R language: a tour

2025-01-28

M1 MIDS/MFA/LOGOS Université Paris Cité Année 2024 Course Homepage Moodle



# ! Objectives

This workbook intends to walk you through basic aspects of the R language and programming environment.

#### **Packages**

Base R can do a lot. But the full power of R comes from a fast growing collection of packages.

Packages are first *installed* (that is downloaded from **cran** and copied somewhere on the hard drive), and if needed, *loaded* during a session.

- Installation can usually be performed using command install.packages(). In some circumstances, ad hoc installation commands (often from packages devtools) are needed
- Package pak offers an interesting alternative to base R install.packages()
- Once a package has been installed/downloaded on your hard drive
  - if you want all objects exported by the package to be available in your session, you should load the package, using library() or require() (what's the difference?).
     Technically, this loads the NameSpace defined by the package.
  - if you just want to pick some objects exported from the package, you can use qualified names like package\_name::object\_name to access the object (function, dataset, ...).

For example, when we write

#### gapminder <- gapminder::gapminder</pre>

we assign dataframe/tibble gapminder from package gapminder to identifier "gapminder" in global environment .

Function p\_load() from pacman (package manager) blends installation and loading: if the package named in the argument of p\_load() is not installed (not among the installed.packages()), p\_load() attempts to install the package. If installation is successful, the package is loaded.

```
if (! require(pak)){
    install.packages("pak")
}

stopifnot(
    require("tidyverse"),
    require("lobstr"),
    require("ggforce"),
    require("mycflights13"),
    require("patchwork"),
    require("viridis"),
    require("WMASS"),
    require("gapminder"),
    require("gryr"),
    require("pak")
)
```

#### Optional arguments

A very nice feature of R is that functions from base R as well as from packages have *optional* arguments with sensible *default* values. Look for example at documentation of require() using expression ?require.

Optional settings may concern individual functions or the collection of functions exported by some packages. In the next *chunk*, we reset the default color scales used by graphical functions from ggplot2.

```
opts <- options() # save old options

options(ggplot2.discrete.colour="viridis")
options(ggplot2.continuous.colour="viridis")</pre>
```

You shall not confuse installing (on your hard-drive) and loading (in session) a package.

#### i Question for Pythonistas

- In **?** what is the analogue of install.packages()?
- In **\rightarrow** what is the analogue of require()/library()?

# Numerical (atomic) vectors

Numerical (atomic) vectors form the most primitive type of R.

#### Vector creation and assignment

The next three lines create three numerical atomic vectors.

In IDE Rstudio, have a look at the environment pane on the right before running the chunk, and after.

Use ls() to investigate the *environment* before and after the execution of the three assignments.

```
1 ls()
2 x <- c(1, 2, 12)
3 y <- 5:7
4 z <- 10:1
5 x; y; z
6 ls()</pre>
```

#### i Question

- What are the identifiers known in the global environment before execution of lines 2-4?
- What are the identifiers known in the global environment after execution of lines 2-4?
- Which objects are attached to identifiers x, y, and z?

#### i Question

What does the next chunk?

```
ls()
w <- y
ls()
```

#### i Question

- Is the content of object denoted by y copied to a new object bound to w?
- Interpret the result of w == y.
- Interpret the result of identical(w,y) (use help("identical") if needed).

```
w == y
identical(w,y)
```

# Indexation, slicing, modification

Slicing a vector can be done in two ways:

- providing a vector of indices to be selected. Indices need not be consecutive.
- providing a Boolean mask, that is a logical vector to select a set of positions.

```
x \leftarrow c(1, 2, 12); y \leftarrow 5:7; z \leftarrow 10:1
```

# i Question

Explain the next lines

```
z[1]  # slice of length 1
z[0]  # What did you expect?
z[x]  # slice of length ??? index error ?
z[y]
z[x %% 2]  # what happens with x[0] ?
z[0 == (x %% 2)]  # masking
z[c(2, 1, 1)]
```

If the length of mask and and the length of the sliced vector do not coincide, what happens?

#### .

A scalar is just a vector of length 1!

```
class(z)
[1] "integer"

class(z[1])
[1] "integer"

class(z[c(2,1)])
[1] "integer"
```

#### Question

Explain the next lines

```
y[2:3] <- z[2:3]
y == z[-10]
z[-11]
```

# i Question

Explain the next line

```
z[-(1:5)]
```

#### Question

How would you select the last element from a vector (say z)?

#### i Question

Reverse the entries of a vector. Find two ways to do that.

In statistics, machine learning, we are often faced with the task of building grids of regularly spaced elements (these elements can be numeric or not). R offers a collection of tools to perform this. The most basic tool is rep().

#### Question

- Repeat a vector 2 times
- Repeat each element of a vector twice

Let us remove objects from the global environment.

```
rm(w, x, y, z)
```

#### Numbers

So far, we told about numeric vectors. Numeric vectors are vectors of floating point numbers. R distinguishes several kinds of numbers.

- Integers
- Floating point numbers (double)

To check whether a vector is made of numeric or of integer, use is.numeric() or is.integer(). Use as.integer, as.numeric() to enforce type conversion.

```
Explain the outcome of the next chunks

class(113L); class(113); class(113L + 113); class(2 * 113L); class(pi); as.intege

[1] "integer"
[1] "numeric"
[1] "numeric"
[1] "numeric"
[1] 3

floor(pi); class(floor(pi)) # mind the floor

[1] 3
[1] "numeric"
```

#### Integer arithmetic

```
29L * 31L ; 899L %/% 32L ; 899L %% 30L

[1] 899

[1] 28

[1] 29
```

```
R integers are not the natural numbers from Mathematics
R numerics are not the real numbers from Mathematics
.Machine$double.eps
[1] 2.220446e-16
.Machine$double.xmax
[1] 1.797693e+308
.Machine$sizeof.longlong
[1] 8
u <- double(19L)
v <- numeric(5L)</pre>
w <- integer(7L)
lapply(list(u, v, w), typeof)
[[1]]
[1] "double"
[[2]]
[1] "double"
[[3]]
[1] "integer"
length(c(u, v, w))
[1] 31
typeof(c(u, v, w))
[1] "double"
```

R is (sometimes) able to make sensible use of Infinite.

```
log(0)

[1] -Inf
log(Inf)

[1] Inf
1/0

[1] Inf
0/0

[1] NaN
max(c( 0/0,1,10))

[1] NaN
max(c(NA,1,10))

[1] NA
max(c(-Inf,1,10))

[1] 10
```

```
is.finite(c(-Inf,1,10))
[1] FALSE TRUE TRUE
is.na(c(NA,1,10))
[1] TRUE FALSE FALSE
is.nan(c(NaN,1,10))
[1] TRUE FALSE FALSE
Computing with vectors
Summing, scalar multiplication
x < -1:3
y <- 9:7
sum(x); prod(x)
[1] 6
[1] 6
z <- cumsum(1:3)
w <- cumprod(3:5)
x + y
[1] 10 10 10
x + z
[1] 2 5 9
2 * w
[1] 6 24 120
2 + w
[1] 5 14 62
w / 2
[1] 1.5 6.0 30.0
    Question
     How would you compute a factorial?
  i Question
    Approximate \sum_{n=1}^{\infty} 1/n^2 within 10^{-3}?
```

How would you compute the inner product between two (atomic numeric) vectors?

- What we have called vectors so far are indeed atomic vectors.
  - Read Chapter on Vectors in R advanced Programming
  - Keep an eye on package vctrs for getting insights into the R vectors.

#### Numerical matrices

R offers a matrix class.

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

[1] "matrix" "array"

# i Question

From the evaluation of the preceding chunk, can you guess whether it is easier the traverse a matrix in row-first order or in column-first order?

#### Creation, transposition and reshaping

A vector can be turned into a column matrix.

```
v <- as.matrix(1:5)</pre>
      [,1]
[1,]
[2,]
          2
[3,]
          3
[4,]
          4
[5,]
          5
```

A matrix can be transposed

[1,]

[2,]

1

1

1

1

```
t(v) # transpose
     [,1] [,2] [,3] [,4] [,5]
[1,]
             2
                  3
                       4
cat(dim(v), ' ', dim(t(v)), ' n')
5 1
      1 5
A <- matrix(1, nrow=5, ncol=2); A
     [,1] [,2]
```

[3,] 1 1 [4,] 1 1 [5,] 1 1

#### Question

lobstr::mem\_used() allows us to keep track of the amount of memory used by
our R session. lobstr::obj\_size() tells us the amount of memory used by the
representation of an object.

Comment the next chunk

```
m1 <-lobstr::mem_used()
A <- matrix(rnorm(100000L), nrow=1000L)
m2 <- lobstr::mem_used()
lobstr::obj_size(A)

800.22 kB
B <- t(A)
lobstr::obj_size(B)

800.22 kB
m3 <- lobstr::mem_used()
m2-m1; m3-m2

802.65 kB
1.09 MB</pre>
```

#### i Question

- Is there a difference between the next two assignments?
- How would you assign value to all entries of a matrix?

```
A <- matrix(rnorm(16), nrow=4)
A[] \leftarrow 0 ; A
     [,1] [,2] [,3] [,4]
[1,]
        0
             0
                  0
[2,]
        0
             0
                  0
                       0
[3,]
             0
                  0
                       0
        0
                       0
[4,] 0
             0
                  0
A < -0; A
[1] 0
```

#### i Question

What is the final shape of A?

```
A <- matrix(1, nrow=5, ncol=2)
A
A[] <- 1:15
A
```

We can easily generate diagonal matrices and constant matrices.

```
diag(1, 3) # building identity matrix
```

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

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

```
matrix(0, 3, 3) # building null matrix
```

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

Is there any difference between the next two assignments?

```
B <- A[]
B ; A
Γ17 0
[1] 0
lobstr::obj_addr(B) ; lobstr::obj_addr(A)
[1] "0x5f08e2236e58"
[1] "0x5f08e2a7c718"
B <- A
lobstr::obj_addr(B) ; lobstr::obj_addr(A)
[1] "0x5f08e2a7c718"
[1] "0x5f08e2a7c718"
```

#### Indexation, slicing, modification

Indexation consists in getting one item from a vector/list/matrix/array/dataframe.

Slicing and subsetting consists in picking a substructure:

- subsetting a vector returns a vector
- subsetting a list returns a list
- subsetting a matrix/array returns a matrix/array (beware of implicit simplifications and dimension dropping)
- subsetting a dataframe returns a dataframe or a vector (again, beware of implicit simplifications).

# Question Explain the next results

```
A <- matrix(1, nrow=5, ncol=2)
dim(A[sample(5, 3), -1])
dim(A[sample(5, 3), 1])
length(A[sample(5, 3), 1])
is.vector(A[sample(5, 3), 1])
A[10:15]
A[60]
dim(A[])
```

How would you create a fresh copy of a matrix?

#### Computing with matrices

\* versus %\*% %\*% stands for matrix multiplication. In order to use it, the two matrices should have conformant dimensions.

[1,] -7.476049 21.18286

There are a variety of reasonable products around. Some of them are available in R.

#### i Question

How would you compute the Hilbert-Schmidt inner product between two matrices?

$$\langle A, B \rangle_{\mathrm{HS}} = \mathrm{Trace}(A \times B^{\top})$$

# i Question

How can you invert a square (invertible) matrix?

# Logicals

- R has constants TRUE and FALSE.
- Numbers can be coerced to logicals.

#### Question

- Which numbers are truthies? falsies?
- What is the value (if any) of ! pi & TRUE?
- What is the meaning of all()?
- What is the meaning of any()?
- Recall De Morgan's laws. Check them with R.
- Is | denoting an inclusive or an exclusive OR?

#### Handling three-valued logic

```
TRUE & (1> (0/0))
(1> (0/0)) | TRUE
(1> (0/0)) | FALSE

TRUE || (1> (0/0))

TRUE | (1> (0/0))

TRUE || stopifnot(4<3)

# TRUE | stopifnot(4<3)

# FALSE && stopifnot(4<3)

# FALSE & stopifnot(4<3)
```

#### i Question

What is the difference between logical operators | | and | |?

Remark: favor &, | over &&, | |.

## all and any

Look at the definition of all and any.

#### i Question

- How would you check that a square matrix is symmetric?
- How would you check that a matrix is diagonal?

#### Lists

While an instance of an atomic vector contains objects of the same type/class, an instance of list may contain objects of widely different types.

#### Question

Check an explain the output of the next chunk

```
check an explain the output of the next chunk

p <- c(2, 7, 8)
q <- c("A", "B", "C")
x <- list(p, q)
x[2]
x
length(x)
rlang::is_vector(x)
rlang::is_atomic(x)
y <- c(p, q)
y
length(y)
rlang::is_atomic(y)
rlang::is_list(y)</pre>
```

- How would you build a list made of p, q, and x?
- What is x[2] made of?
- How does it compare with x[[2]]?

#### i Question

Read and understand the next expressions.

```
is_atomic(p); is_atomic(p[2]); is_atomic(p[[2]])

is_list(q); is_atomic(q)

is_list(x); is_atomic(x); class(x)

class(x[2]); class(x[[2]])
length(x[2]); length(x[[2]])

identical(q, x[[2]]); identical(q, x[2])

obj_addr(q); obj_addr(x[[2]]); obj_addr(x[2])

ref(x)
obj_addrs(x)
identical(x[2],x[[2]])
```

Functions is\_atomic(), is\_list(), ..., obj\_addr() are from packages rlang and lobstr. See https://rlang.r-lib.org and https://lobstr.r-lib.org

#### i Question

How would you replace "A" in x with "K"?

Read Chapter on Lists in R advanced Programming

# Lookup tables (aka dictionaries) using named vectors

A lookup table maps strings to values. It can be implemented using named vectors. If we want to map: "seine" to "75", "loire" to "42", "rhone" to "69", "savoie" to "73" we can proceed in the following way:

```
codes <- c(75L, 42L, 69L, 73L)
names(codes) <- c("seine", "loire", "rhône", "savoie")

codes["rhône"]; codes["aube"]

rhône
69</pre>
```

<NA>

What is the class of codes?

#### i Question

Capitalize the names used by codes

•

Package stringr offers a function str\_to\_title() that could be of interest.

#### **Factors**

Factors exist in Base R. They play a very important role. Qualitative/Categorical variables are implemented as Factors.

Meta-package tidyverse offers a package dedicated to factor engineering: forcats.

```
yraw <- c("g1","g1","g2","g2","g3")
print(yraw)</pre>
```

```
[1] "g1" "g1" "g2" "g2" "g2" "g3"
```

summary(yraw)

Length Class Mode 6 character character

```
is.vector(yraw) ; is.atomic(yraw)
```

- [1] TRUE
- [1] TRUE

# i Question

yraw takes few values. It makes sense to make it a factor. How does it change the behavior of *generic* function summary?

Load the (celebrated) iris dataset, and inspect variable Species

```
data(iris)
species <- iris$Species
levels(species)</pre>
```

```
[1] "setosa" "versicolor" "virginica" summary(species)
```

```
setosa versicolor virginica
50 50 50
```

We may want to collapse virginica and versicolor into a single level called versinica

forcats offer a function fct\_collapse().

Factors are used to represent *categorical* variables.

# Question

- Load the whiteside data from package MASS.
- Have a glimpse.
- Assign column Insul to y

#### i Question

- What is the class of y?
- Is y a vector
- Is y ordered? What does ordered mean here?
- What are the levels of y? How many levels has y?
- Can you slice y?
- What are the binary representations of the different levels of y?

#### Question

Summarize factor y

#### Factors nuts and bolts

When coercing a vector (integer, character, ...) to a factor, use forcats::as\_factor() rather than base R as.factor().

💡 🖝 Useful function to make nice barplots when constructing barplots.

Recall that when you want to display counts for a univariate categorical sample, you use a barplot. It is often desirable to rank the levels according to the displayed statistics (usually a count).

This can be done in a seamless way using functions like forcats::fct\_infreq().

```
forcats::fct_count(y, prop = TRUE)
```

```
# A tibble: 2 x 3
  f
             n
  <fct> <int> <dbl>
1 Before
            26 0.464
2 After
            30 0.536
z <- sample(y, length(y), replace = TRUE) # permutation of whiteside$Insul
sort(forcats::fct_infreq(z))
                                   # first level is most frequent one
```

- [1] Before Before Before Before Before Before Before Before Before
- [11] Before Before Before Before Before Before Before Before Before

```
[21] Before [31] Before Before Before After After
```

```
forcats::fct_count(z)
```

Make z ordered with level After preceding Before. Does ordering impact the behavior of forcats::fct\_count()?

Read Chapter on Factors in R for Data Science

# Dataframes, tibbles and data.tables

A dataframe is a list of vectors with equal lengths. This is the way R represents and manipulates multivariate samples.

Any software geared at data science supports some kind of dataframe

- Python Pandas
- Python Dask
- Spark
- ..

The iris dataset is the "Hello world!" of dataframes.

```
data(iris)
iris |>
  glimpse()
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ Sepal.Width <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.5, 1.~
$ Petal.Width <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ Species <fct> setosa, setosa
```

A matrix can be transformed into a data.frame

```
A <- matrix(rnorm(10), ncol=2)
data.frame(A)
```

```
X1 X2
1 0.2029087 1.68377132
2 0.2521046 0.29735485
```

```
3 0.3384742 -0.03710351
4 0.3303436 -0.66581150
5 0.2253168 1.43674703
```

There are several flavors of dataframes in R: tibble and data.table are modern variants of data.frame.

```
t <- tibble::tibble(
  x=1:3,
  a=letters[11:13],
  d=Sys.Date() + 1:3)
head(t)</pre>
```

#### glimpse(t)

```
Rows: 3
Columns: 3
$ x <int> 1, 2, 3
$ a <chr> "k", "l", "m"
$ d <date> 2025-01-29, 2025-01-30, 2025-01-31
```

#### ref(t)

```
[1:0x5f08e2315ae8] <tibble[,3]>
x = [2:0x5f08e0b8a4c8] <int>
a = [3:0x5f08e23a4258] <chr>
d = [4:0x5f08e235f198] <date>
```

! Read Chapter on data frames and tibbles in Advanced R

#### i Question

Perform a random permutation of the columns of a data.frame/tibble.

 $\P$  Function sample() from base R is very convenient

#### nycflights data

Wrestling with tables is part of the data scientist job. Out of the box data are often messy. In order to perform useful data analysis, we need tidy data. The notion of tidy data was elaborated during the last decade by experienced data scientists.

You may benefit from looking at the following online documents.

#### Tidy data in R for Data Science

Introduction to Table manipulation in R for Data Science in R.

More data of that kind is available following guidelines from https://github.com/hadley/nycflights13

In this exercise, you are advised to use functions from dplyr.

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges.

data <- nycflights13::flights

#### Question

- Have a glimpse at the data.
- What is the class of object data?
- What kind of object is data?

Hint: use class(), is.data.frame() tibble::is\_tibble()

#### i Question

Extract the name and the type of each column.

#### Compute the mean of the numerical columns

Base R has plenty of functions that perform statistical computations on univariate samples. Look at the documentation of mean (just type ?mean). For a while, leave aside the optional arguments.

In database parlance, we are performing aggregation

```
mean(data$dep_delay)
```

#### [1] NA

```
# mean(data[["dep_delay"]])
```

If we want the mean of all numerical columns, we need to project the data frame on numerical columns.

A verb of the summarize family can be useful.



Have a look at across in latest versions of dplyr() Use across() from dplyr 1.x. See Documentation

If applied to a data.frame, summary(), produces a summary of each column. The summary depends on the column type. The output of summary is a shortened version the list of outputs obtained from applying summary to each column (lapply(data, summary)).

```
data |>
  summary()
```

year	month	day	dep_time	sched_dep_time
Min. :2013	Min. : 1.000	Min. : 1.00	Min. : 1	Min. : 106
1st Qu.:2013	1st Qu.: 4.000	1st Qu.: 8.00	1st Qu.: 907	1st Qu.: 906
Median :2013	Median : 7.000	Median :16.00	Median :1401	Median :1359
Mean :2013	Mean : 6.549	Mean :15.71	Mean :1349	Mean :1344
3rd Qu.:2013	3rd Qu.:10.000	3rd Qu.:23.00	3rd Qu.:1744	3rd Qu.:1729
Max. :2013	Max. :12.000	Max. :31.00	Max. :2400	Max. :2359

```
NA's
                                                         :8255
  dep_delay
                      arr_time
                                  sched_arr_time
                                                    arr_delay
                                                         : -86.000
Min.
       : -43.00
                         : 1
                                  Min.
                                         : 1
                                                  Min.
                  Min.
1st Qu.:
          -5.00
                   1st Qu.:1104
                                  1st Qu.:1124
                                                  1st Qu.: -17.000
Median :
          -2.00
                  Median:1535
                                  Median:1556
                                                  Median :
                                                            -5.000
Mean
       : 12.64
                          :1502
                                  Mean
                                                             6.895
                  Mean
                                          :1536
                                                  Mean
3rd Qu.: 11.00
                   3rd Qu.:1940
                                  3rd Qu.:1945
                                                  3rd Qu.: 14.000
       :1301.00
                                          :2359
                                                         :1272.000
Max.
                  Max.
                          :2400
                                  Max.
                                                  Max.
NA's
       :8255
                  NA's
                          :8713
                                                  NA's
                                                         :9430
  carrier
                        flight
                                     tailnum
                                                          origin
Length: 336776
                   Min.
                                   Length:336776
                                                       Length: 336776
                           :
Class : character
                    1st Qu.: 553
                                   Class :character
                                                       Class : character
Mode :character
                   Median:1496
                                   Mode :character
                                                       Mode :character
                   Mean
                           :1972
                    3rd Qu.:3465
                   Max.
                           :8500
    dest
                       air_time
                                       distance
                                                         hour
Length: 336776
                           : 20.0
                                           : 17
                                                           : 1.00
                   Min.
                                    Min.
                                                    Min.
Class : character
                   1st Qu.: 82.0
                                    1st Qu.: 502
                                                    1st Qu.: 9.00
Mode :character
                   Median :129.0
                                    Median: 872
                                                    Median :13.00
                   Mean
                           :150.7
                                    Mean
                                            :1040
                                                    Mean
                                                           :13.18
                   3rd Qu.:192.0
                                    3rd Qu.:1389
                                                    3rd Qu.:17.00
                   Max.
                           :695.0
                                    Max.
                                            :4983
                                                    Max.
                                                           :23.00
                   NA's
                           :9430
    minute
                  time_hour
       : 0.00
                        :2013-01-01 05:00:00.00
Min.
                Min.
1st Qu.: 8.00
                1st Qu.:2013-04-04 13:00:00.00
Median :29.00
                Median :2013-07-03 10:00:00.00
       :26.23
                        :2013-07-03 05:22:54.64
Mean
                Mean
                3rd Qu.:2013-10-01 07:00:00.00
3rd Qu.:44.00
                        :2013-12-31 23:00:00.00
Max.
       :59.00
                Max.
```

#### Handling NAs

We add now a few NAs to the data....

```
data2 <- data
data2$arr_time[1:10] <- NA</pre>
```

Houston, we have a problem!

#### i Question

How should we compute the column means now?

It is time to look at optional arguments of function mean.

#### Question

Decide to ignore NA and to compute the mean with the available data

It is possible to remove all rows that contain at least one NA. Show this leads to a different result.

#### Question

Compute the minimum, the median, the mean and the maximum of numerical columns

#### i Question

Obtain a nicer output!

Check with https://dplyr.tidyverse.org/reference/scoped.html?q=funs#arguments

# i Question

Mimic summary on numeric columns

#### i Question

Compute a new itinerary column concatenating the origin and dest one. Have a look at Section Operate on a selection of variables

#### i Question

Compute the coefficient of variation (ratio between the standard deviation and the mean) for each itinerary. Can you find several ways?

#### i Question

Compute for each flight the ratio between the distance and the air\_time in different ways and compare the execution time (use Sys.time()).

#### Question

Which carrier suffers the most delay?

#### Puzzle

```
year <- 2012L

data |>
  dplyr::select(year, dest, origin) |>
  head()
```

```
3
  2013 MIA
              JFK
4 2013 BQN
              JFK
5 2013 ATL
              LGA
6 2013 ORD
              EWR
data |>
  dplyr::filter(year==year) |>
  dplyr::summarize(n())
# A tibble: 1 x 1
   `n()`
   <int>
1 336776
data |>
  dplyr::filter(year==2012L) |>
  dplyr::summarize(n())
# A tibble: 1 x 1
  `n()`
  <int>
1
      0
data |>
  dplyr::filter(year==.env$year) |>
  dplyr::summarize(n())
# A tibble: 1 x 1
  `n()`
  <int>
1
      0
data |>
  dplyr::filter(year==.data$year) |>
  dplyr::summarize(n())
# A tibble: 1 x 1
   `n()`
   <int>
1 336776
  Question
    Can you explain what happens?
```

#### Flow control

R offers the usual flow control constructs:

- branching/alternative if (...) {...} else {...}
- iterations (while/for) while (...) {...} for (it in iterable) {...}
- function calling callable(...) (how do we pass arguments? how do we rely on defaults?)

# If () then {} else {}

If expressions  $yes_expr$  and  $no_expr$  are complicated it makes sense to use the if (...)  $\{...\}$  else  $\{...\}$  construct

There is also a conditional statement with an optional else {}

```
#| label: if-else
#| eval: false
#| collapse: false
if (condition) {
    ...
} else {
    ...
}
```

#### i Question

Is there an elif construct in R?

r R also offers a switch

```
#| label: switch
switch (object,
   case1 = {action1},
   case2 = {action2},
   ...
)
```

```
i There exists a selection function ifelse(test, yes_expr, no_expr).
ifelse(test, yes, no)
Note that ifelse(...) is vectorized.

x <- 1L:6L
y <- rep("odd", 6)
z <- rep("even", 6)

ifelse(x %% 2L, y, z)
[1] "odd" "even" "odd" "even" "odd" "even"

This is a vectorized function</pre>
```

#### Iterations for (it in iterable) {...}

Have a look at Iteration section in R for Data Science

#### i Question

Create a lower triangular matrix which represents the 5 first lines of the Pascal triangle.

Recall

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Locate the smallest element in a numerical vector

## While (condition) {...}

#### i Question

Find the location of the minimum in a vector v

#### i Question

Write a loop that checks whether vector v is non-decreasing.

#### **Functions**

To define a function, whether named or not, you can use the function constructor.

```
foo <- function(arg1, arg2=def2) {
    # body
}</pre>
```

#### i Question

Write a function that checks whether vector **v** is non-decreasing.

# Question

Write a function with integer parameter n, that returns the Pascal Triangle with n+1 rows.

#### Question

How would you generate a Fibonacci sequence of length n?

Recall the Fibonacci sequence is defined by

$$F_{n+2} = F_{n+1} + F_n$$
  $F_1 = F_2 = 1$ 

#### Question

Write a function that perform binary search in a non-decreasing vector.

# ! 4

Read Chapter on functions in Advanced R

In R, argument evaluation is surprising, powerful but taming argument evaluation is real work.

# Functional programming

In R, functions are first class entities, they can be defined at run-time, they can be used as function arguments. You can define list of functions, and iterate over them.

Try to use https://purrr.tidyverse.org.

# ! Anonymous functions

```
\(x) body
is a shorthand for
function (x) {
  body
}
```

#### Operators purrr::map\_???

#### i Question

```
Write truth tables for &, |, &&, ||, ! and xor 
Hint: use purrr::map, function outer()
```

# i Question

Write a function that takes as input a square matrix and returns TRUE if it is lower triangular.

#### i Question

Use map, choose and proper use of pronouns to deliver the n first lines of the Pascal triangle using one line of code.

As far as the total number of operations is concerned, would you recommend this way of computing the Pascal triangle?

Read Chapter on Functional Programming in Advanced R

# Further exploration

This notebook walked you through some aspects of R and its packages. We just saw the tip of the iceberg.

We barely mentioned:

- (Non-standard) Lazy evaluation
- Different flavors of object oriented programming
- Connection with C++: RCpp
- Connection with databases: dbplyr
- Building modeling pipelines: tidymodels
- Concurrency
- Building packages
- Building interactive Apps: Shiny

- Attributes (metadata)
- Formulae formula
- Strings stringi, stringr
- Dates lubridate
- and plenty other things ....

#### References

- https://www.statmethods.net/index.html
- https://www.datacamp.com/courses/free-introduction-to-r
- dplyr videos
- ggplot2 video tutorial
- cheatsheets

A Readers who really want to learn R should spend time on

- R for Data Science by Wickham, Çetinkaya-Rundel, and Grolemund.
- Advanced R 2nd Edition by Wickham
- Advanced R Solutions by Grosser and Bumann
- Hands-On Programming with R by Grolemund

Don't go without Base R cheatsheet