Introduction to R

Valentina Zangirolami

University of Milano-Bicocca
Department of Economics, Management and Statistics

May 12, 2025

Contact: `valentina.zangirolami@unimib.it`

R resources

- To install R: `https://cran.r-project.org/`
- To install R studio (R is required): `https://posit.co/downloads/`
- Documentation: `https://cran.r-project.org/manuals.html`
- Cheat sheets: `https://support--rstudio-com.netlify.app/resources/cheatsheets/`
- Book:
  - *Venables, W. N., Smith D. M. & the R Core Team (2021).* An Introduction to R.
  - Efficient R programming: `https://csgillespie.github.io/efficientR/index.html`
- To find specific topics: `https://www.bigbookofr.com/`

Basic commands - Help

- help.start(): shows the main links for programming in R
- help(): to obtain the description of a specific function (e.g. help(mean) or ?mean shows the description of the function "mean")
- help.search(): to search a string on function's documentation

Basic commands - Workspace

- getwd(): to visualize the working directory
- setwd("mydirectory"): to set a new directory
- ls(): to visualize all the objects saved in R space
- rm(myobject): to remove myobject from R space
- rm(list=ls()): to remove all files from R space

Basic commands - Libraries

- library(): list of packages available
- library(name package): to load a package
- install.packages("name package"): to install a package
- update.packages(ask=FALSE): to update all packages
- data(package='name package'): to show the list of datasets available on the "name package"
- data(): to show the list of datasets available on all loaded packages
- data(name dataset): to load a dataset from a loaded package

Basic commands - Guideline

- lowercase and uppercase are different on R
- '#' can be used to make comments
- **Symbols for maths operation**: + (addition), - (difference), * (product with numbers), / (division with numbers), %/% (integer division), ** (exponentiation), %% (division remainder)
- **Logic operators**: != (not equal), & (AND), | (OR)

Data Types

In R, there are several kind of data types:

- **Numeric**: real numbers with or without decimal points (e.g. 2.5, 2, ...)
- **Integer**: real numbers without decimal points. To specify an integer number we need to use the suffix L (e.g. integer_number = 18L)
- **Logical**: boolean values (TRUE and FALSE)
- **Complex**: complex numbers whose imaginary values are represented by the suffix i (e.g. 2i)
- **Character**: string values (e.g. fruit = "Cherry")
- **Raw**: values as raw bytes

Special data types

R involves two special data types: NULL and NA. When assigning **NULL** to a variable, we convert it into an **empty object**. Instead, **NA** means non-available entry and usually represents a **missing value** in data analysis

```
> null_object <- NULL
> missing <- NA

> null_object
NULL
> missing
[1] NA

> print(1 + missing)
[1] NA
```

Variables and Basic Arithmetic

R can be used as calculator to evaluate simple expression.
Some example:

```
# Sum of two numbers
> sum_n <- 2 + 2.5 #NB: the decimal separator is .
> cat("The sum of the two numbers is", sum_n)
The sum of the two numbers is 4.5
```

```
# Division between two numbers with scientific notation
> x <- 1
> y <- 2.73e4
> print(x/y)
[1] 3.663004e-05
```

Math functions

The most common math functions are already implemented in R:

```
# Exp and log functions
> z <- exp(3)
> y <- log(1)
> print(z - y)
[1] 20.08554
# Pi greco
> pi
[1] 3.141593
# Trigonometric functions
> sin(0); cos(pi)
[1] 0
[1] -1
```

If we want to calculate $\log_2(4)$:

```
> log(x=4, base=2)
[1] 2
```

It's your turn!

1. Given a circle with a radius equal to 4. Calculate perimeter and area of the circle using R
2. Install the library dplyr and show the datasets available.
3. Try to read the help page of factorial and choose.
4. In how many ways can you arrange 7 out of 20 books on a bookshelf? Show the result in R. (Hint: ex.3 can be useful)
5. Round the number n=3.78957 to two decimal places. (Hint: use round function looking the documentation)
6. Visualize all objects saved in R space and then remove them.

Vectors

A vector is an array of elements having the same data type. In R, to create a vector we can use c():

```
> # Different ways to assign x to a vector
> x <- c(23.4, 6, 7.1, 6.9, 34.7)
> assign("x", c(23.4, 6, 7.1, 6.9, 34.7))
> x = c(23.4, 6, 7.1, 6.9, 34.7)
```

The length of the vector we just created can be retrieved by

```
> length(x)
[1] 5
```

The "[. . .]" operator can be used to draw out parts of a vector by specifying one or more indices of numeric type. **In R, the indices of a vector are defined starting with the value 1.**

```
> x[1]; x[c(2,3)]; x[c(1:3)]
[1] 23.4
[1] 6.0 7.1
[1] 23.4  6.0  7.1
```

Vectors

If you want to create a sequence of values by following a specific condition, you can use seq() or rep()

```
> n<-10
> seq(1, n); seq(1, n, by = 2); seq(1, n, length.out = 5)
[1]  1  2  3  4  5  6  7  8  9 10
[1] 1 3 5 7 9
[1]  1.00  3.25  5.50  7.75 10.00
> rep(n, times=5); rep(n, each=5) # same
[1] 10 10 10 10 10
[1] 10 10 10 10 10
> rep(1:4, each=5); rep(1:4, times=5) # different
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
```

Math operations and useful functions

There are some functions to tackle vectors or do math operations

```
> x<-c(x, -6.8) # add a new value to the previous vector
> sort(x) # default: decreasing=F, i.e. ascending order
[1] -6.8  6.0  6.9  7.1 23.4 34.7
> floor(x) # extract the integer part of a decimal number
[1] 23  6  7  6 34 -7
> sign(x) # sign: -1 = negative, 0 = 0, 1 = positive
[1]  1  1  1  1  1 -1
> any(x>4) # is there at least a value in our vector >4?
[1] TRUE
```

- abs(), sum(), mean(), sd(), median(), prod(), cumsum() compute absolute value, sum, average, standard deviation, median, product and cumulative sum
- min(), max() and range() show min and max of a vector
- head() and tail() extract the head and the tail of a vector (6 elements)
- summary() shows summary statistics

Vectors of characters

The R language implements a data structure called ”factor” that facilitates the analysis of categorical data. Factors are particularly useful for representing variables that can take only one category among a known and finite set of values (e.g., gender, type of employment, month of birth, . . .). Values can have an ordering (e.g. hierarchies in the military) or not (e.g. hair color).

```
> days_list <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
> factor(c("Monday", "Sunday"), levels = days_list)
[1] Monday   Sunday
7 Levels: Monday Tuesday Wednesday Thursday ... Sunday
```

The table() function is used to generate a table of absolute frequencies

```
> table(c("A", "B", "B"))
data
A B
1 2
```

Matrix

Matrices are a generalization of vectors in two dimensions. As the vectors, all elements of a matrix must be of the same data type.

```
> matrix(1:10, nrow = 2, ncol = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> matrix(letters[1:10], nrow = 2, ncol = 5, byrow = TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,] "a"  "b"  "c"  "d"  "e"
[2,] "f"  "g"  "h"  "i"  "j"
```

The dim() can be used to show the dimensions of a matrix

```
> x <- matrix(1:10, nrow = 2, ncol = 5)
> dim(x) # dimensions of the matrix x
[1] 2 5
> nrow(x); ncol(x) # total number of rows and columns
[1] 2
[1] 5
```

Math operations

- Matrix multiplication: %*%
- t() and det() to compute matrix transpose and the determinant

```
> x <- matrix(1:4, nrow=2); y <- matrix(c(5,10,33,2), ncol
   =2)
> x; solve(x) # solve the equation a %*% m = b for m. In our
    case, a=x and b=diag(2) (identity matrix)
     [,1] [,2]        [,1] [,2]
 [1,]    1    3   [1,]   -2  1.5
 [2,]    2    4   [2,]    1 -0.5
> crossprod(x,y) # t(x) %*% y
     [,1] [,2]
[1,]   25   37
[2,]   55  107
> solve(x,x) # solve(x) %*% x
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Math operations

- eigen(), svd(), and qr() can be used to compute eigenvalues and eigenvectors, singular value decomposition and QR decomposition
- cbind() and rbind() mean column-binding and row-binding. Let's see an example

```
> cbind(x,y) # join matrices by column
     [,1] [,2] [,3] [,4]
[1,]    1    3    5   33
[2,]    2    4   10    2
```

It's your turn!

7. Calculate absolute and relative frequencies of vector ("Cat","Dog", "Bird", "Bird", "Cat","Cat","Dog"). Print the maximum absolute frequency and the correspondent animal. (Hint: use which() function)
8. Define a vector called z by "pasting" the vectors x=("Red", "Green", "Yellow") and y=("Tomato", "Grass", "Sun") and select only those elements of z that begin with "r" and end in "o". (Hint: use paste() and startsWith() functions)
9. Create a matrix A containing the integers from 10 to 24 having 5 rows and 3 columns where A must be filled by column. Show the firsts 3 rows of the firsts two columns. Calculate the sum of A by row and column (Hint: rowSums(), colSums()). Calculate the transpose of A. Define a new matrix B of appropriate size to calculate the AB product (and calculate that product). Calculate the standard deviation of B by row and then by column (Hint: apply())

Lists

Lists can contain elements with different data type. They are also called "recursive" structures in that one list can also contain another list.

```
> list(FALSE, 0L, pi, "ABC")
[[1]]           [[2]]
[1] FALSE       [1] 0
[[3]]           [[4]]
[1] 3.141593    [1] "ABC"
> x <- list(1, "A", matrix(1:4, 2, 2))
> x[2] # to extract a list
[[1]]
[1] "A"
> x[[2]] # to extract the second element
[1] "A"
```

- If you want to transform a list to a vector, you can use unlist()

Exercise

10. Given a matrix X 2x2 containing the integers from 1 to 4. Try to apply svd() function to X and then reconstruct the input matrix X with the outputs of svd.

Probability functions

R software implements a set of functions having a common syntax to handle random variables (e.g. rnorm, qnorm, dnorm, pnorm).
The first letter identifies the objective of the function:

- **p** means ”probability”, the cumulative distribution function (c.d.f.)
- **d** means ”density”, the density function (p.f. or p.d.f.)
- **r** means ”random”, to generate random numbers
- **q** means ”quantile”, the inverse c.d.f.

The second part describes the random variable (e.g. norm for normal distribution, pois for poisson, gamma for gamma, unif for uniform).

```
> rnorm(n = 5, mean = 0, sd = 1) # vector of 5 normally
    distributed random numbers
> pnorm(0, mean = 0, sd = 1) # P(X<=0) where X ~ N(0,1)
```

R Exercises

11. Let $X \sim N(5, 5)$. Calculate the 0.7 quantile of X.
12. Let $Y \sim Gamma(1, 2)$. Simulate the extraction of n=500 random numbers from Y and save them into a object called "data". Show the summary statistics about it. (To guarantee the reproducibility you need to use set.seed())

Control structures

There are several kind of control structures in R:

- if/else: to test a condition
- for: to execute a loop a fixed number of times
- while: to execute a loop while a condition is true
- repeat: to execute an infinite loop
- break: to break the execution of a loop
- next: to skip an iteration of a loop

The most used control structures are if/else and for.

If/Else

There are different ways to implement if/else conditions:

```r
# One condition
if (TEST) {
    do - something}
# Two conditions
if (TEST) {
    do - something
} else {
    do - something - else}
# More conditions
if (TEST) {
    do - something
} else if (ANOTHER TEST) {
    do - something - else
} else {
    do - something - else}
```

For

A for loop can be implemented using the following

```
for (sequence of indices) {
    body of the loop}
```

Let's see an example: Generate 10 random numbers $X \sim \mathit{Poisson}(10)$ and divided them into even and odd vectors.

```
> set.seed(123)
> x <- rpois(10, lambda = 10) # data generation
> x <- sort(x)
> even_x <- c(); odd_x <- c() # initialize vectors
> for (i in seq_along(x)) { # seq_along => 1:length(10)
+   if(x[i]%%2==0){
+     even_x <- c(even_x, x[i]) # add value x[i] to vector even_x
+   } else {
+     odd_x <- c(odd_x, x[i])
+   }
+ }
```

Functions in R

In R to create your own functions, you can call function() and assign it to an object.

```
> fahrenheit_to_celsius <- function(temp_F) {
+ temp_C <- (temp_F - 32) * 5 / 9
+ return(temp_C)}
> fahrenheit_to_celsius(55) # to show the result
[1] 12.77778
```

In R, it's not necessary to specify the return statement

```
> fahrenheit_to_celsius <- function(temp_F) {
+  (temp_F - 32) * 5 / 9} # as above
> fahrenheit_to_celsius(55)
[1] 12.77778
```

It’s your turn!

13. Try to define a function f which, given a numerical vector as input, returns a list containing the mean, variance, minimum and maximum of that vector. Assign an appropriate name to the elements of the list.
14. Write an R function, num_seq(n), that returns a vector that has the form $\{2^2 \cdot 3, 3^2 \cdot 5, 4^2 \cdot 7, 5^2 \cdot 9, \dots\}$and has exactly n terms.
15. Assume that a data set is stored in the vector data in R. Write a function skewness(data) that calculates the skewness index ($I = \frac{3(\bar{x} - Q_2)}{s}$) of the data set and returns either "The data is significantly skewed" or "The data is not significantly skewed" as appropriate.

Dataframe

R uses an object called ”data.frame” (i.e., class ”data.frame”) to represent matrix of data. The rows of a data.frame correspond to statistical units (i.e., observations) and columns to variables.

```
> x <- data.frame(hair_color = c("Blonde", "Blonde", "Black"
    , "Brown"), gender = c("F", "M", "M", "F"))
> x
  hair_color gender
1     Blonde      F
2     Blonde      M
3      Black      M
4      Brown      F
```

For external datasets, you can use two functions: read.table() and read.csv().

Apply, sapply, tapply, lapply

Instead of for loop, in R you can use apply, sapply, tapply or lapply.This family of functions allows us to apply a certain function to a certain data frame, list or vector and return the result as a list or vector depending on the function we use.

- apply(): to apply a function to the rows or columns of a matrix or data frame.
- sapply(): to apply functions on a list/ vector/ data frame and returns an array or matrix object.
- tapply(): to compute functions to each factor variable in a vector.
- lapply(): to apply functions on list/ vector/ matrix objects and returns a list object.

Examples

1. Let consider two variables: salaries and jobs. Compute the mean for each level of jobs.

```
> salaries <- c(80000, 62000, 113000, 68000, 75000, 79000, 112000, 118000, 65000, 117000)
> jobs <- c(’DS’, ’DA’, ’DE’, ’DA’, ’DS’, ’DS’, ’DE’, ’DE’, ’DA’, ’DE’) # factor variable
> print(tapply(salaries, jobs, mean))
    DA     DE     DS
 65000 115000  78000
```

2. Create a new function add_one that, given an input, adds 1 unit. Then, define a vector my_vector and print a vector by applying add_one at each element.

```
> add_one <- function(x) x+1
> my_vector <- c(1, 2, 3)
> print(sapply(my_vector, add_one))
[1] 2 3 4
```