Data science with R

Miguel-Angel Canela, IESE Business School

October 30, 2017

Contents

Data Science and Machine Learning

The expression **data scientist** is trending these days in job descriptions, referred to a mix of data analysis skills and a background of programming languages and databases. But it is, somewhat, a new name for an old job. 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) and the technology for processing these data (cloud computing), data science is hot now, not only in tech companies.

An ancestor of Data Science is **Data Mining**, 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 was born in the computer science field. The typical applications in management are related to Customer Relationship Management (CRM): market basket analysis, churn modelling, credit scoring, etc.

Also closely related to Data Science is **Machine Learning** (ML), born in the golden age of **Artificial Intelligence** (AI). The objective of Machine Learning 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 popular in the business world, due to the increasing interest of giants like IBM, Google, Baidu, etc, and Artificial Intelligence is a hot topic in tech companies, where **algorithms** are taken today as valuable assets.

Data Mining textbooks (e.g. Larose, 2005) describe 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 **variables** (also called attributes and fields). Typically, the variables are either **numeric**, as the amount paid in a transaction, or **categorical** (also called nominal), as gender. Nevertheless, there are also methods for dealing with string (text) data and dates. Categorical variables are frequently managed through **dummies**, or attributes with 1/0 values.

Data Science software

The user interacts with the Data Science software applications 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. More specifically, 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), were using visual programming or menus in their examples, but, nowadays, almost all the Data Science books are based on `R` or **Python**, and the examples include code.

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. So, tabular data are managed as data frames in the actual data management technology.

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 column of a data frame is identified as `dataframe$variable`, as we see in the following simple example.

```
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
```

Subsetting

Data frames can be subsetted in an easy way. Some examples follow.

```
df[1:3, 1:2]
```

```
##   v1 v2
## 1  1 10
## 2  2  9
## 3  3  8
```

```
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
```

Importing and exporting data sets

Data sets in tabular form can be imported to `R` as data frames from many formats. 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 can lead to errors with text data. 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 `dataset <- 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. To export a data frame to a csv file, we use the reverse function, `write.csv`. The syntax is `write.csv(dataset, file=filename)`. Again, the file name, supplied by the user, includes the path.

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`.

Example: Employees leaving prematurely

The data set for this example contains data on employee turnover. It has 14,999 rows and 10 columns:

- The employee satisfaction level, in the 0-1 range (`satisfaction`).
- The last evaluation by his/her superior, in the 0-1 range (`eval`).
- The number of projects in which the employee has participated (`projects`).
- The average monthly hours worked (`hours`).
- The time spent at the company, in years (`time`).
- A dummy indicating whether the employee has had a work accident (`accident`).
- A dummy indicating whether the employee has had a promotion in the last 5 years (`promotion`).
- The actual department to which the employee is assigned, with 10 values (`dept`).
- The actual salary, with three levels, low, medium and high (`salary`).
- A dummy indicating whether the employee has left the company (`left`).

We import the file with the function `read.csv`,

```
turnover <- read.csv(file="turnover.csv")
```

Note that, where I have written `"turnover.csv"`, you have to write the complete path of this file in your computer, for the file to be found by `R`. As I am writing it, my code will work only if the file is in the **working**

directory. You can learn where the working directory by means of the function `getwd`, and you can change it with the function `setwd`.

```
getwd()
```

```
## [1] "C:/Users/mcanela/Dropbox (Personal)/DATA-2018-1/[DATA-02] Data science with R"
```

I perform some checks on the data frame. What the function `dim`, `head` and `tail` do is obvious. With the last two, there is an second argument that controls the number of rows returned. The default is 6.

```
dim(turnover)
```

```
## [1] 14999    10
```

```
head(turnover)
```

```
##   satisfaction eval projects hours time accident left promotion  dept
## 1         0.38 0.53        2   157    3        0    1         0 sales
## 2         0.80 0.86        5   262    6        0    1         0 sales
## 3         0.11 0.88        7   272    4        0    1         0 sales
## 4         0.72 0.87        5   223    5        0    1         0 sales
## 5         0.37 0.52        2   159    3        0    1         0 sales
## 6         0.41 0.50        2   153    3        0    1         0 sales
##   salary
## 1    low
## 2 medium
## 3 medium
## 4    low
## 5    low
## 6    low
```

```
tail(turnover)
```

```
##       satisfaction eval projects hours time accident left promotion
## 14994         0.76 0.83        6   293    6        0    1         0
## 14995         0.40 0.57        2   151    3        0    1         0
## 14996         0.37 0.48        2   160    3        0    1         0
## 14997         0.37 0.53        2   143    3        0    1         0
## 14998         0.11 0.96        6   280    4        0    1         0
## 14999         0.37 0.52        2   158    3        0    1         0
##          dept salary
## 14994 support    low
## 14995 support    low
## 14996 support    low
## 14997 support    low
## 14998 support    low
## 14999 support    low
```

Also, the structure of an `R` object can be explored with the function `str`. Actually, `turnover` is a data frame, with 14,999 and 10 columns. Note that `dept` and `salary`, which come as string data in the csv file, have been imported as factors. We will discuss this later in this course.

```
str(turnover)
```

```
## 'data.frame':    14999 obs. of  10 variables:
##  $ satisfaction: num  0.38 0.8 0.11 0.72 0.37 0.41 0.1 0.92 0.89 0.42 ...
##  $ eval        : num  0.53 0.86 0.88 0.87 0.52 0.5 0.77 0.85 1 0.53 ...
##  $ projects    : int  2 5 7 5 2 2 6 5 5 2 ...
##  $ hours       : int  157 262 272 223 159 153 247 259 224 142 ...
##  $ time        : int  3 6 4 5 3 3 4 5 5 3 ...
##  $ accident    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ left        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ promotion   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ dept        : Factor w/ 10 levels "accounting","hr",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ salary      : Factor w/ 3 levels "high","low","medium": 2 3 3 2 2 2 2 2 2 2 ...
```

Summarizing

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

```
summary(turnover)
```

```
##   satisfaction         eval           projects         hours      
##  Min.   :0.0900   Min.   :0.3600   Min.   :2.000   Min.   : 96.0  
##  1st Qu.:0.4400   1st Qu.:0.5600   1st Qu.:3.000   1st Qu.:156.0  
##  Median :0.6400   Median :0.7200   Median :4.000   Median :200.0  
##  Mean   :0.6128   Mean   :0.7161   Mean   :3.803   Mean   :201.1  
##  3rd Qu.:0.8200   3rd Qu.:0.8700   3rd Qu.:5.000   3rd Qu.:245.0  
##  Max.   :1.0000   Max.   :1.0000   Max.   :7.000   Max.   :310.0  
##                                                                   
##       time           accident           left          promotion      
##  Min.   : 2.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000  
##  1st Qu.: 3.000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000  
##  Median : 3.000   Median :0.0000   Median :0.0000   Median :0.00000  
##  Mean   : 3.498   Mean   :0.1446   Mean   :0.2381   Mean   :0.02127  
##  3rd Qu.: 4.000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000  
##  Max.   :10.000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000  
##                                                                      
##           dept         salary    
##  sales      :4140   high  :1237  
##  technical  :2720   low   :7316  
##  support    :2229   medium:6446  
##  IT         :1227                
##  product_mng: 902                
##  marketing  : 858                
##  (Other)    :2923                
```

With one argument, the function `table` counts the observations for the values of a categorical variable.

```
table(turnover$dept)
```

```
## 
##  accounting          hr          IT  management   marketing product_mng 
##         767         739        1227         630         858         902 
##       RandD       sales     support   technical 
##         787        4140        2229        2720 
```

As any other vector, the oputput of table can come sorted.

```
sort(table(turnover$dept), decreasing=T)
```

```
## 
##       sales   technical     support          IT product_mng   marketing 
##        4140        2720        2229        1227         902         858 
##       RandD  accounting          hr  management
##         787         767         739         630
```

With two arguments, the function `table` is used for cross tabulation. Note that the first argument identifies the variable that come in then rows.

```
table(turnover$dept, turnover$salary)
```

```
##
##               high  low medium
##   accounting    74  358    335
##   hr            45  335    359
##   IT            83  609    535
##   management   225  180    225
##   marketing     80  402    376
##   product_mng   68  451    383
##   RandD         51  364    372
##   sales        269 2099   1772
##   support      141 1146    942
##   technical    201 1372   1147
```

The function `tapply` is typically used to summarize a variable by group. The first argument is the varaible that we wish to summarize, the second argument is the grouping variable and the third one the summary statistic. In this case, since `left` is a dummy, the mean is equal to the proportion of ones, that is, to the actual turnover rate. Note that, although we use this function for statistical summaries, it can be applied as far as the function in the third argument makes sense for the vector in the first argument.

```
tapply(turnover$left, turnover$salary, mean)
```

```
##       high        low     medium
## 0.06628941 0.29688354 0.20431275
```

Some people prefer a graphical version of such tables.

```
barplot(tapply(turnover$left, turnover$salary, mean),
  main="Figure 1. Barplot")
```

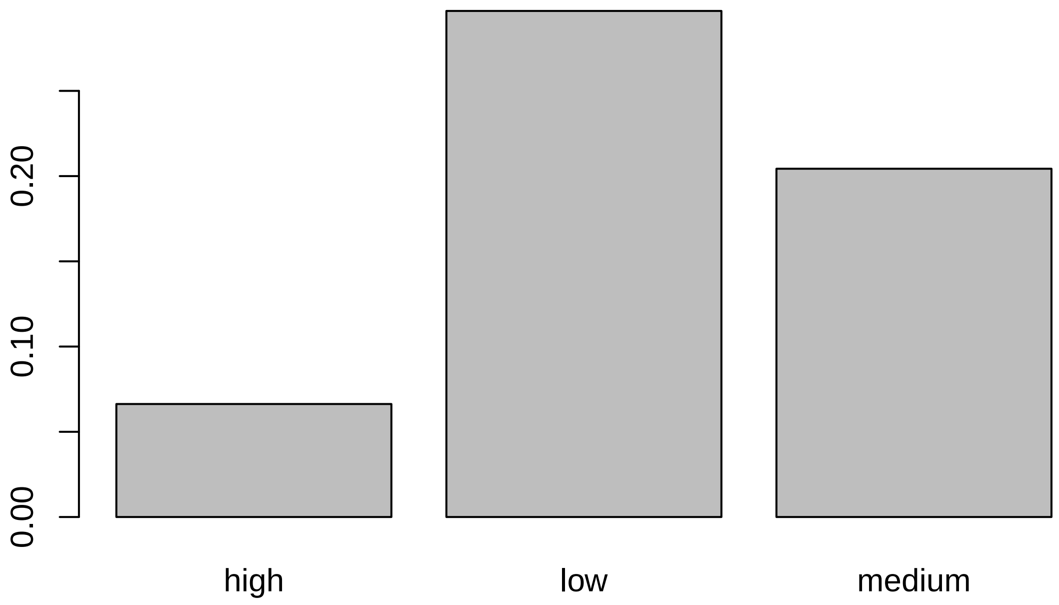


Figure 1. Barplot

More examples follow.

```
tapply(turnover$left, turnover$dept, mean)
```

```
##  accounting          hr          IT  management   marketing product_mng
##   0.2659713   0.2909337   0.2224939   0.1444444   0.2365967   0.2195122
##       RandD       sales     support   technical
##   0.1537484   0.2449275   0.2489906   0.2562500
```

```
sort(tapply(turnover$left, turnover$dept, mean), decreasing=T)
```

```
##          hr  accounting   technical     support       sales   marketing
##   0.2909337   0.2659713   0.2562500   0.2489906   0.2449275   0.2365967
##          IT product_mng       RandD  management
##   0.2224939   0.2195122   0.1537484   0.1444444
```

Correlation

With two arguments, the function `cor` applies to a pair of vectors, and its value is a number. With one argument, applies to a data frame, and produeces a correlation matrix.

```
cor(turnover[, 1:8])
```

```
##              satisfaction         eval     projects        hours
## satisfaction   1.00000000  0.105021214 -0.142969586 -0.020048113
## eval           0.10502121  1.000000000  0.349332589  0.339741800
## projects      -0.14296959  0.349332589  1.000000000  0.417210634
## hours         -0.02004811  0.339741800  0.417210634  1.000000000
## time          -0.10086607  0.131590722  0.196785891  0.127754910
## accident       0.05869724 -0.007104289 -0.004740548 -0.010142888
## left          -0.38837498  0.006567120  0.023787185  0.071287179
## promotion      0.02560519 -0.008683768 -0.006063958 -0.003544414
##                      time     accident        left    promotion
## satisfaction -0.100866073  0.058697241 -0.38837498  0.025605186
## eval          0.131590722 -0.007104289  0.00656712 -0.008683768
## projects      0.196785891 -0.004740548  0.02378719 -0.006063958
## hours         0.127754910 -0.010142888  0.07128718 -0.003544414
## time          1.000000000  0.002120418  0.14482217  0.067432925
## accident      0.002120418  1.000000000 -0.15462163  0.039245435
## left          0.144822175 -0.154621634  1.00000000 -0.061788107
## promotion     0.067432925  0.039245435 -0.06178811  1.000000000
```

The default of `cor` gives too many decimals. we can round to two decimals with the function `round`.

```
round(cor(turnover[, 1:8]), 2)
```

```
##              satisfaction  eval projects hours  time accident  left
## satisfaction         1.00  0.11    -0.14 -0.02 -0.10     0.06 -0.39
## eval                 0.11  1.00     0.35  0.34  0.13    -0.01  0.01
## projects            -0.14  0.35     1.00  0.42  0.20     0.00  0.02
## hours               -0.02  0.34     0.42  1.00  0.13    -0.01  0.07
## time                -0.10  0.13     0.20  0.13  1.00     0.00  0.14
## accident             0.06 -0.01     0.00 -0.01  0.00     1.00 -0.15
## left                -0.39  0.01     0.02  0.07  0.14    -0.15  1.00
## promotion            0.03 -0.01    -0.01  0.00  0.07     0.04 -0.06
##              promotion
## satisfaction      0.03
## eval             -0.01
## projects         -0.01
## hours             0.00
## time              0.07
## accident          0.04
## left             -0.06
## promotion         1.00
```

Plotting

Histograms allow a quick glance at the distribution of a numeric varioable.

```
hist(turnover$eval,
  main="Figure 2. Histogram",
  xlab="Evaluation")
```

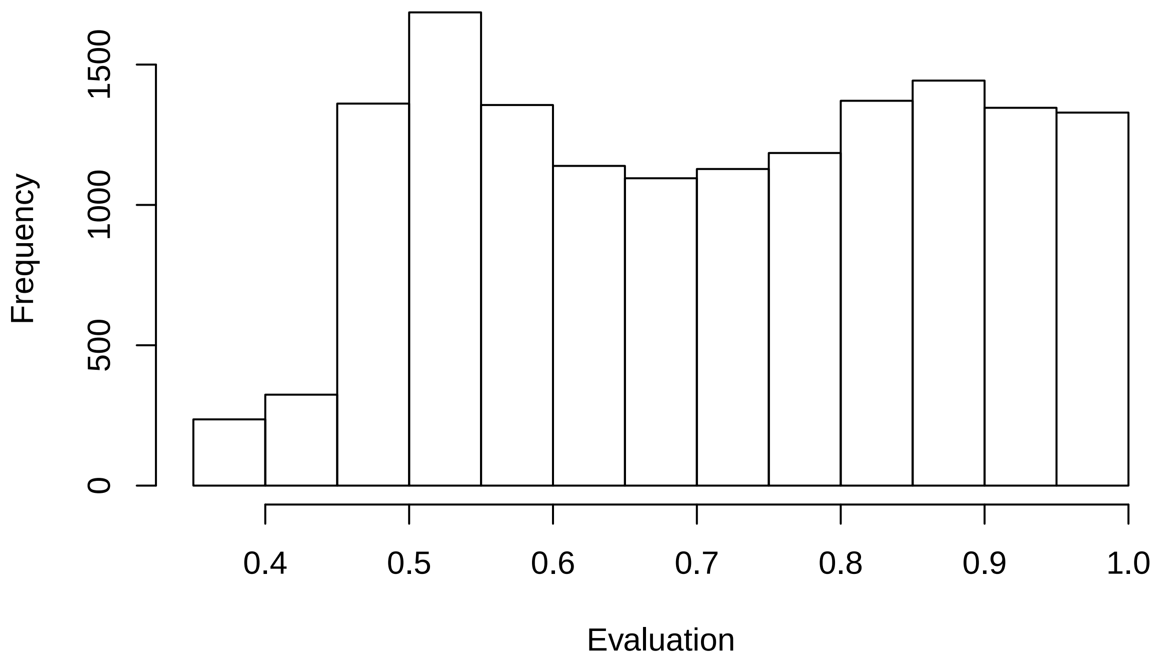
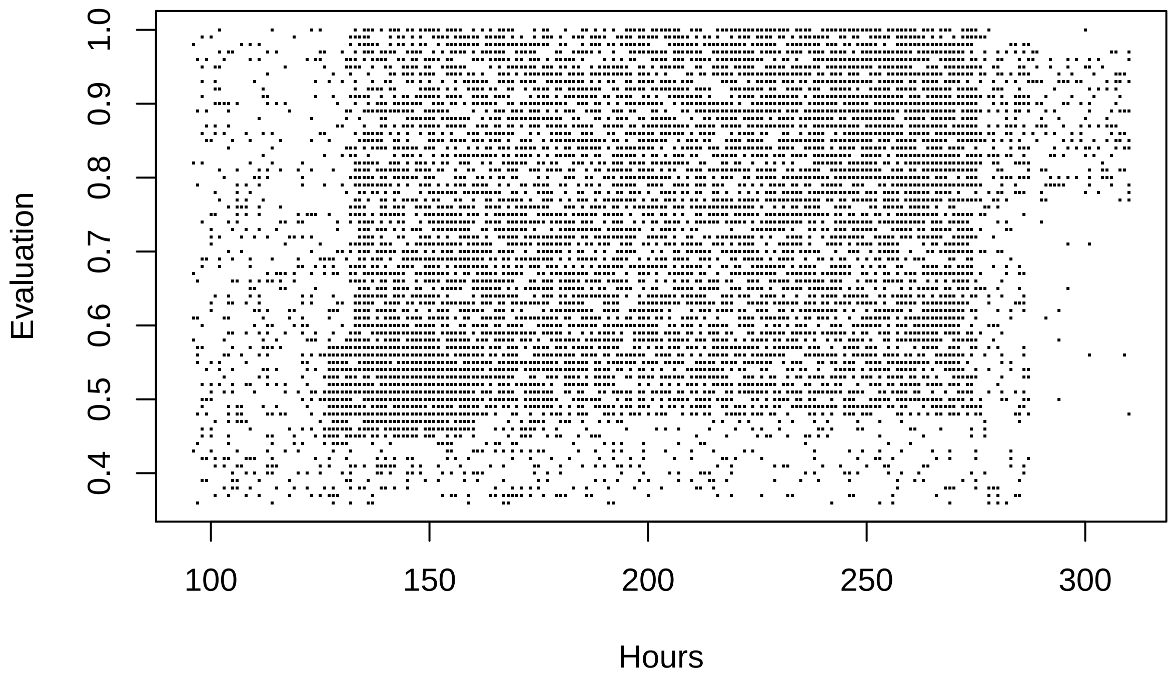


Figure 2. Histogram

Scatterplots are very simple in `R` and allow for a lot of customization. The syntax is `plot(y ~ x)`. If the two vectors are extracted from the same data frame, is practical to use the argument `data`. The argument `pch` controls the dots used in the plot. `pch=16` and `pch=20` produce black circles of different size (my choice). With `pch="."`, we get a dot with the same size as in text, adequate when the sample big gets big.

```r
plot(eval ~ hours, data=turnover, pch=".",
  main="Figure 3. Scatterplot",
  xlab="Hours", ylab="Evaluation")
```

Figure 3. Scatterplot



Homework

The file `tata.csv` contains daily OCHL (Open/Close/High/Low) data for Tata Steel, from the National Stock Exchange in Mumbai, extracted from Yahoo Finance India. The dates go from 2003-01-01 to 2015-12-31.

- Summarize the adjusted prices.
- Calculate the daily returns and the logarithmic returns for the adjusted prices and compare them. Are the differences relevant?
- Plot a histogram of the returns. Can a normal distribution be used a model here?
- Produce a line plot for the returns, with the syntax `plot(returns, type="l")`.

References

1. E Alpaydin (2016), *Machine Learning*, MIT Press.
2. MJ Crawley (2012), *The R Book*, Wiley. Free access at ‘ftp://ftp.tuebingen.mpg.de/pub/kyb/bresciani/Crawley%20-%20The%20R%20Book.pdf
3. RI Kabacoff (2010), *R in Action*, Manning. Free access at `http://kek.ksu.ru/eos/DataMining/1379968983.pdf`.
4. DT Larose (2005), *Discovering Knowledge in Data*, Wiley.
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. H Wickham & G Grolemund (2016), *R for Data Science*, O’Reilly.