
Instituto Superior de Engenharia de Lisboa

**Mineração de Dados em Larga Escala**

Mestrado Engenharia Informática e Multimédia &
Mestrado Engenharia Informática e Computadores

# Laboratório 1

# R language and environment setup

Semestre de Verão, 2022/2023

Nome: Gonçalo Fonseca | Número: A50185
Nome: Pedro Diogo | Número: A47573

# Índice

# 1 R Primer

## 1.1 b)

No ponto abaixo, a explicação de cada linha de código encontra-se acima de cada linha de código a explicar.

### 1.1.1 i)

```
# Guarda o valor numa variável de ambiente com o nome "var1"
var1 <- 3

# Apresenta o valor no terminal
show ( var1 )

# O valor da multiplicação entre var1 e var1 é guardado na variável
# var2, que neste caso é o valor de 1.
var2 <- var1  var1

# O valor de 1 expoente de 2 é guardado na variável var3, que
# neste caso é o valor de 1.
var3 <- var1 ** 2

# O valor de var4 é o mesmo que o var3 visto ambos serem o cálculo
# para o expoente, mas com caracteres diferentes.
var4 <- var1 ^ 2

# Observa-se se o valor de var1 é maior que o valor de var1.
# Neste caso será falso.
var1 < var1

# Observa-se se o valor de var3 é diferente de var4, que neste
# caso é falso.
var3 != var4

# Observa-se se o valor de var32 é igual de var4, que neste
# caso é verdadeiro.
var2 == var4

# O valor da subtração de var2 por var2 (valor = 0) é
# guardado na mesma variável
var2 <- var2 - var2

# O resultado da divisão de var3 por var2 é guardado em var5
var5 <- var3 / var2
```

```r
# O valor de var5 é incrementado por 1
var5 + 1
```

### 1.1.2 ii)

```r
# A função c cria vetores, sendo que neste caso o vetor
# inclui todos os valores dentro dos parêntises. O
# resultado é guardado na variável a. Neste caso as
# variáveis são numéricas.
a <- c ( 1 , 2 , 5.3 , 6 , -2 , 4 , 3.14159265359)

# Apresenta o vetor guardado na memória na variável a no terminal
a

# É criado um vetor de caracteres, guardado na variável b.
b <- c ( "1" , "2" , "3" )

# Verifica se o caractere "2" está dentro do vetor b, sendo
# o resultado TRUE.
"2" %in% b

# Verifica se o caractere "5" está dentro do vetor b, sendo
# o resultado FALSE.
"5" %in% b

# É criado um vetor de valores lógicos, guardado na variável c.
c <- c (TRUE,TRUE,FALSE,TRUE)

# O valor na variável c é guardado na variável a
a <- c

# O valor dá logical(0) pois os valores no vetor são lógicos,
# mas as posições no vetor começam por 1.
a[0]

# Obtém-se o valor da primeira posição do vetor.
a[1]

# Apresenta os valores do vetor sem o valor da posição 1.
a[1]

# Apresenta o valor da posição 8 no vetor, que neste caso
# é nulo pois não existe nenhum valor nessa posição
a[8]

# É criado um vetor com os valores 1 e 3, que ficará por exemplo
```

```r
# a[1, 3], e com isto obtém os valores do vetor a das posições
# 1 e posição 3
a [ c( 1 , 3) ]

# É o mesmo código de cima, só que o valor da primeira posição
# é o 3.
a [ c(3 , 1) ]

# Inicialmente faz-se a > 2 que, para cada valor no vetor,
# verifica-se se o valor é maior que 2. Neste caso dá tudo false porque
# TRUE e FALSE maior que 2 dá falso. De seguida, verifica-se
# no vetor, a posição FALSE no vetor a. Como FALSE não é um
# valor para se verificar a posição, dá logical(0), ou seja,
# não dá nenhum valor.
a [ a > 2 ]
```

#### 1.1.3 iii)

```r
# A função abaixo cria uma matriz com valores entre 1 e 6,
# com 3 linhas e 2 colunas
m <- matrix ( 1 : 6 , nrow=3, ncol=2)

# Apresenta a matriz guardado na memória na variável m no terminal
m

# A função abaixo cria uma matriz com valores entre 2 e 7,
# com 2 linhas e 3 colunas
n <- matrix ( 2 : 7 , nrow=2, ncol=3)

# Apresenta a matriz guardado na memória na variável n no terminal
n

# Obtém os valores da 2ª coluna
m[ , 2 ]

# Obtém os valores da 1ª linha
n [ 1 , ]

# Obtém-se as linhas 2 e 3, e as colunas 1 e 2
m[2:3,1:2]

# Produto de n por m
n %% m

# Produto de m por n
m %% n
```

```r
# Produto de n por n
n %% n

# Na matriz n, aplica-se a cada valor o seu quadrado.
n^2

# Na matriz n, é feito a cada valor a raiz quadrada.
sqrt(n)
```

#### 1.1.4 iv)

```r
# Cria um vetor número e guarda na variável d.
d <- c ( 1 , 2 , 3 , 4 )

# Cria um vetor de caracteres (string) e guarda na variável e.
e <- c ( "Bob", "Alice", NA, "Joe" )

# É criado um vetor de valores lógicos, guardado na variável f.
f < c (TRUE,TRUE,FALSE,TRUE)

# Cria uma tabela (como dataframe em Python), em que a primeira
# coluna é o vetor d, etc.
my.data <- data.frame ( d , e , f , stringsAsFactors=FALSE)

# Apresenta a tabela na linha de comandos
show (my.data)

# Define o nome das colunas
names(my.data ) <- c ( "ID" , "Name" , "Passed " )

# Mostra a tabela de uma forma mais visual (sem ser na linha
# de comandos)
View(my.data)

# Mostra os valores da coluna Name
my.data$Name

# Adiciona uma coluna nova à tabela my.data, com os valores
# contrários de f, na coluna Failed
my.data <- cbind (my.data , Failed =! f )

# Adiciona uma linha no final da tabela, com os valores do vetor.
my.data <- rbind(my.data , c(5 , "Carol" ,FALSE,TRUE) )
```

```r
# Apresenta a documentação de uma data frame.
?data.frame
```

#### 1.1.5 v)

```r
# Cria um vetor de caracteres, que junta 20 valores "red" e
30 valores "blue"
colour <- c ( rep ( "red", 20 ) , rep ( "blue" , 30 ) )

# Cria uma espécie de enumerado
colour <- factor(colour)

# Apresenta um sumário de quantas vezes aparece um valor
summary(colour)

# Cria um vetor de caracteres com os valores abaixo.
dimensions <- c("large", "medium", "small")

# Neste caso faz o mesmo que o factor. No entanto, o objetivo
# do ordered é ordenar por níveis, e por essa razão esta
# função tem um parâmetro levels que damos os níveis que
# pretendemos
dimensions <- ordered(dimensions)
```

#### 1.1.6 vi)

```r
# Apresenta o número de elemntos que compõem a variável
length(colour)

# O mesmo de cima
length(a)

# É criado um vetor de valores lógicos, guardado na variável f.
f < c (TRUE,TRUE,FALSE,TRUE)

# Apresenta o tipo de variável de uma certa variável. Neste caso,
# baseado nas linhas anteriores, é um factor.
class(colour)

# Apresenta o número de linhas na tabela my.data
nrow(my.data)

# Apresenta o número de colunas na tabela my.data
```

```r
ncol(my.data)

# Converte uma variável em string
str(my.data)

# Apresenta as variáveis guardadas
ls()

# Remove a variável a
rm(a)

# Apresenta a tabela com algumas inforamções adicionais como
# o número de linhas, colunas e o tipo de variáveis.
glimpse (my.data )
```

#### 1.1.7 vii)

```r
# Apresenta os packages extras instalados
library()

# Apresenta os packages a serem usados
search()

# Importa o package MASS
library("MASS")

# Apresenta os mesmos packages do código 2 linhas antes,
# mas com o package MASS incluido
search()

# Apresenta todas as pastas que o R conhece
.libPaths()

# Instala o package "e1071"
install.packages("e1071")

# Cria um vetor com 3 strings
x<-c ( "MASS" , " dplyr" , " e1071 " )

# Aplica uma função a todos os valores do vetor
lapply(x, require, character.only = TRUE)

# Importa o package
require(funModeling)
```

```
# Apresenta a tabela com algumas inforamções adicionais como
# o número de linhas, colunas e o tipo de variáveis.
glimpse (my.data )
```

#### 1.1.8 viii)

```
# Importa o dataset da IRIS
iris

# Mostra as caracterisitcas de cada atributo
summary(iris)

# Apresenta, para cada caracteristica, o máximo, mínimo e quartis.
fivenum(iris$Sepal.Length )

# Apresenta a quantidade de zeros, valores NA e valores infinitos
status(iris)
```

### 1.2 c)

#### 1.2.1 ii.

```
#### Ex1 ####
x<-0
for (i in seq(0,10,1))
{
        x<-x+i
}
```

O código do exercício 1 acima corre um ciclo for numa sequência de valores entre 0 e 10, com o stop de 1. A cada ciclo, o valor **x** que começa com o valor 0, adiciona-se o valor de **i** que é incrementado.

```
#### Ex2 ####
for (i in rep(0,10))
{
        print(i)
}
```

O código do exercício 2 acima corre um ciclo for com uma sequência valores entre 10 valores do número 0. A cada ciclo, faz o print do valor i, que será sempre 0, 10 vezes

```
#### Ex3 ####
v<-c(1,2,3,4,5,6,7,8,9)
while(length(v)>0)
{
        v<-v[-1]
        print(v)
        if( (6 %in% v) == FALSE)
        break
}
```

Inicia-se com um vetor com valores entre 1 e 0. Corre-se um loop que corre continuamente desde que o tamanho do vetor seja maior que 0. De seguida, o primeiro elemento do vetor é removido, fazendo print do vetor atualizado. Quando o valor 6 deixar de existir no vetor, o ciclo para.

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

```
my.data[, 1:2]
```

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

```
my.data[, c(1,2)]
```

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

```
install.packages( "Hmisc" )
library(Hmisc)
```

# 2 Setup Spark

## 2.1 a) *Install the package sparklyr*

```
install.packages("sparklyr")
```

## 2.2 b) *Install Spark using library(sparklyr) spark_install(version = '3.3.2', hadoop_version = '3')*

Para instalar o spark, usaram-se os comandos abaixo.

```
library(sparklyr)
spark_install(version = '3.3.2', hadoop_version = '3.2')
```

No entanto, após correr estes comandos, apareceu o erro "The Spark version specified may not be available".
Por essa razão, decidiu-se procurar por versões abaixo, como a do seguinte código que instalou corretamente.

```
spark_install(version = '3.2.2', hadoop_version = '3')
```

# 3 Spark primer

## 3.1 a) *Check programatically if sparklyr package is loaded.*

```
requireNamespace("sparklyr")
if (requireNamespace("sparklyr", quietly = TRUE)) {
        "Is installed"
        } else {
        "Is not installed"
}
```

De seguida, conectou-se ao Spark, usando as versões acima instaladas, e guardando a conexão na variável ss.

```
ss <- spark_connect('local', version = '3.2.2', hadoop_version = '3.2',
config = list())
```

## 3.2 b) *View the content of the variable ss*

```
View(ss)
```

Ao correr o código acima, foi apresentada uma tabela com várias informações sobre a conexão como a versão, o id da sessão, o *path* do spark, entre outros.

## 3.3 c) *Copy the iris dataset to spark*

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

O que o código acima faz é importar a library dplyr, copiar uma base de dados local (neste caso a base de dados da iris) para um *data source* remoto, e por fim apresentar os dados desse dataset na linha de comandos, apresentando algumas informações sobre os dados.

## 3.4 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
```

A primeira linha mostra apenas as colunas Petal_Width e Species. A segunda linha mostra todas as linhas com Petal_Width é maior que 0.3. Por fim, a terceira linha mostra as 5 primeiras rows do dataframe.

## 3.5 e) *Using SQL*

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

O código acima importa uma library chamada DBI, que faz com que seja possível fazer consultas SQL no dataframe que temos. Neste caso, a consulta mostra as 5 primeiras linhas com a Petal_Width maior que 0.3

## 3.6 f) *Get data from Spark nodes*

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

O código acima cria uma variável local_df onde guarda os dados da dataframe df que está guardada remotamente, como explicado acima. As últimas duas linhas servem para mostrar que os dados são iguais, apenas guardados em locais diferentes (um na memória e outro remotamente).

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

```
spark_disconnect(ss)
connection_is_open(ss)
```

A 2ª linha "connection_is_open(ss)"verifica se o Spark está conectado ou não. Caso esteja, dá o resultado de TRUE. Caso não esteja (que é o suposto depois de correr o primeiro comando), o resultado suposto é FALSE.

## 3.8 h) *Indicate, as an example, a supervised learning algorithm implemented on Spark, and available through sparklyr*

Um exemplo de um algoritmo de aprendizagem supervisionada é o algoritmo de **Random Forest**, presente na biblioteca **sparklyr**. [1]

## 3.9 i) *Indicate, as an example, a unsupervised learning algorithm implemented on Spark, and available through sparklyr*

Um exemplo de um algoritmo de aprendizagem supervisionada é o algoritmo de **K-Means Clustering**, presente na biblioteca **sparklyr**. [1]

## 3.10 j) *Tell how Spark can support the Big data process pipeline.*

Algumas razões pelas quais o Spark pode suportar o processo de Big Data são, por exemplo, que o Spark consegue consumir grandes volumes de dados de diferentes fontes, consegue processar dados em paralelo, conseguindo ganhos significativos de desempenho. Também, pode realizar tarefas avançadas de análise e visualização de dados.

# Referências

[1] sparklyr - [Em linha] [Consult. 19 mar. 2023]. Disponível em WWW:https://spark.rstudio.com/packages/sparklyr/latest/reference/#spark-machine-learning.