Teil 3: Mehr R

Die R-Arbeitsumgebung

- R-Hauptfenster

Benutzeroberfläche(Rgui – graphical user interface)
zur Eingabe von Befehlen, Ausgabe von Output
besteht aus mehreren Subfenstern

- Workspace

Arbeitsspeicher des Computers
alle während einer R-Sitzung definierten Variablen, Funktionen, etc.

- Working Directory

zugeordneter Verzeichnispfad
zum Lesen und Schreiben externer Dateien (ohne Pfadangabe)

Das R-Hauptfenster (Benutzeroberfläche)

besteht aus:

- Container- oder Master-Fenster
  enthält
  - R-Konsole (Eingabe von Befehlen, numerischer Output)
  - R-Grafikfenster (Grafik-Output)
  - R-Editor
  - R-Dateneditor

- Menü- und Symbolleisten
  je nach Sub-Fenster unterschiedliche Funktionalitäten

Die R-Konsole

- Schaltzentrale und **zentrales Sub-Fenster**
- Eingabe von R-Befehlen und Retournierung von Ergebnissen
- Für Menü- und Symbolleisten existieren auch R-Befehle
- Ermöglicht auch vollautomatische Aufgabenabwicklung ohne direkte Interaktion mit R
- Durch schließen der R-Konsole beenden von R

Das R-Grafikfenster

- Angeforderte Grafiken werden in einem Grafikfenster dargestellt
- Erzeugen weiterer Grafiken überschreibt die Vorhergehenden
- bleibt so lange geöffnet, bis man es aktiv schließt oder R beendet

Der R-Editor

- Eingabe oder Einfügen von R-Befehlen
- Einlesen oder modifizieren von Daten

Der R-Dateneditor

- Hier können Daten eingegeben und modifiziert werden.

Die Menü- und Symbolleisten

- änderen sich je nach aktivem Fenster
- Beschreibung des jeweiligen Symbols, wenn mit der Maus – ohne zu klicken – auf das Symbol gezeigt wird
- Menüpunkt *Bearbeiten* → *GUI Einstellungen*
  Definieren des Erscheinungsbilds des R-Hauptfensters (z.B. Schrifttyp und Farbe der Ausgabe, Zeilenbreite usw.)
  Einstellungen können gespeichert und wieder geladen werden
- Menüpunkt *Hilfe* → *Konsole*
  Informationen zur Tastensteuerung

Der Workspace

hier liegt eine Sammlung von Objekten

- Vektoren, Matrizen, Data Frames, Funktionen, . . .
- werden bei der interaktiven Bedienung von R erzeugt
- solange R läuft sind diese verfügbar (befinden sich im Arbeitsspeicher)
- `objects()` bzw. `ls()` retourniert die Namen der momentan im Workspace gespeicherten Objekte
- mit `rm()` löschen einzelner Objekte
  Menüpunkt *Entferne alle Objekte*: löschen aller Objekte
- Menüpunkte *Datei* → *Sichere Workspace. . .*
  speichern des gesamten Workspace in `.RData` Datei

Beim Beenden von R erscheint die Frage: *Workspace sichern?*
bei JA werden im aktuellen Arbeitsverzeichnis die Dateien `.Rhistory` und `.RData` angelegt.

Working Directory – das Arbeitsverzeichnis

Definition eines Standardverzeichnisses erfolgt automatisch beim Starten von R

- `getwd()` Abfragen des aktuellen Arbeitsverzeichnisses
  Beispiel:

```
> getwd()
[1] "C:/Users/Ruser/"
```

- `dir()` auflisten aller Dateien im aktuellen Arbeitsverzeichnis
- `setwd()` wechseln des aktuellen Arbeitsverzeichnisses
  Beispiel:

```
> setwd("D:/Data/Work/")
```

  alternativ: Menüpunkte *Datei* → *Verzeichnis wechseln...*

Festlegen eines eigenen Working Directories ist praktisch:

- man braucht beim Lesen/Schreiben von Dateien keinen Pfad angeben
- leichter Ordnung halten

Achtung: bei Dateien die nicht im Arbeitsverzeichnis liegen

- R benutzt / bzw. \\ als Trennzeichen bei Pfadangaben
- z.B. `D:/ProjektXY/Daten` bzw. `D:\\ProjektXY\\Daten`
- nicht wie in Windows: \

Hinweis:
ein bestimmtes Arbeitsverzeichnis beim Aufruf von R definieren

- Rechtsklick auf R-Icon, *Eigenschaften* öffnen
- den Pfad zum Arbeitsverzeichnis unter *Verknüpfung* und *Ausführen in:* eintragen

R-Grafik

Unterteilung von Grafikbefehlen in R in zwei Hauptgruppen:

- High-level Plotting Functions:
  Funktionen für vollständige Grafiken
- Low-level Plotting Functions:
  Funktionen für einzelne Grafikelemente

Zusätzlich:

- Grafikparameter:
  steuern und modifizieren einzelner Aspekte einer Grafik

High-level Plotting Functions

- Funktionen zum Erzeugen vollständiger Grafiken
- erzeugt je nach Datenstruktur automatisch bestimmte Grafikelemente (z.B. Titel, Achsen, Beschriftungen)
- Ausgabe im R-Grafikfenster
- einige wichtige High-level-Plotfunktionen:
  `plot()`, `barplot()`, `dotchart()`, `hist()`

Low-level Plotting Functions

Anwendungsbereiche:

- man fügt zu einer Grafik etwas hinzu
- man erzeugt eine Grafik von Grund auf
- einige wichtige Low-level-Plotfunktionen:
  `points()`, `lines()`, `text()`, `axis()`

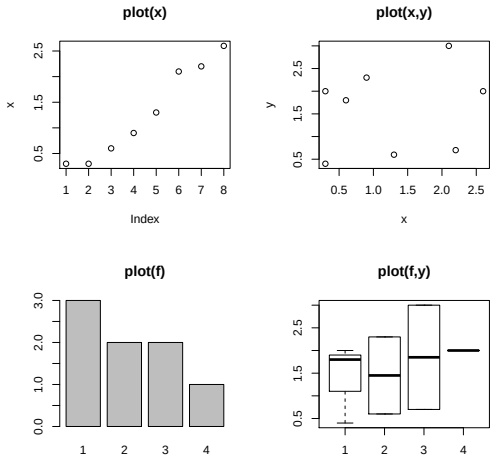
Die Funktion `plot()`

Produziert, abhängig vom Datentyp des ersten Arguments, verschiedene Arten von Grafiken.

Beispiel:
zwei metrische Variablen `x` und `y` sowie ein Faktor `f`

```
> x <- c(0.3, 0.3, 0.6, 0.9, 1.3, 2.1, 2.2, 2.6)
> y <- c(0.4, 2, 1.8, 2.3, 0.6, 3, 0.7, 2)
> f <- factor(c(1, 1, 1, 2, 2, 3, 3, 4))

> oldpar <- par(mfrow = c(2, 2))
> plot(x, main = "plot(x)")
> plot(x, y, main = "plot(x,y)")
> plot(f, main = "plot(f)")
> plot(f, y, main = "plot(f,y)")
> par(oldpar)
```

im Einzelnen:

- `plot(x)`
  links oben: Darstellung der Werte von `x` gegen ihren Index auf der y-Achse.
- `plot(x,y)`
  rechts oben: Streudiagramm (Auftragen der Punkte von `x` gegen `y`). Erstes Argument `x` wird auf x-Achse und Zweites `y` auf y-Achse dargestellt.
- `plot(f)`
  links unten: Balkendiagramm (Häufigkeiten der Kategorien des Faktors `f`).
- `plot(f,x)`
  rechts unten: sog. Boxplots (Verteilung der Werte von `x` nach den Kategorien von `f`)

Weitere Argumente bei High-level-Plotfunktionen

die ersten Argumente sind immer Daten

weitere Argumente und Optionen:

- `axes = FALSE` unterdrückt die Erzeugung von Achsen.
- `type =`
  - `"l"` verbindet Datenpunkte mit Linien (ohne Punkte)
  - `"b"` zeichnet sowohl Linien als auch Punkte
  - `"n"` leerer Plot
- `xlab = "Zeichenkette"` Beschriftung horizontale Achse
- `ylab = "Zeichenkette"` Beschriftung vertikale Achse
- `xlim = c(x1, x2)`
- `ylim = c(y1, y2)`
  Beschränkung des Bereichs, in dem geplottet wird
  wenn `x1 > x2`, dann Achse umgedreht
- `main = "Zeichenkette"` Titel eines Plots
- `sub = "Zeichenkette"` Untertitel eines Plots

Beispiel
Logarithmusfunktion für Werte zwischen 1 und 10

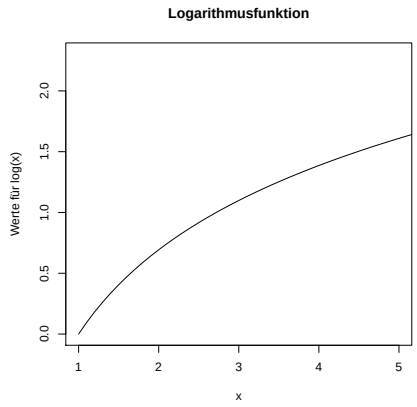
die einzelnen Schritte:

- zunächst Erzeugen eines Vektors `x` mit den Zahlen 1 bis 10, in 0.1-Schritten
  mittels `seq(from, to, by)`

```
> x <- seq(from = 1, to = 10, by = 0.1)
> head(x)
[1] 1.0 1.1 1.2 1.3 1.4 1.5
```

- die Funktion soll als Kurve gezeichnet werden (`type = "l"`)
- die y-Achse soll mit `Werte für log(x)` beschriftet werden (`ylab = "Werte für log(x)"`)
- es sollen nur Werte von 1 bis 5 dargestellt werden (`xlab = c(1,5)`)
- die Grafik soll mit „Logarithmusfunktion“ betitelt werden (`main="Logarithmusfunktion"`)


```
> plot(x, log(x), type = "l", ylab = "Werte für log(x)", xlim = c(1,
+     5), main = "Logarithmusfunktion")
```



Low-level Plotting Functions (Hinzufügen von Grafikelementen)

die wichtigsten Low-level-Plotfunktionen:

- `points(x, y)`
  zeichnet Punkte an den Koordinaten, die in den Vektoren `x` und `y` stehen (analog zu `plot(x, y, type = "p")`)
- `lines(x, y)`
  verbindet die Punkte, deren Koordinaten in den Vektoren `x` und `y` stehen (analog zu `plot(x, y, type = "l")`)
- `text(x, y, labels)`
  schreibt Text, der im Vektor `labels` steht, an den Koordinaten `x` und `y` (ähnlich `points()` nur mit Text)
- `abline()` Zeichnen einer (Regressions-)Geraden
- `legend()` Erzeugen einer Legende
- `title(main, sub)` erzeugt Titel und Untertitel
- `axis(side, ...)` wird gleich beschrieben.

Beispiel:
Streudiagramm: Gewicht und Körpergröße von 5 Personen

```
> GEWICHT <- c(56, 63, 80, 49, 75)
> GRÖSSE <- c(1.64, 1.73, 1.85, 1.6, 1.81)
> namen <- c("Gerda", "Karin", "Hans", "Doris", "Ludwig")
```

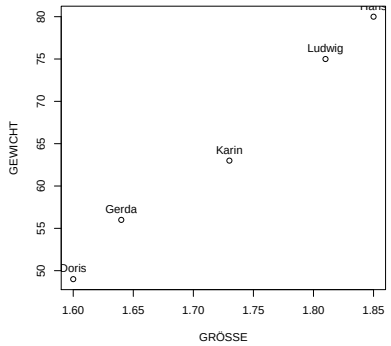
Erstellen der Grafik

```
> plot(GRÖSSE, GEWICHT)
```

Modifikation mittels

- `text()` hinzufügen der Namen zu den Punkten
- `pos =` Zahl – Plotposition des Texts
  unterhalb (1), links (2), oberhalb (3), rechts (4) der jeweiligen Koordinaten

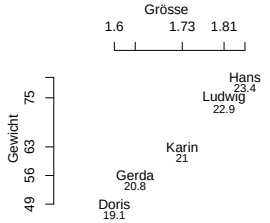
```
> text(GRÖSSE, GEWICHT, labels = namen, pos = 3)
```

Anmerkung:
Beschriftung teils ausserhalb des Darstellungsbereichs
weil durch `plot()` schon festgelegt, `text()` kam erst nachher
→ Ändern des Darstellungsbereiches mittels `xlim =` und `ylim =`

Beispiel: gleiche Grafik aber

- statt der Punkte sollen die Namen verwendet werden
- unterhalb der Namen soll der Body-Mass-Index stehen
- um zu wissen, wie groß und schwer die Personen sind, sollen diese Werte an den Achsen stehen
- die Achsen sollen oben und rechts gezeichnet werden



Dazu notwendige Befehle:

- erzeugen der Grafik, wird nicht dargestellt: `type = "n"`
  Unterdrücken der Achsen: `axes = FALSE`
  und deren Beschriftung: `xlab = "", ylab = ""`

```
> plot(GRÖSSE, GEWICHT, type = "n", axes = FALSE, xlab = "", ylab = "",
+     xlim = c(1.5, 1.9), ylim = c(45, 85))
```

- schreiben der Namen der Personen: `labels =`
  zentriert an die Stellen, definiert durch `GRÖSSE` und `GEWICHT`

```
> text(GRÖSSE, GEWICHT, labels = namen)
```

- zeichnen der Achsen links: `side = 2` und oben: `side = 3`
  Ticks an den Stellen `at` der Werte von `GRÖSSE` und `GEWICHT`
  Beschriftung der Ticks: `labels` mit entsprechenden Werten

```
> axis(side = 2, at = GEWICHT, labels = GEWICHT)
> axis(side = 3, at = GRÖSSE, labels = GRÖSSE)
```

- berechnen und des Body-Mass-Index und runden

```
> bmi <- round(GEWICHT/(GRÖSSE^2), digits = 1)
```


- schreiben der BMI-Werte unterhalb: `pos = 1` der Namen
  reduzieren der Schriftgröße auf 80%: `cex = 0.8`

```
> text(GRÖSSE, GEWICHT, labels = bmi, pos = 1, cex = 0.8)
```

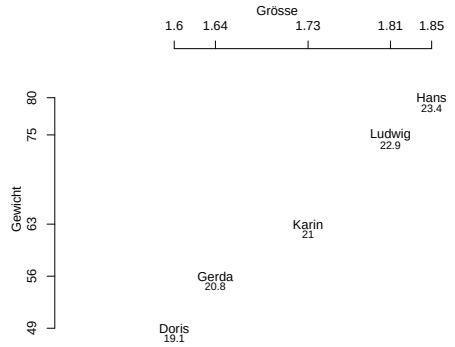
- Achsennamen: welche Achse: `side`, Abstand von Achse: `line`

```
> mtext("Gewicht", side = 2, line = 2)
> mtext("Grösse", side = 3, line = 2)
```

Alles zusammen:

```
> plot(GRÖSSE, GEWICHT, type = "n", axes = FALSE, xlab = "", ylab = "",
+     xlim = c(1.5, 1.9), ylim = c(45, 85))
> text(GRÖSSE, GEWICHT, labels = namen)
> axis(side = 2, at = GEWICHT, labels = GEWICHT)
> axis(side = 3, at = GRÖSSE, labels = GRÖSSE)
> bmi <- round(GEWICHT/(GRÖSSE^2), digits = 1)
> text(GRÖSSE, GEWICHT, labels = bmi, pos = 1, cex = 0.8)
> mtext("Gewicht", side = 2, line = 2)
> mtext("Grösse", side = 3, line = 2)
```


Resultat:



Spezielle Einstellungen (Graphical Parameters)

es gibt eine Vielzahl von Grafik-Parametern
sie kontrollieren Farben, Linienstärken, Anordnung einzelner Grafiken, Textausrichtung etc.

Standardeinstellungen erhält man mit `par()`

Einteilung in zwei Klassen:

- einfache Charakteristika (Farbe, Linientyp, -stärke, etc.)
- übergeordnete Aspekte (Achseneigenschaften, multiple Grafiken, etc.)

sie können auf zwei Arten gesetzt bzw. geändert werden:

- mit der Funktion `par()`, z.B. `par(lty = 1)`
  Einstellungen bleiben bestehen bis sie durch weiteren `par()`-Befehl geändert werden oder Grafikfenster geschlossen wird
- beim Aufruf von Grafikfunktionen als Argument
  dann Wirksamkeit nur temporär

Einige wichtige Grafikparameter

Eine Auswahl an Grafikparametern:

- `pch=` definiert den *plotting character* (z.B.: Zahl zw. `0` - `25`, oder Zeichen wie Buchstaben, Zahlen, etc.)
- `lty=` spezifiziert den Linientyp (engl. *linetype*) z.B. Zahl zw. `0` - `6` (unsichtbar = `0`, durchgezogen = `1` etc.) - oder `"blank"` (unsichtbar), `"solid"` (durchgezogen), `"dashed"` (strichliert), `"dotted"` (punktiert)
- `lwd=` spezifiziert die Linienstärke (engl. *linewidth*) über eine Zahl, die das Vielfache der Standardstärke festlegt.
- `cex=` definiert die Textgröße als Vielfaches der Standardgröße.
- `col=` definiert die Farbe von Grafikelementen Standardfarben `1` - `8`, konkrete Farbnamen mittels Zeichenkette (Abfrage in R verfügbarer Farben mittels `colors()`)
- `mfrow=` bzw. `mfcol=` für multiple Grafiken

Eine komplette Auflistung in der Hilfe zu `par()` (`?par`).

Permanente Änderungen der Grafikparameter mit `par()`

`par()` kann auf zwei Arten verwendet werden:

- Abfragen von Grafikparametern
  z.B. Abfrage des eingestellten Linientyps

```
> par("lty")
[1] "dashed"
```

- Setzen von Grafikparametern
  z.B. standardmäßiges Setzen auf gestrichelt und doppelte Linienstärke

```
> par(lty = 2, lwd = 2)
```

meistens jedoch nur Änderung einiger Parameter
danach Rückkehr zu ursprünglichen Einstellungen
→ einfachste Möglichkeit: Grafikfenster schließen
oder
`def.par<-par(no.readonly=TRUE)` speichern der Standardwerte
`par(def.par)` Rückkehr zu ursprünglichen Einstellungen

Multiple Grafiken

mittels `par(mfrow=c(nr,nc))` oder `par(mfcol=c(2,2))`

Definition einer „Grafikmatrix“ in der Form `c(nr, nc)`
hierbei ist `nr` Anzahl der Zeilen und `nc` Anzahl der Spalten

bei

- `mfrow=` zeilenweises Auffüllen
- `mfcol=` spaltenweises Auffüllen

Beispiel: `par(mfrow=c(2,2))`

- teilt das Grafikfenster in 4 Bereiche: 2×2
- Felder werden sukzessive mit einzelnen Plots aufgefüllt
- wenn alle Felder belegt sind:
  - neues Grafikfenster mit neuer (leerer) Grafik-Matrix
  - Auffüllen beginnt von Neuem
- Beenden mit `par(mfrow=c(1,1))`

Beispiel Hinzufügen Elemente

Daten

```
> bank <- read.table("http://statmath.wu-wien.ac.at/~hatz/Rkurs/bnk.dat",
+     header = TRUE)
> attach(bank)
```

Voreinstellung speichern, Hintergrundfarbe

```
> old.par <- par(no.readonly = TRUE)
> par(bg = "lightblue1")
```

High Level Grafik

```
> hist(SALNOW, col = "lightgreen", freq = FALSE, xlim = c(0, 50000),
+     main = "Gehalt in Dollar")
```

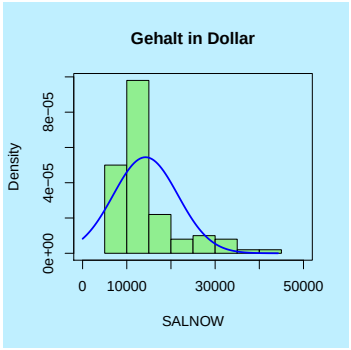
Berechnen der geschätzten Normalverteilungskurve für `SALNOW`

```
> x <- SALNOW
> sal.x <- seq(0, max(x), length.out = length(x))
> sal.y <- dnorm(sal.x, mean(x), sd(x))
```

Hinzufügen der geschätzten Normalverteilungskurve

```
> lines(sal.x, sal.y, col = "blue", lwd = 2)
> par(old.par)
```

Beispiel Hinzufügen Elemente

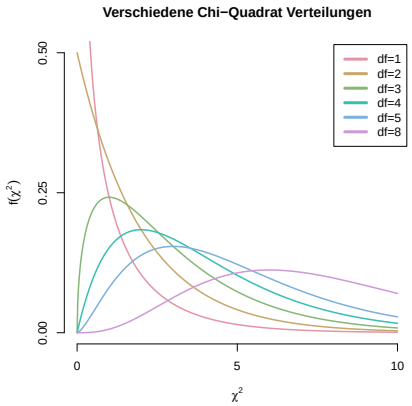


Beispiel von Null weg

χ^2-Verteilungen mit verschiedenen Freiheitsgraden:

```
> df = c(1, 2, 3, 4, 5, 8)
> library(colorspace)
> cols = rainbow_hcl(length(df))
> lw = 2
> x = seq(0, 10, length = 500)
> plot(x, type = "n", ylim = c(0, 0.5), xlim = c(0, 10), axes = FALSE,
+     xlab = expression(chi^2), ylab = expression(f(chi^2)))
> axis(1, at = c(0, 5, 10))
> axis(2, at = c(0, 0.25, 0.5))
> for (i in seq(along = df)) {
+     lines(x, dchisq(x = x, df = df[i]), col = cols[i], lwd = lw)
+ }
> legend(x = "topright", inset = 0.01, legend = paste("df=", df,
+     sep = ""), col = cols, lwd = lw + 2)
> title("Verschiedene Chi-Quadrat Verteilungen")
```


Beispiel von Null weg



Weitere Beispiele mit Code

http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html

http://de.wikipedia.org/wiki/Benutzer:Thire/Bilder#Mathematische_Grafiken

http://addictedtor.free.fr/graphiques/

Weiterverwenden des R-Outputs

Text- bzw. Zahlenmaterial:
- markieren – kopieren – in Zielanwendung einfügen

Grafiken:
- direktes Kopieren: (zwei Möglichkeiten)
  - Maus: *Rechtsklick → Kopieren als Metafile/Bitmap*
  - Menü: *Datei → Kopiere in Zwischenablage → Metafile/Bitmap*

  Grafik ist nun in Zwischenablage (dann einfügen mit *Strg+V*)

- als Datei speichern (zwei Möglichkeiten, wie oben)

Hinweise:
- für Word, etc., Metafiles verwenden (Dateiendung `.emf`)
- vor Kopieren bzw. Speichern der Grafik sollte Größe des Grafikfensters in R so geändern werden, wie sie in der Zielanwendung gewünscht wird

Einlesen von R-Skripts

die in einer `.R`-Datei gespeicherten Befehlssequenzen können direkt daraus eingelesen und ausgeführt werden
- Menüpunkte *Datei → Lese R Code ein...*

oder
- `source("path/filename.R")` bzw. `source(file.choose())`

Befehle werden der Reihe nach abgearbeitet
Output wird in der R-Konsole bzw. im R-Grafikfenster dargestellt
Einlesevorgang bricht bei Fehlern im Code ab

geht auch direkt aus dem Internet, z.B.

```
> source("http://statmath.wu.ac.at/data/exmpl.R")
```

direktes Kopieren aus externen Dokumenten (auch pdf)
einfügen in R: *Rechtsklick → Füge nur Befehle ein*

Das `.Rhistory` File

- enthält alle während einer R-Sitzung eingegebenen Befehle (bis zum Zeitpunkt der Speicherung)
- eignet sich sehr gut zum Speichern von R-Code in eigenem Skript
- wird meist nicht direkt eingelesen sondern üblicherweise vorher überarbeitet (wegen etwaiger Fehler, die während der Eingabe gemacht wurden)

Hinweise:
- Speicherung der R-Befehle erfolgt kummulativ bei mehrmaliger Speicherung hintereinander
- aktuelle `.Rhistory` mittels `history()` im R-Editor

Einlesen externer Daten-Files: Excel

direktes Einlesen aus Excel-Dateien möglich
aber einfacher ist Zwischenschritt über CSV-Dateien:
speichern der Daten in Excel als *CSV (Trennzeichen-getrennt) (*.csv)*

Hinweise:
- nur Daten eines bestimmten Typs (*numeric/character*) in einer Spalte
- wenn Variablennamen, dann in erster Zeile
- *character*-Variablen werden automatisch in Faktoren umgewandelt

CSV-Dateien in R einlesen mit
- `read.csv()` – für englische Excel Version
- `read.csv2()` – für deutsche Excel Version

Einlesen der CSV-Daten in R:

Beispiel:

```
> fragdat1 <- read.csv2("fragebogen.csv", header = TRUE)
> head(fragdat1)
```

```
  id sex lalt gross mon date entsch proj i1 i2 i3 i4 i5
1 11   w    2   173 266    4      3    2  2  3  3  2  2
2 16   w    3   166 241    5      4    1  4  2  3  1  1
3 17   m    2   178 231    3      4    2  2  1  3  2  4
4 18   w    3   154 265    3      5    2  5  3  2  4  1
5 19   w    1   164 225    2      3    2  1  4  2  2  3
6 20   m    1   389 229    4      1    1  5  2  2  1  4
```

Argumente:

- File-Name
- `header = TRUE`
  wenn Variablennamen in erster Zeile der Datei
- sonst: `header = FALSE`
  Benennung in R ist dann `V1`, `V2`, ...)

Einlesen externer Daten-Files: SPSS

mit `read.spss()`

notwendiges Package `foreign` (kann SPSS, SAS, Stata, etc.)

Beispiel:

```
> library("foreign")
> frag3 <- read.spss("file.sav", to.data.frame = TRUE, use.value.labels = TRUE)
```

Argumente:

- `file` File-Namen mit Endung `.sav`
- `to.data.frame = TRUE` sehr wichtig!
  falls nicht angegeben, wird kein Data Frame erzeugt
- `use.missings = to.data.frame` sehr wichtig!
  wandelt SPSS-fehlende Werte in `NA` um
- `use.value.labels = TRUE`
  verwendet SPSS Werte-Labels als Kategorien von Faktoren

Direktes Kopieren – Einfügen über die Zwischenablage

Vorgehen für SPSS, Excel oder andere Programme:

- Markieren des interessierenden Bereichs mit der Maus, Kopieren in die Zwischenablage (aus einem Menü oder durch rechten Mausklick).
- In R verwenden der Funktion `read.table()`, genauso wie vorhin beschrieben, man schreibt aber statt des File-Namens `"clipboard"`.

Beispiel:

- Webseite: `http://lib.stat.cmu.edu/datasets/Andrews/T02.1`
  markieren der Daten → *Strg+C* zum Kopieren
- in R verwenden von `"clipboard"` als Filename

  ```
  > andrews <- read.table("clipboard", header = FALSE)
  ```

  Daten stehen damit im Data Frame `andrews` zur Verfügung

Schreiben von Dateien

`write.table()` schreibt Daten im Text-Format in eine Datei

`write.csv()` bzw. `write.csv2()` erzeugen CSV-Dateien

Argumente:

- file
- `row.names = FALSE` unterdrückt Abspeichern der Zeilennamen
- `col.names = FALSE` unterdrückt analog die Variablennamen

Beispiel:

Abspeichern eines Data Frames

```
> write.table(andrews, file = "andrews.txt", row.names = FALSE)
```

erzeugt eine Datei `andrews.txt`
kann in Windows mit dem Programm *Editor* geöffnet werden

Das R-Hilfesystem und weiterführende Information

Hilfe zu R gibt es in unterschiedlichster Form:
- Einzelinformation zu Funktionen und Packages
- ausführliches Dokumentationsmaterial (Manuals, Tutorials)
- Suchmaschinen
- Wikis und Hilfe-Foren
- Webseiten mit Überblick über Methoden zu einzelnen wissenschaftlichen Fachbereichen

Einzelinformation zu Funktionen und Packages:
- über Menü: *R Funktionen (Text)...*
- mittels `help()`
- mittels `?`

Beispiel:
```
> help("subset")  # oder
> ?subset
```


R Hilfeseiten

Hilfeseiten zu Funktionen in R sind standardisiert und umfassen immer die gleichen Elemente:

- Usage: Beschreibung der Verwendung der Funktion sowie Auflistung aller Argumente (Optionen) die verwendet werden können.
- Arguments: Beschreibung der Optionen und deren Spezifikation.
- Value: Beschreibung des Output der Funktion (was und in welcher Form von der Funktion erzeugt wird).
- Example: Beispiele zur Verwendung der Funktion (man kann diese leicht mit Kopieren und Einfügen in R ausprobieren), alternativ kann mit der Funktion `example()`, alles, was unter Examples angeführt ist, auf einmal ausgeführt werden.

Daneben gibt es manchmal noch:
- Details: hier erfolgt eine detailliertere Beschreibung der Funktion.
- See also: hier gibt es manchmal Links zur Hilfe zu ähnlichen oder verknüpften Funktionen.

Tipps:
- Hilfeseiten sind oft sehr detailliert und beschreiben manchmal technische Details, die für Anfänger nicht wichtig sind
- nur das lesen, was man versteht
- ausprobieren der Examples

Suche im WWW

- `http://search.r-project.org`
  (auch über das R Hilfe-Menü erreichbar)
- `http://www.rseek.org`
- `http://www.dangoldstein.com/search_r.html`

Bücher:

- The R Book von Michael J. Crawley: Eine sehr ausführliche Darstellung von R und der Verwendung auch fortgeschrittener statistischer Methoden
- R Graphics von Paul Murrell: Referenzbuch zu Grafiken
- Programmieren mit R von Uwe Ligges: zum Programmieren mit R und/oder dem grundlegenden Verständnis wesentlicher Strukturen von R
- R Einführung durch angewandte Statistik von Hatzinger et.al