Einführung in R

Clemens Brunner

14.-15.2.2019

Daten importieren

Mit R kann man Daten aus vielen unterschiedlichen Formaten importieren, beispielsweise aus Excel-Tabellen oder aus SPSS. Diese Formate sind allerdings proprietär und daher für das Abspeichern eigener Daten unter Umständen weniger gut geeignet. Idealerweise speichert man Daten in einem offenen und möglichst einfach aufgebauten Format ab, welches man mit einer Vielzahl an (frei verfügbaren) Programmen öffnen kann. Ein Beispiel für ein einfaches Format ist eine Textdatei, die man mit jedem beliebigen Texteditor öffnen kann (R-Scripts sind aus diesem Grund ebenfalls einfache Textdateien).

Textdateien

Werte in Textdateien werden sehr häufig mit Kommas voneinander getrennt - so kann man Werte aus verschiedenen Spalten einer Tabelle darstellen. Solche speziell formatierten Textdateien haben sehr häufig die Endung .csv (comma-separated values). Es gibt aber auch andere Möglichkeiten, die Werte voneinander zu trennen, z.B. mit Semikolons (Strichpunkte) oder Tabulatoren. Solche Dateien haben oft die Endungen .dat oder .tsv. Die Endungen sind aber eigentlich gar nicht relevant, denn im Kern handelt es sich bei allen Varianten um normale Textdateien.

Importieren mit R-Bordmitteln (Base R)

Daten, die in solchen Textdateien tabellarisch abgespeichert sind, können in R mit der Funktion `read.table` eingelesen werden. Bei dieser Funktion kann man sehr viele Argumente genau auf die einzulesende Datei abstimmen (z.B. das Trennzeichen, ob es eine Kopfzeile gibt, ob es Kommentare oder fehlende Werte gibt, usw.). Für Daten im CSV-Format (Datenwerte sind durch Kommas getrennt) gibt es eine Wrapper-Funktion `read.csv`, welche sinnvolle Standardwerte für diverse Argumente annimmt. Für Dateien im TSV-Format gibt es die Wrapper-Funktion `read.delim`.

Als Beispiel sehen wir uns eine Textdatei namens `lecturer.dat` an (diese muss sich im Arbeitsverzeichnis befinden). Wenn man nicht weiß, wie die Daten in einer Datei formatiert sind, kann man deren Inhalt mit der Funktion `file.show` anzeigen:

```
file.show("lecturer.dat")
```

Alternativ kann man die Datei auch in einem beliebigen Texteditor öffnen. Man erkennt, dass bei dieser konkreten Datei die einzelnen Spalten nicht durch sichtbare Zeichen wie Kommas oder Semikolons getrennt sind. Offensichtlich sind die Spalten durch Tabulatoren getrennt, denn diese Zeichen sind (so wie auch Leerzeichen) in der Ausgabe nicht sichtbar. Weiters ist eine Kopfzeile mit den Variablennamen vorhanden.

Man kann versuchen, die Funktion `read.table` mit Standardargumenten aufzurufen und dann Argumente so lange anzupassen, bis die Daten korrekt eingelesen werden:

```
read.table("lecturer.dat")
```

```
       V1         V2  V3      V4      V5     V6       V7
1    name birth_date job friends alcohol income neurotic
2     Ben   7/3/1977   1       5      10  20000       10
3  Martin  5/24/1969   1       2      15  40000       17
4    Andy  6/21/1973   1       0      20  35000       14
5    Paul  7/16/1970   1       4       5  22000       13
6  Graham 10/10/1949   1       1      30  50000       21
```

```
7  Carina  11/5/1983   2      10      25   5000        7
8  Karina  10/8/1987   2      12      20    100       13
9     Doug  1/23/1989   2      15      16   3000        9
10    Mark  5/20/1973   2      12      17  10000       14
11     Zoe 11/12/1984   2      17      18     10       13
```

Das Ergebnis sieht gar nicht so falsch aus, nur die erste Zeile, die in der Datei die Variablennamen enthält, wurde fälschlicherweise als Datenzeile erkannt (und die Spaltennamen wurden automatisch auf generische Namen V1, V2, usw. gesetzt). Dies können wir ändern, indem wir das zusätzliche Argument `header=TRUE` setzen:

```
read.table("lecturer.dat", header=TRUE)
```

```
     name birth_date job friends alcohol income neurotic
1     Ben   7/3/1977   1       5      10  20000       10
2  Martin  5/24/1969   1       2      15  40000       17
3    Andy  6/21/1973   1       0      20  35000       14
4    Paul  7/16/1970   1       4       5  22000       13
5  Graham 10/10/1949   1       1      30  50000       21
6  Carina  11/5/1983   2      10      25   5000        7
7  Karina  10/8/1987   2      12      20    100       13
8     Doug  1/23/1989   2      15      16   3000        9
9     Mark  5/20/1973   2      12      17  10000       14
10     Zoe 11/12/1984   2      17      18     10       13
```

Diesmal sieht das entstandene Data Frame korrekt aus - wir können es direkt einer Variable zuweisen, um damit weiterarbeiten zu können:

```
df <- read.table("lecturer.dat", header=TRUE)
```

Für diese Datei hätten wir auch `read.delim` verwenden können, hier wird standardmäßig davon ausgegangen dass eine Kopfzeile in der Datei vorhanden ist:

```
df <- read.delim("lecturer.dat")
```

Sehen wir uns als zweites Beispiel dieselben Daten an, die aber diesmal mit Kommas getrennt in einer .csv-Datei vorliegen. Um diese Datei einzulesen, können wir entweder die generische Funktion `read.table` mit entsprechendem `sep`-Argument verwenden oder die spezialisierte Funktion `read.csv`:

```
df <- read.table("lecturer.csv", header=TRUE, sep=",")
df <- read.csv("lecturer.csv")
```

Ein weiterer wichtiger Parameter solcher Dateien ist das verwendete Dezimaltrennzeichen. In der englischen Schreibweise werden Kommazahlen durch einen Punkt getrennt (z.B. 12.3, 3.1415). In der deutschen Schreibweise wird hingegen ein Komma verwendet (z.B. 12,3; 3,1415). Das Dezimaltrennzeichen kann in der Funktion `read.table` mit dem Argument `dec` festgelegt werden. Es ist standardmäßig auf einen Punkt gesetzt, ebenso bei der Funktion `read.csv`. Sollten die Zahlen in der Datei jedoch in der deutschen Schreibweise vorliegen, können die Spalten nicht auch durch Kommas getrennt sein - hier werden die Spalten dann oft durch Semikolons getrennt. Für deutsche Zahlenformate setzt man daher die Argumente `sep=";"` und `dec=","` bzw. verwendet die spezialisierte Funktion `read.csv2`.

Wichtig: Unabhängig davon, welche Dezimaltrennzeichen in den Dateien verwendet werden, verwendet R *immer* einen *Punkt* als Dezimaltrennzeichen. Sobald die Daten korrekt eingelesen wurden und als Data Frame vorliegen, muss man sich an die englische Schreibweise halten.

Importieren mit dem `readr`-Paket

Das Tidyverse beinhaltet das Paket `readr`, welches Textdateien einlesen kann. Dies funktioniert oft besser und schneller als mit den Funktionen aus Base R. Die mit dem `readr`-Paket eingelesenen Daten stehen

außerdem als Tibble statt als Data Frame zur Verfügung.

Die Hauptfunktion zum Einlesen von generischen Textdateien mit `readr` ist `read_delim`. Für CSV-Dateien gibt es den spezialisierten Wrapper `read_csv`, für mit Tabulatoren getrennte Daten heißt die Funktion `read_tsv` (bzw. `read_table`). Die letztgenannten Funktionen rufen aber lediglich `read_delim` mit geeigneten Argumenten auf.

Die vorigen Beispieldateien liest man mit `readr`-Funktionen wie folgt ein:

```
library(readr)
df <- read_delim("lecturer.dat", "\t")  # \t ist das Tabulator-Zeichen
df <- read_tsv("lecturer.dat")

df <- read_delim("lecturer.csv", ",")
df <- read_csv("lecturer.csv")
```

Die folgende Tabelle fasst die verschiedenen Funktionen aus Base R bzw. dem `readr`-Paket übersichtlich zusammen.

Dateiformat	Trennzeichen	Base R	`readr`
Generisch	verschieden	`read.table`	`read_delim`
CSV	`,`	`read.csv`	`read_csv`
TSV/DAT	Tabulator (`\t`)	`read.delim`	`read_tsv` bzw. `read_table`

Daten aus SPSS

Sollen bereits vorhandene SPSS-Daten (.sav) importiert werden, kann man dazu die Funktion `read_sav` aus dem Paket `haven` verwenden. Das Ergebnis ist ein Tibble (da `haven` Teil des Tidyverse ist). Das `haven`-Paket kann übrigens auch Daten aus SAS und Stata importieren.

```
library(haven)
df <- read_sav("lecturer.sav")
```

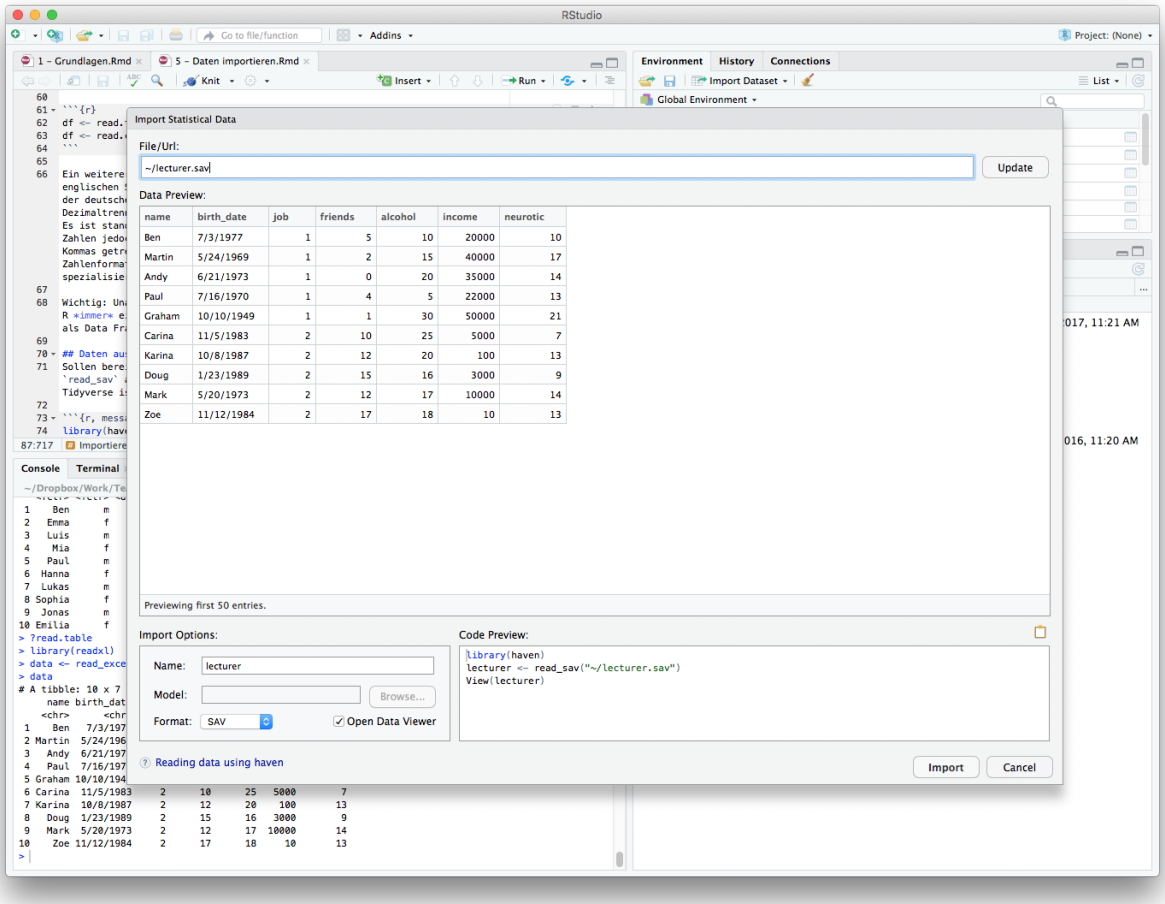
Daten aus Excel

Wenn Daten in einer Excel-Datei (Endung `.xlsx` oder `.xls`) vorliegen, verwendet man zum Einlesen die Funktion `read_excel` aus dem Paket `readxl`. Da auch dieses Paket Teil des Tidyverse ist, bekommt man auch hier ein Tibble zurückgeliefert.

```
library(readxl)
data <- read_excel("lecturer.xlsx")
```

Importieren mit RStudio

Mit RStudio kann man Daten in vielen Formaten auch mit Hilfe der GUI importieren. Praktischerweise bekommt man immer den dazugehörigen R-Code mitgeliefert, welcher die Daten korrekt importiert - diesen Code kann man dann in eigenen Scripts verwenden. Dazu klickt man in der *Files*-Ansicht rechts unten auf die gewünschte Datei und wählt *Import Dataset* aus. Alternativ kann man in der *Environment*-Ansicht rechts oben auf *Import Dataset* gefolgt vom gewünschten Format klicken. Es öffnet sich ein neues Fenster, in dem man eine Vorschau der zu importierenden Datei sieht. Es gibt hier auch die Möglichkeit, Optionen zu ändern - wie sich das auf die eingelesenen Daten auswirkt ist in der Vorschau unmittelbar ersichtlich. Der zugehörige Code befindet sich dann rechts unten. Im folgenden Bild sieht man diesen Dialog beispielhaft für eine zu importierende SPSS-Datei.



Daten aufbereiten

In welchem Dateiformat die Daten auch immer vorhanden sind, schlussendlich landen sie in einem Data Frame (oder Tibble), welches wir dann in R weiterverwenden können. In den allermeisten Fällen wird man das Data Frame noch aufbereiten müssen, damit jede Spalte auch wirklich im gewünschten Format vorliegt. Sehen wir uns das anhand unserer Beispieldaten an, welche wir (wie oben gezeigt) in ein Data Frame einlesen:

```
df <- read.csv("lecturer.csv")
df
```

```
     name birth_date job friends alcohol income neurotic
1     Ben   7/3/1977   1       5      10  20000       10
2  Martin  5/24/1969   1       2      15  40000       17
3    Andy  6/21/1973   1       0      20  35000       14
4    Paul  7/16/1970   1       4       5  22000       13
5  Graham 10/10/1949   1       1      30  50000       21
6  Carina  11/5/1983   2      10      25   5000        7
7  Karina  10/8/1987   2      12      20    100       13
8    Doug  1/23/1989   2      15      16   3000        9
9    Mark  5/20/1973   2      12      17  10000       14
10    Zoe 11/12/1984   2      17      18     10       13
```

Die Funktion `str` zeigt die Struktur eines Objektes an. Im Falle eines Data Frames werden unter anderem die Datentypen jeder Spalte angezeigt.

```
str(df)
```

```
'data.frame':   10 obs. of  7 variables:
 $ name      : Factor w/ 10 levels "Andy","Ben","Carina",..: 2 8 1 9 5 3 6 4 7 10
 $ birth_date: Factor w/ 10 levels "1/23/1989","10/10/1949",..: 10 7 8 9 2 5 3 1 6 4
 $ job       : int  1 1 1 1 1 2 2 2 2 2
 $ friends   : int  5 2 0 4 1 10 12 15 12 17
 $ alcohol   : int  10 15 20 5 30 25 20 16 17 18
 $ income    : int  20000 40000 35000 22000 50000 5000 100 3000 10000 10
 $ neurotic  : int  10 17 14 13 21 7 13 9 14 13
```

Man erkennt, dass die numerischen Werte korrekt als Ganzzahlen erkannt wurden (Typ `int` in den Spalten 3 bis 7). Allerdings besitzt die Spalte 3 (`job`) lediglich zwei Werte, d.h. hier wäre eine kategorische Variable (ein Faktor) besser geeignet als eine numerische. Die ersten beiden Spalten wurden als Faktoren erkannt. Dies ist in unserem Beispiel nicht unbedingt gewünscht, und daher müssen wir diese beiden Spalten in passende Typen umwandeln.

Bei Verwendung der Tidyverse-Funktionen hat die `name`-Spalte bereits den passenden Typ `character`, aber auch hier sollten die Spalten `birth_date` und `job` in geeignetere Typen umgewandelt werden:

```
read_csv("lecturer.csv")
```

```
# A tibble: 10 x 7
   name   birth_date   job friends alcohol income neurotic
   <chr>  <chr>      <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
 1 Ben    7/3/1977       1       5      10  20000       10
 2 Martin 5/24/1969      1       2      15  40000       17
 3 Andy   6/21/1973      1       0      20  35000       14
 4 Paul   7/16/1970      1       4       5  22000       13
 5 Graham 10/10/1949     1       1      30  50000       21
 6 Carina 11/5/1983      2      10      25   5000        7
 7 Karina 10/8/1987      2      12      20    100       13
 8 Doug   1/23/1989      2      15      16   3000        9
 9 Mark   5/20/1973      2      12      17  10000       14
10 Zoe    11/12/1984     2      17      18     10       13
```

Faktoren

Kategorische Variablen, d.h. Variablen die nur eine bestimmte Anzahl an Werten annehmen können, werden in R mit dem Typ `factor` dargestellt. Die Funktion `factor` kann einen entsprechenden Datentyp erzeugen. Standardmäßig werden nicht geordnete (also nominale) Faktoren erzeugt. Setzt man das Argument `ordered=TRUE`, kann man einen geordneten Faktor (also eine ordinale Variable) erzeugen.

Im Beispiel ist die Spalte `job` vom Typ `int`, sollte aber als Faktor interpretiert werden. Die Spalte kann entsprechend neu erstellt und überschrieben werden:

```
df$job <- factor(df$job, levels=c(1, 2), labels=c("Lecturer", "Student"))
df
```

```
     name birth_date      job friends alcohol income neurotic
1     Ben   7/3/1977 Lecturer       5      10  20000       10
2  Martin  5/24/1969 Lecturer       2      15  40000       17
3    Andy  6/21/1973 Lecturer       0      20  35000       14
4    Paul  7/16/1970 Lecturer       4       5  22000       13
5  Graham 10/10/1949 Lecturer       1      30  50000       21
6  Carina  11/5/1983  Student      10      25   5000        7
7  Karina  10/8/1987  Student      12      20    100       13
```

```
8    Doug  1/23/1989  Student      15      16   3000        9
9    Mark  5/20/1973  Student      12      17  10000       14
10    Zoe 11/12/1984  Student      17      18     10       13
```

Datumsangaben

Für Datumsangaben gibt es in R einen eigenen Datentyp, der das Rechnen mit diesen Werten erleichtert. Die Funktion `as.Date` wandelt eine Datumsangabe in Textform in diesen speziellen Typ um. Anzugeben ist hier insbesondere das Argument `format`, welches das Format der vorliegenden Datumswerte spezifiziert.

```
df$birth_date <- as.Date(df$birth_date, format="%m/%d/%Y")
df
```

```
     name birth_date      job friends alcohol income neurotic
1     Ben 1977-07-03 Lecturer       5      10  20000       10
2  Martin 1969-05-24 Lecturer       2      15  40000       17
3    Andy 1973-06-21 Lecturer       0      20  35000       14
4    Paul 1970-07-16 Lecturer       4       5  22000       13
5  Graham 1949-10-10 Lecturer       1      30  50000       21
6  Carina 1983-11-05  Student      10      25   5000        7
7  Karina 1987-10-08  Student      12      20    100       13
8    Doug 1989-01-23  Student      15      16   3000        9
9    Mark 1973-05-20  Student      12      17  10000       14
10    Zoe 1984-11-12  Student      17      18     10       13
```

In diesem Beispiel bedeutet das Argument `%m/%d/%Y`, dass die Werte in der Reihenfolge Monat/Tag/Jahr vorliegen und mit einem Schrägstrich getrennt sind (`%Y` steht hier für eine vierstellige Jahreszahl - Details dazu sind dem Hilfetext von `as.Date` zu entnehmen).

Zeichenketten

Die Spalte `name` wird beim Einlesen mit Base-R-Funktionen als Faktor interpretiert. Die Funktion `as.character` erzeugt daraus einen Character-Vektor.

```
df$name <- as.character(df$name)
```

Nach diesen Konvertierungsoperationen sieht die Datenstruktur dann wie folgt aus:

```
df
```

```
     name birth_date      job friends alcohol income neurotic
1     Ben 1977-07-03 Lecturer       5      10  20000       10
2  Martin 1969-05-24 Lecturer       2      15  40000       17
3    Andy 1973-06-21 Lecturer       0      20  35000       14
4    Paul 1970-07-16 Lecturer       4       5  22000       13
5  Graham 1949-10-10 Lecturer       1      30  50000       21
6  Carina 1983-11-05  Student      10      25   5000        7
7  Karina 1987-10-08  Student      12      20    100       13
8    Doug 1989-01-23  Student      15      16   3000        9
9    Mark 1973-05-20  Student      12      17  10000       14
10    Zoe 1984-11-12  Student      17      18     10       13
```

```
str(df)
```

```
'data.frame':   10 obs. of  7 variables:
 $ name      : chr  "Ben" "Martin" "Andy" "Paul" ...
 $ birth_date: Date, format: "1977-07-03" "1969-05-24" "1973-06-21" ...
 $ job       : Factor w/ 2 levels "Lecturer","Student": 1 1 1 1 1 2 2 2 2 2
 $ friends   : int  5 2 0 4 1 10 12 15 12 17
```

```
 $ alcohol   : int  10 15 20 5 30 25 20 16 17 18
 $ income    : int  20000 40000 35000 22000 50000 5000 100 3000 10000 10
 $ neurotic  : int  10 17 14 13 21 7 13 9 14 13
```

Bei Tibbles kann man analog vorgehen um einzelne Spalten in geeignetere Typen zu konvertieren.

Daten speichern

Wenn man ein Data Frame oder Tibble in einer Datei abspeichern möchte, geht das am einfachsten mit der Funktion `write.table` (bzw. `write.csv`) und den gewünschten Parametern. Dies funktioniert prinzipiell ganz analog zu den oben beschriebenen Lesefunktionen. Hier sollte man allerdings das Argument `row.names=FALSE` setzen, da ansonsten die Zeilennummern in die Datei geschrieben werden, was meist nicht erwünscht ist. Die relevanten Tidyverse-Funktionen lauten `write_delim`, `write_csv` bzw. `write_tsv` (hier werden standardmäßig keine Zeilennummern geschrieben).

Daten umformen (Wide und Long)

Bei Daten im Wide-Format gibt es für jede Variable eine eigene Spalte. Daten im Long-Format haben nur eine Spalte mit allen Werten und eine oder mehrere Spalte(n) mit Indikator-Variablen, welche den Kontext der Werte definieren. Die folgende Tabelle zeigt Beispieldaten im Wide-Format:

Person	Age	Weight	Height
Bob	32	98	188
Al	24	61	176
Sue	64	87	174

Man sieht, dass es drei Wertespalten (`Age`, `Weight` und `Height`) gibt, sowie eine Spalte, welche die Person identifiziert. Dieselben Daten sehen im Long-Format so aus:

Person	Variable	Value
Bob	Age	32
Bob	Weight	98
Bob	Height	188
Al	Age	24
Al	Weight	61
Al	Height	176
Sue	Age	64
Sue	Weight	87
Sue	Height	174

In R kann man zwischen den beiden Formaten hin- und herwechseln, d.h. wenn die Daten in einem Format vorliegen, kann man relativ einfach das andere Format produzieren. Die Daten von oben können wir in R wie folgt im Wide-Format eingeben:

```
library(tibble)
df <- tibble(Person=c("Bob", "Al", "Sue"), Age=c(32, 24, 64), Weight=c(98, 61, 87), Height=c(188, 176, 
df
```

```
# A tibble: 3 x 4
  Person   Age Weight Height
  <chr>  <dbl>  <dbl>  <dbl>
1 Bob       32     98    188
2 Al        24     61    176
```

```
3 Sue       64     87    174
```

Das Umformen von Daten nimmt einen großen Stellenwert im Tidyverse ein, daher gibt es auch eine Menge an nützlichen Funktionen um Daten ins gewünschte Format zu bringen. Das Paket `tidyr` beinhaltet die Funktionen `gather` und `spread`, welche Data Frames von wide nach long bzw. von long nach wide umwandeln können. Das Beispiel von vorher könnte man mit diesen beiden Funktionen wie folgt schreiben:

```
library(tidyr)
(df_long <- gather(df, variable, value, -Person))
```

```
# A tibble: 9 x 3
  Person variable value
  <chr>  <chr>    <dbl>
1 Bob    Age         32
2 Al     Age         24
3 Sue    Age         64
4 Bob    Weight      98
5 Al     Weight      61
6 Sue    Weight      87
7 Bob    Height     188
8 Al     Height     176
9 Sue    Height     174
```

Hier übergibt man zuerst die Daten im Wide-Format, gefolgt vom gewünschten Namen der Key-Spalte (Indikatorspalte) und vom gewünschten Namen der Value-Spalte (Wertespalte). Danach gibt man alle Spalten an, die man zusammenfassen möchte (in unserem Beispiel wählen wir alle Spalten bis auf die `Person`-Spalte aus).

Der umgekehrte Weg wird mit `spread` beschritten; diese Funktion kann eine Spalte auf mehrere Spalten aufteilen:

```
(df_wide <- spread(df_long, variable, value))
```

```
# A tibble: 3 x 4
  Person   Age Height Weight
  <chr>  <dbl>  <dbl>  <dbl>
1 Al        24    176     61
2 Bob       32    188     98
3 Sue       64    174     87
```

Hier gibt man zunächst die Daten im Long-Format an, gefolgt vom Spaltennamen, welcher die Namen der neuen Spalten beinhaltet (Indikatorspalte). Schließlich gibt man als drittes Argument noch den Namen der Wertespalte an.

Übungen

Übung 1

Sehen Sie sich die Hilfe zur Funktion `read.table` an. Welches Argument setzt das Dezimaltrennzeichen? Welches Argument enthält das Trennzeichen der Spalten?

Beantworten Sie diese Fragen auch für die Funktion `read_delim` aus dem `readr`-Paket!

Übung 2

Laden Sie die Datei `wahl16.csv` von hier herunter:

http://bit.ly/r_example_data

In dieser Datei befinden sich die Ergebnisse der Bundespräsidentenwahl 2016 (und zwar nach dem ersten Wahlgang mit den sechs ursprünglichen Kandidaten). Laden Sie diese Daten in ein Data Frame oder Tibble namens `wahl16`und berechnen Sie die relative Gesamtanzahl an Stimmen für alle Kandidaten (die Funktionen `colSums`, `rowSums` sowie `sum` könnten dabei hilfreich sein).

Übung 3

Das UCI Machine Learning Repository ist eine der bekanntesten Websites, welche Datensätze zur freien Verwendung zur Verfügung stellt. Laden Sie für diese Übung den Datensatz Individual Household Electric Power Consumption als .zip-Datei herunter und entpacken Sie die Datei im Arbeitsverzeichnis (wenn Sie nachfolgend die Funktion `read_delim` aus dem Paket `readr` verwenden, müssen Sie die Datei nicht entpacken sondern können direkt die .zip-Datei laden).

Diese Datei beinhaltet die minütliche elektrische Leistungsaufnahme eines Haushalts in einem Zeitraum von fast vier Jahren. Insgesamt gibt es über 2 Millionen Messpunkte und 9 Variablen, welche durch Strichpunkte voneinander `;` getrennt sind. Fehlende Werte sind mit einem Fragezeichen `?` codiert.

Führen Sie folgende Schritte durch:

- Lesen Sie die Daten in ein Data Frame oder Tibble namens `df` ein.
- Sehen Sie sich die ersten paar Zeilen und die Struktur der Daten an.
- Wie viele Zeilen und Spalten hat das Data Frame?
- Wie viele fehlende Werte gibt es insgesamt? Wie viele fehlende Werte gibt es pro Spalte?
- Welche Spalten könnte man eventuell noch in einen passenderen Typ umwandeln (Sie müssen das aber nicht tatsächlich durchführen)?

Hinweis: Es ist wichtig, dass R beim Einlesen der Daten fehlende Werte korrekt erkennt. In dieser Datei werden fehlende Werte durch `?` markiert. Stellen Sie durch Setzen des entsprechenden Argumentes der Funktion `read.table` bzw. `read_delim` sicher, dass diese fehlenden Werte richtig eingelesen werden (und somit korrekt als `NA` interpretiert werden).