R & Rstudio
Basics for NGS data analysis

Fatma Guerfali, PhD
Institut Pasteur de Tunis
fatma.guerfali@gmail.com

OVERVIEW

PARTS 1 - 2

Part 1 Basics of R

Basics of R langage and few exercises

Part 2 R and RStudio

Basics of RStudio and few exercises

OVERVIEW

PARTS 1 - 2

Part 1 Basics of R

Basics of R langage and few exercises

Part 2 R and RStudio

Basics of RStudio and few exercises

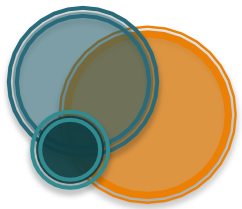
INTODUCTION

R & RStudio

R si a langage

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has:

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either directly at the computer or on hard- copy
- R is very much a vehicle for newly developing methods of interactive data analysis. It has developed rapidly, and has been extended by a large collection of packages.



Working under R

- Create a working directory and start the program (use separate working directories for analyses conducted with R)

```
$ cd exerciseR
$ R
```

- Use the R program

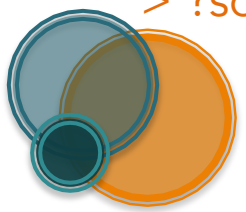
- Terminate with the q() command at the end of the session.

- Inbuilt help similar to the man in Unix

```
> help(solve)
```

An alternative is

```
> ?solve
```



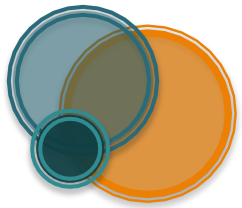
Basic concepts

- Case sensitive

- If a command is not complete at the end of a line, R will give a different prompt, by default:

+

- At the end of each R session you are given the opportunity to save all the currently available objects. If « yes », the objects are written to a file called .RData in the current directory, and the command lines used in the session are saved to a file called .Rhistory.



Basic concepts

- The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions,….

- Vectors are the most important type of object in R
- Matrices or more generally arrays : multi-dimensional generalizations of vectors. In fact, they are vectors that can be indexed by two or more indices and will be printed in special ways.
- Factors provide compact ways to handle categorical data.
- Lists are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists.
- Dataframes are matrix like structures. Think of it as 'data matrices' with one row per observational unit but with (possibly) both numerical and categorical variables.
- Functions are themselves objects in R which can be stored in the project's workspace.
- …

Basic definitions : vectors

- A vector
  - is a data structure
  - It can be constructed using the c() function
  - and assigned to a named object using the <- operator.

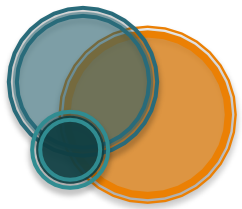
- A vector is, simply speaking, just a collection of :
  - one or more numbers 389.3491
  - more than one number 10 150 30 45 20.3
  - or character strings "Darth Vader" "Luke Skywalker" "Han Solo"

- An "empty" object may still have a mode. For example

```
> e <- numeric()     ##makes e an empty vector structure of mode numeric
> e <- character()   ##is an empty character vector
```

Basic definitions :

```
> c(1.1, 1.2, 1.3)                    # numeric
[1] 1.1 1.2 1.3

> c(FALSE, TRUE, FALSE)               # logical
[1] FALSE TRUE FALSE

> c("foo", "bar", "baz")              # character, single or double quote ok
[1] "foo" "bar" "baz"

> as.character(x)                     # convert 'x' to character
[1] "5" "4" "3" "2" "1"
```



https://www.r-exercises.com/2017/05/01/how-to-create-your-first-vector-in-r/
https://www.bioconductor.org/help/

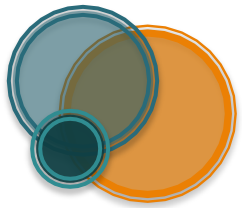
Constructing a vector

- Example of a vector

In R, a vector is considered a data structure (a very simple data structure)

We can construct a vector from a series of individual elements, using the c() function, as follows:

```
> c(10, 150, 30, 45, 20.3)
[1]  10.0 150.0  30.0  45.0  20.3
```



Assigning a vector

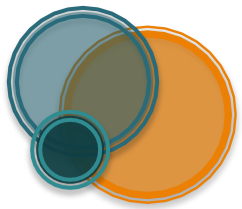
• After constructing the vector, it will be convenient to refer to this vector using a name, instead of having to enter it over and over again.

• We can accomplish this using :

- the assign() function — assign('x', c(10, 150, 30, 45, 20.3))
- or the <- operator — x <- c(10, 150, 30, 45, 20.3)
- or the = operator — x = c(10, 150, 30, 45, 20.3)

• Once done, we can refer to the vector using this name (we call it an « object »)
→ if we type a, R will now show the elements of vector a.

```
> x
[1]  10.0 150.0  30.0  45.0  20.3
```



https://www.r-exercises.com/2017/05/01/how-to-create-your-first-vector-in-r/

Assigning a vector

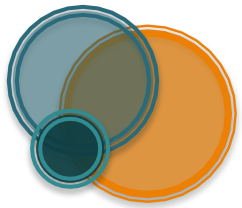
```
> x
> [1]  10.0 150.0  30.0  45.0  20.3
```

The further assignment

```
> y <- c(x, 0, x)
```

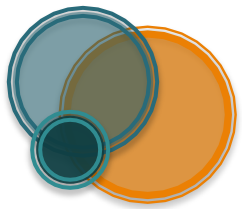
would create a vector y with 11 entries consisting of two copies of x with a zero in the middle place.

```
> y
[1]  10.0 150.0  30.0  45.0  20.3   0.0  10.0 150.0  30.0  45.0  20.3
```



Vectors used in arithmetic expressions

```
> x
> [1]  10.0 150.0  30.0  45.0  20.3

> v <- 2*x
would create
> v
[1]  20.0 300.0  60.0  90.0  40.6

> v <- 2*x + 1
> v
[1]  21.0 301.0  61.0  91.0  41.6
```



https://www.r-exercises.com/2017/05/01/how-to-create-your-first-vector-in-r/

Vectors used in arithmetic expressions

- Two important statistical functions are
  - mean(x) which calculates the sample mean which is sum(x)/length(x)
  - var(x) which gives sum((x-mean(x))^2)/(length(x)-1)

```
> mean(x)
[1] 51.06

> var(x)
[1] 3225.368

> sort(x)
[1]  10.0  20.3  30.0  45.0 150.0
```

Generate regular sequences

• R has a number of facilities for generating commonly used sequences of numbers.
For example 1:30 is the vector c(1, 2, ..., 29, 30).

```
> s3 <- seq(-5, 5, by=.2)  ##generates the vector c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)
> s4 <- seq(length=51, from=-5, by=.2)
```

• A related function is rep() which can be used for replicating an object in various complicated ways. The simplest form is

```
> s5 <- rep(x, times=5) ##put 5 copies of x end-to-end in s5.
```

• Another useful version is

```
> s6 <- rep(x, each=5) ##repeats elements of x 5 times before moving on to the next.
```

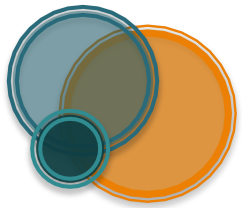
Vectors of character strings

• This possibility only applies where an object has a names attribute to identify its components.

In this case a sub-vector of the names vector may be used

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> lunch <- fruit[c("apple","orange")]
```

→ The advantage is that alphanumeric names are often easier to remember than numeric indices. This option is particularly useful in connection with data frames.



The class of an object

- All objects in R have a class, reported by the function class.

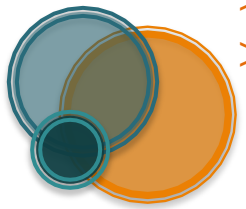
- For simple vectors this is just the mode, for example "numeric", "logical", "character" or "list", but "matrix", "array", "factor" and "data.frame" are other possible values.

- The class of the object is used to allow for an object-oriented style of programming in R.

→ if an object has class "data.frame", it will be printed in a certain way, the plot() function will display it graphically in a certain way, and other so-called generic functions (summary()…) will react to this class in a specific way.

- To remove temporarily the effects of class, use the function unclass(). Ex:

```
> winter          ##will print it in data frame form (rather like a matrix)
> unclass(winter) ##will print it as an ordinary list.
```



Basic concepts : reminder

- The entities that R creates and manipulates are known as objects. These may be variables, arrays of numbers, character strings, functions,….

- Vectors are the most important type of object in R
- Matrices or more generally arrays : multi-dimensional generalizations of vectors. In fact, they are vectors that can be indexed by two or more indices and will be printed in special ways.
- Factors provide compact ways to handle categorical data.
- Lists are a general form of vector in which the various elements need not be of the same type, and are often themselves vectors or lists.
- Dataframes are matrix like structures. Think of it as 'data matrices' with one row per observational unit but with (possibly) both numerical and categorical variables.
- Functions are themselves objects in R which can be stored in the project's workspace.
- …

Index matrices

- R matrix = multi-dimensional generalizations of vectors. In fact, they are vectors that can be indexed by two or more indices and will be printed in special ways.

- In the case of a doubly indexed array, an index matrix may be given consisting of two columns and as many rows as desired.

- Generate a 4 by 5 array:

```
> x <- array(1:20, dim=c(4,5))
> x
```

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

- Exercise:
- Extract elements X[1,3], X[2,2] and X[3,1] as a vector structure
- And Replace these entries in the array X by zeroes.

Index matrices

- « i » is a 3 by 2 index array.

```
> i <- array(c(1:3,3:1), dim=c(3,2))
> i
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
```

- Extract specific elements

```
> x[i]
[1] 9 6 3
```

- Replace them by zero

```
> x[i] <- 0
> x
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
```

NB: Negative indices are not allowed in index matrices. NA and zero values are allowed: rows in the index matrix containing a zero are ignored, and rows containing an NA produce an NA in the result.

Lists

- R list = object consisting of an ordered collection of objects known as its components.

- There is no particular need for the components to be of the same mode or type, and, for example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on. Here is a simple example of how to make a list:
- > Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))

- Components are *always numbered* and may always be referred to as such.

→ if Lst is the name of a list with four components, these may be individually referred to as Lst[[1]], Lst[[2]], Lst[[3]] and Lst[[4]].

→ If, further, Lst[[4]] is a vector subscripted array then Lst[[4]][1] is its first entry.

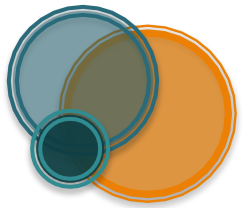
Lists

- R list = object consisting of an ordered collection of objects known as its components.

```
> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

- If Lst is a list, then the function length(Lst) gives the number of (top level) components it has.

```
> length(Lst)
[1] 4
```



Lists

- Components of lists may also be named, and in this case the component may be referred to either by giving the component name as a character string

> name$component_name

- Makes it easier to get the right component if you forget the number. Use:

- in place of the number the double square brackets

Lst[["name"]]

- or, more conveniently, by giving an expression of the form

Lst$name

- So in the previous example :

> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))

Lst$name is the same as Lst[[1]] and is the string "Fred",
Lst$wife is the same as Lst[[2]] and is the string "Mary",
Lst$child.ages[1] is the same as Lst[[4]][1] and is the number 4.

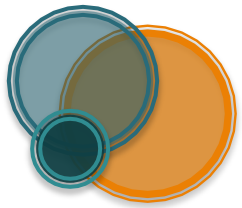
Data frames

- A data frame is a list with class "data.frame". There are restrictions on lists that may be made into data frames, namely:

- The components must be vectors (numeric,character,orlogical), factors, numeric matrices, lists, or other data frames.

- Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.

- Numeric vectors, logicals and factors are included as is, and by default1 character vectors are coerced to be factors, whose levels are the unique values appearing in the vector..

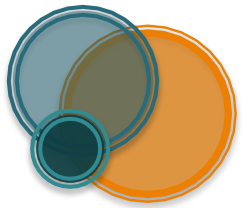


Data frames

- A data frame is a list with class "data.frame". There are restrictions on lists that may be made into data frames, namely:

- Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

- A data frame may for many purposes be regarded as a matrix with columns possibly of differing modes and attributes. It may be displayed in matrix form, and its rows and columns extracted using matrix indexing conventions

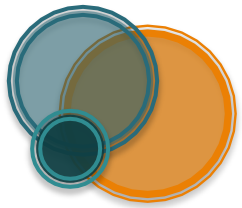


Data frames

- Exercise 1

Create the following data frame
Afterwards invert Sex for all individuals.

	Age	Height	Weight	Sex
Alex	25	177	57	F
Lilly	31	163	69	F
Mark	23	190	83	M
Oliver	52	179	75	M
Martha	76	163	70	F
Lucas	49	183	83	M
Caroline	26	164	53	F



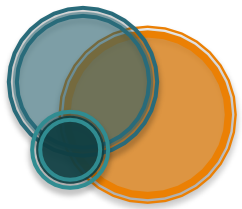
Data frames

- Exercise 1

- Create the following data frame
- Afterwards invert Sex for all individuals

Solution

```
Name <- c("Alex", "Lilly", "Mark", "Oliver", "Martha", "Lucas", "Caroline")
Age <- c(25, 31, 23, 52, 76, 49, 26)
Height <- c(177, 163, 190, 179, 163, 183, 164)
Weight <- c(57, 69, 83, 75, 70, 83, 53)
Sex <- as.factor(c("F", "F", "M", "M", "F", "M", "F"))
df <- data.frame (row.names = Name, Age, Height, Weight, Sex)
levels(df$Sex) <- c("M", "F")
```



Data frames

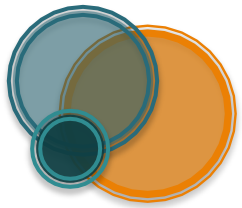
- Exercise 2

- Create this data frame

- Add this data frame column-wise to the previous one.

a) How many rows and columns does the new data frame have?
b) What class of data is in each column?

	Working
Alex	Yes
Lilly	No
Mark	No
Oliver	Yes
Martha	Yes
Lucas	No
Caroline	Yes



Data frames

- Exercise 2

- Create this data frame

	Working
Alex	Yes
Lilly	No
Mark	No
Oliver	Yes
Martha	Yes
Lucas	No
Caroline	Yes

Solution

```
Name <- c("Alex", "Lilly", "Mark", "Oliver", "Martha", "Lucas", "Caroline")
Working <- c("Yes", "No", "No", "Yes", "Yes", "No", "Yes")

dfa <- data.frame(row.names = Name, Working)
```

Data frames

- Exercise 2

- Add this data frame column-wise to the previous one.

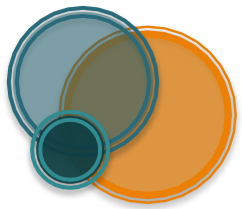
a) How many rows and columns ?
b) What class of data is in each column?

Solution a)

```
> dfa <- cbind (df,dfa)
```

```
> dim(dfa)
[1] 7 5
```

```
# or:
> nrow(dfa)
[1] 7
> ncol(dfa)
[1] 5
```



Data frames

- Exercise 2

- Add this data frame column-wise to the previous one.

a) How many rows and columns ?

b) What class of data is in each column?

Solution b)

```
> sapply(dfa, class)
```

```
##   Age    Height    Weight  Sex   Working
## "numeric" "numeric" "numeric"  "factor"  "factor"
```

Alternative Solution b)

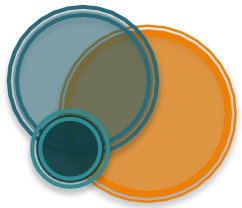
```
> str(dfa)
```

```
## 'data.frame':     7 obs. of  5 variables:
##  $ Age    : num  25 31 23 52 76 49 26
##  $ Height : num  177 163 190 179 163 183 164
##  $ Weight : num  57 69 83 75 70 83 53
##  $ Sex    : Factor w/ 2 levels "M","F": 1 1 2 2 1 2 1
##  $ Working: Factor w/ 2 levels "No","Yes": 2 1 1 2 2 1 2
```

Data frames

- Exercise 3

- How to check for the class of a dataset called X?
- Convert it to a dataframe



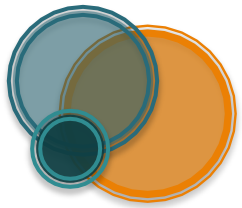
Data frames

- Exercise 3

- How to check for the class of a dataset called X?
- Convert it to a dataframe

Solution

```
> class (X)
[1] "list"

> df <- as.data.frame(X)
```



Data frames

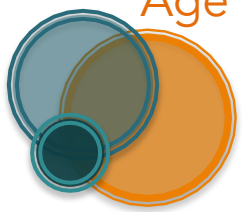
- Exercise 4

- Create a simple data frame from 3 vectors.
- Order the entire data frame by the first column.

Help
Suppose the 3 vectors are:

```
v <- c(45:41, 30:33)
b <- LETTERS[rep(1:3, 3)]
n <- round(rnorm(9, 65, 5))
```

And that they should correspond to the following columns in the dataframe:
Age Class Grade



Data frames

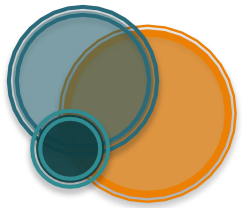
- Exercise 4

- Create a simple data frame from 3 vectors.
- Order the entire data frame by the first column.

Solution

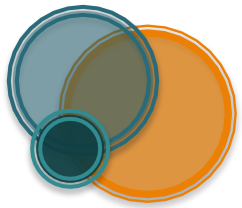
```
> df <- data.frame(Age = v, Class = b, Grade = n)

> df[with (df, order(Age)),]

# alternative solution
> df[order(df$Age), ]
```

```
##    Age Class Grade
## 6   30     C    57
## 7   31     A    64
## 8   32     B    59
## 9   33     C    73
## 5   41     B    61
## 4   42     A    71
## 3   43     C    70
## 2   44     B    63
## 1   45     A    62
```



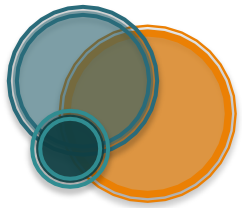
Reading data from files

- Large data objects will usually be read as values from external files rather than entered during an R session at the keyboard.
- R input facilities are simple and their requirements are fairly strict and even rather inflexible.
- Usually, you will be able to modify your input files using other tools, such as file editors or Perl to fit in with the requirements of R. This allows you to simply cope with R requirmeents.



Reading data from files

- Large data objects will usually be read as values from external files rather than entered during an R session at the keyboard.
- R input facilities are simple and their requirements are fairly strict and even rather inflexible.
- Usually, you will be able to modify your input files using other tools, such as file editors or Perl to fit in with the requirements of R. This allows you to simply cope with R requirmeents.



Reading data from files :The read.table() function

- To read an entire data frame directly, the external file would normally have a special form:

- The first line of the file should have a name for each variable in the data frame.
- Each additional line of the file has as its first item a row label and the values for each variable.

```
     Price    Floor     Area    Rooms      Age  Cent.heat
01   52.00    111.0      830      5        6.2       no
02   54.75    128.0      710      5        7.5       no
03   57.50    101.0     1000      5        4.2       no
04   57.50    131.0      690      6        8.8       no
05   59.75     93.0      900      5        1.9      yes
...
```

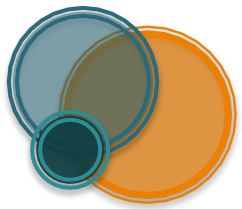
Reading data from files :The read.table() function

- By default numeric items (except row labels) are read as numeric variables and non-numeric variables, such as Cent.heat in the example, as factors. This can be changed if necessary.

- The function read.table() can then be used to read the data frame directly

```
> HousePrice <- read.table("houses.data")
```

```
    Price    Floor     Area   Rooms     Age  Cent.heat
01  52.00    111.0      830     5       6.2       no
02  54.75    128.0      710     5       7.5       no
03  57.50    101.0     1000     5       4.2       no
04  57.50    131.0      690     6       8.8       no
05  59.75     93.0      900     5       1.9      yes
...
```



Reading data from files :The read.table() function

- Often you will want to omit reading the first line as variables. The data frame may then be read as

```
> HousePrice <- read.table("houses.data", header=TRUE)
```

#header=TRUE option specifies that the first line is a line of headings, and hence, by implication from the form of the file, that no explicit row labels are given.

```
    Price    Floor     Area   Rooms     Age  Cent.heat
01  52.00    111.0      830     5       6.2       no
02  54.75    128.0      710     5       7.5       no
03  57.50    101.0     1000     5       4.2       no
04  57.50    131.0      690     6       8.8       no
05  59.75     93.0      900     5       1.9      yes
...
```

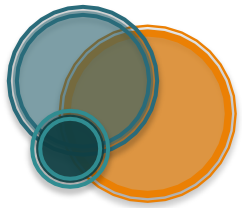
Reading data from files and making simple plots

- Simple dataset available for download

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/chol.txt?tap_a=5644-dce66f&tap_s=10907-287229

- Exercise:

- Download the data table into a file called test.txt
- Place it using the read.table option in an object called « X »
- Draw a histogram of the « AGE » data column



https://cran.r-project.org/doc/manuals/R-intro.pdf
https://www.r-bloggers.com/how-to-make-a-histogram-with-basic-r/

Reading data from files and making simple plots

- Exercise:

- Download the data table into a file called test.txt

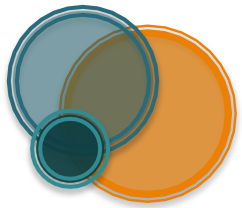
make sure the file is placed into the current working directory (exerciseR) and named correctly

- Place it using the read.table option in an object called « X »

```
> X <- read.table("test.txt", header=TRUE)
```

- Draw a histogram of the « AGE » data column

```
> hist(X$AGE)
```

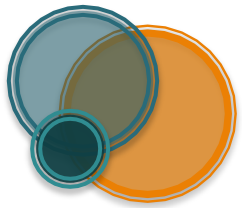


Reading data from files and making simple plots

- Exercise:

- Modify the output of your histogram using the following parameters

```
> hist(X$AGE, main="Histogram for AGE data", xlab="AGE", border="blue", col="lightblue", xlim=c(10,60), breaks=5)

> hist(X$AGE, main="Histogram for AGE data", xlab="AGE", border="blue", col="lightblue", xlim=c(10,60), breaks=10)

> hist(X$AGE, main="Histogram for AGE data", xlab="AGE", border="blue", col="lightblue", xlim=c(10,60), breaks=20)
```



Reading data from files and making simple plots

- Exercise:

- Probability Density

By default: hist() function shows you the frequency of a certain bin on the y-axis. However, if you want to see how likely it is that an interval of values of the x-axis occurs, you will need a probability density rather than frequency.

 # probability density expressed through the y-axis instead of the regular frequency

```
> hist(X$AGE, main="Histogram for AGE data", xlab="AGE", border="blue", col="lightblue", xlim=c(10,60), breaks=20, prob = TRUE)
```

 # then add a density curve to your dataset by using the lines() function

```
> lines(density(X$AGE))
```


https://cran.r-project.org/doc/manuals/R-intro.pdf
https://www.r-bloggers.com/how-to-make-a-histogram-with-basic-r/

OVERVIEW

PARTS 1 - 2

Part 1 Basics of R

Basics of R langage and few exercises

Part 2 R and RStudio

Basics of RStudio and few exercises

INTODUCTION

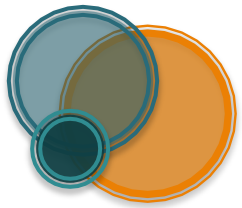
R & RStudio

R si a langage

RStudio is a (software products related to the R langage)

RStudio IDE (Integrated Development Environment)
is a product. There are 3 versions of this product:

- Rstudio Desktop IDE (local version to install on your own computer)
- RStudio Server Open Source
- Rstudio Server Pro

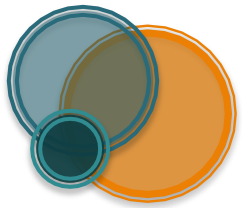


RStudio

an open source project maintained on Github

Runs on all major platforms (Windows, Mac, and Linux) and can also be run as a server, enabling multiple users to access the RStudio IDE using a web browser.

You can work:
- at the interactive prompt as you would classicly with R installation
- or you can write and run R Code in an editor

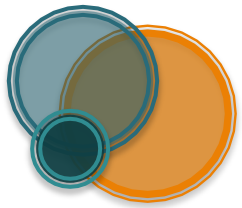


RStudio

Different panels exist to display environment and project settings, so you have immediate visibility and a generally superior presentation that simply working at the command line.

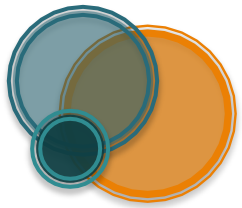
You can :
- generate plots
- generate HTML files and entire HTML presentations
- …

→ RStudio makes R easier to use. It includes a code editor, debugging & visualization tools.



RStudio

- Open Rstudio
- Get familiarized with the display
- Get familiarized with how to load packages



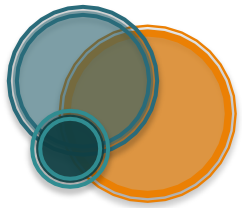
RStudio

Exercise

#Load library ggplot and the diamond dataset included
> library(ggplot2)
> data(diamonds)

#Have a look at a portion of the dataset
> head(diamonds)

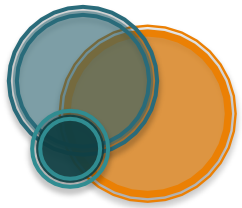
#Have a look at some statistics on the dataset
> summary(diamonds)



RStudio

Exercise

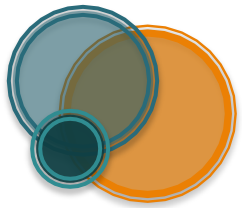
```
# How many observations are in the data set?
> nrow(diamonds)
[1] 53940

# How many variables are in the data set?
> ncol(diamonds)
[1] 10

# Alternative solution
> dim(diamonds)
[1] 53940    10
```



RStudio

Exercise

```
# How many letters represents the best color for a diamonds?
> levels(diamonds$color)
[1] "D" "E" "F" "G" "H" "I" "J”

# Create a histogram of the price of all the diamonds in the diamond data set.
> ggplot(diamonds, aes(x = price)) +
+     geom_histogram(color = "black", fill = "DarkOrange", binwidth = 500) +
+     scale_x_continuous(breaks = seq(0, 20000, 1000)) +
+     theme(axis.text.x = element_text(angle = 90)) +
+     xlab("Price") + ylab("Count")
```



RStudio

Exercise

 # Try limiting the x-axis
 # Try altering the bin width
 # Try setting different breaks on the x-axis.

> ...