Instituto Superior de Engenharia de Lisboa
Mestrado em Engenharia Informática e de Computadores
Mestrado em Engenharia Informática e Multimédia
Big data mining (MDLE)

Laboratory Class #1 — R language and environment setup

2nd semester, 2023/2024 (February, 28th)

Code and Report are due by March, 06th

---

1. **Data Resources and Software Tools.**
   For this laboratory class, you will need the following software:

   - Access to RServer (`http://datalys.dyn.fil.isel.pt:8787/http://10.62.73.53:8787`) or,
   - R, https://cran.r-project.org/, and
   - R Studio, https://posit.co/downloads/.

   As a preparation to the lab, you should:

   - download the Sparklyr and RStudio IDE cheat sheets from https://posit.co/resources/cheatsheets/.

---

2. **R Primer**.

   (a) Go to http://www.r-tutor.com/r-introduction and read the basic data types section.

   (b) Open R Studio, execute and explain each of the following statements, one by one (and check the values on the 'Global Environment' tab). Beware with the cut+paste:

   i. Scalars and Operators.

```
var1 <- 3 Allocate the number 3 to the variable var1, var1 stores 3
show(var1) Print var1 to the console: "[1] 3"
var2 <- var1 * var1 Allocate the square of var1 to var2, var2 stores 9
var3 <- var1 ** 2 Allocate the square of var1 to var3, var3 stores 9
var4 <- var1 ^ 2 Allocate the square of var1 to var4, var4 stores 9
var1 < var1 Assign var1 to itself, var1 stores 3
var3 != var4 Check if var3 is not equal to var4: "[1] FALSE"
var2 == var4 Check if var2 is equal to var4: "[1] TRUE"
var2 <- var2 - var2 Subtract var2 from itself and assign the result to var2, var2 stores 0
var5 <- var3 / var2 Divide var3 by var2 and assign the result to var5, var5 stores Inf (infinite)
var5 + 1 Add 1 to var5, var5 stores Inf
```

   ii. Manipulating vectors.

```
a <- c(1,2,5.3,6,-2,4,3.14159265359) Allocates the list of numeric values to a, a stores num [1:7] 1 2 5.3 6 -2...
a  Prints a values to the console: "[1] 1.000000  2.000000  5.300000  6.000000 -2.000000  4.000000  3.141593"
b <- c("1","2","3") Allocates the list of values to b, b stores char [1:3] "1" "2" "3"
"2" %in% b Checks if "2" is in b: "[1] TRUE"
"5" %in% b Checks if "5" is in b: "[1] FALSE"
c <- c(TRUE,TRUE,FALSE,TRUE) Allocates the list of logical values to c, c stores logi [1:4] TRUE TRUE FALSE TRUE
a <- c Replicates the values of c in a, a stores logi [1:4] TRUE TRUE FALSE TRUE
```

```
a[0] Shows the value of a at the position 0: "[1] logical(0)", because R starts indexing lists at the position 1
a[1] Shows the value of a at the position 1: "[1] TRUE"
a[-1] Shows all values of a except the first one which is the absolute number of the index: "[1] TRUE FALSE TRUE"
a[8]  Shows the value of a at the position 8: "[1] NA" (not available), a only has 4 values
a[c(1,3)] Shows the values of the position 1 and 3 of a: "[1] TRUE FALSE"
a[c(3,1)] Shows the values of the position 3 and 1 of a: "[1] FALSE TRUE"
a[a > 2]  Shows all values in a that are bigger than 2: "[1] logical(0)", a only has logical values and not numeric
```

iii. Manipulating matrices.

```
m <- matrix(1:6, nrow=3,ncol=2) Allocates the matrix to m, m stores int[1:3, 1:2] 1, 2, 3, 4, 5, 6
show(m) Shows the values of m, shows:
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
n <- matrix(2:7, nrow=2,ncol=3) Allocates the matrix to n, n stores int[1:2, 1:3] 2, 3, 4, 5, 6, 7
n Shows the values of n, shows:
      [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7
m[,2] Shows the second column of m: "[1] 4 5 6"
n[1,] Shows the first line of n: "[1] 2 4 6"
m[2:3,1:2] Shows the subdivision of m that only contains the lines from 2 to 3 and the columns from 1 to 2, shows:
      [,1] [,2]
[1,]    2    5
[2,]    3    6
n %*% m Shows the matrixical multiplication of n and m,, shows:
      [,1] [,2]
[1,]   28   64
[2,]   34   79
m %*% n Shows the matrixical multiplication of m and n, shows:
      [,1] [,2] [,3]
[1,]   14   24   34
[2,]   19   33   47
[3,]   24   42   60
n %*% n Shows: "Error in n %*% n : non-conformable arguments" *
n^2 Shows the matrix n with all its values squared, shows:
      [,1] [,2] [,3]
[1,]    4   16   36
[2,]    9   25   49
sqrt(n) Shows the square root of the matrix n, shows:
          [,1]     [,2]     [,3]
[1,] 1.414214 2.000000 2.449490
[2,] 1.732051 2.236068 2.645751
```

*For the matrix multiplication of 2 matrix to work requires that the number of columns in the first matrix equals the number of rows in the second matrix.

iv. Manipulating data frames.

```
d <- c(1,2,3,4)  Allocates the list of numeric values to d, d stores num [1:4] 1 2 3 4
e <- c("Bob", "Alice", NA,"Joe") Allocates the list of numeric values to e, e stores char [1:4] "Bob" "Alice" NA "Joe"
f <- c(TRUE,TRUE,FALSE,TRUE)   Allocates the list of logical values to f, f stores logi [1:4] TRUE TRUE FALSE TRUE
my.data <- data.frame(d,e,f,stringsAsFactors=FALSE) Creates the data frame my.data
show(my.data) Shows the values of my.data, shows:
  d     e     f
1 1   Bob  TRUE
2 2 Alice  TRUE
3 3  <NA> FALSE
4 4   Joe  TRUE
names(my.data) <- c("ID","Name","Passed") Name the columns of my.data: d->"ID", e->"Name", f->"Passed"
View(my.data) Invoke a spreadsheet-style data viewer of my.data
my.data$Name Shows a list with the values in column Name: "[1] "Bob" "Alice" NA "Joe""
my.data <- cbind(my.data, Failed=!f) Binds the columns of 2 data frame, my.data and Failed which as the opposite of f list
View(my.data) Invoke a spreadsheet-style data viewer of my.data with the new column(Failed)
my.data <- rbind(my.data, c(5,"Carol",FALSE,TRUE)) Bind the rows of data frame my.data and a new list
View(my.data) Invoke a spreadsheet-style data viewer of my.data with the new row(5 "Carol, FALSE, TRUE)
?data.frame Help from R studio about data frames
```

v. Manipulating factors.

```
colour <- c(rep("red",20), rep("blue", 30)) Creates the list colour with 20 instances of red and 30 of blue
colour <- factor(colour) Replaces the list colour with the factor of colour, in this case is a factor with 2 levels, "blue" and "red"
summary(colour) Shows how many instances are in each level, shows:
blue  red
  30   20
dimensions <- c("large", "medium", "small") Creates the list dimensions with the respective values
show(dimensions) Shows: "[1]  large  medium small"
dimensions <- ordered(dimensions) Creates a factor with 3 levels from dimensions that contains an order
show(dimensions) Shows said order: "3 Levels: large < medium < small"
```

vi. Some useful functions.

```
length(colour) Returns the length of the list colour: "[1] 50"
length(a) Returns the length of the list a: "[1] 4"
class(colour) Returns the values of the class attribute of an R object(colour): "[1] "factor""
class(dimensions) Returns the values of the class attribute of an R object(dimensions): "[1] "ordered" "factor""
class(a)  Returns the values of the class attribute of an R object(a):"[1] "logical""
class(my.data) Returns the values of the class attribute of an R object(my.data): "[1] "data.frame""
nrow(my.data)  Returns the number of rows of the data frame my.data: "[1] 5"
ncol(my.data)  Returns the number of columns of the data frame my.data: "[1] 4"
str(my.data) Displays the internal structure of an R object(my.data)
sessionInfo() Query and print the information of the current session
ls() Returns a vector of character strings giving the names of the objects in the specified environment.
rm(a) Removes the object a
glimpse(my.data) transposed version of print. First we had to install dplyr package( "library(dplyr)"). Doing the command "glimpse(my.data)" it
shows:
Rows: 5
Columns: 4
$ ID     <chr> "1", "2", "3", "4", "5"
$ Name   <chr> "Bob", "Alice", NA, "Joe", "Carol"
$ Passed <chr> "TRUE", "TRUE", "FALSE", "TRUE", "FALSE"
$ Failed <chr> "FALSE", "FALSE", "TRUE", "FALSE", "TRUE"
```

```
[1] ".GlobalEnv"        "tools:rstudio"
[3] "package:stats"     "package:graphics"
[5] "package:grDevices" "package:utils"
[7] "package:datasets"  "package:methods"
[9] "Autoloads"         "package:base"
```

vii. Packages.

```
library() Opens the available libraries
search() shows the packages:
library("MASS") adds the package MASS to the library
search() shows the packages:
.libPaths() shows the path to the library of the machine
install.packages("e1071") installs the package e1071
install.packages("funModeling") installs the package funModeling, doesn't work because of version compatibility

x<-c("MASS","dplyr", "e1071") set the list of package names x
lapply(x, require, character.only = TRUE) apply to all package names the require function, result package MASS and e1071 are
                                          set as true because they were found, package dplyr is false because it wasn't found

require(funModeling) make funModeling required, doesn't work because the package was not found
```

```
[1] ".GlobalEnv"        "tools:rstudio"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

viii. Basic descriptive statistics and more.

```
iris Shows the matrix iris
summary(iris) Shows the minimum, maximum and median value and the 1st, 2nd and 3rd quartiles for each feature
fivenum(iris$Sepal.Length) shows the min, 1st Qu, Median, 3rd Qu and max values for the Sepal-Length feature

status(iris) Belongs to the funModeling library, so it doesn't run
```

(c) Open the file "**ControlFlow.R**", handed with this guide, and:

i. Set a breakpoint at line 6 and run the code delimited by region *EX.1*, step by step.

ii. Explain the purpose of the three examples, named *EX.1*, *EX.2*, and *EX.3*.

(d) Implement and show a code that gets the first two columns of *my.data*, using the range operator indexation.

(e) Implement and show a code that gets the first two columns of *my.data*, using vector indexation.

(f) Install the package `Hmisc`, and use the function `describe` on *my.data* variable.

3. **Setup Spark**.

(a) Install the package `sparklyr` using `install.packages("sparklyr")`. It may take some minutes, so be patient.

(b) Install *Spark* using `library(sparklyr) spark_install(version = '3.4.2', hadoop_version = '3')`

4. **Spark primer**.
Describe the commands used (if not listed), the problems (if any), and the results of the following points, line by line:

(a) Check, programmatically, if `sparklyr` package is loaded. Next, connect to *Spark* using
`ss <- spark_connect('local', version = '3.4.2', hadoop_version = '3', config = list())`.

(b) View the content of variable `ss`.

(c) Use function `copy_to` from package `dplyr` to copy the *iris* data set to *Spark*.

```
library(dplyr)
df <- copy_to(ss, iris)
show(df)
```

(d) Show some sample data from the loaded data set.

```
head( select( df, Petal_Width, Species))
head( filter( df, Petal_Width > 0.3))
df %>% head
```

(e) Using SQL

```
library(DBI)
df_sql <- dbGetQuery(ss, "SELECT * FROM iris WHERE Petal_Width > 0.3 LIMIT 5")
show(df_sql)
```

(f) Get data from *Spark* nodes.

```
local_df <- collect(df)
show(local_df)
show(df)
```

(g) Disconnect spark using `spark_disconnect(ss)`. Confirm, programmatically, that *Spark* is closed.

(h) Indicate, as an example, a supervised learning algorithm implemented on *Spark*, and available through `sparklyr`.

(i) Indicate, as an example, a unsupervised learning algorithm implemented on *Spark*, and available through `sparklyr`.

(j) Tell how *Spark* can support the *Big data process pipeline*. Try to start at https://spark.rstudio.com