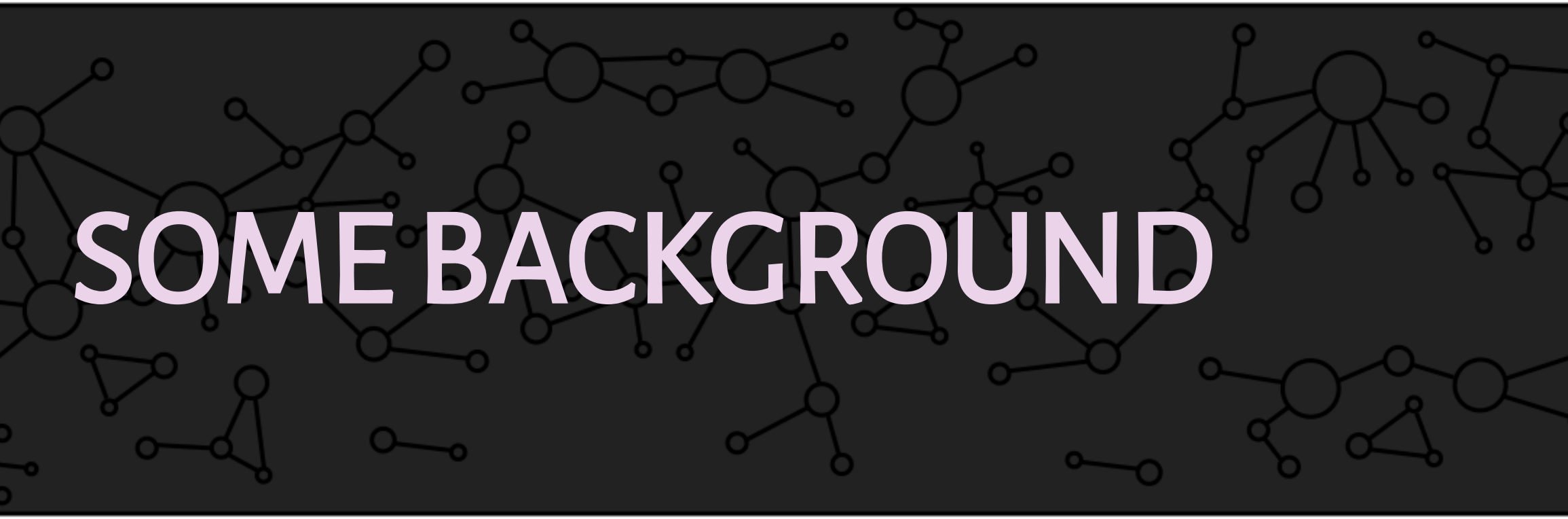
DATA HANDLING, VALIDATION AND MANIPULATION IN R.

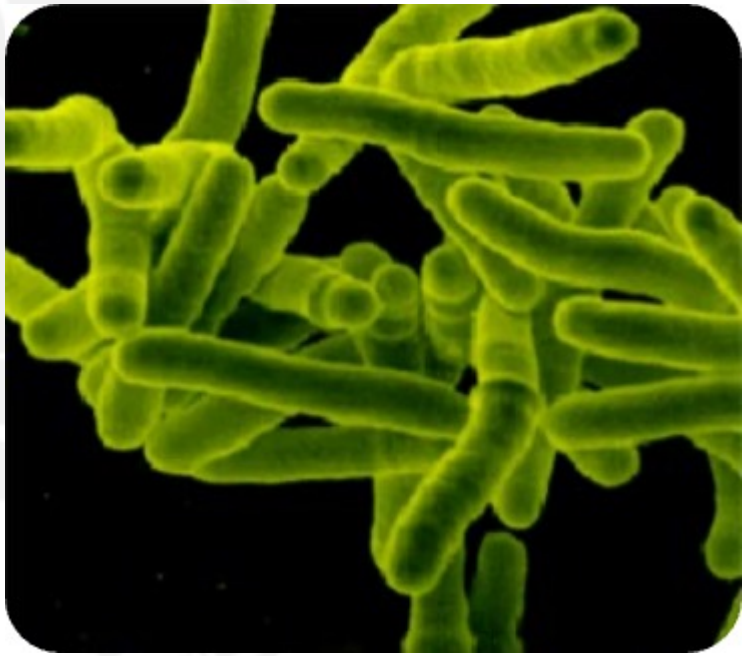
Pablo Gomez

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.



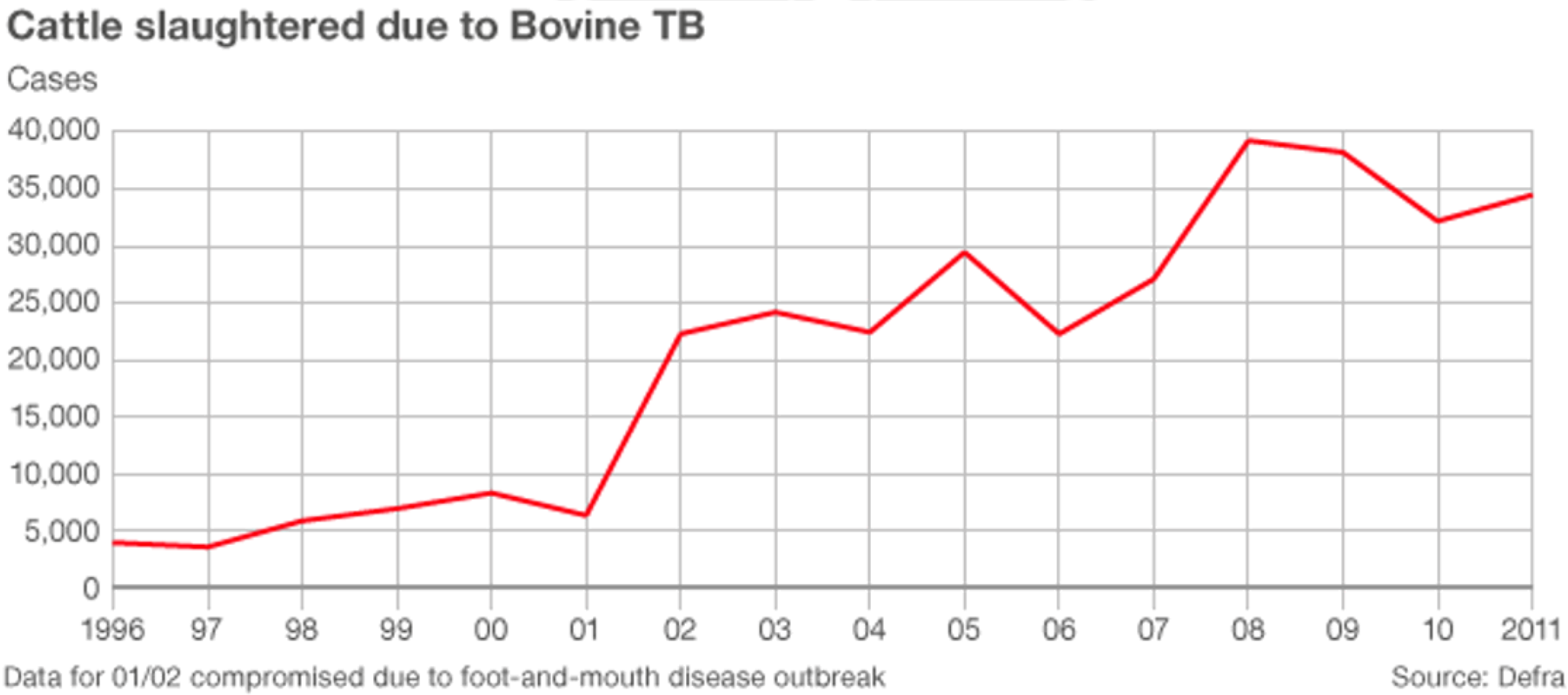
WHAT IS THE PROBLEM?

Public Health

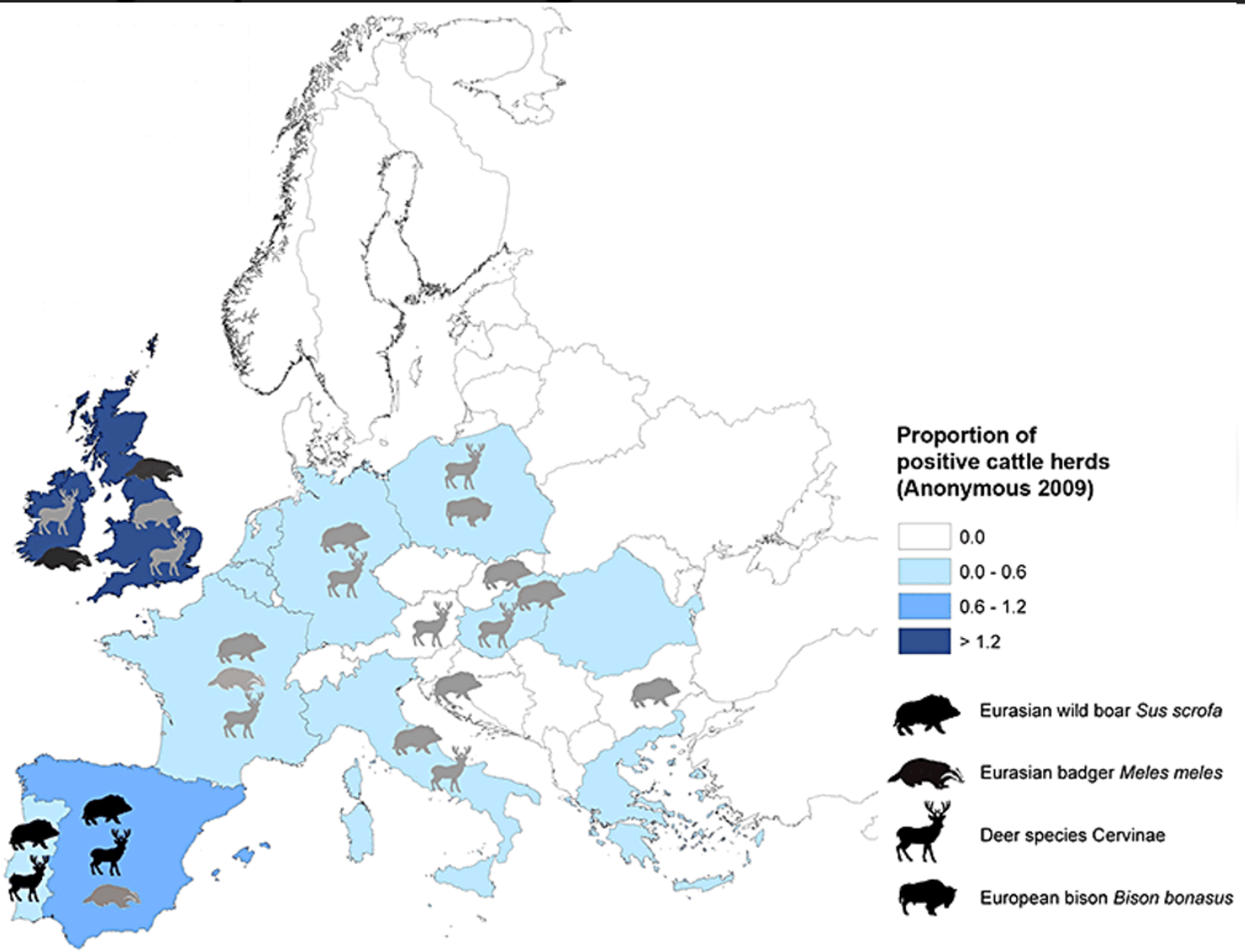
Economic Impact

WHAT IS THE PROBLEM?

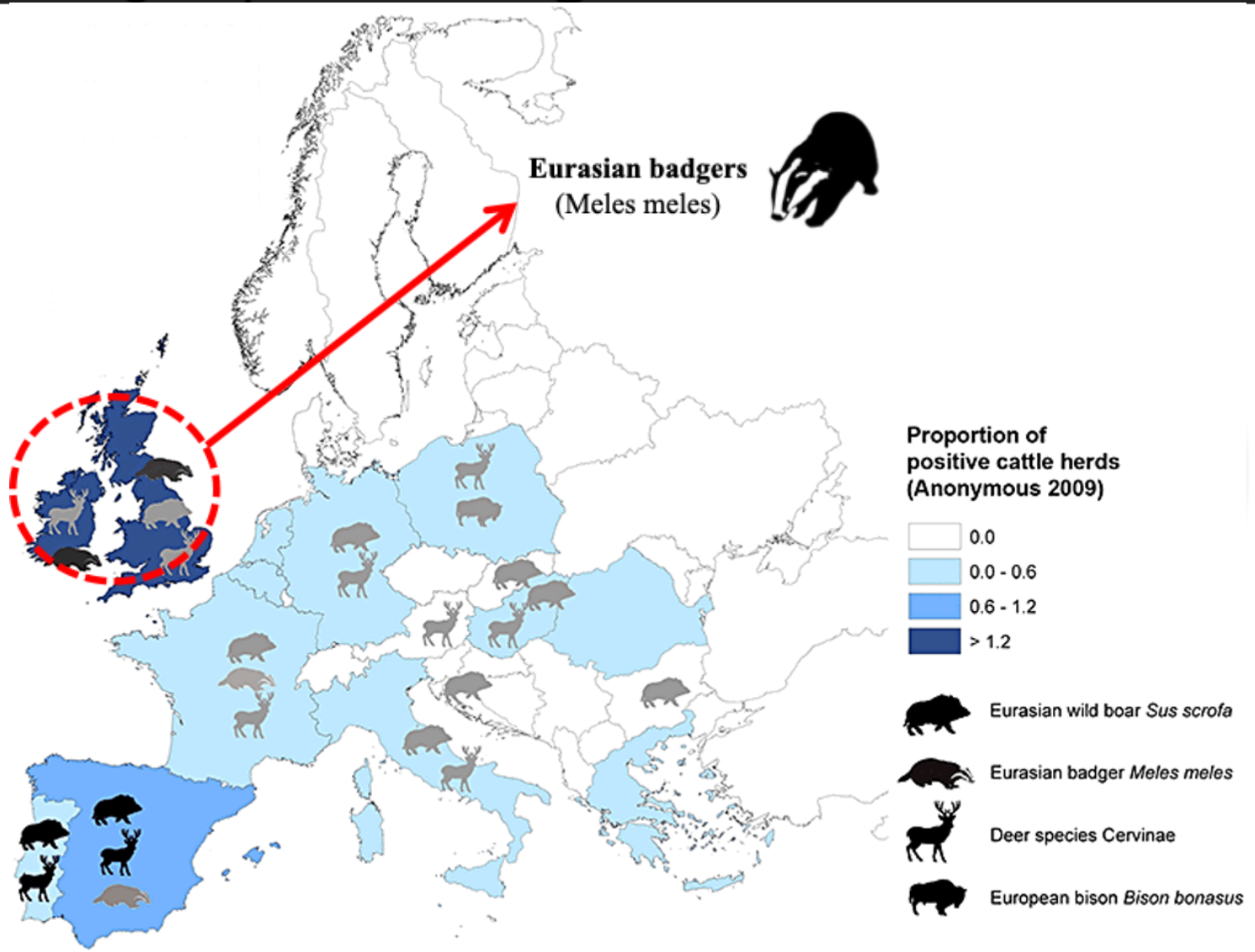
Although bTB eradication has been a major objective for developed countries in the last decades, this goal is still far to be accomplished:



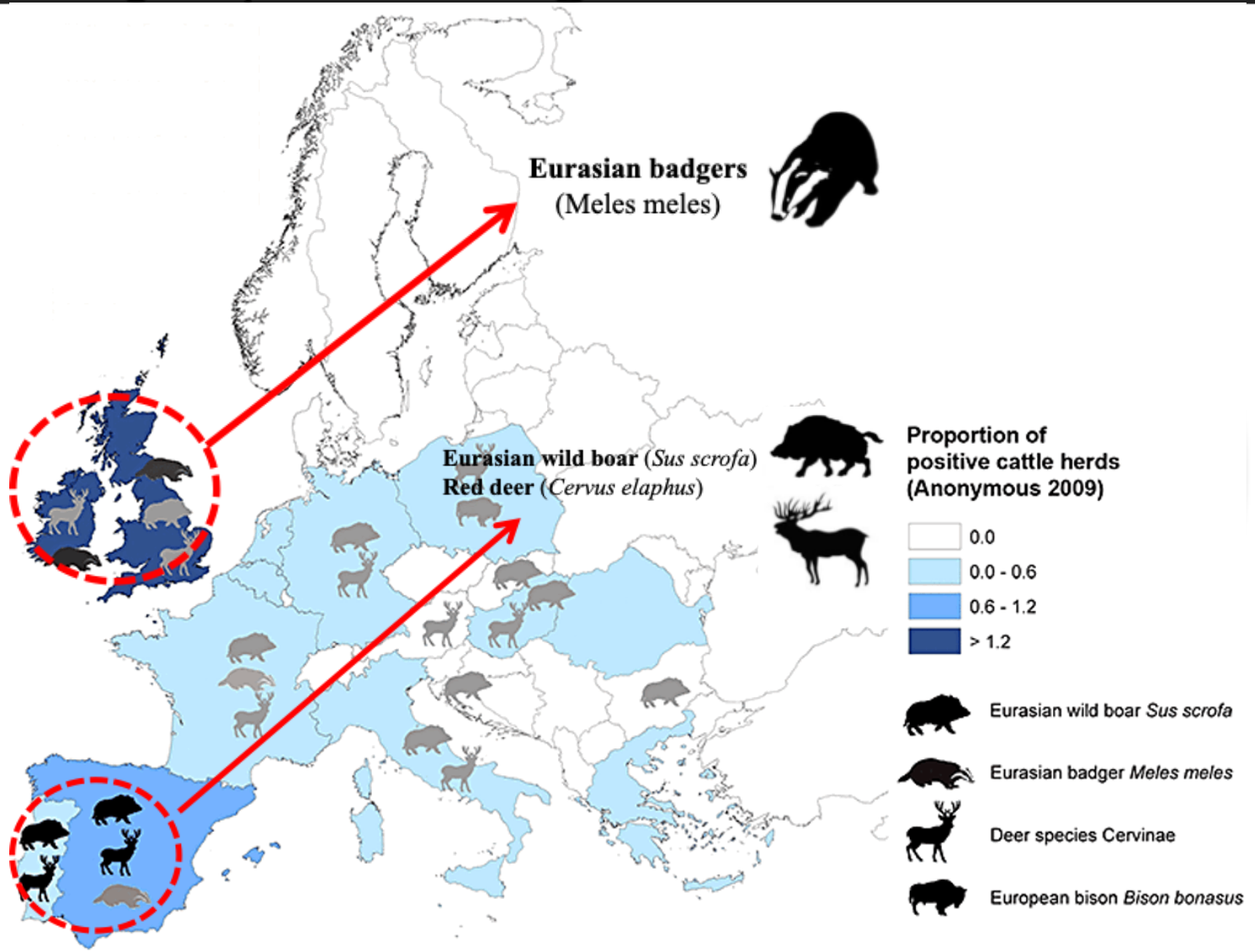
WHAT IS THE PROBLEM?



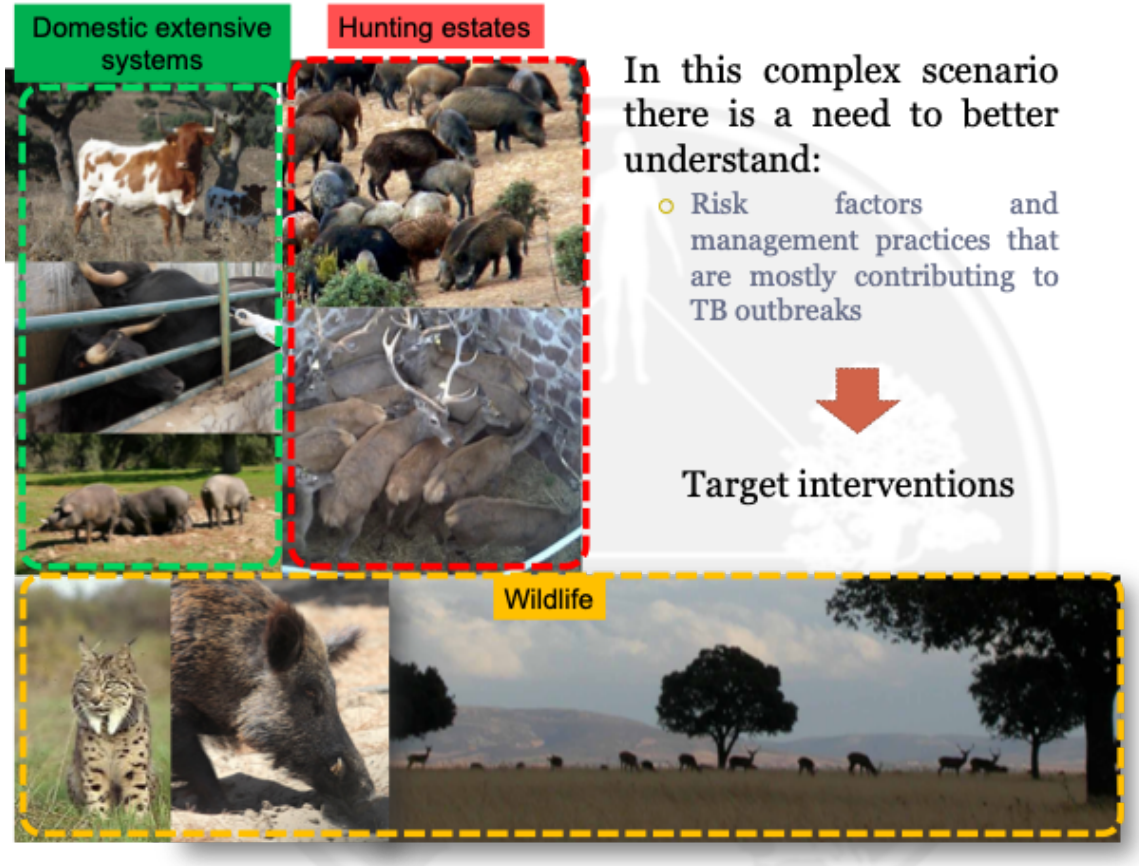
WHAT IS THE PROBLEM?



WHAT IS THE PROBLEM?



STUDY AREA



In this complex scenario there is a need to better understand:

- Risk factors and management practices that are mostly contributing to TB outbreaks

Target interventions

STUDY AREA

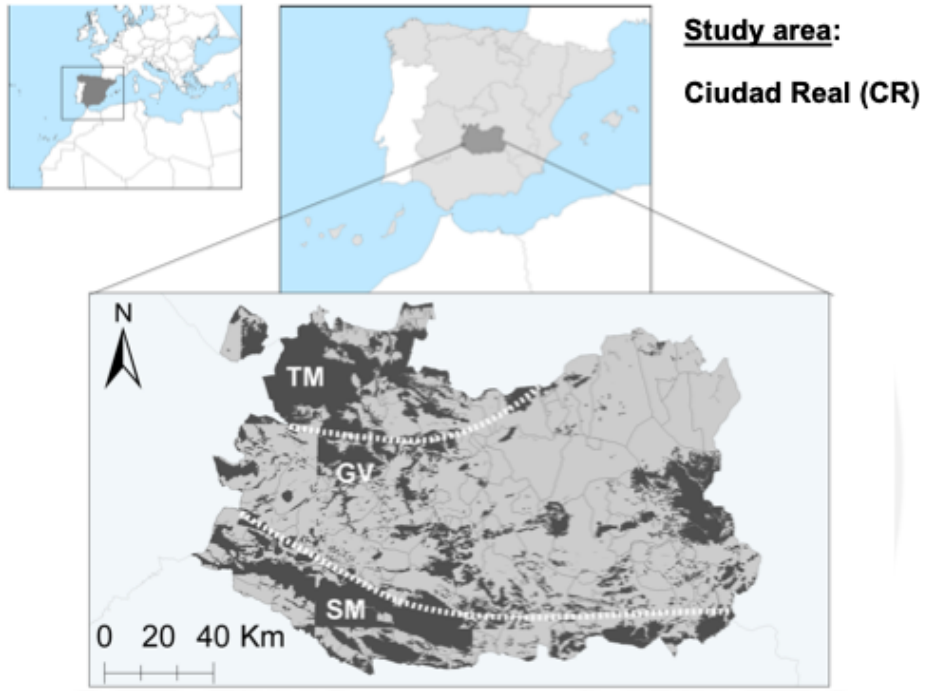


Figure 1. Ciudad Real. TM= Toledo Mountains; SM=Sierra Morena Mountains. GV=Guadiana River Valley.

DATA

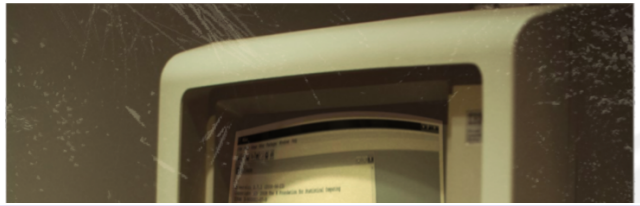
Example: An observational study to evaluate factors associated with TB prev/breakdowns in Ciudad Real (CR), in which we have information at a farm-level (766 farms).

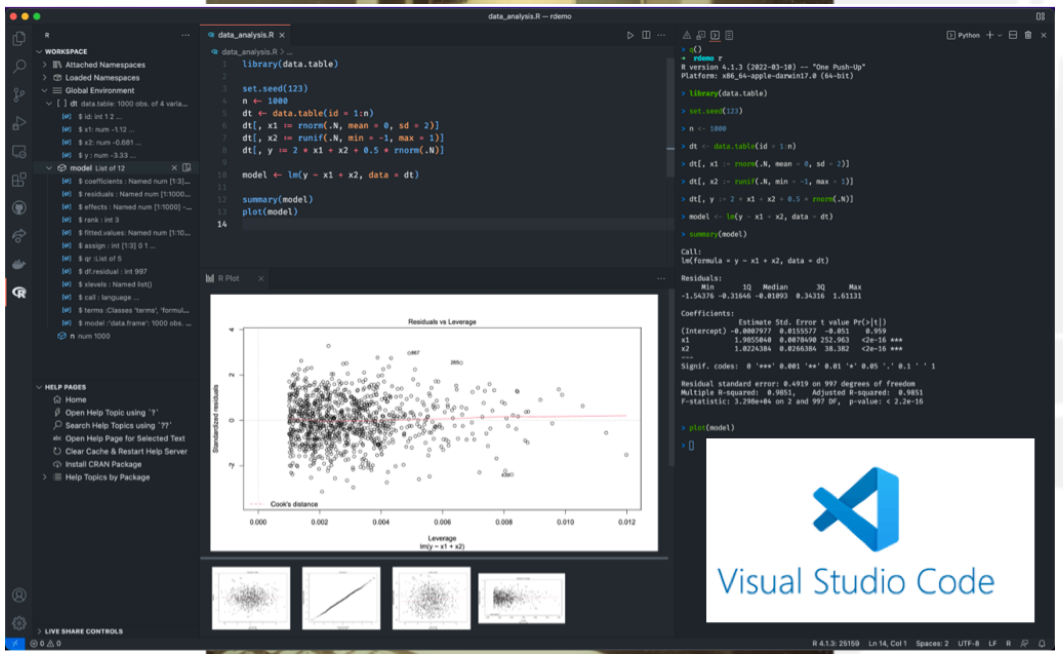
- ID_farm: Unique identifier of the cattle farm
- Mun: Name of municipality where farm is located
- Com: Name of county (i.e. "Comarca") where farm is located
- Census: Census or farm size (number of animals on farm)
- Prev: Prevalence of TB on farm the previous year (2004) (numerically perturbed to 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.

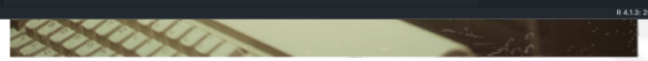
HOW ARE WE DOING THIS?

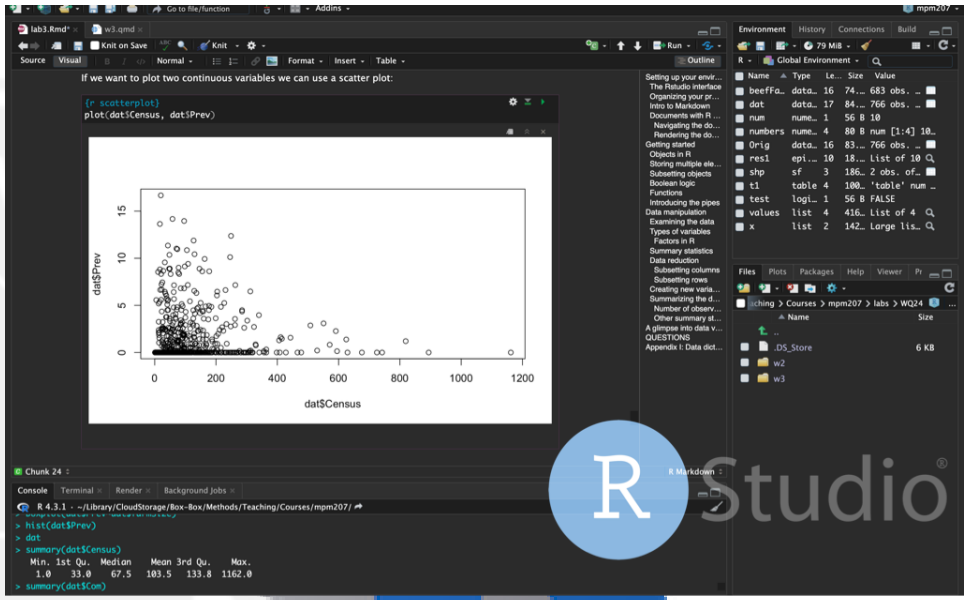
GETTING STARTED WITH R

GETTING STARTED WITH R

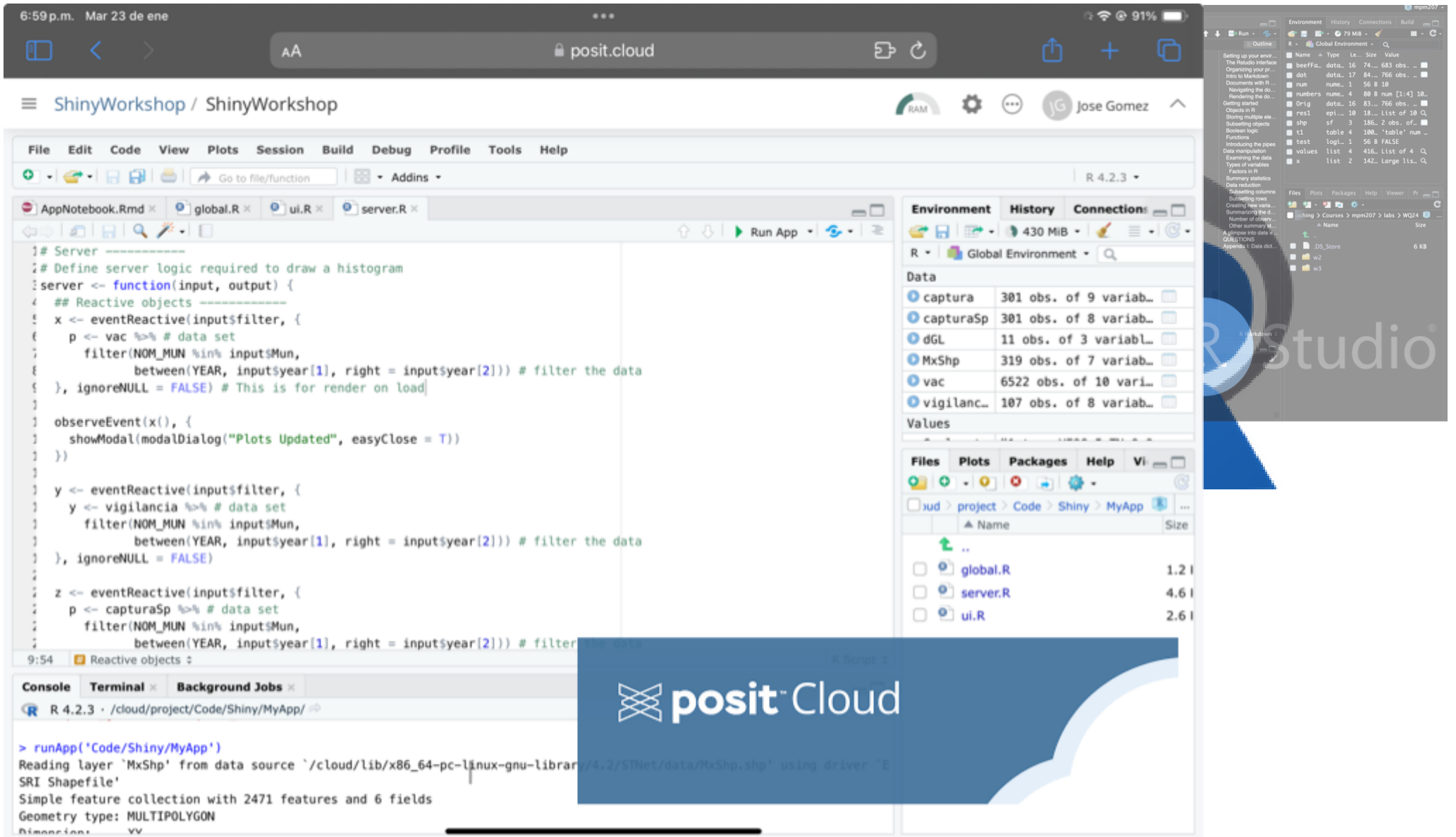




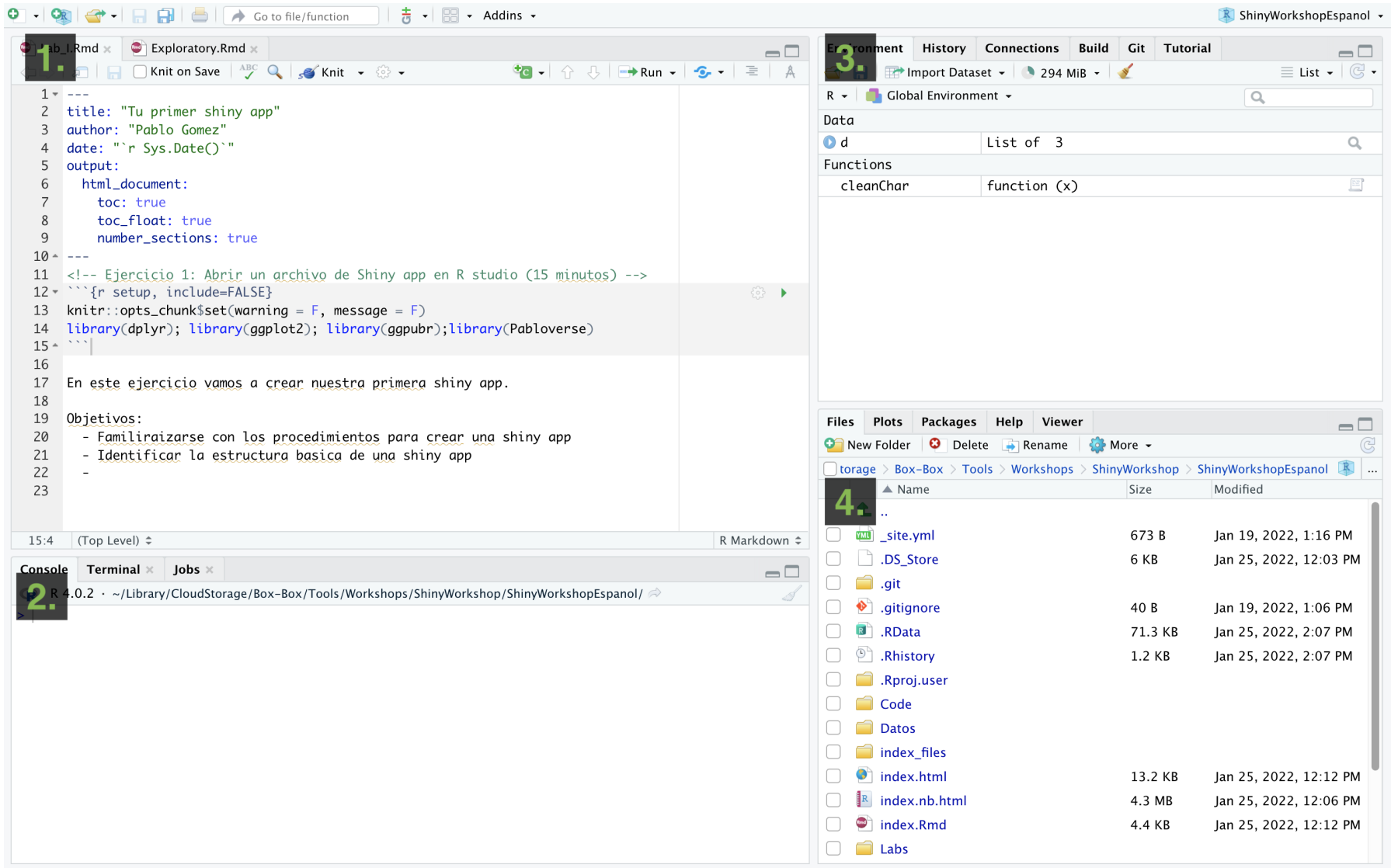




GETTING STARTED WITH R



R STUDIO


Go to file/function
Addins
ShinyWorkshopEspanol
1.
lab_1.Rmd
Exploratory.Rmd
Knit on Save
Knit
Run
---
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) -->
```{r setup, include=FALSE}
knitr::opts_chunk$set(warning = F, message = F)
library(dplyr); library(ggplot2); library(ggpubr);library(Pabloverse)
```
En este ejercicio vamos a crear nuestra primera shiny app.
Objetivos:
- Familiraizarse con los procedimientos para crear una shiny app
- Identificar la estructura basica de una shiny app
-
15:4
(Top Level)
R Markdown
2.
Console
Terminal
Jobs
R 4.0.2 · ~/Library/CloudStorage/Box-Box/Tools/Workshops/ShinyWorkshop/ShinyWorkshopEspanol/
3.
Environment
History
Connections
Build
Git
Tutorial
Import Dataset
294 MiB
List
R
Global Environment
Data
d
List of 3
Functions
cleanChar
function (x)
4.
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
torage
Box-Box
Tools
Workshops
ShinyWorkshop
ShinyWorkshopEspanol
Name
Size
Modified
..
_site.yml
673 B
Jan 19, 2022, 1:16 PM
.DS_Store
6 KB
Jan 25, 2022, 12:03 PM
.git
.gitignore
40 B
Jan 19, 2022, 1:06 PM
.RData
71.3 KB
Jan 25, 2022, 2:07 PM
.Rhistory
1.2 KB
Jan 25, 2022, 2:07 PM
.Rproj.user
Code
Datos
index_files
index.html
13.2 KB
Jan 25, 2022, 12:12 PM
index.nb.html
4.3 MB
Jan 25, 2022, 12:06 PM
index.Rmd
4.4 KB
Jan 25, 2022, 12:12 PM
Labs

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

```
3 + 3 # this is a sum operator
```

```
[1] 6
```

```
3 - 2 # this is a subtract operator
```

```
[1] 1
```

```
4 * 4 # This is a multiplication
```

```
[1] 16
```

OPERATORS

Operators are characters with a specific function in R for example

```
3 + 3 # this is a sum operator
```

```
[1] 6
```

```
3 - 2 # this is a subtract operator
```

```
[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 # is it equal?
```

```
[1] TRUE
```

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

```
[1] FALSE
```

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

```
[1] TRUE
```

BOOLEAN LOGIC

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

```
[1] TRUE
```

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

```
[1] FALSE
```

```
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.

```r
# create a sequence of numbers
seq(
  from = 0, # Starting number
  to = 80, # Ending number
  by = 20 # number increment of the sequence
)
```

```
[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.

```r
# create a sequence of numbers
seq(
  from = 0, # Starting number
  to = 80, # Ending number
  by = 20 # number increment of the sequence
)
```

```
[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.

```
# create a sequence of numbers
seq(
  from = 0, # Starting number
  to = 80, # Ending number
  by = 20 # number increment of the sequence
)
```

```
[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!

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

```
[1] 7.406667
```

TEST TIME!

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

```
[1] 7.406667
```

Objects:

- x
- y

Operators:

- *
- <-
- =

TEST TIME!

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

```
[1] 7.406667
```

Objects:

- x
- y

Operators:

- *
- <-
- =

Functions:

- seq()
- mean()

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:

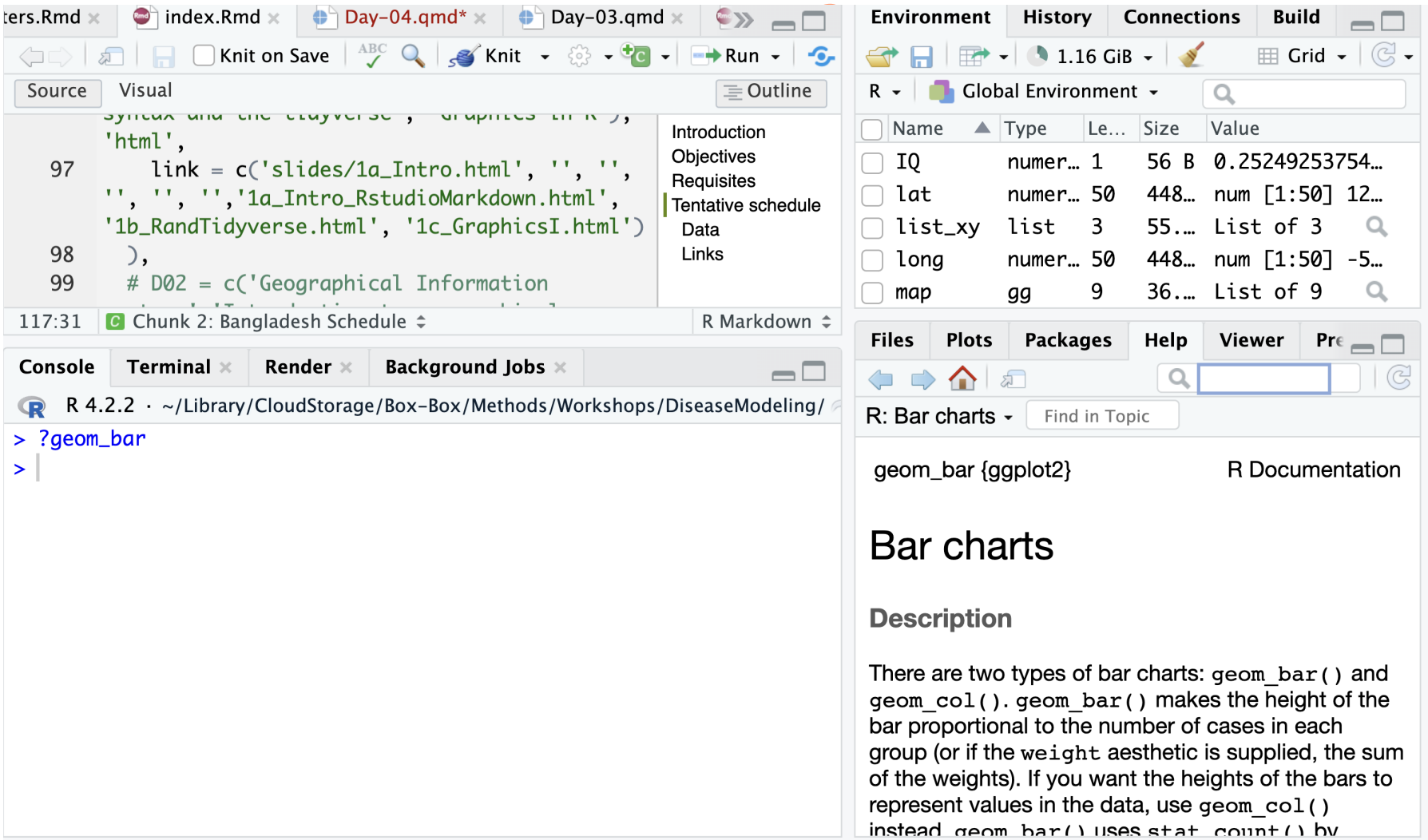
```
1 # Get the number of outgoing and incoming shipments
2 Out <- rename(summarise(group_by(mov, id_orig), Outgoing = n()), id = id_or
```

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?



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

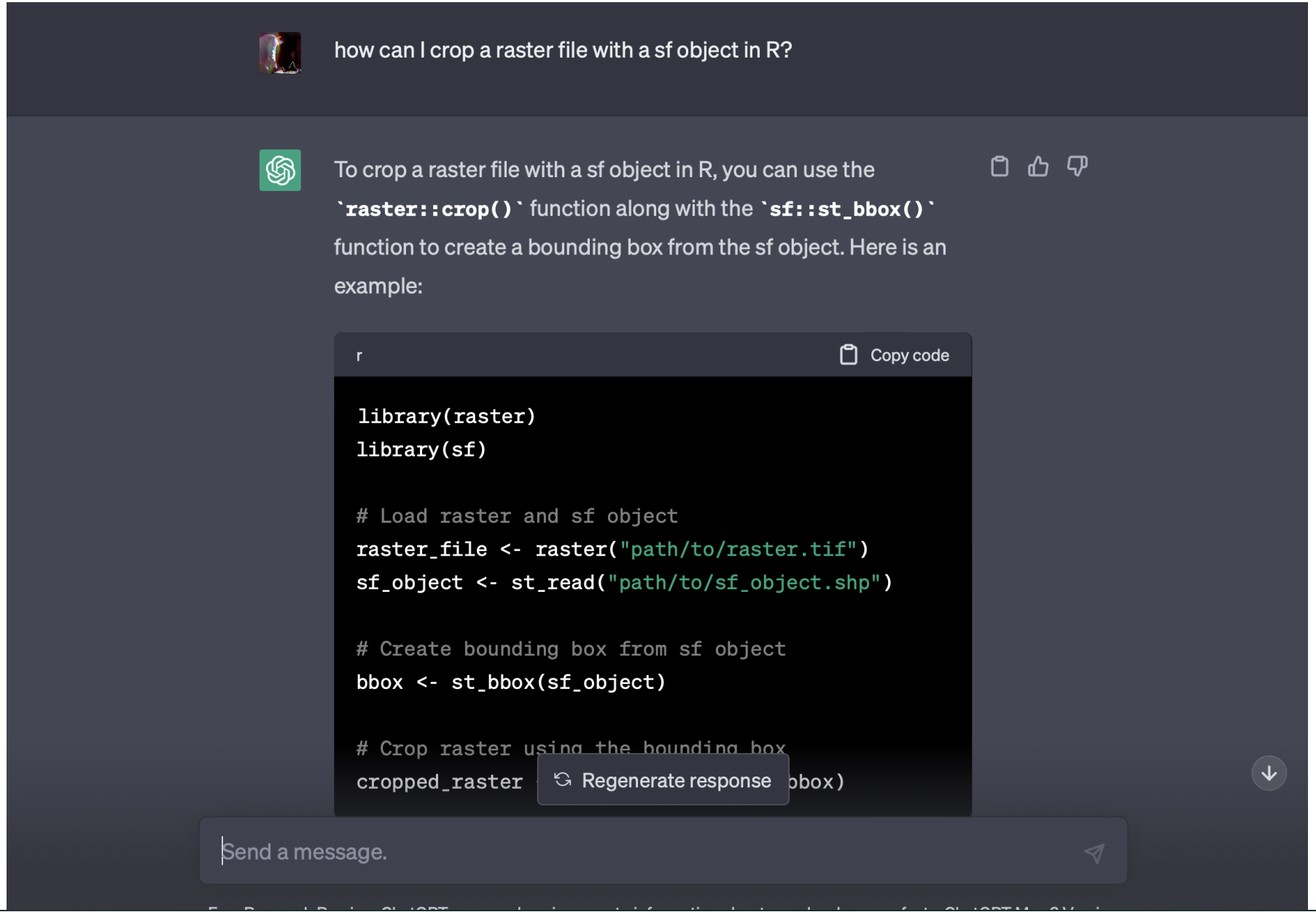
Run examples

```
# 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?



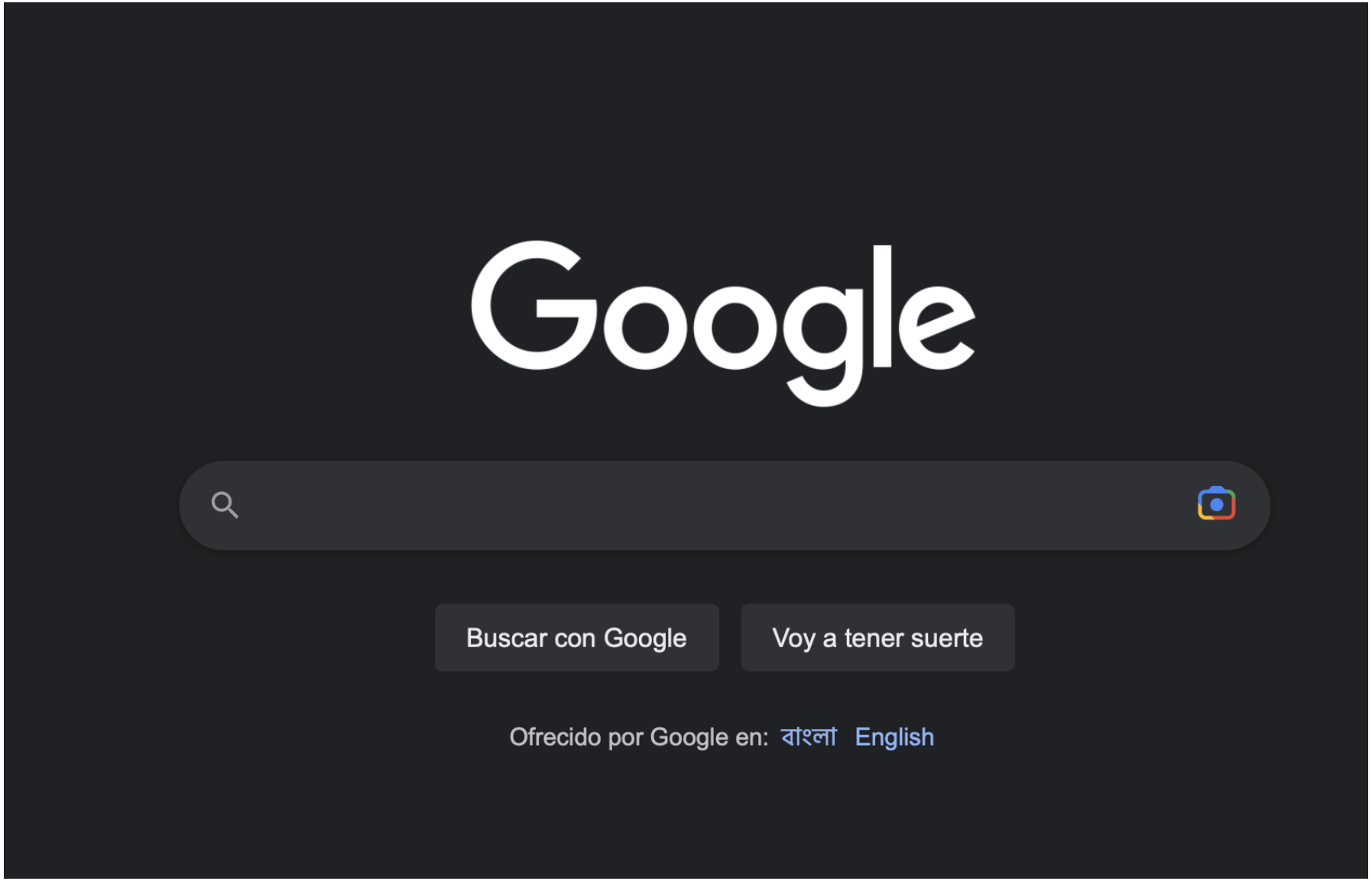
HOW CAN WE FIND HELP WITH R?

ChatGPT:

! WARNING: CHAT GPT CAN GIVE INCORRECT INFORMATION !

- If chat GPT does not knows 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