

Introduction to R

Lecture 1

Louis SIRUGUE

CPES 2 - Fall 2022

Welcome to the course!

About me:

- PhD student at the Paris School of Economics
- I work primarily on:
 - Intergenerational (income) mobility
 - Residential segregation
 - Discrimination
- I do empirical research, so I use Econometrics and (R) programming on a daily basis
- You can reach me at louis.sirugue@psemail for any question or comment about the course

About the course:

- **Objective:**
 - Give you the necessary statistical and data visualization tools to perform data analyses
- **Prerequisites:**
 - None (middle-school maths level)
- **What you'll learn:**
 - Find and manipulate data
 - Summarize data with relevant statistics and compelling graphics
 - Carry out an empirical research project
- **Thus, this course is a mix of:**
 - Programming on R
 - Basic Statistics
 - Introductory Econometrics

First semester

Course format:

→ 15 formal lectures divided into 3 parts:

1. Introduction to R Programming
2. Data analysis
3. Data visualization

Grading:

- Weekly short online quizzes (25% of the grade)
 - 5 short questions related to the content of the previous class
 - The quiz opens at the end of the lecture and closes at the beginning of the following one
 - Possibility to retry the quiz as many times as you want before submitting
- Homeworks (3, one for each part, 25% of the grade each)
 - Homeworks are already available [here](#), work on it progressively as we go through the course

Second semester

Course format:

- Apart from 2 refreshers at the beginning, the whole semester will be dedicated to your research projects
- At each session a T.A. will guide you through the research process
 - From framing a relevant question and finding appropriate data
 - To the final details of your data analysis

Grading:

- Basically the whole grade will be based on your research project
- The detailed (indicative) grading scheme is available [here](#).

→ Visit the [course webpage](#) for more detail, everything's there!

Let's delve into it!

1. Getting started

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages

- 4.1. Functions
- 4.2. Packages

5. A few words on learning R

6. Wrap up!

Let's delve into it!

1. Getting started

- 1.1. About R
- 1.2. The R Studio environment

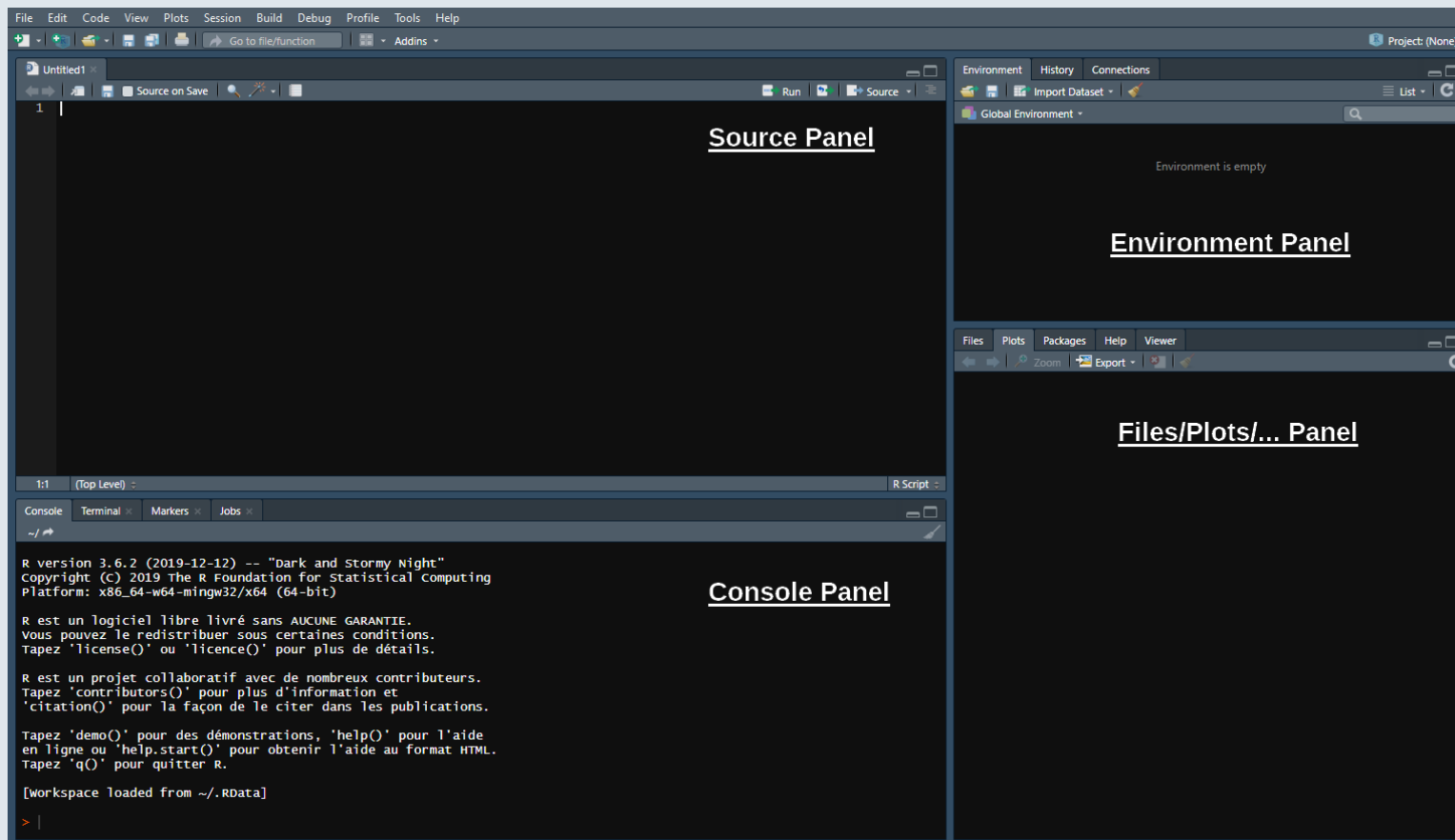
1. Getting started

1.1. About R

- R is a **programming language** and free software environment for **statistical computing and graphics**
- The R language is widely (and increasingly) used in **academic and non-academic research** in fields like:
 - Economics
 - Statistics
 - Biostats
- Things you can do with R:
 - Reports
 - Nice plots
 - All the material of this course
 - Academic research
 - Win kaggle competitions
 - Interactive data visualization
 - Art

1. Getting started

1.2. The R studio environment



1. Getting started

1.2. The R studio environment

→ The Console panel

- This is where you **communicate with R**. You can write instructions after the `>`, press enter and R will **execute**
 - Try with `1+1`:

```
1+1
```

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

→ The Source panel

- This is where you **write and save your code** (File > New File > R Script)
 - Separate different commands with a line break
 - The `#` symbol allows to annotate the code, everything after `#` will be ignored by R until the next line break

```
1+1 # Do not put 2+2 on the same line, press enter to go to next line
2+2
```

1. Getting started

1.2. The R studio environment

→ The Source panel

- To send the command from the source panel to the console panel:
 1. Highlight the lines you want to execute
 2. Press `ctrl + enter`
- If you do not highlight anything the line of code where your cursor stands will be executed
- Check the console to see the output of your code

→ The Environment panel

- Data analysis requires manipulating datasets, vectors, functions, etc.
 - These **elements are stored in the environment** panel.
- For instance we can assign a value to an object using `<-`

```
x <- 1
```

→ You now have an object called 'x' in your environment, which takes the value 1

1. Getting started

1.2. The R studio environment

→ The Environment panel

- Now that the object `x` is stored in your environment, you can use it:

```
x + 1
```

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

- You can also modify that object at any point:

```
x <- x + 1 # What's gonna be the value of x?
```

```
x
```

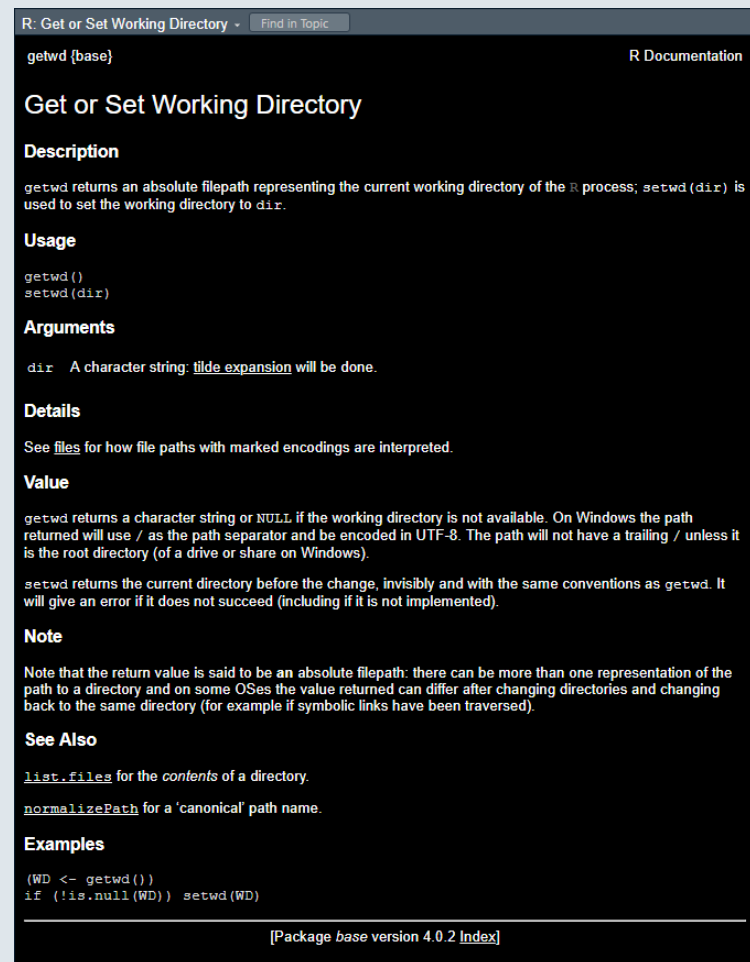
```
## [1] 2
```

1. Getting started

1.2. The R studio environment

→ The Files/Plots/... panel

- In this panel we'll mainly be interested in the following 4 tabs
 - **Files:** Shows your working directory
 - **Plots:** Where R returns plots
 - **Packages:** A library of tools that we can load if needed
 - **Help:** Where to look for documentation on R functions
- Enter `?getwd()` in the console to see what a **help file** looks like
 - It **describes** what the command does
 - It **explains** the different parameters of the command
 - It **gives examples** of how to use the command



The screenshot shows the R help documentation for the `getwd` and `setwd` functions. The title is "Get or Set Working Directory". The description states that `getwd` returns an absolute filepath representing the current working directory, and `setwd` is used to set the working directory to `dir`. The usage section shows `getwd()` and `setwd(dir)`. The arguments section indicates that `dir` is a character string where tilde expansion will be done. The details section refers to `files` for file path encodings. The value section explains that `getwd` returns a character string or NULL, and `setwd` returns the current directory before the change. A note mentions that the return value is an absolute filepath. The "See Also" section lists `list.files` and `normalizePath`. Examples show how to assign `getwd()` to a variable and check if it is NULL before setting it.

R: Get or Set Working Directory - Find in Topic R Documentation

`getwd` (base)

Get or Set Working Directory

Description

`getwd` returns an absolute filepath representing the current working directory of the R process; `setwd` (dir) is used to set the working directory to dir.

Usage

```
getwd()
setwd(dir)
```

Arguments

dir A character string. [tilde expansion](#) will be done.

Details

See [files](#) for how file paths with marked encodings are interpreted.

Value

`getwd` returns a character string or NULL if the working directory is not available. On Windows the path returned will use / as the path separator and be encoded in UTF-8. The path will not have a trailing / unless it is the root directory (of a drive or share on Windows).

`setwd` returns the current directory before the change, invisibly and with the same conventions as `getwd`. It will give an error if it does not succeed (including if it is not implemented).

Note

Note that the return value is said to be an absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the same directory (for example if symbolic links have been traversed).

See Also

[list.files](#) for the contents of a directory.

[normalizePath](#) for a 'canonical' path name.

Examples

```
(WD <- getwd())
if (!is.null(WD)) setwd(WD)
```

[Package base version 4.0.2 [Index](#)]

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages

- 4.1. Functions
- 4.2. Packages

5. A few words on learning R

6. Wrap up!

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

2. Classes of R objects

→ There are 3 main classes of R objects

- **Numeric values:**

- 42
- -10^3
- 2.6879098767

- **Character values:**

- A
- Hello world
- PM2.5

- **Logical values:**

- TRUE (*or equivalently T*)
- FALSE (*or equivalently F*)

The class of an object is a **key attribute**:

- Different types of operations are available for different classes of object
- Objects of different classes may react differently to a same operation
- Class mistakes is a common source of programming errors

2. Classes of R objects

2.1. Numeric values

Basic operations with numeric values:

```
2 + 3 # Addition
```

```
## [1] 5
```

```
2 - 3 # Subtraction
```

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

```
2 * 3 # Multiplication
```

```
## [1] 6
```

```
2 / 3 # Division
```

```
## [1] 0.6666667
```

```
2 ^ 2 # Exponent
```

```
## [1] 4
```

```
4 ^ (1/2) # Root
```

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

- Store the result of a computation in an object:

```
result <-((2 + 2)^(2 / (2 * 2))) - 2  
result
```

```
## [1] 0
```


2. Classes of R objects

2.2. Character values

Character values need to be surrounded by quotation marks (' or ")

```
language <- "R"  
language
```

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

Without quotation marks, R will think that you refer to an object

```
R
```

```
## Error in eval(expr, envir, enclos): objet 'R' introuvable
```

Characters are not necessarily letters

```
character_string <- "R' {ç^*μ,,, _hi3"
```

2. Classes of R objects

2.2. Character values

Useful functions with characters

Combining strings

```
paste("Hello", "world")
```

```
## [1] "Hello world"
```

```
paste("a", "b", "c", sep = "-")
```

```
## [1] "a-b-c"
```

```
paste0("a", "b", "c")
```

```
## [1] "abc"
```

Splitting strings

```
substr("abcdefghijklmnopqrstuvwxyz", 3, 5)
```

```
## [1] "cde"
```

```
nchar("Hello world")
```

```
## [1] 11
```

```
substr("Hello world", 5, nchar("Hello world"))
```

```
## [1] "o world"
```

2. Classes of R objects

2.3. Logical values

A logical value/**Boolean**, is a value that can be either:

- **TRUE** (can be abbreviated as **T**)

```
T
```

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

- or **FALSE** (can be abbreviated as **F**)

```
F
```

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

```
2 + 2 == 4 # Equality
```

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

```
2 + 2 != 4 # Inequality
```

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

```
1 > 2 # Strictly lower/greater
```

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

```
2 <= 2 # Weakly lower/greater
```

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

2. Classes of R objects

2.3. Logical values

And (&) / or (|)

```
(1 == 1) & (1 == 2) #AND
```

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

```
(1 == 1) | (1 == 2) #OR
```

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

```
1 == 1 | (2 == 2 & 1 == 2) #Parentheses matter
```

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

```
(1 == 1 | 2 == 2) & 1 == 2 #Parentheses matter
```

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

Opposite (!)

```
is.numeric(1)
```

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

```
!is.numeric(1)
```

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

```
3 < 2
```

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

```
!3 < 2
```

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

2. Classes of R objects

2.4. Manipulate classes

- It is important to **keep in mind the class** of the object you are working with because they work differently
 - For instance, you can add two numeric values together but not two characters
- You can **find the class** of an object by hovering your mouse over its name in the environment panel, or using the **class()** function:

```
class(2)
```

```
## [1] "numeric"
```

```
class("a")
```

```
## [1] "character"
```

```
class(FALSE)
```

```
## [1] "logical"
```

2. Classes of R objects

2.4. Manipulate classes

- You can **change the class** of an object using functions of the form **as.[character/numeric/logical]()**

Class conversion table:

	as.numeric()	as.character()	as.logical()
numeric	No effect	Converts numeric values into strings of numbers	Returns TRUE if != 0 Returns FALSE if 0
character	Converts strings of numbers into numeric values Returns NA if characters in the string	No effect	Returns TRUE if "T" or "TRUE" Returns FALSE if "F" or "FALSE" Returns NA otherwise
logical	Returns 1 if TRUE Returns 0 if FALSE	Returns "TRUE" if TRUE Returns "FALSE" if FALSE	No effect

- NA** stands for *'Not Available'*, it corresponds to a **missing value**
 - Beware of NAs**, it is a common source of programming errors (more on that next week)

2. Classes of R objects

2.4. Manipulate classes

Some examples

```
as.numeric("2022")
```

```
## [1] 2022
```

```
as.numeric("2022-2023")
```

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

```
as.numeric(T)
```

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

```
as.numeric(FALSE)
```

```
## [1] 0
```

```
as.character(2022)
```

```
## [1] "2022"
```

```
as.logical(0)
```

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

```
as.logical(1)
```

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

```
as.logical(-.75)
```

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

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages

- 4.1. Functions
- 4.2. Packages

5. A few words on learning

6. Wrap up!

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

3. Vectors

3.1. Definition

- In R, a vector is a **sequence of objects that have the same class**
 - To create a vector you should list its elements separated by commas inside `c()`

```
months <- c("January", "February", "March", "April", "May", "June", "July",  
            "August", "September", "October", "November", "December")
```

- Vectors are **ordered**, you can recover elements of a vector using their position in the sequence using `[]`

```
months[3]
```

```
## [1] "March"
```

- Conversely, the function `match()` allows you to recover the position of specific elements in a vector:

```
match(c("August", "December"), months)
```

```
## [1] 8 12
```

3. Vectors

3.2. Operations with vectors

- Numeric operations apply to every element of the vector:

```
4 + c(10, 20, 30)
```

```
## [1] 14 24 34
```

```
c(1, 2, 3) * 4
```

```
## [1] 4 8 12
```

```
4 ^ c(1, 2, 3)
```

```
## [1] 4 16 64
```

```
c(1, 2, 3) ^ 4
```

```
## [1] 1 16 81
```

3. Vectors

3.3. Useful functions

- The `length()` function for the **number of elements** in a vector

```
length(months)
```

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

- The `rep()` function for **repeating a vector**

- Vector-wise:

```
rep(c(1, 2, 3), 2)
```

```
## [1] 1 2 3 1 2 3
```

- Or element-wise:

```
rep(c(1, 2, 3), each = 2)
```

```
## [1] 1 1 2 2 3 3
```

3. Vectors

3.3. Useful functions

- The `seq()` function for **sequences**

```
seq(0, 100, 10)
```

```
## [1] 0 10 20 30 40 50 60 70 80 90 100
```

```
seq(2, 3, .2)
```

```
## [1] 2.0 2.2 2.4 2.6 2.8 3.0
```

- Sequences of **consecutive integers** can be easily produced using `:`

```
1:10
```

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

3. Vectors

3.3. Useful functions

- The `paste()` function also applies to all the elements of the vector

```
paste0("A", 1:5)
```

```
## [1] "A1" "A2" "A3" "A4" "A5"
```

- And can merge the elements of a vector with the `collapse` argument

```
sentence <- c("Lorem", "ipsum", "dolor", "sit", "amet")  
sentence
```

```
## [1] "Lorem" "ipsum" "dolor" "sit"   "amet"
```

```
paste0(sentence, collapse = " ")
```

```
## [1] "Lorem ipsum dolor sit amet"
```

3. Vectors

3.4. Classes of vectors

- As the elements of a vector should be of the same class, the class of a vector is always that of its elements:

```
class(c(1, 2, 3))
```

```
## [1] "numeric"
```

- But there is **another class** you should know for vectors: the **factor** class
 - It's made for vectors with numeric values that can't be interpreted but simply indicate a **group/category**
- Consider for instance the following age vectors:
 1. Vector age1 indicates the actual age of an individual, it should be of class numeric
 2. Vector age2 indicates an age group (1 = 1 to 10, 2 = 10 to 20, ...) it should be of class factor

→ *In vector age2, numbers don't mean anything, they're just group indicators and R should interpret it as such*

3. Vectors

3.4. Classes of vectors

- R understands that multiplying the values of a factor by 2 makes no sense

```
as.factor(c(1, 2, 2)) * 2
```

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

- R converts a factor to the numeric format by attributing a number from 1 to n to each category

```
as.numeric(as.factor(c(46, 44, 46, 45)))
```

```
## [1] 3 1 3 2
```

- To have a numeric vector with the values of a factor, you should thus go through the character class

```
as.numeric(as.character(as.factor(c(46, 44, 46, 45))))
```

```
## [1] 46 44 46 45
```


Practice

→ You have the social security number of 4 individuals

```
ssn <- c("1970248765890", "2891135987473", "17875109483", "29931298043")
```

How SSNs are constructed:

1	97	02	48	765	890
Sex	Year of birth	Month of birth	Department of birth	Municipality INSEE code	Registration number

Task: Use R programming to compute the age of these 4 individuals

1. Copy/paste the line above into an .R script and run it
2. Extract the second to third characters from the SSNs
3. Convert these values to the numeric class
4. Add 1900 to these number to obtain the year of birth
5. Subtract these number from 2022 to get the age

You've got 10 minutes!

Solution

1) Copy/paste the line above into an .R script and run it

```
ssn <- c("1970248765890", "2891135987473", "17875109483", "29931298043")
```

2) Extract the second to third characters from the SSNs

```
birth_year <- substr(ssn, 2, 3)
```

3) Convert these values to the numeric class

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

4) Add 1900 to these numbers and subtract from 2022 to get the age

```
age <- 2022 - (birth_year + 1900)  
age
```

```
## [1] 25 33 44 23
```

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors ✓

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages

- 4.1. Functions
- 4.2. Packages

5. A few words on learning

6. Wrap up!

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors ✓

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages

- 4.1. Functions
- 4.2. Packages

4. Functions and packages

4.1. Functions

- **Functions**, like `paste()`, `match()`, `sep()`, etc., are **sequences of operations** condensed into one command.
- Functions take **arguments as inputs**, specified in the parentheses and separated by commas, and **return an output** given these arguments.

Consider the following example:

```
paste("Hello", "world")
```

```
## [1] "Hello world"
```

- Here the **function** `paste()`:
 - Takes `"Hello"` and `"world"` as **inputs**
 - Produces `"Hello world"` as an **output**

→ Let's make our own function!

4. Functions and packages

4.1. Functions

- Imagine you have the following grade distribution

```
grades <- c(8, 10, 12, 13, 15, 15, 16, 17.5, 18, 19.5, 20)
```

- You want to add two points to everybody but grades should not be higher than 20

```
grades <- grades + 2 # add 2 each element of the vector  
grades # Some grades are greater than 20
```

```
## [1] 10.0 12.0 14.0 15.0 17.0 17.0 18.0 19.5 20.0 21.5 22.0
```

```
grades > 20 # Their location in the vector can be found with a logical statement...
```

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

```
grades[grades > 20] <- 20 # ... so we can easily replace their values
```

4. Functions and packages

4.1. Functions

- If you need to do such computations frequently, it is handy to make a function so that it boils down to a single line of code

```
shift_grades <- function(grades, shift) {  
  grades <- grades + shift # Shift grades  
  grades[grades > 20] <- 20 # Cap to 20  
  return(grades) # Return output  
}
```

```
shift_grades(grades = c(0, 10, 20), shift = 5)
```

```
## [1]  5 15 20
```

```
shift_grades(grades = 10:20, shift = 3)
```

```
## [1] 13 14 15 16 17 18 19 20 20 20 20
```

4. Functions and packages

4.2. Packages

It is often the case that user-created functions can be very useful to many other users. That's what packages are about. **Packages are bundles of functions** that R users put to the disposal of other R users. Packages are centralized on the [Comprehensive R Archive Network \(CRAN\)](#). **To download a CRAN package** you can simply use `install.packages()`. Next time we will use functions from the `tidyverse` package. Let's install it already.

```
install.packages("tidyverse") # Requires and internet connection
```

- The `tidyverse` package is now installed on your computer and you won't have to do it again. But to be able to use `tidyverse` functions the **package should also be loaded** in our R session with the function `library()`.

```
library(tidyverse)
```

- Once this command is run you have access to all the `tidyverse` functions in your environment. But even though the package is permanently installed, it is **loaded only for your current session**. So each time you start a new R session, you'll have to load the packages you need with the `library()` command.

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors ✓

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages ✓

- 4.1. Functions
- 4.2. Packages

5. A few words on learning R

6. Wrap up!

Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors ✓

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages ✓

- 4.1. Functions
- 4.2. Packages

5. A few words on learning R

5. A few words on learning R

We're finally done with the "theoretical" part of programming!

- That's normal if some concepts we saw are not completely obvious to you, fluency will come with practice
- Take time to go through the first two lectures by yourself and tell me if anything's unclear.
- There are still some important coding tools you need to learn but we'll see that progressively throughout the course.

When your code won't work

To finish the Lecture I'd like to give you some tips about learning programming. Often times it can be quite frustrating when you can't make your code work.

- When it doesn't not work the way you want
- When it doesn't work at all

5. A few words on learning R

When it doesn't work the way you want

- When things do not work the way you want, NAs are the usual suspects. For instance, this is how the mean function reacts to NAs.

```
mean(c(1, 2, NA))
```

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

```
mean(c(1, 2, NA), na.rm = T)
```

```
## [1] 1.5
```

- From the output it obvious in this case that NAs are the problem, but sometimes it's not that transparent. So take a look at your data to see whether NAs could mess things up. You can seek for NAs using `is.na()`:

```
is.na(c(1, 2, NA))
```

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

5. A few words on learning R

Where to find help

- Help files are made for that. Sometimes things don't work just because you did not understand a function's argument or you don't know that there exists an argument you should use. Every function has a help file, just enter the name of the function preceded by a question mark in your console and it will appear in the Help tab of R studio.

```
?pivot_longer
```

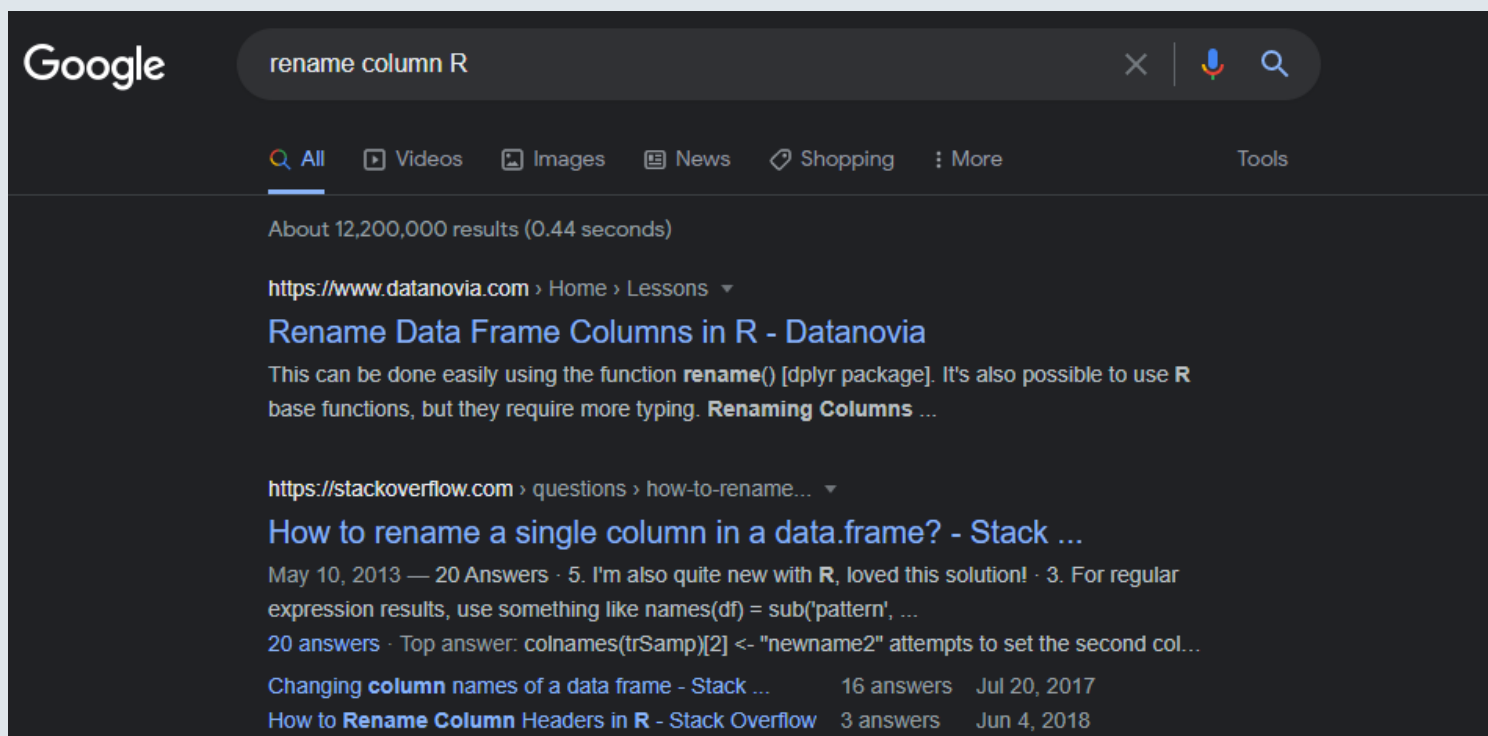
Arguments

<code>data</code>	A data frame to pivot.
<code>cols</code>	<code><tidy-select></code> Columns to pivot into longer format.
<code>names_to</code>	<p>A string specifying the name of the column to create from the data stored in the column names of <code>data</code>.</p> <p>Can be a character vector, creating multiple columns, if <code>names_sep</code> or <code>names_pattern</code> is provided. In this case, there are two special values you can take advantage of:</p> <ul style="list-style-type: none">• <code>NA</code> will discard that component of the name.• <code>.value</code> indicates that component of the name defines the name of the column containing the cell values, overriding <code>values_to</code>.
<code>names_prefix</code>	A regular expression used to remove matching text from the start of each variable name.

5. A few words on learning R

Where to find help

- Search on the internet. I'm pretty sure that any question you might have during these first months of programming is already asked and answered on <https://stackoverflow.com/>.



5. A few words on learning R

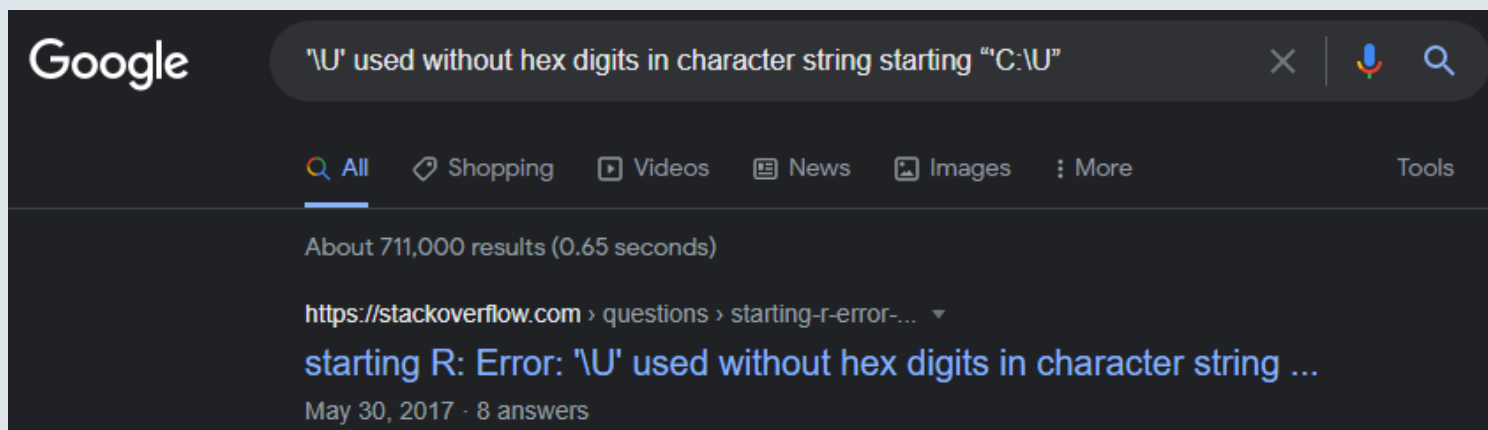
When it doesn't work at all

- Sometimes R breaks and returns an error. And errors are usually kind of cryptic.

```
read.csv("C:\\Users\\l.sirugue\\Documents\\R")
```

Error: '\U' non suivi de chiffres hexadécimaux dans la chaîne de caractères débutant "'C:\\U"

- Try to look for keywords that might help you understand where it comes from, and paste your it in Google with the name of your command. Chances are many people already struggled with that



Overview

1. Getting started ✓

- 1.1. About R
- 1.2. The R Studio environment

2. Classes of R objects ✓

- 2.1. Numeric values
- 2.2. Character values
- 2.3. Logical values
- 2.4. Manipulate classes

3. Vectors ✓

- 3.1. Definition
- 3.2. Operations with vectors
- 3.3. Useful functions
- 3.4. Classes of vectors

4. Functions and packages ✓

- 4.1. Functions
- 4.2. Packages

5. A few words on learning R ✓

6. Wrap up!

6. Wrap up!

1. Main classes of R objects: numeric, character, logical

```
is.numeric("a") # What would be the output?
```

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

2. Vectors

```
week <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")  
length(week) # What would be the output?
```

```
## [1] 7
```

```
match(7, week) # What would be the output?
```

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

6. Wrap up!

3. Functions

```
shift_grades <- function(grades, shift) { # Inputs
  grades <- grades + shift
  grades[grades > 20] <- 20
  return(grades) # Output
}
```

```
shift_grades(grades = c(0, 10, 20), shift = 5)
```

```
## [1]  5 15 20
```

4. Packages

```
install.packages("tidyverse") # Run only once to download the package
library(tidyverse) # Run at each session to load the package
```

To do by next time

- 1) Go through the slides once again
- 2) Do Quiz 1 and send me your results
- 3) Install all the packages required for this course

```
install.packages()
```

- 4) Send me your questions if you have some

See you next week!