R Guide

für das Neurolinguistics Lab Mainz

Stand: 04.02.2021

Contents

To-do:

- (automatische Signifkanzssterne in Plots)
- (Generalized Linear Mixed Effects Models)
- (Bayesian methods)

- (interaktive Aufgaben)

Bei Fragen, Verbesserungsvorschlägen und Anregungen gerne an josh.ziegler@icloud.com wenden.

---

1 R

Was ist R?

> *R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.* (Quelle: https://www.r-project.org)

Wie installiere ich R?
R kann kostenlos über das Comprehensive R Archive Network (CRAN) heruntergeladen und anschließend installiert werden.

Wie installiere ich RStudio (ein beliebtes GUI für R)?
Rstudio ist ein Graphic User Interface (GUI) für R, welches die Arbeit mit R übersichtlicher machen kann. Es kann kostenlos von der RStudio Homepage heruntergeladen und anschließend installiert werden.

Was sind Pakete (*packages*)?
Pakete sind Erweiterungen für R, die sich direkt über die Konsole installieren und laden lassen. Sie beinhalten z.B. neue Funktionen, überarbeitete Funktionen oder auch Datensätze und ergänzen somit R um die vorinstallierten Funktionen (= *baseR*).

Wie lerne ich R?
Um R zu lernen gibt es viele Resourcen. Ich empfehle das Paket **swirl**, da es direkt innerhalb RStudio durchgeführt werden kann. Um das Tutorial zu starten müssen folgende Schritte in der *Console* ausgeführt werden:

1. Das Paket "swirl" installieren: `install.packages("swirl")`
2. Das Paket "swirl" laden: `library(swirl)`
3. Die Funktion swirl() ausführen: `swirl()`

swirl leitet dann innerhalb der *Console* durch die Tutorials.

Des Weiteren empfielt es sich, die eigenen R-Projekte so zu organisieren, dass sie auch nach langer Zeit noch ersichtlich sind. Neben eindeutigen Benennungen von Variablen, Daten, Plots, etc. lohnt es sich vor allem eine gute Ordnerstruktur anzulegen. Hilfreiche Links dazu:

- RStudio Projects
- Version Control: GitHub & RStudio Guide von Jennifer Bryan

Was ist ein R Markdown?
R Markdown ermöglicht es, Code in einer schriftlichen Arbeit zu integrieren. Der Code befindet sich in sogenannten "Chunks", die sich in RStudio direkt in der Datei ausführen lassen. Das ganze Dokument, inklusive Code und Output, lässt sich unter Anderem als html- oder pdf-Datei exportieren. Mehr Informationen gibt es auf der offiziellen Website.

R Markdown verfügt über viele gestalterische Optionen. Ein Cheatsheet lässt sich in RStudio wie folgt aufrufen:
Help > Cheatsheets > R Markdown Cheat Sheet

Wie werden R, RStudio und Pakete geupdated?
David von *R for the Rest of us* hat eine wunderbare Guide zusammengestellt, wie sich R, RStudio und die Pakete auf Mac und Windows updaten lassen, ohne dass dabei Probleme entstehen (*sollten...*):

https://rfortherestofus.com/2020/09/how-to-update-rstudio-r-packages/

Post-Tutorial: Troubleshooting!
Generell gilt, dass sich viele Probleme lösen lassen, indem die Dokumentation einer Funktion aufgerufen wird. Dies geht direkt in der *Console*, indem ein ? vor die Funktion gesetzt wird (z.B.: `?summary`).

Da auch das nicht immer hilft, empfiehlt es sich Foren, wie bspw. Cross Validated und Stack Overflow, zu durchforsten, sich mit den Regeln des Forums vertraut zu machen und bei Bedarf selbst eine Frage zu stellen.

- Wie lese ich meine Daten ein?
  - .txt-Datei: `txt.data <- read.delim(("Dateipfad", header = TRUE, sep = ";", dec = ",")` (hier lässt sich bestimmen, ob die erste Zeile als Variablennamen übernommen werden soll, sowie durch welche Zeichen die Zellen getrennt sind und welches Zeichen vor Dezimalstellen verwendet wird)
  - .csv-Datei: `csv.data <- read.csv("Dateipfad", header = TRUE, sep = ";", dec = ",")` (hier lässt sich bestimmen, ob die erste Zeile als Variablennamen übernommen werden soll, sowie durch welche Zeichen die Zellen getrennt sind und welches Zeichen vor Dezimalstellen verwendet wird)
  - .sav-Datei: `sav.data <- read_sav("Dateipfad")`(benötigt das Paket "haven"!)
- Wie bekomme ich meine Daten ins richtige Format?
  - Umwandeln einer Variable in das Factor-, Character- oder Numeric-Format: `as.factor(data$Variable)`, `as.character(data$Variable)` oder `as.numeric(data$Variable)`

  - Daten vom Weit- ins Langformat wandeln (Beispiel dazu im Kapitel McNemar-Test): mit den Funktionen `pivot_wider()` oder `pivot_longer()`
- R sagt mir, dass meine Variable nicht vorhanden ist, was jedoch nicht stimmen kann. Was nun?
  - Oft lohnt es sich zu überprüfen, ob noch ungewünschte Gruppierungen in den Daten vorhanden sind, die durch die Funktion `group_by`ausgelöst wurden. Diese lassen sich mit der folgenden Funktion aufheben: `data <- data %>% ungroup()`.

---

2 Setup

Diese Guide arbeitet mit exemplarischen Daten als eine Art Blaupause für eure R-Projekte. In erster Linie wird der Datensatz **sleepstudy** aus dem Paket *lme4* verwendet:

> *"The average reaction time per day for subjects in a sleep deprivation study. On day 0 the subjects had their normal amount of sleep. Starting that night they were restricted to 3 hours of sleep per night. The observations represent the average reaction time on a series of tests given each day to each subject."* (Quelle: `?sleepstudy`)

Die Kapitel Stepwise Regression und Pairwise Comparison bei Linear Mixed Effect Models arbeiten mit dem *etwas* weniger ansprechenenden Datensatz **ham** aus dem Paket *lmerTest*:

> *"One of the purposes of the study was to investigate the effect of information given to the consumers measured in hedonic liking for the hams. Two of the hams were Spanish and two were Norwegian, each origin representing different salt levels and different aging time. The information about origin was given in such way that both true and false information was given. Essentially a 4x2 design with 4 samples and 2 information levels. A total of 81 Consumers participated in the study."* (Quelle: `?ham`)

Zunächst müssen die Pakete geladen werden:

```
# Laden der Pakete "lme4", "tidyverse" und "car"
library(lme4)  # beinhaltet die "sleepstudy"-Daten & die Funktionen lmer(), glmer()
library(tidyverse) # beinhaltet u.A. ggplot2, dplyr (und andere nützliche Pakete)
library(car) # beinhaltet u.A. die Funktion leveneTest()
library(rstatix) # beinhaltet u.A. die Funktion identify_outliers()
library(lmerTest) # erweitert lme4 (berechnet u.A. p-Werte)
library(influence.ME) # wird für influencial data bei linear mixed effects models benötigt
library(emmeans) # ermöglicht pairwise comparison bei linear mixed effect models
library(MuMIn) # ermöglicht die Ausgabe von R-Squared-Werten bei linear mixed effect models
library(ggsignif) # ermöglicht manuelles Hinzufügen von Signifikanzsternen in Plots
```

Sollten diese Pakete nicht installiert sein, folgende Kommandos in der Konsole eintragen (ohne "#"):

```
# install.packages("lme4")
# install.packages("tidyverse")
# install.packages("car")
# install.packages("rstatix")
# install.packages("lmerTest")
# install.packages("influence.ME")
# install.packages("emmeans")
# install.packages("MuMIn")
# install.packages("ggsignif")
```

---

3 Deskriptive Statistik

3.1 Übersicht der Daten

Das Kommando `head()` zeigt die ersten Zeilen eines Datensatzes und liefert so einen kurzen Überblick:

```
head(sleepstudy)
```

```
##   Reaction Days Subject
## 1 249.5600    0     308
## 2 258.7047    1     308
## 3 250.8006    2     308
## 4 321.4398    3     308
## 5 356.8519    4     308
## 6 414.6901    5     308
```

Übersicht über die Variablentypen:

```
str(sleepstudy)
```

```
## 'data.frame':    180 obs. of  3 variables:
##  $ Reaction: num  250 259 251 321 357 ...
##  $ Days    : num  0 1 2 3 4 5 6 7 8 9 ...
##  $ Subject : Factor w/ 18 levels "308","309","310",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Zusammenfassung der Daten:

```
summary(sleepstudy)
```

```
##     Reaction          Days        Subject
##  Min.   :194.3   Min.   :0.0   308    : 10
##  1st Qu.:255.4   1st Qu.:2.0   309    : 10
##  Median :288.7   Median :4.5   310    : 10
##  Mean   :298.5   Mean   :4.5   330    : 10
##  3rd Qu.:336.8   3rd Qu.:7.0   331    : 10
##  Max.   :466.4   Max.   :9.0   332    : 10
##                                (Other):120
```

Für den Zweck dieser Guide erstellen wir an dieser Stelle zusätzliche Variablen (Geschlecht, Fremdsprache, Wortflüssigkeit):

```
# Geschlecht
sleepstudy$Sex <- factor(rep(c("m","f"), each = 90))
# Fremdsprache
sleepstudy$Language <- factor(rep(c("DE", "ENG", "FRA"), each = 60))
# Wortflüssigkeit
sleepstudy <- sleepstudy %>%
  mutate(VerbalFluency = rnorm(180, 18, 2))
# Datenkopf
head(sleepstudy)
```

```
##   Reaction Days Subject Sex Language VerbalFluency
## 1 249.5600    0     308   m       DE      20.67091
## 2 258.7047    1     308   m       DE      17.26837
## 3 250.8006    2     308   m       DE      18.16458
## 4 321.4398    3     308   m       DE      19.91645
## 5 356.8519    4     308   m       DE      18.51281
```

```
## 6 414.6901    5     308    m       DE      16.93990
```

Wie sich neue Variablen erstellen lassen, wird im *swirl*-Tutorial behandelt. Weitere Übersichten zu diesem Thema finden sich bspw. hier und hier.

3.2 Weitere nützliche Funktionen für die deskriptive Statistik:

- **Arithmetisches Mittel** einer Variable: `mean(sleepstudy$Reaction)`
- **Median** einer Variable: `median(sleepstudy$Reaction)`
- **Quantile** einer Variable: `quantile(sleepstudy$Reaction)`
- **Summe** einer Variable: `sum(sleepstudy$Reaction)`
- **Standardabweichung** einer Variable: `sd(sleepstudy$Reaction)`
- **Anzahl** der Datenpunkte: `nrow(sleepstudy)`
- **Variable erstellen**: `sleepstudy$NewVariable <- NA`
- **Variable entfernen**: `sleepstudy$NewVariable <- NULL`

3.3 z-Transformation

Eine einfache z-Transformation wird mit der Funktion `scale()` durchgeführt. Da die Reaktionszeiten der sleepstudy zu verscheidenen Messzeitpunkten erhoben wurden, kalkulieren wir die z-Werte gruppiert nach Erhebungszeitpunkt ("group_by()").

```
sleepstudy <- sleepstudy %>%
  group_by(Days) %>%
  mutate(zReaction = scale(Reaction)) %>%
  ungroup()
```

3.4 log-Transformation

Die log-Transformation wird mit der Funktion `log()` durchgeführt:

```
sleepstudy$logReaction <- log(sleepstudy$Reaction)
```

3.5 Outlier

Eine elegante Methode zum identifizieren und entfernen von Outliern bietet das Paket *rstatix*:
Beschreibung von `identify_outliers()` (*rstatix* Paket):
Values above Q3 + 1.5xIQR or below Q1 - 1.5xIQR are considered as outliers. Values above Q3 + 3xIQR or below Q1 - 3xIQR are considered as extreme points (or extreme outliers).
Q1 and Q3 are the first and third quartile, respectively. IQR is the interquartile range (IQR = Q3 - Q1).

```
# Outlier einer Variable
outliers <- sleepstudy %>%
              identify_outliers(Reaction)
```

```r
# Outlier ausgeben
print(outliers)
```

```
## # A tibble: 2 x 10
##   Reaction  Days Subject Sex   Language VerbalFluency zReaction[,1] logReaction
##      <dbl> <dbl> <fct>   <fct> <fct>            <dbl>         <dbl>       <dbl>
## 1     466.     9 308     m     DE                16.4          1.72        6.14
## 2     459.     9 337     f     ENG               15.9          1.61        6.13
## # ... with 2 more variables: is.outlier <lgl>, is.extreme <lgl>
```

```r
# Outlier einer Variable, gruppiert nach einer anderen Variable
outliers_grouped <- sleepstudy %>%
                      group_by(Days) %>%
                      identify_outliers(Reaction)
# Outlier ausgeben
print(outliers_grouped)
```

```
## # A tibble: 8 x 10
##    Days Reaction Subject Sex   Language VerbalFluency zReaction[,1] logReaction
##   <dbl>    <dbl> <fct>   <fct> <fct>            <dbl>         <dbl>       <dbl>
## 1     5     415. 308     m     DE                16.9          2.05        6.03
## 2     6     454. 332     m     DE                18.3          2.25        6.12
## 3     9     466. 308     m     DE                16.4          1.72        6.14
## 4     9     237. 309     m     DE                19.4         -1.69        5.47
## 5     9     248. 310     m     DE                17.7         -1.54        5.51
## 6     9     254. 332     m     DE                17.3         -1.45        5.54
## 7     9     237. 335     m     ENG               19.9         -1.70        5.47
## 8     9     459. 337     f     ENG               15.9          1.61        6.13
## # ... with 2 more variables: is.outlier <lgl>, is.extreme <lgl>
```

Wie entferne ich die Outlier?
Die Anwendung der Funktion `identify_outliers()` auf die nach Erhebungstag (Days) gruppierten Daten ergab **acht** Outlier, wovon **sechs** als extrem eingestuft werden. Die Outliers sind wie folgt zu entfernen:

```r
# Alle Outlier
sleep_outliers <-
  sleepstudy[-which(sleepstudy$Reaction %in% outliers_grouped$Reaction), ]
# Nur extreme Outlier:
# Filtern
outliers_grouped_extreme <-
  filter(outliers_grouped, is.extreme == "TRUE")
# Entfernen
sleep_outliers_extreme <-
  sleepstudy[-which(sleepstudy$Reaction %in% outliers_grouped_extreme$Reaction), ]
```

Überprüfen wir die Anzahl der Datenpunkte, um sicher zu gehen, dass die richtige Anzahl an Spalten entfernt wurde:

```r
# Alle Outliers
nrow(sleepstudy) - nrow(sleep_outliers)
```

```
## [1] 8
```

```r
# Nur extreme Outliers:
nrow(sleepstudy) - nrow(sleep_outliers_extreme)
```

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

(Wir speicherten hier die um die Outlier reduzierten Daten in neuen Datensätzen, damit wir die alten nicht überschreiben. Der Einfachheit halber wird mit den Daten inklusive der Outlier weitergearbeitet.)

---

4 Plots

R bietet zahlreiche Möglichkeiten zur grafischen Darstellen von Daten (mit baseR oder zusätzlichen Paketen). Eines der am häufigsten verwendeten Pakete ist **ggplot2**, da es sich sehr frei gestalten lässt. Bis auf einige wenige Ausnahmen wird auch in dieser Guide "ggplot2" verwendet.

Zunächst erstellen wir ein neues ggplot-Element mit unserem Datensatz als Argument. Dies gilt uns als Grundlage für die Plots:

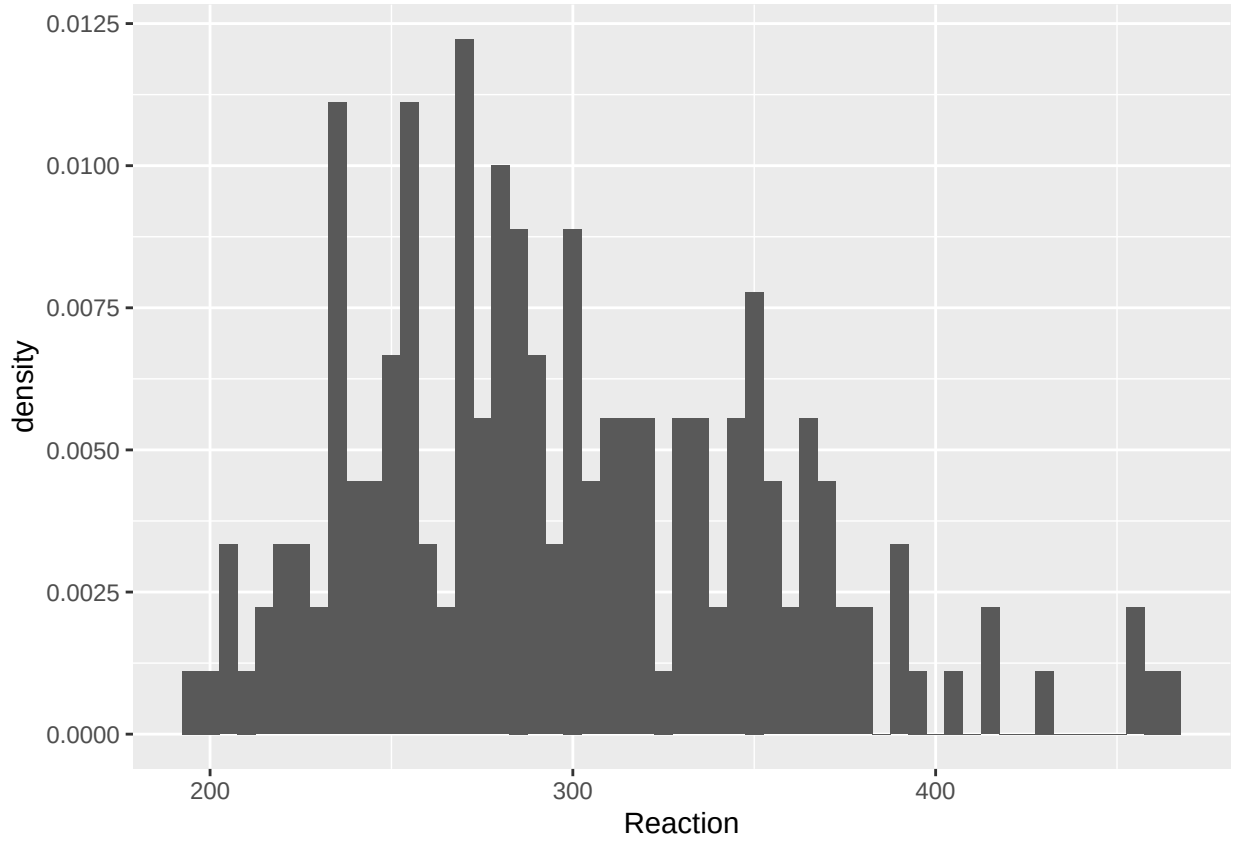
```
gg_sleep <- ggplot(sleepstudy)
```

Hinweis:
Es wäre ebenfalls möglich, bereits in der `ggplot()`-Funktion die *aesthetics* (`aes()`) zu bestimmen. So müssten wir dies nicht bei jedem "geom" neu definieren, wie wir es in den folgenden Kapiteln tun. Ich denke, dass es im Rahmen dieser Guide der praktischere Ansatz ist, die *aesthetics* hier noch nicht zu bestimmen. Somit haben wir alle Variablen der Daten in unserem `gg_sleep`-Element vorhanden. Dies ist jedoch sicher von Person zu Person und von Projekt zu Projekt unterschiedlich. Die verschiedenen Herangehensweisen scheinen unendlich und lassen viel Raum für experimentelles Herumprobieren.
In diesem Sinne: Frohes Plotten :-)

4.1 Histogramm

Um ein Histogramm mit ggplot2 zu erstellen, ergänzen wir unser ggplot-Element um die Funktion `geom_histogram()`. Als x-Aesthetic legen wir die Reaktionsdaten fest. Die y-Achse soll die relative Häufigkeit angeben. Um die absolute Häufigkeit darzustellen, würden wir "..density.." durch "..count.." ersetzen.
Mit dem Argument `binwidth =` bestimmen wir die Breite der *bins*.

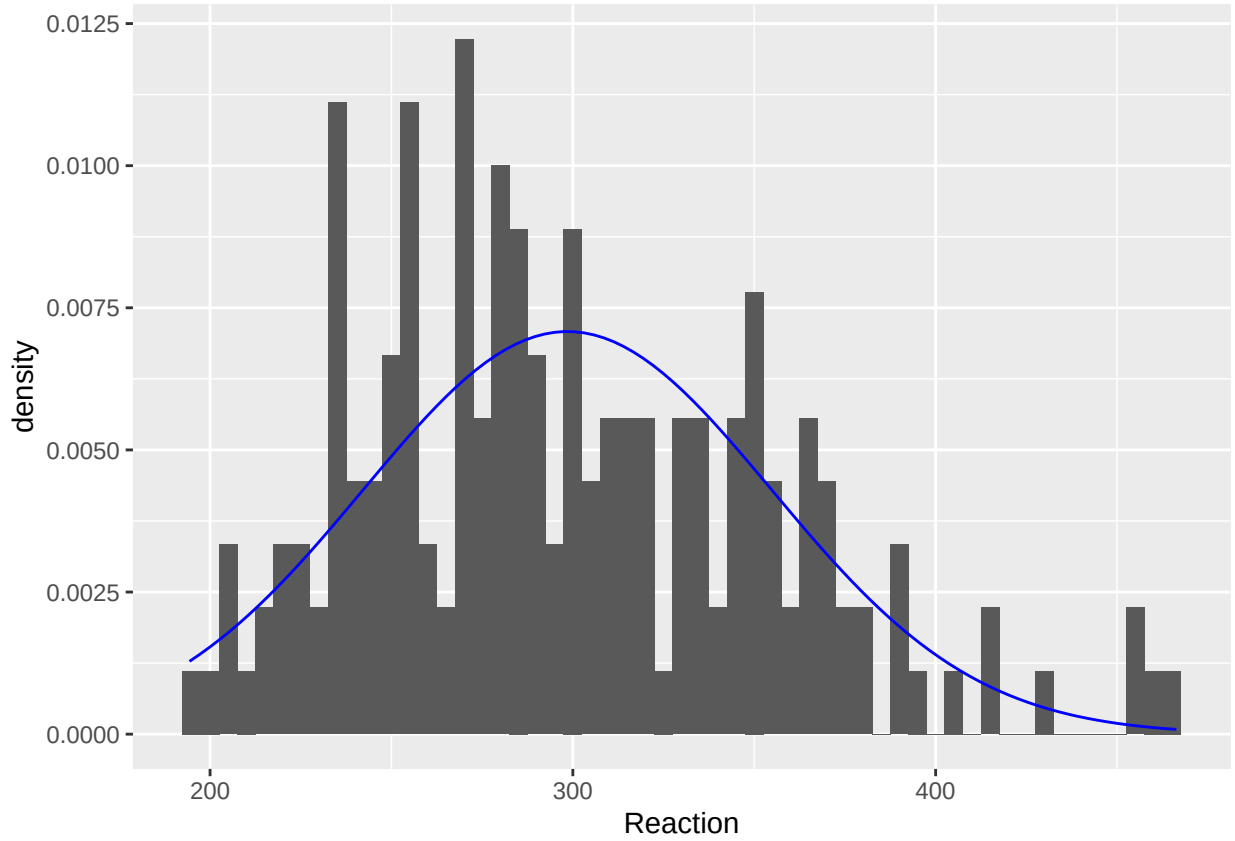
```
# Plot speichern
sleep_hist <-
  gg_sleep +
  geom_histogram(aes(x = Reaction, y = ..density..),
                 binwidth = 5
                 )
# Plot ausgeben
sleep_hist
```



4.1.1 Histogramm mit Normalverteilungskurve

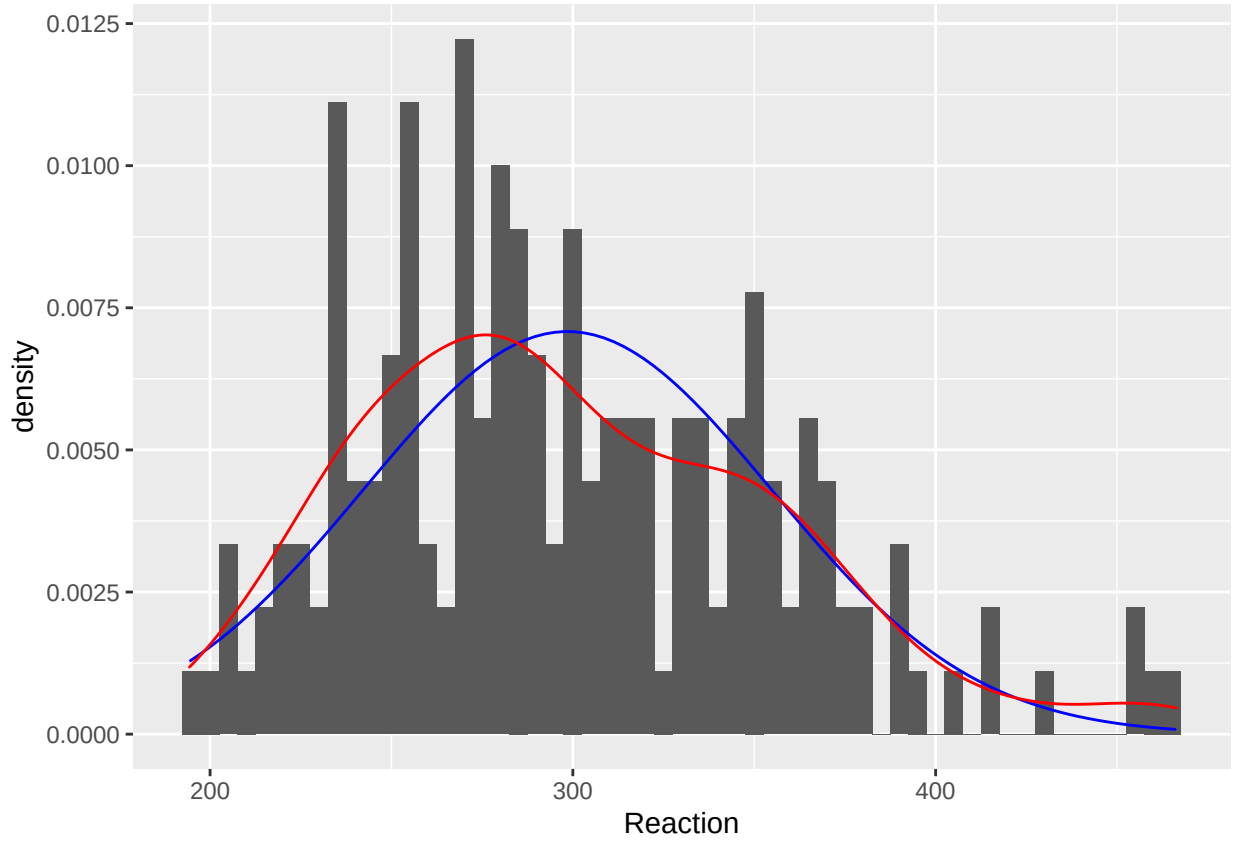
Um unserem Plot eine Normalverteilungskurve hinzuzufügen, muss das ggplot-Element um die Funktion `stat_function()` mit den Argumenten `fun = dnorm` und `args = list()` ergänzt werden. Innerhalb der `list()`-Klammer geben wir die Variable an, deren Histogramm betrachtet wird. Mit dem Argument `col =` können wir bestimmen, in Welcher Farbe die Kurve abgebildet werden soll:

```
# Normalverteilungskurve ergänzen und Plot in neuer Variable speichern
sleep_hist2 <-
  sleep_hist +
  stat_function(fun = dnorm,
                args = list(mean = mean(sleepstudy$Reaction),
                            sd = sd(sleepstudy$Reaction)),
                col = "blue"
                )
# Plotten
sleep_hist2
```



Mit der Funktion `geom_density()` kann zusätzlich eine geglättete Version des Histogramms hinzugefügt werden:

```
# Geglättete Density Kurve ergänzen und in neuer Variable speichern
sleep_hist3 <-
  sleep_hist2 +
  geom_density(aes(x = Reaction),
               col = "red"
               )
# Plotten
sleep_hist3
```



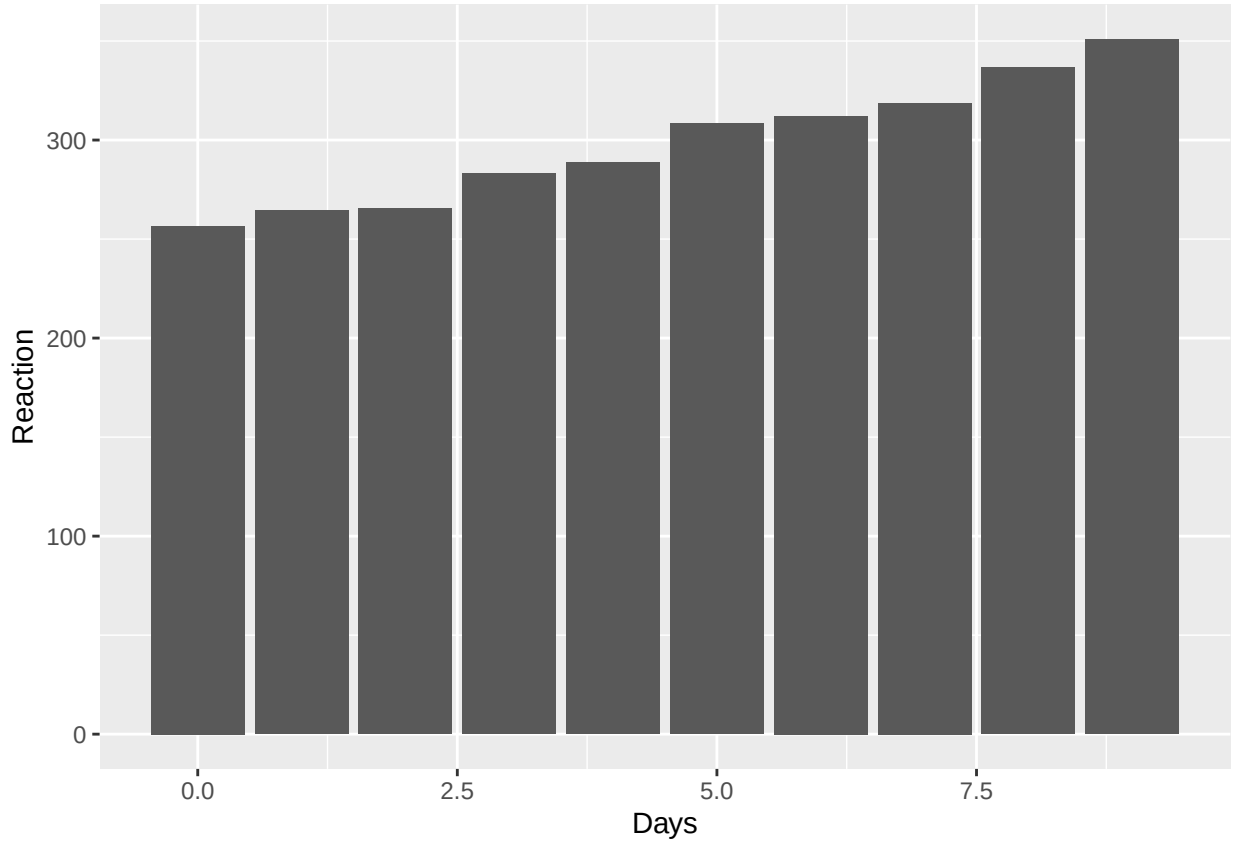
4.2 Barplots

Barplots lassen sich durch die Funktion `geom_bar()` erstellen. In der `aes()`-Klammer werden die x- und y-Achse definiert.
Das Argument `stat =` hat folgende Inputmöglichkeiten:

- "summary" -> kalkuliert standardmäßig den Mittelwert
- "identity" -> bildet die Daten ab, wie sie vorliegen
- "count" -> zählt die Anzahl der Ausprägungen

```
gg_sleep +
  geom_bar(aes(x = Days, y = Reaction),
           stat = "summary"
           )
```

```
## No summary function supplied, defaulting to `mean_se()`
```

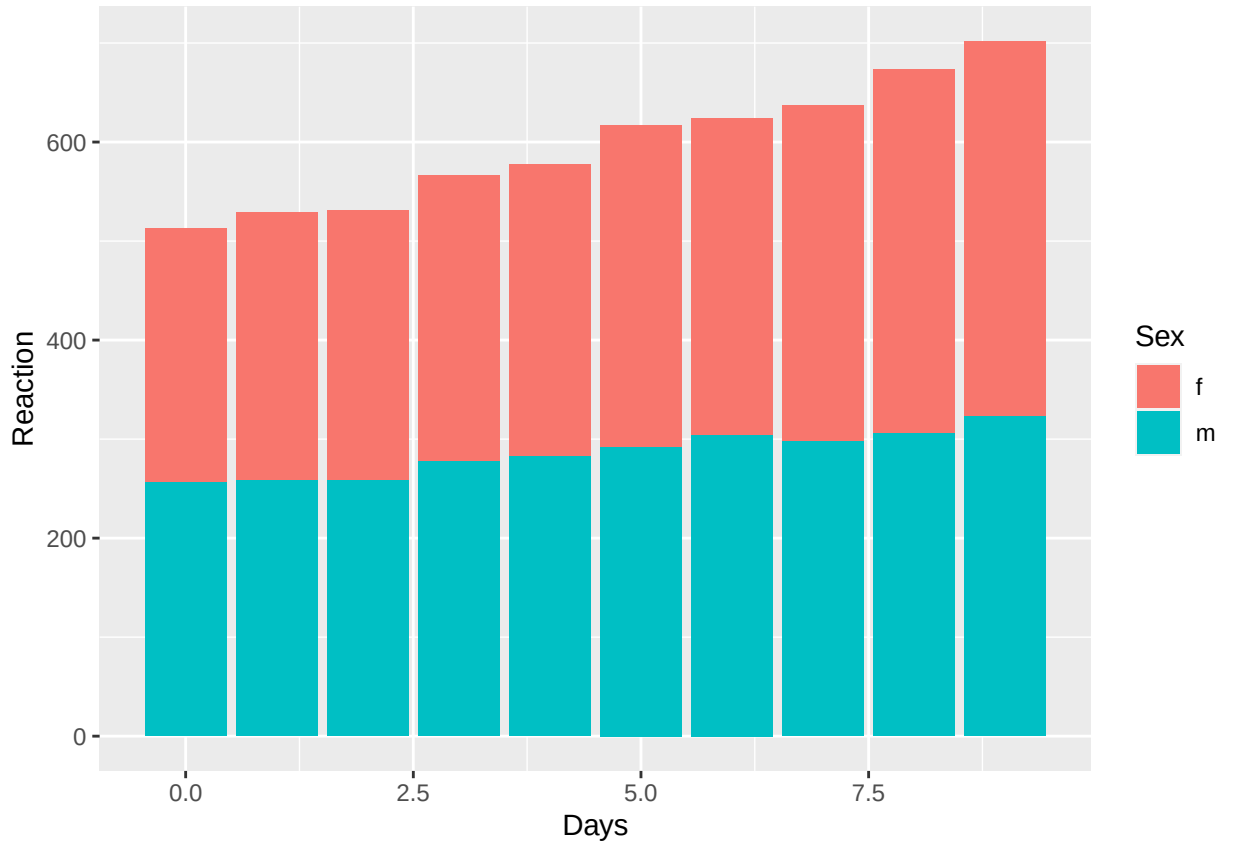


4.2.1 Gestapelte Barplots

Um die Daten gruppiert darzustellen, ergänzen wir in der `aes()`-Klammer entweder das Argument `fill =` oder `col =`. R entstellt automatisch eine Legende für die Variable, nach der gruppiert werden soll. Standardmäßig werden die Balken bei Gruppierungen gestapelt angeordnet:

```
gg_sleep +
  geom_bar(aes(x = Days, y = Reaction, fill = Sex),
           stat = "summary"
           )
```

```
## No summary function supplied, defaulting to `mean_se()`
```

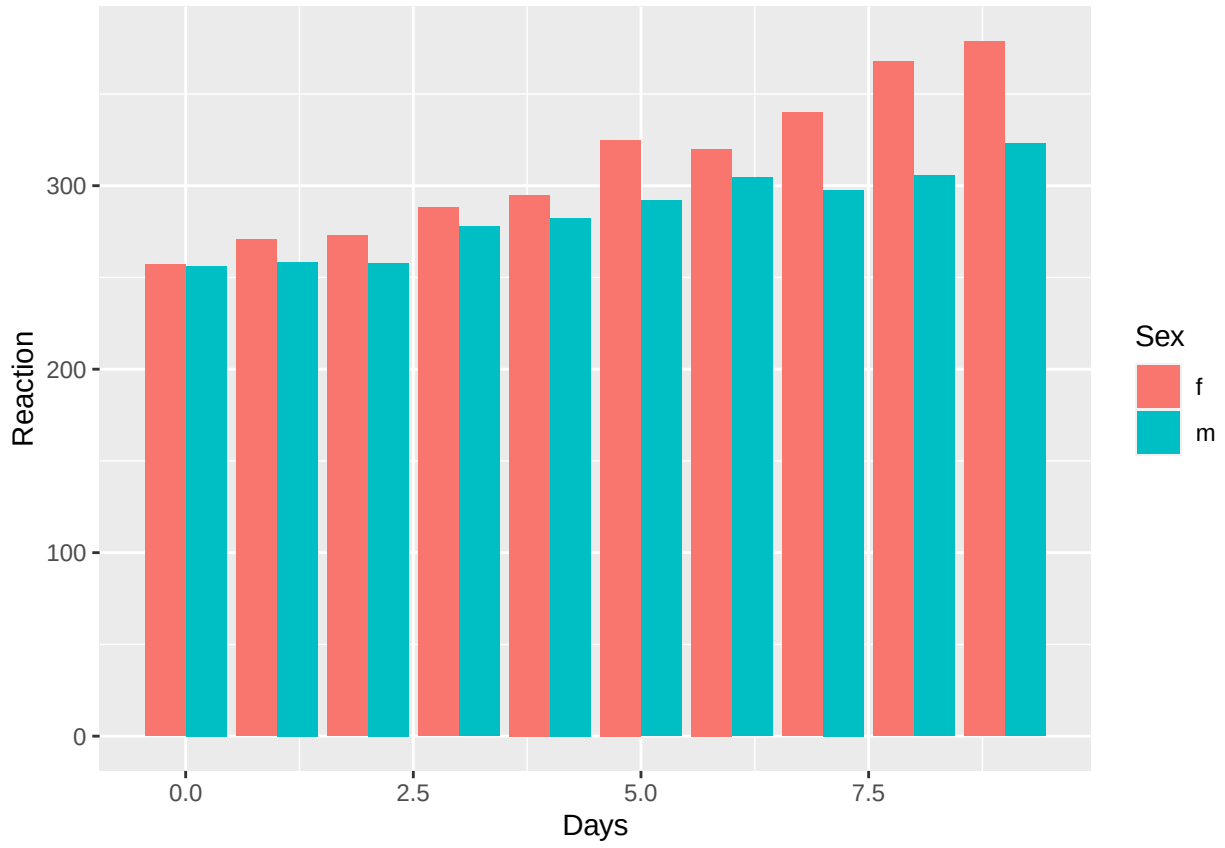


4.2.2 Nebeneinander angeordnete Barplots

Um die Balken nebeneinander anzuordnen muss `position = "dodge"` in der geom_bar()-Klammer ergänzt werden:

```
# Plot erstellen
sleep_bar <- gg_sleep +
              geom_bar(aes(x = Days, y = Reaction, fill = Sex),
                       stat = "summary",
                       position = "dodge"
                       )
# Plot ausgeben
sleep_bar
```

```
## No summary function supplied, defaulting to `mean_se()`
```



4.2.3 Barplots mit Fehlerbalken

Um barplots mit Fehlerbalken mit ggplot zu erstellen, wird eine Funktion benötigt, die nicht in einem Paket erhalten ist. Wie dies funktioniert ist einer Anleitung auf sthda.com entnommen:

```
# Erstellen der Funktion data_summary()
data_summary <- function(data, varname, groupnames){
  require(plyr)
  summary_func <- function(x, col){
    c(mean = mean(x[[col]], na.rm=TRUE),
      sd = sd(x[[col]], na.rm=TRUE))
  }
  data_sum<-ddply(data, groupnames, .fun=summary_func,
                  varname)
  data_sum <- rename(data_sum, c("mean" = varname))
 return(data_sum)
}

# Neue Datentabelle
sleep_errorbar <-
  data_summary(sleepstudy, varname = "Reaction",
               groupnames = "Days")
# Betrachten der neuen Datentabelle
head(sleep_errorbar)
```

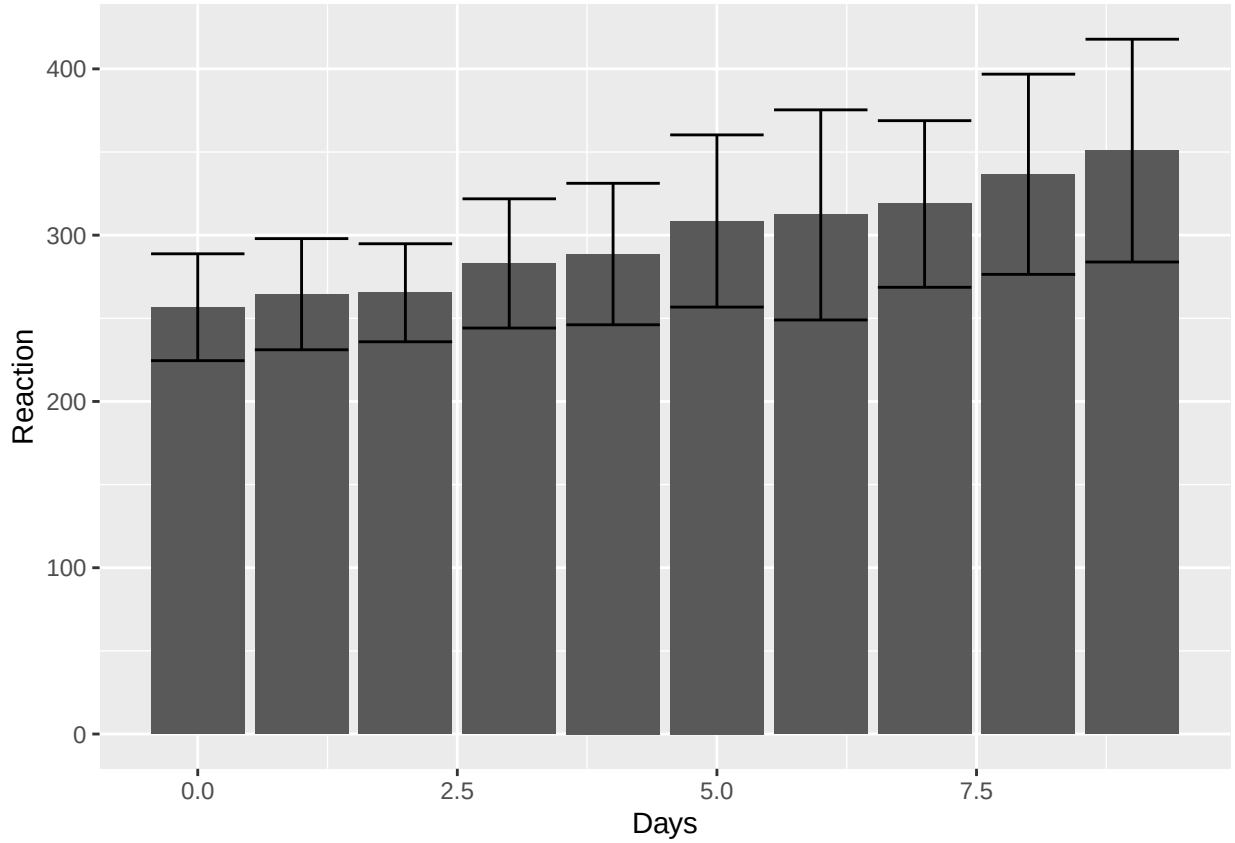
```
##   Days Reaction       sd
```

```
## 1    0 256.6518 32.12945
## 2    1 264.4958 33.43033
## 3    2 265.3619 29.47342
## 4    3 282.9920 38.85774
## 5    4 288.6494 42.53789
## 6    5 308.5185 51.76962
```

Die neu erstellte Datentabelle wird daraufhin zum Erstellen der Grafik verwendet:

```
ggplot(sleep_errorbar,
       aes(x = Days, y = Reaction)) +
  geom_bar(aes(x = Days, y = Reaction),
           stat = "summary") +
  geom_errorbar(aes(ymin = Reaction-sd, ymax = Reaction+sd))
```

```
## No summary function supplied, defaulting to `mean_se()`
```



Ein Barplot, der dies alles **kombiniert** (also gruppiert, nebeneinander & inkl. Fehlerbalken) lässt sich wie folgt erstellen:

```
# Erstellen der neuen Datentabelle inkl. der Variable "Sex"
sleep_errorbar2 <-
  data_summary(sleepstudy, varname = "Reaction",
               groupnames = c("Days", "Sex"))
# Betrachten der zweiten neuen Datentabelle:
head(sleep_errorbar2)
```

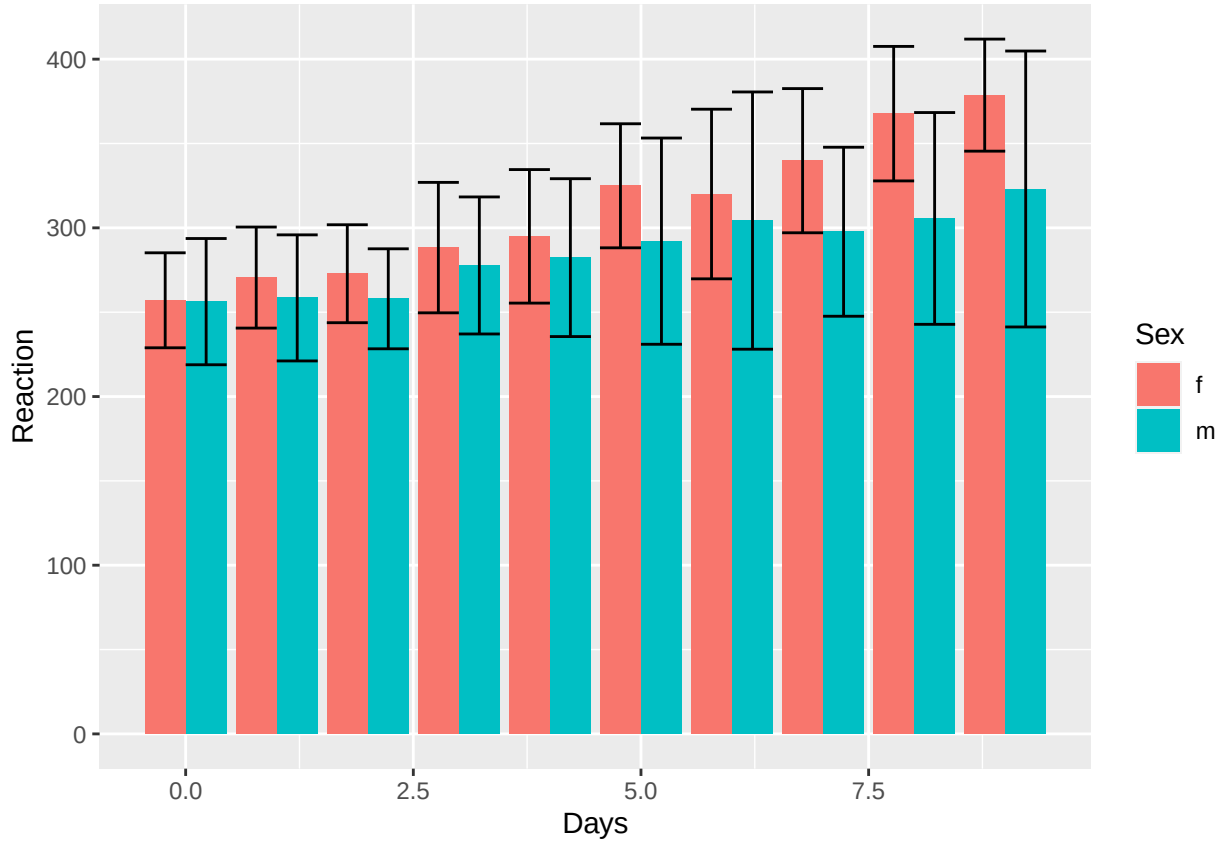
```
##   Days Sex Reaction       sd
## 1    0   f 257.0500 28.15595
```

```
## 2    0   m 256.2536 37.42361
## 3    1   f 270.5228 29.94441
## 4    1   m 258.4687 37.36940
## 5    2   f 272.7947 29.02961
## 6    2   m 257.9291 29.64669
```

Danach plotten:

```
ggplot(sleep_errorbar2,
       aes(x = Days, y = Reaction, fill = Sex)) +
  geom_bar(aes(x = Days, y = Reaction, fill = Sex),
           stat = "summary",
           position = "dodge") +
  geom_errorbar(aes(ymin = Reaction-sd, ymax = Reaction+sd),
                position = "dodge")
```

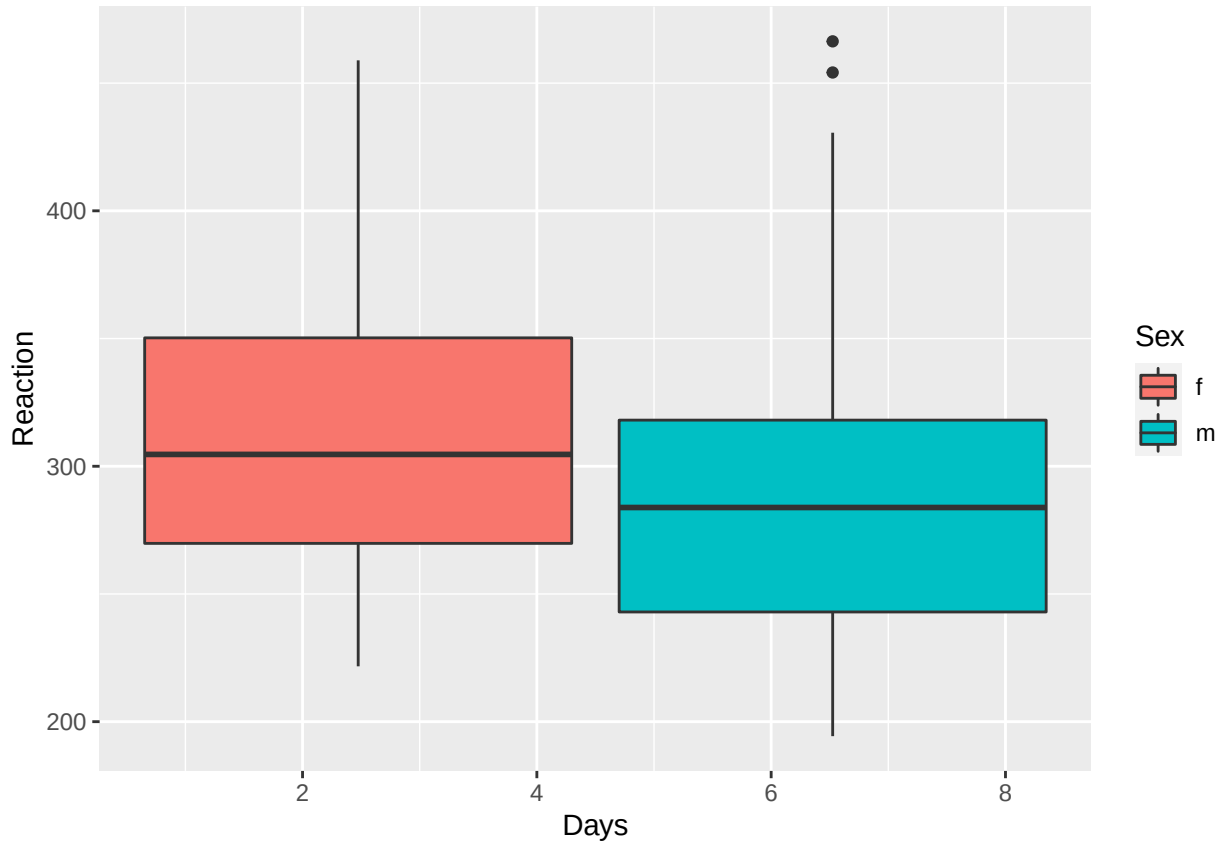
```
## No summary function supplied, defaulting to `mean_se()`
```



4.3 Boxplots

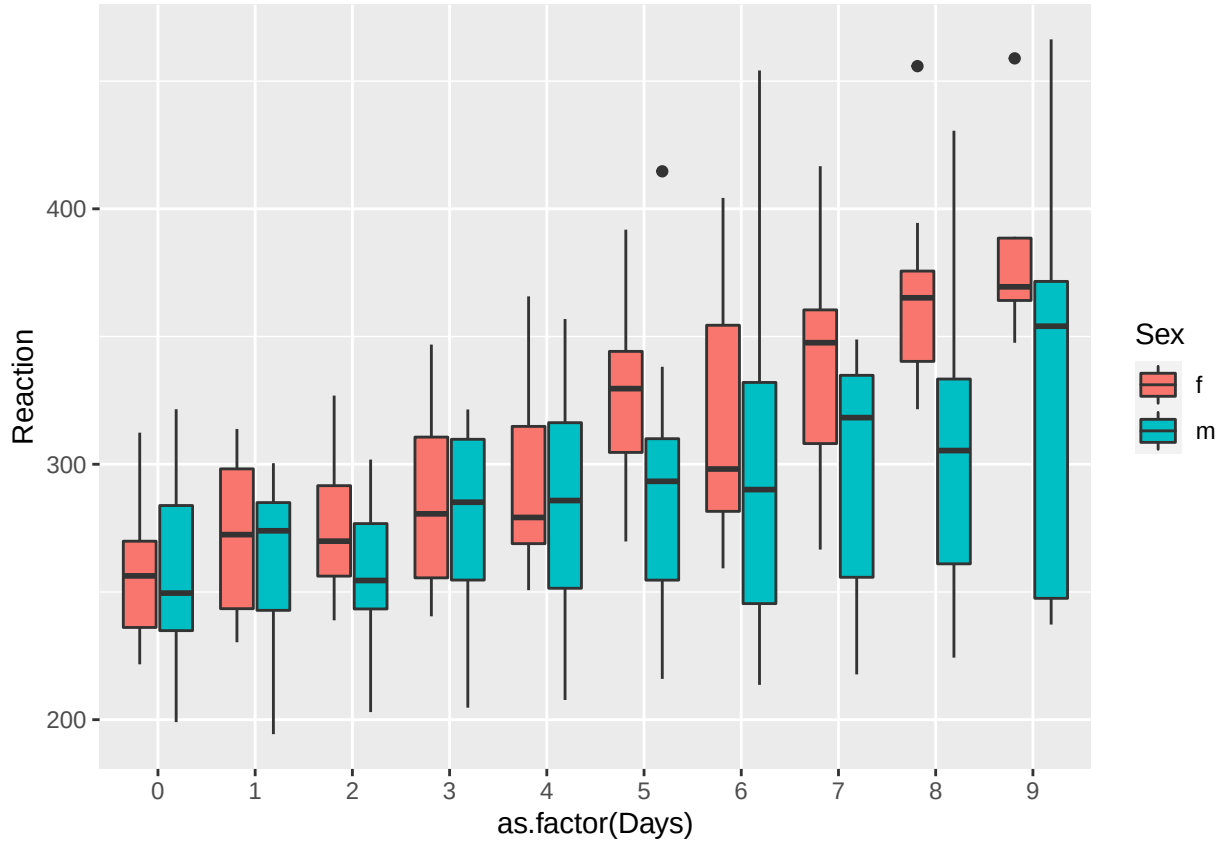
Boxplots werden in ggplot mit der Funktion `geom_boxplot()`erstellt:

```
gg_sleep +
  geom_boxplot(aes(x = Days, y = Reaction, fill = Sex))
```



Dieser Plot stellt nicht das dar, was wir wollten, da "Days" als numerische Variable konfiguriert ist. Mit dem Befehl `as.factor()` oder `as.character()` lässt sich dies korrigieren:

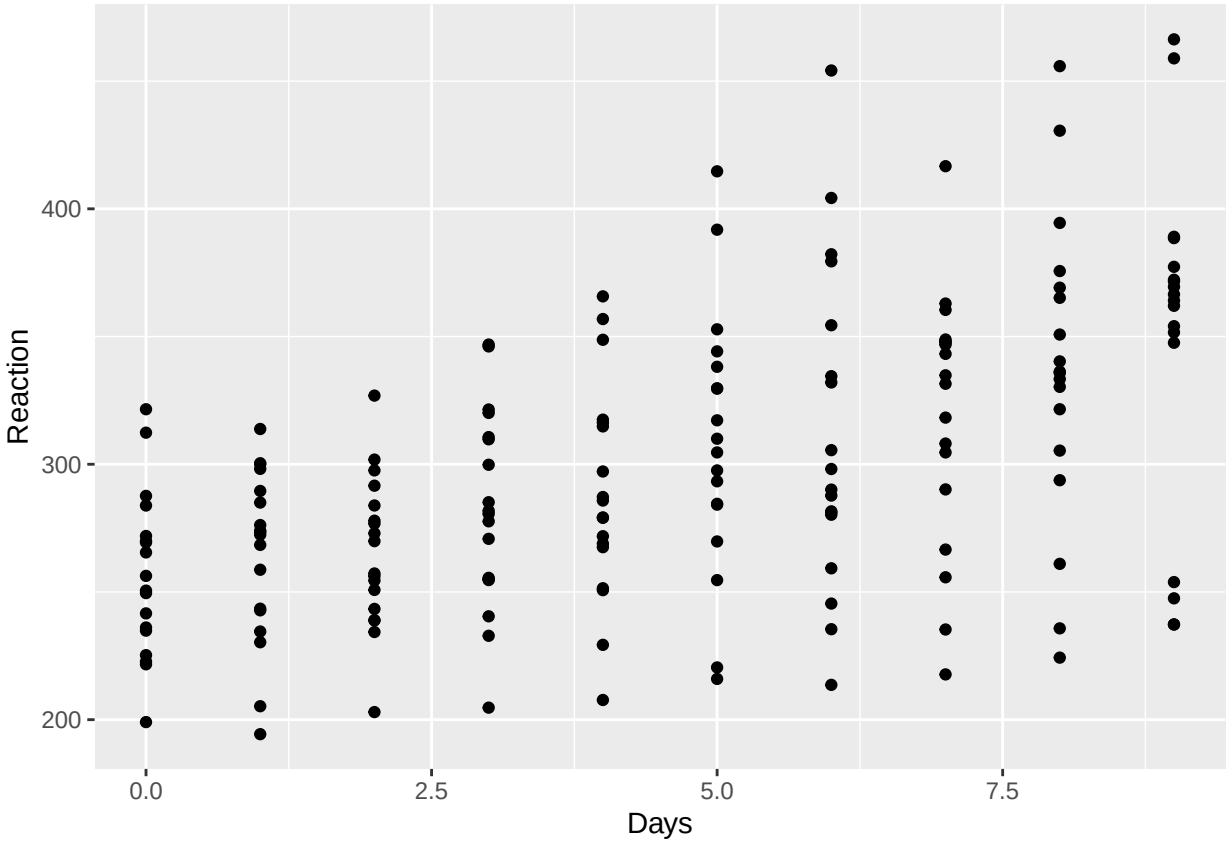
```
gg_sleep +
  geom_boxplot(aes(x = as.factor(Days), y = Reaction, fill = Sex))
```



4.4 Scatterplots

Für Scatterplots wird die Funktion `geom_point()` verwendet:

```
# Plot erstellen
sleep_point <- gg_sleep +
  geom_point(aes(x = Days, y = Reaction))
# Plotten
sleep_point
```

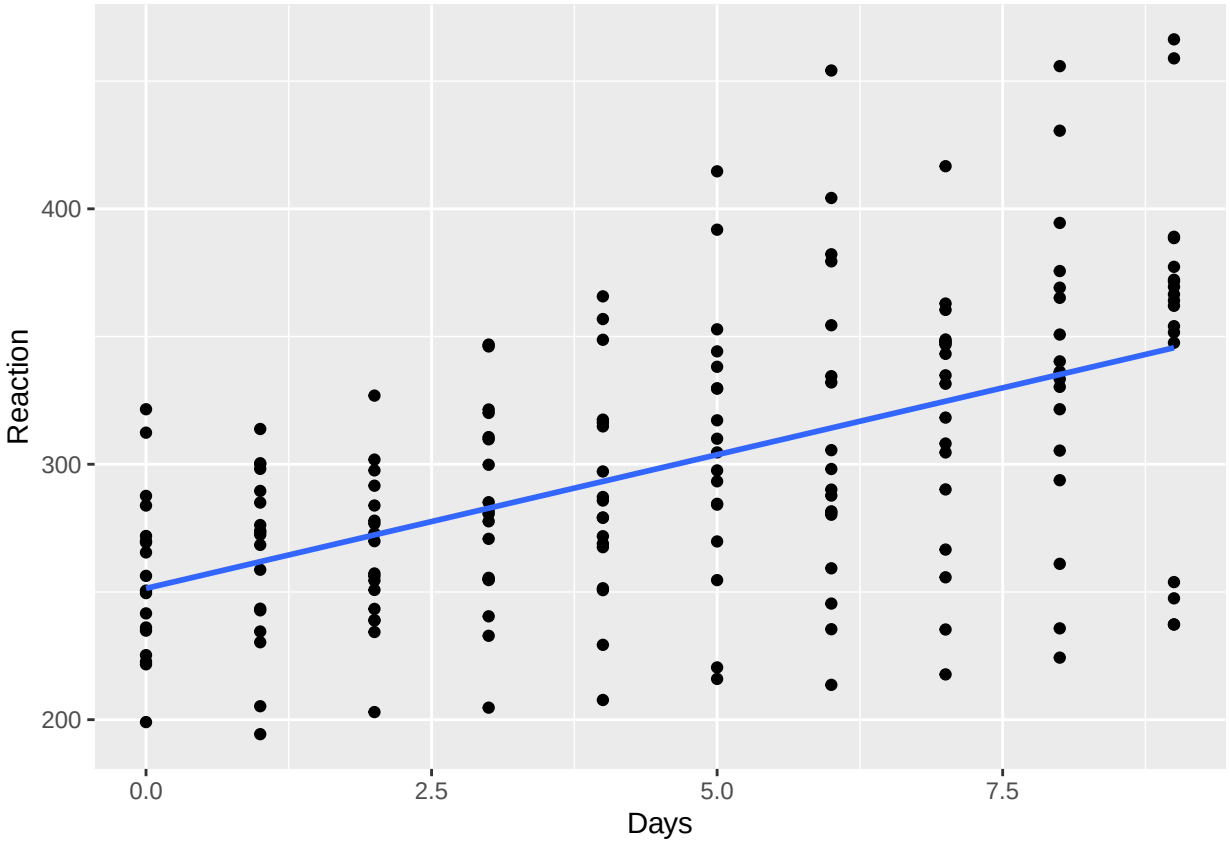


4.4.1 Regressionsgeraden

Um eine Regressionsgerade zu ergänzen, wird die Funktion `geom_smooth()` verwendet.
Ein Blick in die *Help-Page* mittels `?geom_smooth` liefert weitere Informationen zu den Argumenten `method =` und `se =`.

```
sleep_point +
  geom_smooth(aes(x = Days, y = Reaction),
              method=lm,
              se = F
              )
```

```
## `geom_smooth()` using formula 'y ~ x'
```

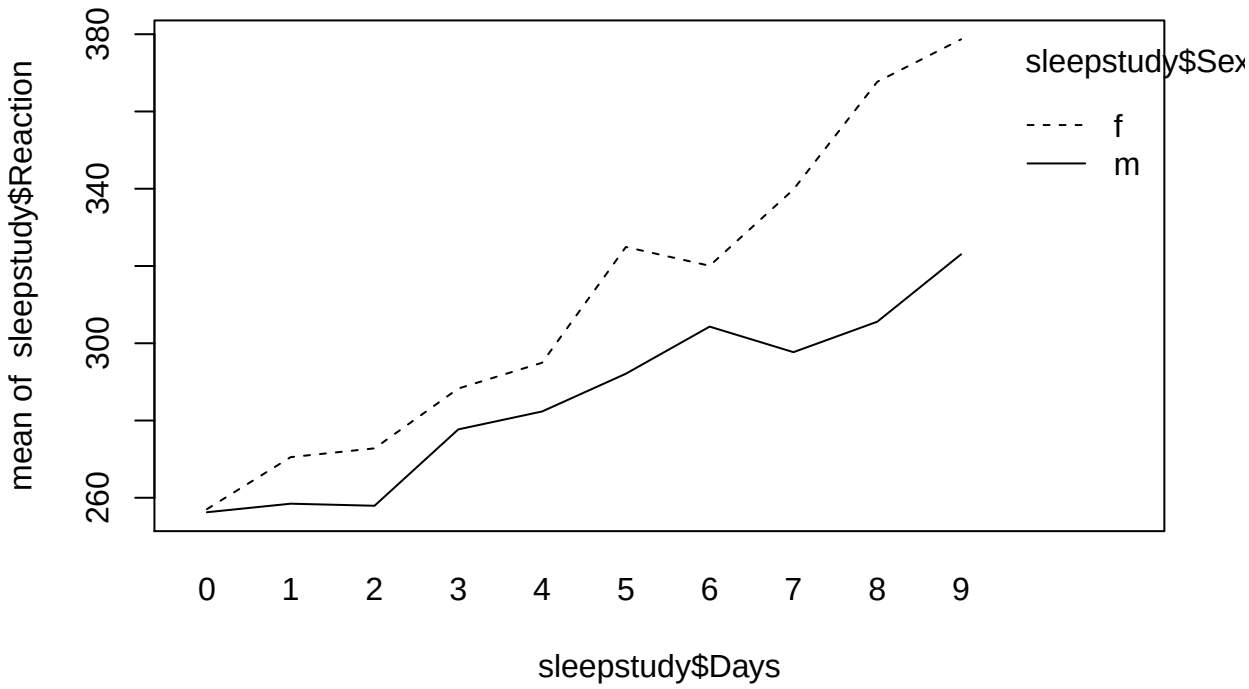


Auch hier ließen sich mit dem Argument `fill =` gruppierte Regressionsgeraden erstellen.

4.4.2 Interaktionsplot

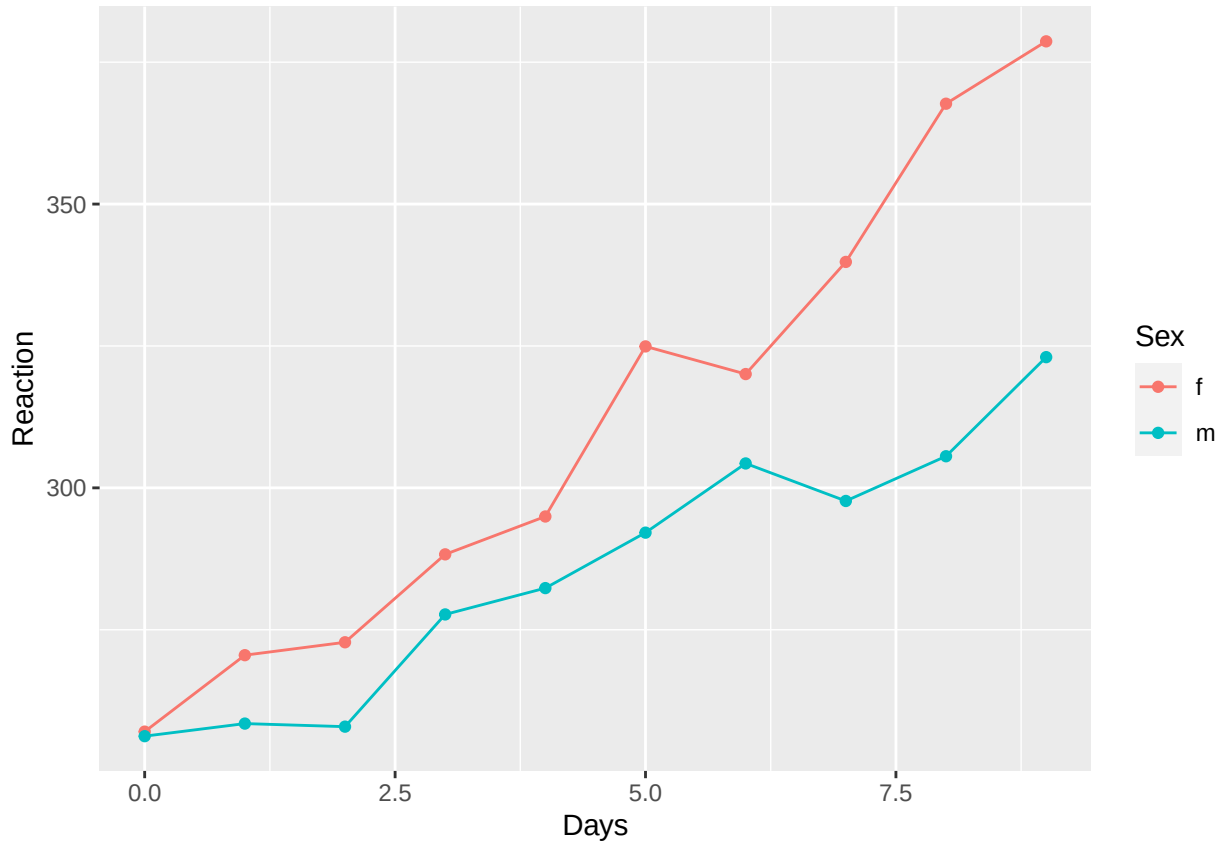
Ein Interaktionsplot lässt sich einfach mit *baseR* erstellen. Dazu werden die benötigten Variablen in folgender Reihenfolge als Argumente der Funktion `interaction.plot()` geschrieben:
`interaction.plot(x-Achse, Gruppierung, y-Achse)`

```
interaction.plot(sleepstudy$Days, sleepstudy$Sex, sleepstudy$Reaction)
```



Mit ggplot sind Interaktionsplots ein wenig komplizierter. Benötigt wird die Funktion `geom_line()`. Wichtig ist hierbei, dass die Argumente `stat = "summary"`und `fun = "mean"` ergänzt werden, sowie die Gruppierung auch mit dem Argument `group =` gekennzeichnet wird.
Die Funktion `geom_point()` kann optional verwendet werden, um Punkte an den unterschiedlichen Ausprägungen der unabhängigen Variable auf der x-Achse anzuzeigen:

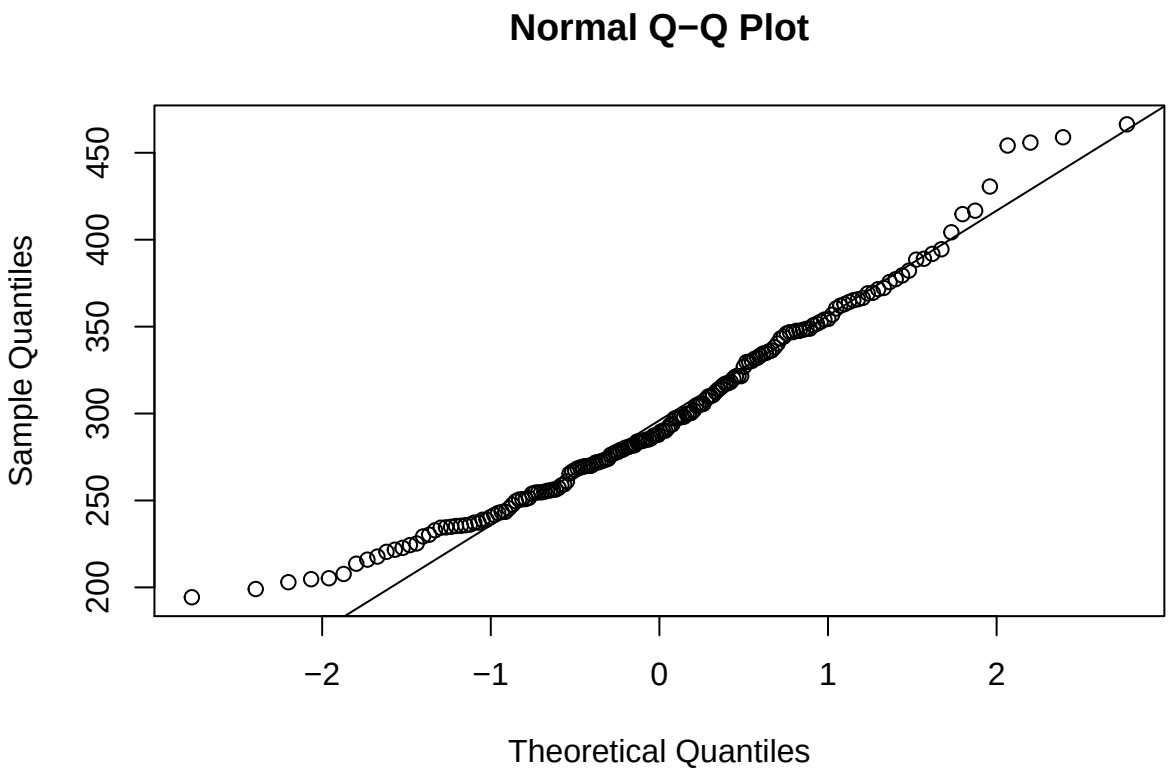
```
gg_sleep +
  geom_line(aes(x = Days, y = Reaction, col = Sex, group = Sex),
            stat = "summary", fun = "mean"
            ) +
  geom_point(aes(x = Days, y = Reaction, col = Sex),
            stat = "summary", fun = "mean"
            )
```



4.5 Q-Q-Plot

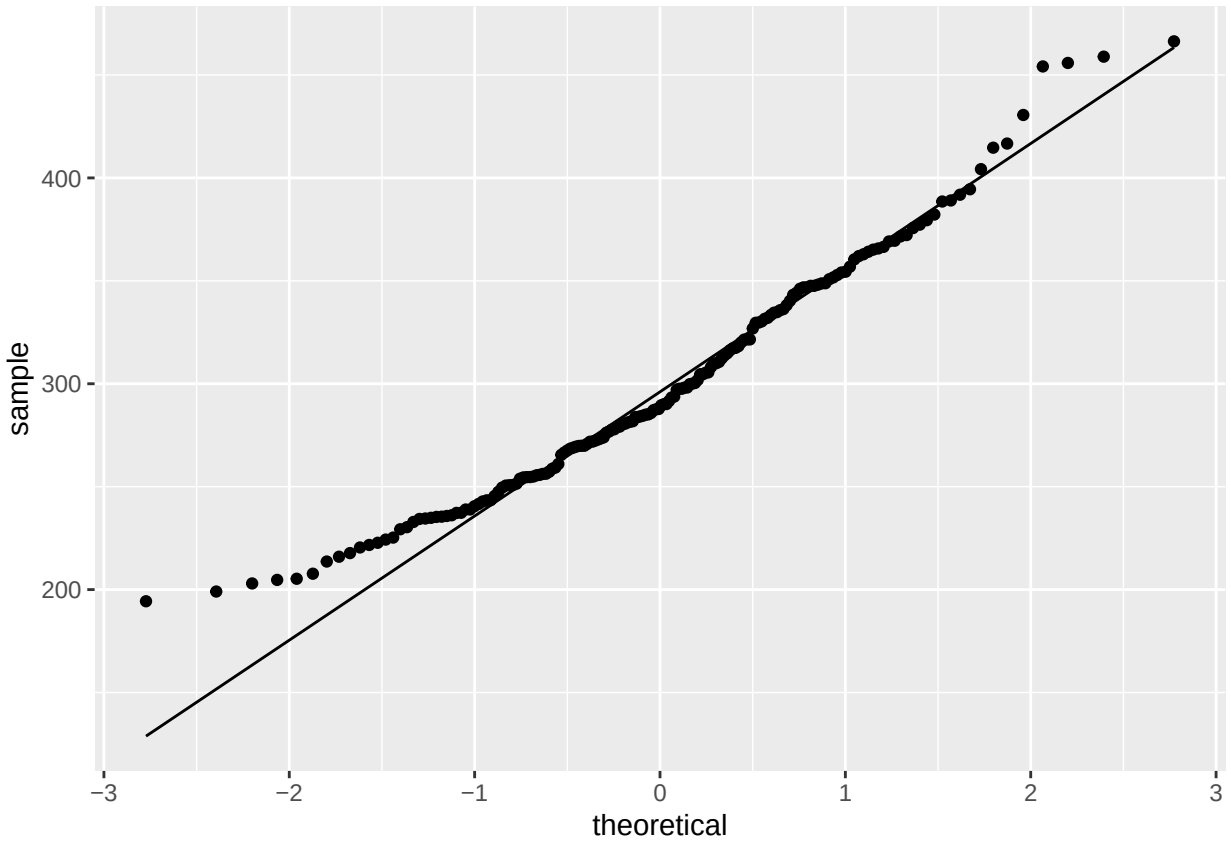
Mit *baseR*:

```
qqnorm(sleepstudy$Reaction)    # Punkte
qqline(sleepstudy$Reaction)    # Linie
```



Mit *ggplot*:

```
gg_sleep +
  stat_qq(aes(sample = Reaction)) +    # Punkte
  stat_qq_line(aes(sample = Reaction)) # Linie
```



4.5.1 Textplots

Um Text zu plotten werden die Funktionen `geom_text` oder `geom_label` verwendet. Der Unterschied besteht darin, dass letztere zusätzlich einen Kasten um den Text generiert. Innerhalb der `aes()`-Klammer wird das Argument `label =` benötigt, welches definiert, aus welcher Variable der Text entnommen werden soll.

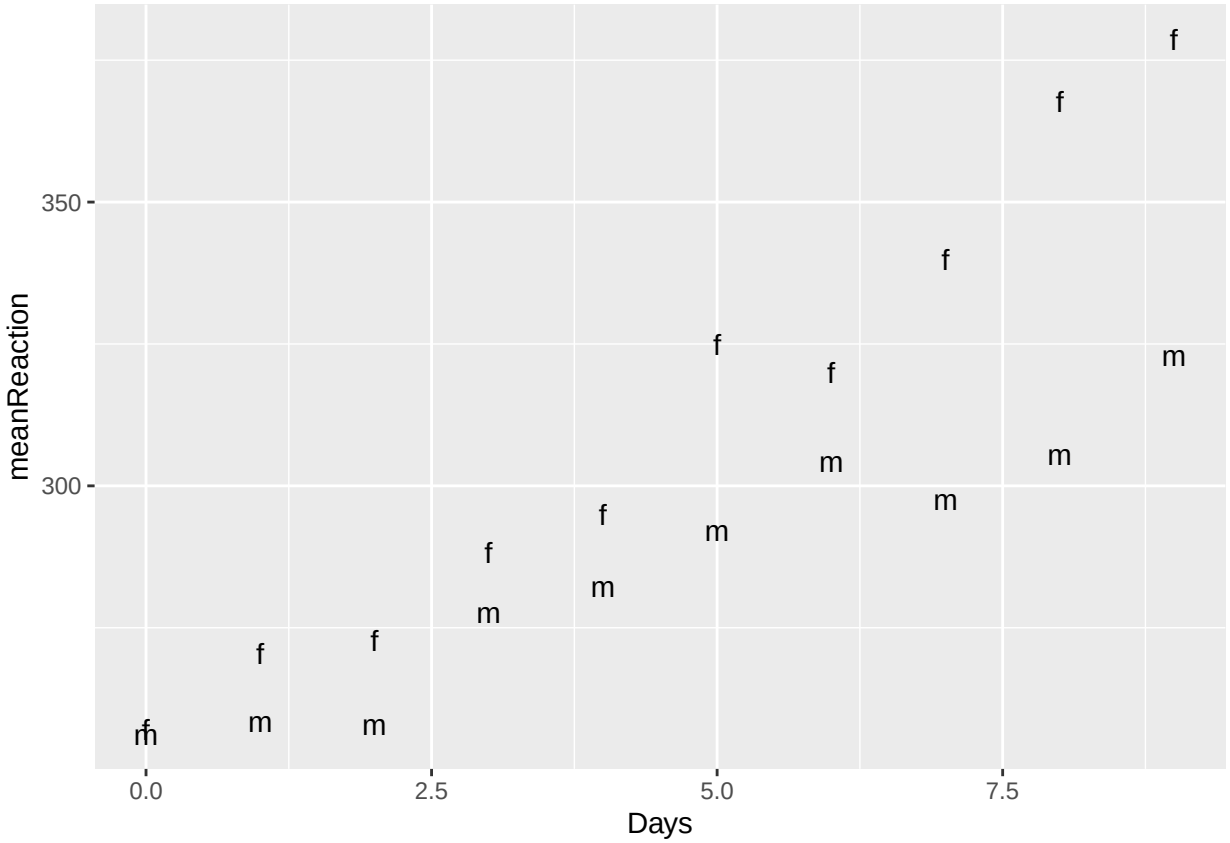
Anmerkung:
Da es sich beim Datensatz `sleepstudy` um Messwiederholungsdaten handelt, wäre eine Darstellung der Datenpunkte mit Text sehr unübersichtlich. Wir generieren also zunächst eine neue Tabelle, die den Mittelwert der Reaktionsszeit nach Tag und Geschlecht berechnet.
Die Notation `dplyr::` wird verwendet, damit R weiß, dass die Funktionen aus dem "dplyr"-Paket verwendet werden, nicht die gleichnamigen aus dem "plyr"-Paket.

```
# Tabelle täglicher means
daily_means <- sleepstudy %>%
  dplyr::group_by(Days, Sex) %>%      # Gruppiert nach Tage und Geschlecht
  dplyr::summarise(meanReaction = mean(Reaction))  # mean der Reaktionszeit berechnen
```

```
## `summarise()` regrouping output by 'Days' (override with `.groups` argument)
```

```
                                                  # und in neuer Variable speichern

# Plotten
ggplot(daily_means) +
  geom_text(aes(x = Days, y = meanReaction, label = Sex))
```



4.6 Weitere Funktionen für Plots

In diesem Kapitel wird dargestellt, wie *ggplot2*-Plots modifiziert werden können.
Hierzu werden die Plots `sleep_bar` aus Kapitel Nebeneinander angeordnete Barplots und X verwendet.
Die hier gezeigten Funktionen stellen nur einen kleinen Ausschnitt dessen dar, was mit *ggplot2* möglich ist. Außerdem gilt auch hier, dass es oft viele verschiedene Herangehensweisen gibt. Eine umfassende Guide gibt es unter anderem auf sthda.com.

Achtung: Eine Erweiterung des Plots in einzelnen Schritten eignet sich hier zur Darstellung der Funktionen. In der Praxis kann dies zu Problemen führen, da sich einige Funktionen gegenseitig überschreiben (z.B. `scale_x_continuous` und `scale_x_reverse`, siehe Kapitel Achsen- und Legendenbeschriftung und Weitere (grafische) Transformationen).

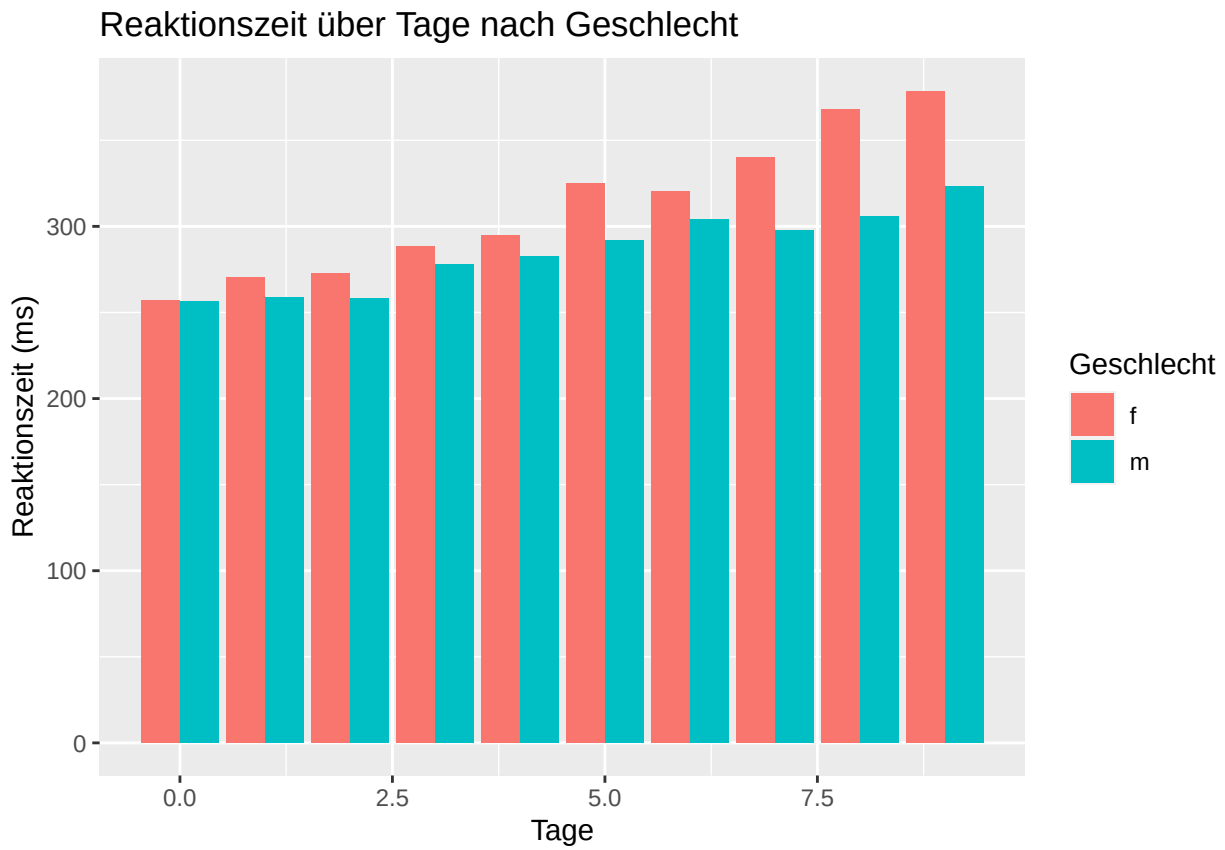
4.6.1 Titel

Durch die Funktion `labs()` können der **Haupttitel** (-> `title =`), die **Achsentitel** (-> `x =` & `y =`) und **Legendentitel** (-> `fill =` & `colour =`) definiert werden:

```
# Plot erstellen
sleep_bar_title <- sleep_bar +
                   labs(title = "Reaktionszeit über Tage nach Geschlecht",
                        x = "Tage",
                        y = "Reaktionszeit (ms)",
                        fill = "Geschlecht")
```

```r
# Plot ausgeben
sleep_bar_title
```

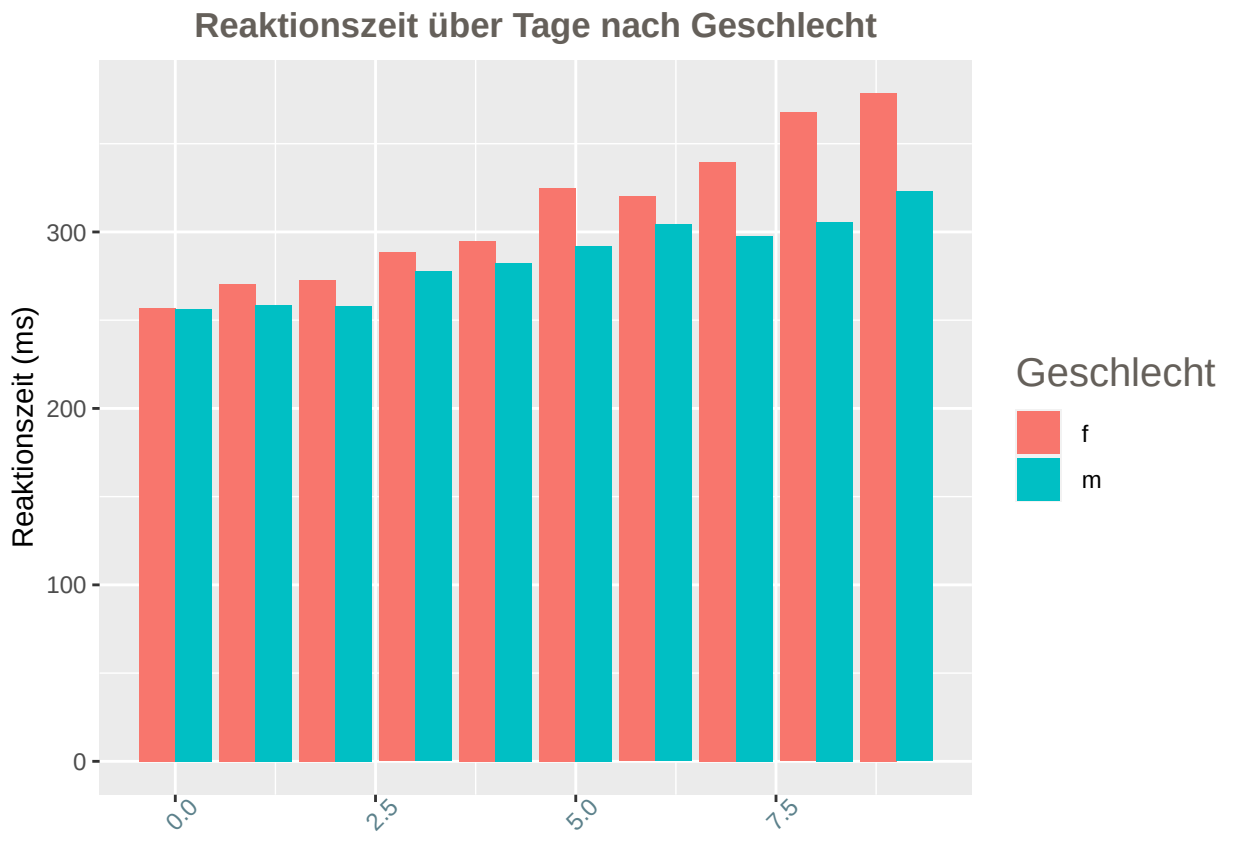
```
## No summary function supplied, defaulting to `mean_se()`
```



Die Form der Titel kann mit der Funktion `theme()` verändert werden. Ein Blick in die Help-Page (mit `?theme`) liefert eine Übersicht der zahlreichen Möglichkeiten und deren Anwendungen.
Ein Beispiel:

```r
# Plot erstellen
sleep_bar_title2 <- sleep_bar_title +
                    theme(plot.title = element_text(colour = "#65605a", # Farbe
                                                    face = "bold",      # fett
                                                    hjust = 0.5),       # zentrieren
                          axis.title.x = element_blank(),               # Titel entfernen
                          axis.text.x = element_text(angle = 45,        # Rotation
                                                     colour = "#5c818b"), # Farbe
                          legend.title = element_text(size = 15,        # Größe
                                                      colour = "#65605a") # Farbe
                          )
# Plot ausgeben
sleep_bar_title2
```

```
## No summary function supplied, defaulting to `mean_se()`
```



4.6.2 Achsen- und Legendenbeschriftung

Die Beschriftungen der Achsen- und Legendeneinheiten lassen sich durch die Funktionen mit Präfix `scale_` bearbeiten.
Numerische Variablen erfordern das Suffix `_conituous`, Faktoren und *Characters* das Suffix `_discrete`.

Dementsprechend können wir die x-Achse, auf der die numerische Variable "Days" liegt mit der Funktion `scale_x_continuous` bearbeiten.
Mit dem Argument `breaks =` geben wir an, an welchen Zeitpunkten eine Achsenbeschriftung vorhanden sein soll. Mit dem Argument `labels =` können diese noch eine neue Beschriftung erhalten.

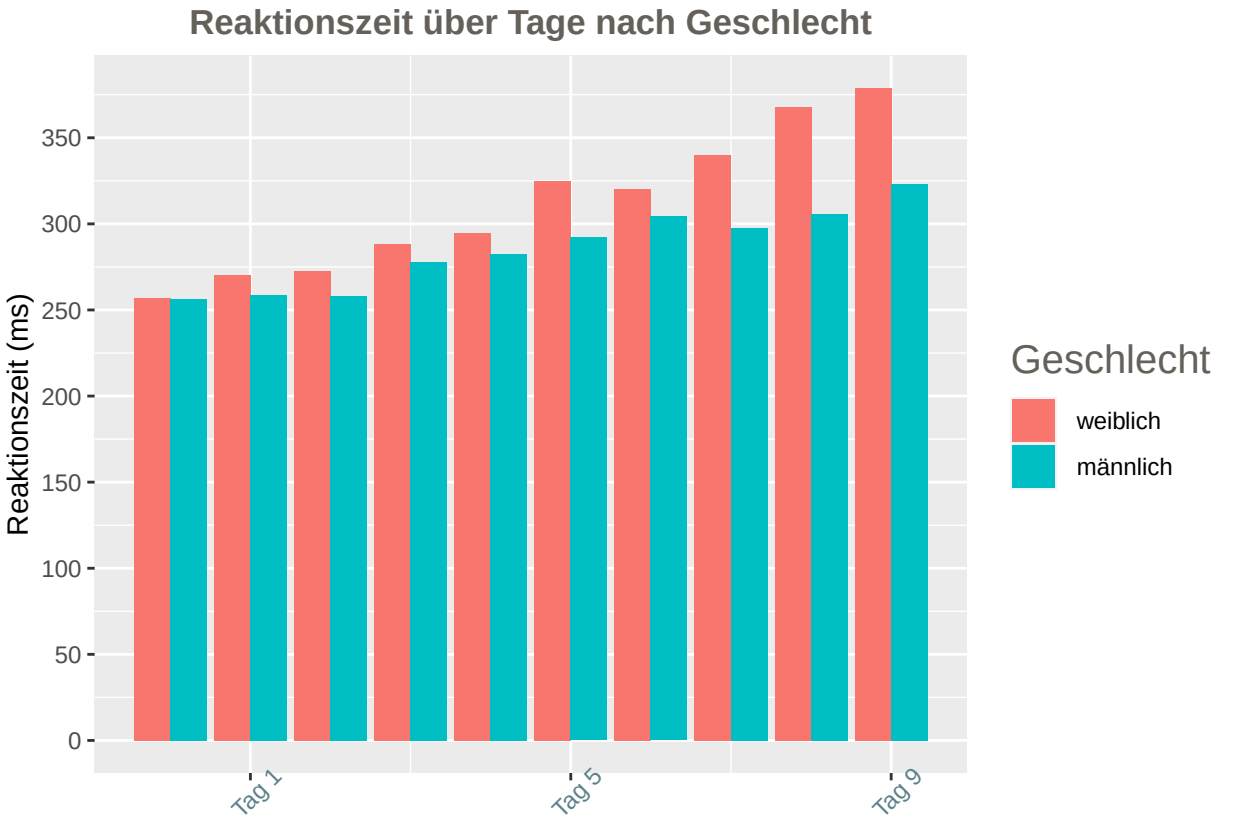
Die *breaks* der y-Achse definieren wir durch die Funktion `seq`, wobei die erste und zweite Zahl den Bereich markieren, für welchen die Beschriftung im Abstand der dritten Zahl erfolgen.

Die Legende, welche wir durch `fill = Sex` definiert haben erfordert für den Faktor "Sex" die Funktion `scale_fill_discrete`.

```
# Plot erstellen
sleep_bar_axis <- sleep_bar_title2 +
  scale_x_continuous(breaks = c(1, 5, 9),                    # Breaks bestimmen
                     labels = c("Tag 1", "Tag 5", "Tag 9")) +  # Breaks umbenennen
  scale_y_continuous(breaks = seq(0, 500, 50)) +             # Breaks alle 50 ms
  scale_fill_discrete(labels = c("weiblich", "männlich"))    # Ausprägungen umbenennen

# plotten
sleep_bar_axis
```

```
## No summary function supplied, defaulting to `mean_se()`
```



R ordnet *Factors* und *Characters* automatisch alphabetisch. Wenn diese auch in der Tabelle anders angeordnet sein sollen, müsste das bereits vor der Erstellung des *ggplot*-Objekts (in unserem Fall `gg_sleep`) wie folgt passieren:

```
sleepstudy$Sex <- factor(sleepstudy$Sex, levels = c("m", "f"))
```

4.6.3 Farben

Die Farben des Plots können mithilfe der folgenden Funktionern verändert werden:

- `scale_fill_manual` (wenn wir die Gruppierung über `fill =` erstellt haben) oder
- `scale_color_manual` (wen wir die Gruppierung über `colour =` erstellt haben)

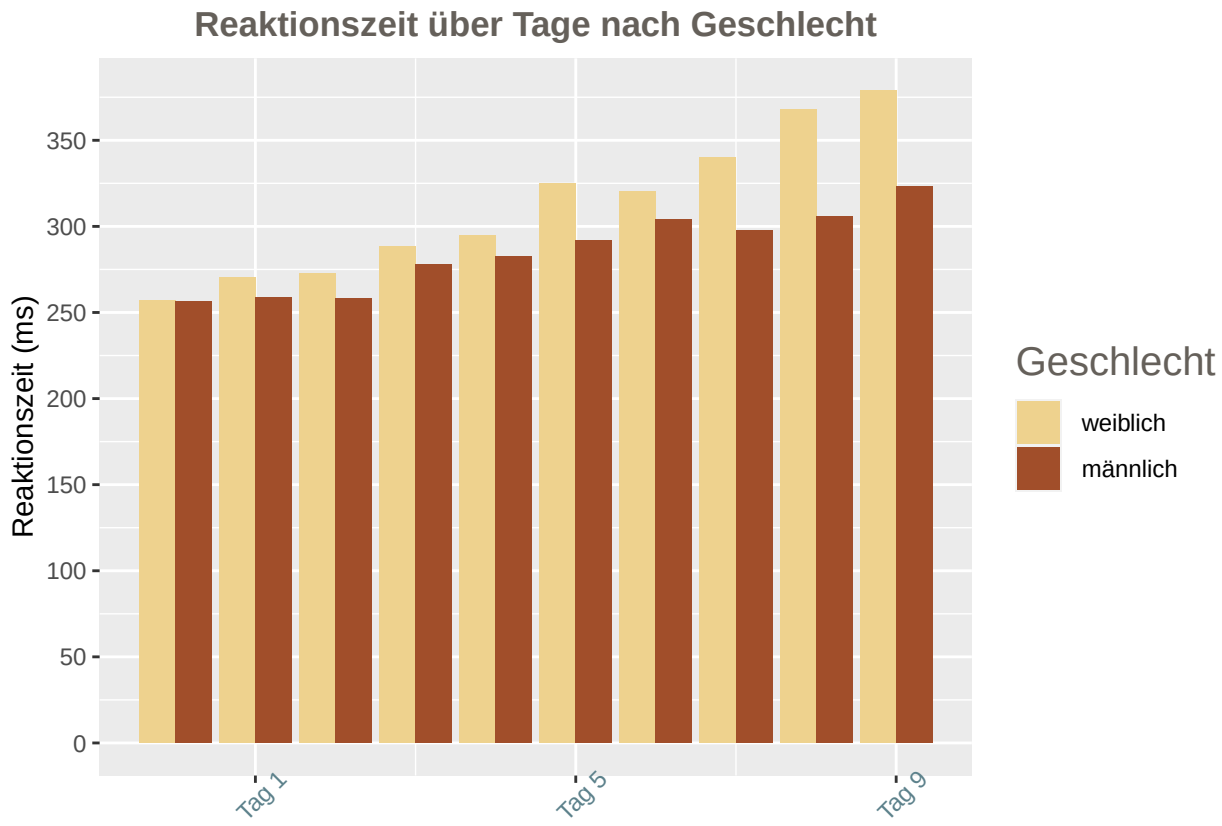
Damit die Legendenbeschriftungen "weiblich" und "männlich" nicht überschrieben werden, definieren wir sie auch hier in der `scale_fill_manual`-Klammer. Täten wir dies nicht, würden sie wieder in ihrem Ausgangsformat ("f", "m") erscheinen.

```
# Plot erstellen
sleep_bar_col <- sleep_bar_axis +
  scale_fill_manual(labels = c("weiblich", "männlich"),      # Labels
                    values = c("#eed28e", "#a14e2a")         # Farben
                    )
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```

```
# Plotten
sleep_bar_col
```

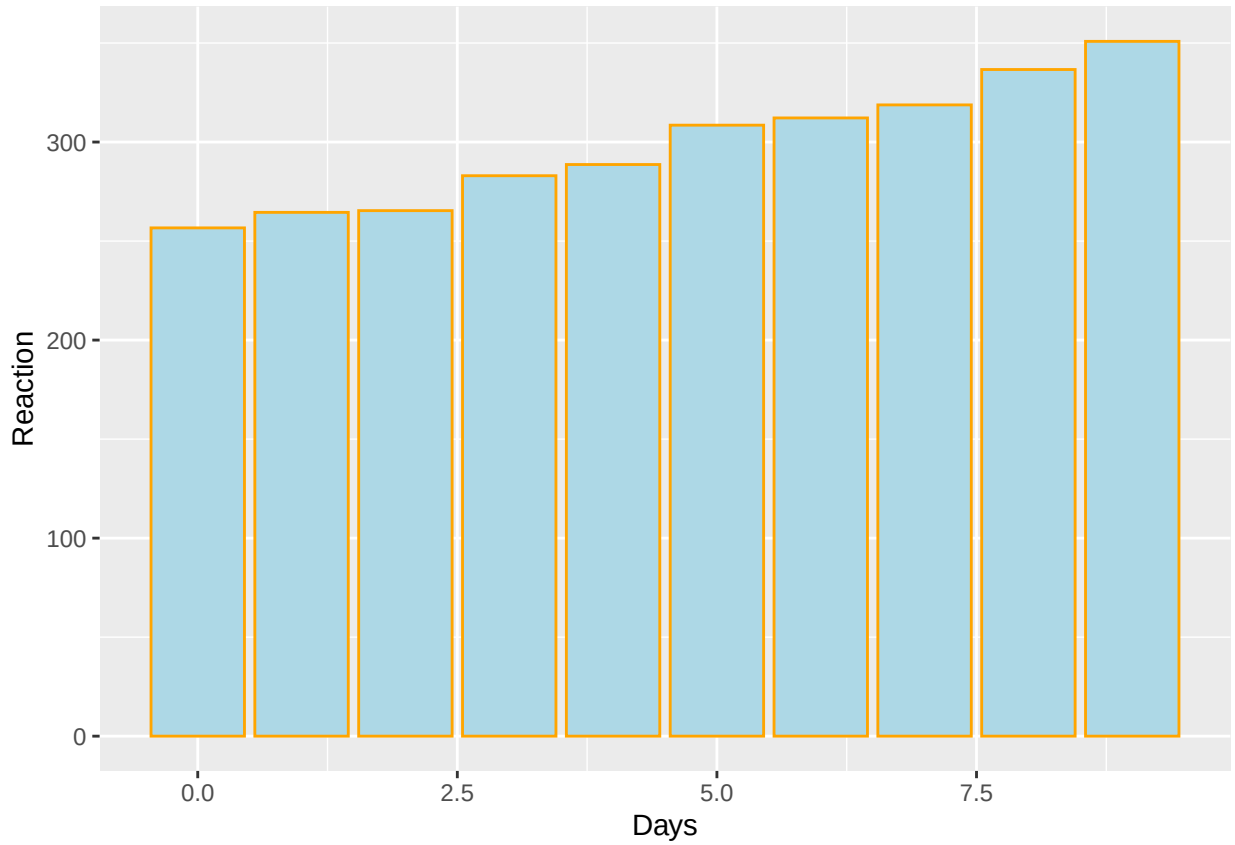
```
## No summary function supplied, defaulting to `mean_se()`
```



Ungruppierte Plots lassen sich durch die Argumente `fill =` und `col =` färben. Diese Argumente stehen **nicht** in der `aes()`-Klammer!
Ein Beispiel, wie wir den Plot aus dem Kapitel Barplots ergänzen könnten:

```
gg_sleep + geom_bar(aes(x = Days, y = Reaction),
                    stat = "summary",
                    fill = "lightblue",     # Füllung
                    col = "orange"          # Rand
                    )
```

```
## No summary function supplied, defaulting to `mean_se()`
```



4.6.3.1 Paletten

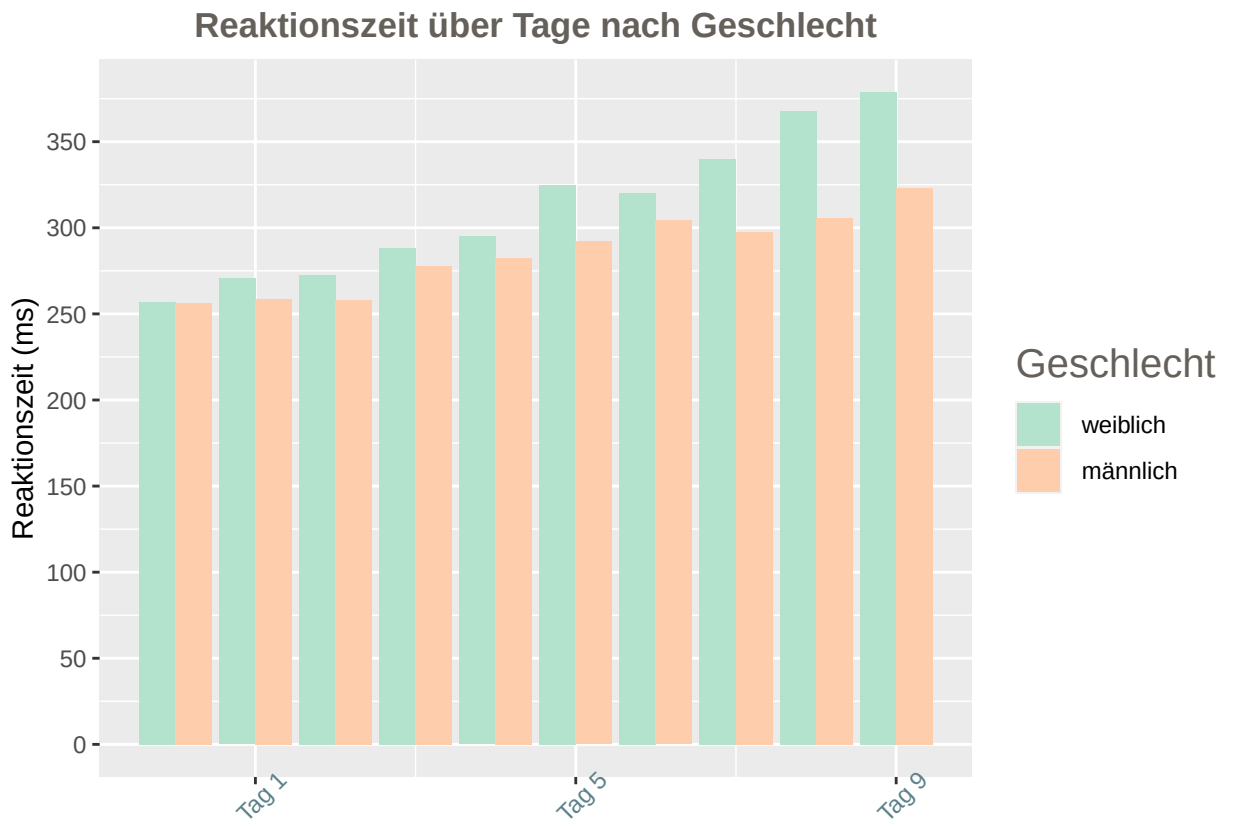
ggplot2 verfügt über die Farbpaletten des Pakets RColorBrewer. Diese lassen sich mit den Funktionen `scale_fill_brewer` oder `scale_colour_brewer` abrufen - je nachdem, ob `fill =` und/oder `colour =` für die Gruppierung verwendet wurden.
Im folgenden Beispiel wird die Palette "Pastel2" verwendet. Die *labels* werden wie bereits im Kapitel Farben erneut definiert, damit sie nicht überschrieben werden.

```
# Plot erstellen
sleep_bar_pal <- sleep_bar_col +
  scale_fill_brewer(labels = c("weiblich", "männlich"),  # Labels
                    palette="Pastel2")                   # Palette
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```

```
# Plotten
sleep_bar_pal
```

```
## No summary function supplied, defaulting to `mean_se()`
```



4.6.3.2 Entfärben

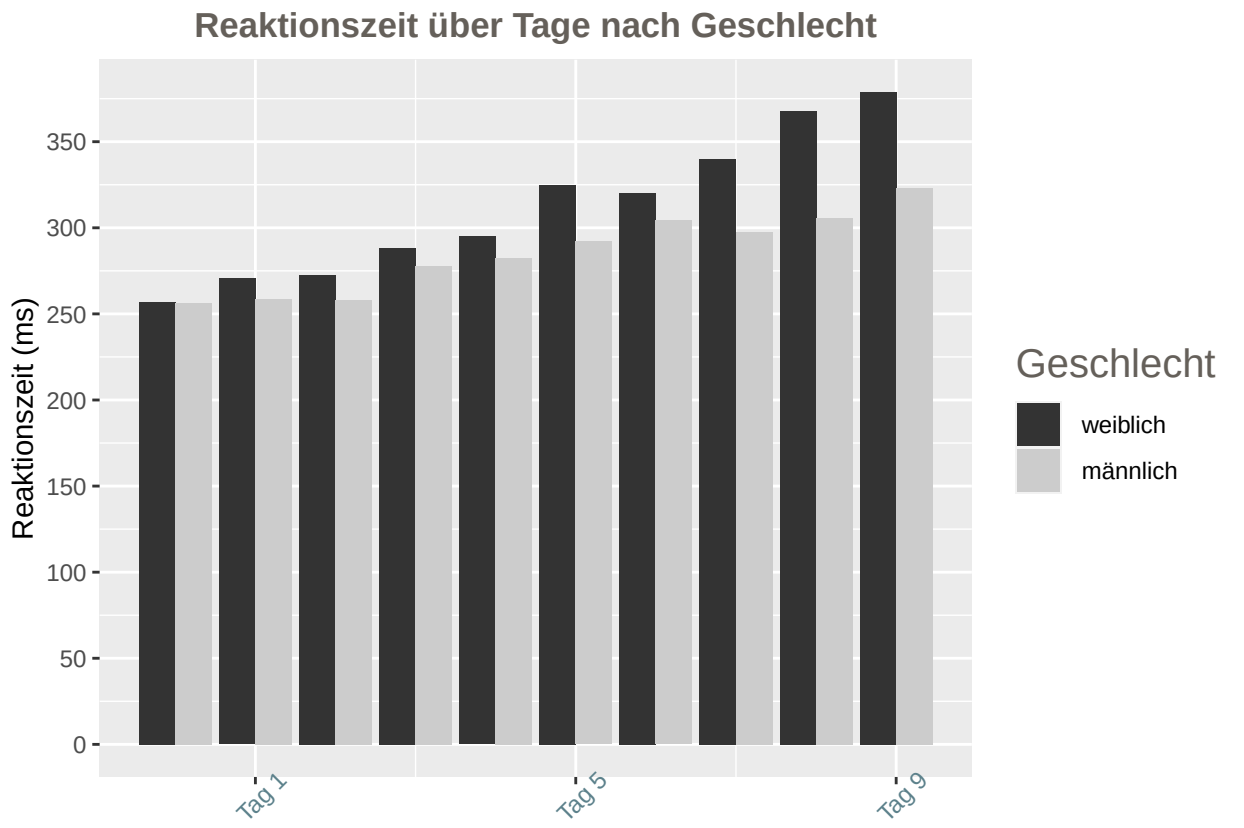
Die Funktionen `scale_fill_grey` und `scale_colour_grey` legen eine Palette aus verschiedenen Grautönen über die Gruppierungen:

```
# Plot erstellen
sleep_bar_bw <- sleep_bar_pal +
  scale_fill_grey(labels = c("weiblich", "männlich"))
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```

```
# Plotten
sleep_bar_bw
```

```
## No summary function supplied, defaulting to `mean_se()`
```



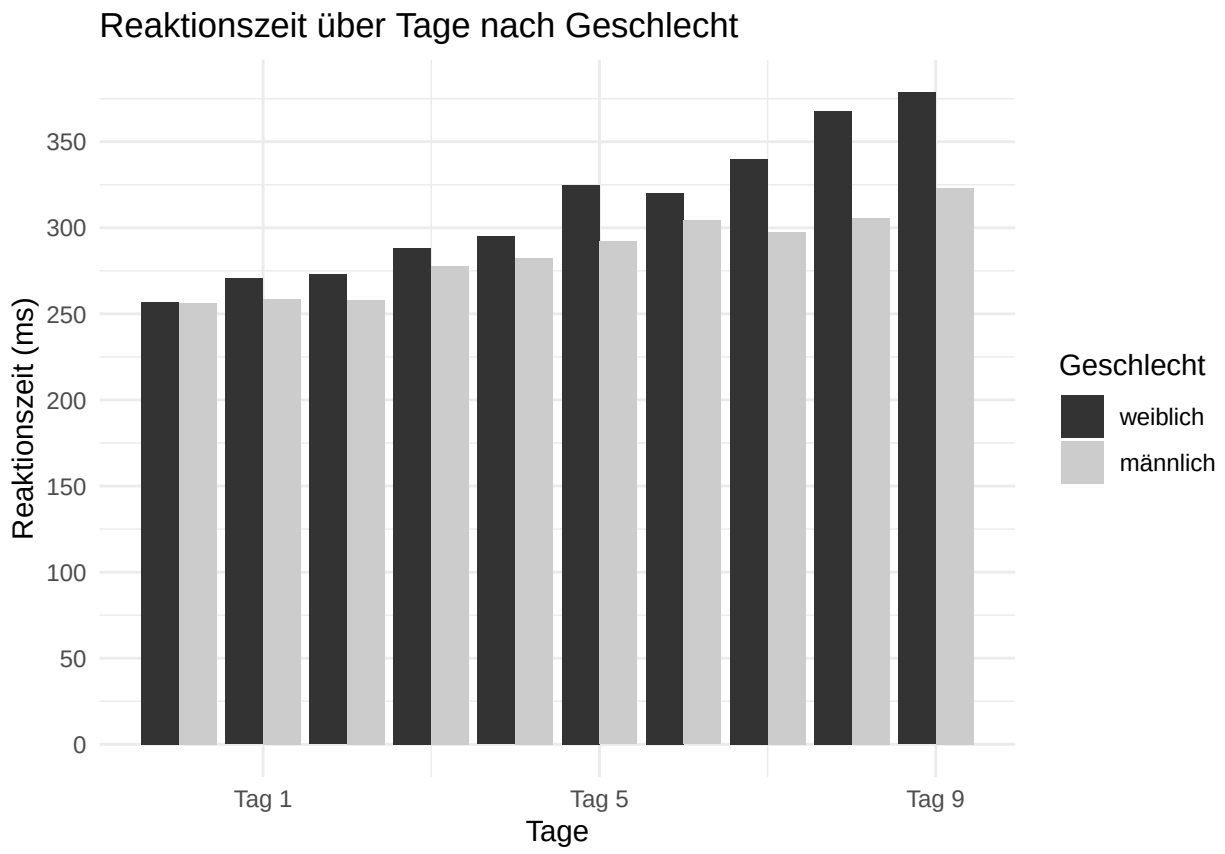
4.6.3.3 *Themes*

Das Aussehen der nicht-Daten komponenten lässt sich auch über sogenannte *themes* anpassen. Dies resultiert in einem einheitlichen Erscheinungsbild. Es stehen die Funktionen `theme_grey`, `theme_bw`, `theme_linedraw`, `theme_light`, `theme_dark`, `theme_minimal`, `theme_classic` & `theme_void` zur Verfügung.

Die Funktion `theme_minimal` überschreibt hier, was wir im Kapitel Titel in der `themes()`-Klammer definiert haben, weshalb der Titel der x-Achse wieder angezeigt wird, der Legendentitel in der ursprünglichen Größe vorhanden ist und der Haupttitel nicht mehr zentriert ist.

```
# Plot erstellen
sleep_bar_min <- sleep_bar_bw +
  theme_minimal()
# Plotten
sleep_bar_min
```

```
## No summary function supplied, defaulting to `mean_se()`
```



4.6.4 Weitere (grafische) Transformationen

- Achsen spiegeln: `scale_x_reverse()` oder `scale_y_reverse()`
  - **Achtung:** Überschreibt andere `scale_`-Funktionen der jeweiligen Achse!
- Achsen begrenzen: `xlim(min, max)` oder `ylim(min, max)` (min und max stehen für Zahlenwerte)
- Legendenposition verändern: `theme(legend.position = "")` (in Anführungsstrichen ergänzen: bottom, top, right, left)

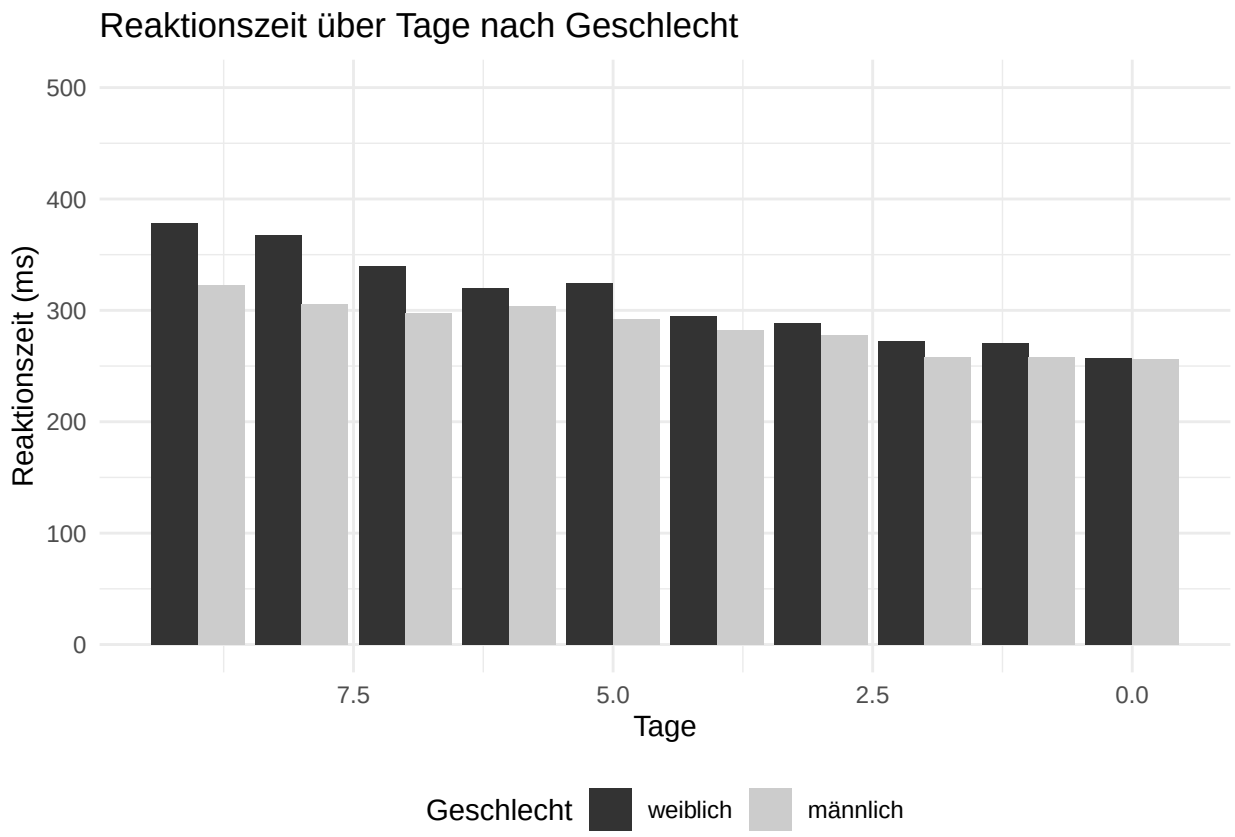
```
# Plot erstellen
sleep_bar_trans <- sleep_bar_min +
  scale_x_reverse() +       # x-Achse spiegeln
  ylim(0, 500) +            # Grenzen der y-Achse
  theme(legend.position = "bottom")   # Position der Legende
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```

```
# Plotten
sleep_bar_trans
```

```
## No summary function supplied, defaulting to `mean_se()`
```



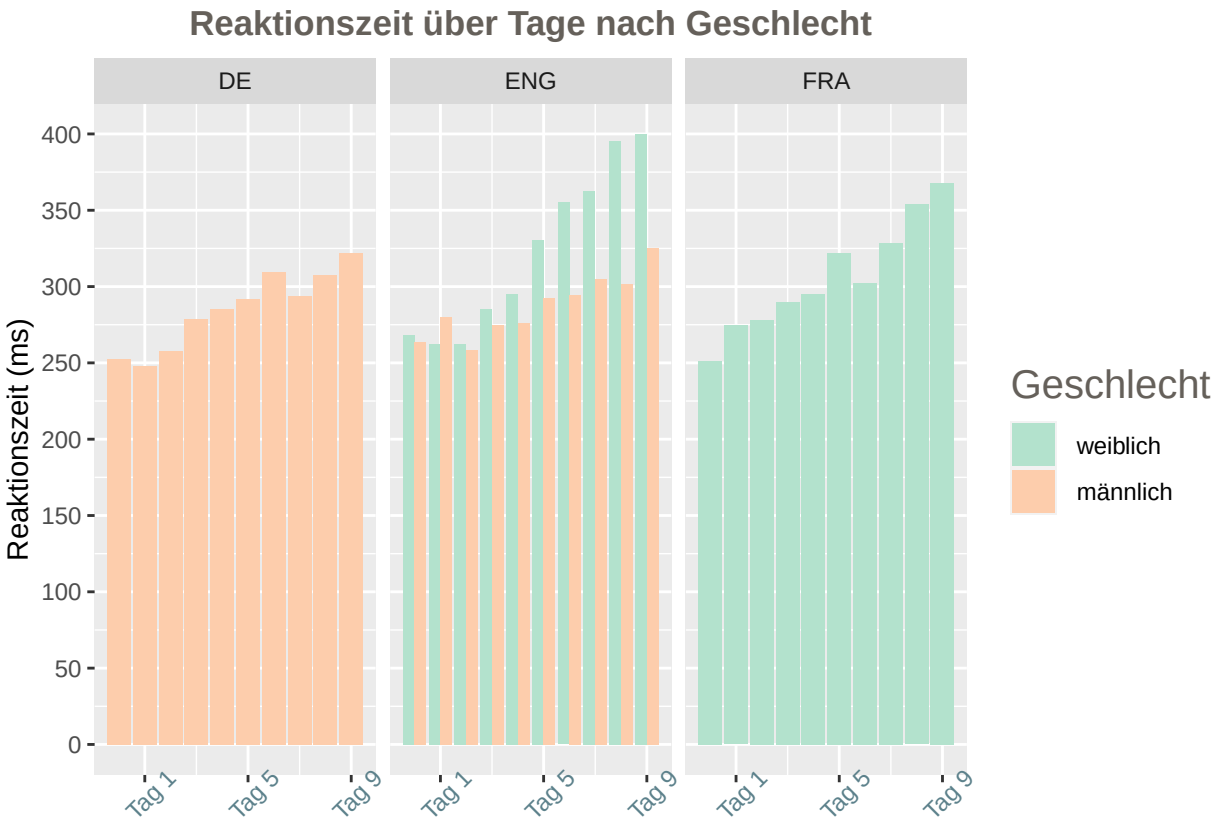
4.6.5 Aufgeteilte Plots

Plots lassen sich mit Hilfe der Funktion `facet_grid()` nach Variablen Aufteilen. Wir verwenden hierfür wieder den farbigen Plot aus dem Kapitel Paletten.

Plots aufgeteilt nach Sprache:

```
sleep_bar_pal +
  facet_grid(~ Language)
```

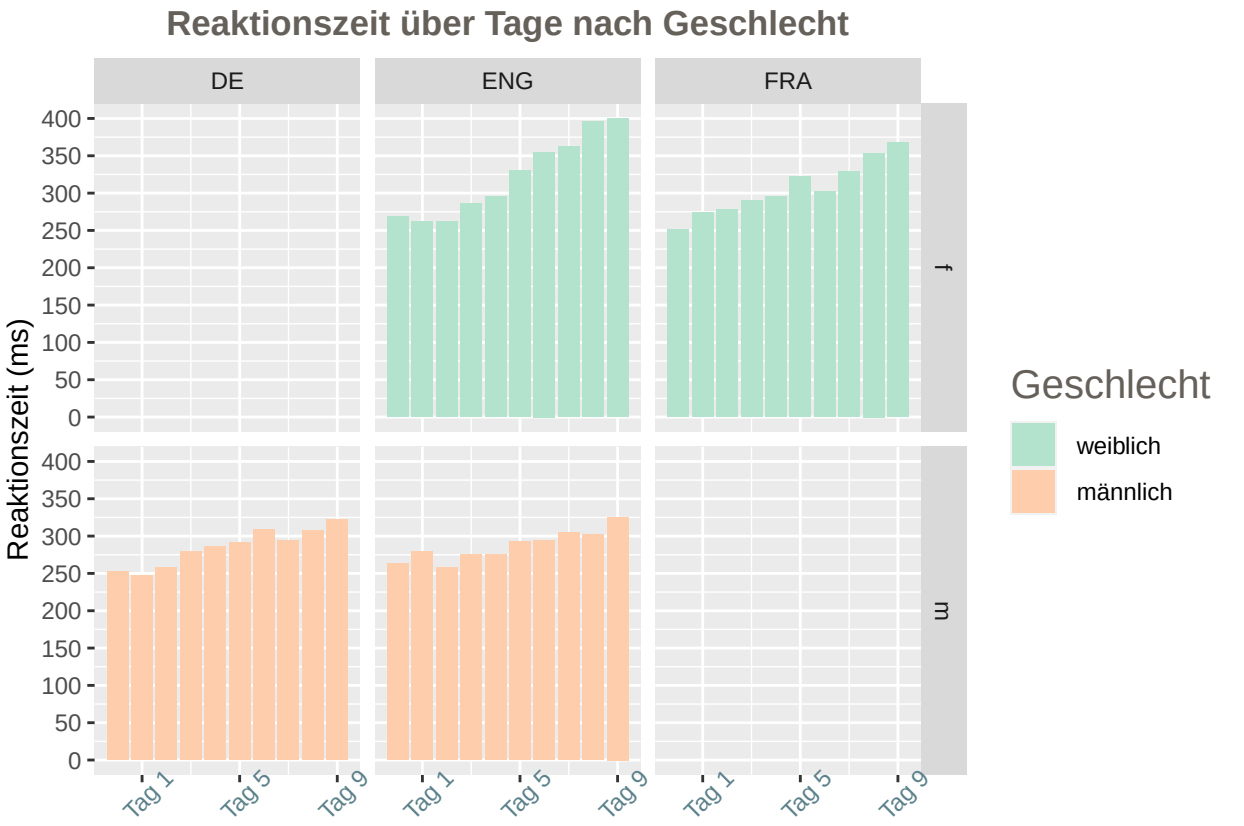
```
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
```



Plots aufgeteilt nach Sprache und Geschlecht:

```
sleep_bar_pal +
  facet_grid(Sex ~ Language)
```

```
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
```



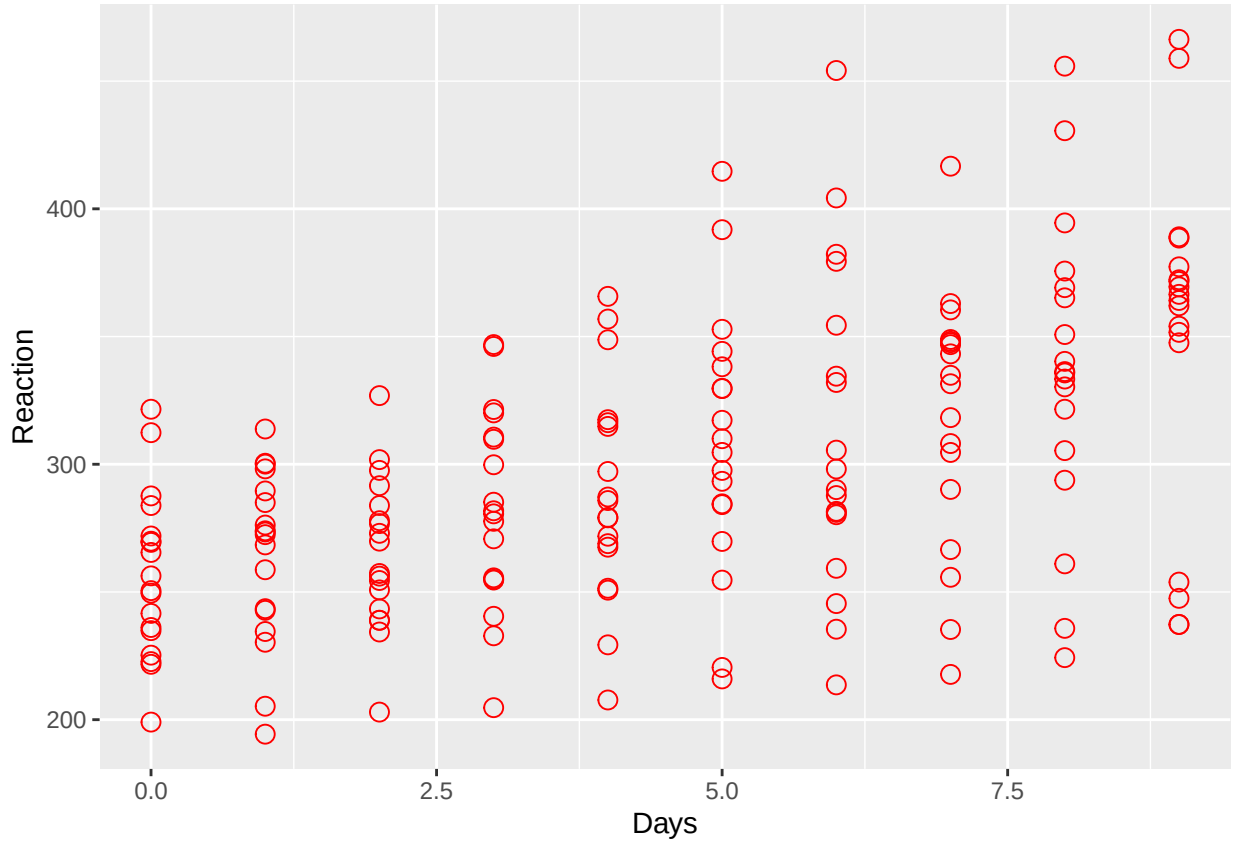
(Dieser Plot macht in diesem Fall natürlich nur bedingt Sinn, da wir bereits nach Geschlecht farbig gruppiert haben.)

4.6.6 Formen von Punkten

Die Formen von Scatterplots, bzw. Punktediagrammen, lassen sich in *ggplot2* auf verschiedenste Arten gestalten. Eine Übersicht über die Formen gibt es auf sthda.com.

Es stehen die Argumente `shape =`, `col =`, und `size =` zur Verfügung. Werden diese außerhalb der `aes()`-Klammer definiert, gilt die Formatierung für alle Punkte, und ist somit **ungruppiert:**

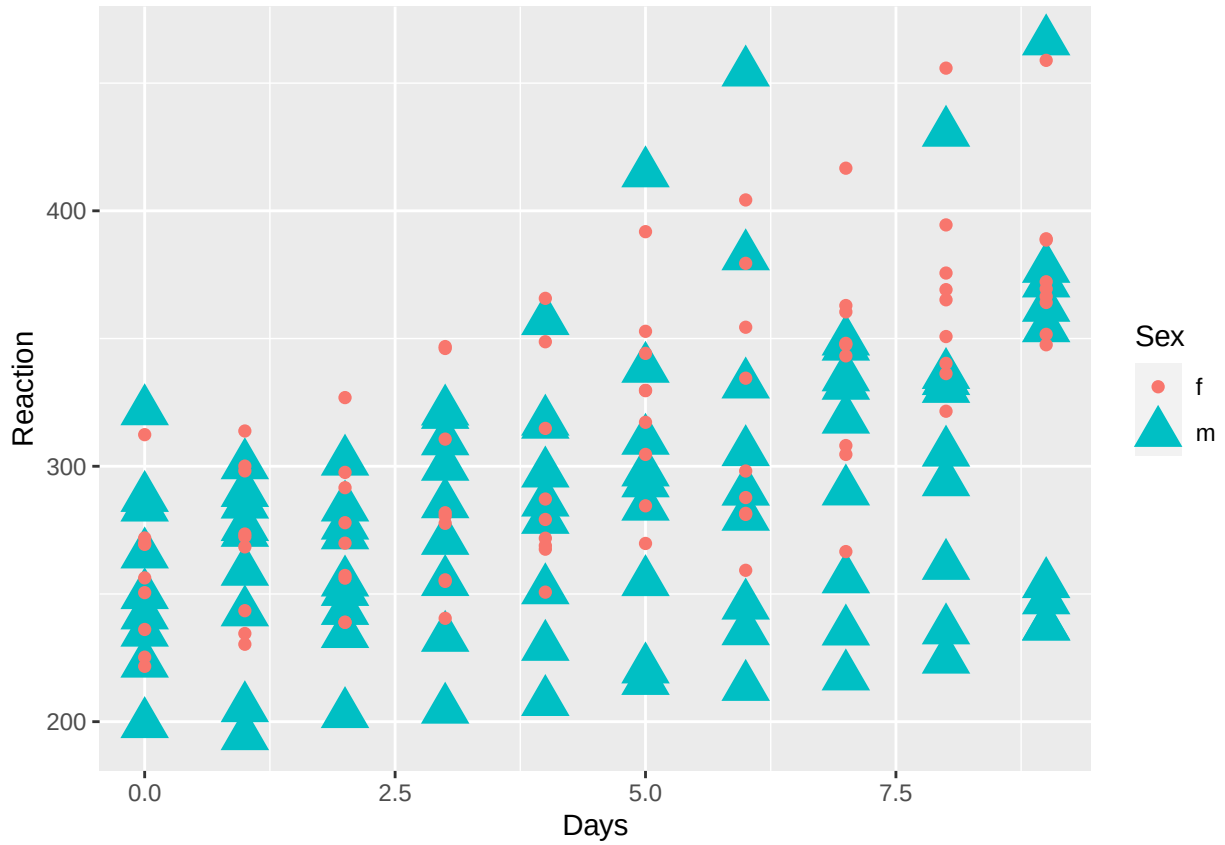
```
gg_sleep +
  geom_point(aes(x = Days, y = Reaction),
             shape = 1,
             col = "red",
             size = 3)
```



Werden sie innerhalb der Klammer definiert, werden die Punkte nach der angegebenen Variable in ihrer Form verändert **gruppiert**:

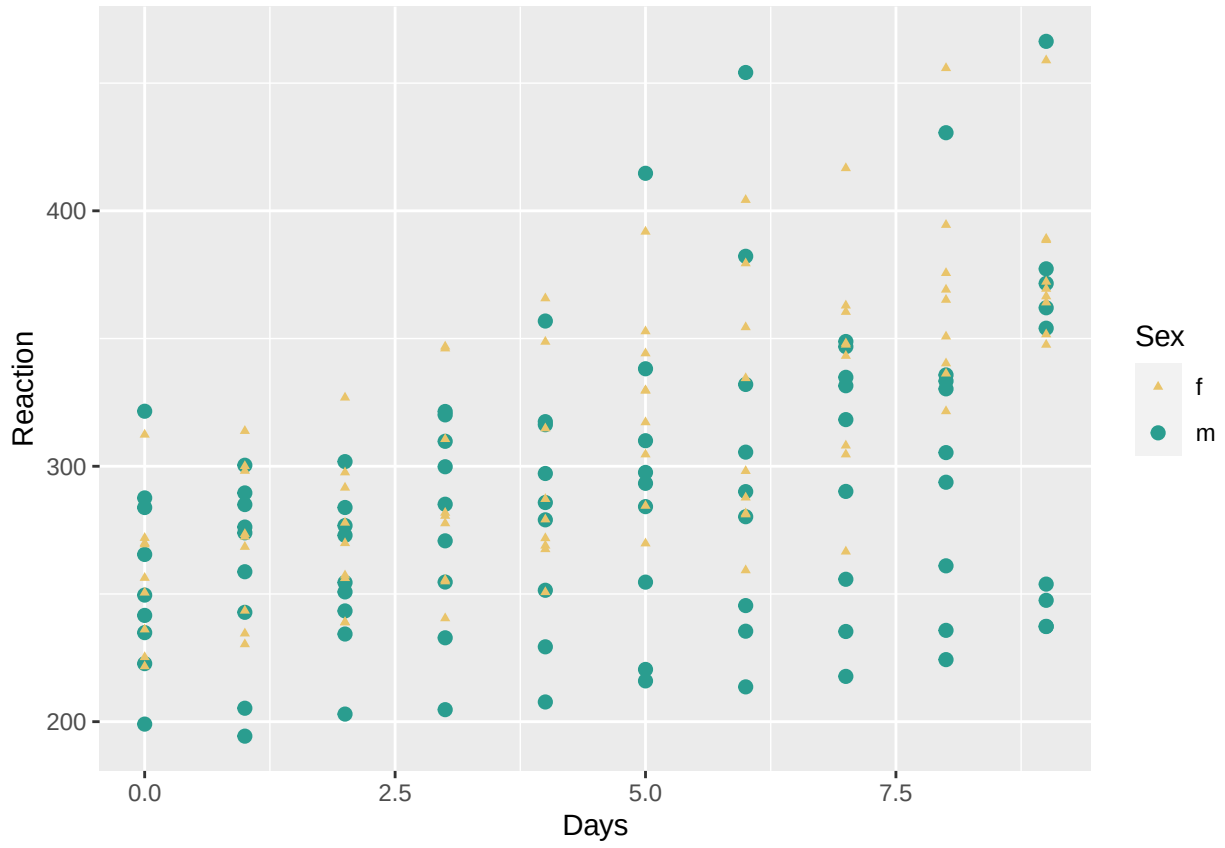
```
# Plot erstellen
sleep_point2 <- gg_sleep +
  geom_point(aes(x = Days, y = Reaction, shape = Sex, col = Sex, size = Sex))
# Plotten
sleep_point2
```

```
## Warning: Using size for a discrete variable is not advised.
```



Um die Form im nachhinein zu verändern können die Funktionen `scale_shape_manual`, `scale_colour_manual` und `scale_size_manual` wie folgt verwendet werden:

```
sleep_point2 +
  scale_shape_manual(values = c(17,19)) +
  scale_colour_manual(values = c("#e9c46a", "#2a9d8f")) +
  scale_size_manual(values = c(1,2))
```



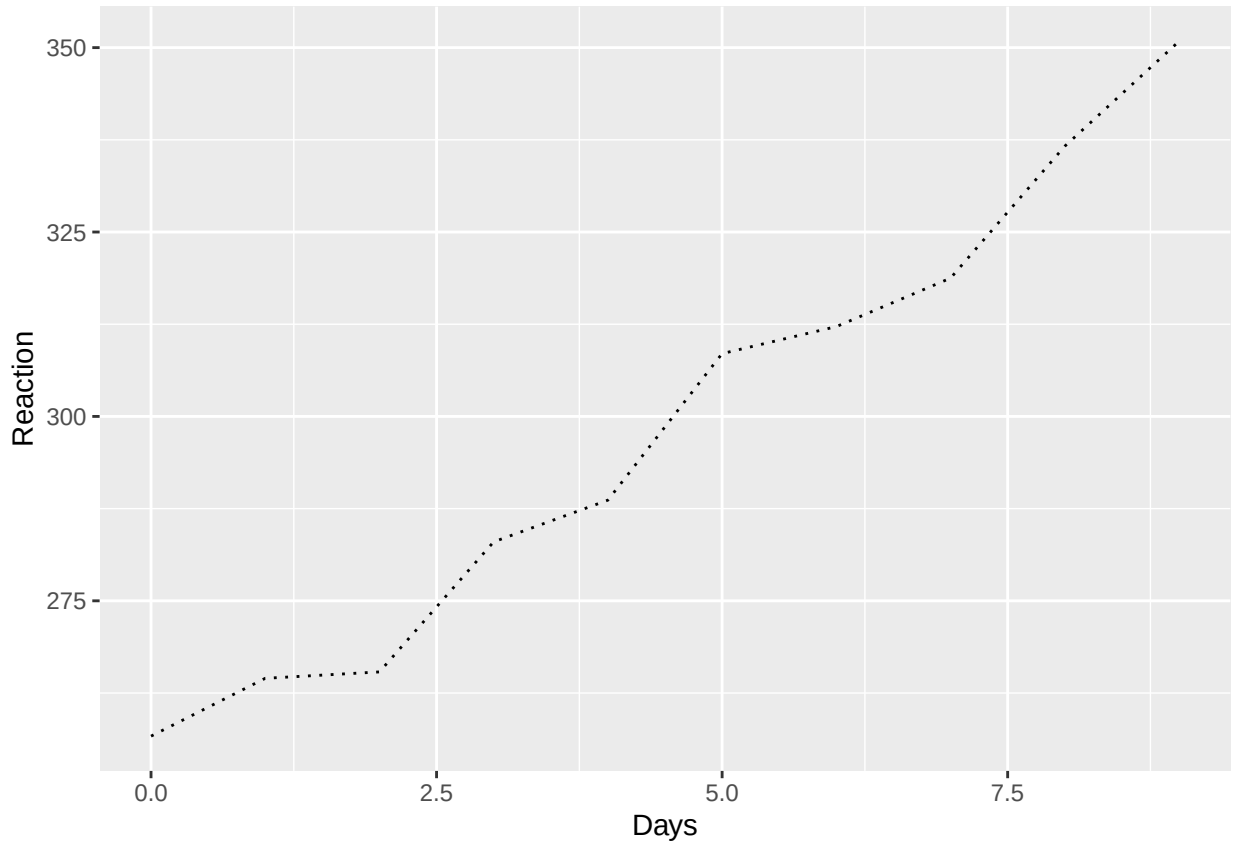
4.6.7 Formen von Linien

Die Formen von Linien lassen sich durch das Argument `linetype =`bestimmen. Es stehen folgende Möglichkeiten zur Auswahl: “blank”, “solid”, “dashed”, “dotted”, “dotdash”, “longdash”, “twodash”.

Wie in vorherigen Beispielen zu `shape =` und `fill =` kommt es darauf an, ob `linetype =` innerhalb der `aes()`-Klammer steht oder nicht.

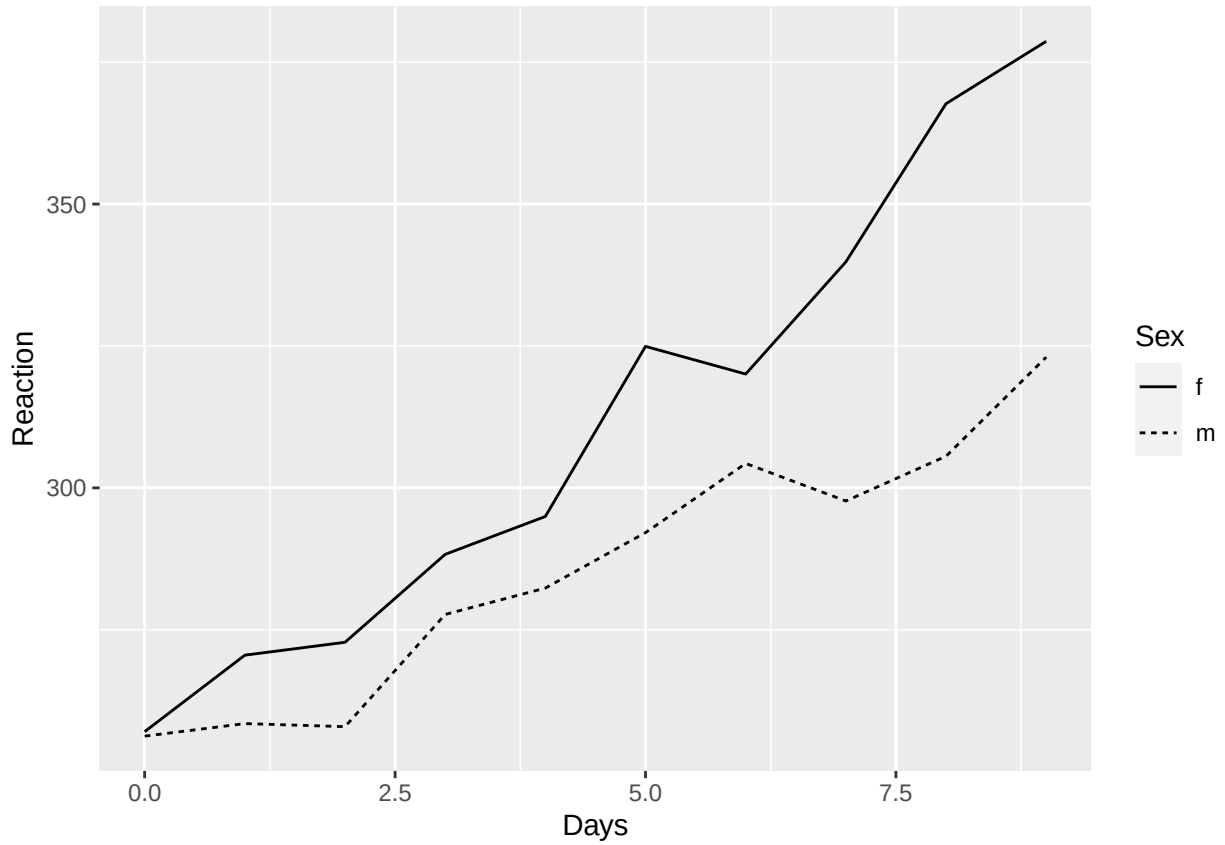
Steht es nicht in der Klammer gilt die Formatierung für alle Linien, und ist somit **ungruppiert:**

```
gg_sleep +
  geom_line(aes(x = Days, y = Reaction),
            stat = "summary", fun = "mean",
            linetype = "dotted")
```



Steht `linetype =` innerhalb der `aes()`-Klammer werden die Linien nach der angegeben Variable in unterschiedlichen Linienformen **gruppiert:**

```
# Plot erstellen
sleep_line <- gg_sleep +
  geom_line(aes(x = Days, y = Reaction, linetype = Sex),
            stat = "summary", fun = "mean")
# Plotten
sleep_line
```

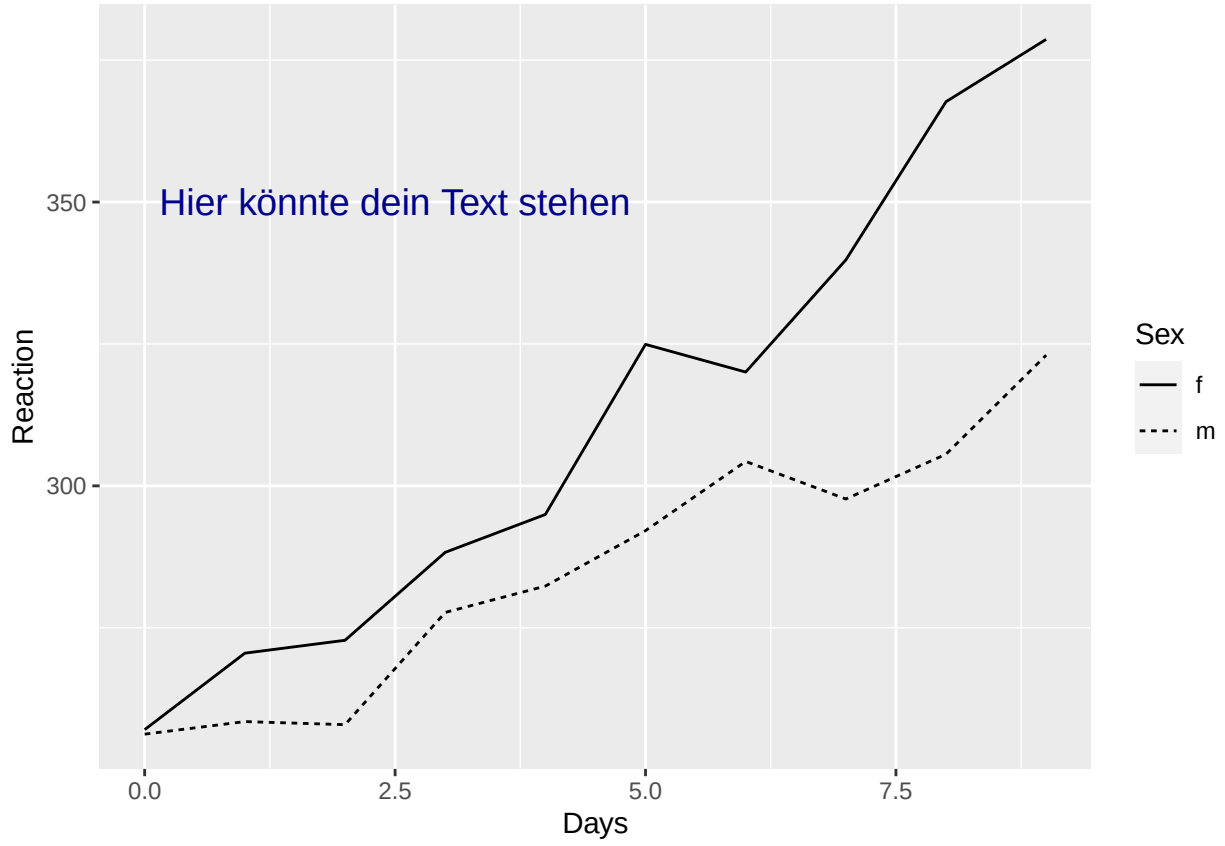


Um die Linienformen manuell zu verändern wird ähnlich der Beeinflussung von Farbe und Form die Funktion `scale_linetype_manual(values = c())`.

4.6.8 Text im Plot

Um Text innerhalb der Plots zu ergänzen wird die Funktion `annotate` verwendet:

```
sleep_line +
  annotate(x = 2.5,        # Position auf der x-Achse
           y = 350,        # Position auf der y-Achse
           geom = "text",    # "text" oder "label"
           label = "Hier könnte dein Text stehen",  # Text
           colour = "darkblue",  # Farbe
           size = 5)             # Größe
```

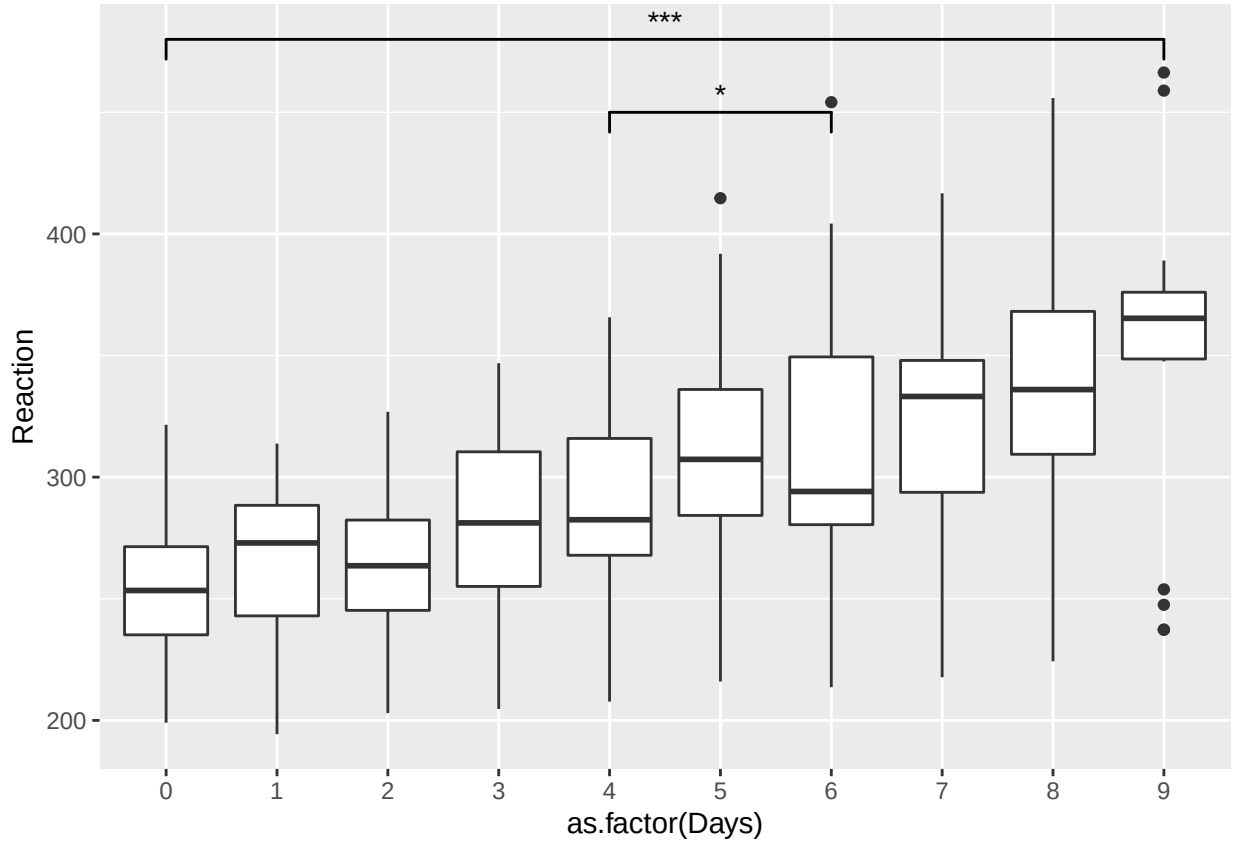


4.6.9 Signifikanzsterne

Signifikanzsterne lassen sich **manuell** hinzufügen. Dazu wird die Funktion `geom_signif` aus dem "ggpubr"-Paket verwendet. Die benötigten Argumente werden wie folgt benutzt:

- Mit `xmin()` und `xmax()` wird angegeben, welche Punkte auf der x-Achse mit Signifikanzssternen versehen werden sollen.
- `y_position =` gibt die jeweilige Position auf der y-Achse an
- `annotations =` bestimmt den Text (in diesem Fall die Signifikanzsterne)
- Die jeweiligen Stellen korrellieren nach der Stelle, an der sie stehen. Im folgenden Beispiel geben wir an, dass Tag "0" mit Tag "9" verglichen wurde und "***" auf der Höhe von y = 480 abgebildet werden soll.

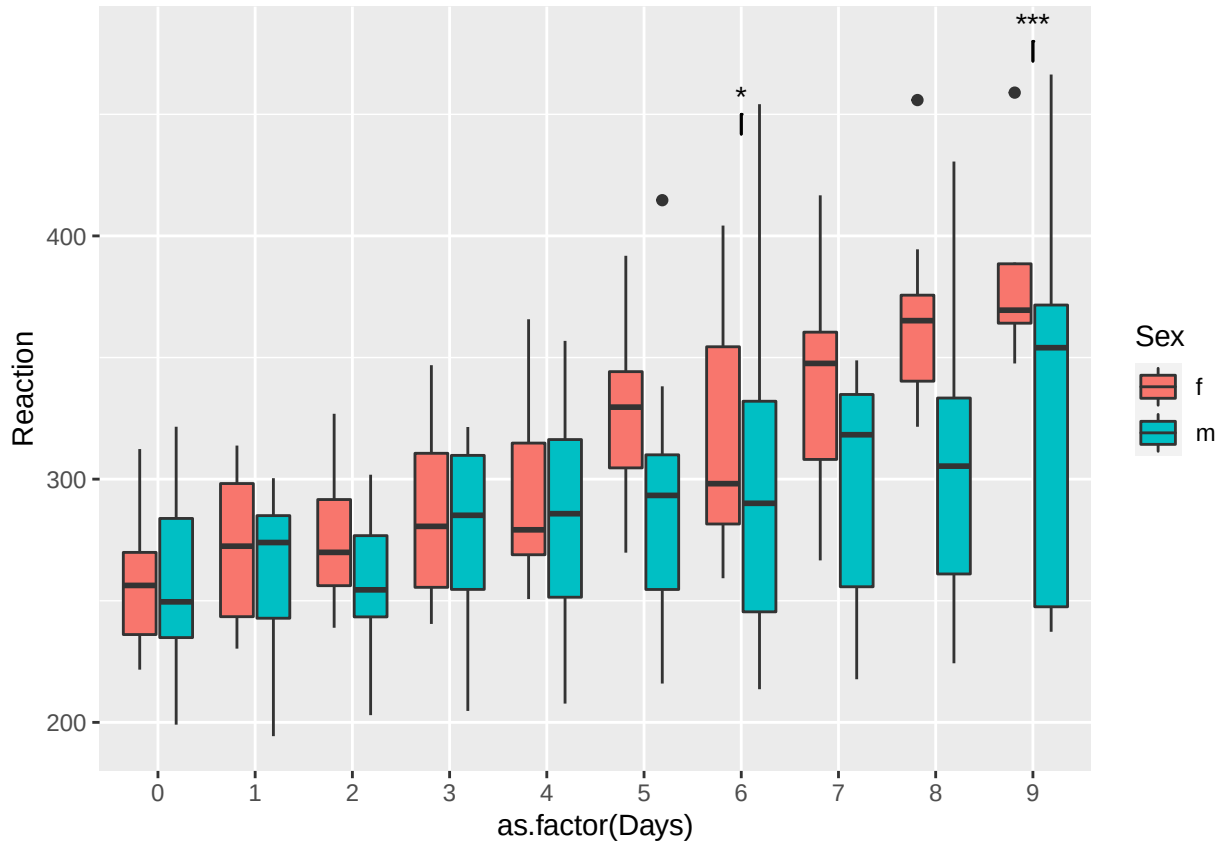
```
gg_sleep +
  geom_boxplot(aes(x=as.factor(Days), y=Reaction)) +
  geom_signif(aes(x=as.factor(Days), y=Reaction),
              xmin = c("0", "4"),          # x-Werte min
              xmax = c("9", "6"),          # x-Werte max
              y_position = c(480, 450),    # y-Werte
              annotations = c("***", "*")  # Signifikanzsterne/Text
              )
```



Um signifikante Unterschiede zwischen Gruppen zu markieren, wird bei `xmin` und `xmax` der gleiche Wert angegeben.

```
gg_sleep +
  geom_boxplot(aes(x=as.factor(Days), y=Reaction, fill = Sex)) +
  geom_signif(aes(x=as.factor(Days), y=Reaction, fill = Sex),
              xmin = c("9", "6"),           # x-Werte min
              xmax = c("9", "6"),           # x-Werte max
              y_position = c(480, 450),     # y-Werte
              annotations = c("***", "*")   # Signifikanzsterne/Text
              )
```

```
## Warning: Ignoring unknown aesthetics: fill
```



Signifkanzsterne können auch **automatisch** mit ggsignif oder dem Paket ggpubr - einer Erweiterung von "ggsignif" - ergänzt werden. Wie das geht, steht in den verlinkten Guides (evtl. wird dies hier zu einem späteren Zeitpunkt ergänzt).

4.6.10 Plot speichern

ggplots lassen sich entweder direkt über das "Plots"-Fenster in *RStudio* speichern, oder aber mit der Funktion `ggsave()`.
Der folgende Beispielcode ist auskommentiert, damit nicht bei jedem *Knit* eine Datei gespeichert wird.

Achtung: Ist im Ordner in dem die Datei gespeichert werden soll eine Datei mit dem gleichen Namen, wird diese überschieben, ohne das R euch warnt oder fragt.

```
# ggsave("filename.jpg", plot = sleep_point2)
```

Weitere Informationen:

- In den Klammern, in denen der Dateiname definiert wird, kann auch der Pfad angegeben werden (z.B.: `"user/documents/plots/filename.jpg"`)
- Wird der Plot nicht mit `plot =` spezifiziert, speichert R automatisch den zuletzt erstellten Plot
- Weitere Möglichkeiten (z.B. Bestimmen der Größe der Grafik) können der *Help-Page* entnommen werden (`?ggsave`)

---

5 Inferenzstatistik

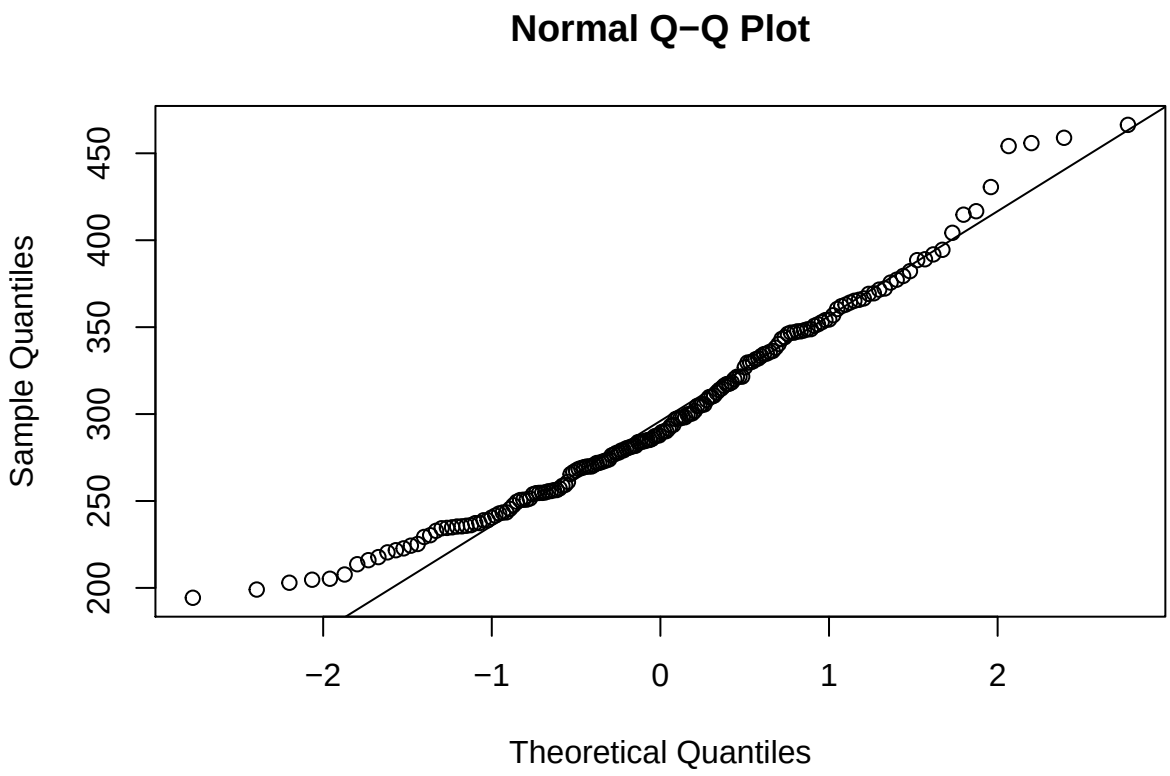
5.0.1 Testen der Normalverteilung

Die Normalverteilung lässt sich mit Hilfe des Shapiro-Wilk Tests oder anhand eines Q-Q Plots überprüfen:

```
# Shapiro-Wilk Test
shapiro.test(sleepstudy$Reaction)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sleepstudy$Reaction
## W = 0.97074, p-value = 0.00078
```

```
# Visuelle Darstellung (Q-Q Plot)
qqnorm(sleepstudy$Reaction)
qqline(sleepstudy$Reaction)
```



Gruppierter Shapiro-Wilk Test mit dem *rstatix* Paket:

```
sleepstudy %>%
  group_by(Days) %>%  # weitere unabhängige Variablen durch Komma getrennt
  shapiro_test(Reaction)
```

```
## # A tibble: 10 x 4
##     Days variable statistic      p
##    <dbl> <chr>        <dbl>  <dbl>
```

```
##  1     0 Reaction     0.977 0.909
##  2     1 Reaction     0.948 0.388
##  3     2 Reaction     0.987 0.994
##  4     3 Reaction     0.977 0.919
##  5     4 Reaction     0.972 0.843
##  6     5 Reaction     0.978 0.927
##  7     6 Reaction     0.959 0.585
##  8     7 Reaction     0.946 0.372
##  9     8 Reaction     0.971 0.819
## 10     9 Reaction     0.863 0.0134
```

5.0.2 Testen der Varianzhomogenität

Die Varianzhomogenität (auch Homoskedastizität) lässt sich mit dem Levene Test überprüfen:

```
# bei einer abhängigen Variable:
leveneTest(Reaction ~ Sex, data = sleepstudy)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   1  0.0593 0.8079
##       178
```

```
# bei mehreren abhängigen Variablen:
leveneTest(Reaction ~ Sex*Language, data = sleepstudy)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value   Pr(>F)
## group   3    4.46 0.004786 **
##       176
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Gruppierter Levene Test mit dem *rstatix* Paket:

```
sleepstudy %>%
  group_by(Days) %>% # weitere unabhängige Variablen durch Komma getrennt
  levene_test(Reaction ~ Sex)
```

```
## # A tibble: 10 x 5
##     Days   df1   df2 statistic      p
##    <dbl> <int> <int>     <dbl>  <dbl>
##  1     0     1    16  0.682    0.421
##  2     1     1    16  0.173    0.683
##  3     2     1    16  0.000283 0.987
##  4     3     1    16  0.0903   0.768
##  5     4     1    16  0.257    0.619
##  6     5     1    16  1.01     0.330
##  7     6     1    16  0.540    0.473
##  8     7     1    16  0.510    0.486
##  9     8     1    16  1.38     0.258
## 10     9     1    16  5.66     0.0302
```

5.1 Parametrische Tests

5.1.1 t-Test

5.1.1.1 t-Test für unabhängige Stichproben

Für den t-Test für unabhängige Stichproben erstellen wir hier einen Datensatz **ohne Messwiederholungen**, der nur aus den Messungen des **ersten Tages** besteht. Anschließend testen wir zur Verdeutlichung erneut, ob die Daten normalverteilt sind und ob Varianzhomogenität vorliegt.
Der t-Test wird nach dem folgenden Schema durchgeführt:
`t.test(dependent variable ~ independent variable, data = data)`

```
# Neuer Datensatz, nur mit Daten des ersten Tages
day1 <-
  filter(sleepstudy, Days == "1")
# Normalverteilung
shapiro.test(day1$Reaction)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  day1$Reaction
## W = 0.94756, p-value = 0.388
```

```
# Varianzhomogenität
leveneTest(Reaction ~ Sex, data = day1)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  1  0.1729 0.6831
##       16
```

```
# t-test für unabhängige Stichproben
t.test(Reaction ~ Sex, data = day1)
```

```
##
##  Welch Two Sample t-test
##
## data:  Reaction by Sex
## t = 0.75517, df = 15.274, p-value = 0.4616
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -21.91541  46.02373
## sample estimates:
## mean in group f mean in group m
##        270.5228        258.4687
```

5.1.1.2 t-Test für abhängige Stichproben

Für den t-Test für abhängige Stichproben erstellen wir hier einen Datensatz mit **einer Messwiederholung**, der nur aus den Messungen des **ersten** und des **neunten Tages** besteht. Anschließend testen wir zur Verdeutlichung erneut, ob die Daten normalverteilt sind und ob Varianzhomogenität vorliegt.
Damit R weiß, dass es sich um einen t-test für abhängige Stichproben handelt, wird in der Funktionsklammer `paired = T` ergänzt.

```r
# Neuer Datensatz, nur mit Daten des ersten und neunten Tages
day1_9 <-
     filter(sleepstudy, Days %in% c("1", "9"))
# Normalverteilung
shapiro.test(day1_9$Reaction)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  day1_9$Reaction
## W = 0.94603, p-value = 0.07846
```

```r
# Varianzhomogenität
# für die Funktion leveneTest() darf die unabhänigige Variable nicht numerisch sein, daher verwenden wi
leveneTest(Reaction ~ as.factor(Days), data = day1_9)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  1  2.1278 0.1538
##       34
```

```r
# t-test für unabhängige Stichproben
t.test(Reaction ~ Days, data = day1_9, paired = T)
```

```
##
##  Paired t-test
##
## data:  Reaction by Days
## t = -6.5205, df = 17, p-value = 5.236e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -114.29724  -58.41369
## sample estimates:
## mean of the differences
##               -86.35547
```

5.1.2 Anova

Die Tests nach Normalverteilung und Varianzhomogenität sind analog zur Darstellung in den jeweiligen Kapiteln:

- Testen der Normalverteilung
- Testen der Varianzhomogenität

5.1.2.1 Univariate einfaktorielle ANOVA

Die univariate einfaktorielle ANOVA wird nach dem folgenden Schema durchgeführt:
`aov(dependent variable ~ independent variable, data = data)`
Die Ergebnisse der ANOVA erhalten wir, indem wir die ANOVA in die Funktion `summary()` einbetten.

```r
# Speichern der ANOVA in einer Variable
anova1 <-
  aov(Reaction ~ Language, data = day1)
```

```r
# Ergebnisse der ANOVA
summary(anova1)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## Language     2   2556    1278   1.166  0.338
## Residuals   15  16443    1096
```

```r
# post-hoc Bonferroni
pairwise.t.test(day1$Reaction, day1$Language, p.adj = "bonf")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  day1$Reaction and day1$Language
##
##     DE   ENG
## ENG 0.72 -
## FRA 0.55 1.00
##
## P value adjustment method: bonferroni
```

```r
# Oder: post-hoc TukeyHSD
TukeyHSD(anova1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = Reaction ~ Language, data = day1)
##
## $Language
##              diff       lwr      upr     p adj
## ENG-DE  23.460917 -26.19103 73.11286 0.4560021
## FRA-DE  26.769000 -22.88295 76.42095 0.3655879
## FRA-ENG  3.308083 -46.34386 52.96003 0.9836429
```

5.1.2.2 Univariate multifaktorielle ANOVA

Die univariate multifaktorielle ANOVA wird nach dem folgenden Schema durchgeführt:
`aov(dependent variable ~ independent variable1 * independent variable2, data = data)`

```r
# Speichern der ANOVA in einer Variable
anova2 <-
  aov(Reaction ~ Language * Sex, data = day1)
# Ergebnisse der ANOVA
summary(anova2)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## Language     2   2556  1277.9   1.119  0.354
## Sex          1    453   452.9   0.396  0.539
## Residuals   14  15990  1142.2
```

```r
# post-hoc TukeyHSD
TukeyHSD(anova2)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
```

```
##
## Fit: aov(formula = Reaction ~ Language * Sex, data = day1)
##
## $Language
##              diff       lwr      upr     p adj
## ENG-DE  23.460917 -27.60759 74.52942 0.4711094
## FRA-DE  26.769000 -24.29951 77.83751 0.3813117
## FRA-ENG  3.308083 -47.76042 54.37659 0.9842962
##
## $Sex
##         diff       lwr      upr    p adj
## m-f 5.791844 -28.37787 39.96156 0.721628
```

5.1.2.3 Multivariate einfaktorielle ANOVA

Die univariate einfaktorielle ANOVA wird nach dem folgenden Schema durchgeführt:
`manova(cbind(dependent variable1, dependent variable2) ~ independent variable, data = data)`

```
# Speichern der MANOVA in einer Variable
manova1 <-
  manova(cbind(Reaction, VerbalFluency) ~ Language, data = day1)
# Ergebnisse (gesamt)
summary(manova1)
```

```
##           Df Pillai approx F num Df den Df Pr(>F)
## Language   2 0.1471  0.59541      4     30 0.6687
## Residuals 15
```

```
# Ergebnisse (aufgeteilt nach abhängigen Variablen)
summary.aov(manova1)
```

```
##  Response Reaction :
##             Df  Sum Sq Mean Sq F value Pr(>F)
## Language     2  2555.9  1277.9  1.1658 0.3384
## Residuals   15 16443.1  1096.2
##
##  Response VerbalFluency :
##             Df  Sum Sq Mean Sq F value Pr(>F)
## Language     2   1.603  0.8014   0.118 0.8895
## Residuals   15 101.845  6.7896
```

Post-hoc Tests für Multivariate ANOVA müssen für jede Variable einzeln berechnet werden, siehe Kapitel Univariate einfaktorielle ANOVA.

5.1.2.4 Multivariate multifaktorielle ANOVA

Die univariate einfaktorielle ANOVA wird nach dem folgenden Schema durchgeführt:
`manova(cbind(dependent variable1, dependent variable2) ~ independent variable1 * independent variable2, data = data)`

```
# Speichern der MANOVA in einer Variable
manova2 <-
  manova(cbind(Reaction, VerbalFluency) ~ Language*Sex, data = day1)
```

```r
# Ergebnisse (gesamt)
summary(manova2)
```

```
##           Df   Pillai approx F num Df den Df Pr(>F)
## Language   2 0.150357  0.56903      4     28 0.6872
## Sex        1 0.054171  0.37228      2     13 0.6963
## Residuals 14
```

```r
# Ergebnisse (aufgeteilt nach abhängigen Variablen)
summary.aov(manova2)
```

```
##  Response Reaction :
##             Df  Sum Sq Mean Sq F value Pr(>F)
## Language     2  2555.9 1277.94  1.1189 0.3542
## Sex          1   452.9  452.86  0.3965 0.5390
## Residuals   14 15990.2 1142.16
##
##  Response VerbalFluency :
##             Df Sum Sq Mean Sq F value Pr(>F)
## Language     2  1.603  0.8014  0.1127 0.8943
## Sex          1  2.258  2.2583  0.3175 0.5820
## Residuals   14 99.586  7.1133
```

Post-hoc Tests für Multivariate ANOVA müssen für jede Variable einzeln berechnet werden, siehe Kapitel Univariate multifaktorielle ANOVA.

5.1.2.5 One-way repeated measures ANOVA

One-way repeated measures ANOVAs lassen sich zwar auch gut mit *baseR*-Funktionen durchführen, allerdings möchte ich an dieser Stelle auch auf das Paket *rstatix* hinweisen.

5.1.2.5 mit *rstatix*

Damit die Funktionen des *rstatix* Pakets funktionieren, dürfen sich im Datensatz nur die abhängigen Variablen im Datensatz befinden, die auch beobachtet werden sollen:

```r
# Nur die abhängigen Variablen, die auch benötigt werden im Datensatz:
sleepstudy2 <-
  sleepstudy[, c("Reaction", "Subject", "Language", "Days")]
```

Berechnung:
Aus der Anleitung entnommene Definitionen für die Argumente der Funktion `anova_test()`:

- *dv:* (numeric) dependent variable name.
- *wid:* (factor) column name containing individuals/subjects identifier. Should be unique per individual
- *between:* (optional) between-subject factor variables
- *within:* (optional) within-subjects factor variables
- *covariate:* (optional) covariate names (for ANCOVA)

```r
# Speichern der ANOVA in einer Variable
ranova1x <- anova_test(
```

```
  data = sleepstudy2, dv = Reaction, wid = Subject,
  within = Days
  )
# Ergebnisse
get_anova_table(ranova1x)
```

```
## ANOVA Table (type III tests)
##
##   Effect  DFn   DFd      F        p p<.05   ges
## 1   Days 3.32 56.46 18.703 5.46e-09     * 0.293
```

Post-Hoc Test:
Der post-hoc Test für *one-way repeated measures ANOVAs* mit *rstatix* wird mit Hilfe der Funktion `pairwise_t_test()`durchgeführt:

```
sleepstudy2 %>%
  pairwise_t_test(
    Reaction ~ Days, paired = TRUE,
    p.adjust.method = "bonferroni"
    )
```

```
## # A tibble: 45 x 10
##    .y.    group1 group2    n1    n2 statistic    df       p   p.adj p.adj.signif
##  * <chr>  <chr>  <chr>  <int> <int>     <dbl> <dbl>   <dbl>   <dbl> <chr>
##  1 React~ 0      1         18    18    -1.40     17 1.80e-1 1.00e+0 ns
##  2 React~ 0      2         18    18    -1.16     17 2.61e-1 1.00e+0 ns
##  3 React~ 0      3         18    18    -3.00     17 8.00e-3 3.60e-1 ns
##  4 React~ 0      4         18    18    -3.38     17 4.00e-3 1.61e-1 ns
##  5 React~ 0      5         18    18    -4.42     17 3.72e-4 1.70e-2 *
##  6 React~ 0      6         18    18    -3.67     17 2.00e-3 8.60e-2 ns
##  7 React~ 0      7         18    18    -5.96     17 1.56e-5 7.02e-4 ***
##  8 React~ 0      8         18    18    -5.83     17 1.99e-5 8.95e-4 ***
##  9 React~ 0      9         18    18    -6.96     17 2.31e-6 1.04e-4 ***
## 10 React~ 1      2         18    18    -0.170    17 8.67e-1 1.00e+0 ns
## # ... with 35 more rows
```

5.1.2.5 mit *baseR*

```
ranova1 <- aov(Reaction ~ Days + Error(Subject), data = sleepstudy)
summary(ranova1)
```

```
##
## Error: Subject
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals 17 250618   14742
##
## Error: Within
##             Df Sum Sq Mean Sq F value Pr(>F)
## Days         1 162703  162703   169.4 <2e-16 ***
## Residuals  161 154634     960
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Der post-Hoc Test für *one-way repeated measures ANOVAs* mit *baseR* wird wie im Kapitel Univariate einfaktorielle ANOVA durchgeführt, jedoch ergänzen wir `paired = TRUE`:

```r
pairwise.t.test(sleepstudy$Reaction, sleepstudy$Days,
                p.adj = "bonf",
                paired = TRUE)
```

```
##
##  Pairwise comparisons using paired t tests
##
## data:  sleepstudy$Reaction and sleepstudy$Days
##
##   0       1       2       3       4       5       6       7       8
## 1 1.00000 -       -       -       -       -       -       -       -
## 2 1.00000 1.00000 -       -       -       -       -       -       -
## 3 0.35979 0.39174 0.05572 -       -       -       -       -       -
## 4 0.16075 0.27762 0.22326 1.00000 -       -       -       -       -
## 5 0.01675 0.02640 0.03769 0.34640 0.27753 -       -       -       -
## 6 0.08559 0.19400 0.11594 0.76197 1.00000 1.00000 -       -       -
## 7 0.00070 0.00396 0.00143 0.08087 0.13300 1.00000 1.00000 -       -
## 8 0.00090 0.00194 0.00185 0.00932 0.00437 0.01288 1.00000 1.00000 -
## 9 0.00010 0.00024 0.00068 0.00318 0.00143 0.00641 1.00000 0.70714 1.00000
##
## P value adjustment method: bonferroni
```

5.1.2.6 Two-way repeated measures ANOVA

Auch *two-way repeated measures ANOVAs* lassen sich mit *baseR*-Funktionen und *rstatix* durchführen. Für post-hoc Tests muss in jedem Fall auf *rstatix* zurückgegriffen werden.

5.1.2.6 mit *rstatix*

Auch hier gelten die Voraussetzungen, damit die Funktionen des *rstatix* Pakets funktionieren (siehe [One-way repeated measures ANOVA mit *rstatix*]).

Berechnung:
Wir ergänzen hier beispielsweise den *between-subjects* Faktor "Language":

```r
# Speichern der ANOVA in einer Variable
ranova2x <- anova_test(
  data = sleepstudy2, dv = Reaction, wid = Subject,
  within = Days, between = Language
  )
# Ergebnisse
get_anova_table(ranova2x)
```

```
## ANOVA Table (type II tests)
##
##          Effect  DFn   DFd      F        p p<.05   ges
## 1      Language 2.00 15.00  0.553 5.86e-01       0.044
## 2          Days 3.10 46.55 18.151 4.57e-08     * 0.310
## 3 Language:Days 6.21 46.55  0.749 6.17e-01       0.036
```

Es ließen sich auch weitere *within-* oder *between-subjects* Faktoren einbeziehen. Dies erfolgt nach dem folgenden Muster: `within = c(Variable1, Variable2)`

Post-Hoc-Tests
(vgl. https://www.datanovia.com/en/lessons/repeated-measures-anova-in-r/)

Zunächst können in einem ersten Schritt die p-Werte des Effekts unserer unabhängigen Variable pro Zeitpunkt berechnet werden:

```
# Effekt der Sprache pro Tag
sleepstudy2 %>%
  group_by(Days) %>%
  anova_test(dv = Reaction, wid = Subject, between = Language) %>%
  get_anova_table() %>%
  adjust_pvalue(method = "bonferroni")
```

```
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
```

```
## # A tibble: 10 x 9
##     Days Effect     DFn   DFd     F     p `p<.05`   ges p.adj
##    <dbl> <chr>    <dbl> <dbl> <dbl> <dbl> <chr>   <dbl> <dbl>
##  1     0 Language     2    15 0.35  0.711 ""      0.045     1
##  2     1 Language     2    15 1.17  0.338 ""      0.135     1
##  3     2 Language     2    15 0.83  0.455 ""      0.1       1
##  4     3 Language     2    15 0.121 0.887 ""      0.016     1
##  5     4 Language     2    15 0.086 0.918 ""      0.011     1
##  6     5 Language     2    15 0.492 0.621 ""      0.062     1
##  7     6 Language     2    15 0.175 0.841 ""      0.023     1
##  8     7 Language     2    15 1.14  0.346 ""      0.132     1
##  9     8 Language     2    15 1.07  0.368 ""      0.125     1
## 10     9 Language     2    15 0.841 0.451 ""      0.101     1
```

Für die Berechnung der post-hoc Tests von two-way repeated measures ANOVAs wird im Vergleich zum Kapitel **One-way repeated measures ANOVA** mit *rstatix* eine Gruppierung ("group_by()") nach Messzeitpunkt vorgenommen. In der Klammer der "pairwise_t_test()"-Funktion steht unsere abhängige und die unabhängige Variable, die uns interessiert:

```
# Pairwise comparison zwischen Sprachen pro Tag
sleepstudy2 %>%
  group_by(Days) %>%
  pairwise_t_test(
    Reaction ~ Language, paired = TRUE,
    p.adjust.method = "bonferroni"
    )
```

```
## # A tibble: 30 x 11
##     Days .y.   group1 group2    n1    n2 statistic    df     p p.adj
##  * <dbl> <chr> <chr>  <chr>  <int> <int>     <dbl> <dbl> <dbl> <dbl>
##  1     0 Reac~ DE     ENG        6     6   -0.881      5 0.419 1
##  2     0 Reac~ DE     FRA        6     6    0.0481     5 0.964 1
##  3     0 Reac~ ENG    FRA        6     6    0.738      5 0.494 1
##  4     1 Reac~ DE     ENG        6     6   -1.16       5 0.297 0.891
##  5     1 Reac~ DE     FRA        6     6   -1.13       5 0.312 0.936
##  6     1 Reac~ ENG    FRA        6     6   -0.186      5 0.86  1
```

```
##  7     2 Reac~ DE     ENG        6     6   -0.160      5 0.879 1
##  8     2 Reac~ DE     FRA        6     6   -0.853      5 0.433 1
##  9     2 Reac~ ENG    FRA        6     6   -0.926      5 0.397 1
## 10     3 Reac~ DE     ENG        6     6   -0.0579     5 0.956 1
## # ... with 20 more rows, and 1 more variable: p.adj.signif <chr>
```

5.1.2.6 mit *baseR*

Um eine *two-way repeated measures ANOVA* in *baseR* durchzuführen, gehen wir nach folgendem Schema vor:
`aov(dependent variable ~ independent variable1 * independent variable2 + Error(ID/within-subject-conditi`
`data = data)`
Das bedeutet:

- unsere *independent variables* werden durch einen Asterisk getrennt
- wir ergänzen einen Error()-Term
- in dieser Error()-Klammer steht die Variable für unsere Subject-IDs, gefolgt von einem slash (/) und unseren *within-subjects conditions*, jedoch **nicht** den *between-subject conditions* -> In diesem Fall also **nicht** `Error(Subject/Language*Days)`

```r
# Speichern der ANOVA in einer Variable
ranova2 <- aov(Reaction ~ Language*Days + Error(Subject/Days),
               data = sleepstudy)
# Ergebnisse
summary(ranova2)
```

```
##
## Error: Subject
##           Df Sum Sq Mean Sq F value Pr(>F)
## Language   2  17215    8608   0.553  0.586
## Residuals 15 233403   15560
##
## Error: Subject:Days
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Days           1 162703  162703  43.746 8.23e-06 ***
## Language:Days  2   4534    2267   0.609    0.557
## Residuals     15  55788    3719
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Error: Within
##            Df Sum Sq Mean Sq F value Pr(>F)
## Residuals 144  94312   654.9
```

Für den Post-Hoc Test müssen wir bei der two-way repeated measures ANOVA mit *baseR* auf das *rstatix* Paket zurückgreifen:

```r
sleepstudy %>%
  group_by(Days) %>%
  anova_test(dv = Reaction, wid = Subject, between = Language) %>%
  get_anova_table() %>%
  adjust_pvalue(method = "bonferroni")
```

```
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
```

```
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
## Coefficient covariances computed by hccm()
```

```
## # A tibble: 10 x 9
##     Days Effect     DFn   DFd     F     p `p<.05`   ges p.adj
##    <dbl> <chr>    <dbl> <dbl> <dbl> <dbl> <chr>   <dbl> <dbl>
##  1     0 Language     2    15 0.35  0.711 ""      0.045     1
##  2     1 Language     2    15 1.17  0.338 ""      0.135     1
##  3     2 Language     2    15 0.83  0.455 ""      0.1       1
##  4     3 Language     2    15 0.121 0.887 ""      0.016     1
##  5     4 Language     2    15 0.086 0.918 ""      0.011     1
##  6     5 Language     2    15 0.492 0.621 ""      0.062     1
##  7     6 Language     2    15 0.175 0.841 ""      0.023     1
##  8     7 Language     2    15 1.14  0.346 ""      0.132     1
##  9     8 Language     2    15 1.07  0.368 ""      0.125     1
## 10     9 Language     2    15 0.841 0.451 ""      0.101     1
```

5.2 Non-parametrische Tests

5.2.1 Mann-Whitney-U Test

Non-parametrische Alternative für t-Test für unabhängige Stichproben. Wir verwenden wieder die gefilterten Daten, nur mit Tag 1 (day_1) und gehen davon aus, dass unsere Reaktionszeiten nicht Normalverteilt sind.

```
wilcox.test(Reaction ~ Sex, data = day1)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  Reaction by Sex
## W = 44, p-value = 0.7962
## alternative hypothesis: true location shift is not equal to 0
```

5.2.2 Wilcoxon-Test

Non-parametrische Alternative für t-Test für abhängige Stichproben. Wir verwenden wieder die gefilterten Daten, nur mit Tag 1 und 9 (day1_9) und gehen davon aus, dass unsere Reaktionszeiten nicht Normalverteilt sind.

```
wilcox.test(day1_9$Reaction, day1_9$Days)
```

```
## Warning in wilcox.test.default(day1_9$Reaction, day1_9$Days): cannot compute
## exact p-value with ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  day1_9$Reaction and day1_9$Days
## W = 1296, p-value = 1.276e-13
```

```
## alternative hypothesis: true location shift is not equal to 0
```

5.2.3 Kruskal-Wallis-Test

Non-parametrische Alternative für Univariate einfaktorielle ANOVA.

```
kruskal.test(Reaction ~ Language, data = day1)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Reaction by Language
## Kruskal-Wallis chi-squared = 1.2749, df = 2, p-value = 0.5287
```

5.2.4 Friedman-Test

Non-parametrische Alternative für One-way repeated measures ANOVA. Wir verwenden hierfür die Funktion `friedman_test()`aus dem *rstatix* Paket und nehmen zur Demonstration an, dass die Reaktionszeiten nicht normalverteilt sind.

```
sleepstudy %>% friedman_test(Reaction ~ Days | Subject)
```

```
## # A tibble: 1 x 6
##   .y.          n statistic    df        p method
## * <chr>    <int>     <dbl> <dbl>    <dbl> <chr>
## 1 Reaction    18      86.1     9 9.90e-15 Friedman test
```

Zusätzlich zum p-Wert wird uns unter *statistic* der Chi-Quadrat Wert ausgegeben.

Post-Hoc-Tests:

```
sleepstudy %>%
  wilcox_test(Reaction ~ Days,
              paired = TRUE,
              p.adjust.method = "bonferroni")
```

```
## # A tibble: 45 x 9
##    .y.      group1 group2    n1    n2 statistic         p    p.adj p.adj.signif
##  * <chr>    <chr>  <chr>  <int> <int>     <dbl>     <dbl>    <dbl> <chr>
##  1 Reaction 0      1         18    18        63 0.347     1        ns
##  2 Reaction 0      2         18    18        66 0.417     1        ns
##  3 Reaction 0      3         18    18        25 0.007     0.296    ns
##  4 Reaction 0      4         18    18        19 0.002     0.105    ns
##  5 Reaction 0      5         18    18         9 0.000252  0.011    *
##  6 Reaction 0      6         18    18        16 0.001     0.058    ns
##  7 Reaction 0      7         18    18         6 0.000107  0.005    **
##  8 Reaction 0      8         18    18         6 0.000107  0.005    **
##  9 Reaction 0      9         18    18         1 0.0000153 0.000688 ***
## 10 Reaction 1      2         18    18        84 0.966     1        ns
## # ... with 35 more rows
```

5.2.5 Chi-Quadrat-Test

Der Chi-Quadrat-Test wird hier verwendet, um zu testen, ob Geschlecht und Sprache unabhängig voneinander sind.

```r
# Kreuztabelle
table(day1$Sex, day1$Language)
```

```
##
##     DE ENG FRA
##   f  0   3   6
##   m  6   3   0
```

```r
# Chi-Quadrat-Test
chisq.test(day1$Sex, day1$Language)
```

```
## Warning in chisq.test(day1$Sex, day1$Language): Chi-squared approximation may be
## incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  day1$Sex and day1$Language
## X-squared = 12, df = 2, p-value = 0.002479
```

Die Warnungmeldung "Chi-Quadrat-Approximation kann inkorrekt sein" deutet darauf hin, dass zu viele Zellhäufigkeiten (mehr als 20%) unter 5 liegen. Dazu schauen wir uns die erwarteten Häufigkeiten an.

```r
# Erwartete Häufigkeiten
chisq.test(day1$Sex, day1$Language)$expected
```

```
## Warning in chisq.test(day1$Sex, day1$Language): Chi-squared approximation may be
## incorrect
```

```
##         day1$Language
## day1$Sex DE ENG FRA
##        f  3   3   3
##        m  3   3   3
```

In diesem Fall liegen alle (>20%) Zellhäufigkeiten unter 5. Es sollte also zusätzlich der Fisher-Test gerechnet werden.

5.2.6 Fisher-Test

Zusätzlich zum Chi-Quadrat-Test bei Zellhäufigkeiten unter 5.

```r
# Fisher-Test
fisher.test(day1$Sex, day1$Language)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  day1$Sex and day1$Language
## p-value = 0.002468
## alternative hypothesis: two.sided
```

5.2.7 McNemar-Test

Zur Veranschaulichung des McNemar-Tests verwenden wir die `day1_9`-Daten und kreiieren eine neue Variable (*RT_Verteilung*), die angibt, ob die Reaktionszeit über oder unter dem Mittelwert liegt.

```
day1_9$RT_Verteilung <-
  ifelse(day1_9$Reaction >= mean(day1_9$Reaction), "higher", "lower")
```

Als nächstes sind einige Schritte notwendig:

1. Alle für den McNemar-Test unwichtigen Spalten entfernen. Wir benötigen nur die Variablen "Subject", "Days" und "RT_Verteilung".

2. Die Daten mithilfe der Funktion *pivot_wider()* [tidyr] aus dem *long format* in das *wide format* umwandeln.

3. Hier haben wir ein Problem: Unsere neuen Variablen tragen die Namen der Tage (Variable "Days"), also "1" und "9". Variablennamen die nur aus Zahlen bestehen gefallen R gar nicht. Also benennen wir sie mithilfe der colnames()-Funktion um.

4. Erstellen einer Contingency-Tabelle.

5. Ausgeben der Tabelle inklusive Summen.

6. Rechnen des McNemar-Tests

```
# Schritt 1
day1_9_mc <- day1_9[, c("Subject", "Days", "RT_Verteilung")]
# Schritt 2
day1_9_wide <- day1_9_mc %>%
  pivot_wider(names_from = Days,
              values_from = RT_Verteilung
              )
# Schritt 3
colnames(day1_9_wide) <- c("Subject", "Day1", "Day9")
# Schritt 4
day1_9_mcntest <- table(day1_9_wide$Day1, day1_9_wide$Day9)
# Schritt 5
addmargins(day1_9_mcntest)
```

```
##
##          higher lower Sum
##   higher      1     0   1
##   lower      13     4  17
##   Sum        14     4  18
```

```
# Schritt 6
mcnemar.test(day1_9_mcntest)
```

```
##
##  McNemar's Chi-squared test with continuity correction
##
## data:  day1_9_mcntest
## McNemar's chi-squared = 11.077, df = 1, p-value = 0.0008741
```

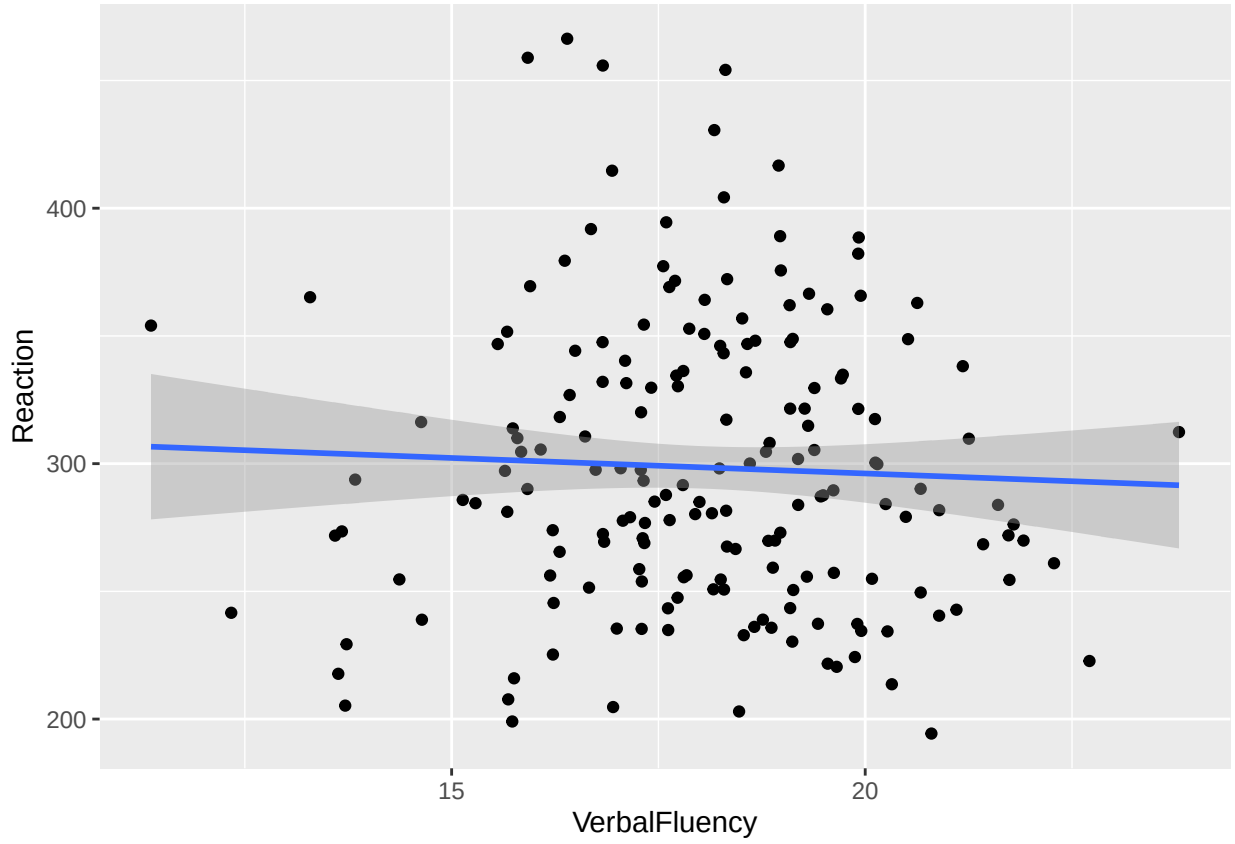
Wie wir der Tabelle entnehmen können, war die Reaktionszeit von 13 der 18 VPs an Tag 1 unter dem Mittelwert und an Tag 9 darüber, bei vier VPs blieb sie darunter und bei einer VP blieb sie darüber. Der Unterschied zwischen den beiden Tagen ist mit $p < 0.001$ signifkant.

5.3 Korrelationen

Für die grafische Untersuchung von Korrelationen empfiehlt sich ein einfacher Scatterplot mit Regressionsgerade, wie in Kapitel Regressionsgeraden beschrieben:

```
gg_sleep +
  geom_point(aes(x = VerbalFluency, y = Reaction)) +  # zu plottende Variablen
  stat_smooth(aes(x = VerbalFluency, y = Reaction),   # Regressionsgerade
              method = lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Korrelationen können mit der Funktion `cor.test()` nach dem folgenden Muster berechnet werden:
`cor.test(data$x, data$y, method = "pearson")`

Das Argument `method =` gibt an, welcher Korelationskoeffizient berechnet werden soll ("pearson", "kendall", oder "spearman").

Ein Beispiel:

```
cor.test(sleepstudy$VerbalFluency, sleepstudy$Reaction, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  sleepstudy$VerbalFluency and sleepstudy$Reaction
## t = -0.5895, df = 178, p-value = 0.5563
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
```

```
##  -0.1891836  0.1027852
## sample estimates:
##         cor
## -0.04414171
```

TIPP: Das Paket ggpubr beinhaltet weitere nützliche Funktionen zur grafischen Darstellung von Korrelationen.

5.4 General Mixed Models

Für eine Einleitung in das Thema empfehlen sich zwei kurze Tutorials zu *Linear Models* und *Linear Mixed Effect Models* von Bodo Winter:
Tutorial 1 (Linear Models):
http://www.bodowinter.com/uploads/1/2/9/3/129362560/bw_lme_tutorial1.pdf
Tutorial 2 (Linear Mixed Effects Models):
http://www.bodowinter.com/uploads/1/2/9/3/129362560/bw_lme_tutorial2.pdf

5.4.1 Linear Model

Dieses einfache lineare Modell modelliert den Einfluss von Geschlecht auf Reaktionszeit an Tag 1. Die Syntax in R lässt sich wie folgt verallgemeinern:
lm(dependent variable ~ independent variable, data)

```
# Modell definieren
sleep_lm <- lm(Reaction ~ Sex, day1)
# Ausgabe mit der Funktion summary()
summary(sleep_lm)
```

```
##
## Call:
## lm(formula = Reaction ~ Sex, data = day1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -64.136 -24.216   2.436  27.386  43.283
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   270.52      11.29  23.968 5.78e-14 ***
## Sexm          -12.05      15.96  -0.755    0.461
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.86 on 16 degrees of freedom
## Multiple R-squared:  0.03442,    Adjusted R-squared:  -0.02593
## F-statistic: 0.5703 on 1 and 16 DF,  p-value: 0.4611
```

5.4.2 Linear Mixed Effects Model

Modelliert den Einfluss unserer unabhängigen Variablen auf unsere abhängige Variable unter Berücksichtigung anderer Faktoren, wie z.B. der Zwischensubjektvariabilität.

Grober Aufbau:
dependent vairable ~ independent variable + error

Die Syntax für die Funktion lmer() in R lässt sich wie folgt verallgemeinern:
lmer(dependent variable ~ fixed effect1 + fixed effect2 + ... + (random effects), data)

Wie unterscheiden sich *fixed effect* und *random effect*?
Fixed effect:

- systematischer und vorhersagbarer Effekt auf die Daten (z.B. der Effekt von Schlafentzug in Tagen auf Reaktionszeit)
- allen für ein Experiment definierten Ausprägungen sind vorhanden (z.B. der Einfluss einer Variable "Höflichkeit" mit den Ausprägungen "unhöflich" & "höflich" auf unsere abhängige Variable)

Random effect:

- kann einen Einfluss auf die Daten haben, der nicht systematisch und vorhersagbar ist (z.B. Zwischen-subjektvariabilität)
- stell ein zufälliges Sample der Gesamtpopulation dar (z.B. unsere gewählten VPs, Items einer Condition etc.)

Aufbau des *random effects*:

- `(random slope | random intercept)`
- Nur variierender Intercept (also Variation innerhalb eines random effects): `(1|random effect)`
- Nur variierende Slope: `(0+fixed effect|random effect)`
- Variierender Intercept & variierende Slope: `(1+fixed effect|random effect)`

Random slope & intercept grafisch dargestellt:
http://mfviz.com/hierarchical-models/

Wollen wir berücksichtigen, dass sich die Subjects in ihrer Fähigkeit zu reagieren unterscheiden, sagen wir dem Modell, dass es **für jedes Subject einen anderen Intercept** annehmen soll:

```
# Modell definieren
lmm1 <- lmer(Reaction ~ Days + (1|Subject), sleepstudy)
# Ausgabe mit der Funktion summary()
summary(lmm1)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: Reaction ~ Days + (1 | Subject)
##    Data: sleepstudy
##
## REML criterion at convergence: 1786.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.2257 -0.5529  0.0109  0.5188  4.2506
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Subject  (Intercept) 1378.2   37.12
##  Residual              960.5   30.99
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
```

```
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept) 251.4051     9.7467  22.8102   25.79   <2e-16 ***
## Days         10.4673     0.8042 161.0000   13.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.371
```

So können wir beispielsweise berücksichtigen, dass manche Subjects generell schneller reagieren können als Andere.

Wollen wir berücksichtigen, dass sich die Subjects in ihrer Fähigkeit zu reagieren **nicht** unterscheiden, sie jedoch Unterschiede aufgrund des *fixed effect* (hier: Days) aufweisen, sagen wir dem Modell, dass es **für jedes Subjekt den gleichen Intercept, jedoch eine andere Slope** annehmen soll. Wir berücksichtigen also, dass manche Subjects mehr unter dem Schlafentzug leiden als andere:

```
# Modell definieren
lmm2 <- lmer(Reaction ~ Days + (0+Days|Subject), sleepstudy)
# Ausgabe mit der Funktion summary()
summary(lmm2)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: Reaction ~ Days + (0 + Days | Subject)
##    Data: sleepstudy
##
## REML criterion at convergence: 1766.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.5104 -0.5588  0.0541  0.6244  4.6022
##
## Random effects:
##  Groups   Name Variance Std.Dev.
##  Subject  Days  52.71    7.26
##  Residual      842.03   29.02
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error     df t value Pr(>|t|)
## (Intercept)   251.41       4.02 161.00  62.539  < 2e-16 ***
## Days           10.47       1.87  21.68   5.599 1.32e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.340
```

Wollen wir berücksichtigen, dass sich die Subjects in ihrer Fähigkeit zu reagieren unterscheiden **und** sie Unterschiede aufgrund des *fixed effect* (hier: Days) aufweisen, sagen wir dem Modell, dass es **für jedes Subject einen anderen Intercept und eine andere Slope** annehmen soll:

```
# Modell definieren
lmm3 <- lmer(Reaction ~ Days + (1+Days|Subject), sleepstudy)
```

```r
# Ausgabe mit der Funktion summary()
summary(lmm3)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: Reaction ~ Days + (1 + Days | Subject)
##    Data: sleepstudy
##
## REML criterion at convergence: 1743.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9536 -0.4634  0.0231  0.4634  5.1793
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 612.10   24.741
##           Days         35.07    5.922   0.07
##  Residual             654.94   25.592
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error      df t value Pr(>|t|)
## (Intercept)  251.405      6.825  17.000  36.838  < 2e-16 ***
## Days          10.467      1.546  17.000   6.771 3.26e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138
```

Typisch wäre hier z.B. auch eine Berücksichtigung der Unterschiedlichen Items nach kritischer unabhängiger Variable (hier "Condition"): `RT ~ Condition + (1+Condition|Item)`.

Interagierende *fixed effects* werden mit einem * verbunden:
`fixed effect1*fixed effect2`
Dies fasst die folgenden Notation zusammen:
`fixed effect1 + fixed effext2 + fixedeffect1:fixedeffect2`
Wobei `fixedeffect1:fixedeffect2` für die Interaktion zwischen den Effekten steht.

5.4.2.1 R-Squared

Anders als bei einem Linear Model gibt die Funktion `summary()` bei [Linear Mixed Effect Models] keine *R-Squared*-Werte (R2) aus. Um den *conditional R2* und *marginal R2* eines Modells zu erhalten, kann die Funktion `r.squaredGLMM()` aus dem "MuMIn"-Paket verwendet werden:

```r
r.squaredGLMM(lmm3)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##            R2m       R2c
## [1,] 0.2786511 0.7992199
```

Alternativ eignet sich auch die Funktion `r2()` aus dem "sjstats"-Paket.

5.4.2.2 Likelihood Ratio Test

Verschiedene Modelle lassen sich mit der Funktion `anova()` vergleichen, um herauszufinden, welches Modell unsere Daten besser beschreibt. Wir vergleichen hier die Modelle *lmm1* und *lmm3* aus dem Kapitel Linear Mixed Effects Model.

Zunächst müssen wir unsere Modelle noch um das Argument `REML = FALSE` ergänzen. Wir übernehmen die Kommandos der Modelle lmm1 und lmm3 und speichern sie mit der Ergänzung unter den Namen *lmm4* und *lmm5* ab:

```
# Ergänzen des REML = False Kommandos
lmm4 <- lmer(Reaction ~ Days + (1|Subject), sleepstudy, REML = FALSE)
lmm5 <- lmer(Reaction ~ Days + (1+Days|Subject), sleepstudy, REML = FALSE)
# Likelihood ratio test
anova(lmm4, lmm5)
```

```
## Data: sleepstudy
## Models:
## lmm4: Reaction ~ Days + (1 | Subject)
## lmm5: Reaction ~ Days + (1 + Days | Subject)
##      npar    AIC    BIC  logLik deviance  Chisq Df Pr(>Chisq)
## lmm4    4 1802.1 1814.8 -897.04   1794.1
## lmm5    6 1763.9 1783.1 -875.97   1751.9 42.139  2  7.072e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Wir sehen, dass unser Modell **lmm5** (bzw. lmm3) den niedrigeren Akaike-Informationskriterium (AIC)-Wert hat und sich signifikant vom Modell lmm1 unterscheidet. Somit hat das Modell lmm3 den besseren Fit.

Mit dieser Methode lassen sich auch Modelle vergleichen, die sich z.B. bezüglich eines weiteren *fixed effects* unterscheiden.
Der *Likelihood ratio test* wird auf den Seiten 11-14 von Bodo Winters Tutorial 2 genauer erklärt (siehe Abschnitt General Mixed Models).

5.4.2.3 Stepwise Regression

Bei einer *Stepwise Regression* wird ausgehend von einem möglichst maximalen Modell berechnet, welche *fixed* und *random effects* die Daten am besten beschreiben.
In R wird die Funktion `step()` aus dem Paket "lmerTest" wie folgt verwendet:

1. Maximales Modell erstellen
2. Funktion `step()` auf Modell anwenden und in neuer Variable speichern
3. Ergebnisse der *Stepwise Regression* ausgeben
4. Neues Modell mit der Funktion `get_model()` in neuer Variable speichern

Ein Beispiel für eine *Stepwise Regression* anhand der "ham"-Daten aus dem "lmerTest"-Paket:

```
# Schritt 1
lmm6 <- lmer(Informed.liking ~ Product*Information*Gender*Age +
             + (1|Consumer) + (1|Consumer:Product) +
             (1|Consumer:Information),
           data = ham)
# Schritt 2
step_lmm6 <- step(lmm6)
```

```
## boundary (singular) fit: see ?isSingular
```

```
# Schritt 3
step_lmm6      # Display elimination results
```

```
## Backward reduced random-effect table:
##
##                            Eliminated npar  logLik    AIC     LRT Df Pr(>Chisq)
## <none>                                   36 -1380.5 2833.0
## (1 | Consumer:Information)          1    35 -1381.3 2832.5   1.538  1    0.21491
## (1 | Consumer)                      0    34 -1382.7 2833.5   2.941  1    0.08634
## (1 | Consumer:Product)              0    34 -1464.0 2996.1 165.560  1    < 2e-16
##
## <none>
## (1 | Consumer:Information)
## (1 | Consumer)              .
## (1 | Consumer:Product)     ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Backward reduced fixed-effect table:
## Degrees of freedom method: Satterthwaite
##
##                                 Eliminated  Sum Sq Mean Sq NumDF DenDF F value
## Product:Information:Gender:Age           1  7.2065  2.4022     3   308  1.4606
## Product:Gender:Age                       2  0.6234  0.2078     3   231  0.1258
## Product:Information:Gender               3  5.8789  1.9596     3   311  1.1863
## Product:Gender                           4  0.9155  0.3052     3   234  0.1844
## Product:Information:Age                  5  7.2141  2.4047     3   314  1.4531
## Product:Age                              6  4.0334  1.3445     3   237  0.8090
## Product:Information                      7 10.3873  3.4624     3   317  2.0834
## Information:Gender:Age                   8  5.3415  5.3415     1   320  3.1817
## Information:Age                          9  0.0084  0.0084     1   321  0.0050
## Gender:Age                              10  1.2017  1.2017     1    77  0.7132
## Age                                     11  0.0254  0.0254     1    78  0.0151
## Information:Gender                      12  1.4075  1.4075     1   322  0.8353
## Gender                                  13  1.4803  1.4803     1    79  0.8789
## Product                                  0 19.3466  6.4489     3   240  3.8291
## Information                              0  6.5201  6.5201     1   323  3.8714
##                                   Pr(>F)
## Product:Information:Gender:Age 0.22530
## Product:Gender:Age             0.94478
## Product:Information:Gender     0.31505
## Product:Gender                 0.90694
## Product:Information:Age        0.22737
## Product:Age                    0.48998
## Product:Information            0.10232
## Information:Gender:Age         0.07542 .
## Information:Age                0.94376
## Gender:Age                     0.40101
## Age                            0.90262
## Information:Gender             0.36143
## Gender                         0.35135
## Product                        0.01048 *
## Information                    0.04997 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Model found:
## Informed.liking ~ Product + Information + (1 | Consumer) + (1 |
##     Consumer:Product)
```

```
# Schritt 4
final_lmm6 <- get_model(step_lmm6)
```

Kritik an Stepwise Regression (Smith 2018):

> Stepwise regression selects explanatory variables for multiple regression models based on their statistical significance. Although it has often been criticized for the misapplication of single-step statistical tests to a multi-step procedure, stepwise regression has become popular with Big Data because it is a very efficient way of choosing a relatively small number of explanatory variables from a vast array of possibilities. The assumption is that the larger the number of possible predictors, the more useful is stepwise regression. This paper uses Monte Carlo simulations to demonstrate that a stepwise procedure may choose nuisance variables rather than true variables and that the out-of-sample accuracy of the model may be far worse than the in-sample fit. These problems are more likely to be serious when there are a large number of potential predictors. Stepwise regression does not solve the problem of Big Data. Big Data exacerbates the problems of stepwise regression.

5.4.2.4 Pairwise Comparison bei Linear Mixed Effect Models

Das Paket "emmeans" ermöglicht eine *Pairwise Comparison* von linearen gemischten Modellen. Dies ist wie folgt durchzuführen:

1. Modell definieren
2. Summary des Modells ausgeben (**optional**, aber informativ)
3. Mittels der Funktion `anova()` eine allgemeinere Darstellung der Effekte ausgeben (auch **optional**)
4. Interaktionsplot mit der Funktion `emmip()` anzeigen lassen (**optional**, bei mehreren *fixed effects* hilfreich)
5. *Estimated marginal means* der für die *Pairwise Comparison* gewünschten Faktoren mit der Funktion `emmeans()` berechnen und in neuer Variable speichern
6. *Pairwise Comparison* mit der Funktion `pairs()` ausgeben lassen (das Argument `simple =` kann verwendet werden, um die Ausgabe nach einer Variable aufzuteilen)

Ein Beispiel für eine *Pairwise Comparison bei Linear Mixed Effect Models* anhand der "ham"-Daten aus dem "lmerTest"-Paket, mit zwei *fixed factors*:

```
# Schritt 1 (Einfluss von Produkt und gegebener Information auf die Bewertung)
lmm7 <- lmer(Informed.liking ~ Product*Information + (1|Consumer) , data=ham)
# Schritt 2
summary(lmm7)
```

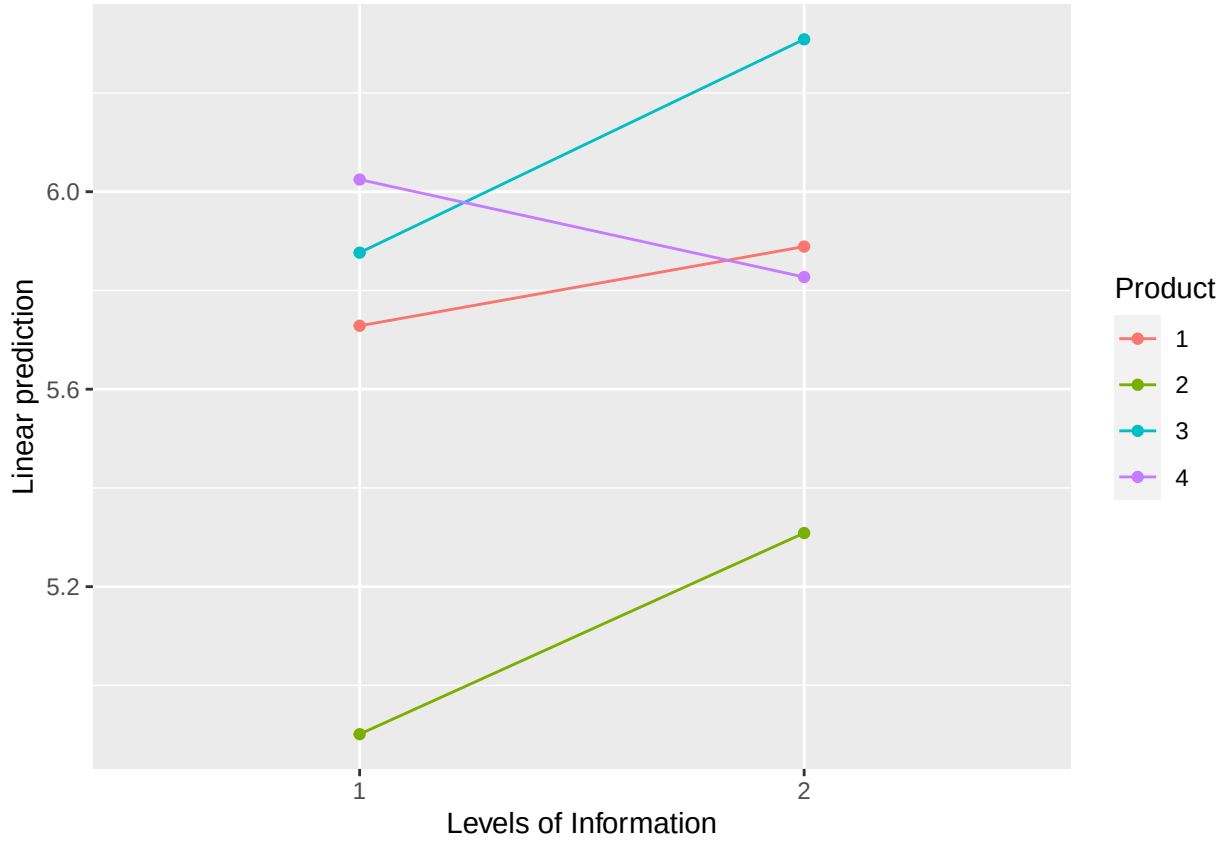
```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: Informed.liking ~ Product * Information + (1 | Consumer)
##    Data: ham
```

```
##
## REML criterion at convergence: 2870
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.5162 -0.6864  0.1000  0.7467  2.6725
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Consumer (Intercept) 0.8253   0.9085
##  Residual             4.3780   2.0924
## Number of obs: 648, groups:  Consumer, 81
##
## Fixed effects:
##                        Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)             5.7284     0.2535 544.1671  22.601   <2e-16 ***
## Product2               -0.8272     0.3288 560.0000  -2.516   0.0122 *
## Product3                0.1481     0.3288 560.0000   0.451   0.6525
## Product4                0.2963     0.3288 560.0000   0.901   0.3679
## Information2            0.1605     0.3288 560.0000   0.488   0.6256
## Product2:Information2   0.2469     0.4650 560.0000   0.531   0.5956
## Product3:Information2   0.2716     0.4650 560.0000   0.584   0.5594
## Product4:Information2  -0.3580     0.4650 560.0000  -0.770   0.4416
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Prdct2 Prdct3 Prdct4 Infrm2 Pr2:I2 Pr3:I2
## Product2    -0.649
## Product3    -0.649  0.500
## Product4    -0.649  0.500  0.500
## Informatin2 -0.649  0.500  0.500  0.500
## Prdct2:Inf2  0.459 -0.707 -0.354 -0.354 -0.707
## Prdct3:Inf2  0.459 -0.354 -0.707 -0.354 -0.707  0.500
## Prdct4:Inf2  0.459 -0.354 -0.354 -0.707 -0.707  0.500  0.500
```

```
# Schritt 3
anova(lmm7)
```

```
## Type III Analysis of Variance Table with Satterthwaite's method
##                     Sum Sq Mean Sq NumDF DenDF F value    Pr(>F)
## Product             91.807 30.6024     3   560  6.9901 0.0001271 ***
## Information          6.520  6.5201     1   560  1.4893 0.2228402
## Product:Information 10.387  3.4624     3   560  0.7909 0.4992920
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Schritt 4
emmip(lmm7, Product ~ Information)
```




```
# Schritt 5
lmm7_pw <- emmeans(lmm7, ~ Product * Information)
# Schritt 6
pairs(lmm7_pw, simple = "Product")
```

```
## Information = 1:
##  contrast estimate    SE  df t.ratio p.value
##  1 - 2      0.8272 0.329 560  2.516  0.0586
##  1 - 3     -0.1481 0.329 560 -0.451  0.9695
##  1 - 4     -0.2963 0.329 560 -0.901  0.8042
##  2 - 3     -0.9753 0.329 560 -2.966  0.0165
##  2 - 4     -1.1235 0.329 560 -3.417  0.0038
##  3 - 4     -0.1481 0.329 560 -0.451  0.9695
##
## Information = 2:
##  contrast estimate    SE  df t.ratio p.value
##  1 - 2      0.5802 0.329 560  1.765  0.2915
##  1 - 3     -0.4198 0.329 560 -1.277  0.5782
##  1 - 4      0.0617 0.329 560  0.188  0.9977
##  2 - 3     -1.0000 0.329 560 -3.042  0.0131
##  2 - 4     -0.5185 0.329 560 -1.577  0.3926
##  3 - 4      0.4815 0.329 560  1.464  0.4598
##
## Degrees-of-freedom method: kenward-roger
## P value adjustment: tukey method for comparing a family of 4 estimates
```

Hinweis: Dies funktioniert nicht, wenn der *fixed effect* (wie "Days" aus den "sleepstudy"-Daten) numerisch

ist.

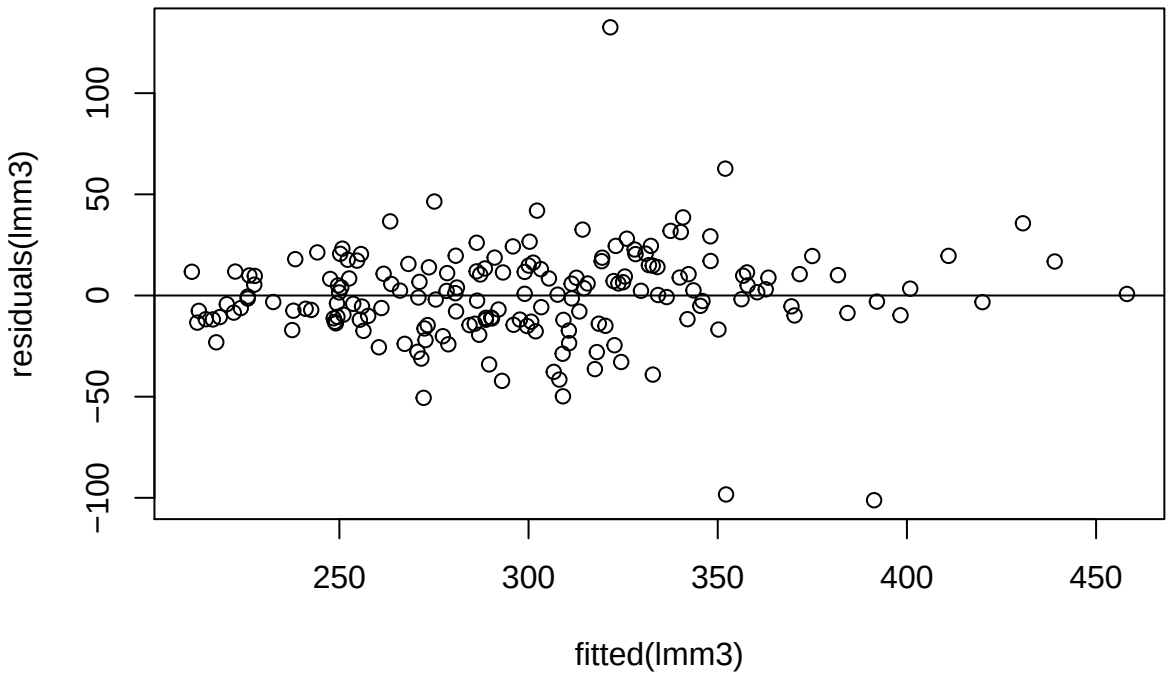
5.4.3 Voraussetzungen für General Mixed Models

Die Voraussetzungen für *Linear Models* und *Linear Mixed Effect Models* werden auf den Seiten 12-21 von Bodo Winters Tutorial 1 genauer erklärt (siehe Abschnitt General Mixed Models). Im Folgenden werden grob die Funktionen in R dargestellt:

5.4.3.1 Linearität

Um ein Modell auf Linearität zu überprüfen, werden die *residuals* wie folgt geplottet:

```
plot(fitted(lmm3), residuals(lmm3))
abline(h = 0) # horizontale Linie bei y = 0
```



Hier sehen wir lineare Daten. Nonlinear wären diese, wenn ein Kurvenmuster zu erkennen wäre.

5.4.3.2 Kollinearität

Kollinearität (also Korrelation zwischen *fixed effects*) lassen sich auf verschiedene Arten und Weisen überprüfen, z.B. können die Variablen wie in Kapitel Korrelationen untersucht werden.

Eine weitere Methode ist das Betrachten der *variance inflation factors* (VIF). Diese Werte sollten unter einem bestimmten Grenzwert liegen. Winter (2019) setzt diesen Wert bspw. bei 10 an. Die Funktion `vif()` ist im *car*-Paket enthalten und nimmt das Modell als Argument.

Zur Demonstration erstellen wir ein Modell (lmm6), welches das Modell lmm3 um den *fixed effect* "VerbalFluency" erweitert:

```
lmm8 <- lmer(Reaction ~ Days + VerbalFluency + (1+Days|Subject), sleepstudy)
vif(lmm8)
```

```
##          Days VerbalFluency
##      1.004528      1.004528
```

Die beiden *fixed effects* zeigen so gut wie keine Kollinearität. Dies ist nicht verwunderlich, da es sich hier bei "VerbalFluency" um eine frei erfundene und von uns generierte Variable handelt.

5.4.3.3 Homoskedastizität

Die Homoskedastizität (bzw. Varianzhomogenität) kann mit Hilfe des Levene Tests überprüft werden (siehe [Varianzhomogenität]):

```
leveneTest(Reaction ~ as.factor(Days), data = sleepstudy)
```

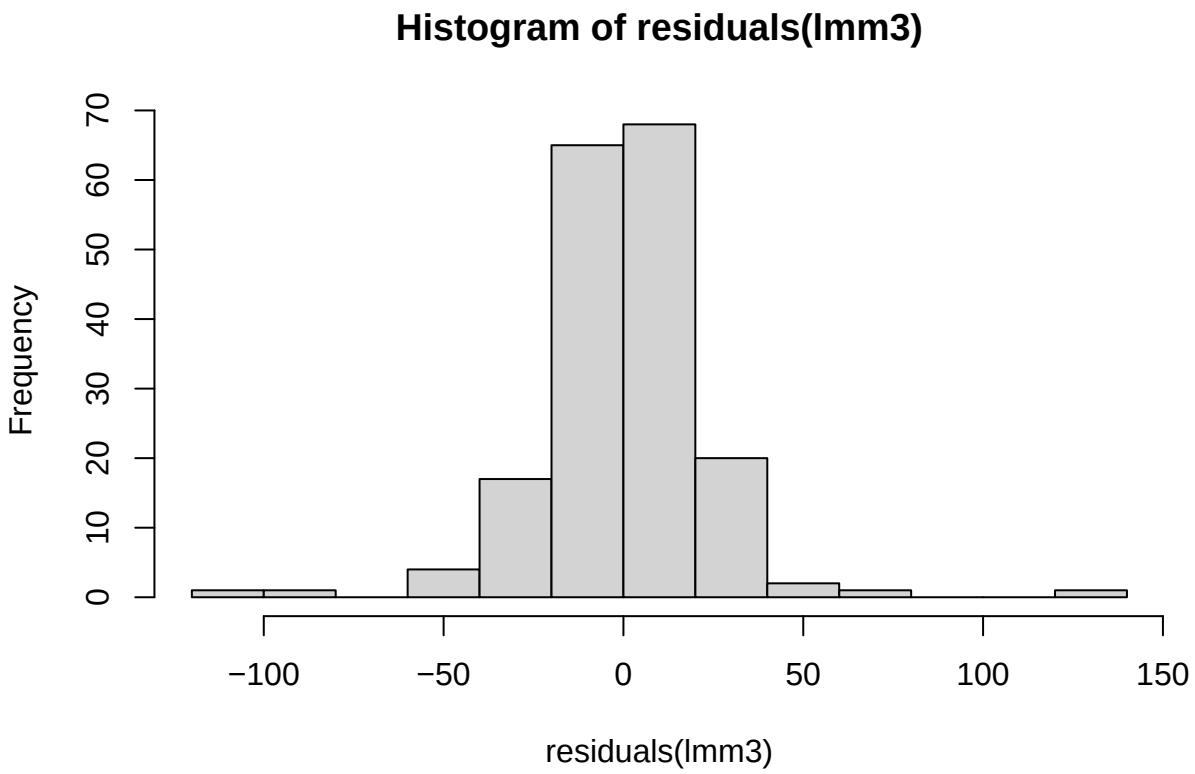
```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   9  1.3052 0.2374
##       170
```

Alternativ kann ein Residual Plot betrachten werden (siehe Linearität).
Beispiele für Residuals Plots mit Homoskedastizität und Heteroskedastizität sind auf den Seiten 17-18 von Bodo Winters Tutorial 1 zu finden (siehe General Mixed Models).

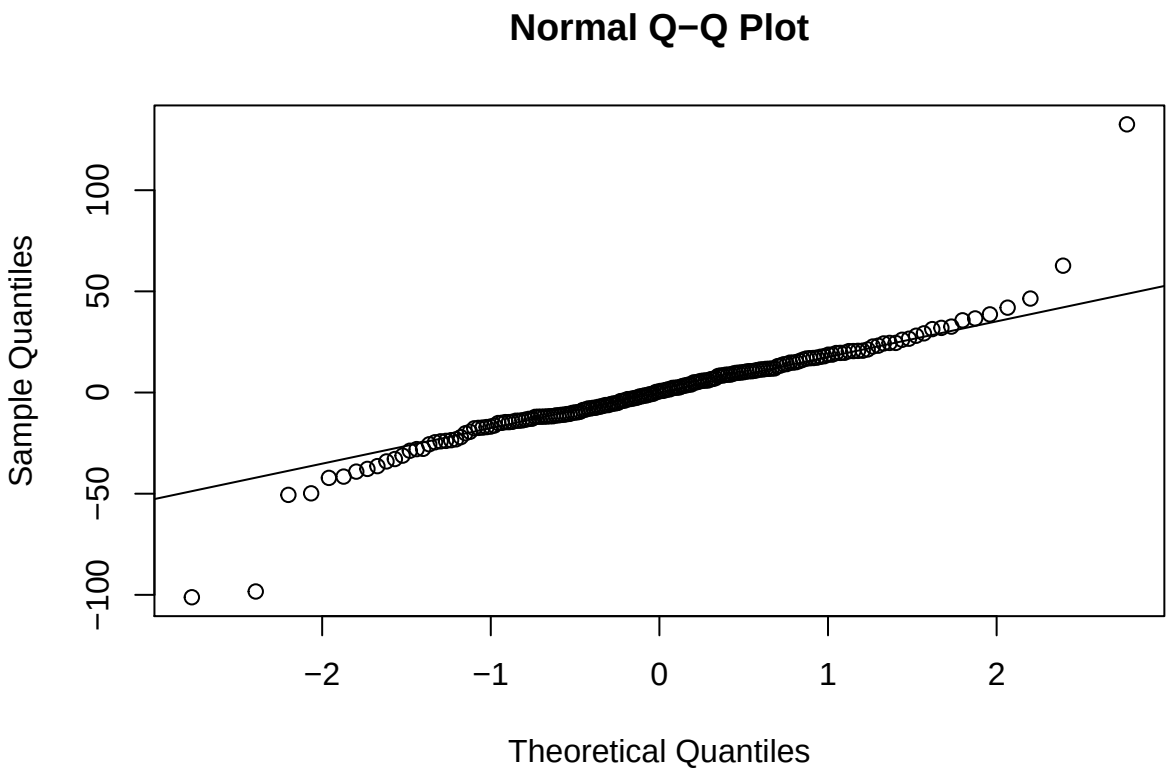
5.4.3.4 Normalverteilung der Residuals

Ob die Residuals normalverteilt sind lässt sich mit einem Histogram oder einem Q-Q Plot überprüfen:

```
# Histogramm
hist(residuals(lmm3))
```




```
# Q-Q Plot
qqnorm(residuals(lmm3))
qqline(residuals(lmm3))
```



Zudem kann der Shapiro-Wilk Test (siehe [Normalverteilung]) auf die Residuals angewendet werden:

```
shapiro.test(residuals(lmm3))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(lmm3)
## W = 0.90146, p-value = 1.408e-09
```

5.4.3.5 Influential Data

Genauere Informationen zu *influential data* siehe Nieuwenhuis et al. (2012) und Winter (2013-1).

5.4.3.5 bei Linear Models

Entsprechend Winter (2013-1) wird die Funktion `dfbeta()` verwendet, welche das Lineare Modell als Argument nimmt:

```
dfbeta(sleep_lm)
```

```
##      (Intercept)        Sexm
## 1   5.211235e-18  0.02950278
## 2  -3.915630e-16 -6.65035972
## 3   0.000000e+00 -8.01705972
## 4   0.000000e+00  5.24144028
```

```
## 5   0.000000e+00  3.31641528
## 6   0.000000e+00 -1.95710972
## 7   0.000000e+00  3.88579028
## 8   0.000000e+00  2.21656528
## 9   0.000000e+00  1.93481528
## 10  5.410371e+00 -5.41037083
## 11 -5.025767e+00  5.02576667
## 12 -3.383567e+00  3.38356667
## 13  3.691846e+00 -3.69184583
## 14  3.458883e+00 -3.45888333
## 15 -2.607417e-01  0.26074167
## 16 -4.499917e+00  4.49991667
## 17  2.399958e-01 -0.23999583
## 18  3.688958e-01 -0.36889583
```

Als Grenzwert legen wir die absolute Hälfte des Slopes fest (s. Linear Model): 12.05 / 2 = *6.025*.

Wie der Tabelle zu entnehmen ist, sind die Datenpunkte aus Reihe 2 und 3 als *influential data* zu betrachten, da ihr absoluter Wert größer als 6.025 ist.

Winter (2013-1) über den Umgang mit *influential data*:
"How to proceed if you have influential data points? Well, it's definitely not legit to simply exclude those points and report only the results on the reduced data set. A better approach would be to run the analysis with the influential points and then again without the influential points ... then you can report both analyses and state whether the interpretation of the results does or doesn't change. The only case when it is o.k. to exclude influential points is when there's an obvious error with them, so for example, a value that doesn't make sense (e.g., negative age, negative height) or a value that obviously is the result due to a technical error in the data acquisition stage (e.g., voice pitch values of 0). Influence diagnostics allow you to spot those points so you can then go back to the original data and see what went wrong."

5.4.3.5 bei Linear Mixed Effect Models

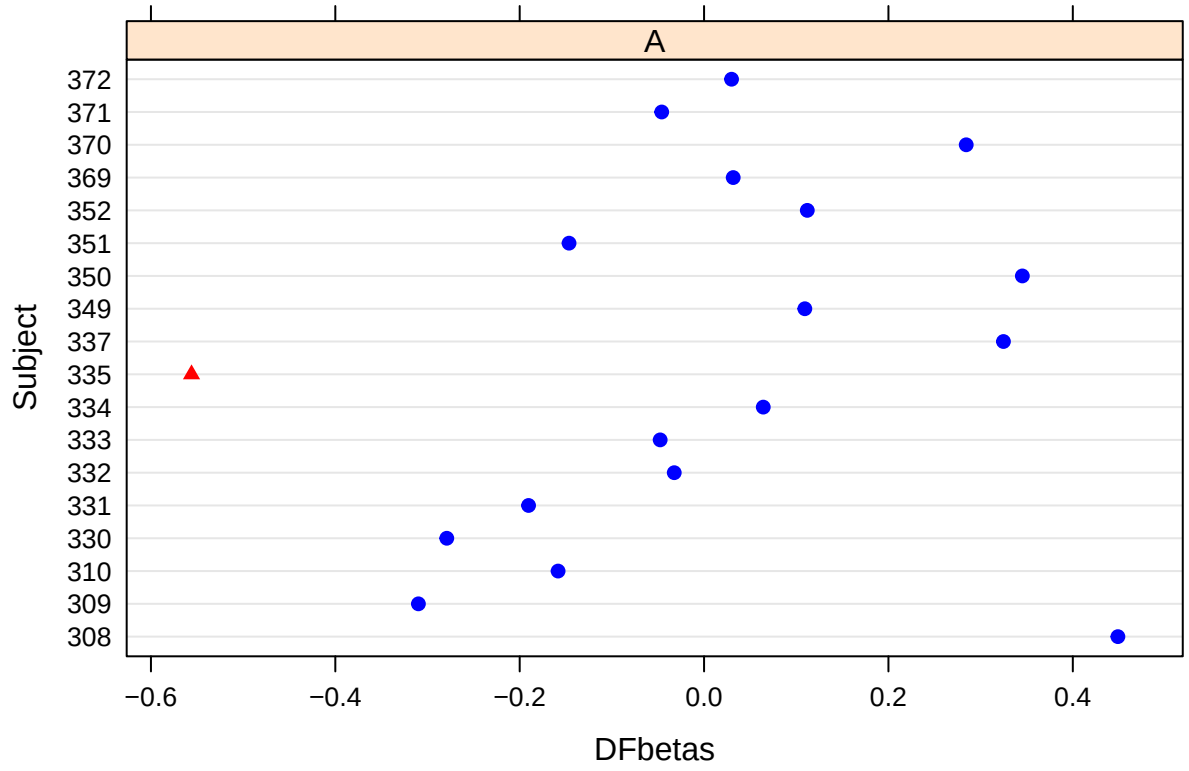
Um die *influential data* bei *Linear Mixed Effect Models* zu betrachten, kann das Paket "influence.ME" von Nieuwenhuis et al. (2012) verwendet werden.
Wir untersuchen hier, ob eine VP unsere Ergebnisse potentiell verfälscht:

```
# Influence Funktion
sleep.inf <- influence(lmm3, "Subject")
# DFBetas ausgeben
dfbetas(sleep.inf)
```

```
##     (Intercept)        Days
## 308 -0.05872589  0.44893116
## 309 -0.41087526 -0.30988234
## 310 -0.42760781 -0.15834084
## 330  0.32953913 -0.27909747
## 331  0.29209668 -0.19037019
## 332  0.10503983 -0.03232726
## 333  0.19600052 -0.04760742
## 334 -0.09178840  0.06422031
## 335  0.09498548 -0.55597555
## 337  0.33359356  0.32477727
## 349 -0.31051770  0.10931584
## 350 -0.21303733  0.34527502
## 351  0.07944720 -0.14645910
## 352  0.20776024  0.11197343
```

```
## 369  0.02896723  0.03162031
## 370 -0.35573097  0.28438839
## 371  0.01813294 -0.04593617
## 372  0.12826533  0.02982169
```

```r
# DFBetas mit Cutoff plotten
plot(sleep.inf,
     which = "dfbetas",
     parameters = 2, # nur die Zweite Spalte ("Days") plotten
     xlab = "DFbetas",
     ylab = "Subject",
     cutoff = 2/sqrt(nlevels(sleepstudy$Subject))) # nach Nieuwenhuis et al. (2012)
```



Anders als Winter (2013-1) wählen Nieuwenhuis et al. (2012) den Grenzwert für die DFBeta-Werte bei 2 geteilt durch die Wurzel von n, wobei n = die Anzahl der Ausprägungen des Gruppenfaktors. In unserem Fall: 2 geteilt durch die Wurzel von 18;
bzw. in R: '`2/sqrt(nlevels(sleepstudy$Subject))`
= 0.4714045

Wie der Tabelle und dem Plot zu entnehmen sind, liegt VP 335 über diesem Wert.

Mit der Funktion `exclude.influence()` kann die VP 335 aus dem Modell entfernt werden:

```r
# exclude.influence - Funktion
lmm3.excl <- exclude.influence(lmm3, "Subject", "335")
# Summary des neuen Modells
summary(lmm3.excl)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
```

```
## lmerModLmerTest]
## Formula: Reaction ~ Days + (1 + Days | Subject)
##    Data: data.update
##
## REML criterion at convergence: 1649.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.8160 -0.4374  0.0142  0.5066  5.0466
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 644.86   25.394
##           Days         25.59    5.059   0.16
##  Residual             685.84   26.188
## Number of obs: 170, groups:  Subject, 17
##
## Fixed effects:
##             Estimate Std. Error      df t value Pr(>|t|)
## (Intercept)  250.721      7.202  16.000  34.812  < 2e-16 ***
## Days          11.252      1.412  16.000   7.968 5.85e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.101
```

Nieuwenhuis et al. (2012) über den Umgang mit *influential data*: *"Generally, there are several strategies, including getting more data, checking data consis- tency, adapting model specification, deleting the in- fluential cases from the model, and obtaining addi- tional measurements on existing cases to account for the overly influential cases."*

5.5 Generalized Linear Models

... folgen.
Für eine erste Übersicht:
https://www.r-bloggers.com/2018/10/generalized-linear-models-understanding-the-link-function/

6 Bayesian Methods

... folgen.

---

7 Quellen

Die Quellenangabe für die genutzte R Version wird mit der Funktion `citation()` *abgefragt. Für Pakete wird einfach der Name des Pakets in der Klammer in Anführungszeichen ergänzt.*

car:
John Fox and Sanford Weisberg (2019). An {R} Companion to Applied Regression, Third Edition. Thousand Oaks CA: Sage. https://socialsciences.mcmaster.ca/jfox/Books/Companion/

ggsignif:
Constantin Ahlmann-Eltze (2019). ggsignif: Significance Brackets for 'ggplot2'. R package version 0.6.0. https://CRAN.R-project.org/package=ggsignif

lme4:
Douglas Bates, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01

lmerTest:
Kuznetsova A, Brockhoff PB, Christensen RHB (2017). "lmerTest Package: Tests in Linear Mixed Effects Models." *Journal of Statistical Software*, *82*(13), 1-26. doi: 10.18637/jss.v082.i13

influence.ME:
Rense Nieuwenhuis, Manfred te Grotenhuis and Ben Pelzer (2012). influence.ME: Tools for Detecting Influential Data in Mixed Effects Models. R Journal, 4(2): pp. 38-47.

MuMIn: Kamil Bartoń (2020). MuMIn: Multi-Model Inference. R package version 1.43.17. https://CRAN.R-project.org/package=MuMIn

R:
R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

RStudio: RStudio Team (2020). RStudio: Integrated Development Environment for R. RStudio, PBC, Boston, MA URL http://www.rstudio.com/.

rstatix:
Alboukadel Kassambara (2020). rstatix: Pipe-Friendly Framework for Basic Statistical Tests. R package version 0.6.0. https://CRAN.R-project.org/package=rstatix

tidyverse:
Wickham et al. (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

Weitere Literatur:
Smith, G. (2018). Step away from stepwise. Journal of Big Data, 5(1), 32. https://doi.org/10.1186/s40537-018-0143-6

Winter, B. (2013-1). Linear models and linear mixed effects models in R: Tutorial 1. http://www.bodowinter.com/uploads/1/2/9/3/129362560/bw_lme_tutorial1.pdf

Winter, B. (2013-2). A very basic tutorial for performing linear mixed effects analyses (Tutorial 2). http://www.bodowinter.com/uploads/1/2/9/3/129362560/bw_lme_tutorial2.pdf

Winter, B. (2019). Statistics for linguists: An introduction using R. Routledge.

Diese Guide wurde zuletzt mit den folgenden Versionen getestet:
Um diese Informationen abzurufen `sessionInfo()` *in der Konsole ausführen.*

R version 4.0.2 (2020-06-22)
RStudio version 1.3.1073 Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS 10.16

attached base packages:
stats, graphics, grDevices, utils, datasets, methods, base

other attached packages:
plyr_1.8.6, emmeans_1.5.0, influence.ME_0.9-9, rstatix_0.6.0, car_3.0-9, carData_3.0-4, forcats_0.5.0,

stringr_1.4.0, dplyr_1.0.2, purrr_0.3.4, readr_1.3.1, tidyr_1.1.2, tibble_3.0.4, ggplot2_3.3.2, tidyverse_1.3.0, lmerTest_3.1-2, lme4_1.1-23, Matrix_1.2-18, ggsignif_0.6.0, MuMIn_1.43.17
