Introduction to R

Table of contents

1. Introduction to R and RStudio
2. Object types
3. Functions and packages
4. Some fundamentals on vector, matrix, list, and data.frame

Introduction to R and RStudio

Learning Objectives

- become familiar with programming
- become capable of using R software to conduct research independently
  - manipulate data
  - visualize data
  - report results
  - spatial data management

R and RStudio

R

- a very popular statistical programming language used in academia and industry
- started out as software to do statistics, designed by a number of statisticians
- is open-source and free
- has been and is evolving rapidly by the contributions of its users
    - state-of-the-art statistical methods (e.g., machine learning algorithms) written by the developers of the methods
    - geographic information system (GIS)
    - big data handling and analysis

RStudio:

- R has a terrible graphic user interface
- by far the most popular graphic user interface of R

Install R and RStudio

- Install R
- Install RStudio

Introduction to RStudio

Four panes

- R script (upper left)
- Console (lower left)
- Environment (upper right)
- Files, plots, packages, and help (lower right)

Small tips

- Appearance
- Pane Layout

Getting started with R and RStudio

- do basic mathematical operations
- define objects in R
- learn different object types
- use RStudio at the same time

Basic element types (atomic mode)

- integer: e.g., 1, 3,
- numeric (double): e.g., 1, 1.3
- complex:
- logical (boolean): true or false
- character: combination of letters (numerical operations not allowed)

Basic arithmetic: R as a calculator

```
#--- addition ---#
2 + 3.3
```

```
## [1] 5.3
```

```
#--- subtraction ---#
6 - 2.7
```

```
## [1] 3.3
```

```
#--- multiplications ---#
6 * 2
```

```
## [1] 12
```

```
#--- exponentiation ---#
2 ^ 2.4
```

```
## [1] 5.278032
```

RStudio tip:

- `command` + `enter` (Mac) runs the code (`Control` + `enter` (Windows))

Basic arithmetic: R as a calculator (cont.)

```
#--- division ---#
6 / 2
```

```
## [1] 3
```

```
#--- remainder ---#
6 %% 4
```

```
## [1] 2
```

```
#--- quotient ---#
6 %/% 4
```

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

RStudio tip: :

- `command` (`Control`) + 1 for Mac (Windows) to move the cursor to the source pane
- `command` (`Control`) + 2 for Mac (Windows) to move the cursor to the source pane

logical values and operators

```
#--- true or false ---#
5 == 5
```

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

```
5 == 4
```

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

```
5 > 4
```

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

```
5 < 4
```

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

Character

Contents enclosed by double (or single) quotation marks will be recognized as characters.

```
#--- character ---#
"R"
```

```
## [1] "R"
```

```
#--- character ---#
" rocks"
```

```
## [1] " rocks"
```

You cannot do addition using characters

```
"R" + "rocks"
```

```
## Error in "R" + "rocks": non-numeric argument to binary operator
```

We will learn string manipulations later using the `stringr` package.

RStudio tip: :

- `Option` + `-` (`Alt` + `-`) for Mac (Windows) to inset the assignment operator (`<-`)

Assignment

You can assign contents (numeric numbers, character, boolean, etc) to an object on R and reuse it later using either `<-` or `=`.

```
object_name <- contents
```

Example

```
#--- numeric ---#
a <- 1
```

Notice that these objects are now in the list of objects on the environment tab of RStudio.

Object evaluation

Once objects are created, you can evaluate them on the console to see what is inside:

```
a
```

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

RStudio tip: :

- `Command` + `up` (`Control` + `up`) for Mac (Windows) to look at the history of codes you have run (or go to the "History tab" on the right upper pane)

Assignment (cont.)

Other examples

```
#--- character ---#
b <- "R rocks"

b
```

```
## [1] "R rocks"
```

```
#--- logical ---#
d <- 1 == 2

d
```

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

Assignment (cont.)

You can also use `=` for assignment

```r
#--- character ---#
a <- "R rocks"

a
```

```
## [1] "R rocks"
```

It does not really matter which of `<-` or `=` to use. You should pick whichever makes sense for you. But, it is a good idea to be consistent.

Assignment (cont.)

Several things to remember about assignment:

- If you assign contents to an object of the same name, the object that had the same name will be overwritten

```
a <- 3
a <- 1
a
```

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

- Object names cannot start with a numeric number. Try the following:

```
1a <- 2
```

- You cannot use a reserved word as the name of an object (complete list found here)

```
#--- try this ---#
if <- 3
```

Objects

Objects

R is an object-oriented programming (OOP), which basically means:

"Everything is an object and everything has a name."

Different object types (classes)

- vector
- matrix
- data.frame
- list
- function

Vector (one-dimensional array)

A vectors is a class of object that consists of elements of the same kind (it can have only one element). You use `c()` to create a vector.

Example

```
#--- create a vector of numbers ---#
a <- c(4, 3, 5, 9, 1)

a
```

```
## [1] 4 3 5 9 1
```

```
#--- create a vector of characters ---#
b <- c("python", "is", "better", "than", "R")

b
```

```
## [1] "python" "is"     "better" "than"   "R"
```

Vector (cont.)

What if we mix elements of different mode

Example

```
#--- create a vector of numbers ---#
a_vector <- c(4, 3, "5", 9, 1)

#--- see its content ---#
a_vector
```

```
## [1] "4" "3" "5" "9" "1"
```

All the numeric values are converted to characters.

List

A `list` is a class of object that consists of elements of mixed types:

Example

```r
#--- create a vector of numbers ---#
a_list <- list(4, 3, "5", 9, 1)

#--- see its content ---#
a_list
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] "5"
##
## [[4]]
## [1] 9
##
## [[5]]
## [1] 1
```

A `list` is very flexible. It can hold basically any type of R objects as its elements.

Matrix (two-dimensional array)

A matrix is a class of object that consists of elements of the same kind (it can have only one element) stored in a two-dimensional array.

Example

```r
#--- create a matrix of numbers ---#
M_num <- matrix(c(2, 4, 3, 1, 5, 7), nrow = 3)

M_num
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    4    5
## [3,]    3    7
```

Matrix (two-dimensional array)

A matrix is a class of object that consists of elements of the same kind (it can have only one element) stored in a two-dimensional array.

Example

```r
#--- create a matrix of characters ---#
M_char <- matrix(c("a", "b" , "c", "d", "e", "f"), nrow = 3)

M_char
```

```
##      [,1] [,2]
## [1,] "a"  "d" 
## [2,] "b"  "e" 
## [3,] "c"  "f" 
```

data.frame

`data.frame` is like a matrix (or a list of columns)

```
#--- create a data.frame ---#
yield_data <- data.frame(
  nitrogen = c(200, 180, 300),
  yield = c(240, 220, 230),
  state = c("Kansas", "Nebraska", "Iowas")
)

yield_data
```

```
##   nitrogen yield    state
## 1      200   240   Kansas
## 2      180   220 Nebraska
## 3      300   230    Iowas
```

There are different kinds of objects that are like "data.frame"

- tibble
- data.table

Objects (cont.)

It is critical to recognize the class of the objects:

- the same function does different things depending on the class of the object to which the function is applied
- some functions work on some object classes, but not on others

Many of the errors you will encounter while working on R has something to do with applying functions that are not applicable to the objects you are working on!

Objects (cont.)

Use `class`, `typeof`, and `str` commands to know more about what kind of objects you are dealing with:

```
#--- check the class ---#
class(yield_data)
```

```
## [1] "data.frame"
```

```
#--- check the "internal" type ---#
typeof(yield_data)
```

```
## [1] "list"
```

```
#--- look into the structure of an object ---#
str(yield_data)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ nitrogen: num  200 180 300
##  $ yield   : num  240 220 230
##  $ state   : chr  "Kansas" "Nebraska" "Iowas"
```

Objects (cont.)

You could also use `View()` function for visual inspection:

```
View(yield_data)
```

Function

Function

What is a function?

A function takes R objects (vector, data.frame, etc), processes them, and returns R objects

Example

`min()` takes a vector of values as an argument and returns the minimum of all the values in the vector

```
min(c(1, 2))
```

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

Some other useful functions

- create a sequence of values

```
v1 <- seq(0, 100, by = 5)
v2 <- seq(0, 100, length = 21)
```

- repeat values

```
v3 <- rep(10,5)
```

- sum values

```
v1_sum <- sum(v1)
```

- find the length of an vector

```
v1_len <- length(v1)
```

`+` and `>` on the console

Notice that once the code is run, you will have `>` sign on the R console. But, sometimes, you see just see `+. This either means,

- The code is still running (in this case, you will see the stop sign on the right upper part of the console pane.)
- You did not close `(` with `)` or close `{` with `}`.

Try to run the following:

```
mean(c(3, 5)
```

Then, you should see `+` and nothing seems to be happening. This is because R is waiting for more codes because you did not close `(` with `)`.

- When you see `+`, look at the codes that you just sent to the console and see if you have not closed `(` or `{`. If you found unclosed `(` or `{`, then close it by typing `)` or `}` on the console.

- If you cannot tell what happened by looking at the code, then you could also type random sequence of letters after `+`, which would cause an error, and then can start run codes again.

Exercises

- generate a vector (call it x) that starts from 1 and increase by 2 until 99

- calculate the sample mean of x

$\frac{1}{n}\sum_{i=1}^{n} x_i$

Next class: Rmarkdown