



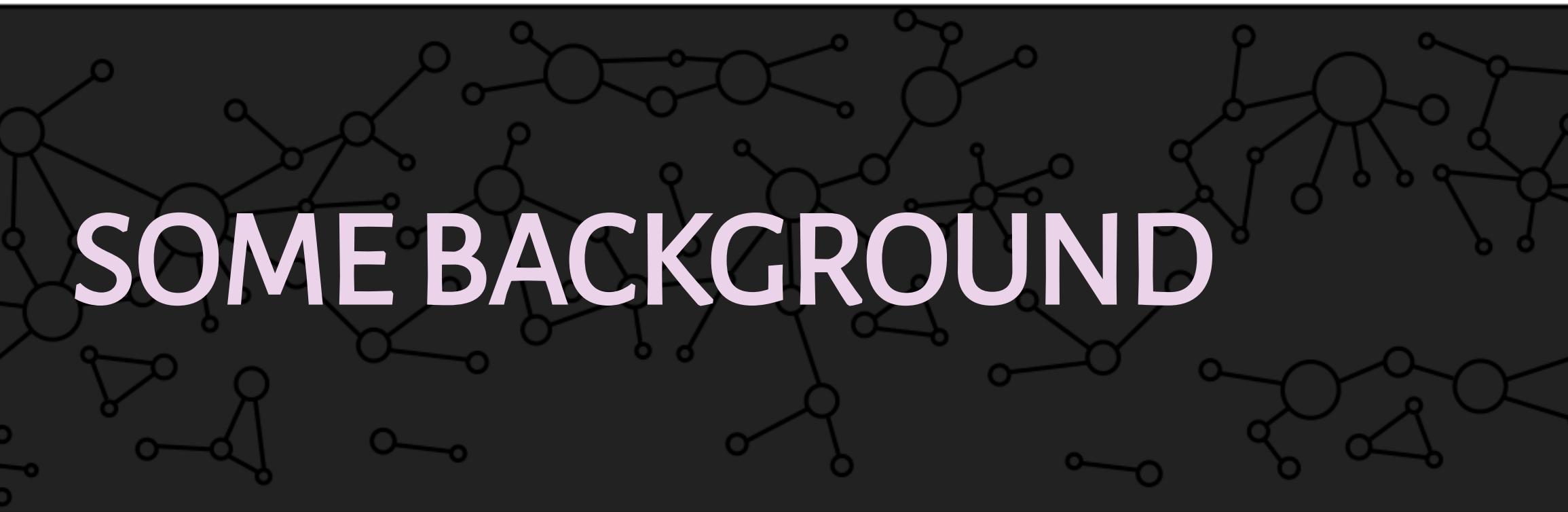
# DATA HANDLING, VALIDATION AND MANIPULATION IN R.

Pablo Gomez

To see this slides online go to:

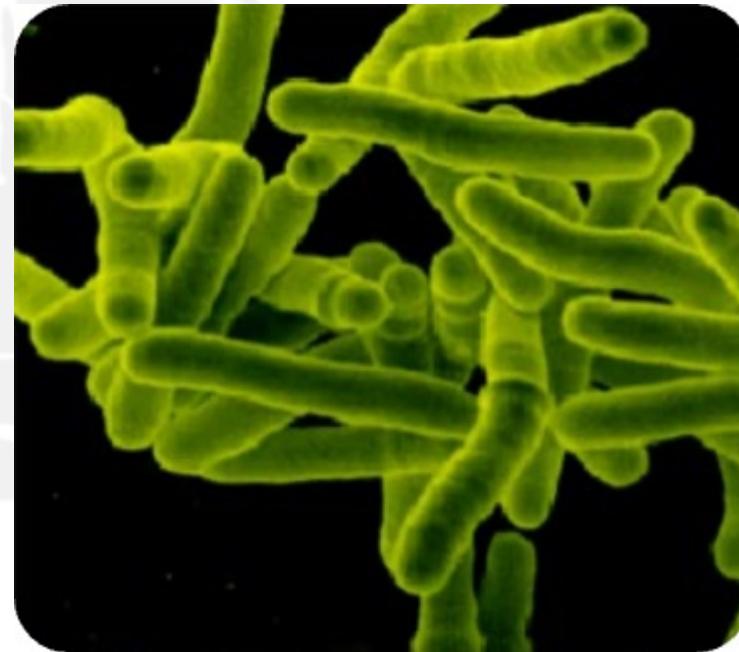
<https://cadms.github.io/mpm207/s/w3.html>

# SOME BACKGROUND



# WHAT IS THE PROBLEM?

Bovine tuberculosis (bTB), caused by *Mycobacterium tuberculosis* complex, particularly *M. bovis* and *M. caprae*, is a zoonotic bacterial disease that affects a wide range of domestic and wild species all over the world.



# WHAT IS THE PROBLEM?

Public  
Health

Economic  
Impact



# WHAT IS THE PROBLEM?

Although bTB eradication has been a major objective for developed countries in the last decades, this goal is still far to be accomplished:

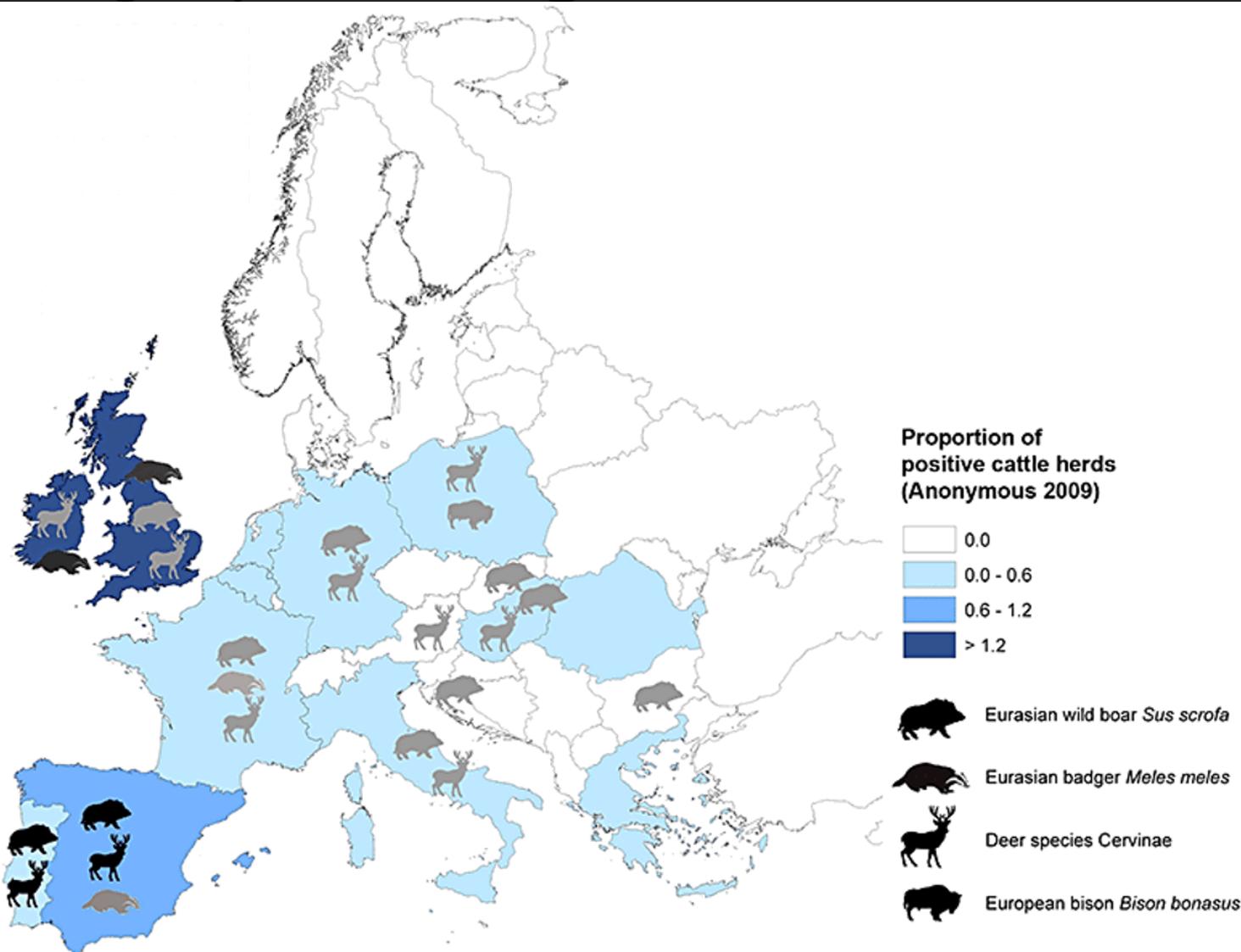
Cattle slaughtered due to Bovine TB



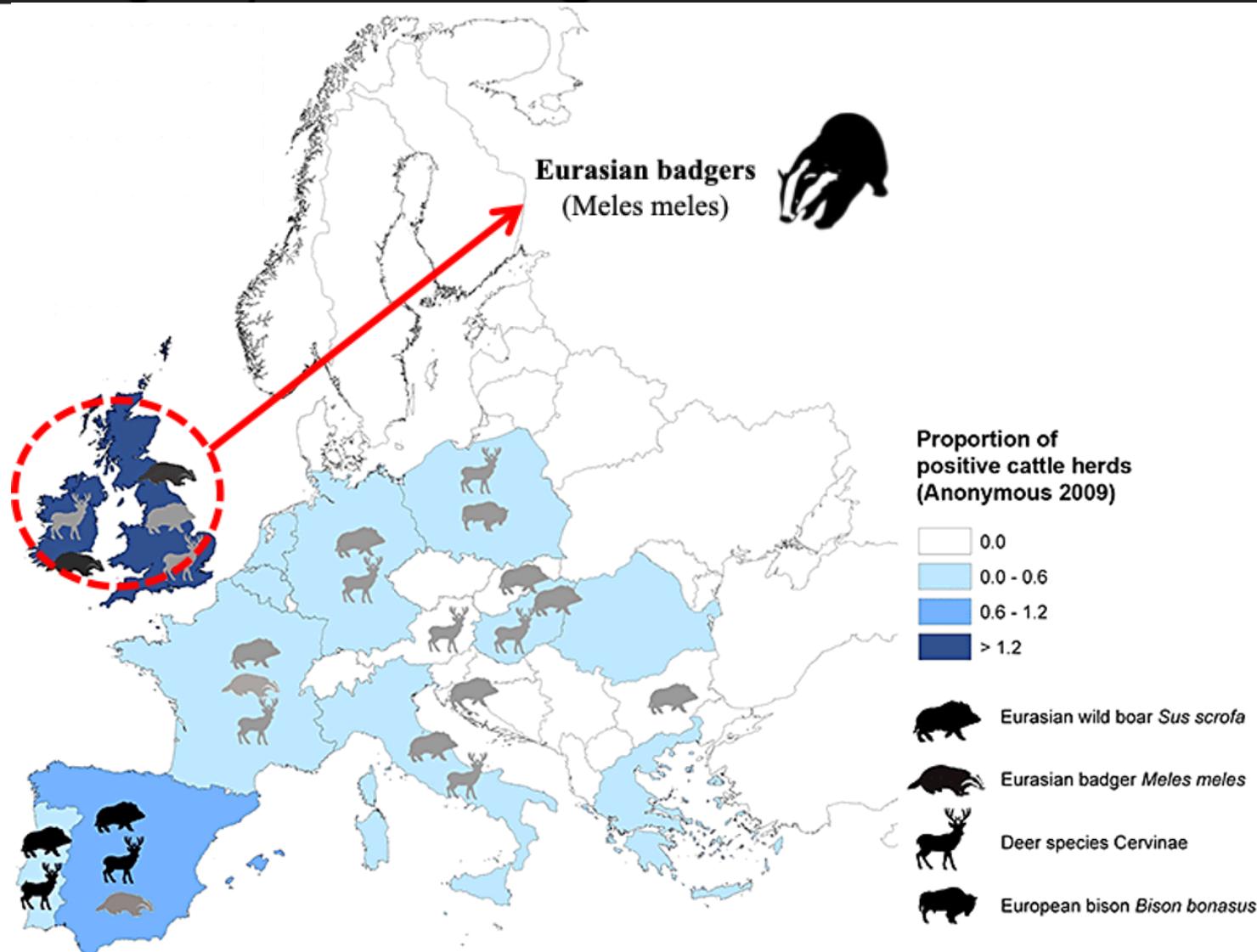
Data for 01/02 compromised due to foot-and-mouth disease outbreak

Source: Defra

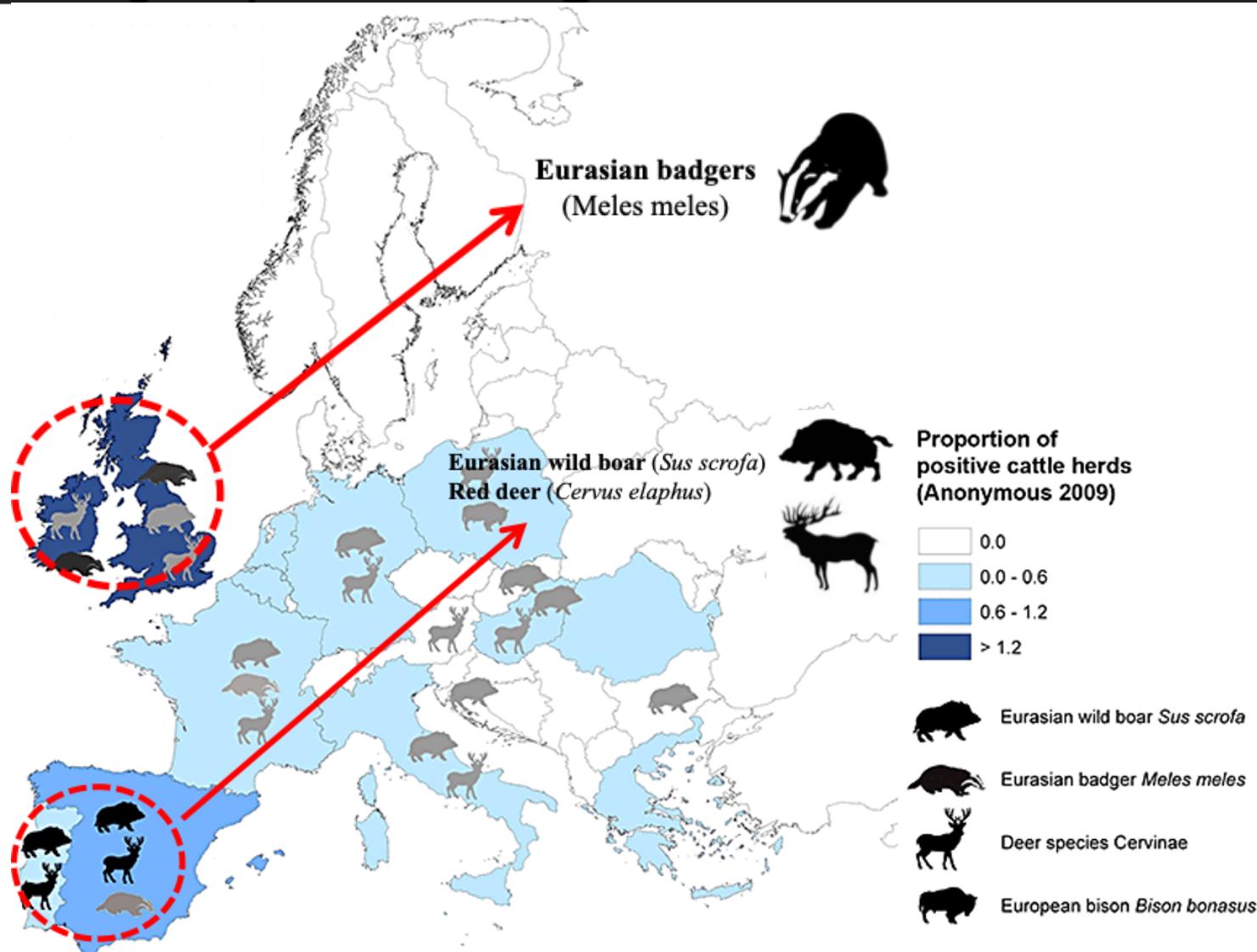
# WHAT IS THE PROBLEM?



# WHAT IS THE PROBLEM?



# WHAT IS THE PROBLEM?



# STUDY AREA



# STUDY AREA

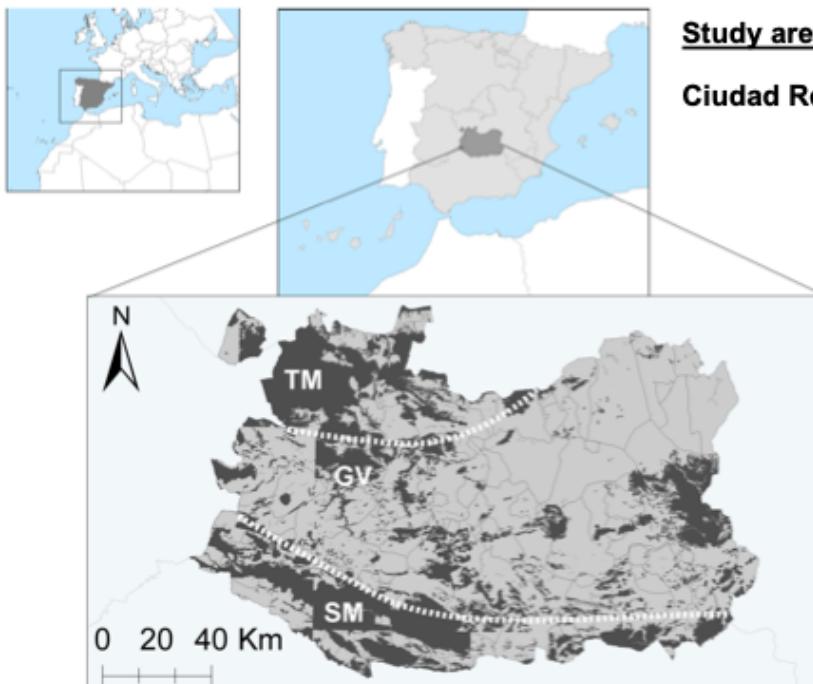


Figure 1. Ciudad Real. TM= Toledo Mountains; SM=Sierra Morena Mountains. GV=Guadiana River Valley.

# DATA

Example: An observational study to evaluate factors associated with TB prev/breakdowns in Ciudad Real (CR), in which we have information at a farm-level (766 farms).

- ID\_farm: Unique identifier of the cattle farm
- Mun: Name of municipality where farm is located
- Com: Name of county (i.e. “Comarca”) where farm is located
- Census: Census or farm size (number of animals on farm)
- Prev: Prevalence of TB on farm the previous year (2004) (numerically perturbed to preserve confidentiality)
- Incid: TB breakdown on farm (2005 campaign, 1= YES, 0=NO)
- Xcoor: X coordinate (UTM system, numerically perturbed to preserve confidentiality)
- Ycoor: Y coordinate (UTM system, numerically perturbed to preserve confidentiality)
- Type: Type of farm (Beef vs Dairy)
- MOV\_year: Average number of cattle incoming shipments per year
- MOV\_cen: Cattle incoming shipments adjusted by farm size (cattle incoming shipments divided by the number of animals on farm \* 100)
- ANIM\_mov: Number of cattle introduced on farm in 2005
- Goat: Goats present on farm (yes=1, no=0)
- Sheep: Sheep present on farm (yes=1, no=0)
- Pig: Pigs present on farm (yes=1, no=0)
- MEANfenced: Mean number of fenced (closed) hunting states in a 3km radius around the farm.

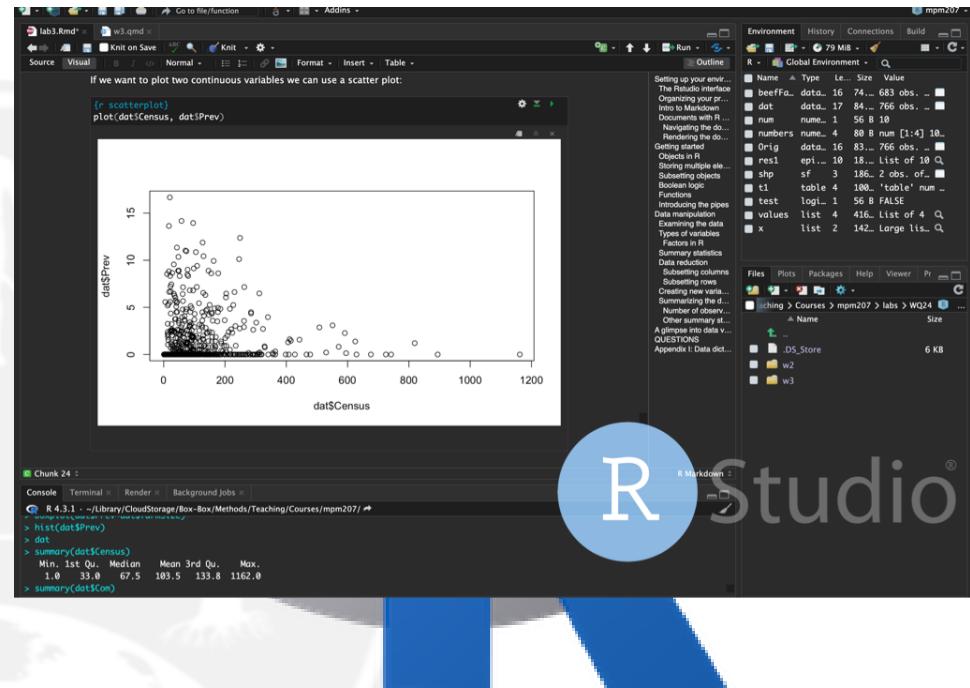
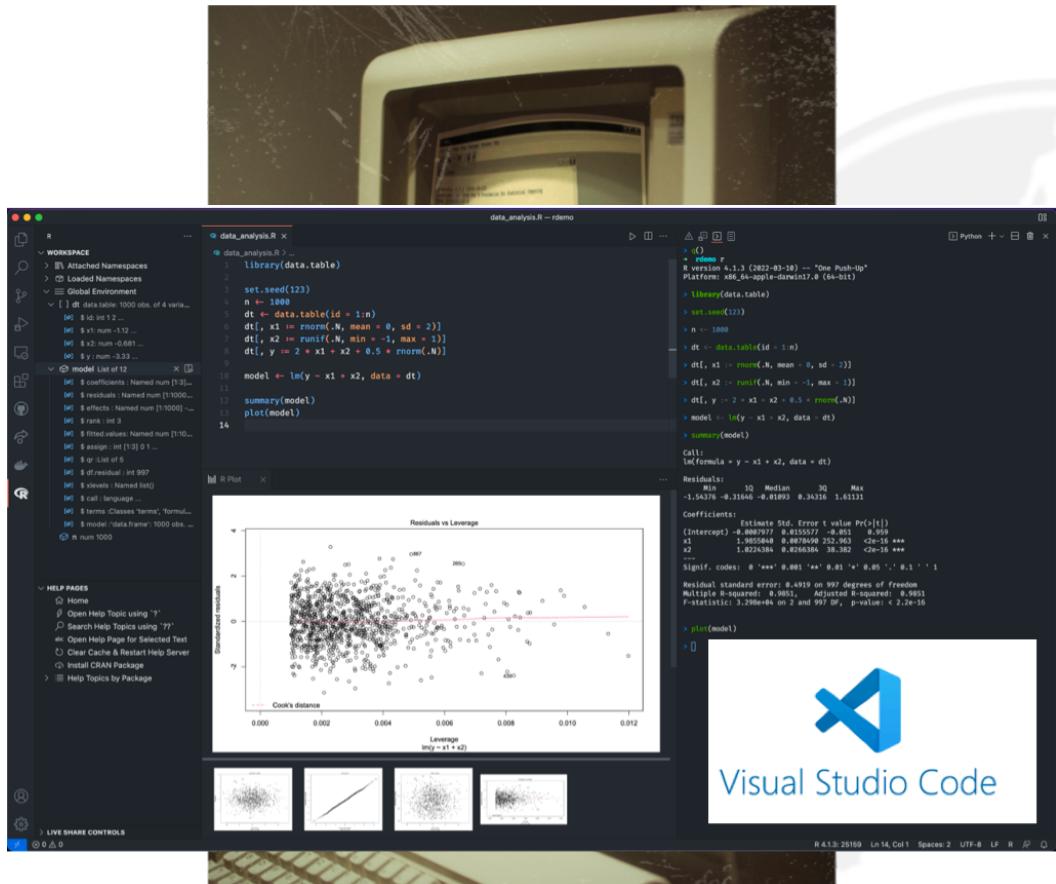


**HOW ARE WE DOING  
THIS?**

# GETTING STARTED WITH R



# GETTING STARTED WITH R



# GETTING STARTED WITH R

The screenshot shows the RStudio interface running on posit.cloud. The top bar displays the time (6:59 p.m.) and date (Mar 23 de ene), battery level (91%), and connection status. The title bar says "posit.cloud". The main window is titled "ShinyWorkshop / ShinyWorkshop". The left pane contains the code editor with an R script named "AppNotebook.Rmd". The code defines a Shiny server logic for a histogram, reactive objects for filtering data based on year, and observeEvent for updating plots. The right pane is divided into several panels: "Environment" (listing variables like "captura", "capturaSp", "dGL", etc.), "History" (empty), "Connections" (empty), "Files" (listing "global.R", "server.R", "ui.R"), "Plots" (empty), "Packages" (empty), and "Help" (empty). The bottom left shows the "Console" tab with the command "runApp('Code/Shiny/MyApp')" and its output. The bottom right features the posit Cloud logo.

```
# Server
# Define server logic required to draw a histogram
server <- function(input, output) {
  ## Reactive objects
  x <- eventReactive(input$filter, {
    p <- vac %>% # data set
    filter(NOM_MUN %in% inputsMun,
           between(YEAR, input$year[1], right = input$year[2])) # filter the data
  }, ignoreNULL = FALSE) # This is for render on load

  observeEvent(x(), {
    showModal(modalDialog("Plots Updated", easyClose = T))
  })

  y <- eventReactive(input$filter, {
    y <- vigilancia %>% # data set
    filter(NOM_MUN %in% inputsMun,
           between(YEAR, input$year[1], right = input$year[2])) # filter the data
  }, ignoreNULL = FALSE)

  z <- eventReactive(input$filter, {
    p <- capturaSp %>% # data set
    filter(NOM_MUN %in% inputsMun,
           between(YEAR, input$year[1], right = input$year[2])) # filter the data
  }, ignoreNULL = FALSE)
}

Reactive objects :
```

Console Terminal Background Jobs

R 4.2.3 /cloud/project/Code/Shiny/MyApp/

```
> runApp('Code/Shiny/MyApp')
Reading layer 'MxShp' from data source '/cloud/lib/x86_64-pc-linux-gnu-library/4.2/STNet/data/MxShp.shp' using driver 'ESRI Shapefile'
Simple feature collection with 2471 features and 6 fields
Geometry type: MULTIPOLYGON
Dimensions: XY
```

# R STUDIO

1.  Lab.Rmd x  Exploratory.Rmd x

Knit on Save ABC Knit Run Addins Environment History Connections Build Git Tutorial

2.  R ShinyWorkshopEspanol

3.  Global Environment

4.  Files Plots Packages Help Viewer

5.  storage > Box-Box > Tools > Workshops > ShinyWorkshop > ShinyWorkshopEspanol

6.  1. ---  
title: "Tu primer shiny app"  
author: "Pablo Gomez"  
date: "r Sys.Date()"  
output:  
html\_document:  
toc: true  
toc\_float: true  
number\_sections: true  
---  
<!-- Ejercicio 1: Abrir un archivo de Shiny app en R studio (15 minutos) --&gt;<br/>```{r setup, include=FALSE}  
knitr::opts\_chunk\$set(warning = F, message = F)  
library(dplyr); library(ggplot2); library(Pabloverse)  
```  
  
En este ejercicio vamos a crear nuestra primera shiny app.  
  
Objetivos:  
- Familiarizarse con los procedimientos para crear una shiny app  
- Identificar la estructura basica de una shiny app  
-

7.  2. R 4.0.2 ~~/Library/CloudStorage/Box-Box/Tools/Workshops/ShinyWorkshop/ShinyWorkshopEspanol/

8.  3. 4. 5. 6. 7. 8.

9.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.

10.  11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.

11.  12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.

12.  13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23.

13.  14. 15. 16. 17. 18. 19. 20. 21. 22. 23.

14.  15. 16. 17. 18. 19. 20. 21. 22. 23.

15.  16. 17. 18. 19. 20. 21. 22. 23.

16.  17. 18. 19. 20. 21. 22. 23.

17.  18. 19. 20. 21. 22. 23.

18.  19. 20. 21. 22. 23.

19.  20. 21. 22. 23.

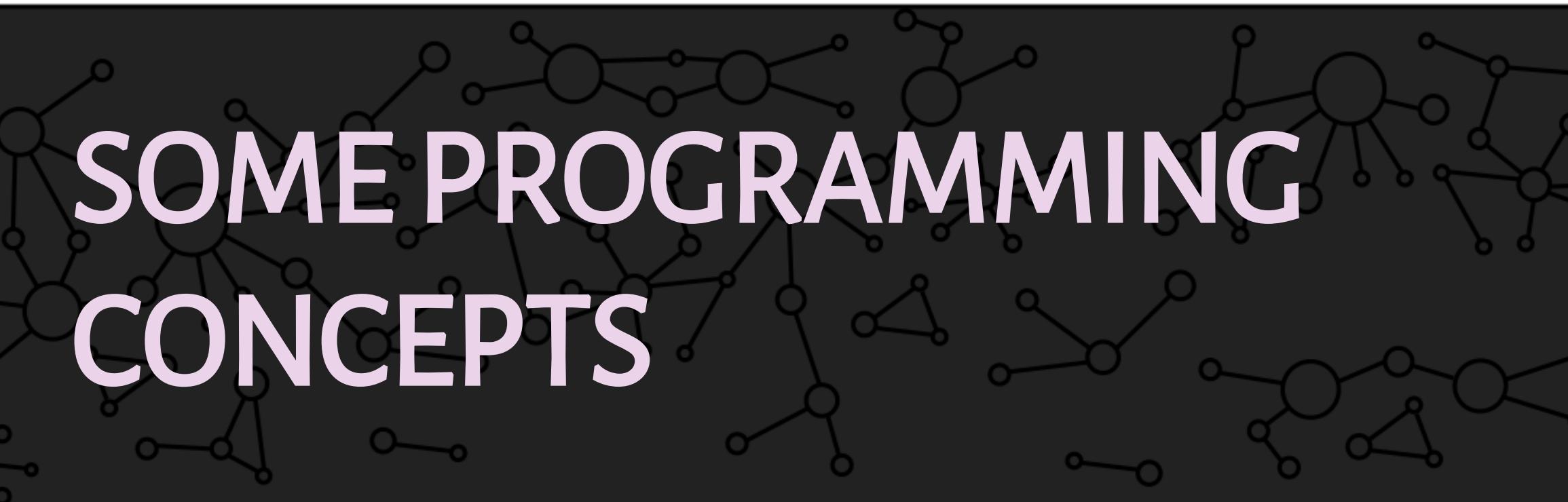
20.  21. 22. 23.

21.  22. 23.

22.  23.

23. 

# SOME PROGRAMMING CONCEPTS



# COMMENTS

COMMENT AS MUCH AS POSSIBLE!

```
1 # This is a comment in R it will be only for the user  
2 This is not a comment and will cause an error
```

What is the difference between line 1 and 2?

# COMMENTS

COMMENT AS MUCH AS POSSIBLE!

```
1 # This is a comment in R it will be only for the user  
2 This is not a comment and will cause an error
```

What is the difference between line 1 and 2?

YES! the **#** character will make everything after it a comment in that line of code

# COMMENTS

COMMENT AS MUCH AS POSSIBLE!

```
1 # This is a comment in R it will be only for the user  
2 This is not a comment and will cause an error
```

What is the difference between line 1 and 2?

YES! the **#** character will make everything after it a comment in that line of code

```
1 10 + 10 # Everything after will be a comment  
2 7 + 4
```

# OPERATORS

Operators are characters with a specific function in R for example

```
1 3 + 3 # this is a sum operator  
[1] 6  
  
1 3 - 2 # this is a subtract operator  
[1] 1  
  
1 4 * 4 # This is a multiplication  
[1] 16
```

# OPERATORS

Operators are characters with a specific function in R for example

```
1 3 + 3 # this is a sum operator  
[1] 6  
  
1 3 - 2 # this is a subtract operator  
[1] 1  
  
1 4 * 4 # This is a multiplication  
[1] 16
```

Later we will see other kind of operators, but... DONT STRESS about learning everything.

# OBJECTS

Objects in R are containers for information, we can create objects with any names we want that start with a letter

```
1 myNumber <- 4  
2 myResult <- 4 * 5
```

# STORING MULTIPLE ELEMENTS

Using the `c()` function

```
1 x <- c(1, 3, 5) # using the c() function
2 x
[1] 1 3 5
```

# STORING MULTIPLE ELEMENTS

Using the `c()` function

```
1 x <- c(1, 3, 5) # using the c() function
2 x
[1] 1 3 5
```

Using the `list()` function

```
1 y <- list(1, 3, 5) # using the list() function
2 y
[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
[1] 5
```

# BOOLEAN LOGIC

```
1 1 == 1 # is it equal?
```

```
[1] TRUE
```

```
1 1 != 1 # is it NOT equal?
```

```
[1] FALSE
```

```
1 1 %in% c(1, 2, 3) # is the number contained in the sequence?
```

```
[1] TRUE
```

# BOOLEAN LOGIC

```
1 1 == 1 # is it equal?
```

```
[1] TRUE
```

```
1 1 != 1 # is it NOT equal?
```

```
[1] FALSE
```

```
1 1 %in% c(1, 2, 3) # is the number contained in the sequence?
```

```
[1] TRUE
```

Notice that we are using operators to make the comparisons

# FUNCTIONS

Functions are a special kind of object. Functions are objects that require arguments, the arguments needs to be inside parentheses.

```
1 # create a sequence of numbers
2 seq(
3   from = 0, # Starting number
4   to = 80, # Ending number
5   by = 20 # number increment of the sequence
6 )
```

```
[1] 0 20 40 60 80
```

# FUNCTIONS

Functions are a special kind of object. Functions are objects that require arguments, the arguments needs to be inside parentheses.

```
1 # create a sequence of numbers
2 seq(
3   from = 0, # Starting number
4   to = 80, # Ending number
5   by = 20 # number increment of the sequence
6 )
```

```
[1] 0 20 40 60 80
```

Notice that the arguments are named in the function, the arguments in the function **seq( )** function are **from, to, by**.

# FUNCTIONS

Functions are a special kind of object. Functions are objects that require arguments, the arguments needs to be inside parentheses.

```
1 # create a sequence of numbers
2 seq(
3   from = 0, # Starting number
4   to = 80, # Ending number
5   by = 20 # number increment of the sequence
6 )
```

```
[1] 0 20 40 60 80
```

Notice that the arguments are named in the function, the arguments in the function **seq( )** function are **from, to, by**.

We can create our own functions, which we will talk more about in the labs

# VARIABLES IN R



# VARIABLES IN R

- *numeric*, continuous numeric variables WITH any decimal values.  
For example: KG of product imported, probability of an event happening.

# VARIABLES IN R

- *numeric*, continuous numeric variables WITH any decimal values.  
For example: KG of product imported, probability of an event happening.
- *integer*, Whole numbers WITHOUT decimal values. For example:  
Number of animals, number of shipments, etc..

# VARIABLES IN R

- *numeric*, continuous numeric variables WITH any decimal values.  
For example: KG of product imported, probability of an event happening.
- *integer*, Whole numbers WITHOUT decimal values. For example:  
Number of animals, number of shipments, etc..
- *character*, Alphanumeric variables. For example: name of a region, name of a disease, farm ID.

# VARIABLES IN R

- *numeric*, continuous numeric variables WITH any decimal values.  
For example: KG of product imported, probability of an event happening.
- *integer*, Whole numbers WITHOUT decimal values. For example:  
Number of animals, number of shipments, etc..
- *character*, Alphanumeric variables. For example: name of a region, name of a disease, farm ID.
- *factor*, Alphanumeric variable with specific categories or levels.  
For example: type of product imported, type of farm, etc...

# TEST TIME!

```
1 x <- seq(from = 5, to = 23, length.out = 10) # create a sequence of numbers
2 y <- seq(from = 0.1, to = 0.78, length.out = 10) # Create another sequence
3 mean(x*y) # Get the mean of the multiplication
[1] 7.406667
```

# TEST TIME!

```
1 x <- seq(from = 5, to = 23, length.out = 10) # create a sequence of numbers
2 y <- seq(from = 0.1, to = 0.78, length.out = 10) # Create another sequence
3 mean(x*y) # Get the mean of the multiplication
[1] 7.406667
```

Objects:

- X
- y

Operators:

- \*
- <-
- ==

# TEST TIME!

```
1 x <- seq(from = 5, to = 23, length.out = 10) # create a sequence of numbers  
2 y <- seq(from = 0.1, to = 0.78, length.out = 10) # Create another sequence  
3 mean(x*y) # Get the mean of the multiplication  
[1] 7.406667
```

Objects:

- x
- y

Functions:

- seq()
- mean()

Operators:

- \*
- <-
- =

Arguments:

- from
- to
- lengt.out

# INTRODUCING THE PIPES %>%

*Pipes* (%>%), can connect several functions to an object.

# INTRODUCING THE PIPES %>%

*Pipes* (%>%), can connect several functions to an object.

For example, if we want to execute a function **F1()** followed by another function **F2()** for the object **X**:

# INTRODUCING THE PIPES %>%

*Pipes* (%>%), can connect several functions to an object.

For example, if we want to execute a function **F1()** followed by another function **F2()** for the object **X**:

```
1 F2(F1(x))
```

# INTRODUCING THE PIPES %>%

*Pipes* (%>%), can connect several functions to an object.

For example, if we want to execute a function F1( ) followed by another function F2( ) for the object X:

```
1 F2(F1(x))
```

is equivalent to:

```
1 x %>% F1() %>% F2()
```

# FOR EXAMPLE

$$\sqrt{\sum_{1}^n x}$$

Instead of this:

```
1 sqrt(sum(x))
```

# FOR EXAMPLE

$$\sqrt{\sum_{1}^n x}$$

Instead of this:

```
1 sqrt(sum(x))
```

We can write it like this:

```
1 x %>% sum() %>% sqrt()
```

# FOR EXAMPLE

Instead of this:

```
1 # Get the number of outgoing and incoming shipments
2 Out <- rename(summarise(group_by(mov, id_orig), Outgoing = n()), id = id_or
```

# FOR EXAMPLE

Instead of this:

```
1 # Get the number of outgoing and incoming shipments
2 Out <- rename(summarise(group_by(mov, id_orig), Outgoing = n()), id = id_or
```

We can write this:

```
1 # Get the number of outgoing and incoming shipments
2 Out <- mov %>%
3   group_by(id_orig) %>%
4   summarise(Outgoing = n()) %>%
5   rename(id = id_orig)
```

# FOR EXAMPLE

Instead of this:

```
1 # Get the number of outgoing and incoming shipments
2 Out <- rename(summarise(group_by(mov, id_orig), Outgoing = n()), id = id_or
```

We can write this:

```
1 # Get the number of outgoing and incoming shipments
2 Out <- mov %>% # This is the movement data set
3   group_by(id_orig) %>% # Group by origin
4   summarise(Outgoing = n()) %>% # Count the number of observations
5   rename(id = id_orig) # Rename the variable
```

And we can break down the code easier!

# HOW CAN WE FIND HELP WITH R?

The screenshot shows the RStudio interface with several panes:

- Code Editor (Top Left):** Displays R code related to Tidyverse graphics, including `geom\_bar`.
- Environment (Top Right):** Shows the global environment with objects like IQ, lat, list\_xy, long, and map.
- Console (Bottom Left):** Shows the command `?geom\_bar` being run in the R console.
- Help Viewer (Bottom Right):** Displays the documentation for `geom\_bar` from the ggplot2 package, explaining its two types and usage.

**Code Editor Content (approximate lines 97-99):**

```
97     'html',
97     link = c('slides/1a_Intro.html', '', '',
97     '', '', '1a_Intro_RstudioMarkdown.html',
97     '1b_RandTidyverse.html', '1c_GraphicsI.html')
98 ),
99 # D02 = c('Geographical Information
```

**Help Viewer Content (geom\_bar {ggplot2} Documentation):**

## Bar charts

### Description

There are two types of bar charts: `geom_bar()` and `geom_col()`. `geom_bar()` makes the height of the bar proportional to the number of cases in each group (or if the `weight` aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use `geom_col()` instead. `geom_bar()` uses `stat_count()` by

# HOW CAN WE FIND HELP WITH R?

geom\_path {ggplot2}

R Documentation

## Connect observations

### Description

`geom_path()` connects the observations in the order in which they appear in the data. `geom_line()` connects them in order of the variable on the x axis. `geom_step()` creates a stairstep plot, highlighting exactly when changes occur. The group aesthetic determines which cases are connected together.

### Usage

```
geom_path(  
  mapping = NULL,  
  data = NULL,  
  stat = "identity",  
  position = "identity",  
  ...,  
  lineend = "butt",  
  linejoin = "round",  
  linemitre = 10,  
  arrow = NULL,  
  na.rm = FALSE,  
  show.legend = NA,  
  inherit.aes = TRUE  
)
```

# HOW CAN WE FIND HELP WITH R?

## Arguments

- mapping** Set of aesthetic mappings created by [aes\(\)](#). If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- data** The data to be displayed in this layer. There are three options:  
If `NULL`, the default, the data is inherited from the plot data as specified in the call to [ggplot\(\)](#).  
A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See [fortify\(\)](#) for which variables will be created.  
A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).
- stat** The statistical transformation to use on the data for this layer, either as a `ggproto` Geom subclass or as a string naming the stat stripped of the `stat_` prefix (e.g. "count" rather than "stat\_count")
- position** Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
- ...** Other arguments passed on to [layer\(\)](#). These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

# HOW CAN WE FIND HELP WITH R?

## Examples

### [Run examples](#)

```
# geom_line() is suitable for time series
ggplot(economics, aes(date, unemploy)) + geom_line()
ggplot(economics_long, aes(date, value01, colour = variable)) +
  geom_line()

# You can get a timeseries that run vertically by setting the orientation
ggplot(economics, aes(unemploy, date)) + geom_line(orientation = "y")

# geom_step() is useful when you want to highlight exactly when
# the y value changes
recent <- economics[economics$date > as.Date("2013-01-01"), ]
ggplot(recent, aes(date, unemploy)) + geom_line()
ggplot(recent, aes(date, unemploy)) + geom_step()
```



# HOW CAN WE FIND HELP WITH R?



how can I crop a raster file with a sf object in R?



To crop a raster file with a sf object in R, you can use the `raster::crop()` function along with the `sf::st\_bbox()` function to create a bounding box from the sf object. Here is an example:



```
r                                     Copy code

library(raster)
library(sf)

# Load raster and sf object
raster_file <- raster("path/to/raster.tif")
sf_object <- st_read("path/to/sf_object.shp")

# Create bounding box from sf object
bbox <- st_bbox(sf_object)

# Crop raster using the bounding box
cropped_raster
```

Regenerate response



Send a message.



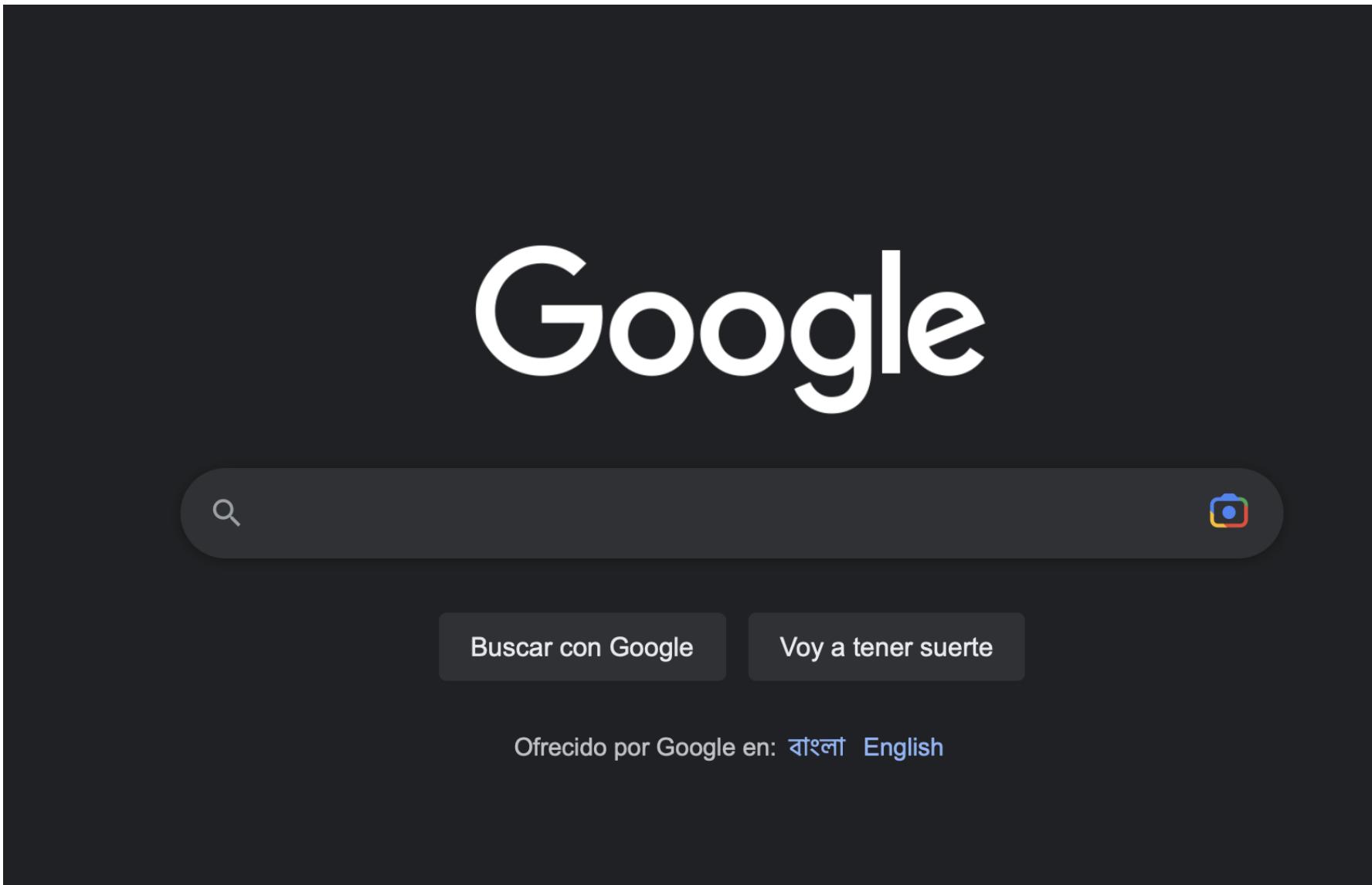
# HOW CAN WE FIND HELP WITH R?

ChatGPT:

**!WARNING: CHAT GPT CAN GIVE INCORRECT INFORMATION !**

- If Chat GPT does not know something, sometimes will make up information (i.e. made up references, name of packages, libraries etc...)
- Make sure to verify the information provided by Chat GPT

# HOW CAN WE FIND HELP WITH R?



# LAB TIME!

<https://cadms.github.io/mpm207/lab3.html>

