Introduction to R - part 1

Tiago Nardi

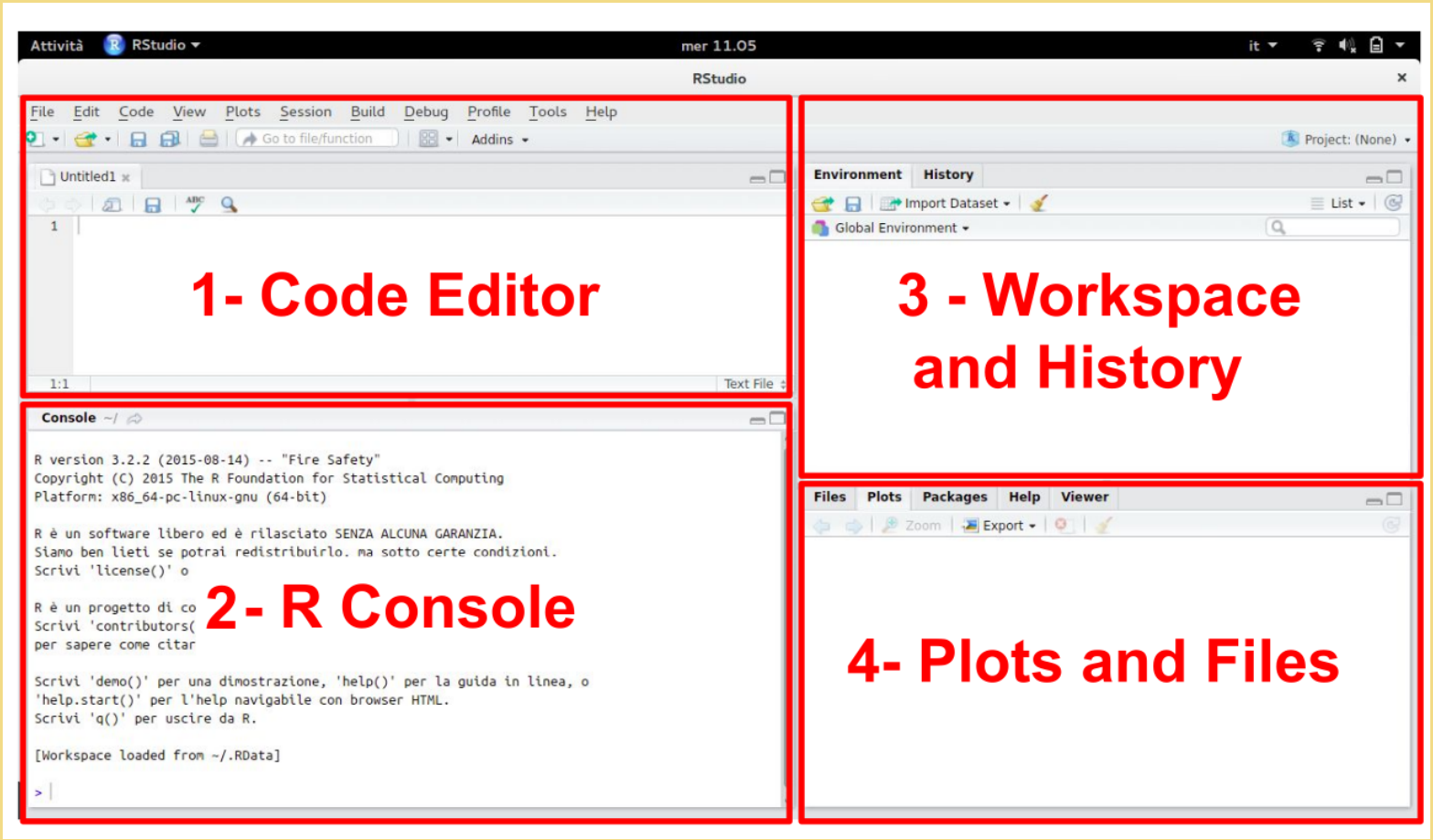
University of Pavia

Open RStudio

To open RStudio you can:

- Double Click on the Rstudio icon
- Open a terminal and run
  - On Linux: **rstudio**
  - On Mac: **open -na Rstudio**

First look into RStudio



You can change the order and the window(pane) size from the settings

1 - Code Editor

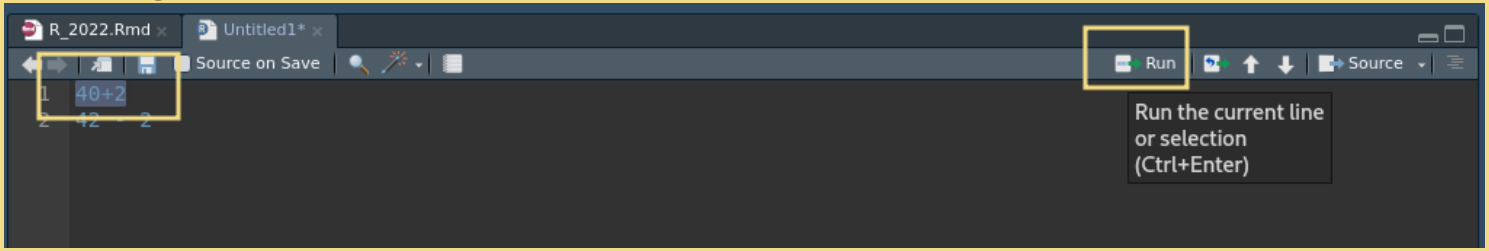
The **Code Editor** - also known as **Source Pane** - is a notepad for your R code

Script files written in R code are typically saved with a .R extension

You can save (and load) your code on your computer

Code written in the **Source Pane** will not be evaluated until you expressly run it

- Executing a Single Line: Ctrl+Enter (or use the Run toolbar button) to execute the line of code where the cursor currently resides
- Executing Multiple Lines: Ctrl+Enter key (or use the Run toolbar button) after selecting the lines

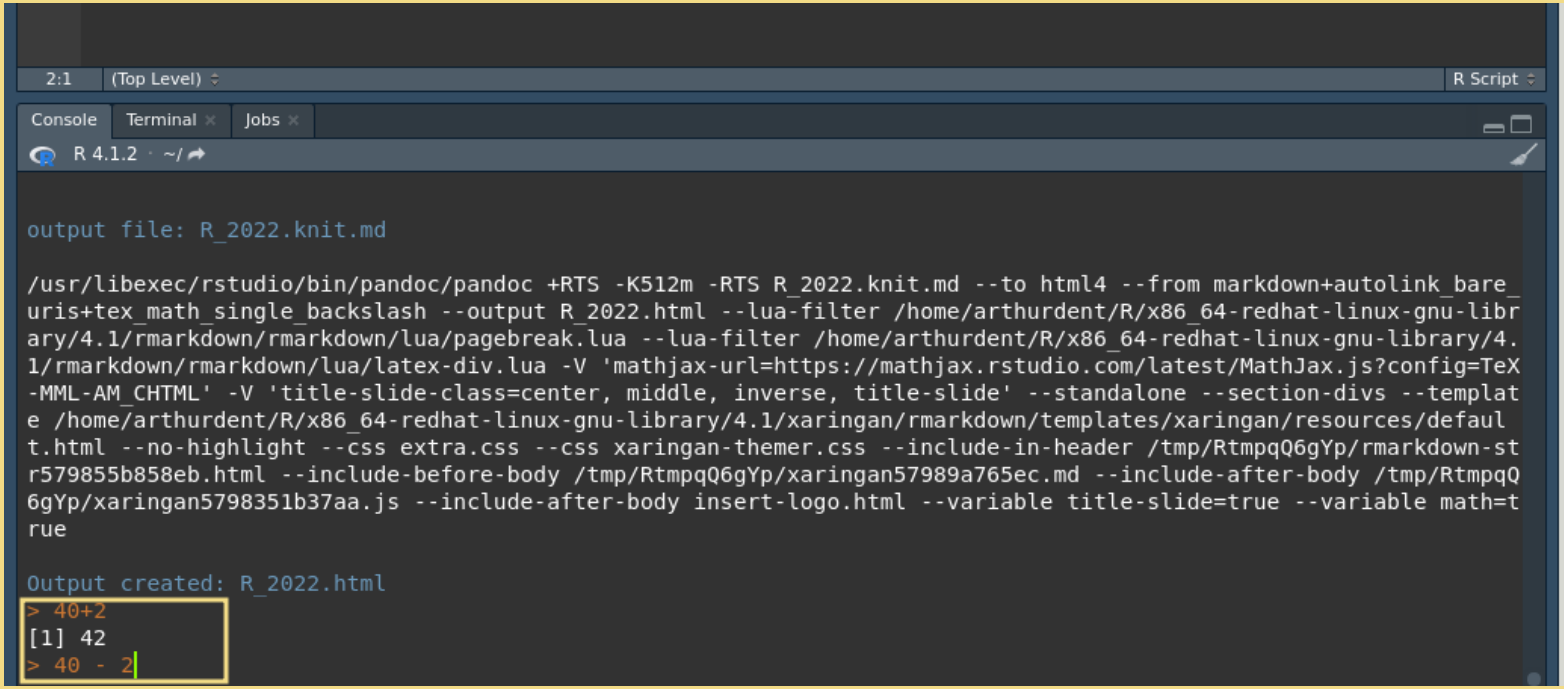


2 - Console

The core of R, it's where the code is evaluated

The prompt (> by default) indicates that the console is ready to run new code

Click on the **console**, type directly the commands and press enter ↵ to run them



2 - Console

Each command will be executed one at a time

You can run commands or in the console or in the **Source Pane**

The console is useful for a quick test or for debugging

Previous commands will be logged in the **History**

But better to write most of your code on the **Code Editor**

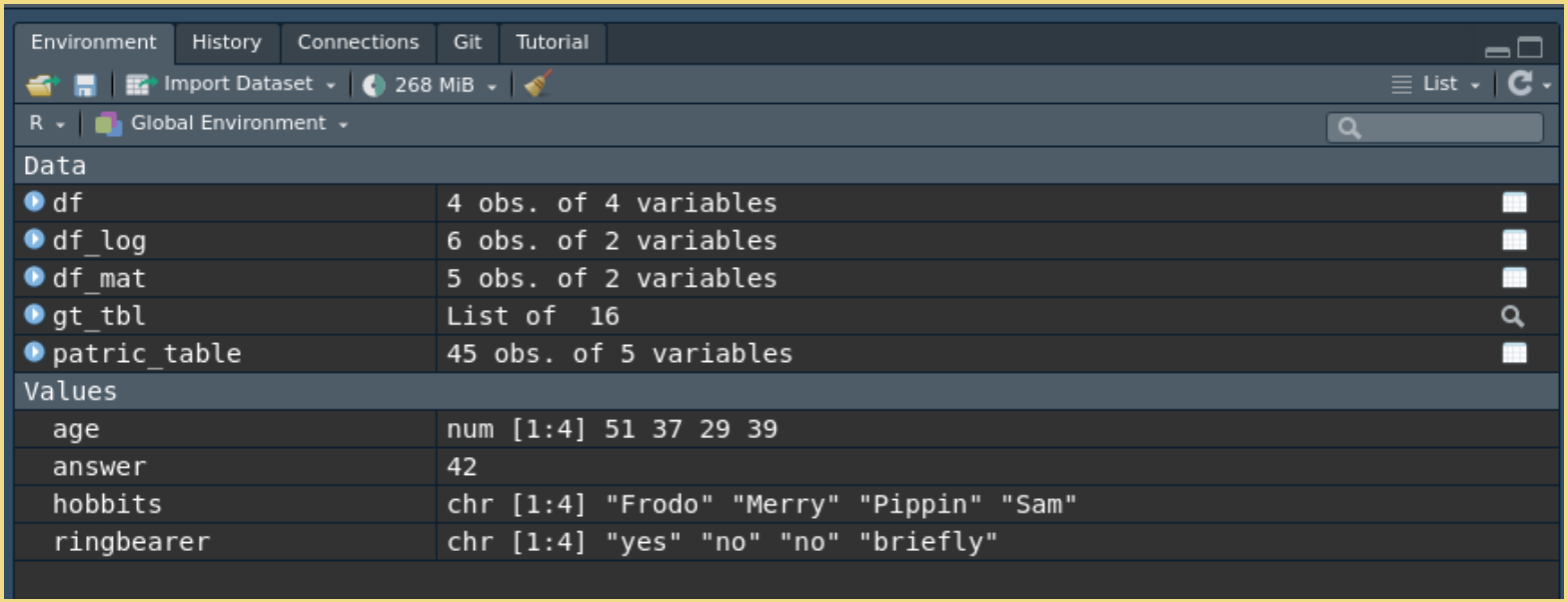
3- Workspace and History

The Environment tab will contain all your data and objects

You can click on objects to inspect them

You will have informations such as the number of rows and observation

History contain previous used commands



4 - Plots and Files

This pane contains various informations:

- Files: your **present working directory**, and the file present in that folder. You can change your working directory with a graphic interface from here
- Plots: your plots (duh), you can zoom on them and export in various formats and sizes
- Packages: you can load (and unload) previously downloaded **packages** (extension of R capability)
- Help: online R documentation, you can search a command on it, or write in the console *?command*

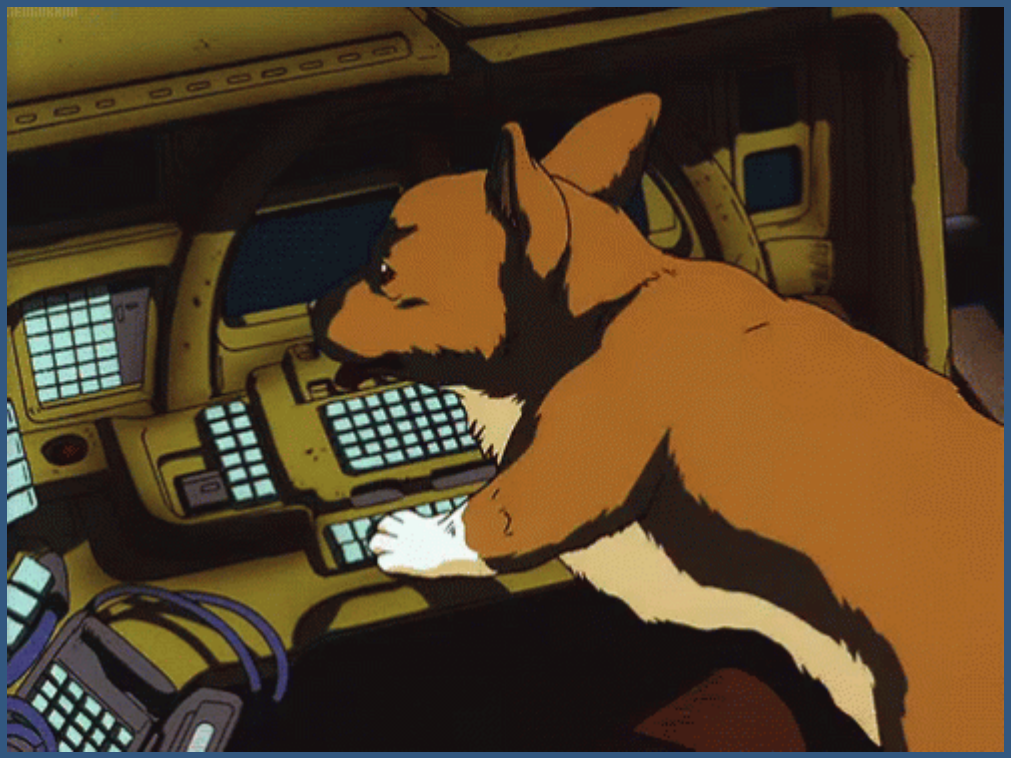
```
?chisq.test
```

Two tips

Remember:

- RStudio (as other IDEs) has a very useful feature called "code completion": use the Tab key (↹) to autocomplete the full name of an object or a function, or a file/folder location
- Comments can be used to explain R code, and to make it more readable. You can also use them to prevent execution when testing alternative code. Comments starts with a #, when executing the code R will ignore anything that starts with #

Let's start



Getting our feet wet

To start, we can use R as a fancy calculator

The commands are the the basic mathematical operators

Operator	Description
+	Sum
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

Getting our feet wet

Try these commands in the **console**

```
40 + 2

44 - 2

42/11

4^3
```

Getting our feet wet

Executing the code should get you these results

```
40 + 2
```

```
## [1] 42
```

```
44 - 2
```

```
## [1] 42
```

```
42/11
```

```
## [1] 3.818182
```

```
4^3
```

```
## [1] 64
```

Logical operators

You can use logical operators: they check if a statement is true and output a Boolean value (TRUE/FALSE)

The logical operators are:

Operator	Description
>	Greater than
>=	Greater than or equal tp
<	Lesser than
<=	Lesser than or equal to
==	Equal to
!=	Not equal to
&	AND
|	OR
!	NOT

Logical operators

Try these commands in the **console**

```
42 > 42

42 >= 42

42 == 42

42 != 71

42 == 42 | 42 != 42

42 == 42 & 42 != 42
```

```r
42 > 42
```

```
## [1] FALSE
```

```r
42 >= 42
```

```
## [1] TRUE
```

```r
42 == 41
```

```
## [1] FALSE
```

```r
42 != 71
```

```
## [1] TRUE
```

```r
42 == 42 | 42 != 42
```

```
## [1] TRUE
```

```r
42 == 42 & 42 != 42
```

```
## [1] FALSE
```

AND OR

OR |

If at least one the statements is **TRUE** -> **TRUE**

If both are **FALSE** -> **FALSE**

AND &

If two statements are both **TRUE** -> **TRUE**

If at least one of the statements is **FALSE** -> **FALSE**

R objects

The entities that R creates and manipulates are **objects**

Objects are stored by name in the active memory of the computer

To create an **object**, we need to give it a name followed by the assignment operator "<-"

```
variable_name <- value
```

After assigning an object it should appear in the **environment tab**, where you can inspect it

You can call the object by its name and use it with any function

R objects

1) **Object** name must start with a letter and can be a combination of letters, digits, period . and underscore _. If it starts with period ., it cannot be followed by a digit

2) **Object** names are case-sensitive (*age*, *Age* and *AGE* are three different variables)

3) Reserved words cannot be used as **Object** name (TRUE, FALSE, NULL, if...)

Try it!

Assign a number to a variable of your choosing

```
answer <- 42
Try some operations (arithmetic and logical)
```

```
answer
answer * 2
answer - answer
answer > 5
```

Try it!

```
answer <- 42
answer
```

```
# [1] 42
```

```
answer * 2
```

```
# [1] 84
```

```
answer - answer
```

```
# [1] 0
```

```
answer > 5
```

```
# [1] TRUE
```

R objects

If you assign a new value to a named **object** you overwrite it

```
answer <- 42
answer
```

```
# [1] 42
```

```
answer <- 20
answer
```

```
# [1] 20
```

You can assign a variable to the variable itself

```
answer <- answer * 2
answer
```

```
# [1] 40
```

Variables and vectors

Strings (a series of letters) must be enclosed in quotes, either double(“ “) or single (‘ ‘)

Numbers enclosed in quotes are considered strings, not numbers and can't be used for mathematical operations

Variable

single value (number or a string)

```
name <- "string"
```

Vector

an ordered collection of values. Create it using c() function with its elements separated by a comma

```
hobbits <- c("Frodo", "Sam")
```

R data types

Numeric

decimal values

```
grade <- 18
```

Character

letters or numbers enclosed by quotes

```
my_name <- “Bilbo”
```

Logical

a variable that can have a value of TRUE or FALSE

```
logic_var1 <- TRUE
```

Exercise

How to check the class of a variable:

```
class(objects)
```

Try:

```
var1 <- "777"
var2 <- 777
class(var1)
class(var2)
```

Changing types

To change the variable/vector type: From any type to character:

```
name <- as.character(name)
```

From any type to numeric (a character cannot be changed into a number):

```
name <- as.numeric(name)
```

From any type to factor:

```
name <- as.factor(name)
```

Factors

Factor: objects which are used to group data into categories. Each group is assigned to a level which identifies the group

How to convert a variable into a factor:

```
variable_name <- as.factor(variable_name)


f <- c("East","West","East","North","North")
class(f) # It's a character
f <- as.factor(f) #Function to convert a variable into a factor
```

Check levels of a factor:

```
levels(variable_name)
levels(f) # "East" "North" "West"
```

By default, R sorts the levels of a factor alphabetically

Data frames

A table in which each **column** contains values of one variable and each row contains one set of values from each column.

	Column_1	Column_2	Column_3	Column_4
Row_1	value	value	value	value
Row_2	value	value	value	value

Data frames

To create a data frame from vectors

```
dataframe_name <- data.frame(column1,column2,column3...)
```

Let's first create 3 vectors (they must be of the same length)

```
hobbits <- c("Frodo","Merry","Pippin","Sam")
age <- c(51,37,29,39)
ringbearer <- c("yes","no","no","briefly")
# and now let's use these vectors to build our data frame
df <- data.frame(hobbits,age,ringbearer)
```

Data frames

Inspect the dataframe clicking on it in the **environment tab**

Or write in the console

```
View(df)
```

hobbits	age	ringbearer
Frodo	51	yes
Merry	37	no
Pippin	29	no
Sam	39	briefly

Probably better to codify the last column as a boolean variable or use only "yes" or "no":

Being consistent is fundamental when using table

Data frames

Also it can be done in a single command

```
data.frame(hobbits <- c("Frodo","Merry","Pippin","Sam"),
           age <- c(51,37,29,39),
           ringbearer <- c("yes","no","no","briefly"))
```

```
##   hobbits....c..Frodo....Merry....Pippin....Sam.. age....c.51..37..29..39.
## 1                                                Frodo                    51
## 2                                                Merry                    37
## 3                                               Pippin                    29
## 4                                                  Sam                    39
##   ringbearer....c..yes....no....no....briefly..
## 1                                           yes
## 2                                            no
## 3                                            no
## 4                                       briefly
```

Data frames

To access the **columns** of a data frame:

```
dataframe_name$column_name
```

To change the **object** type:

```
dataframe_name$column_name <- as.character(dataframe_name$column_name)
```

You can add a column to a **data frame** assigning a **vector** to it

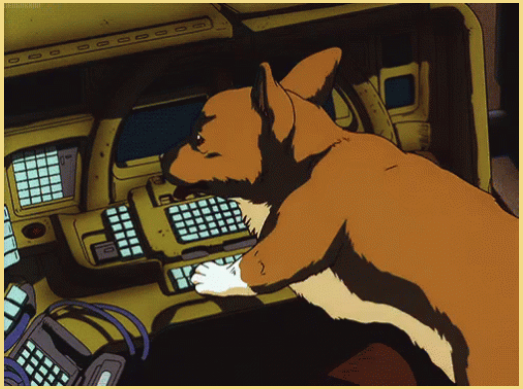
```
dataframe_name$new_column <- vector
```

Exercise

Create a vector to add to the data frame

It should have the same length of the other columns

Add it to the data frame and inspect it



Results

```
breakfast_taken <- c(2,3,2,4)
df$breakfasts <- breakfast_taken
```

hobbits	age	ringbearer	breakfasts
Frodo	51	yes	2
Merry	37	no	3
Pippin	29	no	2
Sam	39	briefly	4

How to manage external files

Set working directory

Every time you start a R session it will start in a specific directory

To know your current **working directory** you can use the command

```
getwd()
```

To change the **working directory** directory

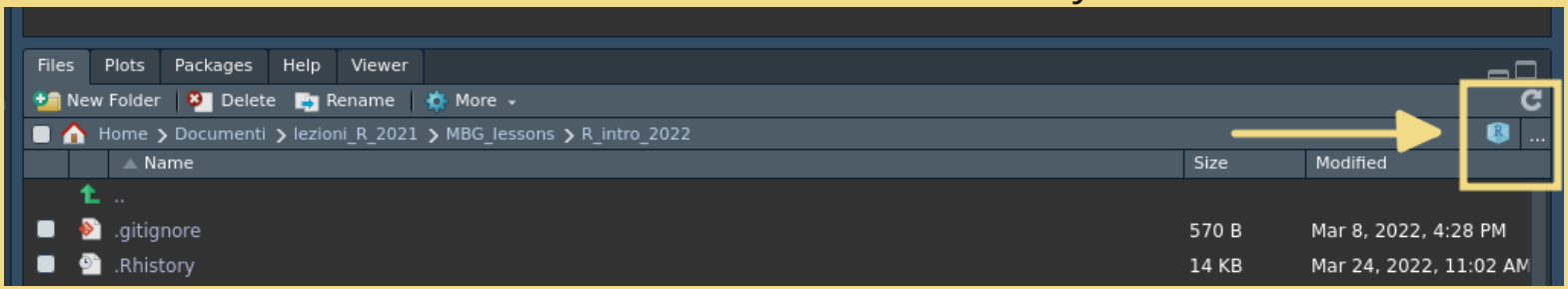
```
setwd("/path/to/my/directory")
```

Use *../* to go to the parent directory (the folder containing your current folder)

Set working directory

You can also set up the directory through the graphical interface:

Click on the three dots in the **files tab** to select a directory



Tables

Table can be (and often are) saved as csv/tsv files

These are text files, in which **v**alues are **s**eparated by a **c**omma (,) or by a **t**ab

If you open a CSV table with a notepad you will read something like

```
Frodo,51,yes
Merry,37,no
Pippin,29,no
Sam,39,briefly
```

A TSV table will look like

```
Frodo   51    yes
Merry   37    no
Pippin  29    no
Sam   39    briefly
```

Other separators are possible but most of the cases will use comma, semicolons or tab

Tables

Another aspect is the presence of a **header**

The **header** is a first row that contains the names of the columns

```
Hobbit,age,ringbearer
Frodo,51,yes
Merry,37,no
Pippin,29,no
Sam,39,briefly
```

When you load a file check if the table contains a **header** Possible mistakes:

- using the first row as column names
- using the column names as values

Open an external table

We will work with the table *patric_redux.csv*

You can download the table from the course gdrive, and save it into a folder of your choice

https://drive.google.com/file/d/1UMmE_KZial63WJVleg4xbX6GXCtCTd45/view?usp=sharing

Otherwise you can download it from github

https://github.com/tiagonardi/R_intro_2022/blob/gh-pages/patric_redux.csv

Change your R **working directory** to the folder where you have saved the table

```
setwd("/path/to/my/directory")
```

Or navigate to the folder using **Files tab**

Reading a table

You can read the table using this command

```
dataframe_name <- read.csv("File_name",
                           header = TRUE,
                           sep=",")
```

The first parameter of "read.csv" is the name of the file and its position relative to the **working directory**

The second specify the presence of a header

The third which separator has been used in the table

Functions have often default value for some of the parameters that are used if are not specified

For example *read.csv* will use *sep=","* if not specified

Reading a table

There are various commands that can be used to read a table in R

They are all variation of the more general command *read.table*, with different defaults

Command	Defaults
read.table	sep='', header=FALSE, dec = '.'
read.csv	sep=',', header=TRUE, dec = '.'
read.csv2	sep=';', header=TRUE, dec = ','
read.delim	sep=' ' ,header=TRUE, dec = '.'
read.delim2	sep=' ', header=TRUE, dec = ','

The read **functions** indicated with a 2 are designed for countries that use "," to indicate decimals (e.g. Italy 🇮🇹)

Exercise

After setting up the **working directory** read the downloaded table, and assign it to an object named *patric_table*

```
patric_table <- read.csv("patric_redux.csv",header = T,
                         sep=",")
```

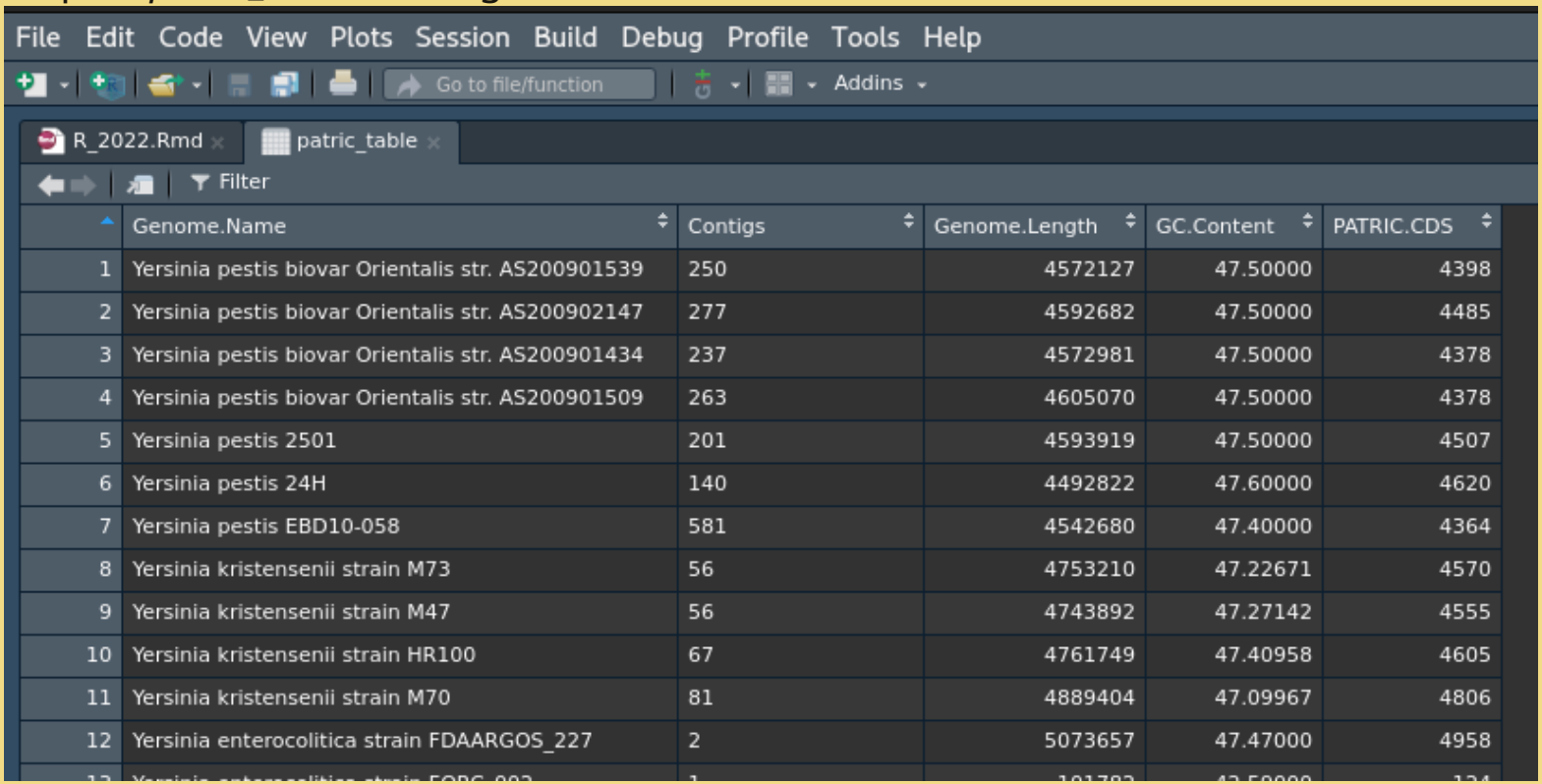
You can run the object name in the **console** to have a look at it, or (better) click the name in the **environment tab** or use *view(patric_table)*

Also try to read the table with a wrong separator (";")

```
wrong_table <- read.csv2("patric_redux.csv")
```

Exercise

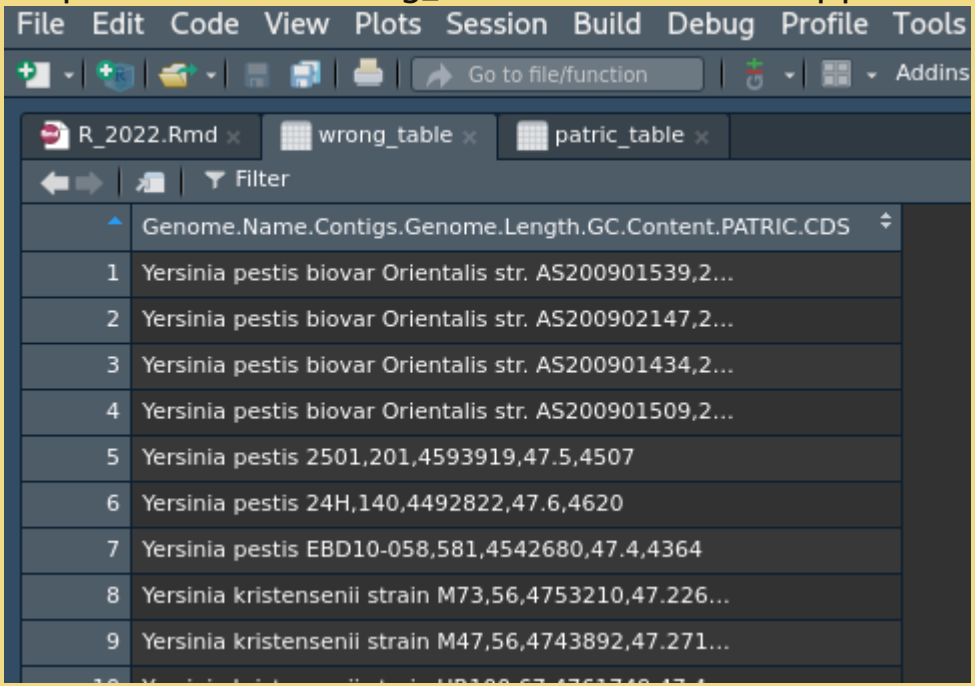
Inspect *patric_table* clicking on it on the **environment tab**



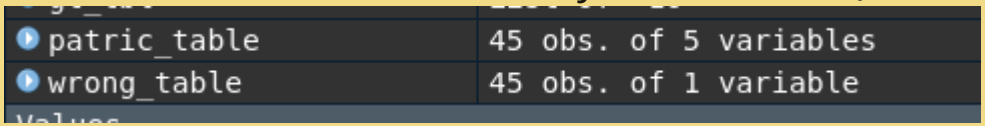
	Genome.Name	Contigs	Genome.Length	GC.Content	PATRIC.CDS
1	Yersinia pestis biovar Orientalis str. AS200901539	250	4572127	47.50000	4398
2	Yersinia pestis biovar Orientalis str. AS200902147	277	4592682	47.50000	4485
3	Yersinia pestis biovar Orientalis str. AS200901434	237	4572981	47.50000	4378
4	Yersinia pestis biovar Orientalis str. AS200901509	263	4605070	47.50000	4378
5	Yersinia pestis 2501	201	4593919	47.50000	4507
6	Yersinia pestis 24H	140	4492822	47.60000	4620
7	Yersinia pestis EBD10-058	581	4542680	47.40000	4364
8	Yersinia kristensenii strain M73	56	4753210	47.22671	4570
9	Yersinia kristensenii strain M47	56	4743892	47.27142	4555
10	Yersinia kristensenii strain HR100	67	4761749	47.40958	4605
11	Yersinia kristensenii strain M70	81	4889404	47.09967	4806
12	Yersinia enterocolitica strain FDAARGOS_227	2	5073657	47.47000	4958

Practicals

Inspect also the *wrong_table* to see what happens with a wrong separator



R was unable to divide correctly the columns ("1 variable")



Exercises

Check if the table has been imported correctly

```
head(dataframe_name) # returns the first part of a data frame
str(dataframe_name) # compactly display the internal structure
summary(dataframe_name) # display statistics for sample and subgroups
```

Check column and row names in *patric_table*:

```
colnames(dataframe_name)
rownames(dataframe_name)
```

Results

```
head(patric_table)
```

```
##           Species    ID Contigs Genome.Length GC.Content PATRIC.CDS
## 1 Yersinia pestis AS539     250       4572127       47.5       4398
## 2 Yersinia pestis AS147     277       4592682       47.5       4485
## 3 Yersinia pestis AS134     237       4572981       47.5       4378
## 4 Yersinia pestis AS509     263       4605070       47.5       4378
## 5 Yersinia pestis  A251     201       4593919       47.5       4507
## 6 Yersinia pestis   24H     140       4492822       47.6       4620
##   Isolation_location Source
## 1           Isengard Hobbit
## 2          The Shire Hobbit
## 3              Rohan  Human
## 4             Mordor  Human
## 5          The Shire Hobbit
## 6             Erebor Dragon
```

Results

```
str(patric_table)
```

```
## 'data.frame':    58 obs. of  8 variables:
##  $ Species           : chr  "Yersinia pestis" "Yersinia pestis" "Yersinia p
##  $ ID                : chr  "AS539" "AS147" "AS134" "AS509" ...
##  $ Contigs           : int  250 277 237 263 201 140 581 56 56 67 ...
##  $ Genome.Length     : int  4572127 4592682 4572981 4605070 4593919 4492822
##  $ GC.Content        : num  47.5 47.5 47.5 47.5 47.5 ...
##  $ PATRIC.CDS        : int  4398 4485 4378 4378 4507 4620 4364 4570 4555 46
##  $ Isolation_location: chr  "Isengard" "The Shire" "Rohan" "Mordor" ...
##  $ Source            : chr  "Hobbit" "Hobbit" "Human" "Human" ...
```

Results

```
summary(patric_table)
```

```
##    Species                ID                  Contigs      Genome.Length
##  Length:58          Length:58          Min.   :  1.0   Min.   :3840239
##  Class :character   Class :character   1st Qu.: 65.5   1st Qu.:4541366
##  Mode  :character   Mode  :character   Median :152.0   Median :4596468
##                                        Mean   :175.4   Mean   :4612834
##                                        3rd Qu.:249.0   3rd Qu.:4738290
##                                        Max.   :581.0   Max.   :5073657
##    GC.Content      PATRIC.CDS   Isolation_location    Source
##  Min.   :42.59   Min.   :3731   Length:58          Length:58
##  1st Qu.:47.09   1st Qu.:4474   Class :character   Class :character
##  Median :47.40   Median :4604   Mode  :character   Mode  :character
##  Mean   :47.23   Mean   :4756
##  3rd Qu.:47.50   3rd Qu.:4770
##  Max.   :48.33   Max.   :9392
```

Results

```
colnames(patric_table)
```

```
## [1] "Species"            "ID"                 "Contigs"
## [4] "Genome.Length"      "GC.Content"         "PATRIC.CDS"
## [7] "Isolation_location" "Source"
```

```
rownames(patric_table)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29"
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44"
## [46] "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56" "57" "58"
```