Einführung in R

Dr. Gabriele Schwiertz & Dr. Denis Arnold

2024-10-15

Skript basiert teilweise auf Ch. 1 aus Winter, Bodo. Statistics for linguists: an introduction using R. Routledge, Taylor & Francis Group, 2020.

R & RStudio

R ist eine Programmiersprache, *RStudio* ist das Programm, in dem die meisten Leute mit R arbeiten. Normalerweise ist das RStudio-Interface in 4 Fenster aufgeteilt.

Vorbereitung: Projekt anlegen

Bevor wir anfangen, legen wir ein neues R-Projekt an: In einem Projekt sind alle relevanten Dateien und Informationen gespeichert. Dann öffnen wir ein neues R-Skript: In dem R-Skript werden wir unsere Befehle eingeben und ausprobieren.

Erste Schritte: R als Taschenrechner

In den grauhinterlegten Zellen stehen die Befehle zur Eingabe in RStudio, in den weißen Zellen darunter steht der Output, also das, was R als Ergebnis ausgibt.

Beispiel:

```
pi
```

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

Man kann sowohl in der Konsole als auch im Skript mit R rechnen.

```
# Addition:

2 + 2
```

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

```
# Subtraktion:

3 - 2
```

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

```
# Beispiel für einen unvollständigen Befehl; abbrechen mit ESC:

# 3 -

# Division:
```

```r
3 / 2
```

```
## [1] 1.5
```

Wir können den Code ausführen mit `Strg+Enter` oder mit dem Run-Button.

```r
# Multiplikation:

3 * 2
```

```
## [1] 6
```

```r
# Zwei hoch drei (2 * 2 * 2):

2 ^ 3
```

```
## [1] 8
```

```r
# Klammerung:

(2 + 3) * 3
```

```
## [1] 15
```

```r
2 + (3 * 3)
```

```
## [1] 11
```

Befehle in R, Bsp. Quadratwurzel

```r
# Wurzel 4:

sqrt(4)
```

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

```r
# Beispiel für einen fehlerhaften Befehl, es fehlt das Argument:

 #sqrt()

# Der Betrag:

abs(-2)
```

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

```r
abs(2)
```

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

Variablen definieren

```r
# Variablen definieren:

x <- 2
y <- 3

# Den Wert kann man ausgeben lassen, indem man die Variable aufruft

x
```

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

```
# Mit Variablen kann man dann weiterarbeiten

x*y
```

```
## [1] 6
```

```
# Wir benutzen x hier wie eine Zahl

x/2
```

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

```
# Man kann auch Ergenisse von Berechnungen in einer Variablen speichern

x<- 2+2
x
```

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

```
# In R ist  Groß- und Kleinschreibung relevant, das heißt die Eingabe eines großen "X" produziert einen

X
```

```
## Error in eval(expr, envir, enclos): Objekt 'X' nicht gefunden
```

```
# Überprüfen, welche Objekte gerade in der R-Umgebung vorhanden sind

ls()
```

```
## [1] "x" "y"
```

In RStudio erscheinen die Variablen im “Environment”-Panel.

Übung

Lege drei Variablen wie folgt an. a soll 3 sein, b soll 4 sein und c soll 5 sein. Berechne a hoch b, dann subtrahiere c. Schreibe die Wurzel des Ergebnisses in die Variable d. Lasse d ausgeben.

```
a <- 3
b <- 4
c <- 5

a^b-c
```

```
## [1] 76
```

```
sqrt(a^b-c)
```

```
## [1] 8.717798
```

```
d <- sqrt(a^b-c)

d
```

```
## [1] 8.717798
```

Numerische Vektoren

Variablen können nicht nur einzelne Werte enthalten, sondern auch z.B. eine Liste aus Zahlen, die mit dem Befehl "concatenate" c() aneinandergereiht werden.

```
x <- c(2.3, 1, 5)
x
```

```
## [1] 2.3 1.0 5.0
```

Wir können überprüfen, wie viele Elemente der Vektor enthält.

```
length(x)
```

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

Wir können überprüfen, was es für ein Typ der Vektor ist.

```
class(x)
```

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

```
mode(x)
```

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

Zahlenfolgen können einfach generiert werden, hier zum Beipsiel vom ersten Argument (10) bis zum letzten (1)

```
mynums <- 10:1
mynums
```

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

Auf diesen numerischen Vektor können wir weitere Funktionen anwenden.

```
# die Summe

sum (mynums)
```

```
## [1] 55
```

```
# den kleinsten Wert finden

min (mynums)
```

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

```
# den größten Wert finden

max (mynums)
```

```
## [1] 10
```

```
# beides gleichzeitig

range (mynums)
```

```
## [1]  1 10
```

```
# den Mittelwert

mean (mynums)
```

```
## [1] 5.5
```

Und genauso kann man auch Rechenoperationen auf den Vektor anwenden.

```
# von jedem Wert 5 subtrahieren

mynums - 5
```

```
##  [1]  5  4  3  2  1  0 -1 -2 -3 -4
```

```
# jeden Wert durch 2 teilen

mynums / 2
```

```
##  [1] 5.0 4.5 4.0 3.5 3.0 2.5 2.0 1.5 1.0 0.5
```

Übung

Lege einen numerischen Vektor *meineZahlen* mit den Zahlen von 20-30 an. Lass den *range* ausgeben und berechne den Mittelwert.

```
meineZahlen<-20:30
range(meineZahlen)
```

```
## [1] 20 30
```

```
mean(meineZahlen)
```

```
## [1] 25
```

Indexieren

Ich kann einzelne Werte in einem Vektor über die Position (Index) rauspicken.

```
# den Wert an erster Stelle  (Achtung, anders als andere Programmiersprachen beginnt R die Zählung bei

mynums [1]
```

```
## [1] 10
```

```
# den Wert an zweiter Stelle

mynums [2]
```

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

```
# die ersten vier Werte

mynums[1:4]
```

```
## [1] 10  9  8  7
```

```
# alle Werte, außer den zweiten

mynums[-2]
```

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

Übung

Addiere den 2. und den 3. Wert unseres Vektors mynums und schreibe den Wert in die Variable a.

```
a<-mynums[2]+mynums[3]
a
```

```
## [1] 17
```

Logische Vektoren

Logische Vektoren enthalten die Werte TRUE oder FALSE.

```
# ist der jeweilige Wert größer als 3?

mynums > 3
```

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

```
# ist der jeweilige Wert kleiner oder gleich 4?

mynums <= 4
```

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

```
# ist der Wert genau 4?

mynums == 4
```

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

```
# Achtung: Doppeltes Gleichzeichen! Ein einfaches Gleichzeichen würde mynums den Wert 4 geben und somit

# ist der Wert nicht gleich 4?

mynums != 4
```

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

Wir speichern den Output einer solchen Abfrage in den logischen Vektor mylog

```
mylog <- mynums >= 3

# und überprüfen, welcher Typ der Vektor ist

class(mylog)
```

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

Man kann diesen logischen Vektor jetzt auch zum Indexieren verwenden

```
mynums[mylog]
```

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

```
# oder etwas länger ausgedrückt

mynums[mynums >= 3]
```

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

Character vector, “Buchstabenvektor”, strings

Wir legen einen Buchstabenvektor an.

```
status <- c('Single', 'Married', 'Married', 'Single', 'Single')

# Überprüfen, was drin steht

status
```

```
## [1] "Single"  "Married" "Married" "Single"  "Single"
```

```
# Und welcher Typ der Vektor ist?

class(status)
```

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

Buchstabenwerte (strings) sind in R durch Anführungszeichen gekennzeichnet.

Auch bei Character-Vektoren kann man per Index oder mit logischen Aussagen Werte selektieren.

```
status[2]
```

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

```
status[status =='Single']
```

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

Mathematische Operationen funktionieren hier nicht.

```
mean (status)
```

```
## Warning in mean.default(status): Argument ist weder numerisch noch boolesch:
## gebe NA zurück
```

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

```
status - 1
```

```
## Error in status - 1: nicht-numerisches Argument für binären Operator
```

Faktoren

Faktoren kategorisieren Daten, haben nur begrenzt viele verschiedene Werte.

Wir konvertieren unseren Buchstabenvektor *status* in einen Faktor-Vektor.

```
status <- as.factor(status)

# Überprüfen, ob es geklappt hat

status
```

```
## [1] Single  Married Married Single  Single
## Levels: Married Single
```

```
# hier sollten die Anführunszeichen verschwunden sein

# wir können die Stufen (levels) anzeigen lassen

levels(status)
```

```
## [1] "Married" "Single"
```

```
# sobald die Stufen festgelegt sind, können keine neuen hinzugefügt werden
```

```
status[3] <- "Divorced"
```

```
## Warning in `[<-.factor`(`*tmp*`, 3, value = "Divorced"): ungültiges
## Faktorniveau, NA erzeugt
```

```
# das funktioniert nicht, man beachte das "NA" (fehlender Wert)

status
```

```
## [1] Single  Married <NA>    Single  Single
## Levels: Married Single
```

Weitere Stufen hinzufügen, funktioniert so:

```
levels(status) <- c('Married', 'Single', 'Divorced')

# jetzt funktioniert der Befehl

status[3] <- "Divorced"
```

NULL und NA

Es gibt in R zwei Arten einen fehlenden Wert auszudrücken: NULL und NA. NULL steht für "no value" und NA für "Not Assigned".

```
NULL
```

```
## NULL
```

```
NA
```

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

NA ist ansteckend: Wenn wir versuchen, damit zu rechnen, ist das Ergebnis auch NA, weil R keine Möglichkeit hat, es tatsächlich auszurechnen:

```
NA + 1
```

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

```
max(NA, 12, 1)
```

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

Sollte man NA's in seinen Werten haben, kann man mit einem Parameter `na.rm=TRUE` trotzdem ein Ergenis ausrechnen lassen – unter Ausschluss der NAs.

```
max(NA, 12, 1, na.rm = TRUE)
```

```
## [1] 12
```

Fehlermeldungen

Wie geht man damit um, einfach googlen! RSTudio auf Englisch umstellen.

Dataframes

Tabellen heißen "dataframes" in R. Wir bauen eine simple Tabelle. Wir beginnen mit einem Vektor, der 3 Namen enthält.

```
participant <- c('louis', 'paula', 'vincenzo')
```

Wir packen das in einen *dataframe* und fügen eine zweite Spalte ("score") mit numerischen Werten hinzu.

```
mydf <- data.frame(participant, score = c(67, 85, 32))


# das ginge auch in 2 Schritten, also score erstmal separat definieren und zwar so:

score <- c(67,85,32)

mydf <- data.frame(participant, score)

# Kurz überprüfen

mydf
```

```
##   participant score
## 1       louis    67
## 2       paula    85
## 3    vincenzo    32
```

```
# wie viele Zeilen?

nrow(mydf)
```

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

```
# wie viele Spalten?

ncol(mydf)
```

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

```
# Wie heißen die Spalten?

colnames(mydf)
```

```
## [1] "participant" "score"
```

```
# kurzer Überblick

summary(mydf)
```

```
##  participant           score
##  Length:3           Min.   :32.00
##  Class :character   1st Qu.:49.50
##  Mode  :character   Median :67.00
##                     Mean   :61.33
##                     3rd Qu.:76.00
##                     Max.   :85.00
```

```
# die Struktur
```

```r
str(mydf)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ participant: chr  "louis" "paula" "vincenzo"
##  $ score      : num  67 85 32
```

Indexierung funktioniert auch bei Dataframes

```r
# Spalten können  mit "$" ausgewählt werden

mydf$score
```

```
## [1] 67 85 32
```

```r
# Ich kann Funktionen auf Spalten anwenden

sum (mydf$score)
```

```
## [1] 184
```

```r
# Wir können die erste Zeile auswählen

mydf[1,]
```

```
##   participant score
## 1       louis    67
```

```r
# Oder die zweite Spalte

mydf[,2]
```

```
## [1] 67 85 32
```

```r
# oder die erste Zelle in der zweiten Spalte

mydf[1,2]
```

```
## [1] 67
```

```r
# oder alle Zeilen, wo in der participant-Spalte "vincenzo" steht

mydf[mydf$participant == "vincenzo",]
```

```
##   participant score
## 3    vincenzo    32
```

Schematisch geht die Indexierung so: dataset [rows, columns]

Übung

Rechne für die Spalte `score` den Mittelwert aus: 2 verschiedene Möglichkeiten.

Lösung:

```r
mean(mydf$score)
```

```
## [1] 61.33333
```

```r
mean(mydf[,2])
```

```
## [1] 61.33333
```

Daten einlesen und speichern

Normalerweise generiert man nicht selbst Tabellen in R, sondern liest vorhandene Daten ein.

```
# Wir überprüfen das aktuelle Arbeitsverzeichnis.

getwd()
```

```
## [1] "C:/Users/schwiertz/Documents/R/DmD_IntroToR"
```

```
# Und was drin ist

list.files()
```

```
##  [1] "DmD.css"               "DmD_IntroToDmD.html"   "DmD_IntroToDmD.qmd"
##  [4] "DmD_IntroToDmD_files"  "DmD_IntroToR.aux"      "DmD_IntroToR.html"
##  [7] "DmD_IntroToR.pdf"      "DmD_IntroToR.Rmd"      "DmD_IntroToR.Rproj"
## [10] "DmD_style.css"         "DmD_style.scss"        "exerciseDataframe.csv"
## [13] "images"                "lösch.r"               "penguins.csv"
## [16] "Pinguine.csv"          "Pinguine.txt"
```

Das kann man auch in RStudio im Fenster "Files" überprüfen.

Wir lesen eine Datei ein

```
df <- read.csv("penguins.csv")
```

Eingelesene Dateien *immer* überprüfen

```
# die ersten 6 Reihen

head(df)

# die letzen 6 Reihen

tail (df)

# eine Zusammenfassung

summary (df)

# komplett angucken

View (df)

# wir gucken uns mal die Spaltennamen an
colnames(df)
```

Der Datensatz enthält Informationen über Pinguine.

#Übung Rechne den Mittelwert für die Schnabellänge aus

```
# erster Versuch
mean(df$bill_length_mm)
```

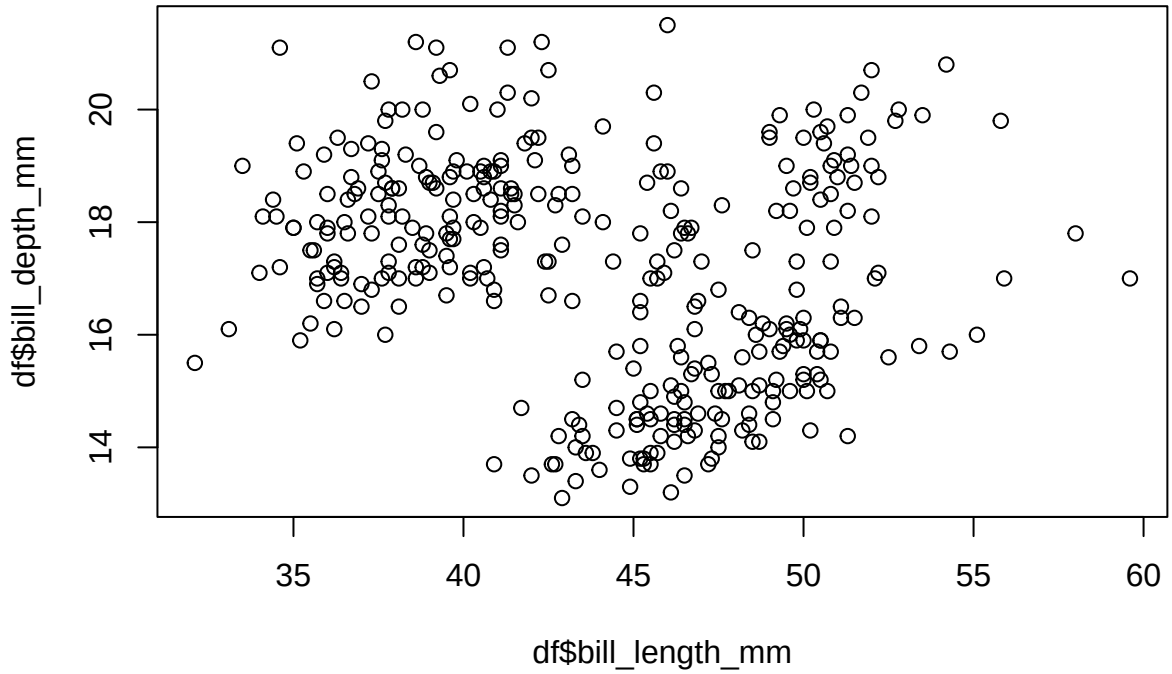
```
## [1] NA
```

```
# es scheint NA's zu geben. Problem beheben durch:

mean(df$bill_length_mm, na.rm=TRUE)
```

```
## [1] 43.92193
```

Mit R kann man auch sehr schön plotten, dazu mehr in der Sitzung zu Datenvisualisierung.

```
# Wir plotten mal was

plot(df$bill_length_mm, df$bill_depth_mm)
```



Wir brauchen nur den Datensatz für die Adelie-Pinguine und wollen den dann speichern.

```
df_adelie<-df[df$species=="Adelie",]
View (df_adelie)

# speichern
write.table(df_adelie, file = "Pinguine.csv")
```

Dateien mit anderen Separatoren können auch eingelesen werden. Wir speichern unsere Tabelle als tabulatorseparierte Textdatei.

```
write.table(df_adelie, file = "Pinguine.txt", sep = "\t")
```

Wir lesen sie wieder ein.

```
mydf <- read.table('Pinguine.txt',
                   sep = '\t', header = TRUE)
```

Pakete

Pakete enthalten zusätzliche Funktionen, die auf bestimmte Nutzungsfälle abgestimmt sind.

Sie müssen erst installiert werden.

```
# install.packages("fortunes")
```

Das geht aber auch über das Menu “Tools>Install Packages...”.

Und dann, wenn man sie benutzen möchte, müssen sie auch geladen werden.

```r
library("fortunes")

# das fortune-Paket wirft alberne Zitate aus, Fortune Cookies for R.

fortune()
```

```
##
## The problem with attach is what it does, not how it does it. If you do the same
## thing attach does any other way, you've managed to create all the problems of
## attach without using attach. If I tell you chopping your finger off with a
## knife is bad, your question shouldn't be "Okay, knives are bad. How do I chop
## my finger off without a knife? Can I use scissors or is there a better way?"
##    -- Gregor (about alternatives to attach())
##       stackoverflow.com (May 2016)
```

Zitieren

Benutzte Pakete sollte man zitieren

```r
citation("fortunes")$textVersion
```

```
## NULL
```

Die Versionsnummer kann man so herausfinden

```r
packageVersion("fortunes")
```

```
## [1] '1.5.4'
```

Und R zitiert man so

```r
citation()$textVersion
```

```
## NULL
```

```r
R.Version()$version.string
```

```
## [1] "R version 4.3.3 (2024-02-29 ucrt)"
```

Hilfe finden

Die Hilfsdateien zu einem Befehl findet man mit "?".

```r
?seq
```

```
## starte den http Server für die Hilfe fertig
```

Darüberhinaus findet man im Internet leicht Hilfe. Einfach Fehlermeldung googlen. Bzw. https://rseek.org/

Wie man eine R-Hilfeseite liest

Übung: Plotten mit Base-R

1. Lese die Datei "exerciseDataframe.csv" in die Variable `meindf` ein mithilfe des Befehls `read.csv()`.
2. Plotte die Daten mit dem Befehl `plot()`. Du kannst mit `col ="FARBE"` die Farbe ändern, probiere einige englische Farbnamen aus. Bzw. mit dem Befehl `colors()` kriegst Du eine Liste mit Farbnamen, die R versteht.

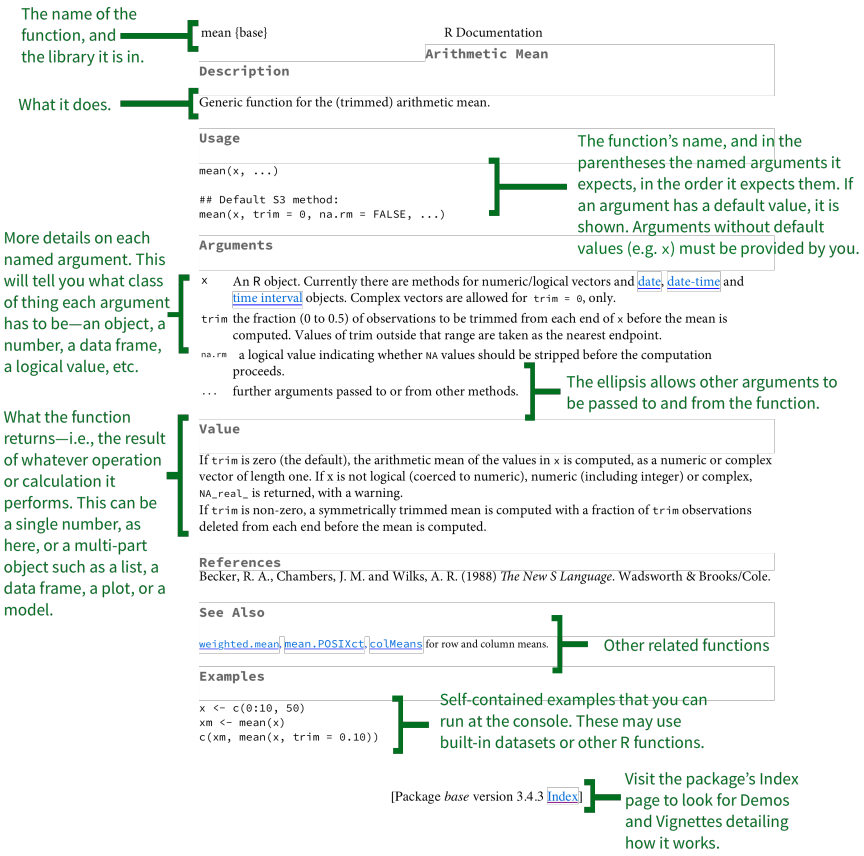


Figure 1: Quelle: Looking at data

3. Man kann auch den Hexadezimalwert für die Farbe eingeben mit `col = "#000000"`. Die ersten beiden Positionen geben den Wert für Rot an, die nächsten beiden den für Grün und der letzte für blau. Probiert das aus!

Zusammenfassen

Die heutige Sitzung sollte einen ersten Eindruck von der Funktionsweise von R vermitteln.

- Wir haben Variablen kennengelernt und Vektoren und dataframes.
- Wir haben verschiedene Funktionen kennengelernt.
- Wir haben gesehen, dass Tippfehler NIE verziehen werden.
- Wir haben Pakete kennengelernt.
- Wir haben gesehen, wie man Datensätze einliest.
- Wir haben gesehen, wie man Tabellen speichert.

Ausblick: Nur zum Ansehen

Eine Gallerie mit den schönsten R-Plots:

https://www.r-graph-gallery.com/index.html

https://www.r-graph-gallery.com/all-graphs.html

Mehr dazu in der Sitzung zu Visualisieren mit ggplot.