Master the basics of R Programming

Introduction

R is a programming language created and developed in 1991 by two statisticians at the University of Auckland, in New Zealand. It officially became free and open-source only in 1995. For its origins, it provides statistical and graphical techniques, linear and non-linear models, techniques for time series, and many other functionalities. Even if Python is the most common in the Data Science field, R is still widely used for specialized purposes, like in financial companies, research, and healthcare.

Assignment

When we program in R, the entities we work with are called objects [1]. They can be numbers, strings, vectors, matrices, arrays, functions. So, any generic data structure is an object. The assignment operator is `<-`, which combines the characters `<` and `-`. We can visualize the output of the object by calling it:

```
# Assignment
x <- 23
```

A more complex example can be:

```
# A more complex example
x <- 1/1+1*1
y <- x^4
z <- sqrt(y)
x
```

```
## [1] 2
```

```
y
```

```
## [1] 16
```

```
z
```

```
## [1] 4
```

As you can notice, the mathematical operators are the ones you use for the calculator on the computer, so you don't need the effort to remember them. There are also mathematical functions available, like `sqrt`, `abs`, `sin`, `cos`, `tan`, `exp`, and `log`.

Vectors in R Programming

In R, the vectors constitute the simplest data structure. The elements within the vector are all of the same types. To create a vector, we only need the function `c()`:

```
# Create vector
v1 <- c(2,4,6,8)
v1
```

```
## [1] 2 4 6 8
```

This function simply concatenates different entities into a vector. There are other ways to create a vector, depending on the purpose. For example, we can be interested in creating a list of consecutive numbers and we don't want to specify them manually. In this case, the syntax is `a:b` , where a and b correspond to the lower and upper extremes of this succession. The same result can be obtained using the function `seq()`

```
# Creating a list of consecutive numbers
1:7
```

```
## [1] 1 2 3 4 5 6 7
```

The function `seq()` can also be applied to create more complex sequences. For example, we can add the argument by the step size and the length of the sequence:

```
# Create list by step size
v4 <- seq(0,1,by=0.1)
v4
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
# Create list by the length of the sequence
v5 <- seq(0,2,len=11)
v5
```

```
##  [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

To repeat the same number more times into a vector, the function rep() can be used:

```
# Repeat the same number more times into a vector
v6 <- rep(2,3)
v6
```

```
## [1] 2 2 2
```

```
v7 <- c(1,rep(2,3),3)
v7
```

```
## [1] 1 2 2 2 3
```

There are not only numerical vectors. There are also logical vectors and character vectors:

```r
# Logical vector
x <- 1:10
y <- 1:5
l <- x==y
l
```

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

```r
# Character vector
c <- c('a','b','c')
c
```

```
## [1] "a" "b" "c"
```

factors in R Programming

factors are specialized vectors used to group elements into categories. There are two types of factors: ordered and unordered. For example, we have the countries of five friends. We can create a factor using the function `factor()`

```r
# Create a factor
states <- c('italy','france','germany','germany','germany')
statesf <- factor(states)
statesf
```

```
## [1] italy   france  germany germany germany
## Levels: france germany italy
```

To check the levels of the factor, the function `levels()` can be applied.

```r
# Check the levels of the factor
levels(statesf)
```

```
## [1] "france"  "germany" "italy"
```

Matrices in R Programming

As you probably know, the matrix is a 2-dimensional array of numbers. It can be built using the function `matrix()`

```r
# Creating a matrix
m1 <- matrix(1:6,nrow=3)
m1
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```r
m2 <- matrix(1:6,ncol=3)
m2
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

It can also be interesting combine different vectors into a matrix row-wise or column-wise. This is possible with `rbind()` and `cbind()`:

```r
# Combining vectors into matrix using rbind()
countries <- c('italy','france','germany')
age <- 25:27
rbind(countries,age)
```

```
##           [,1]     [,2]     [,3]
## countries "italy" "france" "germany"
## age       "25"    "26"     "27"
```

```r
# Or using cbind()
cbind(countries,age)
```

```
##      countries age
## [1,] "italy"   "25"
## [2,] "france"  "26"
## [3,] "germany" "27"
```

Arrays in R Programming

Arrays are objects that can have one, two, or more dimensions. When the array is one-dimensional, it coincides with the vector. In the case it's 2D, it's like to use the matrix function. In other words, arrays are useful to build a data structure with more than 2 dimensions.

```r
# Creating an array
a <- array(1:16,dim=c(6,3,2))
a
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    7   13
## [2,]    2    8   14
## [3,]    3    9   15
## [4,]    4   10   16
## [5,]    5   11    1
## [6,]    6   12    2
##
## , , 2
##
##      [,1] [,2] [,3]
```

```
## [1,]    3    9   15
## [2,]    4   10   16
## [3,]    5   11    1
## [4,]    6   12    2
## [5,]    7   13    3
## [6,]    8   14    4
```

list

The list is a ordered collection of objects. For example, it can a collection of vectors, matrices. Differently from vectors, the lists can contain values of different type. They can be build using the function `list()`:

```
# Creating a list
x <- 1:3
y <- c('a','b','c')
l <- list(x,y)
l
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Data frames in R Programming

A data frame is very similar to a matrix. It's composed of rows and columns, where the columns are considered vectors. The most relevant difference is that it's easier to filter and select elements. We can build manually the data frame using the function `data.frame()`:

```
# Data frame
countries <- c('italy','france','germany')
age <- 25:27
df <- data.frame(countries,age)
df
```

```
##   countries age
## 1     italy  25
## 2    france  26
## 3   germany  27
```

An alternative is to read the content of a file and assign it to a data frame with the function `read.table()`:

```
# read.table() function
df <- read.table('titanic.dat')
```

Like in Pandas, there are other functions to read files with different formats. For example, let's read a csv file:

```
# read.csv() function
df <- read.csv('Data/titanic.csv')
head(df)
```

```
##   PassengerId Survived Pclass
## 1           1        0      3
## 2           2        1      1
## 3           3        1      3
## 4           4        1      1
## 5           5        0      3
## 6           6        0      3
##                                                 Name    Sex Age SibSp Parch
## 1                            Braund, Mr. Owen Harris   male  22     1     0
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1     0
## 3                             Heikkinen, Miss. Laina female  26     0     0
## 4       Futrelle, Mrs. Jacques Heath (Lily May Peel) female  35     1     0
## 5                           Allen, Mr. William Henry   male  35     0     0
## 6                                   Moran, Mr. James   male  NA     0     0
##             Ticket    Fare Cabin Embarked
## 1        A/5 21171  7.2500              S
## 2         PC 17599 71.2833   C85        C
## 3 STON/O2. 3101282  7.9250              S
## 4           113803 53.1000  C123        S
## 5           373450  8.0500              S
## 6           330877  8.4583              Q
```

Like in Python, R provides pre-loaded data using the function `data()`:

```
# Load pre-loaded data
data("mtcars")
head(mtcars)
```

```
##                    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
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

The function `head()` allows visualizing the first 6 rows of the mtcars dataset, which provides the data regarding fuel consumption and ten characteristics of 32 automobiles.

To check all the information about the dataset, you write this line of code:

```
# This code not evaluated
help(mtcars)
```

In this way, a window with all the useful information will open. To have an overview of the dataset's structure, the function `str()` can allow having additional insights into the data:

```r
# Structure of the data
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

From the output, it's clear that there are 32 observations and 11 variables/columns. From the second line, there is a row for each variable that shows the type and the content. We show separately the same information using:

- the function `dim()` to look at the dimensions of the data frame
- the function `names()` to see the names of the variables

```r
# Dimensions of the data frame
dim(mtcars)
```

```
## [1] 32 11
```

```r
# Names of the variables
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

The summary statistics of the variables can be obtained through the function `summary()`.

```r
# Summary of the data
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
```

```
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

We can access specific columns using the expression `namedataset$namevariable`. If we want to avoid specifying every time the name of the dataset, we need the function `attach()`.

```
# Using $ sign expression
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
# Using attach() function
attach(mtcars)
mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

In this way, we attach the data frame to the search path, allowing to refer to the columns with only their names. Once we attached the data frame and we aren't interested anymore to use it, we can do the inverse operation using the function `detach()`.

We can also try to select the first row in the data frame using this syntax:

```
# Select the first row
mtcars[1,]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

Note that the index starts from 1, not from 0! If we want to extract the first columns, it can be done in this way:

```
# Select the first column
mtcars[,1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

We can also try to filter the rows using a logical expression:

```
# Filter with logical expression
mtcars[mpg>20,]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

we can also specify the column while we filter:

```
# Specify the column while filter
mtcars[mpg>20, 'mpg']
```

```
##  [1] 21.0 21.0 22.8 21.4 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
```