R für Psychologen

Jan Philipp Nolte

The Health and Life Science University

Contents

1. Einleitung

"Everybody [...] should learn to program a computer, because it teaches you how to think." - Steve Jobs

1.1 Warum R?

Wenn man sich anschaut mit welchen Statistikprogrammen in den verschiedenen akademischen Fachrichtungen gearbeitet wird, kann sich das Gefühl breit machen, von der Vielfalt erschlagen zu werden. Warum R und nicht SPSS, Stata, SAS, MATLAB, Octave, Julia oder Python? Wenn man nicht gerade plant während seiner restlichen Laufbahn jährlich beträchtliche Summen an kommerzielle Unternehmen zu zahlen, kommen SPSS, Stata, SAS und MATLAB schon einmal nicht in Frage. Natürlich gibt es mehr als nur einen Grund einen großen Bogen um SPSS zu machen, aber da wir uns hier anschauen, was R ausmacht und nicht, was SPSS nicht ausmacht, sei an dieser Stelle darauf verzichtet.

In der Psychologie wird man häufig mit Datensätzen konfrontiert, die nicht nur Zahlen, sondern auch Spalten mit soziodemographischen Daten enthalten. Aufgrund dieser Besonderheit fällt Octave ebenfalls weg, da diese Datenstruktur wie bei MATLAB nicht unterstützt wird. Bleiben noch R, Julia und Python. Die Entscheidung ist vor allem abhängig von der geplanten Anwendung, der Datenmenge und Performanceansprüchen sowie persönlicher Präferenzen. Für Psychologen bietet R allerdings alles, was man sich zur Auswertung wünschen kann. Eine einfache Syntax, zahlreiche Erweiterungen zur Erleichterung vieler Aufgaben und die Möglichkeit einfach Graphen zu erstellen, die APA Richtlinien entsprechen. Wenn man anhand von Datensätzen mit mehreren Millionen Einträgen durchgängig Modelle trainieren möchte, sollte man zum jetztigen Zeitpunkt definitiv lieber auf Julia oder Python zurückgreifen. Allerdings ist Julia noch eine sehr junge Sprache und hat deswegen eine geringere Nutzerzahl und deswegen weniger Erweiterungen und Python ist eine allgemein anwendbare Programmiersprache, wodurch sich der Einstieg als deutlich umständlicher erweist.

Für Psychologen ist R also die beste Wahl. Viele argumentieren immer wieder, dass die *Meisten* SPSS nutzen würden. Innerhalb der letzten Jahre ist R rasant gewachsen und zur 6. beliebtesten Programmiersprache der Welt geworden geworden. Dafür, dass R *nur* eine Sprache für Statistik ist, ist das ziemlich beeindruckend. Eine Trendwende ist nicht in Sicht. Es ist nur eine Frage der Zeit bis R auch die breite arbeitende Masse der Psychologen erreicht. Also worauf warten wir noch?

1.2 Für wen ist dieses Tutorial?

Dieses Tutorial setzt keine Vorkenntnisse in R voraus. Somit ist es für jeden geeignet, der R lernen möchte. Aber auch jene, die bereits Vorkenntnisse mit Base R haben und in die Weiten des **tidyverse** aufbrechen wollen, kommen auf ihre Kosten. Dabei wird ein besonderer

Fokus auf Probleme gelegt, mit denen Psychologen konfrontiert werden. Daher wird keine Vollständigkeit für andere Fachrichtungen beansprucht.

1.3 Bearbeitungsstrategie

Natürlich kann man das Tutorial bearbeiten, wie man möchte. Es wird allerdings empfohlen bis einschließlich 4. zu lesen und dann der Anleitung in Abschnitt 13 zu folgen, damit man die neu kennengelernten Funktionen sofort selbst ausprobieren kann. Da der Fokus auf dem Lernen von R liegt, kann man die statistischen Abschnitte 6. und 9. erst einmal überspringen. Der 12. Abschnitt stellt nützliche Zusatzinformationen bereit, die über die grundlegend notwendigen Fertigkeiten mit R hinaus gehen. Wenn man also Abschnitte 1-5, 7, 8, 10 und 11 gemacht hat, kann man wie unter 13. beschrieben, versuchen, das Gelernte in einer interaktiven Übungsumgebung anzuwenden.

Falls Probleme auftreten oder Dir im Laufe des Tutorials etwas auffällt, was unklar formuliert ist oder gar fehlt, ist es erwünscht auf Github eine neue **Issue** zu eröffnen.

Programmieren lernen ist anders als Fremdsprachen lernen. Man lernt nicht zuerst alle Vokabeln und die Grammatik auswendig. Viel mehr lernt man nur die Grammatik in Form von grundlegenden Regeln und die wichtigsten Vokabeln. Alles Andere schlägt man anwendungsbezogen immer wieder nach, bis man sie auswendig kann. Es ist also keine Schmach für jede aufkommende Frage während der Übungen oder bei eigenen Projekten das Tutorial zur Hilfe zu verwenden. Im Gegenteil! Wichtig ist nur, dass man genau weiß, an welcher Stelle man suchen muss. Viel Spaß!

1.4 Set up your lab

1.4.1 Installiere R und RStudio

Installiere zuerst die Programmiersprache R hier. Anschließend installiere die Programmierumgebung RStudio unter diesem Link.

1.4.2 Never save your workspace

Wenn sich ein Fehler irgendwo im Code eingschlichen hat, kann es wahnsinnig zeitintensiv sein, diesen zum Beheben zu finden. Manche schwer zu behebende Fehler können durch das automatische Speichern des Workspace entstehen. So stellst Du es aus:

1. Gehe zu: Tools/Global Options. . . /General
2. Entferne den Haken bei: **restore .RData into workspace at startup**
3. Ändere **Save workspace to .RData on exit** zu **never**

Keine Sorge, auf das Speichern Deines Codes hat das keine Auswirkung.

1.4.3 Default text encoding

Damit man den geschriebenen Code fehlerfrei auf anderen Geräten lesen kann, ist es wichtig die selbe Zeichencodierung zu wählen. Die modernste ist UTF-8.

1. Gehe zu: Tools/Global Options. . . /Code/Saving
2. Ändere: **Default text encoding** zu **UTF-8**

1.4.4 Konsole, Script und Markdown

Es gibt im Grunde genommen drei Möglichkeiten den eigenen Code in RStudio auszuführen.

Konsole: Einmaliges Ausführen des Codes. Wenn man später erneut den selben Befehl ausführen möchte, muss man ihn neu eingeben.

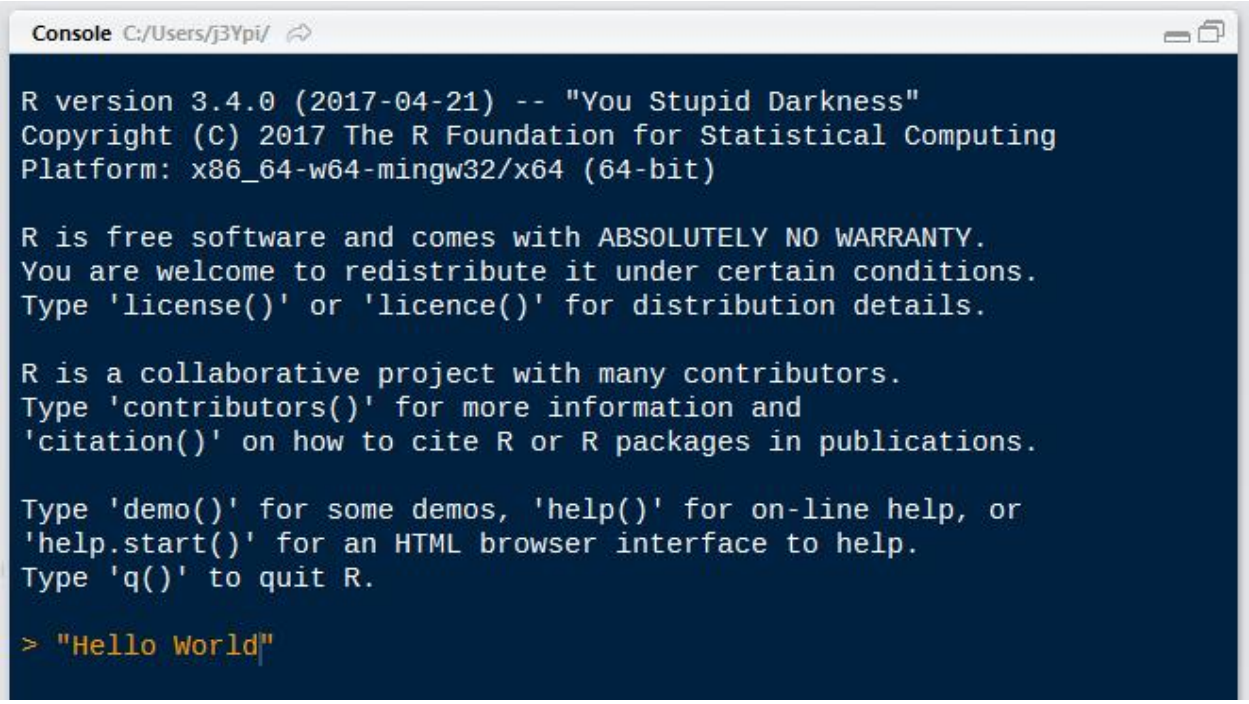


Figure 1: Konsole

Script: Sämtliche Relevanten Befehle sind in einem Script gespeichert. Um ein neues Script zu öffnen, drücke crtl (strg) + shift + N. Zum Ausführen drücke in der jeweiligen Zeile crtl (strg) + enter. Falls alles ausgeführt werden soll, drücke zuerst crtl (strg) + a und führe es dann aus. Gespeichert wird mit crtl (strg) + S.

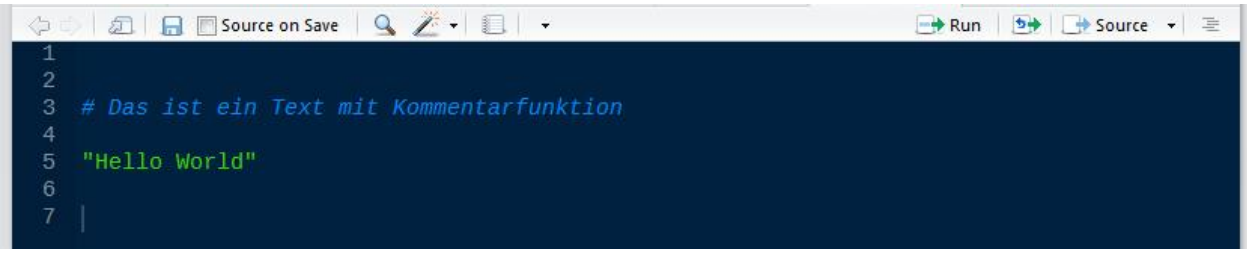


Figure 2: Script

Markdown: Ermöglicht es schnell gut formatierte Reports innerhalb von R zu schreiben. Der Vorteil im Vergleich zu Scripten ist die größere Übersichtlichtkeit und die Möglichkeit Text zu schreiben, ohne die Kommentierfunktion zu benutzen. Ein Markdown Dokument kann man über File/New File und dann R Notebook oder R Markdown erstellen. Mehr Informationen gibt es hier.

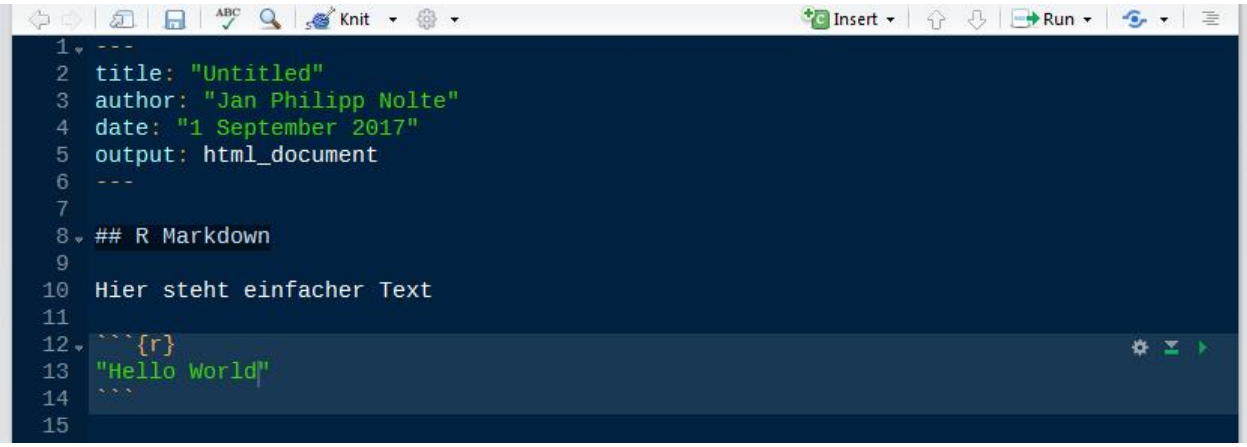


Figure 3: Markdown

2. Was ist ein Package?

R bietet von Beginn an eine Bandbreite von Funktionen. Da diese Funktionen allerdings Basics darstellen und nicht auf spezifisch Anwendungskontexte zugeschnitten sind, gibt es so genannte Packages. Packages sind i.d.R kostenlose Erweiterungen von Dritten, die verschiedenste Aufgaben erheblich erleichtern können. Die Packages werden auf CRAN geteilt. Es gibt mitlerweile mehr als 11 000 Packages.

2.1 Packages installieren und laden

Um ein Package zu installieren, muss man **install.packages("packageName", dependencies = TRUE)** ausführen. Wichtig ist hierbei, dass der Packagename in Anführungszeichen

geschrieben ist. Das Argument **dependencies = TRUE** installiert hierbei zusätzlich alle Packages, auf denen das gewünschte Package basiert. Da Programmierer faul sind, kann man TRUE in R grundsätzlich auch einfach mit T abkürzen. Damit man auf die Funktionen des Packages zurückgreifen kann, muss das Package aus der Bibliothek mithilfe von **library()** geladen werden. Hierbei sind keine Anfürungszeichen notwendig.

```
install.packages("tidyverse", dependencies = T)
library(tidyverse)
```

2.2 Der tidyverse Ansatz

Tidyverse ist eine Zusammenstellung mehrerer Packages, die aufeinander abgestimmt sind. Gegenüber Base R macht es vieles im Kontext von Datenverarbeitung erheblich einfacher. Und wie wir alle wissen, nimmt die Datenvorbereitung mit Abstand die meiste Zeit in der Auswertung ein. Der Großteil der Aufgaben, die wir mithilfe von tidyverse Packages machen werden, kann man auch mit normalen Base R Funktionen ausführen. Allerdings kommt man mithilfe von tidyverse einfacher und schneller ans Ziel, mit dem Nebeneffekt einer nachvollziehbareren Syntax. Gerade wenn man sich Monate später den eigenen Code noch einmal bspw. zur Wiederverwendung ansieht, lernt man den Wert von Lesbarkeit zu schätzen. Zur Lesbarkeit trägt auch die angemessene Verwendung von **Leerzeichen** bei. Hier die Funktionen der Packages, die wir aus tidyverse benutzen werden.

- ggplot2 / ggpubr: Graphen erstellen.
- dplyr / tidyr: Datenvorbereitung und Manipulation
- readr: Einlesen der gängigsten Dateientypen
- haven: Einlesen von SPSS / SAS Dateien
- broom: Umwandlung statistischer Analysen ins tidy Format
- purrr: functional programming

Der Hintergrund von tidyverse ist es, **tidy data** zu kreieren. Man spricht von **tidy data**, wenn:

1. Jede Variable eine Spalte ist
2. Jede Beobachtung eine Zeile ist
3. Jeder Wert in einer Zelle ist

3. Was ist ein Working Directory?

Das Working Directory ist jener Ort, in dem man Arbeiten möchte. In der Regel liegt dort der Datensatz, den man einlesen möchte. Um sicherzugehen, dass R den Datensatz auch findet, muss man vor Einlesen des Datensatzes den Pfad zu dem Datensatz als Working Directory festlegen.

3.1 getwd() und setwd()

Mithilfe von getwd() kann man das momentane Working Directory sehen. Um den neuen Pfad festzulegen muss man setwd() ausführen. Hierbei ist zu beachten, dass der Pfad sich bei Windows und UNIX (macOS, Linux) basierten System grundsätzlich unterscheidet. Optional kann man mit einem kleinen Trick die Schreibweise unter Windows jedoch vereinfachen (nichts ändern unter macOS und Linux):

1. Gehe zu **Global Options. . . /Default Working Directory(When not in a project)**
2. Ändere den Pfad auf "C:/Users/userName"
3. Starte RStudio neu

Nun unterscheiden sich die beiden Pfade nurnoch in einem Zeichen. Beispielhaft der Pfad zum Desktop als Beispiel für Windows und für UNIX Systeme:

```
## Windows:
# vor Änderung
setwd("C:/Users/userName/Desktop")
# nach Änderung
setwd("./Desktop")
## UNIX:
setwd("~/Desktop")
```

3.2 Einlesen und Speichern häufiger Dateientypen

Damit man mit einem Datensatz in R arbeiten kann, muss man ihn zuerst einlesen. Die üblichsten Formate seien hier aufgeführt. Beim Speichern wird anstelle von **read_** der Befehl **write_** verwendet.

3.2.1 .RData

Datensätze im R Format enden mit .RData und werden als einziger Dateityp mit load() geladen.

```
load("data.RData")
save(bigFive, file = "data.RData")
```

3.2.2 .csv und .tsv

csv Files oder comma seperated values, ist die üblichste Dateiform. Dabei sind die Daten entweder mit einem Komma oder mit einem Semikolon getrennt. Ein tsv file hat zwar die selbe Endung wie ein csv, allerdings sind die Werte dort mit einem TAB getrennt (tab

seperated values). Wir greifen beim Einlesen auf Funktionen von readr zurück. Wir müssen das Package nicht seperat laden, weil es durch tidyverse automatisch geladen wird.

```
## Komma
read_csv("data.csv")
write_csv(bigFive, path = "data.csv")
## Semikolon
read_csv2("data.csv")
## Tabulator
read_tsv("data.csv")
```

3.2.3 .xlsx

Microsoft Excel Dateien haben die Endung xlsx. Um diese einlesen und speichern zu können benötigen wir ausnahmsweise tidyverse fremde Packages Namens writexl und readxl.

```
library(readxl)
library(writexl)

read_xlsx("data.xlsx")
write_xlsx(bigFive, "data.xlsx")
```

3.2.4 .sav

SPSS speichert Daten im sav Format. Tidyverse bietet dafür ein Package namens *haven* an, welches allerdings extra geladen werden muss.

```
library(haven)
read_spss("data.sav")
write_sav(bigFive, data = "data.sav")
```

3.3 Zuweisungspfeil

Damit man den Datensatz auch speichert, muss man ihn einer Variable zuweisen. Dies macht man in R mit einem Zuweisungspfeil. Dabei wird die Variable in die lokale **Environment** gespeichert. Wichtig zu verstehen ist, dass in R nichts gespeichert wird, wenn man es nicht einer Variable zuweist. Variablen kann man grundsätzlich fast so benennen wie man möchte. Man darf nur nicht mit einer Zahl anfangen oder nach einem Punkt direkt eine Zahl als Namen wählen wie bei ".2VariablenName". Auf Umlaute sollte im Zusammenhang mit Programmiersprachen ebenfalls immer (!) verzichtet werden.

```
bigFive <- read_tsv("data.csv")
```

4. Der Beispieldatensatz

Im Datensatz wurden 19 719 Personen zu 54 Items befragt. Die ersten vier Spalten sind mit Alter, Geschlecht, Händigkeit und Herkunftsland soziodemographischer Natur.

- Geschlecht: 0 = missed, 1 = männlich, 2 = weiblich, 3 = other
- Händigkeit: 0 = missed, 1 = rechts, 2 = links, 3 = beides

Die restlichen 50 Spalten beinhaltet jeweils die 10 Items zu jedem Big Five Persönlichkeitsfaktor (Extraversion, Neuroticism, Agreeableness, Conscientiousness, Openness to experience) mit der Ausprägung von 1 (Disagree) bis 5 (Agree).

5. Data.frames indizieren

Unser Datensatz ist in einem so genannten data.frame gespeichert (siehe 12.2.3). In vielen Situationen kann es wichtig sein, auf bestimmte Spalten eines Datensatzes zuzugreifen. Deswegen seien hier zwei Methoden erklärt.

5.1 $-Operator

Der $-Operator greift auf eine Spalte über den Spaltennamen zu.

```
head(bigFive$country)
```

```
## [1] "US" "US" "PK" "RO" "US" "US"
```

5.2 Eckige Klammern

Mit den eckigen Klammern kann man auf eine oder mehrere Zeilen und Spalten über die jeweilige Nummer zugreifen. Dabei steht die Zahl links vom Komma für die Zeilennummer und rechts für die Spaltennummer. Lässt man eins der beiden Felder frei, werden jeweils alle Zeilen oder Spalten ausgewählt. Wenn man mehrere Zahlen auswählen mochte, verbindet man diese mit einem Doppelpunkt. Die Funktion c() erlaubt es verschiedene Zahlen zu kombinieren. Ein Minus vor der Zeilen oder Spaltenzahl entfernt diese.

```
bigFive[ ,1:4]
```

```
## # A tibble: 19,719 x 4
##      age gender  hand country
##    <int>  <int> <int>   <chr>
##  1    53      1     1      US
##  2    46      2     1      US
##  3    14      2     1      PK
##  4    19      2     1      RO
```

```
##  5    25      2     1      US
##  6    31      2     1      US
##  7    20      2     1      US
##  8    23      1     1      IN
##  9    39      2     3      US
## 10    18      2     1      US
## # ... with 19,709 more rows
```

```
bigFive[1:5, -4]
```

```
## # A tibble: 5 x 53
##     age gender  hand    E1    E2    E3    E4    E5    E6    E7    E8    E9
##   <int>  <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1    53      1     1     4     2     5     2     5     1     4     3     5
## 2    46      2     1     2     2     3     3     3     3     1     5     1
## 3    14      2     1     5     1     1     4     5     1     1     5     5
## 4    19      2     1     2     5     2     4     3     4     3     4     4
## 5    25      2     1     3     1     3     3     3     1     3     1     3
## # ... with 41 more variables: E10 <int>, N1 <int>, N2 <int>, N3 <int>,
## #   N4 <int>, N5 <int>, N6 <int>, N7 <int>, N8 <int>, N9 <int>, N10 <int>,
## #   A1 <int>, A2 <int>, A3 <int>, A4 <int>, A5 <int>, A6 <int>, A7 <int>,
## #   A8 <int>, A9 <int>, A10 <int>, C1 <int>, C2 <int>, C3 <int>, C4 <int>,
## #   C5 <int>, C6 <int>, C7 <int>, C8 <int>, C9 <int>, C10 <int>, O1 <int>,
## #   O2 <int>, O3 <int>, O4 <int>, O5 <int>, O6 <int>, O7 <int>, O8 <int>,
## #   O9 <int>, O10 <int>
```

```
bigFive[c(1, 4, 5), ]
```

```
## # A tibble: 3 x 54
##     age gender  hand country    E1    E2    E3    E4    E5    E6    E7
##   <int>  <int> <int>   <chr> <int> <int> <int> <int> <int> <int> <int>
## 1    53      1     1      US     4     2     5     2     5     1     4
## 2    19      2     1      RO     2     5     2     4     3     4     3
## 3    25      2     1      US     3     1     3     3     3     1     3
## # ... with 43 more variables: E8 <int>, E9 <int>, E10 <int>, N1 <int>,
## #   N2 <int>, N3 <int>, N4 <int>, N5 <int>, N6 <int>, N7 <int>, N8 <int>,
## #   N9 <int>, N10 <int>, A1 <int>, A2 <int>, A3 <int>, A4 <int>, A5 <int>,
## #   A6 <int>, A7 <int>, A8 <int>, A9 <int>, A10 <int>, C1 <int>, C2 <int>,
## #   C3 <int>, C4 <int>, C5 <int>, C6 <int>, C7 <int>, C8 <int>, C9 <int>,
## #   C10 <int>, O1 <int>, O2 <int>, O3 <int>, O4 <int>, O5 <int>, O6 <int>,
## #   O7 <int>, O8 <int>, O9 <int>, O10 <int>
```

5.3 grep

Wenn man einen großen Datensatz hat, möchte man beim Indizieren des Datensatzes über die eckigen Klammern bestimmt nicht jede einzelne Spalte zählen. Da schafft grep Abhilfe. Mit

names() kann man auf die Spaltennamen zugreifen. Die Spalte gender hat also die Nummer 2.

```
grep("gender", names(bigFive))
```

```
## [1] 2
```

5.4 head und print

head() gibt die ersten 6 Zeilen aller Spalten aus. Für einen ersten Überblick kann das sehr nützlich sein. Wenn man mehr oder weniger als 6 Zeilen sehen möchte, kann man hinter den Namen des Datensatzes die gewünschte Zeilenanzahl schreiben. Wenn man den gesamten Datensatz auf einmal sehen möchte, kann man entweder print() oder schlichtweg den Variablennamen eintippen. Der verständlichkeit halber, werden wir im Folgenden konsistent print() benutzen.

```
head(bigFive, 5)
print(bigFive)
bigFive
```

6. Deskriptive Statistik

6.1 Zusammenfassung und Dimensionen

summary() gibt von jeder Spalte das Minimum, Maximum, 1. Quartil, Mittelwert (2. Quartil), 3. Quartil und Median aus. dim() gibt die Zeilenanzahl und Spaltenanzahl aus.

```
summary(bigFive)
```

```
##       age                gender           hand         country
##  Min.   :       13   Min.   :0.000   Min.   :0.00   Length:19719
##  1st Qu.:       18   1st Qu.:1.000   1st Qu.:1.00   Class :character
##  Median :       22   Median :2.000   Median :1.00   Mode  :character
##  Mean   :    50767   Mean   :1.617   Mean   :1.13
##  3rd Qu.:       31   3rd Qu.:2.000   3rd Qu.:1.00
##  Max.   :999999999   Max.   :3.000   Max.   :3.00
##        E1              E2              E3              E4
##  Min.   :0.000   Min.   :0.00   Min.   :0.000   Min.   :0.000
##  1st Qu.:2.000   1st Qu.:2.00   1st Qu.:3.000   1st Qu.:2.000
##  Median :3.000   Median :3.00   Median :4.000   Median :3.000
##  Mean   :2.629   Mean   :2.76   Mean   :3.417   Mean   :3.152
##  3rd Qu.:4.000   3rd Qu.:4.00   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.00   Max.   :5.000   Max.   :5.000
##        E5              E6              E7              E8
```

```
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :4.000   Median :2.000   Median :3.000   Median :3.000
##  Mean   :3.432   Mean   :2.453   Mean   :2.867   Mean   :3.376
##  3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        E9              E10             N1              N2
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:2.000   1st Qu.:3.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :3.000   Median :4.000   Median :3.000   Median :3.000
##  Mean   :3.094   Mean   :3.585   Mean   :3.262   Mean   :3.235
##  3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        N3              N4              N5              N6
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.00
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.00
##  Median :4.000   Median :3.000   Median :3.000   Median :3.00
##  Mean   :3.843   Mean   :2.756   Mean   :2.952   Mean   :2.98
##  3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.00
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.00
##        N7              N8              N9             N10
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :3.000   Median :3.000   Median :3.000   Median :3.000
##  Mean   :3.152   Mean   :2.803   Mean   :3.135   Mean   :2.834
##  3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        A1              A2              A3              A4
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.00
##  1st Qu.:1.000   1st Qu.:3.000   1st Qu.:1.000   1st Qu.:4.00
##  Median :2.000   Median :4.000   Median :2.000   Median :4.00
##  Mean   :2.312   Mean   :3.927   Mean   :2.163   Mean   :4.03
##  3rd Qu.:3.000   3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:5.00
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.00
##        A5              A6              A7              A8
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:1.000   1st Qu.:3.000   1st Qu.:1.000   1st Qu.:3.000
##  Median :2.000   Median :4.000   Median :2.000   Median :4.000
##  Mean   :2.166   Mean   :3.896   Mean   :2.161   Mean   :3.766
##  3rd Qu.:3.000   3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:5.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        A9             A10              C1              C2
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:3.000   1st Qu.:3.000   1st Qu.:2.000
##  Median :4.000   Median :4.000   Median :3.000   Median :3.000
```

```
##  Mean   :3.945   Mean   :3.682   Mean   :3.318   Mean   :2.979
##  3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        C3              C4              C5              C6
##  Min.   :0.000   Min.   :0.000   Min.   :0.0   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:2.0   1st Qu.:2.000
##  Median :4.000   Median :3.000   Median :3.0   Median :3.000
##  Mean   :3.983   Mean   :2.654   Mean   :2.7   Mean   :2.923
##  3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:4.0   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.0   Max.   :5.000
##        C7              C8              C9              C10
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:3.000
##  Median :4.000   Median :2.000   Median :3.000   Median :4.000
##  Mean   :3.647   Mean   :2.481   Mean   :3.224   Mean   :3.637
##  3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        O1              O2              O3              O4
##  Min.   :0.000   Min.   :0.00   Min.   :0.000   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:1.00   1st Qu.:4.000   1st Qu.:1.000
##  Median :4.000   Median :2.00   Median :4.000   Median :2.000
##  Mean   :3.692   Mean   :2.15   Mean   :4.126   Mean   :2.079
##  3rd Qu.:5.000   3rd Qu.:3.00   3rd Qu.:5.000   3rd Qu.:3.000
##  Max.   :5.000   Max.   :5.00   Max.   :5.000   Max.   :5.000
##        O5              O6              O7              O8
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:1.000   1st Qu.:4.000   1st Qu.:2.000
##  Median :4.000   Median :1.000   Median :4.000   Median :3.000
##  Mean   :3.873   Mean   :1.795   Mean   :4.073   Mean   :3.208
##  3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.:5.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##        O9             O10
##  Min.   :0.000   Min.   :0.000
##  1st Qu.:4.000   1st Qu.:3.000
##  Median :4.000   Median :4.000
##  Mean   :4.134   Mean   :4.005
##  3rd Qu.:5.000   3rd Qu.:5.000
##  Max.   :5.000   Max.   :5.000
```

```
dim(bigFive)
```

```
## [1] 19719    54
```

6.2 Mittelwert, Median, Varianz, Standardabweichung, Summe

```
mean(bigFive$age)
```

```
## [1] 50767.03
```

```
median(bigFive$age)
```

```
## [1] 22
```

```
var(bigFive$age)
```

```
## [1] 5.071251e+13
```

```
sd(bigFive$age)
```

```
## [1] 7121272
```

```
sum(bigFive$age)
```

```
## [1] 1001075023
```

7. Datentypen

Wie man im Output von head() sieht gibt es verschiedene Datentypen, die in den Spalten gespeichert werden. Dabei ist es wichtig zu verstehen, dass in jeder Spalte nur ein Typ gespeichert werden kann. Die drei Hauptdatentypen, seien hier aufgelistet. In R werden sie etwas untypisch im Vergleich zu anderen Programmiersprachen benannt. In Klammern sei der übliche Begriff genannt.

- numeric (integer / double): z.B. age, gender usw.
- character (string): z.B. country
- logical (boolean): TRUE, FALSE

Bei logischen Abfragen ist es wichtig zu beachten, dass wenn man auf Gleichheit von zwei Werten prüft, nicht ein einfaches = sondern ein doppeltes == nötig ist. Man kann logische Aussagen auch kombinieren. Wenn beide logische Abfragen stimmen sollen, kann man ein & (UND) schreiben. Wenn nur einer Aussage richtig sein muss, damit der Wert TRUE zurückgegeben wird, muss man | (ODER) schreiben.

```
## Numeric
42
```

```
## [1] 42
```

```
## Character
"Hallo Welt"
```

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

```
## Logisch
1 < 2 & 1 == 2
```

```
## [1] FALSE
```

```
1 < 2 | 1 == 2
```

```
## [1] TRUE
```

8. Dplyr

Dplyr bietet verschiedene Funktionen, die vor allem die Datenvorbereitung erheblich erleichtern können. Die wichtigsten 6 seien hier gezeigt.

8.1 select

Gibt nur ausgewählte Spalten des Datensatzes aus. Das erste Argument ist der Name des Datensatzes.

```
select(bigFive, gender, age)
```

```
## # A tibble: 19,719 x 2
##    gender   age
##     <int> <int>
##  1      1    53
##  2      2    46
##  3      2    14
##  4      2    19
##  5      2    25
##  6      2    31
##  7      2    20
##  8      1    23
##  9      2    39
## 10      2    18
## # ... with 19,709 more rows
```

8.2 filter

Gibt nur jene Zeilen einer Spalte aus, die mit der gewählten Bedingung übereinstimmen.

```
filter(bigFive, gender == 1)
```

```
## # A tibble: 7,608 x 54
##      age gender  hand country    E1    E2    E3    E4    E5    E6    E7
##    <int>  <int> <int>   <chr> <int> <int> <int> <int> <int> <int> <int>
```

```
##  1    53      1     1      US     4     2     5     2     5     1     4
##  2    23      1     1      IN     4     3     5     3     5     1     4
##  3    15      1     1      IN     3     3     5     3     3     3     2
##  4    21      1     1      US     2     3     2     3     3     1     1
##  5    21      1     2      CA     3     3     4     3     4     3     3
##  6    48      1     1      US     5     5     4     1     5     1     5
##  7    22      1     1      US     3     3     4     3     2     2     3
##  8    37      1     1      DK     3     3     3     3     2     2     4
##  9    25      1     2      US     4     2     4     2     4     2     4
## 10    15      1     1      GB     1     5     2     5     2     2     2
## # ... with 7,598 more rows, and 43 more variables: E8 <int>, E9 <int>,
## #   E10 <int>, N1 <int>, N2 <int>, N3 <int>, N4 <int>, N5 <int>, N6 <int>,
## #   N7 <int>, N8 <int>, N9 <int>, N10 <int>, A1 <int>, A2 <int>, A3 <int>,
## #   A4 <int>, A5 <int>, A6 <int>, A7 <int>, A8 <int>, A9 <int>, A10 <int>,
## #   C1 <int>, C2 <int>, C3 <int>, C4 <int>, C5 <int>, C6 <int>, C7 <int>,
## #   C8 <int>, C9 <int>, C10 <int>, O1 <int>, O2 <int>, O3 <int>, O4 <int>,
## #   O5 <int>, O6 <int>, O7 <int>, O8 <int>, O9 <int>, O10 <int>
```

8.3 mutate

Erstellt eine neue Spalte. rowMeans berechnet die Mittelwerte pro Zeile.

```
mutate(bigFive, Extraversion = rowMeans(bigFive[ ,5:14]))
```

```
## # A tibble: 19,719 x 55
##      age gender  hand country    E1    E2    E3    E4    E5    E6    E7
##    <int>  <int> <int>   <chr> <int> <int> <int> <int> <int> <int> <int>
##  1    53      1     1      US     4     2     5     2     5     1     4
##  2    46      2     1      US     2     2     3     3     3     3     1
##  3    14      2     1      PK     5     1     1     4     5     1     1
##  4    19      2     1      RO     2     5     2     4     3     4     3
##  5    25      2     1      US     3     1     3     3     3     1     3
##  6    31      2     1      US     1     5     2     4     1     3     2
##  7    20      2     1      US     5     1     5     1     5     1     5
##  8    23      1     1      IN     4     3     5     3     5     1     4
##  9    39      2     3      US     3     1     5     1     5     1     5
## 10    18      2     1      US     1     4     2     5     2     4     1
## # ... with 19,709 more rows, and 44 more variables: E8 <int>, E9 <int>,
## #   E10 <int>, N1 <int>, N2 <int>, N3 <int>, N4 <int>, N5 <int>, N6 <int>,
## #   N7 <int>, N8 <int>, N9 <int>, N10 <int>, A1 <int>, A2 <int>, A3 <int>,
## #   A4 <int>, A5 <int>, A6 <int>, A7 <int>, A8 <int>, A9 <int>, A10 <int>,
## #   C1 <int>, C2 <int>, C3 <int>, C4 <int>, C5 <int>, C6 <int>, C7 <int>,
## #   C8 <int>, C9 <int>, C10 <int>, O1 <int>, O2 <int>, O3 <int>, O4 <int>,
## #   O5 <int>, O6 <int>, O7 <int>, O8 <int>, O9 <int>, O10 <int>,
## #   Extraversion <dbl>
```

8.4 summarise und group_by

Summarise kann verschiedene grundlegende statistische Verfahren auf mehrere Spalten anwenden. group_by gruppiert die Daten nach einer Schlüsselvariable.

```
summarise(group_by(bigFive, gender), mean = mean(E1), sd = sd(E1))
```

```
## # A tibble: 4 x 3
##   gender     mean       sd
##    <int>    <dbl>    <dbl>
## 1      0 2.458333 1.350657
## 2      1 2.617114 1.243765
## 3      2 2.640884 1.224553
## 4      3 2.147059 1.213758
```

8.5 arrange

Sortiert die Daten nach einer Schlüsselvariable.

```
arrange(bigFive, age)
```

```
## # A tibble: 19,719 x 54
##      age gender  hand country    E1    E2    E3    E4    E5    E6    E7
##    <int>  <int> <int>   <chr> <int> <int> <int> <int> <int> <int> <int>
##  1    13      1     1      GB     3     4     3     3     3     5     2
##  2    13      2     3      US     5     1     5     5     5     1     5
##  3    13      2     1      US     4     2     5     2     4     1     5
##  4    13      2     1      US     3     1     4     2     4     1     4
##  5    13      1     1      US     1     4     3     5     3     5     1
##  6    13      2     3      SG     1     4     2     3     4     2     1
##  7    13      1     2      GB     1     5     3     5     1     5     1
##  8    13      1     1      US     3     2     3     2     4     2     5
##  9    13      2     1      US     2     3     4     2     3     1     2
## 10    13      1     1      US     2     4     2     4     2     5     1
## # ... with 19,709 more rows, and 43 more variables: E8 <int>, E9 <int>,
## #   E10 <int>, N1 <int>, N2 <int>, N3 <int>, N4 <int>, N5 <int>, N6 <int>,
## #   N7 <int>, N8 <int>, N9 <int>, N10 <int>, A1 <int>, A2 <int>, A3 <int>,
## #   A4 <int>, A5 <int>, A6 <int>, A7 <int>, A8 <int>, A9 <int>, A10 <int>,
## #   C1 <int>, C2 <int>, C3 <int>, C4 <int>, C5 <int>, C6 <int>, C7 <int>,
## #   C8 <int>, C9 <int>, C10 <int>, O1 <int>, O2 <int>, O3 <int>, O4 <int>,
## #   O5 <int>, O6 <int>, O7 <int>, O8 <int>, O9 <int>, O10 <int>
```

8.6 Der Pipe Operator

Der größte Vorteil von Dplyr ist jedoch der %>% (Pipe) Operator. Damit können sämtliche Dplyr Funktionen in einem Befehl zusammengebunden werden. Wenn man also als Output nur die Spalten age, gender und die mittlere Ausprägung der Extraversion männlicher Probanden in aufsteigender Ordnung sehen möchte:

```
bigFive %>%
  arrange(age) %>%
  filter(gender == 1) %>%
  mutate(Extraversion = rowMeans(.[ ,5:14])) %>%
  select(age, gender, Extraversion)
```

```
## # A tibble: 7,608 x 3
##      age gender Extraversion
##    <int>  <int>        <dbl>
##  1    13      1          3.5
##  2    13      1          3.4
##  3    13      1          3.2
##  4    13      1          3.4
##  5    13      1          3.1
##  6    13      1          3.1
##  7    13      1          2.8
##  8    13      1          3.4
##  9    13      1          3.2
## 10    13      1          3.3
## # ... with 7,598 more rows
```

Wie man sieht kann man den Datennamen jeweils weglassen. Der Punkt vor [,5:14] steht für den Namen des Datensatzes. Damit man weitergehend mit den Daten arbeiten kann, muss man sie wie immer in einer Variable speichern. In diesem Fall fügen wir die mittlere Extraversion- und Neurotizismusausprägung pro Person unserem Datensatz hinzu.

```
bigFive <- bigFive %>%
  mutate(Extraversion = rowMeans(.[ ,5:14]),
         Neurotizismus = rowMeans(.[ ,15:24]))
```

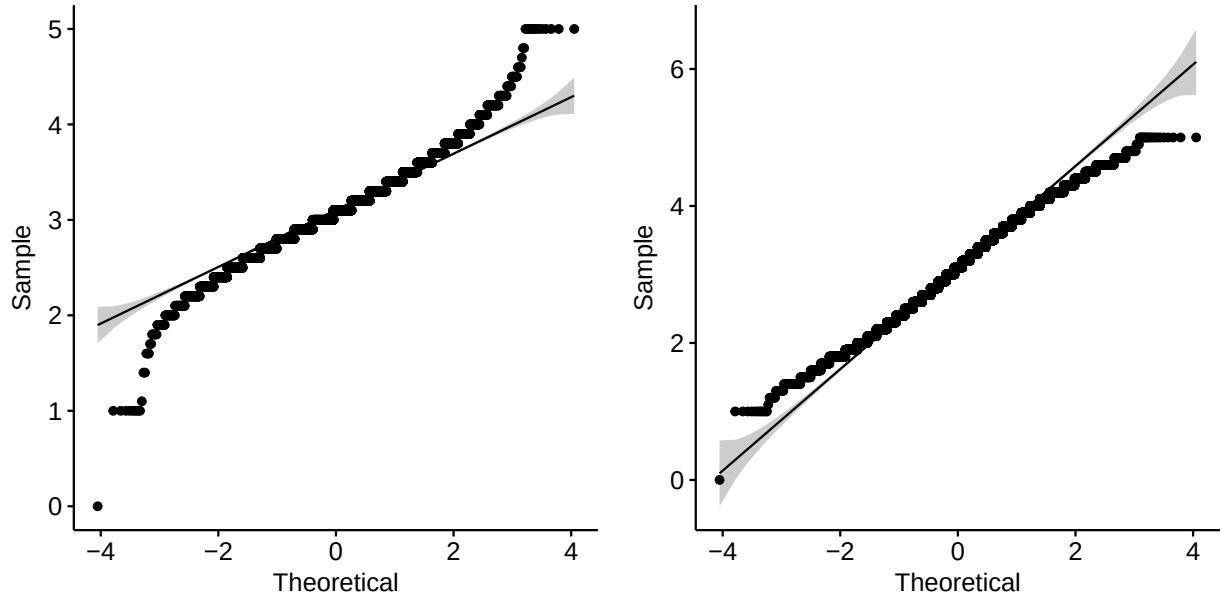
9. Induktive Statistik

9.1 Q-Q Plot

Zur Erstellung der Q-Q Plots, die man zum Testen der Annahme über Normalverteilung benötigt, verwenden wir das Package ggpubr (siehe 8.).

```
library(ggpubr)
```

```
ggqqplot(bigFive, "Extraversion")
ggqqplot(bigFive, "Neurotizismus")
```



9.2 Bonferroni Korrektur

Wenn man mehrere verschiedene statistische Tests anwendet, muss man das gewählte alpha Niveau (Fehler 1. Art) anpassen. Bei der Bonferroni Korrektur muss man alpha durch die Anzahl der Verfahren dividieren (hier als Beispiel 4). Der Einfacheit halber belassen wir das alpha Niveau bei den folgenden Tests jedoch bei 5%.

```
alpha <- .05
alpha / 4
```

```
## [1] 0.0125
```

9.3 F-Test, t-Test, Welch-Test

Zur Testung auf Varianzgleichheit wird der F-test mit der Funktion var.test() aufgerufen. Wenn der F-Test ein signifikantes Ergebnis ausgibt, muss der Welch-test angewendet werden. Ansonsten kann man aufgrund der Varianzgleichheit Students t-Test verwenden. Der Unterschied zwischen Welch- und t-test in R ist lediglich das Argument var.equal, was man je nach Test auf TRUE oder FALSE setzen muss. Falls die Variablen abhängig sind, muss das paired Argument auf TRUE gesetzt werden. Zur einseitigen Testung muss man das Argument alternative auf “less” oder “greater” setzen.

```
## F-Test
var.test(bigFive$Extraversion, bigFive$Neurotizismus,
         alternative = "two.sided", conf.level = .95)
```

```
##
##  F test to compare two variances
##
## data:  bigFive$Extraversion and bigFive$Neurotizismus
## F = 0.27505, num df = 19718, denom df = 19718, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.2674814 0.2828404
## sample estimates:
## ratio of variances
##          0.2750537
```

```
## t-Test
t.test(bigFive$Extraversion, bigFive$Neurotizismus,
       alternative = "two.sided", conf.level = .95, paired = F, var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  bigFive$Extraversion and bigFive$Neurotizismus
## t = -3.4489, df = 39436, p-value = 0.0005634
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.029212225 -0.008041186
## sample estimates:
## mean of x mean of y
##  3.076566  3.095192
```

```
## Welch-Test
t.test(bigFive$Extraversion, bigFive$Neurotizismus,
       alternative = "two.sided", conf.level = .95, paired = F, var.equal = F)
```

```
##
##  Welch Two Sample t-test
##
## data:  bigFive$Extraversion and bigFive$Neurotizismus
## t = -3.4489, df = 29802, p-value = 0.0005636
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.029212330 -0.008041081
## sample estimates:
## mean of x mean of y
##  3.076566  3.095192
```

9.4 Regressionsanalysen

Die verschiedenen Regressions- und Varianzanalysen sind syntaktisch nicht kompliziert. Auf der linken Seite der Formel steht immer die Response Größe und auf der rechten jeweils der oder die Einflüsse, die in Wechselwirkung zueinander stehen können. Die Outputs in Base R sind unübersichtlich und können nicht direkt weiterverarbeitet werden. Unter 9.9 wird eine Lösung dafür angeboten. Pr(>|t|) ist der p-Wert. Bei SPSS wird unter .001 nur noch <.001 angezeigt. Bei R wird bis 2e-16, also 2 x 10^-16 (0.0000000000000002) differenziert.

```
## Einfache lineare
reg <- lm(Extraversion ~ Neurotizismus, data = bigFive)
summary(reg)
```

```
##
## Call:
## lm(formula = Extraversion ~ Neurotizismus, data = bigFive)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0775 -0.1768  0.0232  0.2233  1.9240
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.077485   0.011829  260.15   <2e-16 ***
## Neurotizismus -0.000297   0.003735   -0.08    0.937
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3522 on 19717 degrees of freedom
## Multiple R-squared:  3.207e-07,  Adjusted R-squared:  -5.04e-05
## F-statistic: 0.006323 on 1 and 19717 DF,  p-value: 0.9366
```

```
## Multiple lineare
summary(lm(gender ~ Extraversion + Neurotizismus, data = bigFive))
```

```
##
## Call:
## lm(formula = gender ~ Extraversion + Neurotizismus, data = bigFive)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7555 -0.5702  0.2989  0.3956  1.6113
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.619763   0.034815   46.52   <2e-16 ***
```

```
## Extraversion  -0.108990   0.009955  -10.95   <2e-16 ***
## Neurotizismus  0.107414   0.005221   20.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4924 on 19716 degrees of freedom
## Multiple R-squared:  0.02682,    Adjusted R-squared:  0.02672
## F-statistic: 271.7 on 2 and 19716 DF,  p-value: < 2.2e-16
```

```
## Logistische
summary(glm(Extraversion ~ Neurotizismus, data = bigFive))
```

```
##
## Call:
## glm(formula = Extraversion ~ Neurotizismus, data = bigFive)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.0775  -0.1768   0.0232   0.2233   1.9240
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.077485   0.011829  260.15   <2e-16 ***
## Neurotizismus -0.000297   0.003735   -0.08    0.937
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1240787)
##
##     Null deviance: 2446.5  on 19718  degrees of freedom
## Residual deviance: 2446.5  on 19717  degrees of freedom
## AIC: 14814
##
## Number of Fisher Scoring iterations: 2
```

9.5 Varianzanalysen (ANOVA)

Die Anpassung des p-Wertes wird vor allem bei ANOVA Verfahren gerne vergessen. Da man i.d.R verschiedene Hypothesen auf einmal testet, ist dies allerdings unabdingbar (siehe 9.2).

```
## Einfaktoriell
anova(reg)
```

```
## Analysis of Variance Table
##
## Response: Extraversion
```

```
##                   Df Sum Sq  Mean Sq F value Pr(>F)
## Neurotizismus      1    0.0 0.000785  0.0063 0.9366
## Residuals      19717 2446.5 0.124079
```

```
## Zweifaktoriell mit Wechselwirkung
anova(lm(gender ~ Neurotizismus * Extraversion, data = bigFive))
```

```
## Analysis of Variance Table
##
## Response: gender
##                                Df Sum Sq Mean Sq  F value Pr(>F)
## Neurotizismus                   1  102.7 102.685 423.4853 <2e-16 ***
## Extraversion                    1   29.1  29.061 119.8501 <2e-16 ***
## Neurotizismus:Extraversion      1    0.0   0.030   0.1228 0.7261
## Residuals                   19715 4780.4   0.242
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Zweifaktoriell ohne Wechselwirkung
anova(lm(gender ~ Neurotizismus + Extraversion, data = bigFive))
```

```
## Analysis of Variance Table
##
## Response: gender
##                   Df Sum Sq Mean Sq F value    Pr(>F)
## Neurotizismus      1  102.7 102.685  423.50 < 2.2e-16 ***
## Extraversion       1   29.1  29.061  119.86 < 2.2e-16 ***
## Residuals      19716 4780.5   0.242
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.6 Korrelationskoeffizienten

Für den Korrelationskoeffizienten nach Pearson, Spearman oder Kendall, muss man lediglich das method Argument entsprechend verändern.

```
cor.test(bigFive$Extraversion, bigFive$Neurotizismus,
         method = "pearson", alternative = "two.sided", conf.level = .95)
```

```
##
##  Pearson's product-moment correlation
##
## data:  bigFive$Extraversion and bigFive$Neurotizismus
## t = -0.079516, df = 19717, p-value = 0.9366
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.01452376  0.01339142
```

```
## sample estimates:
##           cor
## -0.0005662811
```

9.7 Kontingenztafeln

Mithilfe von table() kann man schnell Kontingenztafeln erstellen. Als Beispiel sei hier dargestellt, wie viele Männer und Frauen, bei der mittleren Extraversion einen Wert von > 3.5 haben. Dabei greifen wir mit dem $-Operator auf die jeweilige Spalte zu. Anschließend testen wir mit dem Fisher- und Mcnemar-Test auf Signifikanz der Zeilen 2 und 3 also männlichen und weiblichen Personen.

```
tbl <- table(bigFive$gender, bigFive$Extraversion > 3.5)
print(tbl)
```

```
##
##     FALSE  TRUE
##   0    23     1
##   1  6848   760
##   2 11105   880
##   3    97     5
```

```
fisher.test(tbl[c(2, 3), ])
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tbl[c(2, 3), ]
## p-value = 1.172e-10
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.6442448 0.7915376
## sample estimates:
## odds ratio
##  0.7140585
```

```
mcnemar.test(tbl[c(2, 3), ])
```

```
##
##  McNemar's Chi-squared test with continuity correction
##
## data:  tbl[c(2, 3), ]
## McNemar's chi-squared = 9018, df = 1, p-value < 2.2e-16
```

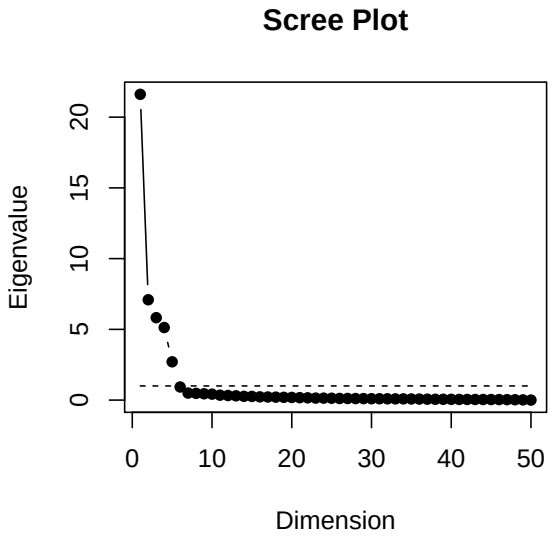
9.8 Exploratorische Faktorenanalyse

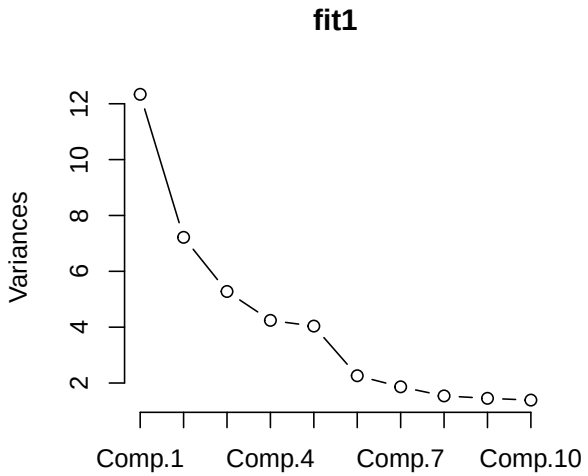
Nun möchten wir schauen, ob die 50 Fragen zu den verschiedenen Persönlichkeitseigenschaften auch tatsächlich durch 5 Faktoren dargestellt werden können. Dafür schauen wir uns zuerst mit der Principal Component Analysis (PCA) eine Methode der reinen Dimensionsreduktion an. Die pca ist zwar keine richtige Faktorenanalyse, wird aber von SPSS fälschlicher Weise als Standard verwendet. Dabei wird eine naive PCA gewählt, bei der man keine Vorannahme über die Anzahl der vorhanden Faktoren machen muss. Anschließend vergleichen wir das Ergebnis mit der Maximum Likelihood Faktor Analysis. Die Outputs seien hier aus Gründen der Übersichtlichkeit ausgelassen, da die folgenden Befehle mehrere Seiten füllen würden. Nur die jeweiligen Screeplots, die Aufschluss über die Faktorenzahl geben, werden gezeigt.

```
## Principal Component Analysis
fit1 <- princomp(bigFive[5:54])
summary(fit1)
loadings(fit1)

## Maximum Likelihood Factor Analysis
fit2 <- factanal(bigFive[5:54], factors = 5, rotation = "varimax")
print(fit2, digits = 2, cutoff = .3, sort = T)
fit2$loadings

## Graphische Überprüfung
library(psy)
scree.plot(fit2$correlation)
screeplot(fit1, type = "lines")
```





9.9 Broom Package

Das Package Broom konvertiert die Ergebnisse verschiedener statistischer Tests mit tidy() aus Base R zu übersichtlicheren Outputs, die man weiterverarbeiten kann. Mit glance() erhält man zusätzliche Informationen wie R Quadrat oder die logistische Likelihood.

```
library(broom)
tidy(reg)
```

```
##            term      estimate  std.error    statistic   p.value
## 1   (Intercept)  3.0774849879 0.01182950 260.15341579 0.0000000
## 2 Neurotizismus -0.0002969893 0.00373498  -0.07951564 0.9366233
```

```
glance(reg)
```

```
##      r.squared adj.r.squared     sigma   statistic   p.value df   logLik
## 1 3.206743e-07 -5.039696e-05 0.3522481 0.006322738 0.9366233  2 -7403.86
##        AIC      BIC deviance df.residual
## 1 14813.72 14837.39  2446.46       19717
```

10. tidyr

Tidyr bietet Funktionen zur Umwandlung von Datensätzen, die nicht tidy sind. Die Funktion gather() kann aber auch für die Vorbereitung des Datensatzes zum Zeichnen eines Graphen praktisch sein. Dabei nimmt die Funktion mehrere Spalten und fasst sie unter Berücksichtigung einer Key-Value zusammen. Sie macht also "wide Data longer". Da wir zum Zeichnen Charakter und nicht Numerics brauchen, ersetzen wir zusätzlich mit replace() alle einsen in der Spalte des Geschlechts mit einem m und alle zweien mit einem f. Zum Schluss muss man den tibble (siehe 12.2.3) in einen data.frame umwandeln (siehe 12.2.5), damit man den Graphen zeichnen kann.

```
df <- bigFive %>%
  group_by(gender) %>%
  summarise(Extraversion = mean(Extraversion),
            Neurotizismus = mean(Neurotizismus)) %>%
  filter(gender == 1 | gender == 2) %>%
  gather(BIG5, Mittelwert, Extraversion:Neurotizismus) %>%
  mutate(gender = replace(gender, gender == 1, "m"),
         gender = replace(gender, gender == 2, "f")) %>%
  as.data.frame()
print(df)
```

```
##   gender          BIG5 Mittelwert
## 1      m  Extraversion   3.110620
## 2      f  Extraversion   3.055912
## 3      m Neurotizismus   2.974040
```

```
## 4      f Neurotizismus   3.171089
```

11. ggplot2 und ggpubr

Eins der populärsten und umfangreichsten Packages zum Erstellen von Graphen in R ist ggplot2. ggpubr basiert auf ggplot2 mit dem Unterschied, dass die Syntax weiter vereinfacht wurde und ein Fokus darauf gelegt wird, möglichst einfach und schnell publikationsreife Graphen zu erstellen.

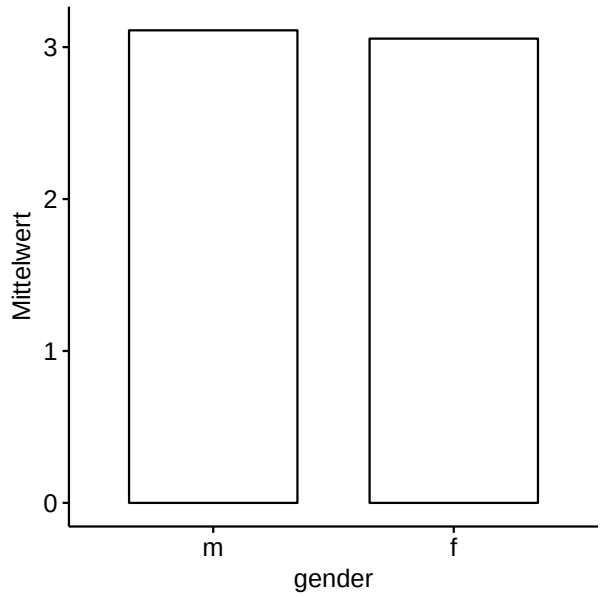
11.1 Säulendiagramm

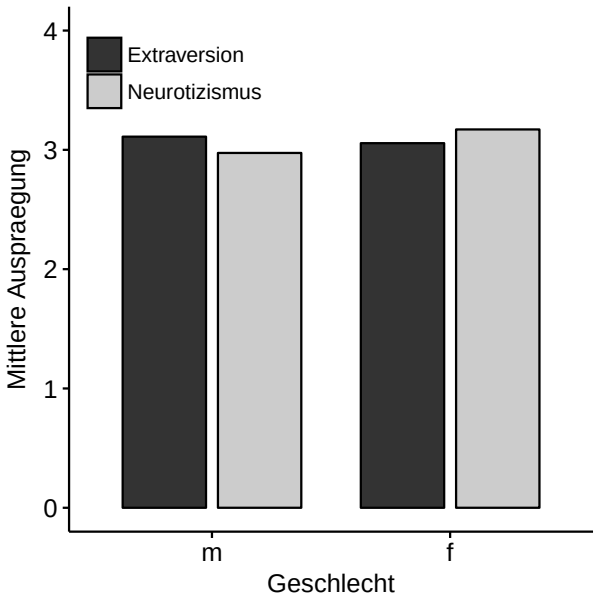
Im folgenden sei ein Beispiel zur Erstellung eines einfach Säulendiagramms sowie eines mit nebeneinander liegenden Säulen gezeigt. Wir greifen dabei auf den zuvor mit gather erstellten Datensatz **df** zu. Mithilfe von ggpar kann man einfach verschiedenste Modifikationen vornehmen. Für weitere Informationen über Anpassungsmöglichkeiten von ggpubr siehe hier.

```
df1 <- df[c(1, 2), ]

a <- ggbarplot(data = df1,
               x = "gender",
               y = "Mittelwert")
print(a)

b <- ggbarplot(data = df,
               x = "gender",
               y = "Mittelwert",
               fill = "BIG5",
               position = position_dodge(0.8)) +
               scale_fill_grey() +
               theme(legend.position = c(.2,.9),
                    axis.title.x = element_text(size = 12),
                    axis.title.y = element_text(size = 12))
b <- ggpar(p = b,
           xlab = "Geschlecht",
           ylab = "Mittlere Auspraegung",
           ylim = c(0, 4),
           legend.title = "")
print(b)
```

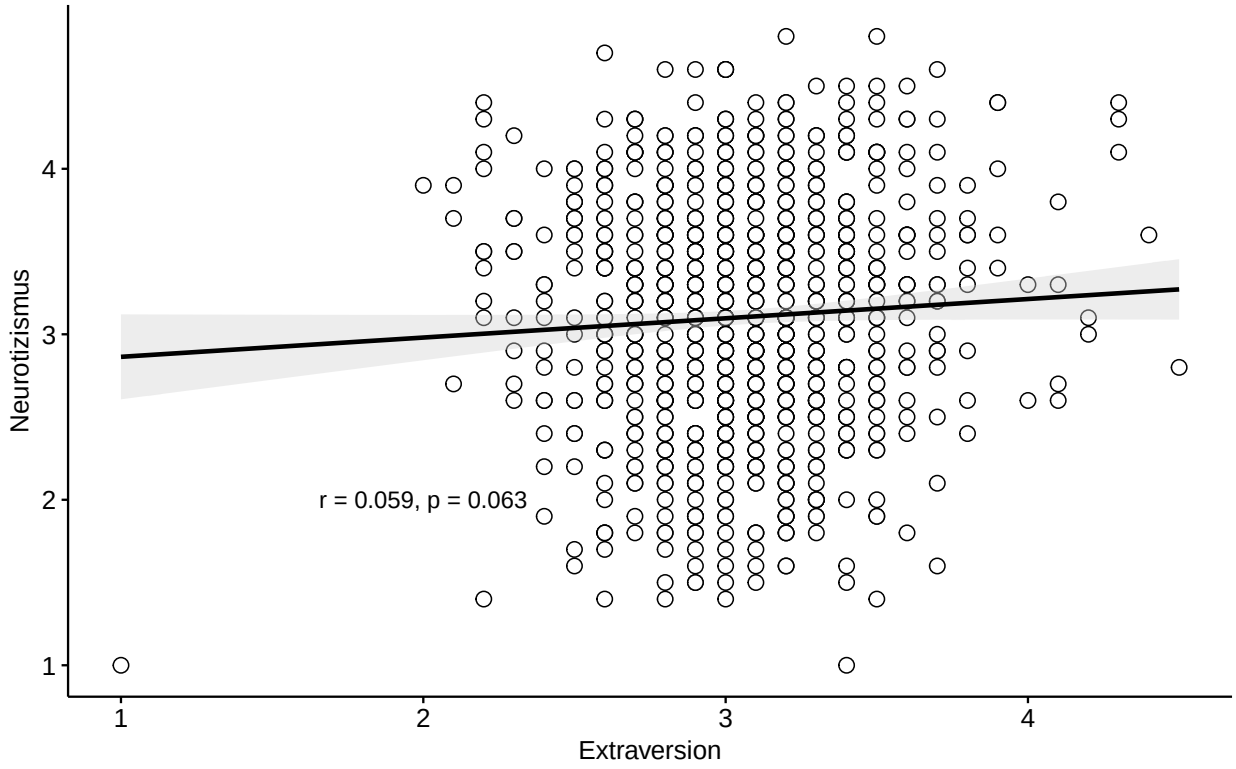




11.2 Scatterplot

Nun stellen wir den Zusammenhang von Extraversion und Neurotizismus der ersten 1000 Personen mit einem Scatterplot dar. Man kann auf einfache Art und Weise zusätzliche Informationen wie eine Regressiongerade, das Konfidenzintervall oder den Korrelationskoeffezienten hinzufügen.

```
df1 <- as.data.frame(bigFive[1:1000, ])
c <- ggscatter(data = df1,
               x = "Extraversion",
               y = "Neurotizismus",
               cor.coef = T,
               cor.coef.coord = c(2, 2),
               cor.method = "pearson",
               conf.int = T,
               add = "reg.line",
               add.params =  list(color = "black",
                                  fill = "lightgray"),
               size = 3,
               shape = 21,
               color = "black") + theme(axis.title.x = element_text(size = 12),
                                        axis.title.y = element_text(size = 12))
print(c)
```



11.3 Speichern

Mit ggsave kann man den Graphen einfach im derzeitigen Working Directory speichern.

```
ggsave(filename = "plotName.jpeg",
       plot = plotName,
       device = "jpeg",
       width = 5,
       height = 5)
```

12. Zusatz

12.1 Wo bekomme ich Hilfe?

Das schöne an Programmiersprachen ist, dass i.d.R irgendjemand schon einmal genau das selbe Problem gehabt hat, wie man selbst. Wenn man nach einer Fehlermeldung oder spezifischen Frage im Internet sucht, ist meist der erste Vorschlag das Forum Stackoverflow. Sollte man da nicht weiterkommen, gibt es für jedes Package und jede Funktion eine Dokumentation auf CRAN. Diese kann man entweder mit F1 bei öffnen der Funktion, per ?funktionsName oder über die Pdf Dokumentationen im Internet aufrufen.

12.2 Verschiedene Datenstrukturen

Bisher bestand unser Datensatz immer aus einem sogenannten data.frame bzw. einem tibble. Es gibt allerdings noch andere Formate, in denen Daten gespeichert werden.

12.2.1 Vektor

Ein Vektor ist eine eindimensionale Datenstruktur, die nur einen Dateitypen enthalten darf. Also entweder Numeric, Character oder Logical. Einen Vektor kann man mit c() erstellen. Aber auch die einzelnen Spalten in unserem Datensatz sind Vektoren. Der Zugriff auf Vektoren erfolgt mit eckigen Klammern. Man kann ebenfalls mit rep() und seq() einen Vektor erzeugen. rep() wiederholt eine Zeichenfrequenz gewünscht oft und seq() verhält sich wie der Doppelpunkt mit dem Unterschied, dass man die Abstände zwischen den Zahlen anpassen kann. Wenn man aus einem Data.frame eine Spalte mit dem $-Operator auswählt, ist auch dieser ein Vektor.

```
vektor <- c(1, 2, 3, 4)
bigFive$age
rep(c(1, 2), 2)
seq(from = 1, to = 4, by = .5)
1:4

## Zugriff auf das erste Element des Vektors
vektor[1]
```

12.2.2 Matrix

Eine Matrix bietet gegenüber dem Vektor den Vorteil der Multidimensionalität. Es kann mehrere Vektoren eines Datentyps enthalten. Eine Matrix kann man entweder mit matrix() oder mit cbind / rbind erstellen. cbind bindet Spalten und rbind bindet Zeilen zusammen. Dabei müssen alle Vektoren die selbe Länge haben. Der Zugriff erfolgt wie beim data.frame / tibble mit eckigen Klammern mit durch ein Komma getrennter Zeilen - und Spaltenzahl. Dabei kann man mehrere Zeilen und Spalten auswählen.

```
matrix <- matrix(c(1:8), ncol = 2)
rbind(c(1:4), c(1:4))
cbind(c("a", "b", "c"), c("a", "b", "c"))

## Zugriff auf die 2. Spalte von allen Beobachtungen
matrix[ ,2]
```

12.2.3 Data.frame / tibble

Ein data.frame kann in verschiedenen Spalten innerhalb des selben data.frames verschiedene Datentypen enthalten. Allerdings muss der Datentyp innerhalb einer Spalte die selbe sein. Wenn man mit einer tidyverse Funktionen einen Datensatz einliest, speichert der diesen jedoch nicht in einem üblichen data.frame, sondern in einem tibble. Die genauen Unterschiede und Vorteile von tibbles können hier nachgelesen werden.

```
## normaler Data.frame
data.frame(a = 1:4, b = c("a", "b", "c", "d"))

## tibble
data_frame(a = 1:4, b = c("a", "b", "c", "d"))
```

12.2.4 List

In einer Liste kann man alles verstauen. Dabei hat man verschiedene Listenelemente auf die man mit doppelten eckigen Klammern zugreifen kann. Wenn sich innerhalb des Listenelements ein Vektor befindet, kann man dann mit den gewohnten einfachen eckigen Klammern auf die gewünschten Elemente zugreifen.

```
list <- list(vektor, matrix)

## Zugriff auf das erste Element der Liste und zweite des Vektors
list[[1]][2]
```

12.2.5 Umwandlungen von Datenstrukturen

Datenstrukturen können grundsätzlich nur in einander umgewandelt werden, wenn sämtliche notwendigen Eigenschaften gegeben sind. Ein tibble, welcher in 2 Spalten verschiedene Datentypen beinhaltet, kann nicht (!) zu einer Matrix konvertiert werden.

```
as.vector()
as.matrix()
as.data.frame()
as.list()
```

12.3 Umgang mit fehlenden Werten

Fehlende Werte oder auch NAs können zu Fehlermeldungen führen. Viele Funktionen haben das logische Argument na.rm, welches man zur Berechnung schlichtweg auf TRUE setzen muss. Wenn man sämtliche NAs entfernen möchte, kann man dies mit na.omit machen. Allerdings verwirft R dann die gesamte Zeile, in der NA vorkommt (je nach Vollständigkeit des Datensatzes lässt dieser Befehl den Umfang extrem schrumpfen).

```r
df_na <- data_frame(Hallo = c(1:5, NA), Welt = c(NA, 1:5))

mean(df_na, na.rm = T)
na.omit(df_na)
```

12.4 suppressPackageStartupMessage

Da es nicht selten vorkommt, dass beim Laden eines Packages mehrere Hinweise in der Konsole angezeigt werden, kann man mithilfe von suppressPackageStartupMessage diese unterdrücken. Dies ist optional und dient nur einem aufgeräumten Output.

```r
suppressPackageStartupMessages(library(tidyverse))
```

12.5 Publikationsreife Tabellen mit LaTeX

Wenn man beispielsweise tibbles oder data.frames in publikationsreife Tabellen in LaTeX Code umwandeln möchte, kann man das einfach mit dem Package xtable.

```r
library(xtable)
xtable(data_frame(a = 1:3, b = 4:6))
```

```
## % latex table generated in R 3.4.2 by xtable 1.8-2 package
## % Sun Oct 22 11:31:30 2017
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrr}
##   \hline
##  & a & b \\
##   \hline
## 1 &   1 &   4 \\
##   2 &   2 &   5 \\
##   3 &   3 &   6 \\
##    \hline
## \end{tabular}
## \end{table}
```

12.6 Microsoft R Open (MRO)

Ist eine von Microsoft entwickelte kostenlose Programmiersprache, die auf R basiert. Der große Vorteil ist die Möglichkeit durch die automatische Nutzung der Intel Math Kernel Library sämtliche zur Verfügung stehende CPU Kerne zu benutzen (parallel Computing), was die Berechnungszeit erheblich reduzieren kann. MRO ist mit sämtlichen Packages aus R vollständig kompatibel und kann hier heruntergeladen werden.

12.7 Functional Programming

Bei fortgeschrittenen Verfahren reicht es häufig nicht mehr aus lediglich die gegeben Funktionen von Base R oder Packages zu verwenden. Es gibt verschiedene Werkzeuge, die es einem ermöglichen komplexere Aufgaben zu lösen.

12.7.1 Funktionen erstellen

Manchmal bietet es sich an, eigene Funktionen zu schreiben. Dabei schreibt man das Argument, was der Funktion weitergegeben werden soll in die Klammern von function(). Erst einmal gespeichert, kann man die Funktion beliebig oft anwenden. Sie verhält sich dabei genau wie sämtliche anderen Funktionen in R, wie z.B. mean(x). Wenn man unserer Funktion eine Zahl gibt, wird die selbe Zahl hinzu addiert.

```
fun <- function(x){
  x + x
}
fun(1:5)
```

```
## [1]  2  4  6  8 10
```

12.7.2 purrr

Mit purrr werden eine Reihe an Funktionen geliefert, die es einem Erlauben Funktionen wiederholt anzuwenden. Dabei seien hier 3 verschiedene Varianten der map Funktion gezeigt. Die normale map Funktion gibt eine Liste zurück. In unserem Beispiel ist jedes Element ein Ergebnis unserer Funktion mit den Zahlen von 1 bis 5. map_int gibt einen Vektor zurück. Wenn die Ergebnisse der angewendeten Funktion jeweils data.frames sind, bindet map_df diese zu einem tibble zusammen.

```
map(1:5, fun)
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 8
##
## [[5]]
```

```
## [1] 10
```

```
map_int(1:5, fun)
```

```
## [1]  2  4  6  8 10
```

```
map_df(1:5, as.data.frame(fun))
```

```
##   value
## 1     2
## 2     4
## 3     6
## 4     8
## 5    10
```

12.7.3 for-Schleife

In for-Schleifen geht man eine bestimmte Anzahl an Iterationen (Wiederholung) der Reihe nach durch und führt für jedes i das aus, was in der geschweiften Klammer steht. In diesem Beispiel addiert sich für jedes i von 1 bis 5 das selbe i hinzu. Sie macht also exakt das selbe wie map mit unserer selbst geschriebenen Funktion.

```
for (i in 1:5) {
  print(i + i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

12.7.4 if...else und ifelse Verzweigung

Manchmal möchte man bestimmte Berechnungen nur unter den gewünschten Bedingungen ausführen. Dafür hat man in R zwei Möglichkeiten. Der Hauptunterschied ist der, dass die ifelse Funktion in bitcode compiliert ist und deswegen bei vektoriellen Berechnungen teils erheblich schneller ist, als die übliche if-Verzweigung in R. In der Klammer der if-Verzweigung wird geprüft, ob die Bedingung wahr ist. Wenn der Wert wahr ist, wird der Befehl innerhalb der geschweiften Klammern von if ausgeführt - wenn dieser falsch ist, der Befehl innerhalb von else.

```
if (4 < 5) {
  "wahr"
} else {
  "falsch"
}
```

```
## [1] "wahr"
```

```
ifelse(6 < 5, "wahr", "falsch")
```

```
## [1] "falsch"
```

13. Übungen

Zuerst installieren wir mit dem Befehl alle notwendigen Packages auf Deinem Computer. Dies kann einige Minuten in Anspruch nehmen.

```
install.packages(c("tidyverse", "readxl", "writexl", "haven",
                   "broom", "ggpubr", "psy", "purrr",
                   "devtools", "learnr", "tidyr"), dependencies = T)
```

Die jeweiligen Packages solltest Du mithilfe des Abschnittes 2.1 selbst laden können. Um auf den Datensatz und das interaktive Tutorial (s.u.) zugreifen zu können, musst du ein weiteres Package installieren. Allerdings ist das diesmal nicht auf CRAN sondern Github erhältlich. Der Befehl lautet wie folgt:

```
devtools::install_github("j3ypi/rBasics")
```

Jetzt sollte es möglich sein, den Datensatz zu laden, so dass Du sofort die jeweiligen Befehle ausprobieren kannst.

```
data(bigFive, package = "rBasics")
```

Gehe jetzt wieder zu Abschnitt 5 und fahre mit dem Tutorial fort. Wenn Du wie unter 1.3 beschrieben, alles Notwendige durchgearbeitet bzw. durchgelesen hast, führe den folgenden Befehl aus, um ein interaktives Tutorial in Deinem Standardbrowser zu öffnen.

```
learnr::run_tutorial("wiederholung", package = "rBasics")
```

Der Übungssatz beinhaltet verschiedene Übungen, die das Verständnis des Gelernten prüfen sollen. Dabei kann man sich Hinweise holen, wenn man nicht weiterkommt. Bei Multiple Choice Fragen wird sofort geprüft, ob die Antwort korrekt ist. Um zu erfahren, ob der geschrieben Code richtig ist, kann man mehrmals auf **Hints** drücken, da der letzte Hinweis immer die Lösung beinhaltet. Der Fortschritt wird automatisch gespeichert, sodass man zu einem späteren Zeitpunkt die Übungen fortführen kann. Viel Erfolg!