Einführung in die Programmiersprache

Verwendung & Installation

Die Programmiersprache R wird meistens für die Daten-Analyse, Visualisierungen, Maschinelles Lernen und das Manipulieren von Datensätzen verwendet. Das zu R äquivalente Python-Paket ist pandas.

Die Programmiersprach R kann über CRAN (https://cran.r-project.org) heruntergeladen werden. Anschließend wird der Download von RStudio als Entwicklungsumgebung empfohlen.

Grundlagen

In R werden die Anweisungen Zeile für Zeile ausgeführt. Um Werte zu speichern, können Variablen mittels `var <- expression` angelegt werden.
Die folgende Tabelle listet grundlegende Methoden auf, die bereits in R eingebaut sind.

Methoden	Beschreibung
`seq(...)`	Erstelle eine Sequenz von Start- bis Endwert
`c(...)`	Kombiniere mehrere Werte und Sequenzen zu einer Sequenz
`rep(...)`	Wiederhole eine Sequenz mehrmals
`rm(...)`	Entferne eine Variable aus dem Speicher
`getwd()`	Lese das aktuelle Verzeichnis aus
`setwd(...)`	Setze das Working Directory
`str(...)`	Zeige die Struktur eines Objekts an
`sqrt(...)`	Wurzel aus einer Zahl ziehen
`a ** b`	Berechne a hoch b

> **Hinweis**
>
> Mit dem Ausdruck `?method` öffnet sich in RStudio automatisch ein Fenster mit der Dokumentation. Hier wird die Methode samt Parametern und Rückgabewert in natürlicher Sprache erklärt.

Pakete

In R gibt es Pakete, die nachträglich installiert werden können. Sie bieten weitere Funktionen an.

Bekannte Pakete:

- tidyverse
- ggplot2
- psych
- readxl

Paket installieren

```
install.packages("car")
```

Paket importieren

```
library(car)
```

Datensätze erstellen

Datensätze werden in R in Dataframes (Abkürzung `df`) gespeichert. Sie sind die Hauptdatenstruktur in R. Anstelle von Zeilen und Spalten wird auch von Merkmalsträgern und Variablen gesprochen.

Es gibt mehrere Wege, um Datensätze zu erstellen:

- Datensatz selbst erstellen
- Datensatz als Datei einlesen
- In R-eingebauten Datensatz laden
- Paket-spezifischen Datensatz laden

Datensatz selbst erstellen

```
names <- c("Tina", "Tim", "Sara")
genders <- c("w", "m", "w")
ages <- c(22, 19, 24)

df <- data.frame(
  Name=names,
  Geschlecht=genders,
  Alter=ages
)
```

	Name	Geschlecht	Alter
1	Tina	w	22
2	Tim	m	19
3	Sara	w	24

Datensatz einlesen

```
library(readxl)
df <- read_excel("./data.xlsx")
# für .Rdata
load("./data.Rdata")
```

> **Hinweis**
>
> Relative Pfade werden ausgehend vom Working Directory gesucht. Setzen und Prüfen des Working Directories ist mit den Befehlen `getwd()` und `setwd("...")` möglich.

In R-eingebauten Datensatz laden

In R gibt es z.B. den Motor Trend Car Road Tests Datensatz, der direkt als Dataframe verwendet werden kann.

```
data(mtcars)
```

Paket-spezifischen Datensatz laden

Manche Pakete beinhalten eigene Datensätze. Zum Beispiel bietet das Paket `car` einen Datensatz über die Passagiere der Titanic im Dataframe `TitanicSurvival` an.

```
library(car)
data("TitanicSurvival")
```

Datensatz-Struktur analysieren

Bevor man mit den geladenen Daten arbeitet, ist es wichtig sich mit der Struktur der Daten auseinanderzusetzen. Die folgenden Methoden können dabei helfen.

Beschreibung der Daten und Erklärung der Variablen (falls vorhanden)

```
?df
```

Ausgabe der ersten 5 Merkmalsträger

```
head(df)
```

Ausgabe des kompletten Datensatzes

```
View(df)
# alternativ
df
```

Ausgabe der Variablennamen

```
colnames(df)
```

Ausgabe der Variablen mit ihren Datentypen

```
str(df)
```

Anzahl der Merkmalsträger

```
nrow(df)
```

> **Hinweis**
>
> Es kann hilfreich sein sich die hinterlegten Datentypen anzuschauen. Gerade bei importierten Datensätzen fehlen manchmal Typ-Informationen.

Auf Daten zugreifen

Der Zugriff auf Daten eines Datensatzes erfolgt über eckige Klammern mit zwei Werten [a, b]. Der erste Wert spezifiziert die Zeilen, der zweite Wert die Spalten.
Folgende Werte können angenommen werden:

Bezeichner	Beispiel	Beschreibung
Index	`df[1,3]`	Der Wert, der in der ersten Zeile und 3. Spalte steht
Leer	`df[1, ]`	Alle Werte, die in der ersten Spalte stehen
Mehrere Indizes	`df[ c(1, 3, 5), ]`	Alle Werte, die in der 1., 3., und 5. Zeile stehen

Boolean	`df[ c(TRUE, FALSE, TRUE), ]`	Bei einem Datensatz von 3 Zeilen werden nur die Zeilen mit TRUE ausgegeben.

> **Hinweis**
>
> Die Boolean-Schreibweise ist nur theoretisch und dient als Grundlage für das Filtern von Daten.

Kurzschreibweise für Spalten-Zugriff

```
df$NameDerVariablen
```

Meistens möchte man nach Eigenschaften filtern, z.B. alle Personen ausgeben, die weiblich sind. Daher kann man Vergleichsoperationen direkt auf Spalten ausführen. Das Ergebnis ist eine Liste an Booleans, die man dann für den Zugriff verwenden kann.

> **Hinweis**
>
> Neben Vergleichsoperationen sind alle entsprechende Datentyp-Operationen (z.B. * 2 für Zahlen) gültig.

Ausgabe aller weiblichen Personen

```
df[df$Geschlecht == "w", ]
```

Ausgabe des Alters aller weiblichen Personen

```
df$Alter[df$Geschlecht == "w"]
```

Datensätze manipulieren

Variable „Alter in Tagen“ anlegen

```
df$AlterInTagen <- df$Alter * 365
```

Variable löschen

```
df$Alter <- NULL
```

Datensatz aufsteigend nach dem Alter sortieren

```
df <- df[order(df$Alter), ]
```

Gewicht auf zwei Nachkommastellen runden

```
df$Gewicht <- round(df$Gewicht, 2)
```

Fallunterscheidung für eine Variable

```
df$Bestanden <- ifelse(
    df$Punkte > 10,
    "ja", "nein"
)
```

Deskriptive Analyse

Um einen Überblick über die Daten und ihre statistischen Kennwerte zu erhalten, kann man Zusammenfassungen ausgeben.

Anzahl, Mittelwert, Standard-Abweichung, Median, usw. aller Variablen

```
library(psych)
describe(df)
```

Alternativ kann man sich die konkreten Kennwerte auch für einzelne Variablen berechnen lassen.

Mittelwert

```
mean(df$Alter)
```

Median

```
median(df$Alter)
```

Varianz

```
var(df$Alter)
```

Standard-Abweichung

```
sd(df$Alter)
```

Spannweite

```
range(df$Alter)
```

Modalwert

```
which.max(table(df$Alter))
```

> **Hinweis**
>
> Bei den meisten Methoden kann man den Parameter `na.rm=TRUE` setzen. Dadurch werden Werte, die NaN sind, ausgelassen.

Um Häufigkeiten darzustellen, können Tabellen und Histogramme verwendet werden.

Häufigkeitstabelle

```
table(df$Geschlecht)
```

Relative Häufigkeitstabelle

```
prop.table(table(df$Geschlecht))
```

Histogramm

```
hist(df$Geschlecht)
```

Korrelation

Der Kennwert „Korrelation“ gibt an, wie stark der Zusammenhang zweier Variablen ist. Der Korrelationswert liegt zwischen -1 und 1.
Bei negativem Vorzeichen spricht man von einem negativen Zusammenhang bzw. einem positiven Zusammenhang für ein positives Vorzeichen.
Ab einem absoluten Wert von 0.5 spricht man von einem starken Zusammenhang.

Die Berechnung des Korrelationskoeffizienten hängt von den Skalen der Variablen ab. Bei ordinal-skalierten Variablen (z.B. Likert-Skalen) wird Kendalls Tau verwendet. Bei Intervall-skalierten Variablen (z.B. Gewicht) kann Pearson verwendet werden.

Pearson-Korrelationskoeffizienten berechnen

```
cor(df$Alter, df$Gewicht,
    use="complete.obs",
    method= "pearson")
```

Signifikanz der Korrelation testen

```
cor.test(df$Alter, df$Gewicht,
    use="complete.obs",
    method= "pearson")
```

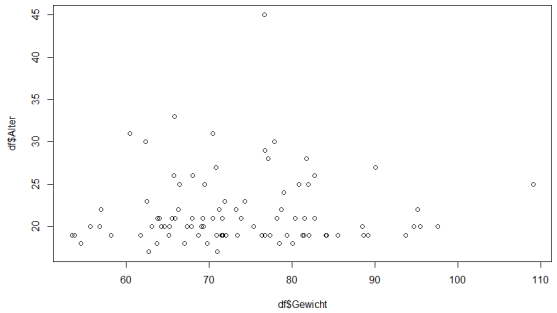
> **Hinweis**
>
> Kendalls Tau kann mit `method="kendall"` berechnet werden.

Visualisierung von Daten

In R kann man mit bereits wenigen Zeilen die Daten eines Datensatzes visualisieren und darstellen.

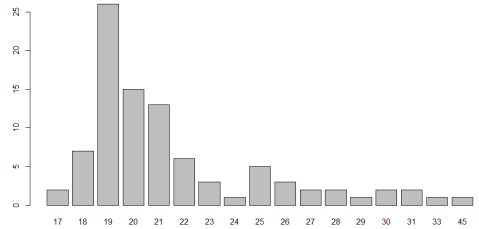
Streudiagramm (x-Achse: Gewicht, y-Achse: Alter)

```
plot(df$Alter ~ df$Gewicht)
```



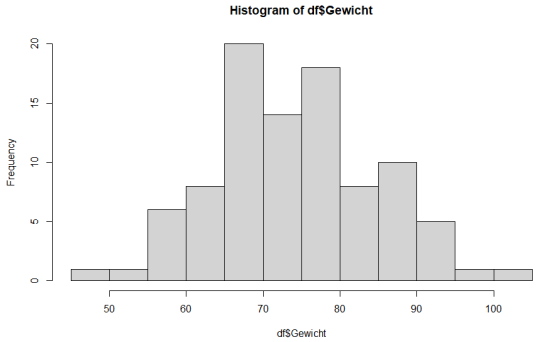
Balkendiagramm (y-Achse: Anzahl der Merkmalsträger)

```
barplot(table(df$Alter))
```



Histogramm

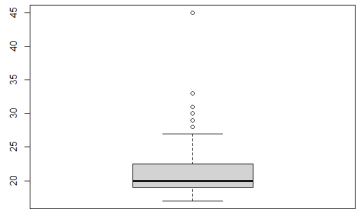
```
hist(df$Gewicht)
```



Das Boxplot ist ein Diagramm, das mehrere Kennwerte visualisiert. Neben dem Median (schwarzer Balken), werden auch das obere und untere Quantil (graue Box) und Ausreiser (Kreise) dargestellt.

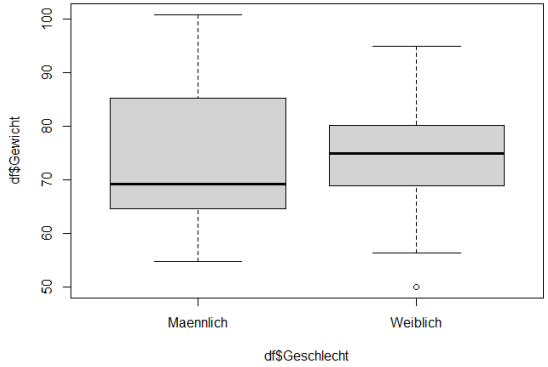
Boxplot

```
boxplot(df$Alter)
```



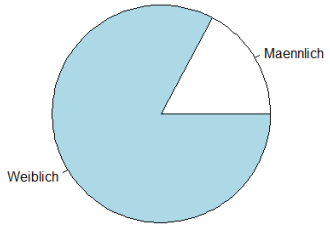
Boxplot in Abhängigkeit einer Gruppierung

```
boxplot(df$Alter)
```



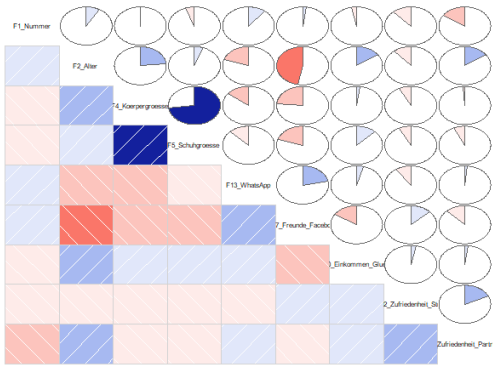
Tortendiagramm

```
pie(table(df$Geschlecht))
```



Visualisierung der Korrelation zwischen allen Variablen

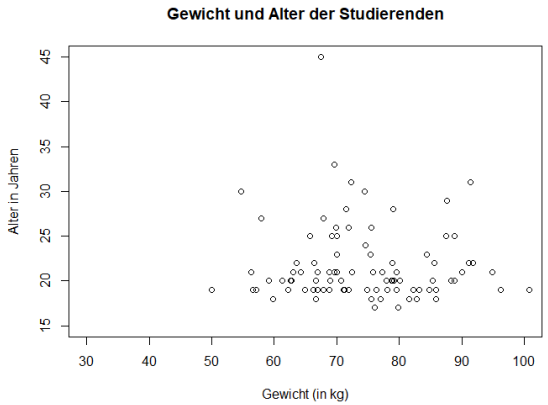
```
library(corrgram)
corrgram(df, upper.panel=panel.pie)
```



Es ist möglich Teile des Plots anzupassen. Zum Beispiel kann man die Achsenbeschriftung und Titel des Plots ändern. Außerdem kann man die Achsen skalieren und Punkte färben.

Angepasstes Streudiagramm

```
plot(df$Alter ~ df$Gewicht,
    main = "Gewicht und Alter der
Studierenden",
    xlab = "Gewicht (in kg)",
    ylab = "Alter in Jahren",
    xlim = c(30, 100),
    ylim = c(15, 45)
)
```



> **Hinweis**
>
> Das Paket `ggplot2` bietet weitere Funktionalitäten, um individuelle Darstellungen zu ermöglichen.

t-Test für unabhängige Stichproben

Bei dem t-Test für unabhängige Stichproben testet man die unabhängigen Stichproben zweier Gruppen. Dazu müssen die Stichproben Intervall-skaliert sein. Konkret wird überprüft, ob sich 2 unabhängige Gruppen hinsichtlich ihrer Mittelwerte unterscheiden.
Dabei wird zwischen gerichtet und ungerichtet unterschieden. Bei einem gerichteten t-Test hat man bereits eine Vermutung, ob ein Mittelwert höher bzw. niedriger ist, z.B. „Frauen sind klüger als Männer“. Bei einem ungerichteten t-Test stellt man die Hypothese auf, dass sich die Mittelwerte nur unterschieden, z.B. „Es gibt einen Unterschied der Intelligenz bzgl. Männer und Frauen“.

Um eine Hypothese mit dem t-Test zu testen, sind folgende Schritte nötig:

1. Daten deskriptiv analysieren
2. Voraussetzungen prüfen
3. Test durchführen
4. Ergebnis interpretieren

Für Schritt 1 kann man die Methoden der deskriptiven Analyse anwenden.

Danach müssen folgende Voraussetzungen geprüft werden:

- Die abhängige Variable muss intervallskaliert sein.
- Die Messwerte müssen unabhängig voneinander sein.
- Stichprobenkennwerteverteilungen sind normalverteilt:
  - für n ≥ 30: gegeben durch das Grenzwerttheorem
  - für n < 30: Messwerte der abhängigen Variablen müssen in beiden Gruppen normalverteilt sein
- Varianzhomogenität muss gegeben sein.

Die ersten beiden Voraussetzungen können nicht mit R überprüft werden. Sie müssen bereits beim Sammeln der Daten sichergestellt werden.

Für die 3. Voraussetzung kann man zunächst prüfen, ob man mehr als 30 Stichproben hat.

In den folgenden Code-Auszügen wird die Hypothese „Männer sind größer als Frauen“ getestet.

Anzahl der Stichproben ermitteln

```
library(psych)

describeBy(
    df$Groesse,
    df$Geschlecht,
    mat=TRUE
)
```

Falls das nicht gegeben ist, muss der Shapiro-Test verwendet werden.

Shapiro-Test für beide Gruppen

```
maenner <- df$Groesse[
     df$Geschlecht == "m"
]
```

```
frauen <- df$Groesse[
     df$Geschlecht == "w"
]

shapiro.test(maenner)
shapiro.test(frauen)
```

Wenn das Ergebnis des Tests nicht signifikant ist, also p-Wert ≥ 0.05 ist, dann liegt die geforderte Normalverteilung vor.

> **Hinweis**
>
> Diese Schlussfolgerung gilt für alle Tests auf eine bestimmte Voraussetzung. Wenn der Test signifikant ist, dann ist die Voraussetzung verletzt. Ist der Test nicht signifikant, dann wird die Voraussetzung erfüllt.

Zuletzt wird auf Varianzhomogenität mithilfe des Levene-Tests getestet.

Levene-Test ausführen

```
library(car)

leveneTest(
    Groesse ~ Geschlecht,
    data=df,
    center=mean
)
```

Beachte, dass bei dem Levene-Test ein Signifikanzniveau von 10% anstelle der typischen 5% genommen wird.
Als Test für eine Voraussetzung gilt erneut: Nicht signifikanter Test impliziert Voraussetzung erfüllt.

Sollte die Voraussetzung nicht erfüllt sein, muss im folgenden Schritt die Welch-Korrektur verwendet werden.

Für Schritt 3 kann man die folgenden Methoden für einen ungerichteten Test verwenden.

ungerichteter t-Test ohne Welch-Korrektur

```
t.test(
    Groesse ~ Geschlecht,
    data=df,
    var.equal=TRUE,
    na.action=na.exclude
)
```

ungerichteter t-Test mit Welch-Korrektur

```
t.test(
    Groesse ~ Geschlecht,
    data=df,
    var.equal=FALSE,
    na.action=na.exclude
)
```

Wenn man gerichtet testen möchte, muss man die „Richtung“ angeben. Dazu ist es entscheidend, wie die Reihenfolge der Ausprägungen ist. In dem Beispiel ist es wichtig, ob zuerst „m“ oder „w“ auftaucht. Je nachdem muss man

Reihenfolge der Ausprägungen bestimmen

```
levels(df$Geschlecht)
```

Wir nehmen an, dass die Ausgabe

```
[1] "m" "w"
```

ist.

Um nun zu prüfen, ob Männer größer sind, braucht man den Vergleich `greater`, da `"m"` an erster Stelle steht.

gerichteter t-Test

```
t.test(
    Groesse ~ Geschlecht,
    data=df,
    alternative = "greater",
    var.equal=FALSE,
    na.action=na.exclude
)
```

Um in Schritt 4 den Test korrekt interpretieren zu können, muss man zunächst untersuchen, ob der Test signifikant ist. Wenn der p-Wert unter 5% liegt, ist der Test signifikant. Es liegt also ein Unterschied vor.

Wenn der Test nicht signifikant ist, kann kein Unterschied festgestellt werden.

Sollte der Test signifikant sein, lässt sich zusätzlich die Effektgröße d angeben. Nach Cohen (1988) ergibt d = 0.2 einen kleinen Effekt, d = 0.5 einen mittleren Effekt und ab d = 0.8 einen starken Effekt.

Gruppen-Größen bestimmen

```
table(df$Geschlecht)
```

Effektgröße berechnen

```
# n = #Gruppe 1
# m = #Gruppe 2
test_res <- t.test(...)
t <- test_res$statistic
d <- t * sqrt(1/n + 1/m)
```

t-Test für abhängige Stichproben

Bei dem t-Test für abhängige Stichproben testet man die abhängigen Stichproben zweier Gruppen. Dazu müssen die Stichproben intervallskaliert sein. Konkret wird überprüft, ob sich 2 abhängige Gruppen hinsichtlich ihrer Mittelwerte unterscheiden.

Das Vorgehen für den abhängigen t-Test entspricht nahezu dem unabhängigen t-Test.

Daher werden nur die Unterschiede aufgelistet und erklärt.

Die Voraussetzungen 1 und 2 stimmen überein. Danach müssen folgende Voraussetzungen gegeben sein:

- Stichprobenkennwerteverteilung der Differenzen ist normalverteilt:
  - für n ≥ 30 pro Gruppe: gegeben durch das Grenzwerttheorem
  - für n < 30: Differenzen der Werte der abhängigen Variablen müssen normalverteilt sein.
- Die Messwerte müssen positiv miteinander korrelieren.

Die Gruppen-Größen können wie unter „Gruppen-Größen bestimmen“ ermittelt werden. Sollten die Anzahl nicht erreicht sein, muss die Differenz auf Normalverteilung mit dem Shapiro-Test getestet werden.

Differenz berechnen

```
maenner <- df$Groesse[
    df$Geschlecht == "m"
]
frauen <- df$Groesse[
    df$Geschlecht == "w"
]
differenz <- maenner - frauen
```

Anschließend kann der Shapiro-Test auf `differenz` ausgeführt werden.

Zuletzt müssen die Gruppen `maenner` und `frauen` miteinander positiv korrelieren (siehe Korrelation).

Im Unterschied zum unabhängigen t-Test muss der Parameter `paired=TRUE` gesetzt werden.

ungerichteter t-Test

```
t.test(
    Groesse ~ Geschlecht,
    data=df,
    paired=TRUE,
    na.action=na.exclude
)
```

Der letzte Unterschied liegt in der Effektgröße d, die sich wie folgt berechnen lässt:

Effektgröße berechnen

```
test_res <- t.test(...)
t <- test_res$statistic
df <- test_res$parameter
d <- t / sqrt(df)
```

Regression

In R kann man ohne viel Aufwand ein lineares Regressionsmodell erstellen, das auch mehrere Variablen miteinbeziehen kann.

Im Folgenden betrachten wir den Fall, dass wir das Gewicht anhand des Alters (und der Größe) vorhersagen wollen (auch, wenn die Daten dafür zu keinem guten Ergebnis führen werden).

Modelle erstellen

```
model <- lm(
    Gewicht ~ Alter,
    data=df

)
# Modell mit mehreren Variablen
model_ext <- lm(
    scale(Gewicht) ~ scale(Alter) +
scale(Groesse),
    data=df
)
```

Informationen über das Modell erhalten

```
summary(model)
```

Mithilfe der (nicht-skalierten) Koeffizienten lässt sich eine interpretierbare Regressionsgleichung der Form

$$y = a \cdot Alter + b \cdot Größe + c$$

aufstellen.

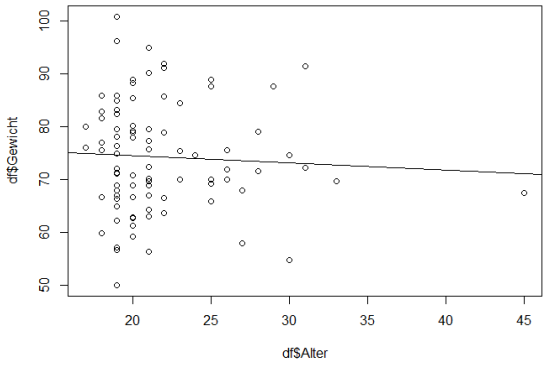
Koeffizienten erhalten

```
summary(model)$coefficients
```

Die Regressionsgerade lässt sich zusammen mit den zugrundeliegenden Daten graphisch darstellen.

Regressionsgerade plotten

```
plot(df$Alter, df$Gewicht)
abline(model)
```



Basierend auf dem Modell lässt sich nun das Gewicht anhand des Alters mithilfe des Modells `model` vorhersagen.

Vorhergesagtes Gewicht im Alter von 40 Jahre

```
alter <- data.frame(Alter=40)
predict(model, alter)
```

> **Hinweis**
>
> Das funktioniert nur für nicht-skalierte Koeffizienten.

Um zu schauen, wie akkurat das Modell Vorhersagen trifft, gibt es die Vorhersagegüte R^2. Ein Score von 1 bedeutet eine perfekte Vorhersage, wobei ein Score von 0 und negativen Werten für eine schlechte Vorhersage spricht.

R^2 bestimmen

```
summary(model)$r.squared
```

Je mehr Variablen man für die Vorhersage verwendet, desto mehr Varianz klärt man auf und dementsprechend höher wird die Vorhersagegüte. Allerdings besteht die Gefahr des Overfittings, sodass die Robustheit des Modells abnimmt.

Um die Robustheit zu bestimmen, werden die Daten in einen Trainings- und Test-Datensatz aufgetrennt. Anschließend wird auf beiden Daten der MSE (Mean Squared Error) berechnet. Je kleiner die absolute Differenz ist, desto robuster das Modell.

Hinweis: Der typische Split ist 80:20, also 80%

Hinweis

Der typische Split ist 80:20, also 80% der Daten werden für das Trainieren des Modells verwendet, während 20% zur Überprüfung der Robustheit verwendet werden.

Robustheit bestimmen

```
# Split in Trainings- und Test-
Daten
train_size <- 0.8 * nrow(df)
train <- sample(
    1:nrow(df), train_size
)
# Modell erstellen
model <- lm(..., subset=train)
# MSE berechnen
MSE.train <- mean(
(
    scale(df$Gewicht) -
    predict(model, df)
)[train] ^ 2)
MSE.test <- mean(
(
    scale(df$Gewicht) -
    predict(model, df)
)[-train] ^ 2)

# Differenz bilden
MSE.train – MSE.test
```