Inhaltsverzeichnis

1 Datenstrukturen

1.1 Vektoren und Data Frames

Sie haben bereits eine Datenstruktur in Form eines Vektors kennengelernt. Ein Vektor ist nichts anderes als die Zusammenfassung von Zahlen in ein Objekt.

```
vektor <- c(2, 3, 5)
vektor
```

```
[1] 2 3 5
```

Eine *mächtigere* Datenstruktur ist der sogenannte **Data Frame**. Im Gegensatz zum Vektor ist der Data Frame 2-D (zweidimensional); er besteht aus benannten Zeilen und Spalten. Normalerweise tritt ein Datensatz in der Psychologie in einer tabellen-ähnlichen Form auf, wobei unterschiedliche Spalten für unterschiedliche Variablen und unterschiedliche Zeilen für unterschiedliche Personen stehen. Der Data Frame kann diese Art von Daten durch seine Zweidimensionalität genau abbilden.

Beispiel:

```
load("dat/erstis.RData")    # laden von Daten
```

- zuerst laden wir den in `erstis.RData` gespeicherten Data Frame in R

```
str(erstis)
```

- mit dem `str()`-Befehl (kurz für *Structure*) kann man sich die (Daten-)Struktur eines Objekts vollständig angeben lassen

- je nach Größe des Objekts, für das man sich die Struktur ausgeben lassen will, kann das Ergebnis in der Konsole sehr viele Zeilen einnehmen (deshalb werden hier nur Ausschnitte des Outputs gezeigt)

Für Data Frames kann man den `str`-Befehl in zwei Bestandteile unterteilen.

```
'data.frame':   191 obs. of  55 variables:
```

- `data.frame` beschreibt die Datenstruktur des Objekts `erstis`.
- das Objekt `erstis` ist also ein Data Frame mit 191 Beobachtungen (Zeilen) und 55 Variablen (Spalten)

```
$ gruppe     : int  2 3 2 2 4 4 3 2 3 4 ...
$ geschl     : int  NA 2 1 2 NA 1 NA NA NA 2 ...
$ gebjahr    : int  1970 1967 1989 1976 1969 1976
$ alter      : int  38 41 19 32 39 32 NA NA NA 23
$ abi        : int  1992 1987 2007 1993 1987 1995
$ kinder     : int  1 1 2 2 1 1 NA NA NA 2 ...
$ job        : int  2 1 1 2 2 1 NA NA NA 2 ...
$ berlin     : int  1 1 1 1 1 1 NA NA NA 1 ...
$ wohnort.alt: int  2 3 4 3 3 3 NA NA NA 3 ...
$ uni1       : int  1 1 0 0 0 0 NA NA NA 0 ...
```

- die Dollarzeichen leiten eine neue Spalte im Data Frame `erstis` ein, d.h., dass `alter` eine Variable (Spalte) im Data Frame `erstis` darstellt
- es hilft, sich Data Frames als eine Anreihung gleichlanger Vektoren vorzustellen. Jeder dieser Vektoren entspricht genau einer Variable (Spalte)
- die Abkürzung `int` steht für `integer` (Ganze Zahlen) und beschreibt den Datentyp (s.u.) der dazugehörigen Variable
- `alter` ist also eine Variable, die die Datenstruktur Vektor hat und nur ganze Zahlen beinhalten darf
- als letztes werden die ersten 10 Elemente angezeigt

Wenn man auf eine Variable in einem Data Frame zugreifen will, dann kann man das mit dem $-Operator.

```
erstis$alter
```

```
  [1] 38 41 19 32 39 32 NA NA NA 23 25 23 23 20 28 25 25 31 32 23 19 NA 18 30 41
 [26] 43 25 21 20 21 21 36 22 NA 20 18 19 19 19 25 20 30 20 20 19 31 23 25 28 19
 [51] 23 23 21 21 30 27 23 19 18 NA 20 21 23 NA 34 19 21 20 26 23 27 21 26 25 36
 [76] 20 20 44 22 36 19 29 30 24 36 23 NA 23 29 22 NA NA 21 23 20 21 28 22 NA 38
[101] 20 NA 27 19 21 29 19 25 26 23 NA 22 19 20 27 20 29 35 17 21 21 27 24 33 20
[126] 34 39 19 28 21 23 26 44 20 22 22 20 27 21 33 33 20 19 26 32 27 19 25 25 28
[151] 30 19 23 27 22 21 27 27 29 37 55 33 22 NA 22 19 27 37 28 21 39 19 31 24 21
[176] 27 30 21 21 48 20 27 25 29 29 20 29 NA 19 25 30
```

Der $-Operator wird zwischen den Namen des Data Frames und den Variablennamen gesetzt. In allgemeiner Form:

```
dataframe$variable
```

Falls man sich nur für die Datenstruktur eines Objekts interessiert, ohne sich den zusätzlichen Output des `str`-Befehls ausgeben lassen zu müssen, kann man den `class`-Befehl benutzen.

```
class(erstis)
```

```
[1] "data.frame"
```

2 Datentypen

Neben den *Integers* gibt es noch eine Reihe anderer Datentypen:

- Numerische Vektoren (*numeric* oder *integer*) enthalten nur Zahlen
  - *numeric*: i.d.R. Dezimalzahlen/ *integer*: i.d.R. ganze Zahlen

  - In dieser Veranstaltung wird nicht weiter zwischen *numeric* und *integer* unterschieden, wir bezeichnen beide Vektoren als *numeric*
- Zeichenvektoren (*characters*) werden als Text behandelt (Buchstaben, Wörter, Zeichen, auch Zahlen als Text)
- Faktoren (*factors*) werden als kategoriale (nominalskalierte) Variablen behandelt (dazu später mehr)

Um einen Zeichenvektor zu erstellen kann wieder der `c`-Befehl benutzt werden. Dafür muss jedes Element des Vektors bei der Erstellung in Anführungszeichen gesetzt werden:

```
planeten <- c("Merkur", "Venus", "Erde", "Mars", "Jupiter", "Saturn", "Uranus", "Neptun")
planeten
```

```
[1] "Merkur"  "Venus"   "Erde"    "Mars"    "Jupiter" "Saturn"  "Uranus"
[8] "Neptun"
```

```
class(planeten)
```

```
[1] "character"
```

Hier gibt der `class`-Befehl für `planeten` die Datenstruktur `character` aus. Dies ist die englische Bezeichnung für Vektoren, die Zeichen beinhalten.
Auf Vektoren können eine Reihe von Funktionen angewendet werden. Die `length`-Funktion gibt die Anzahl der Elemente eines Vektors wieder.

```
length(planeten)                    # 8 Planeten in unserem Sonnensystem
```

```
[1] 8
```

Vorsicht! Es gibt Funktionen, die nur auf bestimmte Datentypen angewendet werden können.

```
sum(planeten)
```

```
Error in sum(planeten) :  invalid 'type' (character) of argument
```

Hier wird der `sum`-Funktion ein Zeichenvektor übergeben, um die Summe aus allen Elementen des Vektors zu bilden. R gibt einen Fehler aus, da R, logischerweise, nicht weiß wie Zeichen summiert werden.

2.1 Erstellen eines Data Frames

Wie Vektoren kann man auch Data Frames in R selbst erzeugen. Hierfür wird, analog zum `c()`-Befehl, der `data.frame()`-Befehl verwendet.

Beispiel:

```
planeten <- c("Merkur", "Venus", "Erde", "Mars", "Jupiter", "Saturn", "Uranus", "Neptun")
planetentyp <- c("Erdplanet", "Erdplanet", "Erdplanet", "Erdplanet", "Gasplanet",
                 "Gasplanet", "Gasplanet", "Gasplanet")

sonnensystem <- data.frame(planeten, planetentyp)
sonnensystem
```

```
  planeten planetentyp
1   Merkur   Erdplanet
2    Venus   Erdplanet
3     Erde   Erdplanet
4     Mars   Erdplanet
5  Jupiter   Gasplanet
6   Saturn   Gasplanet
7   Uranus   Gasplanet
8   Neptun   Gasplanet
```

- Zuerst wurden zwei Zeichenvektoren erstellt. Dem `data.frame()`-Befehl wurden als Argument die Vektoren `planeten` und `planetentyp` übergeben. Der erstellte Data Frame wird schlussendlich im Objekt `sonnensystem` abgelegt.
- Im Output kann man sehen, dass die Spalten des Data Frames den Vektoren entsprechen, und zwar in der Reihenfolge, in der die Vektoren an den `data.frame`-Befehl übergeben wurden.
  - Grundsätzlich gilt: Alle Spalten müssen in einem Data Frame dieselbe Länge haben!

```
id <- c(1, 2, 3, 4, 5, 6, 7) # Vektor mit sieben Elementen
data.frame(planeten, id)     # Sonnensystem hat acht Planeten! -> FEHLER
```

```
Error in data.frame(planeten, id) :  arguments imply differing number of rows:  8, 7
```

Man kann einen erstellten Data Frame auch durch neue Vektoren (also Spalten) erweitern oder enthaltene Informationen löschen.

Hinzufügen von Vektorem zu einem Data Frame:

```
sonnensystem$monde <- c(0, 0, 1, 2, 69, 62, 27, 14)
sonnensystem
```

```
  planeten planetentyp monde
1   Merkur   Erdplanet     0
2    Venus   Erdplanet     0
3     Erde   Erdplanet     1
4     Mars   Erdplanet     2
5  Jupiter   Gasplanet    69
6   Saturn   Gasplanet    62
7   Uranus   Gasplanet    27
8   Neptun   Gasplanet    14
```

- Ein numerischer Vektor wird erzeugt und mit dem Zuweisungspfeil `<-` im Data Frame `sonnensystem` als neue Spalte `monde` abgespeichert

Löschen von Vektoren (Spalten) in einem Data Frame:

```
sonnensystem$monde <- NULL
sonnensystem
```

```
  planeten planetentyp
1   Merkur   Erdplanet
2    Venus   Erdplanet
3     Erde   Erdplanet
4     Mars   Erdplanet
5  Jupiter   Gasplanet
6   Saturn   Gasplanet
7   Uranus   Gasplanet
8   Neptun   Gasplanet
```

- Dem Vektor `monde` im Data Frame `sonnensystem` wurde der Wert `NULL` zugewiesen. Das führt dazu, dass der Vektor `monde` gelöscht wird.

3 Datenkontrolle

Da auf Basis der Daten Berechnungen angestellt und Schlüsse gezogen werden, ist es sehr wichtig, die Daten auf ihre Richtigkeit zu kontrollieren! Fehler können beispielsweise beim Datenimport ...

- falsches Dezimaltrennzeichen
- fehlende Werte nicht berücksichtigt

... oder bei der Dateneingabe auftreten:

- Werte liegen außerhalb des möglichen Wertebereichs
- Werte liegen innerhalb des möglichen Wertebereichs
  - nur identifizierbar über doppelte Dateneingabe

Um Fehler beim Datenimport identifizieren zu können, kann man den `str`-Befehl nutzen. So kann man beispielsweise überprüfen, ob numerische Variablen auch numerische Vektoren sind oder ob der Datensatz alle Personen abbildet, die man beispielsweise einen Fragebogen ausfüllen lassen hat.
Um Werte außerhalb des zulässigen Wertebereichs zu identifizieren, kann man den `summary`-Befehl verwenden. Mit diesem Befehl lassen sich Verteilungskennwerte über die Variablen des Datensatzes ausgeben.

4 Datenüberblick

Der Unterschied zu `str()` ist, dass der der `summary`-Befehl nicht die Datenstruktur, sondern eine Zusammenfassung des Objektinhalts ausgibt.

Beispiel:

```
summary(erstis)
```

```
      code           gruppe          geschl         gebjahr   
 Min.   :  1.0   Min.   :1.000   Min.   :1.000   Min.   :1953  
 1st Qu.: 48.5   1st Qu.:1.500   1st Qu.:1.000   1st Qu.:1979  
 Median : 96.0   Median :2.000   Median :1.000   Median :1985  
 Mean   : 96.0   Mean   :2.461   Mean   :1.324   Mean   :1982  
 3rd Qu.:143.5   3rd Qu.:3.500   3rd Qu.:2.000   3rd Qu.:1987  
 Max.   :191.0   Max.   :4.000   Max.   :2.000   Max.   :1991  
                                 NA's   :21      NA's   :15    
     kinder           job           berlin       wohnort.alt   
 Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000  
 1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000  
 Median :2.000   Median :1.000   Median :1.000   Median :3.000  
 Mean   :1.851   Mean   :1.497   Mean   :1.126   Mean   :2.556  
 3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:1.000   3rd Qu.:3.000  
 Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :4.000  
 NA's   :17      NA's   :18      NA's   :24      NA's   :29     
```

- die Darstellung unterscheidet sich nicht nur im Inhalt zum `str()`-Befehl, sondern auch im Format der Ausgabe
- die Variablen werden nun im Spaltenformat ausgegeben
  - für jeden numerischen Vektor im Datensatz werden das Minimum, Maximum, das erste und dritte Quartil, sowie der Median und das arithmetische Mittel ausgegeben
  - falls die Variable fehlende Werte beinhaltet, werden die fehlenden Werte für diese Variable gezählt und in `NA` angezeigt

Weitere nützliche Befehle für die erste Erstellung eines Datenüberlicks:

```
nrow(erstis)      # Anzahl der Zeilen
ncol(erstis)      # Anzahl der Spalten
names(erstis)     # Spaltennamen
colnames(erstis)  # Spaltennamen
rownames(erstis)  # Zeilennamen
na.omit(erstis)   # Ausschluss ALLER Personen/Zeilen, die mind. einen fehlenden Wert aufweisen
```

5 Indexierung

R bietet eine Vielzahl an Tools an, um einen Datensatz zu bearbeiten oder auf einen Teil des Datensatzes zuzugreifen.

5.1 Indexierung: Vektoren

Um bestimmte Zellen innerhalb eines Vektors anzusprechen, benötigen wir eine Funktion zur Indexierung. In R werden dafür die eckigen Klammern (`[]`) verwendet. So können einzelne Elemente eines Vektors ganz einfach über die Position innerhalb des Vektors ausgewählt werden.

```
vektor <- c(2, 3, 5)
vektor[2]
```

```
[1] 3
```

Das geht auch mit Zeichenvektoren oder Vektoren mit Wahrheitswerten:

```
character_vector <- c("P", "S", "Y", "C", "H", "O", "L", "O", "G", "I", "E")
character_vector[6]
```

```
[1] "O"
```

```
bool_vector <- c(TRUE, FALSE)
bool_vector[2]
```

```
[1] FALSE
```

5.2 Indexierung: Data Frames

Im Gegensatz zu Vektoren sind Data Frames zweidimensional. Wenn also nur auf ein bestimmtes Element des Datensatzes zugegriffen werden soll, muss eine Zeile **und** eine Spalte spezifiziert werden.

`dataframe[Position: ZEILE , Position: SPALTE]`

Zeilen und Spalten werden durch ein Komma getrennt! (s. auch Abbildung 1)

Beispiel:

```
erstis[2, 4]
```

```
[1] 1967
```

- in der 2. Zeile der 4. Spalte des Data Frames `erstis` ist die Zahl 1967 abgespeichert
  - daraus folgt, dass die 2. Person des Datensatzes im Jahr 1967 geboren wurde, da die 4. Spalte der Variable `gebjahr` entspricht

Es ist auch möglich ganze Zeilen oder Spalten auszugeben.

```
erstis$gebjahr              # Indexierung mit $-Operator (ohne "")
erstis[, "gebjahr"]         # Indexierung mit []; Spalte über Namen spezifiziert (mit "")
```

```
erstis[, 4]                 # Indexierung mit []; Spalte über Position spezifiziert
```

```
  [1] 1970 1967 1989 1976 1969 1976   NA   NA   NA 1985 1983 1985 1985 1988 1980
 [16] 1983 1983 1977 1976 1985 1989   NA 1990 1978 1967 1965 1983 1987 1988 1987
 [31] 1987 1972 1986   NA 1988 1990 1989 1989 1989 1983 1988 1978 1988 1988 1989
 [46] 1977 1985 1983 1980 1989 1985 1985 1987 1987 1978 1981 1985 1989 1990   NA
 [61] 1988 1987 1985   NA 1974 1989 1987 1988 1982 1985 1981 1987 1982 1983 1972
 [76] 1988 1988 1964 1986 1972 1989 1979 1978 1984 1972 1985   NA 1985 1979 1986
 [91]   NA   NA 1987 1985 1988 1987 1980 1986   NA 1970 1988   NA 1981 1989 1987
[106] 1979 1989 1983 1982 1985   NA 1986 1989 1988 1981 1988 1979 1973 1991 1987
[121] 1987 1981 1984 1975 1988 1974 1969 1989 1980 1987 1985 1982 1964 1988 1986
[136] 1986 1988 1981 1987 1975 1975 1988 1989 1982 1976 1981 1989 1983 1983 1980
[151] 1978 1989 1985 1981 1986 1987 1981 1981 1979 1971 1953 1975 1986   NA 1986
[166] 1989 1981 1971 1980 1987 1969 1989 1977 1984 1987 1981 1978 1987 1987 1960
[181] 1988 1981 1983 1979 1979 1988 1979   NA 1989 1983 1978
```

- alle Befehle führen zur Ausgabe der Spalte `gebjahr` inkl. aller dazugehörigen Zeilen
  - das Freilassen des Zeilen- oder Spaltenelements bedeutet, dass alle Zeilen, respektive Spalten, ausgewählt werden

Wenn man sich mehrere Spalten oder Zeilen ausgeben lassen möchte, dann geht das mit der Übergabe von Vektoren:

```
erstis[2, c(2, 3, 4)]
```

```
  gruppe geschl gebjahr
2      3      2    1967
```

- hier wurde die 2. Zeile der 2., 3. und 4. Spalte ausgewählt
  - in diesem Fall ist es möglich `2:4` anstatt `c(2, 3, 4)` zu übergeben, da die ausgewählten Spalten (`2,3,4`) aufeinander folgen
- allerdings ist `c()` notwendig für die Indexierung, falls a) separate Spalten ausgewählt werden sollen oder b) mehrere Spalten über ihre Namen ausgewählt werden sollen:

```
erstis[2, c(2, 4)]
```

```
  gruppe gebjahr
2      3    1967
```

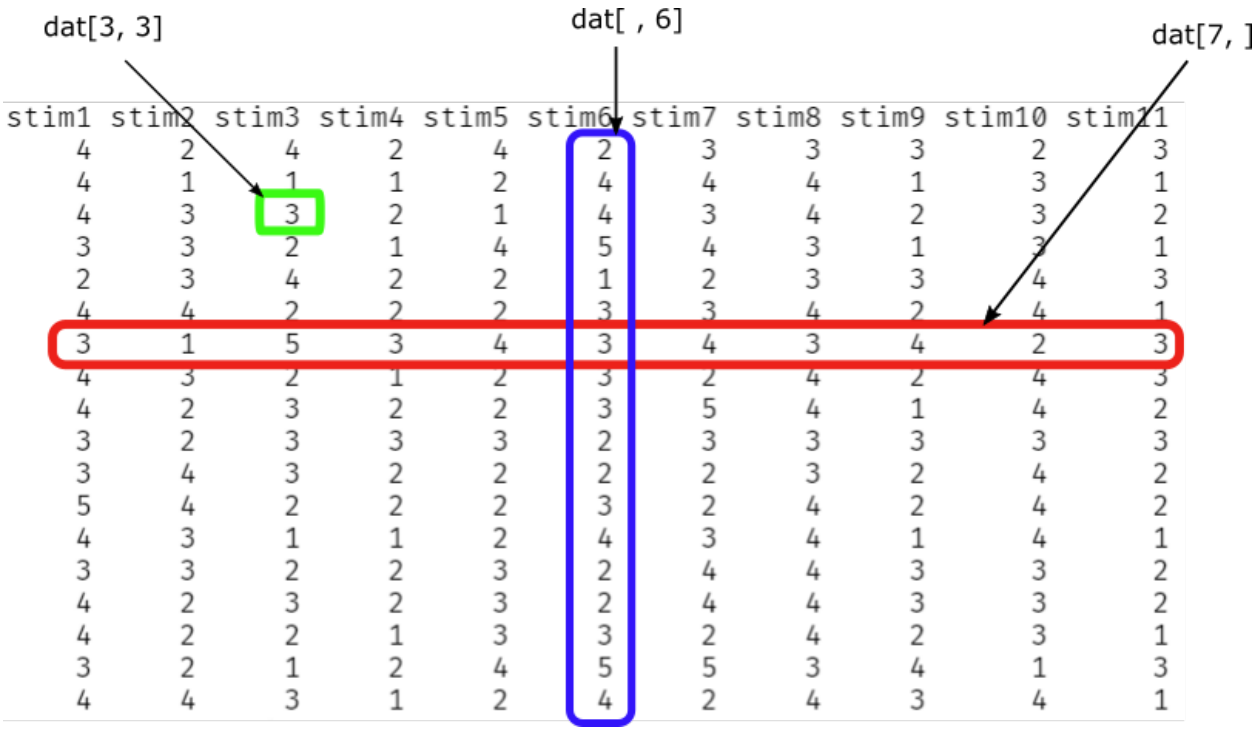

Abbildung 1: Visualisierung der Indexierung eines Dataframes

5.2.1 Indexierung mit Logischen Operatoren

Eine nützliche Form der Indexierung ist die Benutzung von logischen Operatoren.

```
erstis[erstis$alter < 20, "alter"]
```

```
 [1] 19 NA NA NA 19 NA 18 NA 18 19 19 19 19 19 19 18 NA NA 19 19 NA NA NA NA NA
[26] 19 19 NA 19 17 19 19 19 19 NA 19 19 NA 19
```

Es lohnt sich diesen Befehl etwas genauer zu betrachten:

- Mit `erstis$alter` greifen wir auf die Spalte `alter` im Data Frame `erstis` zu
  - Nun wird `erstis$alter` mit dem logischen Operator `<` verknüpft. Wir filtern mit `erstis$alter < 20` alle Werte aus der Spalte `alter`, die kleiner als 20 sind
- Dies wird nun an den Data Frame, in der **Zeilenposition**, übergeben
  - Es sollen also nur die Personenzeilen des Data Frames `erstis` ausgegeben werden, die in der Spalte `alter` Werte kleiner 20 haben
- In der **Spaltenposition** wird die Spalte `alter` spezifiziert
- Der Data Frame wurde also nicht nur auf bestimmte Personen (alter < 20), sondern auch auf die Spalte `alter` gefiltert

Mit logischen Operatoren lassen sich sehr komplexe Filterprozesse konstruieren:

```
erstis[erstis$alter < 20 & !is.na(erstis$alter), "alter"]
```

```
 [1] 19 19 18 18 19 19 19 19 19 19 18 19 19 19 19 19 17 19 19 19 19 19 19 19
```

- Der obige Befehl wurde um `!is.na(erstis$alter)` erweitert
  - `is.na()` schaut sich jeden Wert in der Spalte `alter` an und gibt ein `TRUE` aus, falls der Wert `NA` (Not Available) sein sollte
  - `!` kehrt alle Wahrheitswerte um. `TRUE -> FALSE` oder `FALSE -> TRUE`
- Das `&`-Zeichen verbindet die beiden Operationen.
- Mit dieser Befehlskette werden zusätzlich (zum Filtern der unter 20-jährigen) alle fehlenden Werte (`NA`s) aus dem Datensatz `erstis` gefiltert

6 Faktoren

Variablen mit Ordinalskalenniveau oder höherem Skalenniveau werden in R als Vektoren angelegt. Variablen mit Nominalskalenniveau werden in R hingegen als Faktoren gespeichert. Ein Faktor ist damit der Datentyp einer Variable, deren Ausprägungen diskrete, ungeordnete Kategorien sind. Die Abspeicherung einer Variable im jeweils richtigen Datentyp hat wichtige Konsequenzen für ihre Weiterverarbeitung in R. So können bestimmte Analysen nur für bestimmte Datentypen durchgeführt werden.

Die Umwandlung eines Vektors in einen Faktor bzw. die Erstellung eines Faktors kann man mit der Funktion factor() vornehmen:

```
num <- c(1, 0, 0, 1)
factor_num <- factor(num)

str(factor_num)
```

```
 Factor w/ 2 levels "0","1": 2 1 1 2
```

Mit `str()` lassen wir uns erneut die Struktur des Faktors `factor_num` ausgeben. Der Output gibt uns die Datenstruktur des Objekts `factor_num` aus. Zusätzlich erhalten wir die Anzahl der `levels` des Faktors, also die Anzahl der Kategorien sowie die Kategorienamen. Wie auch bei anderen Datenstrukturen werden nach dem Doppelpunkt die ersten Elemente des Objekts ausgegeben. Vielleicht ist es Ihnen aufgefallen, dass die Elemente nicht den `levels` entsprechen. Das liegt daran, dass R intern die *echten* Werte numerisch abspeichert. Bei Zeichenvektoren kann der Befehl genauso angewendet werden:

```
char <- c("F", "M", "M", "F")
factor_char <- factor(char)

str(factor_char)
```

```
 Factor w/ 2 levels "F","M": 1 2 2 1
```

Der `factor()`-Befehl wandelt also numerische Variablen (Zahlen) und Zeichenvektoren in kategoriale Variablen um. Auch hier werden die Elemente als Zahlen dargestellt! Die wichtigsten Argumente bei Verwendung dieses

Befehls sind:
`factor(x, levels, labels = levels)`
Diese Argumente bedeuten der Reihe nach:

1. `x` gibt den Namen des Objekts an, welcher umgewandelt werden soll.
2. `levels` zeigt an, welche Zahlenwerte die kategoriale Variable grundsätzlich annehmen kann. Wird diese Angabe nicht gemacht, so werden die Zahlenwerte übernommen, welche in den übergebenen Daten vorhanden sind.
3. `labels` ermöglicht die Benennung der Kategorien der kategorialen Variable. Die Benennung entspricht der Reihenfolge der Werte in `levels`. Die Namen werden in einer runden Klammer übergeben, vor welcher ein **c** steht.

Wir betrachten als Beispiel die Variable Geschlecht. Da Geschlecht als dichotome (nominalskalierte) Variable erhoben wurde, würde sich für diese die Umwandlung in einen Faktor anbieten. Durch die Kodierung als Faktor, teilen wir R mit, welche Eigenschaften diese Variable hat und welche Operationen mit dieser Variable sinnvoll sind. In R lassen sich einige Funktionen nur auf Objekte eines bestimmten Datentyps anwenden. Nutzen wir nun den `factor()`-Befehl auf unseren `erstis` Datensatz an:

```
str(erstis$geschl)
```

```
 int [1:191] NA 2 1 2 NA 1 NA NA NA 2 ...
```

- beim Datenimport werden Variablen, welche nur Zahlen enthalten per Voreinstellung als numerische Vektoren abgelegt
- Wir wandeln die Variable `geschl` daher nun in einen Faktor um:

```
erstis$geschl <- factor(erstis$geschl,
                        levels = c(1, 2),
                        labels = c("weiblich", "männlich"))
str(erstis$geschl)
```

```
 Factor w/ 2 levels "weiblich","männlich": NA 2 1 2 NA 1 NA NA NA 2 ...
```

- bei der Umwandlung in einen Faktor ist es wichtig, dass die Reihenfolge der Werte in `levels` zu den Kategorienbezeichnungen in den `labels` passen
  - dafür muss das *Codebook* zurate gezogen werden und unter der Variable *Geschlecht* die Zuweisung der Ausprägungen gelesen werden. Erst dann ist es möglich, die `labels` den `levels` korrekt zuzuordnen
- die neue Variable (die Faktorform von `erstis$geschl`) wurde dem Namen der Ausgangsvariable zugewiesen. Die Ausgangsvariable wurde also überschrieben und ist in R nicht mehr zugänglich

Die Beschriftung der Faktorstufen kann abgefragt werden mit dem `levels()`-Befehl:

```
levels(erstis$geschl)
```

```
[1] "weiblich" "männlich"
```

7 Umkodierung von Variablen

Es gibt mehrere Gründe, warum Variablen umkodiert werden:

- Umpolen von negativ formulierten Items einer Skala
- Zusammenfassen von Kategorien einer Variablen
- Fehlende Werte sind mit einer bestimmten Zahl kodiert und sollen durch NA ersetzt werden. (Falls dies beim Einlesen nicht berücksichtigt wurde.)

Im `erstis` Datensatz wurde die Stimmung der Probanden anhand mehrerer Fragen erhoben (sogenannter *Items*). Um einzelne Variablen (Items einer Skala) umzupolen, muss jede mögliche Ausprägung dieser Variable ausgewählt und dann neu zugewiesen werden. Bei der Veränderung von Werten in einer Variable macht es

Sinn die veränderten Werte in einer neuen Variable abzuspeichern. Diese neue Variable erhält im folgenden die Endung `_r` für rekodiert.

```
erstis$stim4_r[erstis$stim4 == 1] <- 5
erstis$stim4_r[erstis$stim4 == 2] <- 4
erstis$stim4_r[erstis$stim4 == 3] <- 3
erstis$stim4_r[erstis$stim4 == 4] <- 2
erstis$stim4_r[erstis$stim4 == 5] <- 1
```

Einfacher mit `recode()`-Befehl aus dem Paket `car`:

```
library(car)
erstis$stim4_r <- recode(erstis$stim4, recodes = "1=5; 2=4; 3=3; 4=2; 5=1")
erstis$stim11_r <- recode(erstis$stim11, recodes = "1=5; 2=4; 3=3; 4=2; 5=1")
```

- Hier wird mit einer Zeile dasselbe ausgeführt wie oben in fünf Zeilen
  - mit dem `recodes`-Argument wird die Umkodiervorschrift angegeben
  - die Umkodiervorschrift wird in Anführungszeichen eingeschlossen und Wertepaare durch Semikola getrennt

8 Rechnen mit Skalenwerten

Um beispielsweise einen Skalenmittelwert zu berechnen, kann man den `rowMeans()`-Befehl verwenden. Ein Skalenmittelwert wird errechnet, indem man die Werte aller Items einer Skala für eine Person mittelt. Mit dem `rowMeans()`-Befehl lässt sich das ganz einfach für alle Personen im Datensatz durchführen, das heißt zeilenweise.

```
set_gs <- c("stim1", "stim4_r", "stim8", "stim11_r")          # Speichern der Itemnamen
erstis$gs1 <- rowMeans(erstis[, set_gs],
                       na.rm = FALSE)
```

- Wir übergeben der `rowMeans()`-Funktion all diejenigen Spalten des Datensatzes `erstis`, welche die Werte der Items der Skala gute vs. schlechte Stimmung enthalten. Dies sind die Items `stim1` (zufrieden), `stim4` (schlecht), `stim8` (gut) und `stim11` (unwohl). Die Items `stim4` und `stim11` müssen zuvor umgepolt werden (s.o.), damit hohe Werte für alle Items die gleiche Bedeutung haben (hoher Wert = positivere Stimmung).
- Das Objekt `set_gs` enthält die Namen der entsprechenden Variablen
- über `erstis[, set_gs]` übergeben wir der `rowMeans()`-Funktion die Spalten des Datensatzes `erstis` mit diesen vier Variablen
- Die Funktion berechnet dann zeilenweise (d.h., für jede Person) den Mittelwert über die vier Werte der Person auf diesen Items
- Das Ergebnis wird in einer neuen Variablen (neue Spalte) namens `gs1` gespeichert und dem Datensatz hinzugefügt
- Umgang mit fehlenden Werten: bei der Berechnung des Skalenmittelwerts stellt sich die Frage, ob für Personen mit fehlenden Werten ein Skalenmittelwert berechnet werden soll
  - `na.rm = FALSE` impliziert einen strengen Umgang mit fehlenden Werten: Personen, die mindestens einen fehlenden Wert aufweisen (d.h., auf mindestens einer der zu mittelnden Variablen) erhalten keinen Skalenmittelwert, sondern den Wert `NA` auf der neuen Variablen `gs1`.
  - ein liberaler Umgang mit fehlenden Werten ist mit `na.rm = TRUE` möglich. Hier wird jede Person, die mindestens einen gültigen Wert aufweist, mit in die Berechnung aufgenommen. Die Person erhält auf der Variable `gs1` den Mittelwert über alle für sie vorhandenen Werte auf den vier Items.

9 Typische Schritte beim Umgang mit neuen Daten

- Variablennamen festlegen
- Daten eingeben oder einlesen
- Fehlende Werte kodieren
- Kategoriale Variablen kodieren
- Plausibilität prüfen
  - Gibt es Eingabefehler? Werte, die es nicht geben kann?
  - Sind fehlende Werte richtig kodiert?
  - Gibt es unmögliche Werte (z.B., durch Tippfehler)?
- Items invertieren, also Kodierung inverser Fragebogen-Items umkehren
- Neue Variablen berechnen
  - Summenscores/Mittelwerte für Skalen aus mehreren Items
  - Zeitdauer aus Anfangszeit/Endzeit
  - Alter aus Geburtsjahr
  - ...

10 Übersicht

10.1 Neue wichtige Konzepte

- **Datenstrukturen**
- **Datentypen**
- **Datenüberblick**
- **Datenkontrolle**
- **Indexierung**
- **Faktoren**

10.2 Neue wichtige Befehle, Argumente, Operatoren

Funktion	Verwendung
`Datensatz[a, b]`	Auswahl einer oder mehrerer Zeilen (a) und einer oder mehrerer Spalten (b)
`objekt$element`	Auswahl/Ansprache von einem Element in einem Objekt
`objekt$element <- NULL`	Löschen von Element (z.B. einer Variable) in Objekt (z.B. einem Datensatz)
`str(Objekt)`	Abfrage der Objektstruktur
`summary(Objekt)`	Zusammenfassung der Objektinhalte
`length(Objekt)`	Länge eines Objekts (Anzahl der Elemente)
`nrow(Datensatz)`	Anzahl von Personen (Zeilen)
`ncol(Datensatz)`	Anzahl von Variablen (Spalten)
`names(Datensatz)`	Abfrage der Variablennamen
`na.omit(Datensatz)`	Entfernt alle Fälle mit mind. einem fehlendem Wert
`is.na(x)`	Logische Abfrage: Ist der Wert x fehlend?
`factor(x)`	Wandelt x in einen Faktor um
`levels(x)`	Zeigt die Faktorstufen des Faktors x an
`recode(x, "Vorschrift")`	Erstellt eine nach der Umkodiervorschrift veränderte Variante von x
`rowMeans(x)`	Berechnet die Zeilenmittelwerte

11 Appendix

11.1 Funktionen

11.1.1 `lapply`

R stellt eine Reihe an nützlichen Funktionen als Teil des R-Basispakets `base` zur Verfügung. Im folgenden schauen wir uns `lapply` an. Mit `lapply` kann ein Befehl (Funktion), den man an `lapply` übergibt, auf eine beliebige Anzahl an Objekten in nur einer Zeile Code angewendet werden. Das wird am Beispiel der nun bekannten Umkodierung von Items demonstriert:

```
set_2do <- c("stim4", "stim11")
```

- Namen der **zu umkodierenden** Items werden einem Objekt `set_2do` als Zeichenvektor zugewiesen

```
set_new <- c("stim4_r", "stim11_r")
```

- Namen der **umkodierten** Items werden einem Objekt `set_new` als Zeichenvektor zugewiesen

```
erstis[, set_new] <- lapply(erstis[, set_2do],
                            FUN = recode,
                            recodes = "1=5; 2=4; 3=3; 4=2; 5=1")
```

- mit `lapply` werden alle Items des Objekts `set_2do` gleichzeitig und nach **derselben** Umkodiervorschrift umgepolt!
- das erste Argument entspricht dem Objekt, bzw. den Items, die wir umpolen wollen
  - in unserem Fall sind das die Spalten des `erstis`-Datensatzes
  - daher indexieren wir den `erstis`-Datensatz mithilfe dem von uns erstellten Objekt `set_2do`. Denn das enthält die Itemnamen.
- das Argument `FUN` entspricht der Funktion, die wir auf die einzelnen Items anwenden wollen
  - in unserem Fall benutzen die Funktion ‘recode“
- das letzte Argument ist eigentlich ein Argument der Funktion `recode` und nicht der Funktion `lapply`!
- falls also weitere Argumente der Funktion, die wir auf das Objekt anwenden wollen (in unserem Fall `recode`), übergeben werden sollen, müssen diese an `lapply` übergeben werden (nicht an `recode`!). Kodieren Sie die Items einmal mit und einmal ohne `lapply` und vergleich Sie das Ergebnis. Wenn Sie Umkodierung richtig vorgenommen haben, sind die umkodierten Spalten/Items identisch.