SciencesPo
DEPARTMENT OF ECONOMICS

ScPoEconometrics

Introduction

Florian Oswald
SciencesPo Paris
2019-08-20

Welcome to ScPoEconometrics!

- In this course you will learn about the basics of *Econometrics*.

Welcome to ScPoEconometrics!

- In this course you will learn about the basics of *Econometrics*.
- You will also learn to use the `R` software.

Welcome to ScPoEconometrics!

- In this course you will learn about the basics of *Econometrics*.
- You will also learn to use the `R` software.

What is *Econometrics*, actually?

- A set of techniques and methods to answer questions with data.
- Econometrics shares many things with Applied Statistics and Machine Learning.
- Some Examples!

Answering Questions with Econometrics

> Does inauguration of World Bank-financed projects in Sub-Saharan Africa *cause* an electoral benefit for the incumbent?

Answering Questions with Econometrics

Does inauguration of World Bank-financed projects in Sub-Saharan Africa *cause* an electoral benefit for the incumbent?

Does a certain concentration of `airbnb` listings in some urban area *cause* an increase in long-term rents?

Answering Questions with Econometrics

Does inauguration of World Bank-financed projects in Sub-Saharan Africa *cause* an electoral benefit for the incumbent?

Does a certain concentration of `airbnb` listings in some urban area *cause* an increase in long-term rents?

Will increasing the minimum wage *cause* greater unemployment?

Causality

- Notice the keyword **cause** in all of the above.
- Notice also that *many other factors could have caused* each of those outcomes.
- Econometrics is often about spelling out conditions under which we can claim to measure causal relationships.
- We will encounter the most basic of those conditions, and talk about some potential pitfalls.

This Course

- Teach you the basics of *Linear Regression*

This Course

- Teach you the basics of *Linear Regression*
- Introduce you to the `R` software environment.

This Course

- Teach you the basics of *Linear Regression*
- Introduce you to the `R` software environment.
- This is *not* a course about `R`.

This Course

- Teach you the basics of *Linear Regression*
- Introduce you to the `R` software environment.
- This is *not* a course about `R`.

Grades

1. There will be quizzes on Moodle roughly every two weeks.
2. There will be two or three take home exams / case studies.
3. There will be *no* final exam. Your grade will be 40% of 1 and 60% of 2.

R

What is R?[1]

To quote the R project website:

> R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

What is R?[1]

To quote the R project website:

> R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

What does that mean?

- R was created for statistical and graphical work.
- R has a vibrant, thriving online community. (stack overflow)
- Plus it's **free** and **open source**.

[1]: The next 3 slides have been shamelessly copied from Ed Rubin's course

Why are we using R?

1. R is **free** and **open source**—saving both you and the university 💰💵💰.

2. *Related:* Outside of a small group of economists, private- and public-sector **employers favor R** over Stata and most competing softwares.

3. R is very **flexible and powerful**—adaptable to nearly any task, *e.g.*, 'metrics, spatial data analysis, machine learning, web scraping, data cleaning, website building, teaching.

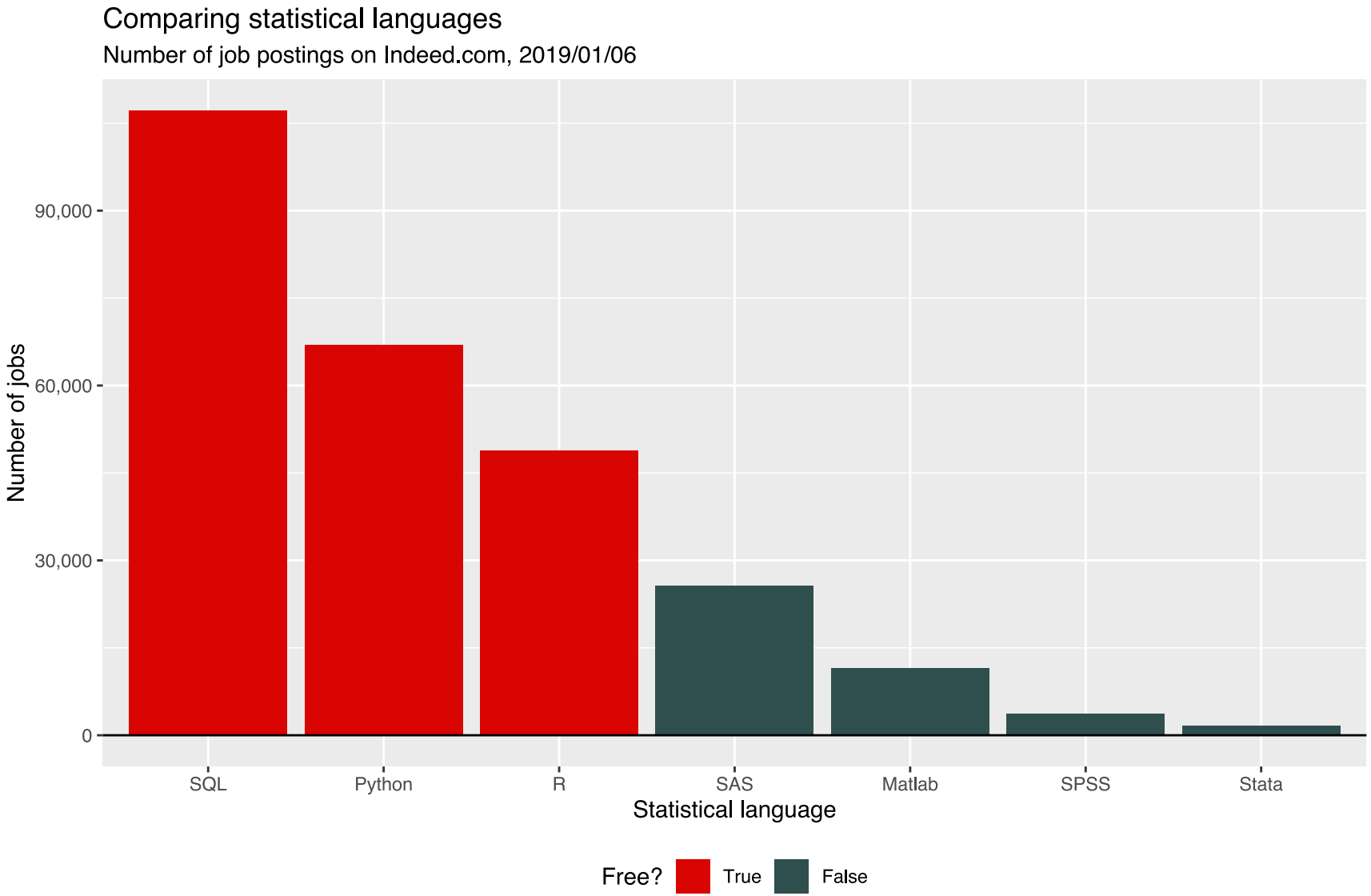
Why are we using R?

4. *Related:* R imposes **no limitations** on your amount of observations, variables, memory, or processing power. (I'm looking at **you**, Stata.)

5. If you put in the work[2], you will come away with a **valuable and marketable** tool.

6. I 💖 **R**

[2]: Learning R definitely requires time and effort.


Comparing statistical languages
Number of job postings on Indeed.com, 2019/01/06
Number of jobs
90,000
60,000
30,000
0
SQL
Python
R
SAS
Matlab
SPSS
Stata
Statistical language
Free?
True
False

R SHOWCASE

Data Wrangling

- The `flights` dataset contains on-time data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013.
- Suppose we want to know the relationship between distance and average delay, as in this example.
- We need to group the data by destination, summarise to get distance, delay and number of flights

```
library(nycflights13)
 # select 3 cols of interest
fl = flights[ ,c("dest","distance","arr_delay")]
# show first 4 lines of this dataframe
head(fl,n = 4)
```

```
## # A tibble: 4 x 3
##   dest  distance arr_delay
##   <chr>    <dbl>     <dbl>
## 1 IAH       1400        11
## 2 IAH       1416        20
## 3 MIA       1089        33
## 4 BQN       1576       -18
```

Data Wrangling

- There are always several ways to achieve a goal. (In life 😄)
- Here are the two leading data packages:

dplyr

```
delays_dplyr <- fl %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

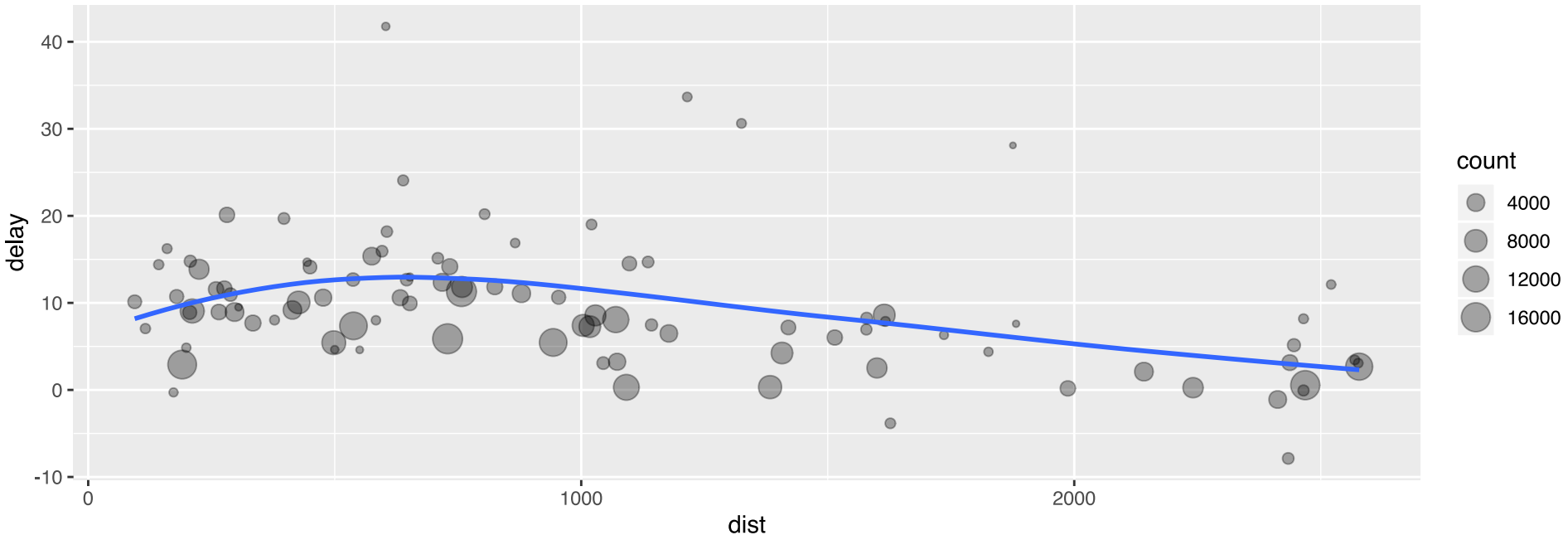
data.table

```
library(data.table)
dl_dt = data.table(fl)
delays_dt <-
  dl_dt[, list(count = .N,
               dist = mean(distance,na.rm=T),
               delay = mean(arr_delay, na.rm=T)),
          by = dest][count > 20 & dest != "HNL"]
```

Plotting

- Now we could *look* at `delays_dt`, or compute some statistics from it.
- Nothing beats a picture, though:

```
ggplot(data = delays_dt,
       mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



Plotting in Base R vs ggplot

- R has very good graphical capabilities outside of contributed packages like ggplot2.
- We will encounter both approaches in this course.
- You will quickly see that some things are easier in base rather than ggplot and vice versa.

Spatial Data

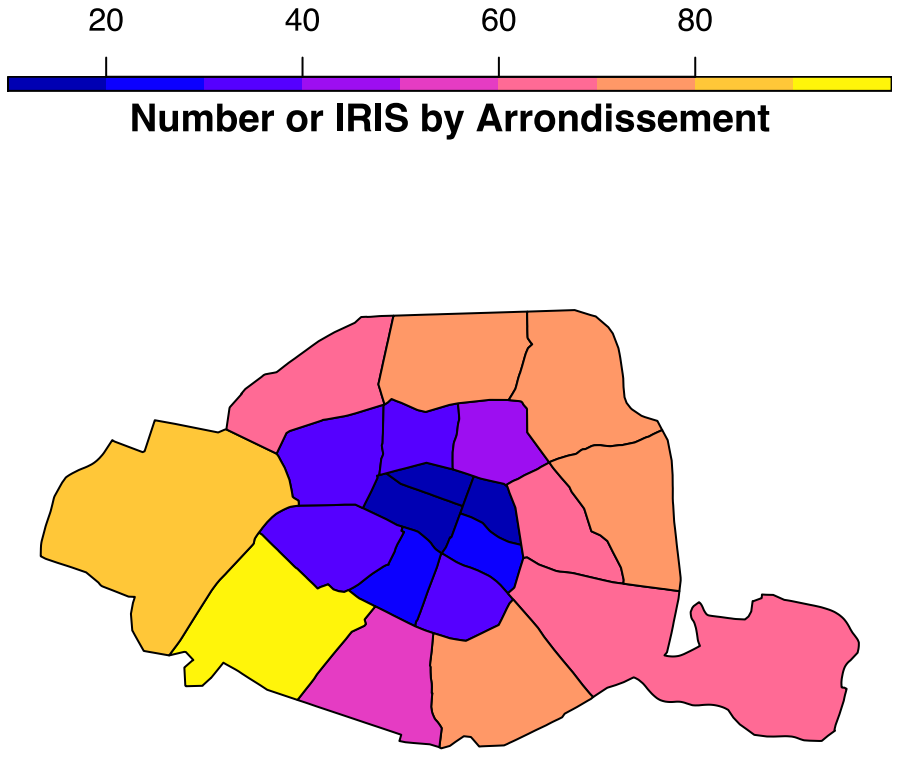
- `R` is very strong with spatial data. In particular via the `sf` package.
- We can represent *any* shape or geometry.
- Maps are the most obvious example:

```
library(sf)
plot(paris_sh[,"n"],
     main = "Number or IRIS by Arrondissement",key.pos
```

Spatial Data

- R is very strong with spatial data. In particular via the sf package.
- We can represent *any* shape or geometry.
- Maps are the most obvious example:

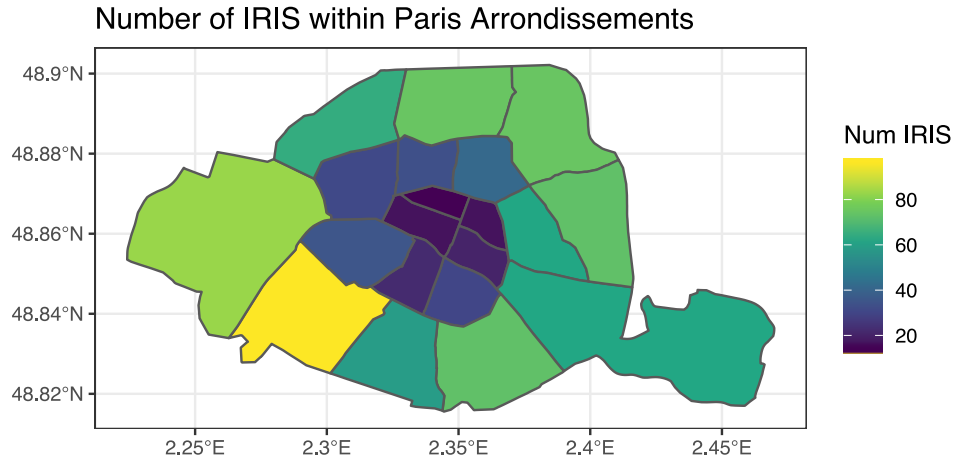
```
library(sf)
plot(paris_sh[,"n"],
     main = "Number or IRIS by Arrondissement",key.pos
```



Spatial Plotting with ggplot

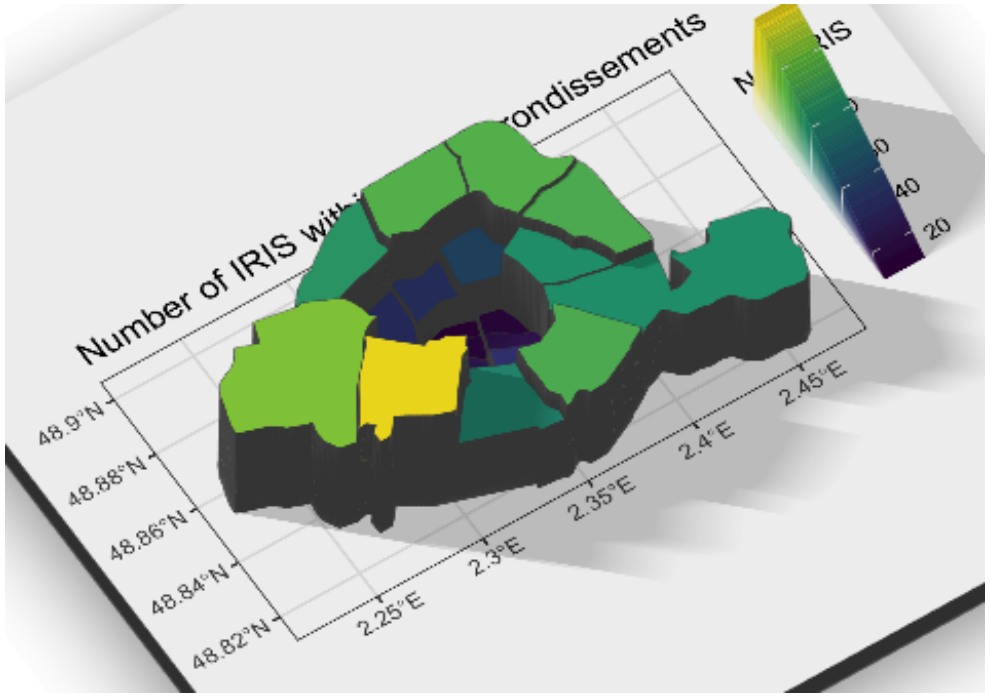
- ggplot can also directly plot spatial data
- here is an example:

```
ggplot(paris_sh) +          # base layer: data
geom_sf(aes(fill = n)) +    # the `sf` geom
scale_fill_viridis_c() +    # greenish fill
theme_bw()                  # simple theme
```



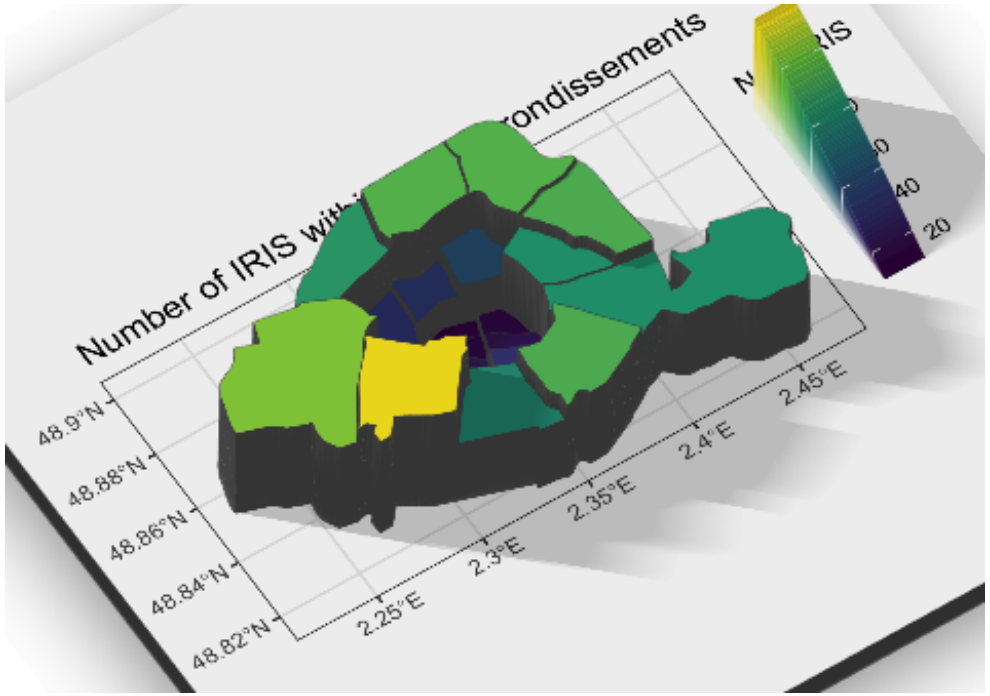
3D ggplots

- A recent *very* cool 🆒 package is `rayshader`
- Sometimes graphs are better to see in 3D.



3D ggplots

- A recent *very* cool 🆒 package is `rayshader`
- Sometimes graphs are better to see in 3D.



Like, *real* 3D...

R 101: Here Is Where You Start.

Start your `Rstudio`!

First Glossary of Terms

- `R`: a statistical programming language
- `RStudio`: an integrated development environment (IDE) to work with `R`
- *command*: user input (text or numbers) that `R` *understands*.
- *script*: a list of commands collected in a text file, each separated by a new line, to be run one after the other.

R as a Calculator

- You can use the R console like a calculator
- Just type an arithmetic operation after > and hit `Enter`!

R as a Calculator

- You can use the R console like a calculator
- Just type an arithmetic operation after > and hit `Enter`!

- Some basic arithmetic first:

```
4 + 1
```

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

```
8 / 2
```

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

- Great! What about this?

```
log(exp(1))
```

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

```
# by the way: this is a comment! (R disregards it
```

Calculator 2

- We can also do exponents with `^`:

```
x = 2
x^3
```

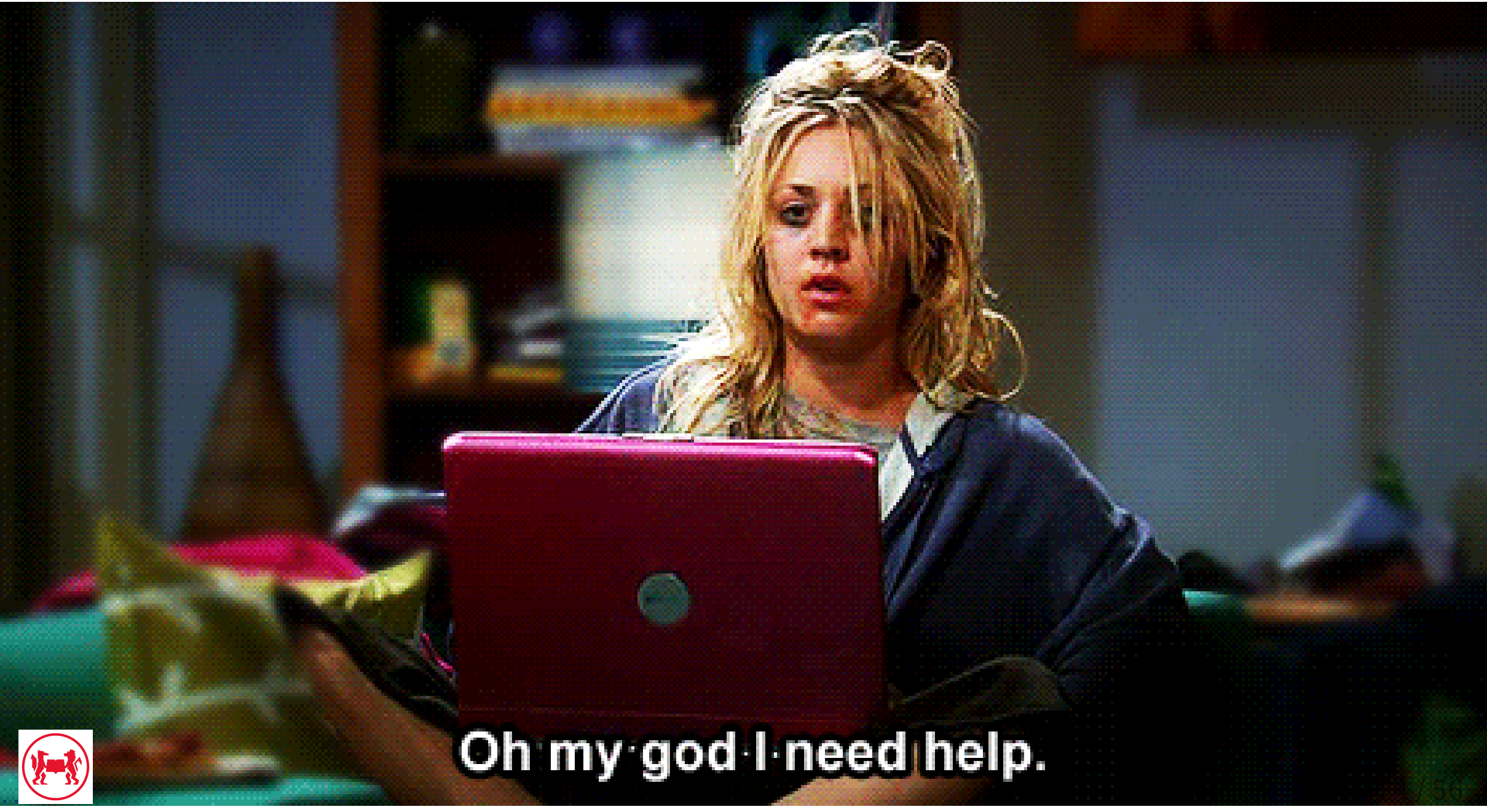
```
## [1] 8
```

- Square roots

```
sqrt(2)
```

```
## [1] 1.414214
```

- and many logarithmic and trigonometric functions.
- many ... What??


Oh my god I need help.

Where to get Help?

- `R` built-in `help`:

```
?log
?sin
?paste
?lm
help(lm)   # help() is equivalent
??plot  # get all help on keyword "plot"
help(ggplot,package="ggplot2")  # show help from a certain package
```

- Help from Humans!
  - stackoverflow.com [SO]
  - Your classroom channel on Slack
  - rstudio forum

HOW to get Help?

1. Describe what you want to do.
2. Describe what you *expect* your code to do.
3. Describe what you your code *does instead*.
    - Provide the entire error message.
4. Provide enough code to *reproduce* your error.
    - You can post post code snippets on slack and SO

R Packages

- `R` users contribute add-on data and functions as *packages*
- Installing packages is easy!

```
install.packages("ggplot2")
```

- To *use* the contents of a packge, we must load it from our library:

```
library(ggplot2)
```

ScPoEconometrics package

- We wrote an R package for you.
- It's hosted on github.com
- You can install (and frequently update!) from there:

```
if (!require("devtools")) install.packages("devtools")
library(devtools)
install_github(repo = "ScPoEcon/ScPoEconometrics")
```

`ScPoEconometrics` package

- We wrote an `R` package for you.
- It's hosted on github.com
- You can install (and frequently update!) from there:

```
if (!require("devtools")) install.packages("devtools")
library(devtools)
install_github(repo = "ScPoEcon/ScPoEconometrics")
```

- Did it work?

```
library(ScPoEconometrics)
packageVersion("ScPoEconometrics")
```

```
## [1] '0.2.2'
```

Data Types and Data Structures

- Numeric: `1.0, 2.1`
- Integer: `1L`, `2L`, `42L`
- Logical: `TRUE` and `FALSE`
- Character: `"a"`, `"Statistics"`, `"1 plus 2."`
- Categorical or `factor`
- You should read more right here!

Vectors

- What *is* a **vector**?
- The `c` function creates vectors.

```
c(1, 3, 5, 7, 8, 9)
```

```
## [1] 1 3 5 7 8 9
```

- Coercion to unique types:

```
c(42, "Statistics", TRUE)
```

```
## [1] "42"         "Statistics" "TRUE"
```

- Creating a *Range*

```
(y = 1:6)
```

```
## [1] 1 2 3 4 5 6
```

Vectors from Sequences and Repetitions

- `seq` creates a sequence `from` `to` in steps of `by`:

```
seq(from = 1.5, to = 2.1, by = 0.1)
```

```
## [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1
```

- `rep` repeats items:

```
rep("A", times = 10)
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

- They also work in combination:

```
c(x, rep(seq(1, 9, 2), 3), c(1, 2, 3), 42, 2:4)
```

```
##  [1]  2  1  3  5  7  9  1  3  5  7  9  1  3  5  7  9  1  2  3 42  2  3  4
```

YOUR TURN

Task 1

- Create a vector of five ones, i.e.

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

- Notice that the colon operator `a:b` is just short for *construct a sequence from `a` to `b`*. Create a vector the counts down from 10 to 0, i.e. it looks like

```
##  [1] 10  9  8  7  6  5  4  3  2  1  0
```

- Use `rep` to create a vector that looks like this:

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

- Find out (using `help()`, google or whatever) how to get the *length* of a vector in `R`!

Indexing or Subsetting a Vector

- R uses 1-based indexing.
- We use [] to get the value at an index

```
x = c(1, 3, 5, 7, 8, 9)
x[2]
```

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

- Works with vectors of indices:

```
x[c(2,5)]
```

```
## [1] 3 8
```

- And can get *all but* indices:

```
x[-4]
```

```
## [1] 1 3 5 8 9
```

Logical Subsetting

- One can use a vector of `TRUE` and `FALSE` to index:

```
x = c(1, 3, 5, 7, 8, 9)
x[c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)]
```

```
## [1] 1 3 5 8
```

- Let's create a *logical vector* for condition `x > 3`

```
x > 3
```

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

- We can get all values of `x` where `x>3` is `TRUE`:

```
x[ x > 3 ]
```

```
## [1] 5 7 8 9
```

Task 2!

- From the `runif` function get 10 numbers drawn from the uniform distribution and store in `x`.
- get all the elements of `x` larger than 0.3, and store them in `y`.
- using the function `which`, store the *indices* of all of those elements in `iy`.

```
## [1]  1  2  3  4  9 10
```

- Check that `y` and `x[iy]` are identical.

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

Matrix

- A `Matrix` is a two-dimensional Array

```
X = matrix(1:9, nrow = 3, ncol = 3)
X
```

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

- Subsetting needs two indices now:

```
X[1,2]
```

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

```
X[3, ]
```

```
## [1] 3 6 9
```

Matrix Operations

- Let's create two matrices.

```
X = matrix(1:4, 2, 2)
Y = matrix(4:1, 2, 2)
```

- Arithmetic!

```
X * Y # equally for +, - and /
```

```
##      [,1] [,2]
## [1,]    4    6
## [2,]    6    4
```

- But `X * Y` is **not** matrix multiplication. All of above are *element by element* operations.
- Matrix multiplication uses %*%. What is `X %*% Y` for you?

Task 3

- Create a vector containing `1,2,3,4,5` called v.
- Create the (2,5) matrix `m`:

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

- Perform matrix multiplication of `m` with `v`. Use the command `%*%`. What dimension does the output have?

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

- Why does the command `v %*% m` not work?

Lists

- Up to now, all containers were *homogeneous*
- `lists` are more flexible:

```
# works with and without fieldnames
ex_list = list(
  a = c(1, 2, 3, 4),
  b = TRUE,
  c = "Hello!",
  d = diag(2)
)
ex_list
```

```
## $a
## [1] 1 2 3 4
##
## $b
## [1] TRUE
##
## $c
## [1] "Hello!"
##
## $d
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Indexing Lists

- `[]` gets a *sublist*
- `[[]]` gets a list element
- Can index by numerical index or name

```
ex_list[1]
```

```
## $a
## [1] 1 2 3 4
```

```
ex_list[[1]]
```

```
## [1] 1 2 3 4
```

```
ex_list$d
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Task 4

1. Copy and paste the above code for `ex_list` into your R session. Remember that `list` can hold any kind of `R` object. Like...another list! So, create a new list `new_list` that has two fields: a first field called "this" with string content `"is awesome"`, and a second field called "ex_list" that contains `ex_list`.

2. Accessing members is like in a plain list, just with several layers now. Get the element `c` from `ex_list` in `new_list`!

3. Compose a new string out of the first element in `new_list`, the element under label `this`. Use the function `paste` to print the string `R is awesome` to your screen.

data.frame's

`data.frame`'s are like spreadsheets.

```
example_data = data.frame(x = c(1, 3, 5, 7),
                          y = c(rep("Hello", 3), "Goodbye"),
                          z = sample(c(TRUE,FALSE),size=4,replace=TRUE))
example_data
```

```
##   x       y     z
## 1 1   Hello  TRUE
## 2 3   Hello FALSE
## 3 5   Hello  TRUE
## 4 7 Goodbye  TRUE
```

data.frames

- Useful methods for a dataframe:

```
nrow(example_data)
```

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

```
ncol(example_data)
```

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

```
names(example_data)
```

```
## [1] "x" "y" "z"
```

Data on Cars

- The `mtcars` dataset is built-in to `R`.

```
head(mtcars,n=3)  # show first 3 rows
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

- To access one of the variables **as a vector**, we use the `$` operator as in `mtcars$mpg`
- Or we use the column name or index: `mtcars[,"mpg"]` or `mtcars[,1]`

Subsetting `data.frames`

- Subsetting is like with matrices, `[,]`

```
# mpg[row condition, col index]
mtcars[mtcars$mpg > 32, c("cyl", "disp", "wt")]
```

```
##                cyl disp    wt
## Fiat 128         4 78.7 2.200
## Toyota Corolla   4 71.1 1.835
```

- But there is a special function which looks nicer.

```
subset(mtcars, subset = mpg > 32, select = c("cyl", "disp", "wt")]
```

Task 4

1. How many observations are there in `mtcars`?
2. How many variables?
3. What is the average value of `mpg`?
4. What is the average value of `mpg` for cars with more than 4 cylinders, i.e. with `cyl>4`?

Basic Programming

- It's useful for us to review some basics for programming.
- We won't be going very deep here, but it's good for you to know some of this.

Variables

- A variable refers to an *object*.
- Another way to say it is that a variable is a name or a *label* for something:

```
x = 1
y = "roses"
z = function(x){sqrt(x)}
```

- *local* variables are only defined (and hence visible) within a certain area of your code, called a *scope*
- *global* variables are defined everywhere.
- Try to **avoid global variables**.

Control Flow

- We can influence which *branch* our code executes based on
- Whether we follow one branch or another depends on a `condition`.

```
if (condition == TRUE) {
  some R code
} else {
  some other R code
}
```

```
x = 1
y = 3
if (x > y) {  # test if x > y
  # if TRUE
  z = x * y  # assign value to z
  print("x is larger than y")
} else {
  # if FALSE
  z = x + 5 * y  # assign other value to z
  print("x is less than or equal to y")
}
```

```
## [1] "x is less than or equal to y"
```

```
z
```

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

Loops

- This is an example for a loop:

```
for (i in 1:3){  # does not have to be 1:3!
  print(i) # loop body: gets executed each time
  # the value of i changes with each iteration
}
```

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

Loops

- This is an example for a loop:

```
for (i in 1:3){  # does not have to be 1:3!
  print(i) # loop body: gets executed each time
  # the value of i changes with each iteration
}
```

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

- We can iterate over things other than numbers:

```
for (i in c("mangos","bananas","apples")){
  print(paste("I love",i))  # the paste function
}
```

```
## [1] "I love mangos"
## [1] "I love bananas"
## [1] "I love apples"
```

Nested Loops

We often also see *nested* loops, which are just what its name suggests:

```
for (i in 2:3){
  # first nest: for each i
  for (j in c("mangos","bananas","apples")){
    # second nest: for each j
    print(paste("Can I get",i,j,"please?"))
  }
}
```

```
## [1] "Can I get 2 mangos please?"
## [1] "Can I get 2 bananas please?"
## [1] "Can I get 2 apples please?"
## [1] "Can I get 3 mangos please?"
## [1] "Can I get 3 bananas please?"
## [1] "Can I get 3 apples please?"
```

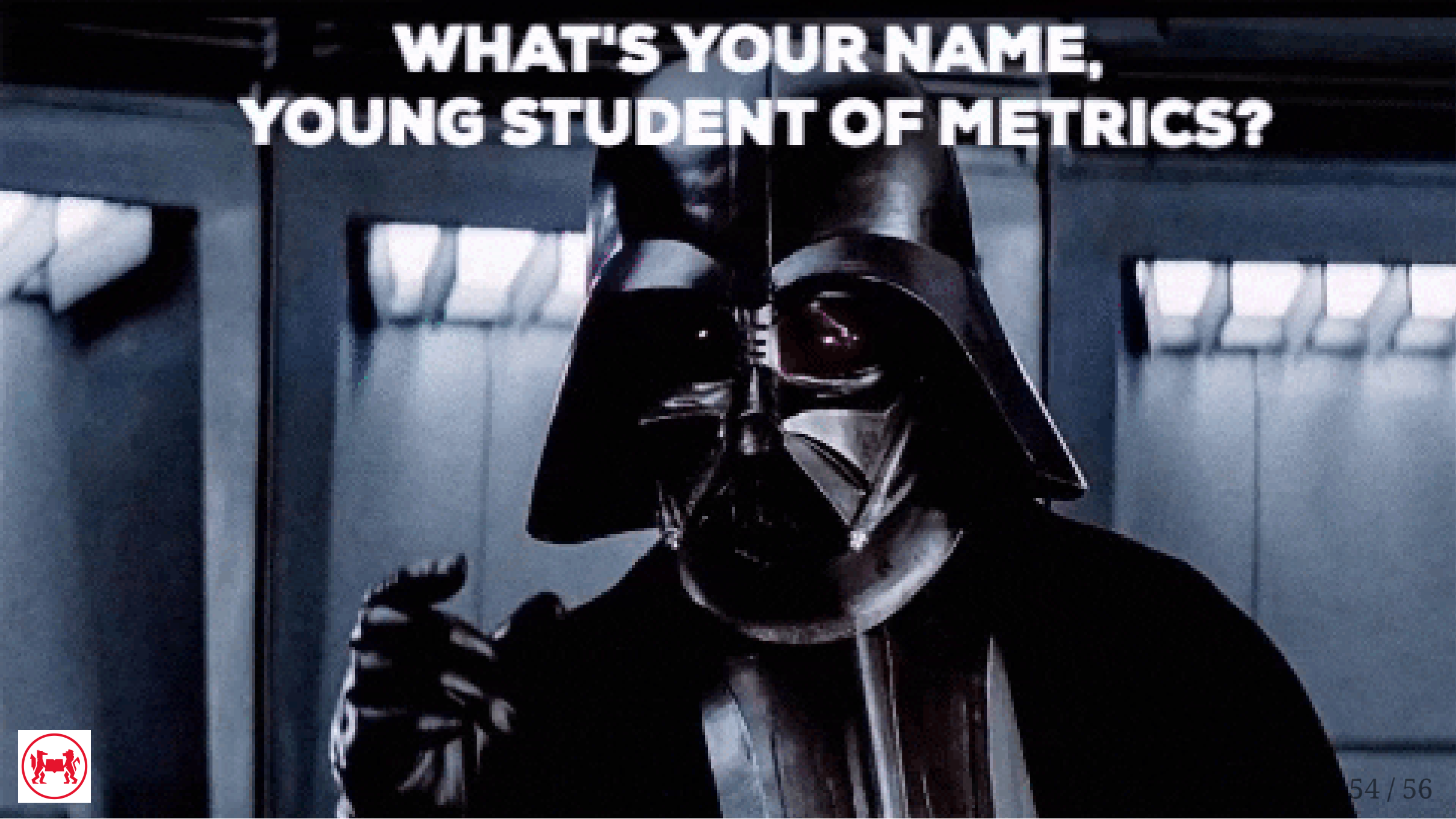
Functions

- Function `say_hello` tells `R` what to do when it you tell it `say_hello()`.

```
say_hello <- function(your_name = "Lord Vader"){
  paste("You R most welcome,",your_name)
}
# we call the function by typing it's name with round brackets
say_hello()
```

```
## [1] "You R most welcome, Lord Vader"
```

- We specified a default argument, but we don't have to.
- Call the function with your name!


WHAT'S YOUR NAME,
YOUNG STUDENT OF METRICS?

Task 5

1. Write a for loop that counts down from 10 to 1, printing the value of the iterator to the screen.

2. Modify that loop to write "i iterations to go" where `i` is the iterator

3. Modify that loop so that each iteration takes roughly one second. You can achieve that by adding the command `Sys.sleep(1)` below the line that prints "i iterations to go".

4. Finally, let's create a function called `ticking_bomb`. it takes no arguments, it's body is the loop you wrote in the preceding question. The only think you should add to the body is a line after the loop finishes, printing "BOOOOM!" with `print("BOOOOM!")`. You can repeatedly redefine the function in the console, and try it out with `ticking_bomb()`.

SciencesPo
DEPARTMENT OF ECONOMICS

END

	florian.oswald@sciencespo.fr
	Slides and Book
	@ScPoEcon
	@ScPoEcon