Data Management [01]

Miguel-Angel Canela, IESE Business School

November 10, 2016

### Introduction to R

R is a language which comes with an environment for computing and graphics, available for both Windows and Macintosh. It includes an extensive variety of techniques for statistical testing, predictive modelling and data visualization. R can be extended by hundreds of additional packages available at the **Comprehensive R Archive Network** (CRAN), which cover virtually every aspect of statistical data analysis and machine learning.

Although it was initially restricted to the academic context, R has been gaining acceptance in the business intelligence industry in the last years: one can use R resources in Oracle, Microsoft Azure Cloud, SAP HANA, etc. It has been adopted as the default analytic tool by many well known companies such as Google, which has even written an R Style Book for its programmers.

R provides a GUI, which, called the **console**, in which we can type (or paste) our code. The console works a bit different in Macintosh and Windows. In the Windows console, what you type is in red and the R response in blue. In Macintosh, we type in blue and the response comes in black. When R is ready for our code, the console shows a **prompt** which is the symbol "greater than" (>). With the Return key, we finish a line of code, which is usually interpreted as request for execution. But R can detect that your input is not finished, and then it waits for more input showing a different prompt, the symbol "plus" (+).

In an **R Markdown** document like the one that you are reading, this is usually seen as follows.

2 + 2

## [1] 4

The R console is not user-friendly, so you will probably prefer to work in an interactive developer environment (IDE). **RStudio** is the leading choice and, nowadays, most R coders prefer RStudio to the console. In RStudio, you have the console plus other windows that may help you to organize your task.

### Objects in R

R is **object-oriented**, with many classes of objects. I comment here briefly of some classes which will appear in the analysis of the examples of this course. First, we have **vectors**. A vector is an ordered collection of elements which are all of the same type. Vectors can be **numeric**, **factors** (explained later), **character** (called string in most languages), **logical** (TRUE/FALSE) or of other types not discussed in this course. The following three examples are numeric (x), character (y) and logic (z), respectively.

x <- 1:10  
x

## [1] 1 2 3 4 5 6 7 8 9 10

y <- c("Messi", "Neymar", "Cristiano")  
y

## [1] "Messi" "Neymar" "Cristiano"

z <- x > 5  
z

## [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE

Comments on these examples:

* R distinguishes, as most languages, between integers and real numbers. Nevertheless, you can ignore this distinction unless you are dealing with very big datasets, for which the memory allocated for each record may matter. Complex numbers are in another class, but I skip them.
* The expression c(a,b) packs the elements a and b as the terms of a vector. This is only possible if they are of the same type.
* The quote marks indicate character type.
* An expression like x > 5 is translated as a logical vector with one term for each term of x.

The first term of the vector x can be extracted as x[1], the second term as x[2], etc. **Matrices** are like vectors, but two-dimensional. They can be numeric, character or logical. The terms of a matrix are identified by two indexes. For instance, A[2,3] is the term in the second row, third column.

A <- matrix(1:24, nrow=4)  
A

## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] 1 5 9 13 17 21  
## [2,] 2 6 10 14 18 22  
## [3,] 3 7 11 15 19 23  
## [4,] 4 8 12 16 20 24

A[2,3]

## [1] 10

A **data frame** (a data set) is a set of vectors (presented as columns). These vectors can have different type, but the same length. A vector of a data frame are identified as dataframe$variable. Rows and columns of a data frame are identified as in a matrix.

df <- data.frame(v1=1:10, v2=10:1, v3=rep(-1,10))  
df

## v1 v2 v3  
## 1 1 10 -1  
## 2 2 9 -1  
## 3 3 8 -1  
## 4 4 7 -1  
## 5 5 6 -1  
## 6 6 5 -1  
## 7 7 4 -1  
## 8 8 3 -1  
## 9 9 2 -1  
## 10 10 1 -1

df$v1

## [1] 1 2 3 4 5 6 7 8 9 10

A **factor** is a numeric vector in which the values have labels, called **levels**. Factors are very natural to statisticians (stat packages, like SPSS or Stata use similar systems), but look weird to computer engineers, since programming languages don't have them.

**Dates** can be handled in many ways, including the usual date and datetime formats of databases. I will talk about this later.

Extracting parts of R objects is called is called **subsetting**. Vectors, matrices and data frames can be subsetted in an easy way. Some examples follow.

x[1:3]

## [1] 1 2 3

x[x>=5]

## [1] 5 6 7 8 9 10

A[1:2,3:6]

## [,1] [,2] [,3] [,4]  
## [1,] 9 13 17 21  
## [2,] 10 14 18 22

df[,-3]

## v1 v2  
## 1 1 10  
## 2 2 9  
## 3 3 8  
## 4 4 7  
## 5 5 6  
## 6 6 5  
## 7 7 4  
## 8 8 3  
## 9 9 2  
## 10 10 1

df[df$v1<df$v2,]

## v1 v2 v3  
## 1 1 10 -1  
## 2 2 9 -1  
## 3 3 8 -1  
## 4 4 7 -1  
## 5 5 6 -1

R is a fully functional language. The real power of R lies in defining the operations that you wish to perform as **functions**, so they can be applied many times. For instance, import and export are usually managed by read/write functions. A simple example follows.

f <- function(x) 1/(1+x^2)  
f(1)

## [1] 0.5

You can replace all the "arrows" (<-) by equal signs, and nothing will change. Nevertheless, it is recommended, to the beginner, to use the arrow system, to avoid mistakes. x <- 2 + 2 is read assign("x", 2+2). We can also write 2 + 2 -> x, but 2 + 2 <- x does not make sense. The equal sign is read as <- (as in any programming language).

### Importing data to R

Data sets can be imported to R data frames from many formats, typically from text files, with the read.table function. For **csv files**, there is a special function read.csv. The default of read.csv is reading the first line of the file as the names of the variables.

To capture the data, we have to specify a **path** in our computer or a URL. Let me illustrate with the following code, which imports a csv file containing daily OCHL (Open/Close/High/Low) data for the NIFTY 500 index, from 2008-01-01 to 2015-12-31, extracted from Yahoo Finance India.

url1 <- "http://real-chart.finance.yahoo.com/table.csv"  
url2 <- "?s=%5ECRSLDX&a=00&b=01&c=2008&d=11&e=31&f=2015"  
url <- paste(url1, url2, sep="")  
nifty <- read.csv(url)

I perform some checks on the data frame.

dim(nifty)

## [1] 1946 7

head(nifty)

## Date Open High Low Close Volume Adj.Close  
## 1 2015-11-05 6734.80 6734.80 6654.10 6661.70 0 6661.70  
## 2 2015-11-04 6790.10 6796.35 6733.55 6741.20 0 6741.20  
## 3 2015-11-03 6771.95 6781.60 6733.50 6755.15 0 6755.15  
## 4 2015-11-02 6747.40 6747.65 6695.40 6740.90 0 6740.90  
## 5 2015-10-30 6796.05 6815.80 6733.85 6750.95 0 6750.95  
## 6 2015-10-29 6833.10 6833.10 6777.65 6784.85 0 6784.85

tail(nifty)

## Date Open High Low Close Volume Adj.Close  
## 1941 2008-01-08 5543.15 5563.50 5402.35 5460.30 0 5460.30  
## 1942 2008-01-07 5443.05 5518.00 5442.90 5500.15 0 5500.15  
## 1943 2008-01-04 5488.75 5527.80 5484.95 5502.60 0 5502.60  
## 1944 2008-01-03 5412.95 5478.70 5411.45 5442.20 0 5442.20  
## 1945 2008-01-02 5407.65 5449.95 5331.20 5437.80 0 5437.80  
## 1946 2008-01-01 5370.35 5399.75 5348.85 5384.55 0 5384.55

The structure of an R object can be explored with the function str. Note that, in this case, the variable Date has been imported as a factor. This is the default for importing string data in R.

str(nifty)

## 'data.frame': 1946 obs. of 7 variables:  
## $ Date : Factor w/ 1946 levels "2008-01-01","2008-01-02",..: 1946 1945 1944 1943 1942 1941 1940 1939 1938 1937 ...  
## $ Open : num 6735 6790 6772 6747 6796 ...  
## $ High : num 6735 6796 6782 6748 6816 ...  
## $ Low : num 6654 6734 6734 6695 6734 ...  
## $ Close : num 6662 6741 6755 6741 6751 ...  
## $ Volume : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ Adj.Close: num 6662 6741 6755 6741 6751 ...

The funtion summary works in different ways in objects of different nature. For numeric variables, it produces some summary statistics.

summary(nifty)

## Date Open High Low   
## 2008-01-01: 1 Min. :1960 Min. :1967 Min. :1797   
## 2008-01-02: 1 1st Qu.:4046 1st Qu.:4075 1st Qu.:4014   
## 2008-01-03: 1 Median :4418 Median :4447 Median :4382   
## 2008-01-04: 1 Mean :4584 Mean :4612 Mean :4548   
## 2008-01-07: 1 3rd Qu.:4855 3rd Qu.:4875 3rd Qu.:4824   
## 2008-01-08: 1 Max. :7427 Max. :7428 Max. :7288   
## (Other) :1940   
## Close Volume Adj.Close   
## Min. :1967 Min. :0.00e+00 Min. :1967   
## 1st Qu.:4045 1st Qu.:0.00e+00 1st Qu.:4045   
## Median :4414 Median :0.00e+00 Median :4414   
## Mean :4579 Mean :3.43e+05 Mean :4579   
## 3rd Qu.:4849 3rd Qu.:0.00e+00 3rd Qu.:4849   
## Max. :7346 Max. :4.36e+08 Max. :7346   
##

As an illustration, the returns of the adjusted closing price are calculated below. The function hist allows exploring the distribution.

return <- nifty$Adj.Close[-1]/nifty$Adj.Close[-1828] - 1  
hist(return, main="NIFTY 500 Adjusted Close", xlab="Daily returns", breaks=20)

