Fachhochschule Vorarlberg - University of Applied Science

Alles Wichtige zu R

1 Starten und Beenden von R

1.1 Starten von R

Da RStudio installiert wurde, geschieht der Start von R über den Start von RStudio
Mit RStudio erhalten Sie eine Oberfläche bestehend aus 4 Fenstern mit folgender Bedeutung (beginnend links oben, im Uhrzeigersinn):

- Skript-Fenster
- Konsole
- Anzeige von z.B. Plots, Hilfetexte, Packages,..
- Umgebung, z.B. verwendete Variable,...

1.2 Beenden von R

Sie können R schließen entweder durch Beenden von RStudio oder mit folgendem Befehl in der Console

```
> q()
```

2 Hilfe zu R

2.1 Hilfe zu Funktionen

```
> ?sort
> ?median
```

oder einfach In RStudio den Suchbegriff ins Suchfeld vom Register Help eintragen
Die Hilfeseiten sind immer gleich aufgebaut:

1. Description
   Überblick darüber, was die Funktion leistet
2. Usage, Arguments
   Wie die Funktion aufgerufen wird, welche Argumente übergeben werden müssen. Die Argumente einer Funktion kann man auch kurz aus

   ```
   > args(median)
   ```

   bestimmen
3. Details
4. Value
   Wie der Rückgabewert der Funktion zu interpretieren ist
5. References
   Hinweis auf relevante Literatur
6. Examples
   Hier sieht man anhand von Beispielen, wie die Funktion verwendet wird
   Das Beispiel kann man sich aber auch direkt im Editor anzeigen lassen

   ```
   > example(median)
   ```

2.2 Hilfe zu unbekannter Funktion

Beispielsweise, wenn Sie Funktionen suchen, die zum Thema Varianz angeboten werden

```
> ??variance
```

oder mit R-Studio im Register Help einen Suchbegriff ins Suchfeld eingeben

2.3 Hilfe zu Packages

Weitere Informationsquellen zu Packages sind Vignetten (nicht obligatorisch), das sind Dokumente zu den packages
Eine Übersicht aller installierter packages erhalten Sie im Register packages von RStudio Mit vignette("packagename") rufen Sie eines dieser beschreibenden Dokumente auf

```
> vignette("grid")
```

3 Eingabemodi von R

R kann in zwei Modi betrieben werden

1. interaktiver Modus
2. Skriptmodus

3.1 interaktiver Modus

Im interaktiven Modus geben Sie R-Anweisungen in der Konsole an, bestätigen dies mit Enter und erhalten sofort das Ergebnis, z.B.

```
> x<-c(1,2,4,6,7)    # es wird ein Vektor mit den Zahlen 1,2,4,6,7 generiert
> mean(x)            # es wird der Mittelwert aus den Zahlen in x gebildet
[1]  4
> y<-mean(x)         # der Mittelwert von x wird in die Variable y geschrieben
> 2*y                # der Mittelwert wird verdoppelt
[1]  8
```

R zeigt typischerweise Ihre Frage und die Antwort in unterschiedlichen Farben an. Fehlermeldungen kommen in rot
Die getätigten Eingaben können Sie sich mit dem Befehl

```
> history()
```

nochmals anzeigen lassen, bzw. finden Sie im history-Fenster von RStudio.

3.2 Skriptmodus

Der Skriptmodus bedeutet, dass Sie die einzelnen R-Befehle in einer R-Datei ablegen.
Die Folge von Anweisungen, kann man in einer normalen Textdatei speichern. Zur Kennzeichnung kann man diese Dateien .r oder .rcode nennnen.
In der praktischen Arbeit mit R wird es meist so sein, dass man die Anweisungen zunächst im interaktiven Modus testet und dann, sobald sie funktionieren, in ein R-Skript übernimmt.

Sie erstellen eine neue Skriptdatei via
File → New → R Script.
Sie sollten jetzt vier Panele wie in Abb. 1 sehen. Speichern Sie die Datei als tutorial.R ab über
File → Save.
Von nun an sollten Sie alle R-Befehle in die Skriptdatei schreiben. Kommentieren Sie immer Ihren Code mittels dem Symbol

```
#
```

, damit Sie oder jemand anderes zu einem späteren Zeitpunkt nachvollziehen kann, was der Code macht.
Geben Sie im Editor-Panel tutorial.R als erste Zeile z <- c(8,13,21) und als zweite Zeile 2*z ein.

Sie haben verschiedene Möglichkeiten R-Code an den R-Prozess zu senden:

1. Klicken Sie auf **source**. Der gesamte Code ihres Skriptes wird an die R-Konsole geschickt.
2. Bewegen Sie den Cursor zur ersten Zeile. Nun klicken Sie auf **Run** . Nur die entsprechende Zeile wird an die R-Konsole geschickt und der Cursor springt zur n?achsten Zeile. Wiederholen Sie den Klick auf **Run** um die zweite Zeile an die R-Konsole zu schicken, etc.
3. Sie k?onnen auch bestimmte Teile des Codes mit der Maus ausw?ahlen und mit **Run** an die R-Konsole senden.
4. Die Tastaturkombination für **Run** ist <Ctrl>+<RETURN> (d.h. <Ctrl> und <RETURN> gleichzeitig drücken).

3.2.1 Bemerkung

: Es kann vorkommen, dass die Ausführung eines R-Skriptes zu lange dauert. Dies passiert oft, wenn man einen Fehler in einer Schleife hat. Sie können die Evaluation jederzeit abbrechen, indem Sie auf **Stop** klicken (dieser Button wird nur sichtbar, wenn R am Rechnen ist). Alternativ kann man die <Esc>-Taste drücken, wenn man in der R-Konsole ist.

4 Packages verwenden

Für R gibt es zahlreiche Erweiterungspakete (packages). Sollten Sie ein spezielles Problem mit R lösen wollen, so informieren Sie sich stets zunächst darüber, ob es dazu nicht schon ein passendes package gibt. Dieses kann man dann herunterladen, installieren und verwenden!

4.1 Installierte Packages anzeigen

Mit RStudio erhalten Sie die Liste der installierten Packages im Register Packages. Dort wird Ihnen mit einem kleinen Haken auch angezeigt, ob das Package aktuell geladen ist

4.2 Installierte Packages verwenden

Da Packages nach dem Installieren nicht einfach verfügbar sind, müssen diese vor Gebrauch erst noch geladen werden. Dies geschieht mit RStudio einfach durch das Setzen des Häkchens in der angezeigten Package Liste.

4.3 Packages aus dem Internet herunterladen und installieren

In RStudio können Sie die Installation mit Hilfe der Option **Install Packages** einleiten. Im sich öffnenden Dialogfenster kann man sich entscheiden, ob man das Package von CRAN herunterladen möchte oder ob man ein Package installieren möchte, das bereits aufs Laufwerk heruntergeladen ist.

5 Arbeitsverzeichnis aufbauen

Hier geht es um die effektive Organisation der Dateien für das Arbeiten mit R. Bearbeitet man ein Projekt fallen zahlreiche Dokumente an. Dabei ist es wichtig von Anfang an eine gute Verzeichnisstruktur vorzusehen, zB

_history
_temp
100 Admin
200 Literatur
300 Daten
400 R-Skripte
500 Schreibwerkstatt
600 Final

Standardmäßig versucht R, eine Datei, die Sie in ein R-Skript einbinden, im gleichen Verzeichnis zu finden, in dem das R-Skript selbst liegt. Liegen die Daten aber im Verzeichnis 300 Daten, so macht es Sinn dieses Datenverzeichnis als Arbeitsverzeichnis zu deklarieren:

```
> setwd("Documents/R-Projekt/300Daten")
```

Typischerweise deklariert man sein Arbeitsverzeichnis zu Beginn eines R-Skriptes.

6 Arbeitsstände sichern und wiederherstellen

Hier wird erläutert, wie man Arbeitsstände, d.h. R-Objekte, wie Datensätze oder Variablen dauerhaft speichern und wiederherstellen kann, um später wieder damit arbeiten zu können.

```
> save()       # speichert die angegebenen R-Objekte in einer Datei

> save.image() # speichert alle existierenden R-Objekte in einer Datei

> load()       # l\"adt die zuvor gespeicherten R-Objekte nach R
```

7 Mit Daten arbeiten

7.1 Variablen

Variablen werden in R nicht deklariert. Variablennamen in R dürfen nicht mit einer Ziffer beginnen. R ist case sensitive. Namen in R beinhalten oft einen Punkt, der als Trennzeichen verwendet wird

```
> x <- 1     # weist der Variablen x den Wert 1 zu
> class(x)   # zeigt den Typ einer Variablen an
> levels()   # zeigt die m\"oglichen Auspr\"agungen einer Faktorvariablen
> ls()       # listet alle aktuell existierenden Variablen auf
> rm(x)      # l\"oscht eine oder mehrere Variablen
```

7.2 Faktoren

Dies sind Variablen, die nur eine bestimmte endliche Menge an Ausprägungen annehmen können.

```
> geschlecht <- factor(c("m", "m", "w", "m", "w", "w")) # erzeugen einer Faktorvariablen
> levels(geschlecht)     # gibt die m\"oglichen Auspr\"agungen der Faktorvariablen geschlecht an
> class(geschlecht)      # zeigt den Typ der Variablen geschlecht
```

7.3 Vektoren

Vektoren fassen mehrere gleichartige Datenelemente zu einem R-Objekt zusammen

```
> x <- c(5,3,7) # Vektor anlegen
> lenght(x)     # gibt die Anzahl der Elemente im Vektor x an
> x[2]          # auf das zweite Element des Vektors zugreifen
> x[2] <- 9     # \"andern eines Elementes des Vektors
```

7.4 Dataframes

Typischerweise hat man es in der Statistik nicht mit einzelnen Variablen, sondern meist mit ganzen Datensätzen zu tun. Dafür verwendet man Dataframes. Dataframes sind tabellenartige Datenschemata, die man sich aus mehreren Vektoren als Spalten zusammengesetzt vorstellen kann. In den Zeilen des Datensatzes stehen die einzelnen Dateneinträge.

```
> vorname <- c("Patrick","Florian","Lea")           # 1. Vektor
> alter <- c(21, 20, 20)                            # 2. Vektor
> semester <- c(3, 3, 3)                            # 3. Vektor
> studenten <- data.frame(vorname, alter, semester) # Dataframe anlegen
> View(studenten)                                   # darstellen des Dataframes
> View(studenten$alter)                             # anzeigen der Variablen alter aus dem Dataframe
> mean(studenten$alter)                             # berechnen des Mittelwertes des Alters der Studenten
```

7.4.1 Auf einzelne Elemente eines Dataframes zugreifen

```
> studenten[3,2]          # auf das Element in 3. Zeile 2. Spalte im Dataframe zugreifen
> studenten[3,2]    <- 21          # Das Element in 3. Zeile 2. Spalte im Dataframe aendern
```

7.4.2 Einen Dataframe um einen Vektor erweitern

```
> nname <- c("Meier","Vallaster","Schertler")       # 4. Vektor
> studenten$nname<- nname                           # 4. Vektor in Dataframe aufnehmen
> studenten$nname<- NULL                            # 4. Vektor wieder entfernen
```

7.4.3 Zwei Dataframes gleicher Struktur zusammenführen

- Dataframes sind hinsichtlich der Zeilen identisch

  ```
  > nname <- c("Meier","Vallaster","Schertler")
  > matrikel<-c(0815,1234, 0007)
  > studentenno<-data.frame(nname,matrikel)        # Zweiter Dataframe
  > studenten<-cbind(studenten,studentenno)        # Dataframe studenten um Zusatzinfo verbreitern
  ```

- Dataframes sind hinsichtlich der Spalten identisch

  ```
  > vorname <- c("Leon","Lisa")                    # 1. Vektor
  > alter <- c(22, 21)                             # 2. Vektor
  > semester <- c(5, 3)                           # 3. Vektor
  > studentenplus<-data.frame(vorname,alter,semester)     # Zweiter Dataframe
  > studenten<-rbind(studenten,studentenplus)             # Zwei Dataframes zeilenweise verbinden
  ```

8 Einlesen von Daten

Hierfür gibt es in R diverse Möglichkeiten

R-Befehl	geeignet für	Argumente
read.table()	.csv .txt .dat	header=... , sep=... , dec=...
read.csv()	.csv	header=... , sep=... , dec=... , fill=TRUE
scan()	.txt .dat ...	file= , what = ..., header= . sep = , fill=

Beispiel:

```
> setwd("Documents/R-Projekt/300Daten")
> x <- read.table("Evaluation.csv", header=TRUE, sep = ";", dec=",") # einlesen der csv-Datei mit Header,
                                                                     # Separator und Dezimalkomma
```

9 Graphische Ausgaben

9.1 Boxplots

```
> boxplot(x,type=5,col=c("red","","yellow"));
```

x	...	Vektor, für den ein Boxplot in einer Grafik gezeichnet werden soll
type	...	1-7 wie bei Quantilen (Type=5 verwenden)
col	...	Farbparameter

9.2 Balkendiagramme

```
> barplot(x,col=rainbow(10),main="Balkendiagramm");
```

x	...	Vektor von Häufigkeiten, für die ein Balkendiagramm erstellt werden soll
col	...	Farbparameter

9.3 Tortendiagramme

```
> pie(x,labels=names,col=rfarbe,main="Balkendiagramm");
```

x	...	Vektor von relativen Häufigkeiten, für den ein Tortendiagramm erstellt werden soll
labels	...	Name der einzelnen Tortenstückchen
col	...	Farbparameter

9.4 Histogramme

```
> hist(x,breaks=c(....));
```

oder

```
> hist(x,freq=TRUE);
```

x	...	Vektor für den ien Histogramm erstellt werden soll
breaks	...	Vektor, der die Stützstellen zwischen den Zellen beschreibt
freq	...	logischer Wert

9.5 Streudiagramm

```
> print(faithful);     # anschauen der Daten
> plot(faithful);      # Streudiagramm der Daten erstellen
```

Daten des Old Faithful Geysirs im Yellowstonepark
Die erste Spalte (eruptions= Ausbruchzeit in MInuten)
Die zweite Spalte (waiting=Wartezeit bis zjm nächsten Ausbruch in Minuten)

10 Spezielle Fragen

1. Funktion für gängige Lagemaße

```
> x <- rnorm(20);     # rnorm(20) generiert 20 normalverteilte Zufallswerte mit Erwartungswert 0 und
                      # Standardabweichung 1
> summary(x);         # generiert Minimum, 1. Quantil, Median, Mittelwert, 3. Quantil, Maximum
```

2. Histogramme mit unterschiedlichen Klassenbreiten
   Die Intervallgrenzen werden von R automatisch bestimmt, können jedoch auch explizit angegeben werden. Die Angabe

   ```
   > breaks=c(-3,-1,0,1,3)
   ```

   liefert beispielsweise die Intervalle [-3,-1), [-1,0), [0,1) und [1,3)