Datenanalysen mit R

Peter Nauroth

1. Was ist R?

Zentrale Strukturen

R ist eine Programmiersprache

- Datenstrukturen & Datentyp
  - Datentyp (`mode()`): numeric, character, boolean, factor
  - Datenstruktur (`str()`)
  - *Viele Fehlermeldungen sind darauf zurückzuführen, dass der Datentyp oder die Datenstruktur nicht zur Funktion passen!*
- Operatoren (z.B. Zuordnung: `x <- 5`)
- Funktionen (z.B. `'+'(4,5)`)
  - Funktionsparameter

2. RStudio

GUI

1. **Konsole**
   - Taschenrechner (`4+4`)
2. **Skript**
3. **Workspace / History / GIT**
   - *Viele Probleme sind auf unbeabsichtigte Objekte im Workspace zurükzuführen.*
     - `rm(list=ls())` –> Säubern des Workspace
     - erfordert mehr Mikromanagement als SPSS

4. Files / Plots / Packages / Help

- Working Directory beachten (`setwd()`)
- Hilfe (`?`; z.B. `?lm`)
  - Packages / Libraries
  - Für (fast) alles gibt es bereits vorgefertigte Funktionen in Paketen
  - Um diese nutzen zu können: `install.packages()` & `library()`

3. Operatoren, Datentypen & Datenstrukturen

3.1 Datentypen

- numeric: ganzzahlige (integer) oder Gleitkomma-Werte (double)
- character: Zeichen [String: Zeichenkette]
- logical: Wahrheitswerte (`TRUE` und `FALSE`)
  - `(5 == 5) == FALSE`
- factor: nominale oder ordinale Daten (intern: integer mit labels)
- (list: rekursive Datenstruktur)

3.2 Operatoren

1. Mathematische Operatoren: `^ %% %/% * / + -`
2. Logische Operatoren: `< > <= >= == != ! & && | ||`
3. Zuordnung: `<-` oder `=`

- Präzedenz: von 1. zu 3., von links nach rechts

  TODO: Was ergibt der folgende Ausdruck:

  ```
  (1+2*3 <= 6) | (5==5) == !FALSE
  ```

3.3 Datenstrukturen

Das Verständnis über den Zugriff auf und die Organisation von Datenstrukturen ist essentiell für das Arbeiten mit R.

3.3.1 Homogene Datenstrukturen

Homogene Datenstrukturen enthalten nur einen Datentyp.

1. Skalare

```
s <- 5
s
str(s)
s + 4
s * 2
```

3.3.1 Homogene Datenstrukturen

2. Vektoren

```
v <- c(5,4,5)
v
str(v)
v + 4
v * 2
# get 3rd element:
v[3]
```

3. Matrizen

```
m1 <- matrix(c(1,123,4,12,3,5), nrow=2)
m1
str(m1)
m1 + 4
m1 * 2
m2 <- matrix(c("1","123","4","12","3","5"), nrow=2)
m2
str(m2)
# get 2nd element in 1st row:
m2[1,2]
# get 2nd column:
m2[,2]
```

3.3.2 Heterogene Datenstrukturen

Datenstruktur mit potentiell unterschiedlichen Datentypen.

1. Data frames

```
subject <- c(2,3,4)
condition <- as.factor(c("EG","KG","EG"))
dv <-  c(5,5,5)
df <- data.frame(subject, condition, dv)
df
str(df)

# get values of dv:
df$dv
# alternatively
df[,3]
# get values of subject 3:
df[df$subject==3,]
# alternatively
df[2,]
```

3.3.2 Heterogene Datenstrukturen

1. Data frames

```
# get condition & dv
df[,c(2,3)]
```

- Data frames sind die Struktur in der unsere Daten normalerweiser organisiert sind.

  TODO: Gebe die Bedingungszugehörigkeit von Subjekt 4 auf der Konsole aus.

3.3.2 Heterogene Datenstrukturen

1. **Data frames**

   TODO: Gebe die Bedingungszugehörigkeit von Subjekt 4 auf der Konsole aus.

```
df[3,2]
df[df$subject==4, names(df)=="condition"]
```

3.3.2 Heterogene Datenstrukturen

2. Listen

```
l <- list(c("a", "b", "c"), c(1,2,3,4),
          c(TRUE, FALSE, TRUE))
l
str(l)

# get 1st value of 3rd object in list
l[[3]][1]
```

- Viele Outputs von statistischen Funktionen sind Listen.

4. Funktionen

Grundsätzliches

Funktionen sind Methoden mit denen wir unsere Daten manipulieren und analyiseren.
Ein Funktion hat (einen):

- *Namen* (z. B.: `mean`),
- *Parameter* bzw. *Argumente* (z. B.: `c(1,2,3)`)
  - notwendige und
  - optionale
- *Rückgabewert* (z. B.: `## [1] 2`).

Grundsätzliches

```
x <- c(1:10)
x
# Calculate mean of x
mean(x) # That's a function call and x is a function
        # parameter/argument
# However, mean() may have more parameters. Try:
?mean
```

TODO: Berechne den Mittelwert von `x <- c(1:78, NA)`.

Grundsätzliches

```r
mean(x, na.rm=TRUE)
```

- NAs stellen ein häufiges Problem für Funktionen in R dar:
- Prinzipiell 2 Möglichkeiten:
- Funktionsparameter vorhanden (z.B. `na.rm=TRUE`)
- vorher ausschließen (z.B. `dplyr::filter(data, !is.na(x))`; dazu später mehr)

Selbstdefinierte Funktionen

- Funktionen können selbst definiert werden:

```
# We want a function that squares its input:
y <- 5
y*y
f1 <- function(x) x*x
f1(y)
```

TODO1: Definiere eine Funktion, die ihren Input verdoppelt und anschließend 5 addiert.

TODO2: Definiere eine Funktion, die den Mittelwert eines Vektors berechnet und vorher mögliche NAs entfernt (Tipp: `?mean`).

Selbstdefinierte Funktionen

```
doublePlusFive <- function(x) 2*x+5
myMean <- function(x) mean(x, na.rm=TRUE)
```

Funktionen

Für fast alles gibt es bereits Funktionen.

- Bevor man anfängt eine eigene Funktion zu schreiben lohnt sich eine Suche im Internet:
  - StackOverflow
  - Quick-R
  - R-bloggers

5. Daten

Einlesen von .csv Dateien

```r
data <- read.csv(file="data.csv")
# read.table offers most important wrappers for importing
# .txt or .csv files
?read.table
```

TODO: Welche Wrapper sind am günstigsten um mit Excel erstellte .csv-Dateien einzulesen? (Tipp: Erstelle eine .csv mit Excel und öffne sie mit einem Texteditor)

Einlesen von .csv Dateien

- R nutzt “/” oder “\\” um Ordner anzusteuern (nicht “\” wie in Windows üblich):

```
d <- read.delim2(
      "H:/Arbeit/Studien/3-SR-S0/data/data_SR.csv")
# Or:
d <- read.delim2(
      "H:\\Arbeit\\Studien\\3-SR-S0\\data\\data_SR.csv")
```

Einlesen von .sav Dateien (SPSS)

- Um SPSS-Files einlesen zu können, brauchen wir ein spezielles Paket

```
install.packages("foreign") # install foreign package
library(foreign)            # load the foreign package
?read.spss                  # check out the documentation
myData <- read.spss("myfile.sav", to.data.frame=TRUE)
```

TODO: Einlesen einer eigenen SPSS Datei in R.

Daten verstehen

```r
#install.packages("ggplot2")
#install.packages("psych")
library(ggplot2)
library(psych)

str(diamonds)
head(diamonds)
View(diamonds)
summary(diamonds)

by(diamonds, diamonds[,"cut"], function(x) lm(carat~price,

pairs.panels(diamonds[c(1:1000),])
```

Manipulation von Daten

- Häufig müssen wir vorliegende Daten verändern ("manipulieren") um sie auswerten zu können.
- Beispiel:
  - Wir wollen die Korrelationen von `carat`, `depth` und `price`
  - Hilfreiche Funktion: `cor()`

TODO: Gebe die Korrelationsmatrix der drei Variablen auf der Konsole aus.

Manipulation von Daten

```
cor(diamonds[,c(1,5,7)])
```

- Ein Problem dabei ist, dass wir immer die Spaltenposition der Variablen wissen müssen.
- Ein Paket das Datenmanipulationen enorm erleichtert ist `dplyr`.

dplyr

Eine kurze und gute Einführung in dplyr findet man hier: Advanced R Programming

```
install.packages("dplyr")
library(dplyr)                  # Did you get any messages?
```

- Zentrale Funktionen:
  - tbl_df
  - filter
  - select
  - arrange

dplyr::tbl_df

```
diamonds
myData <- tbl_df(diamonds)
myData
```

dplyr::filter

Häufig wollen wir nur Untermengen unserer Daten betrachten.
Beispielsweise wollen wir uns nur Daten einer Bedingung anschauen.

```
# get all diamonds with premium cut
myData[myData$cut=="Premium",]
filter(myData, cut %in% c("Premium"))
# not really more efficient
# but what if we want:
#   all I- or J-colored
#   with 'Fair' or 'Good' cut quality
#   and a weight of more than 4 carat?
```

dplyr::filter

```
myData[myData$cut %in% c("Fair","Good") &
       myData$color %in% c("I", "J") &
       myData$carat > 4,]

filter(diamonds, carat > 4 &
         cut %in% c("Fair","Good") &
         color %in% c("I", "J"))
```

dplyr::select

Bestimmte Spalten auszuwählen und damit weiterzuarbeiten ist einer der häufigsten Arbeitsschritte in R.

```
select(myData, carat, cut, color, price)
select(myData, carat:color, price)
```

Mithilfe von `dplyr::select` spart man sich den Zugriff über die Spaltennummern.

dplyr::arrange

Manchmal möchte man einen Datensatz nach bestimmten Kriterien ordnen.

```
# order the data ascending for carat and with
# diamonds of same carat beeing sortet by descending
# prices
arrange(myData, carat, desc(price))
```

dplyr

- Sehr nützliches Paket für einfache Datenmanipulationen
- Andere nützliche Funktionen in `dplyr`
  - `mutate`
  - `summarize`
  - `group by`

6. Gundlegende statistische Verfahren

t-Tests

```r
# independent 2-group t-test
t.test(y~x) # where y is numeric and x is a binary(!) facto

# independent 2-group t-test
t.test(y1, y2) # where y1 and y2 are numeric

# paired t-test
t.test(y1, y2, paired=TRUE) # where y1 & y2 are numeric
```

TODO: Unterscheiden sich die "gute"" Diamanten preislich von "sehr guten" Diamanten?

t-Tests

```r
t.test(diamonds$price[diamonds$cut%in%c("Good", "Very Good"
       factor(diamonds$cut[diamonds$cut%in%c("Good", "Very

tTestData <- diamonds %>%                    # %>% = then (
  filter(cut%in%c("Good", "Very Good"))

t.test(tTestData$price~tTestData$cut)
```

Lineare Modelle

```
testData <- data.frame(y = rnorm(100),
                       x1 = rnorm(100),
                       x2 = rnorm(100),
                       x3 = rnorm(100))

lm(y ~ x1 + x2 + x3, data=testData)

myRegression <- lm(y ~ x1 + x2 + x3, data=testData)
summary(myRegression)

myInteraction <- lm(y ~ x1 * x2 * x3, data=testData)
summary(myInteraction)
```

TODO: Sagen carat, clarity & color den price vorher?

Vorsicht: Datentypen beachten und evlt. transformieren (?as.numeric())

Lineare Modelle

```
myRegression <- lm(price~carat + as.numeric(clarity) + as.n
summary(myRegression)
```

7. Gundlegende Programmierkenntnisse

Selbstverständlicher Code

- Kommentieren!
- Einheitliche und konsistente Namensgebung z.B.: `squareAndDouble` oder `square.and.double`
- selbsterklärende Variablennamen und Funktionsnamen
  - besser lang und verständlich als kurz und kryptisch!

```
# ------------- Intro to R ------------- #

# ----------------- Supplementary Functions

# This is a wrapper for the mean(..., na.rm=TRUE) function
meanNA <- function(x) mean(x, na.rm = TRUE)
```

Nachvollziehbare Programmierung

- Modularisierung
  - Codeblöcke NIE copy & pasten
  - besser eine Funktion extrahieren
  - und in eigene Dateien schreiben:
    `source("supple-functions.R")`

8. Organisatorisches

Organisatorisches zum 2. Termin

- Eigener Laptop (Raum: S2)
- Präsentation:
  - Kommentierter und selbstverständlicher R Code
  - einseitiges Handout mit wichtigsten Infos

Potentielle Themen

- Datenmanipulation 2: `reshape2`
- **Grundlegende Statistik:**
  - Häufigkeiten
  - Kreuztabellen
  - Korrelationen: `stats`, `Hmisc`
  - Nicht-parametrische Verfahren
- **Varianzanalyse**: `car`, `ez`, `multcomp`...
- **Datenvisualisierung:** `ggplot2`
- Skalenanalyse und explorative Faktorenanalyse: z.B. `psych`
- Reporting (RMarkdown) & Version Control Management (GIT)

Potentielle Themen 2

- Effektstärkenberechnung / Poweranalyse
- Meta-analytische Verfahren: z.B. `metafor`
- Latente Modelle / Konfirmatorische Faktorenanalyse: `lavaan`
- Datenvisualisierung 2: `lattice`
- Simulationen mit R