R language

Fondamentals

R studio is a software to use R: R is the programming language.

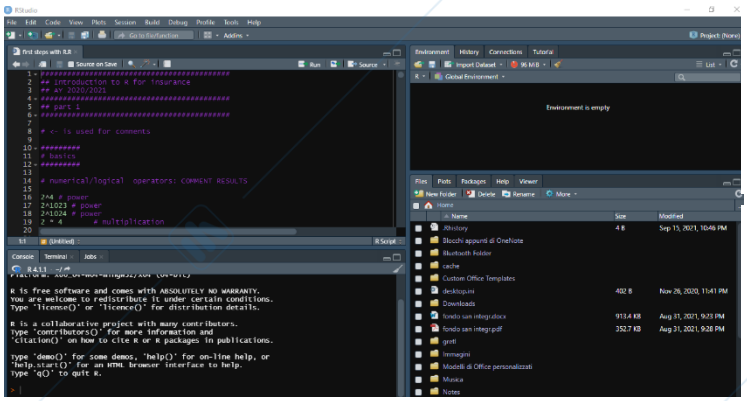
We will use a simple approach because we are not going to use (for now, we will use it in the 2nd module) the Pytline command system which is a very modern way to write a code in R. "pytline" stands for a sort of concatenation of commands which are sequentially applied to just one row in order to achieve a sort of specific result.

Click on BB on "a crash course on R" -> it will open a window: let's open the file using R studio ->it asks sometimes also "what does firefox do with this file? Then if you chose file it's probable that the system will save it in the download directory of our computer or you can simply open with r) -> ok

There are typically 4 windows (but it's possible to reduce the number of them):

Window of the commands: It contains the R codes, so here we are going to type the instructions

Window of the output: you get the output of the code you have run above



Environment window: here you can see something about the dataset uploaded to your memory, the constants and the objects prepared into your code = it's possible to have a sort of quick control of what you have prepared in the previous section.

4th window is made of 4 sections:
->files: to manage the files available in our hard disk
->plots: so to have a quick view with a map, an histogram, a diagram and so on
->helps: in case you have a command with a specific grammar that you don't know well here you can have a nice explanation of it

Some basics:

- To write **comments** (very useful while your coding) write before an #
- To execute commands type ctrl enter (in my system, but it might be different as shift enter and so on; not only enter because it means "go to the next row" not "execute the command")
- Execution of a **sum:** … + … ctrl enter.
- Execution of a **power:** …^… ctrl enter;
  Running for instance 2^1024 R returns us "Inf": the specific result is a very very large number - impossible to compute using my system – and so modern languages (instead of returning for instance "impossible" or "not available") return inf, to signal it's roughly an infinite number / If we would run -1*(2^1024) we expect to obtain instead -inf
  !!! "+Inf" or "-Inf" can be used into the R language as objects (pay attention to use the capital I so that R can recognize them correctly) so they corresponds to specific algorithms that returns a very huge number.
- Execution of a **multiplication**: … * … ctrl enter.
  +Inf * 0 returns us "Nan" = not a number (! Is not 0) while +Inf*-Inf= -Inf
- Execution of a **division**: … / … ctrl enter.
  1/0 returns Inf
  !!! so 0 in this case is seen by R not just as a number exactly equal to 0 but it's a floating number. So I have not said to R that 0=0 but used 0 in that manner.

So if you have a bit o knowledge of how a computer works and of its architecture u know that every type of number that we insert into the system is interpreted not exactly equal to that number, but as a number plus a sequence of 0 (typically 15) and then an additional closing number used to control the quality of numbers added to our code. For instance "1" stands for 1,0000000000000001

Inf/Inf returns "Nan", so we cannot say what's the output of that ratio

- Execution of some **logic commands**:

  ... == ... ctrl enter -> "that number is equal to the next one", true or false?

  ... != ... ctrl enter -> "that number is NOT equal to the next one", true or false?

  ⇨ They are Boolean commands: they return as results "true" or "false"

  ⇨ So for 1 == 1 we get true and for 1 != 2 we get false

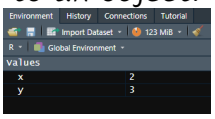
  ! If we run 1=1 we get an error because it's not the correct way to express that command

  !! these commands can be really useful using a dataset: we have to know how to fill a dataset or how to select a specific detail that comes from it (filtering option of a dataset, for instance selecting all the females in a cluster of the data)

- Creating an **object**: x -> ... or ... <- y (or even x = ...) ctlr enter

  The arrow is used to attribute to the object (x for instance) the number, the direction of the arrow is used to attribute something to an object.

  We can see our objects in the environment window

  Obviously the object can be...

  ➔ A scalar: for instance x <-4

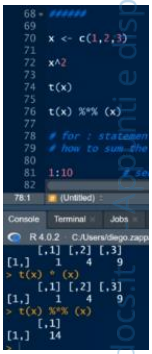
  ➔ A vector:

  here c is a specific command that stands for "concatenate"; o it concatenates some elements reported inside the brackets, separated by commas

  ↳ A *numerical vector* = the elements are numbers, as in x <- c(1,2,3)

  We can use this kind of vector for some basic operations...

  - The **power**: x^2 ctrl enter -> we obtain 1 4 9 so it applies to each element the power
  - The **transposition**: t(x) ctrl enter, where t stands for "transpose" -> we obtain a row vector with one row and 3 columns
  - The **multiplication**: x * x ctrl enter -> we get again 1 4 9

    But !!! ... The trick is that the product of vectors cannot be executed using just the asterisk (*): in order to be able to do linear product of matrices or vectors of linear algebra using the R commands we need to add the percentage before and after the asterisk

    t(x) * x ctrl enter -> we get something similar to 1 4 9 => so we have to use t(x) %*% x and we will do so a row vector times a column vector, so 1*1 + 2*2 + 3*3 = 14

    x * t(x) ctrl enter -> we get something quite similar to 1 4 9 but now in a vector notation => so we have to use x %*% t(x), we obtain a matrix, because now we are multiplying a column vector times a row vector.

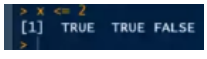
    

    x is a numerical vector, but if we execute some logical commands, we can obtain logical vectors. We can for instance...:

    ->x ==1 ctrl enter: R checks whether the elements of the vector x (so: 1,2,3) are equal to 1 or not... the result is

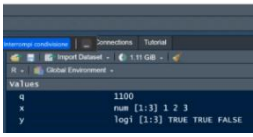
    ->x<=2 ctrl enter: R checks whether the elements of the vector x (so: 1,2,3) are less or equal to 2 or not...so we get

    If we want those results to become new vectors (logical vectors) we can make them new objects. For instance we can...:

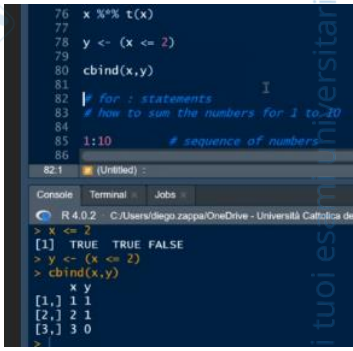
    So that in the environment window we will have

  ↳ A *logical vector* = the elements consist of "true" (1) or "false" (0)

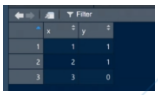
(Boolean vector)

We could chose to join/combine the vectors (x and y)...

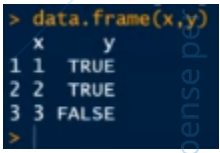


- **cbind(x,y)** ctrl enter -> c corresponds to concatenation and bind stands for link, here with vectors it stands for the concatenation of a column (column bind). cbind must be used only when the variables that you are willing to join are of the same type (= all numeric, or all characters or all boulean). Typically, when you combine vectors of different types you have that some of them will be changed into the most weak? type of vector present
  Here the logical vector is interpreted numerically using as true 1,as false 0 (we can rename this as a new object, X: a numerical object that contains 3 rows and 2 columns. If we want to have a nice view of that object it's possible to click on the very small item reported in the environment window (next we can close it)

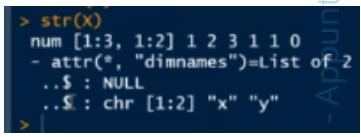
```
X <- cbind(x,y)
```



- **data.frame(x,y)** ctrl enter -> it's used to prepare a dataset
  Here we have the opportunity to save the original meaning of the variable y, without transformation.
  Data frame is very similar to a dataset/database: in it you have many columns (variables) and a lot of rows (statistical units).

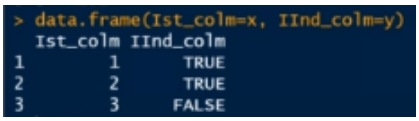


How to have a specific view of the characteristics of an object? **str(X)** ctrl enter -> str stands for structure, and says to us that this is a numerical matrix 3 rows 2 columns, where the two columns are named x and y.

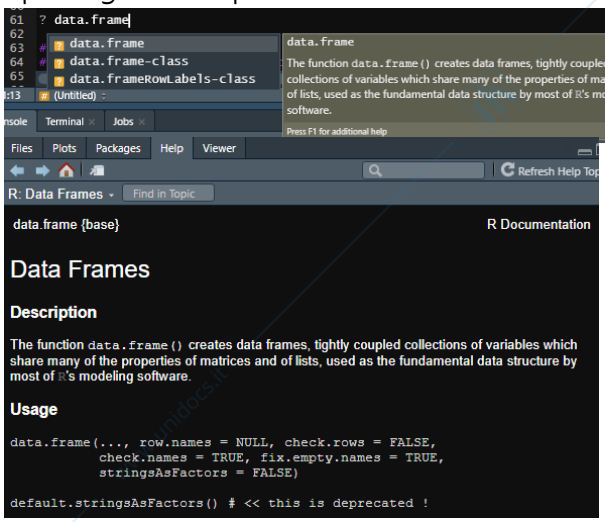


It's also possible to change the name of the columns:

```
data.frame(Ist_colm=x, IInd_colm=y)
```



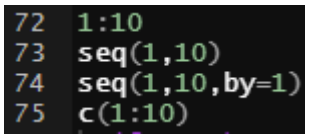
- Opening the help section to learn about a command: ? ... ctrl enter

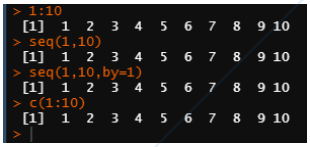


So for instance: ?data.frame or ?cbind
Doing so you automatically are asking to R to enable the help system: in it we can find an explanation of the function of the command, how to use it, details about it (for instance data frame contains lots of arguments), and even some examples. They are really useful because you can run the commands there proposed and observe what happen and so learn.

It's really useful because there are a lot of commands that we can use and we may not know them all.

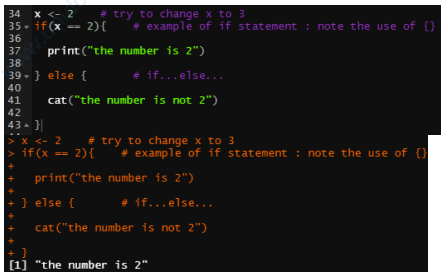
- Getting a **sequence of integers**: there are 4 equivalent ways, supposing a domain from 1 (starting point) to 10 (end/last point)
  ! in reality they are not exactly the same...it depends on what we want to achieve with our code. So, according to our problem we can apply one of the strategy proposed here, choosing the best.



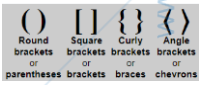


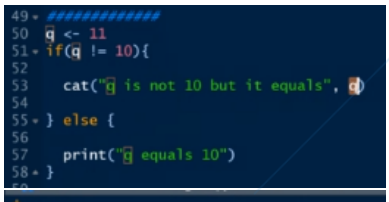
  You can also... :
  - Typing seq(1,10,) and pointing the cursor where there is nothing after the last command and pressing "tab"  , it will be opened a brand new menu where we have the objects available in our memory. So 'it's a sort of strategy to insert every quickly additional detail to your command
  - **seq(1,10,length.out=...)** ctrl enter allows us to split the domain proposed into a certain number of regular (length -1) intervals/classes/subsegments by proposing into the output the number of extremes we selected as length
    If we select a length of 5, we would have our domain (1-10) divided into 4 groups, classes by a selection of 5 equally distant extremes.
  - **seq(1,10,by=...)** ctrl enter we would get 1 and 6 . what? Try to do that with seq(1,10,by=1) by 1 corresponds again simply to seq(1,10) or 1:10 so to the sequence so from 1 up to the maximum adding 1 to which step of the command.
    With seq(1,10,by=5) we would obtain 1 and 6: so from the starting point (1) we have summed +5 obtaing 6. 6+5=11, out of the domain and so not reported
    With seq(1,10,by=2) we get 1 3 5 7 9. 9+2=11, outside the domain and so not reported

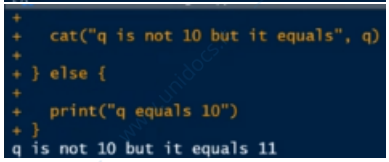
- Execution of the **if statement:** fundamental in every language



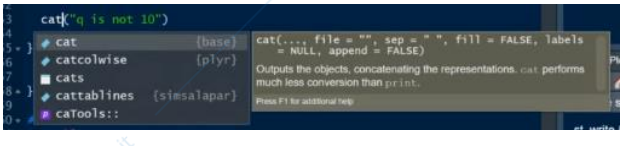
we start from an object and "if" something happens R run a command, if it doesn't, R runs something "else". It's very important to use correctly the parentheses: when we open them R automatically remind us to even close them;
insert square brackets: alt Gr and the key
insert curly brackets: shift, alt Gr and the key
If we did: x <- 4, we would expect the application of the "else" statement: we would obtain "the number is not 2".







->**print()**: R "print" what is written between the round brackets and the inverted commas
->**cat()**: = we have not used the written "11" in the command (row 53) but – thanks to the concatenation – the output embodied it (because we used a variable, available in our memory).



!!! To run this sequence of commands we need to select all those rows and press ctrl enter = so R know to execute them all in one shot (not one up to the other, but jointly!)
!!! The "if statement" consists of the "if" and "else" statement, but in reality it's possible to use it even without the else statement. In that case R execute the first part is the statement is correct, it executes nothing if it's false.

```
# 0
# 0+1
# 0+1+2
# 0+1+2+3
# 0+1+2+3+4
# 0+1+2+3+4+5
# ...
# 0+1+2+3+4+...+10
```

Summing a sequence of integers

if you want to sum numbers from 1 to 10 (i.e) you have to implement such a loop. We will use as starting point 0 (neutral scalar, since 0+... returns ...). Then, considering the sequence of rows we observe that each of them contains the output proposed in the row just reported above plus something else (i.e: 0+1+2 of the 3rd row contains the info in the previous one +2) => So this is a *concatenating process*: we're updating the summation adding a brand new number to the part of summation used before.
Here we have few numbers, and we could show this sequence and compute the result manually, but if they were much more we would need an *algorithm*...

How can we built it?

We can reproduce that idea of concatenation using the object "j".
At the beginning j= 0 (starting point, scalar), then – step by step – we sum the numbers of that domain (1-10) to the updated value of "j"
! you have to press ctrl enter after each row: you will see that in the environment window at the beginning you have j=0, then 1, then, 3, 6 and so on.
-> this is an algorithm because you are concatenating the same variable.

```
# i.e.
 j=0    # (j=0)
 j=j+1  # (j=0+1)
 j=j+2  # (j=0+1+2)
 j=j+3  # (j=0+1+2+3)
# j=j+4  (j=0+1+2+3+4)
# j=j+5  (j=0+1+2+3+4+5)
# ...
# j=j+10
```

Now we want to understand how to execute/shrink this procedure in a more elegant and quick manner (=all in one shot)
Underlying idea: A BOX We have 10 pieces of paper, numbered from 1 to 10. At the beginning our box is empty, then Zappa (our i) picks up one by one the pieces of paper and insert them into the box (=puts in progressively the numbers to sum) = he picks up the piece "1" and put into the box, now it's not anymore empty, it values 1! Then it picks up "2" and put in it: its value is updated to 3 (0+1+2) and so on!

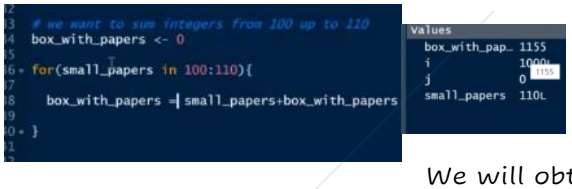
```
94  ########################
95  j=0
96  for(i in 1:10){
97    j=j+i          # i =1,2,3,4,5,6...,10
98  }
99
100 print(j)
101 # or simply
102 j
```

! There's no problem in mixing the sequence of numbers: there's no need to have a strict order from 1 to 10 to obtain the overall summation. We can even think to a random selection of papers.
The summation will be all the same.
(We can imagine I as "Zappa" or as another box from which the numbers are picked up)

Our starting point is again j=0 *(empty box).*
"for" statements now: that "for" means "for all the arguments we propose inside the brackets". i = 1,2,3, ...,10 and so are the numbers we have progressively to sum.
The key command so is now: j=j+i. Print(j) ctrl enter to show us the result: 55!!!
I.e (silly): for(i in1:1) means for any arguments available in the silly vector 1:1 (=1, scalar) So adding 1 to 0 the output will be 1.
I.e: The result we get is 500503 !!!

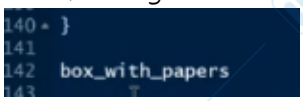
```
109 ##### how to sum numbers from 1 to... 1000 !!!
110 j=0
111 for(i in seq(1,1000,length.out = 10000)){
112   j=j+i
113 }
114 j
```

I.e:

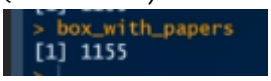


When you apply the procedure of adding small papers to the empty box your new box will contain what present at the beginning of the process (nothing) plus the paper that you have selected. We will obtain as result 1155

In order to get now the output (so not only in the environment window) let's type

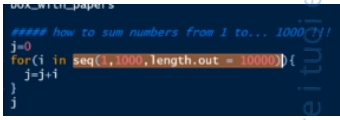


➔

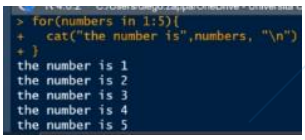


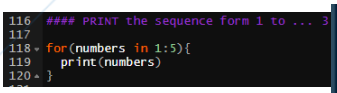
What's evident is that such a procedure can be now generalized and used in many relevant contexts.

I.e: we could choose for instance to sum one by one the extremes obtained using the command length.out in a sequence
! We can select only that part and execute it = so we discover that it's possible to write complex codes and to execute only a part of them by selecteing the part we want and doing ctrl enter. It's not necessary to "compile" = transform a code into the language of your operation system.



- Execution of the **for statement:**
"for" statements now: that "for" means "for all the arguments we propose inside the brackets". we can have for instance for (i in x) were x is a vector = for all the elements of the vector
Then we can apply different things to what we apply the "for":
- a sum (as seen before)
- the "print" command
- the "cat" command





- **Executing a factorial of integers**
3! = 1*2*3 … we want to prepare a simple code in R to do that.
The factorial needs a couple of objects:
  - First of all for we need to specify the number that you want to consider in your computation: here n <- 3

!!! attention there in the file there is a big error: it's j=i*j
Here you're not going to sum numbers, but to multiply them
Also here we would have a starting point: 1 (here the neutral factor can't be 0!)
i.e: 5! = 120, 69! = 5,075802+83, so a very huge number

- **Obtaining the position of an element in a vector**
We should have a vector (ie y) that will contains some elements.
y[*numb of the position*] ctrl enter -> we will obtain as output the element that is in the position recalled.
I.e. y <- c("a","b","c","a",b","c") y[2] ctrl enter -> b y[5] ctrl enter -> b

EX1: **multiply only odds integers between 1 and 100**

Hint use the trick: number %% 2 to compute the module of a division *8 %% 2 = 0 7 %% 2 = 1*

```
> j <- 1
> for (i in 1:100){
+   if((i %% 2 != 0)){
+     j=j*i
+   }
+ }
> print(j)
[1] 2.725392e+78
```

(if we want to copy the text of the exercise in R we can simply do that, but remind to use before every row an #, so that R knows that it's a text and not a command for him)
We wil use as starting point j <-1 because here we will have a product.
Then we can apply a for statement for I that ranges between 1 and 100 BUT we have a problem: we want to have the product of only the odd numbers, not of all the numbers in the range!!!
We can separate the possible scenarios (odd or even number)...
-*Adding an if statement* (inside the for statement!): we want to pick up only odd numbers, otherwise (else statement) do nothing
-*Using the trick of number %% 2* (number %% number it's the module of the division and tell us the rest of the division. So, if the division is exact, we will have as output 0. If it is not, we will get the rest of the division. i.e. 4 %% 3 -> 1 5 %%3 -> 2)
We can use so this trick to identify and separate odd and even numbers: If we set number %% 2 we will obtain that if the number is an even number, the rest will be 0; if the number is an odd number there will be a rest and so we will have an output different from 0.
We want to pick up so only the cases in which i %% 2 is different from 0 because that means that the division with 2 is not exact and so that we have an odd number -> So this is a sort of algorithm that can be implemented in order to select only the odd numbers

```
j=1
n <- 7
for(i in 1:n){
  if((i %% 2 != 0) ){
    j=j*i
  }
}
j
```

! if you wish to generalize your program, so to do for instance the same example with a generic (so for a generic number that you want to insert exogenously, - say - according to a sort of input of your program) you can do so:
!! Another way to do the exercise was using "seq()"

EX2. Generate 2 vectors: X <-c(1,2,3,4,5,6) y <- c("a","b","c","a","b","c")

```
> x <- c(1,2,3,4,5,6)
> y <- c("a","b","c","a","b","c")
>
> for (i in x){
+   if((i %% 2 == 0)){
+     print(y[i])
+   } else {
+     print(y[i])
+   }
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "a"
[1] "b"
[1] "c"
```

If x is 2,4,6 print y in position 2,4,6 else if x is 1 3 5 print y in position 1,3,5
Hint: loop x position from 1 to 6 and inside the for loop insert the condition if-then-else
Insert the vector in R (even with ctrl c ctrl v ctrl enter)
x is a numerical vector y is a character vector
We have to find a way to impost that if statement.
We have different manners to do this exercise.
ΔIt's possible to use the for statement -> (!!! this is not an exercise related to something similar to the computation of a factorial, of a summation of integers so we don't need 2 boxes, but just one in which we have x and y). Here we will have the range of the i of the for statement that is about all the elements of the vector x, sequentially. We need now to propose the if statement, and so we have to add an additional block (open curly brackets). 2 4 6 are even numbers, while 1 3 5 odd numbers, so we can use this aspect to separate the two cases. We know that a y[] returns the element of y in that position. Running the first part of our command we get in fact this output. To finish our exercise, we need the else statement, for the position 1,3,5.

```
> for (i in x){
+   if( (i %% 2 == 0) ){
+     print(y[i])
+   }
+ }
[1] "b"
[1] "a"
[1] "c"
```

In case you're willing to have adding commands to understand well the output you can add the cat statement for instance like that.
...it's not a very clear output: we have to remind using also "\n"

```
for( i in x){

  if( (i %% 2 == 0) ){
    cat("the index in not an odd number", y[i], "\n")
  } else {
    cat("the index in an odd number", y[i], "\n")
  }

}
```

```
  if( (i %% 2 == 0) ){
    cat("the index in not an odd number", y[i])
  } else {
    cat("the index in an odd number", y[i])
  }

}
the index in an odd number athe index in not an odd number bthe index in an odd
 number cthe index in not an odd number athe index in an odd number bthe index i
n not an odd number c
```

```
the index in an odd number a
the index in not an odd number b
the index in an odd number c
the index in not an odd number a
the index in an odd number b
the index in not an odd number c
```

```
XY                    6 obs. of 2 variables
```

```
6 6 c
> XY <- data.frame(x,y)
> view(XY)
> str(XY)
'data.frame':   6 obs. of  2 variables:
 $ x: num  1 2 3 4 5 6
 $ y: chr  "a" "b" "c" "a" ...
>
```

ΔIt's possible to combine the 2 vectors...using dataframe(x,y) (we can make it an object, XY)

*why not cb*ind(x,y)? -> y is a character vector, x is numerical so using cbind(x,y) here all the vectors are transformed into character ones and that's not good

We can also do str(XY) then to see its characteristics.

⌘ Working with a distribution

Typing ?rnorm for instance we have the explanation of the command, with the summary of all the commands related.

- ❋ dnorm() -> density
- ❋ pnorm() -> distribution function
  pnorm (0,0,1) => 0,5 pnorm (1.64,0,1) => 0.9494974
  pnorm (-1.64,0,1) => 0.05050258= 1- 0.9494974
- ❋ qnorm() -> cdf (=quantile function)
  qnorm (0.5,0,1) => 0
- ❋ rnorm() -> random generation of numbers

*In all modern languages we need not the variance but the standard deviation.

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

⌘ Generating random numbers from a distribution

I.e. runif(3,0,1) ctrl enter -> using a U(0,1) the "min=0" and "max=1" are not so necessary so r + abbreviated name of the distr + (n° of random numbers that we want to generate, parameters of the distribution -> arguments required) ctrl enter

```
> runif(3,0,1)
[1] 0.8093060 0.6955518 0.8359093
```

!! in any case typing for instance ***?r...*** we have the complete explanation
!! typing ***r...()*** and with the mouse inside the brakctes the **tab** we get all the options we can have
!! pressing **F9** you can change your random numbers

*The numbers that I have obtained are different from the ones obtained by you, with another pc; that's because our pc are starting from a different "seed" (starting point of a sequence of random numbers)
I few want to obtain all the same numbers we should fix a common starting point, to set up before the seed -> ***set.seed(...)*** the output is nothing, but then with r...(...) we will obtain all the same results
For instance set.seed(123) and then runif(3,0,1)
->This is a nice exercise to show "the reproducible properties of a program"... when you propose a solution, an algorithm it's very important to allow any other user to have the same output that you obtained in your application

*The generation of numbers from a uniform standard distribution is the most important, because it's the base for the Theory in which we can see the application of the inverse of the cdf of a distribution.

vect_unif_numb <- runif(n=1000000) we can so generate for instance a very huge vector composed of random numbers (we won't have the numbers reported in the output part)
We can also enlarge the amount of random numbers generated, but how much depends on the technical capabilities of your specific laptop. For very very huge vectors you need to have a pc with a very large ram (32, 64; we typically have 8 and it's the same quite ok)

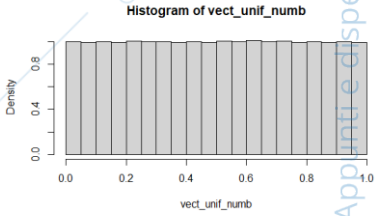
- **Obtaining the principle descriptive statistics about some data**

summary(vect_unif_numb) -> allows you to have a very quick evidence of the contents o an object.

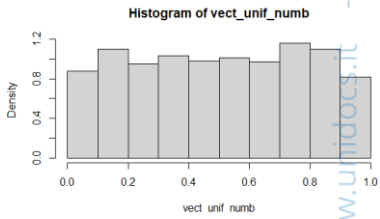
```
> summary(vect_unif_numb)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000008 0.2502778 0.5004569 0.5002597 0.7506405 0.9999978
```

- **Drawing an histogram**

!! The hist statement assumes that the vector contains continuous data
hist(vect_unif_numb, freq = FALSE) ->
the distribution is almost uniform, the densities are almost around 1, it's easy to see that the Uniform distribution fit the data



Repeating the exercise using for instance 1000 instead of 1000000 we get a an histogram with density's values much more different from 1 than before. We know the data comes form a uniform because we have generated them, but if we didn't know that, we couldn't have said it immediately.
When the sample size, coming from the real population, is not so large the histogram is a bit far from the original; it's possible to assume that it's uniform according to specific tests of hypothesis.



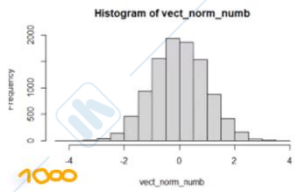
(In excel a bit advanced is the function INV.NORM that returns the inverse of the cdf of a Gaussian, useful to apply the theory recalled (with it you can put inside a random number that comes from a uniform distribution (0,1) getting a random number from a Gaussian distribution (or lognormal and so on doesn't matter)

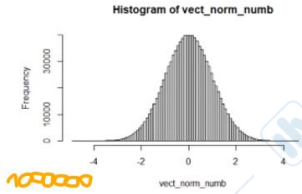
!! Starting from the theorem recalled we can generate random numbers from a distribution starting from the uniform distribution.

```
qnorm(runif(1) , mean = 0, sd = 1)
```

->GENERATING NUMBERS FROM A NORMAL DISTRIBUTION rnorm(...,0,1) or runif(1) will be a generic number between 0 and 1. Using that trick we obtain below a random number from a Gaussian distribution that depends on the random number from a uniform distribution.

Instead of 1 you can insert bigger numbers, up to very very large vectors. The bigger they are, the closer will be the histogram of our data to the theoric curve of the Normal



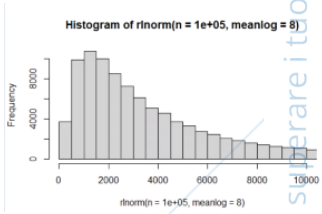


->GENERATING NUMBERS FROM A LOGNORMAL DISTRIBUTION qlnorm(runif(1,0,1)) or rlnorm(..., meanlog)

!!!! the 2 arguments of the lognormal ARE NOT the mean and the standard deviation of the lognormal distribution, but the mean and the standard deviation of the corresponding normal =

The parameters related to the 2 distributions are not the basic and silly transformation of the log of the average and the standard deviation

hist(rlnorm(n=100000, meanlog = 8), nclass=500, xlim=c(0,10000)) ctrl enter -> Meanlog does not mean that the lognormal has avg 8 (it's exp8 + ...); Inside we have an rlnorm statement, so we have the concatenation of 2 functions (=it's possible to insert in a function an external one that exploit the output of what there is inside to get here the corresponding histogram)

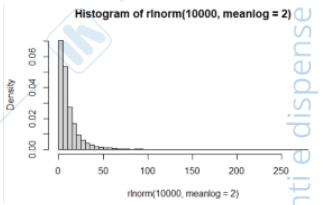


*the output is not as expected around 8

n represents the n° of random numbers we will obtain, nclass the n° of classes, xlim allows the user to restrict the view of the user to a specific domain (we can change to see what changes!)

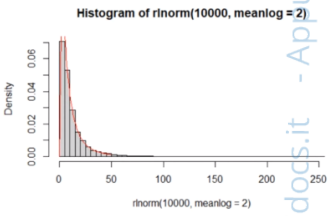
hist(rlnorm(10000, meanlog=2), freq=F, nclass=50) ctrl enter

->Here we have a meanlog of 2 instead of 8...the shape of the histogram changes: we have a much more shrink distribution towards the y axis



curve(expr = dlnorm(x, meanlog=2), from = 0, to = 50, add=T, col="red")

aim: having, on the histogram built from the data, also the theoretical curve that corresponds to the distribution of the real lognormal with those parameters, so that we can do a sort of comparison



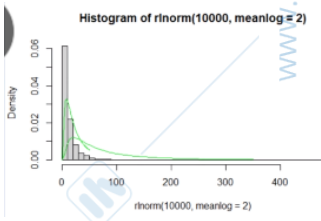
! We use dlnorm as "expr" because we want to plot the density of a lognormal with meanlog 2

! we want the domain between 0 and 50

! Add=T allow you to add the curve to the histogram already preset (with Add = F we would have only the curve)

! col=red is just a graphical option in order to have it not in the default color (black). We can change it in "green for instance, yellow and so on.

For instance with also the curve corresponding to lognormal with meanlog 3 and meanlog 4:

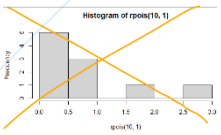


=>So it's a quick and intuitive descriptive manner to have a view of the comparison of the dataset collected from the real word and the theorical distribution Of course, intuitively what we are willing to observe is data collecting from real world and a mathematical model which is able to reproduce in a nice manner what we have observed.

->GENERATING NUMBERS FROM A POISSON DISTRIBUTION

```
> rpois(10,1)
 [1] 0 1 0 2 0 0 1 1 2 2
```

rpois(n=10, lambda =1) ctrl enter -> that sequence can be interpreted as a sequence of accidents that may happen into the future, supposing that the avg of the accidents is 1.



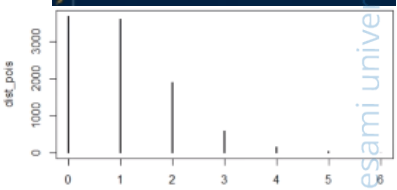
(assume that now n=10000)

!!! The Poisson is discrete, random numbers generated are integers ...

The "hist statement" assumes that the argument is a vector of continuous data, so here it's not good. So how can we represent those data?

Fist of all, use the "table statement": it returns the frequencies of how many times 0,1,2...7 (the domain of the poisson is from 0 up to oo, here we have up to 7 but if we have a different sequence doesn't matter) were generated (the sum is 10000 obviously) ... it's roughly similar to the pivot system used in excel!

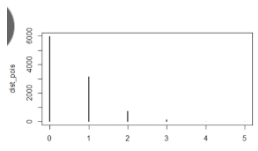
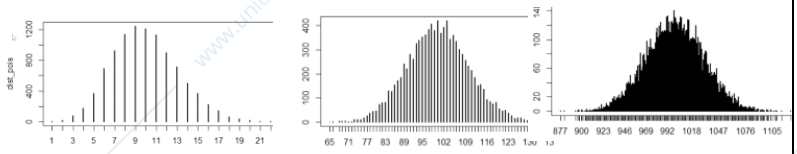
```
> table(rpois(n = 10000, lambda = 1))
   0    1    2    3    4    5    6    7
3701 3689 1796  604  175   28    4    3
```



Then we can add the object dist_pois `dist_pois <- table(rpois(n = 10000, lambda = 1))`

We will use the plot statement, a *generic* function for plotting data.

Some comments about changes in value of the λ

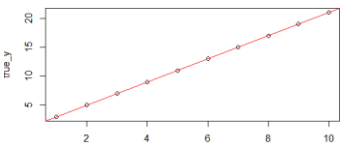
If λ decreases *(i.e λ=0,14, as in the exercise)*	If λ increases *(i.e. 10, then 100, then 10000)*
	

➔ The Poisson distribution is characterized by the *reproductive property*: the λ can be seen as the summation of many λ, supposing that the data comes from independent distribution. When λ increases we observe here implicitly the application of the CENTRAL LIMIT THEOREM, with a shape more and more close to the one of a Gaussian

DEALING WITH (SIMPLE) LINEAR MODEL

✂ How to get the estimates of a simple linear model-> *lm statement*

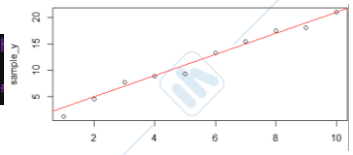
```
?lm

x <- seq(1,10)
true_y <- 1+2*x
plot(x , true_y) # basic s
abline(a=1,b=2, col="red")
```



?lm gives us lot of details, there are also a lot of corresponding function

Suppose to have a "true model" ("theoretical", in reality this would be unknwon) that is a linear combination, a straight line.

```
sample_y <- 1+2*x+rnorm(10) # wh
plot(x,sample_y)
abline(a = 1, b = 2, col="red")
```



In the real world we would collect some data, masked by some randomness: they will constitute a sample from Y. Plotting them we would have points a bit messy 'cause they are not exactly the true ones.

*here we have hp N(10,mean=0,sd=1) but we could have chosen a different sigma (in general the expected value of the error is 0): if the variance increase the dispersive effect is more evident (intuitive looking at a plot)

So, now we want to get $\hat{a}$ and $\hat{b}$, the estimates of the true unknown a and b

```
> lm( sample_y   ~  x  ) #

call:
lm(formula = sample_y ~ x)

Coefficients:
(Intercept)            x
     0.8168       2.1359
```

!! in the argument of the lm statement we have to use before the dependent variable and then the independent one

Y ~ X -> R

≡

$Y = a + b \cdot X$ -> Math

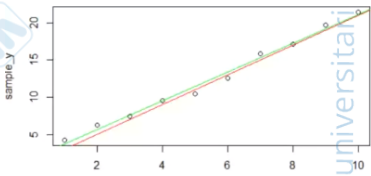
$$\hat{y} = \hat{a} + \hat{b}x$$

R found them using the ordinary least square approach (OLS)

We can now use the summary statement in order to compute the basic statistics related to our model.

We can plot it and compare it to the true model we have
We have two strategies to do that.

1) Using a simple abline statement manually inserting as a and b the value obtained by the estimation process
2) Using a much more general procedure: generate a new (complex) object and use it in the abline statement



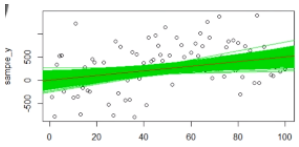

```
abline(a=0.8168,b=2.1359, col="green")
```

```
MY_MODEL <- lm( sample_y   ~  x  )
abline(MY_MODEL, col="green")
```

How to simulate the model a lot of times

```
for (i in 1:1000){
  sample_y <- true_y + rnorm(100,sd=sigma)
  MY_MODEL <-lm(sample_y~ x)
  abline(MY_MODEL, col="green")
}
abline(a=2, b=5, col="red")
```

*For i in 1:1 is a sort of loop just for one time, it's the previous case: we would get only a pair of estimates and so only a straight line estimated

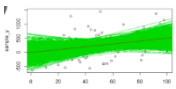


We want to use this procedure so for a 1000 of times: we would have 1000 new vectors of samples (assuming that the error component is time by time updated with the value of sigma; value of 200), 1000 pairs of estimates and so 1000 estimated straight lines. To compare them with the one of our true model we use the abline statement with the true parameters.

We "theoretical" straight line is inside the region that we have individuated (quite in the middle of it)

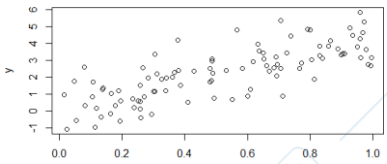
!!! an increase in the number of simulation causes an increase in the precision of results

!!! an increase in the sigma (=so in the variance; for instance here from 200 to 300) causes an increase in the uncertainty
(even if we can observe that the "theoretical" straight line is still inside and in the middle of the region)



Ex3:
1-Generate x <- runif(100)
2- generate y <- 4*x+rnorm(100)
3- plot the data

```
158  x <- runif(100)
159  y <- 4*x + rnorm(100)
160  plot(x,y)
```



A general specification: X is a vector of random numbers from a *continuous* variable (continuity property). It represents a sample extract from a wide population, continuous (here is the Uniform, but it's the same for the Gaussian, Gamma, ...): in order to collect the data, to observe the realization of the continuous variable we have to get a sequence of numbers (not the real line) => SO, in a computer and statistical context we are assuming that x is continuous but we observe a **discretization** of the variable (so we are not able to see its continuity)

4- estimate the coefficients of the model y=a+b*x

```
> lm (y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.2198       3.1766
```

(supposing to not know that there is not intercept => unconstrained model)

With lm (y ~ x) we are saying to R to apply the OLS approach to estimate the coefficients of our model. The intercept will be so $\hat{a}$ and $\hat{b}$

In statistics the formula was... ratio of the covariance between X and Y over variance of X

```
> lm (y ~ -1+x)

Call:
lm(formula = y ~ -1 + x)

Coefficients:
     x
3.528
```

(supposing to know there is no intercept / we don't want it => constrained model)

With lm (y ~ x) we are saying to R we want it to estimate only the multiplier

In statistics the formula was...moment of the product of X and Y divided by the moment of X squared.

5- assume the estimates are the only coefficients at your disposal

i.e. do not consider the " true model" = you don't know a=0 b= 4

We have 2 strategies:

1) Not general strategy, specific to this precise model (and so to these sample data): copy and pace the estimates proposed before into you model

```
a_hat <- 0.2198
b_hat <- 3.1766
```

2) General strategy. It allows to update the value of the estimates once we change the related sample data and model (we don't have so to insert values of estimates manually time by tyme)

```
> coefficients(MY_MODEL)
(Intercept)           x
  0.2198049   3.1766311
```

First of all we need make the lm statement an object, we will call it MY_MODEL. Now we can apply the "coefficient statement": it's the R command to extract from the model the estimates, it works with objects (that why we need MY_MODEL).
Also coefficients can become an object: estim <- coefficients(MY_MODEL) !!
->We can use str(estim) to get info about them
->Typing estim[1]=estimate in position 1 we get the intercept, estim[2] we get the multiplier
Calling a_hat <- estim[1] and b_hat <- estim[2] we arrived at the same result, in a much more generalized manner because now we can change the values of the sample, and executing again all the commands -> the amounts will be updated.

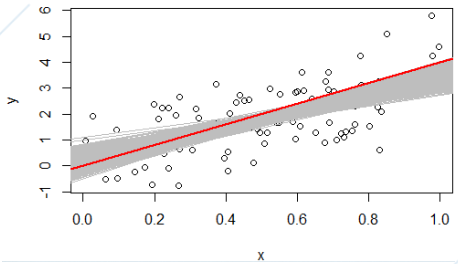
6- simulate 1000 of time the model in 5

7- plot the results and plot the "true" model in 1-2

```
a_hat <- 0.2198
b_hat <- 3.1766
y_new <- a_hat + b_hat*x + rnorm(100)

for( numb_sim in 1:1000){
  y_new <- a_hat + b_hat*x + rnorm(100)
  MY_MODEL <- lm(y_new ~ x)
  abline(MY_MODEL,col="grey")
}
```

First thing to do is creating a new object (y_new or y_hat) representing our model calculated with the estimated coefficients.

We will then use the for look statement to run the simulation. We would have for 1000 times the estimation of the model and the graphically representation.



Typing also

```
abline(a=0,b=4,col="red", lwd=2)
```

We get the line that represent the true (but unknwon) model

->with *lwd* we obtain a bord line!

It's possible to appreciate that the red line belongs to the region depicted by my simulation process (even if not really in the middle) We have now just compared the model simulated with respect to the true model.

(to see the values of the pairs of estimates computed at n.6 you should have done:

```
for(num_simu in 1:1000){

  y_new <- a_hat + b_hat * x + rnorm(100)
  MY_MODEL <- lm(y_new ~ x)
  print(coefficients(MY_MODEL))
  abline(MY_MODEL, col="grey")

}
abline(a=0, b=4, col="red", lwd=2)
```

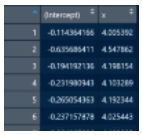
(advanced) 8- how can you save the 1000 of coefficient estimates in 6 ?

1)Prepare a brand new object in which saving them: it will be our empty box, a sort of empty space in the ram dedicated to our results.

It's intuitive thinking that – since we have *1000 pairs* of estimates – our box should consist of 2 columns (1st: intercpets; 2nd: multipliers) and 1000 rows.

```
for(num_sim in 1:1000){
  y_new <- a_hat + b_hat * x + rnorm(100)
  MY_MODEL <- lm(y_new ~ x)
  ESTIM_COEFFs <- rbind(ESTIM_COEFFs , coefficients(MY_MODEL) )
}
```

Now we have to fill up the box with the estimates: we will insert in the for statements (but without the plot part, because we won't plot anything here) an rbind statement to obtain column vectors -> if I use 1:1 we would get only a pair of estimates, put inside the empty box with a mechanism such as "nothing + something = something". With 1:2 we get 2 pairs of estimates and so on. For 1000 we would have them all inside the box!



If from the environment we show the matrix we have a quite complex matrix

Cbind and rbind used with scalars (=a vector)

Cbind = column bind; it allows to put something aside an object => so to create a row vector	**Rbind** = row bind; it allows to put something below an object => so to create a column vector

```
> cbind(1,2)
     [,1] [,2]
[1,]    1    2
```

```
> rbind(1,2,3)
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

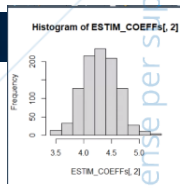
We can now use the "ESTIM_COEFFs" object to...

```
> summary(ESTIM_COEFFs)
  (Intercept)          x
 Min.   :-1.0264   Min.   :3.432
 1st Qu.:-0.3861   1st Qu.:4.087
 Median :-0.2523   Median :4.291
 Mean   :-0.2542   Mean   :4.298
 3rd Qu.:-0.1283   3rd Qu.:4.519
 Max.   : 0.4102   Max.   :5.373
```

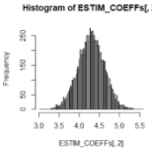
-> have a quick view of some basic statistics in just one shot through the summary statement (extremely important)

->built the related histogram for the intercepts [,1] and for the multipliers [,2] through the "hist" stat

```
hist(ESTIM_COEFFs[,2])
```



Does it remind us something? Let's try with 10000 and 100000 times (instead of 1000) – it takes some seconds to complete the job – and try also to insert nclass = 100



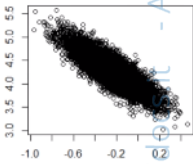

```
hist(ESTIM_COEFFs[,2], nclass=100)
hist(ESTIM_COEFFs[,1], nclass=100)
```

It reminds quite closely a *Gaussian*

If the error component of a linear model is supposed to be Gaussian, the distribution of the random variable which corresponds to the estimators of the model is supposed to be as well Gaussian

->have the plot: it will be a scatterplot of the many estimates found above

```
plot(ESTIM_COEFFs)
```



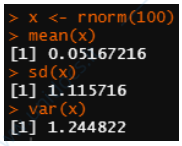
We can observe that the estimators are negatively correlated. So, the joint distribution of the two estimators of the linear model according to this picture is supposed to be a bivariate Gaussian distribution with a negative correlation coefficient.
Otherwise, it's possible to show that using central limit theorem the same result it's true but you need to apply a lot of simulations to achieve this aim.

Block models in languages are for instance the for look statement, the lm statement and function

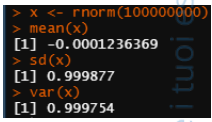
Using functions in R

Function in any language (and also in math) is something very general, describes something (a straight line, a curve, ...). It should be used whatever is the input of the object.
To express this as a function in R we have to use the function statement. In the brackets

we should insert the things from which the function that we want to study depends. Here we have a parameter and a vector of data. Then into the {} we insert the function to express!



Let's generate 100 random numbers from a Gaussian and let's compute the basic statistics as mean, standard deviation and mean.
...We know that the increase of the sample size causes a conversion of the value of the mean to 1 and of the value of the variance to 1.



Suppose now to have only the vector of x <- rnorm(100) and no other information. We will see an example of the research of the minimum with respect to an unknown theta (basic standard statistic problem)

Solving an optimization problem (*we want the best solution*)

$\min_{\theta} \pi\left[(x-\theta)^2\right]$

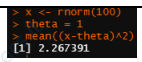
$\theta = \pi(x)$

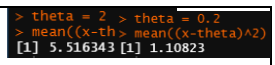
X is a statistical variable. We now in advance that the minimum in this minimization problem is the E(x) because if it is so that is the *variance*

-> this problem is so relevant because the variance represents the risk. In order to compute the credibility of something you want to compute the variability of it = when it's huge, our results are not credible so the premium is really high. Differently, if you are almost sure about the occurrence of accidents, you will be ready to estimate in a precise manner the corresponding premium, that typically will be lower.

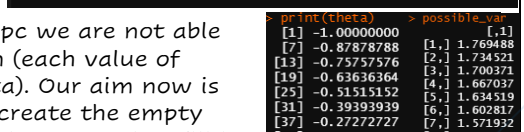
But...how can I solve this problem numerically?

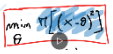
TRIAL AND ERROR APPROACH = texting different values of θ
Suppose $\theta = 1$ and then $\theta = 2$. The solution for $\theta = 2$ is worst than for $\theta = 1$ so we can exclude 2 as a solution and try with values closer to 1.
With $\theta = 0.2$ we obtain an even better solution and so on



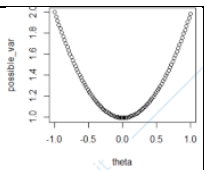


OPTIMIZE()
Here we suppose that θ is inside a possible domain (***)
Suppose also to split that interval into 100 small pieces = you are discretizing that domain !!! the function that we are studying is continuous with respect to θ, but with the pc we are not able to represent the continuity -> discretization of the domain (each value of θ can be seen using the print statement or typing just theta). Our aim now is to compute that quantity for each possible value of θ : we create the empty box "possible_var" and use the for statement with the rbind command to fill it up (save in it)with that quantity computed for each value of θ.
***which domain should I choose? It's a arbitrary subjective selection, an opinion -> once I did it, we have to think about the graphical output.
The locus of point plotted returns the function: by changing θ we obtain different values of

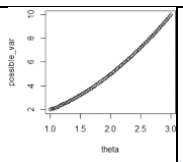




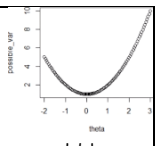
here we chose (-1;1): it seems to be a good choice



If we had chosen (1;3): that domain is not good for our purpose (find out the minimum) 'cause there the minimum would be the extreme of the domain (negative derivative): it's intuitive that the best solution should be located on the left. We son should try to enlarge the domain.



If we had chosen (-3;3) we would have observed that the best solution would be around 0: we could so shrink our domain!



The best solution (and so our aim) corresponds to the value of θ that returns the minimum of the locus of point proposed there. We can imagine it will be around 0 through these comments but we can't for now compute it exactly...

First thing is expressing the function that we want to study through the function statement. We will make it a new object ("my_first_func").
Then we will use the optimize statement!!: it need as arguments the object and the interval chosen. We then get the result... we can control it computing the mean of x.
You can then compute the quantity of the corresponding variance with that value and the use the command var(x) to control. You won't obtain exactly the same value because

```
my_first_func <- function(theta, dataset=x){
  mean((dataset-theta)^2)
}

optimize(f = my_first_func, interval = c(-3,3))
```

```
$minimum
[1] 0.00357423

$objective
[1] 0.9921044
```

$\hat{\sigma}^2 = \frac{1}{m} \sum (x_i - \bar{x})^2$ $\quad \hat{\sigma}_c^2 = \frac{1}{m-1} \sum (x_i - \bar{x})^2$

=S2 in the R code we use this

OPTIM()

The starting point is also here the implementation of a function

```
my_first_func <- function(theta, datase
  mean((dataset-theta)^2)
}

optim(par = c(0), fn = my_first_func,
```

```
> optim(par = c(0)
$par
[1] 1.645313

$value
[1] 4.787092

$counts
function gradient
      30       NA

$convergence
[1] 0

$message
NULL
```

The very first argument of the optim statement is/are the free coefficient(s): here par=c(0) is a starting point from which starting in finding the best solution.

Next he want to know the arguments of your functions are just related to the free parameters of to something else, here to something else. Because inside we have not only the free parameter but also in our case the vector systematically used by the function to find the solution.

Running that command we get the solution but also a warning message here: (if I change the starting point from 0 to 1 (I get the same solution) and to -1 (again) and so on...!)

```
Warning message:
In optim(par = c(0), fn = my_first_func, dataset = x) :
  one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
```

= user you have used optime but your problem has just an unknown quantity/ parameter.
A much more efficient algorithm in R is available as optimize or brent.

$Y = a + bX$

Also the OLS is a optimization problem -> in order to find the best estimates to a and b we need to find the minimum with respect to a and b of the deviance of the residuals.

$$\min_{a,b} \sum (y_i - (a+bx_i))^2$$

The solution is easily founded using the **lm statement** (ex 4)
(so that we can compare the solution founded?)
but what is the reasoning numerically speaking? What's the logic behind?

OPTIM()

Now we should use optim (not optimize, because we have 2 parameters to estimate!) to solve the implicit OLS problem (min of the deviance of the residuals)

n.b. Param[1] corresponds to a, param[2] corresponds to b in the standard notation y=a+b*x

```
x <- runif(100)
y <- 1+3*x + rnorm(100)
lm(y ~ x)

OLS_problem <- function(param,depend=y,indep=x){
  sum( depend - (param[1]+param[2]*indep)^2 )
  }
optim(par=c(0,2), fn=OLS_problem, depend=y, indep=x)
```

Hint: the code optim(par=c(0,1), fn=...., dataset=...) can be used to initialize the function that must be ex-ante prepared to implement the deviance of the residuals = First of all we need a the function to use. The function will depend on "param" and depend=y, indep=x
*you could obviously type only param, depend, indep leaving them general
The argument will be " sum ((depend – (param[1]+param[2]*indep)^2)".
In the optim statement we should also chose a subjective point, such as (0,1)

-> !! the solution we find it's not exactly the same, but close

Lm -> exploit liner algebra, much more precise

Optim -> exploit numerical procedure, less precise even because it depends on a subjective starting point... In general, we have always to try more starting points: if we see that the solutions are always quite the same, it's stable it's okay, if not you have to understand which is the best situation from which start.

In this case we have just one vector, but we could have a matrix. So that that param would have had much more elements.

TRIAL AND ERROR APPROACH

Hint: two nested for-loop statements must be considered

→ **Showing the validity of the CENTRAL LIMIT THEOREM through R**

We did a previous example of it simply showing graphically through histograms the convergence towards the Gaussian of a Poisson with a certain λ
~the Poisson enjoys the reproductive property, so can be interpreted as the sum of Poissons iid

Now we want to do show the same thing but numerically:

```
#CLT
how_many_rv <- 2
sample_size <- 10000

random_num <- matrix(NA, ncol=how_many_rv, nrow=sample_size )
for(j in 1:how_many_rv){
  random_num[,j] <- runif(sample_size)
}
```

We chose to consider 2 random variables (so **n**=2*) from which we will obtain a 10000 records (**sample size** = 10000)*
*We suppose that: **HP1:** the random variables are iid*
***HP2:** the random numbers comes from uniform*
To reproduce the presence of the sequence of random variables (x1, …, xn…here only x_1 and x_2 as we had only 2 customers) we need a specific object: it will be a matrix in which the columns will contain each random variable and the rows the data collected: so here 2 columns and 10000 rows.
At the beginning I have an empty matrix with that structure ("NA" is a specific technology used in R to represent something not know as we can see in the environment window) . so…

- **Using the matrix statement matrix(…,ncol=…,nrow=…)**
  It helps us preparing a matrix. If we don't know the data inside it we can put at the beginning NA, if we know them we can use the object that contains them

Now, using the for loop statement we will fill it up: for each of our 2 customers we would collect the observations, that can be interpreted as random numbers from a Uniform distribution (for hp2)
[,j] remind that before this procedure is done from the customer 1 (x1) and then for 2 (x2)
Obviously you can have more and more customers: our portfolio could for instance contain 1000 customers and so we will have x_1, …, x_{1000} and for each of them 10000 observations.
Next we have to compute some useful quantities…

- **Using the apply statement**

```
data <- c(1,2,3,4,5,6)
ex_matr <- matrix(data, ncol = 2, nrow = 3)
ex_matr
apply(ex_matr, MARGIN = 2, mean)
> apply(ex_matr, MARGIN = 2, mean)
[1] 2 5
> apply(ex_matr, MARGIN = 1, mean)
[1] 2.5 3.5 4.5
```

We have so an example of how a complex object as a matrix can be analyzed in a compact manner, just using an implicit procedure that exploit its dimensions (margin 1->rows; margin 2->columns). The arguments of the apply statement should be: the object, the dimension to use and then the computation we want to do.

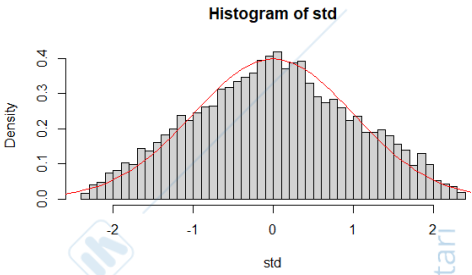
There we did first by column (2), then by rows (1)

```
avgs = (apply(random_num, MARGIN=1,mean))
sds = (apply(random_num, MARGIN=1,sd))
std = (avgs-mean(avgs))/sd(avgs)
hist(std, nclass=50, freq=F)
curve(dnorm(x),from=-4, to=4, add=T, col="red")
```

We are doing these computations by row (x bar is the avg by row, avgs) because the theorem says that you are willing to estimate the average of the sample size of x1,…,xn and so on. They are the columns, not the rows. So, each row represents say a random evidence of what can be collected from each customer available in my portfolio. In that case we get a vector of 10000 estimates where each position corresponds to the sample avg by row.
The sd is not so necessary in reality

!! Instead of apply we could have also he first one is again a for loop statement: given it it's possible to pick up a specific row, and for each row it's possible to compute the average, to save them into a specific vector and using those output to compute the final solution to our problem

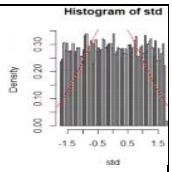
So we want to show that the distribution of the standardized x bar (→empirical data) converges asymptotically to the standard gaussian distribution (theoretical model).
We see that they are quite similar



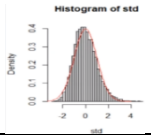
->Of course theorem works in distribution so theoretically when n->oo
->Of course an increase in the sample size causes a graphical result closest to the theoretical model

Some comments/points/changes:

Considering n=1
In our matrix now we will have just 1 column
The average for each row will be the same number, the average of the averages will corresponds to the avg of that column, and the sd of the averages corresponds to the avg of the column.
...At the end of the procedure we observe not at all a Gaussian distribution, but a U(0,1)

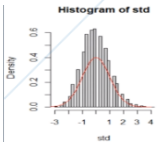


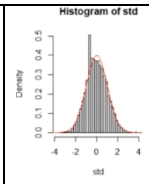
Let's change the HP2: considering the lognormal distr as the generating process of the random numbers
⇨ So a much more asymmetric distribution
Let's use n=10
!!! When we are dealing with asymmetrical distributions the theorem need a higher n to show the convergence = when n small the convergence to the normal distribution is not so nice
Let's use n=1000
The fit could be better but it's better



Let's change the HP2: considering the Poisson as the generating process of the random numbers
⇨ So a discrete distribution

Let's use n=10
Very bad fitting





Let's use n=100
We have a strange picture...R has used 50 classes as chosen at the beginning. But what if they were 20? Or 30?
We have so a different view of the same vector.
...if they are too large you have a sort of high density in specific c

⇨ hist is a quite powerful tool but it's strictly dependent in the number you classes you selected. So in order to see that the data are roughly to our continuous distribution we have to set up a test of hypothesis the distribution observed collected data and the theoretical one.

ΔWhenever you move from theory (mathematical) into practice (numerical) you must think that your mathematical model (=something abstract) has to be somehow represented using an algorithm, that samples the domain to have of course evidence of how works. This "sampling" consists in discretizing the domain

To represent a line can use the an object, drawn through a function (y=a+bx). Now try to think about how to implement a straight line into an algorithm -> if you type in R y=a+bx you get nothing: you have to explain to R what's a, b, x and so on.

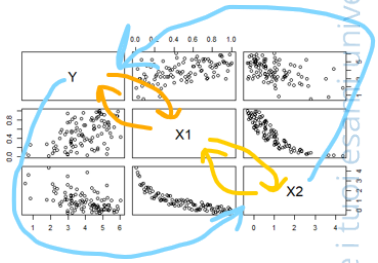
CREATING A DATA FRAME

Now we will see how to save a previous vector into a dataframe and how to use it.

Dataframe -> is a sort of database, a "special" matrix: it can be a mix of numeric and alphanumeric data ≠
Matrix -> homogeneous object, just numeric, or all not numeric

```
n <- 100 # sample_size

X1 <- seq(1,n)/n # normalization
X2 <- -log(X1)+runif(n)-.5
Y <- 3 + 2*X1 - 0.1*X2 + rnorm(n)
```

- **Creating a dataframe**
  DATASET <- *data.frame*(Y=Y, X1=X1, X2=X2)
  this notation allows the user to save into the data frame Y, X1 and so on with the corresponding names. If we typed losses=X1 we would have the 2nd column of the data frame with the label/name "losses"
  (we can see it in the environment window)
- **Showing only the first 6 rows of the data frame**
  head(DATASET) *very useful since the dataset could contain thousand of rows*
- **Showing only the last 6 rows of the data frame**
  tail(DATASET)
- **Getting the multiple plots**
  plot(DATASET) -> **scatterplot matrix:** *allows us to have a quick view of the dependence of the variables, two by two*
- **Getting the names of the fields/labels**
  names(DATASET)
- **Obtaining the characteristics of the object DATASET**
  str(DATASET) -> *we can see i.e that the 3 colm are numerical.*
- **Showing only the content of a specific column**
  DATASET $ X1




```
'data.frame':   100 obs. of  3 variables:
 $ Y : num  1.875 4.121 0.735 2.977 2.986 ...
 $ X1: num  0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 ...
 $ X2: num  4.34 3.91 3.65 3.52 2.6 ...
```

DEALING WITH MULTIPLE LINEAR MODEL

Now we want to extend the linear approach proposed above to a model where we have not just one explanatory variable, but more than one.

- **How to get the estimates of a multiple linear model-> *lm statement***

(In our example we have just 2 explanatory variables)
Our theoretical model has as coefficients 3, 2 and -0.1
...Now how can I fit a multiple model?

```
n <- 100 # sample_size

X1 <- seq(1,n)/n # normalization
X2 <- -log(X1)+runif(n)-.5
Y <- 3 + 2*X1 - 0.1*X2 + rnorm(n)
```

```
regression <- lm( Y ~  X1+X2, data=DATASET ) # fit a multiple model
```

What about the contents of regressions? How can we have a view of them?
Look at Elementi of Econometria material

```
> summary(regression)

Call:
lm(formula = Y ~ X1 + X2, data = DATASET)

Residuals:
     Min       1Q   Median       3Q      Max
-2.72505 -0.68032  0.00714  0.70775  2.36437

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.6987     0.5536   6.681  1.5e-09 ***
X1            1.0841     0.7134   1.520    0.132
X2           -0.3463     0.2133  -1.624    0.108
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.013 on 97 degrees of freedom
Multiple R-squared:  0.2824,    Adjusted R-squared:  0.2676
F-statistic: 19.08 on 2 and 97 DF,  p-value: 1.027e-07
```

```
str(regression)
```
-> running that command we get a lot of thinks

```
> head(model.matrix(~  X1+X2, data=DATASET ))
  (Intercept)   X1       X2
1           1 0.01 4.337401
2           1 0.02 3.906633
3           1 0.03 3.654906
4           1 0.04 3.516370
5           1 0.05 2.595589
6           1 0.06 2.570896
```

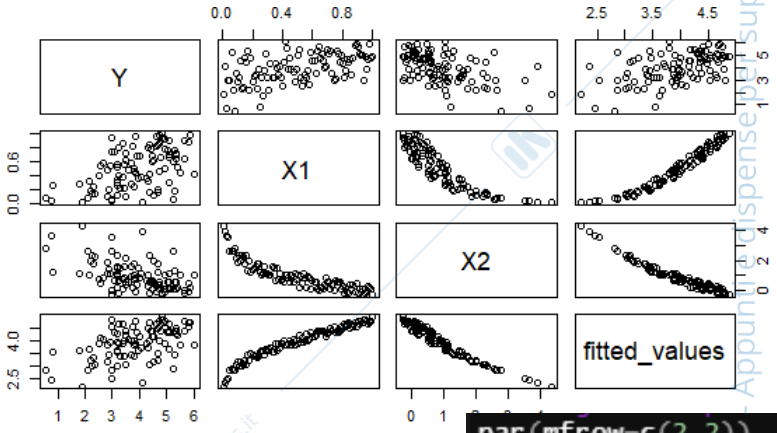
Here we have the application of the head + model matrix function
ΔHead -> in fact in the output we have only the first 6 rows
ΔModel.matrix: technical, powerful and useful command in data sciences because allows to grasp all the details related to the explanatory variables from the data we have (it points out the vector of 1s of the intercept)

- **Using the fitted statement**
  Allows you to compute the y hat vector through the data we have
- **Using the predict statement**
  Here you are assuming brand new data for the estimation…

```
###################
fitted_values <- fitted(regression) # What is the meaning?
predicted_values <- predict(regression,
                            newdata = data.frame(X1=0.5, X2=1))
```

```
# add to DATASET the vector of fitted values
DATASET <- data.frame(DATASET, fitted_values=fitted_values)
plot(DATASET)
```

It's possible to save the fitted values founded above
And even to draw a new scatterplot matrix, with also them

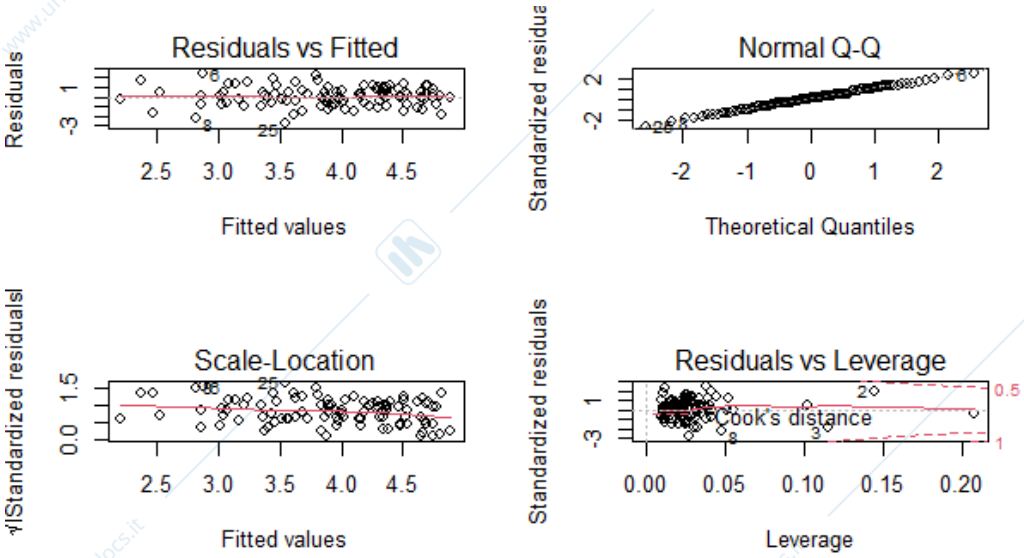


- **Graphical capability of R**

With just plot(regression) we get the scatterplot matrix of just one object, one by one. R says to us "hit <Return> to see next plot:" and typing ctrl enter we get the other one and so on… But, how can I organize my windows in order to appreciate jointly my pictures?

```
par(mfrow=c(2,2))
  plot(regression)
par(mfrow=c(1,1))
```

Typing par(mfrom=c(n of rows,n of columns). There par stands for "parameters", mfrow for "multiple framefork" window and c stands for a vector -> So par(mfrow=c(4,1) -> to split the window into a row vector of 4 plots
So par(mfrow=c(1,4) -> to split the window into a column vector with 4 plots



The command plot applied to the regression object allows us to get 4 different pictures.
Each of them contains some details about regression.

(Using zoom you can have a better view of the plots)

Residuals vs fitted -> shows the relation between residuals and fitted values.
A basic property of the OLs method is that the correlation between the covariance between the residuals and the fitted values is nominally equal to U. So the aim of the plot is to investigate empirically that this theoretical result is true.
In case, the relationship between the residuals and the fitted values is supposed to be not exactly a sort of random allocation of points in our picture it means that we are underestimating the model. In this case the fitting is quite nothing

Normal Q-q
Q-Q Plot is the graphic representation of the quantiles of a distribution.
Quantile to quantile the Q-q plot allows you to compare each empirical data (y axis) of your vector the theoretical quantiles (x axis, computed using a specific model that here, and often, is the Gaussian distribution).
So, if the locus of empirical data is a sort of straight line (like above) it means that you have a one of one correspondence between empirical data and theoretical model => it's possible to say that the residuals are supposed to be Gaussian distributed
So this Plot is really important because very often we assume to have disturbance terms Gaussian distributed, but this has to be verified.
... If the plot says that that hp is not satisfied/true you are overestimating or overestimating your model, getting so a wrong interpretation

Scale location
Residuals vs Leverage

With "par(mfrow=c(1,1)) we say at the end to R to go back to a single representation of the plots. If I did not insert that command it would remain valid for R the previous structure in showing the plots .

OLS and loss FUNCTION : an example of function() and OPTIM() !!!
This is an exercise we already did, but assuming to have 2 explanatory variables and not just one.

```
#####################################################
## OLS and loss FUNCTION : an example of function() and OPTIM() !!!
#####################################################
deviance <- function(param, INPUT=DATASET){
  with(INPUT, sum((Y - (param[1]+param[2]*X1+param[3]*X2))^2))
}

optim(par = c(0,0,0) , fn = deviance , control=list(reltol=1e-16))
```

A new data frame with a CATEGORICAL variable

-> gender, as other important characteristics of the profile of a customer, is not a numerical information. The logical categories (levels) in which consists are characters!

we have to learn how to treat them

⌘ **Selecting randomly labels from a population**

```
## suppose Y = b0 + b1*X1 + b2*gender + eps
n <- 100 # sample_size
X1 <- seq(1,n)/n

# select at random n labels from c("M","F","SOC") with unequal weights
gender <- sample(size = n,
                 x = c("M","F","SOC"),prob = c(.4,.5,.1),
                 replace = TRUE)
```

sample statement -> we have to interpret the mechanism below the sample statement as the extraction of balls from a box.

Here we want to do n extraction, our (balls') labels are "males", "females" and "society" and the extraction are made with replacement.

pick up balls with replacement = you pick up the ball, record the result and put it inside again
No replacement = then you put the ball outside the box.

In reality the customers the company can deal are M, F or SOC
with We know we have 40% of males, 50% of males and 10% of society.
Potentially the company has 40% of prob to sell one of the products to M, 50% to F and 10% to SOC... So, "gender" consists of the random selection of balls from the box with 40% of M 50% F and 10% SOC: this is a sort of strategy to prepare a sort of scenario.

!!!! R uses the double quotes because "M" "F" and "SOC" are not numerical, are characters => they can't be used in this way in a model, into for instance a fitting process.

(for instance: can I deal with a model in which I use The cost -> vector of number and Gender → vector of characters ... no!)

```
> gender
  [1] "M"   "F"   "M"   "F"   "SOC" "M"   "F"
 [26] "F"   "M"   "SOC" "F"   "F"   "M"   "F"
 [51] "M"   "M"   "M"   "M"   "F"   "M"   "F"
 [76] "F"   "M"   "F"   "F"   "M"   "M"   "F"
> str(gender)
 chr [1:100] "M" "F" "M" "F" "SOC" "M" "F" "F"
> # change gender into a factor variable: USE f
> factor(gender)
  [1] M   F   M   F   SOC M   F   F   F   M   M
 [39] F   M   M   F   M   F   M   M   M   F   F
 [77] M   F   F   M   M   F   M   F   M   M   M
Levels: F M SOC
> gender <- factor(gender)
> str(gender)
 Factor w/ 3 levels "F","M","SOC": 2 1 2 1 3 2
```

⌘ **Transforming a character vector into a factor**

Factor statement -> allows us to transform the character vector into a numerical vector (*integers: M -> 1 F -> 2 SOC -> 3) where the labels (M F SOC) are assigned to a specific combination of numbers => now we have an output with no "", ready to be used in a model!!!

*But this strategy is a subjective transformation

We could have chose M -> 2 F->1 and SOC ->3 as 1000 1001 as 1002

** The transformation of labels into numbers is a quick method used by any software for a basic technological and practice reason (efficiency of the algorithm): when the dataset is really big, having in the memory "F", "M", and "SOC" every time (even because the pc works in a numerical way) takes a very big amount of RAM (really larger than using only 1,2,3 and so on)

So, aim = transforming in a not subjective manner characters into numbers in order to grasp from the data / to split in a disjoint manner the contribution of males, females and society (with respect to...)

We have a sort of partition of the domain: splitting the domain into disjoin subregions such that the union of them returns the entire population. So disjoint means that the contribution to our model from males, females, society must be transformed into a manner in which their contribution does not depends on the contribution on the other categories.

Imagine 2 have just 2 columns accidents and gender. If you want to have the avg of accidents of only the males for instance you have just to compute the conditional avg with respect to males.

We will use dummy variables (-> a T/F strategy, a Boolean logic) to do that

Let's try to use only 2 labels (simplification) to better understand:

```
n <- 100 # sample_size
X1 <- seq(1,n)/n

# select at random n labels from c("M","F","so
gender <- sample(size = n,
                 x = c("M","F"),
                 prob = c(.4,.6),
                 replace = TRUE)
gender

data.frame(factor(gender), as.numeric(gender))
> data.frame(model.matrix(~  gender))
   X.Intercept. genderM
1             1       0
2             1       1
3             1       0
4             1       0
5             1       1
```

as.numeric means we are trying to transform gender into numeric structure...we get a warning message, then a vector of F, M and then NA because gender is not available, is a vector of characters.
Now suppose to fit gender into factor: we transform it from a character vector into a factor. M=1 F= 2 is it the possible solution for all our problems?
No

```
> head(data.frame(model.matrix(~  gender), gender), n=10)
   X.Intercept. genderM gender
1             1       0      F
2             1       1      M
3             1       0      F
4             1       0      F
5             1       1      M
6             1       1      M
7             1       0      F
8             1       0      F
9             1       0      F
10            1       0      F
```

```
> data.frame(factor(gender), as.numeric(factor(gender)))
   factor.gender. as.numeric.factor.gender..
1               F                          1
2               M                          2
3               F                          1
4               F                          1
5               M                          2
```

Data.frame is a very important statement -> helps us in understanding what is the independent variable, that R is going to use in the model. The vector of intercept is a vector of 1, in gender M when we have 1 we have a male, with 0 a female. So gender_M contains nominally only the information related males

```
gender <- relevel(gender, ref="M")
contrasts(gender)
plot(gender)
> contrasts(gender)
le ?
    M SOC
F   0   0
M   1   0
SOC 0   1
```

Each column corresponds now to a dummy variable
(2= n. of labels -1 = n. of dummy)

SOC 0,1 -> we have false in the M column, and true in the society one
M 1,0 -> we have true in the M column, and false in the society one
F 0,0 -> if it's not M neither SOC
= So given 3 cat it's possible to use 2 columns to represent univocally them
!! We have 2 dummies (one for M and one for SOC... and F?): implicitly we are able to understand the contribution of the nominally, but not represented, category females by removing the information about M and SOC

For the interpretation of the regression this table is fundamental

```
> head(data.frame(model.matrix(~  gender), gender), n=25)
   X.Intercept. genderM genderSOC gender
1             1       0         0      F
2             1       0         0      F
3             1       1         0      M
4             1       0         0      F
5             1       0         0      F
6             1       0         0      F
7             1       0         0      F
8             1       0         0      F
```

Here we have another representation of the same thing.
mathematically if you try to multiply column gender M and gender Soc you will get the vector of 0 = they are orthogonal = the contribution of the column M doesn't affect the contribution of the column SOC

When we then get a regression...: in case, execute dataset and in case you have a lot of columns names by gender you need to select the last one and call it in case gender 2 (it's only to avoid the repetition)

```
> regression

Call:
lm(formula = Y ~ X1 + gender, data = DATASET)

Coefficients:
(Intercept)           X1      genderM    genderSOC
     2.5905       2.2455       0.1461       0.3816
```

```
DATASET <- data.frame(X1=X1, gender=gender)
regression <- lm( Y ~  X1+gender, data=DATASET ) # fit a mul
```

We have 3 labels, but what is the contribution of a one of them our model?

- How can we fit the contribution to the risk of the only category male?
  I need to consider of course the coefficient related to males, but not the one related to society. Gender_M*0 genderSoc*1
- How can we fit the contribution to the risk of the only category male?
  I have to remove the contribution of males and society: So *0 and *0
  So implicitly the model considers also the contribution of females, but it is not present explicitly in the output: it doesn't matter because we what we are willing to know is not specifically what is the coefficient related to females but what is the model, having removed the contribution of males and society

EXERCISE

```
# exercise suppose
gender <- c("M","F","M","F","M","M","F","M","F","F",
            "M","F","M","F","M","M","F","F","M","F","M")
accidents <- c(1,0,0,0,0,2,1,0,1,0,0,0,0,0,0,0,0,1,0,1,0)
# create a data frame. Name it : "dataset"
dataset <- data.frame (gender, accidents)
dataset
```

 # compute the average of accidents for each category of gender

...= conditional avg of accidents given a category (M, F or SOC).

We can apply different strategies

- *Manually: dividing the sum of recorded accidents for the n. of cust of a specific category*
- *For... loop statement*

```
mean_F=0
count_F=0
mean_M=0
count_M=0
for (i in 1:dim(dataset)[1]){
  if(dataset$gender[i]=="M"){
    mean_M=dataset$accidents[i]+mean_M
    count_M=1+count_M
  } else {
    mean_F=dataset$accidents[i]+mean_F
    count_F=1+count_F
  }
}
```

I need some "empty boxes"
dim = dimension, applied to a matrix
returns n° of rows and n° of columns. Here we use 1:21

```
> dim(dataset)
[1] 21  2
> 1:dim(dataset)[1]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
```

So for each row consider that... if in gender column we have M increase the box by the number related to column accidents, if you have F do the same for the F computation.

```
> mean_F
[1] 0.4
> mean_M
[1] 0.2727273
>
```

You get the solutions, but it's not a so elegant code (too complex and long)

- *Install library:*
  R environment can be enriched by lots of function using install.packages and then the name of the package you want. Then you have to open it using library.
  The last command says to R: "pick up the object dataset and consider it, next apply "group_by" to implicitly allows the system to split it into the 2 categories and summarize to have the basic statistics (here the mean of the accidents) to each group reported
  So avg conditional to F = 0.4, avg conditional to M = 0.273

```
install.packages("dplyr")
library(dplyr)
dataset %>% group_by(gender) %>% summarise(avgaccidents=mean(accidents))
```

```
# A tibble: 2 x 2
  gender avgaccidents
  <chr>         <dbl>
1 F             0.4
2 M             0.273
```

 # change gender into factor

```
dataset = data.frame(gender=factor(gender), accidents)
```

 # fit the model accidents ~ gender

```
> lm(accidents ~ gender)

Call:
lm(formula = accidents ~ gender)

Coefficients:
(Intercept)      genderM
     0.4000      -0.1273
```

(Now that "gender" is a factor we can do it!)

!! Suppose you are interested into the conditional avg related to F for instance. In order to understand how can you do let's…

 # print the model
 # how can you compute the averages by gender?
 # check the command contrasts(dataset$gender)
 # check model matrix(~ gender)

```
> contrasts(dataset$gender)
  M
F 0
M 1
```

```
> model.matrix(~ gender)
   (Intercept) genderM
1            1       1
2            1       0
3            1       1
4            1       0
5            1       1
6            1       1
```

Gender_M (M!!!!) = that column is used to represent the cat males, the females are implicitly in the model (->If I remove males I have females)

 # check data.frame

```
> data.frame(model=model.matrix(~ gender, data=dataset), gender)
  model..Intercept. model.genderM gender
1                 1             1      M
2                 1             0      F
3                 1             1      M
4                 1             0      F
5                 1             1      M
6                 1             1      M
7                 1             0      F
```

Conditional avg of accidents for males?
0.4 + 1-- = 0.27 and so on obtain before*
Conditional avg of accidents for females?
I must remove contribution of the males to the model so 0.4 is left, exactly the one obtained before

 # repeat with the vectors
gender <-
("M","F","SOC","F","M","M","F","M","F","F",”M","F","M","F","M","M","F","F","M","F","SOC")
accidents <- c(1,0,1,0,0,2,1,0,1,0,0,0,0,0,0,0,0,1,0,1,3)

```
> library(dplyr)
> dataset %>% group_by(gender) %>% summarise(avgaccidents=mean(accidents))
# A tibble: 3 x 2
  gender avgaccidents
  <chr>         <dbl>
1 F               0.4
2 M               0.333
3 SOC             2
```

To understand how to compute the others we need also contrasts

The for loop statement is not anymore ready, because it has now to consider 3 labels and not 2. Let's try to do that (we skip this part of the code now).

Cond avg of females -> 0.4

```
> # change gender into factor
> dataset = data.frame(gender=factor(gender), accidents)
> # fit the model accidents  ~ gender
> lm(accidents  ~ gender)

Call:
lm(formula = accidents ~ gender)

Coefficients:
(Intercept)      genderM    genderSOC
    0.40000     -0.06667      1.60000
```

```
> # check data.frame(model=model.matrix(~ gender, data=dataset), gender)
> data.frame(model=model.matrix(~ gender, data=dataset), gender)
   model..Intercept. model.genderM model.genderSOC gender
1                  1             1              0      M
2                  1             0              0      F
3                  1             0              1    SOC
4                  1             0              0      F
5                  1             1              0      M
6                  1             1              0      M
7                  1             0              0      F
8                  1             1              0      M
9                  1             0              0      F
10                 1             0              0      F
```

Cond avg of males (1 0)-> *0.4 -0.06667*1 + 1.6*0 = 0.333*
Cond avg of society (0 1)-> *0.4 -0.06667*0 + 1.6*1*

```
> contrasts(dataset$gender)
    M SOC
F   0   0
M   1   0
SOC 0   1
```

Dataset 2

It's another csv file
!it's very important you save that file and you remember well in which directory it's saved and which is his name (often the one assigned to save it's not the same name you have when you download it from BB)
*When you double click on it the system applies automatically excel to open the file, already separated (sometimes)

Age	ZipCode	RiskExp	NbrClaims	ClaimAmc	AgeCar	Sex	Fuel	Split	Use	Fleet	Sport	Cover	Power
64	1000	1	0	0	5-Feb	Female	Petrol	Once	Private	No	No	MTPL+	66-110
28	1000	0.046575	1	155.9746	10-Jun	Female	Petrol	Twice	Private	No	No	MTPL	66-110
58	1000	0.40274	0	0	>10	Female	Petrol	Thrice	Private	No	No	MTPL	<66
37	1030	0.169863	0	0	5-Feb	Female	Petrol	Once	Professior	No	No	MTPL+++	66-110
29	1030	1	0	0	10-Jun	Female	Petrol	Once	Private	No	No	MTPL+	<66
25	1030	0.29589	0	0	>10	Female	Petrol	Twice	Private	Yes	No	MTPL+	66-110

How open, read and use an external files in in R? ->analytics framework
The read statement allows us to import and read the external file

It's important see also when you get some errors...:
>read.csv(file="2.2 - Dataset (from portfolio).csv") -> we get an error because R is directing into your disk in a wrong position... how can we do? Two ways
- setwd()but it's very complex because it need the exact name of the directory we want to use
(you get probably an eror)
- getwd() where wd= working directory

```
> getwd()
[1] "C:/Users/diego.zappa/OneDrive - 
dati da blackbox/files per prototipo"
```

- Go to Files(4th window) -> click on the 3 small dots -> go to the directory in which the file is -> model -> Click one more into a small tringle (if you run it now, it doesn't work again: R uses again the previous directory selected)-> click on "Set as Working directory"

```
> setwd("C:/Users/diego.zappa/Downloads")
> getwd()
[1] "C:/Users/diego.zappa/Downloads"
>
```

You can have apart the full lists of files that we have in our directory, among these we have also our file
If you save directly the file into the original working directory it works anyway?

Yes, it works automatically in that case

```
> read.csv(file="2.2 - Dataset (from portfolio).csv")
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file '2.2 - Dataset (from portfolio).csv':
tory
```

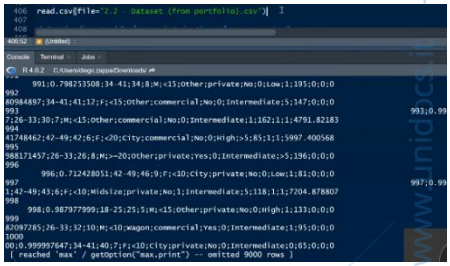
We get again an error... that's because the name doesn't correspond to the name I have in the file in my directory.
I have so to change

```
read.csv(file="2.2 - Dataset (from portfolio)(1)(1).csv")
```

it into

We can remove those suffixes and rename the file
According to this strategy, you can remove that 1 1 initially proposed in your code

!! all these steps are completed using facilities but it's also possible to use R codes (especially when you have lots of strings)



We have used the read statement simply and so you theoretically don't get a table (if you get it, you're ready to use the dataset)

```
DATASET <- read.csv(file="2.2 - Dataset (from portfolio).csv", sep = ";", dec=".", header = TRUE)
```

With "sep" you say to R to separate in different columns what is separated with ; (of for instance the ###, as often is in SAS)
With "dec" (if you have 0,9 and not 0.9) you change your decimals separator using the .
With "header" =TRUE the very first row contains the column =FALSE does not contain already data
(if you are problem uses from BB DatasetRdata)