Einführung in R

Clemens Brunner

18.-19.10.2019

Daten einlesen

Mit R kann man Daten in 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 (bzw. Spalten) 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.

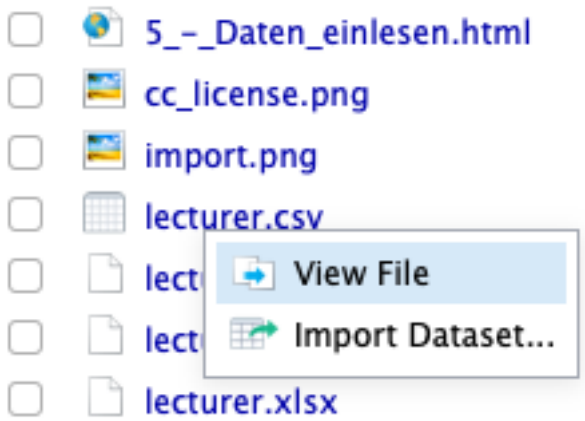
Das Tidyverse beinhaltet das Paket `readr`, welches Textdateien einlesen kann. Dies funktioniert oft besser und schneller als mit den Funktionen aus Base R. Die mit dem `readr`-Paket eingelesenen Daten stehen außerdem als Tibble zur Verfügung.

Die Hauptfunktion zum Einlesen von generischen Textdateien mit `readr` ist `read_delim`. Bei dieser Funktion kann man sehr viele Argumente genau auf die einzulesende Datei abstimmen (z.B. das Trennzeichen, ob es eine Kopfzeile mit den Spaltennamen gibt, ob es Kommentare oder fehlende Werte gibt, usw.). Für Daten im CSV-Format (Datenwerte sind durch Kommas getrennt) gibt es die Wrapper-Funktion `read_csv`, welche sinnvolle Standardwerte für diverse Argumente annimmt. Für Daten die mit Tabulatoren getrennt sind gibt es die Wrapper-Funktion `read_tsv`.

Als Beispiel sehen wir uns eine Textdatei namens `lecturer.csv` 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` im RStudio-Editor öffnen:

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

Alternativ kann man die Datei auch öffnen, in dem man in der "Files"-Ansicht in RStudio auf die Datei klickt und "View File" auswählt.



Man erkennt, dass bei dieser konkreten Datei die einzelnen Spalten durch Kommas getrennt sind. Außerdem ist eine Kopfzeile mit den Variablennamen vorhanden.

Man kann nun versuchen, die Funktion `read_delim` mit dem Argument `delim=","` aufzurufen:

```
library(readr)
read_delim("lecturer.csv", delim=",")
```

```
# 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
```

Das Ergebnis (ein Tibble) sieht korrekt aus - das Tibble hat 10 Zeilen und 7 Spalten mit sinnvollen Datentypen in allen Spalten. Wir können es direkt einer Variable zuweisen, um damit weiterarbeiten zu können:

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

Für diese Datei hätten wir auch `read_csv` verwenden können, hier wird standardmäßig ein Komma als Spaltentrennzeichen angenommen (man spart sich also die Angabe dieses Argumentes):

```
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
```

Sehen wir uns als zweites Beispiel dieselben Daten an, die aber diesmal mit Tabulatoren voneinander getrennt in einer .dat-Datei vorliegen. Um diese Datei einzulesen, können wir entweder die generische Funktion `read_delim` mit entsprechendem `delim`-Argument verwenden (das Tabulator-Zeichen wird als `"\t"` geschrieben) oder direkt die spezialisierte Funktion `read_tsv`:

```
read_delim("lecturer.dat", delim="\t")
```

```
# 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
```

```
read_tsv("lecturer.dat")
```

```
# 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
```

Ein weiterer wichtiger Parameter solcher Textdateien 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_delim` mit dem Argument `locale` festgelegt werden. Es ist standardmäßig auf einen Punkt gesetzt, ebenso bei den Funktionen `read_csv` und `read_tsv`. Sollten die Zahlen 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 `delim=";"` und `locale=locale(decimal_mark=",")` bzw. verwendet die spezialisierte Funktion `read_csv2` (welche ein Semikolon als Spaltentrennzeichen und ein Komma als Dezimaltrennzeichen annimmt).

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

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. 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")
```

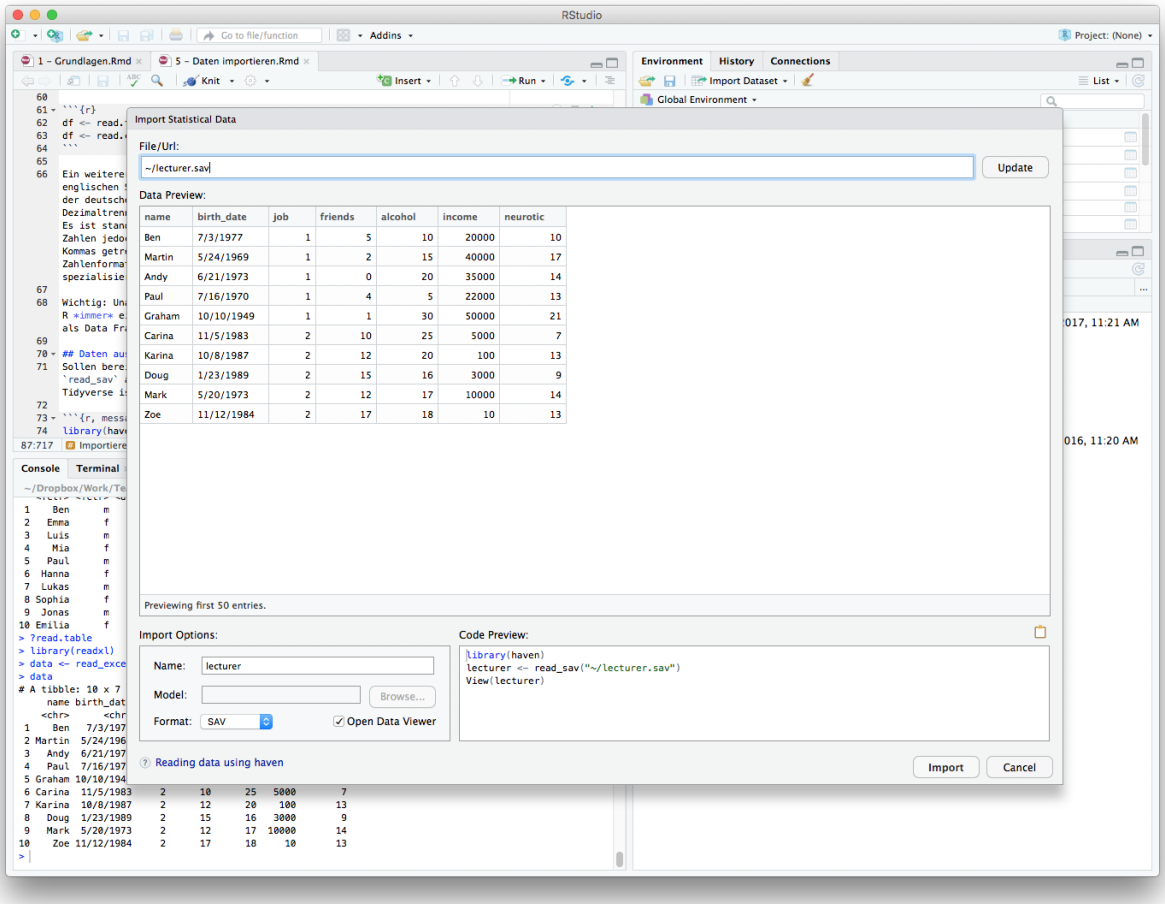
Zusammenfassung

Die folgende Tabelle fasst die Funktionen zum Importieren von unterschiedlichen Dateiformaten zusammen.

Dateiformat	Trennzeichen	Paket	Funktion
Text	verschieden	`readr`	`read_delim`
CSV	`,`	`readr`	`read_csv`
CSV	`;`	`readr`	`read_csv2` (Dezimaltrennzeichen `,`)
TSV/DAT	Tabulator (`\t`)	`readr`	`read_tsv`
SAV	-	`haven`	`read_sav`
XLS/XLSX	-	`readxl`	`read_excel`

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:

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

```
# 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
```

Man erkennt, dass die numerischen Werte korrekt als Zahlen erkannt wurden (der Typ `dbl` entspricht Kommazahlen, d.h. einem numerischen Vektor). Allerdings besitzt die Spalte 3 (`job`) lediglich zwei Werte, d.h. hier wäre eine kategorische Variable (ein Faktor) besser geeignet als eine numerische. Die erste Spalte (`name`) wurde korrekt als `chr` (Character-Vektor) erkannt. Auch die zweite Spalte (`birth_date`) wurde als Charakter-Vektor erkannt, aber es gibt in R einen eigenen Datentyp für Datumswerte (was u.a. bewirkt dass man mit Datumsangaben z.B. rechnen kann).

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 `dbl`, sollte aber als Faktor interpretiert werden. Die Spalte kann entsprechend neu erstellt und überschrieben werden (hier wird vorausgesetzt dass wir wissen, welche Zahlen den jeweiligen Faktorstufen entsprechen):

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

```
# A tibble: 10 x 7
   name   birth_date job      friends alcohol income neurotic
   <chr>  <chr>      <fct>      <dbl>   <dbl>  <dbl>    <dbl>
 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
```

```
# A tibble: 10 x 7
   name   birth_date job      friends alcohol income neurotic
   <chr>  <date>     <fct>      <dbl>   <dbl>  <dbl>    <dbl>
 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).

Daten speichern

Wenn man ein bestehendes Data Frame oder Tibble in einer Datei abspeichern möchte, geht das am einfachsten mit der Funktion `write_delim` (bzw. `write_csv` und `write_tsv`). Dies funktioniert prinzipiell ganz analog zu den oben beschriebenen Lesefunktionen, nur gibt man hier sowohl das zu speichernde Data Frame als auch den Dateinamen an.

Übungen

Übung 1

Sehen Sie sich die Hilfe zur Funktion `read_delim` aus dem `readr`-Paket an. Welches Argument setzt das Trennzeichen der Spalten? Welches Argument setzt das Dezimaltrennzeichen? Mit welchem Argument können Sie das Zeichen für fehlende Werte festlegen?

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 (nach dem ersten Wahlgang). Laden Sie diese Daten 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.

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 Tibble namens `df` ein.
- Geben Sie die ersten paar Zeilen am Bildschirm aus.
- Wie viele Zeilen und Spalten hat der Datensatz?
- 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_delim` sicher, dass diese fehlenden Werte richtig eingelesen werden (und somit korrekt als `NA` interpretiert werden).