Einführung

R für Psychologen v1.0.2

Jan Philipp Nolte

The Health and Life Science University

Inhaltsverzeichnis

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, näher auf SPSS einzugehen.

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 jetzigen 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. 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 jedoch rasant gewachsen und zur 6. beliebtesten Programmiersprache der Welt 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

Das Tutorial ist grundlegend in zwei Teile gegliedert. Dieser Teil dient der Einführung. Das andere Dokument gibt fortgeschrittenere Einblicke in R als Programmiersprache. Unter 6. wird ein Datensatz eingeführt, an dem sämtliche Beispiele gezeigt werden. Es wird eindeutig empfohlen, während der Bearbeitung dieses Tutorials den Datensatz in eine eigene R Session zu laden, um das gelernte direkt auszuprobieren. Unter 13. soll das Gelernte dann in einer interaktiven Lernumgebung, an einem anderen Datensatz angewendet werden. Kapitel 7 und 10 dienen als Komprehendium und können zunächst übersprungen werden, falls das Lernen von R als Programmiersprache als solche und nicht die Statistik zunächst Priorität hat.

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

Bevor wir loslegen können, müssen wir erst einmal alles Notwendige installieren.

1.4.1 Installiere die Programmiersprache R

Windows & macOS:

Für Windows und macOS unterscheidet sich die Installation kaum. Für Windows gehe auf CRAN dann `Download R for Windows` -> `base` -> `Download`. Benutze für macOS den selben Link, dann `Download R for (Mac) OS X` und klicke abschließend auf das aktuellste `.pkg`.

Ubuntu:

Für Ubuntu besteht der Installationsprozess an sich lediglich aus 2 Zeilen Code. Je nach Ubuntu Version, muss man allerdings noch das Repository hinzufügen (14.04, 16.04).

```
sudo apt-get update
sudo apt-get install r-base
```

1.4.2 Installation der Programmierumgebung RStudio

Wenn R installiert ist, kann man unter folgenden Link die Programmierungebung RStudio herunterladen. Drücke dazu auf den Download Button für RStudio Desktop (Open Source License) und Downloade die für dein Betriessystem richtige Version.

1.4.3 Aufbau von RStudio

Die Oberfläche von RStudio ist in vier Teile unterteilt (siehe Abbildung 1). Unten links befindet sich die Konsole. Dort kannst du eine Zeile Code nur einmalig ausführen. Wenn man später erneut den selben Befehl ausführen möchte, muss man ihn neu eingeben. Oben links befindet sich das so genannte Script. Du kannst mehrere Scripts gleichzeitig geöffnet haben. Der große Unterschied zur Konsole ist das Speicher jeder Zeile deines Codes, sodass du später lediglich erneut das Script ausführen musst, um genau die selben Ergebnisse erneut zu erhalten. Der vom Script erzeugte Output wird in der Konsole angezeigt. Man kann zwischen Script und Konsole mit `crtl (strg) + 1 / 2` wechseln. 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`. Oben rechts befindet sich die `Environment`. In dieser kannst du sämtliche Datensätze und Variablen betrachten, die du erstellt hast. Unten rechts sind verschiedene Reiter abgebildet. Die wichtigsten für dich sind Plots und Help. In Plots werden dir wenig überraschend die erstellten Plots angezeigt. Unter Help werden intuitiver Weise die Hilfen angezeigt. Wie diese aufgerufen werden, wird im folgenden Tutorial **Die nächsten Schitte** erklärt.

In Abbildung 2 ist ein Beispiel für eine Variable in der `Environment` dargestellt. Außerdem kann man die Farbe des Editors verändern. Dies kann unter `Global Options...` -> `Appearance` -> `Editor Theme` vollzogen werden.

Darüber hinaus kann die Anordnung von Script, Konsole etc. geändert werden (siehe Abbildung 3). Dies hängt wie das Aussehen des Editors von persönlichen Präferenzen ab. Dies ist lediglich ein Beispiel zur höheren Übersichtlichkeit. Falls du deine Anordnung verändert möchtest, gehe zu `Global Options...` -> `Pane Layout`.

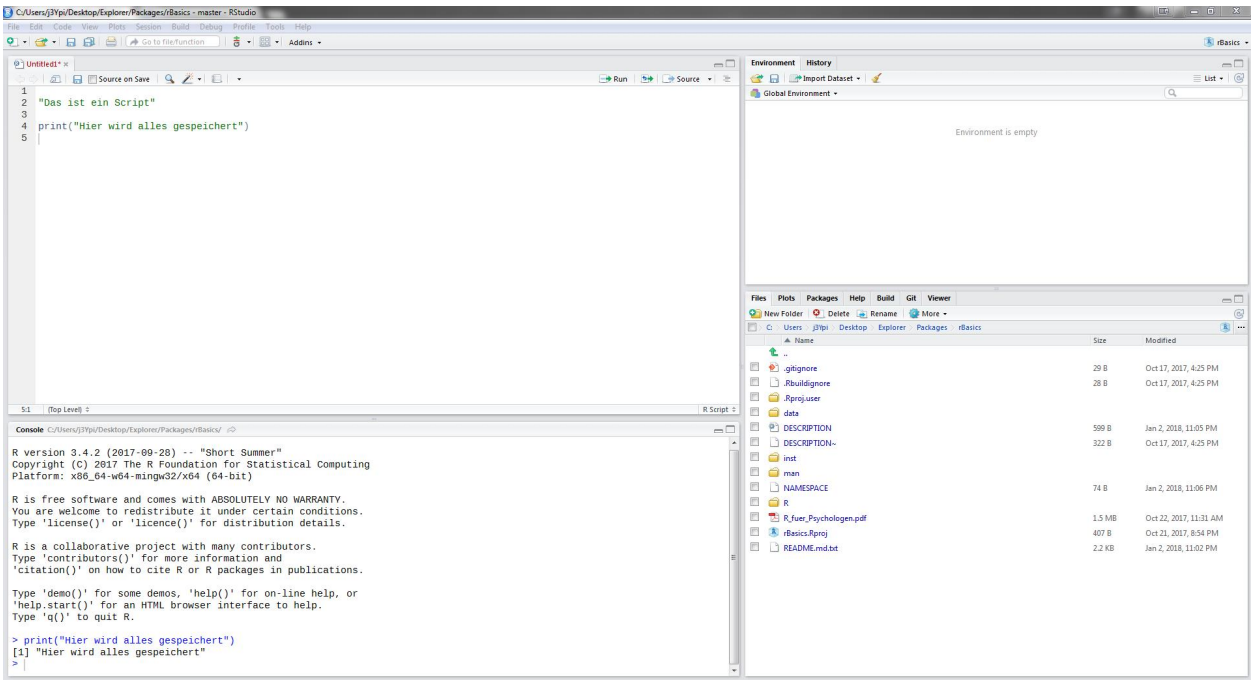


Abbildung 1: Aufbau von RStudio nach frischer Installation

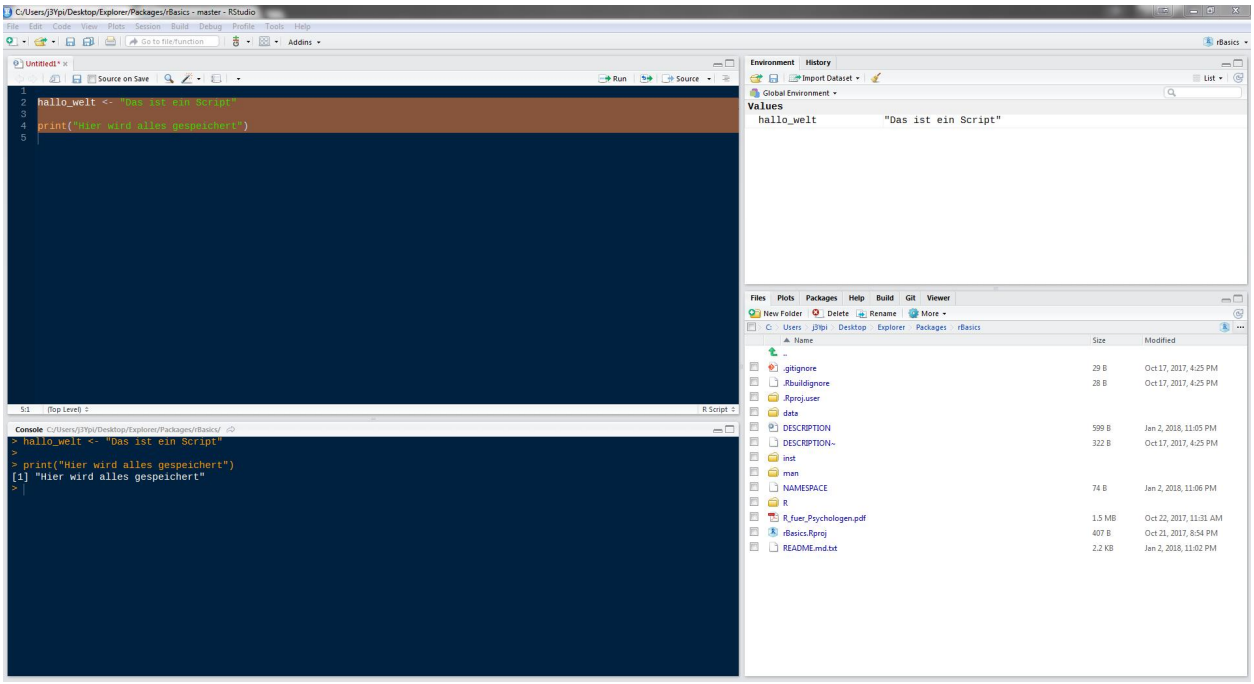


Abbildung 2: Aufbau nach Farbveränderung des Editors

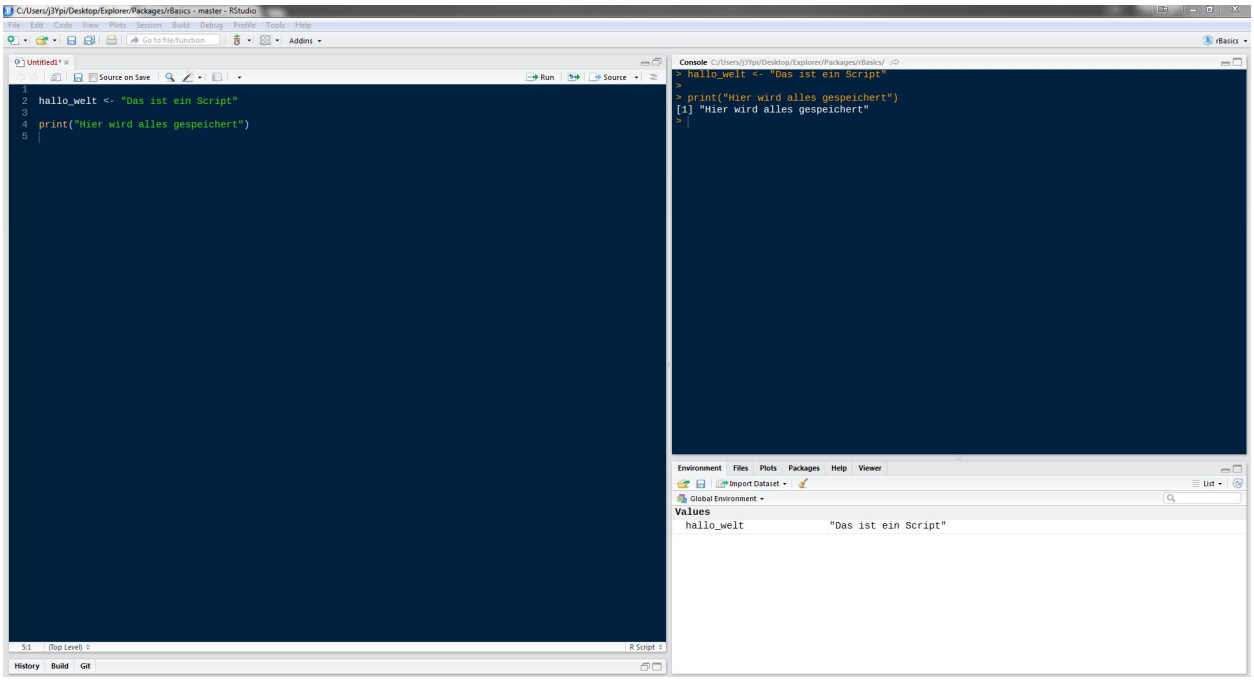


Abbildung 3: Variierbare Anordnungen in RStudio

1.4.4 Never save your workspace

Wenn sich ein Fehler irgendwo im Code eingschlichen hat, kann es sehr 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.

1.4.5 Default text encoding

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

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

2 Was ist ein Package?

R bietet von Beginn an eine Bandbreite von Funktionen. Da diese Funktionen allerdings Basics darstellen und nicht auf spezifische 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 mittlerweile 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 Anführungszeichen 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.

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

2.3 Installieren der notwendigen Packages

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

```
install.packages(c("tidyverse", "here", "rio", "broom", "car"
                   "ggpubr", "psy", "devtools", "learnr"), 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")
```

3 Projektorientierter Workflow

Der ein oder andere, der bereits mit R gearbeitet hat, weiß, dass das Einlesen von Dateien nur dann funktioniert, wenn man R sagt, wo genau sich die Datei befindet. Genau an dieser Stelle beginnt ein Problem - vor allem, wenn man mit Kollegen zusammenarbeitet, mehr als einen Computer hat oder die Datei verschiebt und zu einem späteren Zeitpunkt wiederverwenden möchte. Denn die weit verbreitete Vorgehensweise ist das Festlegen des Arbeitsverzeichnisses (Working Directory) am Anfang jedes R Scriptes mit `setwd()`. Das sieht dann, wenn die Files auf dem Desktop liegen, in etwa so aus:

```
## Windows:
setwd("C:/Users/userName/Desktop")
## UNIX:
setwd("~/Desktop")
```

Nicht nur gibt es Plattformabhängige Unterschiede, sondern viel schwerwiegender, hat jeder die Files an einer anderen Position liegen. Man muss also um das Script laufen zu lassen, erst einmal das Argument `setwd()` verändern. Die Lösung für dieses Problem ist ein projektorientierter Workflow. Man kann ein Projekt direkt innerhalb von RStudio erstellen. Oben rechts ist ein Reiter, über den man einfach zwischen den Projekten wechseln kann. Beim öffnen der Projektdatei, werden alle zugehörigen R Scripte - vorausgesetzt sie wurden innerhalb des Projekts einmal geöffnet - gemeinsam geladen. Nun muss R nur noch mittgeteilt werden, wo genau auf dem Computer das Projekt existiert. Dazu wurde das `here` Package entwickelt. Die Funktion `here()` findet selbstständig den relativen Pfad heraus, an dem die Projektdatei gerade liegt. Wichtig für das Funktionieren dieser Funktion ist entweder eine `.here` oder `.Rproj` Datei. Dabei kann man sich entscheiden, welche man bevorzugt. Erstere kann man mit `set_here()` erstellen. Letztere wird automatisch von RStudio beim Kreieren eines Projektes erstellt. Diese Datei muss sich im Hauptverzeichnis - also da wo die R Scripte liegen - befinden. Gerade bei größeren Projekten bietet es sich an, nicht alle Dateien in einem Ordner zu haben. Es wird empfohlen, alle R Scripte und die `.Rproj` Datei

ins Hautverzeichnis zu legen, alle Datensätze in einem Unterordner `data` und alle Graphiken in einem Unterordner `plot`. Zum Finden des Datensatzes muss man der Funktion `here()` als erstes Argument den Unterordner `data` und als zweites Argument den Datensatz geben. Falls der Datensatz im selben Ordner wie die `.rproj` Datei liegt, kann das erste Argument (hier `data`) weggelassen werden.

```
library(here)
here("data", "test.csv")
## Alternativ
here("test.csv")
```

Aber wie kann man den Datensatz nun einlesen?

4 Einlesen und speichern häufiger Dateientypen

Datensätze können in verschiedenen Formaten vorliegen. Dies ist vor allem abhängig davon mit welchen Programmen Unternehmen, Universitäten oder Kollegen arbeiten. Die in der Psychologie am häufigsten verwendeten sind:

- R (.RData | .rda | .rds)
- Excel (.xlsx | .xls)
- SPSS (.sav)
- Comma seperated values (.csv)
- Tabulator seperated values (.tsv)

4.1 Einlesen eines Datensatzes

Eingelesen werden können sämtliche Dateintypen mithilfe von `import()` aus dem `rio` Package. Dabei erkennt die Funktion die Dateiendung und übernimmt hinter den Kulissen alles Weitere. Damit der Datensatz als tibble (siehe 6.1) eingelesen wird, muss man `setclass = "tbl"` setzen.

```
library(rio)
import(here("data", "test.csv"), setclass = "tbl")
## Alternativ
import(here("test.csv"), setclass = "tbl")
```

4.2 Einlesen multipler Datensätze

Nicht selten kommt es vor, dass man nicht nur einen, sondern gleich mehrere Datensätze einlesen möchte. Dabei werden die verschiedenen Files in einer Liste gespeichert. `list.files()` listet alle Files innerhalb eines Ordners auf. Beachte, dass immer, wenn man den Pfad zu einem Datensatz festlegt, die zuvor kennengelernte `here()` Funktion verwendet werden muss.

```r
files <- list.files(here("data"))
import_list(here("data", files), setclass = "tbl")
```

4.3 Speichern

Auch das speichern von Datensätzen könnte in R durch das rio Package intuitiver nicht sein. Anstelle von import() muss man lediglich export() benutzen. Das erste Argument der Funktion ist der Datensatzname, mit dem man innerhalb von R auf selbigen zugreift.

```r
export(datensatz, here("data", "test.csv"))
```

4.4 Konvertieren von Dateien

Oft kann man in die Situation kommen, in der man den Dateientyp konvertieren möchte. Zum Beispiel wenn man ein SPSS (.sav) File hat, aber auch mit Office Produkten den Datensatz anschauen möchte. In dem Fall könnte man einfach `convert()` aus dem `rio` Package benutzen. Auf `here()` sei aus Gründen der Übersichtlichkeit an dieser Stelle verzichtet.

```r
convert(in_file = "test.sav", out_file = "test.xlsx")
```

5 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. In diesem Fall ist der Variablenname `bigfive`.

```r
bigFive <- import(here("data", "bigFive.csv"))
```

6 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).

Da der Datensatz im Package enthalten ist, kannst du diesen wie folgt einfach in R einlesen.

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

Mit `print()` kann man den Datensatz ausgeben lassen. Alternativ kann man ebenfalls eine Ansicht unter der **Environment** aufrufen.

6.1 tibbles

Es gibt in R verschiedene Datenstrukturen. Diese werden im weiterführenden Tutorial **Die nächsten Schritte** genauer erklärt. Wichtig ist für den Moment, wie man einen Datensatz innerhalb von R erstellt - ihn also nicht aus einer externen Datei einliest. Wir halten uns der Konsistenz halber auch hier an die Funktionen des `tidyverse`. Wenn man also einen Datensatz mit zwei Spalten (Spalte1, Spalte2) erstellen möchte, kann man das mit dem Befehl `tibble()` machen. `c()` erlaubt es mehrere Werte mit einander zu einer Spalte zu kombinieren. Für die genaue Funktionsweise und Struktur von `c()`, sei auch an dieser Stelle an das weiterführende Tutorial verwiesen.

```
tibble(
  Spalte1 = c(1, 2, 3),
  Spalte2 = c("a", "b", "c")
)
```

```
## # A tibble: 3 x 2
##   Spalte1 Spalte2
##     <dbl>   <chr>
## 1       1       a
## 2       2       b
## 3       3       c
```

6.2 Auf Spalten zugreifen

Manchmal ist es notwendig auf einzelne Spalten eines Datensatzes zuzugreifen. Beispielsweise wenn man das mittlere Alter der Versuchspersonen ausrechnen möchte. Es gibt eine Methode aus dem `tidyverse`, mit der man Spalten auswählen kann. Das erste Argument ist dabei der Datensatz und alle weiteren die gewünschten Spalten. Dabei können beliebig viele per Spaltennamen angesprochen werden. In dem Fall werden vom `bigFive` Datensatz die Spalten gender und age ausgewählt.

```
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
```

Wenn man lediglich eine Spalte auswählen möchte - z.B. um den Mittelwert des Alters zu berechnen (siehe 6.) - kann man mit dem Dollar-Operator Code sparen.

```
bigFive$age
```

7 Deskriptive Statistik

`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        Extraversion   Neurotizismus
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:4.000   1st Qu.:3.000   1st Qu.:2.900   1st Qu.:2.600
##  Median :4.000   Median :4.000   Median :3.100   Median :3.100
##  Mean   :4.134   Mean   :4.005   Mean   :3.077   Mean   :3.095
##  3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.:3.300   3rd Qu.:3.600
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
```

```
dim(bigFive)
```

```
## [1] 19719    56
```

Die in `summary()` ausgegebenen deskriptive Werte, kann man ebenfalls manuell ausrechnen:

```
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
```

8 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 ist 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. Ungleichheit kann man mit `!=` abfragen.

```
## Numeric
42
```

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

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

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

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

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

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

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

9 Datenmanipulation mit dplyr

`dplyr` ist ein Package innerhalb des `tidyverse`. Es ist also kein zusätzliches Laden notwendig, wenn man bereits `library(tidyverse)` geschrieben hat. Die Vorteile im Vergleich zu Base R Funktionen sind unter anderem die Konsistente Syntax, die eindeutigen Funktionsnamen und der Pipe Operator. Die vier wichtigsten Funktionen sind hier nachfolgend aufgeführt.

9.1 filter

Gibt nur jene Zeilen einer Spalte aus, die mit der gewählten Bedingung übereinstimmen. Das erste Argument ist wie bei `select()` der Datensatz.

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

```
## # A tibble: 7,608 x 56
##      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 45 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>, Neurotizismus <dbl>
```

9.2 mutate

Erstellt eine neue Spalte. `rowMeans()` berechnet die Mittelwerte pro Zeile.

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

```
## # A tibble: 19,719 x 56
##      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 45 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>, Neurotizismus <dbl>
```

9.3 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
```

9.4 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 sehen möchte:

```
bigFive %>%
  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    53      1          3.2
##  2    23      1          3.5
##  3    15      1          3.2
##  4    21      1          2.6
##  5    21      1          3.3
##  6    48      1          3.4
##  7    22      1          3.0
##  8    37      1          2.9
##  9    25      1          3.0
## 10    15      1          3.1
## # ... 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]))
```

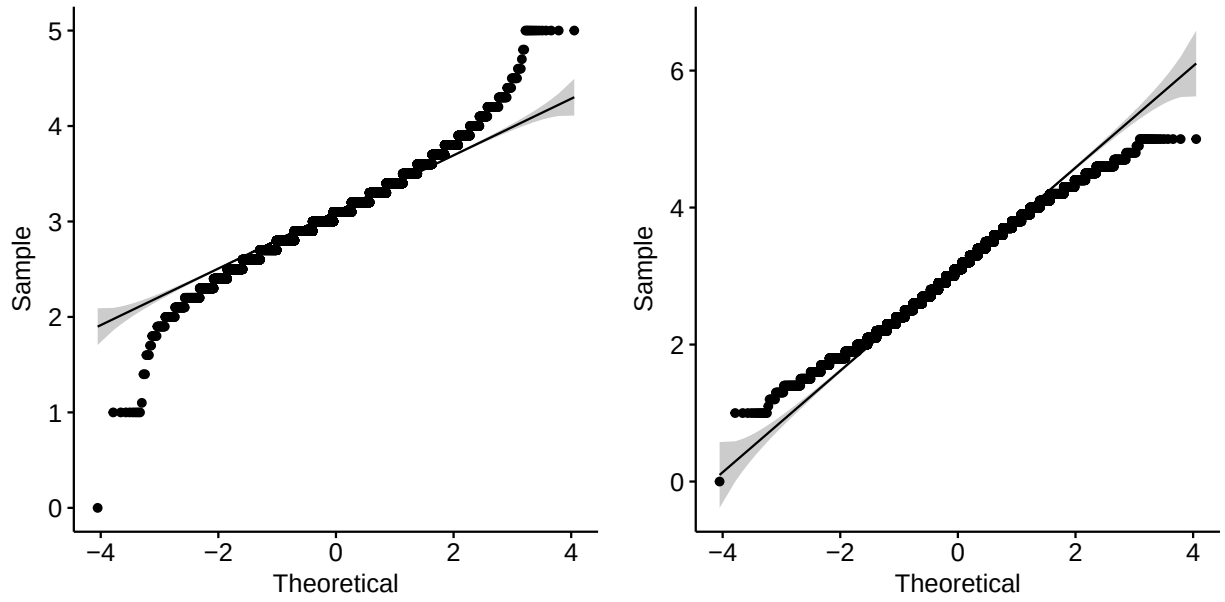
10 Induktive Statistik

10.1 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")
```



10.2 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
```

10.3 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 10.10 wird eine Lösung dafür vorgeschlagen. `Pr(>|t|)` ist der p-Wert. Bei SPSS wird unter .001 nur noch <.001 angezeigt. Bei R hingegen wird bis 2e-16, also 2 x 10^-16 (0.0000000000000002) differenziert. Um die Ergebnisse zu erhalten, muss man das Modell durch `summary()` ausgeben lassen.

```
## 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
```

10.4 Varianzanalysen (ANOVA)

Die Anpassung des p-Wertes wird vor allem bei ANOVA Verfahren gerne vergessen. Da man in der Regel verschiedene Hypothesen auf einmal testet, ist dies jedoch unabdingbar (siehe 10.7).

```
## 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
```

Der ANOVA Befehl aus Base R gibt den Typ I Sums of squares (SS) zurück. Andere Statistikprogramme wie SAS und SPSS hingegen geben den Typ III SS aus. Dies führt zu unterschiedlichen Ergebnissen. Der Typ I SS darf nur dann verwendet werden, wenn in den verschiedenen Gruppen die selbe Anzahl an Versuchspersonen eingeplant werden. Falls diese Anzahl variiert, muss Typ II oder III verwendet werden. Wenn keine Interaktionseffekte bestehen Typ II, ansonsten Typ III.

In R müssen wir dafür das `car` Package laden. Die Funktion für beide Typen ist `Anova()`. Beachte den großgeschriebenen Anfangsbuchstaben. Der einzige Unterschied ist das Argument `type`, welches entsprechend auf II oder III gesetzt werden muss. Damit der Typ III SS die richtigen Ergebnisse ausgibt, muss außerdem das `contrasts` Argument in den Optionen von R verändert werden.

```
library(car)
Anova(reg, type = 2)
```

```
## Anova Table (Type II tests)
##
## Response: Extraversion
##                Sum Sq    Df F value Pr(>F)
## Neurotizismus     0.0     1  0.0063 0.9366
## Residuals      2446.5 19717
```

```
options(contrasts = c("contr.sum", "contr.poly"))
Anova(reg, type = 3)
```

```
## Anova Table (Type III tests)
##
```

```
## Response: Extraversion
##                Sum Sq    Df   F value Pr(>F)
## (Intercept)   8397.6     1 6.768e+04 <2e-16 ***
## Neurotizismus    0.0     1 6.300e-03 0.9366
## Residuals     2446.5 19717
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

10.4.1 TukeyHSD

Für den Post-Hoc Test von Tukey, müssen wir das Modell mit aov() erstellen. Das Ergebnis von aov wird dann der Base R Funktion TukeyHSD weitergegeben. In diesem Beispiel wird gender als Einflussgröße gewählt, da diese aus Faktoren bestehen muss.

```
anov <- aov(Extraversion ~ as.factor(gender), data = bigFive)
TukeyHSD(anov)
```

10.5 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
```

10.6 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 über 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. Wie genau der Zugriff auf verschiedene Zeilen funktioniert, wird im nachfolgenden Tutorial **Die nächsten Schritte** erklärt.

```
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
```

10.7 p-Wert Korrektur

Wenn man mehrere verschiedene statistische Tests anwendet, muss man entweder das gewählte alpha Niveau (Fehler 1. Art) oder die aus den Tests resultierenden p-Werte anpassen. Wir konzentrieren uns auf Letzteres. Dies kann man auf verschiedene Arten machen. Eine sehr häufig benutzte, aber je nach Anwendungskontext zu konventionelle Methode, ist die Bonferroni Korrektur. Bei der Bonferroni Korrektur werden die resultierenden p-Werte mit der Anzahl der Tests multipliziert. `p.adjust()` aus Base R übernimmt das für uns. Die Benjamini-Hochberg (BH) Korrektur ist etwas komplizierter, hat aber den Vorteil keine Inflation des Fehlers 2. Art zu verursachen. Man muss also die p-Werte, die man aus den induktiven Tests erhält, korrigieren. Wenn man also vier p-Werte aus vier Tests erhält:

```r
p_values <- c(0.05, 0.3, 0.04, 0.001)

p.adjust(p = p_values, method = "bonferroni", n = 4)
```

```
## [1] 0.200 1.000 0.160 0.004
```

```r
p.adjust(p = p_values, method = "BH", n = 4)
```

```
## [1] 0.06666667 0.30000000 0.06666667 0.00400000
```

10.8 Statistische Power

Die statistische Power ist eines der wichtigsten Konzepte in der Inferenzstatistik. Auf der einen Seite kann man mit der Power den Fehler 2. Art kontrollieren, der je nach Kontext teils schwerwiegender als der alpha Fehler sein kann. Auf der anderen Seite erhält man Auskunft über die Bedeutsamkeit des potentiell signifkanten Ergebnisses. Bei einer Studie mit zu viel Power wird jeder noch so kleine Unterschied signifikant. Umgekehrt ist auch zu wenig Power fatal, da Effekte ggf. nicht gefunden werden. Um die Power auszurechnen, kann man das `pwr` Package verwenden. Dieses hat die selbe Funktionalität wie das häufig empfohlende Programm g*power. Die Powerberechnung für die verschiedenen Testarten unterscheidet sich nur durch das Verfahren (X) in der Mitte nach dem Schema `pwr.X.test()`. Exemplarisch sei hier die Powerberechnung für eine ANOVA angeführt. Dabei wurden 2 Gruppen mit jeweils 30 Personen bei einer Effektstärke von .3 und einem alpha Fehler von 5 Prozent verwendet.

```r
library(pwr)
pwr.anova.test(k = 2, n = 30, f = .3, sig.level = .05)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 2
##               n = 30
##               f = 0.3
##       sig.level = 0.05
##           power = 0.6275046
##
## NOTE: n is number in each group
```

10.9 Effektstärke

Ein weiteres wichtiges Maß zur Einschätzung der Bedeutsamkeit eines signifikanten Ergebnisses, ist die Effektstärke. Cohens d kann man mit `cohen.d()` ausrechnen. Dabei sind die Argumente `d` und `f` die jeweiligen Gruppen.

```r
library(effsize)
cohen.d(d = c(10, 5, 4, 10, 2), f = c(5, 10, 20, 3, 50, 7))
```

```
##
## Cohen's d
##
## d estimate: -0.715462 (medium)
## 95 percent confidence interval:
##        inf        sup
## -2.1280595  0.6971355
```

10.10 Aufgeräumte Outputs mit Broom

Das Package Broom konvertiert die Ergebnisse verschiedener statistischer Tests mit `tidy()` zu übersichtlicheren Outputs, die man weiterverarbeiten kann. Mit `glance()` erhält man zusätzliche Informationen wie R^2 oder die logistische Likelihood. `reg` ist das lineare Regressionsmodell von zuvor.

```r
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
```

```r
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
```

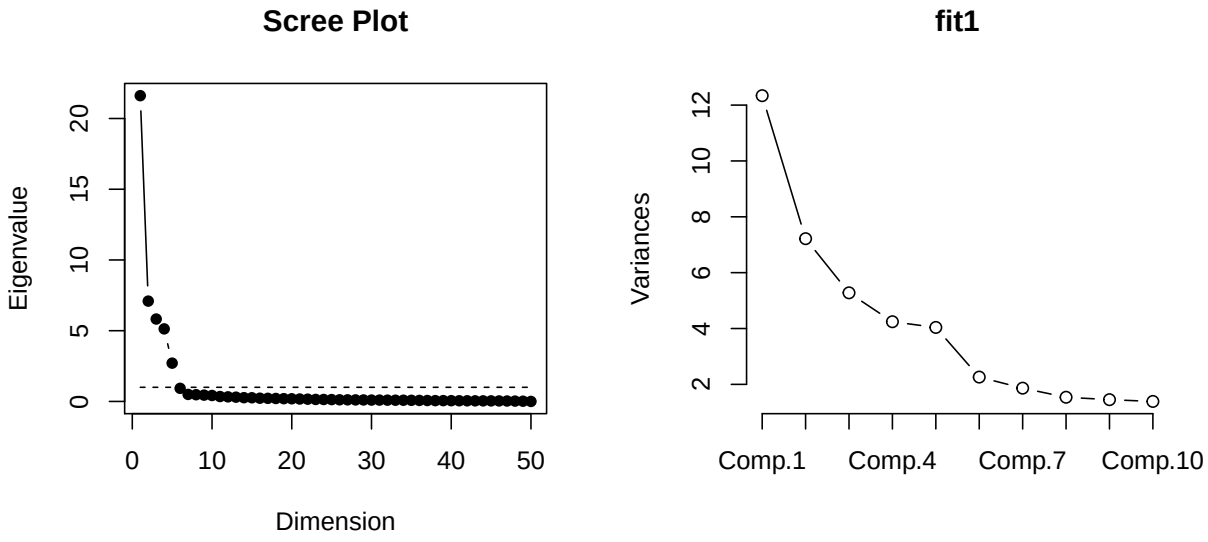
10.11 Exploratorische Faktorenanalyse

Nun möchten wir schauen, ob die 50 Fragen zu den verschiedenen Persönlichkeitseigenschaften auch tatsächlich durch fünf 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.

```r
## 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
```

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



11 Aufräumen des Datensatzes mit tidyr

`tidyr` bietet Funktionen zur Umwandlung von Datensätzen, die nicht tidy sind. Auch `tidyr` wird durch das `tidyverse` mit geladen. 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 Charakters und keine 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 in einen data.frame umwandeln, damit man den Graph mit `ggpubr` zeichnen kann. Für den genauen Unterschied zwischen data.frames und tibbles sei an dieser Stelle erneut auf das fortführende Tutorial

verwiesen.

```
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
```

Um die Tranformation wieder rückgängig zu machen, kann man auf die selbe Art und Weise `spread()` verwenden.

12 Visualisierungen

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. Solange man die Schriftgröße der Achsenbeschriftung anpasst, sowie die Legende innerhalb der Grenzen des Graphen verlegt, entspricht der Stil von `ggpubr` den APA-Richtlinien.

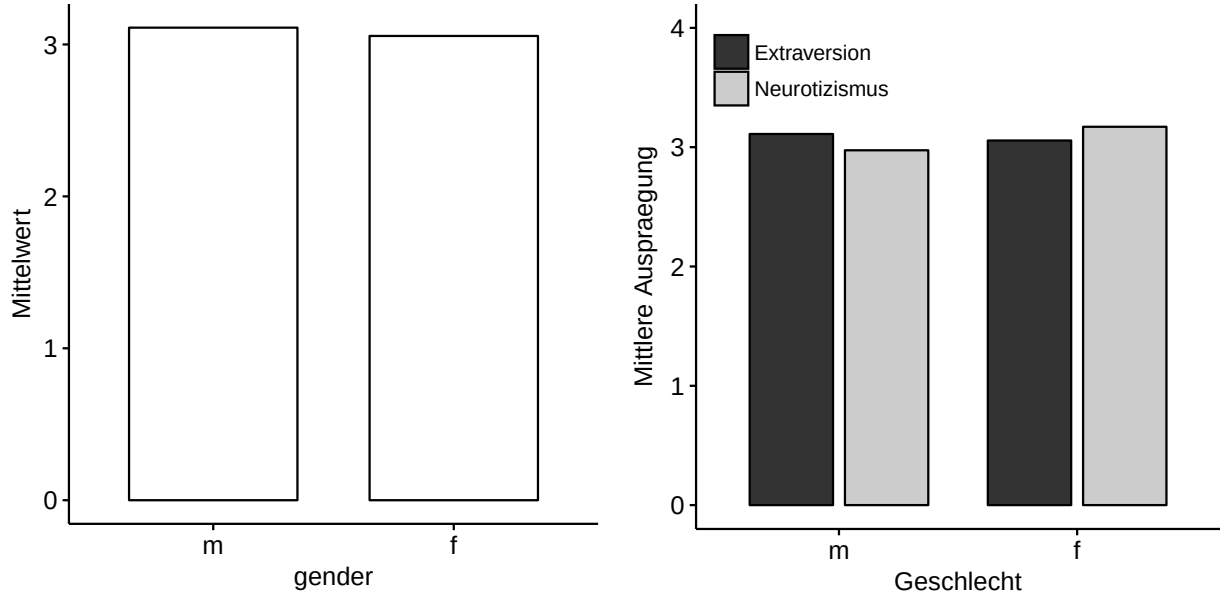
12.1 Säulendiagramm

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.

```
ggbarplot(data = df[c(1, 2), ],
          x = "gender",
          y = "Mittelwert")

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))
ggpar(p = b,
      xlab = "Geschlecht",
      ylab = "Mittlere Auspraegung",
      ylim = c(0, 4),
      legend.title = "")
```

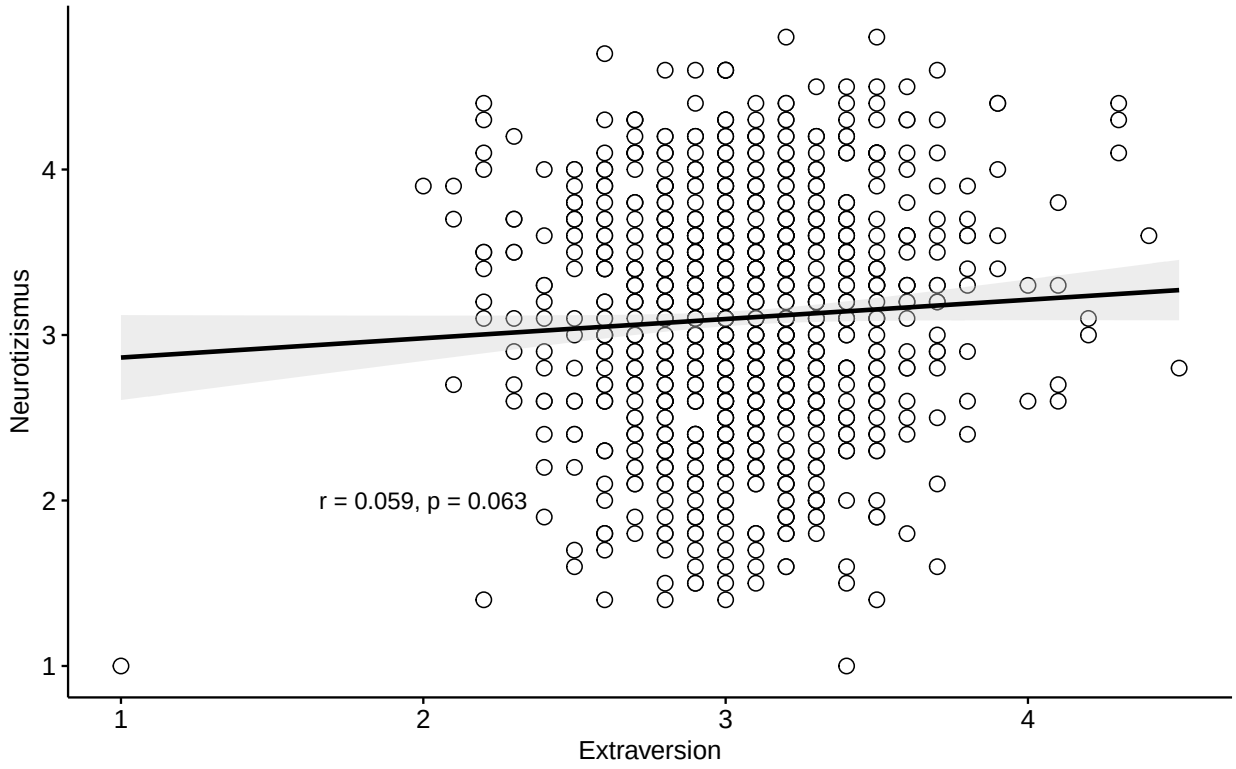


12.2 Streudiagramm

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

```
ggscatter(data = as.data.frame(bigFive[1:1000, ]),
          x = "Extraversion",
          y = "Neurotizismus",
          cor.coef = T,
          cor.coef.coord = c(2, 2),
          cor.method = "pearson",
          conf.int = T,
```

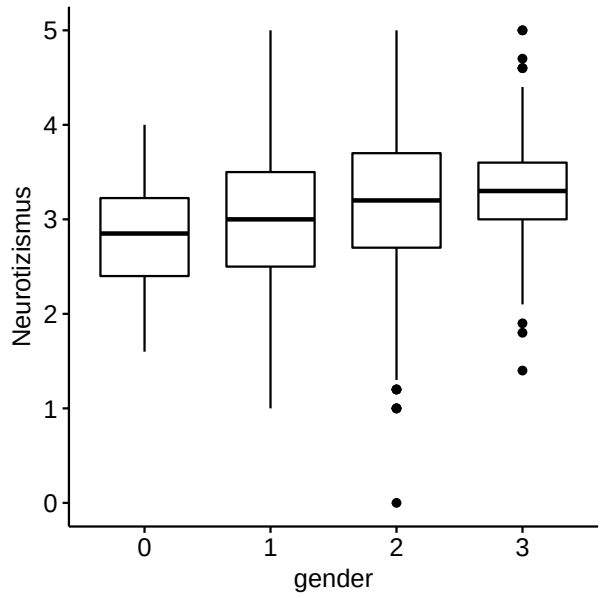
```
          add = "reg.line",
          add.params =  list(color = "black",
                             fill = "lightgray"),
          size = 3,
          shape = 21,
          color = "black") + theme(axis.title.x = element_text(size = 12),
                                   axis.title.y = element_text(size = 12))
```

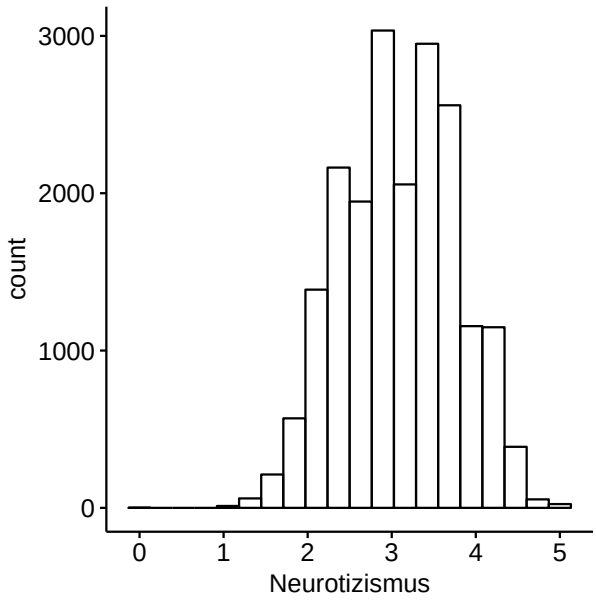


12.3 Boxplot und Histogramm

```
ggboxplot(data = bigFive,
          x = "gender",
          y = "Neurotizismus")

gghistogram(data = bigFive,
            x = "Neurotizismus",
            bins = 20)
```

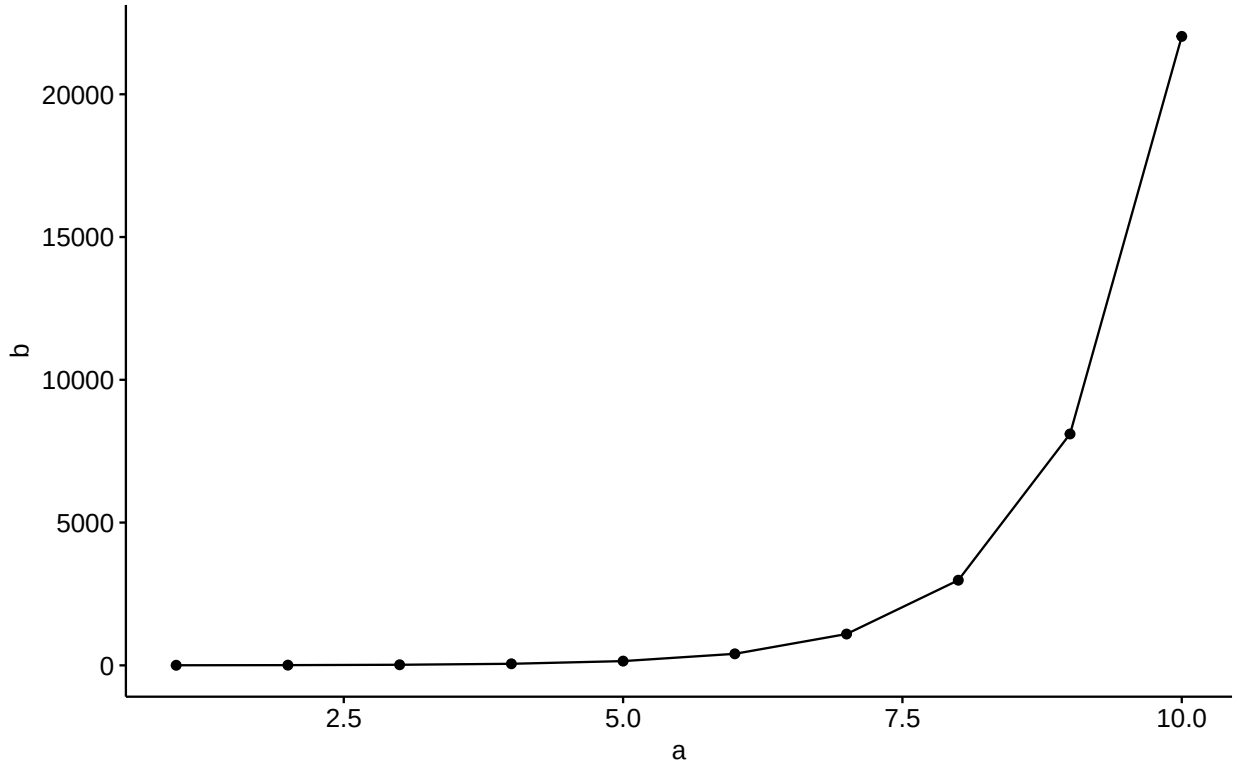




12.4 Liniendiagramm

```
df2 <- tibble(
  a = 1:10,
  b = exp(1:10)
)

ggline(data = df2, x = "a", y = "b")
```



12.5 Speichern

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

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

13 Übungen

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!