Table of Contents

Importing data into R

Importing data from spreadsheet (*.txt, *.csv, Excel file)

This is a common situation - you have data in some spreadsheet software (eg Excel in Windows, or Numbers in Apple) and you want to upload the data into R. There are several ways how to do it. **I strongly recommend you to save the datasheet from Excel as a *.txt (preferably) or *.csv file** (choose *Save as* option, and in the Excel saving wizard in the field "Save as type" (below the name of the file) choose *Text (Tab delimited)(*.txt)* for plain text format where values are separated by a tab character (recommended), or *CSV (Comma delimited)(*.csv)* for the file where cells are separated by commas (or semicolon, depending on your language setting). Store the file in a chosen folder, **and load the data into R from this file**. Alternatively (but less optimally) you may copy the data spreadsheet from Excel to clipboard and load to R via the clipboard (however, this is not a reproducible way, since you cannot code it). Or, you may upload data directly from the Excel file stored on your computer (this may or may not work, depending on the version of Excel and R you are using). Alternatively, if data are available online (e.g. shared as Dropbox link, or stored on some website), you can import them directly from the online source, or (if data can be public) create their online copy on www.pastebin.com (check options here for how to make it).

When you **load the file from your computer** using some of the functions below, there are two ways how to do it:

- together with the name of the file, include also the absolute path to the file on your computer, using either forward slash (`/`), or double backslash (`\\`) to separate the folder names (e.g. "`c:/path/to/data/folder/data-for-import.txt`" or "`c:\\path\\to\\data\\folder\\data-for-import.txt`"), or
- change the R working directory to the folder in which the file is stored by `setwd("c:/path/to/data/folder")` function and then in the loading function you can use only the name of the file without an absolute path (e.g. "`data-for-import.txt`").

Example data. The data we will use here as example (`data-for-import.*` files) look like this (note that the first column are row names, and the first row is header = column names):

rownames	x	y
P001	4.925152656	1.874884168
P002	4.915280266	1.874884168
P003	4.890599291	1.889368528
P004	4.836301144	1.913509128
P005	4.831364949	1.918337248
P006	4.796811584	1.952134088
...	...	...

Using *.txt (tab delimited) format

Use the file data-for-import.txt, which is the plain text, with cells separated by tabulators. Save it into some folder, and then specify the address to the folder in the `file` argument, for example (if the file is saved to folder "`c:/path/to/data/folder`")):

```
imported_data_1 <- read.delim (file = "c:/path/to/data/folder/data-for-import.txt", row.names = 1)
```

The object `imported_data_1` is a dataframe. Note that the function `read.delim` is derived from more general function `read.table` by setting some of the `read.table` arguments to different default values (`sep = '\t', header = TRUE`, meaning that the function expects the data have values separated by tab characters (`\t`) and the first row of the data will become header = names of variables). If the first column of the data should become row names (as in this case), include argument `row.names = 1`.

The argument `file` can also be a URL link to the location of the file on internet:

```
imported_data_1 <- read.delim ("https://www.davidzeleny.net/wiki/lib/exe/fetch.php/recol:data:data-for-import.txt", row.names = 1)
```

Alternative option is to use `read_delim` function from the package `readr` (part of `tidyverse`):

```
imported_data_1a <- readr::read_delim ("https://www.davidzeleny.net/wiki/lib/exe/fetch.php/recol:data:data-for-import.txt", row.names = 1)
```

Note that there is no default setting for the argument `delim` in `read_delim` function, you need to always specify (here tabulator, `delim = "\t"`). Using `readr` package has it's specialities: imported data are not `data.frame`, but they are `tibbles` (a special data frame format used within tidiverse). There are number of differences, one important is that by default `tibble` does not have row names. The `tibble` can be converted into standard data frames using `as.data.frame` function, but if the first column of the `tibble` is row name, it needs to be converted into `row.names` in `data.frame`:

```
imported_data_1b <- as.data.frame (imported_data_1a[,-1], row.names = imported_data_1a[1,])
```

(the function `as.data.frame` is applied on the tibble with the first column removed, and the first column is supplied into the argument `row.names`).

Note: **different countries may have different specifications of what decimal separator is used**; for English (and Chinese) setting the decimal separator is a dot (`.`), while for some other countries (Germany, France, French, among others) it may be a comma (`,`). This behaviour can be changed in the Regional Setting of your computer. If your setting is using comma as decimal separator, then *.txt file saved from Excel will most likely also contain commas. However, the `read.delim` (and also `read.table` or `readr::read_delim`) function by default expects the separator to be dot (default setting of argument `dec = "."`). A simple solution is to use `read.delim` or other functions with argument `dec = ","`; in this way, the comma will be interpreted as decimal separator. Values with decimal commas imported into R without changing `dec` into comma will be imported as factors (you can easily check that by applying `summary` function on the imported data frame: if the column is numeric, the summary will return the basic statistics (mean, median, min and max), while if the column is a factor, it will return a list of individual factor values)

Using *.csv (comma separated values) format

Use the file data-for-import.csv, in which cells are separated by commas (,) and decimal signs are represented by dots (.). **Note that *.csv format is a tricky one, since it follows different standards in different countries and on different platforms** - so while in Taiwan, western Europe and elsewhere the cells are delimited by commas (,) and decimals are separated by dots (.), in France, Czech and elsewhere the *.csv format has cells delimited by semicolons (;) and decimals by commas (,).

```
imported_data_2 <- read.csv (file = 'data-for-import.csv', row.names = 1)
```

Alternatively, you can use general function `read.table` with setting appropriate arguments: `head = T` means that the first row of data is taken as header (column names), `sep = ","` sets the delimiter to be semicolon, and `dec = "."` sets decimal separator to be a dot (this is default of `read.table` function, not necessary to change).

If your *.csv file is following "semicolon/comma" standard instead of "comma/dot" one, you may use the function `read.csv2`, which expects separators to be semicolon (;) and decimals commas (,).

Alternatively, `readr::read_delim` function with argument `delim = ","` can be used (check Using *.txt (tab delimited) format for the specialities around `read_delim` function and `readr` package in general).

Using clipboard

Use the file data-for-import.xlsx, the sheet *pig*, and copy the table into clipboard (take care to copy really just cells with data, not empty cells around - use of CTRL+A is thus not recommended). Then use:

```
imported-data_3 <- read.delim (file = 'clipboard', row.names = 1)
```

The function `read.delim` expects that the input (in this case the one stored in clipboard) is a plain text delimited by tabulators.

Using clipboard is a quick and dirty way of importing data into R; it is relatively fast, but a considerable drawback is that it is not reproducible - you cannot code it, instead, you need to always manually open the file, copy the table to the clipboard and upload into R (this is prone to errors if you want somebody to replicate the same thing).

Import directly from Excel (*.xls or *.xlsx) file

Importing data directly from Excel used to be quite complicated (see e.g. this website), but the package `readxl` made it much easier. Still, I do not suggest you use this option, since it may not be replicable on every platform, and it may change with a newer version of Excel. Alternatively, if you need to do this often (manipulate data manually in a spreadsheet and upload them into R, or even export from R back to the spreadsheet), consider using Google Sheets solution instead (see e.g. here).

Use of the package `readxl` is pretty straightforward. There is a suite of functions for reading Excel file, like read_excel, read_xls and read_xlsx. Important arguments are `path` and `sheet`, first for the name of the file (optinally including the path to the folder) and the second the name of the sheet which should be imported. We may try it on the Excel file data-for-import.xlsx - save it somewhere on your computer, and use:

```
# install.packages ('readxl')
library (readxl)
imported_data_4 <- read_excel ('c:/path/to/data/folder/data-for-
import.xlsx', sheet = 'pig')
```

The object `imported_data_4` created in R is not data.frame, but `tibble` (alternative to data.frame in the `tidyverse` packages). If you don't like that (or you don't know how to use it), you can simply convert it into data.frame using `as.data.frame` function. If the first column of the data are in fact row names, they may need to be assigned as such:

```
imported_data_4a <- as.data.frame (imported_data_4[,-1], row.names =
imported_data_4[,1])
```

Note that the functions from package `readxl` cannot read Excel files directly from internet, unlike e.g. `read.table`. But there is a workaround - first, download the Excel file into R as a temporary file, and the read it using `readxl` function:

```
# install.packages ('readxl')
library (readxl)
url <-
'https://www.davidzeleny.net/wiki/lib/exe/fetch.php/recol:data:data-for-impo
rt.xlsx'
destfile <- tempfile ()
download.file (url, destfile, mode = 'wb')
imported_data_5 <- read_excel(destfile, sheet = 'pig')
```

Note that in the function `download.file` it is important to specify the argument `mode = 'wb'` (on Windows, if the argument `mode` is not set up, the type of the file will be determined from the file extension; in case of “xls” and “xlsx”, it would attempt to download these files as plain text, but in fact, these files need to be downloaded as binaries!).

The library `readxl` is a part of the `tidyverse` packages, and as such it does not use standard `data.frame` format for data frames, but unique `tibble` (check `class (imported_data_5)`). One feature of `tibble` is that **it does not have rownames, and rownames are therefore imported as the first column of the data frame**. This may or may not be handy for future analysis. To convert `tibble` into standard `data.frame`, use the function `as.data.frame`. Then, still, you need to move first column into the rownames of the newly created data frame. Alternatively, there are functions like `column_to_rownames` in the package `tibbles` which can help you with that.

```
imported_data_5_df <- as.data.frame (imported_data_5)
rownames (imported_data_5_df) <- imported_data_5_df[,1]
imported_data_5_df <- imported_data_5_df[,-1]
```

Import *.RData file

Binary data storing the R object can be loaded into R using function `load`. Download data data-for-import.rdata to your computer and use:

```
load ('data-for-import.RData')
```

This should create the variable `data_for_import` in your Global environment (it will appear among variables). You do not need to assign the result to a new variable (nothing like `imported_values <- load ('data-for-import.RData')` will work).

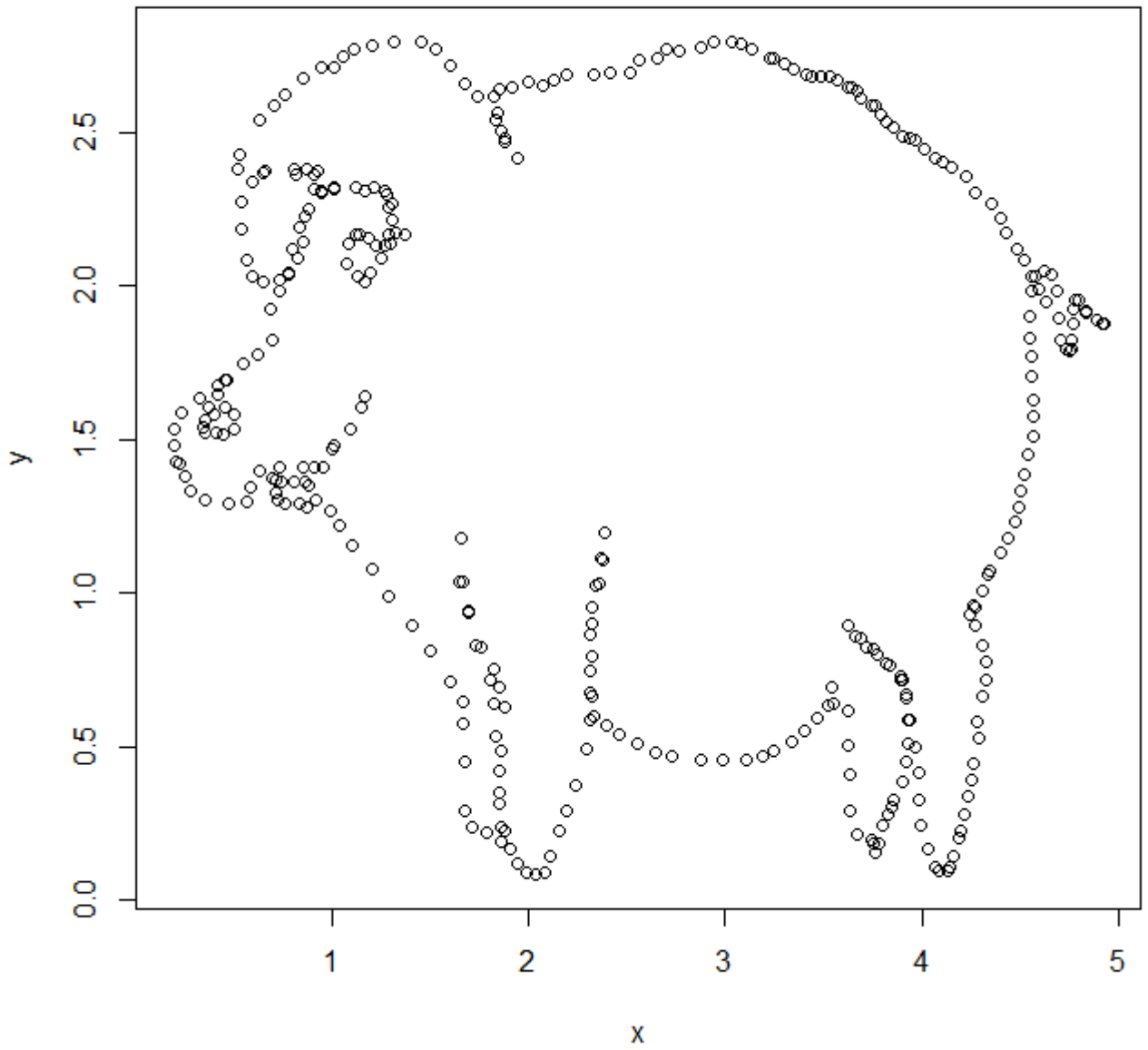
Alternatively, `load` function can read directly data from website, if the URL link is wrapped by function `url`:

```
load (url
('https://www.davidzeleny.net/wiki/lib/exe/fetch.php/recol:data:data-for-import.rdata'))
```

And what are the data in the example data-for-import file?

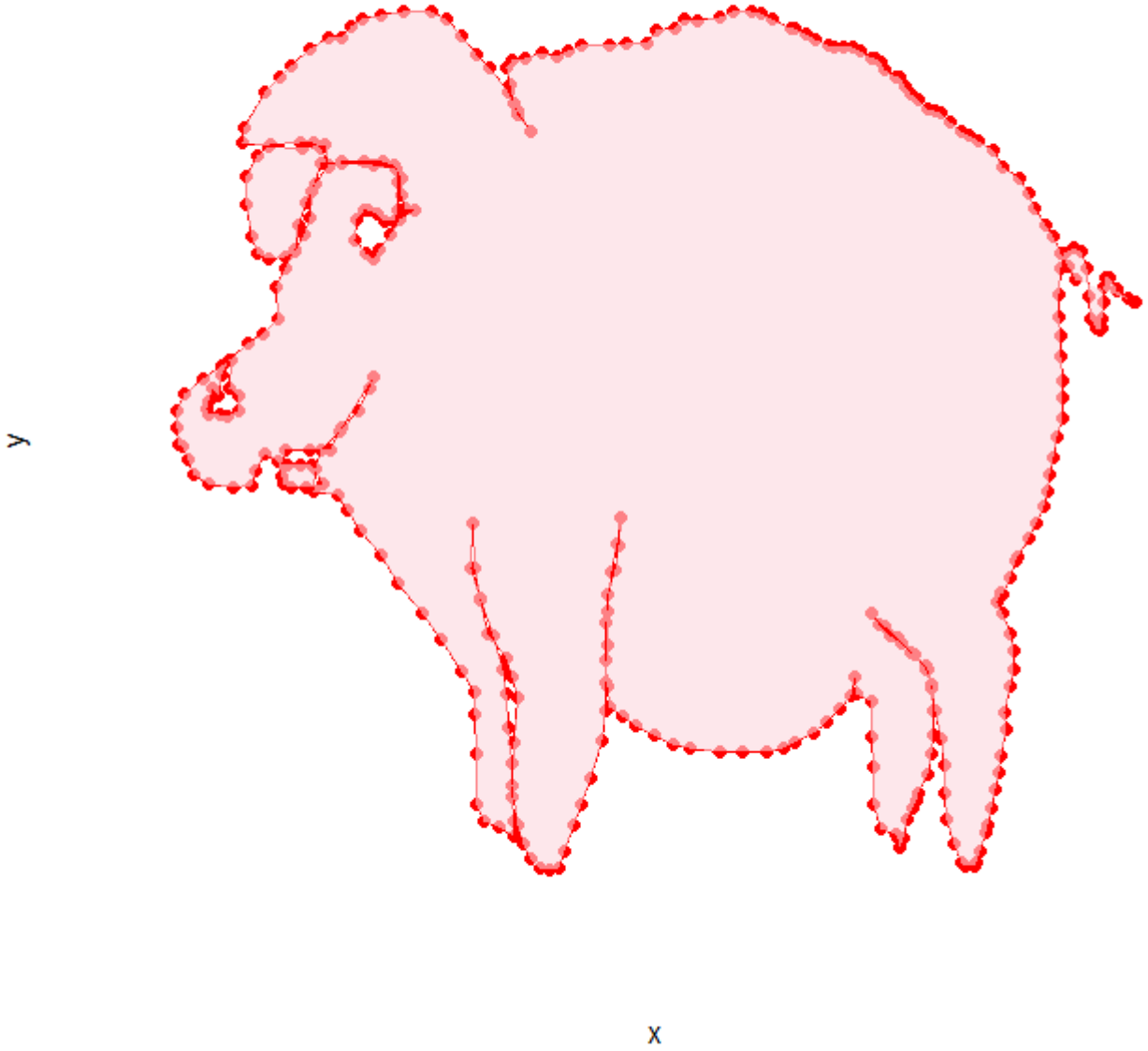
It's actually really a pig!

```
load (url
('https://www.davidzeleny.net/wiki/lib/exe/fetch.php/recol:data:data-for-import.rdata'))
plot (data_for_import)
```



The pig! (note that the color I used to fill the polygon here is called Pig Pink :)

```
plot (data_for_import, type = 'o', pch = 16, col = 'red', frame.plot = F,
axes = F)
polygon (data_for_import, pch = 16, col = '#FCD7DE99', border = NA)
```


y
x