

R-Vorkurs

Teil 1

Jens Klenke

04.10.2021

Übersicht

1. und RStudio
2. Grundlagen
3. Vektoren
4. Matrizen

# und RStudio

R / ist eine freie Programmiersprache für statistische Berechnungen/Grafiken und wurde von Statistikern für Anwender mit statistischen Aufgaben entwickelt.

Was ist RStudio? Brauche ich das?

RStudio ist eine integrierte Entwicklungsumgebung und grafische Benutzeroberfläche für .

RStudio bietet eine Menge an Komfortfunktionen, die das Arbeiten mit vereinfachen.

⇒ **Wir arbeiten daher mit RStudio**

**Wichtig: RStudio ist nicht gleich ! Ohne eine -Installation bringt RStudio nichts.**

## Grundrechenarten

kann als einfacher Taschenrechner benutzt werden.

5

+

7

2

-

8

12

\*

12

8

/

3

## [1] 12

## [1] -6

## [1] 144

## [1] 2.666667

## ... und mehr

#

"

1

2

h

o

c

h

2

"

12

^

2

#

L

o

g

a

r

i

t

h

m

u

s

v

o

n

1

4

4

z

u

r

B

a

s

i

s

1

2

log(

144

, base =

12

)

#

"

e

h

o

c

h

3

"

exp(

3

)

#

"

L

o

g

a

r

i

t

h

m

u

s

v

o

n

e

h

o

c

h

3

"

log(exp(

3

))

#

"

S

i

n

u

s

v

o

n

2

P

i

"

sin(

2

\* pi)

## [1] 144

## [1] 2

## [1] 20.08554

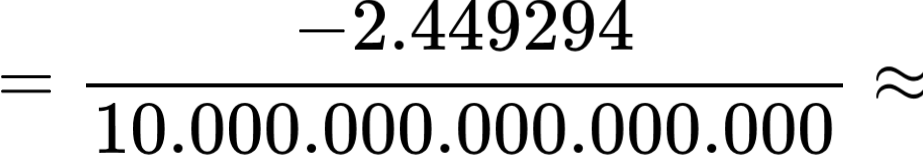
## [1] 3

## [1] -2.449213e-16

## Wissenschaftliche Notation

stellt besonders große bzw. besonders kleine Zahlen mit Hilfe der [wissenschaftlichen Notation](https://de.wikipedia.org/wiki/Wissenschaftliche_Notation) dar.

Die Zahl -2.449294e-16 aus dem -Output ist



(Auf den zweiten Blick ist die wissenschaftliche Notation also tatsächlich nutzerfreundlicher.)

## Variablen

Variablen werden mit dem Zuweisungsoperator <- ("Kleinerzeichen" gefolgt von "Minuszeichen") erzeugt bzw. überschrieben.

x <-

13

y <-

24

Mit den so erzeugten Variablen können wir natürlich auch rechnen.

x - y

## [1] -11

Achtung — Häufiger Fehler:

Vergessen von \*, wenn eine Variable mit einem Skalar multipliziert werden soll.

#

R

i

c

h

t

i

g

:

3

\* x

## [1] 39

#

F

a

l

s

c

h

:

3

x

**Problem:**  interpretiert 3x als Variable, jedoch dürfen Variablennamen nicht mit Ziffern beginnen! Wir erhalten eine Fehlermeldung!

## Kommentare

Auf der vorherigen Folie steht im letzten Codesnippet eine kurze Erklärung angeführt von einem #. Dies kennzeichnet einen [Kommentar](https://de.wikipedia.org/wiki/Kommentar_(Programmierung)):

Kommentare werden beim Ausführen des Codes nicht berücksichtigt. Wir nutzen Kommentare, um unseren Code (für uns und andere) verständlicher zu machen.

## Datentypen

Neben den Zahlen gibt es in noch weitere Datentypen. Die für uns Wichtigsten (Zahlen eingeschlossen) sind:

a <-

5

#

n

u

m

e

r

i

c

(

Z

a

h

l

e

n

)

class(a)

## [1] "numeric"

b <-

"R ist toll!"

#

c

h

a

r

a

c

t

e

r

(

S

c

h

r

i

f

t

z

e

i

c

h

e

n

)

class(b)

## [1] "character"

## Datentypen

d <-

TRUE

#

l

o

g

i

c

a

l

(

T

R

U

E

/

F

A

L

S

E

)

class(d)

## [1] "logical"

**Häufiger Fehler:**

Vergessen von Anführungszeichen bei Schriftzeichen und Kleinschreiben von TRUE oder FALSE.

## Erzeugen eines Vektors

Bisher haben wir nur Variablen betrachtet, die einen einzigen Wert enthalten. Die große Stärke von sind vektor- und matrixbasierte Funktionen.

Ein Vektor wird gewöhnlich mit der Funktion c() (für **c**ombine) erzeugt und besteht aus einem oder mehreren Elementen **eines** Datentyps.

numeric\_v <- c(

1

,

4

,

8

)

character\_v <- c(

"Pommes"

,

"Falafel"

,

"Ketchup"

)

logical\_v <- c(

TRUE

,

FALSE

,

FALSE

)

#

o

d

e

r

k

ü

r

z

e

r

`

T

`

f

ü

r

`

T

R

U

E

`

u

n

d

`

F

`

f

ü

r

`

F

A

L

S

E

`

logical\_v <- c(

T

,

F

,

F

)

**Achtung:** T und F sollten nicht als Variablen verwendet werden (warum?). Die lange Version ist außerdem weniger fehleranfällig.

## Erzeugen eines Vektors

Insbesondere für numerische Vektoren gibt es ein paar hilfreiche Shortcuts: Jede der folgenden Zeilen liefert denselben Output.

#

#

c

(

1

,

2

,

3

,

4

,

5

,

6

,

7

,

8

,

9

,

1

0

)

#

#

1

:

1

0

seq(from =

1

, to =

10

, by =

1

)

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

## Missing Values

Bei "echten" Daten kommt es häu g vor, dass es zu einigen Beobachtungen keine Daten für manche Variablen gibt. **Praxisbeispiel**

Angenommen in einer Umfrage werden die Variablen Alter, Geschlecht und Körpergröße erhoben.

Problem: Eine Person hat keine Angabe zur Körpergröße gemacht.

In kann der fehlende Wert speziell codiert werden: Anstelle von einer −99 oder 0 (sieht man oft in echten Datensätzen) wird für den fehlenden Wert ein NA eingesetzt.

Für viele Funktionen kann das Vorgehen bei Missing Values festlegt werden (siehe auch Übungsaufgabe 5).

## Missing Values

Beim Programmieren werden i.d.R. englische Namen verwendet. Meist sind diese kürzer und man vermeidet Probleme aufgrund von Umlauten.

age <- c(

25

,

28

,

29

)

sex <- c(

"m"

,

"f"

,

"m"

)

height <- c(

184

,

165

,

NA

)

### Subsetting von Vektoren

Um eine Untermenge der Elemente eines Vektors anzusprechen (Subsetting genannt), nutzen wir in eckigen Klammern eingeschlossene Indizes.

numeric\_v[2] # Gibt das 2. Element von numeric\_v aus

## [1] 4 character\_v[c(1, 3)] # Gibt das 1. und 3. Element von character\_v aus

## [1] "Pommes" "Ketchup" logical\_v[1:2] # Gibt das 1. und 2. Element von logical\_v aus

## [1] TRUE FALSE

### Subsetting von Vektoren

**Häufiger Fehler:** Vergessen von c() im 2. Fall. Mehrere Indizes müssen als Vektor übergeben werden.

Wir können auch logische Vektoren für das Subsetting nutzen. Diese müssen dieselbe Länge haben wie der Vektor der "gesubsetted" werden soll. Es werden die Elemente ausgegeben, an deren Stellen der logische Vektor den Wert TRUE hat.

x <- c(

7

,

2

,

5

,

3

,

9

)

#

E

l

e

m

e

n

t

e

2

u

n

d

5

s

o

l

l

e

n

a

n

g

e

s

p

r

o

c

h

e

n

w

e

r

d

e

n

hv <- c(

F

,

T

,

F

,

F

,

T

)

x[hv]

## [1] 2 9

Das ist hilfreich, da viele Funktionen einen logischen Vektor zurückgeben (z. B. is.na() und logische Operatoren, später mehr dazu).

### Ändern von Elementen

Manchmal müssen Elemente eines Vektors geändert werden. Dazu nutzen wir Subsetting zusammen mit dem Zuweisungsoperator.

**Beispiel**: Ändere das dritte Element des Vektors x

x <- c(

7

,

2

,

5

,

3

,

9

)

x[

3

]

<-

49

Wenn wir mehrere Elemente ändern wollen:

x[c(1, 5)] <- c(25, 14) x[c(2, 4)] <- -1 # Setzt das 2. und das 4. Element gleich -1

Wie sieht der erzeugte Vektor aus?

Wir betrachten nun den Vektor age. Dieser enthält das Alter von 5 Studierenden in diesem Kurs.

age <- c(

24

,

25

,

29

,

23

,

26

)

Wie können wir das Durchschnittsalter berechnen?

**Antwort:**

age\_sum <- age[

1

]

+ age

[

2

]

+ age

[

3

]

+ age

[

4

]

+ age

[

5

]

age\_sum /

5

## [1] 25.4

Dieses Vorgehen ist für kleine Vektoren machbar, aber umständlich. Für große Vektoren ist es sehr nervig und nicht mit zeitlichen Restriktionen vereinbar!

Bessere Alternative: Vordenierte Funktionen nutzen:

#

M

i

t

t

e

l

w

e

r

t

b

e

r

e

c

h

n

e

n

age\_sum <- sum(age)

age\_sum / length(age)

## [1] 25.4

Oder noch einfacher:

mean(age)

## [1] 25.4

### I Need Help

Damit haben wir schon einige -Funktionen kennengelernt. Oft ist unklar, wie diese Funktionen funktionieren.

⇒ -Hilfe nutzen:

Angenommen wir sind nicht sicher, was mean() genau berechnet oder wie die Funktion aufgerufen werden muss.

-Hilfe ist insb. für Anfänger manchmal schwer zu verstehen...

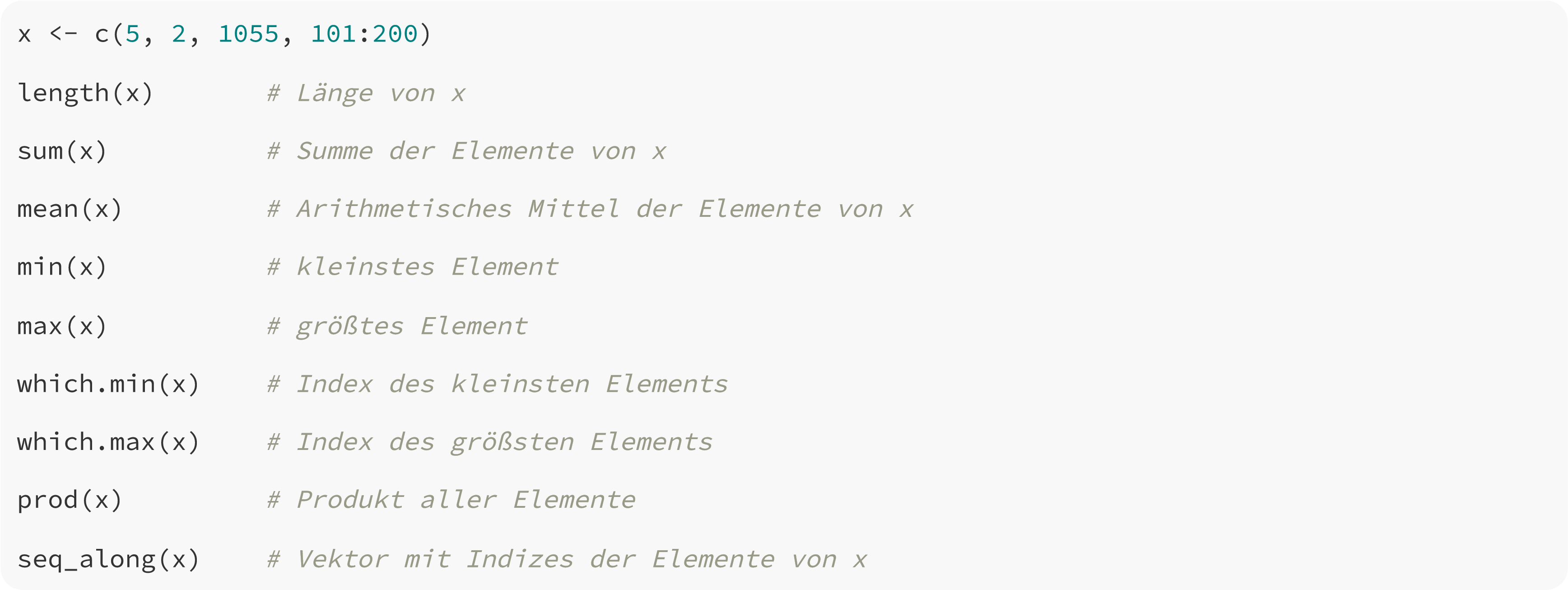
?mean

Die

Nicht verzweifeln, sondern einfach ein bisschen rumprobieren oder eine Suchmaschine bemühen!

**Zwei extrem wichtige Fähigkeiten beim Programmieren, die gleich schon mal geübt werden können!**

### Vektorbasierte Funktionen — Eine kleine Übersicht



Vektoren

Übungsaufgaben zu Vektoren

1

.

Erzeuge einen Vektor

numbers

mit den Elementen

Hinweis:

Die Zahl

ist in

bereits als

pi

vorde niert.

2

.

Berechne das arithmetische und das harmonische Mittel von

numbers

.

Hinweis:

Für einen numerischen Vektor

der Länge

ist das arithmetische Mittel

und das harmonische Mittel

.

3

.

Wir vermuten, dass die höchste und die niedrigste Zahl zu verzerrten Ergebnissen führen und

entscheiden darum, diese Werte zu ignorieren. Ersetze beide Werte durch

NA

und berechne die

Mittelwerte aus Aufgabe 2 erneut.

{

4

,

6

,

−

3

,

2.5

,

18

,

π

,

85

}

π

X

n

¯

¯¯¯

¯

X

=

∑

n

i

=

1

X

i

1

n

¯

¯¯¯

¯

X

h

a

r

m

=

n

∑

n

i

=

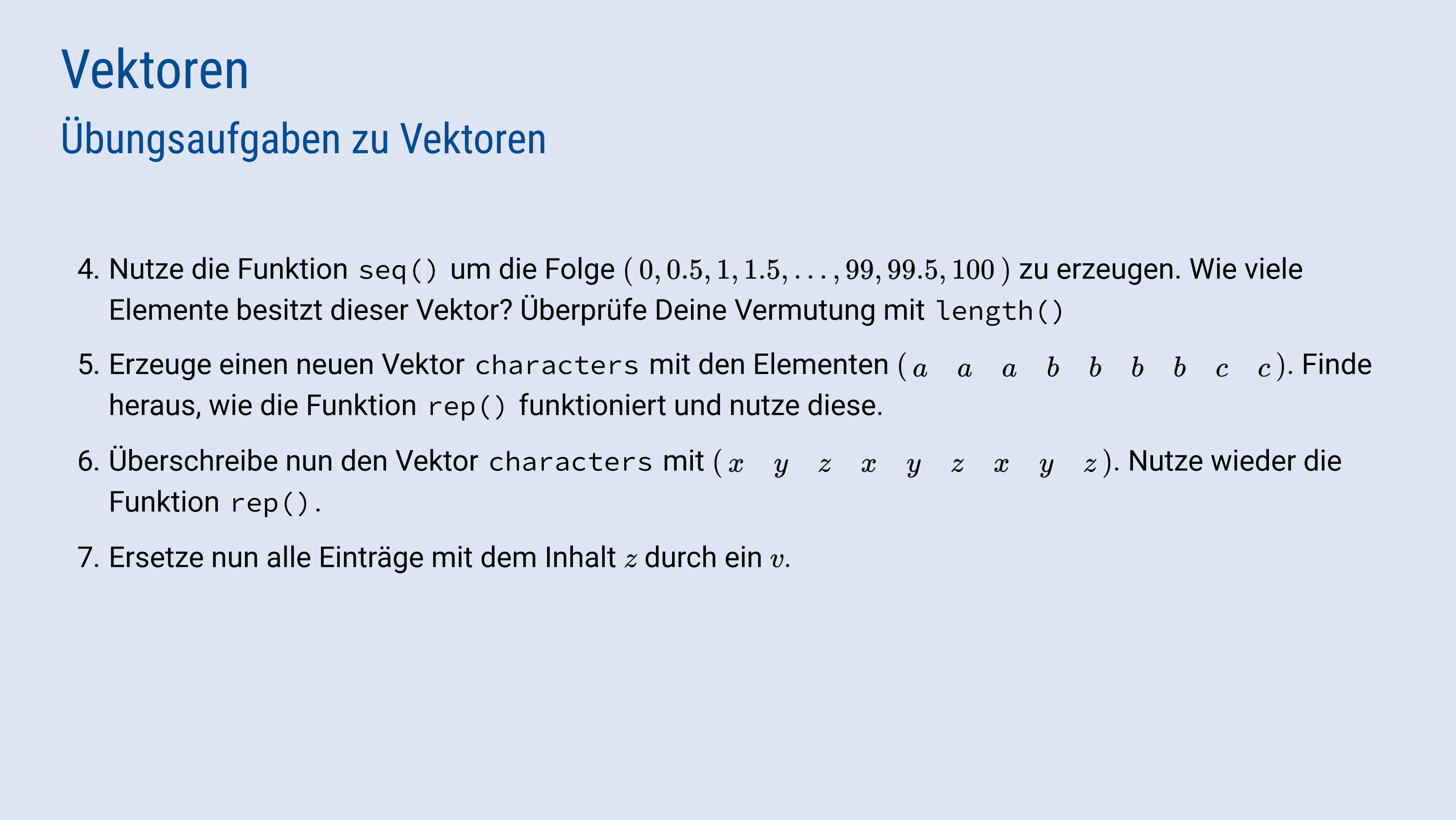
1

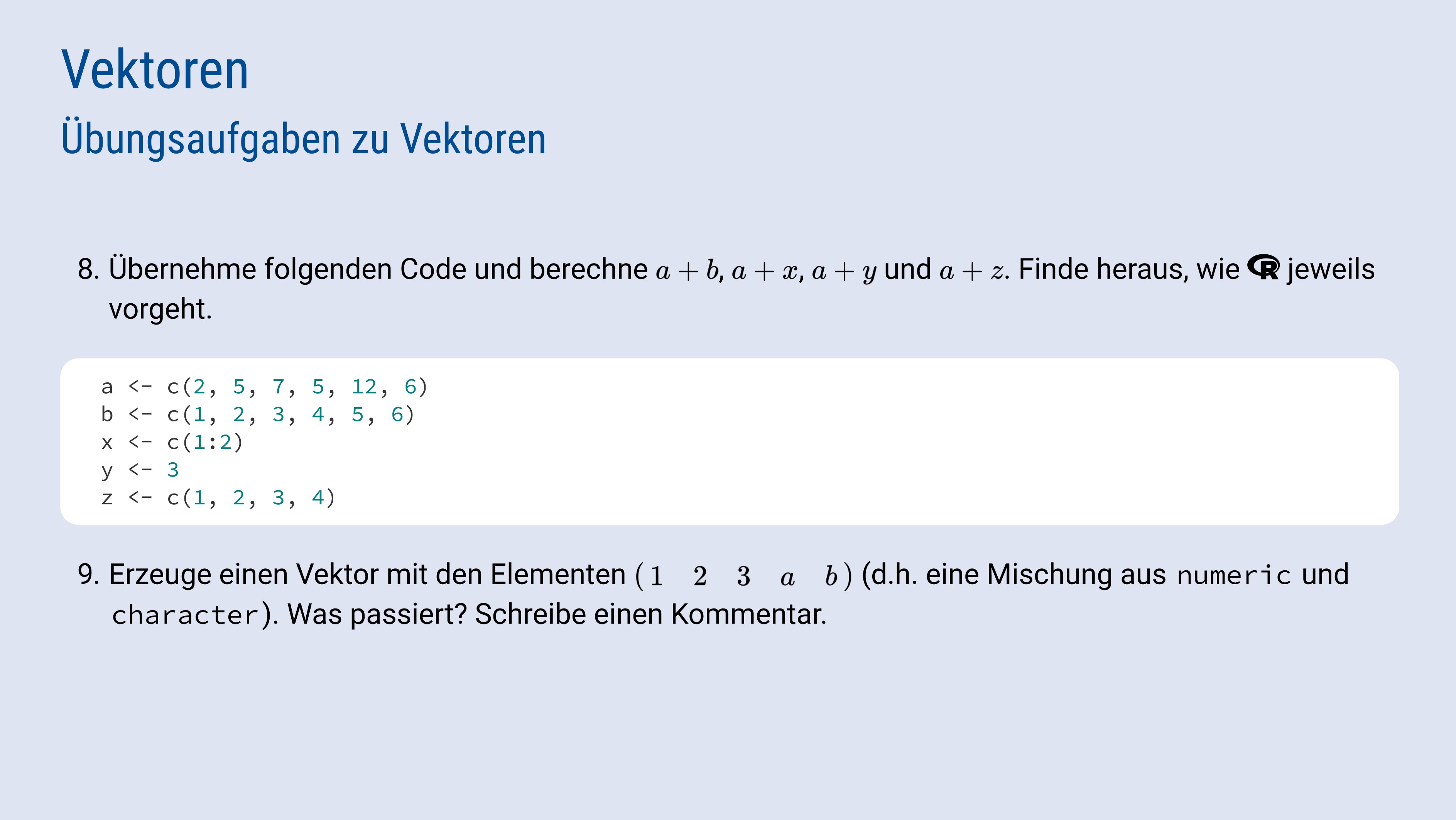
1

/

X

i





### Matrizen erzeugen

Wir können mehrere Vektoren des gleichen Datentyps und gleicher Länge zu einer Matrix zusammenfassen.

x <-

1

:

3

y <-

4

:

6

xy <- cbind(x, y)

xy

## x y

## [1,] 1 4

## [2,] 2 5

## [3,] 3 6

### Matrizen erzeugen

Außerdem können wir auch einen einzelnen Vektor in Matrixform bringen.

matrix(

1

:

6

, ncol =

2

)

## [,1] [,2]

## [1,] 1 4

## [2,] 2 5

## [3,] 3 6

Um ein einzelnes Element einer Matrix anzusprechen, müssen wir **zwei** Indizes verwenden: einen für die Zeile und einen für die Spalte des gewünschten Elements.

Das folgende Beispiel zeigt, wie man das Element in der 3. Zeile und der 2. Spalte erhält.

X <- matrix(

1

:

6

, ncol =

2

)

X[

3

,

2

]

## [1] 6

Um dieses Element zu ändern, nutzen wir wieder den Zuweisungsoperator <-.

X[

3

,

2

]

<-

13

X

## [,1] [,2]

## [1,] 1 4

## [2,] 2 5

## [3,] 3 13

Um ganze Spalten einer Matrix zu erhalten, wird der Index für die Zeile freigelassen.

X[,

1

]

#

g

i

b

t

d

i

e

1

.

S

p

a

l

t

e

z

u

r

ü

c

k

## [1] 1 2 3

Um ganze Zeilen zu erhalten, wird der Index für die Spalte freigelassen.

X[

2

:

3

, ]

#

g

i

b

t

d

i

e

2

.

u

n

d

3

.

Z

e

i

l

e

z

u

r

ü

c

k

## [,1] [,2]

## [1,] 2 5

## [2,] 3 13

### Matrixalgebra — Übersicht

A <- cbind(c(

13

,

4

,

8

)

, c

(

2

,

8.2

,

1

))

B <- cbind(c(

9

,

2.3

, -

1

)

, c

(

12

,

53

, -

3

))

A \* B

#

E

l

e

m

e

n

t

w

e

i

s

e

M

u

l

t

i

p

l

i

k

a

t

i

o

n

C <- t(A) %\*% B

#

M

a

t

r

i

x

m

u

l

t

i

p

l

i

k

a

t

i

o

n

t(A)

#

T

r

a

n

s

p

o

n

i

e

r

e

n

diag(C)

#

E

l

e

m

e

n

t

e

d

e

r

H

a

u

p

t

d

i

a

g

o

n

a

l

e

solve(C)

#

I

n

v

e

r

t

i

e

r

e

n

eigen(C)

#

E

i

g

e

n

w

e

r

t

e

u

n

d

E

i

g

e

n

v

e

k

t

o

r

e

n

### Matrixbasierte Funktionen

Wie bei Vektoren gibt es auch eine Reihe von Funktionen für Matrizen.

X <- matrix(

1

:

40

, ncol =

4

)

colSums(X)

#

S

p

a

l

t

e

n

s

u

m

m

e

n

rowSums(X)

#

Z

e

i

l

e

n

e

n

s

u

m

m

e

n

colMeans(X)

#

S

p

a

l

t

e

n

m

i

t

t

e

l

w

e

r

t

e

rowMeans(X)

#

Z

e

i

l

e

n

m

i

t

t

e

l

w

e

r

t

e

Matrizen

Übungsaufgaben zu Matrizen

1

.

Erzeuge mit dem Inputvektor

1:12

und

matrix()

folgende Matrix

X

.

2

.

Nimm die Matrix aus 1. und vertausche die Spalten. Das Ergebnis soll an die Variable

Y

übergeben

werden.

3

.

Berechne

.

X

=

⎛

⎜

⎜

⎜

⎜

⎜

⎜

⎜

⎜

⎝

1

2

3

4

5

6

7

8

9

10

11

12

⎞

⎟

⎟

⎟

⎟

⎟

⎟

⎟

⎟

⎠

X

Y

T

Matrizen

Übungsaufgaben zu Matrizen

4

.

Erzeuge eine

Matrix aus der 2. und 5. Zeile der Matrix

X

.

5

.

Erzeuge die Matrix

X

mit

X <- matrix(8:-7, nrow = 4)

.

Ersetze die Elemente auf der Hauptdiagonalen durch

NA

s.

Ersetze jetzt alle

NA

s in der Matrix durch

1

. Nutze dazu die Funktion

is.na()

.

2

×

2