Handling data in R

Antonio Gasparrini

01 November 2023

Contents

This session illustrates the practical but very important tasks of managing data in R, and in particular how to import/export data from/to your R session in formats compatible with different statistical programs. The basic R tools used for this purpose are very specific and inflexible, as is often the case with input/export tools. Therefore, only the standard methods for importing and exporting data are presented here. The first section describes how to load datasets already included in the R installation. The second section introduces data formats used by R, while the third section explains the steps to import data stored in basic formats used in any statistical program, with a mention of specialised formats as well. The next section illustrates how to display and summarize data in R, while the last one deals with exporting data outside R.

Loading data

The R distribution comes with several datasets included in standard and recommended packages. Many others can be accessed in other contributed packages, downloadable from CRAN or other repositories. Most of the datasets are small collections of real or simulated data and are included for running examples on the use of functions in the related packages. Others constitute real and full datasets made available through R. A list of datasets available in the packages loaded in the session is produced by:

```r
data()
```

with a window opening in the EDITOR panel of RStudio or a sub-window of the standard R GUI. As shown in the list, the standard package DATASETS contain a long list of different types of datasets. To access datasets in contributed packages, these need to be loaded into the session with `library()`. These datasets are stored in different objects, mostly data frames, but also lists, tables and others.

Almost all the datasets included in R packages are handled by a mechanism called *lazy loading*. In practice, they are immediately available for use once the package is loaded into the session, although they are not pre-loaded in order to save memory. However, if you want the dataset to be visible in the global environment, you need to load it directly using `data()` again. For example:

```r
data(esoph)
```

loads the data frame `esoph` from the package DATASETS. You can see that the data frame `esoph` now appears in your Environment window.

Datasets included in contributed but not recommended packages (not included in the main R distribution) can be accessed as well, provided the package has been installed. The package EPI can be installed by:

```r
install.packages("Epi")
```

or alternatively using the tools in the Packages window, specifically the INSTALL button. The data frame object `births` in EPI can then be loaded by:

```r
library("Epi")
data(births)
```

All the datasets in R are documented in a related help page. Information on the two datasets is accessed by `help(esoph)` and by `help(births)`, with the help page appearing in your Help window. Specifically, the former includes data from a case-control study of oesophageal cancer in France, while the latter stores data from 500 singleton births in a hospital in London (see the help pages).

In many cases, the data are not already available through R packages, and need to be created or imported into the session. A data frame can be created with the function `data.frame()`. Alternatively, basic functions such as `readline()`, `scan()` or `data.entry()` can be used to manually input the data from the keyboard.

R data formats

Luckily enough, in almost all the analyses you are not expected to input the data manually, but you may import them from external files. R uses specific formats to store data compatible with the software. First, datasets can be included as objects in an environment stored in an RDATA workspace file. In this case, the dataset can be imported simply by loading the environment with the function `load()`. For instance:

```r
load("files/mtcars.RData")
```

The expression above loads the file MTCARS.RDATA from the folder FILES in the working directory. The file stores an R environment that includes the data frame `mtcars`, representing a dataset originally stored in the DATASET package (see `help(mtcars)`). Different paths can be specified to access data in other locations, and you can also input a URL to download and read a file stored on the web. As always, remember to use the forward slash '/' or double back slash '\\'. Note that RDATA files can store multiple objects, optionally including data frames referring to several datasets as well as different types of objects. More importantly, consider that, when you load an environment, objects in the global environment with the same name will be replaced without any warning.

A more natural and safe method is to use RDS files, another R data format that can store a single object (usually a data frame) and can be imported directly using the `readRDS()` function and a proper assignment operation, thus avoiding the issue of accidentally overwriting existing objects. An example:

```r
airquality <- readRDS("files/airquality.RDS")
```

Here, R reads the file and assigns it to the data frame `airquality` in the same folder FILES, also originally stored in the DATASET package (see `help(airquality)`).

Note that, in RStudio, you can simply use the related icon in the ENVIRONMENT window to load a workspace, or (double) click on the RDS and RDATA files in the related folder or in the browser available in the FILES window. Finally, consider that both processes use internal file compression to store data efficiently.

Importing data in different formats

For most of the analyses, the data are not already stored in R-compatible files and need to be imported from files created in different formats, depending on the program used to create them. As mentioned earlier, basic R input facilities are simple, and their rules are fairly strict and even rather inflexible.

The most general format to store data is represented by text files with a tabular structure, with values for each record included in the same row and separated by some symbol. The most common examples are *comma-separated value* files (with extension .csv). The basic tool for reading tabular data stored in text files into R is the function `read.table()`. The function expects the external data to comply with a specific format and allows a (quite extended) set of options for importing data, specified through its arguments. As an example, let's imagine that you want to import a text file named FAITHFUL.TXT (see `?faithful`) from the same folder FILES. The command is the following:

```
faithful <- read.table("files/faithful.txt", header=TRUE, sep=",")
```

The argument `header`, set to `TRUE` (but with default to `FALSE` in `read.table()`), specifies that the first line of the text file contains the names of the variables, while the argument `sep` (as separator) states that the values in the original file are separated by commas. The function has a long list of additional arguments, for example `row.names` and `col.names`, for setting the names of records and variables, respectively. See `?read.table` for more detailed information.

Other functions are wrappers for `read.table()` with alternative pre-defined separators. Specifically, `read.csv()` and `read.delim()` import comma-separated and tab-separated value files, respectively.

Text files represent a well-suited format for importing and exporting files into and from R and other programs (although not always the most efficient). However, R can easily reads data compatible with other major statistical programs (*e.g.*, Stata, SAS, or SPSS), in addition to more specialised formats. There are several dedicated packages with specific functions for different data types. Here we provide an example with basic functions from the package FOREIGN (which must be loaded into the session):

```
library(foreign)
sleep <- read.dta("files/sleep.dta")
```

Specifically, the expression imports a dataset stored in a Stata format with extension .DTA and saves it in a data frame `sleep`, again originally available in the dataset package.

As usual, everything is much simpler in RStudio, at least for standard imports such as that described above. Text files and other formats mentioned above can be imported with the IMPORT DATASET menu in the ENVIRONMENT window. The file can be searched through a browser, and the basic choices about header and separators can be made through a dialogue box.

Displaying and summarising data

A dataset stored in a data frame object can be displayed in the Console window simply by printing the object, *i.e.*, calling its name. However, for big datasets, the visualization in the Console window can be poor. A data frame can also be more effectively displayed in an external window by the function `View()`. For example, with:

```
View(births)
```

the object `births`, previously loaded into the session, now appears in spreadsheet-style data viewer in the Editor window. In RStudio, the same result is obtained simply by clicking on the data frame in the list shown in the Environment window.

Other functions can be useful for displaying datasets. Usually, you are not interested in visualizing the whole data frame: just a few lines are enough to check which types of variables are stored in the dataset. The function `head()` can be used to display the first *n* lines. Let's try it with `mtcars`:

```r
head(mtcars, 3)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

The second argument `n`, not directly named in the command above, determines how many rows are displayed (default to 6, see `?head`). The related function `tail()` prints instead the last *n* records. Both functions work with matrices as well. The command shows the observations for the first three records in `mtcars` for eight variables.

Examining the actual observations in a big dataset is rarely productive, and several tools are available for summarizing the information stored in it. A very compact overview of a dataset is offered by the function `str()`, which displays the internal structure of any R object. When called with the data frame `esoph`:

```r
str(esoph)
```

```
## 'data.frame':    88 obs. of  5 variables:
##  $ agegp    : Ord.factor w/ 6 levels "25-34"<"35-44"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ alcgp    : Ord.factor w/ 4 levels "0-39g/day"<"40-79"<..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ tobgp    : Ord.factor w/ 4 levels "0-9g/day"<"10-19"<..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ ncases   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ncontrols: num  40 10 6 5 27 7 4 7 2 1 ...
```

The function displays the content, providing information about the number of records and variables, and the type of variables (in this case two numeric variables with counts of cases and controls stratified for three ordered factors). A selection of levels and actual observations are also displayed.

The function `summary()` can be used to inspect specific variables, which are normally vectors or factors:

```r
summary(births$gestwks)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   24.69   37.94   39.12   38.72   40.09   43.16      10
```

The function returns the distribution of the numeric variable `gestwks` in the data frame `births`, with information on the gestational week of each birth. The number of missing values is also reported. The function `summary()` can also be called with the whole data frame:

```r
summary(sleep)
```

```
##      extra        group        ID   
##  Min.   :-1.600   1:10   1      :2  
##  1st Qu.:-0.025   2:10   2      :2  
##  Median : 0.950          3      :2  
##  Mean   : 1.540          4      :2  
##  3rd Qu.: 3.400          5      :2  
##  Max.   : 5.500          6      :2  
##                          (Other):8
```

In this case, information on all the variables, according to their classes, is returned.

Exporting data

Exporting data from R follows the same lines of importing them, as described earlier. Datasets can be easily stored as data frame objects and saved in a .RData workspace file, using the function `save()`. For instance, you can save the dataset stored in births as births.RData file with:

```r
save(births, file="files/births.RData")
```

The first argument(s) specifies which object(s) needs to be saved, while the argument `file` states the name of the file. The two names can of course be different. The new file will be placed in the working directory, or different directories if a path is specified. Remember to include the file extension .RDATA for the name in the argument `file`, and optionally the slash '/' or '\\' in the path.

Alternatively, the whole set of objects in the workspace can be saved with `save.image()` or through the related icon in the menu in the Environment window of RStudio. For example:

```r
save.image("files/all.RData")
```

The counterpart of `read.table()` for exporting data in text files is `write.table()`, or related wrapper functions such as `write.csv()`. These functions save the dataset in a text file. For example, the data frame `esoph` can be saved externally in the file ESOPH.TXT with:

```r
write.table(esoph, "files/esoph.txt", sep=",")
```

Similarly to `save()` above, the first two arguments specify the object to be saved and the file of the new external file, respectively. Similarly to `read.table()`, the argument `sep` specifies the separator (default to the blank space " "). In this case, the output can be read as a comma-separated value file. The wrapper `write.csv()`, which sets the separator to comma by default, can also be used. The argument `quote` (default to `TRUE`) specifies if character values will be quoted in the text file.

Again, functions for exporting datasets to files in formats used in other statistical programs are provided by other packages. For instance, the package FOREIGN provides writing functions to export datasets in Stata, SAS, and SPSS. For example, we can save the data frame `airquality` in a Stata format with:

```r
write.dta(airquality, file="files/airquality.dta")
```

Other contributed packages includes functions for exporting data in more specialised format.