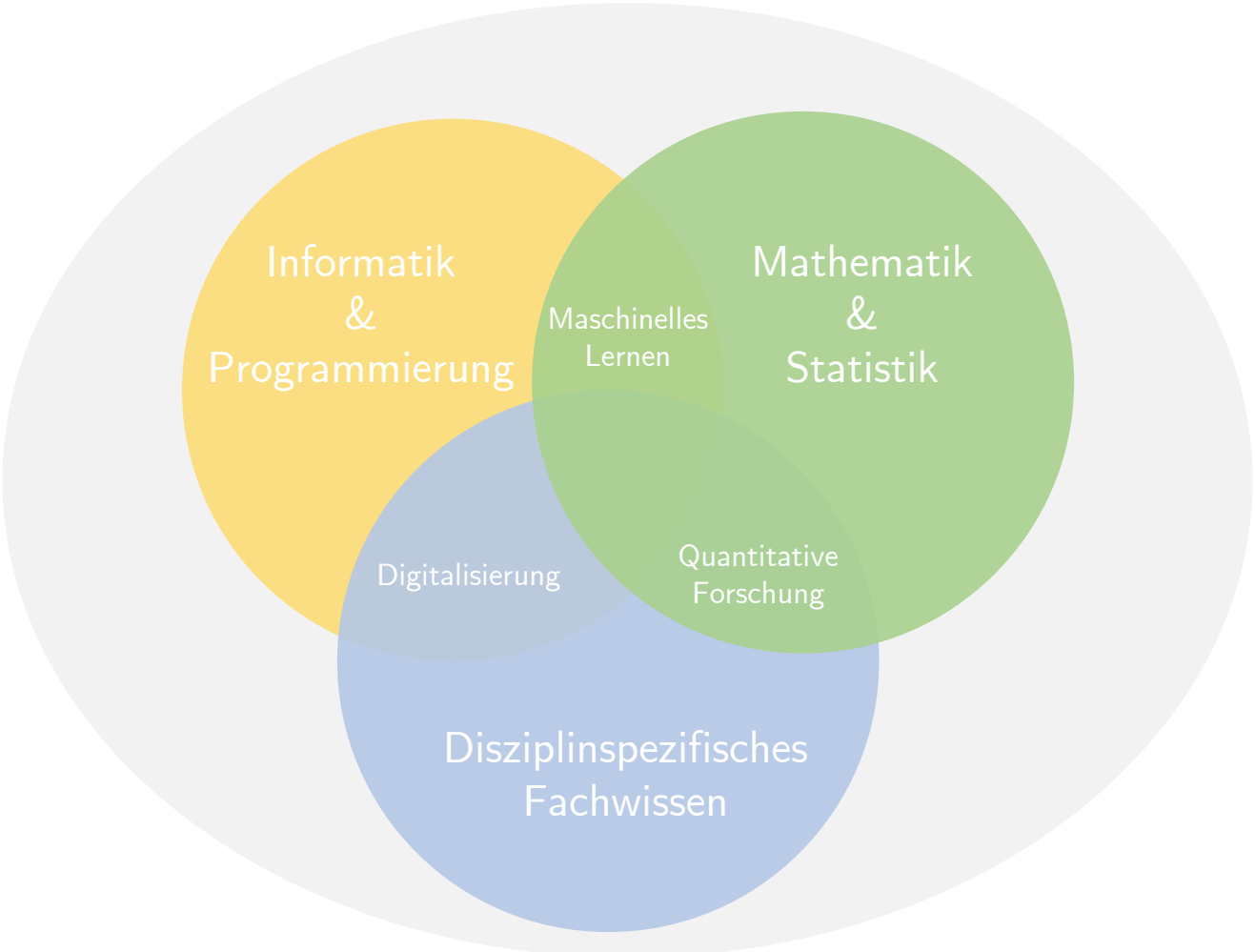
Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(1) Datenanalyse und Programmierung

Datenwissenschaft



Datenanalyse und Programmierung

- Computergestützte Datenanalyse
- Informatik und Rechnerarchitektur
- Algorithmen und Programme
- Selbstkontrollfragen

Datenanalyse und Programmierung

- **Computergestützte Datenanalyse**
- Informatik und Rechnerarchitektur
- Algorithmen und Programme
- Selbstkontrollfragen

Computergestützte Datenanalyse

- Wissenschaftliche Daten liegen typischerweise in digitaler Form vor.
- Daten werden typischerweise mit Hilfe eines Computers analysiert.
- Zur Analyse von digitalen schreibt man Computerprogramme.
- Diese Computerprogramme heißen *Datenanalyseskripte*.

Struktur computergestützter Datenanalyse

1. Einlesen und Bereinigen eines Datensatzes
2. Berechnung und Visualisierung deskriptiver Statistiken
3. Probabilistische Datenmodellierung, Modellinferenz, Visualisierung
4. Dokumentation und Präsentation der Ergebnisse

Typische Werkzeuge zur Analyse psychologischer Daten

Heute

- R (frei verfügbar, Datenwissenschaft, Psychologie)
- Python (frei verfügbar, Datenwissenschaft)
- Matlab (kommerziell, Ingenieurswissenschaften, Neuroimaging)

Früher

- SPSS (kommerziell, Sozialwissenschaften, Psychologie)
- JMP (kommerziell, Biologie, Psychologie)

PYPL Index März 2021

Worldwide, Mar 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.17 %	-0.2 %
2		Java	17.18 %	-1.2 %
3		JavaScript	8.21 %	+0.2 %
4		C#	6.76 %	-0.6 %
5	↑	C/C++	6.71 %	+0.8 %
6	↓	PHP	6.13 %	+0.0 %
7		R	3.81 %	+0.0 %
8		Objective-C	3.56 %	+1.1 %
9		Swift	1.82 %	-0.4 %
10	↑	Matlab	1.8 %	-0.0 %

- PopularitY of Programming Language
- Googlesuchanfragen zu Programmiersprachentutorials

Datenanalyseskripte

- Dokumentation aller Schritte vom Rohdatum zur Datenvisualisierung.
- Reproduktion wissenschaftlicher Ergebnisse durch Dritte.
- Essentieller Teil wissenschaftlicher Publikationen.
- Wesentlicher Teil wissenschaftlicher Arbeit im 21. Jahrhundert.

Programmierung ist das zentrale Handwerkzeug wissenschaftlicher Arbeit

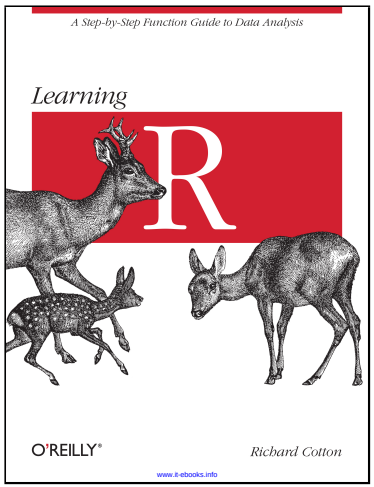
Kursziel ist das Sammeln erster datenanalytischer Programmiererfahrung

Modul C2 Computergestützte Datenanalyse

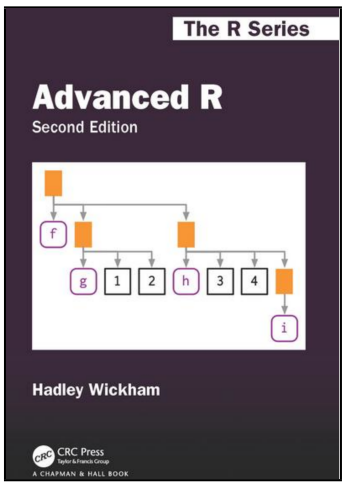
Einheit	Datum	Thema
(1) Datenanalyse und Programmierung	08.04.2021	Grundkenntnisse
(2) R und RStudio Grundlagen	15.04.2021	Programmierung
(3) Vektoren, Matrizen, Arrays	22.04.2021	Programmierung
(4) Listen und Dataframes	29.04.2021	Programmierung
(5) Environments und Funktionen	06.05.2021	Programmierung
(6) Kontrollstruktur, Schleifen, Apply	13.05.2021	Programmierung
(7) Datenimport und Datenbereinigung	20.05.2021	Datenanalyse
(8) Deskriptive Statistiken	27.05.2021	Datenanalyse
(9) Visualisierung I	03.06.2021	Visualisierung
(10) Visualisierung II	10.06.2021	Visualisierung
(11) T-Tests	17.06.2021	Datenanalyse
(12) Optimale Stichprobenumfänge	24.06.2021	Datenanalyse
(13) Einfaktorielle Varianzanalyse	01.06.2021	Datenanalyse
(14) Zweifaktorielle Varianzanalyse	08.07.2021	Datenanalyse
Klausurtermin	Juli	
Klausurwiederholungstermin	September	

Modul C2 Computergestützte Datenanalyse

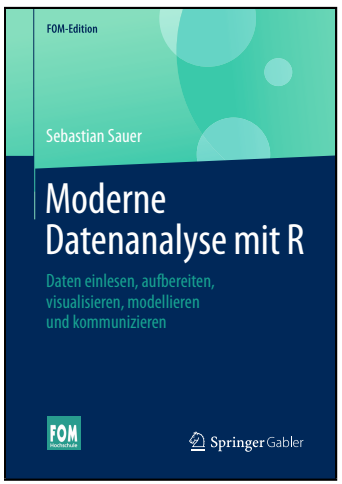
Weiterführende Literatur



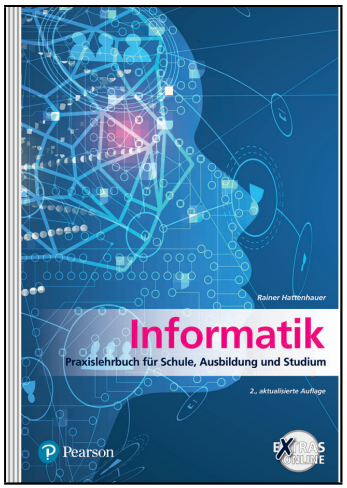
Cotton (2013)



Wickham (2019)



Sauer (2019)



Hattenhauer (2020)

Modul C2 Computergestützte Datenanalyse

Ziele des Moduls

- Grundlegende Datenanalysekompetenz
- Grundlagen für den Erwerb fortgeschrittener Datenanalysekompetenz
- Kompetenz im Umgang mit fachspezifischer statistischer Anwendungssoftware
- Anwendung von in Modul B2 Inferenzstatistik vorgestellten Methoden

Leistungsnachweis

- Klausur am Ende des Semesters
- Prüfungsvorleistung: Erfolgreiche Teilnahme Teilmodul 1
- Die Modulnote entspricht der Klausurnote des Teilmoduls 2
- Klausurfragen angelehnt an Selbstkontrollfragen

Datenanalyse und Programmierung

- Computergestützte Datenanalyse
- **Informatik und Rechnerarchitektur**
- Algorithmen und Programme
- Programmiersprachen
- Selbstkontrollfragen

Informatik (engl. Computer Science)

Bei der Informatik handelt es sich um die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, wobei besonders die automatische Verarbeitung mit Computern betrachtet wird. Sie ist zugleich Grundlagen- und Formalwissenschaft als auch Ingenieurdisziplin.

Wikipedia

Zentrale Komponenten der Informatik

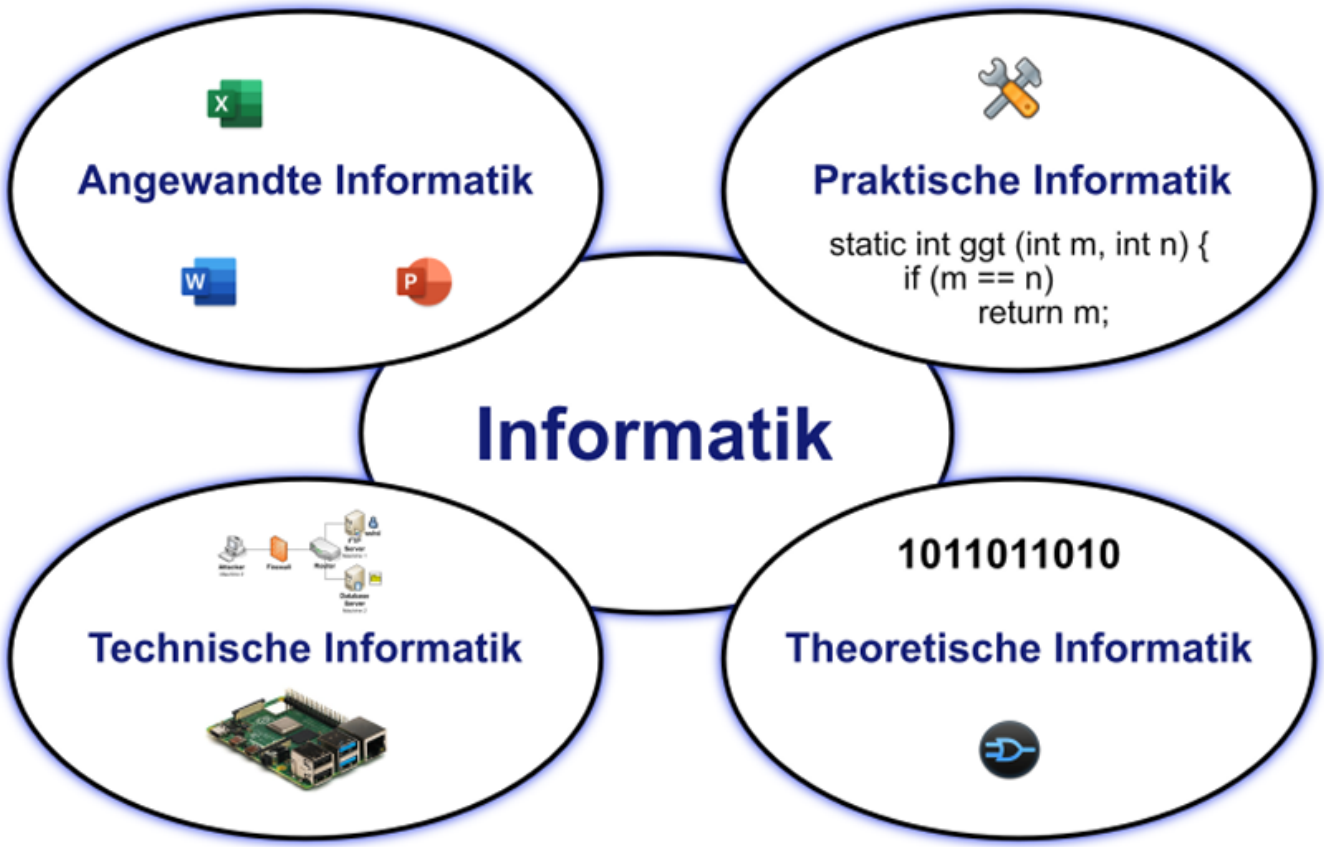
Computer

- Maschinen zum Datenspeichern und Ausführen einfacher Datenoperationen.
- Einfache Operationen mit extrem hoher Geschwindigkeit.
- Universalität durch Speicherung von Daten und Programmen.

Algorithmen und Programme

- *Programme* sind in einer *Programmiersprache* verfasste *Algorithmen*.
- Algorithmen sind Folgen von Anweisungen durchzuführender Operationen.
- Bei Algorithmen unterscheidet man
  - Beschreibung (Kochrezept, IKEA Bauanleitung, R Skript)
  - Anweisungen ("Mehl und Wasser vermengen", o - - -, x = c(1,2,3))
  - Durchführung (Kochvorgang, Zusammenbau, R Skript laufen lassen)

Teilgebiete der Informatik



Hattenhauer (2020) Informatik

Teilgebiete der Informatik (mit Relevanz für Psychologie)

Angewandte Informatik

Anwendungssoftware, Human-Computer-Interaction, Informatik und Gesellschaft

Technische Informatik

Mikroprozessortechnik, Rechnerarchitektur, Netzwerktechnik

Praktische Informatik

Programmierung, Algorithmen, Datenbanken

Theoretische Informatik

Automatentheorie, Berechenbarkeitstheorie, Komplexitätstheorie

Spezialgebiete der Informatik (mit Relevanz für Psychologie)

Maschinelles Lernen und Künstliche Intelligenz

Datenanalyse aus Sicht der Informatik

Computervisualistik

Bilderkennung und Bildsynthese, Virtuelle Realität, Augmented Reality

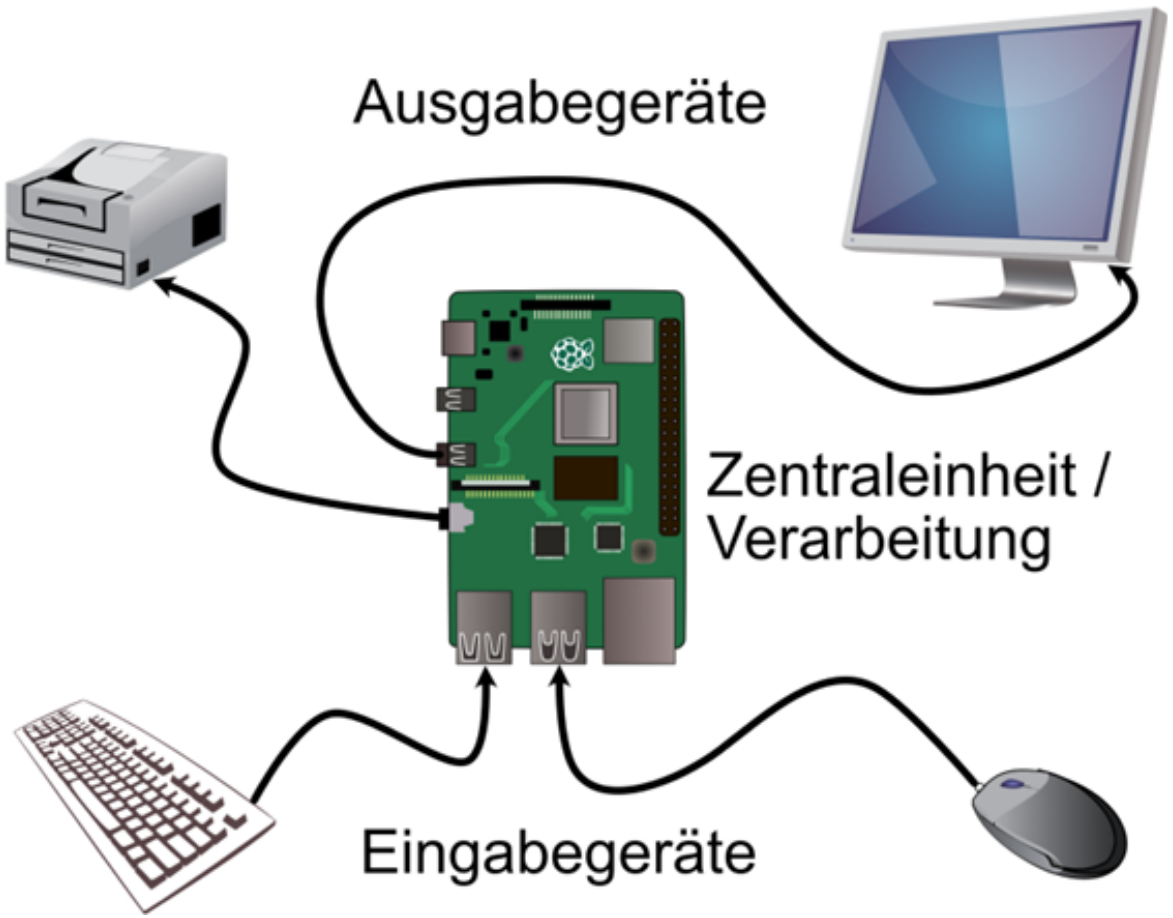
Computerlinguistik

Spracherkennung und Sprachsynthese

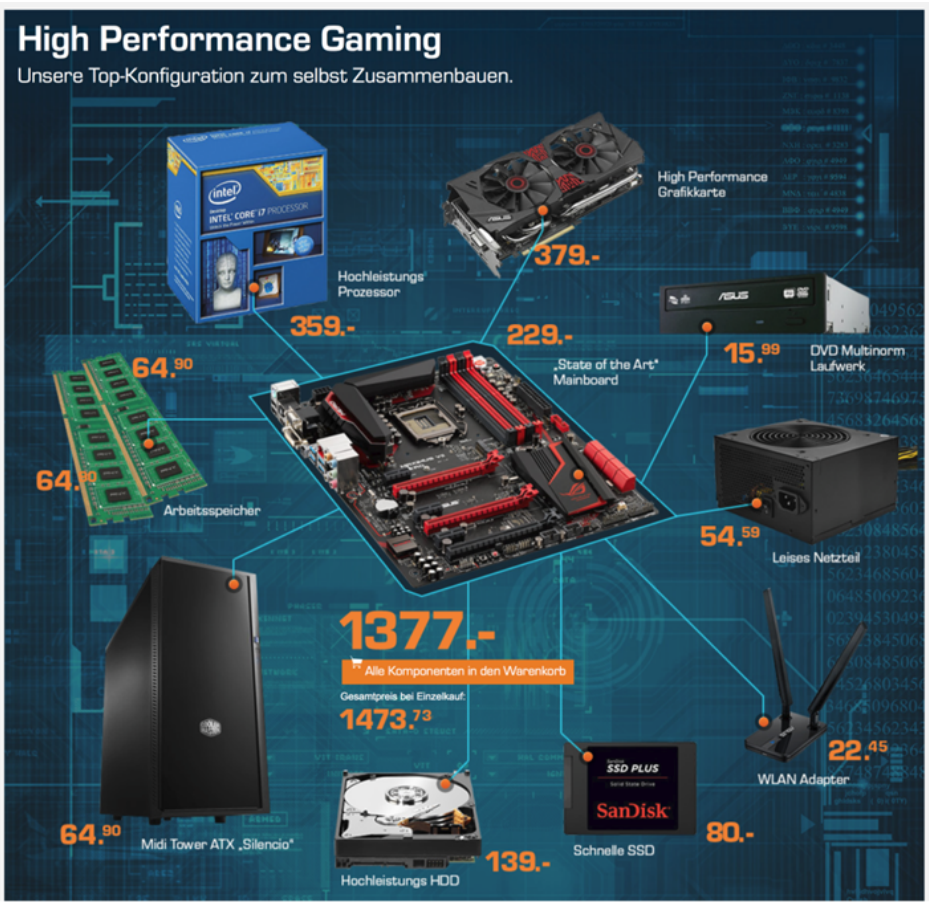
Bioinformatik

Lebenswissenschaften, Genomik, Bildgebende Verfahren der Medizin

Hardwarekomponenten eines Computers



Zentraleinheit eines Computers



Zentraleinheit (Hauptplatine, Motherboard, Mainboard)

CPU (Central Processing Unit/Mikroprozessor)

- Rechenwerk, Steuerwerk, und Leitwerk des Systems
- Cache (flüchtiger schneller Speicher)
- Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz

RAM (Random Access Memory)

- Temporärer, flüchtiger Arbeitsspeicher des Systems
- Begrenzt, z.B. 16 GB

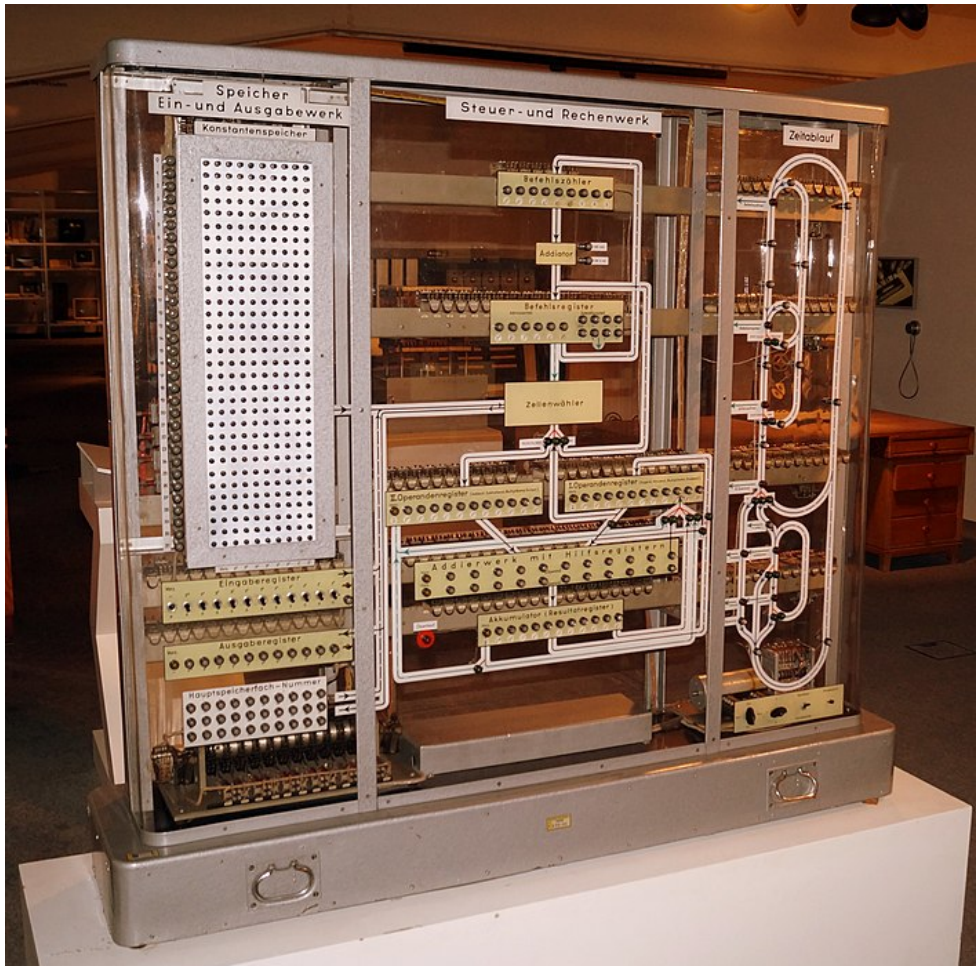
Massenspeicher

- Stationärer Speicher des Systems
- SSD (Solid State Drive), Cloudspeicher

GPU (Graphical Processing Unit)

- Leistungsstarke, speziell für Visualisierung optimierte Prozessoren
- Unterstützung der CPU in manchen Anwendungen, z.b. Neuronale Netze

Von-Neumann-Architektur

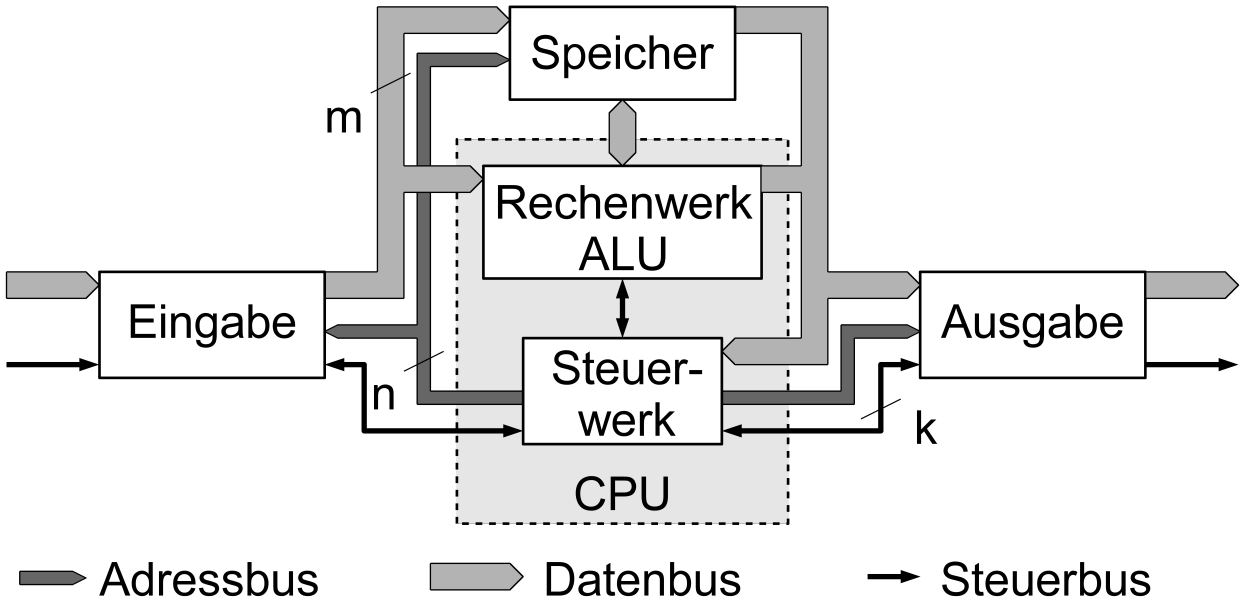


Quelle: Wikipedia

Von-Neumann-Architektur

- Rechner := Steuerwerk, Rechenwerk, Speicher, Eingabewerk, Ausgabewerk.
- Eingabe von Programmen und Daten in den Speicher.
- Daten, Programme, Zwischen- und Endergebnisse liegen im gleichen Speicher.
- Speicher ist in gleichgroße nummerierte (addressierte) Zellen unterteilt.
- Über die Adresse einer Speicherzelle kann deren Inhalt abgerufen/verändert werden.
- Aufeinanderfolgende Befehle eines Programms liegen in benachbarten Speicherzellen.
- Steuerwerk ruft den nächsten Befehl durch Erhöhen der Befehlsaddresse um 1 auf.
- Sprungbefehle erlauben eine Abweichung von der gespeicherten Reihenfolge
- Grundlegende Befehle sind
  - Arithmetische Befehle (z.B. Addition, Multiplikation)
  - Logische Vergleiche (z.B. logisches UND, logisches ODER)
  - Transportbefehle (z.B. Eingabewerk $\rightarrow$ Speicher, Speicher $\rightarrow$ Rechenwert)
- Alle Daten (z.b. Befehle, Adressen) werden binär codiert
- Binäre Enkodierung/Dekodierung geschieht durch geeignete Schaltwerke.

Von-Neumann-Architektur



Quelle: Wikipedia

- SISD System (single instruction stream, single data stream)
- Befehls- und Operandenfolge mit streng sequentieller Abarbeitung

Sequentielle Abarbeitung von Befehlen ist Grundprinzip der Progammierung

Datenanalyse und Programmierung

- Computergestützte Datenanalyse
- Informatik und Rechnerarchitektur
- **Algorithmen und Programme**
- Selbstkontrollfragen

Vom Realwertproblem zum Programm

Realwertproblem

- Das Problem, das mithilfe eines Computers gelöst werden soll.
- Bspw. Auswertung von Fragebogendaten einer psychologischen Studie.

Problemspezifikation

- Genaue sprachliche Fassung des Realweltproblems
- Bspw. der Methodenteil einer wissenschaftlichen Publikation.

Algorithmus

- Folge von Anweisungen zur Lösung des Problems.
- Bspw. Dateneinlesen, deskriptive Statistiken berechnen, T-Test durchführen.

Programm

- Ein Algorithmus, der von einem Computer ausgeführt werden kann.
- Eine in einer Programmiersprache verfasste Textdatei.

Definition (Algorithmus)

Ein *Algorithmus* ist eine Folge von Anweisungen, um aus gewissen Eingabedaten bestimmte Ausgabedaten herzuleiten, wobei folgende Bedingungen erfüllt sein müssen

- *Finitheit.* Die Anweisungsfolge ist in einem endlichen Text vollständig beschrieben sein.
- *Effektivität.* Jede Anweisung muss tatsächlich ausführbar sein.
- *Terminierung.* Der Algorithmus endet nach endlich vielen Anweisungen.
- *Determiniertheit.* Der Ablauf des Algorithmus ist zu jedem Punkt fest vorgeschrieben.

Wenn E die Menge der zulässigen Eingabedaten und A die Menge der zulässigen Ausgabedaten bezeichnet, dann ist ein Algorithmus eine Funktion

$$f : E \to A, e \mapsto f(e) \tag{1}$$

Umgekehrt heißen Funktionen, die durch einen Algorithmus beschrieben werden können, *berechenbare Funktionen*.

Bemerkung

- Effektivität sollte nicht mit Effizienz verwechselt werden.

Programmiersprache

- Bestimmt die Regeln, denen ein Programm gehorchen muss
- Definiert eine Syntax, also Vokabular und Programmaufbau
- Definiert Semantik, also die Bedeutung der erlaubten Anweisungen

```
1
2  // Quellcodebeispiel in C++
3
4  #include <cstdlib>
5  #include <iostream>
6
7  using namespace std;
8
9  int main(int argc, char *argv[])
10 {
11     int alter; // Variable vom Typ Integer
12
13     cout << "Wie alt bist du?";
14     cin >> alter;
15     cout << "Du bist " << alter << " Jahre alt" << endl;
16     getc();
17     return 0;
18 }
19
```

Quelle: Wikipedia

Maschinensprache

- Elementare Operationsbefehle (z.b. Speichern, Vergleichen, Addieren)
- Elementare Operationsbefehle werden als Binärzahlen kodiert

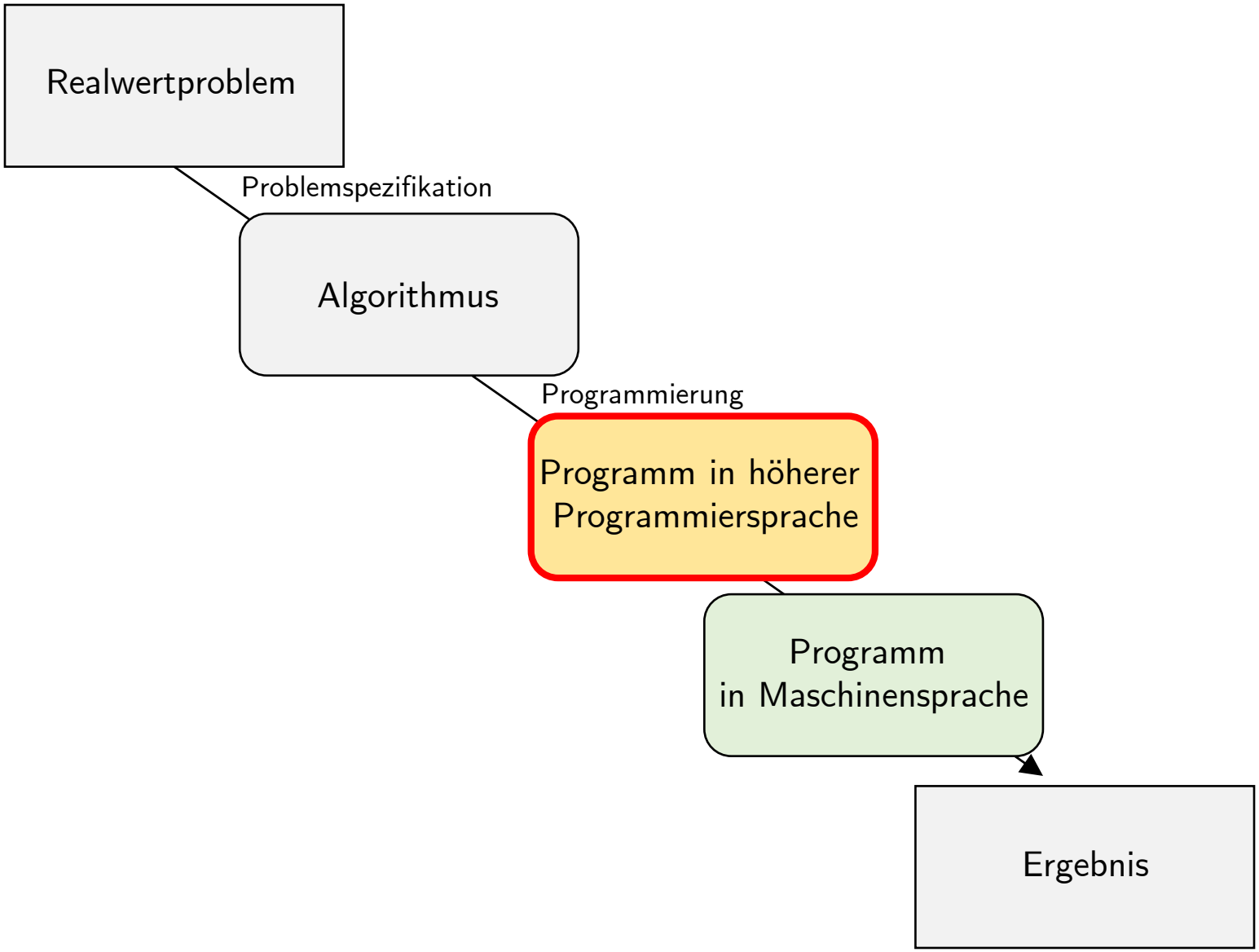
Addiere Inhalt R1 zu Inhalt R2	$\Rightarrow$ 1001 0010
Erhöhe Inhalt R um 1	$\Rightarrow$ 1001 0110
Übertrage Inhalt R1 nach R3	$\Rightarrow$ 0010 0011

- Programme in Maschinensprache heißen *Maschinenprogramme*
- De facto führt ein Computer nur Maschinenprogramme aus
- Für Menschen ist die Programmierung in Maschinensprache mühselig.

Höhere Programmiersprache

- An die menschliche Sprache angelehnte Wörter und Sätze
- Interpreter oder Compiler übersetzen Programme in Maschinensprache
- R, Python, Matlab, C++, Java, FORTRAN, COBOL,...

Algorithmen und Programme



Programmierparadigmen

Imperative Programmierung

Problemlösungsweg wird als Folge von Anweisungen (Befehlen) vorgegeben

Befehle verarbeiten Daten, die mithilfe von *Variablen* adressiert werden.

- Prozedurale imperative Programmierung
  - Daten und sie manipulierende Befehle werden separat behandelt
  - Prozeduren (Funktionen) als zentrales Strukturkonzept
- Objektorientierte imperative Programmierung
  - Daten und manipulierende Befehle werden als *Objekt* zusammengefasst
  - Objekte als zentrales Strukturkonzept

Deklarative Programmierung (hier nicht von Relevanz)

Beschreibung des Problems steht im Vordergrund.

Der Lösungsweg wird nach vorgegebenen Regeln vom Computer ermittelt.

Generationen von Programmiersprachen

1. Generation (1GL)
- Maschinensprachen
- 10110000 01100001 (in hexadezimaler Darstellung: B0 61)

2. Generation (2GL)
- Assemblersprachen ab 1950, erste Form der symbolischen Programmierung
- Bspw. “MOV Al, 61H” # Intel-Prozessor-spezifische Sprache

3. Generation (3GL)
- Höhere Programmiersprachen ab 1970 wie FORTRAN, C, C++, Java
- Programmierfreundlich, prozessor-unabhängig

4. Generation (4GL)
- Höhere Programmiersprachen ab 1980 wie Python, Matlab, R
- Codeoverhead Minimisierung, Automation, Flexibilität, Multiparadigmatisch

Kompilierte Programmiersprachen

- Gesamter Quellcode wird vor Ausführung in Maschinensprache übersetzt.
- Das Übersetzungsprogramm heißt *Compiler*.
- Der übersetzte Maschinencode wird vom Prozessor ausgeführt.
- Das ausführbare Programm wird nicht übersetzt und läuft schnell.
- Bei Änderungen des Quellcodes muss neu kompiliert werden.
- Beispiele für kompilierte Sprachen sind Java, C, C++.

Interpretierte Programmiersprachen

- Quellcode wird während der Ausführung in maschinennahe Sprache übersetzt.
- Das Ausführungsprogramm heißt *Interpreter*.
- Das Programm läuft aufgrund der Interpretation langsamer.
- Bei Änderungen des Quellcodes muss nicht neu interpretiert werden.
- Beispiele für interpretierte Sprachen sind Python und R.

Die Programmiersprache R ist

... eine imperative Programmiersprache ,

... perse objektorientiert, kann aber prozedural genutzt werden,

... eine höhere Programmiersprache der 4. Generation,

... eine interpretierte Sprache,

... auf die statistische Analyse von Daten zugeschnitten.

Datenanalyse und Programmierung

- Computergestützte Datenanalyse
- Informatik und Rechnerarchitektur
- Algorithmen und Programme
- **Selbstkontrollfragen**

Selbstkontrollfragen

1. Geben Sie die typische Struktur einer computergestützten Datenanalyse wieder.
2. Erläutern Sie den Begriff "Datenanalyseskript".
3. Definieren Sie den Begriff "Informatik".
4. Erläutern Sie die Akronyme CPU, RAM, SSD, und GPU.
5. Nennen Sie wesentliche Aspekte der Von-Neumann Rechnerarchitektur.
6. Definieren Sie den Begriff des Algorithmus.
7. Erläutern Sie den Zusammenhang von Algorithmen und Programmen.
8. Was bezeichnen die Syntax und Semantik einer Programmiersprache?
9. Differenzieren Sie die Begriffe "Maschinensprache" und "höhere Programmiersprache".
10. Erläutern Sie die Prinzipien der prozeduralen und objektorientierten imperativen Programmierung.
11. Skizzieren Sie die Entwicklung der Programmiersprachen der ersten bis vierten Generation.
12. Grenzen Sie die Begriffe der kompilierten und interpretierten Programmiersprache voneinander ab.

Literatur

Cotton, R. (2013). *Learning R*. O'Reilly, Beijing ; Sebastopol, CA, first edition edition.

Hattenhauer, R. (2020). *Informatik*. Pearson, second edition.

Sauer, S. (2019). *Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren*. FOM-Edition. Springer Fachmedien Wiesbaden, Wiesbaden.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(2) R und RStudio Grundlagen

Literaturhinweise

Als allgemeine Grundlage dienen Cotton (2013) und Sauer (2019). Die Diskussion der Variablenrepräsentation in R folgt Wickham (2019, Section 2). Die einführende Diskussion der Datenstrukturen in R basiert auf Fußeder (2018).

R und RStudio Grundlagen

- R und RStudio
- Arithmetik, Logik und Präzedenz
- Variablen
- Datenstrukturen
- Übungen und Selbstkontrollfragen

R und RStudio Grundlagen

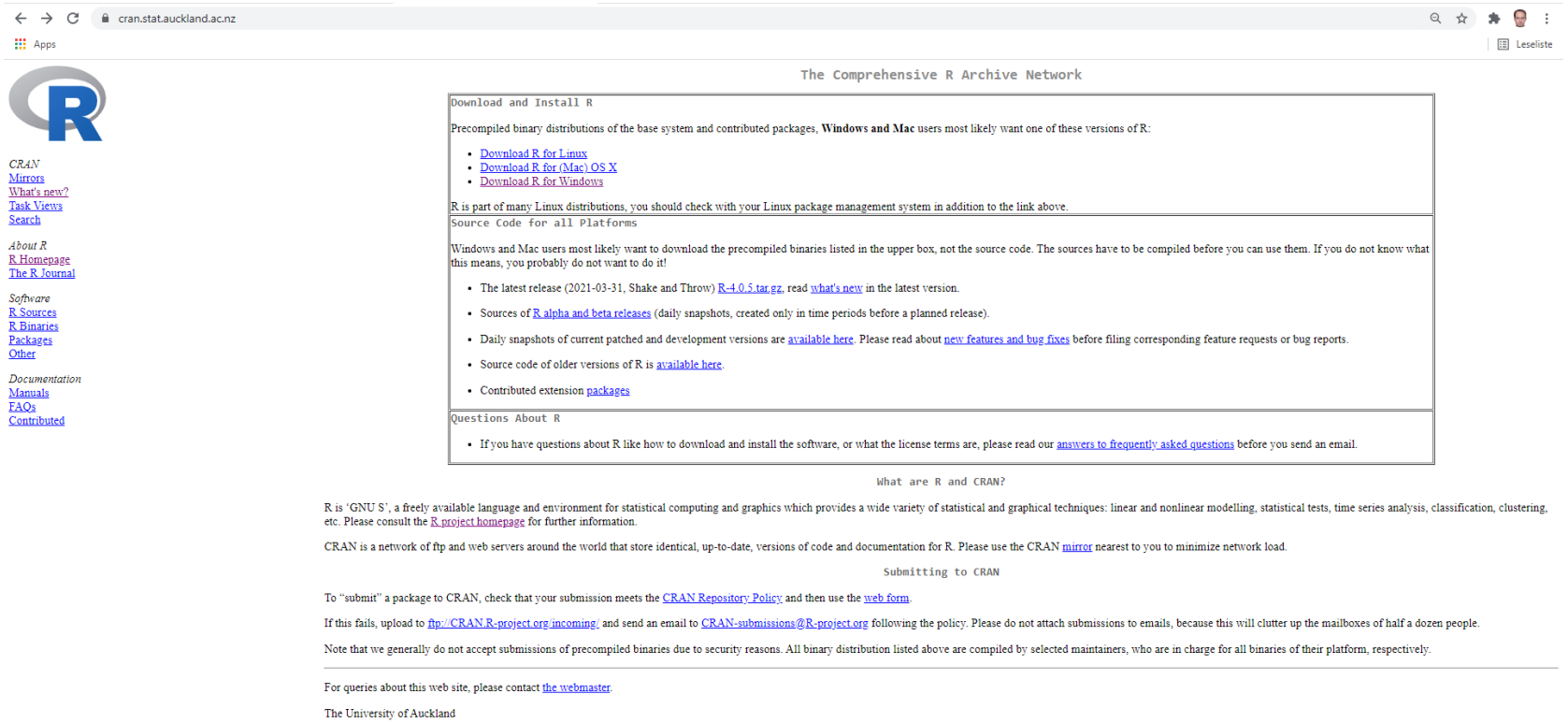
- **R und RStudio**
- Arithmetik, Logik und Präzedenz
- Variablen
- Datenstrukturen
- Übungen und Selbstkontrollfragen

Was ist R?

- Eine Programmiersprache und ein Softwarepaket.
- Entwickelt von Ihaka and Gentleman (1996).
- Freier Dialekt der propietären Software S (Becker et al., 1988).
- Weiterentwickelt und gepflegt durch R Core Team und R Foundation.
- Interpretierte imperativ-objektorientierte 4GL Sprache.
- Optimiert und populär für statistische Datenanalysen.
- Große Community mit etwa 20000 beigetragenen R Paketen (Erweiterungen)
- Evolviert und konservativ im Kern, konsistent und progressiv in R Paketen.

Wie bekommt man R?

Runterladen (z.B. `https://cran.stat.auckland.ac.nz/`) und installieren



Was kann man mit R machen?

- Datensätze laden, manipulieren, und speichern.
- Eine Vielzahl von Berechnungen an verschiedenen Datenstrukturen durchführen.
- Eine Vielzahl statistischer Analyse Methoden auf Daten anwenden.
- Datenanalyseskripte schreiben und Abbildungen generieren.

Was kann man mit R (nicht so gut) machen?

- In einer ansprechenden Umgebung programmieren ($\rightarrow$ RStudio).
- Scientific Computing ($\rightarrow$ Matlab, Julia).
- Psychologische Experimente programmieren ($\rightarrow$ Python, Matlab)
- Apps oder Webseiten programmieren ($\rightarrow$ Java)

Wie bekommt man Hilfe zu R?

- Googlen
- `https://stackoverflow.com/`
- Während der Programmierung und bei bekanntem Funktionsnamen

```
> ?mean
> help(mean)
```

- Für längere Tutorials

```
>browseVignettes()
```

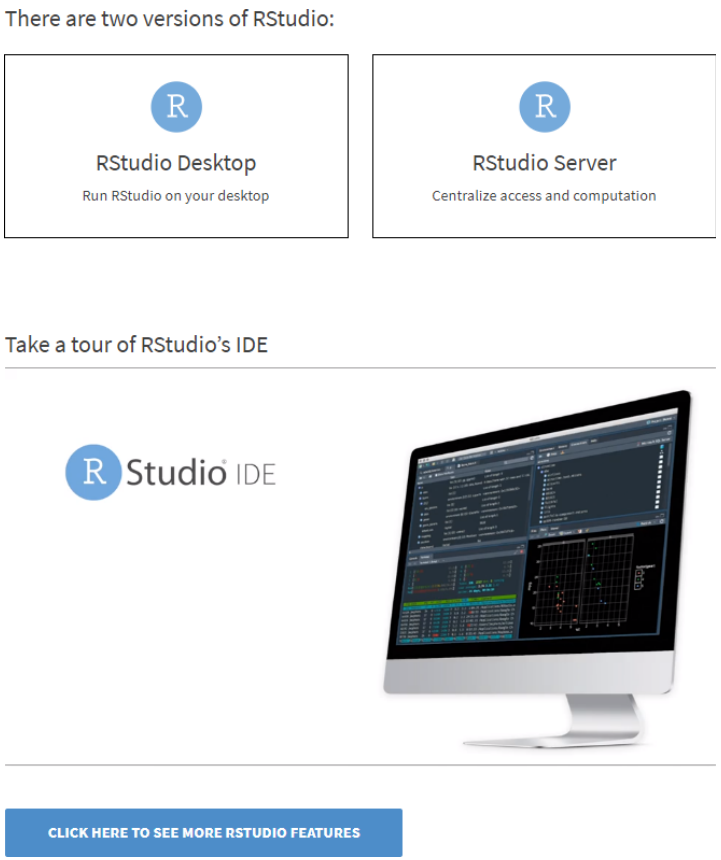
- `https://rseek.org/`
- `https://www.rstudio.com/resources/cheatsheets/`
- `https://www.r-bloggers.com/`

Was ist RStudio?

- Eine Softwareentwicklungsumgebung für R
- Softwareentwicklungsumgebung = Integrated Development Environment
- IDEs sind Programme zum Programmieren mit einer Programmiersprache
- Kommandozeile, Skripteditor, Vielzahl weiterer Tools
- Freemium Produkt von RStudio, Inc. (IDE frei, Server kostenpflichtig)
- Initial Release 2011, Affero General Public License
- Keine Verbindung zu R Core Team oder R Foundation

Wie bekommt man RStudio?

Runterladen (`https://www.rstudio.com/products/rstudio/`) und installieren

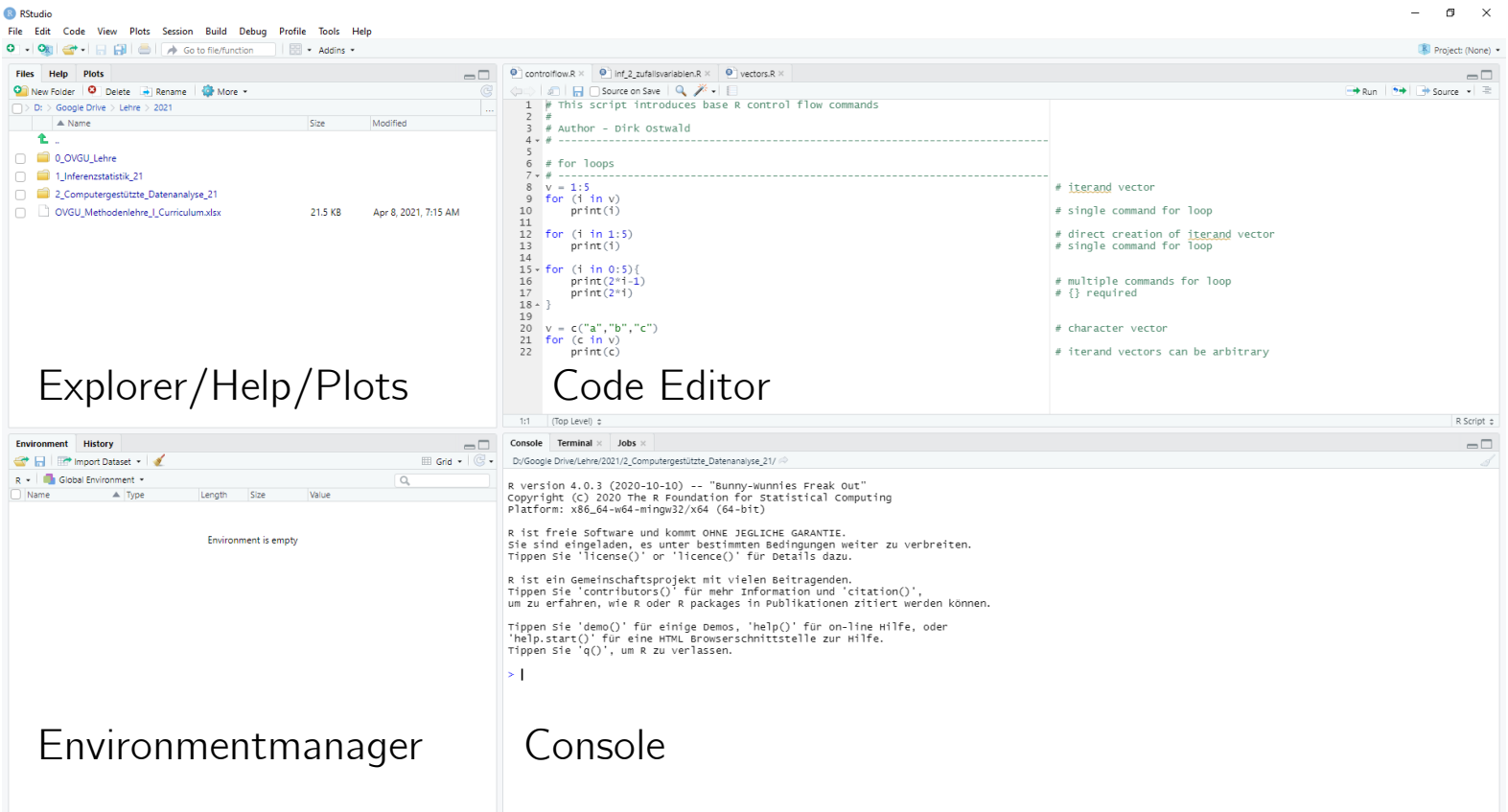


Was kann man mit RStudio machen?

- R Skripte erzeugen, bearbeiten, und laufen lassen
- Laut Eigenwerbung
  - Access RStudio locally
  - Syntax highlighting, code completion, and smart indentation
  - Execute R code directly from the source editor
  - Quickly jump to function definitions
  - View content changes in real-time with the Visual Markdown Editor
  - Easily manage multiple working directories using projects
  - Integrated R help and documentation
  - Interactive debugger to diagnose and fix errors
  - Extensive package development tools

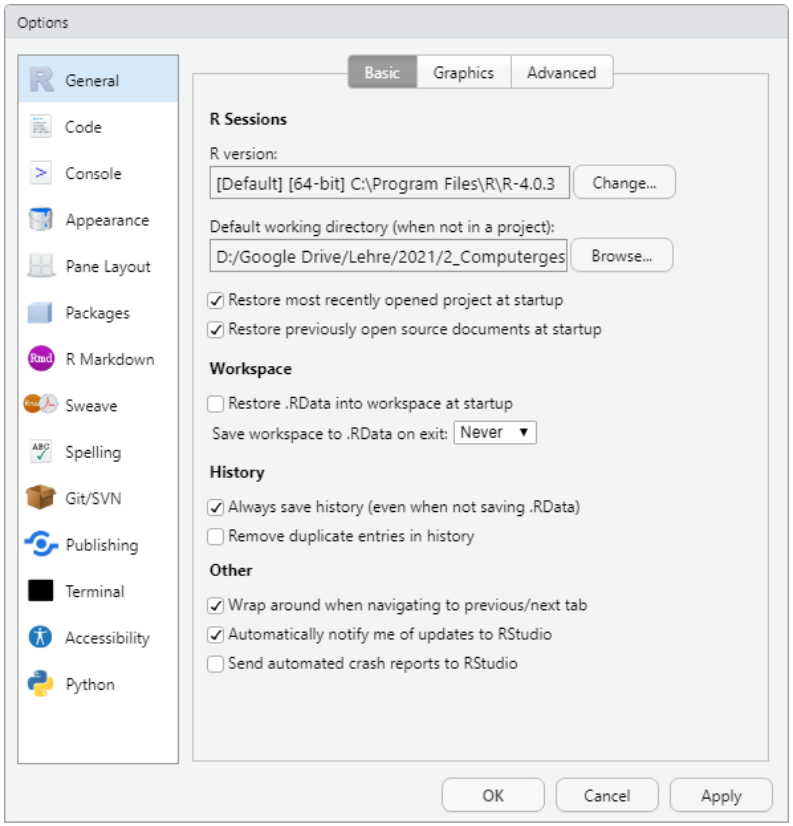
Was kann man mit RStudio machen?

Custom Layout



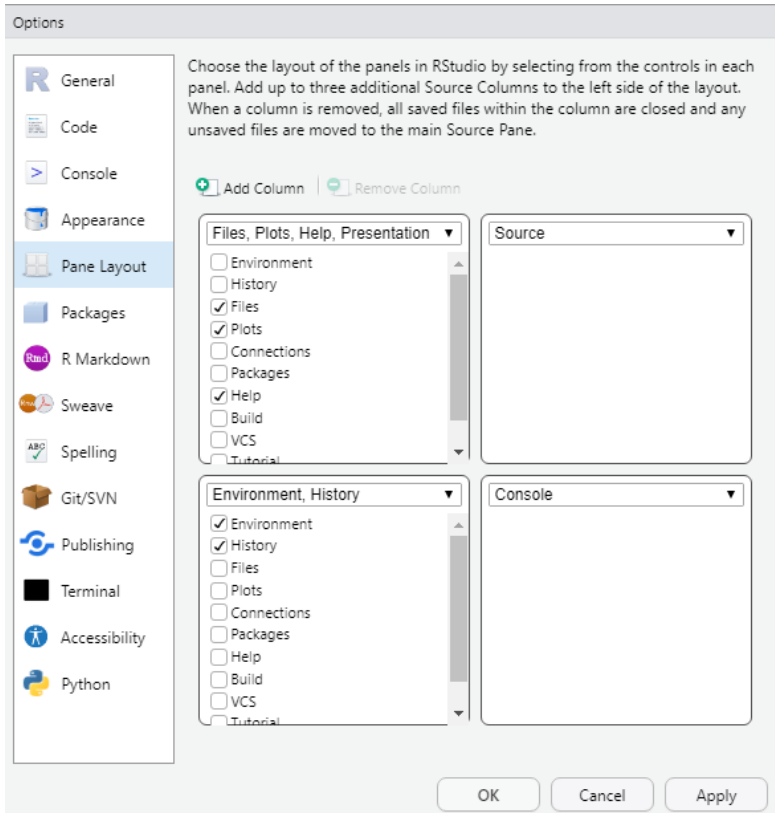
Was kann man mit RStudio machen?

→ Tools → Global Options ...



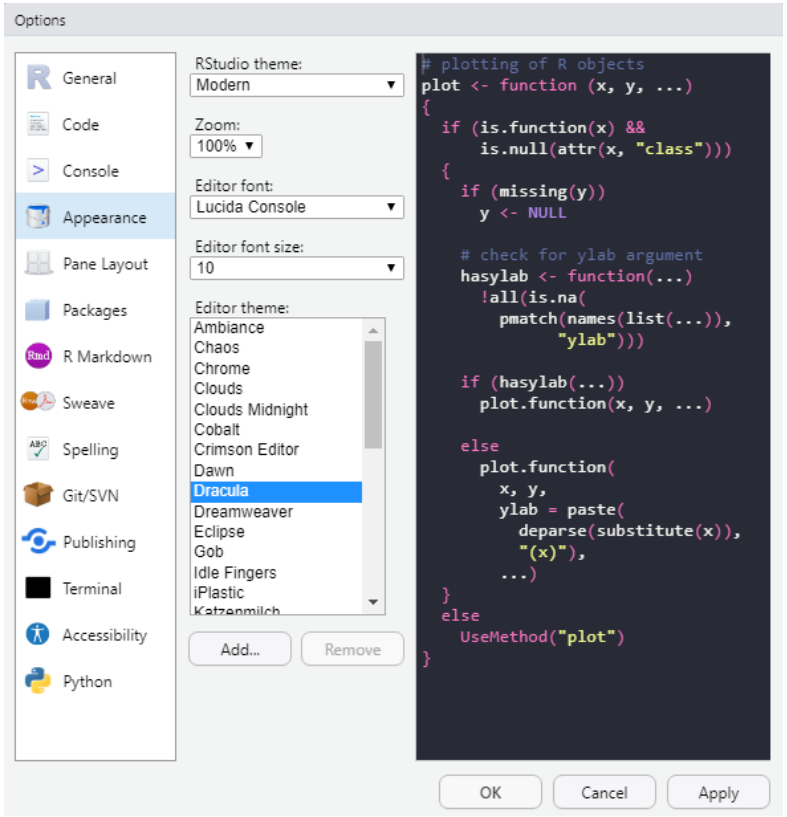
Was kann man mit RStudio machen?

→ Tools → Global Options ...



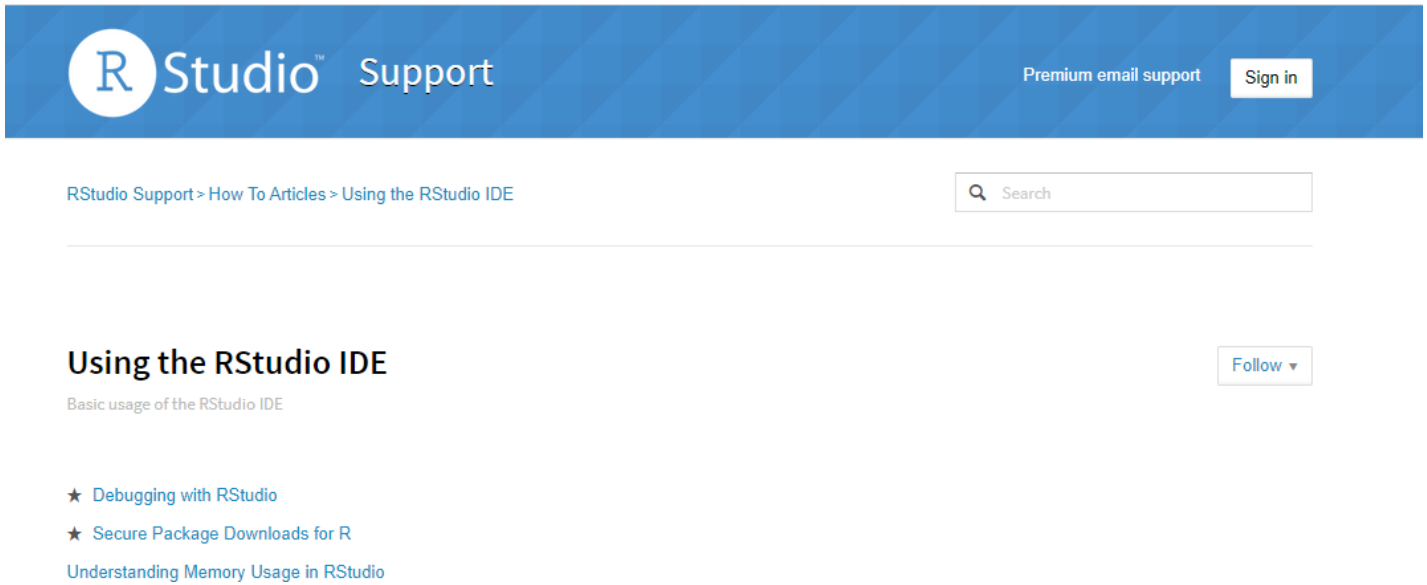
Was kann man mit RStudio machen?

→ Tools → Global Options ...



Wie bekommt man Hilfe zu RStudio?

- Googlen
- `https://support.rstudio.com/hc/en-us/sections/200107586-Using-the-RStudio-IDE`



R Kommandozeile | Working in the Console

- Eingabe von R Befehlen bei >
- Autocomplete mit Tab
- Vorherige Befehle mit Cursor ↑
- Bereinigen des Konsolenoutputs mit Ctrl + L
- Code Ausführungsstopp mit Esc
- Code-Snippets auf diesen Folien der Form

```
print("Hallo Welt!")
```

sollten Sie immer in der Konsole und einem R Skript nachvollziehen!

R Skripte | Executing and Editing Code

- File → New File → R Script oder Ctrl + Shift + N für neue .R Datei
- Open File oder Ctrl + O zum Öffnen bestehender .R Datei
- Eintippen von

```
print("Hallo Welt!")  # Hinter Hashtags stehen dokumentierende Kommentare
print("Hallo R!")     # Kommentare werden nicht ausgefuehrt
```

- Ausführen der einzelnen Zeile, auf welcher der Cursor ruht

  ⇒ Run oder Ctrl + Enter

- Ausführen aller Zeilen

  ⇒ Source oder Ctrl + Shift + Enter oder

  ⇒ Tickmark bei Source on Save setzen und Ctrl + S

R und RStudio Grundlagen

- R und RStudio
- **Arithmetik, Logik und Präzedenz**
- Variablen
- Datenstrukturen
- Übungen und Selbstkontrollfragen

R Konsole als Taschenrechner

```
1+1
[1] 2
2*3
[1] 6
6/2
[1] 3
sqrt(2)
[1] 1.414214
exp(0)
[1] 1
log(0)
[1] -Inf
log(1)
[1] 0
```

- $[1]$ zeigt das erste und einzige Element des Ausgabevektors an
- Vektoren werden noch im Detail behandelt.

Arithmetische Operatoren

Operator	Bedeutung
$+$	Addition
$-$	Subtraktion
$*$	Multiplikation
$/$	Division
$\hat{}$ oder $**$	Potenz
$\%*\%$	Matrixmultiplikation
$\%/\%$	Ganzzahlige Teilung ($5\%/\%2 = 2$)
$\%\%$	Modulo ($5\%/\%2 = 1$)

- Matrixmultiplikation, Modulo, ganzzahlige Teilung benötigen wir zunächst nicht.
- Ganzzahlige Teilung gibt das Resultat der ganzzahligen Teilung an.
- Modulo gibt den ganzzahligen Rest bei ganzzahliger Teilung an.

Logische Operatoren

- Die Boolesche Algebra und R kennen zwei logische Werte: TRUE und FALSE
- Bei Auswertung von Relationsoperatoren ergeben sich logische Werte

Relationsoperator	Bedeutung
==	Gleich
!=	Ungleich
<, >	Kleiner, Größer
<=, >=	Kleiner gleich, Größer gleich
|	ODER
&	UND

- $<, <=, >, >=$ werden zumeist auf numerische Werte angewendet.
- $==, !=$ werden zumeist auf beliebige Datenstrukturen angewendet.
- | und & werden zumeist auf logische Werte angewendet.
- Die Funktion xor() implementiert das exklusive ODER.

Mathematische Funktionen

Aufruf	Bedeutung
abs(x)	Betrag
sqrt(x)	Wurzel
ceiling(x)	Aufrunden (ceiling(2.7) = 3)
floor(x)	Abrunden (floor(2.7) = 2)
round(x)	Mathematisches Runden (round(2.5) = 2)
exp(x)	Exponentialfunktion
log(x)	Logarithmus Funktion

- Es handelt sich um eine Auswahl, einen vollständigen Überblick gibt

```
names(methods::.BasicFunsList)
```

- R unterscheidet formal nicht zwischen Operatoren und Funktionen
- Operatoren können mit der Infix Notation als Funktionen genutzt werden

```
2+3
`+`(2,3)
```

Operatorpräzedenz

- Operatorrangfolge
- Regeln der Form “Punktrechnung vor Strichrechnung”
- Vordefinierte Operatorpräzedenz kann durch Klammern überschrieben werden

```
2 * 3 + 4
[1] 10
2 * (3 + 4)
[1] 14
```

- Generell gilt
  - Operatorrangfolge nicht raten oder folgern, sondern nachschauen!

```
?Syntax
```

  - Lieber Klammern setzen, als keine Klammern setzen!
  - Immer nachschauen, ob Berechnungen die erwarteten Ergebnisse liefern!

Operatorpräzedenz

Präzedenz und Ausführungsreihenfolge arithmetischer Operatoren

Operator	Reihenfolge
^	Rechts nach links
−x,+x	Unitäres Vorzeichen, links nach rechts
*, /	Links nach Rechts
+, −	Links nach Rechts

Beispiele

```
2^2^3       # 2^(2^3)   = 2^8 = 256
(2^2)^3     # (2^2)^3   = 4^3 = 64
-1^2        # -(1^2)    = -1
(-1)^2      # (-1)^2    =  1
2+3/4*5     # 2+(3/4)*5 = 2+(0.75*5) = 2+3.75 = 5.75
2+3/(4*5)   # 2+3/(4*5) = 2+3/20 = 2+0.15 = 2.15
```

Operatorpräzedenz

Operator Syntax and Precedence

Description

Outlines R syntax and gives the precedence of operators.

Details

The following unary and binary operators are defined. They are listed in precedence groups, from highest to lowest.

`:: :::`	access variables in a namespace
`$ @`	component / slot extraction
`[ [[`	indexing
`^`	exponentiation (right to left)
`- +`	unary minus and plus
`:`	sequence operator
`%any%`	special operators (including `%%` and `%/%`)
`* /`	multiply, divide
`+ -`	(binary) add, subtract
`< > <= >= == !=`	ordering and comparison
`!`	negation
`& &&`	and
`| ||`	or
`~`	as in formulae
`-> ->>`	rightwards assignment
`<- <<-`	assignment (right to left)
`=`	assignment (right to left)
`?`	help (unary and binary)

Within an expression operators of equal precedence are evaluated from left to right except where indicated. (Note that = is not necessarily an operator.)

R und RStudio Grundlagen

- R und RStudio
- Arithmetik, Logik und Präzedenz
- **Variablen**
- Datenstrukturen
- Übungen und Selbstkontrollfragen

Definition

In der Programmierung ist eine Variable ein abstrakter Behälter für eine Größe, welche im Verlauf eines Rechenprozesses auftritt. Im Normalfall wird eine Variable im Quelltext durch einen Namen bezeichnet und hat eine Adresse im Speicher einer Maschine. Der durch eine Variable repräsentierte Wert kann – im Unterschied zu einer Konstante – zur Laufzeit des Rechenprozesses verändert werden.

(adaptiert von Wikipedia)

Grundlagen

- Variablen sind vom Programmierenden benannte Platzhalter für Werte
- In 3GL Sprachen wird der Variablentyp durch eine Initialisierungsanweisung festgelegt:
  VAR A : INTEGER # A ist eine Variable vom Typ Integer (ganze Zahl)
- In 3GL Sprachen wird Variablen durch eine Zuweisungsanweisung ein Wert zugeschrieben:
  A := 1 # Der Variable A wird der numerische Wert 1 zugewiesen
- In 4GL Sprachen wie Matlab, Python, R werden Variablen durch Zuweisung initialisiert:
  a = 1 # a ist eine Variable vom Typ double, ihr Wert ist 1.
- Der Zuweisungsbefehl in Matlab und Python ist $=$, der Zuweisungsbefehl in R ist $<-$ oder $=$
- Offiziell empfohlen für R ist $<-$, aus Kohärenzgründen benutzen wir hier $=$

Beispiel

Uta geht ins Schreibwarengeschäft und kauft vier Hefte, zwei Stifte, und einen Füller. Wie viele Gegenstände kauft Uta insgesamt?

```
hefte  = 4    # Definition der Variable 'hefte'  und Wertzuweisung 4
stifte = 2    # Definition der Variable 'stifte' und Wertzuweisung 2
fuller = 1    # Definition der Variable 'fuller' und Wertzuweisung 1
```

- Nach Zuweisung existieren die Variablen im Arbeitsspeicher, dem sogenannten *Workspace*
- Die Variablen können jetzt wie Zahlen in Berechnungen genutzt werden

```
gesamt = hefte + stifte + fuller # Berechnung der Gegenstandsanzahl
[1] 7
```

Ein Heft kostet einen Euro, ein Stift kostet zwei Euro, und ein Füller kostet 10 Euro. Wie viel Euro muss Uta insgesamt bezahlen?

```
gesamtpreis = hefte*1 + stifte*2 + fuller*10 # Berechung des Preises
[1] 18
```

Workspace

ls() zeigt die existierenden benutzbaren Variablen im Arbeitsspeicher an

```
ls()            # Anzeigen aller Variablennamen im Workspace
[1] "fuller"   "gesamt"   "gesamtpreis"   "hefte"    "stifte"
```

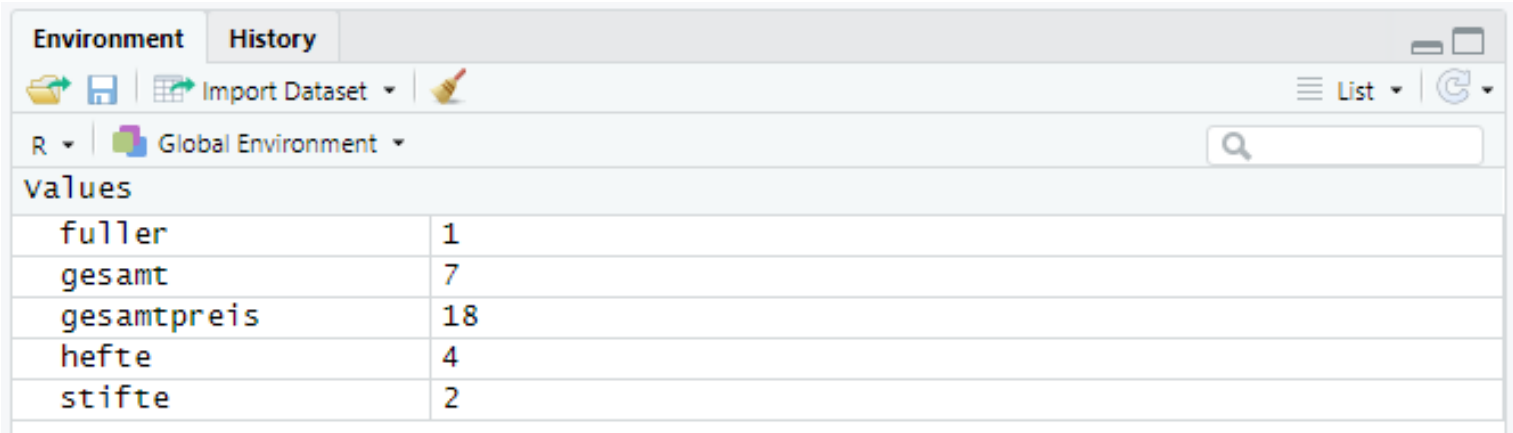
rm() erlaubt das Löschen von Variablen

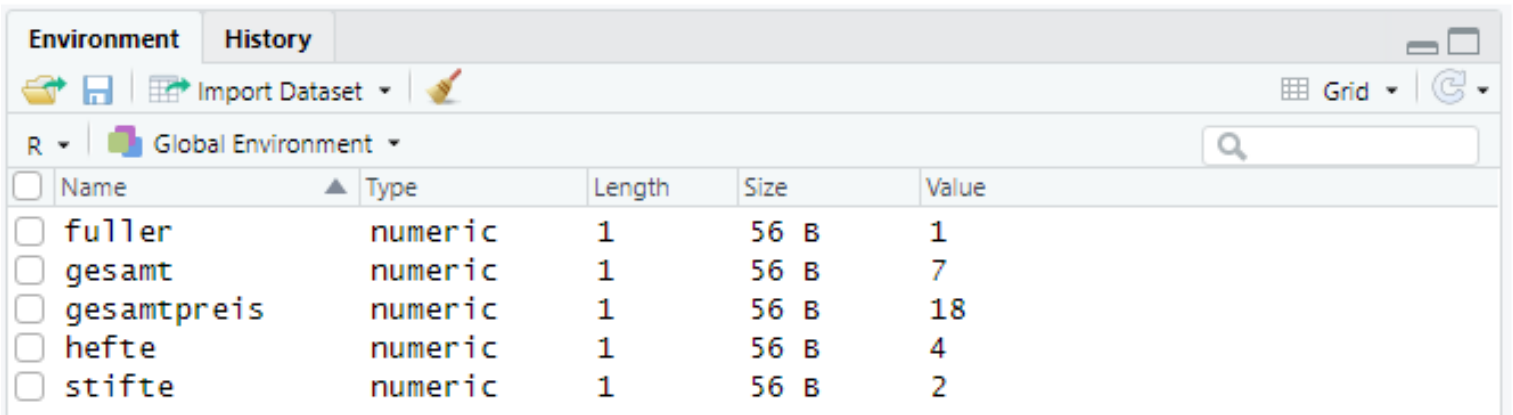
```
rm(gesamtpreis)    # Loeschen der Variable Gesamtpreis
ls()
[1] "fuller"   "gesamt"   "hefte"    "stifte"
```

rm(list=ls()) löscht alle Variablen

```
rm(list = ls())    # Loeschen aller Variablen
ls()
[1] character(0)     # Leerer Workspace
```

Workspace


Environment
History
Import Dataset
List
R
Global Environment
Values
fuller 1
gesamt 7
gesamtpreis 18
hefte 4
stifte 2


Environment
History
Import Dataset
Grid
R
Global Environment
Name
Type
Length
Size
Value
fuller numeric 1 56 B 1
gesamt numeric 1 56 B 7
gesamtpreis numeric 1 56 B 18
hefte numeric 1 56 B 4
stifte numeric 1 56 B 2

Variablennamen

Zulässige Variablennamen

- ... bestehen aus Buchstaben, Zahlen, Punkten (.) und Unterstrichen (_)
- ... beginnen mit einem Buchstaben oder . nicht gefolgt von einer Zahl
- ... dürfen keine reserverd words wie for, if, NaN, usw. sein (>?reserved)
- ... werden unter ?make.names() beschrieben

Sinnvolle Variablennamen

- ... sind kurz ($\approx$ 1 bis 7 Zeichen) und aussagekräftig
- ... bestehen nur aus Kleinbuchstaben und Unterstrichen

Variablenrepräsentation | Binding

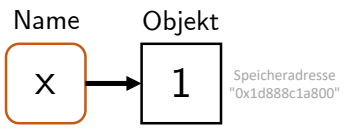
```
x = 1
```

Intuitiv wird eine Variable genannt x mit dem Wert 1 erzeugt.

De-facto geschehen zwei Dinge:

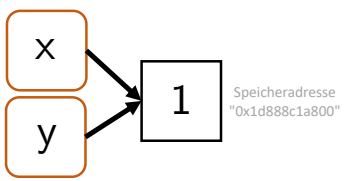
(1) R erzeugt ein Objekt (Vektor mit Wert 1) mit Speicheradresse lobstr::obj_addr(x).

(2) R verbindet dieses Objekt mit dem Namen x, der das Objekt im Speicher referenziert.




```
y = x
```

Intuitiv wird eine Variable genannt y mit Wert gleich dem Wert von x erzeugt.
De-facto wird ein neuer Name y erzeugt, der dasselbe Objekt referenziert wie x:

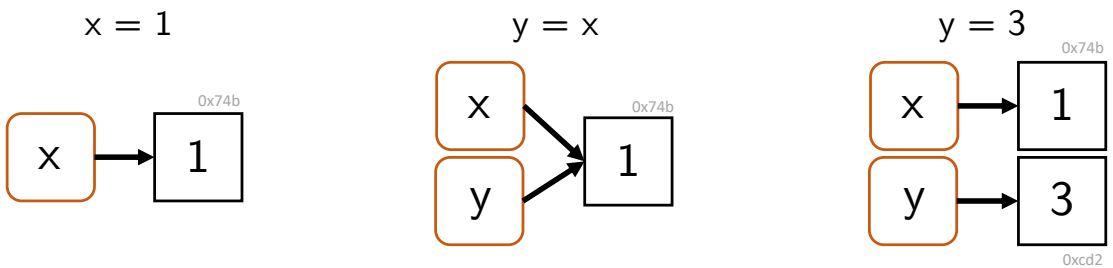


Man überzeuge sich mit lobstr::obj_addr(x) und lobstr::obj_addr(y).

Das Objekt (Vektor mit Wert 1) wird nicht kopiert, R spart Arbeitsspeicher.

Variablenrepräsentation | Copy-on-modify

```
x = 1      # Objekt (0x74b) erzeugt, x referenziert Speicheradresse des Objektes
y = x      # y referenziert dieselbe Speicheradresse wie x (0x74b)
y = 3      # y modifiziert, modifizierte Kopie (0xcd2) wird gespeichert
y
[1] = 3    # y referenziert jetzt (0xcd2)
x
[1] = 1    # x referenziert weiterhin (0x74b)
```



R Objekte sind *immutable*, können also nicht verändert werden

Es gibt allerdings zwei Ausnahmen (*modify-in-place*)

- Objekte mit nur einem gebundenem Namen werden in-place modifiziert

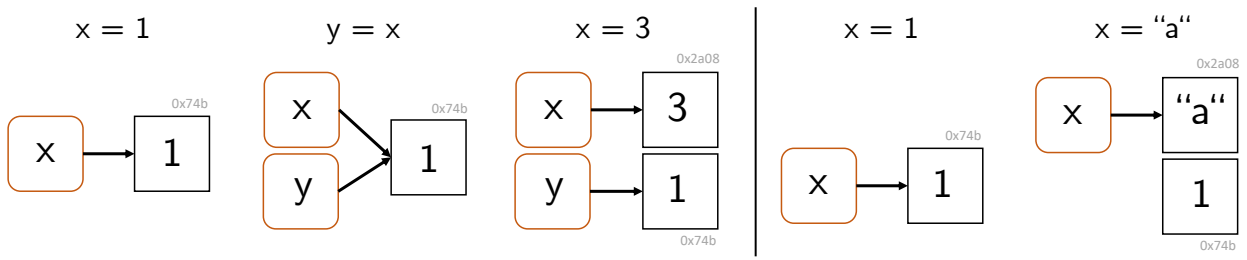
```
x = 1      # Objekt (0x74b) erzeugt, x referenziert Speicheradresse des Objektes
x[1] = 2   # Objekt (0x74b) veraendert
```

- Dieses Verhalten ist allerdings nur in R, nicht innerhalb RStudios reproduzierbar.
- Environments werden in-place modifiziert (→ Environments und Funktionen).

(NEU) Variablenrepräsentation | Unbinding und Carbage Collection

Copy-on-modify gilt auch in umgekehrter Reihenfolge

```
x = 1     # Objekt (0x74b) erzeugt, x referenziert Speicheradresse des Objektes
y = x     # y referenziert dieselbe Speicheradresse wie x (0x74b)
x = 3     # Ein neues Objekt (0x2a08) wird erzeugt, x referenziert (0x2a08)
y
[1] 1     # y referenziert weiterhin Objekt (0x74b)
```



Unbinding

```
x = 1     #  x referenziert Objekt (0x74b)
x = "a"   #  x referenziert Objekt (0x2a08), Objekt (0x74b) jetzt ohne Referenz
```

Carbage collection

- Nicht referenzierte Objekte im Arbeitsspeicher werden automatisch gelöscht.
- Das Löschen geschieht meist erst dann, wenn es wirklich nötig ist.
- Es ist nicht nötig, aktiv die Garbage Collection Funktion **gc()** zu benutzen.

R und RStudio Grundlagen

- R und RStudio
- Arithmetik, Logik und Präzedenz
- Variablen
- **Datenstrukturen**
- Übungen und Selbstkontrollfragen

Klassische Datenstrukturen einer 3GL Programmiersprache

Fundamentale Datenstrukturen

- Vordefiniert innerhalb der Programmiersprache
  - Logische Werte (logical): TRUE, FALSE
  - Ganze Zahlen (integer): int8 (-128,...,127), int16 (–32768,..., 32767)
  - Gleitkommazahlen (single, double): 1.23456, 12.3456, 123.456, ...
  - Zeichen (character): “a”, “b”, “c”, “!”
- Datentyp-spezifische assoziierte Operationen
  - Z.B. AND, OR (logical) +, - (integer) +,-,*, / (single), Zeichenkonkatenation (character)

Zusammengesetzte Datenstrukturen

- Vordefinierte Container zur Zusammenfassung mehrerer Variablen gleichen Datentyps
- Zum Beispiel Vektoren, Listen, Arrays, Matrizen, ...
- Container-spezifische Operationen (Z.B. Vektorindizierung, Matrixmultiplikation, ...)

Selbstdefinierte Datenstrukturen

- Definition eigener Datenstrukturen aus vordefinierten Datenstrukturen und Containern
- Definition eigener Operationen

Datenstrukturenkennenlernen beim Erlernen einer Programmiersprache

Fundamentale Datenstrukturen

- Welche fundamentalen Datenstrukturen bietet die Sprache an?
- Welche Operationen darauf sind bereits definiert?
- Wie lautet die Syntax zur Definition einer Variable eines fundamentalen Datentyps?
- Wie lautet die Syntax, um vordefinierte Operationen aufzurufen?

Zusammengesetzte Datenstrukturen

- Welche Container und zugehörige Operationen bietet die Programmiersprache?
- Wie lautet die Syntax zum Umgang mit einem Containers?

Selbstdefinierte Datenstrukturen

- Wie erzeugt man selbstdefinierte Datenstrukturen und zugehörige Operationen?
- Wie lautet die Syntax zum Umgang mit einer selbstdefinierten Datenstruktur?

Organisation von Daten in R

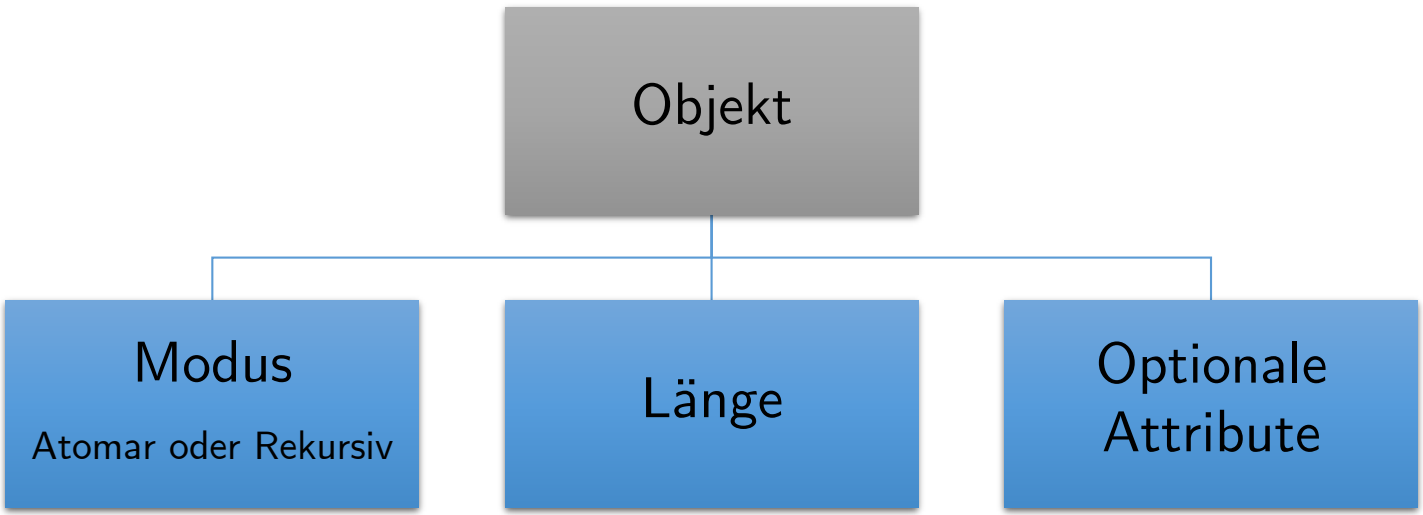
Alles, was in R vorkommt, ist ein **Objekt**

Jedem Objekt kann eindeutig zugeordnet werden

- ein **Modus**
  - Atomar | Komponenten sind vom gleichen Datentyp.
  - Rekursiv | Komponenten können von unterschiedlichem Datentyp sein.
- eine **Länge**
- optional weitere **Attribute**

Organisation der Datenstrukturen in R

Alles, was in R vorkommt, ist ein **Objekt**



Übersicht der R Datentypen

Datentyp	Erläuterung
logical	Die beiden logischen Werte TRUE und FALSE
double	Gleitkommazahlen
integer	Ganze Zahlen
complex	Komplexe Zahlen, hier nicht weiter besprochen
character	Zeichen und Zeichenketten (strings), 'x' oder "Hallo Welt!"
raw	Bytes, hier nicht weiter besprochen

Double und integer werden zusammen auch als numeric bezeichnet.

Viele weitere Typen, hier relevant sind **logical**, **double**, **integer**, **character**.

Übersicht Typen in R

Automatische Festlegung von Datentypen durch Zuweisung

```
b = TRUE          # logical
x = 2.5           # double
y = 1L            # (long) integer
c = 'a'           # character
```

Testen von Datentypen durch **typeof()**

```
typeof(b)
[1] "logical"
typeof(x)
[1] "double"
typeof(y)
[1] "integer"
typeof(c)
[1] "character"
```

Testen von Datetypen durch **is.*()**

```
is.logical(x)
[1] FALSE
is.double(x)
[1] TRUE
```

Übersicht atomare Datenstrukturen in R

Datenstruktur	Erläuterung
Vektor	Container von indizierte Komponenten identischen Typs
Matrix	Interpretation eines Vektors als zweidimensionaler Container
Array	Interpretation eines Vektors als mehrdimensionaler Container
Faktor	Einteilung von Daten in Kategorien

$\Rightarrow$ (3) Vektoren, Matrizen, Arrays, Faktoren

Übersicht rekursive Datenstrukturen in R

Datenstruktur	Erläuterung
Liste	Container von indizierten Komponenten beliebigen Typs
	Insbesondere auch rekursive Struktur, z.B. Liste von Listen
Dataframe	Symbiose aus Liste und Matrix
	Jede Komponente ist Vektor beliebigen Typs identischer Länge
Tibble	Optimierter Dataframe

$\Rightarrow$ (4) Listen, Dataframes, Tibbles

R und RStudio Grundlagen

- R und RStudio
- Arithmetik, Logik und Präzedenz
- Variablen
- Datenstrukturen
- **Übungen und Selbstkontrollfragen**

Übungen und Selbstkontrollfragen

1. Installieren Sie R und RStudio auf Ihrem Rechner.
2. Führen Sie die Befehlssequenz auf Folie R Skripte | Executing and Editing Code aus.
3. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem kommentierten R Skript.
4. Erläutern Sie den Begriff der Operatorpräzedenz.
5. Definieren Sie den Begriff der Variable im Kontext der Programmierung.
6. Erläutern Sie die Begriffe Initialisierungsanweisung und Zuweisungsanweisung für Variablen.
7. Erläutern Sie den Begriff Workspace.
8. Geben Sie jeweils ein Beispiel für einen zulässigen und einen unzulässigen Variablennamen in R.
9. Erläutern Sie die Prozesse, die R im Rahmen einer Zuweisungsanweisung der Form $x = 1$ durchführt.
10. Erläutern Sie den Begriffe Copy-on-modify und Modify-in-place.
11. Diskutieren Sie die klassischen Datenstrukturen einer 3GL Programmiersprache.
12. Diskutieren Sie die Organisation von Datenstrukturen in R.
13. Wodurch unterscheiden sich eine atomare und ein rekursive Datenstruktur in R?
14. Nennen und erläutern Sie vier zentrale Datentypen in R.
15. Nennen und erläutern Sie vier zentrale atomare Datenstrukturen in R.
16. Nennen und erläutern Sie zwei zentrale rekursive Datenstrukturen in R.

Literatur

Becker, R. A., Chambers, J. M., and Wilks, A. R. (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*. Chapman & Hall, London, reprint edition.

Cotton, R. (2013). *Learning R*. O'Reilly, Beijing ; Sebastopol, CA, first edition edition.

Fußeder, W. (2018). übersicht über Datentypen in R.

Ihaka, R. and Gentleman, R. (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3):2999–314.

Sauer, S. (2019). *Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren*. FOM-Edition. Springer Fachmedien Wiesbaden, Wiesbaden.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(3) Vektoren, Matrizen, Arrays, Faktoren

Literaturhinweise

Die Diskussion folgt Cotton (2013, Kapitel 4) und Wickham (2019, Kapitel 3).

Vektoren, Matrizen, Arrays, Faktoren

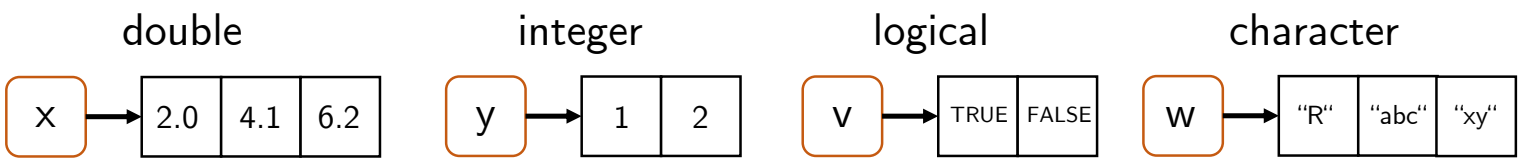
- Vektoren
- Matrizen
- Arrays
- Faktoren
- Übungen und Selbstkontrollfragen

Vektoren, Matrizen, Arrays, Faktoren

- **Vektoren**
- Matrizen
- Arrays
- Faktoren
- Übungen und Selbstkontrollfragen

Übersicht

- Vektoren sind geordnete Folgen von Werten.
- Die einzelnen Werte eines Vektors heißen Elemente.
- Vektoren, deren Elemente vom gleichen Typ sind, heißen **atomic vectors**.
- Die zentralen Typen sind **numeric** (**double**, **integer**), **logical**, **character**.



- Mit dem Begriff **Vektor** ist hier **atomic vector** gemeint.

Elementarwerte

Numeric

- Double wird in Dezimalnotation oder wissenschaftlicher Notation spezifiziert
- Weitere mögliche Werte sind Inf, -Inf, und NaN (Not-a-Number)

```
x = 1                    # Einelementiger Vektor vom Typ double
y = 2.1e2                # Einelementiger Vektor vom Typ double
z = Inf                  # Einelementiger Vektor vom Typ double
```

- Integer wird wie double ohne Dezimalstellen spezifiziert, gefolgt von L (long integer)

```
x = 1L                   # Einelementiger Vektor vom Typ integer
y = 200L                 # Einelementiger Vektor vom Typ integer
```

Logical

- TRUE oder FALSE, abgekürzt T oder F

```
x = TRUE                 # Einelementiger Vektor vom Typ logical
y = F                    # Einelementiger Vektor vom Typ logical
```

Character

- Anführungszeichen ("a") oder Hochkommata ('a')

```
x = "a"                  # Einelementiger Vektor vom Typ character
y = 'test'               # Einelementiger Vektor vom Typ character
```

Erzeugung

Direkte Konkatenation von Elementarwerten mit **c()**

```
x = c(1,2,3)                # numeric vector [1,2,3]
y = c(0,x,4)                # numeric vector [0,1,2,3,4]
s = c("a", "b", "c")        # character vector ["a", "b", "c"]
l = c(TRUE, FALSE)          # logical  vector [TRUE, FALSE]
```

c() konkateniert die Eingabeargumente und erzwingt einen einheitlichen Datentyp

```
x = c(1, "a", TRUE)         # character vector ["1", "a", "TRUE"]
```

Erzeugen "leerer" Vektoren mit **vector()**

```
v = vector("double",3)      # double vector [0,0,0]
w = vector("integer",3)     # integer vector [0,0,0]
l = vector("logical",2)     # logical vector [FALSE, FALSE]
s = vector("character",4)   # character vector ["", "", "", ""]
```

Erzeugen "leerer" Vektoren mit double(), integer(), logical(), character()

```
v = double(3)               # double vector [0,0,0]
w = integer(3)              # integer vector [0,0,0]
l = logical(2)              # logical vector [FALSE, FALSE]
s = character(4)            # character vector ["", "", "", ""]
```

Erzeugung

Erzeugen von ganzzahligen Sequenzen mithilfe des **Colonoperators**

- **a:b** erzeugt ganzzahlige Sequenzen von a (inklusive) bis b (maximal)

```
x = 0:5                    # [0,1,2,3,4,5]
y = 1.5:6.1                # [1.5, 2.5, 3.5, 4.5, 5.5]
```

Erzeugen von Sequenzen mit **seq()**

- **seq**(from, to, by = ((to - from)/(len - 1), len = NULL, ...)

```
x_1 = seq(0,5)             # wie 0:5, [0,1,2,3,4,5]
x_2 = seq(0,1,len = 5)     # 5 Zahlen zwischen 0 (inkl.) und 1 (inkl.)
                           # [0.00, 0.25, 0.50, 0.75, 1.00]
x_3 = seq(0,2,by = .15)    # 0.15 Schritte zwischen 0 (inkl.) und 2 (max.)
                           # [0.00, 0.15, 0.30, ..., 1.50 1.65 1.80 1.95]
x_4 = seq(1,0,by = -.1)    # -0.1 Schritte zwischen 1 (inkl.) und 0 (min.)
```

- **seq.int()**, **seq_len()**, **seq_along()** als weitere Varianten

```
x_1 = seq.int(0,5)         # wie 0:5, [0,1,2,3,4,5]
x_2 = seq_len(5)           # Natuerliche Zahlen bis 5, [1,2,3,4,5]
x_3 = seq_along(c("a","b")) # wie seq_len(length(c("a", "b")))
```

Charakterisierung

length() gibt die Anzahl der Elemente eines Vektors aus

```
x = 0:10                    # Vektor
length(x)                   # Die Anzahl der Elemente des Vektors
[1] 11                      # ... ist 11
```

typeof() gibt den elementaren Datentyp eines Vektors aus

```
x = 1:3L                    # Vektor
typeof(x)                   # Der Typ des atomic vectors
[1] "integer"               # ... ist integer
y = c(T,F,T)                # Vektor
typeof(y)                   # Der Typ des atomic vectors
[1] "logical"               # ... ist logical
```

- mode(), storage.mode() werden nicht empfohlen, sie existieren für S Kompatibilität

is.logical(), **is.double()**, **is.integer()**, **is.character()** testen den Datentyp

```
is.double(x)                # Testen obigen Vektors
[1] FALSE                   # Der Vektor ist nicht vom Typ double
is.logical(y)               # Testen obigen Vektors
[1] TRUE                    # Der Vektor ist vom Typ logical
```

- is.vector(), is.atomic(), is.numeric() haben spezifischere Funktionen

Typangleichung

Bei Kombination verschiedene Typen wird einheitlicher Datentyp erzwungen (coercion)

Es gilt logical $<$ integer $<$ double $<$ character

```
x = c(1.2,"a")              # Kombination gemischter Elementarwerttypen
[1] "1.2" "a"               # character schlaegt double
typeof(x)                   # Erzeugter Vektor
[1] "character"             # ... ist vom Typ character

y = c(1L, TRUE)             # Kombination  gemischter Elementarwerttypen
[1] 1 1                     # integer schlaegt logical
typeof(y)                   # Erzeugter Vektor
[1] "integer"               # ... ist vom Typ integer
```

as.logical(), **as.integer()**, **as.double()**, **as.character()** erlauben Typangleichung

```
x = c(0,1,1,0)              # double Vektor
y = as.logical(x)           # ... umgewandelt in logical
[1] FALSE  TRUE  TRUE FALSE # logical vector
```

Typangleichung geschieht oft implizit

```
x = c(T, F, T, T)           # logical Vektor
s = sum(x)                  # Summation in integer gewandelter logical Elemente
[1] 3                       # 1 + 0 + 1 + 1
```

Indizierung

Indizierung

- Einzelne oder mehrere Vektorkomponenten werden durch Indizierung adressiert.
- Indizierung wird auch Indexing, Subsetting, oder Slicing genannt.
- Zur Indizierung werden eckige Klammern [ ] benutzt.
- Indizierung kann zur Kopie oder Manipulation von Komponenten benutzt werden.
- Der Index des ersten Elements ist 1 (nicht 0, wie in anderen Sprachen).

```
x     = c("a", "b", "c")    # character vector ["a", "b", "c"]
y     = x[2]                # Kopie von "b" in y
x[3]  = "d"                 # Aenderung von x zu x = ["a", "b", "d"]
```

Prinzipien der Indizierung in R

Indizierung mit ...

- ... einem Vektor positiver Zahlen adressiert entsprechende Komponenten.
- ... mit einem Vektor negativer Zahlen adressiert komplementäre Komponenten.
- ... einem logischen Vektor adressiert die Komponenten mit TRUE.
- ... einem character Vektor adressiert benannte Komponenten.

Indizierung

Beispiele

Indizierung mit einem Vektor positiver Zahlen

```
x = c(1,4,9,16,25)          # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y = x[1:3]                  # 1:3 erzeugt Vektor [1,2,3], x[1:3] = [1,4,9]
z = x[c(1,3,5)]             # c(1,3,5) erzeugt Vektor [1,3,5], x[c(1,3,5)] = [1,9,25]
```

Indizierung mit einem Vektor negativer Zahlen

```
x = c(1,4,9,16,25)          # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y = x[c(-2,-4)]             # Alle Komponenten ausseer 2 und 4, x[c(-2,-4)] = [1,9,25]
z = x[c(-1,2)]              # Gemischte Indizierung nicht erlaubt (Fehlermeldung)
```

Indizierung mit einem logischen Vektor

```
x = c(1,4,9,16,25)          # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y = x[c(T,T,F,F,T)]         # TRUE Komponenten,  x[c(T,T,F,F,T)] = [1,4,25]
z = x[x > 5]                # x > 5 = [F,F,T,T,T], x[x > 5] = [9,16,25]
```

Indizierung mit einem character Vektor

```
x = c(1,4,9,16,25)          # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
names(x) = c("a","b")       # Benennung der Komponenten als [a  b <NA> <NA> <NA>]
y = x["a"]                  # x["a"] = 1
```

Indizierung

R hat eine (zu) hohe Flexibilität bei Indizierung

Out-of-range Indizes verursachen keine Fehler, sondern geben NA aus

```
x = c(1,4,9,16,25)          # [1,4,9,16,25] = [1^2, 2^2, 3^2, 4^2, 5^2]
y = x[10]                   # x[10] = NA (Not Applicable)
```

Nichtganzzahlige Indizes verursachen keine Fehler, sondern werden abgerundet

```
y = x[4.9]                  # x[4.9]  = x[4]  = 16
z = x[-4.9]                 # x[-4.9] = x[-4] = [1,4,9,25]
```

Leere Indizes indizieren den gesamten Vektor

```
y = x[]                     # y = x
```

Arithmetik

Unitäre arithmetische Operatoren und Funktionen werden elementweise ausgewertet

```
a    = seq(0,1,len=11)      # a = [ 0.0    ,  0.1    , ...,  0.9     , 1.0    ]
b    = -a                   # b = [-0.0    , -0.1    , ..., -0.9     ,-1.0    ]
v    = a^2                  # v = [ 0.0^2  ,  0.1^2  , ...,  0.9^2   , 1.0^2  ]
w    = log(a)               # w = [ln(0.0), ln(0.1), ..., ln(0.9), ln(1.0)]
```

Binäre arithmetische Operatoren werden elementweise ausgewertet

Vektoren gleicher Länge

```
a = c(1,2,3)                # a = [1,2,3]
b = c(2,1,4)                # b = [2,1,4]
v = a + b                   # v = [1,2,3] + [2,1,4] = [1+2,2+1,3+4] = [   3,  3,    7]
w = a - b                   # w = [1,2,3] - [2,1,4] = [1-2,2-1,3-4] = [  -1,  1,   -1]
x = a * b                   # x = [1,2,3] * [2,1,4] = [1*2,2*1,3*4] = [   2,  2,   12]
y = a / b                   # y = [1,2,3] / [2,1,4] = [1/2,2/1,3/4] = [0.50,  2, 0.75]
```

Vektoren und Skalare (= Vektoren der Länge 1)

```
a = c(1,2,3)                # a = [1,2,3]
b = 2                       # b = [2]
v = a + b                   # v = [1,2,3] + [2,2,2] = [1+2,2+2,3+2] = [   3,  4,    5]
w = a - b                   # w = [1,2,3] - [2,2,2] = [1-2,2-2,3-2] = [  -1,  2,    1]
x = a * b                   # x = [1,2,3] * [2,2,2] = [1*2,2*2,3*2] = [   2,  4,    6]
y = a / b                   # y = [1,2,3] / [2,2,2] = [1/2,2/2,3/2] = [0.5,  1,  1.5]
```

Recycling

R erlaubt (leider) auch Arithmetik mit Vektoren unterschiedlicher Länge

Bei ganzzahligen Vielfachen der Länge wird der kürzere Vektor wiederholt

```
x = 1:2                    # x = [1,2], length(x) = 2
y = 3:6                    # y = [3,4,5,6], length(y) = 4
v = x + y                  # v = [1,2,1,2] + [3,4,5,6] = [4,6,6,8]
```

Arithmetik von Vektoren und Skalaren ist ein Spezialfall dieses Prinzips

Andernfalls werden die ersten Komponenten des kürzeren Vektors wiederholt

```
x = c(1,3,5)               # x = [1,3,5], length(x) = 3
y = c(2,4,6,8,10)          # y = [2,4,6,8,10], length(y) = 5
v = x + y                  # v = [1,3,5,1,3] + [2,4,6,8,10] = [3,7,11,9,13]
```

Generell sollten nur Vektoren gleicher Länge arithmetisch verknüpft werden!

Fehlende Werte (NA)

Fehlende Werte werden in R mit NA (not applicable) repräsentiert

Das Rechnen mit NAs ergibt (meist) wieder NA

```
3 * NA                    # Multiplikation eines NA Wertes
[1] NA                    # ... ergibt NA
NA < 2                    # Relationaler Vergleich eines NA Wertes
[1] NA                    # ... ergibt NA

NA^0                      # NA hoch 0
[1] 1                     # ... ergibt 1, weil jeder Wert hoch 0 eins ergibt (?)
NA & FALSE                # NA UND FALSE
[1] FALSE                 # ... ergibt FALSE, weil jeder Wert UND FALSE FALSE ergibt
```

Auf NA testet man mit is.na()

```
x = c(NA, 5, NA, 10)      # Vektor mit NAs
x == NA                   # Kein Testen auf NAs
[1] NA NA NA NA           # 5 == NA ist NA, nicht FALSE

is.na(x)                  # Logisches Testen auf NA
[1] TRUE FALSE TRUE FALSE # Resultat
```

Attribute

Attribute sind Metadaten von R Objekten in Form von Schlüssel-Wert-Paaren

attributes() ruft alle Attribute eines Objektes auf

Perse haben atomic vectors keine Attribute

```
a = 1:3                 # ein numerischer Vektor
attributes(a)           # Aufrufen aller Attribute
NULL                    # perse hat ein atomic vector keine Attribute
```

attr() kann zum Aufrufen und Definieren von Attributen genutzt werden

```
attr(a, "S") = "W"      # a bekommt Attribut mit Schluessel S und Wert W
attr(a, "S")            # Das Attribut mit Schluessel S
[1] "W"                 # ... hat den Wert W
```

Attribute werden bei Operationen oft entfernt (Ausnahmen sind **names** und **dim**)

```
b = a[1]                # Kopie des ersten Elements von a in Vektor b
attributes(b)           # Aufrufen aller Attribute von b
NULL                    # Das Attribut S mit Wert W wurde nicht mit kopiert
```

Attribute

Spezifikation des Attributs **names** gibt den Elementen eines Vektors Namen

```
v = c(x=1,y=2,z=3)          # Elementnamengeneration bei Vektorerzeugung
x y z                       # Elementnamenzeile
1 2 3                       # Elementwertezeile
```

Die Namen können zur Indizierung benutzt werden

```
v["y"]                      # Indizierung per Namen
y                           # Elementname
2                           # Elementwert
```

names() kann zum Definieren und Aufrufen von Namen benutzt werden

```
y = 4:6                     # Erzeugung eines Vektors
names(y) = c("a","b","c")
names(y)                    # Elementnamenaufruf
[1] "a" "b" "c"             # Elementnamen
```

Benannte Namen können hilfreich sein, wenn der Vektor eine Sinneinheit bildet

```
p = c(age = 31, height = 198, weight = 75)
age height weight           # Alter (Jahre), Groesse (cm), Gewicht (kg) einer Person
 31    198     75
```

Vektoren, Matrizen, Arrays, Faktoren

- Vektoren
- **Matrizen**
- Arrays
- Faktoren
- Übungen und Selbstkontrollfragen

Übersicht

- Matrizen sind zweidimensionale, rechteckige Datenstrukturen der Form

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n_c} \\ m_{21} & m_{22} & \cdots & m_{2n_c} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n_r1} & m_{n_r2} & \cdots & m_{n_rn_c} \end{pmatrix} \tag{1}$$

- Die Elemente $m_{ij}, i = 1, ..., r, j = 1, ..., c$ sind vom gleichen Typ.
- n_r ist die Anzahl der Zeilen (rows), n_c ist die Anzahl der Spalten (columns).
- Jedes Element einer Matrix hat einen Zeilenindex i und einen Spaltenindex j.
- Intuitiv sind Matrizen numerisch indizierte Tabellen.
- Formal sind Matrizen in R zweidimensional interpretierte atomic vectors.
- Matrizen in R sind nicht identisch mit dem mathematischen Matrixbegriff.
- Matrizen in R können allerdings für Lineare Algebra verwendet werden.
- Lineare Algebra ist die Sprache (linearer) statistischer Modelle.

Erzeugung

Die **matrix()** Funktion befüllt Matrizen mit Vektorelementen

- matrix(data, nrow, ncol, byrow)

```
matrix(c(1:12), nrow = 3)           # 3 x 4 Matrix der Zahlen 1,...,12, byrow = F
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

matrix(c(1:12), ncol = 4)           # 3 x 4 Matrix der Zahlen 1,...,12, byrow = F
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

matrix(c(1:12), nrow = 3, byrow = T)  # 3 x 4 Matrix der Zahlen 1,...,12, byrow = T
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Erzeugung

Die Funktion **cbind()** konkateniert passende Matrizen spaltenweise

```
A = matrix(c(1:4) , nrow = 2)      # 2 x 2 Matrix der Zahlen 1,...,4
     [,1] [,2]
[1,]    1    3
[2,]    2    4


B = matrix(c(5:10), nrow = 2)      # 2 x 3 Matrix der Zahlen 5,...,10
     [,1] [,2] [,3]
[1,]    5    7    9
[2,]    6    8   10


C = cbind(A,B)                     # spaltenweise Konkatenierung von A und B
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Erzeugung

Die Funktion **rbind()** konkateniert passende Matrizen reihenweise

```
A = matrix(c(1:6) , nrow = 2, byrow = T)  # 2 x 3 Matrix der Zahlen 1,...,6
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6


B = matrix(c(7:9), nrow = 1)          # 1 x 3 Matrix der Zahlen 5,...,10
     [,1] [,2] [,3]
[1,]    7    8    9


C = rbind(A,B)                        # reihenweise Konkatenierung von A und B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Charakterisierung

typeof() gibt den elementaren Datentyp einer Matrix aus

```
A = matrix(c(T,T,F,F), nrow = 2)      # 2 x 2 Matrix von Elementen vom Typ logical
typeof(A)
[1] "logical"

B = matrix(c("a","b","c"), nrow = 1)  # 1 x 3 Matrix von Elementen vom Typ character
typeof(B)
[1] "character"
```

nrow() und **ncol()** geben die Zeilen- bzw. Spaltenanzahl aus

```
C = matrix(1:12, nrow = 3)            # 3 x 4 Matrix
nrow(C)                               # Anzahl Zeilen
[1] 3
ncol(C)                               # Anzahl Spalten
[1] 4
```

Indizierung

Generell gilt

- Matrixelemente werden mit einen Zeilenindex und einem Spaltenindex indiziert.
- Die Indexreihenfolge ist immer 1. Zeile, 2. Spalte.
- Die Prinzipien der Indizierung entsprechen der Vektorindizierung.
- Indizes verschiedener Dimensionen können unterschiedlich indiziert werden.
- Eindimensionale Resultate liegen als Vektor, nicht als Matrix vor.

```
A = matrix(c(2:7)^2, nrow = 2)     # 2 x 3 Matrix der Zahlen 2^2,...,7^2
     [,1] [,2] [,3]
[1,]    4   16   36
[2,]    9   25   49

a_13  = A[1,3]                     # Element in 1. Zeile, 3. Spalte von A [36]
a_22  = A[2,2]                     # Element in 2. Zeile, 2. Spalte von A [35]
a_2.  = A[2,]                      # Alle Elemente der 2. Zeile [9,25,49]
a_.3  = A[,3]                      # Alle Elemente der 3. Spalte [36,49]
A_12  = A[1:2,1:2]                 # Submatrix der ersten zwei Zeilen und Spalten
A10   = A[A>10]                    # Elemente von A groesser 10 [16,25,36,49]
A_13  = A[1,c(F,F,T)]              # Element in 1. Zeile, 3. Spalte von A [36]
```

Arithmetik

Unitäre arithmetische Operatoren und Funktionen werden elementweise ausgewertet

```
A = matrix(c(1:4), nrow = 2)        # 2 x 2 Matrix der Zahlen 1,2,3,4
     [,1] [,2]
[1,]    1    3
[2,]    2    4

B = A^2                             # B[i,j]  = A[i,j]^2,  1 <= i,j <= 2
     [,1] [,2]
[1,]    1    9                      # 1^2, 3^2
[2,]    4   16                      # 2^2, 4^2

C = sqrt(B)                         # C[i,j]  = sqrt(A[i,j]^2),  1 <= i,j <= 2
     [,1] [,2]
[1,]    1    3                      # sqrt(1^2), sqrt(3^2)
[2,]    2    4                      # sqrt(2^2), sqrt(4^2)

D = exp(A)                          # D[i,j] = exp(A[i,j]),  1 <= i,j <= 2
     [,1] [,2]
[1,]  2.7 20.0                      # exp(1), exp(3)
[2,]  7.4 54.6                      # exp(2), exp(4)
```

Arithmetik

Matrizen passender Größe können mit binären arithmetischen Operatoren verknüpft werden

Binäre arithmetische Operatoren +,-,*,\werden bei gleicher Größe elementweise ausgewertet

```
A = matrix(c(1:4), nrow = 2)        # 2 x 2 Matrix der Zahlen 1,2,3,4
     [,1] [,2]
[1,]    1    3
[2,]    2    4

B = matrix(c(5:8), nrow = 2)        # 2 x 2 Matrix der Zahlen 5,6,7,8
     [,1] [,2]
[1,]    5    7
[2,]    6    8

C = A + B                           # C[i,j] = A[i,j] + B[i,j], 1 <= i,j <= 2
     [,1] [,2]
[1,]    6   10                      # 1 + 5, 3 + 7
[2,]    8   12                      # 2 + 6, 4 + 8

D = A * B
     [,1] [,2]
[1,]    5   21                      # 1 * 5, 3 * 7
[2,]   12   32                      # 2 * 6, 4 * 8
```

Arithmetik

Mit R Matrizen kann Lineare Algebra betrieben werden

- Addition, Subtraktion, Hadamardprodukt elementweise definiert wie oben

Multiplikation, Transposition, Inversion, Determinante

```
C = A  %*% B                    # 2 x 2 Matrixprodukt
      [,1] [,2]
[1,]   23   31                  # 1*5 + 3*6, 1*7+3*8
[2,]   34   46                  # 2*5 + 4*6, 2*7+4*8

A_T = t(A)                      # Transposition von A
      [,1] [,2]
[1,]    1    2                  # A[1,1], A[2,1]
[2,]    3    4                  # A[1,2], A[2,2]

A_inv = solve(A)                # Inverse von A
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

A_det = det(A)                  # Determinante von A
[1] -2                          # 1*4 - 2*3
```

Attribute

Formal sind Matrizen atomic vectors mit einem **dim** Attribut

```
A = matrix(1:12, nrow = 4 )          #  4 x 3 Matrix
attributes(A)                        # Aufrufen der Attribute von A
$dim                                 # Schluessel
[1] 4 3                              # Wert: Anzahl Zeilen, Anzahl Spalten
```

rownames() und **colnames()** spezifizieren das Attribut **dimnames**

```
rownames(A) = c("P1","P2","P3","P4")  # Benennung der Zeilen von A
colnames(A) = c("Age", "Hgt", "Wgt")  # Benennung der Spalten von A
A                                     # A mit Attribut dimnames
   Age Hgt Wgt
P1   1   5   9
P2   2   6  10
P3   3   7  11
P4   4   8  12
attr(A, "dimnames")                   # Aufrufen des Attributs dimnames
[[1]]
[1] "P1" "P2" "P3" "P4"
[[2]]
[1] "Age" "Hgt" "Wgt"
```

Bei Matrizen ist die Benennung von Zeilen und Spalten eher ungewöhnlich.

Vektoren, Matrizen, Arrays, Faktoren

- Vektoren
- Matrizen
- **Arrays**
- Faktoren
- Übungen und Selbstkontrollfragen

Übersicht

- Arrays sind d-dimensionale, hyperrechteckige Datenstrukturen.
- Arrays sind die Generalisierung von Matrizen auf mehr als zwei Dimensionen.
- Für $d = 1$ entspricht ein Array einem Vektor, für $d = 2$ einer Matrix.
- Die Elemente eines Arrays sind vom gleichen Typ.
- $d_i, i = 1, ..., d$ ist die maximalen Anzahl von Elementen der $i$ten Dimension.
- Jedes Element eines Arrays hat einen d-dimensionalen Index $i_1, i_2, ..., i_d$.
- Formal sind Arrays in R d-dimensional interpretierte atomic vectors.

Erzeugung

Die array() Funktion befüllt Arrays mit Vektorelementen

- array(data, dim,...)
- dim enkodiert den maximalen Index in jeder der length(dim) Dimensionen

```
A = array(1:12, dim = c(2,2,3))        # 2 x 2 x 3 Array der Zahlen 1,...,12
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8

, , 3
     [,1] [,2]
[1,]    9   11
[2,]   10   12
```

- Es ist sinnvoll, sich von “räumlichen Vorstellungen” höherdimensionaler Arrays zu lösen.
- Arrays sind Datencontainer, Elemente werden durch Indexkombinationen addressiert.

Charakterisierung

length() gibt die Anzahl der Elemente eines Arrays aus

```
A = array(1:12, dim = c(2,2,3))          # 2 x 2 x 3 Array der Zahlen 1,...,12
length(A)                                # Die Anzahl der Elemente des Arrays
[1] 11                                   # ... ist 11
```

typeof() gibt den elementaren Datentyp eines Arrays aus

```
typeof(A)                                # Der Typ des Arrays
[1] "double"                             # ... ist double
```

dim() gibt die Dimensionen eines Arrays aus

```
dim(A)                                   # Dimension des Arrays
[1] 2 2 3                                # ... 2 x 2 x 3
```

Indizierung

Arrayindizierung erfolgt analog zu Vektor- und Matrixindizierung

```
A      = array(1:12, dim = c(2,2,3))   # 2 x 2 x 3 Array der Zahlen 1,...,12
a_223 = A[2,2,3]                       # Arrayelement mit Indexaddresse [2,2,3] (12)
```

Attribute

Formal sind Arrays atomic vectors mit einem **dim** Attribut

```
A = array(1:12, dim = c(2,2,3))        # 2 x 2 x 3 Array der Zahlen 1,...,12
attributes(A)                          # Aufrufen der Attribute von A
$dim                                   # Schluessel
[1] 2 2 3                              # Wert: Anzahl Zeilen, Anzahl Spalten
```

dimnames() kann zur Benennung der Arraydimensionen mit einer Liste benutzt werden.

Die Benennung von Arraydimensionen ist aber eher ungewöhnlich.

Arithmetik

Unitäre arithmetische Operatoren werden elementweise ausgewertet

Binäre arithmetische Operatoren werden bei Arraykonformität elementweise ausgwertet

```
A     = array(1:4, dim = c(1,2,2))    # 1 x 2 x 2 Array der Zahlen 1,...,4
B     = array(5:8, dim = c(1,2,2))    # 1 x 2 x 2 Array der Zahlen 5,...,8
C     = A^2                           # elementweise Potenzierung
, , 1
     [,1] [,2]
[1,]    1    4                        # 1^2, 2^2
, , 2
     [,1] [,2]
[1,]    9   16                        # 3^2, 4^2
D     = A + B                         # elementweise Addition
, , 1
     [,1] [,2]
[1,]    6    8                        # 1+5, 2+6
, , 2
     [,1] [,2]
[1,]   10   12                        # 3+7, 4+8
```

Vektoren, Matrizen, Arrays, Faktoren

- Vektoren
- Matrizen
- Arrays
- **Faktoren**
- Übungen und Selbstkontrollfragen

Übersicht

- Faktoren sind Vektoren, die nur vorab festgelegte Werte enthalten können.
- Faktoren können zum Speichern kategorialer Daten benutzt werden.
- Formal basieren Faktoren auf integer Vektoren.
- Faktoren haben die Attribute class “factor” und levels.
- Faktoren werden häufig beim Einlesen von Daten automatisch generiert.

Erzeugung

Faktoren können mit factor() aus Vektoren erzeugt werden.

- factor(x = character(), {levels, labels = levels, exclude = NA}, ...)

```
x = factor(c("a", "b", "b", "a"))   # Umwandlung eines character vectors
[1] a b b a                         # Inhalt des factors x
Levels: a b                         # Levels des factors x

y = factor(c(1,2,1,2,1,1))          # Umwandlung eines numerischen vectors
[1] 1 2 1 2 1 1                     # Inhalt des factors y
Levels: 1 2                         # Levels des factors y

v = c("a","b","b","a")              # Vektor zur Konversion in factor
z = factor(v,levels=c("a","c"))     # andere Levels als Werte in Vektor
[1] a    <NA> <NA> a                # fehlende Werte bei "b"
Levels: a c                         # Levels des factors z
```

Indizierung

Faktorindizierung geschieht analog zur Vektorindizierung

Attribute

Durch das **class** Attribut verhalten sich Faktoren nicht wie integer Vektoren

```
f = factor(c("f", "m", "m", "f"))   # Faktordefinition

attributes(f)                       # Attribute
$levels                             # Faktorlevels
[1] "f" "m"                         # f und m
$class                              # class Attribut
[1] "factor"                        # "factor"

typeof(f)                           # elementarer Datentyp
[1] "integer"                       # ... ist integer
```

Vektoren, Matrizen, Arrays, Faktoren

- Vektoren
- Matrizen
- Arrays
- Faktoren
- **Übungen und Selbstkontrollfragen**

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem kommentierten R Skript.
2. Beschreiben Sie in einer Übersicht die R Datenstruktur " Atomic Vector".
3. Beschreiben Sie in einer Übersicht die R Datenstruktur "Matrix".
4. Beschreiben Sie in einer Übersicht die R Datenstruktur "Array".
5. Beschreiben Sie in einer Übersicht die R Datenstruktur "Faktor".
6. Nennen Sie die R Befehle zum Erzeugen eines Vektors, einer Matrix, und eines Arrays.
7. Erzeugen Sie einen Vektor der Dezimalzahlen 0.0, 0.05, 0.10 , 0.15, ..., 0.90, 0.95, 1.0.
8. Wählen Sie mithilfe positiver Indices die Elemente 0.0, 0.1,...,0.9, 1.0 dieses Vektors aus.
9. Wählen Sie mithilfe negativer Indices die Elemente 0.0, 0.1,...,0.9, 1.0 dieses Vektors aus.
10. Wählen Sie die letzten 10 Elemente dieses Vektors aus.
11. Erzeugen Sie eine 4 x 5 Matrix M, deren Elemente dem Zeilenindex des Elements entsprechen.
12. Wählen Sie die 2. und 4. Zeile von M mithilfe positiver Indices aus.
13. Erzeugen Sie eine 4 x 5 Matrix N, deren Elemente dem Spaltenindex des Elements entsprechen.
14. Wählen Sie die 2. und 4. Zeile von N mithilfe negativer Indices aus.
15. Addieren Sie das Doppelte von M zu N.
16. Erzeugen Sie eine 4 x 5 Matrix, deren Elemente den Zeilen- und Spaltenindices entsprechen.

Literatur

Cotton, R. (2013). *Learning R*. O'Reilly, Beijing ; Sebastopol, CA, first edition edition.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(4) Listen, Dataframes, Tibbles

Literaturhinweise

Die Diskussion folgt Cotton (2013, Kapitel 5) und Wickham (2019, Kapitel 3).

Listen, Dataframes, Tibbles

- Listen
- Dataframes
- Tibbles
- Übungen und Selbstkontrollfragen

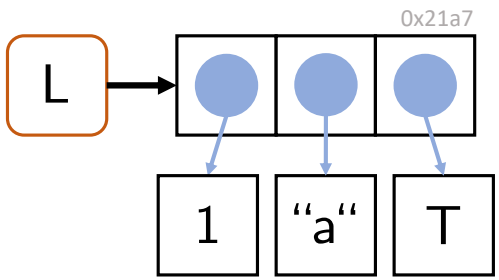
Listen, Dataframes, Tibbles

- **Listen**
- Dataframes
- Tibbles
- Übungen und Selbstkontrollfragen

Übersicht

- Listen sind geordnete Folgen von R Objekten.
- Listen sind rekursiv, können also Objekte verschiedenen Datentyps enthalten.
- Defacto enthalten Listen keine Objekte, sondern Referenzen zu Objekten.

$$L = \text{list}(1,\text{“a“},T)$$



- Listen sind ein wesentlicher Baustein von Dataframes.

Erzeugung

Direkte Konkatenation von Listenelementen mit **list()**

```
L = list(c(1,4,5),                          # Liste mit einem Vektor,
         matrix(1:8, nrow = 2),             # einer Matrix und
         exp)                               # einer Funktion
[[1]]                                       # 1. Listenelement
[1] 1 4 5
[[2]]                                       # 2. Listenelement
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
[[3]]
function (x)  .Primitive("exp")             # 3. Listenelement
```

Listen können Elemente von Listen sein

```
L = list(list(1))                           # Liste mit Element 1 in einer Liste
[[1]]
[[1]][[1]]
[1] 1
```

c() kann zum Verbinden von Listen genutzt werden

```
L = c(list(pi), list("a"))                  # Konkatenation zweier Listen
[[1]]                                       # 1. Listenelement
[1] 3.141593
[[2]]                                       # 2. Listenelement
[1] "a"
```

Charakterisierung

Der Typ von Listen ist **list**

```
L = list(1:2, "a", log)                    # Erzeugung einer Liste
typeof(L)                                  # Typenbestimmung
[1] "list"                                 # Datentyp von Listen ist "list"
```

length() gibt die Anzahl der Toplevel Listenelemente aus

```
L = list(1:2, list("a", pi), exp)          # Liste mit drei Toplevelelementen
length(L)                                  # length() ignoriert Elementinhalte,
[1] 3                                      # length() von L ist also 3
```

Die Dimension, Zeilen-, und Spaltenanzahl von Listen ist **NULL**

```
L = list(1:2, "a", sin)                    # eine Liste
dim(L)                                     # Die Dimension von Listen
NULL                                       # ... ist NULL
nrow(L)                                    # Die Zeilenanzahl von Listen
NULL                                       # ... ist NULL
ncol(L)                                    # Die Spaltenanzahl von Listen
NULL                                       # ... ist NULL
```

Indizierung

Einfache eckige Klammern [ ] indizieren Listenelemente als Listen

```
L  = list(1:3, "a", exp)                    # eine Liste
l1 = L[1]                                   # Indizierung eines Listenelements
[[1]]                                       # das Listenelement l1
[1] 1 2 3                                   # der Inhalt von Listenelement l1
typeof(l1)                                  # Typbestimmung von l1
[1] "list"                                  # L[] gibt eine Liste aus
```

Doppelte eckige Klammern [[ ]] indizieren den Inhalt von Listenelementen

```
L  = list(1:3, "a", exp)                    # eine Liste
i2 = L[[2]]                                 # Indizierung des Listenelementinhalts
[1] "a"                                     # der Inhalt von Listenelement L[2]
typeof(i2)                                  # Typbestimmung von i2
[1] "character"                             # L[[]] gibt den Listenelementinhalt aus
```

Ersetzen von Listenelement(inhalt)en

```
L     = list(1:3, "a", exp)                 # eine Liste
L[1]  = 4:6                                 # keine Typkonversion, Fehlermeldung
L[1]  = list(4:6)                           # Ersetzung des 1. Listenelementes
L[[3]]  = "c"                               # Ersetzung des 3. Listenelementinhaltes
```

Indizierung

Die Prinzipien der Listenindizierung sind analog zur Vektorindizierung

Vektoren positiver Zahlen adressieren entsprechende Elemente

```
L = list(1:3, "a", pi)                    # eine Liste
l = L[c(1,3)]                             # 1. und 3. Listenelement
[[1]]                                     # 1. Listenelement
[1] 1 2 3
[[2]]                                     # 2. Listenelement
[1] 3.141593
```

Vektoren negativer Zahlen adressieren komplementäre Elemente

```
L = list(1:3, "a", pi)                    # eine Liste
l = L[-c(1,3)]                            # 2. Listenelement
[1] "a"
```

Logische Vektoren adressieren Elemente mit TRUE.

```
L = list(1:3, "a", pi)                    # eine Liste
l = L[c(T,T,F)]                           # 1. und 2. Listenelement
[[1]]                                     # 1. Listenelement
[1] 1 2 3
[[2]]                                     # 2. Listenelement
[1] "a"
```

Indizierung

Listenelementen können bei Erzeugung Namen gegeben werden

```
L     = list(peter  = 1:3,                    # eine Liste mit benannten Elementen
             paul   = "a",
              mary  = exp)
$peter                                        # 1. Listenelement
[1] 1 2 3
$paul                                         # 2. Listenelement
[1] "a"
$mary                                         # 3. Listenelement
function (x)  .Primitive("exp")
```

Listenelementen können mit **names()** Namen gegeben werden

```
K       = list(1:2, TRUE)                     # eine unbenannte Liste
names(K) = c("Ebony", "Ivory")                # Namensgebung mit names()
$Ebony                                        # 1.Listenelement
[1] 1 2
$Ivory                                        # 2. Listelement
[1] TRUE
```

Indizierung

Listenelemente und Listenelementinhalte können mit Namen indiziert werden

```
L     = list(peter  = 1:3,                    # eine Liste mit benannten Elementen
             paul   = "a",
              mary  = exp)
l3    = L["mary"]                             # Listenelementindizierung
$mary
function (x)  .Primitive("exp")
i3    = L[["mary"]]                           # Listenelementinhaltsinindizierung
function (x)  .Primitive("exp")
```

Listenelementinhalte können mit dem **$** Operator indiziert werden

```
L     = list(peter  = 1:3,                    # eine Liste mit benannten Elementen
             paul   = "a",
              mary  = exp)
L$peter                                       # Listenelementinhalt
[1] 1 2 3
L$paul                                        # Listenelementinhalt
[1] "a"
L$mary                                        # Listenelementinhalt
function (x)  .Primitive("exp")
```

Arithmetik

Listenarithmetik ist nicht definiert, da Listenelemente unterschiedlichen Typs sein können

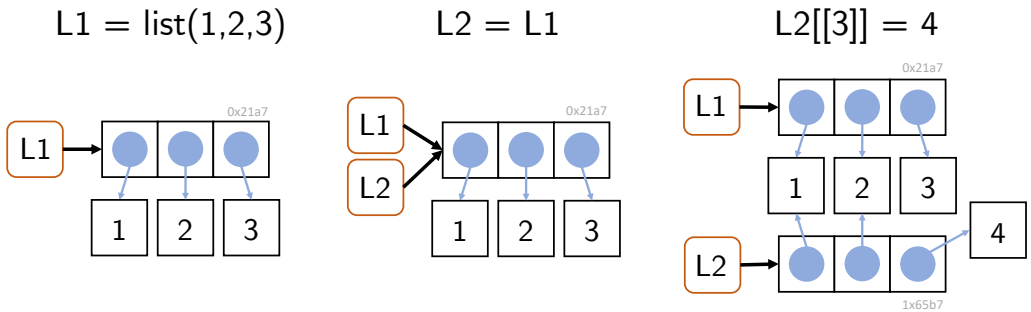
```
L1 = list(1:3, "a")                          # eine Liste
L2 = list(T, exp)                            # eine Liste
L1+L2                                        # Versuch der Listenaddition
Fehler in L1 + L2: nicht-numerisches Argument fuer binaeren Operator
```

Listenelementinhalte können bei Passung jedoch arithmetisch verknüpft werden

```
L1 = list(1:3, pi)                           # eine Liste
L2 = list(4:6, exp)                          # eine Liste
v  = L1[[1]] + L2[[1]]                       # Addition der 1. Listenelementinhalte
[1] 5 7 9                                    # [1+4, 2+5,3+6]
L2[[2]](1)                                   # Anwendung des 2. Listenelementinhalts
[1] 2.718282                                 # exp(1)
```

Copy-on-modify

- Wie bei Vektoren gilt bei Listen das Copy-on-Modify Prinzip.
- "Shallow copy": Listenobjekt wird kopiert, aber nicht die gebundenen Objekte.
- lobstr::ref() erlaubt es, dieses Verhalten zu studieren.




```
L1 = list(1,2,3)                    # Erzeugen einer Liste als Objekt (0x1a3)
L2 = L1                             # L1 und L2 referenzieren das Objekt (0x1a3)
L2[[3]] = 4                         # Copy-on-Modify mit shallow Objekt Kopie
ref(L1,L2)                          # Ausgabe der Referenzen
o [1:0x1a3ae20a0a0] <list>
+-[2:0x1a3a91a1578] <dbl>
+-[3:0x1a3a91a15b0] <dbl>
\-[4:0x1a3a91a15e8] <dbl>
o [5:0x1a3afd9bbb8] <list>
+-[2:0x1a3a91a1578]
+-[3:0x1a3a91a15b0]
\-[6:0x1a3af5bb368] <dbl>
```

Listen, Dataframes, Tibbles

- Listen
- **Dataframes**
- Tibbles
- Übungen und Selbstkontrollfragen

Übersicht

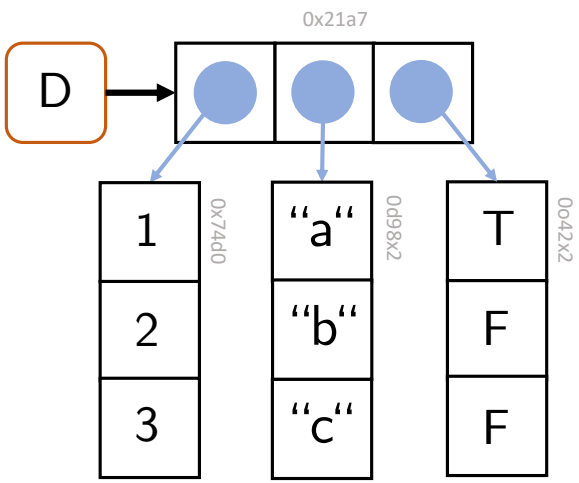
- Dataframes sind die zentrale Datenstruktur in R.
- Dataframes stellt man sich am besten als Tabelle vor.
- Die Zeilen und Spalten der Tabelle haben Namen.

	Control	drug1	drug2L	drug2R	delta1	delta2L	delta2R
1	0.6	1.3	2.5	2.1	0.7	1.9	1.5
2	3.0	1.4	3.8	4.4	-1.6	0.8	1.4
3	4.7	4.5	5.8	4.7	-0.2	1.1	0.0
4	5.5	4.3	5.6	4.8	-1.2	0.1	-0.7
5	6.2	6.1	6.1	6.7	-0.1	-0.1	0.5
6	3.2	6.6	7.6	8.3	3.4	4.4	5.1
7	2.5	6.2	8.0	8.2	3.7	5.5	5.7
8	2.8	3.6	4.4	4.3	0.8	1.6	1.5
9	1.1	1.1	5.7	5.8	0.0	4.6	4.7
10	2.9	4.9	6.3	6.4	2.0	3.4	3.5

Übersicht

- Formal ist ein Dataframe eine Liste, deren Elemente Vektoren gleicher Länge sind.
- Die Listenelemente entsprechen den Spalten einer Tabelle.
- Die Vektorelement gleicher Position entsprechen den Zeilen einer Tabelle.

```
D = data.frame(c(1,2,3),
               c("a", "b", "c"),
               c(T,F,F))
```



Erzeugung

data.frame() erzeugt einen Dataframe

```
D = data.frame( x = letters[1:4],                    # 1. Spalte mit Name x
                y = 1:4,                             # 2. Spalte mit Name y
                z = c(T,T,F,T))                      # 3. Spalte mit Name z
  x y     z                                          # Tabellenform eines Dataframes
1 a 1  TRUE
2 b 2  TRUE
3 c 3 FALSE
4 d 4  TRUE
```

Die Spalten des Dataframes müssen gleiche Länge haben

```
D = data.frame( x = letters[1:4],                    # 1. Spalte mit Name x
                y = 1:4,                             # 2. Spalte mit Name y
                z = c(T,T,F))                        # 3. Spalte mit Name z

Fehler in data.frame(x = letters[1:4], y = 1:4, z = c(T, T, F)) :
Argumente implizieren unterschiedliche Anzahl Zeilen: 4, 3
```

Die Spalten eines Dataframes können offenbar unterschiedlichen Typs sein.

Charakterisierung

Ein Dataframe hat **names()**, **rownames()**,**colnames()**

```
D = data.frame( age    = c(30,35,40,45),        # 1. Spalte
                height = c(178,189,165,171),    # 2. Spalte
                weight = c(67, 76, 81, 92))     # 3. Spalte
names(D)                                        # names gibt die Spaltennamen aus
[1] "age"    "height" "weight"
colnames(D)                                     # colnames entspricht names
[1] "age"    "height" "weight"
rownames(D)
[1] "1" "2" "3" "4"                            # default rownames sind 1,2,...
```

Ein Dataframe **nrow()** Zeilen und **length()** bzw. **ncol** Spalten

```
nrow(D)                                         # Zeilenanzahl
[1] 4                                           # ... ist 4
ncol(D)                                         # Spaltenanzahl
[1] 3                                           # ... ist 3
length(D)                                       # Laenge ist
[1] 3                                           # ... die Spaltenanzahl
```

Charakterisierung

View() öffnet den RStudio Data Viewer.

```
View(D)
```

	age	height	weight
1	30	178	67
2	35	189	76
3	40	165	81
4	45	171	92

str() zeigt in kompakter Form wesentliche Aspekte eines Dataframes an.

```
str(D)
'data.frame': 4 obs. of  3 variables:
 $ age    : num  30 35 40 45
 $ height: num  178 189 165 171
 $ weight: num  67 76 81 92
```

Allgemein zeigt **str()** in kompakter Form wesentliche Aspekte eines R Objektes an.

Attribute

- Dataframes sind Listen mit Attributen für (column) **names** und **row.names**
- Dataframes haben **class "data.frame"**

```
typeof(D)
[1] "list"

attributes(D)
$names
[1] "age"    "height" "weight"

$class
[1] "data.frame"

$row.names
[1] 1 2 3 4
```

Indizierung

Die Prinzipien der Indizierung für Vektoren und Matrizen gelten auch für Dataframes

- Bei einem Index verhalten sich Dataframes wie Listen

```
D = data.frame( x = letters[1:4],   # 1. Spalte mit Name x
                y = 1:4,            # 2. Spalte mit Name y
                z = c(T,T,F,T))     # 3. Spalte mit Name z
class(D)
[1] "data.frame"                    # D ist ein data frame
v = D[1]                            # 1. Listenelement als Dataframe
v
1 a
2 b
3 c
4 d
class(v)                            # v ist
[1] "data.frame"                    # ... ein Dataframe
w = D[[1]]                          # Inhalt des 1. Listenelements
[1] "a" "b" "c" "d"
class(w)                            # w ist
[1] "character"                     # ... ein character vector
v = D$y                             # $ zur Indizierung der y Spalte
[1] 1 2 3 4                         # y Spalte
class(y)                            # y ist ein Vektor
"integer"                           # ... vom Typ "integer" (!)
```

Indizierung

Die Prinzipien der Indizierung für Vektoren und Matrizen gelten auch für Dataframes

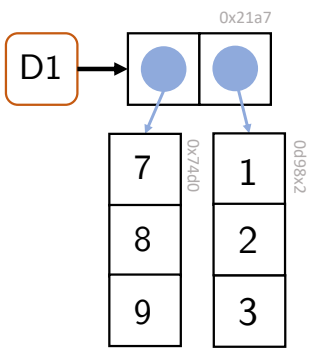
- Bei zwei Indices verhalten sich Dataframes wie Matrizen

```
D = data.frame( x = letters[1:4],    # 1. Spalte mit Name x
                y = 1:4,             # 2. Spalte mit Name y
                z = c(T,T,F,T))      # 3. Spalte mit Name z

D[2:3,-2]                            # 1. Index fuer Zeilen, 2. Index fuer Spalten
  x     z                            # negative Zahlen fuer komplementaere Elemente
2 b  TRUE                            # positive Zahlen fuer entsprechende Elemente
3 c FALSE
D[c(T,F,T,F),]                       # 1. Index fuer Zeilen, 2. Index fuer Spalten
  x y     z                          # leerer Index fuer alle Spalten
1 a 1  TRUE                          # logical TRUE fuer Auswahl
3 c 3 FALSE
 D[,c("x", "z")]                     # 1. Index fuer Zeilen, 2. Index fuer Spalten
  x     z                            # characters fuer Spaltennamen
1 a  TRUE                            # leerer Index fuer alle Zeilen
2 b  TRUE
3 c FALSE
4 d  TRUE
```

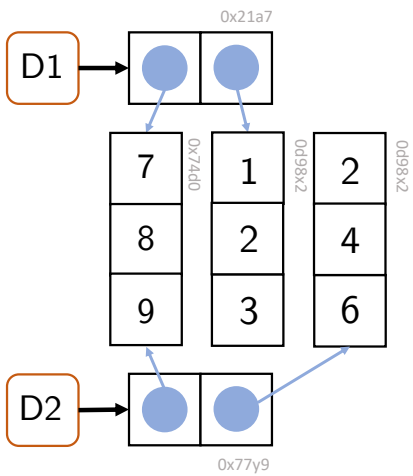
Copy-on-modify

- Die Copy-on-Modify Prinzipien für Listen gelten auch für Dataframes
- Modifikation einer Spalte führt zur Kopie der entsprechenden Spalte
- Modifikation einer Zeile führt zur Kopie des gesamten Dataframes

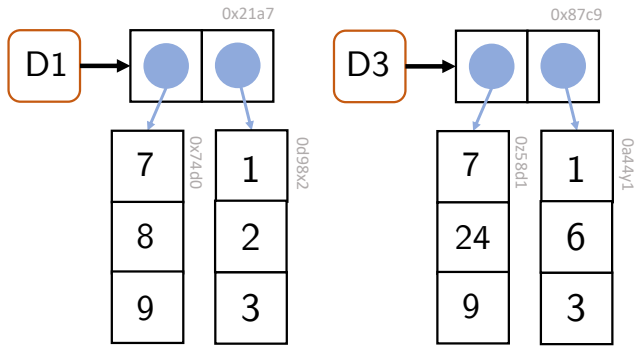
```
D1 = data.frame(
       c(7,8,9),c(1,2,3))
```




```
D2 = D1
D2[,1] = D2[,1] * 2
```




```
D3 = D1
D3[2,] = D3[2,] * 3
```

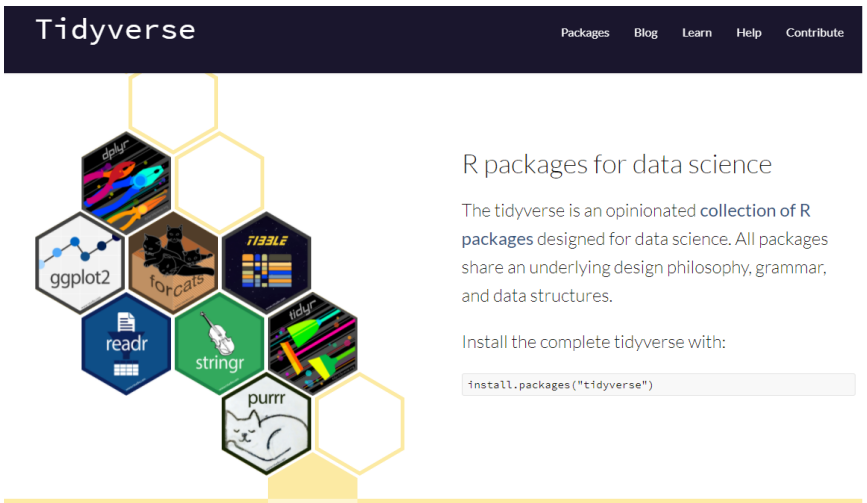


Listen, Dataframes, Tibbles

- Listen
- Dataframes
- **Tibbles**
- Übungen und Selbstkontrollfragen

Übersicht

- Tibbles sind programmiertechnisch optimierte Dataframes.
- Tibbles sind ein Beispiel für die Data Science Hinwendung von R.
- Tibbles sind Teil des tidyverse Pakets.



- Tibbles wurden von Hadley Wickham entwickelt und promotet.

```
install.packages("tidyverse")      # Einmaliger Paket Download und Installation
library(tibble)                    # Sessionspezifisches Laden des tibble Codes
```

Erzeugung und Charaktersierung

- Die Tibble Struktur entspricht Dataframes (Liste mit Vektorreferenzen)
- Im Vergleich zu Dataframes schönere Anzeige (ähnlich Data Viewer)
- Im Vergleich zu Dataframes erweiterter class vector

```
T = tibble(x = c("a", "c", "d"),          # tibble Erzeugung
           y = c(1,2,3),
           z = c(T,F,F))

T                                         # tibble Anzeige
# A tibble: 3 x 3
  x           y z
  <chr> <dbl> <lgl>                       # Typinformation zu Spalten
1 a           1 TRUE                      # chr bleibt chr
2 c           2 FALSE
3 d           3 FALSE

attributes(T)                             # Attribute
$names
[1] "x" "y" "z"
$row.names
[1] 1 2 3
$class
[1] "tbl_df"     "tbl"        "data.frame"          # Erweiterter class vector
```

Schwächen von Dataframes

- data.frames wandelte bis 2018 per default character vectors in factors um
- Seit 2019 ist dies allerdings nicht mehr der Fall

```
D = data.frame(x = 1:3,
               y = c("a", "b", "c"))          # character vector

str(D)
'data.frame': 3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr  "a" "b" "c"                        # ... bleibt character vector
```

- data.frames wandelt per default nicht legale Namen in legale Namen um

```
names(data.frame(`1` = 1))                    # per default Umwandlung von
[1] "X1"                                      # ... nicht legalen Variablennamen

names(data.frame(`1` = 1,
                 check.names = FALSE))        # Dieser default kann jedoch
[1] "1"                                       # ... abgestellt werden.
```

Schwächen von Dataframes

- data.frame recyclet Vektoren bei ganzzahligem Vielfachen der Länge.
- tibble recylcet nur Vektoren der Länge 1.

```
D = data.frame(x = 1:2, y = 3:6)        # length(y) = 2 * length(x)
  x y
1 1 3                                   # Die erste Spalte von D
2 2 4                                   # ist ab
3 1 5                                   # der dritten Zeile ein
4 2 6                                   # reycleter Vektor

T = tibble(x = 1:2, y = 3:6)            # tibble gibt hier eine Fehler aus
Fehler: Tibble columns must have compatible sizes.
* Size 2: Existing data.
* Size 4: Column `y`.
i Only values of size one are recycled.

T = tibble(x = 1:3, y = 3)              # Vektoren mit length 1
# A tibble: 3 x 2                       # ... werden auch von tibble recycled.
      x     y
  <int> <dbl>
1     1     3
2     2     3
3     3     3
```

Schwächen von Dataframes

- Dataframes geben manchmal Dataframes, manchmal Vektoren aus.
- Tibbles sind in dieser Hinsicht konsistenter.

```
D = data.frame(x = 1:3, y = 4:6)                    # Dataframe

str(D["x"])                                         # Listenindizierung mit Namen
'data.frame': 3 obs. of  1 variable:                # ... gibt einen Dataframe aus
 $ x: int  1 2 3

str(D[,"x"])                                        # Matrixindizierung mit Namen
int [1:3] 1 2 3                                     # ... gibt einen integer (!) vector
     aus

T = tibble(x = 1:3, y = 4:6)                        # Tibble

str(T["x"])                                         # Listenindizierung mit Namen
tibble [3 x 1] (S3: tbl_df/tbl/data.frame)          # ... gibt einen tibble aus
 $ x: int [1:3] 1 2

str(T[,"x"])                                        # Matrixindizierung mit Namen
tibble [3 x 1] (S3: tbl_df/tbl/data.frame)          # ... gibt einen tibble aus
 $ x: int [1:3] 1 2
```

Zusätzliches Feature von Tibbles

- Tibble erlaubt die Referenzierung von Variablen bei Erzeugung

```
T = tibble(x = 1, y = 2*x)                    # x wird benannt und benutzt

# A tibble: 1 x 2
      x     y
  <dbl> <dbl>
1     1     2

D = data.frame(x = 1, y = 2*x)                # data.frame erlaubt dies nicht
Fehler in data.frame(x = 1, y = 2 * x) : Objekt 'x' nicht gefunden
```

Fazit

- Tibbles sind weniger flexibel und deshalb genauer als Dataframes.
- Generell sind Tibbles eine gute Idee, gerade für neue Projekte.
- Der Umgang mit Dataframes lehrt auch den Umgang mit Tibbles.
- Dataframes werden wohl nicht so schnell ganz verschwinden.

Listen, Dataframes, Tibbles

- Listen
- Dataframes
- Tibbles
- **Übungen und Selbstkontrollfragen**

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle.
2. Beschreiben Sie in einer Übersicht die R Datenstruktur " List".
3. Beschreiben Sie in einer Übersicht die R Datenstruktur "Dataframe".
4. Beschreiben Sie in einer Übersicht die R Datenstruktur "Tibble".
5. Erzeugen Sie eine Liste, einen Dataframe, und ein Tibble.
6. L sei eine Liste. Was ist der Unterschied zwischen L[1] und L[[1]]?

Literatur

Cotton, R. (2013). *Learning R*. O'Reilly, Beijing ; Sebastopol, CA, first edition edition.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(5) Datenmanagement

Datenmanagement

- FAIR Prinzipien
- Datenformate
- Directory Management
- Datenimport und Datenexport
- Übungen und Selbstkontrollfragen

Datenmanagement

- **FAIR Prinzipien**
- Datenformate
- Directory Management
- Datenimport und Datenexport
- Übungen und Selbstkontrollfragen

Daten

- Zahlenarrays
- Characterarrays
- Software
- Digitale Werkzeuge
- Workflows
- Analysispipelines
- u.v.a.m.

Forschungsdaten

"Grundsätzlich handelt es sich bei Forschungsdaten um elektronisch repräsentierte analoge oder digitale Daten, die im Zuge wissenschaftlicher Vorhaben entstehen oder genutzt werden, z.B. durch Beobachtungen, Experimente, Simulationsrechnungen, Erhebungen, Befragungen, Quellenforschungen, Aufzeichnungen von Audio- und Videosequenzen, Digitalisierung von Objekten, und Auswertungen."

Rat für Informationsinfrastrukturen
Herausforderung Datenqualität (11/2019)
Digitale Kompetenzen – dringend gesucht! (07/2019)
Aktuelle Empfehlungen zu Datenschutz und Forschungsdaten (03/2017)

Metadaten

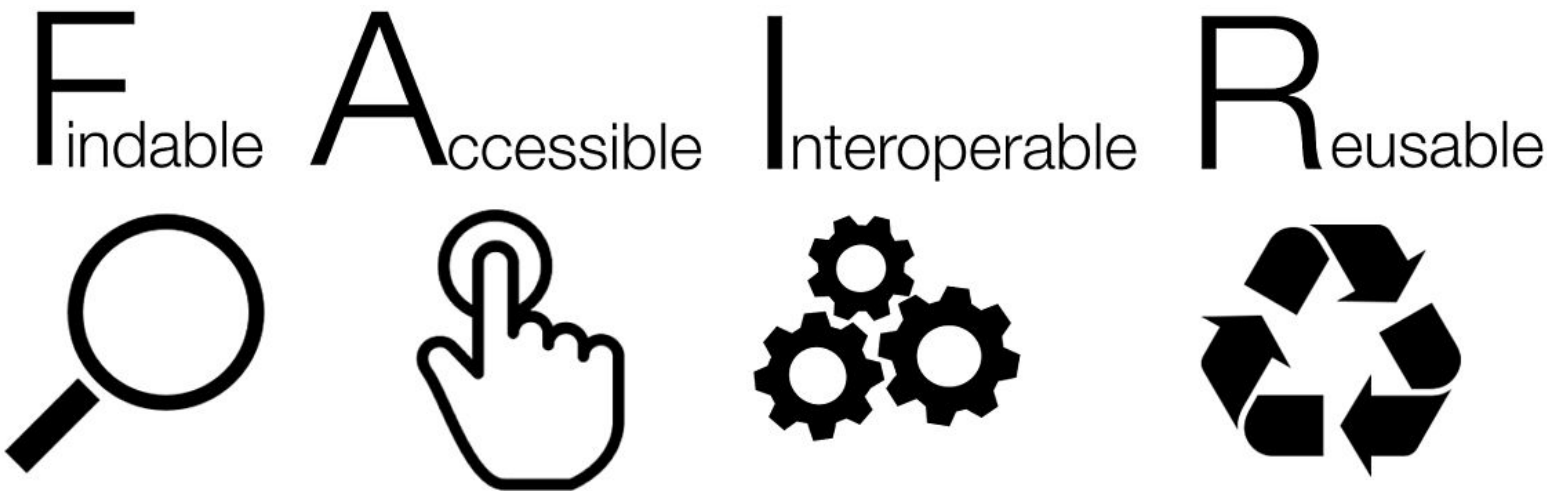
Metadaten repräsentieren Information über Daten

Deskriptive Metadaten dienen dem Auffinden und der Identifikation einer Datenquelle. Beispiele für deskriptive Metadaten sind Titel, Abstrakt, Autor:in, oder Keywords einer wissenschaftlichen Publikationen.

Strukturelle Metadaten sind Metadaten über Datencontainer und repräsentieren den strukturellen Aufbau einer Datenquelle. Beispiele sind die Ordnung der Seiten eines Buches, oder die Schleifenenkodierung dreidimensionaler Datenobjekte.

Administrative Metadaten sind Daten, die das Management einer Datenquelle erleichtern. Beispiele sind die Provenienz, das Dateiformat, die Zugangsrechte, oder weitere technische Informationen zu einer Datenquelle.

Das FAIR Datenideal



für Menschen und Maschinen

Ursprünge und Dokumentation

„Jointly designing a data fairport“ workshop in Leiden 2014

FORCE11

Wilkinson et al. (2016) The FAIR Guiding Principles for scientific data management and stewardship Scientific Data Vol 3, 160018

go-fair.org/FAIR Principles

Findability (Auffindbarkeit)

F1. (Meta)Daten haben einen persistenten global einzigartigen Identifikator.

F2. Daten werden mit Metadaten angereichert.

F3. Metadaten sind zweifelsfrei einem Datensatz zuzuordnen.

F4. (Meta)Daten sind in einer durchsuchbaren Ressource indexiert.

Accessibility (Zugänglichkeit)

A1. (Meta)Daten sind mit standardisierten Protokollen abrufbar.

A1.1. Das genutzte Protokoll ist offen, kostenlos und nutzbar.

A1.2. Das Protokoll ermöglicht Authentifizierung und Rechtevergabe.

A2. Metadaten bleiben zugänglich, auch wenn Daten nicht mehr vorliegen.

Interoperability (Interoperabilität)

I1. (Meta)Daten nutzen eine formale, zugängliche, gemeinsam genutzte und breit anwendbare Sprache zur Wissensrepräsentation.

I2. (Meta)Daten nutzen Vokabularien, die den FAIR-Prinzipien folgen.

I3. (Meta)Daten enthalten qualifizierte Referenzen auf andere (Meta)Daten.

Reusability (Wiederverwendbarkeit)

R1. (Meta)Daten haben eine Vielzahl genauer und relevanter Attribute.

R1.1. (Meta)Daten enthalten eine eindeutige Nutzungslizenz.

R1.2. (Meta)Daten enthalten detaillierte Provenienz-Informationen.

R1.3. (Meta)Daten genügen den Standards der jeweiligen Fachcommunity.

Fazit

- Die FAIR Prinzipien sind ein anzustrebendes Datenmanagementideal.
- Der Umgang mit digitalen Forschungsdaten ist oft noch sehr unstrukturiert.
- Die Universitäten begreifen das digitale Datenmanagement nur sehr langsam.
- Die Digitalisierung bleibt eine Hauptaufgabe nach dem Ende Merkel Ära.
- Nicht alle Wissenschaftler:innen wollen ihre Daten organisieren und teilen.
- Open Science bleibt eine wichtige Initiative verantwortungsvoller Forscher:innen.

Datenmanagement

- FAIR Prinzipien
- **Datenformate**
- Directory Management
- Datenimport und Datenexport
- Übungen und Selbstkontrollfragen

Dateiformate

- Ein Dateiformat definiert Syntax und Semantik von Daten innerhalb einer Datei.
- Dateiformate sind bijektive Abbildungen von Information auf binären Speicher.
- Allgemein unterscheidet man
  - Daten- gegenüber Softwareformaten,
  - textuelle gegenüber binären Dateiformaten, und
  - offene gegenüber propietären (urheberrechtlich geschützten) Dateiformaten.

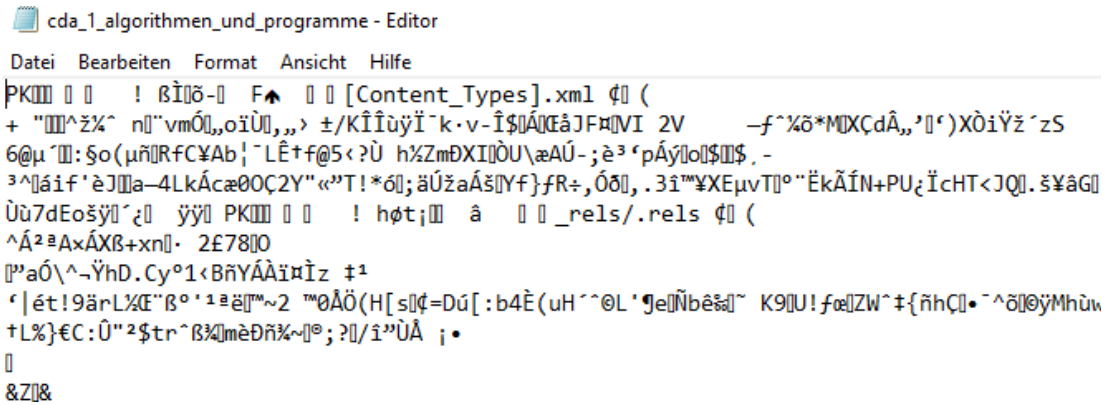
Binäre Dateiformate

- Einlesen, Inspektion, und Manipulation ist nur mit spezieller Software möglich.
- .pdf, .xlsx, .jpg, .mp4 sind binäre Dateiformate.
- Binäre Dateiformate sind oft propietär.
- Binäre Dateiformate wurden früher aufgrund ihrer kleineren Größe bevorzugt eingesetzt.

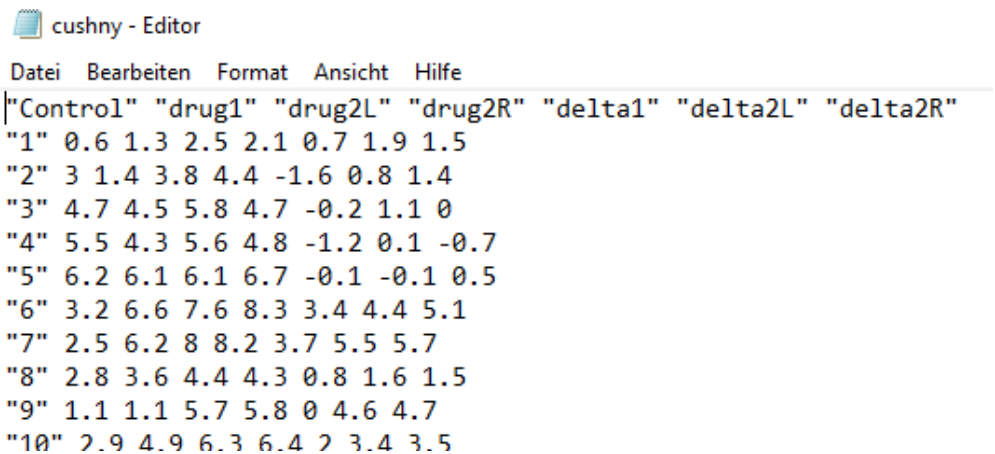
Textuelle Dateiformate

- Einlesen, Inspektion, und Manipulation ist mit einfachen allgemeinen Editoren möglich.
- .txt, .csv., .tsv, .json sind textuelle Dateiformate.
- Textuelle Dateiformate sind generell offene Dateiformate.

Binäres Dateiformat



Textuelles Dateiformat



Textuelle Dateiformate | CSV

- CSV = Comma- (oder auch character)-separated values, Dateiendung .csv
- Zentrales Format zur Speicherung einfach strukturierter Daten
- Repräsentation zeilenweise miteinander verknüpfter Datensätze
  - Trennung von Datenfeldern (Spalten) durch Komma oder Tab (TSV, .tsv)
  - Trennung von Datensätzen (Zeilen) durch Zeilenumbruch
- Erster Datensatz typischerweise Kopfdatensatz (Header) mit Spaltennamendefinition

Beispiel

- Einheit (experimental unit) repräsentiert z.B. eine Versuchsperson

.csv Dateiinhalt

```
Einheit, Variable 1, Variable 2
1, 10.1, 67.5
2, 12.9, 51.2
3, 20.4, 70.8
```

Tabellenrepräsentation

Einheit	Variable 1	Variable 2
1	10.1	67.5
2	12.9	51.2
3	20.4	70.8

Textuelle Dateiformate | CSV

Wide Format: Alle Variablen einer Einheit in einer Zeile

Einheit	Variable 1	Variable 2
1	10.1	67.5
2	12.9	51.2
3	20.4	70.8

Long Format: Variablen einer Einheit über Zeilen verteilt

Einheit	Variable	Messwert
1	Variable 1	10.1
1	Variable 2	67.5
2	Variable 1	12.9
2	Variable 2	51.2
3	Variable 1	20.4
3	Variable 2	70.8

Das Wide Format ist generell übersichtlicher als das Long Format

Textuelle Dateiformate | JSON

Übersicht

- JSON = JavaScript Object Notation
- Textuelles Datenformat zum Speichern strukturierter Daten in Key-Value Form.
- Ähnlichkeit mit R Listen mit benannten Listenelementen.
- Sinnvolles Format für das Speichern von Metadaten.

Elemente von JSON Dateien

- *Objekte* enthalten durch Kommata geteilte Liste von *Eigenschaften* in {}
- *Eigenschaften* bestehen aus Key-Value Paaren
- *Key* ist immer ein String mit Hochkommata “ ”
- *Value* ist ein Objekt, ein Array, ein String, ein Boolean, oder eine Zahl

Textuelle Dateiformate | JSON

Beispiel

```
{
    "Vorname" : "Maxi",
    "Nachname" : "Musterfrau",
    "Matrikelnummer" : 12345,
    "Fachsemester" : 2,
    "Studiengang" : "BSc Psychologie",
    "Module" :
    {
        "Deskriptive Statistik" : {"Abgeschlossen"' : TRUE, "Note" : 1.0 },
        "Inferenzstatistik" : {"Abgeschlossen" : FALSE, "Note": NA }
    }
}
```

Datenmanagement

- FAIR Prinzipien
- Datenformate
- **Directory Management**
- Datenimport und Datenexport
- Übungen und Selbstkontrollfragen

Arbeiten mit Strings

- Die Grundeinheit für Text in R sind atomic vectors vom Typ character.
- Die Elemente von character vectors sind strings, nicht einzelne characters.
- Der Begriff "String" in R ist also nur informeller Natur.
- Strings werden mit Anführungszeichen oder Hochkommata erzeugt

```
s = c("Dies ist ein character vector")        # Anfuehrungszeichen sind ...
[1] "Dies ist ein character vector"           # der String Standard
s = c('Dies ist ein "string"')                # Hochkommata koennen bei ...
[1] "Dies ist ein \"string\""                 # Anfuehrungszeichen im String helfen
```

- **paste()** konvertiert Vektoren in character und fügt sie elementweise zusammen.

```
s = paste(1,2)                                # Konvertierung und Konkatenation
[1] "1 2"                                     # ...einelementiger double vectors
s = paste("Dies ist", "ein String").          # Konkatenation einelementiger
[1] "Dies ist ein String"                     # ...  character strings
```

Arbeiten mit Strings

- **paste()** hat eine Reihe von weiteren Funktionalitäten

```
s = paste(c("Rote", "Gelbe"), "Blume")          # vector recylcing und
[1] "Rote Blume" "Gelbe Blume"                  # elementweise Veknuepfungen
s = paste(c("Rote", "Gelbe"), "Blume",          # Separatorspezifkation
          sep = "-")
[1] "Rote-Blume" "Gelbe-Blume"
s = paste(c("Rote", "Gelbe"), "Blume",          # Zusammenfuegen mit spezifiziertem
          collapse = ", ")                      # Separator
[1] "Rote Blume, Gelbe Blume"
```

- **toString()** ist eine **paste()** Variation für numerische Vektoren

```
s = toString(1:10)                              # Konversion eines double Vektors
[1] "1, 2, 3, 4, 5, 6, 7, 8, 9, 10"             # in formatierten String
s = toString(1:10, width = 10)                  # mit Moeglichkeit der
[1] "1, 2, ..."                                 # Beschraenkung auf width Zeichen.
```

- **cat()** ist eine low-level paste Alternative mit wenig Funktionalität

```
s = cat(c("Gelbe", "Rote"), "Blume")            # Kein Recyling und
Gelbe Rote Blume                                # insbesondere kein
typeof(s)                                       # resultierender
NULL                                            # character vector
```

Working directory

- R hat ein working directory aus dem per default Dateien gelesen werden.
- In RStudio wird das working directory unter Tools → Global Options ... spezifiziert.

getwd() gibt das working directory an.

```
wd = getwd()
[1] "D:/Google Drive/Lehre/2021/1_Inferenzstatistik_21/Code"
```

setwd() ändert das working directory

- Windowspfade haben backward slashes \, R arbeitet mit forward slashes /.
- Manuelle Spezifikation von Windowspfaden benötigt doppelte backward slashes \\.

```
setwd("D:\\Google Drive\\Lehre\\2021")
getwd()
[1] "D:/Google Drive/Lehre/2021"
```

Dateipfadspezifikation

file.path() konstruiert Verzeichnis- und Dateipfade.

```
p = file.path("D:", "Google Drive", "Lehre", "2021")
[1] "D:/Google Drive/Lehre/2021"
```

dirname() gibt das Verzeichnis an, das ein Verzeichnis oder eine Datei enthält.

```
d = getwd()
[1] "D:/Google Drive/Lehre/2021"
p = dirname(getwd())
[1] "D:/Google Drive/Lehre"
```

basename() gibt die unterste Ebene eines Datei- oder Verzeichnispfades an.

```
d = getwd()
[1] "D:/Google Drive/Lehre/2021"
p = basename(getwd())
[1] "2021"
```

Datenmanagement

- FAIR Prinzipien
- Datenformate
- Directory Management
- **Datenimport und Datenexport**
- Übungen und Selbstkontrollfragen

Datenimport mit read.table()

- **read.table()** ist die zentrale Funktion zum Einlesen von CSV Dateien.
- **read.table()** liest eine Datei ein und speichert ihre Inhalte in einem Dataframe.
- Im einfachsten Fall reicht die Angabe des Dateinamens zum Einlesen.
- **read.table()** bietet eine Vielzahl weiterer Spezifikationsmöglichkeiten

```
rootdir = "D:\\...\\...\\"                          # root directory name
codedir = file.path(rootdir, "\\...\\Data_Analysis") # Analysecodeverzeichnis
wdir    = setwd(codedir)                            # working directory
ddir    = file.path(dirname(wdir), "Data")          # data directory
fname   = "cushny.csv"                              # (base) filename
D       = read.table(file.path(ddir, fname))        # Einlesen der Datei
print(D)                                            # Konsolenanzeige
```

	Control	drug1	drug2L	drug2R	delta1	delta2L	delta2R
1	0.6	1.3	2.5	2.1	0.7	1.9	1.5
2	3.0	1.4	3.8	4.4	−1.6	0.8	1.4
3	4.7	4.5	5.8	4.7	−0.2	1.1	0.0
4	5.5	4.3	5.6	4.8	−1.2	0.1	−0.7
5	6.2	6.1	6.1	6.7	−0.1	−0.1	0.5
6	3.2	6.6	7.6	8.3	3.4	4.4	5.1
7	2.5	6.2	8.0	8.2	3.7	5.5	5.7
8	2.8	3.6	4.4	4.3	0.8	1.6	1.5
9	1.1	1.1	5.7	5.8	0.0	4.6	4.7
10	2.9	4.9	6.3	6.4	2.0	3.4	3.5

Datenimport mit read.table()

Einige weitere Spezifikationen bei Anwendung von **read.table()** sind

- **sep** für die Auswahl des Separators, **dec** für die Auswahl des Dezimalpunktes
- **nrow** für die Anzahl der einzulesenden Zeilen
- **skip** für die Anzahl der am Anfang der Datei zu überspringenden Zeilen

```
D = read.table(file.path(ddir, fname), nrow = 4)
  Control drug1 drug2L drug2R delta1 delta2L delta2R
1     0.6   1.3    2.5    2.1    0.7     1.9     1.5
2     3.0   1.4    3.8    4.4   -1.6     0.8     1.4
3     4.7   4.5    5.8    4.7   -0.2     1.1     0.0
4     5.5   4.3    5.6    4.8   -1.2     0.1    -0.7

D = read.table(file.path(ddir, fname), skip = 6)
  V1  V2  V3  V4  V5  V6  V7  V8
1  6 3.2 6.6 7.6 8.3 3.4 4.4 5.1
2  7 2.5 6.2 8.0 8.2 3.7 5.5 5.7
3  8 2.8 3.6 4.4 4.3 0.8 1.6 1.5
4  9 1.1 1.1 5.7 5.8 0.0 4.6 4.7
5 10 2.9 4.9 6.3 6.4 2.0 3.4 3.5
```

Import interner R Datensätze

R und R packages beinhalten eine Vielzahl von Beispieldatensätzen

- Die Core R Datensätze werden aus der R Konsole mit **data()** angezeigt.
- Die Datensätze in Paket P werden mit **data**(package = "P") angezeigt.

```
install.packages("psychTools")     # Installation des Pakets psychTools
data(package = "psychTools")       # Anzeige der psychTools Datensaetze
```

```
Data sets in package 'psychTools':

Damian                          Project Talent data set from Marion Spengler and Rodica Damian
Pollack                         Pollack et al (2012) correlation matrix for mediation example
Schutz                          The Schutz correlation matrix example from Shapiro and ten Berge
Spengler (Damian)               Project Talent data set from Marion Spengler and Rodica Damian
Spengler.stat (Damian)          Project Talent data set from Marion Spengler and Rodica Damian
USAF                            17 anthropometric measures from the USAF showing a general factor
ability                         16 ability items scored as correct or incorrect.
ability.keys (ability)          16 ability items scored as correct or incorrect.
affect                          Two data sets of affect and arousal scores as a function of personality and movie
                                conditions
all.income (income)             US family income from US census 2008
bfi                             25 Personality items representing 5 factors
```

...

- Alle Datensätze werden mit **data**(package = .packages(TRUE)) angezeigt.
- Nach Installation und Laden eines Pakets werden Datensätze mit **data()** geladen.

```
library(psychTools)                # Laden des Paktes psychTools
data(cushny)                       # Laden des cushny Datensatzes aus psychTools
```

Beispiele für weitere Möglichkeiten des Datenimports

CSV und Text Dateien

- **read.csv()**, **read.csv2()**, **read.delim()**, **read.delim2()** als **read.table()** Varianten.
- **readlines** für low-level Textdateiimport.
- **fromJSON()** aus dem Paket **rjson** für .json Dateien.

Binäre Dateien

- **read.xlsx()** und **read.xlsx2()** aus dem Paket **xlsx** für Excel .xlsx Dateien.
- **read.spss()** aus dem Paket **foreign** für SPSS .sav Dateien.
- **readMat** aus dem Paket **R.matlab** für Matlab .mat Dateien.

Webdaten und Datenbanken

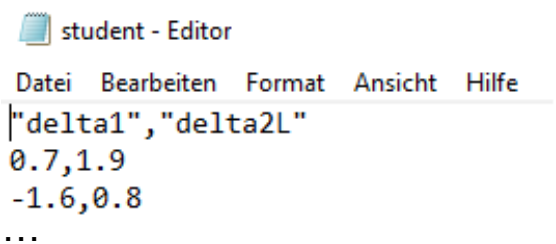
- Twitterdaten können mithilfe der Pakete **rtweet** oder **twitteR** eingelesen werden.
- SQL Datenbanken können mithilfe der Pakete **DBI** und **RSQLite** abgefragt werden.

Datenexport mit write.table()

- **write.table()** ist die zentrale Funktion zum Speichern von Daten in CSV Dateien.
- **write.table()** erzeugt eine Datei und schreibt Daten eines Dataframes hinein.
- Der Dateiname wird mit dem Argument **file** angegeben, der Werteseparator mit **sep**
- Das Argument **row.names** = FALSE unterdrückt das Schreiben von Zeilennahmen

```
fname       = "cushny.csv"                          # Dateiname (input)
rname       = "student.csv"                         # Dateiname (output)
D           = read.table(file.path(ddir, fname))    # Dateneinlesen
D           = D[,5:6]                               # Reduktion des Dataframes
R           = write.table(                          # .csv Schreibfunktion
                D,                                  # Zu speichernder Dataframe
                file = file.path(rdir, rname),      # Dateinname
                sep = ",",                          # Werteseparator fuer .csv
                row.names = F)                      # keine Zeilennamen
```

- Ergebnisdatei student.csv



Datenmanagement

- FAIR Prinzipien
- Datenformate
- Directory Management
- Datenimport und Datenexport
- **Übungen und Selbstkontrollfragen**

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem R Skript.
2. Erläutern Sie den Begriff "Forschungsdaten".
3. Erläutern Sie den Begriff "Metadaten".
4. Erläutern Sie das FAIR Datenideal.
5. Diskutieren Sie Unterschiede und Gemeinsamkeiten von binären und textuellen Dateien.
6. Nennen und erläutern Sie zwei textuelle Dateiformate.
7. Erläutern Sie den Unterschied zwischen dem Wide und Long Format von Tabellen.
8. Erläutern Sie den Begriff des "Working Directories" in R.
9. Nennen Sie eine R Funktion zum Einlesen von .csv Dateien.
10. Nennen Sie eine R Funktion zum Schreiben von .csv Dateien.

Literatur

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(6) Deskriptive Statistiken

Literaturhinweise

Fahrmeir et al. (2016, Kapitel 2) und Henze (2018, Kapitel 5) geben einen Überblick.

Deskriptive Statistiken

- Verteilungsdarstellung
- Maße der zentralen Tendenz
- Maße der Datenvariabilität
- Bivariate Deskriptivstatistik
- Übungen und Selbstkontrollfragen

Deskriptive Statistiken

- **Verteilungsdarstellung**
- Maße der zentralen Tendenz
- Maße der Datenvariabilität
- Bivariate Deskriptivstatistik
- Übungen und Selbstkontrollfragen

Der Affect Beispieldatensatz

- Daten zweier Studien ("maps" und "flat") des Personality, Motivation, and Cognition Laboratory mit $n = 330$ Versuchspersonen, Teil des psychTools R Pakets
- Fragebogenbasierte Messung von Affektvariablen vor und nach Anschauen eines Filmclips
- Filmclipbedingungen
  (1) Frontline, Dokumentation über die Befreiung des Konzentrationslagers Bergen-Belsen
  (2) Halloween, ein Horrorfilm
  (3) National Geographic, eine Naturdokumentation über die Serengeti
  (4) Parenthood, eine Komödie
- Messung von Tense Arousal (TA), Energetic Arousal (EA), Positive Affect (PA) und Negative Affect (NA) vor (1) und nach (2) Anschauen des Filmclips
- Erhebung von Daten mit fünf Skalen des Eysenck Persönlichkeitsinventar sowie State- und Trait-Anxiety Items. In der "maps" Studie außerdem Erhebung des BDI
- Für Details, siehe Rafaeli and Revelle (2006)

Der Affect Beispieldatensatz

```
install.package("psychTools")          # einmalige Installation des psychTools Pakets
library(psychTools)                    # Laden des psychTools Pakets
?affect                                # Erlaeuterung des affect Datensatzes
data(affect)                           # Laden des affect Dataframes
View(affect)                           # Inspektion des affect Dataframes
```

	Study	Film	ext	neur	imp	soc	lie	traitanx	state1	EA1	TA1	PA1	NA1	EA2	TA2	PA2	NA2	state2	MEQ	BDI
1	maps	3	18.0	9.0	7.0	10.0	3.0	24.0	22.0	24.0	14.0	26.0	2.0	6.0	5.0	7.0	4.0	NA	NA	0.04761905
2	maps	3	16.0	12.0	5.0	8.0	1.0	41.0	40.0	9.0	13.0	10.0	4.0	4.0	14.0	5.0	5.0	NA	NA	0.33333333
3	maps	3	6.0	5.0	3.0	1.0	2.0	37.0	44.0	1.0	14.0	4.0	2.0	2.0	15.0	3.0	1.0	NA	NA	0.19047619
4	maps	3	12.0	15.0	4.0	6.0	3.0	54.0	40.0	5.0	15.0	1.0	0.0	4.0	15.0	0.0	2.0	NA	NA	0.38461538
5	maps	3	14.0	2.0	5.0	6.0	3.0	39.0	67.0	12.0	20.0	7.0	13.0	14.0	15.0	16.0	13.0	NA	NA	0.38095238
6	maps	1	6.0	15.0	2.0	4.0	5.0	51.0	38.0	9.0	14.0	5.0	1.0	7.0	12.0	2.0	2.0	NA	NA	0.23809524
7	maps	1	15.0	12.0	4.0	9.0	3.0	40.0	32.0	1.0	5.0	7.0	0.0	13.0	14.0	8.0	8.0	NA	NA	0.30769231
8	maps	2	18.0	10.0	7.0	9.0	2.0	32.0	41.0	17.0	11.0	10.0	1.0	19.0	15.0	16.0	0.0	NA	NA	0.00000000
9	maps	2	15.0	1.0	3.0	11.0	3.0	22.0	26.0	19.0	5.0	14.0	0.0	19.0	6.0	14.0	0.0	NA	NA	0.00000000
10	maps	2	8.0	10.0	2.0	5.0	2.0	35.0	31.0	15.0	8.0	7.0	0.0	28.0	19.0	11.0	2.0	NA	NA	0.33333333
11	maps	1	13.0	9.0	3.0	9.0	3.0	43.0	39.0	14.0	13.0	10.0	2.0	9.0	21.0	3.0	7.0	NA	NA	0.38095238
12	maps	4	14.0	1.0	3.0	12.0	6.0	33.0	25.0	24.0	15.0	23.0	2.0	27.0	11.0	29.0	0.0	NA	NA	0.14285714
13	maps	4	15.0	2.0	4.0	10.0	5.0	23.0	32.0	7.0	14.0	1.0	0.0	11.0	17.0	4.0	0.0	NA	NA	0.00000000
14	maps	4	19.0	3.0	7.0	11.0	0.0	23.0	23.0	21.0	13.0	20.0	1.0	23.0	18.0	21.0	2.0	NA	NA	0.00000000
15	maps	1	15.0	7.0	4.0	10.0	2.0	27.0	28.0	22.0	15.0	16.0	1.0	16.0	18.0	12.0	4.0	NA	NA	0.04761905
16	maps	1	11.0	13.0	6.0	5.0	7.0	45.0	28.0	2.0	0.0	5.0	0.0	23.0	26.0	21.0	9.0	NA	NA	0.19047619
17	maps	1	16.0	18.0	5.0	10.0	0.0	58.0	56.0	3.0	11.0	3.0	7.0	8.0	17.0	4.0	18.0	NA	NA	0.66666667
18	maps	1	17.0	11.0	6.0	11.0	4.0	39.0	44.0	19.0	18.0	23.0	1.0	21.0	20.0	21.0	6.0	NA	NA	0.33333333
19	maps	1	7.0	10.0	2.0	4.0	1.0	43.0	56.0	14.0	21.0	17.0	8.0	18.0	22.0	12.0	12.0	NA	NA	0.42857143
20	maps	1	13.0	12.0	4.0	8.0	4.0	38.0	35.0	9.0	6.0	1.0	0.0	11.0	10.0	8.0	1.0	NA	NA	0.00000000
21	maps	2	14.0	3.0	5.0	7.0	3.0	35.0	28.0	18.0	8.0	12.0	0.0	16.0	17.0	7.0	2.0	NA	NA	0.04761905
22	maps	2	11.0	10.0	3.0	7.0	1.0	37.0	47.0	11.0	12.0	5.0	1.0	16.0	22.0	12.0	4.0	NA	NA	0.52380952
23	maps	2	20.0	10.0	7.0	10.0	2.0	43.0	43.0	5.0	10.0	2.0	1.0	10.0	25.0	2.0	6.0	NA	NA	0.04761905

...

Der Affect Beispieldatensatz

"Tense Arousal" und " Energetic Arousal"

- *Activation-Deactivation Adjective Check List* nach Thayer (1986).
- Circa 25 Adjektive mit Vierpunkteskala
  - Wie gut beschreibt folgendes [Adjektiv] Ihre momentane Gefühlslage?
  - "I (1) do not feel, (2) cannot decide, (3) feel slightly, (4) definitely feel [Adjektiv]"
- Hohe TA Werte entsprechen hohen Selbsteinschätzungen bei Adjektiven wie
  - tense, clutched-up, fearful, jittery, intense
- Niedrige TA Werte entsprechen hohen Selbsteinschätzungen bei Adjektiven wie
  - still, at-rest, calm, quiet, placid
- Hohe EA Werte entsprechen hohen Selbsteinschätzungen bei Adjektiven wie
  - energetic, lively, active, vigorous, full-of-pep, wakeful, wide-awake
- Niedrige EA Werte entsprechen hohen Selbsteinschätzungen bei Adjektiven wie
  - sleepy, drowsy, tired

Häufigkeitsverteilungen

Definition (Absolute und relative Häufigkeitsverteilungen)

$x = (x_1, ..., x_n)$ sei ein *Datensatz* ("Urliste") und $A := \{a_1, ..., a_k\}$ mit $k \leq n$ seien die im Datensatz vorkommenden verschiedenen Zahlenwerte ("Merkmalsausprägungen"). Dann heißt die Funktion

$$h : A \to \mathbb{N}, a \mapsto h(a) := \text{Anzahl der } x_i \text{ aus } x \text{ mit } x_i = a \tag{1}$$

die *absolute Häufigkeitsverteilung* der Zahlwerte von x und die Funktion

$$r : A \to [0, 1], a \mapsto r(a) := \frac{h(a)}{n} \tag{2}$$

die *relative Häufigkeitsverteilung* der Zahlwerte von x.

Häufigkeitsverteilungen

table() zum Erzeugen einer absoluten Häufigkeitsverteilung

Division durch n zum Berechnen der relativen Häufigkeitsverteilung

```
x        = affect$TA1                    # double vector der TA1 Werte
n        = length(x)                     # Anzahl der Datenwerte (330)
H        = as.data.frame(table(x))       # absolute Haeufigkeitsverteilung (dataframe)
names(H)= c("a", "h")                    # Spaltenbenennung
H$r      = H$h/n                         # relative Haeufigkeitsverteilung
print(H, digits = 1)                     #  Ausgabe
      a  h     r
1     0  1 0.003
2   2.5  1 0.003
3     3  2 0.006
4     4  2 0.006
5     5  8 0.024
6     6  7 0.021
7     7  9 0.027
8     8 19 0.058
9     9 13 0.039
10   10 40 0.121
11   11 28 0.085
12   12 27 0.082
```

Häufigkeitsverteilungen

barplot() zur Visualisierung der absoluten Häufigkeitsverteilung

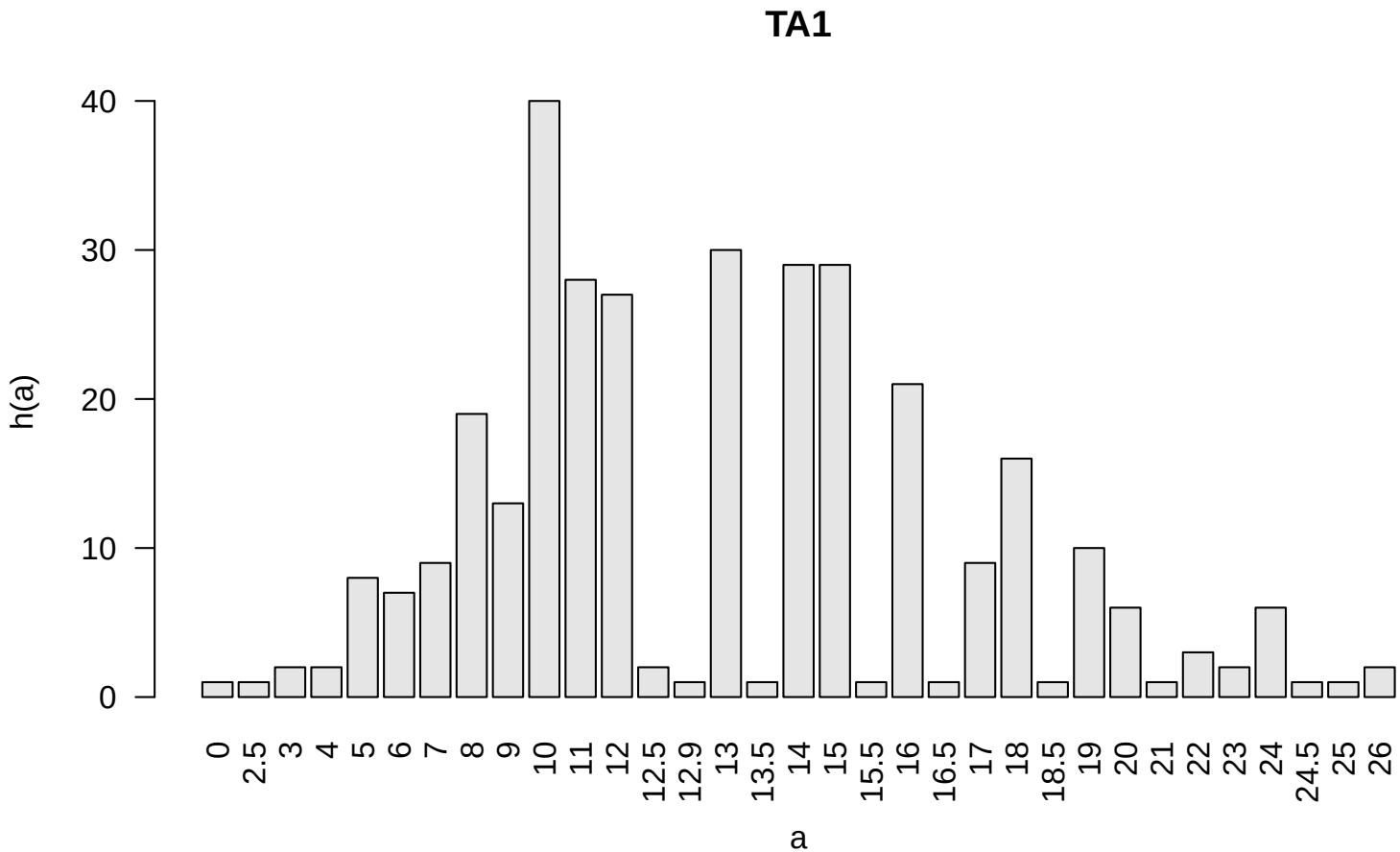
```
h          = H$h                    # h(a) Werte
names(h)   = H$a                    # barplot braucht a Werte als names
dev.new()                           # Abbildungsinitialisierung
barplot(                            # Balkendiagramm
h,                                  # absolute Haeufigkeiten
col        = "gray90",              # Balkenfarbe
xlab       = "a",                   # x Achsenbeschriftung
ylab       = "h(a)",                # y Achsenbeschriftung
las        = 2,                     # Anzeigen aller x Werte
main       = "TA1"                  # Titel
)
```

dev.copy2pdf() zum Speichern der Abbildung als .pdf Datei

```
dev.copy2pdf(                       # PDF Kopiefunktion
file       = "...  .pdf",           # Dateiname
width      = 8,                     # Breite (inch)
height     = 5                      # Hoehe (inch)
)
```

Häufigkeitsverteilungen

barplot() zur Visualisierung der relativen Häufigkeitsverteilung



Häufigkeitsverteilungen

barplot() zur Visualisierung der relativen Häufigkeitsverteilung

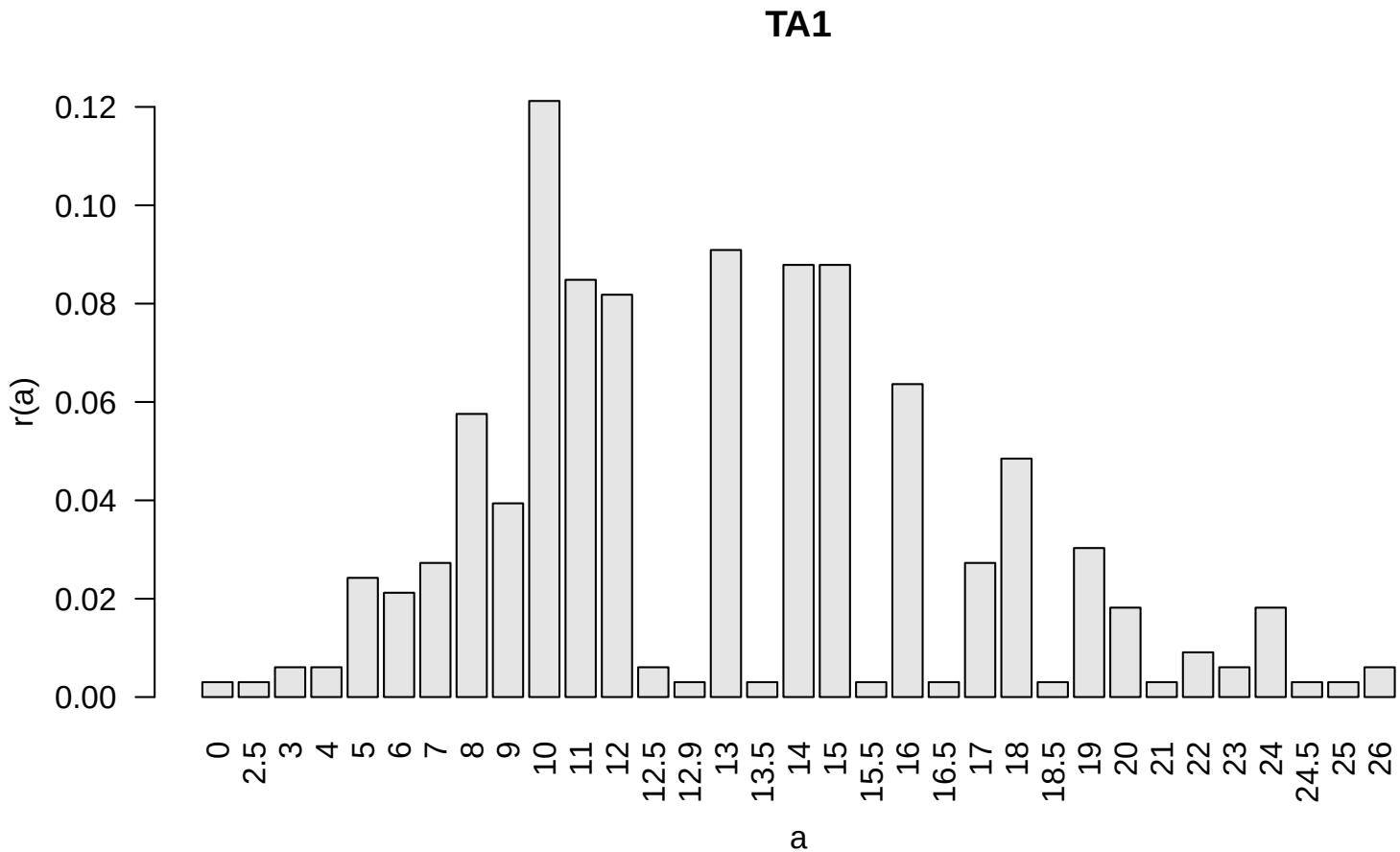
```
r          = H$r                    # r(a) Werte
names(h)   = H$r                    # barplot braucht a Werte als names
dev.new()                           # Abbildungsinitialisierung
barplot(                            # Balkendiagramm
h,                                  # relative Haeufigkeiten
col        = "gray90",              # Balkenfarbe
xlab       = "a",                   # x Achsenbeschriftung
ylab       = "r(a)",                # y Achsenbeschriftung
las        = 2,                     # Anzeigen aller x Werte
main       = "TA1"                  # Titel
)
```

dev.copy2pdf() zum Speichern der Abbildung als .pdf Datei

```
dev.copy2pdf(                       # PDF Kopiefunktion
file       = "...  .pdf",           # Dateiname
width      = 8,                     # Breite (inch)
height     = 5                      # Hoehe (inch)
)
```

Häufigkeitsverteilung

barplot() zur Visualisierung der relativen Häufigkeitsverteilung



Histogramme

Definition (Histogramm)

Ein *Histogramm* ist ein Diagramm, in dem zu einem Datensatz $x = (x_1, ..., x_n)$ mit verschiedenen Zahlwerten $A := \{a_1, ..., a_m\}, m \leq n$ über benachbarten Intervallen $[b_{j-1}, b_j[$, welche *Klassen* oder *Bins* genannt werden, für $j = 1, ..., k$ Rechtecke mit

Breite	$d_j = b_j - b_{j-1}$
Höhe	$h(a)$ oder $r(a)$ mit $a \in [b_{j-1}, b_j[$

abgebildet sind, wobei $b_0 := \min A$ und $b_k := \max A$ angenommen werden soll.

Bemerkungen

- Das Aussehen eines Histogramms ist stark von der Anzahl k der Klassen abhängig.
- Mit der Aufrundungsfunktion $\lceil \cdot \rceil$ sind konventionelle Werte für k

$k := \lceil (b_k - b_0)h \rceil$	h ist die gewünschte Klassenbreite
$k := \lceil \sqrt{n} \rceil$	Excelstandard
$k := \lceil \log_2 n + 1 \rceil$	Implizite Normalverteilungsannahme (Sturges, 1926)
$h := 3.49S/\sqrt[3]{n}$	Min. MSE Dichteschätzung bei Normalverteilung (Scott, 1979)

Histogramme

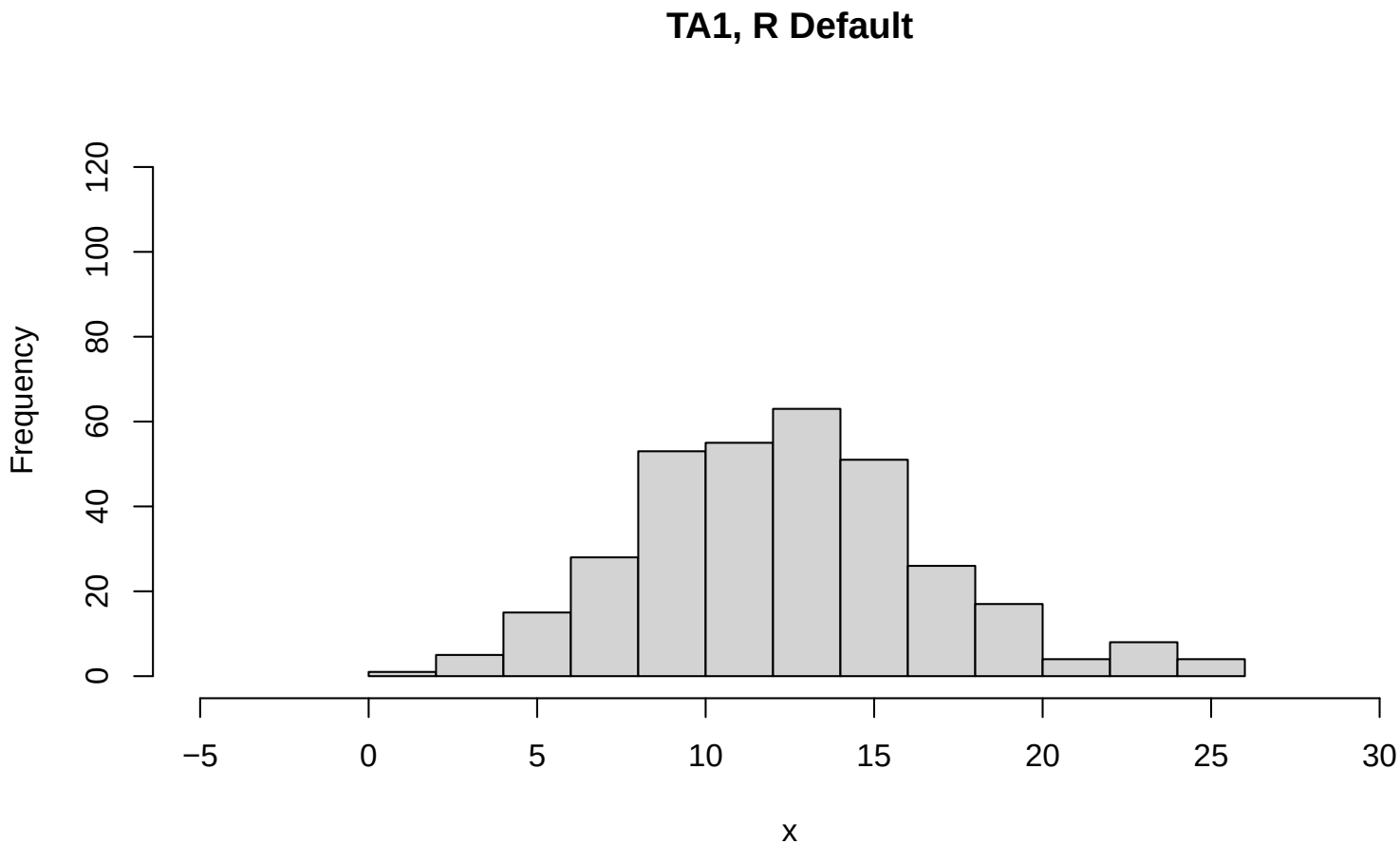
Die Funktion **hist()** berechnet und visualisiert Histogramme

- Die Klassen $[b_{j-1}, b_j[, j = 1, ..., k$ werden als Argument *breaks* festgelegt
- *breaks* ist der atomic vector c$(b_0, b_1, ..., b_k)$ mit Länge $k + 1$
- Per default benutzt **hist()** eine Modifikation der Sturges Empfehlung $k = \lceil \log_2 n + 1 \rceil$
- **hist()** bietet eine Vielzahl weiterer Spezifikationsmöglichkeiten

```
# Default Histogramm
x       = affect$TA1                    # Datensatz
x_min   = -5                            # x Achsengrenze (unten)
x_max   = 30                            # x Achsengrenze (oben)
y_min   = 0                             # y Achsengrenze (oben)
y_max   = 130                           # y Achsengrenze (unten)
hist(                                   # Histogramm
x,                                      # Datensatz
xlim    = c(x_min, x_max),              # x Achsengrenzen
ylim    = c(y_min, y_max),              # y Achsengrenzen
main    =  "TA1, R Default"             # Titel
)
```

Histogramme

Default Histogramm

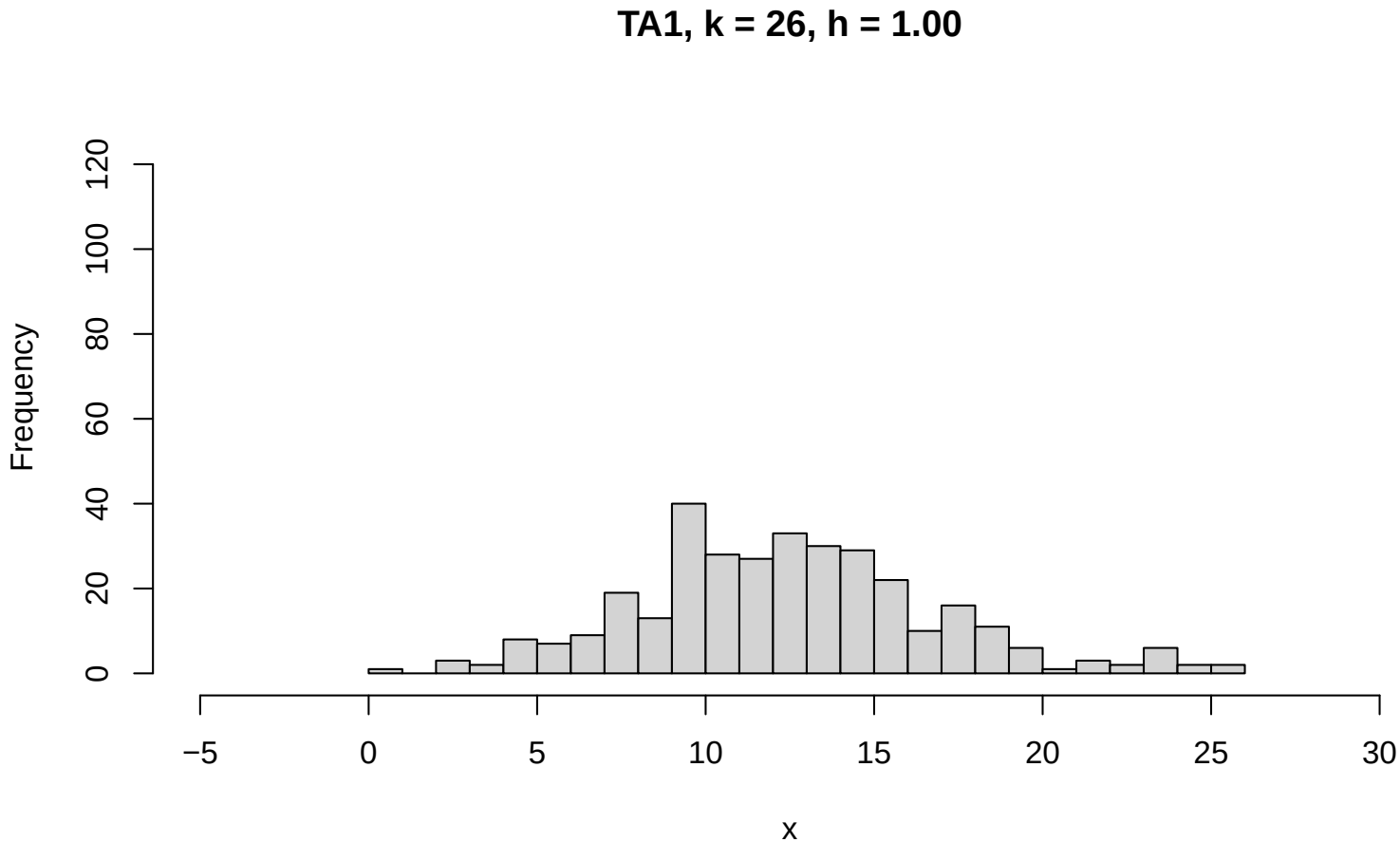


Histogramme

```
# Histogramm mit gewuenschter Klassenbreite
h   = 1                                  # gewuenschte Klassenbreite
b_0 = min(x)                             # b_0
b_k = max(x)                             # b_k
k   = ceiling((b_k - b_0)/h)             # Anzahl der Klassen
b   = seq(b_0, b_k, by = h)              # Klassen [b_{j-1}, b_j[

# Excelstandard
n   = length(x)                          # Anzahl Datenwerte
k   = ceiling(sqrt(n))                   # Anzahl der Klassen
b   = seq(b_0, b_k, len = k)             # Klassen [b_{j-1}, b_j[
h   = b[2] - b[1]                        # Klassenbreite

# Sturges
n   = length(x)                          # Anzahl Datenwerte
k   = ceiling(log2(n)+1)                 # Anzahl der Klassen
b   = seq(b_0, b_k, len = k)             # Klassen [b_{j-1}, b_j[
h   = b[2] - b[1]                        # Klassenbreite

# Scott
n   = length(x)                          # Anzahl Datenwerte
S   = sd(x)                              # Stichprobenstandardabweichung
h   = ceiling(3.49*S/(n^(1/3)))          # Klassenbreite
k   = ceiling((b_k - b_0)/h)             # Anzahl der Klassen
b   = seq(b_0, b_k, len = k)             # Klassen [b_{j-1}, b_j[
```

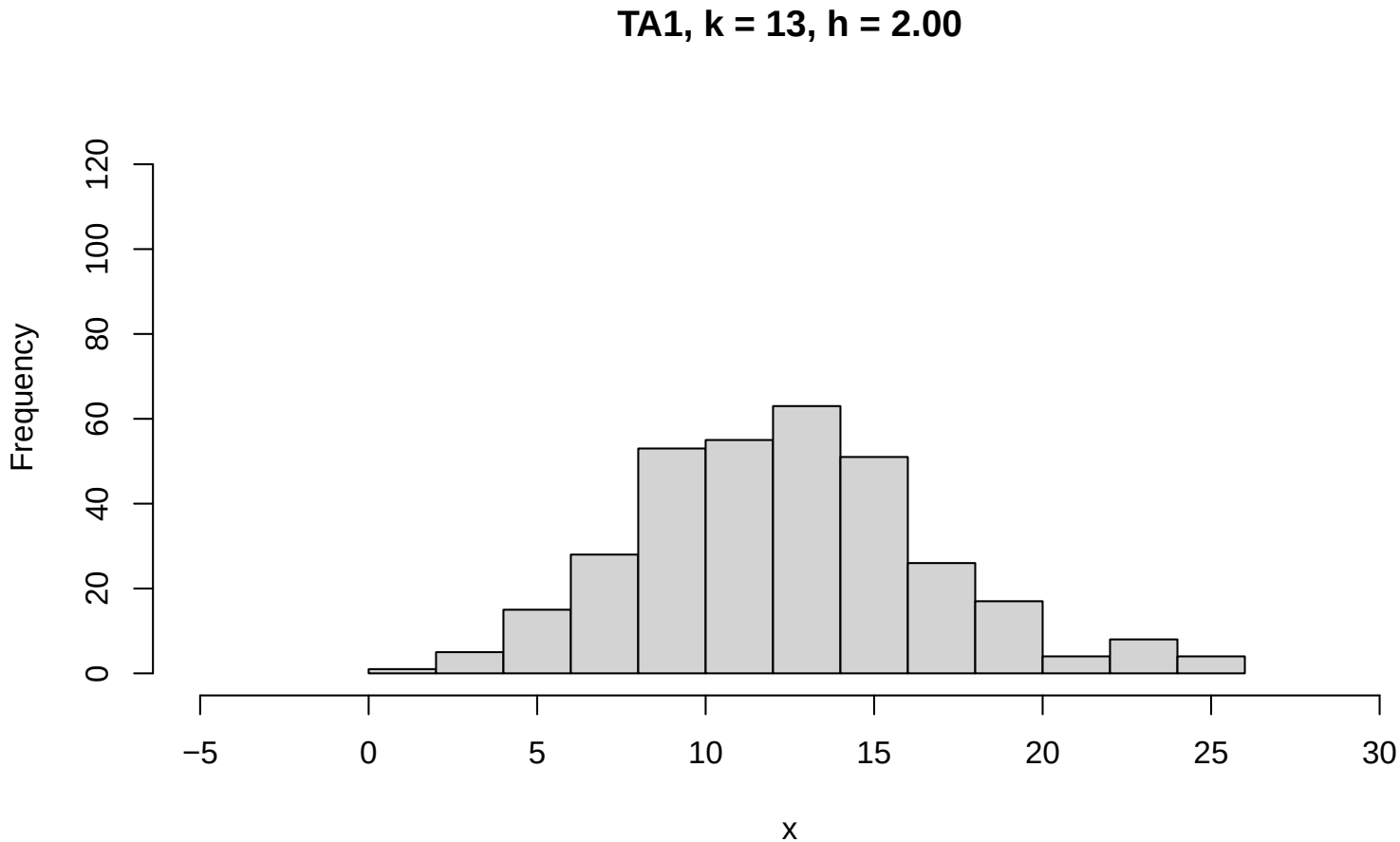
Histogramme

Gewünschte Klassenbreite $h := 1$



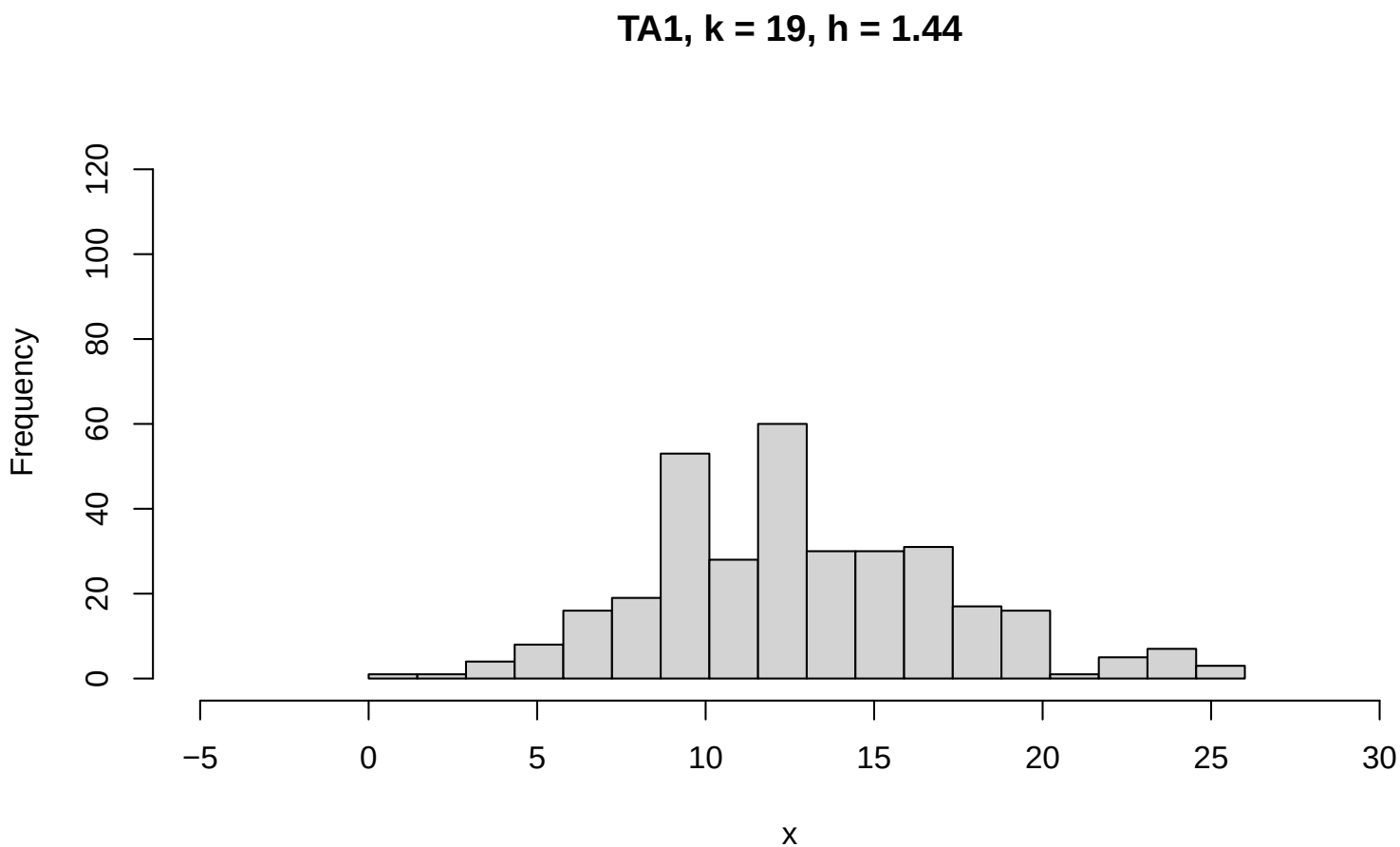
Histogramme

Gewünschte Klassenbreite $h := 2$



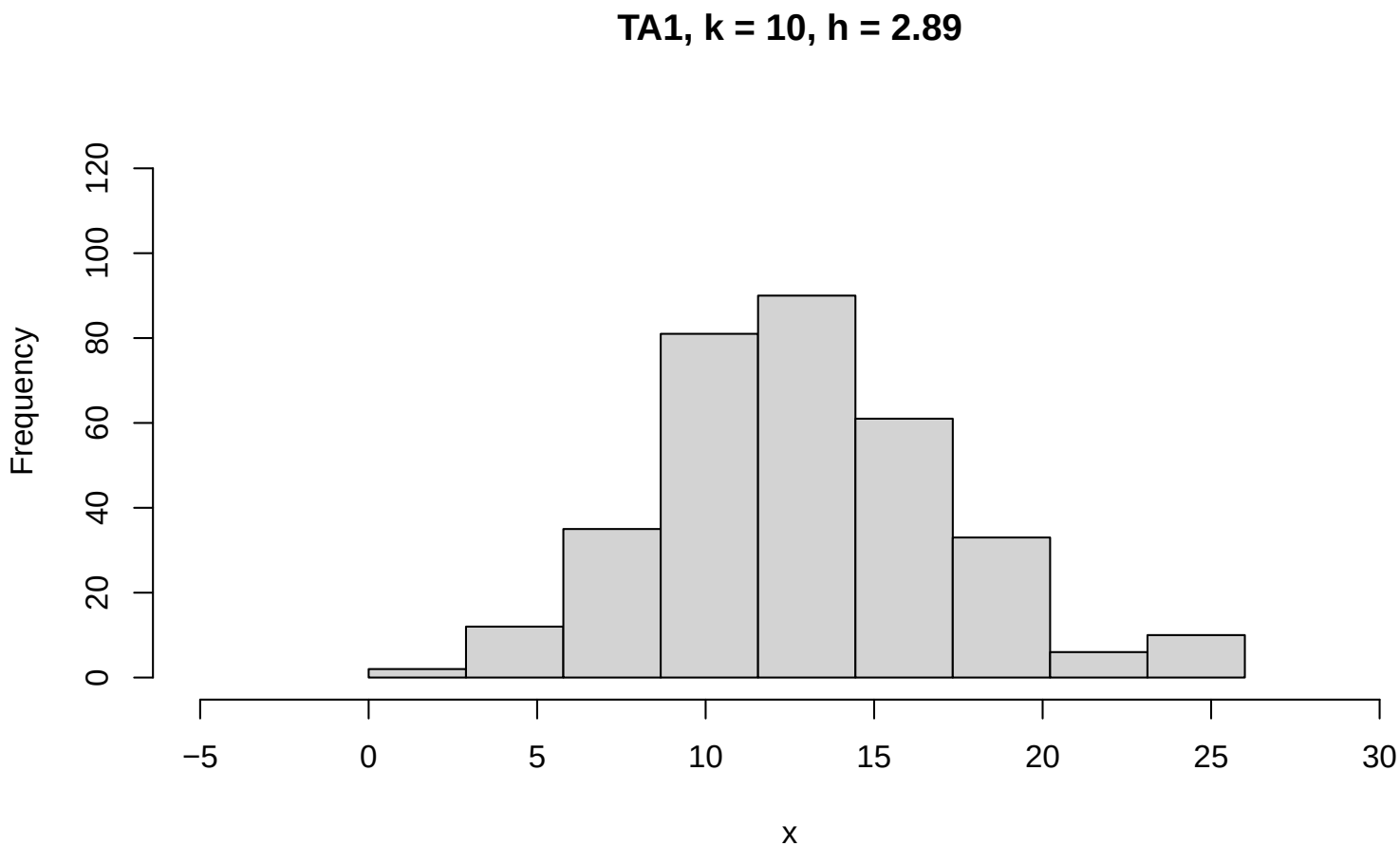
Histogramme

Excelstandard $k := \lceil \sqrt{n} \rceil$



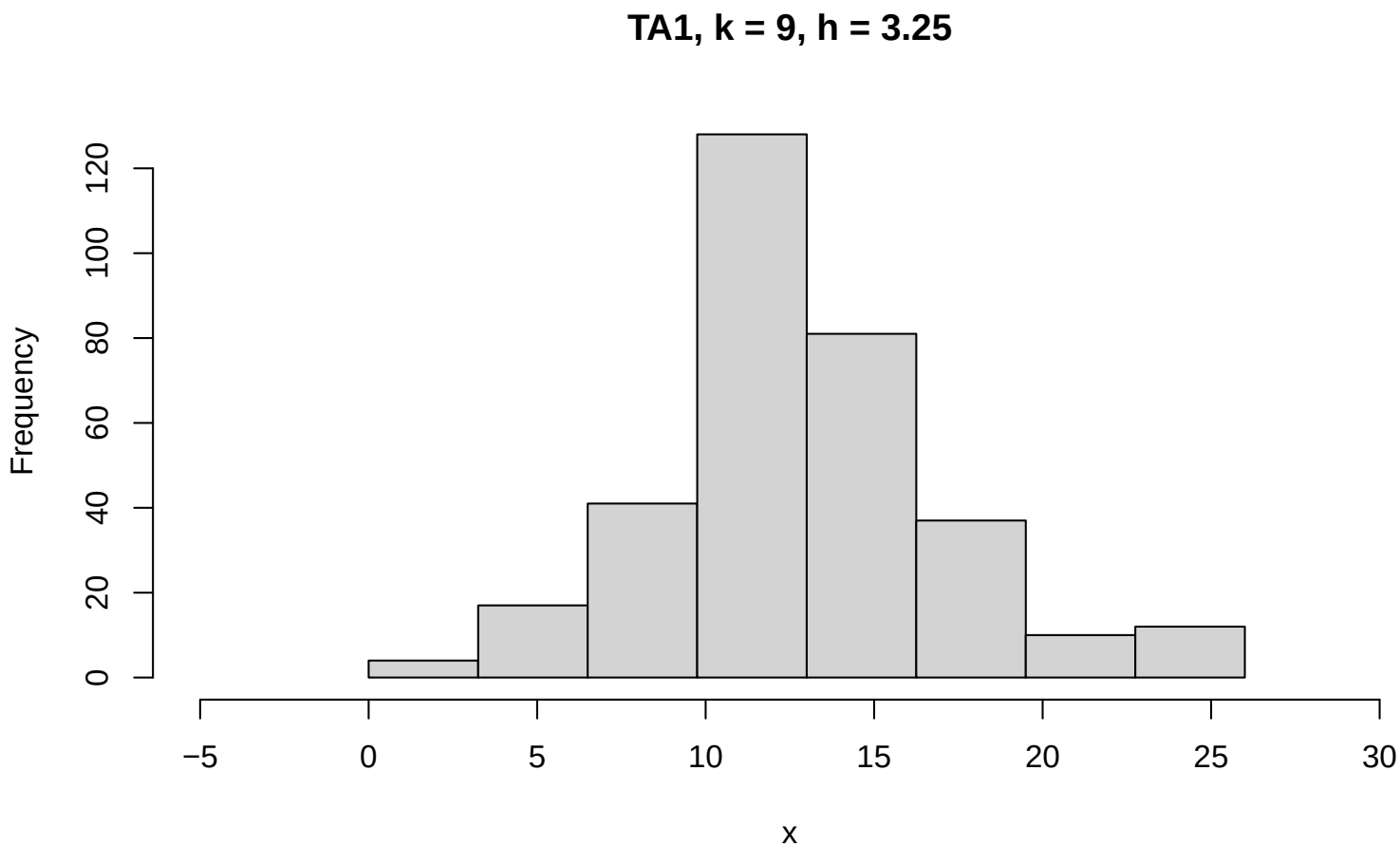
Histogramme

Sturges (1926) $k := \lceil \log_2 n + 1 \rceil$



Histogramme

Scott (1979) $h := 3.49S/\sqrt[3]{n}$



Empirische Verteilungsfunktion

Definition (Kumulative absolute und relative Häufigkeitsverteilungen)

$x = (x_1, ..., x_n)$ sei ein Datensatz, $A := \{a_1, ..., a_k\}$ mit $k \leq n$ die im Datensatz vorkommenden verschiedenen Zahlenwerte und h und r die absoluten und relativen Häufigkeitsverteilungen von x, respektive. Dann heißt

$$H : A \to \mathbb{N}, a \mapsto H(a) := \sum_{a' \leq a} h(a') \tag{3}$$

die *kumulative absolute Häufigkeitsverteilung* von x und die Funktion

$$R : A \to [0, 1], a \mapsto R(a) := \sum_{a' \leq a} r(a') \tag{4}$$

die *kumulative relative Häufigkeitsverteilung* der Zahlwerte von x.

Mit den Definitionen der absoluten und relativen Häufigkeitsverteilungen gilt also

$$H(a) = \text{Anzahl der } x_i \text{ aus } x \text{ mit } x_i \leq a$$

und

$$R(a) = \text{Anzahl der } x_i \text{ aus } x \text{ mit } x_i \leq a \text{ geteilt durch } n.$$

Empirische Verteilungsfunktion

cumsum() erlaubt die Berechnung kumulativer Summen von atomic vector Elementen

```
x         = affect$TA1                  # double vector der TA1 Werte
n         = length(x)                   # Anzahl der Datenwerte
H         = as.data.frame(table(x))     # absolute Hauefigkeitsverteilung dataframe
names(H)= c("a", "h")                   # Spaltenbenennung
H$H       = cumsum(H$h)                 # kumulative absolute Haeufigkeitsverteilung
H$r       = H$h/n                       # relative Haeufigkeitsverteilung
H$R       = cumsum(H$r)                 # kumulative relative Haeufigkeitsverteilung
```

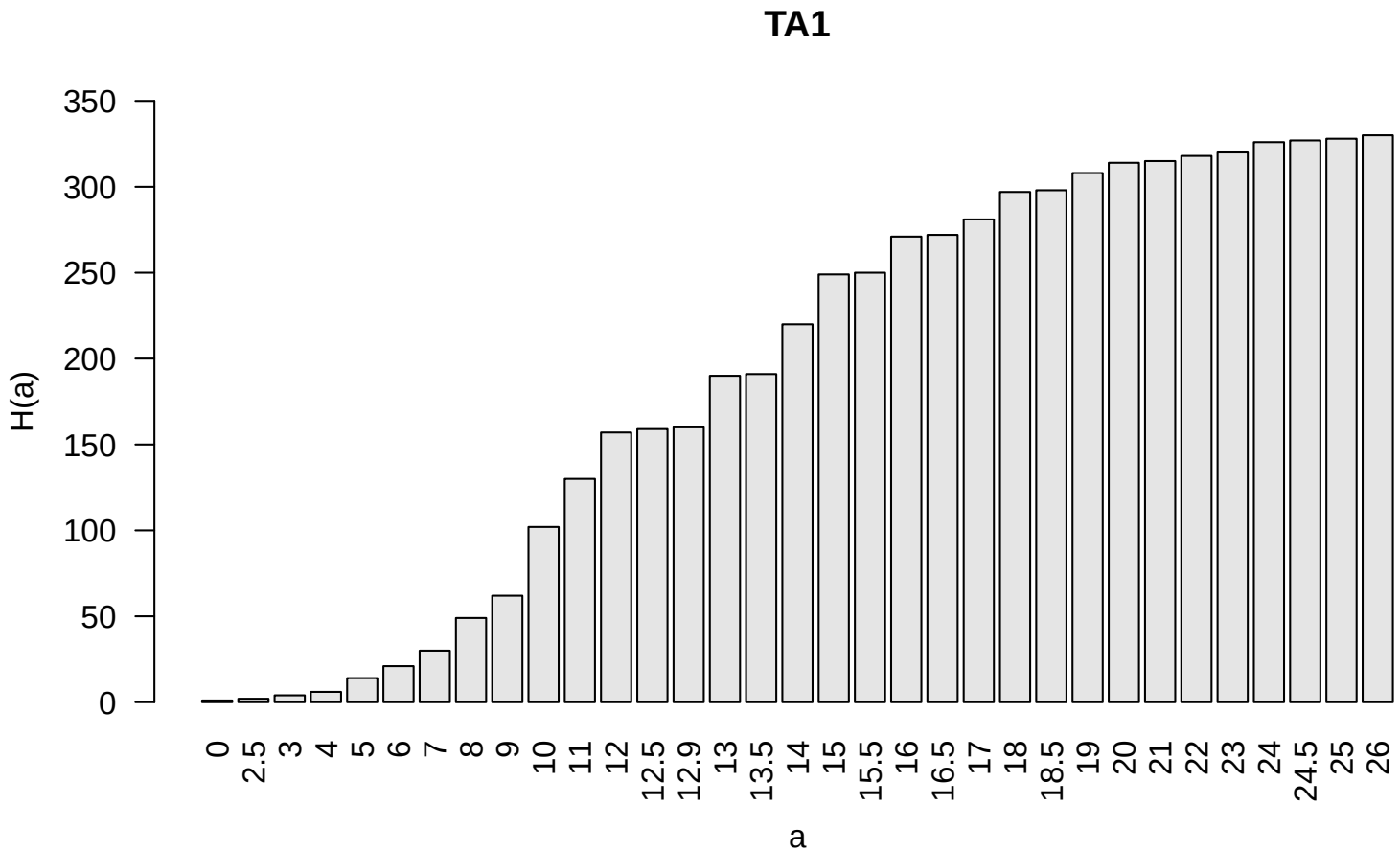
	a	h	H	r	R
1	0	1	1	0.003	0.003
2	2.5	1	2	0.003	0.006
3	3	2	4	0.006	0.012
4	4	2	6	0.006	0.018
5	5	8	14	0.024	0.042
6	6	7	21	0.021	0.064
7	7	9	30	0.027	0.091
8	8	19	49	0.058	0.148
9	9	13	62	0.039	0.188
10	10	40	102	0.121	0.309
11	11	28	130	0.085	0.394
12	12	27	157	0.082	0.476
13	12.5	2	159	0.006	0.482
14	12.9	1	160	0.003	0.485

Empirische Verteilungsfunktion

```
# Visualisierung der kumulativen absoluten Haeufigkeitsverteilung
Ha          = H$H                   # H(a) Werte
names(Ha)   = H$a                   # barplot braucht a Werte als names
dev.new()                           # Abbildungsinitialisierung
barplot(                            # Balkendiagramm
Ha,                                 # H(a) Werte
col         = "gray90",             # Balkenfarbe
xlab        = "a",                  # x Achsenbeschriftung
ylab        = "H(a)",               # y Achsenbeschriftung
las         = 2,                    # Anzeigen aller x Werte
ylim        = c(0,350),             # y Achsenlimits
main        = "TA1")                # Titel

# Visualisierung der kumulativen relativen Haeufigkeitsverteilung
R           = H$R                   # R(a) Werte
names(R)    = H$a                   # barplot braucht a Werte als names
dev.new()                           # Abbildungsinitialisierung
barplot(                            #  Balkendiagramm
R,                                  # R(a) Werte
col         = "gray90",             # Balkenfarbe
xlab        = "a",                  # x Achsenbeschriftung
ylab        = "R(a)",               # y Achsenbeschriftung
las         = 2,                    # Anzeigen aller x Werte
ylim        = c(0,1),               # y Achsenlimits
main        = "TA1")                # Titel
```

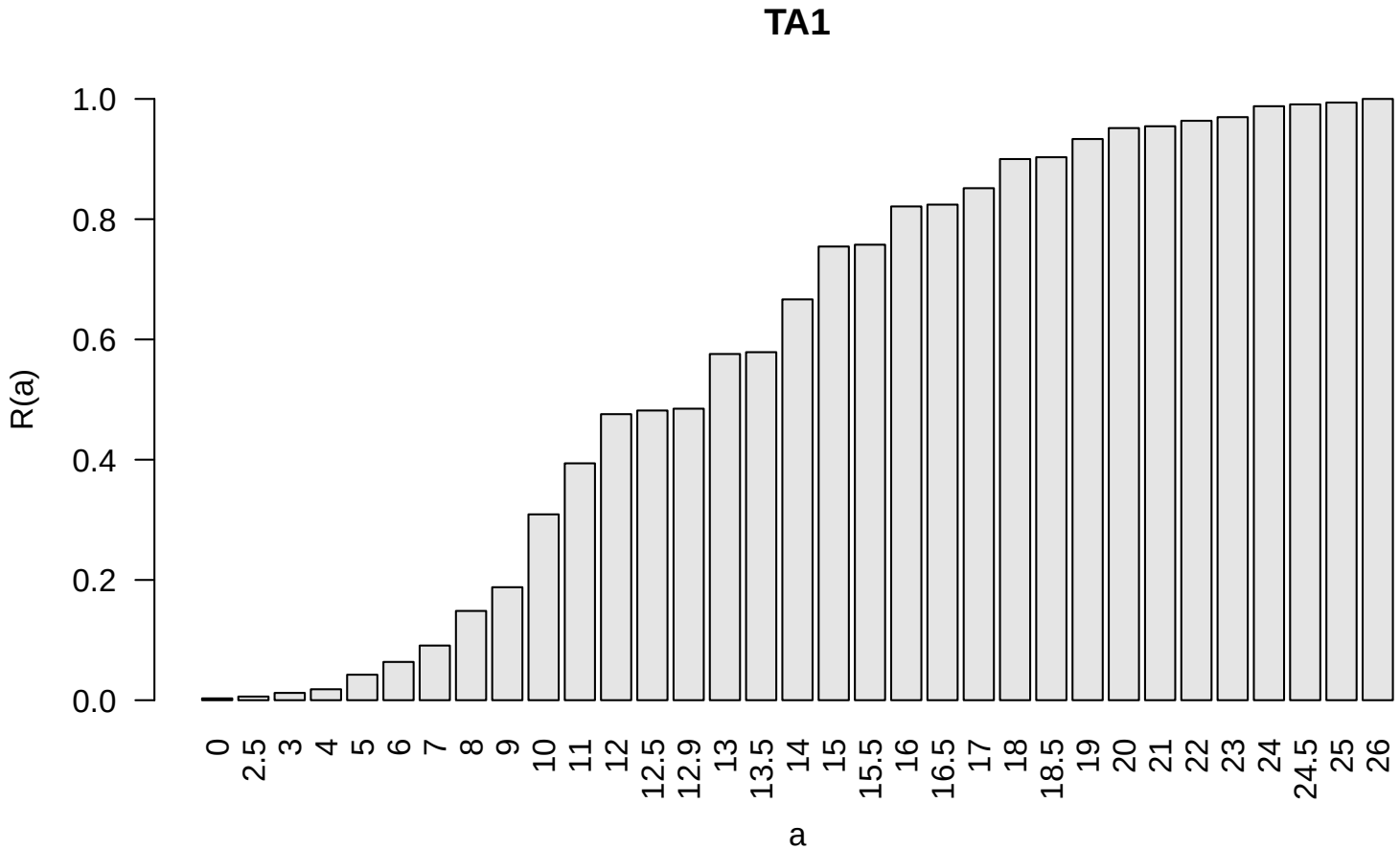
Empirische Verteilungsfunktion

barplot() zur Visualisierung der kumulativen absoluten Häufigkeitsverteilung



Empirische Verteilungsfunktion

barplot() zur Visualisierung der kumulativen relativen Häufigkeitsverteilung



Empirische Verteilungsfunktion

Definition (Empirische Verteilungsfunktion)

$x = (x_1, ..., x_n)$ sei ein Datensatz. Dann heißt die Funktion

$$F : \mathbb{R} \to [0, 1], \xi \mapsto F(\xi) := \frac{\text{Anzahl der } x_i \text{ aus } x \text{ mit } x_i \leq \xi}{n} \tag{5}$$

die empirische Verteilungsfunktion (EVF) von x.

Bemerkungen

- Die empirische Verteilungsfunktion wird auch *empirische kumulative Verteilungsfunktion* genannt.
- Die Definitionsmenge der EVF ist im Gegensatz zu Häufigkeitsverteilungen $\mathbb{R}$ und nicht A; die EVF verhält sich zu kumulativen Häufigkeitsverteilungen wie Histogramme zu Häufigkeitsverteilungen.
- Typischerweise sind empirische Verteilungsfunktionen Treppenfunktionen.
- Die (visuelle) Umkehrfunktion der EVF kann zur Bestimmung von Quantilen genutzt werden.

Empirische Verteilungsfunktion

ecdf() erlaubt die Evaluation der empirischen Verteilungsfunktion

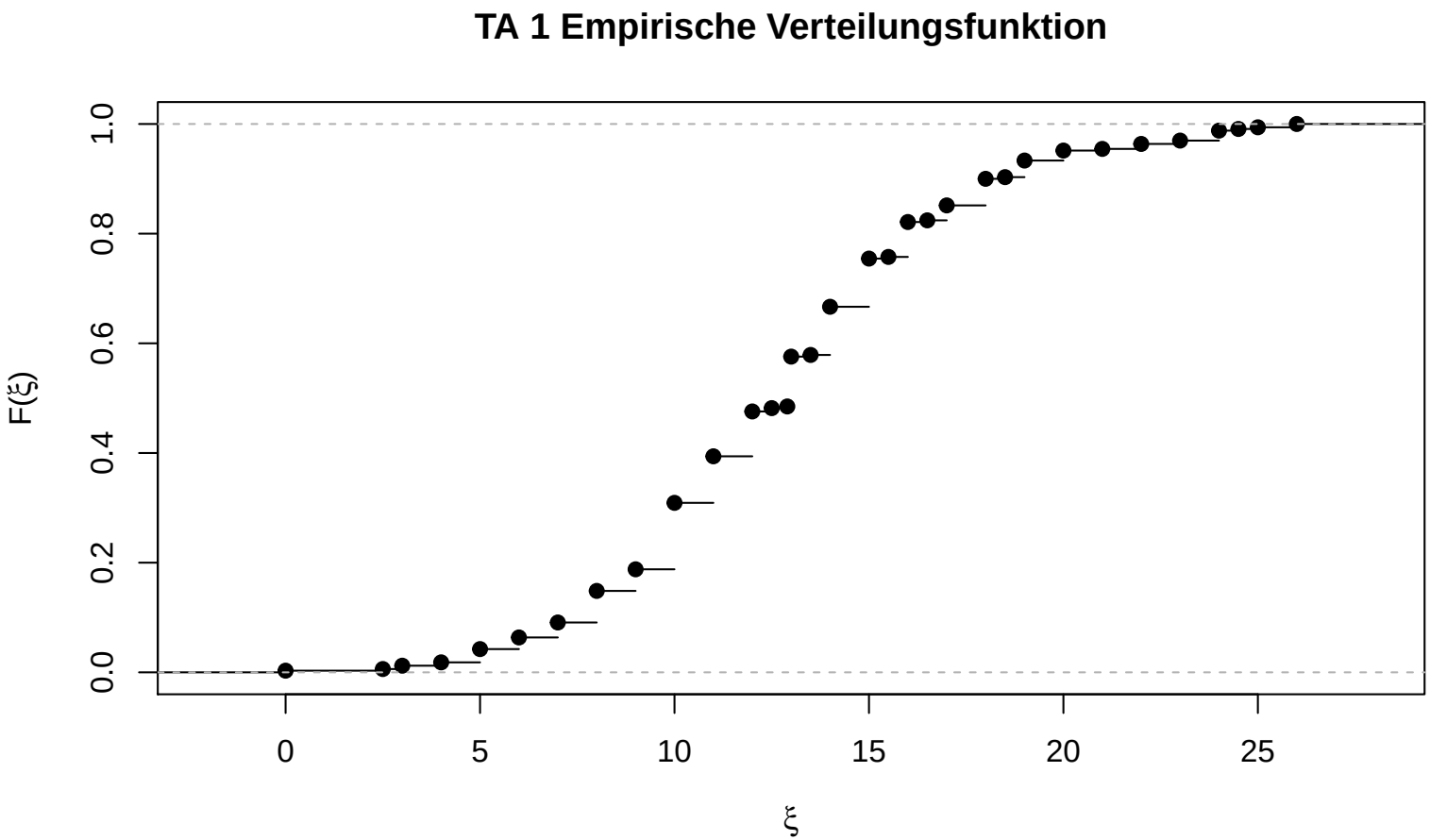
```
x    = affect$TA1                                  # double vector der TA1 Werte
evf  = ecdf(x)                                     # Evaluation der EVF
plot(                                              # plot weiss mit ecdf object umzugehen
evf,                                               # ecdf Objekt
xlab = TeX("$\\xi$"),                              # x Achsenbeschriftung
ylab = TeX("$F(\\xi)$") ,                          # y Achsenbeschriftung
main = "TA1 Empirische Verteilungsfunktion")       # Titel
```

Kombination mit **abline()** erlaubt das Ablesen von *Quantilen*

```
plot(                                              # plot weiss mit ecdf Objekt umzugehen
evf,                                               # ecdf Objekt
verticals    = TRUE,                               # vertikale Linien
do.points    = FALSE,                              # keine Punkte
xlab         = TeX("$\\xi$"),                      # x Achsenbeschriftung
ylab         = TeX("$F(\\xi)$") ,                  # y Achsenbeschriftung
main  = "TA 1 Empirische Verteilungsfunktion")     # Titel
abline(                                            # horizontale Linie
h            = 0.25,                               # y Ordinate der Linie
lty          = 3,                                  # fein gestrichelt
col          = "blue")                             # blau
```

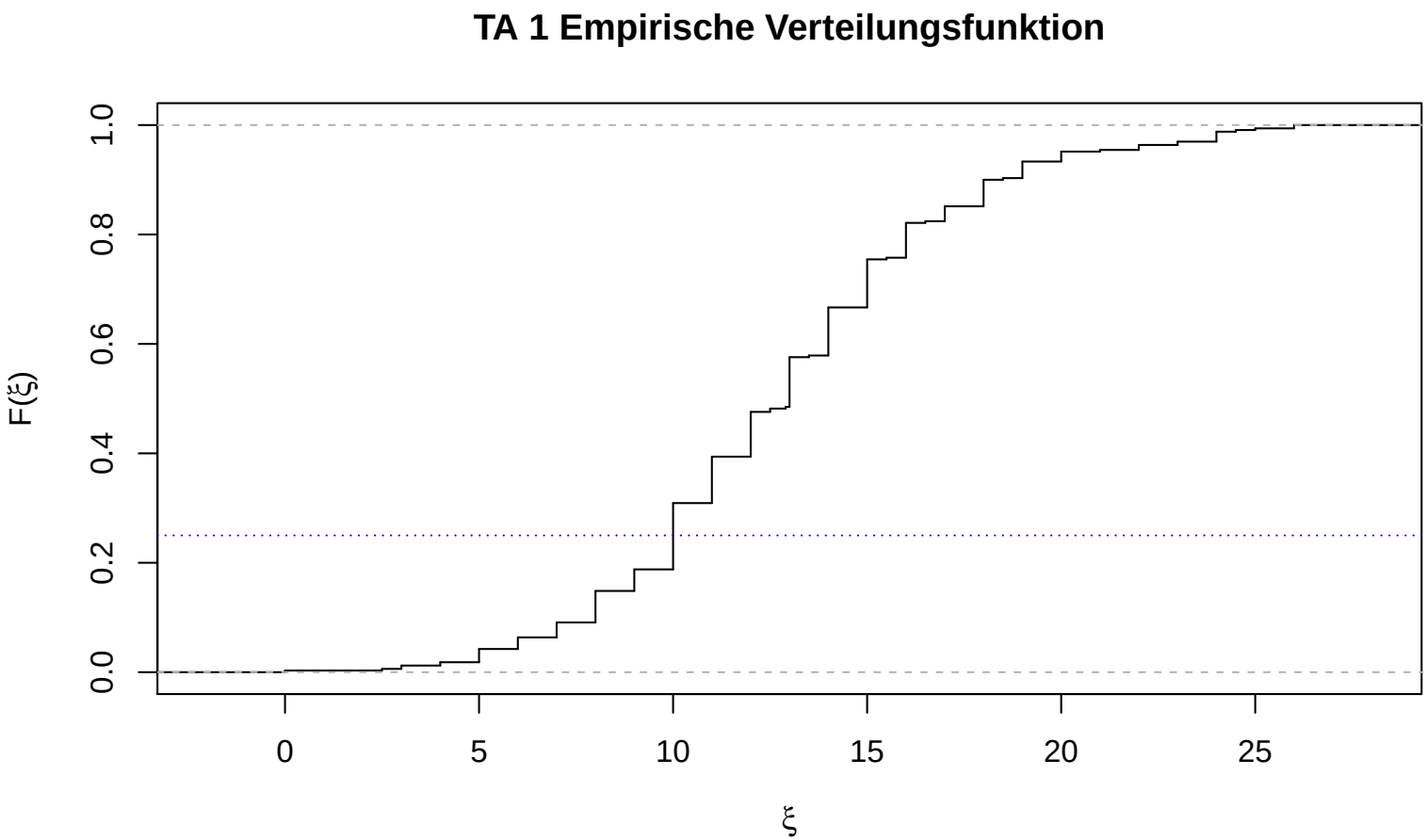
Empirische Verteilungsfunktion

Die Punkt-Liniendarstellung betont die rechtsseitige Stetigkeit.



Empirische Verteilungsfunktion

Kombination mit **abline()** zum Ablesen von *Quantilen*



Quantile und Boxplots

Definition (p-Quantil)

$x = (x_1, ..., x_n)$ sei ein Datensatz und

$$x_s = \left(x_{(1)}, x_{(2)}, ..., x_{(n)}\right) \text{ mit } \min_{1 \leq i \leq n} x_i = x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)} = \max_{1 \leq i \leq n} x_i \tag{6}$$

der zugehörige aufsteigend sortierte Datensatz. Weiterhin bezeichne $\lfloor \cdot \rfloor$ die Abrundungsfunktion. Dann heißt für ein $p \in [0, 1]$ die Zahl

$$x_p := \begin{cases} x_{(\lfloor np+1 \rfloor)} & \text{falls } np \neq \mathbb{N} \\ \frac{1}{2}\left(x_{(np)} + x_{(np+1)}\right) & \text{falls } np \in \mathbb{N} \end{cases} \tag{7}$$

das p-*Quantil* von x.

Bemerkungen

- Mindestens $p \cdot 100\%$ aller Werte in x sind kleiner oder gleich x_p.
- Mindestens $(1 - p) \cdot 100\%$ aller Werte in x sind größer als x_p .
- Das p-Quantil teilt den geordneten Datensatz im Verhältnis p zu $(1 - p)$ auf.
- $x_{0.25}, x_{0.50}, x_{0.75}$ heißen *unteres Quartil, Median*, und *oberes Quartil*, respektive.
- $x_{j \cdot 0.10}$ für $j = 1, ..., 9$ heißen *Dezile*, $x_{j \cdot 0.01}$ für $j = 1, ..., 99$ heißen *Percentile*.

Quantile und Boxplots

Beispiel (p-Quantil, Henze (2018, Kapitel 5))

i	1	2	3	4	5	6	7	8	9	10
x_i	8.5	1.5	75	4.5	6.0	3.0	3.0	2.5	6.0	9.0
$x_{(i)}$	1.5	2.5	3.0	3.0	4.5	6.0	6.0	8.5	9.0	75

0.25-Quantil

Es ist $n = 10$ und es sei $p := 0.25$. Dann gilt $np = 10 \cdot 0.25 = 2.5 \notin \mathbb{N}$. Also folgt

$$x_{0.25} = x_{(\lfloor 2.5+1 \rfloor)} = x_{(3)} = 3.0 \tag{8}$$

0.80-Quantil

Es ist $n = 10$ und es sei $p := 0.80$. Dann gilt $np = 10 \cdot 0.80 = 8 \in \mathbb{N}$. Also folgt

$$x_{0.80} = \frac{1}{2}\left(x_{(8)} + x_{(8+1)}\right) = \frac{1}{2}\left(x_{(8)} + x_{(9)}\right) = \frac{8.5 + 9.0}{2} = 8.75. \tag{9}$$

Quantile und Boxplots

Beispiel (p-Quantil, Henze (2018, Kapitel 5))

- "Manuelle" Quantilbestimmung anhand obiger Definition

```
x   = c(8.5,1.5,75,4.5,6.0,3.0,3.0,2.5,6.0,9.0)   # Beispieldaten
n   = length(x)                                    # Anzahl Datenwerte
x_s = sort(x)                                      # sortierter Datensatz
p   = 0.25                                         # np \notin \mathbb{N}
x_p = x_s[floor(n*p + 1)]                          # 0.25 Quantil
[1] 3.0
p   = 0.80                                         # np \in \mathbb{N}
x_p = (1/2)*(x_s[n*p] + x_s[n*p + 1])              # 0.80 Quantil
[1] 8.75
```

- **quantile()** wertet Quantile anhand der Quantildefinition *type* = aus
- Es gibt mindestens neun verschiedene Quantildefinitionen (Hyndman and Fan, 1996)

```
x_p = quantile(x, 0.25, type = 2)                  # 0.25 Quantil, Definition 2
[1] 3.0
x_p = quantile(x, 0.80, type = 2)                  # 0.80 Quantil, Definition 2
[1] 8.75
x_p = quantile(x, 0.80, type = 1)                  # 0.80 Quantil, Definition 1
[1] 8.5
```

Quantile und Boxplots

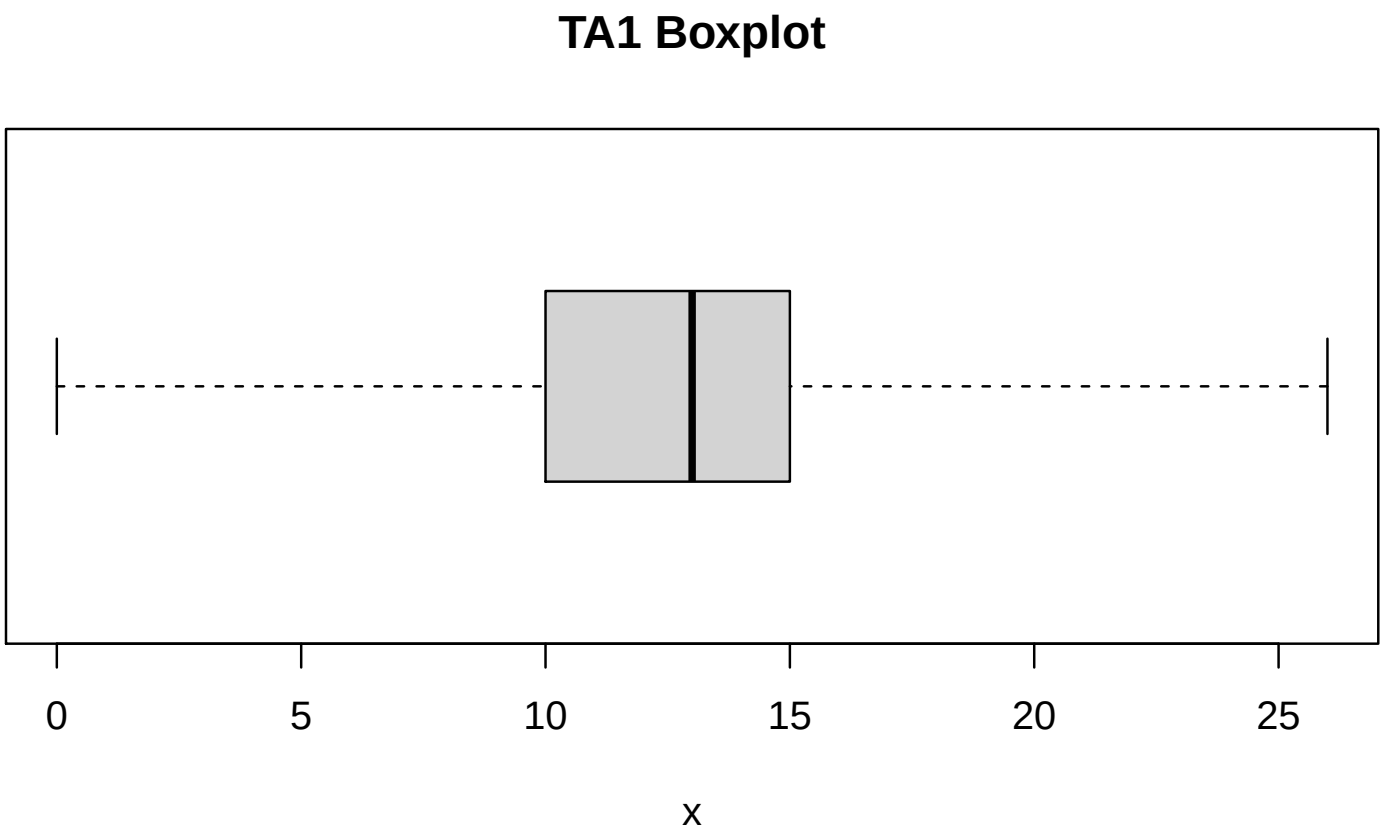
Boxplot

- Ein Boxplot visualisiert eine Quantil-basierte Zusammenfassung eines Datensatzes
- Typischerweise werden $\min x, x_{0.25}, x_{0.50}, x_{0.75}, \max x$ visualisiert
  - $\min x$ und $\max x$ werden oft als "Whiskerendpunkte" dargestellt
  - $x_{0.25}$ und $x_{0.75}$ sind untere und obere Grenze der zentralen grauen Box
  - $x_{0.50}$ wird als Strich in der zentralen grauen Box abgebildet
- $d_Q := x_{0.75} - x_{0.25}$ heißt *Interquartilsabstand* und dient als Verteilungsbreitenmaß
- **boxplot()** erstellt einen Boxplot, **summary()** liefert wesentliche Kennzahlen

```
# Sechswertezusammenfassung
summary(affect$TA1)
Min.   1st Qu.    Median    Mean   3rd Qu.  Max.
0.00   10.00      13.00     12.91   15.00    26.00

# Boxplot
boxplot(                              # Boxplot
affect$TA1,                           # Datensatz
horizontal  = T,                      # horizontale Darstellung
range       = 0,                      # Whiskers bis zu min x und max x
xlab        = "x",                    # x Achsenbeschriftung
main        = "TA1 Boxplot")          # Titel
```

Empirische Verteilungsfunktion

Boxplot



Es gibt viele Boxplotvariationen(z.B. McGill et al., 1978), Erläuterung ist immer nötig!

Univariate Deskriptivstatistiken

- Verteilungsdarstellung
- **Maße der zentralen Tendenz**
- Maße der Datenvariabilität
- Übungen und Selbstkontrollfragen

Mittelwert

Definition (Mittelwert)

$x = (x_1, ..., x_n)$ sei ein Datensatz. Dann heißt $\bar{x} := \frac{1}{n} \sum_{i=1}^{n} x_i$ der Mittelwert von x.

mean() zur Berechnung des Mittelwerts

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
n       = length(x)                     # Anzahl der Werte
x_bar   = (1/n)*sum(x)                  # "manuelle" Mittelwertsberechnung
x_bar   = mean(x)                       # "automatische" Mittelwertsberechnung
```

Die Summe der Abweichungen vom Mittelwert ist Null

```
s    = sum(x - mean(x))                 # Summe der Abweichungen vom Mittelwert
[1] -8.704149e-14                       # Rundungsfehler
```

Die absoluten Summen positiver und negativer Abweichungen vom Mittelwert sind gleich.

```
s_1 = sum(x[x <= mean(x)] - mean(x))    # Summe aller negativer Abweichungen
s_1
[1] -565.4909
s_2 = sum(x[x > mean(x)] - mean(x))     # Summe aller positiver Abweichungen
s_2
[1] 565.4909
```

Mittelwert

Der Mittelwert der Summe zweier Datenreihen entspricht der Summe ihrer Mittelwerte:

$$\overline{x+y} := \frac{1}{n}\sum_{i=1}^{n}(x_i + y_i) = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=1}^{n} y_i =: \bar{x} + \bar{y}$$

```
x     = affect$TA1                    # double Vektor der TA1 Werte
x_bar = mean(x)                       # Mittelwert der TA1 Werte
y     = affect$EA1                    # double Vektor der EA1 Werte
y_bar = mean(y)                       # Mittelwert der EA1 Wert
z     = x + y                         # double Vektor der TA und EA Werte
z_bar = mean(z)                       # Mittelwert der Summe der TA und EA Werte
[1] 22.10576
xy_bar  = x_bar + y_bar               # Summe der Mittelwerte der TA und EA Werte
[1] 22.10576
```

Mittelwert

Linear-affine Transformation der Daten transformiert den Mittelwert linear-affin:

$$\overline{ax + b} = a\bar{x} + b$$

Beweis

$$\begin{aligned}\overline{ax + b} &:= \frac{1}{n}\sum_{i=1}^{n}(ax_i + b) \\ &= \sum_{i=1}^{n}\left(\frac{1}{n}ax_i + \frac{1}{n}b\right) = \sum_{i=1}^{n}\left(\frac{1}{n}ax_i\right) + \sum_{i=1}^{n}\left(\frac{1}{n}b\right) = a\frac{1}{n}\sum_{i=1}^{n}x_i + \frac{1}{n}\sum_{i=1}^{n}b = a\bar{x} + b\end{aligned}$$

□

```
x     = affect$TA1                    # double Vektor der TA1 Werte
x_bar = mean(x)                       # Mittelwert der TA1 Werte
a     = 2                             # Multiplikationskonstante
b     = 5                             # Additionskonstante
y     = a*x + b                       # linear-affine Transformation der TA1 Werte
y_bar = mean(y)                       # Mittelwert der transfomierten TA1 Werte
[1] 30.82364
y_bar   = a*x_bar + b                 # Transformation des TA1 Mittelwerts
[1] 30.82364
```

Mittelwert

Beim *getrimmten Mittelwert* wird ein relativer Anteil an Extremwerten ausgeschlossen.

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
n       = length(x)                     # Anzahl der Werte
t       = 0.3                           # getrimmter relativer Anteil
n_t     = round(0.3*n)                  # Anzahl zu trimmender Werte
x_sort    = sort(x)                     # aufsteigend sortierter Vektor
x_trim  = x_sort[(n_t+1):(n - n_t)]     # getrimmter Datensatz
x_tbar  = mean(x_trim)                  # getrimmter Mittelwert
[1] 12.68485
x_tbar  = mean(x, trim = 0.3)           # "automatische" Berechnung
[1] 12.68485
```

Median

Definition (Median)

$x = (x_1, ..., x_n)$ sei ein Datensatz und $x_s = (x_{(1)}, ..., x_{(n)})$ der zugehörige aufsteigend sortierte Datensatz. Dann ist der Median von x definiert als

$$\tilde{x} := \begin{cases} x_{((n+1)/2)} & \text{falls } n \text{ ungerade} \\ \frac{x_{(n/2)} + x_{((n+1)/2)}}{2} & \text{falls } n \text{ gerade} \end{cases} \tag{10}$$

median() zur Berechnung des Medians

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
n       = length(x)                     # Anzahl der Werte
x_s     = sort(x)                       # aufsteigend sortierter Vektor
if(n %% 2 == 1)                         # n ungerade, n mod 2 == 1
{
    x_tilde = x_s[(n+1)/2]
} else                                  # n gerade, n mod 2 == 0
{
    x_tilde = (x_s[n/2] + x_s[(n+1)/2])/2
}
x_tilde   = median(x)                   # "automatische" Berechnung
```

Median

Die Summe der Abweichungsbeträge vom Median ist minimal

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
x_bar   = mean(x)                       # Mittelwert der TA1 Werte
x_tilde = median(x)                     # Median der TA1 Werte
s_1     = sum(abs(x - x_bar))           # Summe Abweichungsbetraege vom Mittelwert
[1] 1130.982
s_2     = sum(abs(x - x_tilde))         # Summe Abweichungsbetraege vom Median
[1] 1130.1
```

Der Median ist weniger anfällig für Ausreißer als der Mittelwert

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
x_bar   = mean(x)                       # Mittelwert der TA1 Werte
[1] 12.91182
x_tilde = median(x)                     # Median der TA1 Werte
[1] 13
y       = x                             # neuer Datensatz mit
y[1]    = 10000                         # ... einem Extremwert
y_bar   = mean(y)                       # Mittelwert der des neuen Datensatzes
[1] 43.17242
y_tilde = median(y)                     # Median bleibt unveraendert
[1] 13
```

Deskriptive Statistiken

- Verteilungsdarstellung
- Maße der zentralen Tendenz
- **Maße der Datenvariabilität**
- Bivariate Deskriptivstatistik
- Übungen und Selbstkontrollfragen

Spannbreite

Definition

$x = (x_1, ..., x_n)$ sei ein Datensatz. Dann ist die *Spannbreite* von $x_1, ..., x_n$ definiert als

$$S := \max(x_1, ..., x_n) - \min(x_1, ..., x_n). \tag{11}$$

range() zur Berechnung der Spannbreite

```
x       = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
x_max   = max(x)                        # Maximum der TA1 Werte
x_min   = min(x)                        # Mininum der TA1 Werte
S       = x_max - x_min                 # Spannbreite
[1] 26
MinMax  = range(x)                      # "automatische" Berechnung von min(x), max(x)
[1] 0 26
S       = MinMax[2] - MinMax[1]         # Spannbreite
```

Stichprobenvarianz

Definition (Stichprobenvarianz, empirische Stichprobenvarianz)

$x = (x_1, ..., x_n)$ sei ein Datensatz. Die *Stichprobenvarianz* von x ist definiert als u

$$S^2 := \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

und die *empirische Stichprobenvarianz* von x ist definiert als

$$\tilde{S}^2 := \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2.$$

Inferenzstatistische Anmerkungen

- S^2 ist ein unverzerrter Schätzer von $\mathbb{V}(X)$, $\tilde{S}^2$ ist ein verzerrter Schätzer $\mathbb{V}(X)$.
- Für $n \to \infty$ gilt $\frac{1}{n} \approx \frac{1}{n-1}$, $\tilde{S}^2$ ist ein asymptotisch unverzerrter Schätzer von $\mathbb{V}(X)$.
- $\tilde{S}^2$ ist der ML Schätzer, S^2 ist der ReML Schätzer von σ^2 bei $X_1, ..., X_n \sim N(\mu, \sigma^2)$.
- Es gelten $\tilde{S}^2 = \frac{n-1}{n} S^2$, $S^2 = \frac{n}{n-1} \tilde{S}^2$ und $0 \leq \tilde{S}^2 \leq S^2$.

Stichprobenvarianz

var() zum Berechnen der Stichprobenvarianz

```
x         = affect$TA1                          # double Vektor der Tense Arousal 1 Werte
n         = length(x)                           # Anzahl der Werte
S2        = (1/(n-1))*sum((x - mean(x))^2)      # Stichprobenvarianz
[1] 19.53934
S2        = var(x)                              # "automatische" Stichprobenvarianz
[1] 19.53934
S2_tilde  = (1/n)*sum((x - mean(x))^2)          # Empirische Stichprobenvarianz
[1] 19.48013
S2_tilde  = ((n-1)/n)*var(x)                    # "automatische" emp. Stichprobenvarianz
[1] 19.48013
```

Stichprobenvarianz

Für einen Datensatz $x_1, ..., x_n$, den linear-affin transformierten Datentsatz $y_1 := ax_1 + b, ..., y_n := ax_n + b$ und mit Stichprobenvarianzen S_x^2 und S_y^2 der jeweiligen Datensätze gilt

$$S_y^2 = a^2 S_x^2.$$

Beweis

$$\begin{aligned}
S_y^2 &:= \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2 \\
&= \frac{1}{n-1} \sum_{i=1}^{n} (ax_i + b - (a\bar{x} + b))^2 \\
&= \frac{1}{n-1} \sum_{i=1}^{n} (ax_i + b - a\bar{x} - b)^2 \\
&= \frac{1}{n-1} \sum_{i=1}^{n} (a(x_i - \bar{x}))^2 \\
&= \frac{1}{n-1} \sum_{i=1}^{n} a^2 (x_i - \bar{x})^2 \\
&= a^2 \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \\
&= a^2 S_x^2
\end{aligned}$$

□

Stichprobenvarianz

Für einen Datensatz $x_1, ..., x_n$, den linear-affin transformierten Datentsatz $y_1 := ax_1 + b, ..., y_n := ax_n + b$ und mit Stichprobenvarianzen S_x^2 und S_y^2 der jeweiligen Datensätze gilt

$$S_y^2 = a^2 S_x^2.$$

```
x     = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
S2x   = var(x)                        # Stichprobenvarianz von x_1,...,x_n
a   = 2                               # Multiplikationskonstante
b     = 5                             # Additionskonstante
y     = a*x + b                       # y_i = ax_i + b
S2y   = var(y)                        # Stichprobenvarianz y_1,...,y_n
[1] 78.15737
S2y   = a^2 * S2x                     # Stichprobenvarianz y_1,...,y_n
[1] 78.15737
```

Stichprobenvarianz

Für einen Datensatz x und sein elementweises Quadrat x^2 gilt

$$\tilde{S}^2 = \overline{x^2} - \bar{x}^2 \tag{12}$$

Beweis

$$\begin{aligned}
\tilde{S}^2 &:= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(x_i^2 - 2x_i\bar{x} + \bar{x}^2\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - 2\bar{x}\frac{1}{n}\sum_{i=1}^{n}x_i + \frac{1}{n}\sum_{i=1}^{n}\bar{x}^2 \\
&= \overline{x^2} - 2\bar{x}\bar{x} + \frac{1}{n}n\bar{x}^2 \\
&= \overline{x^2} - 2\bar{x}^2 + \bar{x}^2 \\
&= \overline{x^2} - \bar{x}^2
\end{aligned}$$

□

Stichprobenvarianz

Für einen Datensatz x und sein elementweises Quadrat x^2 gilt

$$\tilde{S}^2 = \overline{x^2} - \bar{x}^2 \tag{13}$$

```
x          = affect$TA1                    # double Vektor der Tense Arousal 1 Werte
x_bar      = mean(x)                       # Stichprobenmittel
S2_tilde   = ((n-1)/n)*var(x)              # empirische Stichprobenvarianz
[1] 19.48013
S2_tilde   = mean(x^2) - (mean(x))^2       # \bar{x^2} - \bar{x}^2
[1] 19.48013
S2         = var(x)                        # S^2 \neq \bar{x^2} - \bar{x}^2
[1] 19.53934
```

Stichprobenstandardabweichung

Definition (Stichprobenstandardabweichung, empirische)

$x = (x_1, ..., x_n)$ sei ein Datensatz. Die *Stichprobenstandardabweichung* von x ist definiert als

$$S := \sqrt{S^2}$$

und die *empirische Stichprobenstandardabweichung* von x ist definiert als

$$\tilde{S} := \sqrt{\tilde{S}^2}.$$

Inferenzstatistische Anmerkungen

- S ist ein verzerrter Schätzer von $\mathbb{S}(X)$.
- S^2 misst Variabilität in quadrierten Einheiten, zum Beispiel Quadratmeter (m^2).
- S misst Variabilität in unquadrierten Einheiten, zum Beispiel Meter (m).
- Es gilt $\tilde{S} = \sqrt{(n-1)/n}S$.

Stichprobenstandardabweichung

sd() zur Berechnung der Stichprobenstandardabweichung

```
x      = affect$TA1                                  # double Vektor der TA 1 Werte
n      = length(x)                                   # Anzahl der Werte
S      = sqrt((1/(n-1))*sum((x - mean(x))^2))        # Standardabweichung
[1] 4.420333
S      = sd(x)                                       # "automatische" Berechnung
[1] 4.420333
S_tilde = sqrt((1/(n))*sum((x - mean(x))^2))         # empirische Standardabweichung
[1] 4.41363
S_tilde   = sqrt((n-1)/n)*sd(x)                      # empirische Standardabweichung
[1] 4.41363
```

Stichprobenstandardabweichung

Für einen Datensatz $x_1, ..., x_n$, den linear-affin transformierten Datentsatz $y_1 := ax_1 + b, ..., y_n := ax_n + b$ und mit Stichprobenstandardabweichung S_x und S_y der jeweiligen Datensätze gilt

$$S_y = |a| S_x.$$

Beweis

$$\begin{aligned}
S_y &:= \left(\frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2 \right)^{1/2} \\
&= \left(\frac{1}{n-1} \sum_{i=1}^{n} (ax_i + b - (a\bar{x} + b))^2 \right)^{1/2} \\
&= \left(\frac{1}{n-1} \sum_{i=1}^{n} (a(x_i - \bar{x}))^2 \right)^{1/2} \\
&= \left(\frac{1}{n-1} \sum_{i=1}^{n} a^2 (x_i - \bar{x})^2 \right)^{1/2} \\
&= \left(a^2\right)^{1/2} \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^{1/2}
\end{aligned}$$

Also gilt $S_y = aS_x$, wenn $a \geq 0$ und $S_y = -aS_x$, wenn $a < 0$. Dies aber entspricht $S_y = |a| S_x$. □

Stichprobenvarianz

Für einen Datensatz $x_1, ..., x_n$, den linear-affin transformierten Datentsatz $y_1 := ax_1 + b, ..., y_n := ax_n + b$ und mit Stichprobenstandardabweichung S_x und S_y der jeweiligen Datensätze gilt

$$S_y = |a| S_x.$$

```
# a >= 0
x   = affect$TA1                    # double Vektor der TA1 Werte
Sx  = sd(x)                         # Stichprobenvarianz von x
a = 2                               # Multiplikationskonstante
b   = 5                             # Additionskonstante
y   = a*x + b                       # y_i = ax_i + b
Sy  = sd(y)                         # Stichprobenvarianz von y
Sy  = a*Sx                          # Stichprobenvarianz von y

# a < 0
x   = affect$TA1                    # double Vektor der TA1 Werte
Sx  = sd(x)                         # Stichprobenvarianz von x
a = -3                              # Multiplikationskonstante
b   = 10                            # Additionskonstante
y   = a*x + b                       # y_i = ax_i + b
Sy  = sd(y)                         # Stichprobenvarianz von y
Sy  = (-a)*Sx                       # Stichprobenvarianz von y
```

Deskriptive Statistiken

- Verteilungsdarstellung
- Maße der zentralen Tendenz
- Maße der Datenvariabilität
- **Bivariate Deskriptivstatistik**
- Übungen und Selbstkontrollfragen

Bivariate Datensätze

- Wir bezeichnen einen bivariaten Datensatz mit

$$x = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n)) \,. \tag{14}$$

- x_i und y_i bezeichnen das erste und zweite Merkmal der $i$ten Einheit, respektive.
- n ist die Anzahl an bivariaten Datenpunkten (x_i, y_i).
- Untenstehende Tabelle zeigt ein Beispiel.

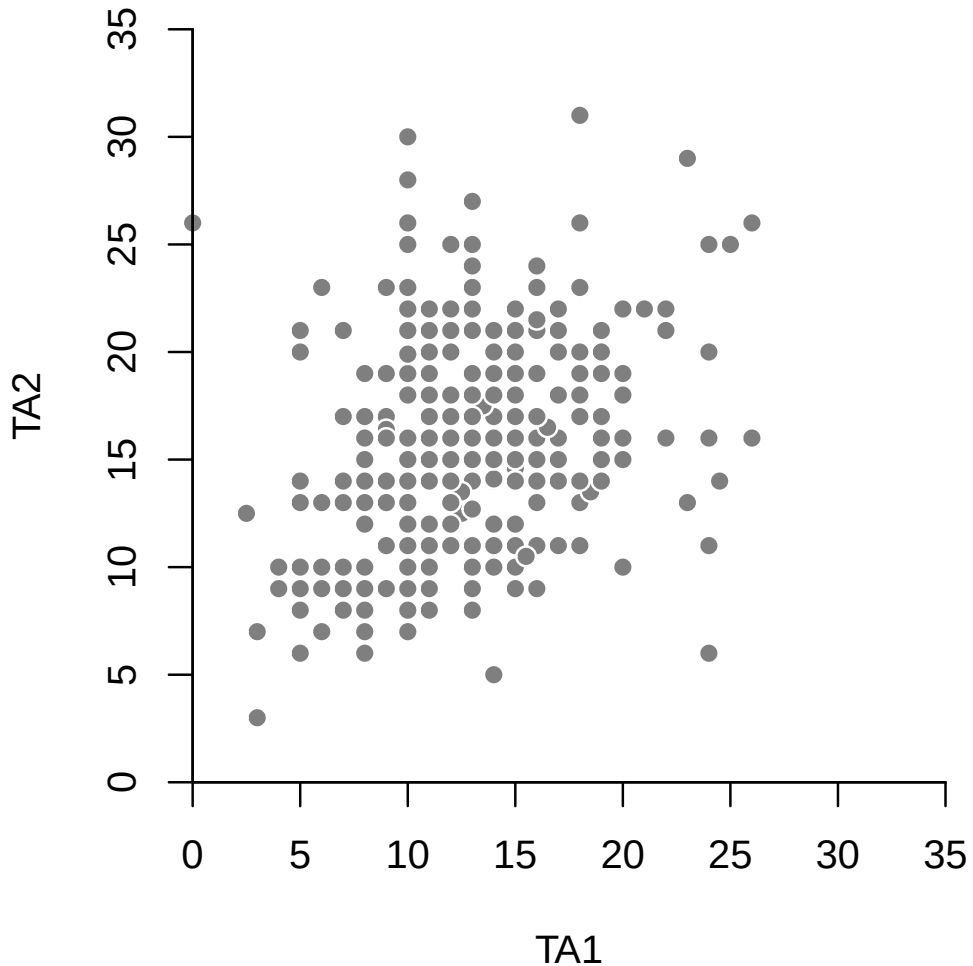
i	1	2	3	4	5	6	7	8	9	10
x_i	3.4	1.5	2.7	4.5	6.1	3.8	2.0	2.5	6.2	9.1
y_i	5.5	7.3	1.1	1.9	4.5	2.3	8.4	8.6	3.9	1.6

Streudiagramme

Die Darstellung der $(x_i, y_i), i = 1, ..., n$ in einem Koordinatensystem heißt *Streudiagramm*.

```
x       = affect$TA1                    # atomic vector x_1,...,x_n
y       = affect$TA2                    # atomic vector y_1,...,y_n
dev.new()                               # Abbildungsinitialisierung
par(                                    # Abbildungsparameter
pty     = "s",                          # Square Abbildung
bty     = "l",                          # Plot Box L
xaxs    = "i",                          # internal (tight) x Achsenstil
yaxs    = "i")                          # internal (tight) y Achsenstil
plot(                                   # Visualisierung
x,                                      # x
y,                                      # y
type    = "p",                          # Punkte
pch     = 21,                           # Punkttyp (?pch)
col     = "white",                      # Punktlinienfarbe
bg      = "gray50",                     # Punktfuellfarbe
xpd     = TRUE,                         # Punkte vor Achsen
cex     = 1.1,                          # Punktgroessenfaktor
xlab    = "TA1",                        # x Achsenbeschriftung
ylab    = "TA2",                        # y Achsenbeschriftung
xlim    = c(0,35),                      # x Achsengrenzen
ylim    = c(0,35))                      # y Achsengrenzen
```

Streudiagramme



Bivariate Histogramme

Definition (Bivariates Histogramm)

Ein *bivariates Histogramm* ist ein Diagramm, in dem zu einem Datensatz $x = ((x_i, y_i))$ mit $i = 1, ..., n$ über den Rechteckklassen

$$[b^x_{j-1}, b^x_j[\times [b^y_{k-1}, b^y_k[\text{ für } j = 1, ..., m_x, k = 1, ..., m_y \tag{15}$$

Blöcke mit Grundkante $[b^x_{j-1}, b^x_j[$ in der x-Ordinate und Grundkante $[b^y_{k-1}, b^y_k[$ in der y-Ordinate und Höhe

$$h_{jk} := \text{Anzahl der } (x_i, y_i) \text{ in } x \text{ mit } (x_i, y_i) \in [b^x_{j-1}, b^x_j[\times [b^y_{k-1}, b^y_k[\tag{16}$$

bzw.

$$r_{jk} := \frac{h_{jk}}{n} \tag{17}$$

abgebildet sind.

Bemerkungen

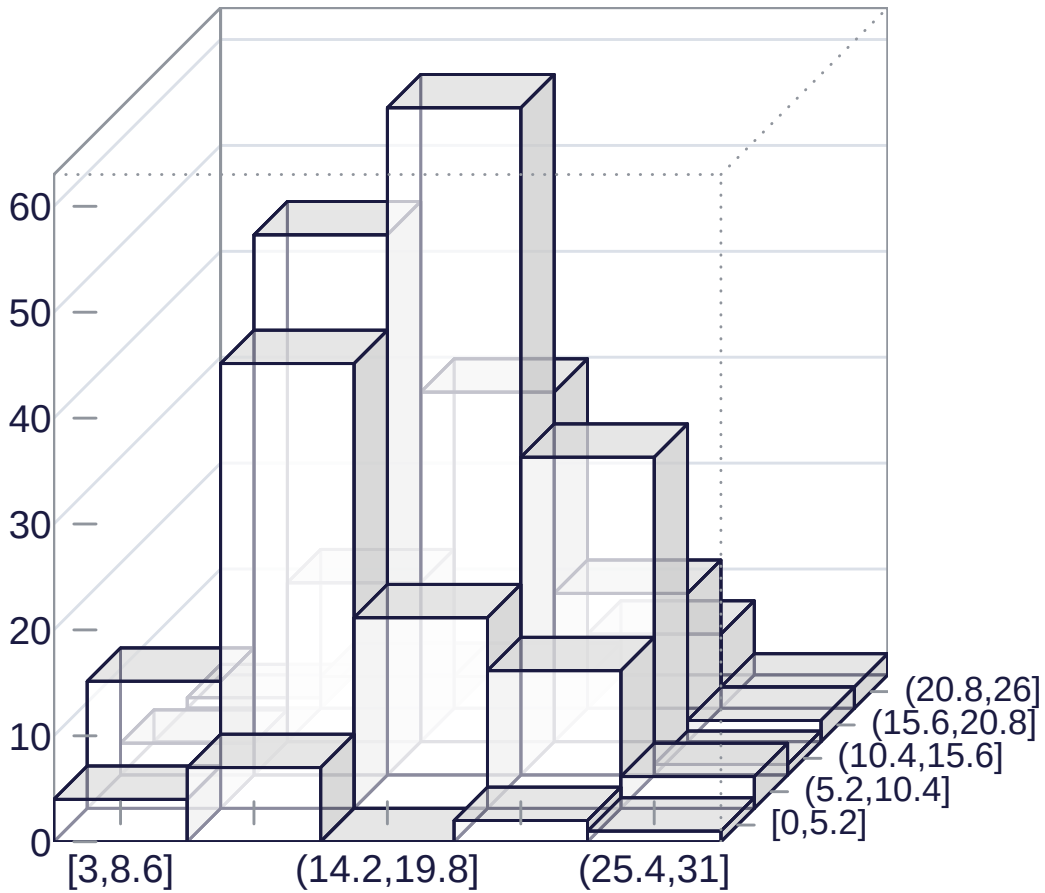
- Ein bivariates Histogramm ist die Generalisierung eines Histogramms für zwei Dimensionen.
- Das Aussehen eines bivariaten Histogramms hängt stark von der Wahl der Rechteckklassen ab.

Bivariate Histogramme

Erzeugung eines einfachen bivariaten Histogramms mithilfe der Pakete **gplots** und **epade**

```
library(gplots)                                  # hist2d Funktionalitaet
library(epade)                                   # bar3d Funktionalitaet
x      = affect$TA1                              # x_1,...,x_n
y      = affect$TA2                              # y_1, ..., y_n
h      = hist2d(x,y,nbins = c(5,5),show = F)     # Bivariate Haeufigkeitenbestimmung
dev.new()                                        # Abbildungsinitialisierung
par(                                             # Abbildungsparameter
pty    = "s",                                    # Square Abbildung
bty    = "l",                                    # Plot Box L
xaxs   = "i",                                    # "internal" (tight) x Achsenstil
yaxs   = "i")                                    # "internal" (tight) y Achsenstil
bar3d.ade(                                       # 3D Blockvisualisierung
h$counts,                                        # Tabelle der h_{ij} Werte
col    = "white",                                # Blockfarbe
wall   = 2,                                      # Dekorationsstil
xw     = 1)                                      # x Blockbreite
```

Bivariate Histogramme



Stichprobenkovarianz

Definition (Empirische Stichprobenkovarianz)

$x = ((x_1, y_1), ..., (x_n, y_n))$ sei ein bivariater Datensatz. Dann heißt die Zahl

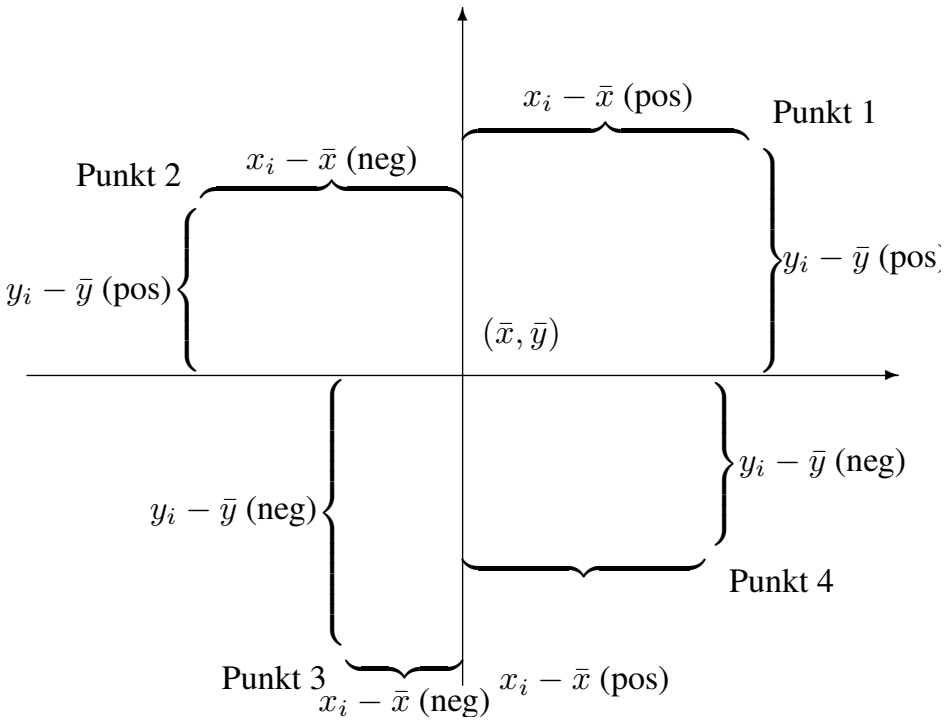
$$C := \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \tag{18}$$

empirische Stichprobenkovarianz von $x_1, ..., x_n$ und $y_1, ..., y_n$.

Bemerkungen

- $\bar{x}$ und $\bar{y}$ bezeichnen die Mittelwerte der $x_1, ..., x_n$ und $y_1, ..., y_n$, respektive.
- Der Faktor $1/n$ normiert für die Stichprobengröße.
- Eine Intuition vermittelt untenstehende Abbildung (Fahrmeir et al., 2016, Kapitel 3.4)

Stichprobenkovarianz



	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$
Punkt 1 (1.Quadrant)	positiv	positiv	positiv
Punkt 2 (2.Quadrant)	negativ	positiv	negativ
Punkt 3 (3.Quadrant)	negativ	negativ	positiv
Punkt 4 (4.Quadrant)	positiv	negativ	negativ

Abbildung 3.10 und Tabelle 3.9 aus Fahrmeir et al. (2016, Kapitel 3.4)

Pearson's Stichprobenkorrelationskoeffizient

Definition (Pearson's Stichprobenkorrelationskoeffizient)

$x = ((x_1, y_1), ..., (x_n, y_n))$ sei ein bivariater Datensatz, C sei die empirische Stichprobenvarianz von $x_1, ..., x_n$ und $y_1, ..., y_n$ und S_x bzw. S_y seien die empirischen Stichprobenstandardabweichungen von $x_1, ..., x_n$ bwz. $y_1, ..., y_n$. Dann heißt die Zahl

$$r := \frac{C}{S_x S_y} \tag{19}$$

Pearson's Stichprobenkorrelationskoeffizient oder *empirischer Korrelationskoeffizient*.

Bemerkungen

- Es gilt $-1 \leq r \leq 1$
- r misst die Stärke des positive oder negativen linearen Zusammenhangs von x und y.
- Eine Intuition vermittelt untenstehende Abbildung (Fahrmeir et al., 2016, Kapitel 3.4)
- Korrelationstärken werden in etwa nach

$$|r| < 0.5,\ 0.5 \leq |r| \leq 0.8,\ |r| > 0.8 \tag{20}$$

als "schwache", "mittelere",oder "starke" Korrelation bezeichnet

Pearson's Stichprobenkorrelationskoeffizient

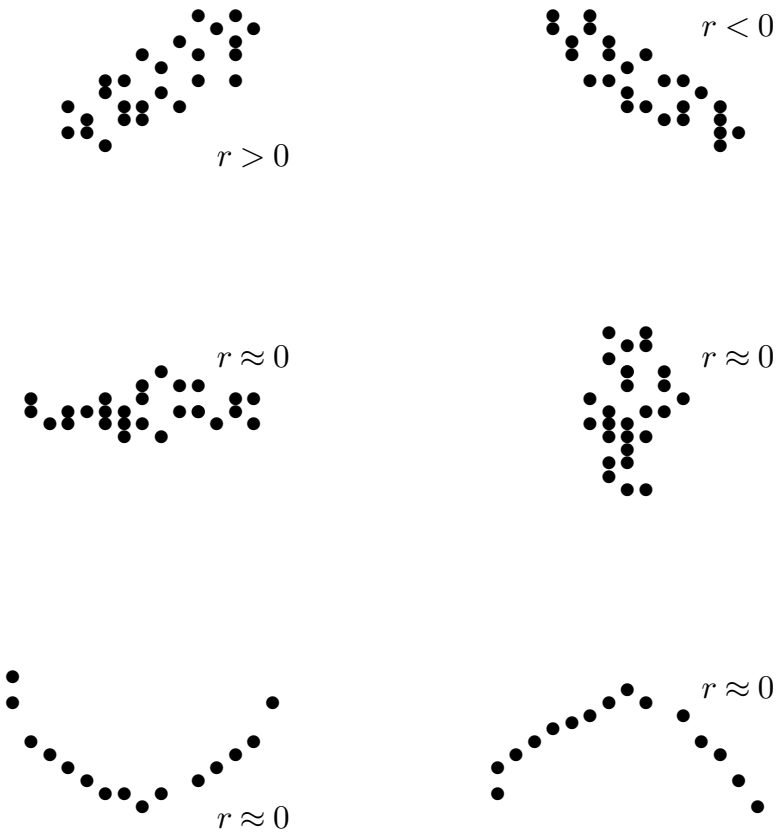


Abbildung 3.11 aus Fahrmeir et al. (2016, Kapitel 3.4)

Pearson's Stichprobenkorrelationskoeffizient

```
# Bivariater Datensatz
x      = affect$TA1                                    # x_1,..., x_n
y      = affect$TA2                                    # y_1,..., y_n

"manuelle" Berechnung
n      = length(x)                                      # Anzahl Datenpunkte
x_bar  = mean(x)                                        # \bar{x}
y_bar  = mean(y)                                        # \bar{y}
c_xy   = (1/n)*sum((x - mean(x))*(y - mean(y)))         # c
s_x    = sqrt((1/n)*sum((x - mean(x))^2))               # s_x
s_y    = sqrt((1/n)*sum((y - mean(y))^2))               # s_y
r_m    = c_xy/(s_x*s_y)                                 # "manuelles" r
[1] 0.3117369

# "automatische" Berechnung
r_a    = cor(x,y, method = "pearson")                   # "automatisches" r
[1] 0.3117369
```

Univariate Deskriptivstatistiken

- Verteilungsdarstellung
- Maße der zentralen Tendenz
- Maße der Datenvariabilität
- **Übungen und Selbstkontrollfragen**

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem R Skript.

2. x sei ein als double vector vorliegender univariater Datensatz, z.B. x = affect$TA2.

   - Berechnen Sie Minimum, Maximum, Median, und Interquartilsabstand von x.
   - Erzeugen Sie einen Boxplot von x
   - Berechnen Sie Mittelwert, Median, Varianz, und Standardabweichung von x.
   - Erzeugen Sie ein Histogramm von x.
   - Visualisieren Sie die empirische Verteilungsfunktion von x.

3. x und y seien zwei gleich große Datensätze, z.B. x = affect$ext und y = affect$neur.

   - Stellen Sie x und y in einem Streudiagramm dar.
   - Berechnen Sie die empirische Kovarianz von x und y.
   - Berechnen Sie Pearson's Stichprobenkorrelationskoeffizienten zu x und y.

Literatur

Fahrmeir, L., Heumann, C., Künstler, R., Pigeot, I., and Tutz, G. (2016). *Statistik*. Springer-Lehrbuch. Springer Berlin Heidelberg, Berlin, Heidelberg.

Henze, N. (2018). *Stochastik für Einsteiger*. Springer Fachmedien Wiesbaden, Wiesbaden.

Hyndman, R. J. and Fan, Y. (1996). Sample Quantiles in Statistical Packages. *The American Statistician*, 50(4):361.

McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of Box Plots. *The American Statistician*, 32(1):12.

Rafaeli, E. and Revelle, W. (2006). A premature consensus: Are happiness and sadness truly opposite affects? *Motivation and Emotion*, 30(1):1–12.

Scott, D. W. (1979). On optimal and data-based histograms. page 6.

Sturges, H. A. (1926). The Choice of a Class Interval. *Journal of the American Statistical Association*, 21(153):65–66.

Thayer, R. E. (1986). Activation-Deactivation Adjective Check List: Current Overview and Structural Analysis. *Psychological Reports*, 58(2):607–614.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(7) Kontrollstruktur und Schleifen

Literaturhinweise

Wir folgen Cotton (2013), Kapitel 8 und Kapitel 9, sowie Wickham (2019), Kapitel 5.

Überblick

Perse wird Programmiercode streng sequentiell Befehl für Befehl ausgeführt.

Manchmal möchte man von dieser rein sequentiellen Befehlsreihenfolge abweichen.

Die prinzipiellen Werkzeuge dafür sind **Kontrollstrukturen** und **Schleifen**

- Kontrollstrukturen bedingen die Ausführung von Befehlen
    - R bietet **if()** und **switch()** als Kontrollstrukturen
- Schleifen erlauben die adaptive Wiederholung von Codeabschnitten
    - R bietet **for()**, **while()**, und **repeat()** Loops

Kontrollstruktur und Schleifen

- if-statements
- for-loops
- Übungen und Selbstkontrollaufgaben

Kontrollstruktur und Schleifen

- **if-statements**
- for-loops
- Übungen und Selbstkontrollaufgaben

if-statements

```
if (Bedingung){
  TrueAktion
}
```

Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` evaluiert.

Wenn `Bedingung` `FALSE` ist, wird `TrueAktion` nicht evaluiert.

if-else-statements

```
if (Bedingung){
  TrueAktion
} else {
  FalseAktion
}
```

Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` evaluiert.

Wenn `Bedingung` `FALSE` ist, wird `FalseAktion` evaluiert.

Beispiele

```
x = 1
if(x > 0){
  print("x ist größer als 0")
}
```

```
> [1] "x ist größer als 0"
```

```
y = 1
if(y > 0){
  print("y ist größer als 0")
} else{
  print("y ist nicht größer als 0")
}
```

```
> [1] "y ist größer als 0"
```

Logische Operatoren in R

Die Boolesche Algebra und R kennen zwei logische Werte: TRUE und FALSE

Bei Auswertung von Relationsoperatoren ergeben sich logische Werte

Relationsoperator	Bedeutung
==	Gleich
!=	Ungleich
<, >	Kleiner, Größer
<=, >=	Kleiner gleich, Größer gleich
|	ODER
&	UND

<, <=, >, >= werden zumeist auf numerische Werte angewendet.

==, != werden zumeist auf beliebige Datenstrukturen angewendet.

| und & werden zumeist auf logische Werte angewendet.

Die Funktion xor() implementiert das exklusive ODER.

Beispiele

```
x = 1
y = 2
```

```
# Test auf Gleichheit
if(x == y){
  print("x ist gleich y")
} else {
  print("x ist ungleich y")
}
```

```
> [1] "x ist ungleich y"
```

```
# Test auf Ungleichheit
if(x < y){
  print("x ist kleiner als y")
} else {
  print("x ist gfößer oder gleich y")
}
```

```
> [1] "x ist kleiner als y"
```

Beispiele

```
x = 2
y = 2
```

```
# logisches UND/ODER
if(x == y | x < y){
  print("x ist kleiner oder gleich y")
} else {
  print("x ist größer als y")
}
```

```
> [1] "x ist kleiner oder gleich y"
```

```
# logisches UND
if(x > y & x != y){
  print("x ist größer als y")
} else {
  print("x ist kleiner oder gleich y")
}
```

```
> [1] "x ist kleiner oder gleich y"
```

Fehlerhafte Bedingungen

Die Bedingung sollte einem einzigen logischen Wert entsprechen

```
if("x") 1
```

```
> Error in if ("x") 1: Argument kann nicht als logischer Wert
> interpretiert werden
```

```
if(NA) 1
```

```
> Error in if (NA) 1: Fehlender Wert, wo TRUE/FALSE nötig ist
```

```
if(c(T,F)) 1
```

```
> Warning in if (c(T, F)) 1: Bedingung hat Länge > 1 und nur das erste
> Element wird benutzt
```

```
> [1] 1
```

Motivation

Kombinierte if-else Statements werden leicht unübersichtlich

```
x = 2
if (x == 1){
  print("Aktion 1")
} else if(x == 2){
  print("Aktion 2")
} else if(x == 3){
  print("Aktion 3")
} else if(x == 4){
  print("Aktion 4")
}
```

```
> [1] "Aktion 2"
```

switch-statement mit Integervariable

Bei switch Variable *i* vom Typ Integer wird die *i*te Aktion ausgeführt.

```
x = 2                                                    # switch Variable
switch(x,                                                # x = 2
print("Aktion 1"),                                       # 1. Aktion
print("Aktion 2"),                                       # 2. Aktion
print("Aktion 3"),                                       # 3. Aktion
print("Aktion 4"))                                       # 4. Aktion
```

```
> [1] "Aktion 2"
```

switch-statement mit Charaktervariable

Bei switch Variable **x** vom Typ Character wird die Aktion mit Namen **x** ausgeführt.

```
x = "a"                                          # switch Variable
switch(x,                                        # x = 2
a = print("Aktion a"),                           # a Aktion
b = print("Aktion b"),                           # b Aktion
c = print("Aktion c"),                           # c Aktion
d = print("Aktion d"))                           # d Aktion
```

```
> [1] "Aktion a"
```

if vs. switch-statements

- Jedes if-statement kann als switch-statement formuliert werden
- Jedes switch-statement kann als if-statement formuliert werden
- if-statements sind in der Anwendung häufiger
- if-else-statements sollten nicht zu komplex designed werden
- Bei komplexe if-else-statements kann sich eine Formulierung als switch lohnen

Kontrollstruktur und Schleifen

- if-statements
- **for-loops**
- Übungen und Selbstkontrollaufgaben

for-loops

Generelle Form

```
for (item in vector) perform_action
```

- `perform_action` wird für jedes `item` in `vector` einmal evaluiert
- Der Wert von `item` wird jeweils aktualisiert.

Beispiel

```
for (i in 1:3){
  print(i)
}
```

```
> [1] 1
> [1] 2
> [1] 3
```

Simulationsbeispiele

Typischerweise wird innerhalb eines for-loops etwas erzeugt und gespeichert

- Die entsprechende Speicherstruktur sollte vorallokiert werden
- Der entsprechende Arbeitsspeicher ist schon bereitgestellt (Speed)
- Fehler können leichter detektiert werden (NaNs)

```
ns    = 3                          # Anzahl an Simulationen
X_bar = rep(NaN, ns)               # Speicherstruktur

# Simulationsiterationen
for(i in 1:ns){                    # Iterationsindices sind typischerweise i,j,k
  X         = rnorm(12)            # Realisierung von 12 Z-Variablen
  X_bar[i]  = mean(X)              # Mittelwert der i-ten Realisierung
  print(X_bar)                     # Anzeige zur Demonstration
}
```

```
> [1] -0.362    NaN    NaN
> [1] -0.362 -0.133    NaN
> [1] -0.362 -0.133  0.121
```

Simulationsbeispiele

Typischerweise ändern sich innerhalb eines for-loops Parameter

- `seq_along` sollte zur linearen Indizierung genutzt werden

```
mu    = c(0,5,50)                    # Drei Erwartungswertparameter
X_bar = rep(NaN, ns)                 # Speicherstruktur

# Simulationsiterationen
for(i in seq_along(mu)){             # Iterationsindices sind typischerweise i,j,k
  X         = rnorm(12, mu[i], 1)    # Realisierung von 12 N(\mu,1) Varialben
  X_bar[i]  = mean(X)                # Mittelwert der i-ten Realisierung
  print(X_bar)                       # Anzeige zur Demonstration
}
```

```
> [1] 0.127   NaN   NaN
> [1] 0.127 4.957   NaN
> [1]  0.127  4.957 49.827
```

Simulationsbeispiele

for-loops können auch geschachtelt werden

Für **jede** Iteration der äußeren Schleife werden **alle** Iterationen der inneren Schleife evaluiert

```
n_i = 2                                 # Anzahl der Iterationen äußere Schleife
n_j = 3                                 # Anzahl der Iterationen innere Schleife
for(i in 1:n_i){                        # Äußere Schleife
  for (j in 1:n_j){                     # Innere Schleife
    print(c(i,j))                       # Anzeigen der Iternationsindices [i,j]
    }
}
```

```
> [1] 1 1
> [1] 1 2
> [1] 1 3
> [1] 2 1
> [1] 2 2
> [1] 2 3
```

Beispiel

Definition (Schätzerkonsistenz)

$X_1, ..., X_n \sim p_\theta$ sei die Stichprobe eines parametrischen statistischen Produktmodells $\mathcal{M}$ und $\hat{\tau}_n$ sei ein Schätzer von τ. Eine Folge von Schätzern $\hat{\tau}_1, \hat{\tau}_2, \ldots$ wird dann eine *konsistente Folge von Schätzern* genannt, wenn für jedes $\epsilon > 0$ und jedes $\theta \in \Theta$ gilt, dass

$$\lim_{n \to \infty} \mathbb{P}_\theta \left(|\hat{\tau}_n(X) - \tau(\theta)| \geq \epsilon \right) = 0.$$

Wenn $\hat{\tau}_1, \hat{\tau}_2, \ldots$ eine konsistente Folge von Schätzern ist, dann heißt $\hat{\tau}_n$ *konsistenter Schätzer*.

$\mathbb{P}_\theta$ hängt von ϵ und n ab.

Beispiel

```
# Definitionen
mu      = 1                                          # w.a.u. Erwartungswertparameter
sigsqr  = 2                                          # w.a.u. Varianzparameter
n       = seq(1,1e3,by = 10)                         # Stichprobengröße n
eps     = c(0.15, 0.10, 0.05)                        # \epsilon Werte
ne      = length(eps)                                # Anzahl \epsilon Werte
nn      = length(n)                                  # Anzahl Stichprobengrößen
ns      = 1e1                                        # Anzahl der Simulationen
E       = array(rep(NaN,nn,ne,ns), dim = c(nn,ne,ns))  # Ereignisindikator Array

# Simulationen
for(e in seq_along(eps)){                            # \epsilon Wert Iterationen
  for(i in seq_along(n)){                            # Stichprobengrößeniterationen
    for(s in 1:ns){                                  # Simulationsiterationen
      x   = rnorm(n[i], mu, sqrt(sigsqr))            # Stichprobenrealisationen
      if(abs(mean(x) - mu) >= eps[e]){               # |X_bar - \mu)| \ge \epsilon
                E[i,e,s] = 1                         # Ereignisindikator
      } else {                                       # |X_bar - \mu)| < \epsilon
                E[i,e,s] = 0                         # Ereignisindikator
            }
        }
    }
}
```

E kann zum Schätzen von $\mathbb{P}_\theta$ gemittelt werden.

for-loop Alternativen

while-loops iterieren Codeabschnitte basierend auf einer Bedingung

```
while(condition){
  TrueAktion          # TrueAktion wird ausgeführt, solange condition == TRUE
}
```

repeat-loops wiederholen Codeabschnitte bis zu einem 'break' Befehl

```
repeat{
  Aktion              # Aktion wird ausgeführt, bis ein break Befehl evaluiert wird
}
```

for-loops können als while-loops reformuliert werden, while-loops als repeat-loops

Generell wird in R die weniger flexible **apply()** Funktionalität bevorzugt.

Kontrollstruktur und Schleifen

- if-statements
- for-loops
- **Übungen und Selbstkontrollaufgaben**

Übungen und Selbstkontrollaufgaben

1. Erzeugen Sie mithilfe eines for-loops eine 4 x 5 Matrix, deren Elemente gleich den Spaltenindizes sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    2    3    4    5
> [2,]    1    2    3    4    5
> [3,]    1    2    3    4    5
> [4,]    1    2    3    4    5
```

2. Erzeugen Sie mithilfe eines for-loops eine 4 x 5 Matrix, deren Elemente gleich den Zeilenindizes sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    1    1    1    1
> [2,]    2    2    2    2    2
> [3,]    3    3    3    3    3
> [4,]    4    4    4    4    4
```

3. Erzeugen Sie mithilfe zweier geschachtelter for-loops und eines if-statements eine 4 x 5 Matrix, deren Elemente gleich den Spaltenindizes sind, wenn der jeweilige Spaltenindex größer oder gleich dem Zeilenindex ist, und deren Elemente ansonsten Null sind. Die Matrix sollte wie folgt aussehen:

```
>      [,1] [,2] [,3] [,4] [,5]
> [1,]    1    2    3    4    5
> [2,]    0    2    3    4    5
> [3,]    0    0    3    4    5
> [4,]    0    0    0    4    5
```

References

Cotton, Richard. 2013. *Learning R*. First Edition. Beijing ; Sebastopol, CA: O'Reilly.

Wickham, Hadley. 2019. *Advanced R, Second Edition*. CRC Press.

Computergestützte Datenanalyse

BSc Psychologie SoSe 2021

Prof. Dr. Dirk Ostwald

(8) Anwendungsbeispiel

Der Affect Beispieldatensatz

- Daten zweier Studien ("maps" und "flat") des Personality, Motivation, and Cognition Laboratory mit $n = 330$ Versuchspersonen, Teil des psychTools R Pakets
- Fragebogenbasierte Messung von Affektvariablen vor und nach Anschauen eines Filmclips
- Filmclipbedingungen
  (1) Frontline, Dokumentation über die Befreiung des Konzentrationslagers Bergen-Belsen
  (2) Halloween, ein Horrorfilm
  (3) National Geographic, eine Naturdokumentation über die Serengeti
  (4) Parenthood, eine Komödie
- Messung von Tense Arousal (TA), Energetic Arousal (EA), Positive Affect (PA) und Negative Affect (NA) vor (1) und nach (2) Anschauen des Filmclips
- Erhebung von Daten mit fünf Skalen des Eysenck Persönlichkeitsinventar sowie State- und Trait-Anxiety Items. In der "maps" Studie außerdem Erhebung des BDI
- Für Details, siehe Rafaeli and Revelle (2006)

Der Affect Beispieldatensatz

```
install.packages("psychTools")     # einmalige Installation des psychTools Pakets
library(psychTools)                # Laden des psychTools Pakets
?affect                            # Erlaeuterung des affect Datensatzes
data(affect)                       # Laden des affect Dataframes
View(affect)                       # Inspektion des affect Dataframes
```

	Study	Film	ext	neur	imp	soc	lie	traitanx	state1	EA1	TA1	PA1	NA1	EA2	TA2	PA2	NA2	state2	MEQ	BDI
1	maps	3	18.0	9.0	7.0	10.0	3.0	24.0	22.0	24.0	14.0	26.0	2.0	6.0	5.0	7.0	4.0	NA	NA	0.04761905
2	maps	3	16.0	12.0	5.0	8.0	1.0	41.0	40.0	9.0	13.0	10.0	4.0	4.0	14.0	5.0	5.0	NA	NA	0.33333333
3	maps	3	6.0	5.0	3.0	1.0	2.0	37.0	44.0	1.0	14.0	4.0	2.0	2.0	15.0	3.0	1.0	NA	NA	0.19047619
4	maps	3	12.0	15.0	4.0	6.0	3.0	54.0	40.0	5.0	15.0	1.0	0.0	4.0	15.0	0.0	2.0	NA	NA	0.38461538
5	maps	3	14.0	2.0	5.0	6.0	3.0	39.0	67.0	12.0	20.0	7.0	13.0	14.0	15.0	16.0	13.0	NA	NA	0.38095238
6	maps	1	6.0	15.0	2.0	4.0	5.0	51.0	38.0	9.0	14.0	5.0	1.0	7.0	12.0	2.0	2.0	NA	NA	0.23809524
7	maps	1	15.0	12.0	4.0	9.0	3.0	40.0	32.0	1.0	5.0	7.0	0.0	13.0	14.0	8.0	8.0	NA	NA	0.30769231
8	maps	2	18.0	10.0	7.0	9.0	2.0	32.0	41.0	17.0	11.0	10.0	1.0	19.0	15.0	16.0	0.0	NA	NA	0.00000000
9	maps	2	15.0	1.0	3.0	11.0	3.0	22.0	26.0	19.0	5.0	14.0	0.0	19.0	6.0	14.0	0.0	NA	NA	0.00000000
10	maps	2	8.0	10.0	2.0	5.0	2.0	35.0	31.0	15.0	8.0	7.0	0.0	28.0	19.0	11.0	2.0	NA	NA	0.33333333
11	maps	1	13.0	9.0	3.0	9.0	3.0	43.0	39.0	14.0	13.0	10.0	2.0	9.0	21.0	3.0	7.0	NA	NA	0.38095238
12	maps	4	14.0	1.0	3.0	12.0	6.0	33.0	25.0	24.0	15.0	23.0	2.0	27.0	11.0	29.0	0.0	NA	NA	0.14285714
13	maps	4	15.0	2.0	4.0	10.0	5.0	23.0	32.0	7.0	14.0	1.0	0.0	11.0	17.0	4.0	0.0	NA	NA	0.00000000
14	maps	4	19.0	3.0	7.0	11.0	0.0	23.0	23.0	21.0	13.0	20.0	1.0	23.0	18.0	21.0	2.0	NA	NA	0.00000000
15	maps	1	15.0	7.0	4.0	10.0	2.0	27.0	28.0	22.0	15.0	16.0	1.0	16.0	18.0	12.0	4.0	NA	NA	0.04761905
16	maps	1	11.0	13.0	6.0	5.0	7.0	45.0	28.0	2.0	0.0	5.0	0.0	23.0	26.0	21.0	9.0	NA	NA	0.19047619
17	maps	1	16.0	18.0	5.0	10.0	0.0	58.0	56.0	3.0	11.0	3.0	7.0	8.0	17.0	4.0	18.0	NA	NA	0.66666667
18	maps	1	17.0	11.0	6.0	11.0	4.0	39.0	44.0	19.0	18.0	23.0	1.0	21.0	20.0	21.0	6.0	NA	NA	0.33333333
19	maps	1	7.0	10.0	2.0	4.0	1.0	43.0	56.0	14.0	21.0	17.0	8.0	18.0	22.0	12.0	12.0	NA	NA	0.42857143
20	maps	1	13.0	12.0	4.0	8.0	4.0	38.0	35.0	9.0	6.0	1.0	0.0	11.0	10.0	8.0	1.0	NA	NA	0.00000000
21	maps	2	14.0	3.0	5.0	7.0	3.0	35.0	28.0	18.0	8.0	12.0	0.0	16.0	17.0	7.0	2.0	NA	NA	0.04761905
22	maps	2	11.0	10.0	3.0	7.0	1.0	37.0	47.0	11.0	12.0	5.0	1.0	16.0	22.0	12.0	4.0	NA	NA	0.52380952
23	maps	2	20.0	10.0	7.0	10.0	2.0	43.0	43.0	5.0	10.0	2.0	1.0	10.0	25.0	2.0	6.0	NA	NA	0.04761905

Tense Arousal

- *Activation-Deactivation Adjective Check List* nach Thayer (1986).
- Circa 25 Adjektive mit Vierpunkteskala
  - Wie gut beschreibt folgendes [Adjektiv] Ihre momentane Gefühlslage?
  - "I (1) do not feel, (2) cannot decide, (3) feel slightly, (4) definitely feel [Adjektiv]"
- Hohe TA Werte $\rightarrow$ Hohe Selbsteinschätzung bei *tense, clutched-up, fearful, jittery, intense, ...*
- Niedrige TA Werte $\rightarrow$ Hohe Selbsteinschätzung bei *calm, still, at-rest, quiet, placid, ...*

Wir wollen im Folgenden exemplarisch untersuchen, ob die Filmclipbedingung Einfluss auf die Tense Arousal Selbsteinschätzung hatte.

Anwendungsbeispiel

- Exploration und Deskription
- T-Tests
- Einfaktorielle Varianzanalyse
- Zweifaktorielle Varianzanalyse
- Übungen und Selbstkontrollaufgaben

Anwendungsbeispiel

- **Exploration und Deskription**
- T-Tests
- Einfaktorielle Varianzanalyse
- Zweifaktorielle Varianzanalyse
- Übungen und Selbstkontrollaufgaben

Visualisierung der Rohdatenverteilungen

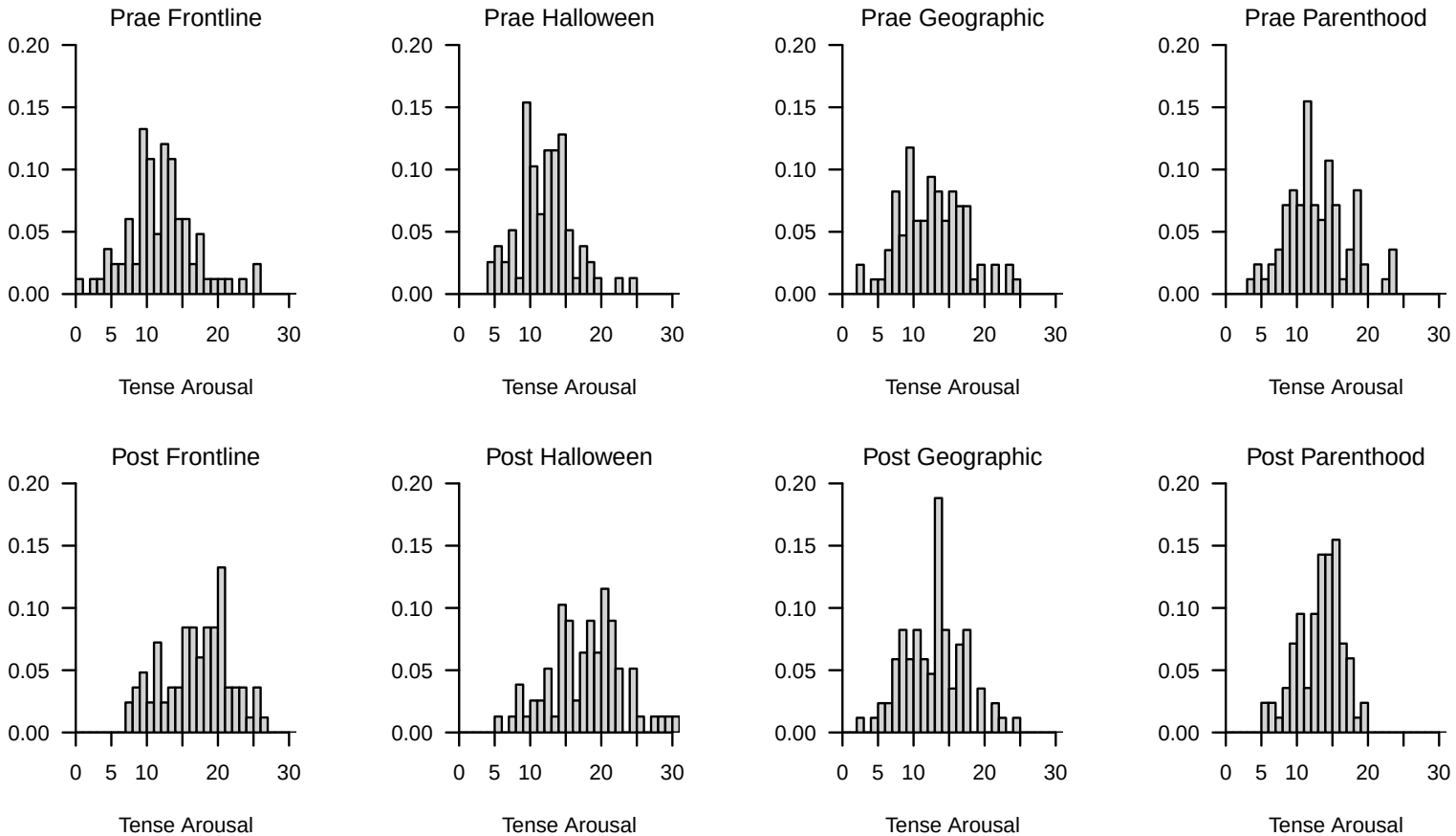
```
# Histogrammparameter
TA_min      = min(min(affect$TA1),          # kleinster TA Wert
                  min(affect$TA2))
TA_max      = max(max(affect$TA1),          # kleinster TA Wert
                  max(affect$TA2))
h           = 1                             # gewünschte Klassenbreite
b_0         = TA_min                        # b_0
b_k         = TA_max                        # b_k
k           = ceiling((b_k - b_0)/h)        # Anzahl der Klassen
b           = seq(b_0, b_k, by = h)         # Klassen [b_{j-1}, b_j[
ylimits     = c(0,.2)                       # y-Achsenlimits
xlimits     = c(0,31)                       # x-Achsenlimits
film        = c("Frontline",                # Filmbedingungen
                "Halloween",
                "Geographic",
                "Parenthood")
```

Visualisierung der Rohdatenverteilungen (fortgeführt)

```
# Abbildungsparameter
par(                                    # für Details siehe ?par
mfcol       = c(2,4),                   # 2 x 4 Panelstruktur
family      = "sans",                   # Serif-freier Fonttyp
pty         = "m",                      # Maximale Abbildungsregion
bty         = "l",                      # L förmige Box
lwd         = 1,                        # Liniendicke
las         = 1,                        # Horizontale Achsenbeschriftung
xaxs        = "i",                      # x-Achse bei y = 0
yaxs        = "i",                      # y-Achse bei x = 0
font.main   = 1,                        # Non-Bold Titel
cex         = .6,                       # Textvergrößerungsfaktor
cex.main    = 1.1)                      # Titeltextvergrößerungsfaktor

# Iteration über Filmclipbedingungen
for(i in 1:4){

  # Prä-Filmclipexposition TA Werte
  hist(
  affect$TA1[affect$Film == i],         # TA Werte von Filmclipbedingung i
  breaks    = b,                        # Histogrammklassen
  freq      = F,                        # normierte relative Häufigkeit
  xlim      = xlimits,                  # x-Achsenlimits
  ylim      = ylimits,                  # y-Achsenlimits
  xlab      = "Tense Arousal",          # x-Achsenbeschriftung
  ylab      = "",                       # y-Achsenbeschriftung
  main      = paste("Prae", film[i]))   # Titelbeschriftung

  # Post-Filmclipexposition TA Werte
  hist(
  affect$TA2[affect$Film == i],         # TA Werte von Filmclipbedingung i
  breaks    = b,                        # Histogrammklassen
  freq      = F,                        # normierte relative Häufigkeit
  xlim      = xlimits,                  # x-Achsenlimits
  ylim      = ylimits,                  # y-Achsenlimits
  xlab      = "Tense Arousal",          # x-Achsenbeschriftung
  ylab      = "",                       # y-Achsenbeschriftung
  main      = paste("Post", film[i])    # Titelbeschriftung
  )
}
```

Visualisierung der Rohdatenverteilungen (fortgeführt)

- Normalisierte relative Häufigkeiten der Tense Arousal Selbsteinschätzungen



- Visuelle Inspektion legt Erhöhung der TA Werte nach Frontline und Halloween nahe

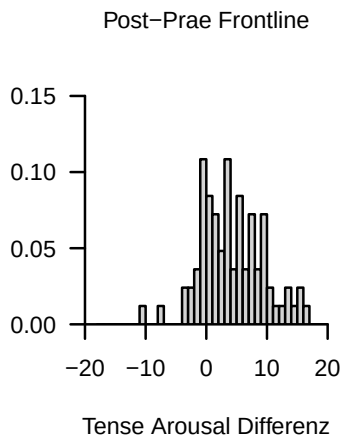
Evaluation der Post-Prae-Filmexposition Tense Arousal Wertdifferenz (DTA)

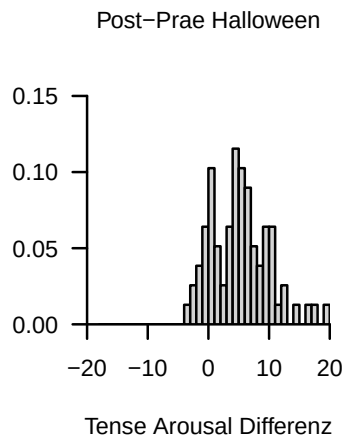
```
data(affect)                                  # Laden des affect Datensatzes

affect$DTA = affect$TA2 - affect$TA1          # Post-TA minus Prae-TA Werte
```

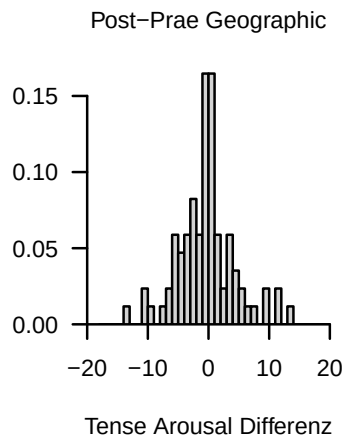
Visualisierung DTA (fortgeführt)

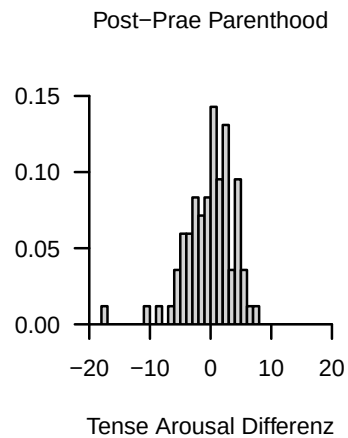
```
# Histogrammparameter
h           = 1                                 # gewünschte Klassenbreite
b_0         = min(affect$DTA)                   # b_0
b_k         = max(affect$DTA)                   # b_k
k           = ceiling((b_k - b_0)/h)            # Anzahl der Klassen
b           = seq(b_0, b_k, by = h)             # Klassen [b_{j-1}, b_j[
ylimits     = c(0,.18)                          # y-Achsenlimits
xlimits     = c(-20,20)                         # x-Achsenlimits
film        = c("Frontline" , "Halloween",      # Filmclipbedingungen
               "Geographic", "Parenthood")

# Abbildungsparameter
par(                                            # für Details siehe ?par
mfcol       = c(1,4),                           # 1 x 4 Panelstruktur
family      = "sans",                           # Serif-freier Fonttyp
pty         = "m",                              # Maximale Abbildungsregion
bty         = "l",                              # L förmige Box
las         = 1,                                # Horizontale Achsenbeschriftung
xaxs        = "i",                              # x-Achse bei y = 0
yaxs        = "i",                              # y-Achse bei x = 0
font.main   = 1,                                # Non-Bold Titel
cex         = .6,                               # Textvergrößerungsfaktor
cex.main    = 1)                                # Titeltextvergrößerungsfaktor

# Iteration über Filmclipbedingungen
for(i in 1:4){
  hist(
  affect$DTA[affect$Film == i],                 # TA Werte von Filmclipbedingung i
  breaks    = b,                                # Histogrammklassen
  freq      = F,                                # normierte relative Häufigkeit
  xlim      = xlimits,                          # x-Achsenlimits
  ylim      = ylimits,                          # y-Achsenlimits
  xlab      = "Tense Arousal Differenz",        # x-Achsenbeschriftung
  ylab      = "",                               # y-Achsenbeschriftung
  main      = paste("Post-Prae", film[i]))      # Titelbeschriftung
}
```

Visualisierung DTA (fortgeführt)

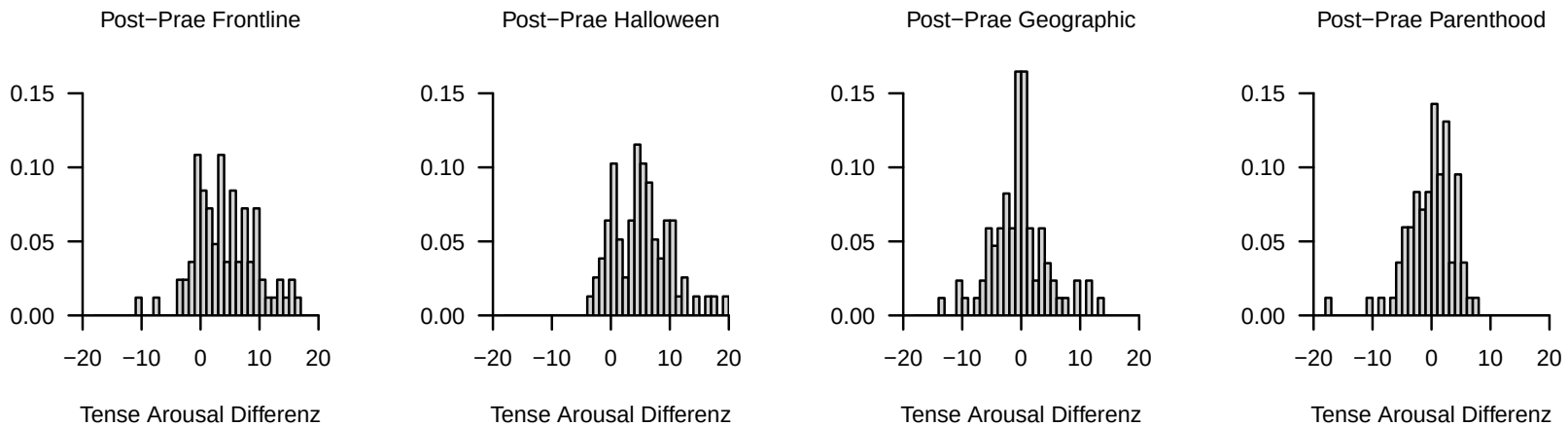








Visualisierung DTA (fortgeführt)



- Visuelle Inspektion legt Erhöhung der TA Werte nach Frontline und Halloween nahe
- Visuelle Inspektion legt leichtes Sinken der TA Werte nach Parenthood nahe

Zusammenfassung und Visualisierung der DTA Werte

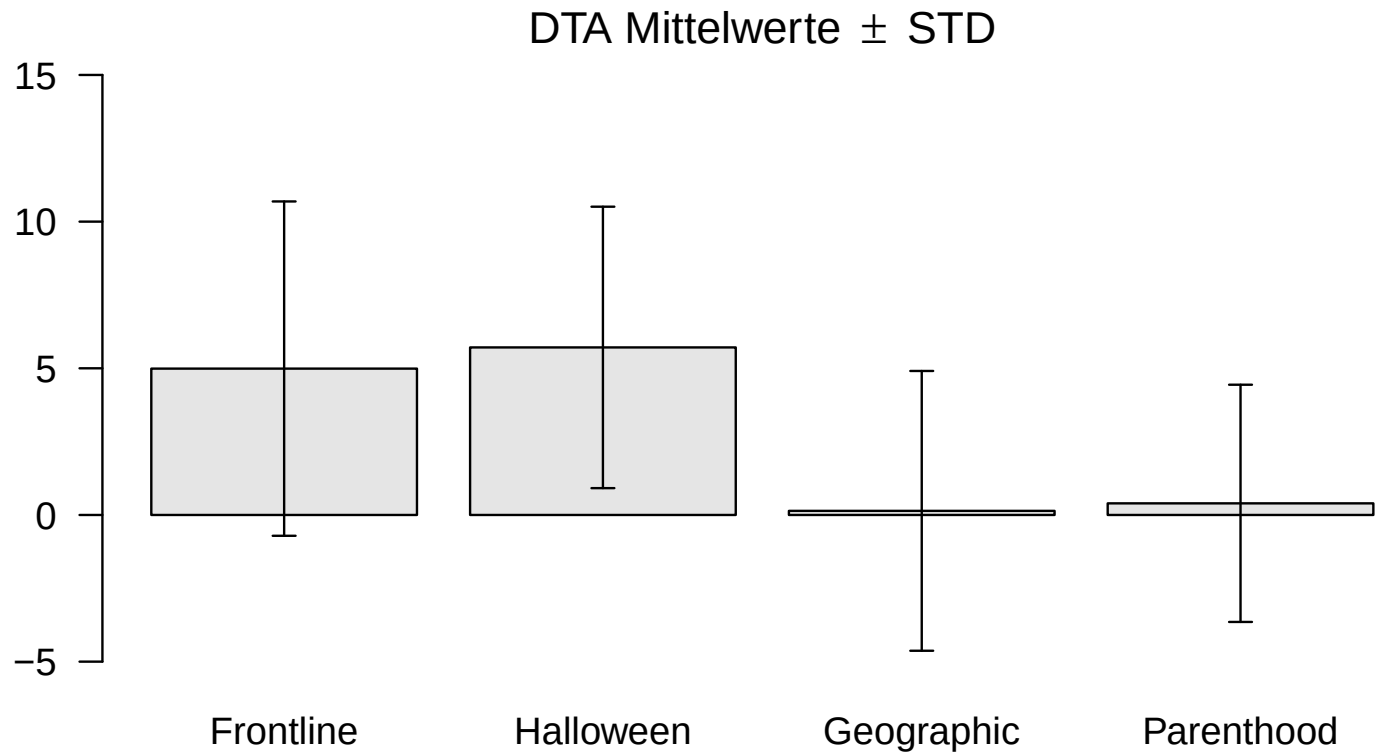
```
# Summary Statistics Dataframe Initialisierung
library(psychTools)                                    # Datensatzladen
data(affect)                                           # Datensatzladen
affect$DTA  = affect$TA2 - affect$TA1                  # Post-TA minus Prae-TA Werte
con         = 4                                        # Anzahl Filmclipbedinungen
STA         = data.frame(                              # Datafrarneerzeugung
              n         = rep(NaN,con),                # Stichprobengrößen
              Max       = rep(NaN,con),                # Maxima
              Min       = rep(NaN,con),                # Minima
              Median    = rep(NaN,con),                # Medians
              Mean      = rep(NaN,con),                # Means
              Var       = rep(NaN,con),                # Variances
              Std       = rep(NaN,con),                # Standarddeviations
              Sem       = rep(NaN,con),                # Standarderrors
              row.names = c("Frontline",               # Filmclipbedinungen
                            "Halloween",
                            "Geographic",
                            "Parenthood"))
# Iterationen über Filmclipbedingungen
for(i in 1:con){
  data             = affect$DTA[affect$Film == i]      # Daten
  STA$n[i]         = length(data)                      # Stichprobengröße
  STA$Max[i]       = max(data)                         # Maxima
  STA$Min[i]       = min(data)                         # Minima
  STA$Median[i]    = median(data)                      # Medians
  STA$Mean[i]      = mean(data)                        # Means
  STA$Var[i]       = var(data)                         # Variances
  STA$Std[i]       = sd(data)                          # Standarddeviations
  STA$Sem[i]       = sd(data)/sqrt(length(data))       # Standard errors
  }
print.AsIs(STA)                                        # Ausgabe
```

```
            n Max Min Median  Mean  Var  Std   Sem
Frontline  83  26 -10   4.00 4.989 32.5 5.70 0.626
Halloween  78  20  -3   5.25 5.712 23.0 4.80 0.543
Geographic 85  14 -13   0.00 0.140 22.7 4.77 0.517
Parenthood 84   8 -18   1.00 0.396 16.3 4.04 0.441
```

Visualisierung der DTA Mittelwerte und Standardabweichungen

```
# Abbildungsparameter
library(latex2exp)                                  # TeX Annotations
par(                                                # für Details siehe ?par
family      = "sans",                               # Serif-freier Fonttyp
pty         = "m",                                  # Maximale Abbildungsregion
bty         = "l",                                  # L förmige Box
lwd         = 1,                                    # Liniendicke
las         = 1,                                    # Horizontale Achsenbeschriftung
font.main   = 1,                                    # Non-Bold Titel
cex         = 1,                                    # Textvergrößerungsfaktor
cex.main    = 1.2)                                  # Titeltextvergrößerungsfaktor

# Balkendiagramm
x = barplot(                                        # Ausgabe der x-Ordinaten (?barplot für Details)
STA$Mean,                                           # y-Werte (DTA Mittelwerte)
ylim      = c(-5,15),                               # y-Achsenlimits
names.arg = c("Frontline",                          # x-Achsenbeschriftung
              "Halloween",
              "Geographic",
              "Parenthood"),
col       = "gray90",                               # Balkenfarbe
main      = TeX("DTA Mittelwerte $\\pm$ STD"))      # Titel

# Fehlerbalken
arrows(                                             # für Details siehe ?arrows
x0        = x,                                      # arrow start x-ordinate
y0        = STA$Mean - STA$Std,                     # arrow start y-ordinate
x1        = x,                                      # arrow end   x-ordinate
y1        = STA$Mean + STA$Std,                     # arrow end   y-ordinate
code      = 3,                                      # Pfeilspitzen beiderseits
angle     = 90,                                     # Pfeilspitzenwinkel -> Linie
length    = 0.05)                                   # Linielänge
```

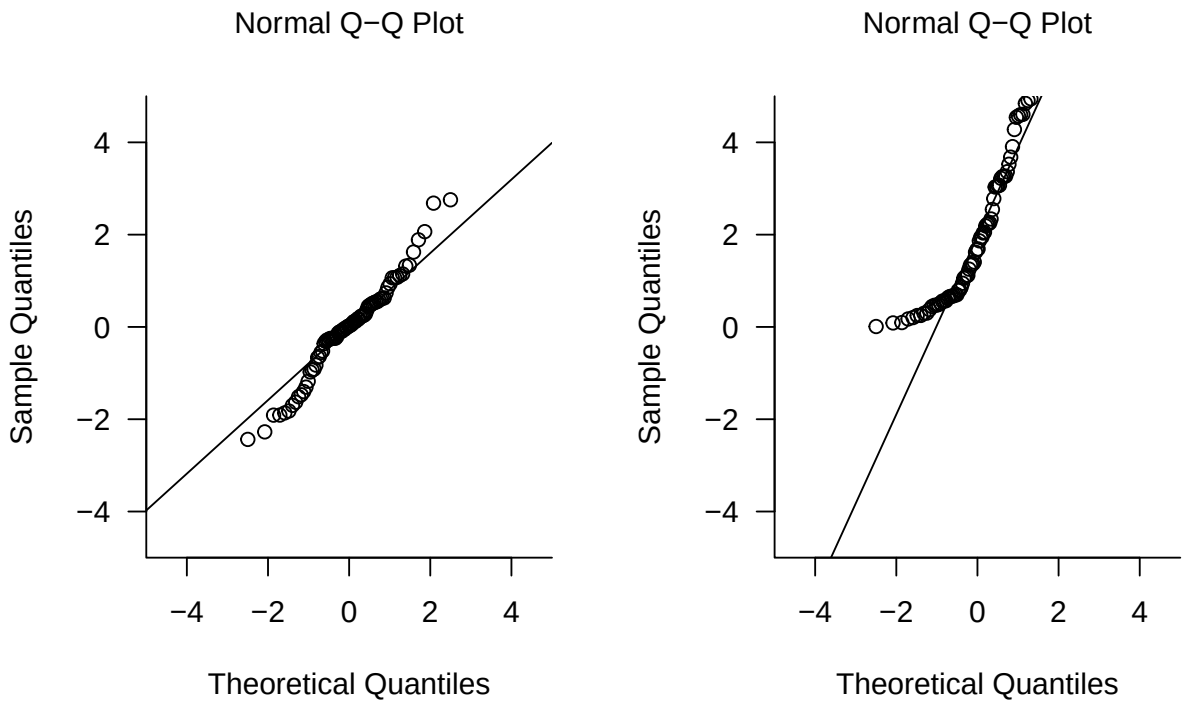
Visualisierung der DTA Mittelwerte und Standardabweichungen



Visuelle Inspektion der Normalität durch Quantil-Quantil-Plots

```
# Abbildungsparameter
par(                                    #für Details siehe ?par
mfcol       = c(1,2),                   # 1 x 4 Panelstruktur
family      = "sans",                   # Serif-freier Fonttyp
pty         = "m",                      # Maximale Abbildungsregion
bty         = "l",                      # L förmige Box
lwd         = 1,                        # Liniendicke
las         = 1,                        # Horizontale Achsenbeschriftung
xaxs        = "i",                      # x-Achse bei y = 0
yaxs        = "i",                      # y-Achse bei x = 0
font.main   = 1,                        # Non-Bold Titel
cex         = .8,                       # Textvergrößerungsfaktor
cex.main    = .8)                       # Titeltextvergrößerungsfaktor
x_norm      = rnorm(80)                 # Normalverteilungsrealisation
x_chisq     = rchisq(80,2)              # \chi^2(2) Realisation
qqnorm(x_norm)                          # Q-Q-Plot
qqline(x_norm)                          # Q-Q-Plotlinie
qqnorm(x_chisq)                         # Q-Q-Plot
qqline(x_chisq)                         # Q-Q-Plotlinie
```

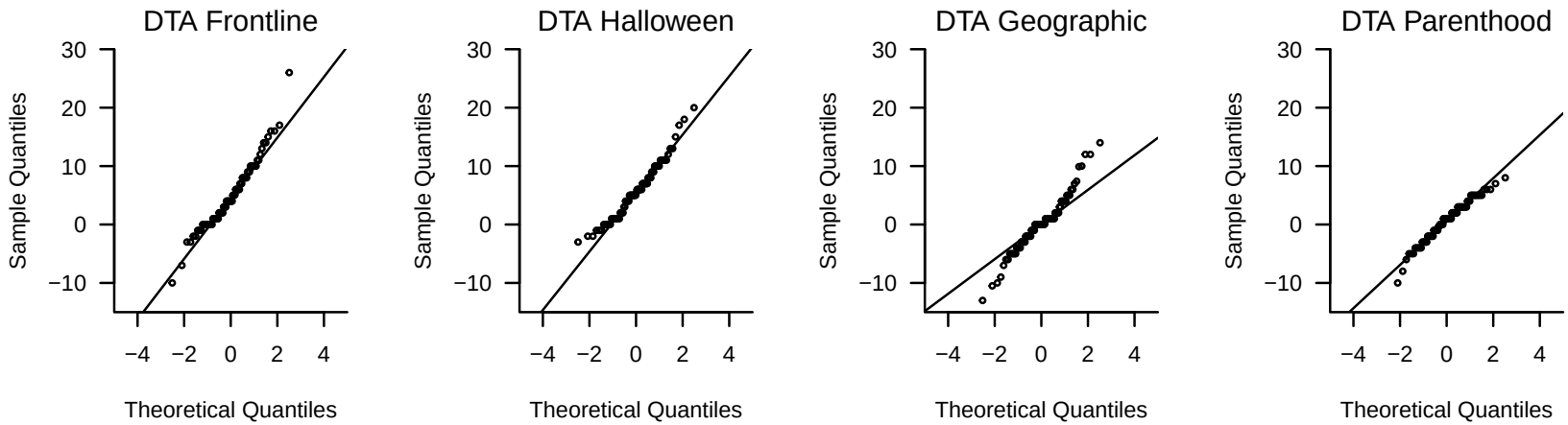
Visuelle Inspektion der Normalität durch Quantil-Quantil-Plots



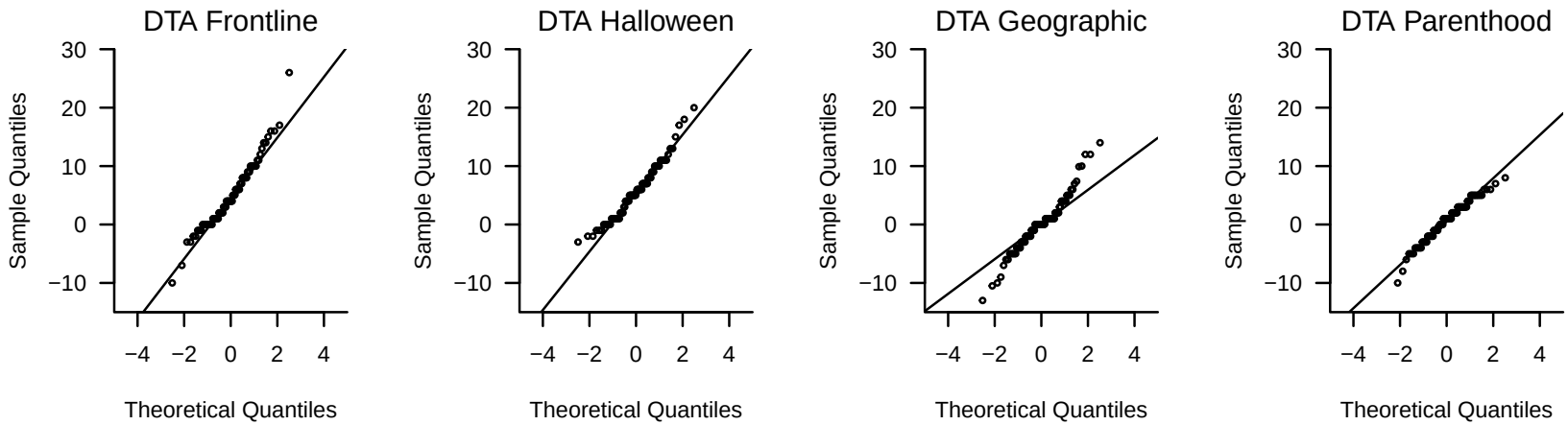
Visuelle Inspektion der Normalität durch Quantil-Quantil-Plots

```
# TA Differenzwerte
data(affect)                                    # Datensatzladen
affect$DTA  = affect$TA2 - affect$TA1           # Post-TA minus Prae-TA Werte

# Abbildungsparameter
par(                                            # Details siehe ?par
mfcol       = c(1,4),                           # 1 x 4Panelstruktur
family      = "sans",                           # Serif-freier Fonttyp
pty         = "m",                              # Maximale Abbildungsregion
bty         = "l",                              # L förmige Box
lwd         = 1,                                # Liniendicke
las         = 1,                                # Horizontale Achsenbeschriftung
xaxs        = "i",                              # x-Achse bei y = 0
yaxs        = "i",                              # y-Achse bei x = 0
font.main   = 1,                                # Non-Bold Titel
cex         = .6,                               # Textvergrößerungsfaktor
cex.main    = 1.2)                              # Titeltextvergrößerungsfaktor
film        = c("Frontline",                    # Filmbedingungen
                "Halloween",
                "Geographic",
                "Parenthood")
# Q-Q-Plots
for(i in 1:4){
  qqnorm(
  affect$DTA[affect$Film == i],                 # Daten
  cex     = .5,                                 # Punktvergrößerungsfaktor
  ylim    = c(-15,30),                          # y-Achsenlimits
  xlim    = c(-5,5),                            # x-Achsenlimits
  main    = paste("DTA", film[i]))              # Titel
  qqline(affect$DTA[affect$Film == i])          # Q-Q-Plotlinie
}
```

Visuelle Inspektion der Normalität durch Quantil-Quantil-Plots



Visuelle Inspektion der Normalität durch Quantil-Quantil-Plots



- Normalität bis auf Geographic im Wesentlichen bestätigt.
- Normalverteilungsannahme nicht ungerechtfertigt

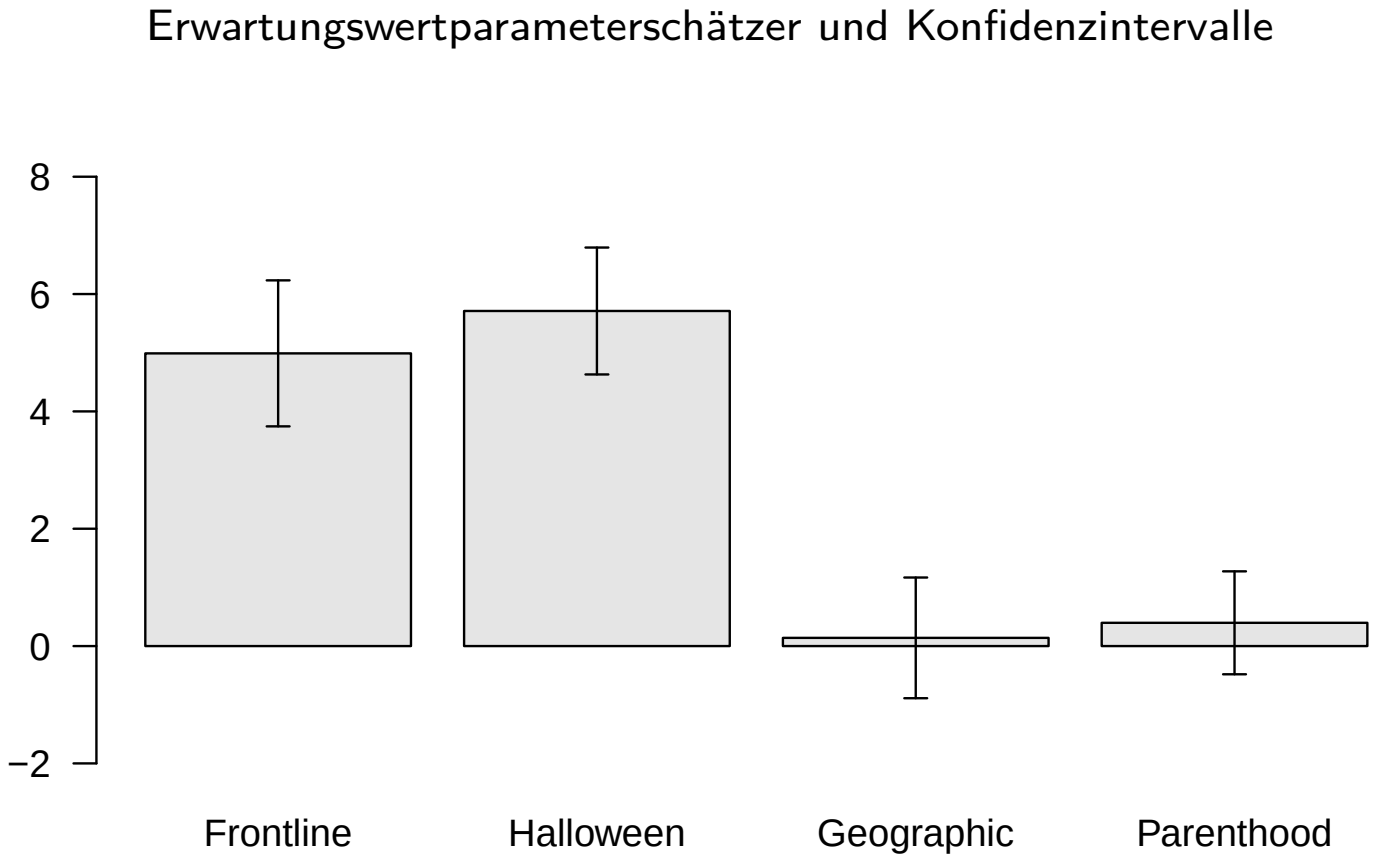
95%-Konfidenzintervalle für die DTA Erwartungwertparameter bei DTA[i] $\sim N(\mu, \sigma^2)$

$\Rightarrow$ Konfidenzintervall für μ bei $N(\mu, \sigma^2)$ mit $\sigma^2 > 0$ unbekannt

```
# KI Berechnung basierend auf dataframe STA
delta       = 0.95                                          # Konfidenzlevel
CI_delta    = STA$Std/sqrt(STA$n)*qt((1+delta)/2,STA$n-1)   # Konfidenzlevel Delta
STA$CI_L    = STA$Mean - CI_delta                           # Untere Konfidenzlevelgrenze
STA$CI_U    = STA$Mean + CI_delta                           # Obere Konfidenzlevelgrenze

# Ausgabe
print.AsIs(STA)
```

```
            n Max Min Median  Mean  Var  Std   Sem   CI_L CI_U
Frontline  83  26 -10   4.00 4.989 32.5 5.70 0.626  3.745 6.23
Halloween  78  20  -3   5.25 5.712 23.0 4.80 0.543  4.630 6.79
Geographic 85  14 -13   0.00 0.140 22.7 4.77 0.517 -0.889 1.17
Parenthood 84   8 -18   1.00 0.396 16.3 4.04 0.441 -0.481 1.27
```

95%-Konfidenzintervalle für die DTA Erwartungswertparameter



Anwendungsbeispiel

- Exploration und Deskription
- **T-Tests**
- Einfaktorielle Varianzanalyse
- Zweifaktorielle Varianzanalyse
- Übungen und Selbstkontrollaufgaben

Einstichproben-T-Test mit $H_0 : \mu = \mu_0$ und $H_1 : \mu \neq \mu_0$

Anwendungsszenario

- Eine Stichprobe experimenteller Einheiten
- Annahme der unabhängigen und identischen Normalverteilung $N(\mu, \sigma^2)$
- μ und σ^2 unbekannt
- Quantifizieren der Unsicherheit beim infererentiellen Vergleich von μ mit einem μ_0 beabsichtigt.

Anwendungsbeispiel

- Analyse der TA Werte vor Filmclipexposition "Halloween" in der "maps" Studie
  - $\mu \neq \mu_0 := 12\,? \Rightarrow$ Vom normalen TA-Ruhewert abweichende Proband:innengruppe?

Datenauswahl

```
library(psychTools)                                  # Datensatzladen
data(affect)                                         # Datensatzladen
film  = 2                                            # Filmbedingung
x     = affect$TA1[affect$Study == "maps" &          # Prae-Filmexpositionsdaten in "maps" Studie
                   affect$Film  == film]             # ... für Filmbedingung "film"
```

Einstichproben-T-Test mit $H_0 : \mu = \mu_0$ und $H_1 : \mu \neq \mu_0$

```
# Manueller Einstichproben-T-Test
n         = length(x)                          # Stichprobengröße
mu_0      = 12                                 # Nullhypothese H_0
alpha_0   = 0.05                               # Signifikanzniveau
k_alpha_0 = qt(1 - (alpha_0/2), n-1)           # kritischer Wert
x_bar     = mean(x)                            # Stichprobenmittel
s_n       = sd(x)                              # Stichprobenstandardabweichung
t         = sqrt(n)*(x_bar - mu_0)/s_n         # Teststatistik
if(abs(t) >= k_alpha_0){                      # Test 1_{|T(X) >= k_alpha_0|}
    phi = 1                                    # Ablehnen von H_0
} else {
    phi = 0                                    # Nicht Ablehnen von H_0
}
pval      = 2*(1 - pt(abs(t), n-1))            # p-Wert
cat("\nfg        = ", n-1,                     # Ausgabe
    "\nx_bar     = ", x_bar,
    "\ns_n       = ", s_n,
    "\nt         = ", T,
    "\nalpha_0   = ", alpha_0,
    "\nk_alpha_0 = ", k_alpha_0,
    "\nphi       = ", phi,
    "\np-Wert    = ", pval)
```

```
fg        =  36
x_bar     =  12.5
s_n       =  4.3
t         =  TRUE
alpha_0   =  0.05
k_alpha_0 =  2.03
phi       =  0
p-Wert    =  0.52
```

Einstichproben-T-Test mit $H_0 : \mu = \mu_0$ und $H_1 : \mu \neq \mu_0$

```
# Automatischer Einstichproben-T-Test
varphi    = t.test(                          # ?t.test für Details
            x,                               # Datensatz
            alternative = c("two.sided"),    # H_1: \mu \neq \mu_0
            mu          = 12,                # \mu_0 (sic!)
            conf.level  = 1-alpha_0)         # \delta = 1 - \alpha_0 (sic!)

# Ausgabe
print(varphi)
```

```
    One Sample t-test

data:  x
t = 0.7, df = 36, p-value = 0.5
alternative hypothesis: true mean is not equal to 12
95 percent confidence interval:
 11.0 13.9
sample estimates:
mean of x
     12.5
```

```
# Genauere Ausgabe t
paste(varphi[1])
```

```
[1] "c(t = 0.650169688685817)"
```

```
# Genauere Ausgabe p
paste(varphi[3])
```

```
[1] "0.519710450116824"
```

Einstichproben-T-Test mit $H_0 : \mu \leq \mu_0$ und $H_1 : \mu > \mu_0$

Anwendungsszenario

- Eine Stichprobe experimenteller Einheiten
- Annahme der unabhängigen und identischen Normalverteilung $N(\mu, \sigma^2)$
- μ und σ^2 unbekannt
- Quantifizieren der Unsicherheit beim infererentiellen Vergleich von μ und σ^2 beabsichtigt.

Anwendungsbeispiel

- Analyse der TA Werte vor Filmclipexposition "Parenthood" in der "maps" Studie
  - $\mu > \mu_0 := 12? \Rightarrow$ Proband:innengruppe mit höheren Werten als TA-Ruhewert?

Datenauswahl

```
library(psychTools)                                  # Datensatzladen
data(affect)                                         # Datensatzladen
film  = 4                                            # Filmbedingung
x     = affect$TA1[affect$Study == "maps" &          # Prae-Filmexpositionsdaten in "maps" Studie
                   affect$Film  == film]             # ... für Filmbedingung "film"
```

Einstichproben-T-Test mit $H_0 : \mu \leq \mu_0$ und $H_1 : \mu > \mu_0$

```
# Manueller Einstichproben-T-Test
n         = length(x)                          # Stichprobengröße
mu_0      = 12                                 # Nullhypothese H_0
alpha_0   = 0.05                               # Signifikanzniveau
k_alpha_0 = qt(1 - alpha_0, n-1)               # kritischer Wert
x_bar     = mean(x)                            # Stichprobenmittel
s_n       = sd(x)                              # Stichprobenstandardabweichung
t       = sqrt(n)*((x_bar - mu_0)/s_n)         # Teststatistik
if(t >= k_alpha_0){                            # Test 1_{T >= k_alpha_0}
    phi = 1                                    # Ablehnen von H_0
} else {
    phi = 0                                    # Nicht Ablehnen von H_0
}
pval      = 1 - pt(t, n-1)                     # p-Wert
cat("\nfg        = ", n-1,                     # Ausgabe
    "\nx_bar     = ", x_bar,
    "\ns_n       = ", s_n,
    "\nt         = ", t,
    "\nalpha_0   = ", alpha_0,
    "\nk_alpha_0 = ", k_alpha_0,
    "\nphi       = ", phi,
    "\np-Wert    = ", pval)
```

```
fg        =  37
x_bar     =  14.2
s_n       =  4.69
t         =  2.94
alpha_0   =  0.05
k_alpha_0 =  1.69
phi       =  1
p-Wert    =  0.00283
```

Einstichproben-T-Test mit $H_0 : \mu \leq \mu_0$ und $H_1 : \mu > \mu_0$

```
# Automatischer Einstichproben-T-Test
varphi    = t.test(                          # ?t.test für Details
            x,                               # Datensatz
            alternative = c("greater"),      # H_1: \mu > \mu_0
            mu          = 12,                # \mu_0 (sic!)
            conf.level  = 1-alpha_0)         # \delta = 1 - \alpha_0 (sic!)

# Ausgabe
print(varphi)
```

```
    One Sample t-test

data:  x
t = 3, df = 37, p-value = 0.003
alternative hypothesis: true mean is greater than 12
95 percent confidence interval:
  13 Inf
sample estimates:
mean of x
     14.2
```

```
# Genauere Ausgabe t
paste(varphi[1])
```

```
[1] "c(t = 2.93821956016834)"
```

```
# Genauere Ausgabe p
paste(varphi[3])
```

```
[1] "0.00282732752593068"
```

Anwendungsszenario

- Zwei Stichproben experimenteller Einheiten
- Annahme der unabhängiger und identischen Normalverteilungen $N(\mu_1, \sigma_1^2)$ und $N(\mu_2, \sigma_2^2)$
- $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ unbekannt
- Quantifizieren der Unsicherheit beim infererentiellen Vergleich von μ_1 und μ_2 beabsichtigt.

Anwendungsbeispiel

- Analyse der TA Werte nach Filmclipexpositionen “Frontline” und “Halloween” in “maps” Studie
  - $\mu_1 \neq \mu_2$? $\Rightarrow$ Induzieren “Frontline” und “Halloween” unterschiedliche TA Level?

Datenauswahl

```
library(psychTools)                                  # Datensatzladen
data(affect)                                         # Datensatzladen
x_1    = affect$TA2[affect$Study == "maps" &         # Post-Filmexpositionsdaten in "maps" Studie
                    affect$Film  == 1]               # ... für Filmbedingung "Frontline"
x_2    = affect$TA2[affect$Study == "maps" &         # Post-Filmexpositionsdaten in "maps" Studie
                    affect$Film  == 2]               # ... für Filmbedingung "Halloween"
```

```
# Manueller Zweistichproben-T-Test
n_1       = length(x_1)                                          # Stichprobengröße n_1
n_2       = length(x_2)                                          # Stichprobengröße n_2
alpha_0   = 0.05                                                 # Signifikanzniveau
k_alpha_0 = qt(1 - (alpha_0/2), n_1+n_2-2)                       # kritischer Wert
x_bar_1   = mean(x_1)                                            # x_bar_1
x_bar_2   = mean(x_2)                                            # x_bar_2
s_12      = sqrt((sum((x_1-x_bar_1)^2)+sum((x_2-x_bar_2)^2))/    # gepoolte Standardabweichung s_12
                 (n_1+n_2-2))
t       = sqrt((n_1*n_2)/(n_1+n_2))*((x_bar_1-x_bar_2)/s_12)  # Zweistichproben-T-Teststatistik
if(abs(t) >= k_alpha_0){                                         # Test 1_{|T(X) >= k_alpha_0|}
    phi = 1                                                      # Ablehnen von H_0
} else {
    phi = 0                                                      # Nicht Ablehnen von H_0
}
pval      = 2*(1 - pt(abs(t), n_1+n_2-2))                        # p-Wert
cat("\nx_bar_1   = ", x_bar_1,                                   # Ausgabe
    "\nx_bar_2   = ", x_bar_2,
    "\nfg        = ", n_1 + n_2 - 2,
    "\nalpha_0   = ", alpha_0,
    "\nk_alpha_0 = ", k_alpha_0,
    "\nt         = ", t,
    "\nphi       = ", phi,
    "\np-Wert    = ", pval)
```

```
x_bar_1   =  18.9
x_bar_2   =  17.9
fg        =  77
alpha_0   =  0.05
k_alpha_0 =  1.99
t         =  0.828
phi       =  0
p-Wert    =  0.41
```

```
# Automatischer Zweistichproben-T-Test
varphi    = t.test(                          # ?t.test für Details
            x_1,                             # Datensatz x_1
            x_2,                             # Datensatz x_2
            var.equal   = TRUE,              # \sigma_1^2 = \sigma_2^2
            alternative = c("two.sided"),    # H_1: \mu_1 \neq \mu_2
            conf.level  = 1-alpha_0)         # \delta = 1 - \alpha_0 (sic!)

# Ausgabe
print(varphi)
```

```
    Two Sample t-test

data:  x_1 and x_2
t = 0.8, df = 77, p-value = 0.4
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.38  3.35
sample estimates:
mean of x mean of y
     18.9      17.9
```

```
# Genauere Ausgabe t
paste(varphi[1])
```

```
[1] "c(t = 0.828099919448388)"
```

```
# Genauere Ausgabe p
paste(varphi[3])
```

```
[1] "0.410173261831605"
```

```
# Automatischer Zweistichproben-T-Test
varphi    = t.test(                          # ?t.test für Details
            x_1,                             # Datensatz x_1
            x_2,                             # Datensatz x_2
            var.equal   = FALSE,             # Behrens-Fisher Problem
            alternative = c("two.sided"),    # H_1: \mu_1 \neq \mu_2
            conf.level  = 1-alpha_0)         # \delta = 1 - \alpha_0 (sic!)

# Ausgabe
print(varphi)
```

```
    Welch Two Sample t-test

data:  x_1 and x_2
t = 0.8, df = 73, p-value = 0.4
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.40  3.37
sample estimates:
mean of x mean of y
     18.9      17.9
```

```
# Genauere Ausgabe t
paste(varphi[1])
```

```
[1] "c(t = 0.821929786001458)"
```

```
# Genauere Ausgabe p
paste(varphi[3])
```

```
[1] "0.413805439616455"
```

Zweistichproben-T-Test bei abhängigen Stichproben

Anwendungsszenario

- **Zwei Messungen** an einer Gruppe experimenteller Einheiten.
- Annahme "abhängiger" Normalverteilungen $N(\mu_1, \sigma_1^2)$ und $N(\mu_2, \sigma_2^2)$
- $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ unbekannt.
- Quantifizieren der Unsicherheit beim inferentiellen Vergleich von μ_1 mit μ_2 beabsichtigt.

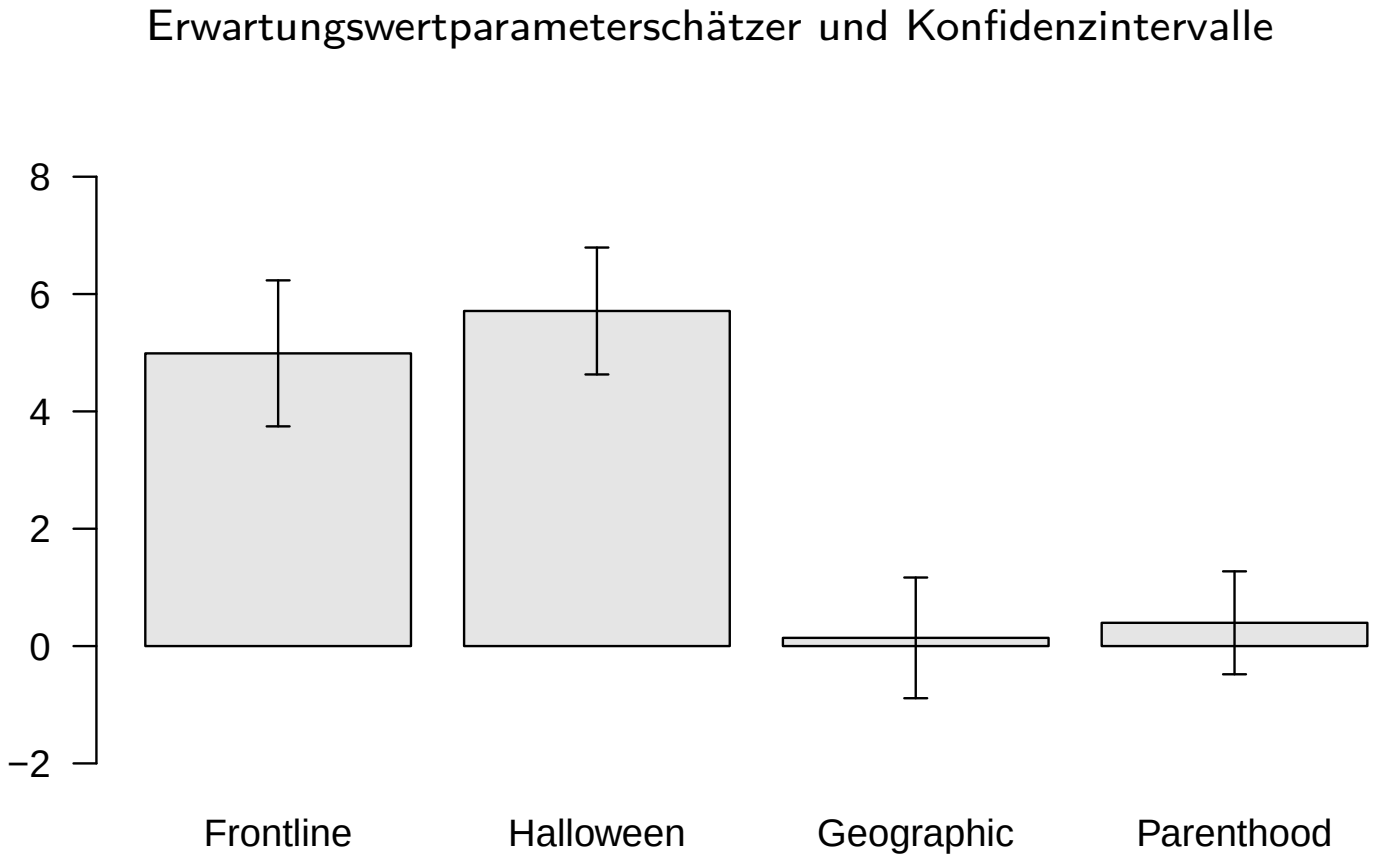
Anwendungsbeipiele

- Tense Arousal Datenanalyse bei einer Filmclipbedingung
  - Erste Messung vor Filmclipexposition, zweite Messung nach Filmclipexposition
  - $\mu_1 \neq \mu_2 ? \Leftrightarrow$ Hat Filmclipexposition einen Einfluss auf TA?

Evaluation der Post-Prae-Filmexposition Tense Arousal Wertdifferenz (DTA)

```
library(psychTools)                          # psychTools library

data(affect)                                 # Laden des affect Datensatzes

affect$DTA = affect$TA2 - affect$TA1         # Post-TA minus Prae-TA Werte
```

95%-Konfidenzintervalle für die DTA Erwartungswertparameter



Zweistichproben-T-Test bei abhängigen Stichproben

Filmclipbedingung “Halloween”

```
x     = affect$DTA[affect$Study == "maps" &          # Post minus Prae Filmexpositions TA Differenzen
                   affect$Film  == 2]                # ... für Filmbedingung Halloween
n         = length(x)                                # Stichprobengröße
mu_0      = 0                                        # Nullhypothese \mu_1 = \mu_2 \Leftrightarrow \mu = 0
alpha_0   = 0.05                                     # Signifikanzniveau
k_alpha_0 = qt(1 - alpha_0/2, n-1)                   # kritischer Wert
x_bar     = mean(x)                                  # Stichprobenmittel
s_n       = sd(x)                                    # Stichprobenstandardabweichung
t       = sqrt(n)*((x_bar - mu_0)/s_n)               # Teststatistik
if(t >= k_alpha_0){                                  # Test 1_{T >= k_alpha_0}
    phi = 1                                          # Ablehnen von H_0
} else {
    phi = 0                                          # Nicht Ablehnen von H_0
}
pval      = 2*(1-pt(abs(t), n-1))                    # p-Wert
cat("\nfg        = ", n-1,                           # Ausgabe
    "\nx_bar     = ", x_bar,
    "\ns_n       = ", s_n,
    "\nt         = ", t,
    "\nalpha_0   = ", alpha_0,
    "\nk_alpha_0 = ", k_alpha_0,
    "\nphi       = ", phi,
    "\np-Wert    = ", pval)
```

```
fg        =  36
x_bar     =  5.49
s_n       =  5.05
t         =  6.6
alpha_0   =  0.05
k_alpha_0 =  2.03
phi       =  1
p-Wert    =  1.09e-07
```

Filmclipbedingung “Parenthood”

```
x     = affect$DTA[affect$Study == "maps" &          # Post minus Prae Filmexpositions TA Differenzen
                   affect$Film  == 4]                # ... für Filmbedingung Halloween
n         = length(x)                                # Stichprobengröße
mu_0      = 0                                        # Nullhypothese \mu_1 = \mu_2 \Leftrightarrow \mu = 0
alpha_0   = 0.05                                     # Signifikanzniveau
k_alpha_0 = qt(1 - alpha_0/2, n-1)                   # kritischer Wert
x_bar     = mean(x)                                  # Stichprobenmittel
s_n       = sd(x)                                    # Stichprobenstandardabweichung
t       = sqrt(n)*((x_bar - mu_0)/s_n)               # Teststatistik
if(t >= k_alpha_0){                                  # Test 1_{T >= k_alpha_0}
    phi = 1                                          # Ablehnen von H_0
} else {
    phi = 0                                          # Nicht Ablehnen von H_0
}
pval      = 2*(1-pt(abs(t), n-1))                    # p-Wert
cat("\nfg        = ", n-1,                           # Ausgabe
    "\nx_bar     = ", x_bar,
    "\ns_n       = ", s_n,
    "\nt         = ", t,
    "\nalpha_0   = ", alpha_0,
    "\nk_alpha_0 = ", k_alpha_0,
    "\nphi       = ", phi,
    "\np-Wert    = ", pval)
```

```
fg        =  37
x_bar     =  0.211
s_n       =  3.74
t         =  0.347
alpha_0   =  0.05
k_alpha_0 =  2.03
phi       =  0
p-Wert    =  0.73
```

Anwendungsbeispiel

- Exploration und Deskription
- T-Tests
- **Einfaktorielle Varianzanalyse**
- Zweifaktorielle Varianzanalyse
- Übungen und Selbstkontrollaufgaben

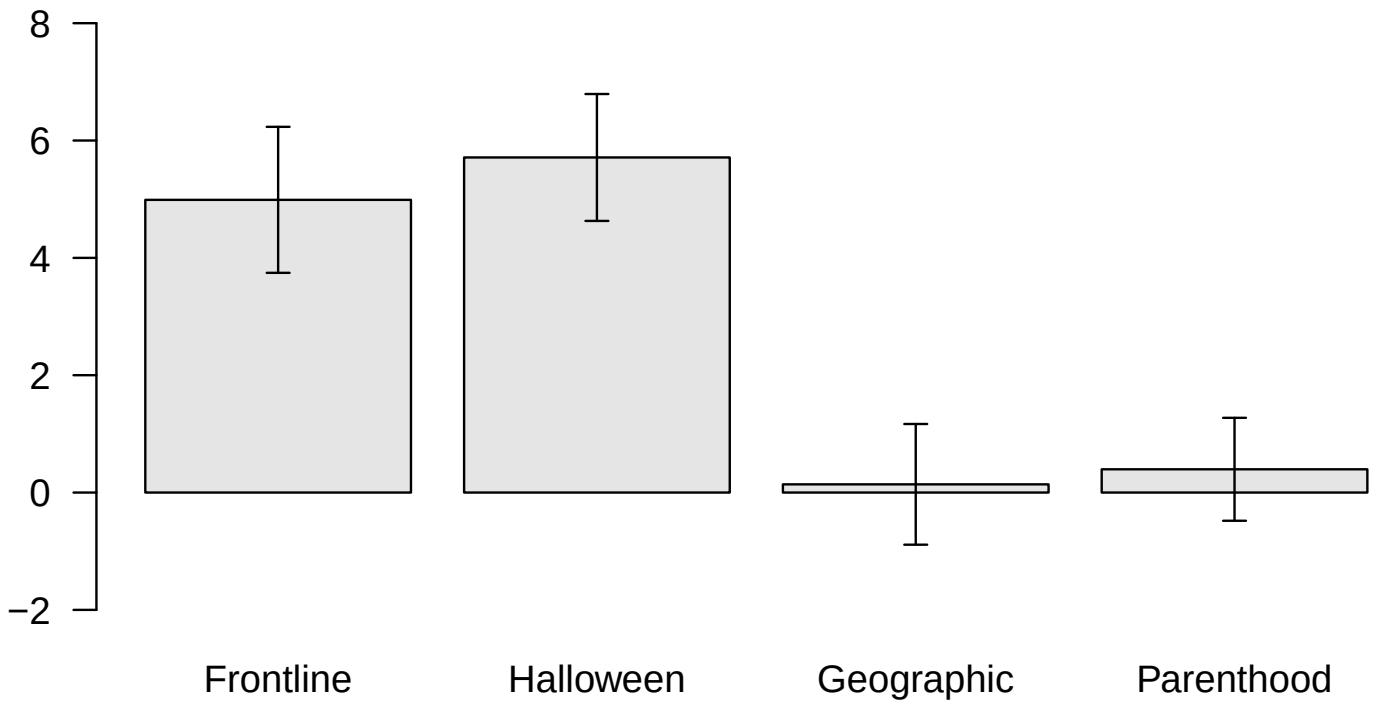
Anwendungsszenario

- **Zwei oder mehr Stichproben** (im Folgenden oft **Gruppen** genannt) experimenteller Einheiten.
- Annahme der unabhängigen und identischen Normalverteilung $N(\mu_i, \sigma^2)$ der Daten.
- μ_i und σ^2 unbekannt.
- Absicht des inferentiellen Testens der Nullhypothese identischer Gruppenerwartungswerte.

Tense Arousal Differenzwertanalyse der vier Filmclipbedingungen im Affect Datensatz

- Inferentielle Evidenz für Gruppenerwartungswertunterschiede?
- Evidenz für unterschiedliche Tense Arousal Induktion durch unterschiedliche Filmclips?

Erwartungswertparameterschätzer und Konfidenzintervalle



Datenaufbereitung

```
L               = list(                                          # Liste der DTA Gruppenwerte
                  x_1j = affect$DTA[affect$Film == 1],
                  x_2j = affect$DTA[affect$Film == 2],
                  x_3j = affect$DTA[affect$Film == 3],
                  x_4j = affect$DTA[affect$Film == 4])
m               = c(
                  length(L$x_1j),                                # Vektor der Gruppengrößen
                  length(L$x_2j),
                  length(L$x_3j),
                  length(L$x_4j))
m_max           = max(m)                                         # Maximale Gruppengröße
X               = data.frame(                                    # Datafrrameerzeugung mit NA Padding
                  x_1j = c(L$x_1j, rep(NA, m_max-length(L$x_1j))),
                  x_2j = c(L$x_2j, rep(NA, m_max-length(L$x_2j))),
                  x_3j = c(L$x_3j, rep(NA, m_max-length(L$x_3j))),
                  x_4j = c(L$x_4j, rep(NA, m_max-length(L$x_4j))))
```

Dataframezeilen ab Zeile 78

	x_1j	x_2j	x_3j	x_4j
78	4	3	-4.0	-1
79	1	NA	0.0	-18
80	11	NA	9.9	2
81	4	NA	1.0	1
82	0	NA	5.0	-5
83	7	NA	-3.0	2
84	NA	NA	1.0	-3
85	NA	NA	0.0	NA

Manuelle Berechnung bei fehlenden Daten (NAs)

```
p           = ncol(X)                                                    # Gruppenanzahl
m           = colSums(!is.na(X))                                         # Gruppengrößen bei gleichen Gruppengrößen
n           = sum(m)                                                     # Gesamtstichprobengröße
x_bar       = mean(as.matrix(X), na.rm = TRUE)                           # Gesamtmittelwert
x_bar_i     = colMeans(as.matrix(X), na.rm = TRUE)                       # Gruppenmittelwerte
alpha_0     = 0.05                                                       # Signifikanzlevel
k_alpha_0   = qf(1 - alpha_0, p-1, n-p)                                  # kritischer Wert
SQT         = sum((as.vector(as.matrix(X)) - x_bar)^2, na.rm = TRUE)     # Total Sum of Squares
SQB         = sum(m*(x_bar_i - x_bar)^2, na.rm = TRUE )                  # Between-Group Sum of Squares
SQW         = sum((t(as.matrix(X)) - as.vector(x_bar_i))^2, na.rm = TRUE )  # Within-Group  Sum of Squares
MSB         = SQB/(p-1)                                                  # Between-Group Mean Squares
MSW         = SQW/(n-p)                                                  # Within-Group  Mean Squares
f_tilde     = MSB/MSW                                                    # Realisierung der F-Teststatistik
if(f_tilde >= k_alpha_0){                                                # \tilde{f} >= k_alpha_0
   phi = 1                                                               # Testwert
   } else {
   phi = 0                                                               # Testwert
}
p_value     = 1 - pf(f_tilde, p-1, n-p)                                  # p-Wert
cat("Einfaktorielle Varianzanalyse",
    "\nGruppenanzahl           = ", p,                                   # Ausgabe
    "\nGruppengrößen           = ", m,
    "\nGesamtstichprobengröße = ", n,
    "\nGesamtmittelwert        = ", x_bar,
    "\nGruppenmittelwerte      = ", x_bar_i,
    "\nalpha_0                 = ", alpha_0,
    "\nk_alpha_0               = ", k_alpha_0,
    "\nSQT                     = ", SQT,
    "\nSQB                     = ", SQB,
    "\nSQW                     = ", SQW,
    "\nMSB                     = ", MSB,
    "\nMSW                     = ", MSW,
    "\nf_tilde                 = ", f_tilde,
    "\nphi                     = ", phi,
    "\np-Wert                  = ", p_value,
   "\n\n")
```

Manuelle Berechnung bei fehlenden Daten (NAs)

```
Einfaktorielle Varianzanalyse
Gruppenanzahl          =  4
Gruppengrößen          =  83 78 85 84
Gesamtstichprobengröße =  330
Gesamtmittelwert       =  2.74
Gruppenmittelwerte     =  4.99 5.71 0.14 0.396
alpha_0                =  0.05
k_alpha_0              =  2.63
SQT                    =  9847
SQB                    =  2145
SQW                    =  7702
MSB                    =  715
MSW                    =  23.6
f_tilde                =  30.3
phi                    =  1
p-Wert                 =  0
```

Automatische Berechnung bei fehlenden Daten (NAs)

```
XAOV    = data.frame(dta   = c(X$x_1j, X$x_2j, X$x_3j, X$x_4j),      # Reformatierung des Datensatzes
                     group = as.factor(c(rep(1,length(X$x_1j)),
                                         rep(2,length(X$x_2j)),
                                         rep(3,length(X$x_3j)),
                                         rep(4,length(X$x_4j)))))

res.aov = aov(dta ~ group, data = XAOV)                              # Benutzung von R's aov Funktion
summary(res.aov)                                                    # Ausgabe
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
group         3   2145     715    30.3 <2e-16 ***
Residuals   326   7702      24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
10 observations deleted due to missingness
```

- Die Nullhypothese identischer Gruppenerwartungswertparameter kann abgelehnt werden.
- Die Filmclipbedingungen induzieren recht sicher unterschiedliche TA Werte.

Anwendungsbeispiel

- Exploration und Deskription
- T-Tests
- Einfaktorielle Varianzanalyse
- **Zweifaktorielle Varianzanalyse**
- Übungen und Selbstkontrollaufgaben

Zweifaktorielle Varianzanalyse

Anwendungsszenario

Zweifaktorielle Versuchspläne mit crossed design (cf. Modul A.2)

- Eine univariate Messvariable bestimmt an individuellen experimentellen Einheiten.
- Zwei diskrete experimentelle Variablen, die mindestens zweistufig sind.
- Die experimentellen Variablen werden *Faktoren* genannt.
- Die Stufen der Faktoren werden auch *Faktorlevel* genannt.
- Jedes Level eines Faktors wird mit allen Level des anderen Faktors kombiniert

TA Score Pre-Post-Differenzwertanalyse für zwei Filmclips und zwei Studien

- Faktor Filmclip (A) mit Level Halloween (1) und Geographic (2)
- Faktor Studien (B) mit Level Flat (1) und Maps (2)
- 2 x 2 faktorieller Versuchsplan mit crossed design $\Rightarrow$ 2 x 2 ANOVA
  - Inferentielle Evidenz für einen Haupteffekt der Filmclipbedingung?
  - Inferentielle Evidenz für einen Haupteffekt der Studie?
  - Inferentielle Evidenz für eine Interaktion = Änderung der Filmclipeffekte zwischen den Studien?

Datenaufbereitung

```
L               = list(                                                    # Liste der DTA Gruppenwerte
                  A1B1 = affect$DTA[affect$Film == 2 & affect$Study == "maps"],  # Halloween , Maps
                  A1B2 = affect$DTA[affect$Film == 2 & affect$Study == "flat"],  # Halloween , Flat
                  A2B1 = affect$DTA[affect$Film == 3 & affect$Study == "maps"],  # Geographic, Maps
                  A2B2 = affect$DTA[affect$Film == 3 & affect$Study == "flat"])  # Geographic, Flat
m               = c(
                  length(L$A1B1),                                          # Vektor der Gruppengrößen
                  length(L$A1B2),
                  length(L$A2B1),
                  length(L$A2B2))
m_min           = min(m)                                                   # minimale  Gruppengröße
X               = data.frame(                                              # balanced design (= gleiche Gruppengrößen)
                  A1B1 = L$A1B1[1:m_min],
                  A1B2 = L$A1B2[1:m_min],
                  A2B1 = L$A2B1[1:m_min],
                  A2B2 = L$A2B2[1:m_min])
```

Dataframezeilen ab Zeile 35

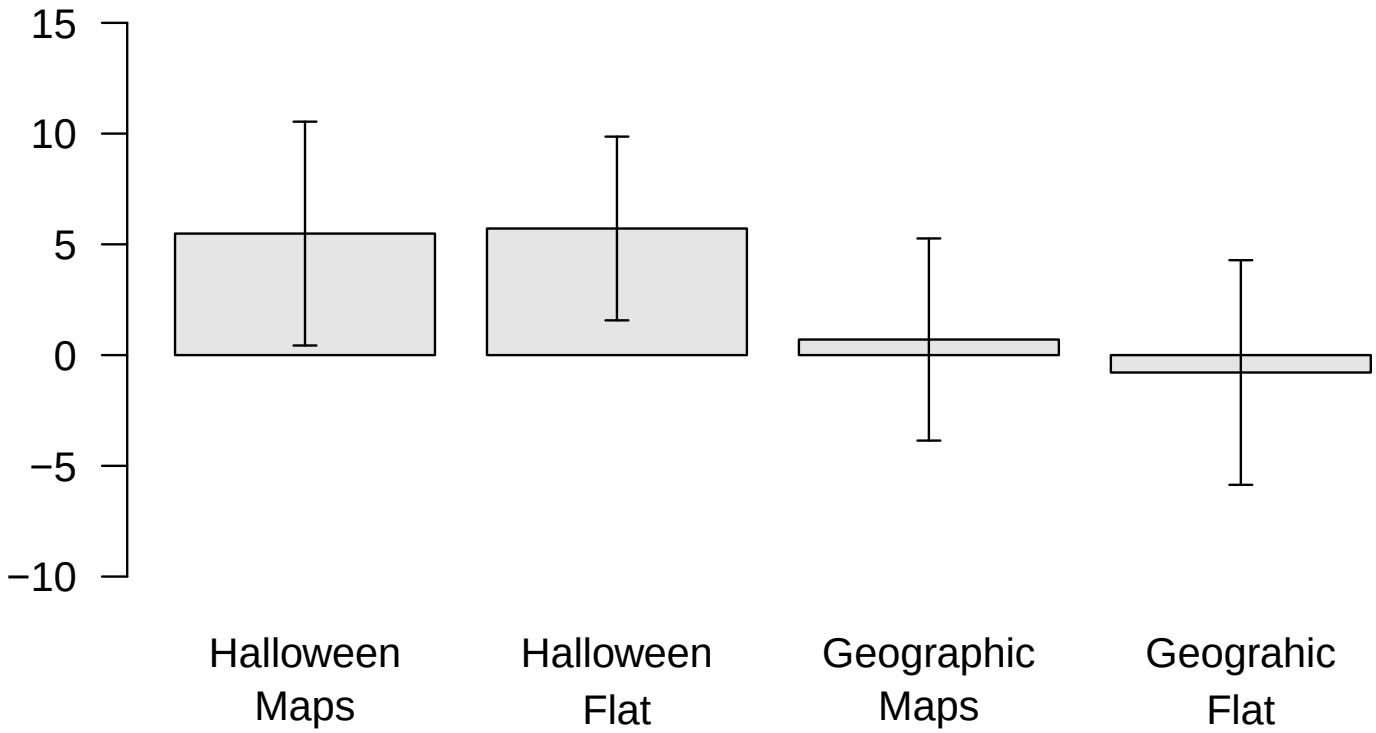
	A1B1	A1B2	A2B1	A2B2
35	5	6	-1	-4.0
36	2	1	-5	0.0
37	2	10	12	9.9

Zweifaktorielle Varianzanalyse

Darstellung der Gruppenmittelwerte und Gruppenstandardabweichungen durch Balkendiagramm

```
dev.new()                                          # Figure Initialisierung
library(latex2exp)                                 # TeX Annotations
par(                                               # für Details siehe ?par
family      = "sans",                              # Serif-freier Fonttyp
pty         = "m",                                 # Maximale Abbildungsregion
bty         = "l",                                 # L förmige Box
lwd         = 1,                                   # Liniendicke
las         = 1,                                   # Horizontale Achsenbeschriftung
font.main   = 1,                                   # Non-Bold Titel
cex         = 1.1,                                 # Textvergrößerungsfaktor
cex.main    = 1.2)                                 # Titeltextvergrößerungsfaktor
groupmeans  = colMeans(X, na.rm = TRUE)            # Gruppenmittelwerte
groupstds   = apply(X,2,sd)                        # Gruppenmittelwerte
x = barplot(                                       # Ausgabe der x-Ordinaten (?barplot für Details)
groupmeans,
ylim        = c(-10,15),
col         = "gray90",
names.arg   = c("Halloween"  , "Halloween",        # Labels
                "Geographic" , "Geograhic"))
text(x[1],-16, "Maps" , xpd = T, cex = 1)
text(x[2],-16, "Flat" , xpd = T, cex = 1)
text(x[3],-16, "Maps" , xpd = T, cex = 1)
text(x[4],-16, "Flat" , xpd = T, cex = 1)
arrows(                                            # für Details siehe ?arrows
x0          = x,                                   # arrow start x-ordinate
y0          = groupmeans - groupstds,              # arrow start y-ordinate
x1          = x,                                   # arrow end   x-ordinate
y1          = groupmeans + groupstds,              # arrow end   y-ordinate
code        = 3,                                   # Pfeilspitzen beiderseits
angle       = 90,                                  # Pfeilspitzenwinkel -> Linie
length      = 0.05)                                # Linienlänge
dev.copy2pdf(
file        = file.path(getwd(), "08_Abbildungen", "cda_08_aov_2_barplot.pdf"),
width       = 7,
height      = 4)
```

Darstellung der Gruppenmittelwerte und Gruppenstandardabweichungen durch Balkendiagramm



Zweifaktorielle Varianzanalyse

Darstellung der Gruppenmittelwerte und Gruppenstandardabweichungen durch Liniendiagramm

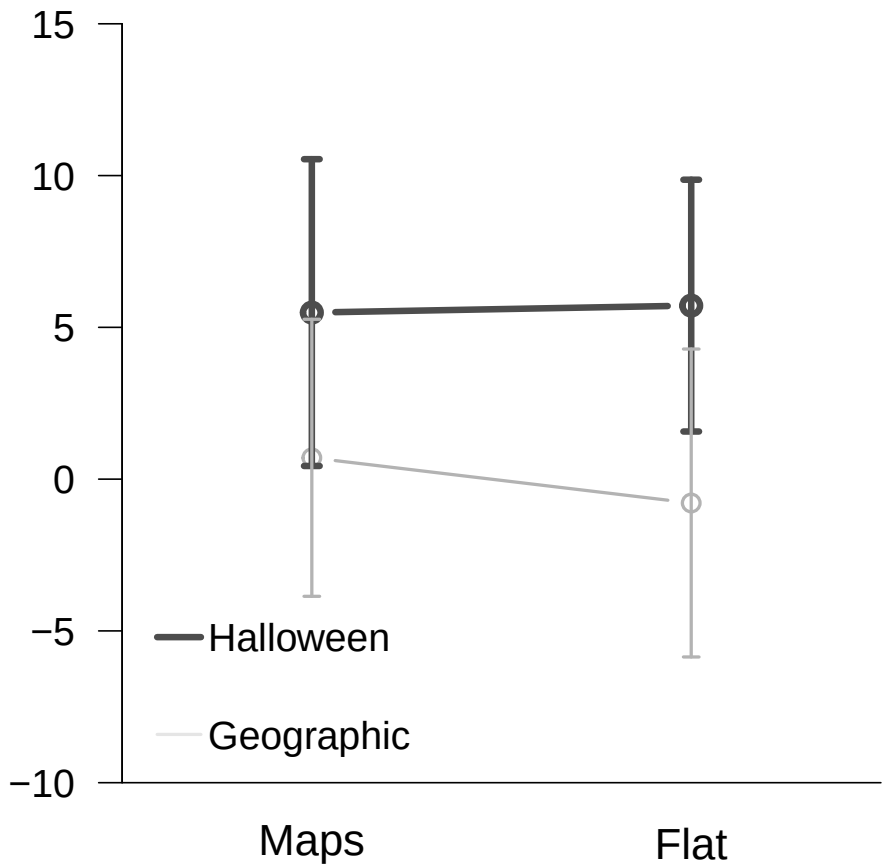
```
dev.new()
library(latex2exp)
x           = 1:2
groupmeans  = matrix(colMeans(X, na.rm = TRUE), nrow = 2)
groupstds   = matrix(apply(X, 2, sd, na.rm = TRUE), nrow = 2)
cols        = c("gray30", "gray70")
lwds        = c(4,2)
graphics.off()
fdir        =  file.path(getwd(), "08_Abbildungen")
dev.new()
par(
family      = "sans",
pty         = "s",
bty         = "l",
lwd         = 1,
las         = 1,
mgp         = c(2,1,0),
xaxs        = "i",
yaxs        = "i",
font.main   = 1,
cex         = 1.5,
cex.main    = 1)
matplot(
x,
groupmeans,
type        = "b",
pch         = 21,
xlim        = c(.5,2.5),
ylim        = c(-10,15),
lty         = 1,
col         = cols,
lwd         = lwds,
xlab        = "",
ylab        = "",
xaxt       = "n")
```

Zweifaktorielle Varianzanalyse

Darstellung der Gruppenmittelwerte und Gruppenstandardabweichungen durch Liniendiagramm

```
# Fehlerbalken und Annotationen
for(i in 1:2){
  arrows(                                          # für Details siehe ?arrows
  x0          = x,                                 # arrow start x-ordinate
  y0          = groupmeans[,i] - groupstds[,i],    # arrow start y-ordinate
  x1          = x,                                 # arrow end   x-ordinate
  y1          = groupmeans[,i] + groupstds[,i],    # arrow end   y-ordinate
  col         = cols[[i]],                         # color
  lwd         = lwds[[i]],                         # linewidth
  code        = 3,                                 # Pfeilspitzen beiderseits
  angle       = 90,                                # Pfeilspitzenwinkel -> Linie
  length      = 0.05)
}
legend(
.4,
-2,
c("Halloween", "Geographic"),
lty         = 1,
col         = c("gray30", "gray90"),
lwd         = lwds,
bty         = "n",
cex         = 1,
x.intersp   = .1,
seg.len     = 0.6,
)
text(1,-12, "Maps" , xpd = T, cex = 1.1)
text(2,-12, "Flat" , xpd = T, cex = 1.1)
dev.copy2pdf(
file        = file.path(getwd(), "08_Abbildungen", "cda_08_aov_2_lineplot.pdf"),
width       = 7,
height      = 7)
```

Darstellung der Gruppenmittelwerte und Gruppenstandardabweichungen durch Liniendiagramm



Manuelle Berechnung

```r
# Analyseparameter
p               = 2                                                    # Anzahl Level Faktor A
q               = 2                                                    # Anzahl Level Faktor B
m               = m_min                                                # Anzahl Datenpunkte pro Zelle
n               = p*q*m                                                # Gesamtstichprobengröße
alpha_0         = 0.05                                                 # Signifikanzlevel

# Mittelwerte
x_bar           = mean(as.matrix(X))                                   # Gesamtmittelwert \bar{X}
x_bar_i         = colMeans(cbind(as.matrix(c(X$A1B1, X$A1B2)),         # Faktor A Mittelwerte \bar{X}_i
                                 as.matrix(c(X$A2B1, X$A2B2))))
x_bar_j         = colMeans(cbind(as.matrix(c(X$A1B1, X$A2B1)),         # Faktor B Mittelwerte \bar{X}_j
                                 as.matrix(c(X$A1B2, X$A2B2))))
x_bar_ij        = colMeans(X)                                          # Gruppenmittelwerte \bar{X}_{ij}

# Parameterschätzer
mu_hat          = x_bar                                                # \mu Schätzer
alpha_hat_i     = x_bar_i - x_bar                                      # \alpha_i Schätzer
beta_hat_j      = x_bar_j - x_bar                                      # \beta_j Schätzer
gamma_hat_ij    = c(x_bar_ij[1] - (mu_hat + alpha_hat_i[1] + beta_hat_j[1]),     # \gamma_ij Schätzer
                    x_bar_ij[2] - (mu_hat + alpha_hat_i[1] + beta_hat_j[2]),
                    x_bar_ij[3] - (mu_hat + alpha_hat_i[2] + beta_hat_j[1]),
                    x_bar_ij[4] - (mu_hat + alpha_hat_i[2] + beta_hat_j[2]))
sigsqr_hat      = sum((t(as.matrix(X)) - as.vector(x_bar_ij))^2)/(p*q*(m-1))      # \sigma^2 Schätzers

# Sum of squares
SQT             = mean((as.matrix(X) - x_bar)^2)                       # Total        SOS
SQA             = q*m*sum(alpha_hat_i^2)                               # Faktor A     SOS
SQB             = p*m*sum(beta_hat_j^2)                                # Faktor B     SOS
SQG             = m*sum(gamma_hat_ij^2)                                # Interaktions SOS
SQR             = p*q*(m-1)*sigsqr_hat                                 # Residual     SOS
```

Manuelle Berechnung

```
# Mean Sum of Squares
MSA             = SQA/(p-1)                                                  # Mean Square Faktor A
MSB             = SQB/(q-1)                                                  # Mean Square Faktor B
MSG             = SQG/((p-1)*(q-1))                                          # Mean Square Interaktion
MSR             = SQR/(p*q*(m-1))                                            # Mean Square Residuals

# Teststatistiken
F_A             = MSA/MSR                                                    # Faktor A F-Teststatistik
F_B             = MSB/MSR                                                    # Faktor B F-Teststatistik
F_AB            = MSG/MSR                                                    # Interaktion F-Teststatistik

# kritische Werte
k_A_alpha_0     = qf(1-alpha_0, p-1        , p*q*(m-1))                      # k^A_\alpha_0
k_B_alpha_0     = qf(1-alpha_0, q-1        , p*q*(m-1))                      # k^B_\alpha_0
k_AB_alpha_0    = qf(1-alpha_0, (p-1)*(q-1), p*q*(m-1))                      # k^AB_\alpha_0

# Tests
phi             = c(F_A, F_B, F_AB) >= c(k_A_alpha_0,k_B_alpha_0, k_AB_alpha_0)  # simultaner Test

# p-Werte
p_values        = 1-c(pf(F_A ,p-1        , p*q*(m-1)),
                      pf(F_B ,q-1        , p*q*(m-1)),
                      pf(F_AB,(p-1)*(q-1), p*q*(m-1)))
```

Manuelle Berechnung

```
Zweifaktorielle Varianzanalyse

Anzahl Level Faktor A  = 2
Anzahl Level Faktor B  = 2
Gruppengrößen          = 37
Gesamtstichprobengröße = 148
alpha_0                = 0.05

Haupteffekt Faktor A
SQA                    = 1178
MSA                    = 1178
F_A                    = 52.8
k_A_alpha_0            = 3.91
phi_A                  = 1
p_A-Wert               = 2.16e-11

Haupteffekt Faktor B
SQB                    = 14.7
MSB                    = 14.7
F_B                    = 0.657
k_B_alpha_0            = 3.91
phi_B                  = 0
p_B-Wert               = 0.419

Interaktion A x B
SQG                    = 27.3
MSG                    = 27.3
F_AxB                  = 1.22
k_AxB_alpha_0          = 3.91
phi_AxB                = 0
p_AxB-Wert             = 0.27
```

Automatische Berechnung

```
XAOV    = data.frame(X = c(X$A1B1, X$A1B2, X$A2B1, X$A2B2)                    ,  # Reformatierung des Datensatzes
                     A = as.factor(c(rep(1,2*m), rep(2,2*m)))                 ,  # Faktor A Level Labels
                     B = as.factor(c(rep(1,m), rep(2,m), rep(1,m), rep(2,m)))    # Faktor B Level Labels
res.aov =  aov(X ~ A + B + A:B, data = XAOV)                                    # Benutzung von R's aov Funktion
summary(res.aov)
```

```
             Df Sum Sq Mean Sq F value  Pr(>F)
A             1   1178    1178   52.79 2.2e-11 ***
B             1     15      15    0.66    0.42
A:B           1     27      27    1.22    0.27
Residuals   144   3214      22
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

Die Nullhypothese keines Filmcliphaupteffektes kann abgelehnt werden.

- Die Filmclips induzieren recht sicher unterschiedliche TA Werte.

Die Nullhypothese keines Studienhaupteffektes kann nicht abgelehnt werden.

- Die Studien induzieren recht sicher keine unterschiedlichen TA Werte.

Die Nullhypothese keiner Interaktion kann nicht abgelehnt werden.

- Die durch die Filmclips induzierten TA Werte unterscheiden sich zwischen den Studien vermutlich nicht.

Anwendungsbeispiel

- Exploration und Deskription
- T-Tests
- Einfaktorielle Varianzanalyse
- Zweifaktorielle Varianzanalyse
- **Übungen und Selbstkontrollaufgaben**

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle in einem R Skript.
2. Nennen Sie eine R Funktion zur visuellen Inspektion der Normalität von Daten.
3. Nennen Sie eine R Funktion zur Durchführung von Einstichproben-T-Tests.
4. Nennen Sie eine R Funktion zur Durchführung von Zweistichproben-T-Tests.
5. Nennen Sie eine R Funktion zur Durchführung einfaktorieller Varianzanalysen.
6. Nennen Sie eine R Funktion zur Durchführung zweifaktorieller Varianzanalysen.

References

Rafaeli, Eshkol, and William Revelle. 2006. "A Premature Consensus: Are Happiness and Sadness Truly Opposite Affects?" *Motivation and Emotion* 30 (1): 1–12. https://doi.org/10.1007/s11031-006-9004-2.

Thayer, Robert E. 1986. "Activation-Deactivation Adjective Check List: Current Overview and Structural Analysis." *Psychological Reports* 58 (2): 607–14. https://doi.org/10.2466/pr0.1986.58.2.607.