Rbasics
PhD toolbox

Matteo Chialva
Martino Adamo

Course organization

RBASICs

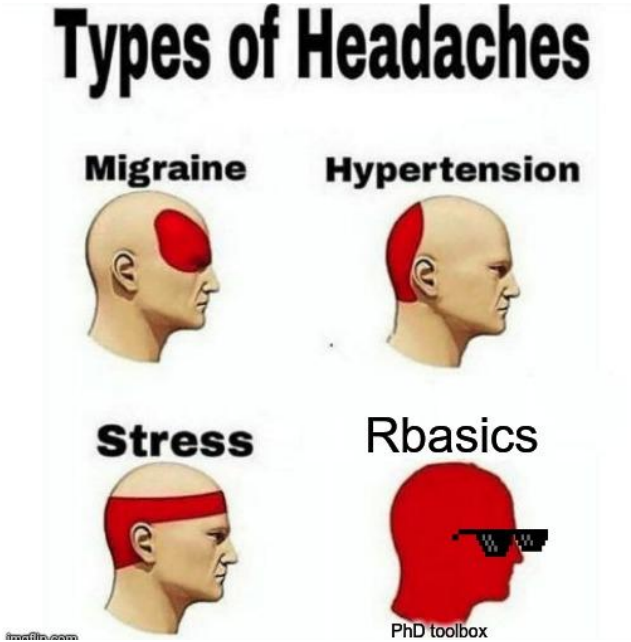
Day 1

- Part I - Operative Introduction to R
- Part II - Spread Sheets
- Part III - Manipulating data with dplyr (may shift in part to day2)

Day 2

- Part IV - Text strings
- Part V - Tips and Tricks
- Part VI - Base graphics in R

Part I Operative introduction to R



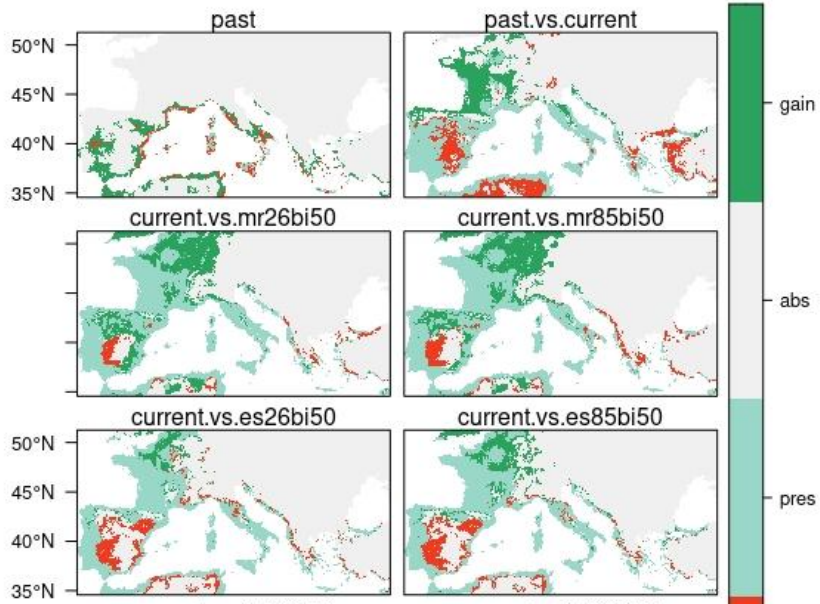
Part 1 contents

- why R instead of other software (excel, SPSS..)
- R is not only a statistics software
- using R through RStudio: the main features
- installing and updating R and RStudio software
- base and advanced packages installation using external repositories (BioConductor, github..).
- systems dependencies and packages
- R objects type: vectors, matrices, dataframes, lists & functions
- data import and export (read.table; write.table; load; WriteXLS)
- practical tips for organizing scripts, files and folders
- “don't panic!”: guidelines to face issues during a data analysis session in R (error messages and warnings, manuals, tutorials, forums, reference sites)

teachers for a week

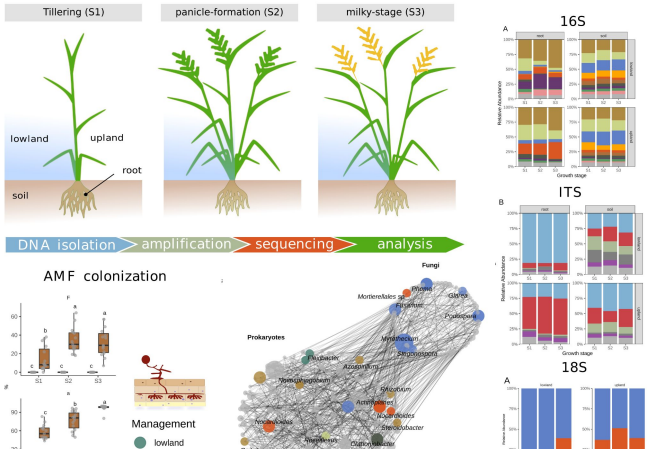
Dr. Martino ADAMO

Postdoc researcher since 2018, I work on plants diversity conservation and orchid mycorrhizas using bio-molecular tools and boring statistical model to study traits influence on species spatial distribution.



Dr. Matteo CHIALVA

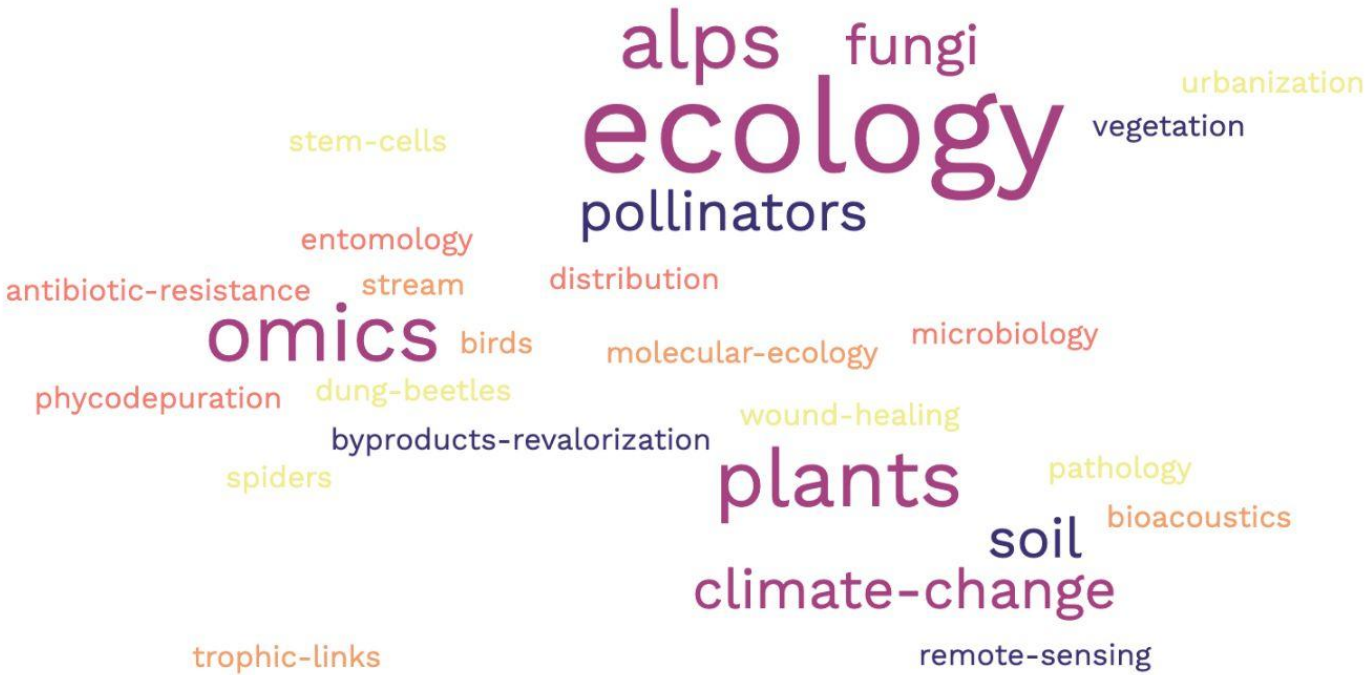
He is post-doctoral resercher since 2017 and his research focuses on plant-microbes interactions in model crop species by using multi-omics approaches from transcriptomics to metagenomics. By coupling these tools with systems biology and biostatistics he is interested in linking soil microbiota diversity and functioning to plant responses and ecosystem services.



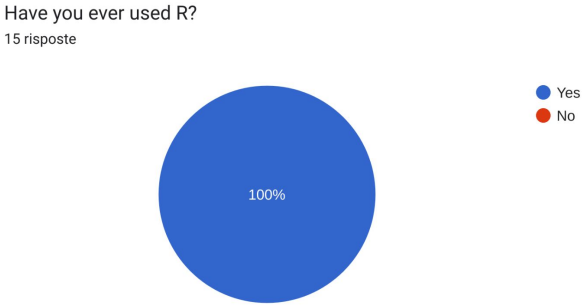
students forever

PhD is the first step of academic career: if you want to continue you are going to study every single day of your working life.

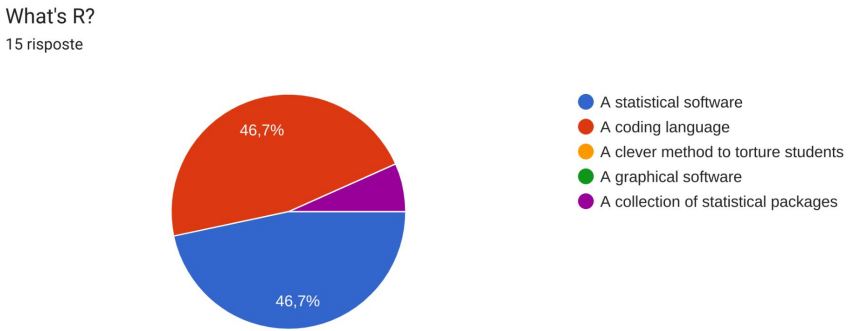
We asked you to explain your study field in three keywords and this is results:



What’s R?



All of you know that R exist and have already used a time in their life



but….less then half of you know what R really is!

R *per se* is a programming environment for statistics and graphics, not a simple standalone software. Much more similar to a coding language rather than a stat. software

why R instead of other software

1. **R is free and open-source**

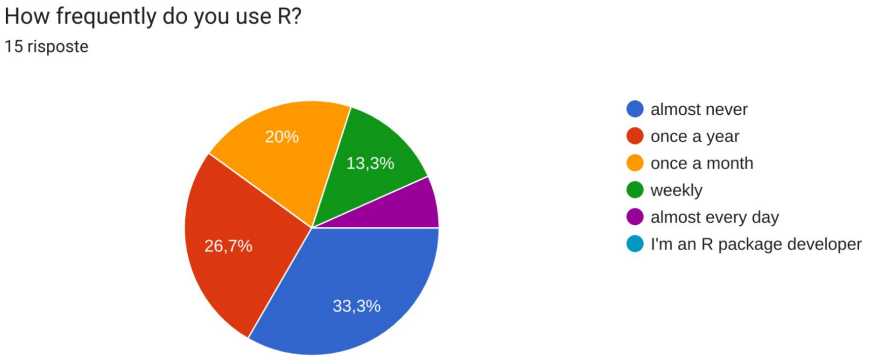
2. **R can do almost everything**.
   No seriously, everything.

3. R can address many of the challenges of performing **reproducible research**

4. Learning **R will make you a more attractive candidate**

What's R?

Most of you rarely used R



Why you should use it **every day**:

- open source and cross platform
- active community of developers: i.e. access to the state-of-the-art methods
- extensive documentation everywhere (books, online, forums etc..)
- great graphics capabilities: you can manage all the type of plots you can imagine (and mainly those that you could not!)

why R instead of other software

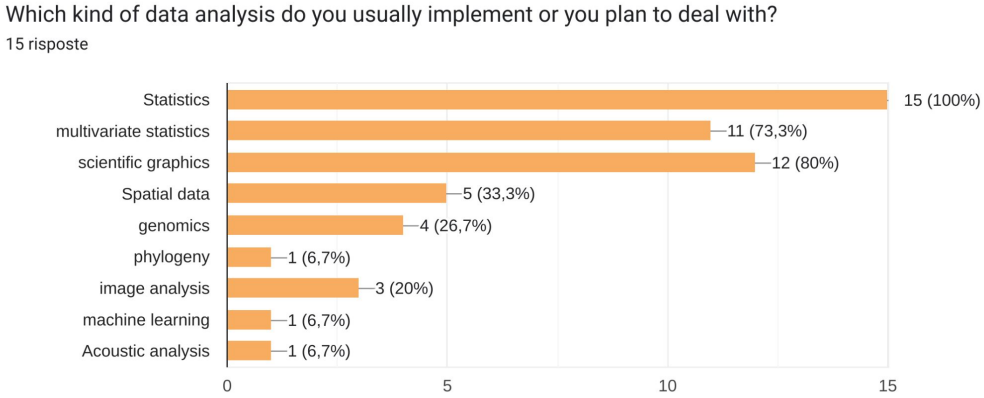
1. **R is open-source** => anyone can access the underlying code used to run the program and add their own code for free. Thus:
    a. will always be able to perform the newest statistical analyses as soon as anyone thinks of them;
    b. will fix its bugs quickly and transparently; and
    c. has brought together a community of programming and stats nerds (a.k.a., useRs) that you can turn to for help.
2. **R can do almost everything**. No seriously, everything.
    a. basic stats
    b. math
    c. frequentist and Bayesian statistics.
    d. excel suffs
    e. it saves money!
    f. LaTeX and Word stuffs
    g. Multicore processing
    h. Phylogeny
    i. genomics
    j. make fun
    k. ...
3. R can address many of the challenges of performing **reproducible research**
4. A final reason you should become a useR is that R is increasingly being used as an industry standard in the realm of data analytics, also known as “data science.” Learning **R will make you a more attractive candidate** if you apply for non-academic jobs and academic positions.

What you'll learn

With R you can do analyses in all the scientific disciplines.

Here you'll learn R and its main functions to deal with tables (any type of data) and graphics.

You'll not learn statistics!



Our aim is to provide you universal tools to deal with your data in a reproducible way

Reproducible research

What is reproducibility?

The possibility to reproduce someone else results. It's part of the scientific method (it should be...)

How can you ensure reproducibility?

One of the best way is to use **CODE** since it is universal and can be easily shared and documented

Reduce (and possibly avoid) manual data manipulation steps (e.g. copy-paste).

- prone to introduce errors
- time inefficient
- impossible to reproduce

If you cannot avoid that you have to document what has been done and what was the purpose.

installing R and RStudio

Almost everyone run R through RStudio GUI (graphic user interface) which is a software which helps you to write in R.

1. Install R - https://cran.r-project.org/

2. Install RStudio (free version) - https://posit.co/downloads/

Note: in both cases installation procedure *dependecies* on your operating system.

(a dependency is additional code that a programmer wants to call)

updating R and RStudio

Periodically you would have to update both R and/or RStudio.

1. Update R - different options available

   The most effective way is to completely remove the previous R version and install the new one.

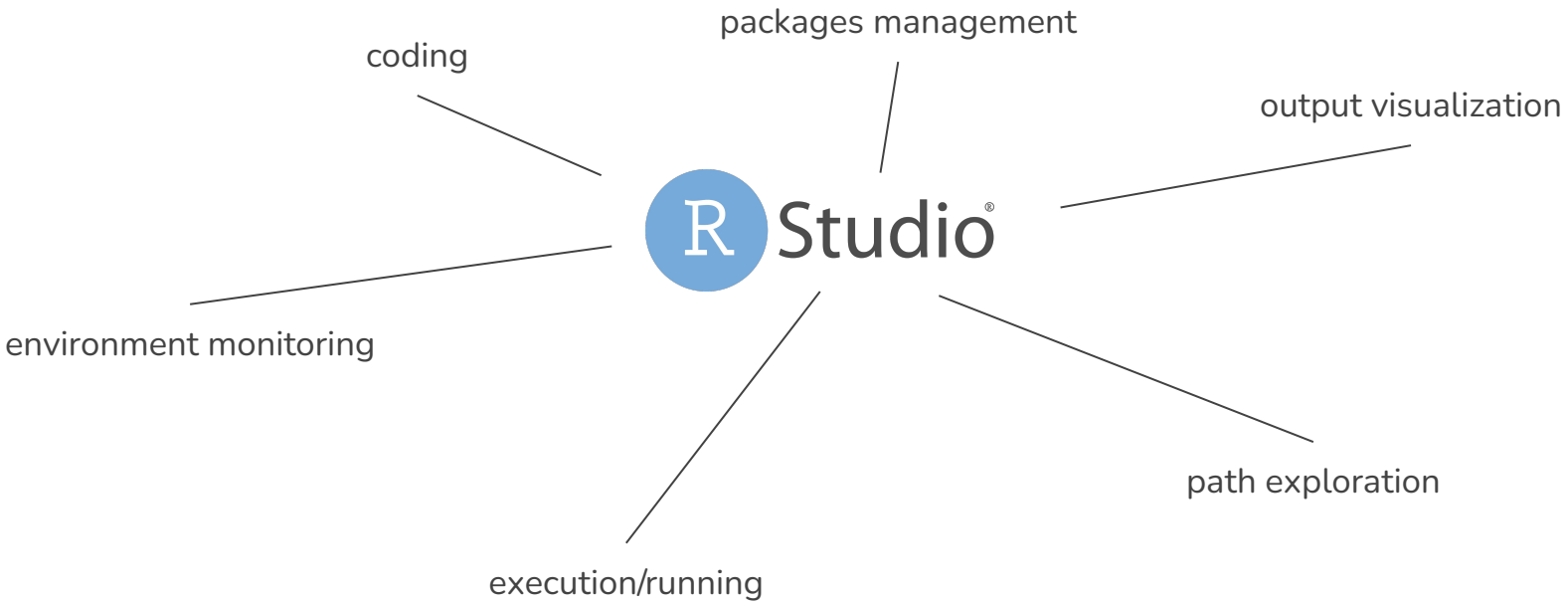
Note: you have to fully re-install all the packages or move the old packages in the new directory.

2. Update RStudio

   Download and install the new version (removal of the previous version is highly suggested!)

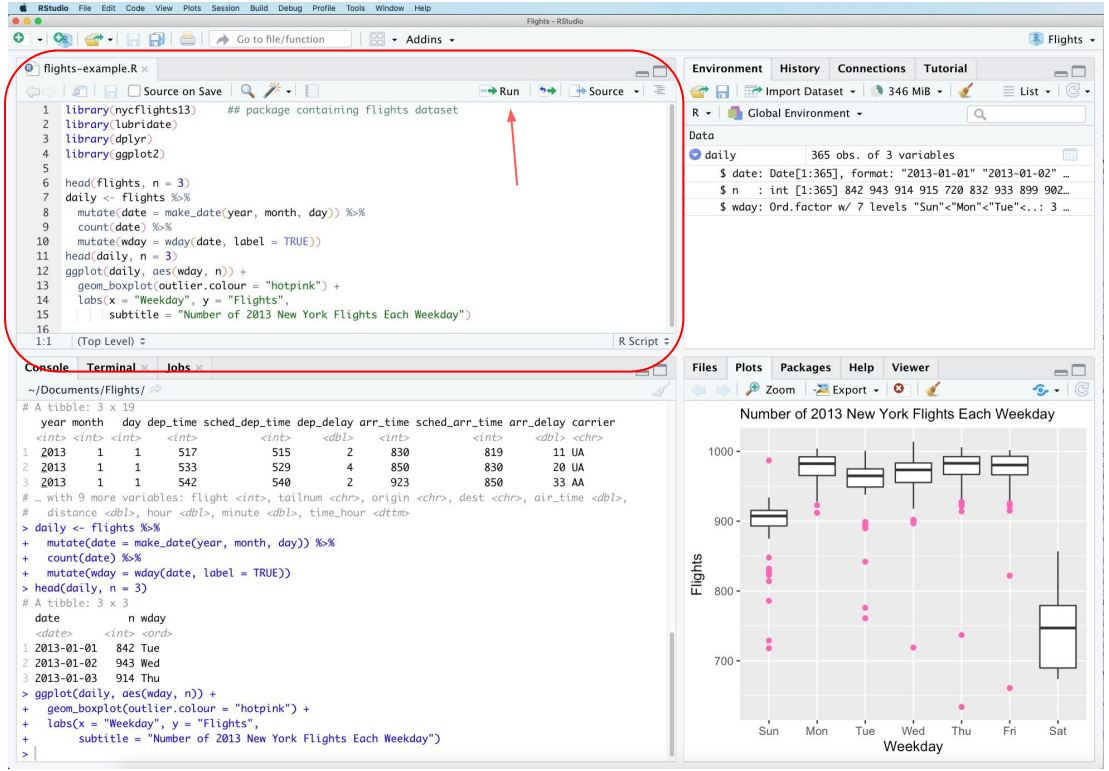
using R through RStudio: the main features

RStudio is an integrated development environment for R



using R through RStudio: the main feature

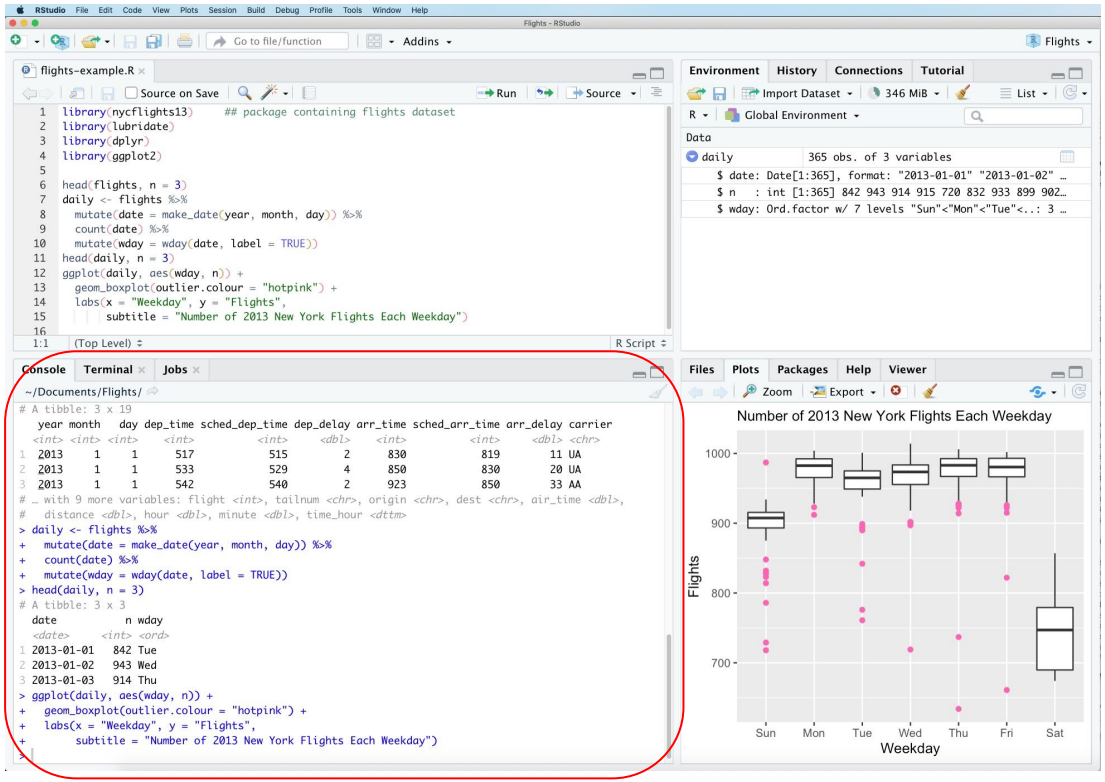
coding
here you can write your code
...
90% of the time on Rstudio you will work here
or here:



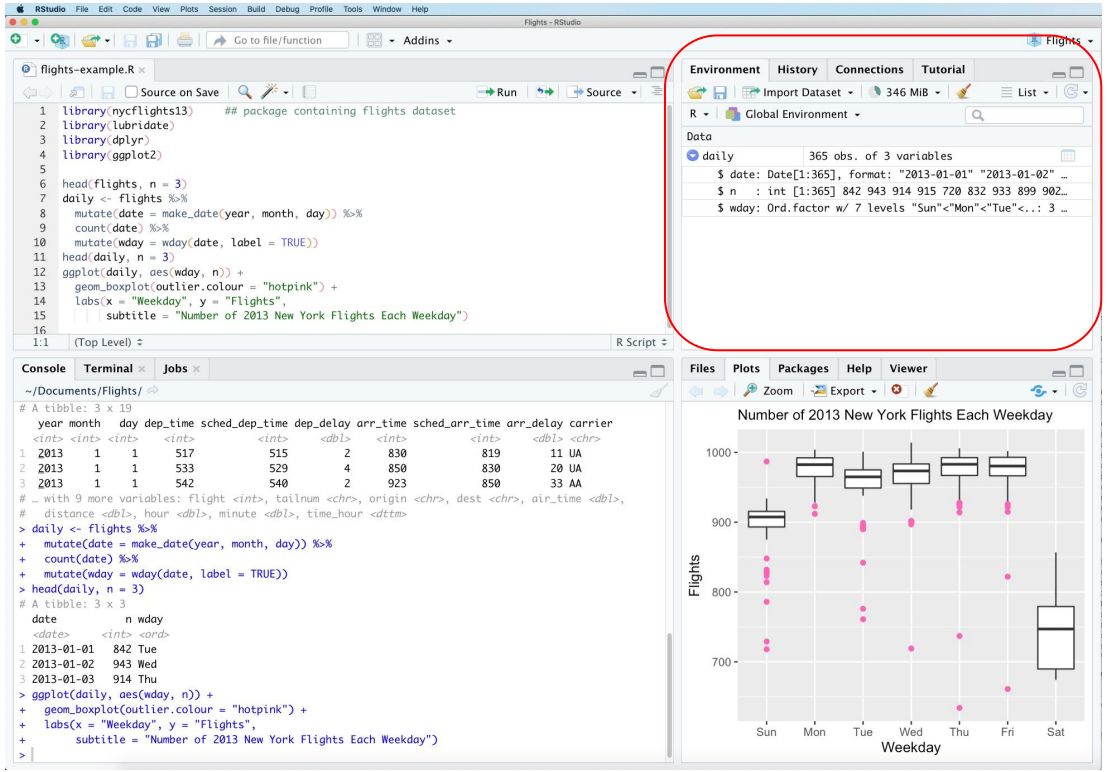
using R through RStudio: the main feature

console
this is actually R

hybrid function
coding::
::running::
::results viz



using R through RStudio: the main feature

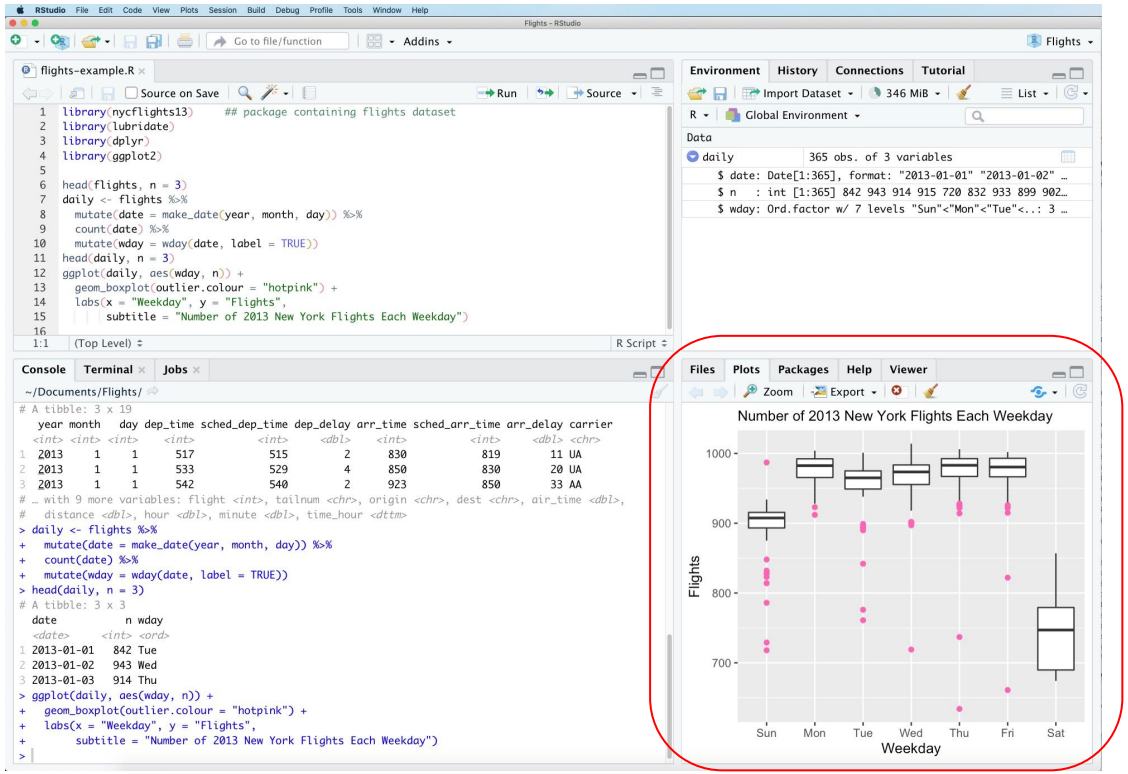


environment
this it the best Rstudio intuition

R commands work on objects

here you can see the objects you're working on

using R through RStudio: the main feature



plots and explorer

here you see graphical outputs, help pages and your packages

moreover you can explore your folders and files

working directory

wd is where you're working and where R looks for files to import or to save outputs ... default *wd* is into the R folder in your home, we suggest to set as *wd* the folder where your input files are stored.

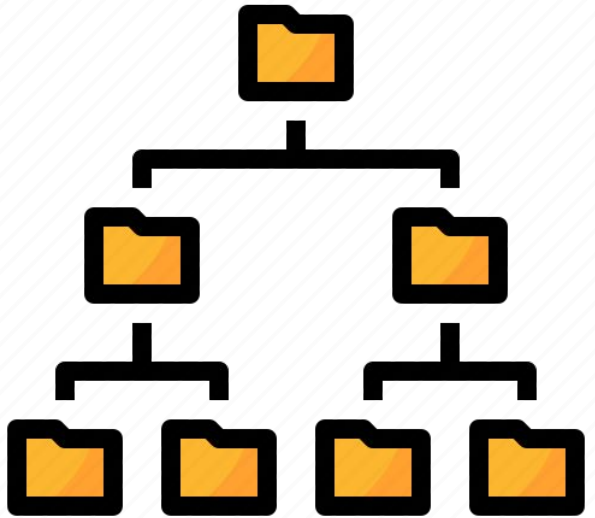
```
setwd("your_path")
```

```
getwd()
```

Example path formats in different OS.

/ubuntu/unix/macos/your_dir/your_filename

"C:\Users\your_folder\your_filename"



packages installation

- **CRAN**: it is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.

```
>install.package("pkg.name")     OR   via RStudio interface
```

- **Bioconductor**: develop, support, and disseminate free open source software that facilitates rigorous and reproducible bioinfo analysis

```
>BiocManager::install(c("package1", "package2"))
```

- **From external repositories, e.g. from GitHub**: it is an online software development platform. It's used for storing, tracking, and collaborating on software projects

- using **devtools**: it makes package development easier by providing R functions that simplify and expedite common tasks

```
>devtools::install_github("repository_name")
```

packages installation exercise

- install CRAN and special packages - we'll use most of these packages later on..
  - install a package from CRAN:
    - 'devtools'
    - 'BiocManager'
    - 'stringr'
    - 'reshape'
    - 'tidyverse'

>install.packages(c("devtools", "BiocManager", ...))

packages installation exercise

- install a package from GitHub (https://github.com/jfq3/QsRutils)
  - https://github.com/jfq3/QsRutils

Github packages are installed through the devtools::**install_github**() function

> install_github("jfq3/QsRutils")

packages installation exercise

- install a package from Bioconductor (Biostrings)
  - https://www.bioconductor.org/packages/release/bioc/html/Biostrings.html

Bioconductor packages are installed through the BiocManager::**install**() function

>BiocManager::install("Biostrings")

packages installation exercise

If for some reason installation fails R reports you some errors
Let’s check:

- they are in the list of the installed packages
- they can be loaded. Use the function **library()**

Few notes on programming

- **R** is the most simple coding language available
- You don’t need to be an experience programmer to use R language
- as every language, R has its own grammar
- Let RStudio help you in writing in R (as Word helps you writing in another language)
- Don’t panic, but…you’ll need to memorize a lot of functions
- most of R functions are simple and intuitive, the best way is to practice!

Few notes on programming

- coding is case sensitive (*e.g.* **object** and **Object** are not the same thing!)
- “**#**” symbol in general excludes lines that the language runs. It is used to comment-out lines with alternative/inactive code or to annotate your script.
- text strings are in quotes: e.g. **“Object” is the string text name of Object**
- objects names cannot contain spaces or mathematical operators
- possibly keep objects and functions name different!
- objects are created, replaced, filled with the symbol “<-”

  object **<-** ”I’m an object”

- “<-” and “=” are alternatives

data type

- numeric (num) = numbers

```
num [1:91] 1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 ..
```

- factor (factor) = categories, ordinal numbers, boolean

```
Factor w/ 2 levels "due","uno": 2 1
```

- character (char) = text

```
chr [1:8] "A" "A" "B" "B" "A" "A" "B" "B"
```

- integer (int) = integer numbers

```
int [1:10, 1:5] 41 18 25 12 13 23 42 10 21 2 …
```

- logical (logi) = TRUE/FALSE values (boolean data type)

```
logi [1:5] TRUE FALSE FALSE FALSE TRUE
```

Boolean are all two-step variables:
True-False
Yes-No
1-0

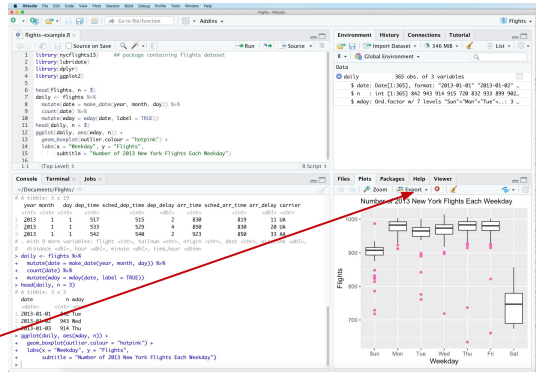
object class

atomic vectors, matrices, dataframes, lists & functions + special features

R object class: functions

`> install.packages()` is a R **function**.

R function are **ALWAYS** followed by brackets



R functions are grouped in **packages**.

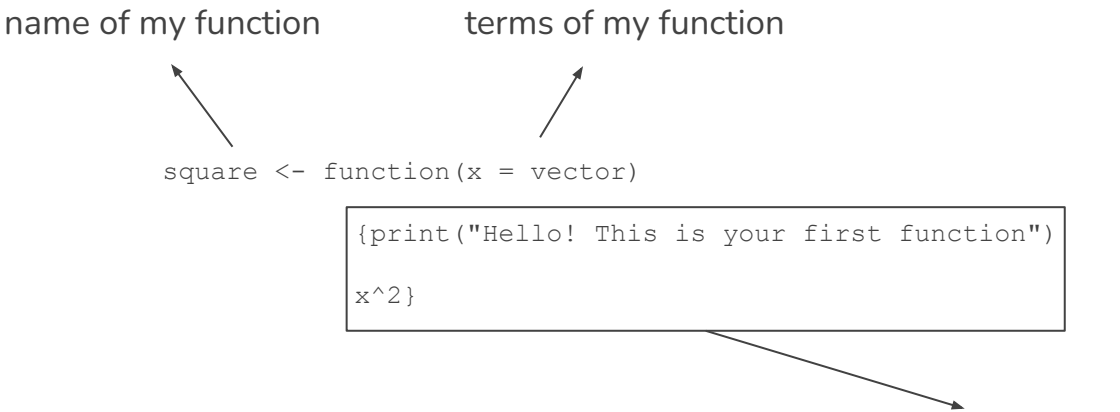
each function is more or less explained in a **help page:**

Usage

```
>factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)
```

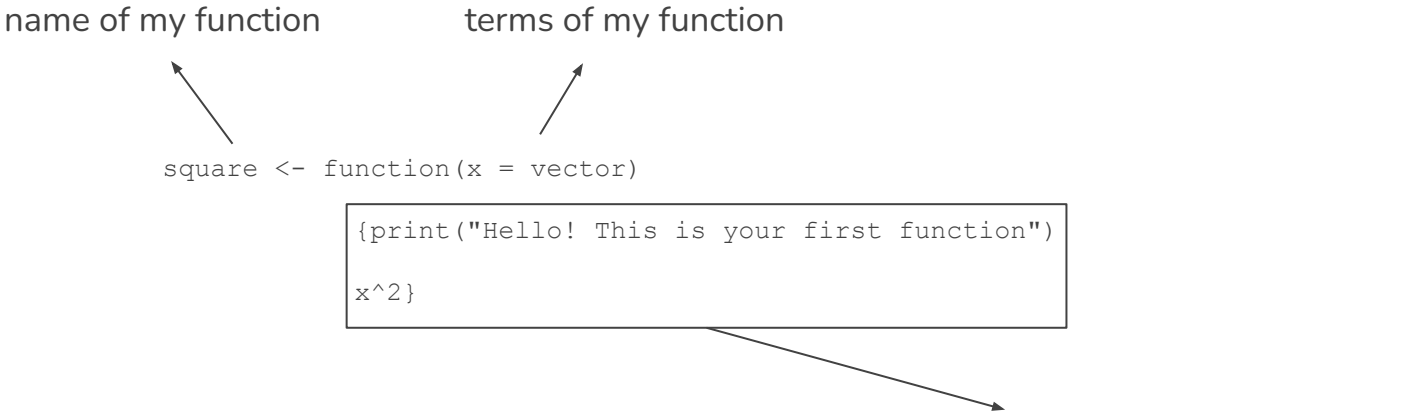
R objects class: functions

Everybody in R can write functions. Just a naive example to understand their grammar:



R objects class: functions

Everybody in R can write functions. Just a naive example to understand their grammar:



print() is a function to visualize something in the console

x^2 squares the x value/object

R objects class: functions

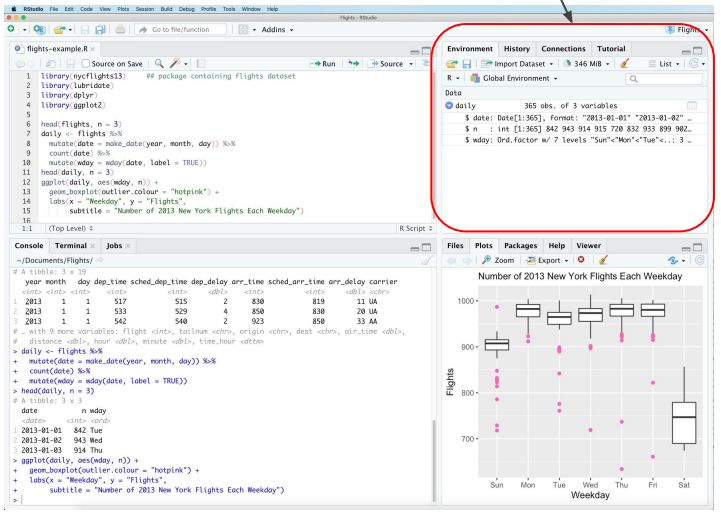
After the previous step you will have new object (a function in this case) in the Rstudio object list

so you can use it!

```
> square(12.567)

[1] "Hello! This is your first function"

[1] 157.9295
```



R objects class: functions

Most of the time you don’t need to write your own functions but to look for needed functions among available R packages.

```
Some shortcuts:

?factor                 # displays the function help page

args(factor)            # prints the arguments the function can take
```

objects class: vector

A sequence of elements (number, characters) assigned using the **c()** function

Vector can contain only one data type, otherwise R tries to convert the content to avoid loosing information.

simple vector

> my_vector <- c(“first_element”, “second_element”, number, ...)

named vector

> my_vector <- c(“name_1”=“first_element”, “name_2”=“second_element”, “name_3”=”second_element”, ...)

display vector elements name

> names(my_vector)

Exercise: write a numeric vector in a R object named “my_vector”

objects class: vector

```
# Write a simple vector:

> colors<-c("red", "blue", "yellow", "orange", "green", "brown")


# Check vector length with length() function

> length(colors)

[1] 6
```

objects class: vector

```
# type  of data in vector
> class(colors)
[1] “character”


# str() gives an overview of the structure of an object and its elements
> str(colors)
 chr [1:6] "red" "blue" "yellow" "orange" "green" "brown"
```

objects class: vector

```
# unique values in vectors                # when there are repeated values
> unique(colors)
[1] "red"    "blue"   "yellow" "orange" "green"  "brown"

# add elements to a vector
> c(colors, "purple")  # add to the end of the vector
[1] "red"    "blue"   "yellow" "orange" "green"  "brown"  "purple"

> c("white", colors)  # add to the beginning of the vector
[1] "white"  "red"    "blue"   "yellow" "orange" "green"  "brown"
```

objects class: vector

some shortcuts to work with vectors

```
> seq(1, 10, by=1)
```
sequence of numbers from x to y

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

```
> vector <- rep(c("A", "B"), 2, each=2)
```
repeat elements in vectors

```
[1] "A" "A" "B" "B" "A" "A" "B" "B"
```

```
> vector_1 <- sample(seq(1,30), 5)
```
generate vector with sampled random elements

```
[1] 22  8  7 18 28
```

objects class: vector

Exercise:

Produce:

- sequence of number (from 3500 to 3475 at 0.5 steps)
- repeated sequence of factors (repeat the words cat, tree and dog two times each and generate a vector of length 18
- vector of two random numbers between 0 and 1 (considering two numbers after the decimal point)

objects class: vector

Exercise:

- sequence of number (from 3500 to 3475 at 0.5 steps)

```
> seq(3500, 3475, by=0.5 )
```

- repeated sequence of factors (repeat the words cat, tree and dog two times each and generate a vector of length 18

```
> rep(c("cat", "tree", "dog"), 3, each=2 )
```

- vector of two random numbers between 0 and 1 (considering two numbers after the decimal point)

```
> sample(seq(0,1, by=0.01), 2)
```

subsetting vectors

To extract values from vectors positional indices should be provided in square brackets **[]**

```
> animals<-c("cat", "dog", "rabbit", "duck", "monkey", "fish")

> animals[2]

[1]  “dog”
```

subsetting vectors

multiple elements can be selected

```
> animals[c(3,2)]
```

```
[1] "rabbit" "dog"
```

conditional subsetting:

```
> numbers<-c(24, 5, 47, 6, 98, 2, 10, 144 )
> numbers[numbers>60]
[1]  98 144
```

operations with vectors

Sum two or more vectors:

```
> c(1,1,1,1)+c(1,1,1,1)
[1]  2 2 2 2
```

if vectors are not the same length, the shorter will be recycled

```
> c(1,2)+c(1,1,1,1)
[1]  2 3 2 3
```

Arithmetic operations on vectors applies element-wise

```
> c(1,1,1,1)*5
[1]  5 5 5 5
```

Exercise:

- write a vector in an object called “num” made of 10000 numbers comprised in the interval between 2 and 20000

- count how many values >3000
- **sum** elements number 385, 1001 and 7521

Solution

```
# generate num vector
> num<-seq(2, 20000, by=2)

# check vector length
> length(num)
[1] 10000

# subset and count elements
> length(num[num>3000])
> [1] 8500
```

sum elements number 385, 1001 and 7521

```
> num[385]+num[1001]+num[7521]
[1] 17814

> sum(num[c(385,1001,7521)])
[1] 17814
```

object class: matrices

```
> matrix(data = sample(seq(1,50)), nrow = 10, ncol = 5, byrow = F)
      [,1] [,2] [,3] [,4] [,5]
 [1,]    2   21   37   30   43
 [2,]   12   42    9    3   46
 [3,]    7   19   29   26   40
 [4,]   17   32   39   23   25
 [5,]   10   27   50   33   15
 [6,]    5   13    1   44    8
 [7,]   47   24    6   22   34
 [8,]   28   18   14   45   36
 [9,]   11   48   16    4   49
[10,]   41   31   38   20   35
```

Matrices are grouped NUMERIC vectors

object class: matrices

Matrices are grouped NUMERIC vectors, thus you can create matrices starting from several different vectors.

Exercise:

1. create 3 different vectors of length 45

- vector 1 should be created using **`seq`**
- vector 2 should be created using **`rep`** `# repeat 45, 45 times`
- vector 3 should be created using **`sample`** `# 45 random numbers between 1 and 100`

2. build your own matrix named “matrix” merging the vectors with the **`rbind()`** function

```
> matrix <- rbind(c(1,2),c(1,2),c(1,2))

     [,1] [,2]

[1,]    1    2

[2,]    1    2

[3,]    1    2
```

object class: matrices

1. create 3 different vectors of length 45
   - vector 1 should be created using **seq**

```
> vector_1<-seq(1,45)
```

   - vector 2 should be created using **rep**

```
# repeat 45, 45 times
> vector_2<-rep(45, 45)
```

   - vector 3 should be created using **sample**

```
# 45 random numbers between 1 and 100
> vector_3<-sample(seq(1,100), 45)
```

2. build your own matrix named “matrix” merging the vectors with the **rbind()** function

```
> my_matrix<-rbind(vector_1, vector_2, vector_3)
```

object class: dataframe

a dataframe is a group of any type of vectors (character, number, integer, factor)...dataframes are the most common object in R

Manually write a dataframe using data.frame() function

```
> df<-data.frame("colname_1"=rep(c("A", "B"), 2, each=2),        # vector 1
                 "colname_2"=sample(seq(1,30), 8),               # vector 2
                 "colname_3"=seq(8,11.5, by=0.5))                # vector 3
```

```
> df
  colname_1 colname_2 colname_3
1         A         9       8.0
2         A        27       8.5
3         B        13       9.0
4         B        30       9.5
5         A        18      10.0
6         A        29      10.5
7         B        21      11.0
8         B        10      11.5
```

object class: list

a list is a group of any type of objects (vector, matrix, data.frame)...function is list(), each object in the list can be identified by a name

```
> my_list<-list("vector"=vector, "matrix"=matrix, "dataframe"=df)

> my_list
$vector
[1] "A" "A" "B" "B" "A" "A" "B" "B"

$matrix
     A B
[1,] 1 2
[2,] 1 2
[3,] 1 2

$dataframe
  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
3         B        20       9.0
4         B        21       9.5
5         A         9      10.0
6         A        28      10.5
7         B         8      11.0
8         B        12      11.5
```

object class

each object can be duplicated (or overwritten!)

```
df2 <- df
```

```
my_list2 <- my_list
```

each object can be removed by **rm()**

```
rm(my_list2)
```

object class

Find the type of object

```
> class(df)
```

[1] "data.frame"

inspect object (find the data structure of a R object)

```
> str(df)
'data.frame':    8 obs. of  3 variables:
 $ colname_1: chr  "A" "A" "B" "B" ...
 $ colname_2: int  13 22 18 23 10 29 8 7
 $ colname_3: num  8 8.5 9 9.5 10 10.5 11 11.5
```

object class

Inspect dimensions

```
> dim(df)    # applies to data.frame and matrices
[1] 8 3
> length(vector) # applies to vectors only
[1] 8
```

object type: special objects

We are not here to teach specific packages, but keep in mind that several specialist package can use the “special objects” they are often lists built with specific parameters!

This is the case of

graphics objects (ggplot...)

APE

POPPR

PHYLOSEQ

DESeq

... and many other...

row- and column names

Each columns or rows in a given object (vector, matrix, dataframe) can have a name.

display column names: **colnames()**

```
> colnames(object)
```

display row names: **rownames()**

```
> rownames(object)
```

The same function can be used to change or set row/column names:

```
> colnames(object)<-c(“colname1”, “colname2”)
```

rows/columns indexing

Columns and rows can be flipped by transposition using the function **t()**

```
> matrix
     [,1] [,2]
[1,]    1    2
[2,]    1    2
[3,]    1    2
> t(matrix)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

rows/columns indexing

Often you need to select only some column/s or rows of a given object to perform operations:

object[rows, columns] **object[columns]**

select row 6 of the object df

```
> df[6,]

  colname_1 colname_2

6         A        12
```

select column 2 of the object df

```
> df[,2]

[1]  9  3 25  4 26 12 20 11
```

rows/columns indexing

Often you need to select only some column/s or rows of a given object to perform operations:

object[rows, columns]

- select from row 6 to row 8 of the object df

```
> df[6:8,]
  colname_1 colname_2
6         A         12
7         B         20
8         B         11
```

… same for columns

- the same way can be used to remove columns/rows using “-”

Remove row 2

```
> df[-2,]
  colname_1 colname_2
1         A          9
3         B         25
4         B          4
5         A         26
6         A         12
7         B         20
8         B         11
```

rows/columns indexing

Often you need to select only some column/s or rows of a given object to perform operations:

object[rows, columns]

non-contiguous columns/rows interval must be provided as a vector!

```
> df[,c(1,3)]
```

```
  colname_1 colname_3
1         A       8.0
2         A       8.5
3         B       9.0
4         B       9.5
5         A      10.0
6         A      10.5
7         B      11.0
8         B      11.5
```

rows/columns indexing (dataframes)

Columns in dataframes objects can be selected by their names or created using “$” symbol

```
> df$colname_1

[1] "A" "A" "B" "B" "A" "A" "B" "B"
```

remove columns in dataframes with $

```
> df2$colname_2 <- NULL
> df2
  colname 1 colname 3
1         A       8.0
2         A       8.5
3         B       9.0
4         B       9.5
5         A      10.0
6         A      10.5
7         B      11.0
8         B      11.5
```

rows/columns indexing (dataframes)

Indexes can be used with logical operators to subset values

```
> df[df$colname_1=="B",]

  colname_1 colname_2 colname_3
3         B        18       9.0
4         B        23       9.5
7         B         8      11.0
8         B         7      11.5

> df[df$colname_1=="B" & df$colname_2<10,]

  colname_1 colname_2 colname_3
7         B         8      11.0
8         B         7      11.5
```

rows/columns/object indexing (lists)

select element in list

```
> my_list[[1]]  # positional selection (select the first element)

[1] "A" "A" "B" "B" "A" "A" "B" "B"
```

```
> my_list[["df"]] # selection byname
  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
3         B        20       9.0
4         B        21       9.5
5         A         9      10.0
6         A        28      10.5
7         B         8      11.0
8         B        12      11.5
```

rows/columns/object indexing (lists)

select column in a dataframe included in a list

```
> my_list[["df"]]$colname_1
[1] "A" "A" "B" "B" "A" "A" "B" "B"
```

select rows (or columns) in element included in a list

```
> my_list[["df"]][1:5,]
  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
3         B        20       9.0
4         B        21       9.5
5         A         9      10.0
```

filter and subsetting

Filtering and subsetting are the empowered version of row/columns selection that we explained before, but they use specific functions in specific packages such as:

subset()

dplyr::filter()

they are based on logics. The grammar for logics in R is simple:

== for equal (not “=”

!= for different

> < major an minor (followed by = to include the limit value)

| for OR (this is the “pipe” character usually is before the 1 in keyboards)

& or **&&** for AND

! for NOT

filter and subsetting

subset() and dplyr::**filter**() are similar and useful to select certain values in a dataframe

```
>library(dplyr)

> filter(df, colname_1 == "A")

  colname_1 colname_2 colname_3
1         A        13       8.0
2         A         5       8.5
5         A         9      10.0
6         A        28      10.5
```

subsetting by factor

```
> filter(df, colname_2 < 13)

  colname_1 colname_2 colname_3
2         A         5       8.5
5         A         9      10.0
7         B         8      11.0
8         B        12      11.5
```

subsetting by number

filter and subsetting

```
>filter(df, colname_1 %in% c("B", "C"))          # reads like: “is contained in..”

colname_1 colname_2 colname_3
1         B        18       9.0
2         B        23       9.5
3         B         8      11.0
4         B         7      11.5

>filter(df, !colname_1 %in% c("B", "C"))         # reads like: “is NOT contained in..”

colname_1 colname_2 colname_3
1         A        13       8.0
2         A        22       8.5
3         A        10      10.0
4         A        29      10.5
```

Exercise

Create a dataframe from the following vectors:

```
> n <- 1:10
> sex <- rep(c('male', 'female'), 5)
> age <- c(23, 22, 21, 22, 24, 30, 23, 29, 19, 29)
> weight <- c(72, 90, 120, 80, 75, 65, 91, 58, 78, 50)
> height <- c(171, 185, 210, 170, 189, 150, 168, 165, 188, 143)
```

Question 1: Which is the mean age? [use mean() function]

Question 2: Which is the minimum male weight? [use function min()]

Exercise: solution

```
> people<- data.frame(n = n, sex = sex, age = age, weight =
weight, height = height)
```

Question 1: Which is the mean age? [use mean() function]

```
> mean(people$age)
[1] 24.2
```

Question 2: Which is the minimum male weight?

```
> min(filter(people, sex=="male")$weight)
[1] 72
```

```
> min(people[people$sex == 'male',]$weight)
[1] 72
```

NOTE: in R problems can have multiple solutions. The best one is the shortest one (less code lines)

Save your session

- *.R files # your main script (often very light text file data)
- *.RData files # your environment (all the objects you saved, often very big files (>1Gb)

- You can easily save *.R and *.RData files through the RStudio interface

- you can also manage your environments through the R command line (save you some time)

```
# save your environment
> save.image(file = "your_path/your_envirnoment_name.RData")
```

Save your session

```
# load a *.Data file in your current session
> load(file = "your_path/your_envirnoment_name.RData")
# make searchable a stored environment without loading it
> attach(file = "your_path/your_envirnoment_name.RData")
```

Note: don’t forget to **detach()** your environment!