.gl/eUm8PI")
```

## Ein Überblick über die Daten:

```
str(segment)
#> 'data.frame': 300 obs. of 7 variables:
#> $ Alter : num 50.2 40.7 43 40.3 41.1 ...
#> $ Geschlecht : Factor w/ 2 levels "Frau", "Mann": 2 2 1 2 1 2 1 2 1 1 ...
#> $ Einkommen : num 51356 64411 71615 42728 71641 ...
#> $ Kinder : int 0 3 2 1 4 2 5 1 1 0 ...
#> $ Eigenheim : Factor w/ 2 levels "Ja", "Nein": 2 2 1 2 2 1 2 2 2 2 ...
#> $ Mitgliedschaft: Factor w/ 2 levels "Ja", "Nein": 2 2 2 2 2 2 1 1 2 2 ...
#> $ Segment : Factor w/ 4 levels "Aufsteiger", "Gemischte Vorstadt", ...: 2 2 ...
```

<sup>46</sup><https://goo.gl/eUm8PI>

```
head(segment)
#>   Alter Geschlecht Einkommen Kinder Eigenheim Mitgliedschaft
#> 1  50.2       Mann     51356      0     Nein      Nein
#> 2  40.7       Mann     64411      3     Nein      Nein
#> 3  43.0       Frau    71615      2      Ja       Nein
#> 4  40.3       Mann     42728      1     Nein      Nein
#> 5  41.1       Frau    71641      4     Nein      Nein
#> 6  40.2       Mann     60325      2      Ja       Nein
#> 
#>           Segment
#> 1 Gemischte Vorstadt
#> 2 Gemischte Vorstadt
#> 3 Gemischte Vorstadt
#> 4 Gemischte Vorstadt
#> 5 Gemischte Vorstadt
#> 6 Gemischte Vorstadt
```

Zur Unterstützung der Analyse wird (wieder) `mosaic` und `tidyverse` verwendet:

```
library(tidyverse)
library(mosaic)
```

Das Ziel einer Clusteranalyse ist es, Gruppen von Beobachtungen (d. h. *Cluster*) zu finden, die innerhalb der Cluster möglichst homogen, zwischen den Clustern möglichst heterogen sind. Um die Ähnlichkeit von Beobachtungen zu bestimmen, können verschiedene Distanzmaße herangezogen werden. Für metrische Merkmale wird z. B. häufig die euklidische Metrik verwendet, d. h., Ähnlichkeit und Distanz werden auf Basis des euklidischen Abstands bestimmt. Aber auch andere Abstände wie Manhatten oder Gower sind möglich. Letztere haben den Vorteil, dass sie nicht nur für metrische Daten sondern auch für gemischte Variablentypen verwendet werden können.

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()`). Die Funktion `daisy()` aus dem Paket `cluster` bietet hier nützliche Möglichkeiten.

```
library(cluster)

daisy(head(segment))
#> Dissimilarities :
#>      1     2     3     4     5
#> 2 0.307
#> 3 0.560 0.390
#> 4 0.219 0.184 0.502
#> 5 0.516 0.220 0.242 0.404
#> 6 0.401 0.206 0.239 0.268 0.426
#>
#> Metric : mixed ; Types = I, N, I, I, N, N, N
#> Number of objects : 6
```

## 0.63 Hierarchische Clusteranalyse

Bei hierarchischen Clusterverfahren werden Beobachtungen sukzessiv zusammengefasst (agglomerativ). Zunächst ist jede Beobachtung ein eigener Clus-

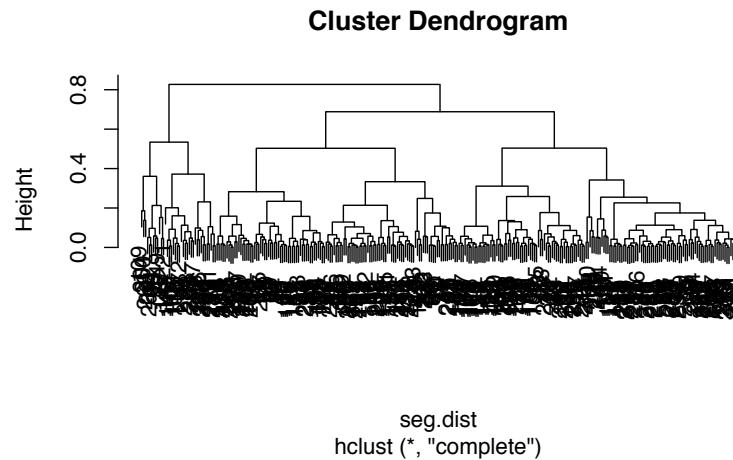
ter, die dann je nach Ähnlichkeitsmaß zusammengefasst werden.

Fassen wir die Beobachtungen *ohne* die Segmentvariable Segment, Variable 7, zusammen:

```
seg.dist <- daisy(segment[,-7]) # Abstände
seg.hc <- hclust(seg.dist) # Hierarchische Clusterung
```

Das Ergebnis lässt sich schön im Dendrogramm darstellen:

```
plot(seg.hc)
```



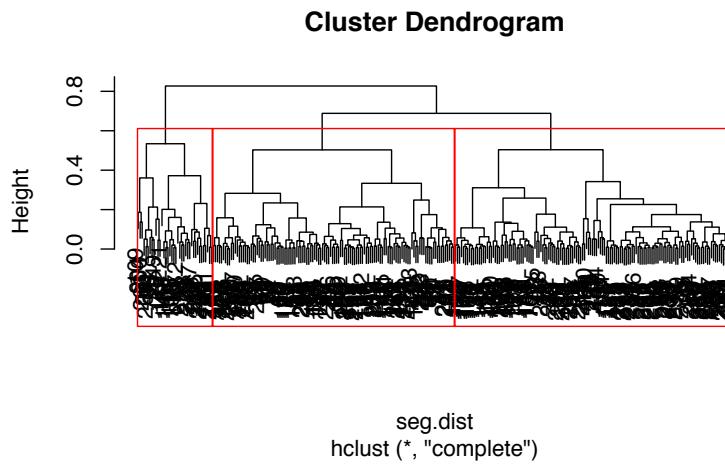
Je höher (**Height**) die Stelle ist, an der zwei Beobachtungen oder Cluster zusammengefasst werden, desto größer ist die Distanz. D. h., Beobachtungen bzw. Cluster, die unten zusammengefasst werden, sind sich ähnlich, die, die oben zusammengefasst werden unähnlich.

Hier wurde übrigens die Standardeinstellung für die Berechnung des Abstands von Clustern verwendet: Complete Linkage bedeutet, dass die Distanz zwischen zwei Clustern auf Basis des maximalen Abstands der Beobachtungen innerhalb des Clusters gebildet wird.

Es ist nicht immer einfach zu entscheiden, wie viele Cluster es gibt. In der Praxis und Literatur finden sich häufig Zahlen zwischen 3 und 10. Evt. gibt es im Dendrogramm eine Stelle, an der der Baum gut geteilt werden kann.

In unserem Fall vielleicht bei einer Höhe von 0.6, da sich dann 3 Cluster ergeben:

```
plot(seg.hc)
rect.hclust(seg.hc, h=0.6, border="red")
```



Das Ergebnis, d. h. die Clusterzuordnung, kann durch den Befehl `cutree()` den Beobachtungen zugeordnet werden.

```
segment$hc.clust <- cutree(seg.hc, k=3)
```

Z. B. haben wir folgende Anzahlen für Beobachtungen je Cluster:

```
mosaic::tally(~hc.clust, data=segment)
#> hc.clust
#>   1   2   3
#> 140 122 38
```

Cluster 3 ist also mit Abstand der kleinste Cluster (mit 38 Beobachtungen).

Für den Mittelwert des Alters je Cluster gilt:

```
segment %>%
```

```

group_by(hc.clust) %>%
  summarise(Alter_nach_Cluster = mean(Alter))
#> # A tibble: 3 × 2
#>   hc.clust Alter_nach_Cluster
#>   <int>           <dbl>
#> 1     1            38.5
#> 2     2            46.4
#> 3     3            34.5

```

D. h. die Durchschnittsalter ist in Cluster der Cluster unterscheiden sich.

Das spiegelt sich auch im Einkommen wieder:

```

segment %>%
  group_by(hc.clust) %>%
  summarise(Einkommen_nach_Cluster = mean(Einkommen))
#> # A tibble: 3 × 2
#>   hc.clust Einkommen_nach_Cluster
#>   <int>           <dbl>
#> 1     1            49452
#> 2     2            54355
#> 3     3            44113

```

Allerdings sind die Unterschiede in der Geschlechtsverteilung eher gering:

```

mosaic::tally(Geschlecht~hc.clust, data=segment, format="proportion")
#> 
#>   hc.clust
#>   Geschlecht    1    2    3
#>   Frau  0.543  0.549  0.526
#>   Mann  0.457  0.451  0.474

```

## 0.64 k-Means Clusteranalyse

Beim k-Means Clusterverfahren handelt es sich im Gegensatz zur 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`<sup>47</sup>.

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, -hc.clust)
```

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

---

<sup>47</sup><https://cran.r-project.org/web/packages/clustMixType/index.html>

Aufgrund von (1.) hängt das Ergebnis einer k-Means Clusteranalyse vom Zufall ab. Aus Gründen der Reproduzierbarkeit sollte daher der Zufallszahlen-generator 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:

```
set.seed(1896)

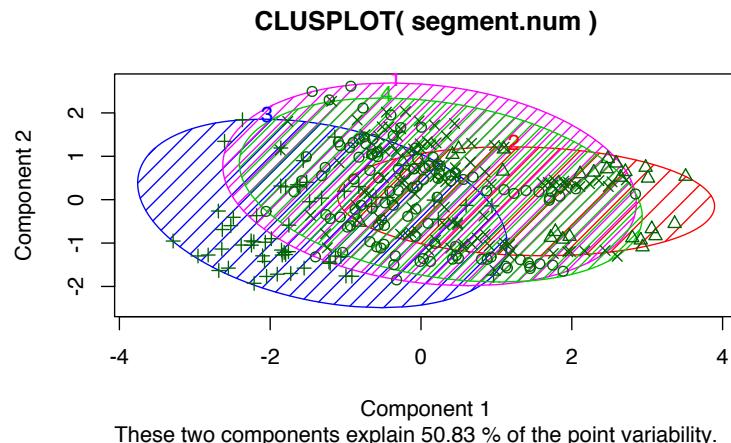
seg.k <- kmeans(segment.num, centers = 4, nstart = 10)
seg.k
#> K-means clustering with 4 clusters of sizes 111, 26, 58, 105
#>
#> Cluster means:
#>   Alter Einkommen Kinder Eigenheim Mitgliedschaft Frau
#> 1  42.9      46049  1.649      0.505      0.1081 0.568
#> 2  56.4      85973  0.385      0.538      0.0385 0.538
#> 3  27.0     22608  1.224      0.276      0.2069 0.414
#> 4  43.6     62600  1.505      0.457      0.1238 0.590
#>
#> Clustering vector:
#> [1] 1 4 4 1 4 4 4 1 2 4 1 1 4 4 1 1 1 1 4 4 4 1 4 1 1 1 1 4 1 4 4 1 1
#> [36] 1 4 1 1 4 4 4 1 4 4 4 4 1 1 1 1 1 2 1 1 4 4 4 4 1 4 1 4 1 1 1 4 4
#> [71] 4 1 1 4 1 1 4 4 4 4 1 4 1 3 1 4 1 1 1 1 4 4 4 4 1 1 4 4 4 3 3 3 3
#> [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
#> [141] 3 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
#> [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
#> [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 1 3 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 1
#> [281] 4 4 4 4 1 4 1 4 4 4 1 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.

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?<sup>48</sup>

---

<sup>48</sup>Das 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.

## 0.65 Übung: B3 Datensatz

Der B3 Datensatz *Heilemann, U. and Münch, H.J. (1996): West German Business Cycles 1963-1994: A Multivariate Discriminant Analysis. CIRET-Conference in Singapore, CIRET-Studien 50.* enthält Quartalsweise Konjunkturdaten aus (West-)Deutschland.

Er kann von <https://goo.gl/OYCEHf> heruntergeladen werden.

1. Wenn die Konjunkturphase PHASEN nicht berücksichtigt wird, wie viele Cluster könnte es geben? Ändert sich das Ergebnis, wenn die Variablen standardisiert werden?
2. Führen Sie eine k-Means Clusteranalyse mit 4 Clustern durch. Worin unterscheiden sich die gefundenen Segmente?

### 0.65.1 Literatur

- Chris Chapman, Elea McDonnell Feit (2015): *R for Marketing Research and Analytics*, Kapitel 11.3
- Reinhold Hatzinger, Kurt Hornik, Herbert Nagel (2011): *R – Einführung durch angewandte Statistik*. Kapitel 12
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013): *An Introduction to Statistical Learning – with Applications in R*, <http://www-bcf.usc.edu/~gareth/ISL/>, Kapitel 10.3, 10.5

---

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 Unported<sup>49</sup>. Der Code steht unter der Apache Lizenz 2.0<sup>50</sup>

---

<sup>49</sup><http://creativecommons.org/licenses/by-sa/3.0>

<sup>50</sup><http://www.apache.org/licenses/LICENSE-2.0>

# Dimensionsreduktion

Datensätze in den Sozialwissenschaften haben oft viele Variablen - oder auch Dimensionen. Es ist vorteilhaft, diese auf eine kleinere Anzahl von Variablen (oder Dimensionen) zu reduzieren: Zusammenhänge zwischen Konstrukten können so klarer identifiziert werden.

In diese Übung 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 (PCA)* versucht, unkorrelierte Linear-kombinationen zu finden, die die maximale Varianz in den Daten erfassen. Die Blickrichtung ist von den Daten zu den Komponenten.
- 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. Die Blickrichtung ist von den Faktoren zu den Daten.

## 0.66 Einführung

### 0.66.1 Gründe für die Notwendigkeit der Datenreduktion

- Im technischen Sinne der Dimensionsreduktion können wir statt Variablen-Sets die Faktorwerte verwenden.

- Wir können Unsicherheit verringern. Wenn wir glauben, dass ein Konstrukt nicht eindeutig messbar ist, dann kann mit einem Variablen-Set die Unsicherheit reduziert werden.
- Wir könnten den Aufwand bei der Datenerfassung vereinfachen, indem wir uns auf Variablen konzentrieren, von denen bekannt ist, dass sie einen hohen Beitrag zum interessierenden Faktor leisten. Wenn wir feststellen, dass einige Variablen für einen Faktor nicht wichtig sind, können wir sie aus dem Datensatz eliminieren.

### 0.66.2 Benötigte Pakete

Pakete, die für die Datenanalyse benötigt werden, müssen vorher einmalig in R installiert werden.

```
# install.packages("corrplot")
# install.packages("gplots")
# install.packages("scatterplot3d")
```

```
library("corrplot")
library("gplots")
library("scatterplot3d")
library("tidyverse")
```

### 0.66.3 Daten einlesen

Wir untersuchen die Dimensionalität mittels eines simulierten Datensatzes der typisch für die Wahrnehmung von Umfragen ist. Die Daten spiegeln Verbraucherbewertungen von Marken in Bezug auf Adjektive wieder, die in Umfragen in folgender Form abgefragt werden:

Auf einer Skala von 1 bis 10 (wobei 1 am wenigsten und 10 am meisten zutrifft)

wie...[ADJECTIV]... ist ...[Marke A]...?

Wir verwenden einen *simulierten* Datensatz aus *Chapman & Feit (2015): R for Marketing Research and Analytics. Springer (<http://r-marketing.r-forge.r-project.org>)*. Die Daten umfassen simulierte Bewertungen von 10 Marken (“a” bis “j”) mit 9 Adjektiven (“performance”, “leader”, “latest”, “fun” usw.) für n = 100 simulierte Befragte.

Das Einlesen der Daten erfolgt direkt über das Internet.

```
brand.ratings <- read.csv("http://goo.gl/IQl8nc")
```

Wir überprüfen zuerst die Struktur des Datensatzes, die ersten 6 Zeilen und die Zusammenfassung

```
str(brand.ratings)
#> 'data.frame': 1000 obs. of 10 variables:
#> $ perform: int 2 1 2 1 1 2 1 2 2 3 ...
#> $ leader : int 4 1 3 6 1 8 1 1 1 1 ...
#> $ latest : int 8 4 5 10 5 9 5 7 8 9 ...
#> $ fun    : int 8 7 9 8 8 5 7 5 10 8 ...
#> $ serious: int 2 1 2 3 1 3 1 2 1 1 ...
#> $ bargain: int 9 1 9 4 9 8 5 8 7 3 ...
#> $ value  : int 7 1 5 5 9 7 1 7 7 3 ...
#> $ trendy : int 4 2 1 2 1 1 1 7 5 4 ...
#> $ rebuy  : int 6 2 6 1 1 2 1 1 1 1 ...
#> $ brand  : Factor w/ 10 levels "a","b","c","d",...: 1 1 1 1 1 1 1 1 1 1 ...
head(brand.ratings)
#>   perform leader latest fun serious bargain value trendy rebuy brand
#> 1      2      4      8      8      2      9      7      4      6      a
#> 2      1      1      4      7      1      1      1      2      2      a
#> 3      2      3      5      9      2      9      5      1      6      a
#> 4      1      6     10      8      3      4      5      2      1      a
#> 5      1      1      5      8      1      9      9      1      1      a
#> 6      2      8      9      5      3      8      7      1      2      a
summary(brand.ratings)
#>   perform           leader          latest          fun 
#> Min.   : 1.00   Min.   : 1.00   Min.   : 1.0   Min.   : 1.00 
#> 1st Qu.: 1.00   1st Qu.: 2.00   1st Qu.: 4.0   1st Qu.: 4.00
```

```

#> Median : 4.00   Median : 4.00   Median : 7.0   Median : 6.00
#> Mean   : 4.49   Mean   : 4.42   Mean   : 6.2   Mean   : 6.07
#> 3rd Qu.: 7.00   3rd Qu.: 6.00   3rd Qu.: 9.0   3rd Qu.: 8.00
#> Max.   :10.00   Max.   :10.00   Max.   :10.0   Max.   :10.00
#>
#>      serious       bargain       value       trendy
#> Min.   : 1.00   Min.   : 1.00   Min.   : 1.00   Min.   : 1.00
#> 1st Qu.: 2.00   1st Qu.: 2.00   1st Qu.: 2.00   1st Qu.: 3.00
#> Median : 4.00   Median : 4.00   Median : 4.00   Median : 5.00
#> Mean   : 4.32   Mean   : 4.26   Mean   : 4.34   Mean   : 5.22
#> 3rd Qu.: 6.00   3rd Qu.: 6.00   3rd Qu.: 6.00   3rd Qu.: 7.00
#> Max.   :10.00   Max.   :10.00   Max.   :10.00   Max.   :10.00
#>
#>      rebuy        brand
#> Min.   : 1.00   a     :100
#> 1st Qu.: 1.00   b     :100
#> Median : 3.00   c     :100
#> Mean   : 3.73   d     :100
#> 3rd Qu.: 5.00   e     :100
#> Max.   :10.00   f     :100
#>                  (Other):400

```

Jeder der 100 simulierten Befragten beurteilt 10 Marken, das ergibt insgesamt 1000 Beobachtungen (Zeilen) im Datensatz.

Wir sehen in der `summary()`, dass die Bereiche der Bewertungen für jedes Adjektiv 1-10 sind. In `str()` sehen wir, dass die Bewertungen als numerisch eingelesen wurden, während die Markennamen als Faktoren eingelesen wurden. Die Daten sind somit richtig formatiert.

#### 0.66.4 Neuskalierung der Daten

In vielen Fällen ist es sinnvoll, Rohdaten neu zu skalieren. Dies wird üblicherweise als **Standardisierung**, **Normierung**, oder **Z Scoring/ Transformation** bezeichnet. Als Ergebnis ist der Mittelwert aller Variablen über alle Beobachtungen dann 0. 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 `brand.sc` zu und skalieren anschließend die Daten. Wir skalieren in unserem Datensatz nur die ersten 9 Variablen, weil die 10. Variable der Faktor für die Markenamen ist.

```
brand.sc <- brand.ratings

brand.ratings %>%
  mutate_each(funs(scale), -brand) -> brand.sc

summary(brand.sc)
#>      perform.V1        leader.V1        latest.V1        fun.V1
#>  Min.   :-1.089   Min.   :-1.310   Min.   :-1.688   Min.   :-1.847
#>  1st Qu.:-1.089   1st Qu.:-0.927   1st Qu.:-0.713   1st Qu.:-0.754
#>  Median :-0.152   Median :-0.160   Median : 0.262   Median :-0.025
#>  Mean    : 0.000   Mean    : 0.000   Mean    : 0.000   Mean    : 0.000
#>  3rd Qu.: 0.784   3rd Qu.: 0.607   3rd Qu.: 0.911   3rd Qu.: 0.704
#>  Max.    : 1.721   Max.    : 2.140   Max.    : 1.236   Max.    : 1.433
#>
#>      serious.V1        bargain.V1        value.V1        trendy.V1
#>  Min.   :-1.196   Min.   :-1.222   Min.   :-1.391   Min.   :-1.539
#>  1st Qu.:-0.836   1st Qu.:-0.847   1st Qu.:-0.974   1st Qu.:-0.810
#>  Median :-0.116   Median :-0.097   Median :-0.140   Median :-0.080
#>  Mean    : 0.000   Mean    : 0.000   Mean    : 0.000   Mean    : 0.000
#>  3rd Qu.: 0.604   3rd Qu.: 0.653   3rd Qu.: 0.693   3rd Qu.: 0.649
#>  Max.    : 2.043   Max.    : 2.153   Max.    : 2.361   Max.    : 1.743
#>
#>      rebuy.V1        brand
#>  Min.   :-1.072   a       :100
#>  1st Qu.:-1.072   b       :100
#>  Median :-0.286   c       :100
#>  Mean    : 0.000   d       :100
#>  3rd Qu.: 0.500   e       :100
#>  Max.    : 2.465   f       :100
```

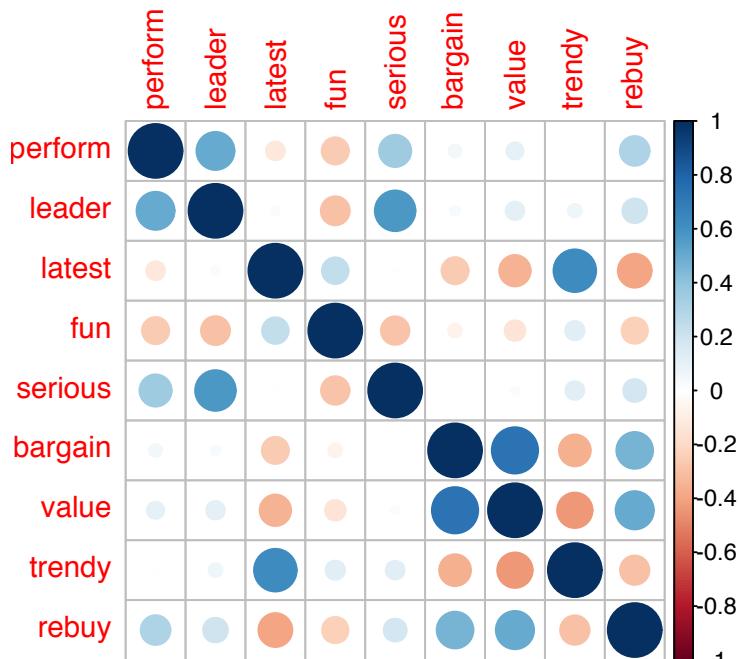
#> (Other):400

Die Daten wurden offenbar richtig skaliert, da Mittelwert aller Variablen über alle Beobachtungen 0 ist.

## 0.66.5 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 Teil Clusteranalyse) neu an.

```
brand.sc %>%  
  dplyr::select(-brand) %>%  
  cor() %>%  
  corplot()
```



Die Visualisierung der Korelation der Adjektive scheint drei allgemeine Cluster zu zeigen:

- fun/latest/trendy
- rebuy/bargain/value
- perform/leader/serious

### 0.66.6 Aggregation der durchschnittlichen Bewertungen nach Marke

Um die Frage “Was ist die durchschnittliche (mittlere) Bewertung der Marke auf jedem Adjektiv?” zu benatworten, können wir den Befel `aggregate()` verwenden. Dieser berechnet den Mittelwert jeder Variable nach Marke.

```
brand.mean <- aggregate(.~ brand, data=brand.sc, mean)
```

```
# brand.mean <-
#   brand.sc %>%
#   group_by(brand) %>%
#   summarise_all(funs(mean))
```

```
brand.mean
#>    brand      V1      V2      V3      V4      V5      V6      V7      V8
#> 1     a -0.8859 -0.528  0.411  0.657 -0.9189  0.2141  0.1847 -0.5251
#> 2     b  0.9309  1.071  0.726 -0.972  1.1831  0.0416  0.1513  0.7403
#> 3     c  0.6499  1.163 -0.102 -0.845  1.2227 -0.6070 -0.4407  0.0255
#> 4     d -0.6799 -0.593  0.352  0.187 -0.6922 -0.8808 -0.9326  0.7367
#> 5     e -0.5644  0.193  0.456  0.296  0.0421  0.5516  0.4182  0.1386
#> 6     f -0.0587  0.270 -1.262 -0.218  0.5892  0.8740  1.0227 -0.8132
#> 7     g  0.9184 -0.168 -1.285 -0.517 -0.5338  0.8965  1.2562 -1.2764
#> 8     h -0.0150 -0.298  0.502  0.715 -0.1415 -0.7383 -0.7825  0.8643
#> 9     i  0.3346 -0.321  0.356  0.412 -0.1487 -0.2546 -0.8034  0.5908
#> 10    j -0.6299 -0.789 -0.154  0.285 -0.6022 -0.0971 -0.0738 -0.4814
#>      V9
#> 1   -0.5962
```

```

#> 2   0.2370
#> 3  -0.1324
#> 4  -0.4940
#> 5   0.0365
#> 6   1.3570
#> 7   1.3609
#> 8  -0.6040
#> 9  -0.2032
#> 10 -0.9616

rownames(brand.mean) <- brand.mean[, 1] # Markenname als Fallbezeichnung setzen
brand.mean <- brand.mean[, -1]           # Variablenname brand entfernen
brand.mean
#>      V1      V2      V3      V4      V5      V6      V7      V8      V9
#> a -0.8859 -0.528  0.411  0.657 -0.9189  0.2141  0.1847 -0.5251 -0.5962
#> b  0.9309  1.071  0.726 -0.972  1.1831  0.0416  0.1513  0.7403  0.2370
#> c  0.6499  1.163 -0.102 -0.845  1.2227 -0.6070 -0.4407  0.0255 -0.1324
#> d -0.6799 -0.593  0.352  0.187 -0.6922 -0.8808 -0.9326  0.7367 -0.4940
#> e -0.5644  0.193  0.456  0.296  0.0421  0.5516  0.4182  0.1386  0.0365
#> f -0.0587  0.270 -1.262 -0.218  0.5892  0.8740  1.0227 -0.8132  1.3570
#> g  0.9184 -0.168 -1.285 -0.517 -0.5338  0.8965  1.2562 -1.2764  1.3609
#> h -0.0150 -0.298  0.502  0.715 -0.1415 -0.7383 -0.7825  0.8643 -0.6040
#> i  0.3346 -0.321  0.356  0.412 -0.1487 -0.2546 -0.8034  0.5908 -0.2032
#> j -0.6299 -0.789 -0.154  0.285 -0.6022 -0.0971 -0.0738 -0.4814 -0.9616

```

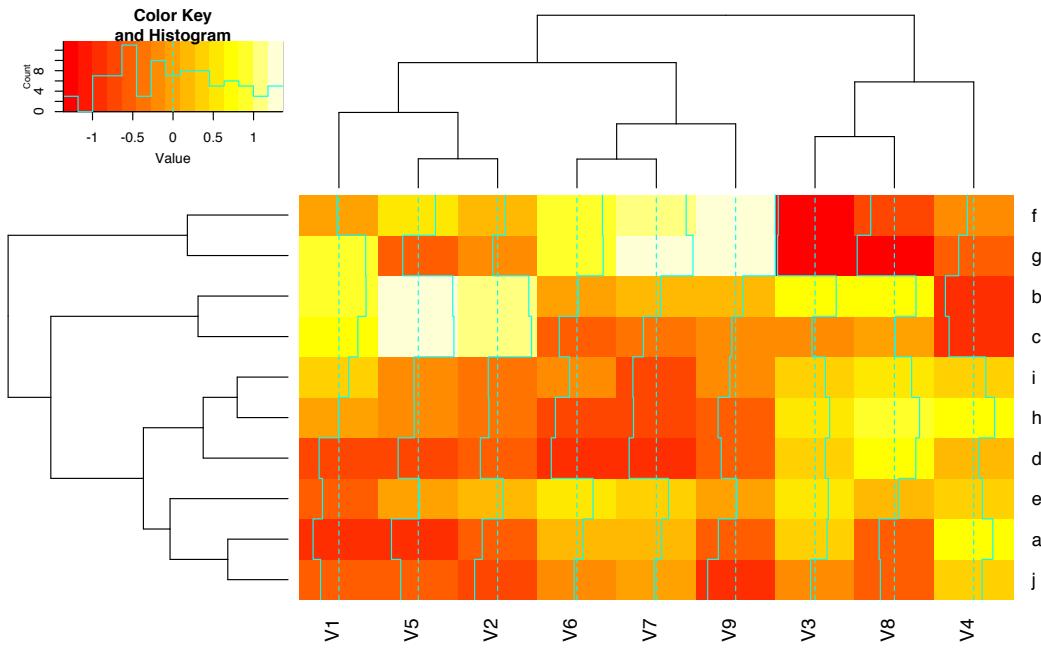
### 0.66.7 Visualisierung der aggregierten Markenbewertungen

Eine **Heatmap** ist eine nützliche Darstellungsmöglichkeit, um solche Ergebnisse zu visualisieren und zu analysieren, da sie Datenpunkte durch die Intensitäten ihrer Werte färbt. Hierzu laden wir das Paket `gplot`.

```

library(gplots)
heatmap.2(as.matrix(brand.mean))

```



`heatmap.2()` sortiert die Spalten und Zeilen, um Ähnlichkeiten und Muster in den Daten hervorzuheben. Eine zusätzliche Analysehilfe ist das Spalten- und Zeilendendrogramm. Hier werden Beobachtungen die nahe beineinanderliegen in einem Baum abgebildet. (Näheres hierzu bei der Clusteranalyse.)

Auch hier sehen wir wieder die gleiche Zuordnung der Adjektive nach

- fun/latest/trendy
- rebuy/bargain/value
- perform/leader/serious

Zusätzlich können die Marken nach Ähnlichkeit bezüglich bestimmter Adjektive zugeordnet werden:

- f und g
- b und c
- i, h und d
- a und j

## 0.67 Hauptkomponentenanalyse (PCA)

Die PCA berechnet ein Variablenset (Komponenten) in Form von linearen Gleichungen, die die lineare 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.

### 0.67.1 Bestimmung der Anzahl der Hauptkomponenten

Betrachten wir in einem ersten Schritt die wichtigsten Komponenten für die Brand-Rating-Daten. Wir finden die Komponenten mit `prcomp()`, wobei wir wieder nur die Bewertungsspalten 1-9 auswählen:

```
brand.pc <- prcomp(brand.sc[, 1:9])
summary(brand.pc)
#> Importance of components:
#>              PC1    PC2    PC3    PC4    PC5    PC6    PC7
#> Standard deviation   1.726 1.448 1.039 0.8528 0.7985 0.7313 0.6246
#> Proportion of Variance 0.331 0.233 0.120 0.0808 0.0708 0.0594 0.0433
#> Cumulative Proportion 0.331 0.564 0.684 0.7647 0.8355 0.8950 0.9383
#>                  PC8    PC9
#> Standard deviation   0.5586 0.493
#> Proportion of Variance 0.0347 0.027
#> Cumulative Proportion 0.9730 1.000
```

```
# Berchnung der Gesamtvarianz
Gesamtvarianz<-1.726^2+1.4479^2+ 1.0389^2+ 0.8528^2+ 0.79846^2+
  0.73133^2+ 0.62458^2 +0.55861^2 +0.49310^2

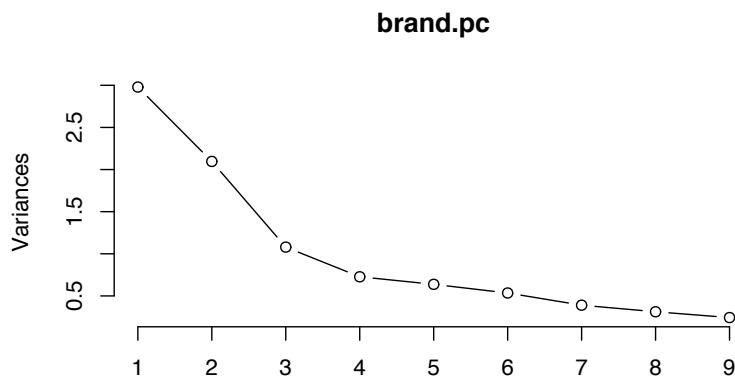
# Varianzanteil der ersten Hauptkomponente
```

```
1.726^2/Gesamtvarianz
#> [1] 0.331
```

## 0.67.2 Scree-Plot

Der Standard-Plot `plot()` für die PCA ist ein **Scree-Plot**, Dieser zeigt uns die Varianzen der Hauptkomponenten und die aufeinanderfolgende Varianzen, die von jeder Komponente berücksichtigt wird. Wir plotten ein Liniediagramm mit dem Argument `typ = "l"` (l für Linie):

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



Wir sehen anhand des Scree-Plots, dass bei den Brand-Rating-Daten der Anteil der Streuung nach der dritten Komponente nicht mehr wesentlich abnimmt.

## 0.67.3 Elbow-Kriterium

Nach diesem Kriterium werden alle Hauptkomponenten berücksichtigt, die links von der Knickstelle im Scree-Plot liegen. Gibt es mehrere Knicks, dann wählt man jene Hauptkomponenten, die links vom rechtenen Knick liegen. Gibt es keinen Knick, dann hilft der Scree-Plot nicht weiter. Bei den Brand-Rating-Daten tritt der Ellbogen, je nach Interpretation, entweder bei drei

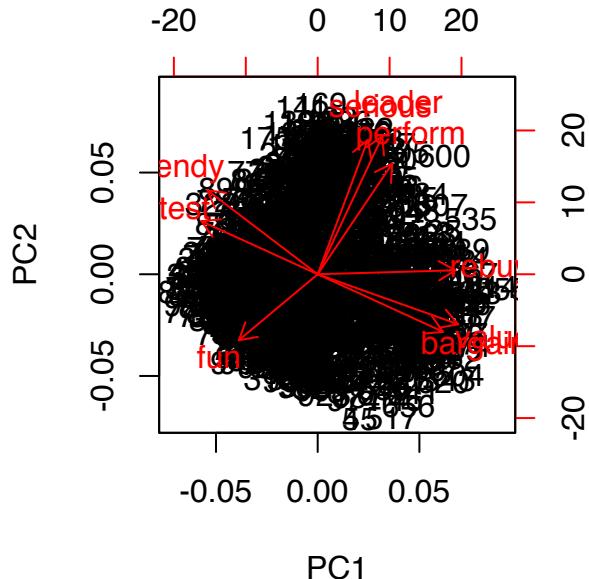
oder vier Komponenten auf. Dies deutet darauf hin, dass die ersten zwei oder drei Komponenten die meiste Streuung in den Markendaten erklären.

#### 0.67.4 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. Dies ist ein zweidimensionales Diagramm von Datenpunkten in Bezug auf die ersten beiden PCA-Komponenten, die mit einer Projektion der Variablen auf die Komponenten überlagert sind.

Dazu verwenden wir `biplot()`:

```
biplot(brand.pc)
```



Die Adjektiv-Gruppierungen auf den Variablen sind als rote Ladungspfeile sichtbar. Zusätzlich erhalten wir einen Einblick in die Bewertungscluster (als dichte Bereiche von Beobachtungspunkten). Der Biplot ist durch die große Anzahl an Beobachtung recht unübersichtlich.

Deshalb führen wir die PCA mit den aggregierten Daten durch:

```

brand.mean
#>      V1      V2      V3      V4      V5      V6      V7      V8      V9
#> a -0.8859 -0.528  0.411  0.657 -0.9189  0.2141  0.1847 -0.5251 -0.5962
#> b  0.9309  1.071  0.726 -0.972  1.1831  0.0416  0.1513  0.7403  0.2370
#> c  0.6499  1.163 -0.102 -0.845  1.2227 -0.6070 -0.4407  0.0255 -0.1324
#> d -0.6799 -0.593  0.352  0.187 -0.6922 -0.8808 -0.9326  0.7367 -0.4940
#> e -0.5644  0.193  0.456  0.296  0.0421  0.5516  0.4182  0.1386  0.0365
#> f -0.0587  0.270 -1.262 -0.218  0.5892  0.8740  1.0227 -0.8132  1.3570
#> g  0.9184 -0.168 -1.285 -0.517 -0.5338  0.8965  1.2562 -1.2764  1.3609
#> h -0.0150 -0.298  0.502  0.715 -0.1415 -0.7383 -0.7825  0.8643 -0.6040
#> i  0.3346 -0.321  0.356  0.412 -0.1487 -0.2546 -0.8034  0.5908 -0.2032
#> j -0.6299 -0.789 -0.154  0.285 -0.6022 -0.0971 -0.0738 -0.4814 -0.9616
brand.mu.pc<- prcomp(brand.mean, scale=TRUE)
summary(brand.mu.pc)
#> Importance of components:
#>                               PC1     PC2     PC3     PC4     PC5     PC6     PC7
#> Standard deviation    2.135  1.735  0.7690  0.615  0.5098  0.3666  0.21506
#> Proportion of Variance 0.506  0.334  0.0657  0.042  0.0289  0.0149  0.00514
#> Cumulative Proportion  0.506  0.841  0.9064  0.948  0.9773  0.9922  0.99737
#>                               PC8     PC9
#> Standard deviation    0.14588 0.04867
#> Proportion of Variance 0.00236 0.00026
#> Cumulative Proportion  0.99974 1.00000

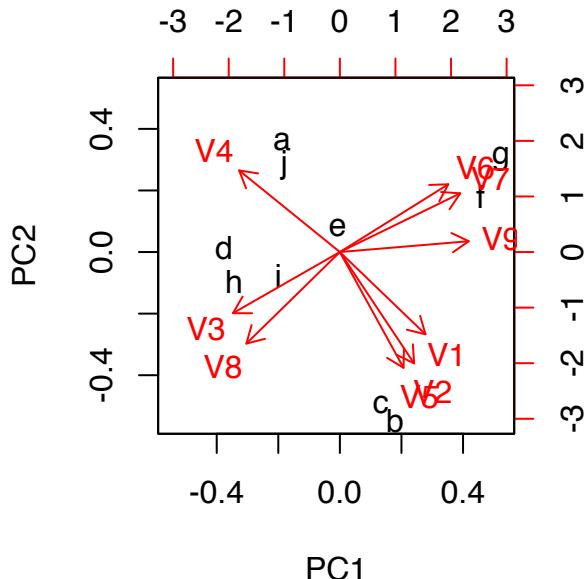
```

Dem Befehl `prcomp()` wurde Skalierung = TRUE hinzugefügt, um die Daten neu zu skalieren. Obwohl die Rohdaten bereits skaliert waren, haben die aggregierten Daten eine etwas andere Skala als die standardisierten Rohdaten. Die Ergebnisse zeigen, dass die ersten beiden Komponenten für 84% der erklärbaren Streuung bei den aggregierten Daten verantwortlich sind.

## 0.67.5 Wahrnehmungsraum

Wenn ein Biplot Marken in einem Zweidimensionalen Raum abbildet, dann nennt man diesen Raum **zweidimensionaler Wahrnehmungsraum**.

```
biplot(brand.mu.pc)
```



Der Biplot der PCA-Lösung für die Mittelwerte gibt einen interpretierbaren Wahrnehmungsraum, der zeigt, wo die Marken in Bezug auf die ersten beiden Hauptkomponenten liegen. Die variablen auf den beiden Komponenten sind mit der PCA auf den gesamten Datensatz konsistent. Wir sehen zunächst vier Bereiche (Positionen) mit gut differenzierten Adjektiven und Marken.

## 0.68 Exploratorische Faktorenanalyse (EFA)

EFA ist 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 (ähn-

lich 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. Innerhalb einer PCA kann die Interpretierbarkeit über eine **Rotation** (z. B. `varimax()`) erhöht werden.

## 0.68.1 Finden einer EFA Lösung

Als erstes muss die Anzahl der zu schätzenden Faktoren bestimmt werden. Hierzu verwenden wir zwei gebräuchliche Methoden:

### 0.68.1.1 Das Elbow-Kriterium

Den Skreeplot haben wir bereits bei der PCA durchgeführt. Ein Knick kontnen wir bei der dritten oder vierten Hauptkomponente feststellen. Somit zeigt der Skreeplot eine 2 oder 3 Faktorenlösung an.

Durch das Paket `nFactors` bekommen wir eine formalisierte Berechnung der Scree-Plot Lösung mit dem Befehl `nScree()`

```
library(nFactors)
nScree(brand.sc[, 1:9])
#>   noc naf nparallel nkaiser
#> 1   3   2       3       3
```

`nScree` gibt vier methodische Schätzungen für die Anzahl an Faktoren durch den Scree-Plot aus. Wir sehen, dass drei von vier Methoden drei Faktoren vorschlagen.

### 0.68.1.2 Das Eigenwert-Kriterium

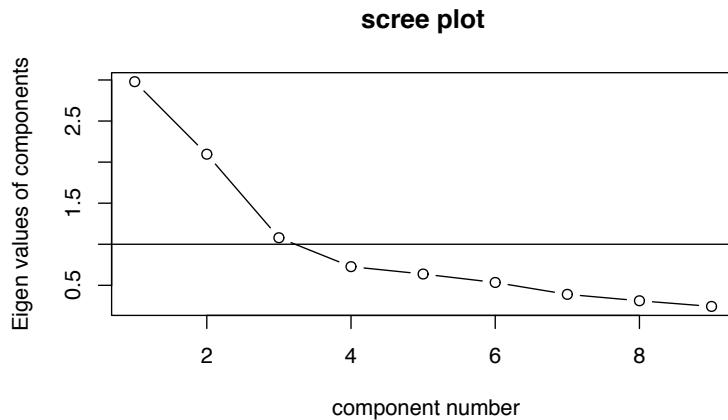
Der Eigenwert ist eine Metrik für den Anteil der erklärten Varianz. Die Anzahl Eigenwerte können wir über den Befehl `eigen()` ausgeben.

```
eigen(cor(brand.sc[, 1:9]))
#> $values
#> [1] 2.979 2.097 1.079 0.727 0.638 0.535 0.390 0.312 0.243
#>
#> $vectors
#>      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
#> [1,] -0.237 -0.4199  0.0385  0.5263  0.4679  0.337  0.36418 -0.1444
#> [2,] -0.206 -0.5238 -0.0951  0.0892 -0.2945  0.297 -0.61367  0.2877
#> [3,]  0.370 -0.2015 -0.5327 -0.2141  0.1059  0.174 -0.18548 -0.6429
#> [4,]  0.251  0.2504 -0.4178  0.7506 -0.3315 -0.141 -0.00711  0.0746
#> [5,] -0.160 -0.5105 -0.0407 -0.0989 -0.5552 -0.392  0.44530 -0.1835
#> [6,] -0.399  0.2185 -0.4899 -0.1673 -0.0126  0.139  0.28826  0.0579
#> [7,] -0.447  0.1898 -0.3692 -0.1512 -0.0633  0.220  0.01716  0.1483
#> [8,]  0.351 -0.3185 -0.3709 -0.1676  0.3665 -0.266  0.15357  0.6145
#> [9,] -0.439 -0.0151 -0.1246  0.1303  0.3557 -0.675 -0.38866 -0.2021
#>      [,9]
#> [1,] -0.0522
#> [2,]  0.1789
#> [3,] -0.0575
#> [4,] -0.0315
#> [5,] -0.0907
#> [6,]  0.6472
#> [7,] -0.7281
#> [8,] -0.0591
#> [9,]  0.0172
```

Der Eigenwert eines Faktors sagt aus, wie viel Varianz dieser Faktor an der Gesamtvarianz aufklärt. Auf dem Eigenwert-Kriterium sollen nur Faktoren mit einem Eigenwert größer 1 extrahiert werden. Dies sind bei den Brand-Rating Daten drei Faktoren, da drei Eigenwerte größer 1 sind.

Dies kann auch grafisch mit dem `VSS.Scree` geplotted werden.

```
VSS.scree(brand.sc[, 1:9])
```



### Schätzung der EFA

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

```
brand.fa<-factanal(brand.sc[, 1:9], factors=3)
brand.fa
#>
#> Call:
#> factanal(x = brand.sc[, 1:9], factors = 3)
#>
#> Uniquenesses:
#>   perform   leader  latest      fun serious bargain    value   trendy   rebuy
#>   0.624    0.327   0.005    0.794    0.530    0.302    0.202    0.524    0.575
#>
#> Loadings:
#>             Factor1 Factor2 Factor3
#> perform          0.607
#> leader           0.810   0.106
#> latest          -0.163
#> fun              -0.398   0.205
#> serious          0.682
#> bargain          0.826   -0.122
```

```

#> value      0.867          -0.198
#> trendy     -0.356          0.586
#> rebuy      0.499   0.296  -0.298
#>
#>           Factor1 Factor2 Factor3
#> SS loadings    1.853    1.752    1.510
#> Proportion Var  0.206    0.195    0.168
#> Cumulative Var  0.206    0.401    0.568
#>
#> Test of the hypothesis that 3 factors are sufficient.
#> The chi square statistic is 64.6 on 12 degrees of freedom.
#> The p-value is 3.28e-09

```

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=.5` und die Ladungen so sortieren, dass die Ladungen die auf einen Faktor laden untereinander stehen mit `sort=TRUE`.

```

print(brand.fa, digits=2, cutoff=.5, sort=TRUE)
#>
#> Call:
#> factanal(x = brand.sc[, 1:9], factors = 3)
#>
#> Uniquenesses:
#> perform  leader  latest      fun serious bargain   value  trendy  rebuy
#>    0.62    0.33    0.00    0.79    0.53    0.30    0.20    0.52    0.58
#>
#> Loadings:
#>           Factor1 Factor2 Factor3
#> bargain      0.83
#> value       0.87
#> perform      0.61
#> leader       0.81
#> serious      0.68
#> latest        0.98

```

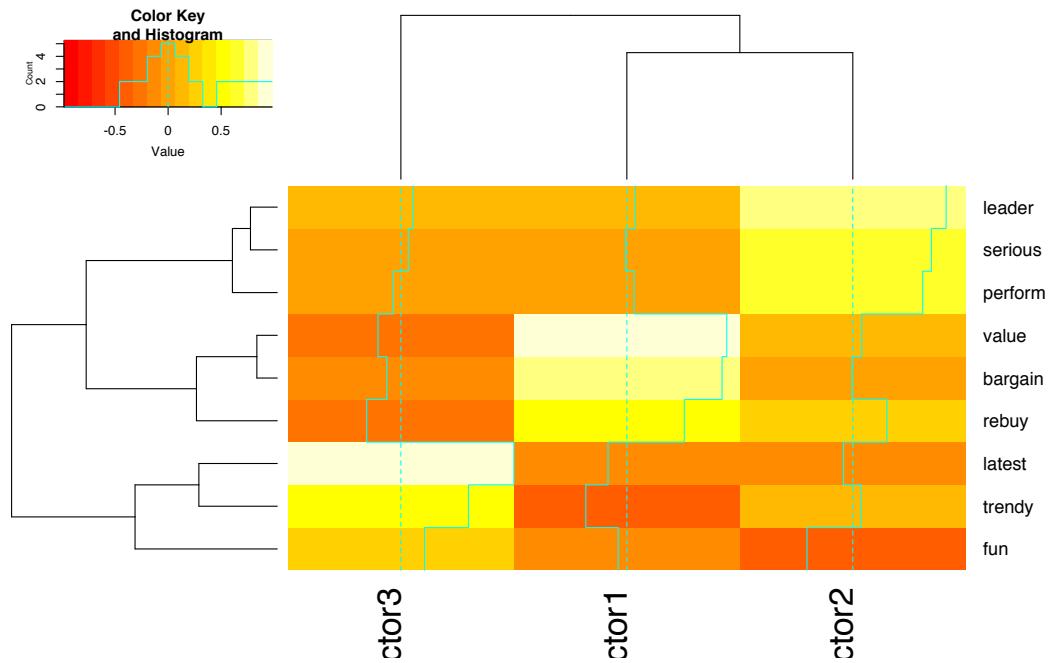
```
#> trendy          0.59
#> fun
#> rebuy
#>
#>           Factor1 Factor2 Factor3
#> SS loadings    1.85   1.75   1.51
#> Proportion Var  0.21   0.19   0.17
#> Cumulative Var  0.21   0.40   0.57
#>
#> Test of the hypothesis that 3 factors are sufficient.
#> The chi square statistic is 64.6 on 12 degrees of freedom.
#> The p-value is 3.28e-09
```

Standardmäßig wird bei der `factanal` eine Varimax-Rotation durchgeführt. Diese verwendet, dass es keine Korrelationen zwischen den Faktoren gibt. Sollen Korrelationen zwischen den Faktoren zugelassen werden, empfiehlt sich die Oblimin-Rotation mit dem Argument `rotation="oblimin"` aus dem Paket `GPArotation`.

## 0.68.2 Heatmap mit Ladungen

In der obigen Ausgabe werden die Item-to-Faktor-Ladungen angezeigt. Im zurückgegebenen Objekt `brand.fa` sind diese als `$loadings` vorhanden. Wir können die Item-Faktor-Beziehungen mit einer Heatmap von `$loadings` visualisieren:

```
heatmap.2(brand.fa$loadings)
```



Das Ergebnis aus der Heatmap zeigt eine deutliche Trennung der Items in 3 Faktoren, die grob interpretierbar sind als **value**, **leader** und **latest**.

### 0.68.3 Berechnung der Faktor-Scores

Zusätzlich zur Schätzung der Faktorstruktur kann die EFA auch die latenten Faktorwerte für jede Beobachtung schätzen. Die gängige Extraktionsmethode ist Bartlett-Methode.

```
brand.fa.ob <- factanal(brand.sc[, 1:9], factors=3, scores="Bartlett")
brand.scores <- data.frame(brand.fa.ob$scores)
head(brand.scores)
#>   Factor1 Factor2 Factor3
#> 1  1.910 -0.867  0.845
#> 2 -1.579 -1.507 -1.119
#> 3  1.172 -1.130 -0.296
#> 4  0.496 -0.430  1.302
#> 5  2.114 -2.047 -0.210
```

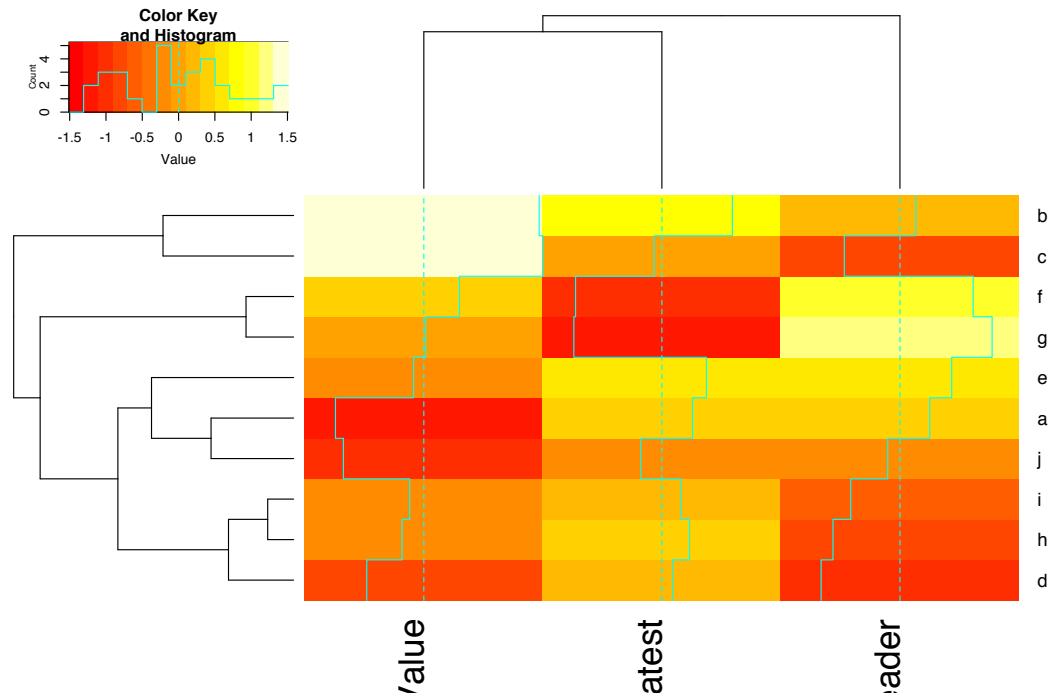
```
#> 6   1.691   0.155   1.219
```

Wir können dann die Faktor-Scores verwenden, um die Positionen der Marken auf den Faktoren zu bestimmen.

```
brand.scores$brand <- brand.sc$brand # Zuweisung der Markennamen zur Scores-Matrix
brand.fa.mean <- aggregate(. ~ brand, data=brand.scores, mean) # Aggregation Marken
rownames(brand.fa.mean) <- brand.fa.mean[, 1] # Fallbezeichnung mit Markennamen setzen
brand.fa.mean <- brand.fa.mean[, -1] # Erste Spalte löschen
names(brand.fa.mean) <- c("Leader", "Value", "Latest") # Spaltennamen neu zuweisen
brand.fa.mean
#>   Leader   Value   Latest
#> a  0.378 -1.1206  0.3888
#> b  0.202  1.4631  0.8938
#> c -0.706  1.5096 -0.0995
#> d -1.000 -0.7239  0.1346
#> e  0.656 -0.1291  0.5657
#> f  0.929  0.4516 -1.0946
#> g  1.169  0.0245 -1.1166
#> h -0.849 -0.2763  0.3500
#> i -0.624 -0.1788  0.2450
#> j -0.155 -1.0202 -0.2671
```

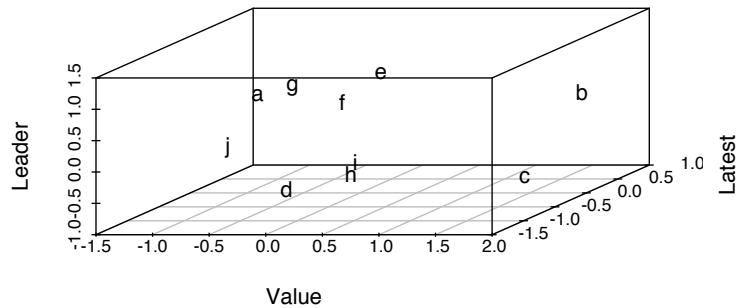
Mittels Heatmap kann dann sehr schnell analysiert werden, welche Marke auf welcher Dimension gute oder schlechte Ausprägungen hat.

```
heatmap.2(as.matrix(brand.fa.mean))
```



Drei Dimensionen lassen sich in einem dreidimensionalen Raum darstellen:

```
library(scatterplot3d)
attach(brand.fa.mean) # Datensatz zum Suchpfad hinzufügen
scatterplot3d(Leader~Value+Latest, pch=row.names(brand.fa.mean))
detach(brand.fa.mean) # Datensatz vom Suchpfad entfernen
```



## 0.69 Interne Konsistenz von 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. Formal ist es ein korrigierter durchschnittlicher Korrelationskoeffizient.

Faustregeln für die Bewertung von Cronbachs Alpha:

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

Wir bewerten nun die interne Konsistenz der Items für die Konstrukte **Leader**, **Value** und **Latest**.

```
alpha(brand.sc[, c("leader", "serious", "perform")], check.keys=TRUE)
#>
#> Reliability analysis
#> Call: alpha(x = brand.sc[, c("leader", "serious", "perform")], check.keys = TRUE)
#>
#>   raw_alpha std.alpha G6(smc) average_r S/N    ase      mean     sd
#>   0.73      0.73      0.66      0.48 2.7 0.015 -7.5e-17 0.81
#>
#>   lower alpha upper      95% confidence boundaries
```

```

#> 0.7 0.73 0.76
#>
#> Reliability if an item is dropped:
#>      raw_alpha std.alpha G6(smc) average_r S/N alpha se
#> leader      0.53      0.53     0.36      0.36 1.1    0.030
#> serious     0.67      0.67     0.50      0.50 2.0    0.021
#> perform     0.73      0.73     0.57      0.57 2.7    0.017
#>
#> Item statistics
#>      n raw.r std.r r.cor r.drop      mean sd
#> leader 1000 0.86 0.86 0.76 0.65 7.0e-17 1
#> serious 1000 0.80 0.80 0.64 0.54 -1.5e-16 1
#> perform 1000 0.77 0.77 0.57 0.48 -1.6e-16 1
alpha(brand.sc[, c("value", "bargain", "rebuy")], check.keys=TRUE)
#>
#> Reliability analysis
#> Call: alpha(x = brand.sc[, c("value", "bargain", "rebuy")], check.keys =
#>
#>      raw_alpha std.alpha G6(smc) average_r S/N ase      mean sd
#>          0.8      0.8     0.75      0.57 4 0.011 9.7e-18 0.84
#>
#> lower alpha upper      95% confidence boundaries
#> 0.78 0.8 0.82
#>
#> Reliability if an item is dropped:
#>      raw_alpha std.alpha G6(smc) average_r S/N alpha se
#> value      0.64      0.64     0.47      0.47 1.8    0.0230
#> bargain     0.67      0.67     0.51      0.51 2.0    0.0207
#> rebuy       0.85      0.85     0.74      0.74 5.7    0.0095
#>
#> Item statistics
#>      n raw.r std.r r.cor r.drop      mean sd
#> value   1000 0.89 0.89 0.83 0.73 1.2e-16 1
#> bargain 1000 0.87 0.87 0.80 0.70 -1.2e-16 1
#> rebuy   1000 0.78 0.78 0.57 0.52 5.2e-17 1
alpha(brand.sc[, c("latest", "trendy", "fun")], check.keys=TRUE)

```

```

#>
#> Reliability analysis
#> Call: alpha(x = brand.sc[, c("latest", "trendy", "fun")], check.keys = TRUE)
#>
#>   raw_alpha std.alpha G6(smc) average_r S/N    ase      mean     sd
#>   0.6       0.6     0.58      0.33 1.5 0.022 4.4e-17 0.75
#>
#>   lower alpha upper      95% confidence boundaries
#> 0.56 0.6 0.64
#>
#>   Reliability if an item is dropped:
#>   raw_alpha std.alpha G6(smc) average_r S/N alpha se
#> latest      0.23      0.23     0.13      0.13 0.29  0.049
#> trendy      0.39      0.39     0.25      0.25 0.65  0.038
#> fun         0.77      0.77     0.63      0.63 3.37  0.014
#>
#>   Item statistics
#>   n raw.r std.r r.cor r.drop      mean sd
#> latest 1000  0.84  0.84  0.76   0.58 -9.7e-17  1
#> trendy 1000  0.79  0.79  0.68   0.48  8.8e-17  1
#> fun    1000  0.61  0.61  0.26   0.21  1.5e-16  1

```

Bis auf `Latest` sind alle Konstrukte bezüglich ihrer internen Konsistenz akzeptabel. Bei dem Konstrukt `Latest` können wir durch Elimination von `fun` das Cronbachs Alpha von einem fragwürdigen Wert auf einen akzeptablen Wert von 0,77 erhöhen.

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

### 0.69.1 Übung

Führen Sie eine Dimensionsreduktion mit den nichtskalierten Original-Daten durch. Berechnen Sie zur Interpretation keine Faktor-Scores, sondern berech-

nen Sie stattdessen den Mittelwert der Variablen, die hoch (mindestens 0,5) auf einen Faktor laden. Für die Berechnung verwenden Sie

```
Datensatz$Neue_Variable <- apply(Datensatz[,c("Variable1",
                                         "Variable2", "etc..")],
                                         1, mean,na.rm=TRUE)
```

## 0.69.2 Literatur

- Chris Chapman, Elea McDonnell Feit (2015): *R for Marketing Research and Analytics*, Kapitel 8.1-8.3
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013): *An Introduction to Statistical Learning – with Applications in R*, <http://www-bcf.usc.edu/~gareth/ISL/>, Kapitel 10.2, 10.4
- Reinhold Hatzinger, Kurt Hornik, Herbert Nagel (2011): *R – Einführung durch angewandte Statistik*. Kapitel 11
- Maike Luhmann (2015): R für Einsteiger, Kapitel 19

---

Diese Übung orientiert sich am Beispiel aus Kapitel 8 aus Chapman und Feit (2015) und steht unter der Lizenz Creative Commons Attribution-ShareAlike 3.0 Unported<sup>51</sup>. Der Code steht unter der Apache Lizenz 2.0<sup>52</sup>

---

<sup>51</sup><http://creativecommons.org/licenses/by-sa/3.0>

<sup>52</sup><http://www.apache.org/licenses/LICENSE-2.0>

# Textmining

In diesem Kapitel benötigte R-Pakete:

```
library(tidyverse)  # Datenjudo
library(okcupiddata) # Daten
library(stringr)  # Textverarbeitung
library(tidytext)  # Textmining
library(pdftools)  # PDF einlesen
library(downloader) # Daten herunterladen
library(knitr)    # HTML-Tabellen
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<sup>53</sup> 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

---

<sup>53</sup>Dank an Karsten Lübke, dessen Fachkompetenz mir mindestens so geholfen hat wie seine Begeisterung an der Statistik ansteckend ist.

Computer ist einfach noch wesentlich *dümmer* als ein Mensch, 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.

## 0.70 Einführung

### 0.70.1 Grundbegriffe

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 and Robinson 2016).

## 0.71 Grundlegende Analyse

### 0.71.1 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-Vektor<sup>54</sup> 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(wort, text)
#> # A tibble: 24 × 2
#>   Zeile     wort
#>   <int> <chr>
#> 1      1   wir
#> 2      1   haben
```

---

<sup>54</sup>Nach dem Gedicht “Jahrgang 1899” von Erich Kästner

```
#> 3      1  die
#> # ... with 21 more rows
```

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 × 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: Diese “dehnen” wandelt automatisch Groß- in Kleinbuchstaben um. UND DANN

filtere die Spalte “wort”, so dass nur noch Kleinbuchstaben übrig bleiben. FERTIG.

## 0.71.2 Text-Daten einlesen

Nun lesen wir Text-Daten ein; das können beliebige Daten sein<sup>55</sup>. Eine gewisse Reichhaltigkeit ist von Vorteil. Nehmen wir das Parteiprogramm der Partei AfD<sup>56</sup>.

```
afd_url <- "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)

afd_raw[3]
#> [1] "3\t Programm für Deutschland / Inhalt\n    7 / Kultur, Sprache und Identität"
```

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.

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(token, afd_raw) %>%
  filter(str_detect(token, "[a-z]")) -> afd_df
```

---

<sup>55</sup>Ggf. benötigen Sie Administrator-Rechte, um Dateien auf Ihre Festplatte zu speichern.

<sup>56</sup>[https://www.alternativefuer.de/wp-content/uploads/sites/7/2016/05/2016-06-27\\_afd\\_grundsatzprogramm\\_web-version.pdf](https://www.alternativefuer.de/wp-content/uploads/sites/7/2016/05/2016-06-27_afd_grundsatzprogramm_web-version.pdf)

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

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

### 0.71.3 Worthäufigkeiten auszählen

```
afd_df %>%
  na.omit() %>%  # fehlende Werte löschen
  count(token, sort = TRUE)
#> # A tibble: 7,087 × 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 `lisa`.

```
data(stopwords_de)

stopwords_de <- data_frame(word = stopwords_de)

stopwords_de <- stopwords_de %>%
  rename(token = word)
```

```
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

afd_count %>%
  top_n(10) %>%
  knitr::kable()
```

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

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.

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

```
afd_count %>%
  top_n(10) %>%
  knitr::kable()
```

token_stem	n
deutschland	219
afd	171
deutsch	119
polit	88
staat	85
programm	81
europa	80
woll	67
burg	66
soll	63

Das ist schon informativer. Dem Befehl `wordStem` füttert man einen Vektor an Wörtern ein und gibt die Sprache an (Default ist Englisch<sup>57</sup>). Das ist schon alles.

#### 0.71.4 Visualisierung

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

```
wordcloud(words = afd_count$token_stem, freq = afd_count$n, max.words = 100, s
```

---

<sup>57</sup>Cleveland fände diese Idee nicht so gut.



Man kann die Anzahl der Wörter, Farben und einige weitere Formatierungen der Wortwolke beeinflussen<sup>58</sup>.

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

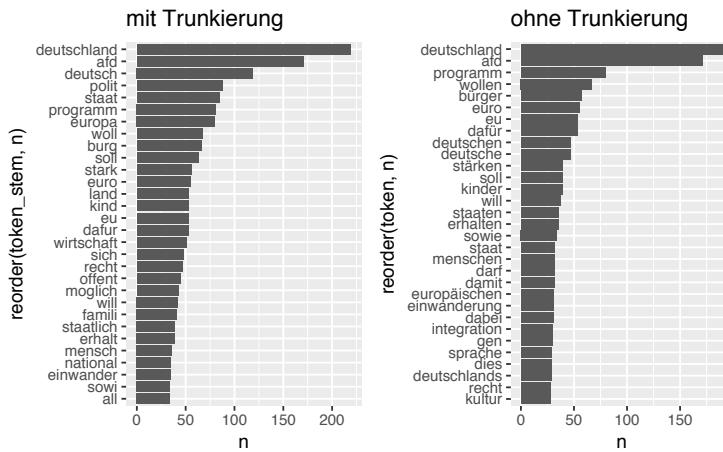
```
afdf_count %>%
  top_n(30) %>%
  ggplot() +
  aes(x = reorder(token_stem, n), y = n) +
  geom_col() +
  labs(title = "mit Trunkierung") +
  coord_flip() -> p1

afdf_df %>%
  count(token, sort = TRUE) %>%
  top_n(30) %>%
  ggplot() +
  aes(x = reorder(token, n), y = n) +
  geom_col() +
  labs(title = "ohne Trunkierung") +
  coord_flip() -> p2
```

---

<sup>58</sup><https://cran.r-project.org/web/packages/wordcloud/index.html>

```
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  $90^\circ$ , d.h. die Achsen sind vertauscht. `grid.arrange` packt beide Plots in eine Abbildung, welche 2 Spalten (`ncol`) hat.

## 0.72 Sentiment-Analyse

Eine weitere interessante Analyse ist, die “Stimmung” oder “Emotionen” (Sentiments) eines Textes auszulesen. Die Anführungszeichen deuten an, dass hier ein Maß an Verständnis suggeriert wird, welches nicht (unbedingt) von der Analyse eingehalten wird. Jedenfalls ist das Prinzip der Sentiment-Analyse im einfachsten Fall so:



Schau dir jeden Token aus dem Text an.  
Prüfe, ob sich das Wort im Lexikon der Sentiments wiederfindet.  
Wenn ja, dann addiere den Sentimentswert dieses Tokens zum bestehenden Sentiments-Wert.

Wenn nein, dann gehe weiter zum nächsten Wort.  
Liefere zum Schluss die Summenwerte pro Sentiment zurück.

Es gibt Sentiment-Lexika, die lediglich einen Punkt für “positive Konnotation” bzw. “negative Konnotation” geben; andere Lexiko weisen differenzierte Gefühlskonnotationen auf. Wir nutzen hier dieses<sup>59</sup> Lexikon (Remus, Quasthoff, and Heyer 2010). Der Einfachheit halber gehen wir im Folgenden davon aus, dass das Lexikon schon aufbereitet vorliegt. Die Aufbereitung unten im Abschnitt zur Vertiefung nachgelesen werden.

Unser Sentiment-Lexikon sieht so aus:

```
library(knitr)  
  
kable(head(sentiment_df))
```

neg_pos	Wort	Wert	Inflektionen
neg	Abbau	-0.058	Abbaus,Abbaues,Abbauen,Abbaue
neg	Abbruch	-0.005	Abbruches,Abbrüche,Abbruchs,Abbrüchen
neg	Abdankung	-0.005	Abdankungen
neg	Abdämpfung	-0.005	Abdämpfungen
neg	Abfall	-0.005	Abfalles,Abfälle,Abfalls,Abfällen
neg	Abfuhr	-0.337	Abfuhrten

### 0.72.1 Ungewichtete Sentiment-Analyse

Nun können wir jedes Token des Textes mit dem Sentiment-Lexikon abgleichen; dabei zählen wir die Treffer für positive bzw. negative Terme. Besser wäre noch: Wir könnten die Sentiment-Werte pro Treffer addieren (und nicht für jeden Term 1 addieren). Aber das heben wir uns für später auf.

```
sentiment_neg <- match(afd_df$token, filter(sentiment_df, neg_pos == "neg")$Wort)  
neg_score <- sum(!is.na(sentiment_neg))
```

---

<sup>59</sup><http://asv.informatik.uni-leipzig.de/download/sentiws.html>

```

sentiment_pos <- match(afd_df$token, filter(sentiment_df, neg_pos == "pos")$Wort)
pos_score <- sum(!is.na(sentiment_pos))

round(pos_score/neg_score, 1)
#> [1] 2.7

```

Hier schauen wir für jedes negative (positive) Token, ob es einen “Match” im Sentiment-Lexikon (`sentiment_df$Wort`) gibt; das geht mit `match`. `match` liefert `NA` zurück, wenn es keinen Match gibt (ansonsten die Nummer des Sentiment-Worts). Wir brauchen also nur die Anzahl der Nicht-Nas (`!is.na`) auszuzählen, um die Anzahl der Matches zu bekommen.

Entgegen dem, was man vielleicht erwarten würde, ist der Text offenbar positiv geprägt. Der “Positiv-Wert” ist ca. 2.6 mal so groß wie der “Negativ-Wert”. Fragt sich, wie sich dieser Wert mit anderen vergleichbaren Texten (z.B. andere Parteien) misst. Hier sei noch einmal betont, dass die Sentiment-Analyse bestenfalls grobe Abschätzungen liefern kann und keinesfalls sich zu einem hermeneutischen Verständnis aufschwingt.

Welche negativen Wörter und welche positiven Wörter wurden wohl verwendet? Schauen wir uns ein paar an.

```

afd_df %>%
  mutate(sentiment_neg = sentiment_neg,
         sentiment_pos = sentiment_pos) -> afd_df

afd_df %>%
  filter(!is.na(sentiment_neg)) %>%
  select(token) -> negative_sentiments

head(negative_sentiments$token, 50)
#> [1] "mindern"      "verbieten"     "unmöglich"    "töten"
#> [5] "träge"        "schädlich"     "unangemessen" "unterlassen"
#> [9] "kalt"          "schwächen"     "ausfallen"    "verringern"
#> [13] "verringern"   "verringern"    "verringern"   "belasten"
#> [17] "belasten"     "fremd"        "schädigenden" "klein"

```

```

#> [21] "klein"           "klein"           "klein"           "eingeschränkt"
#> [25] "eingeschränkt"  "entziehen"        "schwer"          "schwer"
#> [29] "schwer"          "schwer"          "verharmlosen"    "unerwünscht"
#> [33] "abgleiten"       "wirkungslos"     "schwach"         "verschleppen"
#> [37] "vermindern"      "vermindern"      "ungleich"        "widersprechen"
#> [41] "zerstört"        "zerstört"        "erschweren"      "auffallen"
#> [45] "unvereinbar"     "unvereinbar"     "unvereinbar"     "abhängig"
#> [49] "abhängig"        "abhängig"        "abhängig"        "abhängig"

afd_df %>%
  filter(!is.na(sentiment_pos)) %>%
  select(token) -> positive_sentiments

head(positive_sentiments$token, 50)
#> [1] "optimal"          "aufstocken"       "locker"
#> [4] "zulässig"         "gleichwertig"     "wiederbeleben"
#> [7] "beauftragen"      "wertvoll"          "nah"
#> [10] "nah"              "nah"               "überzeugt"
#> [13] "genehmigen"      "genehmigen"       "überleben"
#> [16] "überleben"       "genau"             "verständlich"
#> [19] "erlauben"         "aufbereiten"      "zugänglich"
#> [22] "messbar"          "erzeugen"          "erzeugen"
#> [25] "ausgleichen"      "ausreichen"        "mögen"
#> [28] "kostengünstig"   "gestiegen"         "gestiegen"
#> [31] "bedeuten"         "massiv"            "massiv"
#> [34] "massiv"           "massiv"            "einfach"
#> [37] "finanzieren"      "vertraulich"       "steigen"
#> [40] "erweitern"         "verstehen"         "schnell"
#> [43] "zugreifen"         "tätig"              "unternehmerisch"
#> [46] "entlasten"        "entlasten"         "entlasten"
#> [49] "entlasten"        "helfen"            "helfen"

```

### 0.72.2 Anzahl der unterschiedlichen negativen bzw. positiven Wörter

Allerdings müssen wir unterscheiden zwischen der *Anzahl* der negativen bzw. positiven Wörtern und der Anzahl der *unterschiedlichen* Wörter.

Zählen wir noch die Anzahl der unterschiedlichen Wörter im negativen und positiven Fall.

```
afd_df %>%
  filter(!is.na(sentiment_neg)) %>%
  summarise(n_distinct_neg = n_distinct(token))
#> # A tibble: 1 × 1
#>   n_distinct_neg
#>   <int>
#>   1           96

afd_df %>%
  filter(!is.na(sentiment_pos)) %>%
  summarise(n_distinct_pos = n_distinct(token))
#> # A tibble: 1 × 1
#>   n_distinct_pos
#>   <int>
#>   1           187
```

Dieses Ergebnis passt zum vorherigen: Die Anzahl der positiven Wörter (187) ist ca. doppelt so groß wie die Anzahl der negativen Wörter (96).

### 0.72.3 Gewichtete Sentiment-Analyse

Oben haben wir nur ausgezählt, *ob* ein Term der Sentiment-Liste im Corpus vorkam. Genauer ist es, diesen Term mit seinem Sentiment-Wert zu gewichten, also eine gewichtete Summe zu erstellen.

```

sentiment_df %>%
  rename(token = Wort) -> sentiment_df

afd_df %>%
  left_join(sentiment_df, by = "token") -> afd_df

afd_df %>%
  filter(!is.na(Wert)) %>%
  summarise(Sentimentwert = sum(Wert, na.rm = TRUE)) -> afd_sentiment_summe

afd_sentiment_summe$Sentimentwert
#> [1] -23.9

```

Zuerst benennen wir `Wort` in `token` um, damit es beiden Dataframes (`sentiment_df` und `afd_df`) eine Spalte mit gleichen Namen gibt. Diese Spalte können wir dann zum “Verheiraten” (`left_join`) der beiden Spalten nutzen. Dann summieren wir den Sentiment-Wert jeder nicht-leeren Zeile auf.

Siehe da: Nun ist der Duktus deutlich negativer als positiver. Offenbar werden mehr positive Wörter als negative verwendet, *aber* die negativen sind viel intensiver.

#### 0.72.4 Tokens mit den extremsten Sentimentwerten

Schauen wir uns die intensivesten Wörter mal an.

```

afd_df %>%
  filter(neg_pos == "pos") %>%
  distinct(token, .keep_all = TRUE) %>%
  arrange(-Wert) %>%
  filter(row_number() < 11) %>%

```

```
select(token, Wert) %>%
kable()
```

token	Wert
besonders	0.539
genießen	0.498
wichtig	0.382
sicher	0.373
helfen	0.373
miteinander	0.370
groß	0.369
wertvoll	0.357
motiviert	0.354
gepflegt	0.350

```
afd_df %>%
filter(neg_pos == "neg") %>%
distinct(token, .keep_all = TRUE) %>%
arrange(Wert) %>%
filter(row_number() < 11) %>%
select(token, Wert) %>%
kable()
```

token	Wert
schädlich	-0.927
schwach	-0.921
brechen	-0.799
ungerecht	-0.784
behindern	-0.775
falsch	-0.762
gemein	-0.720
gefährlich	-0.637
verbieten	-0.629
vermeiden	-0.526

Tatsächlich erscheinen die negativen Wörter “dampfender” und “fauchender”

als die positiven.

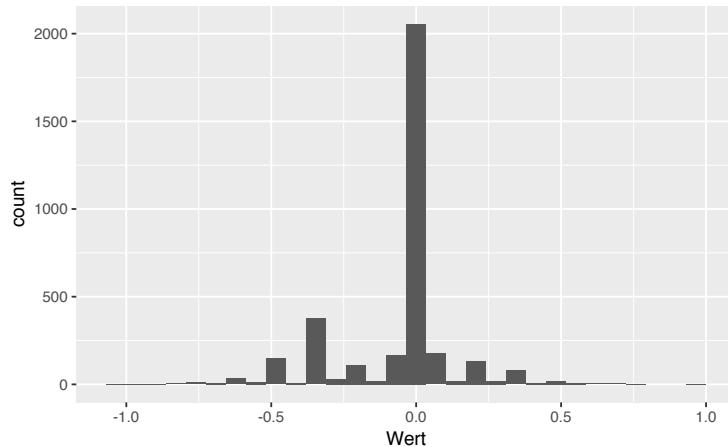
Die Syntax kann hier so übersetzt werden:

Nehmen den Dataframe adf\_df UND DANN  
 filtere die Token mit negativen Sentiment UND DANN  
 lösche doppelte Zeilen UND DANN  
 sortiere (absteigend) UND DANN  
 filtere nur die Top 10 UND DANN  
 zeige nur die Spalten token und Wert UND DANN  
 zeige eine schöne Tabelle.

### 0.72.5 Relativer Sentiments-Wert

Nun könnte man noch den erzielten “Netto-Sentimentswert” des Corpus ins Verhältnis setzen Sentimentswert des Lexikons: Wenn es insgesamt im Sentiment-Lexikon sehr negativ zuginge, wäre ein negativer Sentimentwert in einem beliebigen Corpus nicht überraschend.

```
sentiment_df %>%
  filter(!is.na(Wert)) %>%
  ggplot() +
  aes(x = Wert) +
  geom_histogram()
```



Es scheint einen (leichten) Überhang an negativen Wörtern zu geben. Schauen wir auf die genauen Zahlen.

```
sentiment_df %>%
  filter(!is.na(Wert)) %>%
  count(neg_pos)
#> # A tibble: 2 × 2
#>   neg_pos     n
#>   <chr> <int>
#> 1 neg    1818
#> 2 pos    1650
```

Tatsächlich ist die Zahl negativ konnotierter Terme etwas größer als die Zahl der positiv konnotierten. Jetzt gewichten wir die Zahl mit dem Sentimentswert der Terme, in dem wir die Sentimentswerte (die ein negatives bzw. ein positives Vorzeichen aufweisen) aufaddieren.

```
sentiment_df %>%
  filter(!is.na(Wert)) %>%
  summarise(sentiment_summe = sum(Wert)) -> sentiment_lexikon_sum

sentiment_lexikon_sum$sentiment_summe
#> [1] -187
```

Im Vergleich zum Sentiment der Lexikons ist unser Corpus deutlich negativer. Um genau zu sein, um diesen Faktor:

```
sentiment_lexikon_sum$sentiment_summe / afd_sentiment_summe$Sentimentwert
#> [1] 7.83
```

Der *relative Sentimentswert* (relativ zum Sentiment-Lexikon) beträgt also ~7.8.

## 0.73 Verknüpfung mit anderen Variablen

Kann man die Textdaten mit anderen Daten verknüpfen, so wird die Analyse reichhaltiger. So könnte man überprüfen, ob sich zwischen Sentiment-Gehalt und Zeit oder Autor ein Muster findet/bestätigt. Uns liegen in diesem Beispiel keine andere Daten vor, so dass wir dieses Beispiel nicht weiter verfolgen.

---

## 0.74 Vertiefung

### 0.74.1 Erstellung des Sentiment-Lexikons

Der Zweck dieses Abschnitts ist es, eine Sentiment-Lexikon in deutscher Sprache einzulesen.

Dazu wird das Sentiment-Lexikon dieser Quelle<sup>60</sup> verwendet (CC-BY-NC-SA 3.0). In diesem Paper<sup>61</sup> finden sich Hintergründe. Von dort lassen sich die Daten herunter laden. Im folgenden gehe ich davon aus, dass die Daten herunter geladen sind und sich im Working Directory befinden.

Wir benötigen diese Pakete (es ginge auch über base):

```
library(stringr)
library(readr)
library(dplyr)
```

Dann lesen wir die Daten ein, zuerst die Datei mit den negativen Konnotationen:

---

<sup>60</sup><http://asv.informatik.uni-leipzig.de/download/sentiws.html>

<sup>61</sup>[http://asv.informatik.uni-leipzig.de/publication/file/155/490\\_Paper.pdf](http://asv.informatik.uni-leipzig.de/publication/file/155/490_Paper.pdf)

```
neg_df <- read_tsv("SentiWS_v1.8c_Negative.txt", col_names = FALSE)
names(neg_df) <- c("Wort_POS", "Wert", "Inflektionen")

glimpse(neg_df)
```

Dann parsen wir aus der ersten Spalte (Wort\_POS) zum einen den entsprechenden Begriff (z.B. “Abbau”) und zum anderen die Wortarten-Tags (eine Erläuterung zu den Wortarten-Tags findet sich hier<sup>62</sup>).

```
neg_df %>%
  mutate(Wort = str_sub(Wort_POS, 1, regexp("\\|", .\$Wort_POS)-1),
         POS = str_sub(Wort_POS, start = regexp("\\|", .\$Wort_POS)+1)) -> neg
```

**str\_sub** parst<sup>63</sup> zuerst das Wort. Dazu nehmen wir den Wort-Vektor Wort\_POS, und für jedes Element wird der Text von Position 1 bis vor dem Zeichen | geparsst; da der Querstrich ein Steuerzeichen in Regex muss er escaped werden. Für POS passiert das gleiche von Position |+1 bis zum Ende des Text-Elements.

Das gleiche wiederholen wir für positiv konnotierte Wörter.

```
pos_df <- read_tsv("SentiWS_v1.8c_Positive.txt", col_names = FALSE)
names(pos_df) <- c("Wort_POS", "Wert", "Inflektionen")
pos_df %>%
  mutate(Wort = str_sub(Wort_POS, 1, regexp("\\|", .\$Wort_POS)-1),
         POS = str_sub(Wort_POS, start = regexp("\\|", .\$Wort_POS)+1)) -> pos
```

Schließlich schweißen wir beide Dataframes in einen:

```
bind_rows("neg" = neg_df, "pos" = pos_df, .id = "neg_pos") -> sentiment_df
sentiment_df %>% select(neg_pos, Wort, Wert, Inflektionen, -Wort_POS) -> sentiment
```

<sup>62</sup>[http://www.jlcl.org/2013\\_Heft1/H2013-1.pdf](http://www.jlcl.org/2013_Heft1/H2013-1.pdf)

<sup>63</sup>“parsen” ist denglisch für “einlesen” von engl. “to parse”

```
knitr::kable(head(sentiment_df))
```

neg_pos	token	Wert	Inflektionen
neg	Abbau	-0.058	Abbaus,Abbaues,Abbauen,Abbaue
neg	Abbruch	-0.005	Abbruches,Abbrüche,Abbruchs,Abbrüchen
neg	Abdankung	-0.005	Abdankungen
neg	Abdämpfung	-0.005	Abdämpfungen
neg	Abfall	-0.005	Abfalles,Abfälle,Abfalls,Abfällen
neg	Abfuhr	-0.337	Abfuhen

## 0.75 Verweise

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



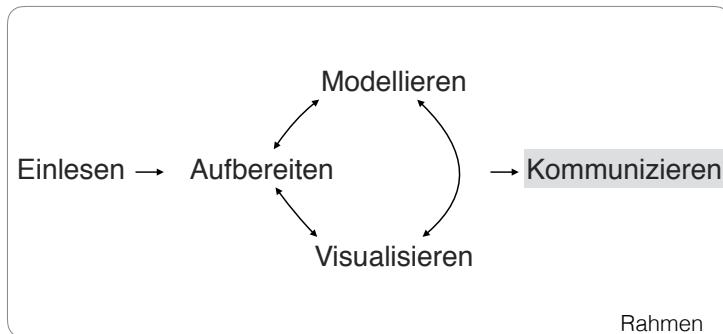
# V ERGEBNISSE KOMMUNIZIEREN



# Was ist RMarkdown?

Benötigte Pakete:

```
library(knitr)  
library(tidyverse)
```



Nehmen wir an, Sie möchten die Ergebnisse Ihrer Datenanalyse niederschreiben. Sei es, dass Sie einen Professor beglücken müssen wollen; sei es, dass Sie einem Kunden eine Bericht der Quartalszahlen inkl. Forecast erstellen oder mit einem Kollegen Ideen austauschen. Welche Anforderungen stellen Sie an ein Werkzeug, mit dem Sie die Ergebnisse niederschreiben?

## 0.76 Forderungen

Einige Anforderungen an Werkzeuge zur Berichterstellung sind im Folgenden aufgelistet:

1. R sollte integrierbar sein; sowohl die Ergebnisse als auch die Syntax.
2. Man sollte sich beim Schreiben auf das Schreiben konzentrieren können - ohne sich auf die Formatierung konzentrieren zu müssen.
3. Das Werkzeug sollte einfach zu erlernen (und zu bedienen) sein.
4. Es sollte verschiedene Ausgabeformate (PDF, HTML,...) unterstützen.
5. Das Dokument sollte optisch ansprechend formatiert sein.
6. Das Werkzeug sollte übergreifend (Betriebssysteme, Zeiten, Versionen) arbeiten.
7. Das Werkzeug sollte mächtig (funktionsreich) sein.
8. Das Werkzeug sollte frei (quelloffen sowie ggf. kostenfrei) sein.
9. Das Werkzeug sollte Kollaboration leicht machen.
10. Das Werkzeug sollte Versionierungen und Änderungsverfolgung unterstützen.

Mit Werkzeug ist hier die Software zur Erstellung des Berichts gemeint; häufig MS Word.

Betrachten wir die einzelnen Forderungen näher.

**R sollte integrierbar sein; sowohl die Ergebnisse als auch die Syntax.**

Sind das Werkzeug der Datenanalyse und das Werkzeug zur Berichterstellung voneinander getrennt, gibt es eine Schnittstselle, die wie eine Sollbuchstelle wirkt. Dieser Graben ist stets zu überwinden. Zwei Dinge, die dann fast zwangsläufig passieren: Copy-Paste und manuelles Aktualisieren. Copy-Paste ist mühsam und fehleranfällig. Mitunter weicht man vielleicht auf Neu-Eintippen aus: "Ok, der Mittelwert war doch 12,34..."; Fehler sind dann vorprogrammiert. Zum manuellen Aktualisieren: Stellen Sie sich vor, Ihr Bericht beinhaltet sagen wir 50 Diagramme und eine Reihe Tabellen und sonstige Zahlen - pro Kunde, pro Berichtszeitraum. Sie haben 100 Kunden, die einen Bericht pro Woche verlangen... Und jetzt für jedes Diagramm etc. in der Berichts-Software auf "Einfügen..." klicken??? Professionell geht anders. Kurz: Die Analyse sollte sich nahtlos in Ihren Bericht einfügen. Das ist vor allem für die Ergebnisse (Zahlen, Diagramme, Tabellen...) wichtig, aber sekundär auch für die Syntax.

Man sollte sich beim Schreiben auf das Schreiben konzentrieren können – ohne sich a

Entscheidend sind die Inhalte, die Formatierung ist zweitrangig und außerdem zeitlich nachgelagert. Bietet das Werkzeug reichhaltige Möglichkeiten zur Formatierung besteht die Gefahr, dass verfrüht der geistige Fokus von den Inhalten zur Dekoration der Inhalte wandert. Besser ist es, sich in gestalterischer Askese zu üben und die mentalen Ressourcen komplett auf die Inhalte konzentrieren zu können. Außerdem ist es wünschenswert, wenn das Werkzeug sich selbstständig um das Formatieren kümmert. Damit soll dem Autor nicht die Möglichkeit verwehrt sein, selber zu gestalten; doch sollte ihm optisch ansprechende Standards automatisch angeboten werden.

Das Werkzeug sollte einfach zu erlernen (und zu bedienen) sein.

Nimmt mir das Werkzeug die Formatierung weitgehend ab, so bleibt (fast nur) das reine Schreiben von Text übrig. Das ist einfach zu bewerkstelligen (oder sollte es sein). Aufwändiges Einarbeiten in neue Werkzeuge sollte entfallen.

Es sollte verschiedene Ausgabeformate (PDF, HTML,...) unterstützen.

Die wichtigsten Formate, um Berichte zu erstellen sind sicherlich PDF, DOC, PPT und neuerdings zunehmend HTML und EPUB-Formate. Diese Formate sollten unterstützt werden. HTML und EPUB gewinnen mit der Verbreitung elektronischer Lesegeräte an Bedeutung. So ist z.B. HTML an einem Tablett oder Handy besser zu lesen als eine PDF-Datei. Ein wesentlicher Grund ist, dass bei HTML Zeilenumbrüche flexibel sind, z.B. je nach Größe des Displays. Bei PDF-Dateien ist dies fix.

Das Dokument sollte optisch ansprechend formatiert sein.

Das Werkzeug sollte - idealerweise als Standard ohne Eingriffe des Nutzers - optisch ansprechende Dokumente erzeugen. Dazu gehört vor allem schöner Schriftsatz: keine hässlichen Löcher zwischen Wörtern oder zwischen Buchstaben, elegante Schriftsätze oder schöne Formeln. Für HTML-Dateien sind "coole" Designs - z.B. mit benutzerfreundlicher Navigationsspalte - sinnvoll.

Das Werkzeug sollte übergreifend (Betriebssysteme, Zeiten, Versionen) arbeiten.

Ärgerlich ist, wenn das Werkzeug nur für bestimmte Betriebssysteme (oder

Versionen davon) funktioniert. Nicht so schlimm, aber auch nervig ist, wenn die Varianten eines Werkzeugs zwar übergreifend funktionieren, aber nicht *ganz* gleich sind - wenn z.B. Farben sich ändern oder Textfelder verrutschen. Auch in ein paar Jahren sollten die Dateien noch lesbar sein.

Das Werkzeug sollte mächtig (funktionsreich) sein.

Die Quadratur des Kreises: Ein Werkzeug sollte einfach zu bedienen sein, aber mächtig, also einen großen Funktionsumfang besitzen. Beide Ziele sind nicht leicht unter einen Hut zu bringen und vielleicht das Grundproblem von Benutzerfreundlichkeit von Computersystemen. Eine pragmatische Lösung ist es (oder sollte sein), dem Nutzer vernünftige Standardwerte anzubieten bzw. diese ungefragt anzuwenden aber gleichzeitig dem Nutzer die Möglichkeit geben, in viele Details einzutreten. Letzteres ist häufig kompliziert(er), aber für viele Nutzer nicht nötig, wenn die Standards gut sind.

Das Werkzeug sollte frei (quelloffen sowie ggf. kostenfrei) sein.

Als "Normalnutzer" denken Sie vielleicht: "Ist mir doch egal, ob das Werkzeug von Firma XYZ angeboten wird, kauf ich mir halt!." Das ist in einigen Fällen stimmig. Aber: Die Erfahrung zeigt, dass einige quelloffene Software tendenziell schneller weiterentwickelt wird (oder Fehler beseitigt werden). Fortgeschrittene Nutzer können so einfach ihre Ideen zur Verfügung stellen. Das geht viel schneller (in der Regel) als wenn es bei einem Konzern durch die Instanzen sickern muss. R ist bestes Beispiel dafür. Außerdem: Manche Werkzeuge der Datenanalyse sind *sehr* teuer; da ist es schön, wenn man (bessere) Leistung kostenfrei bekommt! Nicht jeder ist mit ausreichend Finanzmitteln gesegnet..."Offen" beinhaltet in dem Zusammenhang idealerweise auch, dass die Quelldateien menschenlesbar sind, also reine Text-Dateien. Damit kann jeder, auch ohne ein bestimmtes Werkzeug zu nutzen, die Quelldatei lesen. Reine Textdateien haben wohl von allen Datei-Formaten die beste Aussicht, in 10, 20, 50 oder 100 Jahren noch lesbar zu sein.

Das Werkzeug sollte Kollaboration leicht machen.

Mit Kollaboration ist gemeint, Sie schreiben einen Bericht sagen wir mit 5 Kollegen. Eine beliebte Variante der Zusammenarbeit ist dabei, dass Sie Ihren Entwurf an die Kollegen senden mit der Bitte um Hinweise, Korrekturen oder Überarbeitung. Netterweise bekommen Sie 6 Versionen zurück (ein Kollege schrieb 3 Mails, sie wusste dann selber nicht mehr warum, dafür

meldete sich ein Kollege gar nicht). Ihnen bleibt die glorreiche Aufgabe, diese 6+1 Dokumente zusammenzuführen. Natürlich widersprechen sich die Hinweise der Kollegen, die in Unkenntnis der Hinweise der anderen entstanden... Außerdem nutzten nicht alle die selbe Methode für ihre Hinweise. Kurz: Nach einigen Schreikrämpfen müssen Sie erstmal Spazieren gehen. Schöner wäre ein zentrales Dokument, an dem alle arbeiten. So kann jeder den aktuellen Stand sehen und idealerweise auch die Änderungen der Kollegen (oder die eigenen Änderungen) einsehen. Einfache Werkzeuge dafür sind z.B. Google Docs oder der Dropbox, welche Versionierungsfunktionen bietet. Besser (aber komplexer in der Bedienung) sind spezielle Versionierungs-Werkzeuge. Das bekannteste heißt "Git" (<https://de.wikipedia.org/wiki/Git>).

Das Werkzeug sollte Versionierungen und Änderungsverfolgung unterstützen.

Ähnlich zur Kollaboration ist die Versionierung. Hier geht es darum, die Entwicklungshistorie eines Dokuments nachvollziehen zu können. Das ist umso wichtiger, je größer das Dokument bzw. das Projekt zu dem Dokument ist. Eine studentische Abschlussarbeit ist ein gutes Beispiel, aber viele Berichte in Unternehmen taugen auch als Beispiel. Haben Sie schon mal Dateinamen gesehen wie "Bericht\_V23\_Korrektur>Edit\_Willi\_Final\_2017-02-23\_v3\_final\_Ergänzung.pdf"? Ich gebe zu, das Beispiel ist bewusst auf die Spitze getrieben, aber es schält die Problematik gut heraus. Der Änderungsmodus von MS-Word und ähnlichen Werkzeugen ist hilfreich für eine begrenzte Anzahl von Änderungen. Wurde aber eine bestimmte Passage mehrfach geändert, so kommt dieses Funktionalität schnell an ihre Grenzen. Kurz: Bei wichtigeren oder komplexeren Dokumenten ist eine professionelle Lösung sinnvoll.

RMarkdown erfüllt diese Forderungen. Nicht perfekt, aber besser wohl als jedes andere Werkzeug. Daher ist es ein sehr wertvolles Werkzeug, das Sie kennen sollten, wenn Sie Berichte mit Ergebnissen von Datenanalysen schreiben.

## 0.77 Bevor es losgeht

RMarkdown wird als Teil von RStudio mitgeliefert, kostenlos und großteils quelloffen. Es ist bereits installiert, wenn Sie RStudio installiert haben.

Der Name “RMarkdown” suggeriert, dass R hier eine Rolle spielt. Das ist richtig. Doch was heißt “Markdown”? Der Name röhrt von der Idee von Auszeichnungssprachen (engl. “markup languages”) her. Auszeichnungssprachen sind, einfach gesagt, ein paar Befehle, die man in einen Text hineinschreibt, um den Text zu formatieren. Der Text ist also eine Mischung von Inhalt und Formatierungszeichen. HTML oder TeX sind bekannte Auszeichnungssprachen. Beide sind aber recht komplex, bzw. der Text sieht aus wie Kraut und Rüben vor lauter Auszeichnungsbefehlen. Markdown will anders sein, einfacher. Daher keine “Markup-”, sondern eine “Markdown-”Sprache. Vor “Sprache” braucht man sich hier nicht beunruhigen zu lassen. Die “Sprache” Markdown ist in 10 Minuten gelernt!

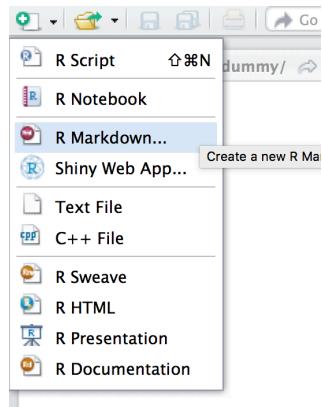
RMarkdown-Dateien haben sinnvollerweise die Endung `.Rmd`. Rmd-Dateien sind, genau wie normale Markdown-Dateien, schnöde Text-Dateien und können entsprechend von jedem Text-Editor (auch mit RStudio, das hier mit Syntax-Highlighting verwöhnt) geöffnet werden.

Erstellt man eine Rmd-Datei, so wird das Verzeichnis, in dem diese Datei liegt, als Arbeitsverzeichnis betrachtet. Möchte man auf eine Datei verweisen, die in einem Unterverzeichnis liegt (z.B. ein Bild laden), so ist es meist am einfachsten, einen relativen Pfad vom Arbeitsverzeichnis anzugeben: `Bilder/mein_bild.png`.

## 0.78 RMarkdown in Action

Werden wir praktisch! Ein Beispiel zur Arbeit mit Markdown.

- Öffnen Sie RStudio.
- Klicken Sie das Icon für “neue Datei” und wählen Sie “R Markdown...”, um eine neue RMarkdown-Datei zu erstellen.
- Dann öffnet sich eine einfache Rmd-Datei, die nicht ganz leer ist, sondern aus didaktischen Gründen ein paar einfache, aber typische, Rmd-



**Figure 19:** Neue Rmd-Datei erstellen

Befehle enthält. Schauen Sie sich den Inhalt kurz an; Sie sehen eine Mischung aus “Prosa” und R.

- Jetzt “verstricken” wir R und Prosa zu einer Datei, HTML in dem Fall, welche sowohl Text, R-Ausgaben als auch R-Syntax enthält. Dafür klicken wir auf das Stricknadel-Icon:

The screenshot shows the RStudio editor window with an Rmd file named 'Untitled.Rmd'. The file contains the following content:

```

1 - ---
2 #title: "Test"
3 output: html_document
4 ---
5
6 ```{r setup, include=FALSE}
7 knitr::opts_chunk$set(echo = TRUE)
8 ---
9
10 ## R Markdown
11
12 This is an R Markdown document. Markdown is a simple formatting syntax for authoring documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.
13
14 When you click the Knit button a document will be generated that includes all
any embedded R code chunks within the document. You can embed an R code chunk
15
16 ```{r cars}
17 knitr::knit()

```

The 'Knit' button in the toolbar is highlighted with a red box.

**Figure 20:** Wir stricken

- Ach ja, RStudio bittet Sie noch, die Rmd-Datei zu speichern. Tun Sie RStudio den Gefallen.
- Als Ergebnis müsste sich im Reiter “Viewer” das Ergebnis zeigen. Sie haben eine, vielleicht Ihre erste HTML-Datei aus einer Rmd-Datei erstellt. Verweilen Sie in Andacht. Sie können sich das Ergebnis, die HTML-Datei im Browser anschauen, in dem Sie betreffende HTML-

Datei im Browser öffnen (diese liegt dort, wo auch die zugehörige Rmd-Datei liegt.)

The screenshot shows the RStudio interface. On the left, the 'Console' and 'R Markdown' panes are visible. The 'R Markdown' pane contains the following Rmd code:

```

1: ---
2: title: "Test"
3: output: html_document
4: ---
5:
6: ````{r setup, include=FALSE}
7: knitr::opts_chunk$set(echo = TRUE)
8: ---
9:
10: # R Markdown
11:
12: This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
13:
14: When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
15:
16: ````{r cars}
17: Test
18: 
```

The right pane, titled 'Test R Markdown', displays the generated HTML output:

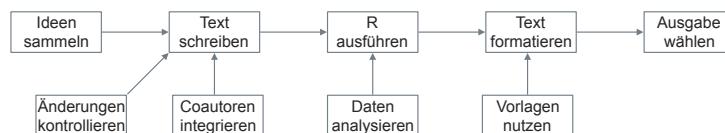
This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

**Figure 21:** HTML-Datei aus Rmd-Datei erstellt

## 0.79 Die Arbeitsschritte mit RMarkdown

Die Arbeitsschritte mit RMarkdown von den ersten Gedanken bis zum fertigen Bericht kann man so zusammenfassen.



**Figure 22:** Die Arbeitsschritte mit RMarkdown

Zuerst bietet es sich an - ganz ohne Gedanken an wohl geformte Sprache und roten Faden - seine Gedanken zu Papier (oder in eine Textdatei) zu kriegen (“Ideen sammeln”). Für viele Menschen sind hier Schmier- und Notizzettel oder Schemata, Mindmaps oder andere Visualisierungen hilfreich. Eine Gliederung zu erstellen gehört auch in diesen Schritt. Hat man schließlich die Struktur erarbeitet, so kann den Text - jetzt mit wohlgeformter(er) Sprache und roten Faden - zusammensetzen (“Text schreiben”). Dabei ist es oft so, dass man die Beiträge der Coautoren ertragen muss integrieren möchte. Dies ist mit am besten mit einer zusätzlichen Software mit dem

Namen “Git” bzw. “Github” zu erreichen, die ebenfalls quelloffen, kostenlos (und weit verbreitet bei Tekkies) ist (“Coautoren integrieren”). Diese Software erlaubt nicht nur Versionierung lokal an einem eigenen Computer, sondern auch in Zusammenarbeit mit Coautoren (“Änderungen nachverfolgen”). Parallel zum Schreiben des Textes sind die Daten zu analysieren; eine Aufgabe, die prinzipiell unabhängig vom Schreiben des Textes ist (“Daten analysieren”). Aber die Ergebnisse der Analyse sind in den Text zu integrieren. Um Schnittstellen zu vermeiden, ist es sinnvoll, den Inhalt-Text sowie die R-Syntax in einer Datei zu “verstricken”. Dieser Schritt (“R ausführen”) wird von einem R-Paket namens “knitr” (Xie 2015) besorgt (engl. “to knit” - stricken). Knitr führt die R-Syntax im Dokument aus und liefert das Ergebnis im Markdown-Format zurück. Aus R+Markdown wird reines Markdown. Die Markdown-Datei kann nun in fast jedes denkbare andere Markup-Format übersetzt werden, die wichtigsten sind HTML und TeX. Die Übersetzung führt RStudio wiederum, genau wie das “Knittern” für uns komfortabel im Hintergrund aus. Dazu wird auf ein Programm names “pandoc” (<http://pandoc.org>) zurückgegriffen. Pandoc übersetzt eine Markup-Sprache in eine andere. Haben wir z.B. in TeX übersetzt, so können wir - vorausgesetzt TeX, LaTeX o.ä. ist installiert - von TeX in PDF umwandeln lassen. Ist TeX auf Ihrem Rechner installiert, so wird dieser Übersetzung wiederum automatisch vorgenommen.

## 0.80 Aufbau einer Markdown-Datei

Das wichtigste ist: Eine Markdown-Datei (.md) ist eine reine Text-Datei, genau wie eine Rmd-Datei (die ein Spezialfall einer Markdown-Datei ist). Mit jedem Texteditor können Sie sie öffnen und ändern.

```
---
```

```
title: "Untitled"
author: "Sebastian Sauer"
date: "23 2 2017"
output: html_document
---
```

```
```{r setup, include=FALSE}
```

```
knitr::opts_chunk$set(echo = TRUE)
```

```

```
## R Markdown
```

This is an R Markdown document. Markdown is a simple formatting syntax for authoring plain-text documents.

When you click the **Knit** button a document will be generated that includes both

```
```{r cars}
summary(cars)
```

```

Typischerweise besteht eine Markdown-Datei aus drei Teilen:

1. Einem (optionalen) **YAML-Header** abgegrenzt durch ---s.
2. **R-Chunks** (R-Syntax-Abschnitte) abgegrenzt durch ```.
3. Normalem Reintext mit einigen Steuerzeichen wie **\*kursiv\*** oder **#Überschrift\_Ebene\_1**.

Der YAML-Header definiert einige Meta-Daten für Ihre Datei; Dinge wie Autor, Titel, Datum oder Ausgabeformat werden hier festgelegt. Sie müssen keine YAML-Header angeben, dann werden einige Standards verwendet. YAML steht übrigens, falls Sie sich das gerade fragen, für “yet another markup language”, also eine Auszeichnungssprache, die sehr einfach ist und nur für Metadaten zuständig ist.

R-Syntax wird (innerhalb von Rmd-Dateien) nur innerhalb der “Chunks” als R verstanden; außerhalb der eingezäunten R-Abschnitte werden Sie als normaler Text verstanden. Sie können Chunks auch nochmal ausführen, in dem Sie z.B. den Button “Run” klicken. Um das ganze Dokument zu “übersetzen”, reicht ein Klick auf das Stricksymbol.

Der Text bei Markdown sieht im Prinzip so aus, wie man eine Plain-Text-Email früher (oder manchmal heute noch) geschrieben hätte. Auch ohne dass man Markdown kennt, kann man ihn ohne Probleme erfassen.

## 0.81 Syntax-Grundlagen von Markdown

In RStudio wird Pandocs Markdown verwendet; die Übersetzung von Markdown in eine andere Auszeichnungssprache wird komplett von Pandoc abgewickelt.

Man braucht etwas Übung, um sich die Syntax zu merken, aber im Grunde ist es ganz einfach. Schauen Sie selbst:

Text formatieren mit Markdown

---

```
*kursiv* oder so _kursiv_
**fett** __fett__
`R-Syntax`
hochgestellt^2 und tiefgestellt~2~
```

Überschriften

---

```
# 1. Ebene
```

```
## 2. Ebene
```

```
### 3. Ebene
```

Aufzählungen

---

```
* Listenpunkt 1
```

```
* Listenpunkt 2
```

```
  * Listenpunkt 2a
```

```
    * Listenpunkt 2b
```

```
1. Element 1 einer nummerierten Liste
```

1. Element 2. Die korrekte Nummer wird automatisch erstellt.

#### Links und Bilder

---

<http://Beispiel.com>

[Bezeichnung des Links] (http://Beispiel.com)

![Optionale Bildbezeichnung] (path/to/img.png)

#### Tabellen

---

| Spaltenkopf1 | Spaltenkopf1 |
|--------------|--------------|
| Zelleninhalt | Zelleninhalt |
| Zelleninhalt | Zelleninhalt |

Wenn man mal etwas vergisst, kann man in RStudio hier nachschauen: *Help > Markdown Quick Reference*.

## 0.82 Tabellen

Praktisch ist, dass man sich Dataframes einfach als Tabellen ausgeben lassen kann. Normalerweise werden in RMarkdown Dataframes in gewohnter Manier ausgegeben:

```
mtcars %>%
  slice(1:3)
#>   mpg cyl disp  hp drat    wt  qsec vs am gear carb
#> 1 21.0   6 160 110 3.90 2.62 16.5  0  1     4     4
#> 2 21.0   6 160 110 3.90 2.88 17.0  0  1     4     4
#> 3 22.8   4 108  93 3.85 2.32 18.6  1  1     4     1
```

**Table 2:** Eine Tabelle mit Kable

| mpg  | cyl | disp | hp  | drat | wt   | qsec | vs | am | gear | carb |
|------|-----|------|-----|------|------|------|----|----|------|------|
| 21.0 | 6   | 160  | 110 | 3.90 | 2.62 | 16.5 | 0  | 1  | 4    | 4    |
| 21.0 | 6   | 160  | 110 | 3.90 | 2.88 | 17.0 | 0  | 1  | 4    | 4    |
| 22.8 | 4   | 108  | 93  | 3.85 | 2.32 | 18.6 | 1  | 1  | 4    | 1    |

(`slice` filtert die angegebenen Zeilen, hier 1 bis 3.)

Möchte man eine “richtige” Tabelle, so kann man z.B. mit dem Befehl `knitr::kable` arbeiten. Die Tabelle unten (2) wurde so erstellt:

```
mtcars %>%
  slice(1:3) %>%
  kable(caption = "Eine Tabelle mit Kable")
```

## 0.83 Zitieren

Zitate sind bei Rmarkdown, genau wie Bilder und Verweise, nichts anderes als Links, mit einer kleinen Änderungen:

Das ist furchtbar wichtig [©WeisOis2017].

Inerhalb von den eckigen Klammern, die einen Linktext kennzeichnen wird ein Klammeraffe geschriebne (@); dieser ist die Bezeichnung für eine Zitation. Dann folgt die ID der Zitation.

Mehrere Zitationen werden mit Strichpunkt getrennt: So steht es geschrieben [©Weis201

Man kann beliebige Kommentare in die Zitation aufnehmen: So war es schon immer [vgl.

Will man eine Zitation ohne Klammer einfügen, so lässt man die eckigen Klammern weg:

Den Namen des Autors/ der Autoren kann man unterdrücken, in dem man ein `--` vor die Z

Eine Reihe von bibliographischen Formaten werden unterstützt; darunter BibLaTEX, BibTeX, endnote und medline. Die bibliographischen Einträge alle Zitationen fügt man in eine Textdatei (z.B. mit Namen `bibliography.bib`); im YAML-Header gibt man dann den Dateinamen mit den bibliographischen Informationen an:

```
bibliography: bibliography.bib  
csl: apa.csl
```

Den Zitationsstil kann man, ebenfalls im YAML-Header mit der Variable `csl` definieren. Dort verweist man auf eine CSL-Stil-Datei. CSL-Dateien sind quelloffen und man kann sie sich z.B. hier herunterladen: <https://github.com/citation-style-language/styles>.

## 0.84 Kollaboration und Versionierung

Kollaboration und Versionierung sind wichtige Aspekte von professionellem Projektmanagement. Das wichtigste Werkzeug hier heißt “Git”. Git ist eine Software, die Versionierungsfunktionen anbietet - auch über mehrere Coautoren hinweg. “Github” ist ein Anbieter, der bekannteste, der “Projektordner” online anbietet, so dass sich man komfortabel synchronisieren kann. Alternativ gibt es Plattformen, die ähnliche Dienste anbieten wie <http://authorea.com>.

Allerdings ist einiges an Einarbeitung vornötig; wir werden hier nicht weiter auf solche Werkzeuge eingehen.

## 0.85 Verweise

RMarkdown ist ein junges Ökosystem, das schnell wächst.

- Ein guter Startpunkt, mehr über Markdown zu lernen, ist das Buch von Wickham und Grolemund (2016).

- Größere Texte können mit `bookdown` geschrieben werden, eine Erweiterung zu Markdown (dieses Buch wurde so geschrieben):  
<https://bookdown.org/yihui/bookdown/>
- Die Cheatsheets von RStudio sind hilfreich: RStudio \* Tools > Cheat-sheets > ...\*.



# **VI Anhang**



# Studienpfade

Je nach Lernziel, Zeit und Interessen bieten sich unterschiedliche Studienpfade durch dieses Buch an. Im folgenden sind einige aufgezählt - untergliedert nach Fachrichtung, Vorerfahrung und Zeit.

## 0.86 konsequutiver Master of Science in Wirtschaftspsychologie

Anahmen:

- Zeitumfang: 48 UE für Lehre
- Vorerfahrung: Deskriptive Statistik, Inferenzstatistik, Grundlagen R, Grundlagen Visualisierung

| Termin | Thema                                     | Kommentar                      |
|--------|---|--------------------------------|
| 1      | Organisatorisches<br>Einführung<br>Rahmen |                                |
| 2      | Daten einlesen und<br>aufbereiten         |                                |
| 3      | Daten visualisieren                       |                                |
| 4      | Inferenzstatistik                         | kurze Wiederholung             |
| 5      | Daten modellieren<br>Lineare Regression   | Wiederholung und<br>Vertiefung |

| Termin | Thema   | Kommentar |
|--------|---|-----------|
| 6      | klassifizierende<br>Regression                          |           |
| 7      | Baumbasierte Verfahren                                  |           |
| 8      | Fallstudie  |           |
| 9      | Clusteranalyse  |           |
| 10     | Textmining  |           |
| 11     | Fallstudie  |           |
| 12     | Wiederholung und Zusammenfassung ggf. plus Probeklausur |           |

Die einzelnen Kapitel sind dabei nicht umfassend abzuarbeiten. Der Lehrende/ Lernende kann hier eine Auswahl treffen. Teilweise ist der Stoff eines Kapitels - ja nach Vorerfahrung der Lernenden - zu viel für einen Termin (mit jeweils 4 UE).

# Literaturverzeichnis

- Allaire, JJ, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, and 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, and 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. <https://www.amazon.com/Breaking-Law-Averages-Probability-Statistics/dp/0557019907%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0557019907>.
- . 2016. *Uncertainty: The Soul of Modeling, Probability & Statistics*. Springer. <https://www.amazon.com/Uncertainty-Soul-Modeling->

Probability-Statistics-ebook/dp/B01JEJNUJK%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB01JEJNUJK.

Bryant, PG, and 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>.

Cleveland, William S. 1993. *Visualizing Data*. Hobart Press. <https://www.amazon.com/Visualizing-Data-William-S-Cleveland/dp/0963488406%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0963488406>.

Cobb, George W. 2007. "The Introductory Statistics Course: A Ptolemaic Curriculum?" *Technology Innovations in Statistics Education* 1 (1).

Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. "Modeling Wine Preferences by Data Mining from Physicochemical Properties." *Decision Support Systems* 47 (4). Elsevier: 547–53.

de Vries, Andrie, and Brian D. Ripley. 2016. *Ggdendro: Create Dendograms and Tree Diagrams Using 'Ggplot2'*. <https://CRAN.R-project.org/package=ggdendro>.

Feinerer, Ingo, and 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>.

Fox, John, and Sanford Weisberg. 2016. *Car: Companion to Applied Regression*. <https://CRAN.R-project.org/package=car>.

Gigerenzer, Gerd. 1980. *Messung Und Modellbildung in Der Psychologie (Uni-Taschenbucher. Psychologie, Padagogik, Soziologie, Psychiatrie) (German Edition)*. E. Reinhardt. <https://www.amazon.com/Modellbildung-Psychologie-Uni-Taschenbucher-Soziologie-Psychiatrie/dp/3497008958%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%>

26creativeASIN%3D3497008958.

———. 2004. “Mindless Statistics.” *The Journal of Socio-Economics* 33 (5). Elsevier BV: 587–606. doi:10.1016/j.socec.2004.09.033<sup>64</sup>.

Grolemund, Garrett, and Hadley Wickham. 2014. “A Cognitive Interpretation of Data Analysis.” *International Statistical Review* 82 (2). Wiley Online Library: 184–204.

Hahsler, Michael, and Sudheer Chelluboina. 2016. *ArulesViz: Visualizing Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arulesViz>.

Hahsler, Michael, Christian Buchta, Bettina Gruen, and Kurt Hornik. 2016. *Arules: Mining Association Rules and Frequent Itemsets*. <https://CRAN.R-project.org/package=arules>.

Hamermesh, Daniel S, and 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, et al. 2015. “Data Science in Statistics Curricula: Preparing Students to ‘Think with Data.’” *The American Statistician* 69 (4). Taylor & Francis: 343–53.

Head, Megan L., Luke Holman, Rob Lanfear, Andrew T. Kahn, and 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.1002106<sup>65</sup>.

Hendricks, Paul. 2015. *Titanic: Titanic Passenger Survival Data Set*. <https://CRAN.R-project.org/package=titanic>.

Ingo Feinerer, Kurt Hornik, and 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. *Corrr: Correlations in R*. <https://CRAN.R-project.org/package=corrr>.

---

<sup>64</sup><https://doi.org/10.1016/j.socec.2004.09.033>

<sup>65</sup><https://doi.org/10.1371/journal.pbio.1002106>

[project.org/package=corr](https://CRAN.R-project.org/package=corr).

James, Gareth, Daniela Witten, Trevor Hastie, and Rob Tibshirani. 2013a. *ISLR: Data for an Introduction to Statistical Learning with Applications in R*. <https://CRAN.R-project.org/package=ISLR>.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013b. *An Introduction to Statistical Learning*. Vol. 6. Springer.

Julia, PhD Silge, and PhD Robinson David. 2017. *Text Mining with R: A Tidy Approach*. O'Reilly Media. <https://www.amazon.com/Text-Mining-R-tidy-approach/dp/1491981652%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1491981652>.

Kim, Albert Y., and Adriana Escobedo-Land. 2015. “OkCupid Data for Introductory Statistics and Data Science Courses.” *Journal of Statistics Education* 23 (2). Citeseer: n2.

Kim, Albert Y., and 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>.

Kuhn, Max, and Kjell Johnson. 2013. *Applied Predictive Modeling*. Vol. 26. Springer.

Ligges, Uwe, Martin Maechler, and Sarah Schnackenberg. 2017. *Scatterplot3d: 3D Scatter Plot*. <https://CRAN.R-project.org/package=scatterplot3d>.

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 Nat-*

- ural 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>.
- Peirce, Charles S. 1955. “Abduction and Induction.” *Philosophical Writings of Peirce* 11. New York.
- Peng, Roger D, and Elizabeth Matsui. 2015. “The Art of Data Science.” *A Guide for Anyone Who Works with Data.* Skybrude Consulting 200: 162.
- Raiche, Gilles, and David Magis. 2011. *NFactors: Parallel Analysis and Non Graphical Solutions to the Cattell Scree Test.* <https://CRAN.R-project.org/package=nFactors>.
- Ram, Karthik, and 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, and 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<sup>66</sup>](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, and Julia Silge. 2016. *Tidytext: Text Mining Using ‘Dplyr’, ‘Ggplot2’, and Other Tidy Tools.* <https://CRAN.R-project.org/package=tidytext>.

---

<sup>66</sup>[http://www.transtats.bts.gov/DL%7B/\\_%7DSelectFields.asp?Table%7B/\\_%7DID=236](http://www.transtats.bts.gov/DL%7B/_%7DSelectFields.asp?Table%7B/_%7DID=236)

**tidytext.**

Robinson, David, Mathieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, and Hadley Wickham. 2015. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*. <https://CRAN.R-project.org/package=broom>.

Romeijn, Jan-Willem. 2016. “Philosophy of Statistics.” In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Winter 2016. <http://plato.stanford.edu/archives/win2016/entries/statistics/>.

Sauer, Sebastian. 2016. “Extraversion Dataset.” Open Science Framework. doi:10.17605/OSF.IO/4KGZH<sup>67</sup>.

———. 2017a. “Dataset ‘Predictors of Performance in Stats Test’.” Open Science Framework. doi:10.17605/OSF.IO/SJHUY<sup>68</sup>.

———. 2017b. “Dataset ‘Height and Shoe Size’.” Open Science Framework. doi:10.17605/OSF.IO/JA9DW<sup>69</sup>.

Schloerke, Barret, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Joseph Larmarange. 2016. *GGally: Extension to 'Ggplot2'*. <https://CRAN.R-project.org/package=GGally>.

Silge, Julia. 2016. *Janeaustenr: Jane Austen’s Complete Novels*. <https://CRAN.R-project.org/package=janeaustenr>.

Silge, Julia, and 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.00037<sup>70</sup>.

Silge, Julia, David Robinson, and Jim Hester. 2016. “Tidytext: Text Mining Using Dplyr, Ggplot2, and Other Tidy Tools.” doi:10.5281/zenodo.56714<sup>71</sup>.

*The Oxford Dictionary of Statistical Terms.* 2006. Oxford University Press. <https://www.amazon.com/Oxford-Dictionary-Statistical-Terms/dp/0199206139%3FSubscriptionId%3D0JYN1NVW651KCA56C102%>

<sup>67</sup><https://doi.org/10.17605/OSF.IO/4KGZH>

<sup>68</sup><https://doi.org/10.17605/OSF.IO/SJHUY>

<sup>69</sup><https://doi.org/10.17605/OSF.IO/JA9DW>

<sup>70</sup><https://doi.org/10.21105/joss.00037>

<sup>71</sup><https://doi.org/10.5281/zenodo.56714>

[26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0199206139.](https://doi.org/10.3758/bf03194105)

Therneau, Terry, Beth Atkinson, and Brian Ripley. 2015. *Rpart: Recursive Partitioning and Regression Trees*. <https://CRAN.R-project.org/package=rpart>.

Tufte, Edward R. 1990. *Envisioning Information*. Graphics Press. <https://www.amazon.com/Envisioning-Information-Edward-R-Tufte/dp/1930824149%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824149>.

———. 2001. *The Visual Display of Quantitative Information*. Graphics Press. <https://www.amazon.com/Visual-Display-Quantitative-Information/dp/1930824130%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824130>.

———. 2006. *Beautiful Evidence*. Graphics Press. <https://www.amazon.com/Beautiful-Evidence-Edward-R-Tufte/dp/1930824165%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1930824165>.

VanDerWal, Jeremy, Lorena Falconi, Stephanie Januchowski, Luke Shoo, and Collin Storlie. 2014. *SDMTools: Species Distribution Modelling Tools: Tools for Processing Data Associated with Species Distribution Modelling Exercises*. <https://CRAN.R-project.org/package=SDMTools>.

Wagenmakers, Eric-Jan. 2007. “A Practical Solution to the Pervasive Problems Of p Values.” *Psychonomic Bulletin & Review* 14 (5). Springer Nature: 779–804. doi:10.3758/bf03194105<sup>72</sup>.

Warnes, Gregory R., Ben Bolker, Lodewijk Bonebakker, Robert Gentleman, Wolfgang Huber Andy Liaw, Thomas Lumley, Martin Maechler, et al. 2016. *Gplots: Various R Programming Tools for Plotting Data*. <https://CRAN.R-project.org/package=gplots>.

Wei, Taiyun, and Viliam Simko. 2016. *Corrplot: Visualization of a Correla-*

---

<sup>72</sup><https://doi.org/10.3758/bf03194105>

- tion 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, and 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.2016.01832<sup>73</sup>.
- 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.i10<sup>74</sup>.
- . 2016a. *Reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package*. <https://CRAN.R-project.org/package=reshape2>.
- . 2016b. *Tidyr: 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, and Romain Francois. 2016. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Visualize, Model, Transform, Tidy, and Import Data*. O'Reilly Media. <https://www.amazon.com/Data-Science-Visualize-Model-Transform/dp/1491910399%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D1491910399>.
- Wickham, Hadley, Jim Hester, and Romain Francois. 2016a. *Readr: Read*

---

<sup>73</sup><https://doi.org/10.3389/fpsyg.2016.01832>

<sup>74</sup><https://doi.org/10.18637/jss.v059.i10>

- Tabular Data.* <https://CRAN.R-project.org/package=readr>.
- . 2016b. *Readr: Read Tabular Data.* <https://CRAN.R-project.org/package=readr>.
- Wild, Chris J, and 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 ed. 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, and Jim Porzak. 2014. *Practical Data Science with R.* Manning.