POLI 30 D: Political Inquiry

TA Sessions

Lab 02 | R Basics I

Before we start

Announcements:

- Quizzes and Participation:
  - Start at week 03. We will give full marks on Quiz 1 for everyone on week 03. You're welcome :)
- Github page:

https://github.com/umbertomig/POLI30Dpublic

Before we start

Recap:

- Last class, you learned how to install R and R Studio on your computer.

Great job!

- Do you have any questions about installations? Is it working fine? Let us know!

Plan for Lab 02

- Become familiar with RStudio

- Become familiar with R
  - do calculations: +, -, *, /
  - create objects: <-, "
  - use functions: (), sqrt(), #

Why is it so hard to code?

If you are worried because...

- You have never done any statistics or coding...
  - Don't worry. We assume you haven't
- Math is your "mortal enemy" or "you are not good at it"...
  - We will start from zero and progress slowly
  - There is plenty of help available
  - As long as you put in the time to do the work every week, you will do great!
- You don't think you will be able to "memorize" all the equations ...
  - Start slow and progress. Understand trumps memorization!

Why is it so hard to code?

- Can you read this sentence: "4 exampel in Ingli$h you kin get prackicly evRiThing rong-rong-rong and sti11 be undr3stud."
- You can do it because English is a *Natural language*.
- R, Python, and other programming languages are *formal languages*.
- This means that you should pass the exact instructions to the computer to it do what you want to get done.
- Learning how to code is learning to speak a very formal language. As with learning any language, **practice makes it perfect**!

Learning your R Studio

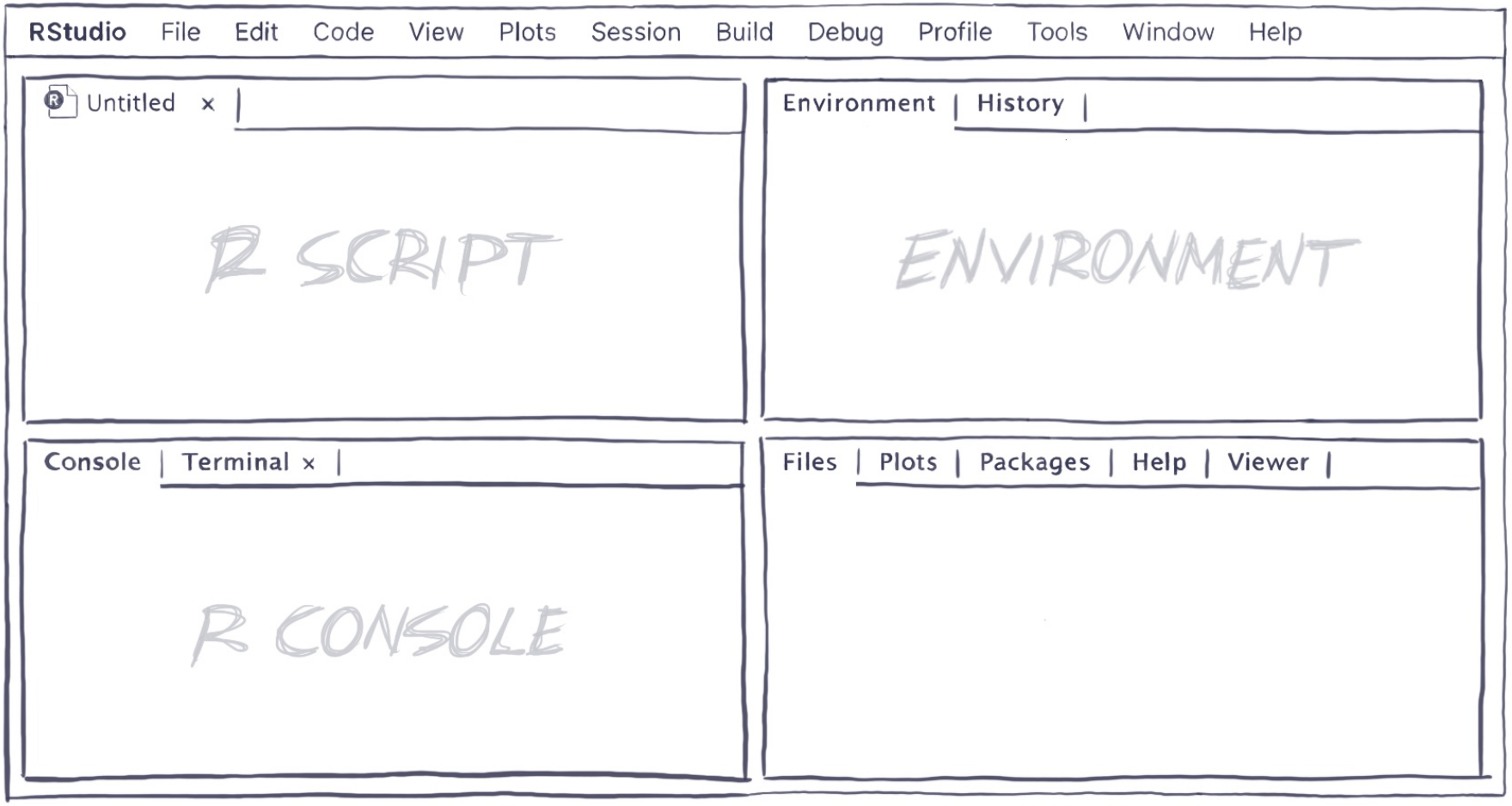
R and RStudio

- In Lab 01, you should have installed two programs on your computer:
  - R () and RStudio ()
- R is the statistical program that will perform calculations and create graphics for us (it's the engine)
- RStudio is the user-friendly interface that we will use to communicate with R
- We will never open R directly; we will always start by opening RStudio
  - RStudio will open R by itself

RStudio

- Go ahead and open RStudio ()
- Then, open a new R script:
  - dropdown menu: File > New File > R Script
- What is an **R script**?
  - type of file we use to store the code we write to analyze data

RStudio Layout



RStudio Layout

- **R Script** (upper left window): where we write and run code
- **R Console** (lower left window): where R provides the executed code and its outputs, including errors
- **Environment** (upper right window): storage room of current R session; lists objects that we have created
- **Help** and **Plots** tabs (lower right window)

We will use R to:

1. Do calculations
2. Create objects
3. Use functions

1. Do calculations

- We can use R as a calculator
  - R understands arithmetic operators such as +, -, *, /
- Let's ask R to calculate 20 plus 5
- First, we type on the R script (upper left window): 20+5
- Then, to run this code: we highlight it and either
  - (a) manually hit the run icon (

) or
  - (b) use the shortcut *command+enter* in Mac or *ctrl+enter* in Windows
- Go ahead and do it

- In the Console, you should see the following:

```
> 20+5
[1] 25
```

  - first, the executed code in blue (after >)
  - then, the output of the executed code in black
  - what does the [1] mean?
  - it indicates that the output immediately to its right is the first (and only, in this case) output

- The title of the R script is now red to indicate that you have unsaved changes
  - to save the R script either use shortcuts (*command*+*S* or *ctr*+*S*) or click on File > Save or Save As...
  - name it "lab02" so that you know what it refers to

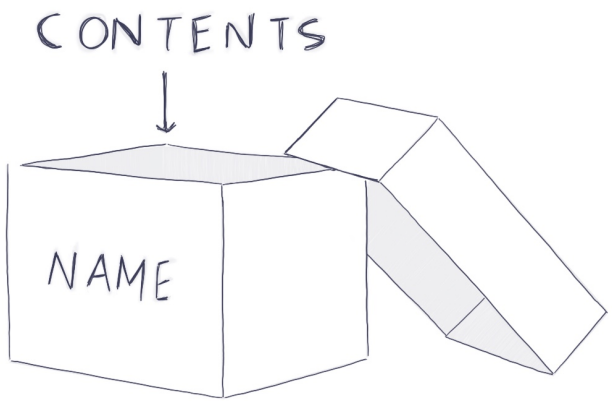
- In the book, as well as in the lectures, we show the output that you should see in the Console right after the code that produces it, like so:

```
20+5
## [1] 25
```

  - `20+5` is the code to be typed and run on the R script (the code that R will execute)
  - the symbol `##` indicates the beginning of the output
  - `[1] 25` is the output (what you should see in the Console after the executed code)

2. Create objects

- R stores information in the form of **objects**
- To analyze data, we will need to create objects
- An object is like a box that can contain anything



To create one, we need to:

- give it a name
- specify its contents
- use the assignment operator <-

In R, we use the assignment operator `<-` to create an object:

- To its left, we specify the name of the object
  - name cannot begin with a number or contain spaces or special symbols like $ or % that are reserved for other purposes
  - name can contain _ underscores, which are good substitutes for spaces
- To its right, we specify the object's content

```
object_name <- object_content
```

object_name <- object_content

- For example, type and run:

```
twentyfive <- 25
```

- After running this code, the object *twentyfive* will show up in the Environment (the upper right window of RStudio)
- To find out the contents of an object, you can run the name of the object in R:

```
twentyfive
## [1] 25
```

- This is equivalent to asking R: what is inside of *twentyfive*?

- Objects can contain text (called strings of characters) as well as numbers
- Run, for example:

```
class <- "POLI30D"
```

- Now, in the environment, there should be two objects
  - What are they?
- Note that in this last piece of code, we used " around the contents, but we did not use " in the previous piece of code

twentyfive <- 25 vs. class <- "POLI30D"

- Why?

When do we need to use " when writing code in R?

- the names of objects, names of functions, and names of arguments as well as special values such as TRUE, FALSE, NA, and NULL should NOT be in quotes
- all other text should be in quotes
- numbers should never be in quotes unless you want R to treat them as text

- What would happen if you run instead: class <- POLI30D?
- ```
  class <- POLI30D
  ## Error: object 'POLI30D' not found
  ```
  - without the ", R thinks that *POLI30D* is the name of an object and R is right; there is no object called *POLI30D* in the environment
- **Running into errors is part of the coding process**
  - do not be discouraged
  - if you have problems figuring out what a particular error means, google it; there are lots of Q&A sites
  - if that doesn't help, post the code and error in our Canvas Discussion
  ```

- R will overwrite objects if you assign new content to an existing object name

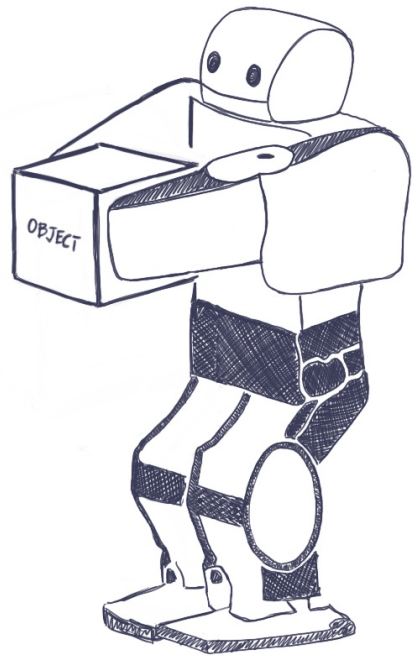
```
class <- "data analysis"
```

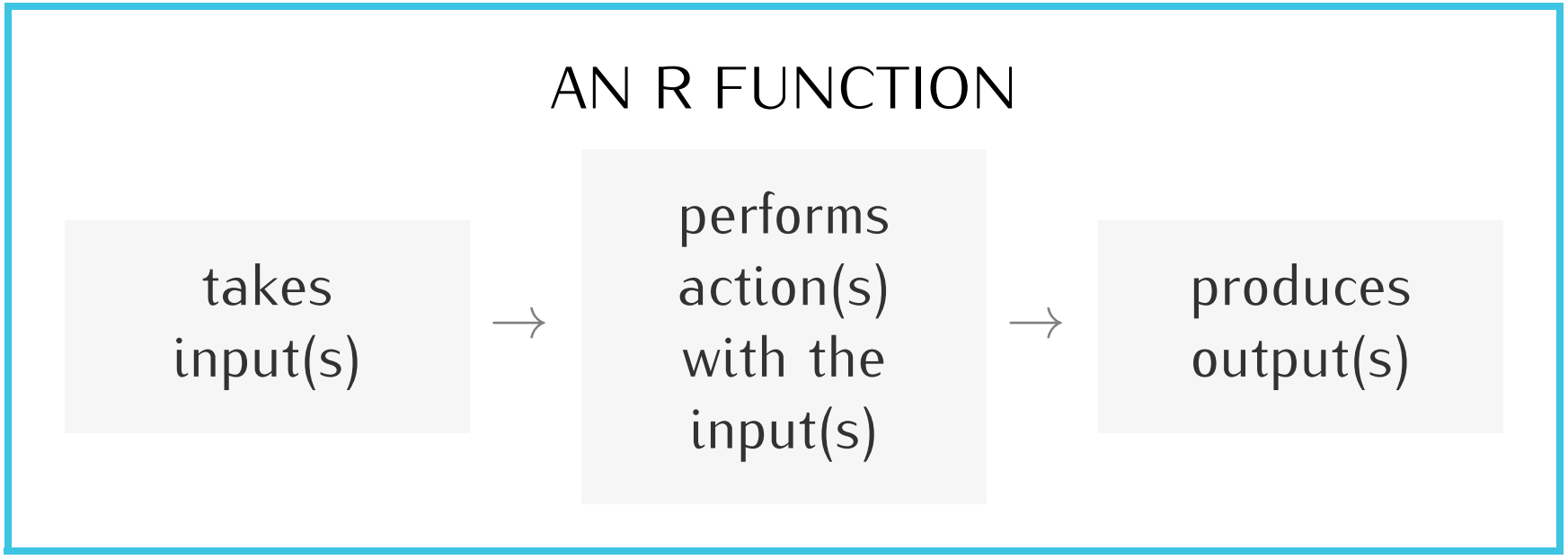
  - After running the code above, *class* will contain the text “data analysis” instead of “POLI30D”

- R is case sensitive:
  - *class* is different than *Class*
  - to avoid confusion, we use lower-case letters when naming objects

3. Use functions

- Think of a function as an action that you request R to perform on a particular object or piece of data, such as calculating the square root of 25





- A **function**:
  - takes input(s)
    - example: takes the number 25
  - performs an action with the input(s)
    - computes $\sqrt{25}$
  - produces an output
    - produces the number 5

- We will learn how to use these functions:
  - sqrt(), setwd(), read.csv(), View(), head(), dim(), mean(), ifelse(), table(), prop.table(), hist(), median(), sd(), var(), plot(), abline(), cor(), lm(), c(), sample(), rnorm(), pnorm(), print(), abs(), and summary()
- In time, we will learn:
  - their names
  - the actions they perform
  - the inputs they require
  - the outputs they produce

- The name of a function (without quotes) is always followed by parentheses: *function_name()*
- Inside the parentheses, we specify the inputs, which we refer to as arguments: *function_name(arguments)*
- Most functions require that we specify at least one argument but can take many optional arguments
  - some arguments are required, others are optional
- When multiple arguments are specified inside the parentheses, they are separated by commas: *function_name(argument1, argument2)*

- To specify the arguments, we enter them in a particular order or include the name of the argument (without quotes) in our specification:
  - *function_name(argument1, argument2)* or
  - *function_name(argument1_name = argument1, argument2_name = argument2)*
- We always specify required arguments first
- If there is more than one required argument, we enter them in the order expected by R
- We specify any optional arguments we want next and include their names:
  - *function_name(required_argument, optional_argument_name = optional_argument)*

USING R FUNCTIONS:

We typically write code in one of these two formats:

function_name(*required_argument*)

or

function_name(*required_argument*,
optional_argument_name = *optional_argument*)

- Fictitious Example: Suppose R were capable of baking and that it had a function named bake() that, by default, bakes the specified ingredient for 60 minutes at 400°F
  - Required argument: the ingredient
    - example: cake mix
  - Optional arguments: named degrees and minutes to change the default temperature and duration of the bake, respectively
    - degrees=350 changes temperature to 350°F
    - minutes=30 changes the duration of the bake to 30 minutes

- The following code would ask R to bake a cake mix for 30 minutes at 350°F, so that we can have cake as the output:

FICTITIOUS EXAMPLE

bake(cake_mix, degrees = 350, minutes = 30)

INPUTS
required argument
= cake_mix
optional argument
degrees = 350
optional argument
minutes = 30

→ ACTION
bake
→ OUTPUT
cake

- Example: sqrt() computes the square root of the argument specified inside the parentheses. To compute $\sqrt{25}$, run:

```
sqrt(25)
## [1] 5
```

  - sqrt is the name of the function, which, as all function names, is followed by parentheses ()
  - 25 is the required argument
  - 5 is the output

- Alternatively, since the object *twentyfive* contains the number 25, we can run:

```
sqrt ( twentyfive )
## [1] 5
```

- R will give you an error message if you run this line of code before creating the object *twentyfive*
- Code is sequential! One must run code in order
  - Whenever returning to work on an R script, run all the code from the beginning

- It is good practice to comment on code
  - include short notes to yourself or your collaborators explaining what the code does
- To comment code, we use #
  - R ignores everything that follows this character until the end of the line
- Examples:

```
sqrt(25) # calculates square root of 25
## [1] 5
```

```
# sqrt(25) calculates square root of 25
```

- Before closing your computer, remember to save the R script, otherwise, you risk losing unsaved changes
  - either use shortcuts (*command*+*S* or *ctr*+*S*) or
  - click on File > Save
- If you quit RStudio, R will ask whether you want to save the workspace image, which contains all the objects you have created during the R session
  - I recommend that you do not save it
  - You can always re-create the objects by re-running the code in your R script

Today's Lab

- RStudio: R script, R console, and the environment
- R: do calculations: +, -, *, /
  create objects: <-, "
  use functions: (), sqrt(), #

Next Lab

- How to load and make sense of data in R
- **Bring your computers!**

Questions?

See you in the next lab!