How to use the R software

Mestiri, Sami

University of Monastir

2019

How to use R software

Sami Mestiri [1]
Applied Economics and Simulation
Faculty of Management and Economic Sciences of Mahdia,
University of Monastir, Tunisia. Rue Ibn Sina Hiboun, Mahdia Tunisia

Abstract: R is a language and software that allows statistical analysis. It includes means that make it possible to manipulate data, calculations and graphical representations. It provides a wide variety of statistical tools (modeling, statistical testing, time series analysis, classification problems, machine learning, ...)
Nowadays, there is no doubt that it is the software par excellence in statistical courses for any level, for theoretical and applied subjects alike.
The goal of this paper is helping to start using the statistical software R. We will cover its benefits, show how to get started and will make interesting recommendations for using R, according to my experience .
JEL codes: C15, C88

Keywords : R software, Statistical analysis, Control Structures.

1 Introduction

R is a software library distributed by the "GNU Public License" (which is a license that fixes the legal conditions of the distribution of the GNU project's software libraries) and drives the language S-PLUS. It presents remarkable defects as possible for the effect of calculus matrix and other complex operations, storage and manipulation of data. It contains numerous functions for statistical analyses and flexible graphics outputs. It is addressed to a large public format of specialists and non-specialists in informatics.

R was initially created, in 1996, by Robert Gentleman and Ross Ihaka of the statistical department of the University of Auckland in New Zealand. Since 1997, it was formed by the "R Core Team" to develop R. It is intended to be used with the exploitation systems Unix, Linux, Windows and MacOS.
It has a logical library and content in 1995. It has a programming language and a statistical function. The base version of R contains a large number of significant statistic functions and graphics, for example, the calculation of a mean or a variance or a tracer of a histogram.

The software is downloaded onto the official web site of R www.R-project.org /. Choose a site near your home (example: in France), the latest possible charges will be more rapid, you will be able to download and install an encrypted legend. For Windows, click on Windows then base. Click on the "exist" file (par example: R-2.14.1-win.exe).

[1] *Email* : *mestirisami*2007@*gmail.com*

The installation program is also charged to your device. It allows you to click on the buttons and provide the instructions. It is important that the name of the version of the downloader is released. It is located on the disk of C, in the series of files connected to program files .

This is where you find the packages based on R. Another item used should be localized: the data file. This is not apparent until now. It contains all the objects that you want to create and store in R. Despite of having been designed for data analysis, R belongs to the top ranks of most employed programming languages, even for general purposes.R provides all the usual statistical tools in its base package, but also the most specific and modern techniques, together with amazing graphical tools, are available in its nearly 20000 packages.

For users with poor or no programming knowledge, GUIs are very helpful. Many of them have been developed, with different purposes. We can highlight the easiness and reproducibility of Jamovi, and the great help R Commander offers to make data analysis in a very simple way, while the user learns programming.RStudio IDE offers a perfect framework to make the most of R for users with programming skills.Some R packages have become tools of great utility in education. For example, exams allows to create exams and lists of personalized exercises in a very simple way, interacting with Moodle and another educational platforms.

There is an important difference in philosophy between R and most other statistical packages. With most packages, a statistical analysis will lead to a large amount of output containing information about the estimation, diagnostic tests, etc. In R, a statistical analysis is normally done as a series of steps, with intermediate results being stored in objects. There is usually minimal output at each step, but the objects obtained at each step can be interrogated by R functions to obtain the information required.

At first, this may seem like a disadvantage, as students will not always be able to find the information they want from an analysis. However, it does teach students the interactive nature of data analysis, and it avoids the confusion that students often experience in being confronted with pages of output that they cannot interpret.

The research paper is organized as follows: We provide creation and manipulation of data in Section 2. Section 3 presents some useful functions. In section 4, we show how created the Graphics. The fifth section is devoted to present how creating functions. And finally, we conclude in section 6.

2 Creation and manipulation of data

When directly connected to the command line, or by editing a text file (with the extension .R) under editor preferred.
The manipulative data under R are stored in a space work. The operating system can be visualized at the tap *getwd()*, which means that the computers are fast enough to use the

command *setwd()*.
At the same time it is not important at the moment of the session to visualize the content of the space to travel to the command *ls()*, and another variable to the command *rm( )*, and you pass the argument the name of the variable. For delete all content variables in the space work using the command: *rm(list=ls())*
By typing *ls()*, we will see that our variables have always been present in the space work . The command *history()* allows you to list the previous commands. Finally, you can save all the commands that can be used to analyse the downloads in a script file with the extension .R, and tap the source ('script.R',echo=T), enter the command, for effect. 'analyse.
It is a language in the command line (interpretation), such as Matlab, and can handle the number types of data in the past commands. what you need to know how to create, manage and treat these objects.
Notes that, as in all languages, certain mots-keys on the servers and none may be used as variable names or functions:
FALSE Inf NA NaN NULL TRUE
break else for function if in next repeat while Although we have only used numeric type variables, there are other ways of representing data (or classes) in a vector: character, integer, logical, complex, list.
Finally, the NULL and NA types play a particular role since they respectively designate the absence of a value (i.e. an empty set) and the default coding of a missing value.

2.1 Vector

Different functions allows to easily generate vectors: c, seq, rep. For example, we can create the vector X = [1 2 3 4 5] in different ways:

$X \leftarrow c(1, 2, 3, 4, 5)$

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

The seq() function is useful for generating sequences of integers, and has several arguments to specify the desired sequence (see ?seq).

$Y \leftarrow seq(1 : 5)$

Y

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

We can duplicate a vector using the rep() command:

$Z \leftarrow rep(X, 2)$

Z

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

The rev() function allows you to reverse the order of the sequence (1 2 3 becomes 3 2 1).

$W \leftarrow rev(X)$

W

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

Arithmetic operations also apply to vectors. For example, we can add two vectors a and b, and perform the classic scalar or member-to-member products:

$a \leftarrow 1:5$

$b \leftarrow 5:9$

$a \times 2$

[1] 2 4 6 8 10

$a + b$

[1] 6 8 10 12 14

$a \times b$

[1] 5 12 21 32 45

Many functions also make it possible to classify the elements (or indices) of a vector, to sum them, etc.

$u \leftarrow c(1, 3, 2, 7, 4)$

u

[1] 1 3 2 7 4

$sort(u)$ # order the elements of vector u

[1] 1 2 3 4 7

$order(u)$ # give the rank of the elements of vector u

[1] 1 3 2 5 4

sum(u) # give the sum of the elements of vector u

[1] 17

length(u) give the length of vector u

[1] 5

Finally, a particularity of R is that the elements of a vector can have names. The names() function allows you to associate a label with each of the elements of a vector:

$x \leftarrow 1:5$

$names(x) \leftarrow c("a","b","c","d","e")$

x

a b c d e

1 2 3 4 5

2.2 Matrix

For matrices, we can either generate a vector of size n, and rearrange it to create a matrix of size (l, m), or directly create a matrix using the matrix() function, specifying in arguments the number of rows and columns, for example:

$M \leftarrow matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), nrow = 2, ncol = 5)$

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]
x	1.00	3.00	5.00	7.00	9.00
y	2.00	4.00	6.00	8.00	10.00

The cbind and rbind functions allow you to manipulate vectors in order to form a matrix

by concatenation on the columns or on the rows.

$x \leftarrow c(1, 3, 5, 7, 9$
$y \leftarrow c(2, 4, 6, 8, 10$
$A \leftarrow cbind(x, y)$

	x	y
[1]	1.00	2.00
[2]	3.00	4.00
[3]	5.00	6.00
[4]	7.00	8.00
[5]	9.00	10.00

$B \leftarrow rbind(x, y)$

	[, 1]	[, 2]	[, 3]	[, 4]	[, 5]
x	1.00	3.00	5.00	7.00	9.00
y	2.00	4.00	6.00	8.00	10.00

To access the elements of a matrix, it's a little different from Matlab: you must specify the indices in square brackets, and the comma serves as a delimiter between row and column indices:

$M[, 1]$
$x \quad y$
$1 \quad 2$
$M[1,]$
$[1] \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$
$M[1, 2]$
$[1] \quad 2$

The matrix product is carried out in the same way as for vectors. For example, if we have a contingency table (the numbers broken down into the modalities of the variables, or, in other words, the frequencies of association between the modalities of two qualitative variables), we can code it in disjunctive form, that is to say, associate a table of indicators with each of the variables: the contingency table is none other than the matrix product of the two tables of indicators.

$x1 \leftarrow matrix(c(1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,$
$0, 1, 0, 0, 0, 1), nrow = 8, byrow = T)$
$x2 \leftarrow matrix(c(1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0), \quad nrow = 8, byrow = T)$
$x1; x2$
$t(x1)\% * \%x2$

	[, 1]	[, 2]
[1,]	2	0
[2,]	2	1
[3,]	1	2

The array() function allows you to generate multidimensional matrices:

$a = array(1 : 20, dim = c(4, 5, 2))$

Under R, this is not the most interesting type of data when working for modeling or statistical testing purposes, and we will see a particular type of data structure that is much more useful.

2.3 The factors

There is a particular type of vector corresponding to the notion of factor which makes it possible to create a sequence of values which presents the modalities of the variable). To do this, we use the functions factor() and gl().
The factor() function is used to explicitly convert a (numeric) vector into a factor.
$v \leftarrow c(1, 2, 1, 2, 2)$
$f \leftarrow factor(v, labels = c("yes", "no"))$
f
[1] yes no yes no no
$Levels : yes$ no
The factor() function is also used to explicitly convert a vector (characters) into a factor.
$u \leftarrow c("yes", "no", "yes", "no""no")$
$g \leftarrow factor(u, levels = c("yes", "no"))$
g
[1] yes no yes no no
$Levels : yes$ no
While the *gl()* function allows you to directly generate a factor vector, by specifying the arrangement of the levels. Here are two ways to create the same factor:
$fbis \leftarrow gl(2, 1, 4, labels = c("A", "B"))$
[1] A B A B
$Levels : A$ B
$x \leftarrow c(1, 3, 2, 4)$
$x[f == "A"]$
[1]12
To see the (numerical) levels of a factor, we can use *unclass()*. The *gl()* function is very useful when it comes to creating a large vector (with many replicates), or when you want to precisely control the arrangement of the factor levels. Note that these functions can essentially be used for balanced plans in terms of numbers and replicas. These two functions allow you to generate nominal qualitative variables, while the *ordered()* function allows you to create an ordinal variable, that is to say one whose modalities are ordered.

2.4 Object lists

The list is a heterogeneous object. It is therefore an ordered set of objects which are not necessarily all of the same mode or the same length. An object of type *list()* makes it possible to group together in the same data structure variables having a different mode

of representation (e.g. numeric and character), and possibly of different sizes
$sami \leftarrow list(alpha = 1:3, beta = c('a','b','c','d'))$
b
$sami\$alpha$
[1] 1 2 3
$sami\$beta$
[1] "a" "b" "c" "d"
This type will not really be used later, but you should know that it is useful for returning multiple values when creating your own functions, and it is this data type that R uses when it returns the result of a statistical test, for example. We can thus easily obtain the value of a t test, without displaying the complete result of the analysis:
$x \leftarrow rnorm(10) + 0.5$
$res \leftarrow t.test(x)$
$res\$p.value$

3 Useful functions

The R language has a very large number (thousands!) of internal functions. This section presents just a few of the basic functions most often used for programming in R and manipulating data. For each function presented in the following sections, we provide one or two examples of use. These examples often fall far short of covering all possible uses of a function.

3.1 Vector manipulation

seq: generation of sequences of numbers
$seq(1, 9, by = 2)$
[1] 1 3 5 7 9
rep: repetition of values or vectors
$rep(2, 10)$
[1] 2 2 2 2 2 2 2 2 2 2
sort: sort in ascending or descending order
$sort(c(4, -1, 2, 6), decreasing = TRUE)$
[1] 6 4 2 −1
rank: rank of the elements of a vector in ascending or descending order
$rank(c(4, -1, 2, 6))$
[1] 3 1 2 4
order: order of extracting elements from a vector to place them in ascending or descending order
$order(c(4, -1, 2, 6))$

[1] 2 3 1 4

rev: reverse a vector

$rev(1:10)$

[1] 10 9 8 7 6 5 4 3 2 1

head: extraction of the first n elements of a vector ($n > 0$) or deletion of the last n ($n < 0$)

$head(1:10,3); head(1:10,-3)$

[1] 1 2 3

[1] 1 2 3 4 5 6 7

tail: extraction of the last n elements of a vector ($n > 0$) or deletion of the first n ($n < 0$)

$tail(1:10,3); tail(1:10,-3)$

[1] 8 9 10

[1] 4 5 6 7 8 9 10

unique: extraction of different elements from a vector

$unique(c(2,4,2,5,9,5,0))$

[1] 2 4 5 9 0

3.2 Finding elements in a vector

The functions in this subsection are all illustrated with the vector

$x \leftarrow c(4,-1,2,-3,6)$

x

[1] 4 −1 2 −3 6

which: positions of TRUE values in a Boolean vector

$which(x < 0)$

[1] 2 4

which.min: position of the minimum in a vector

$which.min(x)$

[1] 4

which.max: position of the maximum in a vector

$which.max(x)$

[1] 5

match: position of the first occurrence of an element in a vector $match(2,x)$

[1] 3

%*in*% membership of one or more values to a vector

$-1:2\%in\%x$

[1] *TRUE FALSE FALSE TRUE*

3.3 District

The functions in this subsection are all illustrated with the vector

$x \leftarrow c(-3.6800000, -0.6666667, 3.1415927, 0.3333333)$

x
[1] -3.6800000 -0.6666667 3.1415927 0.3333333
round: rounded to a defined number of decimal places (default 0)
$round(x)$
[1] -4 -1 3 0 3
$round(x, 3)$
[1] -3.680 -0.667 3.142 0.333
floor: largest integer less than or equal to argument
$floor(x)$
[1] -4 -1 3 0
ceiling: smallest integer greater than or equal to argument
$ceiling(x)$
[1] -3 0 4 1

3.4 Descriptive statistics

The functions in this subsection are all illustrated with the vector
$x \leftarrow c(1, 4, 1, 7, 3, 4)$
x
[1] 1 4 1 7 3 4
sum, prod: sum and product of elements of a vector
$sum(x); prod(x)$
[1] 20
[1] 336
diff: differences between the elements of a vector (mathematical operator ∇)
$diff(x)$
[1] 3 -3 6 -4 1
mean: arithmetic mean (and truncated mean with the trim argument)
$mean(x)$
[1] 3.33
var, sd: variance and type (unbiased versions)
$var(x)$
[1] 5.066
min, max: minimum and maximum of a vector
$min(x); max(x)$
[1] 1
[1] 7
range: vector containing the minimum and maximum of a vector
$range(x)$
[1] 1 7
median: empirical median

$median(x)$
[1] 3.5
quantile: empirical quantiles
$quantile(x)$
0% 25% 50% 75% 100%
1 1.5 3.5 4 7
summary: descriptive statistics of a sample
$summary(x)$
$Min.$ $1stQu.$ $Median$ $Mean$ $3rdQu.Max.$
1 1.5 3.5 3.33 4 7
cumsum, cumprod sum and cumulative product of a vector
$cumsum(x); cumprod(x)$
[1] 1 5 6 13 16 20
[1] 14 4 28 84 335

3.5 Matrix Operations

The functions in this subsection are all illustrated with the matrix
$A \leftarrow matrix(c(2, 1, 4, 3), ncol = 2)$
A
[, 1] [, 2]
[1,] 2 4
[2,] 1 3
nrow, ncol number of rows and columns of a matrix
$nrow(A); ncol(A)$
[1] 2
[1] 2
rowSums, colSums are per row and per column, respectively, elements of a matrix
$rowSums(A)$
[1] 6 4
rowMeans, colMeans : averages per row and per column, respectively, of the elements of a matrix;
$colMeans(A)$
[1] 1.5 3.5
t transposed
$t(A)$
[, 1] [, 2]
[1,] 2 1
[2,] 4 3
det determinant
$det(A)$

[1]2
solve:1) with a single argument (a square matrix): inverse of a matrix; 2) with two arguments (one square matrix and a vector): solution of the system of linear equations $Ax = b$
$solve(x)$
$[,1]$ $[,2]$
$[1,]$ 1.5 -2
$[2,]$ -0.5 1
$solve(x, c(1,2))$
$[1]$ -2.5 1.5
diag: 1) with a matrix as argument: diagonal of the matrix; 2) with a vector as argument: diagonal matrix formed with the vector; 3) with a scalar u as argument: identity matrix p p
$diag(A)$
$[1]$ 2 3

3.6 External product

The outer function calculates the outer product between two vectors. This isn't the most intuitive feature to use, but it's extremely useful for doing multiple operations in a single expression while avoiding loops. The syntax for outer is:
$outer(X, Y, FUN)$
The result is the application of the FUN function (default prod) between each elements of X and each of the elements of Y, in other words
$(X[i], Y[j])$
for all values of indices i and j.
The dimension of the result is therefore c(dim(X), dim(Y)).
For example, the result of the exterior product between two vectors is a matrix containing all the products between the elements of the two vectors:
$outer(c(1,2,5), c(2,3,6))$
$[,1]$ $[,2]$ $[,3]$
$[1,]$ 2 3 6
$[2,]$ 4 6 12
$[3,]$ 10 15 30
The %o% operator is a shortcut for outer(X, Y, prod).

4 Graphics

Ability to see examples of graphics with demo(graphics) or demo(persp). When a graphics function is typed on the console, a graphics window will open with the requested graph.

Partition a graphics window:
by(mfcol=c(nr,nc))
we partition the window into a matrix of nr rows and nc columns and the filling is carried out by column. - mfrow: same but the graphics are drawn in line;
- *layout()*: for more complex partitions
layout(matrix(c(1,2,3,4),2,2)) # to insert into a graph
layout(matrix(c(1,1,2,1),2,2),c(3,1),c(1,3))
layout.show(2) # view the created partition
- *plot (x)*: graph of the values of x (on the ordinate) as a function of the values of x;
- *plot (x,y)*: graph of the values of y (the ordinate) as a function of the values of x;
- *pie(x)*: "pie chart" of the values of x;
- *boxplot(x)*: boxplot of x;
- *hist(x)*: histogram of x (for quantitative x);
- *borplot(x)*: column diagram (for qualitative x) For each function, we have several options but some are common:
- type: "p": points, "l": lines, "b" both, "h": vertical lines, "s": stairs; for stair steps ;
- xlab, ylab: axis names, character variables between " " ;
- main: character type variable; sub: subtitle ;
- *points(x,y)*: adds points;
- *lines(x,y)*: same but with lines;
- *segments(x0,y0,x1,y1)*: draws a line between the points (x0, y0) and (x1, y1);
- *abline(a,b)*: draws a line with slope b and ordinate at the origin a;
- *legend(x,y,legend)*: adds a legend to the point of coordinates (x, y) with the symbols given by legend

5 Functions

General structure for creating functions
The general syntax for defining a function is as follows:
function-name<-function(arg1[=expr1],arg2[=expr2]...)
{
instruction blocks
}
The braces allow you to separate the instructions from the signature of the function, the square brackets allow you to specify default values of the arguments in an optional way.
Example 1: here is an example of simulation of the function $f(x) = x^2$:
square=function(x) { x*x}
square(2)
Example 2: defining the mean of a vector:
mean.vec <- function(x) {

s<-sum(x); # Sum of the elements of x
n<-length(x); # Number of elements of x
res<-round(s/n,2); # rounded result
return(res)
}
Using a text editor using the *fix()* function:
fix(mean.vect)
This command launches a text editor that works with the R system and allows you to define functions. The function code is written from the editor. This method is more comfortable and practical than the first.
The return() function allows you to specify the result of the function, when the corresponding return instruction is not used, R returns the result of the last expression evaluated in the function.

6 Control Structures

Control structures are commands that determine the flow of execution of a program: choice between blocks of code, repetition of commands, or forced exit.

6.1 Conditional execution

if (test. expression) {
statement1
} else {
statement2
} If **condition** is true, **branch.true** is executed, otherwise it will be **branch.false**.
To illustrate control structures, we use these example:
$x \leftarrow x = c(-12, 10, -8, 14)$ # initialization of container x
$y \leftarrow rep(0, 4)$#
$for(i \quad in \quad 1:4)$
{
$if(x[i] > 0) \quad y[i] = x[i] * log(x[i]) \quad else \quad y[i] = 0$
$y = round(y, 3)$ # display the value on the screen
}
y# verification

ifelse(condition, expression.true, expression.false) (ifelse(test, yes, no))
Vector function which returns a vector of the same length as condition formed as follows: for each element **TRUE** of condition we choose the corresponding element of **expression.true** and for each FALSE element we choose the corresponding element of **expression.false**.

To illustrate control structures *ifelse*, we use these example:

```
x <- rnorm(100)
y <- ifelse(x>0, 1, -1)
z <- ifelse(x>0, 1, ifelse(x<0, -1, 0))
```

Multiple conditional execution

```
if. (test.expression1) {
statement1
} else if (test.expression2) {
statement2
} else if ( test .expression3) {
statement3
} else
statement4
```

To illustrate Multiple conditional execution, we use this example:
After an exam, the possibilities are as follows: if you have less than 12, you have a mention, "Passable". If you are between 12 and 14, you have a "Good" mention. If you are over 14, you have a rating, "Very good".

```
note=c(6,12.5,14,7)
for (i in 1:4)  {
if (note[i]<12) print ("Passable") else if(12<= note[i]& note[i]<14)  {
print ("good") }
else print ("Very good")
}
```

6.2 Loops

Loops are and should be used sparingly in R, as they are generally inefficient. In most cases, it is possible to vectorize the calculations to avoid explicit loops, or to rely on the outer, apply, lapply sapply and mapply functions to perform loops more efficiently.
for (variable in suite) expression
Run expression successively for each variable value contained in suite. Again here, we will group the expressions in braces {}. Note that sequence does not have to be composed of consecutive numbers, or even numbers, in fact.

```
ages <- c(12, 18, 32, 2, 4)
names(ages) <- c('Jane','Ed','Bob','Joe','Liz')
for (person in names(ages))
{
message(person, " is ", ages [person], " years old.")
}
```

while (condition) expression
Execute expression as long as condition is true. If condition is false when entering the loop, it is not executed. A while loop is therefore not necessarily always executed.
$z \leftarrow 0$
$while(z < 5)$
{
$z \leftarrow z + 1$
$message(z)$
}
repeat expression
Repeat expression. The latter must include a shutdown test which will use the break command. A repeat loop is always executed at least once.
break
Immediate exit from a for, while or repeat loop.
next
Immediate transition to the next iteration of a for, while or repeat loop.

7 Conclusions

As we mentioned in the introduction, the employment of Statistics as a tool to analyze data from research has become essential, especially in the last years. For most scientific or technological disciplines, even in many social and humanistic fields, there exists the need of handling basic statistical concepts, in order to be capable of extracting conclusions from the researchers' own data or to understand published research. That is the reason why subjects of Statistics, at different levels, are included in very different academic curriculum, with students owning previous knowledges and capacities in a very diverse range. Besides, it is obvious that each statistical technique requires adequate software to perform calculations and create appealing graph reports.

For all these reasons, there is a growing need for a statistical software as simple and versatile as possible. After more than 20 years of experience as students, teachers and researchers, our choice is R, without any doubts. The more we use it, the bigger advantages we find. We highlight the most important ones:
- **Free**. Students can have copies at home.
- **Portable**. Once students invest in learning this program, they can take it with them and install it again wherever they may end up working.
- **Versatile.** The software exists for more platforms than virtually any existing commercial program.
- **General.** A very large number of statistical/econometric tools are available, so the software could be used for many (maybe all) subjects.
- **Cutting-edge.** It includes the very latest methods.

- **Programmable.** It is easy for students to program new methods or develop modifications of existing methods.
- **Matrix language.** The R language handles vectors and matrices directly (as do Gauss, Matlab and Ox). This makes programming much simpler for students and reinforces the matrix notation used in class.
-**Object-oriented** language structure. However, the use of a command console instead of a GUI is a big handicap for a beginner. Due to this, the attempts to introduce the use of R in a first course of Statistics, or in Applied Statistics, can lead to bad results, especially for students with no previous programming experience.

References

[1] The R Project for Statistical Computing. Available online: https://www.r-project.org

[2] Ihaka, R; Gentleman, R. (1996) R: A language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics* , 5(3), 299–314.

[3] The R Graph Gallery. Available online: https://r-graph-gallery.com

[4] Available Packages. Available online: https://cran.r-project.org/web/packages/

[5] R AnalyticFlow: Designed for data analysis. Great for everyone. Available online: https://r.analyticflow. com/en/

[6] Williams, G. Rattle (2009) A Data Mining GUI for R. *The R Journal*, 1(2), 45-55.

[7] Fox, J.(2017) Using the R Commander: A Point-and-Click Interface for R, 1st ed.; Chapman and Hall/CRC Press: BocaRaton, FL.

[8] Mestiri, S. (2023). Initiation à l'utilisation du logiciel R *Éditions universitaires européennes*.DOI: 10.13140/RG.2.2.16938.08641/1

[9] Mestiri, S. (2019). Statistical tests using R Software University of Monastir Press. DOI: 10.13140/RG.2.2.29300.50566

[10] Wickham, H.;Golemund, G.(2017) R for Data Science: Import, Tidy Transform, Visualize, and Model Data, 1st ed.; O'Reilly Media: Sebastopol, CA, .

[11] Tucker, Mary C.;Shaw, Stacy T.; Son, Ji Y. and Stigler, James W. (2022,) Teaching Statistics and Data Analysis with R. Journal of Statistics and Data Science Education