R you ready for R?

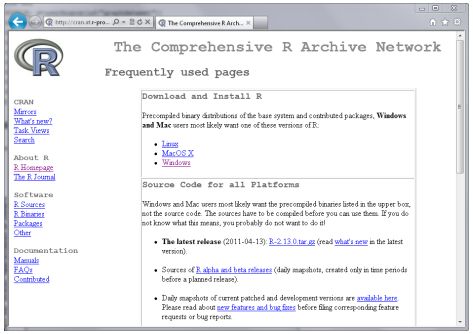
Reinhold Hatzinger

Institute for Statistics and Mathematics
Kompetenzzentrum für empirische Forschungsmethoden

Teil 1: Erste Schritte

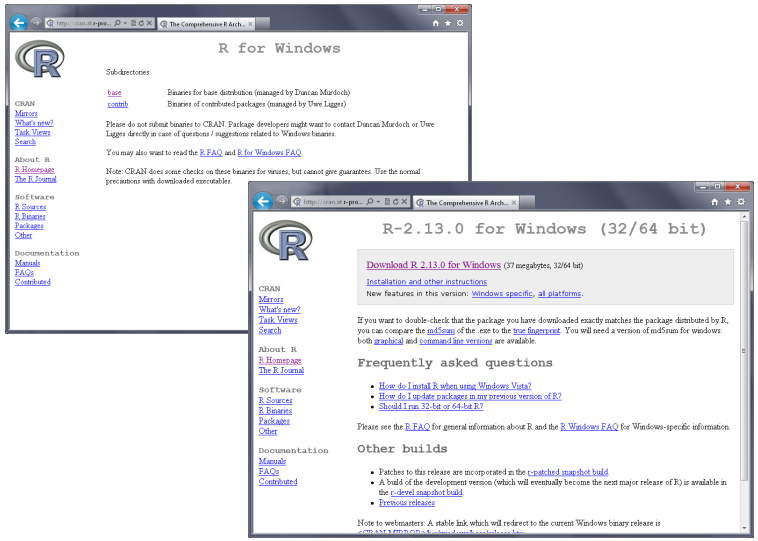
Download und Installation von R

CRAN (Comprehensive R Archive Network)
`http://CRAN.R-project.org/`



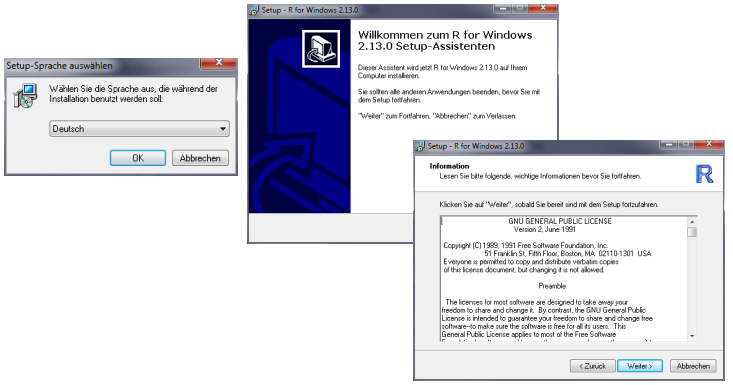
Download

- auf CRAN Hauptseite: Überschrift *Download and Install R*
- klicken auf *Windows* → neue Seite → klicken auf *base*
- hier ist aktuelle Version von R:
  herunterladen unter *Download R x.yy.z for Windows*
  (*x.yy.z* steht für die aktuelle Programmversion, z.B. 2.13.0).

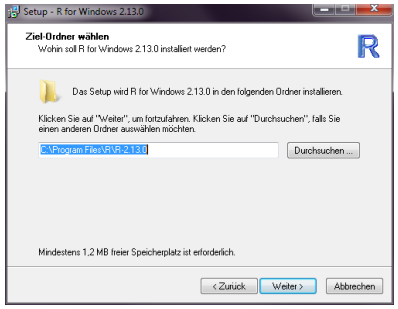


Installation

nach dem Download startet man die Datei (ggf. doppelklicken).
bei den ersten 3 Fenstern kann man jeweils auf *Weiter* klicken



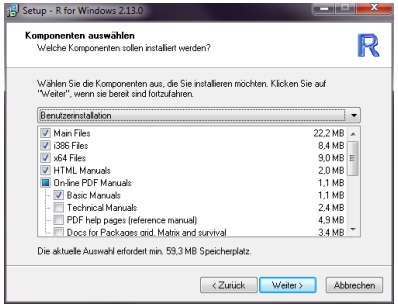
seit Windows Vista: strengere Sicherheitsrichtlinien
R besser nicht unter `C:\Programme\` installieren
(Administratorenrechte).



eine Alternative wäre z.B. `C:\R\` oder `Eigene Dokumente\R\`.
man kann R auch auf einem USB-Stick etc. installieren

es gibt 32- und 64-bit Versionen von R
beide werden in einem Installationprogramm ausgeliefert
über *Benutzerinstallation* werden beide Versionen eingerichtet

vorteilhaft: Basic Manuals, Technical Manuals installieren
eventuell nicht: Message Translations



Aufrufen und Beenden von R

R starten:

- Doppelklick auf das R-Symbol am Desktop
- Im Startmenü unter

  `R` → `R 2.13.0` (32-bit Version)

  `R` → `R x64 2.13.0` (64-bit Version)

R beenden:

- Klicken auf „Schließen"-Kreuz rechts oben
- `quit()` als Befehl eingeben

  in beiden Fällen wird man gefragt: Workspace sichern?

  üblicherweise mit Nein antworten (wird später behandelt)

Installation von Ergänzungen (Contributed Packages)

es gibt viele Erweiterungen für spezielle Methoden
diese Pakete kann man aus sog. Repositories installieren:

im R-Menü *Pakete* klicken auf → *Installiere Paket(e)...*

- auswählen eines nahen Servers (CRAN Mirror)
- auswählen aus Liste mit allen verfügbaren Packages
- nach OK wird das gewählte Package installiert

Info über wichtige Packages aus verschiedenen Bereichen:
auf CRAN (`http://cran.r-project.org/`) links unter *Task Views*

Aller Anfang ist leicht

nach Start von R:

- es öffnet sich das R-Fenster
- oben gibt R allgemeine Information aus
- darunter ist die Einfabeaufforderung: `>`
  hier gibt man R-Befehle ein
- darunter kommt der Output des R-Befehls

Beispiel:
13 eingeben und auf Eingabe drücken → R gibt wieder 13 aus.

```
> 13
[1] 13
```

Tipps:
durch alte Befehle navigieren mit Pfeiltasten ↑ bzw. ↓
Unterbrechen von Befehlen oder laufenden Berechnungen: Esc

Einfaches Rechnen

in R kann man die Grundrechenarten folgendermaßen anwenden:

Addieren

```
> 1 + 2
[1] 3
```

Subtrahieren

```
> 1 - 2
[1] -1
```

Multiplizieren

```
> 4 * 5
[1] 20
```

Dividieren

```
> 4/5
[1] 0.8
```

Potenzieren

```
> 4^2
[1] 16
```

es stehen noch viele weitere Funktionen zu Verfügung wie z.B.

Exponentialfunktion
```
> exp(1)
[1] 2.72
```
Logarithmus
```
> log(1)
[1] 0
```
trigonometrische Funktionen
```
> cos(0)
[1] 1
```

Funktionen können auch geschachtelt werden
```
> log(exp(0))
[1] 0
> exp(cos(log(1)))
[1] 2.72
```


Variablen

Werte können Variablen zugewiesen werden
dies geschieht mit <- (Kleiner-Zeichen und Bindestrich)
```
> x <- 25
```
die Zahl 25 wird in die Variable x gespeichert
man sagt: engl. *„x gets 25“*, deutsch: *„x wird 25“*, *„x ist 25“*
(der Wert wird bei einer Zuweisung nicht ausgegeben)

gibt man nun wiederum x ein, liefert R den Inhalt der Variable
```
> x
[1] 25
```

man kann nahezu beliebige Namen verwenden (Details später):
es ginge also auch
```
> hansi <- 25
> hansi
```


Gespeicherte Variablen kann man bei neuen Zuweisungen verwenden:
```
> y <- x + 1
> y
[1] 26
> x * y
[1] 650
> z <- x/y
> z
[1] 0.962
```

Will man beides auf einmal:
Zuweisung *und* die Ausgabe des Werts → alles einklammern:
```
> (z <- x/y)
[1] 0.962
```


Vektoren

- Vektoren: mehrere Werte zusammengefasst in einer Variablen
- können mit Funktion c() (combine) erzeugt werden
- Argumente von c() werden durch Beistriche getrennt

ein *numerischer* Vektor wäre beispielsweise
```
> ALTER <- c(21, 24, 28)
> ALTER
[1] 21 24 28
```

ein sog. *character* Vektor wäre
```
> Geschlecht <- c("männlich", "weiblich")
> Geschlecht
[1] "männlich" "weiblich"
```

Text muss man unter Anführungszeichen setzen
(doppelt oder einfach, nur nicht mischen!)
auch Zahlen (z.B. "15") in Anführungszeichen werden als Text behandelt!

Vektoren kann man wieder kombinieren und erweitern

```
> ALTER <- c(22, ALTER, 39)
> ALTER
[1] 22 21 24 28 39
```

einige Funktionen für Vektoren sind

Anzahl der Elemente (Länge des Vektors): `length()`

```
> length(ALTER)
[1] 5
```

Ausgabe Elemente in umgekehrter Reihenfolge: `rev()`

```
> rev(ALTER)
[1] 39 28 24 21 22
```

Mittelwert: `mean()`

```
> mean(ALTER)
[1] 26.8
```

Standardabweichung: `sd()`

```
> sd(ALTER)
[1] 7.33
```


Auswählen von Elementen (Indizieren)

Angabe der Indizes in eckigen Klammern [ ]
z.B. das 2. Element in `ALTER`

```
> ALTER[2]
[1] 21
```

das 1., 2. und 4. Element von `ALTER`

```
> ALTER[c(1, 2, 4)]
[1] 22 21 28
```

die Werte von Position 1 bis 3

```
> ALTER[1:3]
[1] 22 21 24
```

der Doppelpunkt bedeutet `von:bis`

```
> (x <- 1:50)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[41] 41 42 43 44 45 46 47 48 49 50

> (x <- 10:1)
 [1] 10  9  8  7  6  5  4  3  2  1
```


Indizierung mit Variablen

```
> u <- 3:5
> ALTER[u]
[1] 24 28 39
```

man kann mittels Indizes gezielte Änderungen vornehmen

```
> ALTER
[1] 22 21 24 28 39
```

der dritte Wert wird durch 42 ersetzt

```
> ALTER[3] <- 42
> ALTER
[1] 22 21 42 28 39
```

mit negativen Indizes bestimmte Werte ausschließen

```
> ALTER[-3]
[1] 22 21 28 39
> ALTER[-u]
[1] 22 21
> ALTER[-(2:4)]
[1] 22 39
```


Matrizen

rechteckige Datenstruktur mit Zeilen und Spalten

wir definieren noch zwei Vektoren
diese sollen zusammen mit `ALTER` eine Matrix bilden

```
> GEWICHT <- c(56, 63, 80, 49, 75)
> GRÖSSE <- c(1.64, 1.73, 1.85, 1.6, 1.81)
```

mit `cbind()` kann man mehrere Vektoren spaltenweise zu einer Matrix zusammenfassen (das ‚c' in `cbind` steht für *columns*)

```
> X <- cbind(ALTER, GEWICHT, GRÖSSE)
> X
     ALTER GEWICHT GRÖSSE
[1,]    22      56   1.64
[2,]    21      63   1.73
[3,]    42      80   1.85
[4,]    28      49   1.60
[5,]    39      75   1.81
```

zeilenweises Zusammenfassen mit `rbind()` (‚r' wie *rows*)

```
> rbind(ALTER, GEWICHT, GRÖSSE)
         [,1]  [,2]  [,3] [,4]  [,5]
ALTER   22.00 21.00 42.00 28.0 39.00
GEWICHT 56.00 63.00 80.00 49.0 75.00
GRÖSSE   1.64  1.73  1.85  1.6  1.81
```

Indizierung von Matrizen auch mit eckigen Klammern:
`[Zeilen(n) , Spalte(n)]`

```
> X[2, 3]
GRÖSSE
  1.73
> X[u, 1:2]
     ALTER GEWICHT
[1,]    42      80
[2,]    28      49
[3,]    39      75
> X[3, ]
  ALTER GEWICHT  GRÖSSE
  42.00   80.00    1.85
```


Zeilen-/Spaltennamen
kann man mit `rownames()` bzw. `colnames()` abfragen und setzen

```
> colnames(X)
[1] "ALTER"   "GEWICHT" "GRÖSSE"

> namen <- c("Gerda", "Karin", "Hans", "Doris", "Ludwig")
> rownames(X) <- namen
> X
       ALTER GEWICHT GRÖSSE
Gerda     22      56   1.64
Karin     21      63   1.73
Hans      42      80   1.85
Doris     28      49   1.60
Ludwig    39      75   1.81
```

Spaltennamen schon vorhanden:
Matrix `X` wurde mit `cbind()` erzeugt
`cbind()` und `rbind()` nehmen Variablennamen mit

Indizierung kann auch über die Namen erfolgen

```
> X["Doris", "GEWICHT"]
[1] 49
> X["Doris", ]
  ALTER GEWICHT  GRÖSSE
   28.0    49.0     1.6
```

Nützliche Funktionen für Matrizen:

- Dimension einer Matrix (Anzahl Zeilen, Anzahl Spalten)

```
> dim(X)
[1] 5 3
```

- Anzahl Spalten

```
> ncol(X)
[1] 3
```

- Anzahl Zeilen

```
> nrow(X)
[1] 5
```

- Gesamtanzahl der Elemente in `X` (Zeilen × Spalten)

```
> length(X)
[1] 15
```


Data Frames

Matrizen können nur Werte eines Datentyps enthalten
z.B. nur *numerisch* oder nur *character*

Data Frames können unterschiedliche Datentypen enthalten

zur Erzeugung verwendet man die Funktion `data.frame()`
funktioniert ähnlich wie `cbind()`

```
> SEX <- c("weiblich", "weiblich", "männlich", "weiblich", "männlich")
> gewichtsdaten <- data.frame(SEX, X)
> gewichtsdaten
            SEX ALTER GEWICHT GRÖSSE
Gerda  weiblich    22      56   1.64
Karin  weiblich    21      63   1.73
Hans   männlich    42      80   1.85
Doris  weiblich    28      49   1.60
Ludwig männlich    39      75   1.81
```

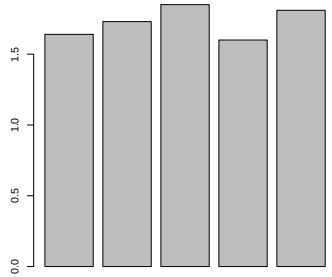
die Indizierung funktioniert wie bei Matrizen:

```
> gewichtsdaten["Doris", ]
          SEX ALTER GEWICHT GRÖSSE
Doris weiblich    28      49    1.6

> gewichtsdaten[2, 3]
[1] 63

> gewichtsdaten[, 4]
[1] 1.64 1.73 1.85 1.60 1.81
```


Einfache Grafiken (später viel mehr)

Balkendiagramm (einen Balken für jeden Wert)

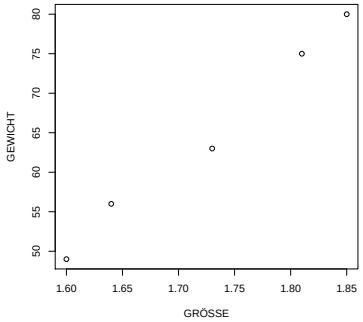
```
> barplot(GRÖSSE)
```



Streudiagramm (Scatterplot)

`plot(x-Werte, y-Werte)`

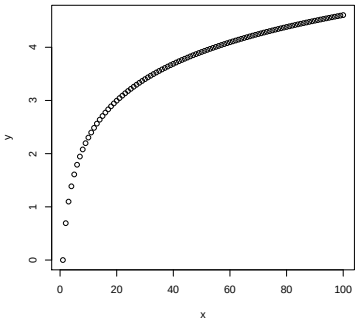
```
> plot(GRÖSSE, GEWICHT)
```



mit plot() kann man sich leicht Funktionen visualisieren

z.B. `log()`

```
> x <- 1:100
> y <- log(x)
> plot(x, y)
```



Übungen

1. Weisen Sie einer Variable `x` den Wert 15 zu und erstellen Sie einen Vektor `y` mit den Werten {1, 2, 3, 10, 100}. Multiplizieren Sie diese miteinander und speichern Sie das Ergebnis in einem neuen Objekt `z`. Bilden Sie anschließend die Summe aller Elemente von `z`.
2. Erzeugen Sie eine Sequenz von 0 bis 10 und eine Sequenz von 5 bis –5.
3. Erzeugen Sie eine Sequenz von –3 bis +3 in 0.1-Schritten. (Tipp: Erzeugen Sie die Sequenz von –30 bis 30 und dividieren Sie durch 10.)
   Zeichnen Sie die Funktion $y = x^2$, wobei Sie für x die erzeugte Sequenz verwenden. Wie sieht die Funktion für $y = 2+x^2$ bzw. $y = 5 - x^2$ aus?

Übungen

4. Definieren Sie zwei Vektoren mit folgenden Daten:
   `t` enthält {mo, di, mi, do, fr, sa} und
   `m` enthält {90, 80, 50, 20, 5, 50}.
   Verbinden Sie beide Vektoren spaltenweise zu einer Matrix mit 5 Zeilen und 2 Spalten und speichern Sie diese im Objekt `studie` ab. Vergeben Sie anschließend die Spaltennamen `Wochentag` für `t` und `Motivation` für `m`.
   Fügen Sie nun am unteren Ende der Matrix eine Zeile mit den Elementen {so, 100} hinzu und überschreiben Sie mit dem Ergebnis das Objekt `studie`.
5. Gehen Sie genauso vor wie im vorigen Beispiel, aber erzeugen Sie einen Data Frame (den Sie `studie2` nennen) anstelle einer Matrix.

Teil 2: Daten in R

Vom Fragebogen zum fertigen Datensatz

Fragebogen und Kodierung

Beispiel: Ausschnitt

ID :_____

Fragebogen: (bitte ankreuzen)

mein Geschlecht?

weiblich	männlich

mein liebstes Alter wäre ?

egal	jünger	älter	genau so wie jetzt es ist	gleich wie jetzt oder jünger	gleich wie jetzt oder älter

meine Größe ? _____ cm

mein Alter ? _____ Monate

bei einem ersten Date bevorzuge ich ?

romantischer Spaziergang	Kino	Abendessen	Disco	spontane Entscheidung	Video ausleihen	auf der Couch knuddeln

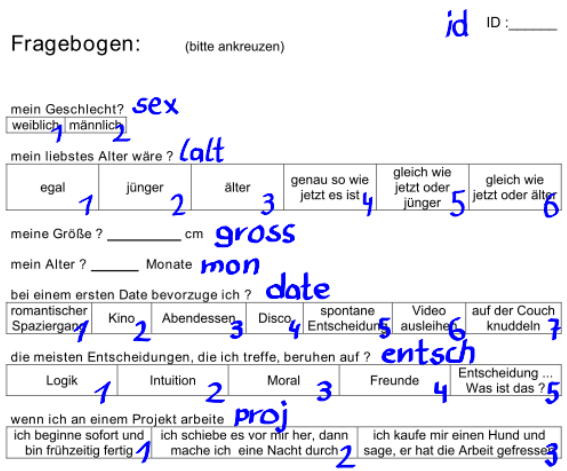
die meisten Entscheidungen, die ich treffe, beruhen auf ?

Logik	Intuition	Moral	Freunde	Entscheidung ... Was ist das ?

wenn ich an einem Projekt arbeite

ich beginne sofort und bin frühzeitig fertig	ich schiebe es vor mir her, dann mache ich eine Nacht durch	ich kaufe mir einen Hund und sage, er hat die Arbeit gefressen

mit Codes versehen – Codebook



Fragebogen: (bitte ankreuzen)

id ID :_____

mein Geschlecht? sex
weiblich 1 | männlich 2

mein liebstes Alter wäre ? lalt

egal	jünger	älter	genau so wie jetzt es ist	gleich wie jetzt oder jünger	gleich wie jetzt oder älter
1	2	3	4	5	6

meine Größe ? _____ cm gross

mein Alter ? _____ Monate mon

bei einem ersten Date bevorzuge ich ? date

romantischer Spaziergang	Kino	Abendessen	Disco	spontane Entscheidung	Video ausleihen	auf der Couch knuddeln
1	2	3	4	5	6	7

die meisten Entscheidungen, die ich treffe, beruhen auf ? entsch

Logik	Intuition	Moral	Freunde	Entscheidung ... Was ist das ?
1	2	3	4	5

wenn ich an einem Projekt arbeite proj

ich beginne sofort und bin frühzeitig fertig	ich schiebe es vor mir her, dann mache ich eine Nacht durch	ich kaufe mir einen Hund und sage, er hat die Arbeit gefressen
1	2	3

beim Kodieren:

- für jede Frage einen Kurznamen
  - beliebige Buchstaben und Zahlen (mit Buchstaben beginnen)
  - Klein- und Großbuchstaben werden unterschieden
  - die Zeichen . und _ sind erlaubt
  - Leerzeichen dürfen nicht verwendet werden
  - Funktionsnamen, wie z.B. `length`, `mean`, oder `log` besser vermeiden
- für kategoriale Variablen:
  - fortlaufende Zahlen beginnend mit 1 verwenden
  - es gehen auch strings (zu Beginn eher vermeiden)

Tipps beim Kodieren:

- möglichst kurze, einprägsame Namen verwenden
- keine Umgestaltungen, besonders keine „händischen Berechnungen“ oder Umkodierungen
- am besten alles so definieren und die Daten dann so eingeben wie der Fragebogen aussieht

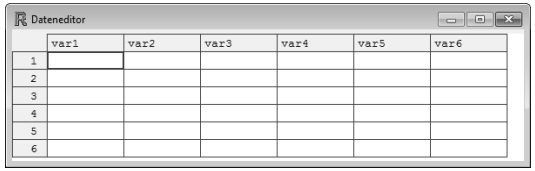
Erfassen der kodierten Daten

verschiedene Möglichkeiten:

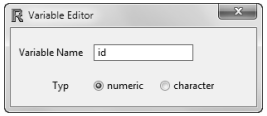
- direkt in R mit Hilfe des Dateneditors
- verwenden externer Programme
  - Excel, andere Spreadsheet-Programme
  - SPSS, Stata, ... (package `foreign`)
  - Texteditor (Edit, Notepad, ...)
- bei kleinen Datenmengen
  (nur wenige Beobachtungen und wenige Variablen):
  - direktes Verwenden von R-Befehlen
    z.B. `x <- c(1, 2, 3)`

Eingabe der Daten (mit dem R-Dateneditor)

```
> fragebogen <- edit(data.frame())
```

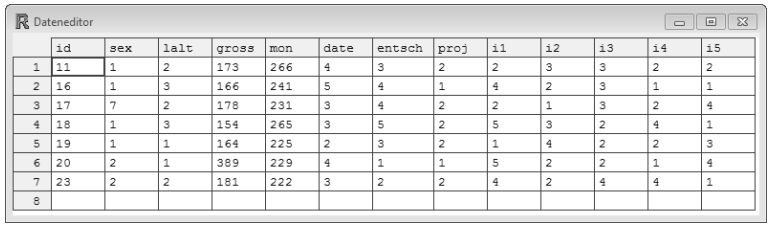


ändern des Variablennamens/Typs durch Klicken auf Spaltenkopf



Einfügen der Daten

- mit Pfeiltasten/Maus navigieren
- wird ein Feld ausgelassen: fehlender Wert (NA)
- hat man Zeile vergessen, kann man sie (meist) unten anhängen



	id	sex	lalt	gross	mon	date	entsch	proj	i1	i2	i3	i4	i5
1	11	1	2	173	266	4	3	2	2	3	3	2	2
2	16	1	3	166	241	5	4	1	4	2	3	1	1
3	17	7	2	178	231	3	4	2	2	1	3	2	4
4	18	1	3	154	265	3	5	2	5	3	2	4	1
5	19	1	1	164	225	2	3	2	1	4	2	2	3
6	20	2	1	389	229	4	1	1	5	2	2	1	4
7	23	2	2	181	222	3	2	2	4	2	4	4	1
8													

Exkurs: fehlende Werte – NA (not available)

es treten immer wieder fehlende Werte auf
radikale Möglichkeit: alle entfernen – Informationsverlust

fehlende Werte in R mit NA kennzeichnen
R nimmt bei statistischen Prozeduren darauf Rücksicht

Achtung bei Anwendung irgendwelcher Berechnungen:
Generell gilt, dass jede Operation, in der ein NA vorkommt, auch ein NA produziert, z.B.

- berechnen eines Mittelwerts für Variable mit NA ergibt NA
- bei Addition von Vektoren werden im Resultat alle jene Elemente NA sein, die in den Ausgangsvektoren auch NA sind.

bei manchen Funktionen kann man NAs mit `na.rm = TRUE` ausschalten:
z.B. `mean(x, na.rm = TRUE)`

Fertigstellen der Datenerfassung

durch Schließen des Dateneditors werden die Daten in das Ergebnis gespeichert

Aufruf war:

```
> fragebogen <- edit(data.frame())
```

Daten sind jetzt in `fragebogen`

mit `head(fragebogen)` kann man sich die ersten Zeilen ansehen

```
> head(fragebogen)
  id sex lalt gross mon date entsch proj i1 i2 i3 i4 i5
1 11   1    2   173 266    4      3    2  2  3  3  2  2
2 16   1    3   166 241    5      4    1  4  2  3  1  1
3 17   7    2   178 231    3      4    2  2  1  3  2  4
4 18   1    3   154 265    3      5    2  5  3  2  4  1
5 19   1    1   164 225    2      3    2  1  4  2  2  3
6 20   2    1   389 229    4      1    1  5  2  2  1  4
```

kleine Korrekturen

in Zeile 3 war hat `sex` den Wert 7
muss korrigiert werden (Nachsehen in Rohdaten) oder `NA`

schon vorhandenen Data Frame kann man mit `fix()` bearbeiten

```
> fix(fragebogen)
```

ist eine Abkürung für `fragebogen <- edit(fragebogen)`

wichtig:
die Daten nach der Eingabe kontrollieren und Fehler ausbessern!
(dazu später)

Abspeichern und Wiedereinlesen der Daten

speichern der Daten in `fragebogen` am besten als komprimierte R-Datei mittels `save()`

```
> save(fragebogen, file = "fragebogen.RData")
```

später wieder in R einlesen mit der Funktion `load()`

```
> load("fragebogen.RData")
```

Hinweise:

- verwenden Sie am besten den gleichen Namen für das zu speichernde Objekt und das komprimierte Datenfile (`fragebogen` – `fragebogen.Rdata`)
- speichern und wieder einlesen beliebiger Objekte mit `save()`

  ```
  > save(a, b, c, file = "objekte_a_b_c.Rdata")
  ```

  und später

  ```
  > load("objekte_a_b_c.Rdata")
  ```


Organisation eines Datensatzes — Data Frames

Data Frame:

- typische Datenstruktur für statistische Analysen in R
- Viele Methoden und Funktionen erwarten einen Data Frame als Input
- besteht üblicherweise aus mehreren Variablen (meist metrische und/oder kategoriale)
- matrixähnliche Struktur:
  – Variablen in Spalten, Beobachtungseinheiten in Zeilen
  – rechteckige Struktur (alle Variablen sind gleich lang)

Das Ansprechen einzelner Variablen eines Data Frame

die ersten fünf Werte der Variable `i1` aus dem `fragebogen`-Datensatz bekommen wir mit

- Indizieren mit *Zeilen/Spalten-Nummern*:

  ```
  > fragebogen[1:5, 9]
  [1] 2 4 2 5 1
  ```

- oder mit *Namen*

  ```
  > fragebogen[1:5, "i1"]
  [1] 2 4 2 5 1
  ```

- weitere Möglichkeit bei Data Frames ist das `$`-*Zeichen*:
  Ansprechen eines Vektors im Data Frame mit seinem Namen

  ```
  > fragebogen$i1[1:5]
  [1] 2 4 2 5 1
  ```

direktes Ansprechen von Variablen eines Data Frames

```
> i1[1:5]
```

funktioniert nicht, aber nach `attach()`

```
> attach(fragebogen)
> i1[1:5]
[1] 2 4 2 5 1
```

wenn man den direkten Zugriff nicht mehr benötigt, sollte man diesen wieder mittels `detach()` aufheben

```
> detach(fragebogen)
```

Achtung:

- Änderungen an `attach()`-ten Variablen, werden nicht im Data Frame gespeichert (sind nach `detach()` verloren)
- permanente Änderungen nur durch
  Ansprechen der Variable im Data Frame
  `dataframe$variable <- ...` oder
  `dataframe[,"variable"] <- ...`

Faktoren

- Faktor ist ein spezieller Datentyp in R
- wird für kategoriale Variablen verwendet
- viele R-Funktionen beruhen darauf
- beim Einlesen von Textvektoren aus externen Dateien diese standardmäßig in Faktoren umgewandelt werden

auch Zahlen können als Kategorienwerte verwendet werden z.B. kategoriale Daten, Kategorien sind die Zahlen 1 bis 3

```
> x <- c(1, 2, 3, 1, 1, 2, 2)
> x
[1] 1 2 3 1 1 2 2
```

definieren eines Faktors mittels `factor()`

```
> x_fact <- factor(x)
> x_fact
[1] 1 2 3 1 1 2 2
Levels: 1 2 3
```


Angabe von Labels (bei der Definition von Faktoren)
z.B. Kategorien 1 bis 3 – Labels `"A"`, `"B"` und `"C"`)

```
> x_fact <- factor(x, labels = c("A", "B", "C"))
> x_fact
[1] A B C A A B B
Levels: A B C
```

Änderung von Labels (bei bestehenden Faktoren):
zwei Möglichkeiten

- mit `factor()`

```
> x_fact1 <- factor(x_fact, labels = c("AA", "BB", "CC"))
> x_fact1
[1] AA BB CC AA AA BB BB
Levels: AA BB CC
```

- oder besser mittels Funktion `levels()`

```
> x_fact2 <- x_fact
> levels(x_fact2) <- c("AA", "BB", "CC")
> x_fact2
[1] AA BB CC AA AA BB BB
Levels: AA BB CC
```


Abfragen von Labels: auch mittels Funktion `levels()`

```
> levels(x_fact2)
[1] "AA" "BB" "CC"
```

`levels()` sowohl zum Abfragen als auch Setzen von Labels

—

wir müssen noch die kategorialen Variablen im Data Frame `fragebogen` in Faktoren umwandeln

```
> fragebogen$sex <- factor(fragebogen$sex, labels = c("w", "m"))
> head(fragebogen$sex)
[1] w w m w w m
Levels: w m
```

Achtung:

- auf Reihenfolge der Kategorien:
  hier *weiblich* (1) und *männlich* (2)
- permanente Änderungen finden nur im Data Frame statt:
  Verwenden von `$` oder andere Möglichkeiten (s.o.)

Auswählen von Beobachtungseinheiten (Fällen)

manchmal möchte man Analyse auf Teilstichprobe beschränken

Beispiel:
Datensatz nur für Personen, die jünger als 20 Jahre sind

mittels Funktion `subset()`

```
> fragebogenJ <- subset(fragebogen, mon < 240)
> head(fragebogenJ, 4)
  id sex lalt gross mon date entsch proj i1 i2 i3 i4 i5
3 17   m    2   178 231    3      4    2  2  1  3  2  4
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
7 23   m    2   181 222    3      2    2  4  2  4  4  1
```

in `fragebogenJ` sind alle Zeilen, in denen der Wert von `mon` kleiner als 240 ist

- `mon < 240` ist eine logische Bedingung, die wahr (`TRUE`) oder falsch (`FALSE`) sein kann
- R kopiert nur jene Fälle für die der logische Ausdruck `TRUE` ist

logische Operatoren

`<`	kleiner als	`>`	größer als
`<=`	kleiner oder gleich	`>=`	größer oder gleich
`==`	(exakt) gleich	`!=`	(exakt) ungleich
`&`	UND (Durchschnitt)	`|`	ODER (Vereinigung)
`!`	NICHT (Negation)		

man kann logische Bedingungen auch verknüpfen:
mit den Operatoren `&` (UND) bzw. `|` (ODER)

Beispiel: nur jüngere und größere Personen
Auswahlbedingung ist dann: `(mon < 240) & (gross >= 180)`

Achtung:
- `&` (UND) bedeutet Durchschnitt (sowohl als auch)
- `|` (ODER) bedeutet Vereinigung (nicht exklusiv)
  entweder das eine oder das andere oder beides

bei metrischen Variablen: vor allem >, >=, <, <=
bei kategorialen Variablen: vor allem == (gleich) und != (ungleich)

Beispiel: Datensatz nur mit Frauen (`sex == "w"`)

```
> fragebogenW <- subset(fragebogen, sex == "w")
> head(fragebogenW, 3)
  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
4 18   w    3   154 265    3      5    2  5  3  2  4  1
```

Achtung bei Verwendung von == bei numerischen Variablen:

```
> log(3)
[1] 1.1
> 1.098612 == log(3)
[1] FALSE
```

das passiert, weil der Rechner intern die Zahlen genauer darstellt
z.B. kann `log(3)` intern 1.09861228866811 sein

logische Funktionen für fehlende Werte

- `na.omit()`
  man will den ganzen Datensatz ohne fehlende Werte

```
> fragebogen_ohneNA <- na.omit(fragebogen)
> dim(fragebogen_ohneNA)
[1] 97 13
> dim(fragebogen)
[1] 100  13
```

- `is.na()`
  Ausschließen fehlenden Werte nur in einzelnen Variablen
  jedes Element wird geprüft ob es NA ist (`x == NA` geht nicht)
  Resultat: `TRUE`, wenn Wert `NA`, `FALSE` wenn nicht `NA`

```
> x <- c(1, 2, NA, NA, 3)
> is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
```

  Beispiel: alle Fälle bei denen `sex` nicht (`"!"`) `NA` ist

```
> subset(fragebogen, !is.na(sex))
```

mit subset() zusätzlich Variablen auswählen: Option `select=`

nur die Variablen `date` und `entsch` und diese nur für Männer

```
> fragebogen_DE_M <- subset(fragebogen, sex == "m", select = c(date,
+     entsch))
> head(fragebogen_DE_M, 3)
  date entsch
3    3      4
6    4      1
7    3      2
```

die Option `select=` ist relativ flexibel

- negative Auswahl, wie beim Indizieren:
  alle Variablen außer `sex` und `mon`: `select = -c(sex, mon)`
- ausnahmsweise auch Bereiche (`":"`) bei Variablennamen:
  Beispiel: Variablen `sex` und `i1` bis `i5`

```
> subset(fragebogen, select = c(sex, i1:i5))
```


Transformieren der Daten: Erzeugen neuer Variablen

wenn man Variablen analysieren will, die sich aus vorhandenen ableiten lassen

zur Erleichterung der folgenden Berechnungen

```
> attach(fragebogen)
```

Beispiel:
Alter (in Monaten erfasst) in Jahre umrechnen

```
> alter <- mon/12
> alter[1:6]
[1] 22.2 20.1 19.2 22.1 18.8 19.1
```

Allgemein: Berechnung von Variablen:

```
variable <- numerischer Ausdruck
```


Numerische Audrücke
sind aus einem oder mehreren der der folgenden Elemente zusammengesetzt

- Bereits definierte Variablen
- Zahlen
- Addition (+)
- Subtraktion (-)
- Multiplikation (*)
- Division (/)
- Potenzfunktion (^ oder **)
- runde Klammern
- Funktionen

„Punkt- vor Strichrechnung“
empfehlenswert immer geeignet Klammern zu setzen
(besser zu viele als zu wenige)

Numerische Funktionen
in numerischen Ausdrücken zahlreiche Funktionen verwendbar (auch geschachtelt)

Beispiele:

- `abs()` Absolutwert (Betrag)
- `exp()` Exponentialfunktion
- `log()` Natürlicher Logarithmus
- `round()` auf eine ganze Zahl gerundeter Wert
- `trunc()` auf eine ganze Zahl abgeschnittener Wert
- `sqrt()` Quadratwurzel
- `min()`, `max()` Minimum, Maximum
- `sum()` Summe
- ...

Beispiel: Alter ganzzahlig gerundet

```
> alter <- round(alter)
```

Beispiel: Erstellung eines Stress-Index (Summenscore)

in `fragebogen`:
die letzten fünf Fragen (i1 bis i5) sollen Stress messen:
wie sehr jemand unter Stress leidet und damit umgeht

Leben unter Zeitdruck Wettbewerb Mangel an Erfolgserlebnissen	vs.	keine Getriebenheit easy going entspannt

übliche Vorgangsweise: Werte zusammenzählen → Index
geht hier nicht wegen Polung der Items: z.B.
Frage 1: „bei Terminen . . . nie zu spät" – Stress
Frage 2: „ich kann gut zuhören" – kein Stress

wenn hohe Werte viel Stress bedeuten müssen Frage 1 und 3 umgepolt werden:
$1 \to 5$, $2 \to 4$, . . . $5 \to 1$ (1 bedeutet „trifft nicht zu")

Umpolung allgemein:

für Werte von $1,\ldots,k$

```
neuer (umgepolter) Wert = (k + 1) - alter Wert
```

für Werte von $0,\ldots,m$

```
neuer (umgepolter) Wert = m - alter Wert
```

für unsere Daten:

```
> i1u <- 6 - i1
> i3u <- 6 - i3
```

den Summenscore `stress` erhalten wir mit

```
> stress <- i1u + i2 + i3u + i4 + i5
> stress[1:10]
 [1] 14  9 14 13 18 12 11 14 11 15
```

neue Variable `stress` ist Zusammenfassung der fünf Items
beschreibt Tendenz einer Person, unter Stress zu stehen

Umkodieren von Variablen

Werte einer Variable ändern
Werte gruppieren bzw. umgruppieren

meistens: Kategorien neu definieren oder Werte zusammenfassen

in R viele Möglichkeiten zur Umkodierung von Variablen:
drei wichtige sind:

- `ifelse()`
- `recode()`
- `cut()`

Die Funktion `ifelse()`

einfachste Möglichkeit: Verwendung logischer Ausdrücke

Beispiel:

Vektor `x`, der aus den Zahlen 1 bis 6 besteht
für alle Werte kleiner als 4 → `"A"`, für die anderen `"B"`

```
> x <- 1:6
> x
[1] 1 2 3 4 5 6
> ifelse(x < 4, "A", "B")
[1] "A" "A" "A" "B" "B" "B"
```

bei `ifelse()`:
jeder Wert von `x` wird geprüft, ob er die Bedingung `x < 4` erfüllt
– bei Ergebnis `TRUE` ist Resultat das zweite Argument, also `"A"`
– bei Ergebnis `FALSE` ist Resultat das dritte Argument, also `"B"`

`ifelse()` kann auch geschachtelt werden

Beispiel:

für 1 und 2 → `"A"`, für 3 bis 5 → `"B"`, für 6 → `"C"`

```
> ifelse(x <= 2, "A", ifelse(x <= 5, "B", "C"))
[1] "A" "A" "B" "B" "B" "C"
```

im Prinzip passiert das Gleiche wie oben, nur durchlaufen jetzt die Werte, bei denen die erste Bedingung (`x <= 2`) `FALSE` ist, eine weitere Prüfung

Die Funktion `recode()`

aus dem R-Package `car`, mit dem Befehl `library()` laden

```
> library("car")
```

Beispiel:

Variable `sex` in `fragebogen` ist mit `"m"` und `"w"` kodiert
stattdessen die Werte `"männlich"` und `"weiblich"`

```
> sex.neu <- recode(sex,
+       recodes = '"m"="männlich" ; "w"="weiblich"')
```

erstes Argument: Variable die umkodiert werden soll (`sex`)
zweites Argument: Option `recodes=`

- Zeichenkette für Umkodierung: `recodes = '...'`
- `...` spezifiziert Umkodierung getrennt durch Strichpunkt
- zwei verschiedene Anführungszeichen nötig
  es ginge auch `recodes = "'m'='männlich' ; 'w'='weiblich'"`

Möglichkeiten für `recode()`

Umkodierungsanweisungen `recodes=` sind in einer Zeichenkette bestehen aus einem oder mehreren Bereich/Wert-Paaren, getrennt durch `;`

vier Möglichkeiten:

- einzelne Werte, z.B. `1 = "männlich"`
- multiple Werte, z.B. `c(1,2,4) = 1`
- Bereiche, z.B. `3:4 = "mittel"`
  Bei Bereichen kann man auch die Werte `lo` bzw. `hi` angeben, um den niedrigsten bzw. höchsten Wert einer Variable berücksichtigen zu können, z.B. `lo:3 = 1`
- die Spezifikation `else`, um Werte, die sonst nicht in einem anderen Bereich/Wert-Paar spezifiziert werden, repräsentieren zu können, z.B. `else = "sonstige"`

Beispiel:
Variable `gross` umkodieren:
Werte unter 160 → 1, 160-174 → 2, größere → 3

```
> head(gross)
[1] 173 166 178 154 164 389
> gross.3 <- recode(gross,
+     recodes = 'lo:159 = 1 ; 160:174 = 2 ; 175:hi = 3')
> head(gross.3)
[1] 2 2 3 1 2 3
```

Resultat ist metrisch, will man einen Faktor
weiteres Argument in `recode()`: `as.factor.result = TRUE`

```
> gross.3 <- recode(gross,
+     recodes = 'lo:159 = 1 ; 160:174 = 2 ; 175:hi = 3',
+     as.factor.result = TRUE)
> head(gross.3)
[1] 2 2 3 1 2 3
Levels: 1 2 3
```

Die Funktion `cut()`

Rekodieren (Gruppieren) von Werten einer metrische Variable
Resultat ist ein Faktor

Beispiel: Variable `alter`
den größten und kleinsten Wert erhält man mit `range()`

```
> range(alter)
[1] 18 27
```

diesen Bereich in 3 Gruppen aufteilen, dabei Faktor erzeugen

```
> alter.3 <- cut(alter, 3, include.lowest = TRUE)
> head(alter.3)
[1] (21,24] [18,21] [18,21] (21,24] [18,21] [18,21]
Levels: [18,21] (21,24] (24,27]
```

es bedeutet (21,24] größer als 21 und höchstens 24
d.h. [bedeutet >=, (bedeutet >, etc.

auch Vergabe von Labels möglich mit Option `labels=`

```
> alter.3kat <- cut(alter, 3, labels = c("jung", "mittel", "alt"),
+     include.lowest = TRUE)
```

bei `cut()` Intervalle selbst bestimmen – Option `breaks=`

Beispiel: Alter in Monaten (`mon`) in 4 Bereiche
als Grenzen: 240, 264 und 288 (entspricht 20, 22 und 24 Jahre)
wir benötigen noch kleinsten und größten Wert (`min()`, `max()`)

```
> grenzen <- c(min(mon), 240, 264, 298, max(mon))
> grenzen
[1] 219 240 264 298 328
```

neue Variable (mit den 4 Bereichen):

```
> mon.4 <- cut(mon, breaks = grenzen, include.lowest = TRUE)
> head(mon.4)
[1] (264,298] (240,264] [219,240] (264,298] [219,240] [219,240]
Levels: [219,240] (240,264] (264,298] (298,328]
```

Werte in `breaks=` legen die oberen Intervallgrenzen] fest

will man Resultat nicht als Faktor sondern numerisch
mittels Option `labels=FALSE`

```
> mon.4n <- cut(mon, breaks = grenzen, include.lowest = TRUE, labels = FALSE)
> head(mon.4n)
[1] 3 2 1 3 1 1
```

alternativ geht es auch mit der Funktion `as.numeric()`

```
> mon.4n <- as.numeric(mon.4)
> head(mon.4n)
[1] 3 2 1 3 1 1
```


Modifikation eines Data Frame

Hinzufügen und Ersetzen von Variablen in einem Data Frame
mit der Funktion `transform()`

Aufbau eines neuen Data Frame `master` – zunächst kopieren

```
> master <- fragebogen
```

Ersetzen einer Variable:

- `sex.neu` statt `sex`, gleich als Faktor definieren

```
> master <- transform(master, sex = factor(sex.neu))
```

erstes Argument: der zu modifizierende Data Frame
die weiteren Argumente sind Zuweisungen zu Variablen
(nur wird jetzt = statt <- verwendet)

- `gross` ersetzen durch Größe in Meter/Zentimeter

```
> master <- transform(master, gross = round(gross/100, digits = 2))
> head(master$gross)
[1] 1.73 1.66 1.78 1.54 1.64 3.89
```

Ersetzen mehrerer Variablen:

- `lalt` und `date` sind noch numerisch definiert → Faktoren

```
> master <- transform(master, lalt = factor(lalt), date = factor(date))
```

Entfernen von Variablen aus einem Data Frame

- entsprechende Variable muss `NULL` gesetzt werden

```
> master <- transform(master, gross = NULL)
```

- geht alternativ auch in `$` Notation

```
> master$gross <- NULL
```


Inspizieren eines Data Frames – mittels `str()`

```
> str(master)
'data.frame':        100 obs. of  13 variables:
 $ id    : num  11 16 17 18 19 20 23 10 66 51 ...
 $ sex   : Factor w/ 2 levels "männlich","weiblich": 2 2 1 2 2 1 1 2 1 2 ...
 $ lalt  : Factor w/ 6 levels "1","2","3","4",..: 2 3 2 3 1 1 2 6 6 3 ...
 $ gross : num  1.73 1.66 1.78 1.54 1.64 3.89 1.81 1.74 1.83 1.67 ...
 $ mon   : num  266 241 231 265 225 229 222 307 299 232 ...
 $ date  : Factor w/ 7 levels "1","2","3","4",..: 4 5 3 3 2 4 3 4 5 7 ...
 $ entsch: num  3 4 4 5 3 1 2 4 2 4 ...
 $ proj  : int  2 1 2 2 2 1 2 1 1 1 ...
 $ i1    : num  2 4 2 5 1 5 4 2 5 2 ...
 $ i2    : num  3 2 1 3 4 2 2 3 3 2 ...
 $ i3    : num  3 3 3 2 2 2 4 1 4 5 ...
 $ i4    : num  2 1 2 4 2 1 4 1 2 4 ...
 $ i5    : num  2 1 4 1 3 4 1 1 3 4 ...
```

man sieht Namen, Typ (z.B. Factor), ev. levels, die ersten Werte

`str()` geht auch für beliebige R Objekte

Fertigstellen des „Master"-Datensatzes

in der Praxis of langwierige Arbeit von Rohdaten zum analysefertigen Datensatz

wir wollen `fragebogen`-Data Frame fertig aufbereiten → `master`

was fehlt? (hierbei hilft uns Output von `str()`)
- `alter` (in Jahren) ist noch nicht in `master` (1)
- `entsch`, `proj` sind noch keine Faktoren (2)
- es fehlen Labels für `lalt`, `date`, `entsch`, `proj` (3)

(1) `alter` einfügen:

```
> master <- transform(master, alter = alter)
```

was heißt `alter = alter`?
erstes `alter` ist Name der neuen Variable im Data Frame
zweites `alter` ist die schon definierte Variable

(2) `entsch`, `proj` zu Faktoren machen

```
> master <- transform(master, entsch = factor(entsch), proj = factor(proj))
```

(3) Labels für `lalt`, `date`, `entsch`, `proj`

geht auch mit `transform()`, einfacher mit `levels()`

```
> levels(master$lalt) <- c("älter", "egal", "gleich", "gleich/älter",
+     "gleich/jünger", "jünger")
> levels(master$date) <- c("Abendessen", "Couch", "Disco", "Kino",
+     "Spazieren", "spontan", "Video")
> levels(master$entsch) <- c("Freunde", "Intuition", "Logik", "Moral",
+     "Was?")
> levels(master$proj) <- c("frühzeitig", "Nacht", "Hund")
```

Achtung auf Reihenfolge der Kategorien (wie im Fragebogen!)

zum Abschluss speichern wir den adaptierten Datensatz `master`.

```
> save(master, file = "master.RData")
```

`detach(fragebogen)` nicht vergessen!

Datenkontrolle (exemplarisch)

bevor man mit der statistischen Analyse von Daten beginnt, sollte man diese immer zuerst auf Korrektheit überprüfen.

bei kleinen Datensätzen:
- Vergleich mit Rohmaterial (Fragebogen)

bei größeren Datensätzen:
- Statistikprozeduren:
  Häufigkeitstabellen `table()`, `summary()`, ...
- Grafikfunktionen:
  Balkendiagramme `barplot()`, Histogramme `hist()`, ...

bei Datenkontrolle sucht man nach Auffälligkeiten
– bei Fehlern: wenn möglich korrigieren (Rohdaten), sonst `NA`
– bei „seltsamen“ Werten: Plausibilitätschecks

Übersicht mittels `summary()`

hier nur Ausschnitt

```
> summary(master[, c("sex", "gross", "mon", "date", "proj")])
      sex         gross            mon              date            proj
 männlich:43   Min.   :1.54   Min.   :219   Abendessen:15   frühzeitig:49
 weiblich:54   1st Qu.:1.66   1st Qu.:236   Couch     :15   Nacht      :51
 NA's    : 3   Median :1.72   Median :264   Disco     :15   Hund       : 0
               Mean   :1.74   Mean   :266   Kino      :18
               3rd Qu.:1.78   3rd Qu.:294   Spazieren :10
               Max.   :3.89   Max.   :328   spontan   :16
                                            Video     :11
```

`summary()` ergibt bei:
kategorial: Häufigkeiten der einzelnen Kategorien
metrisch: Minimum, Maximum, Quartile, Median, Mittelwert
dazu Anzahl fehlender Werte (`NA's`)

Beispiel für numerische Kontrolle: Tabellieren – `table()`

`table()` zählt, wie oft ein bestimmter Wert vorkommt

Beispiel für `sex`:

```
> table(sex)
sex
 w  m
54 43
```

NAs werden auch dargestellt wenn Option `exclude=NULL`

```
> tab.sex <- table(sex, exclude = NULL)
> tab.sex
sex
   w    m <NA>
  54   43    3
```

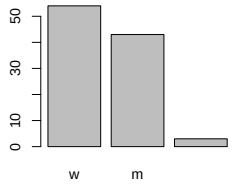
geht auch für mehrere Variablen

```
> table(sex, proj)
   proj
sex  1  2
  w 30 24
  m 17 26
```


Beispiel für grafische Kontrolle: Balkendiagramm – `barplot()`

bei numerischen Variablen:
– `barplot(x)` zeichnet einen Balken für jeden Werte von `x`
bei kategorialen Variablen:
– Werte müssen erst tabelliert werden

```
> barplot(table(sex, exclude = NULL))
```



Auffälligkeiten im `master`-Data Frame

- `NA`'s nur bei `sex`
- Kategorie `Hund` kommt bei `proj` nicht vor
- Maximum bei `gross` ist 3.89
- keine Auffälligkeiten bei den anderen Variablen

zur Variable `gross`:

nachsehen, ob es mehrere Personen mit hohen Werten gibt

```
> which(master$gross > 2)
[1] 6
```

`which()` gibt Positionen in `gross` aus, die Bedingung erfüllen

Abfrage der `id`, damit man in den Rohdaten nachsehen kann

```
> master[master$gross > 2, "id"]
[1] 20
```

gesuchter Fragebogen hat `id` 20, → ausbessern oder NA, z.B.

```
> master[master$gross > 2, "gross"] <- NA
```

nach Abschluß aller Fehlerkorrekturen Abspeichern mit `save()`

Übungen

1. Erstellen Sie einen Data Frame namens `min.dat` und geben Sie Folgendes ein:

a	ges	gr	gew
21	m	181	69
35	w	173	58
829	m	171	75
2	e	166	69

2. Benennen Sie die Variablen in `Geschlecht`, `Alter`, `Grösse` und `Gewicht` um.
3. Erstellen Sie eine Tabelle der Variable `Geschlecht` und prüfen Sie, ob Eingabefehler vorliegen. Der Wert `e` entstand durch einen Tippfehler, wir nehmen an, dass im Fragebogen „weiblich“ stand. Ändern Sie diesen Wert auf `w`.
4. Prüfen Sie die Anzahl der Levels von `Geschlecht` und entfernen Sie Levels, die nicht verwendet werden.

Übungen

5. Benennen Sie die verbleibenden Levels von `Geschlecht` in „weiblich“ für `w` und „männlich“ für `m` um.
6. Kontrollieren Sie das Minimum und Maximum des Alters mit `range()`. Offensichtlich hat es grobe Eingabefehler gegeben, die Sie nun folgendermaßen ersetzen: Erstellen Sie eine Variable `auswahl`, die das Ergebnis der logischen Abfrage für „Alter geringer als 20 oder größer als 80“ enthält. Wir kennen die wahren Werte nicht, also nutzen Sie die Variable `auswahl`, um die Beobachtungen, deren Altersangaben unter 20 oder über 80 liegen, `NA` zu setzen.
7. Verwenden Sie die Funktion `attach()`, um alle Variablen direkt aufrufen zu können.
8. Erstellen Sie ein Balkendiagramm für die Variable `Geschlecht`.

Übungen

9. Berechnen Sie den Body-Mass-Index (BMI) aller Personen nach der Formel BMI $= m/l^2$ (wobei m für das Gewicht in Kilogramm und l für die Größe in Metern steht) und weisen Sie das Ergebnis der Variable `BMI` in Ihrem Data Frame zu. Geben Sie die Variable aus und runden Sie sie dabei auf eine Nachkommastelle.
10. Rekodieren Sie die Variable `Grösse` folgendermaßen: Alle Werte unter 170 nennen Sie `klein` und alle größer oder gleich 170 nennen Sie `gross`. Erstellen Sie hiermit eine neue Variable `Grösse_rec` in Ihrem Data Frame und tabellieren Sie sie.