

Psy\_\_B\_\_11: EDV I - Einheit 2  
elementare Datenverarbeitung

Max Brede

November 30, 2020



# Contents

<b>1</b>	<b>Vorwort</b>	<b>5</b>
<b>2</b>	<b>Lehrplan</b>	<b>7</b>
2.1	Semesterplan . . . . .	7
2.2	Übungsformat . . . . .	7
2.3	Lehrziele für jede Sitzung . . . . .	7
2.4	Prüfungsleistung . . . . .	8
<b>3</b>	<b>Vorlesung I - Rste Schritte</b>	<b>9</b>
3.1	Organisatorisches . . . . .	9
3.2	Einführung . . . . .	10
3.3	Grundlegende Rechenoperationen . . . . .	14
3.4	Ausdrücke, Funktionen, Argumente . . . . .	16
3.5	Objekte . . . . .	16
3.6	Hausaufgabe . . . . .	22
<b>4</b>	<b>elementare Datenverarbeitung</b>	<b>23</b>
4.1	Organisatorisches . . . . .	24
4.2	Vektoren . . . . .	25
4.3	Indizierung . . . . .	26
4.4	Systematische Wertefolgen erzeugen . . . . .	30
4.5	Daten transformieren . . . . .	32
4.6	Einfache deskriptiv-statistische Kennwerte . . . . .	33



# Chapter 1

## Vorwort

Dieses mit `bookdown` erstellte Dokument ist das über das Wintersemester 2020 hinweg wachsende Skript zur Übung “PSY\_B\_11-2: Computerunterstützte Datenanalyse I” der CAU zu Kiel.



# Chapter 2

## Lehrplan

### 2.1 Semesterplan

### 2.2 Übungsformat

Die Übung soll zur Hälfte in 45-minütigen Sitzungen im Vorlesungsformat zur Vorstellung der Funktionen und zur anderen Hälfte als 45-minütige praktische Übung stattfinden. Es wird pro Übungs-Sitzung ein Übungszettel ausgegeben, der mit Hilfe der in der Zugehörigen Vorlesung besprochenen Funktionen bearbeitet werden können soll. Diese Zettel sollen nach der jeweiligen Vorlesung für die Übungen vorbereitet werden, in denen der Zettel dann besprochen und mögliche Fragen geklärt werden. Nach den Übungssitzungen haben die Studierenden dann eine Woche Zeit, zusätzliche Hausaufgaben zu bearbeiten.

Eine Ausnahme von diesem Ablauf ist die erste Sitzung, in der organisatorisches und Grundlagen in 90 minütigem Vorlesungsstil besprochen werden sollen. Auch nach dieser Sitzung werden aber Übungszettel und Hausaufgaben ausgegeben.

### 2.3 Lehrziele für jede Sitzung

Die Studierenden können nach dem Absolvieren der Übung...

#### Einheit 1

- Vor- und Nachteile von R und RStudio nennen und diese installieren.
- erklären was Funktionen und was Argumente sind.
- die Hilfe benutzen.
- das Environment von R benutzen um Objekte anzulegen und zu löschen.

**Einheit 2**

- Vektoren erstellen, transformieren und indizieren.
- verschiedene Datenformate in R erstellen, benutzen und in einander überführen.

**Einheit 3**

- Pakete installieren und benutzen.
- mit Hilfe des “**tidyverse**” Datensätze erstellen, ergänzen, sortieren und indizieren.

**Einheit 4**

- Faktoren erstellen.
- deskriptive Kennwerte berechnen.
- Daten auf Gruppenebene aggregieren.

**Einheit 5**

- Daten auf Gruppenebene noch besser aggregieren.
- Häufigkeiten auszählen und tabellarisch darstellen.
- Datensätze aus verschiedenen Formaten einlesen.

**Einheit 6**

- eine Anzahl von Grafiken erstellen.

**Einheit 7**

- kompliziertere Grafiken erstellen.

**Einheit 8**

- die Funktionsschreibweise lesen und anwenden.
- erfolgreich an der Klausur teilnehmen.

## 2.4 Prüfungsleistung

Die Studierenden **müssen** während des Semesters die nach den Übungssitzungen ausgegebenen Hausaufgaben innerhalb einer Woche sinnvoll bearbeitet abgeben.

Mit maximal einer nicht sinnvoll bearbeiteten Serie werden die Studierenden zur Klausur am Ende des Semesters zugelassen.



## Chapter 3

# Vorlesung I - Rste Schritte

### 3.1 Organisatorisches

#### Semesterplan

Einheit	Vorlesung	Übungswoche	Thema
1	2.11.20	keine Übung	Grundlagen und Begriffe
2	16.11.20	KW 48	Vektoren und Indizierung
			Datenformate erstellen und transformieren
3	30.11.20	KW 50	Pakete installieren und benutzen
			Datensätze erstellen und ergänzen können
			Datensätze sortieren und indizieren können
4	14.12.20	KW 1	Faktoren
			deskriptive Kennwerte
			Aggregation I
5	11.01.21	KW 3	Aggregation II
			In- und Export von Datensätzen
6	25.01.21	KW 5	Grafische Darstellungen I
7	08.02.21	KW 7	Grafische Darstellungen II
8	22.02.21	keine Übung	Puffer
			Probeklausur

## Übungsablauf

Die Übung wird zur Hälfte als Vorlesung, zur anderen Hälfte in Kleingruppen abgehalten.

Die Daten sind im Kalender und im Semesterplan im Olat ersichtlich.

## Prüfungsleistung

Die Prüfungsleistung in dieser Veranstaltung besteht aus:

1. Dem *regelmäßigen Bearbeiten* und *Bestehen* von Hausaufgaben. Diese werden über das OLAT ausgeteilt und abgegeben, zu jeder Veranstaltung wird eine neue Serie herausgegeben. Das Bestehen der Hausaufgaben ist nötig, um zur Klausur zugelassen zu werden.
  - Als *Bestanden* gilt eine Serie, wenn alle Aufgaben **sinnvoll** bearbeitet wurden.
  - Unter *regelmäßigem Bearbeiten* versteht sich das Bestehen aller Serien **mit einer Ausnahme**.
2. Im Klausurzeitraum findet an einem Tag eine praktische Prüfung statt.

## 3.2 Einführung

### Rste Schritte

Diese Veranstaltung und das zugehörige Material sollen Ihnen einen Einstieg in das computergestützte Aufbereiten und Auswerten von empirischen Daten bieten. Dazu werden wir auf die von ihren Autoren als ‘software environment for statistical computing and graphics’ bezeichnete, freie Umgebung R zurückgreifen.

## Wozu brauche ich das?

Christian-Albrechts-Universität zu Kiel  
FPO Psychologie B.Sc. 2016  
(Keine amtliche Bekanntmachung)

Anhang 1: Studien – Verlaufsplan (nicht Bestandteil der Satzung)

Seq.						DWS	LP
1	PSY_B.1 Einführung in das Studium, Geschichte und Perspektiven der Psychologie V (2 SWS / 4 LP)	PSY_B.4 Allgemeine Einführung in die Forschungsmethodik V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.5 Wahrnehmung und Kognition V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.6 Erkenntnis, Motivation, Lernen und Gedächtnis V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.8 Entwicklungspsychologie V (2 SWS / 4 LP)		14
2			PSY_B.9 Persönlichkeitspsychologie V (2 SWS / 4 LP)		PSY_B.10 Schulpsychologie S (2 SWS / 4 LP) V (2 SWS / 4 LP)		16
3	PSY_B.2 Durchführung und Präsentation experimenteller Untersuchungen P (4 SWS / 4 LP)	PSY_B.12 Quantitative Methoden I V (4 SWS / 8 LP)	PSY_B.13 Quantitative Methoden II P (2 SWS / 2 LP)	PSY_B.14 Basismodul Arbeits- und Organisationspsychologie V (2 SWS / 4 LP)	PSY_B.15 Basismodul Klinische Psychologie und Psychotherapie V (2 SWS / 4 LP)		15
4	PSY_B.3 Experimentelles Psychologisches Praktikum P (4 SWS / 4 LP)	PSY_B.16 Experimentelles Psychologisches Praktikum P (4 SWS / 4 LP)	PSY_B.17 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.18 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.19 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)		17
5	PSY_B.16 Basismodul Wirtschaftliche Rechtspsychologie / Forensische Psychologie V (2 SWS / 4 LP)	PSY_B.20 (a-g) Evaluation und Forschungsmethoden V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.21 (a-g) Forschungsorientierte Vertiefung S (2 SWS / 4 LP)	PSY_B.22 (a-c) Schwerpunkt Map Teil 1 V (2 SWS / 4 LP)	PSY_B.23 (a-c) Schwerpunkt Map Teil 2 V (2 SWS / 4 LP)		10
6	PSY_B.17 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.18 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.19 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.20 (a-g) Evaluation und Forschungsmethoden V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.21 (a-g) Forschungsorientierte Vertiefung S (2 SWS / 4 LP)		13
7	PSY_B.18 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.19 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.20 (a-g) Evaluation und Forschungsmethoden V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.21 (a-g) Forschungsorientierte Vertiefung S (2 SWS / 4 LP)	PSY_B.22 (a-c) Schwerpunkt Map Teil 1 V (2 SWS / 4 LP)		14
8	PSY_B.19 Diagnostische Verfahren S (2 SWS / 4 LP) S (2 SWS / 4 LP) S (2 SWS / 2 LP)	PSY_B.20 (a-g) Evaluation und Forschungsmethoden V (2 SWS / 4 LP) S (2 SWS / 4 LP)	PSY_B.21 (a-g) Forschungsorientierte Vertiefung S (2 SWS / 4 LP)	PSY_B.22 (a-c) Schwerpunkt Map Teil 1 V (2 SWS / 4 LP)	PSY_B.23 (a-c) Schwerpunkt Map Teil 2 V (2 SWS / 4 LP)		6
							29
							106

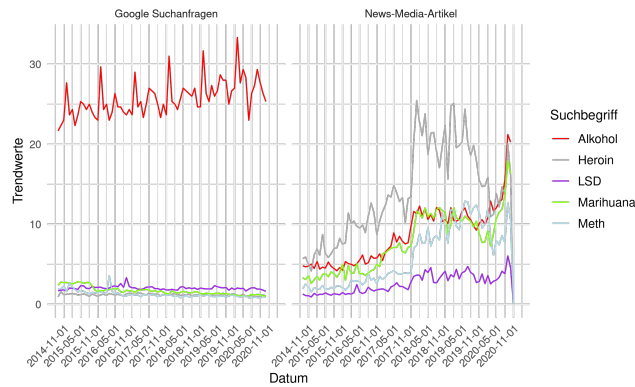
Stand: 17.07.2018 Seite 41 / 48

## Warum R ?

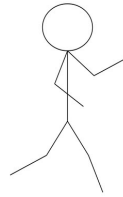
(...und nicht SPSS...)

	SPSS	R
Pro	einfache Bedienung weit verbreitet	das 'CRAN' (Comprehensive R Archive Network) kostenlos macht was angewiesen ist
Contra	kann nicht alles relativ kostenintensive Lizenzen nimmt vieles ab nicht beliebig erweiterbar	etwas Gewöhnung notwendig

Aber die viel wichtigeren Argumente: R kann **Alles**



R macht **S**p



## Literatur

Die Veranstaltung orientiert sich an:

1. Wollschläger (2016) . R kompakt.(Link aus dem Uni-Netz).
2. Golemund and Wickham (2017) . R for Data Science (Link).

## Installation & Verwendung

Es wird die Verwendung der grafischen Benutzeroberfläche RStudio empfohlen.

Beachten Sie, dass für die Verwendung von RStudio zuvor eine Basisinstallation von R erfolgen muss:

1. (R) herunterladen und installieren.
2. (RStudio) herunterladen und installieren.

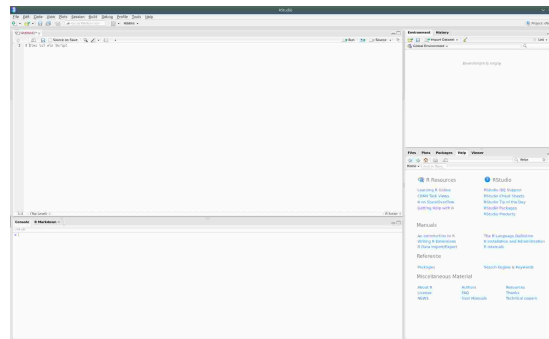


Figure 3.1: Benutzeroberfläche von RStudio. Oben links: Editor; unten links: Konsole; oben rechts: Environment bzw. History; unten rechts: Files, Plots, Help, etc.

## Benutzeroberfläche RStudio

### Allgemeine Hinweise

- Verwenden Sie die Konsole (unten links) nur für einzelzeilige Berechnungen beim “Ausprobieren”
- Verwenden Sie stets den Editor (oben links), um mehrzeilige Berechnungen direkt in ein Skript zu schreiben
- Kommentieren Sie Ihren Code ausreichend und sinnvoll mit Hilfe des #-Zeichens
- Speichern Sie Ihr Skript unter einem sinnvollen Namen in einem sinnvoll benannten Verzeichnis ab
- Speichern Sie regelmäßig mit Strg+S zwischen
- Eine einzelne Skript-Zeile (diejenige, in der sich der Cursor befindet) oder zuvor markierter Code lassen sich mit Strg+Enter ausführen
- In der Konsole bricht ESC die Eingabe ab

### Zum besseren Verständnis

In diesem Skript enthalten die grau hinterlegten Zeilen R-Input, die weiß hinterlegten Zeilen den R-Output. Ein ganz einfaches Beispiel zum Ausprobieren: Die simple Berechnung von  $1 + 1$ .

```
1 + 1
```

```
## [1] 2
```

### Ausdrücke in der R-Konsole

Anweisungen in R funktionieren grundsätzlich über das Ausführen von Ausdrücken. Dabei werden Ausdrücke entweder durch Semikolons oder Zeilenum-

brüche beendet.

```
1 + 1; 2 + 2;
```

```
## [1] 2
```

```
## [1] 4
```

```
1+1
```

```
## [1] 2
```

```
2+2
```

```
## [1] 4
```

### Kommentare

R bietet außerdem die Möglichkeit, im Code Anmerkungen zu machen, die beim Ausführen ignoriert werden. Diese werden mit einem `#`-Symbol eingeleitet.

```
1 + 1 ### +1 +1
```

```
## [1] 2
```

```
#Dies ist ein Kommentar
```

Nutzen Sie Kommentare innerhalb Ihrer Skripte, um Arbeitsschritte kenntlich zu machen und zu erklären. Die übersichtliche Gestaltung Ihrer Skripte ist von wirklich großem Vorteil bei der Arbeit mit R. Dies kann nicht oft genug betont werden.

## 3.3 Grundlegende Rechenoperationen

### Addition, Subtraktion

```
2 + 3
```

```
## [1] 5
```

```
28 - 5
```

```
## [1] 23
```

### Multiplikation, Division

```
2 * 21
```

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

```
92 / 4
```

```
## [1] 23
```

## Rechenregeln

```
1+1*1+1*(1+1)+1
```

```
## [1] 5
```

Wie man sieht, befolgt R die Punkt-vor-Strich-Regel und berücksichtigt Klammerung.

## Potenz, Quadratwurzel (“squareroot”), Betrag (“absolute”)

```
3^2
```

```
## [1] 9
```

```
sqrt(9)
```

```
## [1] 3
```

```
abs(-42)
```

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

## Runden

```
pi
```

```
## [1] 3.141593
```

```
round(pi)
```

```
## [1] 3
```

```
round(pi, digits=2)
```

```
## [1] 3.14
```

```
round(pi, digits=3)
```

```
## [1] 3.142
```

## Aufgabe

```
round(pi, digits = 0) * 3 ### + 5
```

Was kommt raus?

- A) pi
- B) 14

- C) eine Fehlermeldung
- D) 9
- E) NULL

## 3.4 Ausdrücke, Funktionen, Argumente

### Funktionen & Argumente

In R werden sehr häufig *Funktionen* verwendet. Diese repräsentieren eine Reihe von Anweisungen, die beim Aufrufen mit spezifischen Parametern ausgeführt werden sollen. Diese Parameter werden in Form von *Argumenten* übergeben. Beispielsweise enthält die Funktion `round()` die nötigen Anweisungen, um eine Zahl zu runden. Hierfür erwartet `round()` die zu rundende Zahl und die Anzahl an Nachkommastellen auf die zu runden ist. Man schreibt immer *Funktionsname(Argumentliste)*. Bei Funktionen müssen *immer* runde Klammern vorhanden sein, auch wenn keine einzelnen Argumente vorgegeben werden.

Es gibt *obligatorische Argumente*, ohne deren Übergabe das Aufrufen einer Funktion zu einer Fehlermeldung führt:

```
round(pi)

## [1] 3
round() ### Funktionsaufruf ohne Argument

## Error in eval(expr, envir, enclos): 0 arguments passed to 'round' which requires 1 or 2 arguments
... und optionale Argumente:
round(pi, digits=3)

## [1] 3.142
round(pi, digits=pi)

## [1] 3.142
round(pi, digits=15)

## [1] 3.141593

Gibt man den Namen eines Arguments nicht an, entscheidet die Position in der Liste über die
Interpretation des Arguments durch R. Achtung: Fehlerquelle!
round(1/42, 3)

## [1] 0.024
round(3, 1/42)

## [1] 3
```

## 3.5 Objekte

*Objekte* sind für den späteren Gebrauch mit einem Namen versehene und im Arbeitsspeicher abgelegte Ergebnisse von Ausdrücken. Dabei ist Objekt der Überbegriff für eine Vielzahl von möglichen Datenstrukturen.

Ein paar Beispiele für Datenstrukturen in R:

- eindimensionale Vektoren (vector)



- mehrdimensionale Matrizen (`matrix`)
- Funktionen(`function`)

## Objekte benennen

Wählen Sie kurze, aber aussagekräftige Objektnamen! Objektnamen dürfen dabei enthalten: Buchstaben, Zahlen, Punkte, Unterstriche

### Achtung:

- Immer mit einem Buchstaben beginnen
- Groß-/Kleinschreibung ist relevant
- Keine anderen Sonderzeichen
- Keine durch R reservierte Namen von Funktionen, Konstanten, etc. (z.B. “mean”, “pi”, “if”, etc.) (im Zweifel Überprüfen mit `exists()`)

Hier nochmal der nachdrückliche Hinweis: Tun Sie sich selbst den Gefallen, Ihre Objekte eindeutig und nachvollziehbar zu benennen!

## Zuweisungen an Objekte

Ergebnisse von Ausdrücken können benannten Objekten zugewiesen werden.

Dabei sind folgende Ausdrücke äquivalent:

```
firstObject = 42
42 -> firstObject
firstObject <- 42
```

Die letzte Möglichkeit stellt dabei die Beste im Hinblick auf Übersichtlichkeit und Eindeutigkeit dar.

## Verwenden von Objekten:

Die Objektnamen können dann synonym zu ihrem Inhalt verwendet werden.

```
firstObject + 1; 42 + 1;
```

```
## [1] 43
```

```
## [1] 43
```

## Objekte ausgeben

Um diese Ausgabe nachzuholen gibt es folgende Möglichkeiten:

```
print(firstObject)
```

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

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

Diese beiden Versionen sind faktisch dieselbe, da das einfache Aufrufen eines Variablennamens implizit als ein Aufruf von `print()` interpretiert wird.

```
(object2 <- firstObject^2)
```

```
## [1] 1764
```

Bei Setzen eines Befehls in Klammern wird die durch ihn ausgelöste Änderung ausgegeben, im Beispiel die Zuweisung des Ergebnisses zum neuen Objekt `object2`.

Diese Methode ist eine gute Variante, Zwischenergebnisse regelmäßig zu kontrollieren.

## Objekte anzeigen lassen

Alle Objekte im Workspace anzeigen lassen:

```
ls()
```

```
## [1] "a"          "firstObject" "object2"
## [4] "plan"
```

Diese Operation braucht man später nicht unbedingt, da alle angelegten Objekte auch im Environment-Tab in RStudio einsehen kann. Am Anfang kann diese Funktion aber helfen, sich über die Abläufe klar zu werden.

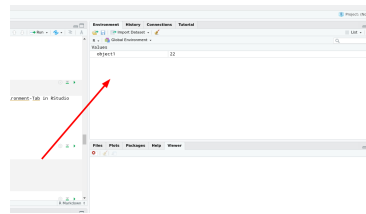


Figure 3.2: Environment

## Objekte entfernen

Vorhandene Objekte lassen sich dann wie folgt entfernen:

```
ls()
```

```
## [1] "a"          "firstObject" "object2"
## [4] "plan"
rm(object2)
ls()
```

```
## [1] "a"          "firstObject" "plan"
```

Mit `rm(list=ls())` lassen sich alle Objekte aus dem Workspace entfernen.

```
ls()
```

```
## [1] "a"          "firstObject" "plan"
rm(list=ls())
ls()
```

```
## character(0)
```

## Datentypen

In R, wie in so gut wie jeder anderen Sprache, werden Objekte in unterschiedliche Subtypen gegliedert, die sich auf die in ihnen gespeicherten Informationen beziehen:

Beschreibung	Beispiel	Datentyp
leere Menge	'NULL'	'NULL'
logische Werte	'TRUE, FALSE, T, F'	'logical'
ganze und reelle Zahlen	'42'	'numeric'
Buchstaben- o. Zeichenfolgen (immer in   Anführungszeichen)	'beware of the leopard.'	'character'

Dabei ist das hier keine vollständige Liste, für den Anfang reicht sie aber.

`mode()` gibt den Datentyp des übergebenen Arguments aus (braucht man selten, hier nur für das Beispiel):

```
mode(answer)
```

```
## [1] "numeric"
```

```
mode('answer')
```

```
## [1] "character"
```

## Datentypen konvertieren

`as.character(answer)` konvertiert den Datentyp des Objekts von `numeric` nach `character` ohne den ursprünglichen Eintrag von `answer` zu überschreiben.

```
mode(answer)
```

```
## [1] "numeric"
```

```
as.character(answer)
```

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

```
mode(answer)
```

```
## [1] "numeric"
```

Um das zu erreichen muss das Objekt überschrieben werden:

```
answer <- as.character(answer)
```

```
mode(answer)
```

```
## [1] "character"
```

Mit `answer` als `character`-Element lässt sich nicht mehr rechnen:

```
answer * 2
```

```
## Error in answer * 2: non-numeric argument to binary operator
```

Um das dann wieder zu ermöglichen muss das Objekt zurück nach `numeric` konvertiert werden:

```
answer <- as.numeric(answer)
```

```
mode(answer)
```

```
## [1] "numeric"
```

```
answer * 2
```

```
## [1] 84
```

Weitere Beispiele für Konvertierung:

```
as.numeric("42") ### konvertiert character nach numeric
```

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

```
as.numeric(TRUE) ### konvertiert logical nach numeric
```

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

```
as.logical(0) ### konvertiert numeric nach logical
```

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

```
as.logical(1) ### konvertiert numeric nach logical
```

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

```
as.logical(23) ### konvertiert numeric nach logical
```

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

```
as.logical("true") ### konvertiert character nach logical
```

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

## Logische Werte, Operatoren und Verknüpfungen

Logische Vergleiche, Verknüpfungen und andere Operatoren:

Operator	Operation
'=='	ist gleich
'!='	ist ungleich
'>'	ist größer
'>='	ist größer gleich
'<'	ist kleiner
'<='	ist kleiner gleich
'!'	logisches NICHT
'&'	logisches UND
' '	logisches ODER
'isTRUE()'	gibt an, ob übergebenes Argument TRUE ist

Das Ergebnis eines logischen Vergleichs sind logische Werte:

WAHR: TRUE = T = 1

FALSCH: FALSE = F = 0

Beispiele:

```
1 == 2
```

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

```
1 != 2
```

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

```
1 < 2
```

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

```
1 >= 2
```

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

```
1>2 & 1<=3
```

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

```
2>1 | 1!=1
```

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

```
6>5 & !(2<=1)
```

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

```
isTRUE(1 == 1)
```

```
## [1] TRUE
(1 == 1)
```

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

## Aufgabe

Was kommt raus?

```
(2 > 1 & 1 < 3) | 1 != 1
```

- A) TRUE
- B) FALSE
- C) NULL

## Umgang mit Dezimalzahlen:

Was kommt hier raus?

```
0.1 + 0.2 == 0.3
```

- A) TRUE
- B) FALSE
- C) NULL

```
0.1 + 0.2 == 0.3
```

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

**0.1 + 0.2 != 0.3?**

‘Falsches’ Ergebnis ist Resultat von Repräsentation von Gleitkommazahlen im Speicher des Rechners.

Die Funktion `all.equal()` löst dieses Problem.

```
all.equal(target=0.1+0.2, current=0.3)
```

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

Mit dem `tolerance`-Argument lässt sich der Bereich der akzeptablen Unterschiede in Dezimalstellen angeben.

```
all.equal(target = 0.424242, current = 0.424243,
          tolerance = 1e-5)
```

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

```
all.equal(target = 0.424242, current = 0.424243,
          tolerance = 1e-6)
```

```
## [1] "Mean relative difference: 2.357145e-06"
```

Hierbei fällt auf, dass bei Ungleichheit nicht `FALSE` sondern die Abweichung ausgegeben wird.

Um `all.equal` sinnvoll in logischen Operationen benutzen zu können wird `isTRUE` benötigt:

```
isTRUE(all.equal(target = 0.424242,
                  current = 0.424243,
                  tolerance = 1e-6))
```

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

## 3.6 Hausaufgabe

### Hausaufgabe: Erstellen eines R-Skripts

Schreiben Sie den dem folgenden Ablauf entsprechenden Code in ein R-*Skript* und führen Sie ihn von dort in der Konsole aus:

Erstellen Sie drei Objekte wie folgt:

- Als erstes ein Objekt namens *whatDoIDoThis* mit der Zahl 4 als Inhalt.
- Als zweites ein Objekt namens *text* mit dem Inhalt : “i\_like\_snake\_case\_better”.
- Als drittes ein Objekt namens *myFavouriteNumber* mit einer Zahl Ihrer Wahl als Inhalt.

Berechnen Sie nun den Mittelwert der Objekte mit numerischem Inhalt und legen Sie diesen in einem weiteren Objekt namens *manualMean* ab.

Lassen Sie sich in der Konsole durch eine Zeile in Ihrem Skript den Text 'I learned about the most important bugfixing tool' ausgeben.

Speichern Sie anschließend das R-*Skript* unter 'R' ab.



## Chapter 4

# elementare Datenverarbeitung

### 4.1 Organisatorisches

#### 4.1.1 Semesterplan

Einheit	Vorlesung	Übungswoche	Thema
1	2.11.20	keine Übung	Grundlagen und Begriffe
2	16.11.20	KW 48	Vektoren und Indizierung
			Datenformate erstellen und transformieren
3	30.11.20	KW 50	Pakete installieren und benutzen
			Datensätze erstellen und ergänzen können
			Datensätze sortieren und indizieren können
4	14.12.20	KW 1	Faktoren
			deskriptive Kennwerte
			Aggregation I
5	11.01.21	KW 3	Aggregation II
			In- und Export von Datensätzen
6	25.01.21	KW 5	Grafische Darstellungen I
7	08.02.21	KW 7	Grafische Darstellungen II
8	22.02.21	keine Übung	Puffer
			Probeklausur



## 4.2 Vektoren

### 4.2.1 Begriff

Im Kontext von R ist ein Vektor als eine sequentiell geordnete Menge von Werten und nicht als das gleichnamige mathematische Konzept zu verstehen.

### 4.2.2 Vektoren erzeugen

Leere Vektoren eines bestimmten Typs lassen sich mit dem Namen des Typs als Funktion und der Anzahl der gewünschten Stellen als Argument erstellen. z.B.:

```
numeric(5);
character(4);
logical(3);

## [1] 0 0 0 0 0
## [1] "" "" "" ""
## [1] FALSE FALSE FALSE
```

### 4.2.3 Vektoren erzeugen

Um mehrere Daten in einer eindimensionalen Anordnung zu verketteten wird die `c()`-Funktion benutzt. Die Argumente werden in Reihenfolge der Eingabe hintereinander angeordnet und können beliebigen Datentypen angehören.

```
c(1,2,3,4);
c('dies', 'ist', 'ein', 'Vektor');
c(T, F, T, F);

## [1] 1 2 3 4
## [1] "dies" "ist" "ein" "Vektor"
## [1] TRUE FALSE TRUE FALSE
```

### 4.2.4 Vektoren erzeugen

Das Ergebnis kann dann wie gewohnt in ein Objekt abgelegt werden.

```
numericVector <-c(4,2,4242,42)
```

Außerdem lassen sich mit dem `c()`-Operator mehrere Vektoren kombinieren.

```
additionToNumericVector <- c(424242, 42400, 42000,
                             4224, 24)
(numericVector <- c(numericVector,
                   additionToNumericVector))
```

```
## [1]      4      2  4242    42 424242  42400  42000
## [8]  4224    24
```

### 4.2.5 Vektoren verwenden

Die Länge eines Vektors lässt sich mit der Funktion `length()` ausgeben.

```
length(numericVector)
```

```
## [1] 9
```

### 4.2.6 Datentypen in Vektoren

Bei dem Versuch Vektoren aus verschiedenen Datentypen anzulegen werden die Daten in den allgemeinsten Datentyp umgewandelt. Dabei gilt im Rahmen der Komplexität für die bisher vorgestellten Datentypen:

```
logical < numeric < character

mode(c(T, T, F));
mode(c(T, T, 0));
mode(c(T, 0, 'false'));

## [1] "logical"
## [1] "numeric"
## [1] "character"
```

In einem Vektor ist im Allgemeinen also immer nur ein Typ an Daten vertreten.

### 4.2.7 Aufgabe

Wenn ich `mode(c('TRUE',FALSE,1))` eingebe, dann...

- A) ... wird `logical` ausgegeben
- B) ... wird `vector` ausgegeben
- C) ... wird `numerical` ausgegeben
- D) ... wird `character` ausgegeben

## 4.3 Indizierung

### 4.3.1 Elemente indizieren

Die beim Erstellen eines Vektors angelegten Positionen der Werte werden in R implizit mit fortlaufenden Indizes versehen und gespeichert. Diese Indizes starten bei jedem Vektor mit 1 und enden mit der Länge desselben. Die einzelnen Elemente eines Vektors lassen sich über ihren Index mit dem `[]` - Operator aufrufen.

```
numericVector[4] ## 4. Element des Vektors numericVector.

## [1] 42
```

### 4.3.2 Elemente indizieren

Wird ein Index über dem des letzten Eintrags eines Vektors aufgerufen, wird `NA` zurückgegeben.

```
numericVector[length(numericVector)+1]

## [1] NA
```

### 4.3.3 mehrere Elemente gleichzeitig indizieren

Es lassen sich auch mehrere Werte eines Vektors über die Indizierung über Zuhilfenahme eines anderen Vektors aufrufen. Dabei kann der Index-Vektor als Objekt vordefiniert oder dem `[]`-Operator direkt übergeben werden.

```
idx <- c(1,2,3,8)
numericVector[idx]

## [1] 4 2 4242 4224
```

```
numericVector[c(4,5,6,7)]

## [1]      42 424242 42400 42000
```

#### 4.3.4 mehrere Elemente gleichzeitig indizieren

Der Index-Vektor kann dabei auch länger als der ursprüngliche Vektor sein, da mehrfacher Aufruf eines Index möglich ist.

```
numericVector ## 9 Werte

## [1]      4      2 4242      42 424242 42400 42000
## [8] 4224      24

idx <- c(1,1,2,2,3,3,4,4,5,5,6,6)
## 12 Aufrufe über die Indizes
numericVector[idx]

## [1]      4      4      2      2 4242 4242      42
## [8] 42 424242 424242 42400 42400
```

#### 4.3.5 Elemente ausschließen

Durch das verwenden negativer Indizes wird das entsprechende Element von der Ausgabe ausgeschlossen.

```
numericVector[-3] ## Vektor ohne drittes Element

## [1]      4      2      42 424242 42400 42000 4224
## [8]      24

idx <- c(1,3,5,7,9)
## Vektor mit Ausnahme der in idx abgelegten Indizes:
numericVector[-idx]

## [1]      2      42 42400 4224
```

#### 4.3.6 Elemente austauschen

Die Indizierung kann außerdem genutzt werden um Elemente eines Vektors zu ersetzen oder als alternative Methode zu oben vorgestelltem Kombinieren von Vektoren via `c(<Vektor1>, <Vektor2>)`.

```
numericVector

## [1]      4      2 4242      42 424242 42400 42000
## [8] 4224      24

numericVector[1] <- 12
numericVector

## [1]     12      2 4242      42 424242 42400 42000
## [8] 4224      24
```

#### 4.3.7 Elemente austauschen

```
numericVector[idx] <-idx
numericVector

## [1]      1      2      3      42      5 42400      7 4224
## [9]      9
```

```
length(numericVector)

## [1] 9
numericVector[c(10,11,12,13,14)] <- idx
numericVector

## [1] 1 2 3 42 5 42400 7 4224
## [9] 9 1 3 5 7 9
```

### 4.3.8 Elemente löschen

Elemente eines Vektors lassen sich nicht im eigentlichen Sinne löschen, man kann aber sehr wohl das Objekt in dem der Vektor abgelegt ist mit einer verkürzten Version überschreiben.

```
numericVector <- numericVector[-idx]
numericVector

## [1] 2 42 42400 4224 1 3 5 7
## [9] 9
```

### 4.3.9 Logische Operatoren

Verarbeitungsschritte mit logischen Operatoren treten häufig bei der Auswahl von Teilmengen von Daten sowie der Recodierung von Datenwerten auf, zwei häufigen Prozeduren in der statistischen Auswertung

### 4.3.10 Logischer Vergleich von Vektoren

Wie vorher Einzelwerte kann man auch Vektoren in logischen Vergleichen verwenden.

```
age <- c(17, 30, 30, 24, 23, 21)
age < 24

## [1] TRUE FALSE FALSE FALSE TRUE TRUE
age >= 18

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

Dabei werden logische Werte als Ergebnis für den Vergleich jeden Wertes ausgegeben.

### 4.3.11 Logische Operatoren

Die vorher gezeigten logischen Verknüpfungen lassen sich genauso anwenden

```
## Alle Werte die mindestens 18 und kleiner als 30 sind
(age >= 18) & (age < 30)

## [1] FALSE FALSE FALSE TRUE TRUE TRUE
## Alle Werte die kleiner als 18 oder mindestens 30 sind
(age < 18) | (age >= 30)

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

### 4.3.12 Logische Vektoren

Die mit `sum()` gebildete Summe eines logischen Vektors gibt einem die Anzahl der wahren Werte im Vektor aus, da `TRUE` für die Berechnung in eine 1 und `FALSE` in eine 0 transformiert wird.

```
res <- !((age < 18) | (age >= 30))
sum(res)

## [1] 3
```

### 4.3.13 Logische Vergleiche von Vektoren

Zwei **gleichlange** Vektoren lassen sich auch mit Hilfe logischer Operatoren vergleichen.

```
age2 <- c(19, 31, 29, 24, 30, 22)
age == age2

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

### 4.3.14 Logische Vergleiche von Vektoren

Und natürlich lassen sich hier alle vorher besprochenen logischen Operatoren anwenden:

```
age == age2

## [1] FALSE FALSE FALSE TRUE FALSE FALSE
age < age2

## [1] TRUE TRUE FALSE FALSE TRUE TRUE
age != age2

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

### 4.3.15 Logische Indizierung

Indizierung funktioniert auch mit logischen Vektoren. Dabei wird im Indexvektor für jeden Wert des indizierten Vektors angegeben, ob dieser ausgewählt werden soll oder nicht. Eine einfache Methode zur Auswahl von Teilmengen von Elementen die einem bestimmten Kriterium entsprechen.

```
(res <- age < 24)

## [1] TRUE FALSE FALSE FALSE TRUE TRUE
age[res]

## [1] 17 23 21
```

### 4.3.16 Logische Indizierung

Der Indexvektor muss nicht vorher als Objekt angelegt werden.

```
age[age<24]

## [1] 17 23 21
```

### 4.3.17 Logische Indizierung und fehlende Werte

Versucht man Vektoren mit fehlenden Werten zu erzeugen, stößt man auf folgendes Problem:

```
age3 <- c(20, 23, 32, NA, 19, 27)
(idx <- age3 < 24)

## [1] TRUE TRUE FALSE NA TRUE FALSE
```

```
age3[idx]

## [1] 20 23 NA 19
```

### 4.3.18 Umgang mit fehlenden Werten bei logischer Indizierung

Der fehlende Wert wird in den Index-Vektor und die Indizierung weitergetragen. Umgehen lässt sich dieses Problem mit der `which()`-Funktion, die die Positionen aller TRUE-Werte des ihre übergebenen Arguments als numerischen Vektor ausgibt.

```
(idx <- which(idx))
```

```
## [1] 1 2 5
```

Dieser kann dann wieder zur Indexierung benutzt werden.

```
age3[idx]
```

```
## [1] 20 23 19
```

## 4.4 Systematische Wertefolgen erzeugen

### 4.4.1 Numerische Sequenzen erstellen

In R lassen sich durch einen Doppelpunkt Zahlensequenzen in Einserschritten zwischen einem Start- und Endwert erstellen.

```
1:20
20:1
-10:10
-(1:20)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## [17] 17 18 19 20
## [1] 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
## [17] 4 3 2 1
## [1] -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1
## [13] 2 3 4 5 6 7 8 9 10
## [1] -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12
## [13] -13 -14 -15 -16 -17 -18 -19 -20
```

### 4.4.2 Numerische Sequenzen mit definierter Schrittgröße

Sequenzen mit anderen Schrittgrößen lassen sich mit der `seq()`-Funktion erstellen.

Dabei lässt sich entweder die Schrittgröße angeben:

```
seq(from = 0, to = 42, by = 6)
```

```
## [1] 0 6 12 18 24 30 36 42
```

```
seq(from = 0, to = 42, by = 5) ## Endpunkt nicht erreicht
```

```
## [1] 0 5 10 15 20 25 30 35 40
```

### 4.4.3 Numerische Sequenzen mit definierter Länge

Oder die gewünschte Anzahl der Werte in der Sequenz:

```
seq(from = 0, to = 42, length.out = 8)
```

```
## [1] 0 6 12 18 24 30 36 42
```

```
seq(from = 0, to = 42, length.out = 6)
```

```
## [1] 0.0 8.4 16.8 25.2 33.6 42.0
```

### 4.4.4 Numerische Sequenzen

Bei zweiterer Methode oder bei Angabe einer nicht-ganzzahligen Schrittgröße können auch nicht zur Indizierung geeignete Dezimalzahlen entstehen.

```
seq(from = 0, to = 1, by = 0.1)
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

### 4.4.5 Numerische Sequenzen mit definierter Länge

Mit dem `along` Argument lässt sich außerdem eine Sequenz in der Länge eines übergebenen Vektors erstellen.

```
age <- c(16, 43, 30, 22, 7, 36)
seq(along = age)
```

```
## [1] 1 2 3 4 5 6
```

```
seq(from = 10, to = 100, along = age)
```

```
## [1] 10 28 46 64 82 100
```

### 4.4.6 Wertefolgen wiederholen

```
someValues <- c(42, 16, 12)
## Vektor 5-mal wiederholen
rep(someValues, times = 5)
```

```
## [1] 42 16 12 42 16 12 42 16 12 42 16 12 42 16 12
```

```
## Jeden Wert des Vektors 5 mal wiederholen
rep(someValues, each = 5)
```

```
## [1] 42 42 42 42 42 16 16 16 16 16 12 12 12 12 12
```

```
## Jeden Wert so oft wie in times individuell angegeben wiederholen
rep(someValues, times = c(42, 1, 5))
```

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

```
## [17] 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42
```

```
## [33] 42 42 42 42 42 42 42 42 42 42 16 12 12 12 12
```

## 4.5 Daten transformieren

### 4.5.1 Werte sortieren

`sort()` sortiert je nach Angabe in auf- oder absteigender Reihenfolge

```
sort(someValues, decreasing = FALSE)
```

```
## [1] 12 16 42
```

```
sort(someValues, decreasing = TRUE)
```

```
## [1] 42 16 12
```

### 4.5.2 Werte sortieren

`order()` macht im Prinzip dasselbe, gibt aber statt des sortierten Vektors die Indizes in entsprechender Ordnung aus.

```
someValues
```

```
## [1] 42 16 12
```

```
(idx <- order(someValues, decreasing = F))
```

```
## [1] 3 2 1
```

```
someValues[idx]
```

```
## [1] 12 16 42
```

### 4.5.3 Werte sortieren

Dies funktioniert auch für `character`-Vektoren in alphabetischer Ordnung

```
(someCharacters <- c("Z", "D", "L", "O", "I", "n", "e", "N", "t", "R"))
```

```
## [1] "Z" "D" "L" "O" "I" "n" "e" "N" "t" "R"
```

```
sort(someCharacters, decreasing = F)
```

```
## [1] "D" "e" "I" "L" "n" "N" "O" "R" "t" "Z"
```

### 4.5.4 Werte sortieren

Sind Zahlen an der ersten Stelle in `character`-Vektoren vertreten, werden diese vor das Alphabet sortiert.

```
someCharacters <- c(
  "42 is fairly overused",
  "India",
  "Zulu",
  "Whiskey",
  "42",
  "a string of characters",
  "Tango",
  "not a number",
  "1"
)
sort(someCharacters, decreasing = F)
```

```
## [1] "1"
```

```
"42"
```

```
## [3] "42 is fairly overused" "a string of characters"
```

```
## [5] "India"
```

```
"not a number"
```



```
## [7] "Tango"           "Whiskey"  
## [9] "Zulu"
```

## 4.6 Einfache deskriptiv-statistische Kennwerte

### 4.6.1 Einfache deskriptiv-statistische Kennwerte

```
age <- c(6, 60, 44, 56, 8, 58, 87, 8, 55, 83)  
IQ <- c(91, 104, 109, 92, 90, 101, 99, 93, 89, 118)  
  
mean(age) ## Mittelwert  
var(age)  ## Varianz (korrigiert)  
sd(age)   ## Streuung (korrigiert)  
  
## [1] 46.5  
## [1] 895.6111  
## [1] 29.92676
```

### 4.6.2 Einfache deskriptiv-statistische Kennwerte

```
N <- length(age)  
sd(age)/sqrt(N) ## SEM  
  
## [1] 9.463673  
sqrt((N-1) / N) * sd(age) ## unkorrigierte Streuung  
  
## [1] 28.39102  
cov(x=age, y=IQ, method="pearson") ## Kovarianz  
  
## [1] 167.6667  
cor(x=age, y=IQ, method="pearson") ## Korrelation  
  
## [1] 0.5875238
```



# Bibliography

Grolemund, G. and Wickham, H. (2017). *R for Data Science*.

Wollschläger, D. (2016). *R kompakt: Der schnelle Einstieg in die Datenanalyse*. Springer-Lehrbuch. Springer Spektrum, second edition.