Statistische Software

Micha Schneider

Institut für Statistik
Ludwig-Maximilians-Universität München

WS 2015/16, R Teil 1

Was ist R?

R ist eine kostenlose Software-Umgebung für statistische Datenanalyse und Graphiken. Es beruht auf einer Implementation der Sprache S. Anfänglich wurde R von Ross Ihaka und Robert Gentleman (Univ. Auckland) entwickelt und wird seit Mitte der 90er Jahre von einem Entwickler-Kollektiv (R-Core) betreut.

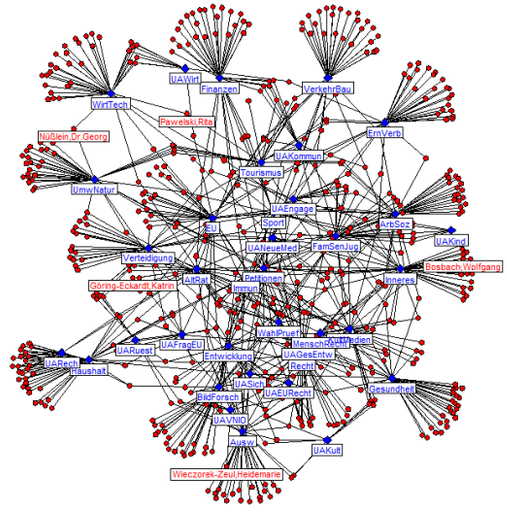
Informationen zu R:

http://www.R-project.org

Vorteile von R

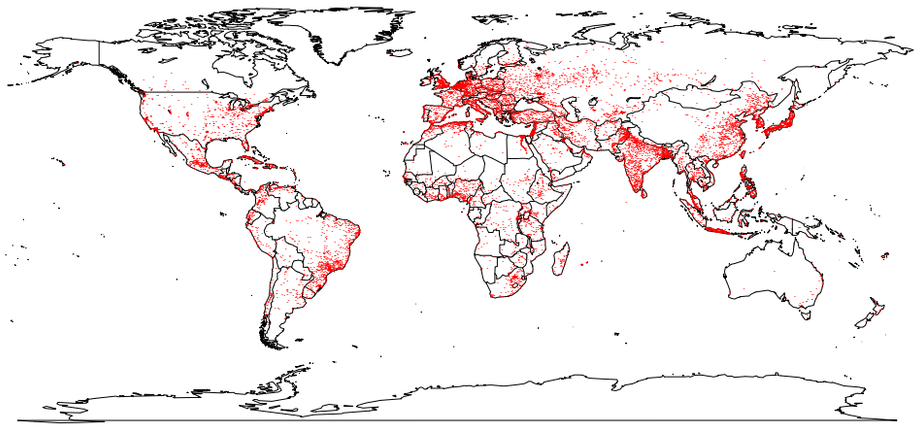
- Open Source
  (Alle Algorithmen und Funktionen im Quelltext nachvollziehbar)
- schnelle Entwicklung
- leichte Erweiterbarkeit mit Paketen
- auf (fast) jedem Betriebssystem / jeder Plattform laufähig
- keine Lizenzgebühren

Beispiel: Netzwerkanalyse



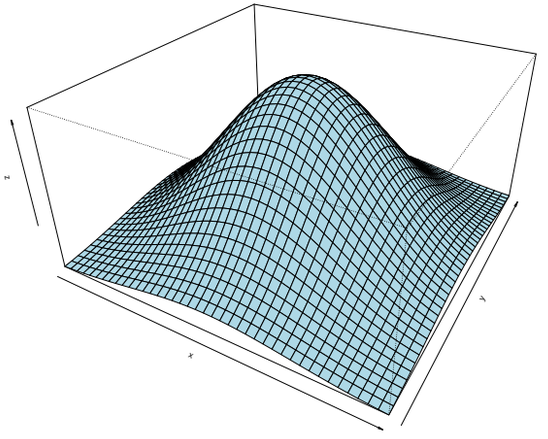
Micha Schneider (2013): Modellidentifikation in Netzwerken, Diplomarbeit. (Paket „statnet“)

Beispiel: Kartengrafik



Städte mit mitdestens 25.000 Einwohner; Daten aus Paket „maps“

Beispiel: 3D-Grafik



Bivariate Normalverteilung mit Hilfe der Pakete „mvtnorm“ und „graphics“

Nachteile von R

- keine vollständige grafische Benutzeroberfläche
- ...

R Versionen

Die Versionsnummern `x.y.z` folgen einem bei Open Source Software weit verbreiteten Schema:

Major Version `x`**:** ändert sich nur selten, ähnlich neuer Version kommerzieller Software.
- `0`: Frühe Entwicklungsversionen (90er Jahre)
- `1`: Sprache S Version 3 vollständig implementiert (2000)
- `2`: S Version 4 und wichtige neue Sprachelemente (2004)
- `3`: Einige neue Funktionen (z.B. längere Vektoren) (2013)

Minor Version `y`**:** Jeweils eine Release im Frühjahr und Herbst jeden Jahres.

Patch Level `z`**:** Der `x.y.0`-Version folgt normalerweise im Abstand von 1–2 Monaten eine `x.y.1`-Version, danach je nach Bedarf.

Wie installiere ich R?

Fertige R Distributionen sind für Windows, MacOS X und viele Linux-Versionen auf CRAN (**C**omprehensive **R** **A**rchive **N**etwork) erältlich:

- Die Website http://www.R-project.org aufrufen und auf „download R“ klicken
- Einen nahegelegenen Server auswählen
- Dem Link mit dem benötigtem Betriebsystem folgen
- Bei Windows: „base“ auswählen und R installieren

R als Taschenrechner

Ok. Aller Anfang ist schwer, also zunächst mal was Bekanntes...

```
> 1 + 1
[1] 2
> 1 + 2 * 3
[1] 7
> (1 + 2) * 3
[1] 9
> 2^3
[1] 8
> 4^0.5
[1] 2
> 200/10
[1] 20
```

Die Grundrechnungsarten folgen in der Abarbeitungsreihenfolge den üblichen Regeln („Punkt vor Strich“). Im Zweifelsfall immer besser ein Klammernpaar zu viel als eins zu wenig verwenden.

Hinweis: In R wird immer der Punkt als Dezimaltrennzeichen verwendet.

Arbeiten mit R

- Kern von R ist das Kommandofenster (Console), in dem alle Befehle ausgeführt werden. Die Eingabeaufforderung wird mit “>“ angezeigt.
- Die Befehle ansich sollten mit einem Editor in eine reine Textdatei geschrieben werden. Die Befehle werden anschließend in der Console ausgeführt.
- Graphiken werden in einem seperaten Fenster angezeigt.

Editoren für R

Für das Erstellen längerer Funktionen und Code-Blöcken ist die R Konsole ungeeignet. Mögliche Editoren sind beispielsweise:

R-Editor: Interner Skript-Editor ohne Zusatzfunktionen wie Syntax-Highlihgting

RStudio: Der Editor funktioniert unter Windows, Mac und Linux und bietet ebenso Syntax-Highlighting und eine Anbindung an R (empfehlenswert):
http://www.rstudio.com/

Eine Übersicht über weitere Editoren findet man unter http://www.sciviews.org/_rgui/projects/Editors.html .

RStudio

Die Oberfläche des RStudios teilt sich in 4 Bereiche:

Skriptfenster (links oben): R-Code, der über Symbole in der Kopfleiste oder mittels Tastenkombinationen ausgeführt werden kann.

Console (links unten): Entspricht der herkömmlichen R-Console. Neben Ausgaben finden sich hier auch Warnungen (Bearbeitung läuft weiter) und Fehlermeldungen (Abbruch der Bearbeitung).

Workspace/History (rechts oben): Anzeige der existierenden Objekte und der zuletzt ausgeführten Befehle.

Files/Plots/Packages/Help: Ordnerstruktur (ausgehend vom Arbeitsverzeichnis), bisher erzeugte Graphiken, Installieren und Updaten von Packages, Zugriff auf die Hilfe

RStudio

Anlegen einer neuen Script-Datei:

```
File -> New -> R Script
```

Öffnen von bereits vorhandenen Script-Dateien:

```
File -> Open File ...
```

Änderungen können über das Menü oder über das Diskettensymbol in der Kopfleiste gespeichert werden. Senden von Befehlen an die Console funktioniert mit Strg+Enter oder dem entsprechendem Symbol 

.

Zuweisungen

Eine Zuweisung erfolgt mit „<-“:

```
> a <- 5
> a
[1] 5
> b <- 10
```

Man sagt auch „dem Objekt „a“ wurde der Wert „5“ zugewiesen“. Objekte können leicht kombiniert werden:

```
> a*b
[1] 50
```

R überschreibt Objekte ohne Vorwarnung:

```
> a <- 1
> a
[1] 1
```

Alle Objekte, die wir während einer Sitzung direkt in der R-Console erzeugen, werden im sogenannten „Workspace“ gespeichert.

Operatoren

Vergleich von Objekten mit Hilfe logischer Operatoren:

```
> 2 > 3
[1] FALSE
> 3 > 2
[1] TRUE
> a == b
[1] FALSE
> a != b
[1] TRUE
```

+ - * /	Grundrechenarten
^	Potenzieren
<-	Zuweisungen
== != < > <= >=	logische Vergleiche
#	Kommentarzeichen

Objekte

- Jede Auswertung kann in einem Objekt abgespeichert werden.
- Ein Objekt in R kann (fast) alles sein: Variablen, Matrizen, Funktionen,...
- Als Objektnamen können fast beliebige Kombinationen aus Buchstaben, Ziffern, Punkt und Unterstrich verwendet werden.
- Objektnamen sollten mit einem Buchstaben beginnen.
- R unterscheidet zwischen Groß- und Kleinschreibung (d.h. case-sensitve).
- Jedes Objekt hat bestimmte Eigenschaften wie z. B. Datentyp.

Funktionen

R ist eine vollwertige Programmiersprache, und der größte Teil von R ist selber in der Sprache R geschrieben. Auch der Benutzer kann zu jeder Zeit neue Funktionen definieren (keine Angst, nicht Teil dieser LVA).

Funktionsaufrufe sind von der Form `funktionsname(arg1=wert1, arg2=wert2)` mit beliebig vielen Argumenten.

Die Namen der Argumente können auch weggelassen werden, jedoch muss die Reihenfolge dann exakt mit der Funktionsdefinition übereinstimmen: `funktionsname(wert1, wert2)`. Aufgrund der Fehleranfälligkeit ist dies in der Regel nicht zu empfehlen.

Achtung: Funktionen werden an den auf den Namen folgenden runden Klammern erkannt. Die Zuweisungen der Form `arg=wert` erfolgt immer mit dem Operator `=`, nie mit `<-` .

Funktionen

Beispiele:

```
> a <- 2
> b <- 2^a
> b
[1] 4

> log(b)
[1] 1.386
> log(b, 2)
[1] 2
> log(base=2, x=b)
[1] 2
> log(2, b)
[1] 0.5
```


Datentypen in R

Das einfachste Datenobjekt ist ein Vektor mit Elementen des Typs

- **numeric**: ganzzahlige oder Gleitkomma-Werte,
- **character**: beliebige Zeichen,
- **logical**: die Zustände `TRUE` und `FALSE`,
- **list**: ein Objekt beliebigen Typs, auch wieder eine Liste (rekursive Datenstrukturen!).

Jeder Vektor hat eine Länge (`length()`) und der Typ kann mittels `mode()` festgestellt werden.

```
> mode(a)
[1] "numeric"
> length(a)
[1] 1
```

Datenstrukturen in R

Unter Datenstrukturen versteht man eine Beschreibung dessen, wie die Daten dargestellt werden und angeordnet sind:

- **Vektor**: Anordnung gleichartiger Elemente (z.B. nur Zahlen, nur characters, etc.)
- **Faktor**: spezielle Anordnung für kategoriale Daten
- **Matrix**: Anordnung gleichartiger Elemente in Zeilen und Spalten
- **dataframe**: Anordnung unterschiedlicher Elemente gleicher Länge
- **Liste**: Anordnung von verschiedenen Elementen (keine Vorgaben bzgl. Typ, Länge, etc.)

Numerische Vektoren

Üblicherweise wird `c()` (concatenate: verbinden) zum Erstellen von mehrelementigen Vektoren verwendet. Weitere wichtige Funktionen sind `seq()` oder `:` (Sequenzen) und `rep()` (Repeat).

```
> v1 <- c(1, 3.14, 17)
> v1
[1]  1.00  3.14 17.00
> v2 <- seq(from = -pi, to = pi, length = 5)
> v2
[1] -3.142 -1.571  0.000  1.571  3.142
> v3 <- 1:5
> v3
[1] 1 2 3 4 5
> v4 <- rep(2, 5)
> v4
[1] 2 2 2 2 2
> rep(c(2,4), times=2)
[1] 2 4 2 4
> rep(c(2,4), each=2)
[1] 2 2 4 4
```


Zeichen-Vektoren

Die Elemente von Zeichen-Vektoren bestehen aus einer Folge von beliebigen Zeichen. Zur Unterscheidung von Variablen werden Zeichenketten durch einfache oder doppelte Hochkomma am Anfang und Ende markiert.

```
> kurs <- "Statistische Software"
> kurs
[1] "Statistische Software"
> mode(kurs)
[1] "character"
> Benutzer <- c("Hansi", "Seppl")
> Benutzer
[1] "Hansi" "Seppl"
```


Logische Vektoren

Es gibt die zwei logischen Zustände `TRUE` und `FALSE`. In den meisten Fällen werden logische Vektoren nicht direkt eingegeben, sondern durch die logischen Operatoren `==`, `!=`, ... erzeugt. Der Operator `!` invertiert einen logischen Vektor.

```
> WoIstSeppl <- (Benutzer == "Seppl")
> WoIstSeppl
[1] FALSE  TRUE
> ! WoIstSeppl
[1]  TRUE FALSE
> which(WoIstSeppl)
[1] 2
```

Zugriff auf Vektorelemente

Generell erfolgt in R der Zugriff auf Teile einer Struktur durch eckige Klammern. Bei einem Vektor können Zahlen und logische Vektoren für den Zugriff auf einzelne Elemente verwendet werden, negative Zahlen schließen einzelne Elemente aus.

```
> v2
[1] -3.142 -1.571  0.000  1.571  3.142
> v2[5]
[1] 3.142
> v2[2:4]
[1] -1.571  0.000  1.571
> v2[-(2:4)]
[1] -3.142  3.142
> v2[v2<0]
[1] -3.142 -1.571
```


Namen für Vektorelemente

Jedes Vektorelement kann einen Namen bekommen, diese können dann für den einfacheren Zugriff verwendet werden.

```
> konst <- c(e = exp(1), pi = pi, zweipi = 2 * pi)
> konst
     e     pi zweipi
 2.718  3.142  6.283
> names(konst)
[1] "e"      "pi"     "zweipi"
> names(konst)[3] <- "2pi"
> konst
    e    pi   2pi
2.718 3.142 6.283
> konst["2pi"]
  2pi
6.283
> konst[c("pi","2pi")]
   pi   2pi
3.142 6.283
```


Faktoren

Nominale oder ordinale Daten werden in R „Faktoren“ genannt. Intern ist dies ein ganzzahliger Vektor, wo jeder Zahl ein „Label“ zugeordnet ist. Zeichen-Vektoren sind keine nominale Variablen, können aber mit der Funktion `factor()` leicht in eine solche verwandelt werden.

```
> behandlung <- rep(c("Placebo", "Medikament"), c(2,3))
> behandlung
[1] "Placebo"    "Placebo"    "Medikament" "Medikament"
[5] "Medikament"
> summary(behandlung)
   Length     Class      Mode
        5 character character
> behandlung <- factor(behandlung)
> behandlung
[1] Placebo    Placebo    Medikament Medikament Medikament
Levels: Medikament Placebo
> summary(behandlung)
Medikament    Placebo
         3          2
```


Matrizen

Matrizen werden mit der Funktion
`matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)`
erzeugt, wobei die Argumente folgende Bedeutung haben:

- data: Ein Vektor mit den Daten
- nrow: Anzahl der Zeilen
- ncol: Anzahl der Spalten
- byrow: Bei TRUE wird die Matrix zeilenweise aufgebaut, sonst spaltenweise

```
> X <- matrix(c(1,2,3,4,5,6),nrow=3)
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Indizierung von Matrizen

Die Indizierung von Matrizen ist ähnlich wie der Zugriff auf Vektorelemente. Um den Wert mit Index `(i ; j)` also (Zeile i , Spalte j) einer Matrix X anzusprechen, verwendet man die Form: `X[i, j]`.
Das Weglassen einer Spaltenangabe, also etwa `X[i, ]`, ermöglicht das Ansprechen des i-ten Zeilenvektors, bzw. `X[, j]` für den j-ten Spaltenvektor.

```
> X[1,]
[1] 1 4
> X[,1]
[1] 1 2 3
> X[3,1]
[1] 3
> X[2:3,]
     [,1] [,2]
[1,]    2    5
[2,]    3    6
```


Eigenschaften von Matrizen

Es ist auch möglich Zeilen und Spaltennamen zu vergeben:

```
> colnames(X) <- c("Apfel", "Birnen")
> rownames(X) <- c("Verkauf Montag","Verkauf Dienstag","Verkauf Mittwoch")
> X
                 Apfel Birnen
Verkauf Montag       1      4
Verkauf Dienstag     2      5
Verkauf Mittwoch     3      6
```

Die Namen beeinflussen so nicht den Datentyp:

```
> mode(X)
[1] "numeric"
```


Datenmatrix

Die wohl wichtigste Struktur zur Haltung von Daten im üblichen Rechteckschema, wo die Beobachtungen in den Zeilen („i“) und die Variablen in den Spalten(„j“) dargestellt werden, ist die Datenmatrix. In R wird diese `data.frame` genannt.

```
> data("iris")
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```


Datenmatrix

Die Spalten von data frames beinhalten in der Regel numerische Vektoren oder Faktoren. Mittels `summary()` erhählt man eine schnelle Übersicht über die einzelnen Variablen:

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1
 1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3
 Median :5.80   Median :3.00   Median :4.35   Median :1.3
 Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
 3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
 Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

Datenmatrix

Mit `str()` kann man sich die Struktur des `data.frame` anzeigen lassen. Insbesondere bei größeren Datensätzen ist dieser Befehl sehr hilfreich, um einen Überblick über die Variablen und deren Kodierung zu bekommen.

```
> str(iris)
'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```


Teilmengen von Datenmatrizen

Man kann wieder den Operator [benutzen, um Zeilen oder Spalten (oder beides) aus data frames zu extrahieren. Die Indizierung funktioniert analog wie bei den Matrizen.

```
> dim(iris)
[1] 150   5
> iris[1:5,1:2]
  Sepal.Length Sepal.Width
1          5.1         3.5
2          4.9         3.0
3          4.7         3.2
4          4.6         3.1
5          5.0         3.6
> iris[1:2,-(1:2)]
  Petal.Length Petal.Width Species
1          1.4         0.2  setosa
2          1.4         0.2  setosa
```


Teilmengen von Datenmatrizen

Auf benannte Spalten (also in der Regel Variablen) eines data frames kann auch einfacher mit dem $-Operator zugegriffen werden:

```
> iris$Sepal.Width
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0
 [16] 4.4 3.9 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2
 [31] 3.1 3.4 4.1 4.2 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8
 [46] 3.0 3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7
 [61] 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7 2.2 2.5 3.2 2.8 2.5 2.8 2.9
 [76] 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0 3.4 3.1 2.3 3.0 2.5
 [91] 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7 3.0 2.9 3.0
[106] 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6 2.2
[121] 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6
[136] 3.0 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```


Listen

Eine Liste erzeugt man mit dem Befehl `list(...)`, wobei `...` die Elemente der Liste enhält. Listen können beliebige Objekte enthalten, auch Objekte verschiedenen Typs:

```
> liste <- list(umsatz=100, Verkauf=X, Produkte=c("Apfel", "Birnen"))
> liste
$umsatz
[1] 100

$Verkauf
                Apfel Birnen
Verkauf Montag      1      4
Verkauf Dienstag    2      5
Verkauf Mittwoch    3      6

$Produkte
[1] "Apfel"  "Birnen"
```

Fehlende Werte

Die Codierungen `NA` (Not Available) und `NaN` (Not a Number) bezeichnen fehlende Werte in einem Vektor oder anderen Datenstruktur. `NA` kann sich aus der Datenerhebung ergeben, kann das Ergebnis einer (fehlgeschlagenen) Berechnung sein oder auch zugewiesen werden. Mit der Funktion `is.na()` kann überprüft werden, wo fehlende Werte auftreten. Bei einigen Funktionen können diese mittels des Arguments `na.rm` ausgeschlossen werden.

```
> fehl <- c(1, 2, NA, 3)
> is.na(fehl)
[1] FALSE FALSE  TRUE FALSE
> mean(fehl)
[1] NA
> mean(fehl, na.rm=TRUE)
[1] 2
```


Arbeitsverzeichnis

Mit der Funktion `getwd()` kann das aktuelle Arbeitsverzeichnis (Working Directory) abgefragt werden. Gegebenenfalls wird es mit `setwd()`, geändert, z.B.

```
> setwd("Z:/REinfuehrung")
```

oder

```
> setwd("Z:\\REinfuehrung")
```

Falls nichts anderes angegeben wird, nimmt R an, dass sich zu ladende Datensätze in diesem Verzeichnis befinden und speichert Graphiken in dieses Verzeichnis ab.

Laden und Einlesen von Daten

Laden einer .RData-Datei, die z.B. die Hundedaten aus der Vorlesung enthält:

```
> load("Daten/Hunde.RData")
```

Text- und .csv-Dateien können mit den Funktionen `read.table()`, `read.csv()` und `read.csv2()` eingelesen werden, z.B.

```
> miete <- read.table("Daten/miete03.asc", header=TRUE)
```

Dabei kann z.B. das Trennzeichen und das Dezimaltrennzeichen spezifiziert werden.

Das Package `foreign` enthält Funktionen zum Einlesen von Formaten aus anderen Programmpaketen, wie z.B.`read.spss()`. Excel-Dateien lassen sich z.B. mit dem Package `xlsx` einlesen.

Speichern von Daten und Objekten

Mit der Funktion `save` können Objekte in einer .RData-Datei abgespeichert werden.

```
> save(Hunde, file="Daten/KopieHunde.RData")
```

Datensätze können mit den Funktionen `write.table()`, `write.csv()` und `write.csv2()` auch als Text- oder .csv-Datei exportiert werden. Die Befehle unterscheiden sich insbesondere darin, welche (default-)Werte z.B. für das Trennzeichen eingestellt sind.

```
> write.csv2(Hunde, file="Daten/Hundedaten.csv")
```

Pakete

Eine Vielzahl von Funktionalitäten sind nicht im Basissystem, sondern in Zusatzpaketen implementiert. Diese werden zentral auf `http://CRAN.R-project.org` gehalten und können direkt von R aus installiert werden.

```
> install.packages("mvtnorm")
```

installiert das Paket zur Berechnung von multivariaten t- und Normalverteilungen. Um das Paket benützen zu können, muss es (in jeder neuen Sitzung) mit

```
> library(mvtnorm)
```

geladen werden. Die Hilfe zu einem Paket kann mit `help(package=mvtnorm)` aufgerufen werden. Ein Erweiterungspaket, das eine SPSS-ähnliche Oberfläche für bestimmte Funktionen bietet, ist der sogenannte R Commander (`Rcmdr`).

Hilfe

Zu einer Funktion (z. B. `log()`) kann die entsprechende Hilfeseite mit

```
> ?log
```

oder

```
> help(log)
```

aufgerufen werden.

Ist der Name der benötigten Funktion unbekannt, kann vorher mit `apropos()` danach gesucht werden. Eine gute Übersicht über R interne Handbücher findet man mit `help.start()`.

Literatur zum Einstieg

- Richard Cotton: *Learning R*
- Peter Dalgaard: *Introductory Statistics with R*
- Uwe Ligges: *Programmieren mit R*
- Daniel Wollschläger: *Grundlagen der Datenanalyse mit R*