Universität Augsburg University

Wirtschaftsingenieurwesen

R-Tutorium

Amelie Schischke, Marcus Röckl

16.04.2026

Universität Augsburg University

Materials Resource Management

Wirtschaftsingenieur

Agenda

1. Allgemeines
2. Basics
3. Packages
4. Datenimport und Datenexport
5. Deskriptive Statistik
6. Plots
7. Regressionsanalyse
8. Weiterführende Literatur

Agenda

Allgemeines

Was ist R?

Kostenfreie, plattformübergreifende Open-Source-Software:

- Optimiert für statistische Programmierung
- Ständige Weiterentwicklung durch Community
- Keine offiziellen Handbücher oder Support

ABER:

- Vignetten / Paper / Dokumentationen zu Packages
- Zahlreiche Tutorials / Foren
- Bücher (häufig kostenfrei über Uni)

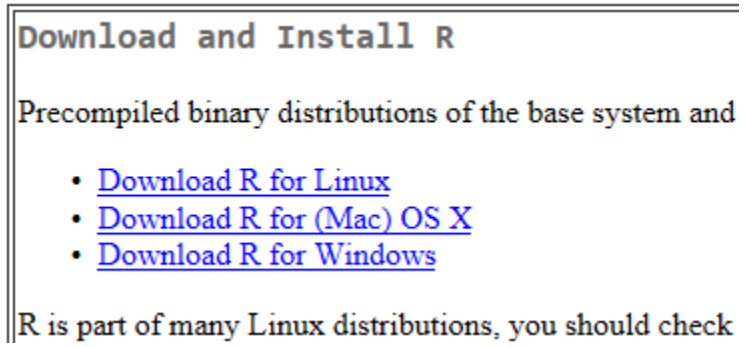
Allgemeines

Was ist R?

Download von R unter: https://cran.rstudio.com/

R ist ein ausführbares Programm und kann in dieser Form bereits verwendet werden. Es gibt graphische Oberflächen, die den Umgang mit R deutlich erleichtern:

- Download von RStudio unter: https://posit.co/download/rstudio-desktop
- ACHTUNG: Unbedingt zuerst R installieren, danach RStudio.
- RStudio ist eine IDE (integrated development environment).

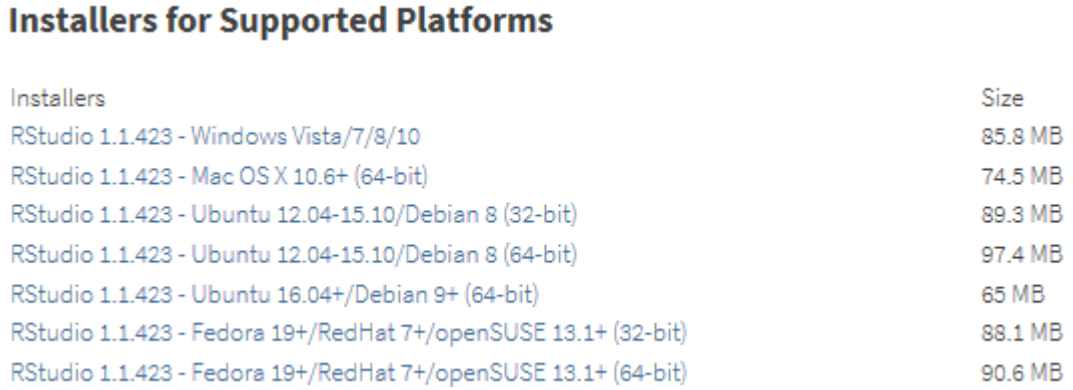


Download and Install R

Precompiled binary distributions of the base system and

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check



Installers for Supported Platforms

Installers	Size
RStudio 1.1.423 - Windows Vista/7/8/10	85.8 MB
RStudio 1.1.423 - Mac OS X 10.6+ (64-bit)	74.5 MB
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB
RStudio 1.1.423 - Ubuntu 16.04+/Debian 9+ (64-bit)	65 MB
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB

Allgemeines

Was ist R?

Einen ersten Einblick zur Logik von Programmierung gibt folgendes Zitat:

> *„Alles, was existiert, ist ein Objekt.*
> *Alles, was passiert, ist ein Funktionsaufruf.“*
> - John M. Chambers

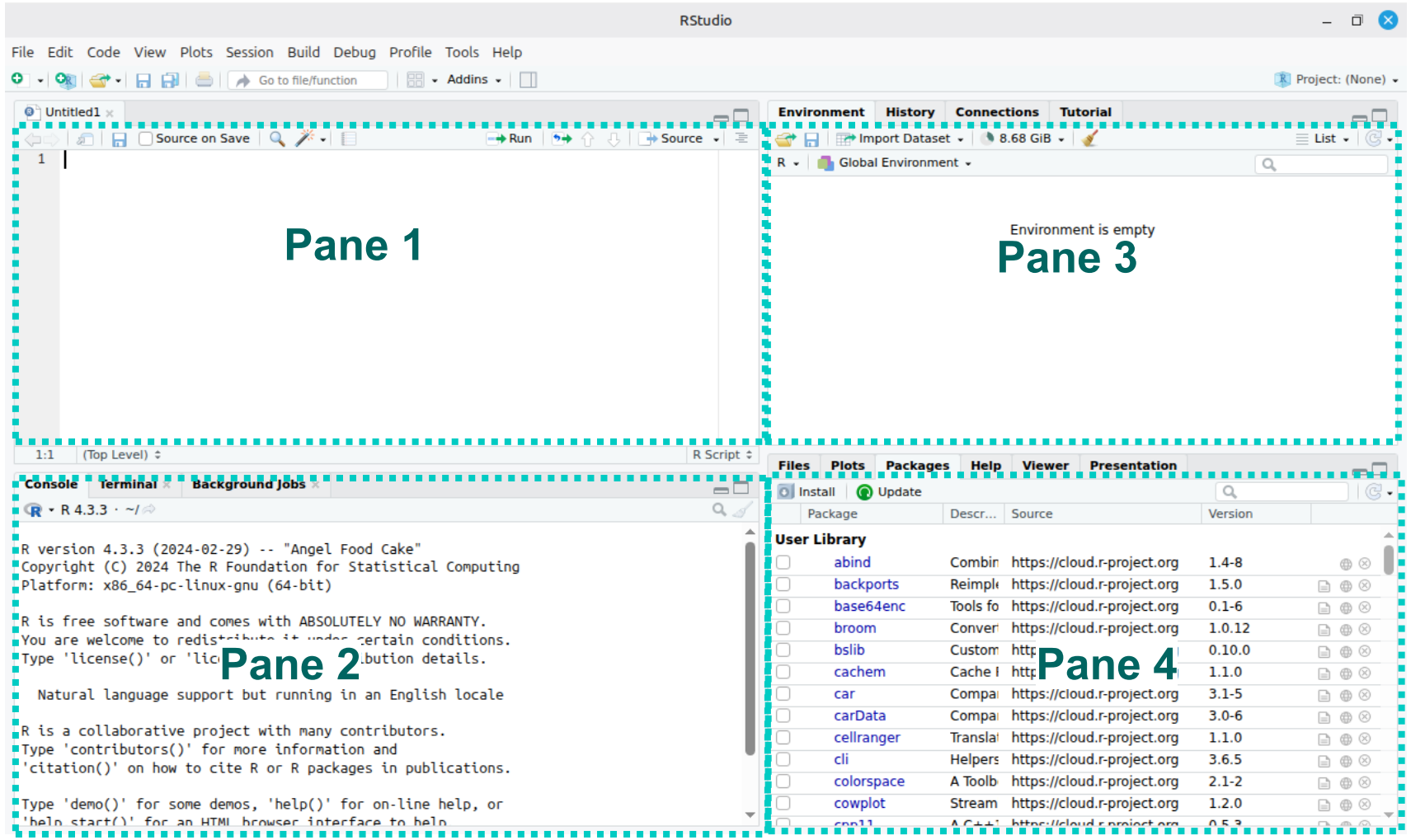
Bei Programmiersprachen gilt wie bei jeder anderen Sprache, nur durch praktischen Anwendung beherrscht man sie.

Erster Schritt nach Programmstart:

- File → New File → R Script (alternativ *Strg+Umschalt+N*).
- Die Oberfläche von RStudio teilt sich in vier Fenster (Panes) auf, diese wollen wir auf den nächsten Folien erkunden.

Allgemeines

RStudio Oberfläche



Allgemeines

RStudio Oberfläche – Pane 1 & 2

Pane 1 – Skript:

- Das „Hauptarbeitsfeld“, in dem der Code eines Programms verfasst wird.
- Um einen Teil bzw. den gesamten Code auszuführen, Bereich markieren und mit *Strg+Enter* ausführen.
- Ohne Markierung wird nur der Code in der “aktuellen Zeile” ausgeführt.

Pane 2 – Konsole:

- Hier wird der Code angezeigt, der von R verarbeitet wurde/wird.
- Auch der zu Grunde liegende Code von Bedienelementen der GUI erscheint in der Konsole.
- Kommandos können hier manuell eingeben werden, diese werden nur einmalig ausgeführt und nicht zwischen Sessions gespeichert.

Allgemeines

Oberfläche in RStudio – Pane 3 & 4

Pane 3 – Datenbereich/Workspace von RStudio:

- Zeigt (fast) alle Objekte, die derzeit verfügbar sind und verwendet werden können.
- RStudio bietet beim Schließen das Speichern des Workspace als .Rdata an. Daten werden als temporäre Datei gespeichert und beim nächsten Start automatisch geladen.
- ABER: Nicht nur Datensätze, sondern v. a. den Code (Pane 1) zur Rekonstruktion der Ergebnisse speichern (Nachvollziehbarkeit).
- Beinhaltet der Code Zufallsprozesse (z.B. `rnorm`, `runif` etc.), muss `set.seed()` am Anfang des Skripts gesetzt werden, um Reproduzierbarkeit zu gewährleisten.

Pane 4 – Zielverzeichnis:

- Plots werden hier gezeigt und können direkt exportiert werden.
- Packages können heruntergeladen, installiert und aktiviert werden.
- Erklärungen und Anwendungsbeispiele aus der Vignette der Packages.

Agenda

Basics

Kommentar

Kommentare helfen, den Überblick im Code zu behalten und dient der Nachvollziehbarkeit in der Zukunft oder für andere.

- Mit „#“ wird ein Kommentar in `R` gestartet.
- Alles, was hinter „#“ steht, wird bei Ausführung des Codes NICHT berücksichtigt.
- Im Allgemeinen sollte ein Code immer (ausführlich) kommentiert werden.
- Nach Konvention sollten Kommentare auf Englisch verfasst werden.

Basics

Variablen & Datentypen

Name der Variable muss mit einem Buchstaben beginnen (Konvention: Kleinbuchstaben).

Datentyp (= Art einer Variablen) wird durch die zugewiesenen Daten oder explizit bestimmt.

Datentyp	Beschreibung
Integer bzw. numeric (skalar)	Ganze bzw. reelle Zahlen
Factor	Kategorielle Variablen (z. B. mid, high, low)
Logical (boolean)	Wahr oder Falsch
Character (string)	Wörter (Zeichenketten)
Date	Datum
Vector	Indexiertes Objekt mit Elementen desselben Datentyps in einer Dimension
Matrix	Indexiertes Objekt mit Elementen desselben Datentyps in zwei Dimensionen
Dataframe	Tabelle, deren Spalte Vektoren gleicher Länge sind
List	Durchnummerierte Kette von Objekten verschiedener Datentypen

Basics

Datentypen – Character & Skalar

Character

```
x <- "Hello World!"        # assign a string to a variable x
```

Skalar

```
x <- 10                    # assign a scalar to a variable x
                           # wichtig:
                           # bei Neuzuordnung eines Werts zur Variable x
                           # wird der vorherige Wert überschrieben
                           # insbesondere ändert sich hier der Datentyp
```

Basics

Datentypen – Vektor

Vektor aus mehreren Elementen

```
x <- c(1, 2, 3, 4, 5)     # c ist die Kurzform für concatenate
x <- seq(1, 10, 2)        # Sequenz von 1 bis 10, in Schrittweite 2
x <- 1 : 10               # Kurzform für Sequenz der Schrittweite 1
```

Selektion von Werten in Vektoren

```
x[1]                      # Wert des Vektors x an der Stelle 1
                          # Indizes starten bei 1 in R
                          # das Element x[0] existiert nicht!
x <- x[-1]                # Vektor x wird überschrieben, das Element an
                          # Stelle 1 dabei ignoriert
x[x < 3]                  # Alle Elemente von x, deren Wert kleiner 3 ist
```

Basics

Datentypen – Matrix

Erstellen einer Matrix

- `d <- c(1 : 10)` `# Sequenz von 1 bis 10`
- `x <- matrix(data = d, nrow = 5, ncol = 2, byrow = TRUE)`
  `# Erstellt eine Matrix aus den Daten d, welche 5 Zeilen und 2 Spalten hat`
- `View(x)` `# Öffne x zur Ansicht`
- `x` `# Gib x in der Konsole aus`

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
```

Basics

Datentypen – Dataframe (I)

Liste von Vektoren gleicher Länge. Verschiedene Spalten können verschiedene Datentypen besitzen.

Erstellen eines `data.frame`-Objektes

- `x_d <- as.data.frame(x)`

```
# Erstellt einen data.frame aus der Matrix x (vorherige Folie)
```

Benennung der Spalten des `data.frame`-Objektes:

- `colnames(x_d) <- c("Spalte1", "Spalte2")`

Aufrufen von Werten eines `data.frame`-Objektes:

- `x_d[,1]` `# alle Werte der ersten Spalte`
- `x_d$Spalte1` `# alle Werte der ersten Spalte`
- `x_d$Spalte1[1 : 3]` `# Werte der ersten drei Zeilen der ersten Spalte`

Basics

Datentypen – Dataframe (II)

Nützliche Befehle für `data.frame`-Objektes

- `x_d$Spalte3 <- x_d$Spalte1 + 2 * x_d$Spalte2`
  `# Hinzufügen der Spalte3: Hier Linearkombination aus Spalte1 und Spalte2`
- `x_d$Spalte2 <- NULL` `# Löschen von Spalte2`
- `x_d[x_d$Spalte3 > 5, ]` `# Filtern der Zeilen`
- `x_d[order(x_d$Spalte3), ]` `# Sortieren der Zeilen nach Spalte3`

Basics

Bedingungen – Übersicht

Normalerweise wird Code zeilenweise ausgeführt. Es kann jedoch sinnvoll sein, Code Blöcke nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist.

Syntax der Wenn-Funktion:

- ```
  if(condition){
      # Code wird ausgeführt, falls condition wahr ist
  }
  ```

Syntax der Wenn-dann-Funktion:

- ```
  if(condition){
      # Code wird ausgeführt, falls condition wahr ist
  }else if(condition2){
      # Code wird ausgeführt, falls condition2 wahr ist
  }else{
      # Code wird ausgeführt, falls keine der oberen Bedingungen wahr ist
  }
  ```

Basics

Bedingungen – `if`-Bedingung

Falls das Klausurergebnis `examResult` besser oder gleich 4.0, gib „Prüfung bestanden“ aus:

- ```
  examResult <- 1.0
  ```
- ```
  if(examResult <= 4.0){
      print("Prüfung bestanden")
  }
  ```

Wenn die Klausur mit 1.0 bestanden ist, beglückwünsche die Kandidatin. Wenn sie bestanden wurde, teile ihr ihre Note mit und ansonsten teile der Kandidatin mit, dass sie in die Nachklausur muss:

- ```
  if(examResult == 1.0){
      print("Glückwunsch zur 1.0!")
  }else if(examResult <= 4.0){
      print(examResult)
  }else{
      print("Leider musst du in die Nachklausur.")
  }
  ```
  ```

Basics

Schleifen – Übersicht

Schleifen ermöglichen es, Code automatisch wiederholt auszuführen.

Faustregel: Wird Code mehr als zweimal untereinander kopiert, um inhaltlich dasselbe auf unterschiedliche Objekte anzuwenden, ist eine Schleife sinnvoll, um

- (Kopier-)Fehler zu minimieren und
- das Skript übersichtlicher zu gestalten.

Allerdings kann es bei Programmierfehlern es zu Endlosschleifen kommen.

Basics

Schleifen – `for`-Schleife (I)

Ziel ist es, einen Code wiederholt z. B. auf verschiedene Elemente eines Objektes (z. B. Vektor, Matrix, Dataframe) auszuführen.

Bsp.: einen Code $n \in \mathbb{N}$ Mal ausführen.

```
for (i in 1:n) {          # In diesem einfachen Beispiel sollen die Zahlen
                          # von 1 bis n in der Konsole ausgegeben werden
  print(i)

}
```

Dabei muss die Variable nicht zwangsweise eine natürliche Zahl sein:

```
for (centralMeasure in c("mean", "median", "mode")) {

  print(centralMeasure)

}
```

Basics

Schleifen – `for`-Schleife (II)

Beispiel für eine `for`-Schleife bei der die Elemente x_i und y_i der Vektoren x und y der Reihe nach addiert und einem neuen Vektor z zugewiesen werden sollen:

```
x <- c(1, 3, 5, 10, 15)# Initialisierung der Vektoren x, y und z
y <- c(2, 7, 8, 12, 17)
z <- c()
```

```
for(i in 1 : 5){

  z[i] <- x[i] + y[i]   # Addiere zu jedem Vektorelement von x das
                        # entsprechende Element von y
}
```

```
print(z)
[1] 3 10 13 22 32       # Ausgabe von z in der Konsole
```

Basics

Schleifen – `while`-Schleife

Ziel ist hier einen Code so lange auszuführen, wie eine bestimmte Bedingung erfüllt ist. Syntax der `while`-Schleife:

```
# Die Schleife wird so lange ausgeführt, wie die Bedingung wahr ist
while(Bedingung){
  # in die geschweiften Klammern schreibt man nun Code
}
```

Beispiel bei dem die Elemente x_i und y_i so lange addiert werden bis so lange $i < 5$, wobei nach jeder Iteration $i = i + 1$ gilt:

```
i <- 1
while(i < 5){
  z[i] <- x[i] + y[i]
  i <- i + 1          # die Zählvariable muss manuell verändert werden
}
```

Meist sind `while`-Schleifen für Bedingungen sinnvoller, `for`-Schleifen, wenn etwas $n \in \mathbb{N}$ mal ausgeführt oder auf bestimmte Elemente angewendet werden soll.

Basics

Funktionen – Übersicht

`R` hat viele Funktionen bereits vorimplementiert, so dass komplexere Operationen nicht manuell durchgeführt werden müssen. Packages bieten weitere Funktionen zu speziellen Themen an.

Beispielsweise muss der Mittelwert nicht manuell berechnet werden, sondern mit der internen und optimierten Funktion `mean`:

- `x <- c(1, 2, 3, 4, 5)`
-
```
mean(x)                    # Objekt x als notwendiger Inputparameter
[1] 3
```

Beinhaltet ein Vektor sog. `NA` (d. h. fehlende Werte) liefert auch `mean` als Output ein `NA`. Wir können `mean` allerdings so anpassen, dass der Mittelwert von allen nicht `NA`-Elementen berechnet wird:

- `y <- c(1, 2, 3, 4, 5, NA)`
-
```
mean(y, na.rm = TRUE)      # Objekt y als notwendiger Inputparameter
[1] 3                      # Zusätzlich Argument na.rm = TRUE um NA bei Berechnung
                           # nicht zu berücksichtigen
```

Basics

Funktionen – eigene Funktionen (I)

Zusätzlich zu den bereitgestellten Funktionen, kann man selbst Funktionen schreiben:

```
myFunction <- function(input1, input2, …, inputN){
     # Code der innerhalb der Funktion ausgeführt werden soll
     return(output)       # die Funktion gibt output zurück
}
```

Beispiel: Einfache Funktion, die namentlich begrüßt.

```
greetings <- function(name){
     print(paste0("Hallo ", name, "!"))
}

# Aufruf der Funktion mit Inputwert "Studis"
greetings("Studis")
[1] "Hallo Studis!"
```

Basics

Funktionen – eigene Funktionen (II)

Berechnung des Kapitalwerts eines Zahlungsstroms `cashFlow` (Vektor) mit Kalkulationszinssatz `discountRate` (Skalar)

```
calculatePresentValue <- function(cashFlow, discountRate){
  pv <- 0
  for(t in 1 : length(cashFlow)){
    pv <- pv + cashFlow[t] / (1 + discountRate)^t
  }
  return(pv)
}
```

Aufruf der Funktion für Zahlungsstrom `cashFlow1` und Kalkulationszinssatz `discountRate1`:

```
cashFlow1 <- c(100, 120, 160, 130, 145, 95)
discountRate1 <- 0.05
calculatePresentValue(cashFlow1, discountRate1)
[1] 633.7487
```

Agenda

Packages

Übersicht (I)

Sammlung von Funktionen, die von der R-Community für einen bestimmten Anwendungsbereich entwickelt wurden. Packages können von einem der offiziellen Mirror heruntergeladen und installiert werden:

- `install.packages("Package")`

Packages müssen vor Verwendung und nach jedem Neustart geladen werden:

- `library("Package")`

Zu einigen Packages sind sog. „CheatSheets“ verfügbar, welche die wichtigsten Funktionen und Kommandos auf einen Blick darstellen.

- Aufruf in `R` über `?` vor der Funktion, z. B. `?install.packages`

Vignetten beschreiben die implementierten Funktionen, insb.

- Input-Variablen (Format etc.) sowie den
- Output, den eine Funktion liefert.

Packages

Übersicht (II)

Zahlreiche Funktionen sind bereits in `base R` enthalten (anders als in `python`). Weitere nützliche Packages:

- `ggplot2` Erstellung von Plots
- `dplyr` Datenaufbereitung
- `e1071` Berechnung von Skewness/Kurtosis
- `readxl` Importieren und Exportieren von Excel-Dateien in/aus `R`

ACHTUNG: Unterschiedliche Packages können Funktionen mit demselben Namen haben. Dies kann zu Fehlern beim Ausführen des Skriptes führen.

- Beispiel: `readxl` und `openxlsx` nutzen Funktionen mit gleichem Namen.

Agenda

Datenimport & Datenexport

Datenimport

Import einer .csv Datei:

- `data <- read.csv("data.csv", header = TRUE, sep = ",")`

Import einer .xlsx Datei:

- `library("readxl")  # Laden des Packages readxl`
- `data <- read_excel("C:/Users/user/Documents/data.xlsx")`

Ansicht der eingelesenen Daten:

- `View(data)`

Anstatt den Pfad im Befehl `read_excel` direkt anzugeben, kann er auch vorab gesetzt werden:

- `getwd()  # Zeige des aktuelle Arbeitsverzeichnis an`
- `setwd("C:/Users/user/Documents/")`
- `data <- read_excel("data.xlsx")`

Datenimport & Datenexport

Datenexport

Speichern von Objekten als Excel Files:

- `write.xlsx(data, file = "data_excel.xlsx", sheetName = "HelloWorld", col.names = TRUE, row.names = FALSE, showNA = TRUE)`

Achtung: R überschreibt vorhandene (Excel-)Dateien, ohne vorherige Warnung.

Export als .csv Datei ebenfalls möglich:

- `write.csv(data, file = "data_Excel.xlsx", row.names = FALSE)`

Plots können direkt mit R exportiert werden:

- ```
  png(filename = „plot_data.png")
  plot()
  dev.off()
  ```

Alle Dateien werden, falls nicht anders spezifiziert, im Arbeitsverzeichnis (*working directory*) abgelegt.
  ```

Agenda

Deskriptive Statistik

Überblick über die Daten

Nützliche Befehle:

```
str(data)          # Struktur des Datensets
names(data)        # (Spalten-)Namen
head(data)         # Ausgabe der obersten Zeilen
data[10 : 20, ]    # Ausgabe der Zeilen 10 bis 20
nrow(data)         # Zeilenanzahl
ncol(data)         # Spaltenanzahl
dim(data)          # Dimensionen (Zeilenanzahl und Spaltenanzahl)
```

Deskriptive Statistik

Berechnung von Kenngrößen

Lagemaße, Streuungsmaße und Korrelation

- `mean(data)` # Berechnung des Erwartungswerts $\left(\bar{x} = \frac{1}{n}\sum_{i=i}^{n} x_i\right)$
- `median(data)` # Berechnung des Medians
- `var(data)` # Berechnung der Varianz $\left(\sigma^2 = \frac{1}{n-1}\sum_{i=i}^{n}(x_i - \bar{x})^2\right)$
- `sd(data)` # Berechnung der Standardabweichung
- `quantile(data, q)` # Berechnung des q-ten Quantils
- `summary(data)` # Summary des Datensatzes
- `cov(data, data2)` # Berechnung der (Stichproben-) Kovarianz zwischen
  # den Datensätzen data, data2 $\left(\mathrm{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=i}^{n}(x_i - \bar{x})(y_i - \bar{y})\right)$
- `cor(data, data2)` # Berechnung der Korrelation $\left(\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_x \cdot \sigma_y}\right)$

Deskriptive Statistik

Eigene Funktion – Bravais-Pearson-Korrelationskoeffizient

Schreibe eine eigene Funktion zur Berechnung des Bravais-Pearson-Korrelationskoeffizienten, gegeben durch

- $\mathrm{Corr}(\mathrm{X},\mathrm{Y}) = \frac{\mathrm{Cov}(\mathrm{X},\mathrm{Y})}{\sigma_x \cdot \sigma_y} = \frac{\sum_{i=i}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=i}^{n}(x_i-\bar{x})^2 \sum_{i=i}^{n}(y_i-\bar{y})^2}}.$

Nutze für die Programmierung nur die Funktionen

- `sum()` und
- `mean()`.

Der Input sollen dabei zwei eindimensionale Vektoren sein und der Output das Skalar $-1 \leq r_{XY} \leq 1$.

Lösung findet ihr im R-Skript in der Zeile 149 ff.

Agenda

Plots

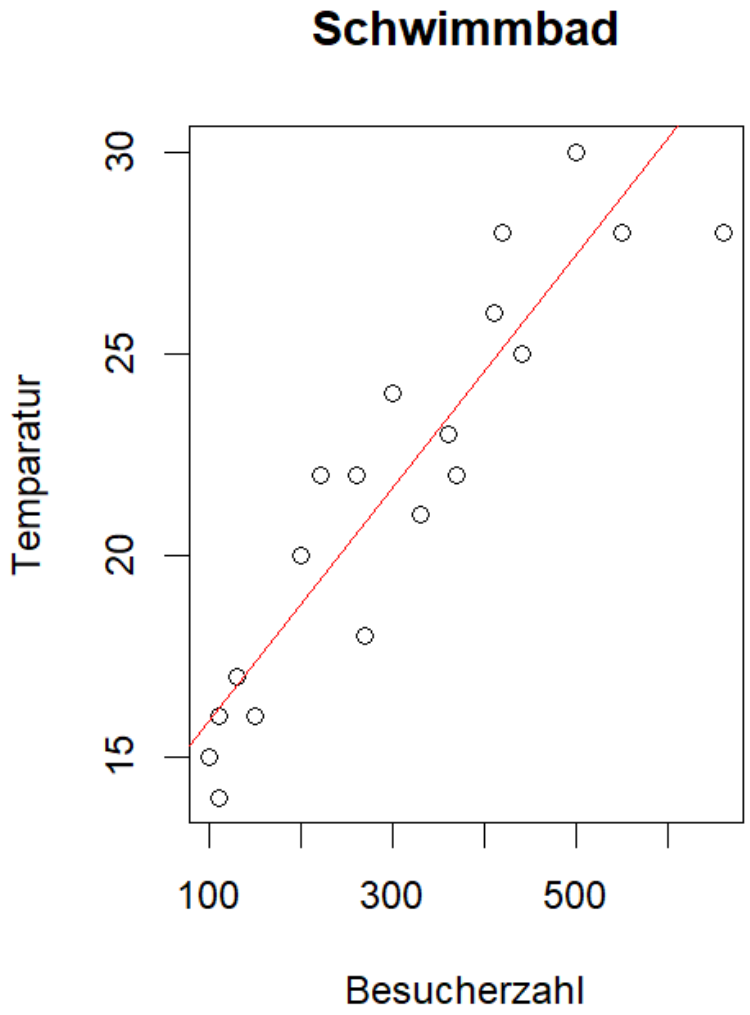
Scatterplot

Plotten zweier Variablen gegeneinander

```
plot(data$Besucheranzahl,
     data$Temperatur,
     col  = "black",
     main = "Schwimmbad",
     ylab = "Temparatur",
     xlab = "Besucherzahl")
# Scatterplot der Variablen Besucherzahl
# und Temperatur des Dataframes data
```

Hinzufügen einer Regressionsgeraden, dazu später mehr.

```
abline(lm(data$Temperatur ~ data$Besucheranzahl),
       col = "red")
# Hinzufügen einer Regressionsgeraden
# Benötigt ein lineares Modell als Input
```



Plots

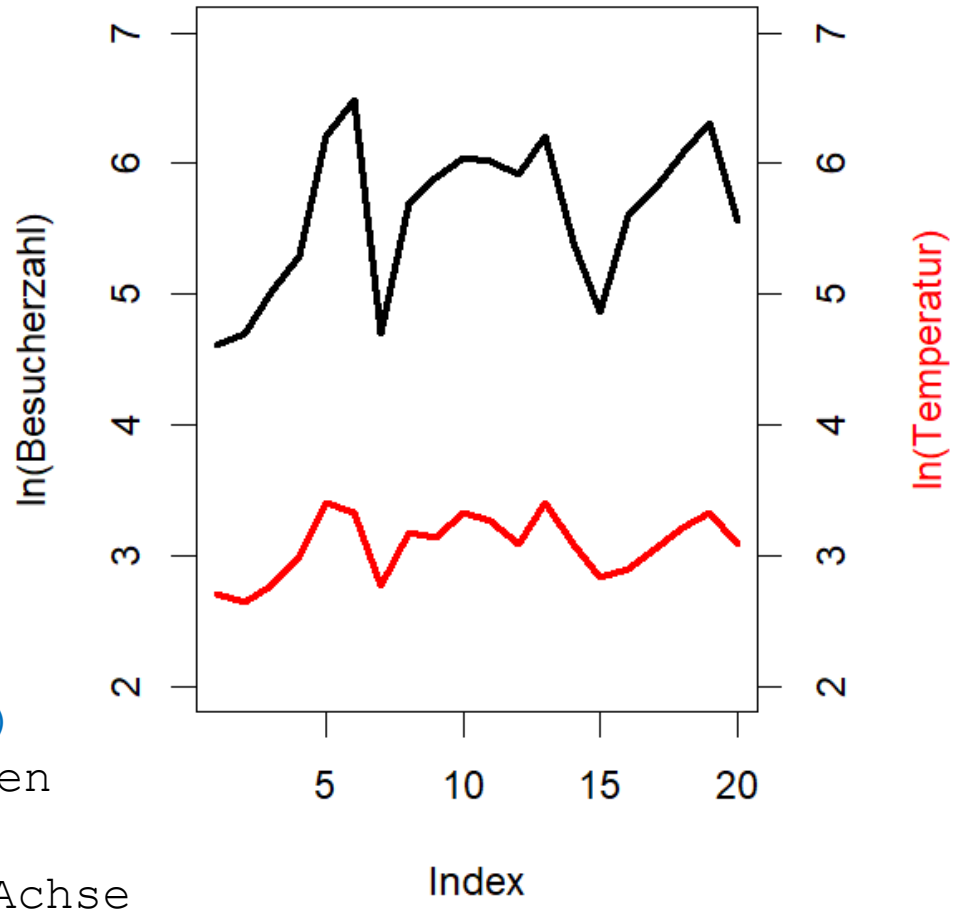
Linienplot

Linienplot mit zwei Variablen:

```
plot(log(data$Besucheranzahl),
     type = "l",
     ylim = c(2, 7),
     lwd = 3,
     ylab = "ln(Besucherzahl)")
# Linienplot der logarithmierten Variable
# Besucheranzahl und manuelle Bestimmung
# der Grenzen der y-Achse
```

Hinzufügen eines weiteren Liniendiagramms. Achtung: `lines` fügt nur Linien hinzu und kann nicht alleine stehen.

```
lines(log(data$Temperatur), lwd = 3, col = "red")
axis(4)     # Achse auf der linken Seite hinzufügen
mtext("ln(Temperatur)", side = 4,
line = 3, col = "red") # Beschriftung der neuen Achse
```



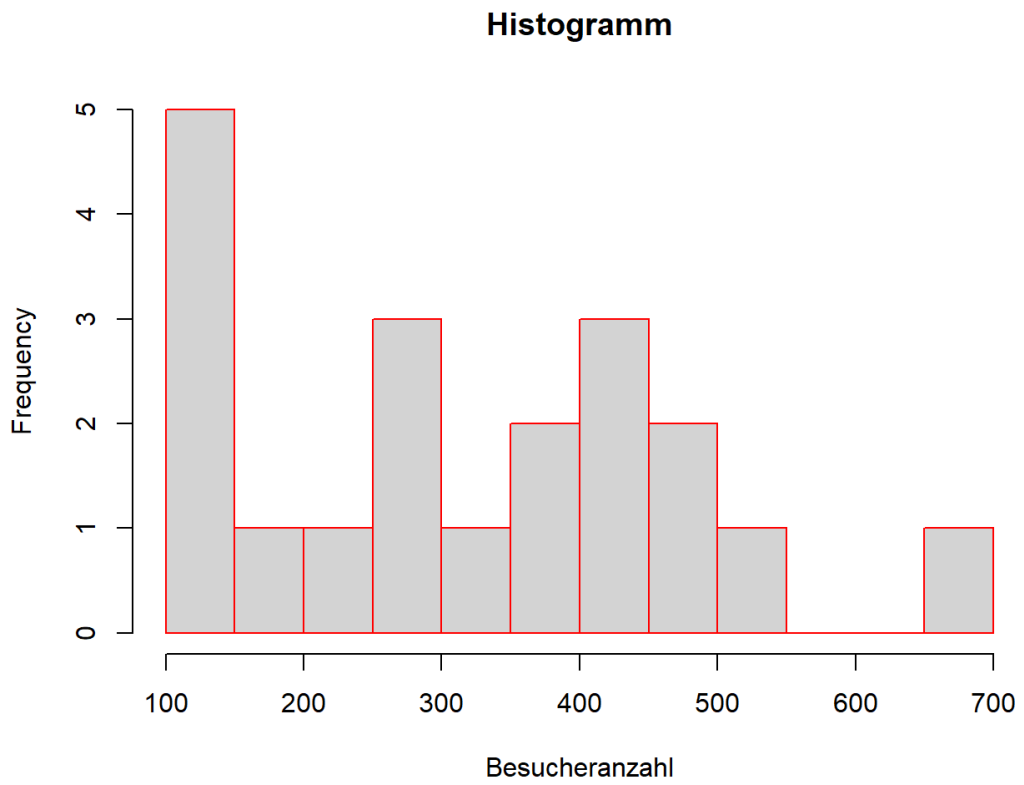
Plots

Histogramm

Darstellung der Häufigkeitsverteilung:

```
hist(data$Besucheranzahl,
     breaks = 12,
     plot = TRUE,
     xlab = "Besucheranzahl",
     main = "Histogramm",
     freq = TRUE,
     col  = "lightgrey",
     border = "red")

# Histogramm für die Variable size
# mit freq = FALSE bekommt man die
# Wahrscheinlichkeitsverteilung
# anstelle der Häufigkeiten
```



Agenda

Regressionsanalyse

Übersicht

Ziel ist die Modellierung von Zusammenhängen zwischen abhängigen und unabhängigen Variablen. Damit versucht man folgende Fragen zu beantworten:

- Gibt es eine Beziehung zwischen den Variablen? Und wenn ja, wie stark ist sie?
- Wie wirkt sich eine Veränderung der unabhängigen Variable auf die abhängige Variable aus?

Wie der Name sagt, können mit der linearen Regression nur lineare Zusammenhänge zwischen den Daten modelliert werden.

Will man z. B. einen Zusammenhang zwischen der Tagesdurchschnittstemperatur und der Besucherzahl im Schwimmbad untersuchen, ist eine mögliche Frage:

- Wie verändert sich die Zahl der Besucher/innen bei einem Anstieg bzw. Rückgang der Temperatur?

Regressionsanalyse

Univariate lineare Regression (I)

Abhängige / zu beschreibende Variable Y mit $n \in \mathbb{N}$ Beobachtungen $y_1, \dots, y_n$.

Unabhängige / beschreibende Variable X mit $n \in \mathbb{N}$ Beobachtungen $x_1, \dots, x_n$.

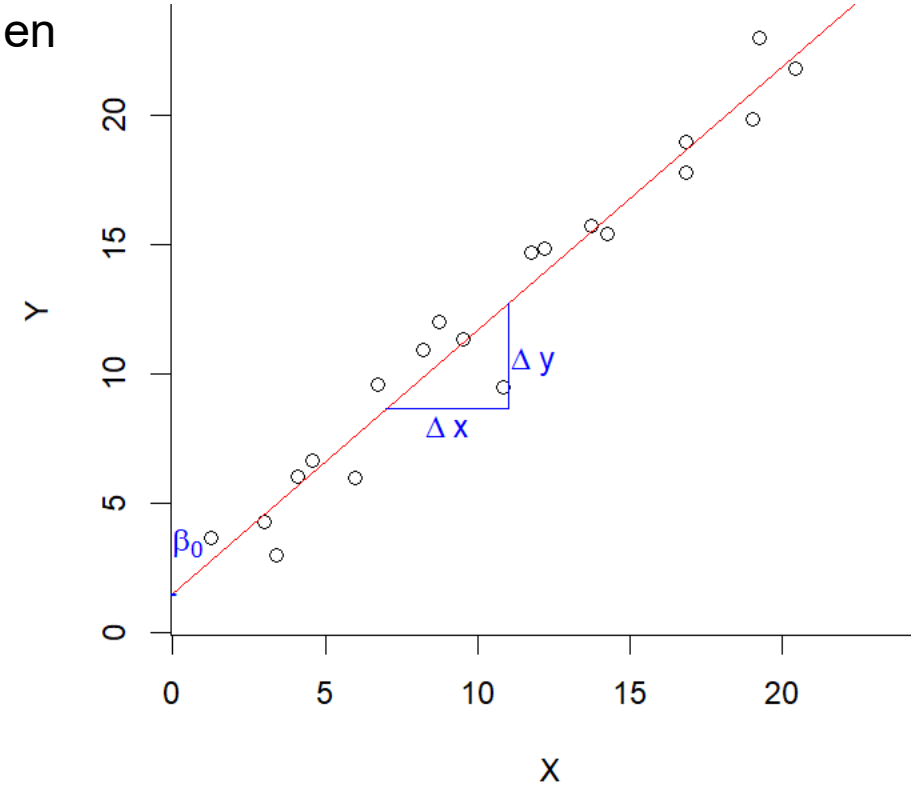
Zusammenfassung der $n \in \mathbb{N}$ Beobachtungen der beiden Variablen in Tupeln: $(x_1, y_1), \dots, (x_n, y_n)$.

Bei der linearen Regression wird ein linearer Zusammenhang zwischen der abhängigen und unabhängigen Variable angenommen:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

mit

- y-Achsenabschnitt β_0
- Steigung β_1 (Richtung des Zusammenhangs zwischen der unabhängigen Variable und der abhängigen Variable)
- Fehlerterm ε



Regressionsanalyse

Univariate lineare Regression (II)

Lineare Regression:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Im Allgemeinen ist der wahre Zusammenhang zwischen den Variablen X und Y und damit die Parameter β_0 und β_1 unbekannt. Daher muss der Zusammenhang mit Hilfe von Daten geschätzt werden.

Wie müssen die Parameter β_0 und β_1 aussehen, sodass die Gerade am besten zu den beobachteten Werten von X und Y (s. Plot) passt und damit den beobachteten Zusammenhang am besten abbildet?

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

→ Wahl der Parameter so, dass das Residuum $\hat{\varepsilon}$ (Abweichung) minimiert wird!

Unter bestimmten Voraussetzungen (s. *Satz von Gauss-Markov*) gibt der „kleinste Quadrate Schätzer“ die besten Schätzwerte für die Parameter. Es wird dabei die Summe der quadratischen Abweichungen der Datenpunkte von der Regressionsgeraden minimiert:

$$\min \sum_i^n \hat{\varepsilon}_i^2 = \min \sum_i^n \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i\right)\right)^2 \Rightarrow \hat{\beta}_0, \hat{\beta}_1$$

Hinweis: `R` berechnet für uns den kleinsten Quadrate Schätzer über die Funktion `lm`

Regressionsanalyse

Multivariate lineare Regression

Bisher haben wir den Zusammenhang einer unabhängigen Variable auf die abhängige Variable betrachtet. Allerdings können mehrere unabhängige Variablen $X_1, \dots, X_K$ die abhängige Variable beeinflussen.

- Erweiterung der univariaten linearen Regression zur multivariaten linearen Regression:
- $Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_K X_K + \varepsilon$
- Der Parameter $\beta_k,\ k = 1, \dots, K$ beschreibt hierbei den Einfluss, den die unabhängige Variable X_k auf die abhängige Variable Y hat.
- Die Schätzung folgt auch hier mit der Methode der kleinsten Quadrate Schätzung (Minimierung der quadratischen Abweichung) und wir erhalten die „besten“ Parameterschätzer $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_K$.

Beispiel (fortgesetzt): Neben der Temperatur ist den Besucher/innen des Schwimmbads auch die Wassertemperatur wichtig. Beeinflusst die Wassertemperatur die Besucheranzahl?

Regressionsanalyse

Güte der Regression – Bestimmtheitsmaß

Das klassische Bestimmtheitsmaß, das für lineare Regressionen genutzt wird, ist R^2.

- Kennzahl zur Beurteilung der Anpassungsgüte einer Regression: „Der Anteil, der durch die Regression erklärten Quadratsumme, an der zu erklärenden totalen Quadratsumme.“
- Es gibt also Antwort auf die Frage, wie viel Streuung der abhängigen Variable durch ein vorliegendes lineares Regressionsmodell „erklärt“ werden kann?

Formell ist das Bestimmtheitsmaß R^2 gegeben durch:

- $R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} = \frac{Summe\ der\ Quadrate\ der\ erklärten\ Abweichung}{totale\ Quadratsumme} \in [0,1]$
- Wobei $R^2 = 1$ einen perfekten Zusammenhang signalisiert und $R^2 = 0$ ein Modell ohne Erklärungsgehalt.

Im Allgemeinen gilt, je höher der Wert des Bestimmtheitsmaßes R^2 ist, desto besser ist das Modell. Allerdings steigt mit der Zahl der Regressoren R^2 automatisch. Um Modellkomplexität zu kontrollieren:

- Adjusted R^2 ist ein erweitertes Maß, welches die Anzahl K der Regressoren mitberücksichtigt.

Regressionsanalyse

Güte der Regression – Test der Parameter

Die Koeffizienten/Parameter $\beta_0, \beta_1, \dots, \beta_k$ werden auf ihre Signifikanz getestet.

- Ist der geschätzte Parameter $\hat{\beta}_k$ signifikant verschieden von 0?
- Falls ja, dann weist das auf einen signifikanten, linearen Einfluss der Variable X auf Y hin.

Mit dem t-Test wird das Hypothesenpaar getestet:

- $H_0: \beta_k = 0$ vs. $H_1: \beta_k \neq 0$

Regressionsanalyse

Regression in R (I)

R gibt bei der Schätzung der linearen Regressionsgleichung viele wichtige Statistiken zur Güte des Modells aus. Man wendet hierzu summary auf das (gespeicherte) Modell an.

Beispiel im univariaten Fall:

- ```
  data_lm <- read_excel("C:/Users/user/Documents/data.xlsx")
  # Berechnung einer linearen Regression mit Besucheranzahl als abhängige,
  # Temperatur als unabhängige Variable. die Daten sind in der Tabelle data_lm
  ```
- ```
  lin_model <- lm(Besucheranzahl ~ Temperatur, data = data_lm)
  ```
- ```
  # Ausgabe der Parameter, Teststatistiken, Bestimmtheitsmaße etc.
  summary(lin_model)
  ```
  ```

Regressionsanalyse

Regression in R (II)

Beispiel im univariaten Fall:

- Geschätzter Beta-Koeffizient für Temperatur von 29.254
- Dieser Koeffizient ist signifikant auf dem 0.001 Niveau mit einer Teststatistik von 9.879 und ein p-Wert von 1.08e-08.
- Eine Erhöhung der Temperatur um 1 Grad führt durchschnittlich zu 29.254 mehr Besucher/-innen. Die Temperatur beeinflusst die Besucheranzahl signifikant.
- Der R^2 Wert von 0.8443 ist hoch. Temperatur erklärt die Besucherzahl sehr gut.

```
> summary(lin_model)

Call:
lm(formula = "Besucheranzahl ~ Temperatur", data = data_lm)

Residuals:
   Min     1Q Median     3Q    Max
-92.19 -47.71 -13.31  41.81 172.29

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -331.394     67.514  -4.908 0.000113 ***
Temperatur    29.254      2.961   9.879 1.08e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 65.93 on 18 degrees of freedom
Multiple R-squared:  0.8443,    Adjusted R-squared:  0.8356
F-statistic:  97.6 on 1 and 18 DF,  p-value: 1.076e-08
```

Regressionsanalyse

Regression in R (III)

Beispiel für eine multivariate Regression, bei der die Besucherzahl durch die Außen- und Wassertemperatur erklärt werden soll :

- ```
  # Laden des Datensatzes
  data_lm <- read_excel("C:/Users/papenfpa/Documents/data.xlsx")
  ```
- ```
  # Berechnung einer linearen Regression mit y als abhängige, alle anderen
  # Spalten des Dataframes data_lm als unabhängige Variablen
  lin_model <- lm("Besucheranzahl ~ . ", data = data_lm)
  # Durch den Punkt in der Regressionsgleichung werden alle Spalten der
  # Tabelle Regressoren - abgesehen von der abhängigen Variable
  ```
- ```
  # Ausgabe der Parameter, Teststatistiken, Bestimmtheitsmaße etc.
  summary(lin_model)
  ```
  ```

Regressionsanalyse

Regression in R (IV)

Beispiel im multivariaten Fall:

- Geschätzter Beta-Koeffizient für Temperatur von 34.189. Dieser Koeffizient ist signifikant auf dem 0.001 Niveau.
- Eine Erhöhung der Temperatur um 1 Grad führt durchschnittlich zu 34.189 mehr BesucherInnen.
- Die Temperatur erklärt die Bewegung der Besucheranzahlen gut.

Geschätzter Beta-Koeffizient für Wassertemperatur von -69.162. Dieser Koeffizient ist nicht signifikant und hat einen p-Wert von 0.242.

- Die Wassertemperatur korreliert mit der Außentemperatur.

```
> summary(lin_model)

Call:
lm(formula = "Besucheranzahl ~ .", data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-83.426 -54.509   2.731  34.564 165.700

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     1100.076   1182.830   0.930    0.365
Temperatur        34.189      5.012   6.821 2.98e-06 ***
Wassertemperatur  -69.162     57.058  -1.212    0.242
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 65.09 on 17 degrees of freedom
Multiple R-squared:  0.8567,    Adjusted R-squared:  0.8398
F-statistic: 50.81 on 2 and 17 DF,  p-value: 6.741e-08
```

Regressionsanalyse

Regression in R (V)

Beispiel im multivariaten Fall: Der R^2 Wert von 0.8567 ist hoch:

- Modell erklärt also rund 86 % der Varianz
- Aber adjusted R^2 ist kaum größer als im univariaten Modell

Außerdem ist der Koeffizient der Wassertemperatur negativ, d. h. höhere Temperaturen des Wassers bedeuten weniger Schwimmbadbesucher/innen → eher kontraintuitiv.

Mögliche Erweiterungen wären:

- Interaktionen zwischen Variablen
- Polynome
- Transformationen

Agenda

Weiterführende Literatur

Nützliche Websites & Tutorials

Websites

- https://www.statmethods.net/r-tutorial/index.html
- https://stackoverflow.com/
- https://cran.r-project.org/
- https://stats.stackexchange.com/questions

Tutorials

- Multiple Linear Regression in R | R Tutorial 5.3 | MarinStatsLectures – YouTube
- R - Multiple Regression (part 2) – YouTube
- https://bookdown.org/joachimfreyberger/r_tutorial_phd_econometrics/_book/index.html

Weiterführende Literatur

Zusätzliche Literatur

Lehrbücher

- Grundlagen der Datenanalyse mit R - Eine anwendungsorientierte Einführung, Daniel Wollschläger, Berlin, Springer Spektrum, 2017
- Angewandte Zeitreihenanalyse mit R, Rainer Schlittgen, München, Oldenbourg, 2012
- Einführung in die Statistik mit R, Andreas Behr, München, Vahlen, 2011

Reference Cards

- https://cran.r-project.org/doc/contrib/Short-refcard.pdf

Online-Skripte

- www.wiwi.uni-bielefeld.de/lehrbereiche/emeriti/jfrohn/Upload/statskript.pdf
- www.uni-muenster.de/Stochastik/lehre/SS17/PrakStat/Materialien/Skript.pdf

Vielen Dank für die Aufmerksamkeit!

Wirtschaftsingenieurwesen
Institut für Materials Resource Management
Universität Augsburg
wing@mrm.uni-augsburg.de
www.uni-augsburg.de/wing