[DS-03] Working with data in R

Miguel-Angel Canela
Associate Professor, IESE Business School

Data frames

In the computer implementation of Data Science, data sets are managed as objects called **data frames**. Data frames were born with R, but have been adopted by other languages like Python and Scala.

In R, a data frame is a list of vectors which are presented as columns. These vectors can have different type, but must have the same length. A simple, artificial example follows.

```
> df = data.frame(v1=1:5, v2=c('a', 'b', 'c', 'd', 'e'), v3=rep(-1, 5))
> df
  v1 v2 v3
1  1  a -1
2  2  b -1
3  3  c -1
4  4  d -1
5  5  e -1
```

The columns of a data frame are identified as the elements of a list, i.e. either as `df$var` or as `df['var']`:

```
> df['v1']
 [1] 1 2 3 4 5
```

As with lists, this course uses the syntax `df['var']` to specify columns of a data frame. Although it is less common than the dollar notation, is facilitates the comparison between R code and Python code. Warning: you may get trouble with this system if you use more than once in the line, as in `table(df['var1'], df['var2'])`. Using a double index, i.e. `df[, 'var']`, is safer in R.

Subsetting

When using indexes for subsetting data frames, we follow the same rules as in vectors, for both rows and columns. Two trivial examples follow.

```
> df[1:3, 1:2]
  v1 v2
1  1  a
2  2  b
3  3  c
> df[, -3]
```

```
  v1 v2
1  1  a
2  2  b
3  3  c
4  4  d
5  5  e
```

To select a subset of the actual set of columns of a data frame, we can use indexes, as in `df[, 1:2]` or the column names, as in `df[c('v1', 'v2')]`. Expressions can be used for extracting rows from a data frame just as they are used as for extracting elements of a vector.

```
> expr = df['v1'] == df['v3']**2
> df[expr, ]
  v1 v2 v3
1  1  a -1
```

¶ Note that, as in many other languages, equality in expressions is denoted with a double equal sign (==).

Importing and exporting data sets

Data sets in tabular form can be imported to R as data frames from many formats. This is typically handled with read/write functions. Most of the data sets used in this course come in **CSV files**, which are text files that use the comma as the column separator. The CSV format is very popular, although it must be handled with care for string data. In CSV files, the names of the variables are in the first row, and every other row corresponds to an instance.

In general, text files are imported to data frames with the function `read.table`. For CSV files, there is a special function called `read.csv`, which is a particular case of `read.table`. The default of `read.csv` takes the first line of the file as the names of the variables. The syntax is `dfname = read.csv(file=filename)`. The name of the data frame is chosen by the user, and the name of the file has to contain the path of that file.

Since columns do not have a data type in CSV files, R guesses from the content. If all the entries in a column (except the first row, which is the name) are numbers, that column is imported as numeric. If there is, at least one entry which is not numeric, the column is imported, by default, as a factor. Many users, including myself, do not like this, so we stop it happening with the argument `stringsAsFactors=FALSE`. When this argument is included, all the columns will be either numeric or character vectors. If the string data contained in a CSV file can contain special characters (such as ñ, or á) which can make trouble, I recommend you to include the argument `encoding='UTF-8'`.

To export a data frame to a CSV file, we use the reverse function, `write.csv`. The syntax is `write.csv(dfname, file=filename)`. Again, the file name, supplied by the user, includes the path. The argument `row.names=FALSE` stops R creating a column in the left with the row names, which you probably do not need.

If Excel is installed in your computer, files with the extension CSV are associated to Excel (so, they have an Excel icon). But, in some countries, the comma is replaced by a semicolon. These alternative CSV files are handled in R with the functions `read.csv2` and `write.csv2`.

Exploring a data set

R provides several functions for exploring a data set right after importing or transforming it. These tools are capital to data scientists, since they are constantly checking that a data frame contains what it is expected to contain. I give only a brief explanation, since I explain this with more detail on the examples of this course.

First, we can check the dimensions of the data frame with the functions `dim`, `nrow` and `ncol`, whose meaning is obvious.

```
> dim(df)
 [1] 5 3
> nrow(df)
 [1] 5
> ncol(df)
 [1] 3
```

The functions `head` and `tail` print either the first or the last rows of a data frame. There is an second argument that controls the number of rows returned. The default is 6. Also, the structure of an R object can be explored with the function `str`. For a data frame, it informs on the dimensions and the type that we have in every column of the data frame.

```
> str(df)
'data.frame':  5 obs.  of 3 variables:
 $ v1:  int 1 2 3 4 5
 $ v2:  Factor w/ 5 levels "a","b","c","d",..:  1 2 3 4 5
 $ v3:  num -1 -1 -1 -1 -1
```

¶ If in the definition of `df` we would have included the argument `stringsAsFactors=FALSE`, now the second column would have been a character vector, instead of a factor.

The function `summary` can have various outputs, depending on the nature of the argument. For a data frame, produces a conventional statistical summary.

```
> summary(df)
       v1   v2         v3
 Min.   :1  a:1  Min.   :-1
 1st Qu.:2  b:1  1st Qu.:-1
 Median :3  c:1  Median :-1
 Mean   :3  d:1  Mean   :-1
 3rd Qu.:4  e:1  3rd Qu.:-1
 Max.   :5       Max.   :-1
```

Missing values

Missing values are denoted by `NA` in R. Two useful functions for handling missing values are `is.na` and `na.omit`. To show you how this works, I first introduce an NA in our data frame.

```
> df[1, 2] = NA
```

`is.na` creates a logical vector when applied to a vector of any type: `TRUE` if the corresponding term is `NA` and `FALSE` otherwise.

```
> is.na(df[, 2])
 [1] TRUE FALSE FALSE FALSE FALSE
> sum(is.na(df[, 2]))
 [1] 1
> mean(is.na(df[, 2]))
 [1] 0.2
```

`na.omit` drops the rows of a data frame that have at least one missing value. Warning: check the amount and the source of these NA's before dropping them.

```
> na.omit(df)
2  2  b
3  3  c
4  4  d
5  5  e
```

Duplicates

The function `unique` drops the duplicated entries (in a vector) or the duplicated rows (in a data frame).

```
> unique(df[, 3]
 [1] -1
```

The function `duplicated` returns a logical vector indicating which entries (for a vector) or which rows (for a data frame) are duplicated. The default version checks duplicates top down, but, with argument `fromLast=TRUE`, you can do it bottom up.

```
> duplicated(df[, 3])
 [1] FALSE TRUE TRUE TRUE TRUE
```

Pivot tables

In exploratory analysis, we often use tables produced with the two following functions:

- `table` counts the number of occurrences for each of the possible values of one or two columns of a data frame. It does not include the missing values.
- `tapply` has three arguments: a column of a data frame that we wish to summarize by groups, the grouping variable and the function to be applied in the summary (e.g. the mean).

These functions are better understood in the examples.

Plotting

We typically visualize the data with bar plots, histograms, scatter plots and line plots. Their use is also better understood in the examples.

- `barplot` applies to a single vector, producing a **bar plot**. The arguments are: `main` is the title, `xlab` the label of the horizontal axis and `ylab` the label of the vertical axis.
- `hist` applies to a single vector, producing a **histogram**. The arguments as for `barplot`.
- `plot` can be applied to a single vector or to a pair of vectors. The arguments work as for the preceding two functions, plus `type`, which specifies the type of plot. The default is a **scatter plot**, but `type='l')` produces a **line plot**.

References

1. BC Boehmke (2016), *Data Wrangling with R*, Springer.
2. MJ Crawley (2012), *The R Book*, Wiley. Free access at `ftp.tuebingen.mpg.de/pub/kyb/bresciani`.
3. RI Kabacoff (2010), *R in Action*, Manning. Free access at `kek.ksu.ru/eos/DataMining`.
4. H Wickham & G Grolemund (2016), *R for Data Science*, O'Reilly. Free access at `r4ds.had.co.nz`.