

# Reading (and writing) data in R

#### R Foundations for Life Scientists

Marcin Kierczak | 06-Nov-2020 NBIS, SciLifeLab

## **Reading data**



- Reading data is one of the most consuming and most cumbersome aspects of bioinformatics...
- R provides a number of ways to read and write data stored on different media (file, database, url, twitter, Facebook, etc.) and in different formats.
- Package foreign contains a number of functions to import less common data formats.





Most often, we will use the read.table() function. It is really, really flexible and nice way to read your data into a data.frame structure with rows corresponding to observations and columns to particular variables.

The function is declared in the following way:

```
read.table(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)*
```

# read.table parameters



You can read more about the *read.table* function on its man page, but the most important arguments are:

- file the path to the file that contains data,
- header a logical indicating whether the first line of the file contains variable names,
- sep a character determining variable delimiter, e.g. comma for csv files,
- quote a character telling R which character surrounds strings,
- dec acharacter determining the decimal separator,
- row/col.names vectors containing row and column names,
- na.strings a character used for missing data,
- nrows how many rows should be read,
- skip how many rows to skip,
- as.is a vector of logicals or numbers indicating which columns shall not be converted to factors,
- fill add NA to the end of shorter rows,
- stringsAsFactors a logical. Rather self explanatory.

## read.table and its siblings



The read.table function has some siblings, functions with particular arguments pre-set to a specific value to spare some time:

- read.csv() and read.csv2() with comma and semicolon as default sep and dot and comma as dec respectively,
- read.delim() and read.delim2() for reading tab-delimited files.

We, however, most often use the canonical read.table() as it is the most flexible one.

## read.table — example use



### What if you encounter errors?



- StackOverflow,
- Google just type R and copy the error you got without your variable names,
- open the file has the header line the same number of columns as the first line?
- in Terminal (on Linux/OsX) you can type some useful commands.

#### Useful commands for debugging

- cat phenos.txt | awk -F';' '{print NF}' prints the number of words in each row. -F';' says that semicolon is the delimiter,
- head -n 5 phenos.txt prints the 5 first lines of the file,
- tail -n 5 phenos.txt prints the 5 last lines of the file,
- head -n 5 phenos.txt | tail -n 2 will print lines 4 and 5...
- wc -l phenos.txt will print the number of lines in the file
- head -n 2 phenos.txt > test.txt
   will write the first 2 lines to a new file

If it still does not give you a clue — just try to load first line of the file.

If this still did not help, split the file in two equal-size parts. Check which part gives the error. Split this part into halves and check which 1/4 gives the error... It is faster than you think!

### Writing with write.table()



read.table() has its counterpart, the write.table() function (as well ass its siblings, like write.csv()). You can read more about it in the documentation, let us show some examples:

## Writing big data



- HINT: write.table() is rather slow on big data it checks types for every column and row and does separate formatting to each. If your data consists of only one type of data, convert it to a matrix using as.matrix before you write it!
- You may want to use function scan() that reads files as vectors. The content does not have to be in the tabular form. You can also use scan to read data from keyboard: typed.data <- scan()</pre>
- If data are written as fixed-width fields, use the read.fwf() function.
- Also check out the readLines() function that enables you to read data from any stream.

#### Read data in XLS and Matlab



```
library(gdata)
# Note, the gdata:: -- not necessary, but
# good to refresh your memory:-)
data <- gdata::read.xls('myfile.xls', sheet = 2)

library(R.matlab)
data <- R.matlab::readMat("mydata.mat")</pre>
```

### Working with remote data



```
url <- 'https://en.wikipedia.org/wiki/List_of_countries_by_average_wage'
conn <- url(url, 'r')
raw.data <- readLines(conn)
raw.data[1:3]

## [1] "<!DOCTYPE html>"
## [2] "<html class=\"client-nojs\" lang=\"en\" dir=\"ltr\">"
## [3] "<head>"
```

But data is often tabularized...

```
library(rvest)
html <- read_html(url)
tables <- html_nodes(html, 'table')
data <- html_table(tables[4])[[1]]
data[1:5, ]</pre>
```

```
## Country PPP Nominal

## 1 Iceland 66,504 70,628

## 2 Luxembourg 65,449 69,895

## 3 Switzerland 64,109 90,127

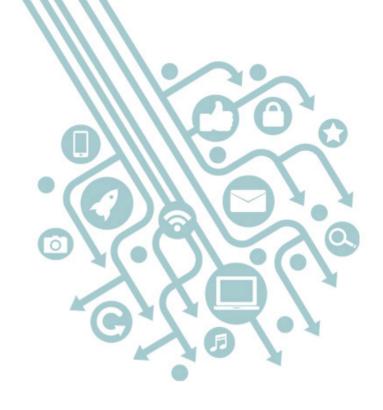
## 4 United States 63,093 63,093

## 5 Denmark 55,253 64,229
```





It is also relatively easy to work with different databases. We will focus on MySQL and present only one example that uses the *RMySQL* package (check also *RODBC* and *RPostgreSQL*).



# See you at the next lecture!

R version 4.0.3 (2020-10-10)

Platform: x86\_64-pc-linux-gnu (64-bit)

OS: Ubuntu 18.04.5 LTS

Built on: 606-Nov-2020 at 32:20:45

2020 • SciLifeLab • NBIS