

Reading in data

Programming in R for Data Science

Anders Stockmarr, Kasper Kristensen, Anders Nielsen

Data Import

R can import data many ways. Packages exist that handle import from software systems like

- ▶ EXCEL;
- ▶ Plain text files;
- ▶ SAS;
- ▶ SPSS;
- ▶ STATA;
- ▶ etc.

Issues that you must attend to in most cases similar; Excel may present specific problems.

We shall look at import from plain text files.

Package installation

For your specific data type, find the relevant package and install it:

- ▶ Open the R GUI;
- ▶ Click on the 'packages' tab;
- ▶ Choose the package to install;
- ▶ Load the package into R with the `library()` function.

The package `Hmisc` contains functions that handles import from SPSS. Once installed, the package contents can be loaded into R (made available to the R system) with the function call

```
> library(Hmisc)
```

Reading data from a text file

- ▶ Frequently data is collected in white space separated columns, where the first line indicate the variable name:

```
x1  x2  x3
2  0.3  0.01
2  1.0  0.11
3  2.1  0.04
3  2.2  0.02
1  0.1  0.10
1  0.2  0.06
```

- ▶ The function `read.table()` is designed to read this format

```
> mydat <- read.table("c:/datadir/filename.dat", header = TRUE)
```

- ▶ The data frame `mydat` now contains

```
> mydat
  x1 x2 x3
1  2 0.3 0.01
2  2 1.0 0.11
3  3 2.1 0.04
4  3 2.2 0.02
5  1 0.1 0.10
6  1 0.2 0.06
```

The R working directory

R has a search path, the *R working directory*, where it stores its workspace and look for files.

You can locate the working directory with the 'get working directory' command,

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

The working directory can be changed with the 'set working directory' command:

```
> setwd("c:/otherdatadir")
> getwd()
[1] "C:/otherdatadir"
```

For files stored in the working directory or subfolders, you can just specify the path from the working directory when reading them.

Example:

- ▶ If the data is located in the 'Data' folder in your working directory, write
`mydat<-read.table("Data/filename.mydat", header=TRUE)`

The read.table() function

- ▶ The read.table() function has a lot of optional arguments:

```
> args(read.table)
```

```
function (file, header = FALSE, sep = "", quote = "\"'",  
  dec = ".", numerals = c("allow.loss", "warn.loss",  
    "no.loss"), row.names, col.names, as.is = !stringsAsFactors,  
  na.strings = "NA", colClasses = NA, nrows = -1,  
  skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
  strip.white = FALSE, blank.lines.skip = TRUE,  
  comment.char = "#", allowEscapes = FALSE, flush = FALSE,  
  stringsAsFactors = default.stringsAsFactors(),  
  fileEncoding = "", encoding = "unknown", text,  
  skipNul = FALSE)
```

NULL

- ▶ Some of the important ones are:
 - ▶ header: Is the first line variable names or not?
 - ▶ sep: What character is used to separate the columns?
 - ▶ dec: What character is used as decimal separator?
 - ▶ nrows: How many rows do we want to read?
 - ▶ na.strings: What string represent a missing value?
 - ▶ skip: How many lines to skip before start reading?
 - ▶ comment.char: What char in the beginning of a line should indicate that the line should be skipped?

read.table() example 1

Consider the data file

```
This file  
has a bit of text  
and an empty line  
before the data  
a b c  
1 2 3  
4 5 6  
and then some more text at the end
```

```
> dat<-read.table("Data/testdat1.dat", header=TRUE, skip=5, nrow=2)  
> dat
```

```
  a b c  
1 1 2 3  
2 4 5 6
```

read.table() example 2

Now, look at the data file

```
| A B C |  
| 1 2 3 |  
| 4 3,2 2 |  
| 1 5 |  
| ; below this line is the extended data |  
| 5 4 6 |
```

```
> dat<-read.table("Data/testdat2.dat", header=TRUE, na.strings=".",  
+                comment.char=";", dec=",")
```

```
> dat
```

```
   A    B    C  
1 1 2.0    3  
2 4 3.2    2  
3 1 5.0 NA  
4 5 4.0    6
```


Variants of `read.table()`

- ▶ Other functions which are useful for reading data frames from files are:
 - ▶ `read.csv()` comma separated, dot as decimal point
 - ▶ `read.csv2()` `sep=";"` and `dec=","`
 - ▶ `read.fwf()` fixed width format
- ▶ Additional arguments are similar to those of `read.table()`

`read.csv()` and `read.csv2()` are adapted to Excel tables saved as csv files. Which one you need to use depends on your system's regional settings; this machine adheres to Western European locales, and matches `read.csv2()`.

Reading text files from Excel

How to read in a table from Excel in text format:

- ▶ Access the sheet in your Excel file where your table is;
- ▶ Save the active sheet in csv (MS-DOS) format;
- ▶ Read in the table with `read.csv2()`.

Saving in other text formats works as well, just use the appropriate reader function.

Reading from more complicated files

- ▶ `scan()` can be a little tricky to use, but is very flexible.
- ▶ Its simplest use is:

```
      | 4.141593 5.141593 6.141593 7.141593 8.141593 |  
> vec<-scan("scantest.txt")  
> vec  
[1] 4.141593 5.141593 6.141593 7.141593 8.141593
```

Reading from more complicated files

- `readLines()` Reads entire lines.

| A | B | C |
|----------|----------|----------------------------|
| 1.324654 | 2.324654 | 3.324654 4.324654 5.324654 |
| How many | roads | |

```
> vec<-readLines("readlinetest.txt")
```

```
> vec
```

```
[1] "A B C"
```

```
[2] "1.324654 2.324654 3.324654 4.324654 5.324654"
```

```
[3] "How many roads"
```

```
> strsplit(vec[2], " ")
```

```
[[1]]
```

```
[1] "1.324654" "2.324654" "3.324654" "4.324654" "5.324654"
```

```
> as.numeric(strsplit(vec[2], " ")[[1]])
```

```
[1] 1.324654 2.324654 3.324654 4.324654 5.324654
```

File connections

- ▶ File connections can open a file for reading different sections in different ways. Consider:

```
> f1<-file("readlinetest.txt", open="r")
```

```
> scan(f1,what="",nlines=1)
```

```
[1] "A" "B" "C"
```

```
> scan(f1,what=double(),nlines=1)
```

```
[1] 1.324654 2.324654 3.324654 4.324654 5.324654
```

```
> readLines(f1)
```

```
[1] "How many roads"
```

```
> close(f1)
```