R ACTUAIRE DATA SCIENCE

ARTHUR CHARPENTIER

In this series of exercices, the files are in the exR.zip file. Extract them in a folder named

...\exRdatascientist\dataset

(1) Get the location of the working directory.

```
> getwd()
[1] "d:/home/documents/"
```

(or somewhere else, depending on the location)

(2) Define ...\exRdatascientist\dataset as the working directory.

```
> setwd("D:/R/prework/exRdatascientist/dataset")
```

In R, we cannot use Windows's style, i.e. backslashes for paths, it should be Unix (forward) slashes. Nevertheless, one can also use

```
> setwd("D:\\R\\prework\\exRdatascientist\\dataset")
```

(3) Import the data1.csv dataset in a data frame object called data1.

```
> data1 <- read.csv2("data1.csv",header=TRUE)
```

A more general command is probably

```
> data1 <- read.table("data1.csv",header=TRUE,sep=";")
```

where we simply mention that it is a text file, with semi-colon delimited columns.

```
> head(data1)
       var1     var2       var3 var4  var5      var6 var7
1 0.7551818 33.01560 1.58589539    c FALSE 0.4694437    8
2 1.1816428 50.45929 1.15616285    d FALSE 0.6852608    5
3 0.1457067 75.94693 0.08695963    c FALSE 0.2556477    3
4 0.1397953 98.16874 1.78660771    b FALSE 2.1000437    2
5 0.4360686 39.56624 1.92524262    e FALSE 5.0076963    1
6 2.8949685 94.08644 0.73748551    e FALSE 1.2553161    5
```

(4) Import the data1.csv dataset, without the 6th line.

```
> data1a <- read.table("data1.csv",header=TRUE,sep=";",
   nrows=5)
> data1b <- read.table("data1.csv",header=FALSE,sep=";",
   skip=7)
> data1b <- data1b[,-1]  # pb with header=FALSE vs TRUE
> names(data1b) <- names(data1a)
> data1r6 <- rbind(data1a,data1b)
> head(data1r6)
       var1     var2       var3 var4  var5      var6 var7
1 0.7551818 33.01560 1.58589539    c FALSE 0.4694437    8
2 1.1816428 50.45929 1.15616285    d FALSE 0.6852608    5
3 0.1457067 75.94693 0.08695963    c FALSE 0.2556477    3
4 0.1397953 98.16874 1.78660771    b FALSE 2.1000437    2
5 0.4360686 39.56624 1.92524262    e FALSE 5.0076963    1
7 1.2295621 95.60495 0.49284122    f  TRUE 0.6699745    5
```

It is a bit technical here, because if we skip some raws, we miss the names of the variables. Observe also that there is an additional column here (because of counts)

```
> data1b <- read.table("data1.csv",header=FALSE,sep=";",
   skip=7)
```

```
> head(data1b,3)
V1 V2         V3        V4         V5 V6    V7         V8 V9
 7  7 1.2295621 95.60495 0.4928412  f  TRUE 0.6699745  5
 8  8 0.5396828 90.03570 7.1819578  d FALSE 0.1883261  2
 9  9 0.9565675 53.85171 0.9139324  a FALSE 4.4402074  9
```

this is why we removed the first column.

(5) Import the data1.csv dataset, without the 6th variable.

Here the code is

```
> mycols <- rep(NA,9)
> mycols[8] <- "NULL"
> data1c6 <- read.table("data1.csv",header=TRUE,sep=";",
+ colClasses=mycols)
> head(data1c)
Id       var1     var2        var3 var4   var5 var7
 1 0.7551818 33.01560 1.58589539    c FALSE    8
 2 1.1816428 50.45929 1.15616285    d FALSE    5
 3 0.1457067 75.94693 0.08695963    c FALSE    3
 4 0.1397953 98.16874 1.78660771    b FALSE    2
 5 0.4360686 39.56624 1.92524262    e FALSE    1
 6 2.8949685 94.08644 0.73748551    e FALSE    5
```

(6) Import the data1.csv dataset, without non-numeric variables.

It is possible to do it - somehow - manually (inspired by the previous question)

```
> mycols <- rep(NA,9)
> mycols[c(5,6)] <- "NULL"
> data1cnn <- read.table("data1.csv",header=TRUE,sep=";",
+ colClasses=mycols)
> head(data1cnn)
Id       var1     var2  var5       var6 var7
```

```
 1 0.7551818 33.01560 FALSE 0.4694437    8
 2 1.1816428 50.45929 FALSE 0.6852608    5
 3 0.1457067 75.94693 FALSE 0.2556477    3
 4 0.1397953 98.16874 FALSE 2.1000437    2
 5 0.4360686 39.56624 FALSE 5.0076963    1
 6 2.8949685 94.08644 FALSE 1.2553161    5
```

It is also possible to read the first raw, and to test whether entries are numeric, or not

```
> data0 <- read.table("data1.csv",header=TRUE,
+ sep=";",skip=0,nrows=1)
> I <- sapply(data0,is.numeric)
> mycols <- rep(NA,ncol(data0))
> names(mycols) <- names(data0)
> mycols[!I] <- "NULL"
> data1cnn <- read.table("data1.csv",header=TRUE,sep=";",
+ colClasses=mycols)
> head(data1cnn)
Id       var1     var2        var3       var6 var7
 1 0.7551818 33.01560 1.58589539 0.4694437    8
 2 1.1816428 50.45929 1.15616285 0.6852608    5
 3 0.1457067 75.94693 0.08695963 0.2556477    3
 4 0.1397953 98.16874 1.78660771 2.1000437    2
 5 0.4360686 39.56624 1.92524262 5.0076963    1
 6 2.8949685 94.08644 0.73748551 1.2553161    5
```

(7) Sort the data frame data1 according to the value of variable `data1$var1`.

```
> data1s <- data1[order(data1$var1), ]
> head(data1s)
 Id         var1      vr2       var3 var4   var5       var6 vr7
254 0.001700975 39.3627 9.5056159    f FALSE 0.5283091    4
```

```
394 0.003509650 37.1241 0.6664607    f FALSE 5.3020584    5
274 0.004501631 19.2581 0.1119306    e FALSE 2.3290662    6
287 0.005634216 60.4258 1.5898802    a FALSE 0.9456810    6
313 0.007649621 11.5346 2.2497970    a FALSE 0.4745822    9
402 0.011686048 81.4552 0.1235825    f FALSE 1.0534607    5
```

(8) What is the mean of the `data1$var2` variable? (make sure that it is a numeric variable, first).

```
> is.numeric(data1$var2)
[1] TRUE
> mean(data1$var2)
[1] 52.35271
```

(9) What is the mean of variable `data1$var3` when `data1$var4=="a"`?

```
> mean(data1$var3[data1$var4=="a"])
[1] 1.867404
```

(10) Print a table that contains all the means of `data1$var3` for all possible values of `data1$var4`?

A first idea might be to use

```
> aggregate(x=data1$var3, by=list(var4=data1$var4), FUN="
   mean")
  var4         x
1    a 1.867404
2    b 1.542238
3    c 1.679678
4    d 1.626285
5    e 1.740542
6    f 1.480313
```

An alternative is to use

```
> tapply(data1$var3,data1$var4,mean)
        a         b         c         d         e         f
1.867404  1.542238  1.679678  1.626285  1.740542  1.480313
```

Another popular function is

```
> library(plyr)
> ddply(data1,~var4,summarise,mean.var3=mean(var3))
  var4  mean.var3
1    a  1.867404
2    b  1.542238
3    c  1.679678
4    d  1.626285
5    e  1.740542
6    f  1.480313
```

Finally, another command worth understanding is

```
> library(data.table)
> dt <- data.table(data1)
> dt[,list(mean=mean(var3)),by=var4]
   var4      mean
1:    c  1.679678
2:    d  1.626285
3:    b  1.542238
4:    e  1.740542
5:    f  1.480313
6:    a  1.867404
```

(11) Using

```
> library(xtable)
```

generate an html page that contains the previous table.

```
> library(xtable)
> var34.table <- ddply(data1,~var4,summarise,mean.var3=
   mean(var3))
> t34 <- xtable(var34.table)
> print(t34, type = "html")
<!-- html table generated in R 3.1.1 by xtable 1.7-4
   package -->
<table border=1>
<tr> <th>  </th> <th> var4 </th> <th> mean.var3 </th>  </
   tr>
  <tr> <td align="right"> 1 </td> <td> a </td> <td align="
     right"> 1.87 </td> </tr>
  <tr> <td align="right"> 2 </td> <td> b </td> <td align="
     right"> 1.54 </td> </tr>
  <tr> <td align="right"> 3 </td> <td> c </td> <td align="
     right"> 1.68 </td> </tr>
  <tr> <td align="right"> 4 </td> <td> d </td> <td align="
     right"> 1.63 </td> </tr>
  <tr> <td align="right"> 5 </td> <td> e </td> <td align="
     right"> 1.74 </td> </tr>
  <tr> <td align="right"> 6 </td> <td> f </td> <td align="
     right"> 1.48 </td> </tr>
   </table>
```

(12) Is the size of the data1 object smaller or larger than 30 kilobytes ?

```
> object.size(data1)
55336 bytes
```

(13) Import the data2.csv dataset, in the data2.zip file, without unzipping it first.

```
> data2 <- read.table(unz("data2.zip", "data2.csv"),
+ sep=";",header=TRUE)
```

(14) Import the dataset in the sheet Sheet2 of the data3.xlsx, by saving the file as data3.csv

See question 3

```
> data3 <- read.table("data3.csv",header=TRUE)
```

(15) Import the dataset in the sheet Sheet2 of the data3.xlsx, using function `read.xls()` in `library(gdata)` ?

This package is available for Windows, Mac or Linux. Unfortunately, it requires to install additional Perl libraries (Perl is usually already installed in Linux and Mac, but sometimes require more effort in Windows platforms)

```
> library(gdata)
> data3 <- read.xls("data3.xlsx",sheet=2)
```

If it works, we get

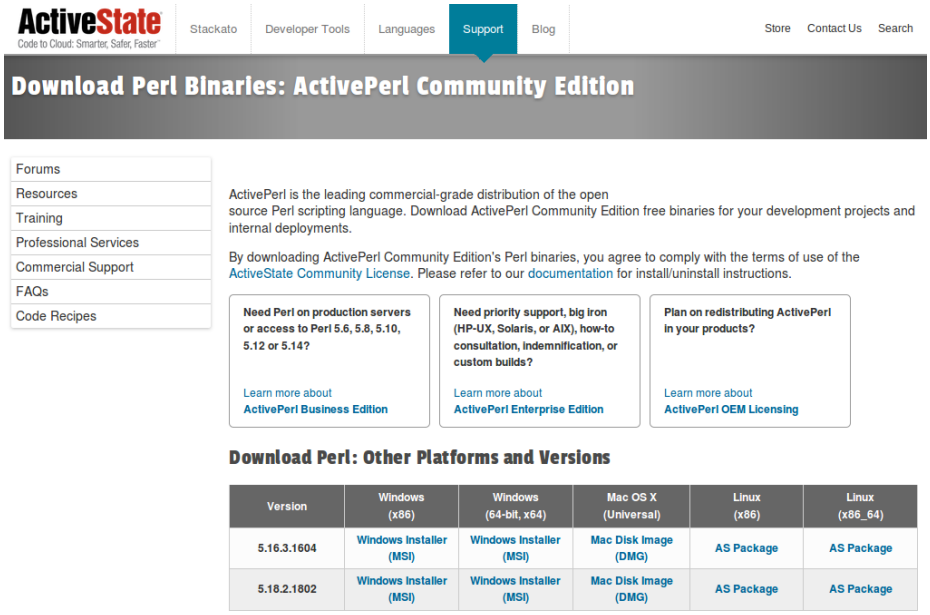
```
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
  LANGUAGE = "fr_CA:fr",
  LC_ALL = (unset),
  LANG = "fr_CA.UTF-8"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
> head(data3[,1:5])
  Id              var1             var2              var3
     var4
1  1  1.73062657183948 55.3600040785968  1.25501332532562
      b
2  2 0.615032798144966 24.7821482941508 0.542722586363832
      d
3  3  1.23291658605905 19.4911944386549  1.71712344967676
      d
```

```
4  4  1.00408444921942 19.7579132029787  2.48421074569453
      c
5  5 0.204201349522918  20.138315009186  4.08214373777409
      c
6  6 0.208788108080626 81.3062246441841 0.469734742496311
      b
```

It should work. But we might also get

```
Error in findPerl(verbose = verbose) :
  perl executable not found. Use perl= argument to specify
      the correct path.
Error in file.exists(tfn) : invalid 'file' argument
```

It means that it did not work, it is till possible to install Perl libraries on a Windows machine. Go on http://www.activestate.com/activeperl, then download and excecute the .exe program.



When installation is complete, it might be necessary to re-install library gdata. Then use simply

```
> library(gdata)
> data3 <- read.xls("data3.xlsx",sheet=2,
+ perl= "C:/Perl64/bin/wperl.exe")
```

(16) Import the dataset in the sheet Sheet2 of the data3.xlsx folder, using function read.xlsx() in library(xlsx) ?

```
> library(xlsx)
> data3 <- read.xlsx("data3.xlsx",sheetName="Sheet2")
> head(data3[,1:5])
  Id                var1              var2                var3
     var4
1  1  1.73062657183948 55.3600040785968  1.25501332532562
      b
2  2 0.615032798144966 24.7821482941508 0.542722586363832
      d
```

```
3  3  1.23291658605905 19.4911944386549  1.71712344967676
      d
4  4  1.00408444921942 19.7579132029787  2.48421074569453
      c
5  5 0.204201349522918  20.138315009186  4.08214373777409
      c
6  6 0.208788108080626 81.3062246441841 0.469734742496311
      b
```

(17) Import the dataset in the sheet Sheet2 of the data3.xlsx folder, using function `readWorksheet()` in `library(XLConnect)` ?

```
> library(XLConnect)
> wb <- loadWorkbook("data3.xlsx")
> data3 <- readWorksheet(wb,sheet="Sheet2")
> head(data3[,1:5])
  Id               var1              var2               var3
     var4
1  1  1.73062657183948 55.3600040785968  1.25501332532562
      b
2  2 0.615032798144966 24.7821482941508 0.542722586363832
      d
3  3  1.23291658605905 19.4911944386549  1.71712344967676
      d
4  4  1.00408444921942 19.7579132029787  2.48421074569453
      c
5  5 0.204201349522918  20.138315009186  4.08214373777409
      c
6  6 0.208788108080626 81.3062246441841 0.469734742496311
      b
```

(18) Import the dataset in the sheet Sheet2 of the data3.xlsx folder, using function `sqlQuery` in `library(RODBC)` ? [hint: use function `odbcConnectExcel` first]

This package used to be popular on Windows plateforms. Unfortunately, it requires Excel ODBC driver to be installed. We need to use the old format of Excel files (and save the .xlsx file in the .xls format first). Then, use

```
> library(RODBC)
> conn <- odbcConnectExcel("data3.xls")
> spreadsheet <- sqlTables(conn)
> query <- "SELECT * FROM [Sheet2$C4:J259]"
> data3 <- qlQuery(conn,query)
> close(conn)
```

or

```
> odbcCloseAll()
```

Then

```
> head(data3[,1:5])
  Id               var1              var2               var3
     var4
1  1  1.73062657183948 55.3600040785968  1.25501332532562
      b
2  2 0.615032798144966 24.7821482941508 0.542722586363832
      d
3  3  1.23291658605905 19.4911944386549  1.71712344967676
      d
4  4  1.00408444921942 19.7579132029787  2.48421074569453
      c
5  5 0.204201349522918  20.138315009186  4.08214373777409
      c
6  6 0.208788108080626 81.3062246441841 0.469734742496311
      b
```

(19) Use function `with()` to compute the mean of variable `var3` of data frame `data1` (with using the `$` symbol).

```
> with(data1,mean(var3))
[1] 1.649216
```

(20) Using function `tempfile()` and `download.file()`, import the data5.dat dataset from http://freakonometrics.free.fr/data5.zip

```
> temp <- tempfile()
> download.file("http://freakonometrics.free.fr/data5.zip"
   ,temp)
trying URL 'http://freakonometrics.free.fr/data5.zip'
Content type 'application/zip' length 219319 bytes (214 Kb
   )
opened URL
==================================================
downloaded 214 Kb

> data5 <- read.table(unz(temp, "data5.dat"))
> unlink(temp)
> head(data5)
  V1 V2  V3   V4
1  1  1   6 4596
2  1  1  29 2882
3  1  1  52 2867
4  1  1  78 5224
5  1  1  97 2986
6  1  1 126 5729
```

(21) Import datasets data1.csv and data4.csv. Add a column named `counts` in `data1` that counts the number of appearences of variable `Id` in `data4`.

```
> data1 <- read.table("data1.csv",header=TRUE,sep=";")
```

```
> data4 <- read.table("data4.csv",header=TRUE,sep=";")
> N <- rep(0,nrow(data1))
> T <- table(data4$Id)
> N[as.numeric(names(T))] <- as.numeric(T)
> data1$counts <- N
> head(data1)
  Id       var1     var2        var3 var4  var5       var6
     var7 cou
1  1 0.7551818 33.01560 1.58589539    c FALSE 0.4694437
      8   2
2  2 1.1816428 50.45929 1.15616285    d FALSE 0.6852608
      5   1
3  3 0.1457067 75.94693 0.08695963    c FALSE 0.2556477
      3   2
4  4 0.1397953 98.16874 1.78660771    b FALSE 2.1000437
      2   3
5  5 0.4360686 39.56624 1.92524262    e FALSE 5.0076963
      1   0
6  6 2.8949685 94.08644 0.73748551    e FALSE 1.2553161
      5   1
```

(22) data1 is a data.frame object. Using function as.data.table in library(data.table), create a data.table object, named data5.

```
> library(data.table)
data.table 1.9.2  For help type: help("data.table")
> data5=as.data.table(data1)
> head(data5)
   Id       var1     var2        var3 var4  var5       var6
      var7
1:  1 0.7551818 33.01560 1.58589539    c FALSE 0.4694437
      8
```

```
2:  2 1.1816428 50.45929 1.15616285    d FALSE 0.6852608
      5
3:  3 0.1457067 75.94693 0.08695963    c FALSE 0.2556477
      3
4:  4 0.1397953 98.16874 1.78660771    b FALSE 2.1000437
      2
5:  5 0.4360686 39.56624 1.92524262    e FALSE 5.0076963
      1
6:  6 2.8949685 94.08644 0.73748551    e FALSE 1.2553161
      5
```

(23) What doe the following commands returns,

```
> setkey(data5,var4)
> data5["a",]
> setkey(data5,var4,var7)
> data5[J("a",1),]
> data5["b",sum(var1)]
> data5[J("b",1),sum(var1)]
> data5["b",sum(var1),by=var7]
```

Reproduce those outputs using the data.frame object data1.

```
> setkey(data5,var4,var7)
> data5[J("a",1),]
   var4 var7  Id       var1     var2       var3  var5
      var6
1:    a    1  87 1.5836961 21.18990 0.4677386 FALSE
   0.7561370
2:    a    1  96 0.2637383 58.04125 0.4544816  TRUE
   2.5037483
3:    a    1 272 1.6295041 34.49076 0.3132506 FALSE
   5.1976029
```

```
4:    a    1 328 2.5896858 69.91569 2.1684339 FALSE
   6.6675872
5:    a    1 403 1.2375457 73.36742 5.4597003 FALSE
   0.4919902
6:    a    1 408 0.1967908 76.03508 0.3221431 FALSE
   1.0507142
7:    a    1 479 0.2659871 30.62531 1.6431458 FALSE
   2.4747742
8:    a    1 490 0.3994575 37.96098 0.5575442  TRUE
   0.5283225
```

To reproduce this output, we can use the following commands

```
> data1[(data1$var4=="a")&(data1$var7==1),]
     Id       var1      var2       var3 var4  var5       var6
        var7
87   87 1.5836961 21.18990 0.4677386    a FALSE 0.7561370
      1
96   96 0.2637383 58.04125 0.4544816    a  TRUE 2.5037483
      1
272 272 1.6295041 34.49076 0.3132506    a FALSE 5.1976029
      1
328 328 2.5896858 69.91569 2.1684339    a FALSE 6.6675872
      1
403 403 1.2375457 73.36742 5.4597003    a FALSE 0.4919902
      1
408 408 0.1967908 76.03508 0.3221431    a FALSE 1.0507142
      1
479 479 0.2659871 30.62531 1.6431458    a FALSE 2.4747742
      1
490 490 0.3994575 37.96098 0.5575442    a  TRUE 0.5283225
      1
```

Then

```
> data5["b",sum(var1)]
   var4       V1
1:    b 82.2133
```

which can be reproduced using

```
> sum(data1[data1$var4=="b","var1"])
[1] 82.2133
```

or equivalently

```
> tapply(data1$var1,data1$var4,sum)["b"]
       b
82.2133
```

Then

```
> data5[J("b",1),sum(var1)]
   var4  var7       V1
1:    b     1 10.5845
```

and finally

```
> data5["b",sum(var1),by=var7]
   var7         V1
1:    1 10.584502
2:    2  3.682740
3:    3  6.323050
4:    4 10.844207
5:    5  9.044560
6:    6  5.364524
7:    7  8.774353
8:    8 18.900576
9:    9  8.694790
```

that can be reproduced using

```
> data1b <- data1[data1$var4=="b",]
> tapply(data1b$var1,data1b$var7,sum)
         1          2          3          4          5
         6
10.584502  3.682740  6.323050 10.844207  9.044560
  5.364524
```

... etc.

(24) What is the function `intersect` used for? What would the following code return?

```
> intersect(seq(4,28,by=7),seq(3,31,by=2))
```

To get more information about a specific function, use

```
> ?intersect
```

Here we get

```
> intersect(seq(4,28,by=7),seq(3,31,by=2))
[1] 11 25
```

which is the vector with elements in both vectors. To get a better understanding, look at the following commands

```
> seq(4,28,by=7)
[1]  4 11 18 25
> seq(3,31,by=2)
 [1]  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31
> seq(4,28,by=7) %in% seq(3,31,by=2)
[1] FALSE  TRUE FALSE  TRUE
>  seq(4,28,by=7)[seq(4,28,by=7) %in% seq(3,31,by=2)]
[1] 11 25
```

(25) What would the following code return?

```
> c(TRUE,TRUE,FALSE,FALSE) & c(TRUE,FALSE,FALSE,TRUE)

[1]  TRUE FALSE FALSE FALSE
```

to understand the output, look at

```
> c(1,1,0,0)*c(1,0,0,1)
[1] 1 0 0 0
```

(26) What would the following code return?

```
> n<<--1
> if (n==0) "yes" else "no"; n
> if (n < - 0) "yes" else "no"; n
> if (n<-0) "yes" else "no"; n
> if (n=0) "yes" else "no"; n
> if (n<-2) "yes" else "no"; n

> n<<--1
```

The first line returns nothing but n takes value -1

```
>  if (n==0) "yes" else "no"; n
[1] "no"
[1] -1
```

Here we test if n is equal to 0. Obviously, the answer is "no" (since its value was -1, and still is)

```
>  if (n < - 0) "yes" else "no"; n
[1] "yes"
[1] -1
```

Here we test if n less than -0. The answer is "yes" (since its value was -1, and still is)

```
>  if (n<-0) "yes" else "no"; n
[1] "no"
[1] 0
```

(27) What will `1:10*1:5` return?

The output is

```
[1]   1   4   9 16 25   6 14 24 36 50
```

Keep in mind that R has a recycling rule : the first vector has 10 values, the second one has 5 values. In order to have vectors with identical length, the second vector will recycle its values. Computrations are

```
> (1:10)*c(1:5,1:5)
 [1]   1   4   9 16 25   6 14 24 36 50
```

(28) Given a numeric vector `x`, write a function which returns only elements of `x` larger than `mean(x)`

Consider for instance

```
> subelements <- function(x) x[x>=mean(x)]
```

In order to check, let us try with a simple vector

```
> subelements(0:9)
[1] 5 6 7 8 9
```

(29) Write a function `seqrep(n)` which returns vector $(1, 2, 2, 3, 3, 3, \ldots, n)$ where integer k is repeated k times. How long is this vector when $n = 50$?

We can use

```
> seqrep <- function(n){
+ x <- 1
+ for(i in 2:n) x <- c(x,rep(i,i))
+ return(x)}
```

With a not too large value for n, we get

```
> seqrep(4)
 [1] 1 2 2 3 3 3 4 4 4 4
```

About the last question, the answer is

```
> length(seqrep(50))
[1] 1275
```

which makes sense

```
> sum(1:50)
[1] 1275
> 50*51/2
[1] 1275
```

Note that it is also possible to use function `sequence`

```
> seqrep <- function(n) sort((n:1)[sequence(1:n)])
```

(30) Write a function that counts the number of `NA`'s in a vector.

Consider

```
> countna <- function(x) sum(is.na(x))
```

for instance, just to check

```
> countna(c(1,NA,NA,2,3,NA,4))
[1] 3
```

(31) Create a function `secdiag(M)` which returns the second diagonal of squared matrix `M`.

Consider

```
> secdiag <- function(M) diag(M[,ncol(M):1])
```

for instance, just to check

```
> M <- matrix(1:25,5,5)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> secdiag(M)
[1] 21 17 13  9  5
```

Actually, it could also mak sense to return the reverse

```
> secdiag <- function(M) rev(diag(M[,ncol(M):1]))
```

(32) What will `M[,2]` return, if

```
> M <- matrix(1:5,3,3)
```

```
> M <- matrix(1:5,3,3)
Warning message:
In matrix(1:5, 3, 3) :
  data length [5] is not a sub-multiple or multiple of the
      number of rows [3]
> M
     [,1] [,2] [,3]
[1,]    1    4    2
[2,]    2    5    3
[3,]    3    1    4
```

Here we have 5 values for a 3×3 matrix, so the recycling rule will be used here

```
> M <- matrix(c(1:5,1:4),3,3)
> M
```

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

Once we under this principle, we can see what the second column is

```
> M[,2]
[1] 4 5 1
```

(33) Get the help page on command %%. What does that mean for x if x %% 3 == 0 is TRUE?

To get the help page, use quotations marks

```
> ?%%
Error: unexpected SPECIAL in "?%%"
> ?"%%"
```

Thus, x %% 3 == 0 means that x is a multiple of 3

```
> (1:22)%%3==0
 [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
     FALSE
[11] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
    FALSE
```

(34) Given matrix

```
> m <- matrix(1:20,5,4)
```

what will the following line return

```
> which(m %% 3 == 0, arr.ind=TRUE)
```

The matrix is the following

```
> m <- matrix(1:20,5,4)
> m
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

without the last part, observe that

```
> m <- matrix(1:20,5,4)
> which((1:20) %% 3 == 0)
[1]  3  6  9 12 15 18
```

Now, with the `arr.ind=TRUE` option, we get the indices in the matrix, row and column,

```
> which(m %% 3 == 0, arr.ind=TRUE)
     row col
[1,]   3   1
[2,]   1   2
[3,]   4   2
[4,]   2   3
[5,]   5   3
[6,]   3   4
```

(35) Write a function that computes the power of any square matrix, `power(M,n)`.

Consider, for instance,

```
> power <- function(M,n){
+ if(n==0) N=diag(ncol(M))
+ N=M
+ if(n>1) for(i in 2:n) N=N %*% M
+ return(N)}
```

Let us check

```
> H=matrix(c(.2,.6,.8,.4),2,2)
> power(H,1)
     [,1] [,2]
[1,]  0.2  0.8
[2,]  0.6  0.4
> power(H,2)
     [,1] [,2]
[1,] 0.52 0.48
[2,] 0.36 0.64
> power(H,3)
      [,1]  [,2]
[1,] 0.392 0.608
[2,] 0.456 0.544
> H %*% H %*% H
      [,1]  [,2]
[1,] 0.392 0.608
[2,] 0.456 0.544
```

(36) Which function should you use to compute M^{-1}?

The function is

```
> ?solve
```

Indeed

```
> H <- matrix(c(.2,.6,.8,.4),2,2)
> solve(H)
     [,1] [,2]
[1,] -1.0  2.0
[2,]  1.5 -0.5
> H %*% solve(H)
     [,1] [,2]
[1,]    1    0
```

```
[2,]    0    1
```

(37) Create the identity matrix, of size 5×5.

```
> diag(5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

(38) Using commands `!` and `%in%` return the subvector of

```
> x <- sample(1:15)
```

where values `c(3,7,12)` are removed.

```
> x[!x%in%c(3,7,12)]
 [1] 14 15  1 11  4  5  2  8  9  6 10 13
```

Since I did not fix the seed of the random number generator, everyone should get so,ething different ! To fix it, use

```
> set.seed(476)
> x <- sample(1:15)
> x[!x%in%c(3,7,12)]
 [1]  6 10  4 13  1  2  8  9 11 15  5 14
```

(39) Given a matrix M, write a function which returns the following Kronecker product,

$$\begin{pmatrix} 1 & 3 & 4 \\ 2 & 0 & 5 \end{pmatrix} \otimes M = \begin{pmatrix} M & 3M & 4M \\ 2M & 0 & 5M \end{pmatrix}$$

```
> K <- matrix(c(1,2,3,0,4,5),2,3)
> kronecker.function <- function(M) kronecker(K,M)
```

Just to check, consider

```
> M <- matrix(c(-1,1,0,2),2,2)
> kronecker.function(M)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   -1    0   -3    0   -4    0
[2,]    1    2    3    6    4    8
[3,]   -2    0    0    0   -5    0
[4,]    2    4    0    0    5   10
```

(40) Compute $\sum_{i=10}^{20}(i^2+4/i)$.

```
> i=10:20
> sum(i^2+4/i)
[1] 2588.075
```

(41) Solve numerically the following system

$$\begin{cases} 3x+2y-z=1 \\ 2x-2y+4z=-2 \\ -x+\frac{1}{2}y-z=0 \end{cases}$$

```
> A <- matrix(c(3,2,-1,2,-2,.5,-1,4,-1),3,3)
> A
     [,1] [,2] [,3]
[1,]    3  2.0   -1
[2,]    2 -2.0    4
[3,]   -1  0.5   -1
> b <- c(1,-2,0)
> solve(A,b)
[1]  1 -2 -2
> A %*% c(1,-2,-2)
     [,1]
[1,]    1
```

```
[2,]   -2
[3,]    0
```

(42) Given a vector $\boldsymbol{x}$ in $\mathbb{R}^n$ and a function $f : \mathbb{R} \to \mathbb{R}$, create a function `sum.function(x,f)` which computes $\sum_{i=1}^{n} i \cdot f(x_i)$.

```
> sum.function <- function(x,f){
sum((1:length(x))*Vectorize(f)(x))}
```

(43) Compute $\sum_{i=1}^{10} \sum_{j=i}^{10} i^2/(5 + i * j)$.

```
> s <- 0
> for(i in 1:10) for(j in i:10) s <- s+i^2/(5*i*j)
> s
[1] 6.5
```

or, if we want to use matrices, instead of loops

```
> f <- function(i,j) i^2/(5*i*j)
> i <- rep(1:10,10)
> j <- rep(1:10,each=10)
> j.exceed.i <- which(j>=i)
> sum( f(i,j)[j.exceed.i] )
[1] 6.5
```

(44) Create a function `mat(n)` which returns the $n \times n$ matrix, such that $M_{i,i} = 2$, $M_{i+1,i} = M_{i,i+1} = 1$ and 0 elsewhere.

Consider

```
> mat <- function(n){
+     M <- matrix(0,n,n)
+     diag(M) <- 2
+     diag(M[1:(n-1),2:n]) <- 1
+     diag(M[2:n,1:(n-1)]) <- 1
+     return(M)
+ }
```

```
> mat(5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    1    0    0    0
[2,]    1    2    1    0    0
[3,]    0    1    2    1    0
[4,]    0    0    1    2    1
[5,]    0    0    0    1    2
```

(45) Are `sqrt(7)` and `7^.5` equal? What about `sqrt(7)^2` and `7` ?

```
> sqrt(7)
[1] 2.645751
> 7^.5
[1] 2.645751
> sqrt(7) == 7^.5
[1] TRUE

> sqrt(7)^2 == 7
[1] FALSE

> sqrt(7)^2 - 7
[1] 8.881784e-16

> all.equal(sqrt(7)^2, 7)
[1] TRUE
```

(46) Given

```
> a <- c(-0.2,0.2,0.49,0.5,0.51,.99,1.2)
```

what is the difference between `trunc(a)`, `floor(a)`, `ceiling(a)` and `round(a)`?

```
> trunc(a)
[1] 0 0 0 0 0 0 1
> floor(a)
```

```
[1] -1  0  0  0  0  0  1
> ceiling(a)
[1] 0 1 1 1 1 1 2
> round(a)
[1] 0 0 0 0 1 1 1
```

Here `ceiling` takes a single numeric argument `a` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `a`, `floor` takes a single numeric argument `a` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `a`, `trunc` takes a single numeric argument `a` and returns a numeric vector containing the integers formed by truncating the values in `a` toward 0, and `round` rounds the values in its first argument to the specified number of decimal places (default 0)

(47) Given a vector `x`, use function `ifelse()` to generate a vector with the same length as `x` with the logarithm of elements of `x` that are positive, and `NA` when elements are negative.

One can use

```
> log(ifelse(x>0,x,NA))
```

For instance

```
> x <- seq(-2,2,by=.7)
> x
[1] -2.0 -1.3 -0.6  0.1  0.8  1.5
> log(ifelse(x>0,x,NA))
[1]          NA          NA          NA -2.3025851 -0.2231436
[6]  0.4054651
```

(48) Create a function `anagram(word1,word2)` which returns `TRUE` if `word1` and `word2` are anagrams.

One can use

```
> anagram <- function(word1,word2){
+     t1<-table(sort(unlist(strsplit(word1,""))))
+     t2<-table(sort(unlist(strsplit(word2,""))))
+     n1<-sum(names(t1)!=names(t2))==0
+     n2<-sum(t1!=t2)==0
+     return(as.logical(n1*n2))
+ }
[1] FALSE
```

Just to make sure that this code works well

```
> anagram("abcde","edcba")
[1] TRUE
> anagram("abcde","abcdf")
[1] FALSE
```

(49) Find roots of polynomial $x^2+x=1$.

One can use

```
> polyroot(c(1,1,-1))
[1] -0.618034-0i  1.618034+0i
```

Since roots belong to $\mathbb{R}$ (and not $\mathbb{C}$) we can also use

```
> Re(polyroot(c(1,1,-1)))
[1] -0.618034  1.618034
```

to get values that can be used easily later on. Further, one can also use (that function is more general since it can be used for any function, not only a polynomial function)

```
> f <- function(x) x^2+x-1
> uniroot(f,c(-10,0))$root
[1] -1.618022
> uniroot(f,c(0,10))$root
[1] 0.6180339
```

(50) Given a vector `x`, write a function `which.closest(x,x0)` which returns the element in `x` which is the closest to `x0`.

One can use

```
> which.closest <- function(x,x0){
+ x[which.min(abs(x0-x))]}
```

Just to check that it actually works

```
> which.closest(seq(0,2,by=0.01),sqrt(2))
[1] 1.41
```

where

```
> sqrt(2)
[1] 1.414214
```

(51) Given two vectors `x` and `y` write a function `subcount(y,x,k)` which returns the number of elements of `y` small than `x[k]`.

Consider

```
> subcount <- function(y,x,k) y[y <= x[k]]
```

(52) Create vector of length 100 `'Ins1','Ins2'`, ..., `'Ins100'`.

One can use

```
> paste("Ins3,1:100,sep="")
```

(53) Create vector `c("London (2012)","Beijing (2008)","athens (2004)","Sydney (2000)")` from `c("London","Beijing","Athens","Sydney")`.

One can use

```
> cities=c("London","Beijing","Athens","Sydney")
> paste(cities," (",seq(2012,2000,by=-4),")",sep="")
[1] "London (2012)"  "Beijing (2008)" "Athens (2004)"
[4] "Sydney (2000)"
```

(54) What will the two following lines return?

```
> paste("a", c("b c","d"), sep="")
> paste("a", c("b c","d"), collapse="")
```

The outputs are

```
> paste("a", c("b c","d"), sep="")
[1] "ab c" "ad"
> paste("a", c("b c","d"), collapse="")
[1] "a b ca d"
```

(55) What will the following command return?

```
> grep("ab",c("abc","b","a","ba","cab"))
```

The standard function is

```
> grep("ab",c("abc","b","a","ba","cab"))
[1] 1 5
```

Indeed, only two elements contain "ab" (in that order)

```
> c("abc","b","a","ba","cab")[c(1,5)]
[1] "abc" "cab"
```

(56) Given a vector x, find some functions can be used to return the location of the largest element? and the location of the second largest ?

The standard function is

```
> which.max(x)
```

or equivalently

```
> which(x==max(x))
```

One can also use

```
> which(rank(X)==length(X))
```

For the second largest

```
n <- length(x)-1
which(x==sort(x)[n])
```

or more efficient

```
which(x==sort(x,partial=n)[n])
```

(57) Define

```
> Z <- ts(rnorm(240), start=c(1960,3), frequency=12)
```

Compute the sum of elements of time series `Z` related to `January`.

To get the date, use

```
> time(Z)
           Jan       Feb       Mar       Apr       May       Jun
1960                     1960.167  1960.250  1960.333  1960.417
1961  1961.000  1961.083  1961.167  1961.250  1961.333  1961.417
1962  1962.000  1962.083  1962.167  1962.250  1962.333  1962.417
1963  1963.000  1963.083  1963.167  1963.250  1963.333  1963.417
1964  1964.000  1964.083  1964.167  1964.250  1964.333  1964.417
1965  1965.000  1965.083  1965.167  1965.250  1965.333  1965.417
```

(etc)

To get only observations in January, use for instance

```
> indx <- which(abs(time(Z)-round(time(Z)))<1e-5)
> Z[indx]
 [1] -0.411510833 -0.690953840 -0.279346282 -0.376702719
 [5] -0.115825322  1.100969102 -0.275778029  0.996543929
 [9] -0.972286836 -0.803213217 -0.955839103 -0.448174237
[13] -0.374854762 -0.928971079  0.981116160  1.790485054
[17]  0.816556449 -1.624364530 -0.061707422  0.005641985
```

Just to check,

```
> Z
               Jan           Feb           Mar           Apr
1960                                -1.539950042 -0.928567035
1961 -0.411510833  0.252223448 -0.891921127  0.435683299
1962 -0.690953840 -1.284599354  0.046726172 -0.235706556
1963 -0.279346282  1.757903090  0.560746091 -0.452783973
1964 -0.376702719  2.441364629 -0.795339117 -0.054877474
```

(58) Create a matrix with 4 columns, that contains all combination of 4 terms in `c(1,2,7,6,12,37,59)`, each row being a combination.

The standard R code is

```
> t(combn(c(1,2,7,6,12,37,59),4))
     [,1] [,2] [,3] [,4]
[1,]    1    2    7    6
[2,]    1    2    7   12
[3,]    1    2    7   37
```

(etc)

```
[33,]    7    6   37   59
[34,]    7   12   37   59
[35,]    6   12   37   59
```

(59) Generate a vector

```
> set.seed(1)
> x <- rpois(9,4)
```

What does `unique(sort(x))[1:3]` will return?

The vector is

```
> set.seed(1)
> x <- rpois(9,4)
> x
[1] 3 3 4 7 2 7 7 5 5
```

if we proceed slowly, here is what we compute

```
> sort(x)
[1] 2 3 3 4 5 5 7 7 7
> unique(sort(x))
[1] 2 3 4 5 7
> unique(sort(x))[1:3]
[1] 2 3 4
```

(60) Given a matrix `M` write a function `range.row` that returns a vector whose ith entry the the difference between the largest and the smallest value on the ith row of `M`.

See

```
> range.row <- function(M) apply(M,1,function(x) max(x)-
  min(x))
```

Consider the following example

```
> M <- matrix (sample(1:30),6,5)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]   18   17   22   24    5
[2,]   11   12    1   13   23
[3,]   21    9   20   27    3
[4,]   28    2    6   10   19
[5,]    7   30   25    8   29
[6,]   26   16   15   14    4
>  range.row(M)
[1] 19 22 24 26 23 22
```

(61) Given two numeric vectors `a` and `b`, write a function that returns the matrix $M = [m_{i,j}]$ with $m_{i,j} = a_i \cdot b_j$.

See

```
> ?outer
```

for instance

```
> a=1:4
> b=1:5
> outer(a,b,function(x,y) x*y)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    4    6    8   10
[3,]    3    6    9   12   15
[4,]    4    8   12   16   20
```

(62) Write a code to estimate the maximum likelihood estimator of a mixture of two Gaussian distributions for sample `sample.x`,

```
> set.seed(123)
> sample.x <- rnorm(427,mean=sample(c(-1,+2),size=427,
+ replace=TRUE))
```

The first step is to define the log-likelihood (actually the opposite of the log-likehood since most functions seek a *minimum*, not a *maximum*

```
> dMixGauss <- function(param, obs) {
+     p <- param[1]
+     m1 <- param[2]
+     sd1 <- param[3]
+     m2 <- param[4]
+     sd2 <- param[5]
+     -sum(log(p*dnorm(x = obs, mean = m1, sd = sd1) +
+    (1-p)*dnorm(x = obs, mean = m2, sd = sd2))) }
> logLik <- function(param) {dMixGauss(param,sample.x)}
```

A popular function for optimization is `nlm`

```
> p0 <- c(.5, mean(X)-sd(X)/5, sd(X), mean(X)+sd(X)/5, sd(
   X))
> fit <- nlm(f = logLik, p = p0)
> fit
$minimum
[1] 441.3367

$estimate
[1]  6041.1951  -949.2413  4002.8215 -5926.7609  7942.2517
```

Here, we have a problem. Maybe just because we could not specify that the probability parameter should lie in $[0, 1]$

```
> dMixGaussC <- function(param, obs) {
+     p <- exp(param[1])/(1+exp(param[1]))
+     m1 <- param[2]
+     sd1 <- param[3]
+     m2 <- param[4]
+     sd2 <- param[5]
+     -sum(log(p*dnorm(x = obs, mean = m1, sd = sd1) +
+                  (1-p)*dnorm(x = obs, mean = m2, sd =
   sd2))) }
> logLik <- function(param) {dMixGaussC(param,sample.x)}
```

The optimazation procedure returns here

```
> fit_C <- nlm(f = logLik, p = p0)
There were 11 warnings (use warnings() to see them)
> fit_C
$minimum
[1] 829.5983

$estimate
```

```
[1]  0.2318832 -1.0018831  1.0465558  2.0372811  0.9342933

$gradient
[1] -3.751666e-06 -3.404195e-05 -6.843659e-06  9.151728e
   -06 -9.094947e-07
```

Here the probability parameter is

```
> exp(fit_C$estimate[1])/(1+exp(fit_C$estimate[1]))
[1] 0.5577124
```

Another popular function for optimization is `optim`

```
> logLik <- function(param) {dMixGauss(param,sample.x)}
> fit_2 <- optim( par = p0,   fn = logLik)
```

This optimazation procedure returns

```
> fit_2
$par
[1]  0.5577248 -1.0016406  1.0468591  2.0371790  0.9347358

$value
[1] 829.5983
```

which is very close to the previous output.Here, the optimization procedure ran fine, but since we dot specify any constraints, it might go wrong in some cases. An alternative is to use `constrOptim` where linear constraints can be added. Here

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^k}\{-\log\mathcal{L}(\boldsymbol{\theta}) \text{ under constraint } \boldsymbol{A\theta}\geq\boldsymbol{b}\}$$

```
> A <- matrix(c(1,-1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1),
    4, 5)
> B <- c(0,-1,0,0)
```

```
> fit_2C <- constrOptim(p0,neglogL, NULL, ui = A, ci = B)
> fit_2C
$par
[1]  0.5575015 -1.0023458  1.0463251  2.0369179  0.9344570

$value
[1] 829.5983
```

(63) Create a function `which.min.mat(M)` which returns the the indices of the minimum of the matrix.

Why not

```
> which.min.mat <- function(M) which(M == min(M), arr.ind
   = TRUE)
```

e.g.

```
> M=matrix(sample(1:20),4,5)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    9   15    4   12    7
[2,]   19   11   20   16    3
[3,]   17   18    1   10    5
[4,]   14    8   13    6    2
> which.min.mat(M)
     row col
[1,]   3   3
```

and observe that with ties

```
> M=matrix(sample(1:10),4,5)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    7    9    6    2
```

```
[2,]    1    5    3   10    8
[3,]    6    2    4    7    9
[4,]   10    8    1    5    3
> which.min.mat(M)
     row col
[1,]   2   1
[2,]   4   3
```

Contacts : charpentier.arthur@uqam.ca.