# [DATA-01] Data science with R

*Miguel-Angel Canela, IESE Business School*

*September 18, 2016*

## What is data mining?

**Data mining** is a generic expression which applies to a heterogeneous set of methods, used to extract information from large data sets. The expression is understood as *mining knowledge from data*. Data mining methods are very popular nowadays, not because there are a real innovation, since most of the methodology has been available for years, but owing to the explosion in the amount of data at hand (the era of big data). The typical applications in management are usually related to Customer Relationship Management (CRM): market basket analysis, churn modelling, credit scoring, etc.

Data mining was born in the computer science field, as the analytic step of the so called **knowledge discovery in databases** (KDD). Although the acronym KDD is still used by organizations and conferences, it is mostly unknown in the business world. In the opposite direction, the expression **data scientist** is trending these days in job descriptions, referred to a mix of data mining proficiency and a background of programming languages and data bases.

**Machine learning** (ML) was born in the (first) golden age of **artificial intelligence** (AI). The objective was the study of systems that could learn from data, but many of the methods were the same as those of data mining. Nowadays, machine learning is also popular in the business world, due to the increasing interest of giants like IBM, Google, Baidu, etc, and artificial intelligence is a hot topic for tech companies.

Data mining textbooks (e.g. Larose, 2005) describe data mining methods that apply to data in structured form, that is, to data sets in tabular form, with rows and columns. The rows correspond to **instances** (also called observations, cases and records), such as individuals, companies or transactions. The columns correspond to **attributes** (also called variables and fields). Typically, the attributes are either **numeric**, as the amount paid in a transaction, or **categorical**, as gender. Nevertheless, there are also methods for dealing with string (text) data and dates. Categorical attributes are frequently managed through **dummies**, or attributes with 1/0 values.

In data mining, the main job is **prediction**, that is, the description of one attribute (Y) in terms of the other attributes (X's). Typically, the predictive model is first developed in a data set called the **training set** and later applied to other data. When a second data set is used to **validate** the model, it is called the **test set**. In the data mining context, the term **regression** applies to the prediction of a numeric attribute, and **classification** to the prediction of a nominal attribute. In an example of regression, we may try to predict the price of a house from a set of attributes of this house. In one of classification, to predict whether a customer is going to quit or not, from his/her demographics plus some measures of customer activity.

## Software for data mining

The user interacts with data mining software in three possible ways: (a) conventional menus, (b) programming code and (c) visual programming, based on flow charts which are a graphical translation of code. This course is based on code, which is what data scientists mostly use. It uses the **R statistical language**, which is, currently, the leading choice of data scientists. Just a few years ago, data mining textbooks, such as Larose (2005), or Nisbet *et al.* (2009), used Clementine (visual programming), Statistica (menus) and SAS Enterprise Miner (menus). But, nowadays, all books are based on R, **Python** or (less frequently) **Java**, and the examples include code.

This section contains brief comments about other choices, at the personal computer level, different from the one made in this course. Among the old stat packages, the main contenders are:

- SPSS (Statistical Package for the Social Sciences) is one of the oldest statistical packages. In 1998, SPSS Inc. adquired Clementine, the first data mining tool using visual programming in a graphical interface. In 2009, SPSS Inc. was adquired by IBM, which now offers updated versions of SPSS and Clementine under the names of SPSS Statistics and **SPSS Modeler**, respectively.
- SAS (Statistical Analysis System) is also a veteran among the statistical packages. It has modular structure, with so many modules that is difficult to keep track. One of the modules is **SAS Enterprise Miner**, which provides a pack of cases, explained with detail.
- **Statistica** is also a statistical package, not as old as SPSS and SAS, which has been increasingly emphasizing data mining resources. As SAS, it is modular, with a lot of modules.

There is also plenty of choice among the data mining suites:

- **Weka** (Waikato Environment for Knowledge Analysis) is the best-known open-source data mining environment (see Witten *et al.*, 2011, for details). It is written in Java (versions for Windows and Macintosh). Advanced users can access its components through Java programming or through a command-line interface. For the rest of us, Weka provides a **graphical user interface** (GUI), the **Weka Explorer**, in which the user can test different models. Weka's community has also developed a set of extensions covering diverse areas, such as text mining, visualization, bioinformatics, and grid computing. Like R in statistics, Weka has been for years the reference in the machine learning community, attracting a number of users and developers. Some of the Weka algorithms can be called from R, through the package **RWeka**.
- **Rapid Miner**, formerly YALE (Yet Another Learning Environment), has been, until recently, the most popular environment for data mining. It is written in Java and can extend its library of algorithms with those provided by R and Weka. It uses visual programming, but the programs are accessible as XML files. Rapid Miner is commercial, but a starter version can be downloaded for free.
- **KNIME** (Konstanz Information Miner) is a nicely designed data mining suite which runs inside the Eclipse development environment, similar to that of Rapid Miner, but better, to my taste. It is also written in Java and can also extend its library of built-in algorithms with those provided by Weka and R.
- **scikit-learn** may be the choice for those who speak Python, a popular language which is gaining ground among data scientists. It comes with excellent on-line documentation (for pythonistas).

# 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. As a rule, I do not use one of these packages unless it makes a real difference.

There are many books for learning about R, but some of them are too statistically-oriented, so they may not be for you. A popular choice, with a broad perspective, is Teetor (2011). Also popular, and a bit more ambitious, is Adler (2012).

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 produced 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 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** (string), **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
```

Some comments on these examples:

- 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.

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 comes from defining the operations that we 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 of the definition of a function follows. More complex examples will appear later in this course.

```
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. As in any programming language, the equal sign is read as `<-`.

# 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 this with the following code, which imports a csv file containing daily OCHL (Open/Close/High/Low) data for ICICI Bank, from 2003-01-01 to 2015-12-31, extracted from Yahoo Finance India.

```
url1 <- "http://real-chart.finance.yahoo.com/table.csv"
url2 <- "?s=ICICIBANK.NS&a=00&b=01&c=2003&d=11&e=31&f=2015"
url <- paste(url1, url2, sep="")
icici <- read.csv(url, stringsAsFactors=F)
```

Note that I have formed the URL pasting two strings. This is not needed, but breaking code into pieces help to avoid mistakes and simplifies updating the code. The `sep` argument specifies that the *glue* to paste the two parts is the empty string (the default is a blank). I perform some checks on the data frame.

```
dim(icici)
```

```
## [1] 3304    7
```

```
head(icici)
```

```
##         Date   Open   High    Low  Close   Volume Adj.Close
## 1 2015-12-31 262.10 263.85 260.15 261.35  7769400    256.08
## 2 2015-12-30 263.25 264.45 261.75 262.35  9114100    257.06
## 3 2015-12-29 264.00 266.50 261.50 264.75  8231500    259.41
## 4 2015-12-28 260.05 265.00 260.00 264.05 10947000    258.73
## 5 2015-12-25 257.95 257.95 257.95 257.95        0    252.75
## 6 2015-12-24 263.00 263.60 257.30 257.95  5543000    252.75
```

```
tail(icici)
```

```
##            Date   Open   High    Low  Close  Volume Adj.Close
## 3299 2003-01-08 133.90 134.55 131.75 133.10 2894400      7.54
## 3300 2003-01-07 134.70 135.50 131.65 132.65 2248300      7.52
## 3301 2003-01-06 137.90 137.90 132.50 133.20 2311100      7.55
## 3302 2003-01-03 139.90 139.90 136.30 136.75 1029700      7.75
## 3303 2003-01-02 141.00 141.80 138.55 138.95 1604200      7.88
## 3304 2003-01-01 141.45 142.00 139.95 140.40  818400      7.96
```

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(icici)
```

```
## 'data.frame':    3304 obs. of  7 variables:
##  $ Date     : chr  "2015-12-31" "2015-12-30" "2015-12-29" "2015-12-28" ...
##  $ Open     : num  262 263 264 260 258 ...
##  $ High     : num  264 264 266 265 258 ...
##  $ Low      : num  260 262 262 260 258 ...
##  $ Close    : num  261 262 265 264 258 ...
##  $ Volume   : int  7769400 9114100 8231500 10947000 0 5543000 10485400 19402600 13007000 1313
7000 ...
##  $ Adj.Close: num  256 257 259 259 253 ...
```

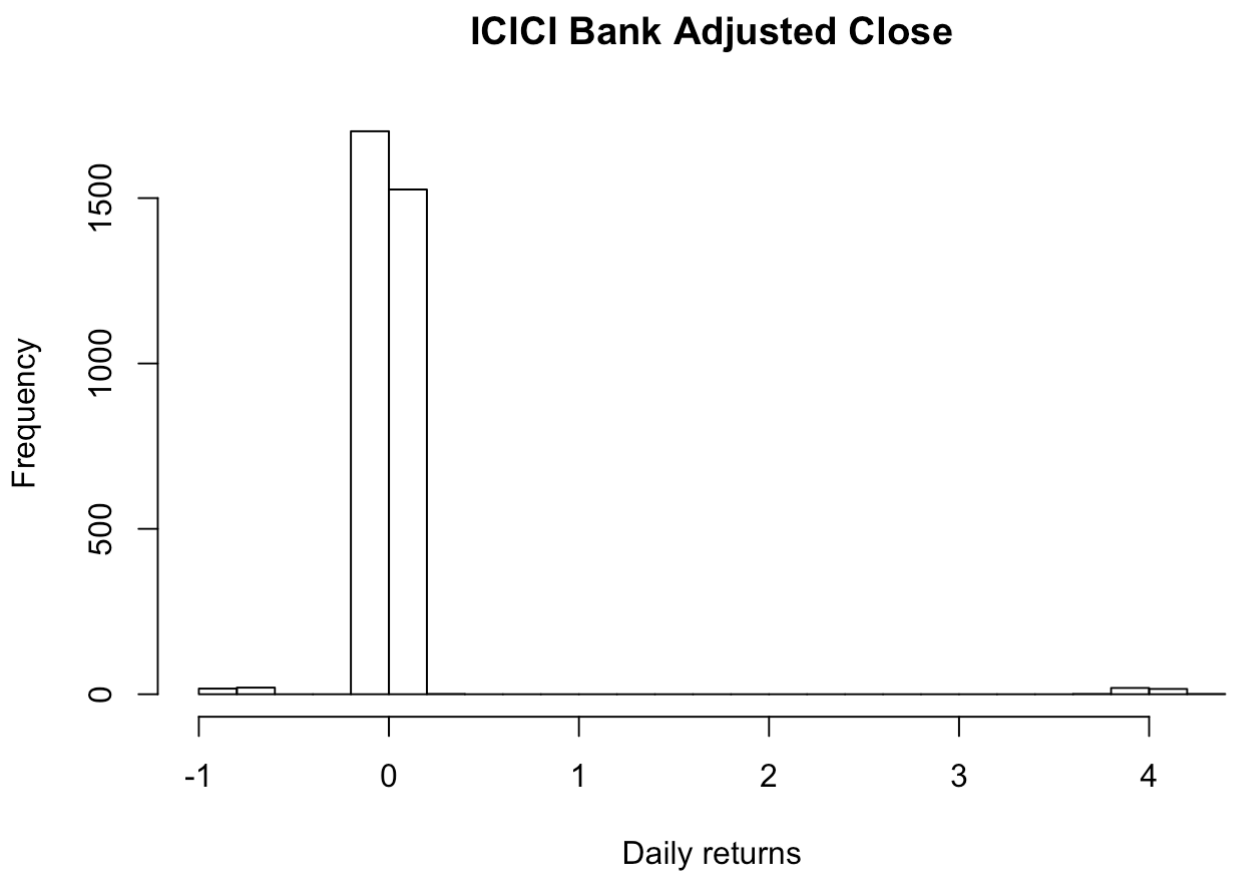
The funtion `summary` works in different ways in objects of different nature. For numeric variables, it produces some summary statistics.

```
summary(icici)
```

```
##      Date               Open             High              Low         
##  Length:3304        Min.   :  27.9   Min.   :  28.05   Min.   :  27.7  
##  Class :character   1st Qu.: 357.4   1st Qu.: 361.99   1st Qu.: 349.8  
##  Mode  :character   Median : 801.0   Median : 823.35   Median : 785.0  
##                     Mean   : 725.0   Mean   : 736.57   Mean   : 712.0  
##                     3rd Qu.:1003.2   3rd Qu.:1020.00   3rd Qu.: 990.0  
##                     Max.   :1767.0   Max.   :1779.00   Max.   :1740.0  
##      Close            Volume            Adj.Close      
##  Min.   :  27.92   Min.   :        0   Min.   :  1.58  
##  1st Qu.: 354.02   1st Qu.:  5826675   1st Qu.: 45.40  
##  Median : 801.58   Median : 12523450   Median : 99.65  
##  Mean   : 724.24   Mean   : 17128310   Mean   :113.85  
##  3rd Qu.:1003.71   3rd Qu.: 20965025   3rd Qu.:151.61  
##  Max.   :1773.30   Max.   :252047400   Max.   :369.94
```

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

```
return <- icici$Adj.Close[-1]/icici$Adj.Close[-3304] - 1
hist(return, main="ICICI Bank Adjusted Close", xlab="Daily returns", breaks=20)
```




# References

1. J Adler (2012), *R in a Nutshell*, O'Reilly.
2. DT Larose (2005), *Discovering Knowledge in Data*, Wiley.
3. V Mayer-Schönberger & K Cukier (2012), *Big Data*, John Murray.
4. R Nisbet, J Elder & G Miner (2009), *Handbook of Statistical Analysis & Data Mining*, Academic Press.
5. F Provost & T Fawcett (2013), *Data Science for Business — What You Need to Know About Data Mining and Data-Analytic Thinking*, O'Reilly.
6. P Teetor (2011), *R Cookbook*, O'Reilly.
7. IH Witten, E Frank & MA Hall (2011), *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann.