Session 1 - Introduction to R

R training

María Reyes Retana
The World Bank | January 2025

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP
Development Economics • Impact

WORLD BANK GROUP
INSTITUTE FOR ECONOMIC DEVELOPMENT

Reproducible Research Repository

Government Analytics and R Training:

Strengthening Public Sector Reporting and Data Analysis

January 13 – January 17, 2025

Introduction

About this training

- This is an **introduction** to government analytics and statistical programming in R
- The training does not require any background in statistical programming
- A computer with R and RStudio installed is required to complete the exercises
- Internet connection is required to download training materials

Introduction

Learning objectives

By the end of the training, you will know:

- How to write **basic** R code
- A notion of how to conduct Government analytics in R and how it differentiates from Excel

Access to Training Materials

All the materials for this training are available at the following link:
https://osf.io/r3fn5/

Here's what you will find there:

- 📂 **Data**: All datasets we'll use throughout the training sessions.
- 📄 **Slides**: The presentation slides for each session.
- 🖍️ **Solutions**: Currently, solutions are not uploaded, but they will be added to the folder after each session.

Course Structure

This training will cover the basics of coding in R. Below is the structure of the course:

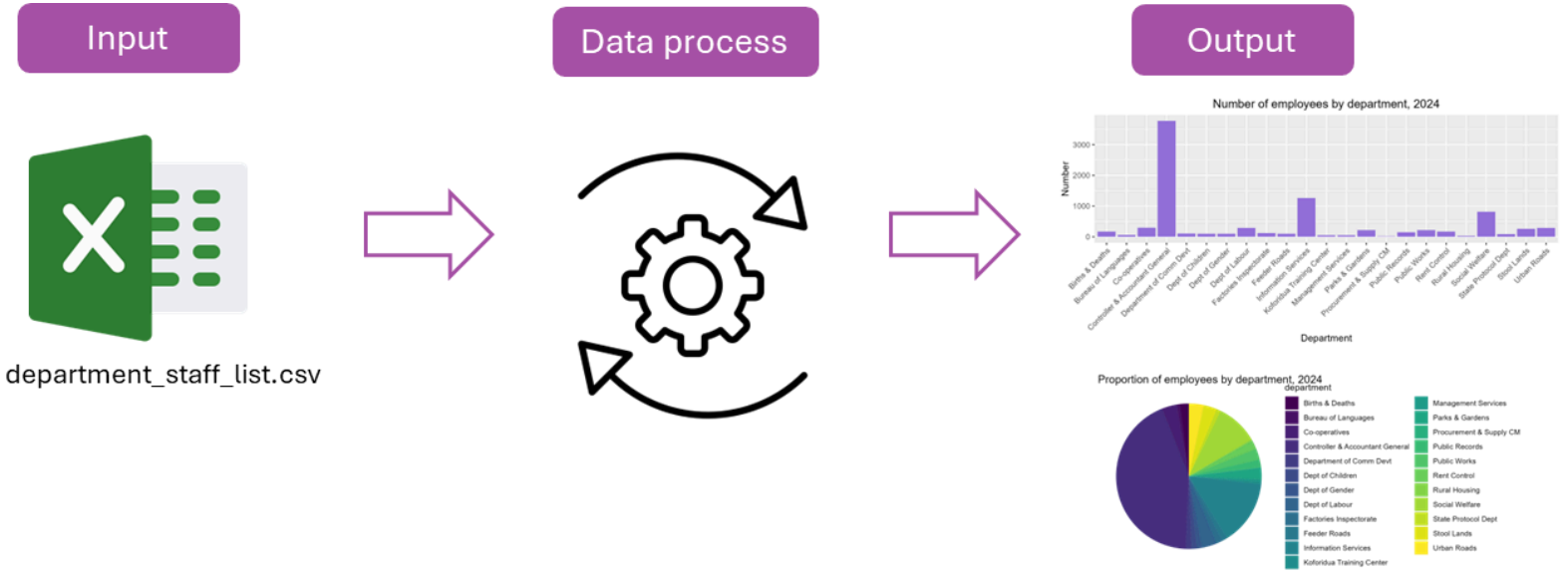
- 🗓️ **Day 1**: **Introduction to R – Get familiar with the R environment and basic syntax.**
- 🛠️ **Day 2**: Data Wrangling – Learn to clean and transform data effectively.
- 📊 **Day 3**: Descriptive Statistics – Explore methods to summarize and analyze data.
- 📈 **Day 4**: Data Visualization – Create meaningful and impactful data visualizations.

Government Analytics and Statistical Programming

Government Analytics

For the context of this training, we'll call Government analytics (or analytics) everything that:

1. Starts with a data input
2. Runs some process with the data
3. Produces an output with the result



Data input example:
CSV, Excel file

Process example:
Obtain sum or averages of data

Output examples:
Bar graphs, maps, tables

Government Analytics

- It's also possible to do analytics with Excel
- However, we will show in this training why using statistical programming (through R) is a better way of doing analytics

Input

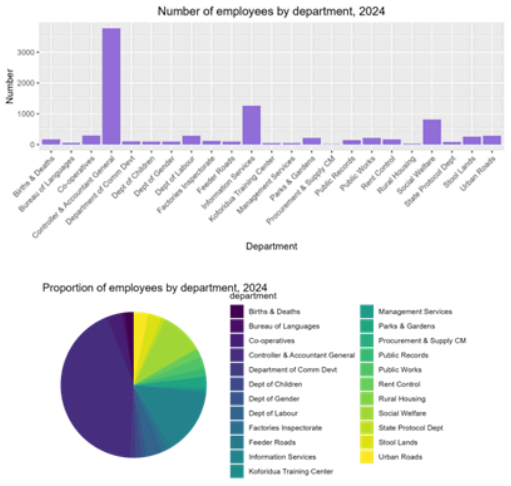
department_staff_list.csv

Data input example:
CSV, Excel file

Data process

Process example:
Obtain sum or
averages of data

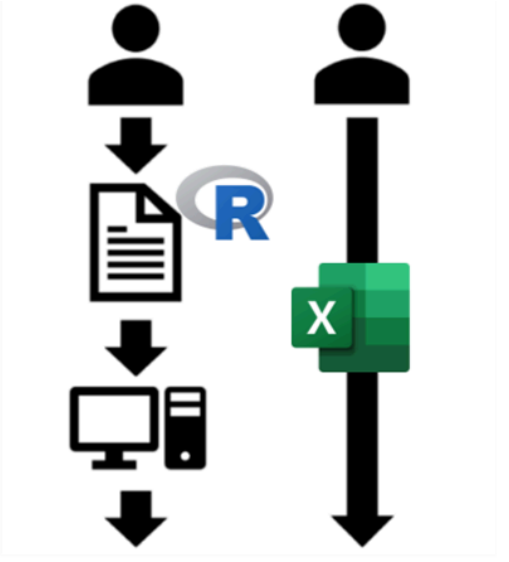
Output

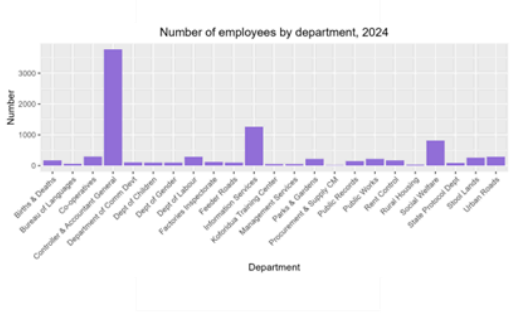


Output examples:
Bar graphs, maps,
tables

How Can it Benefit My Work?

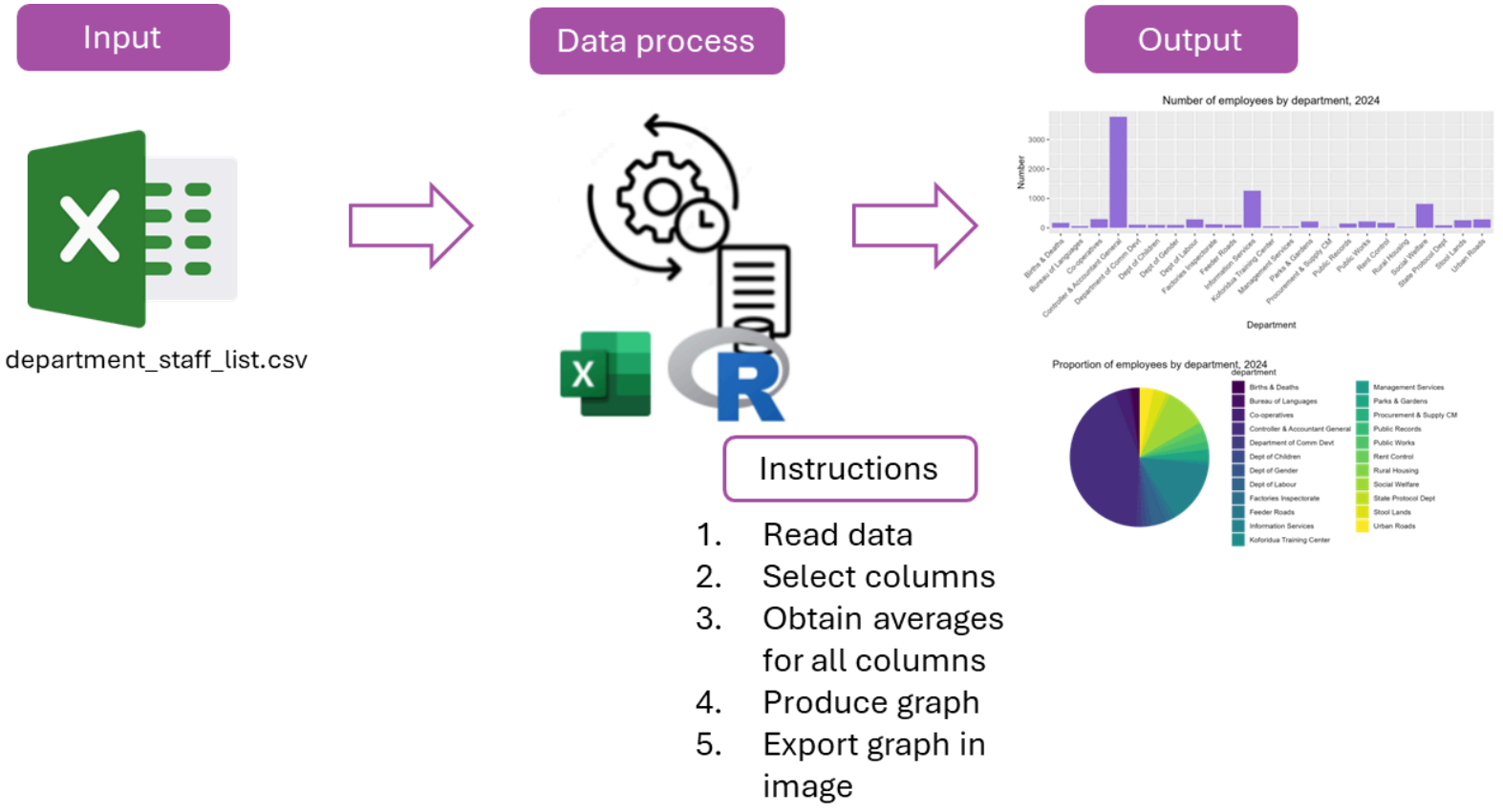
- **Reproducibility**: Code provides an exact record of every step, making results easy to trace and verify.
- **Reusability**: Code can be reused for similar projects, saving time on future reports.
- **Transparency Over Excel**: Instructions are documented and less prone to error compared to manual steps in Excel.





Statistical Programming

- Programming consists of producing instructions to a computer to do something
- In the context of analytics, that "something" is statistical analysis or mathematical operations
- Hence, statistical programming consists of producing instructions so our computers will conduct statistical analysis on data



Statistical Programming

- You can think of statistical programming as writing a recipe

Input

department_staff_list.csv

Data process

Instructions

1. Read data
2. Select columns
3. Obtain averages for all columns
4. Produce graph
5. Export graph in image

Output

Instructions

1. Preheat oven
2. Combine butter and sugar
3. Add eggs
4. Add rest of the ingredients and stir
5. Bake

Statistical Programming

Why use R

- Statistical programming can be implemented through many different software. Other options are Stata and Python
- We recommend using R for these reasons:
  - R is free
  - R was designed specifically for statistical programming
  - There is a large worldwide community of R users. This means you can easily look for help or examples of code in the internet

It's not unusual to struggle at first but it gets better!

- By the end of this training, you will learn how to leverage your existing data to create exhibits for your annual reports using R.
- We know this may feel challenging, but you'll get there!
- Remember to ask questions, be patient with yourself, and take it step by step.

How to write R code?

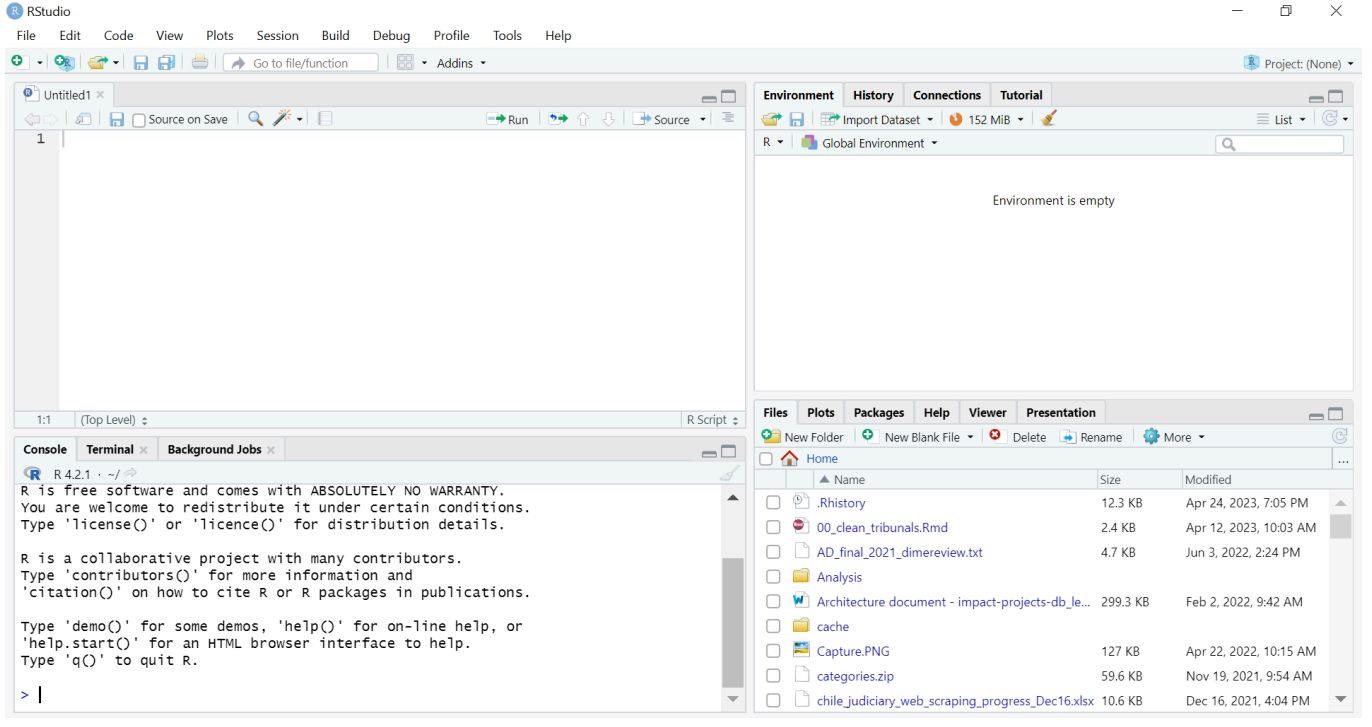
- The rest of today's session focuses on the basics of writing R code
- We'll use RStudio to write R code in this training

How to write R code?

- Now open RStudio in your computer
- Please make sure you're opening RStudio and not R

How to write R code?

- Now open RStudio in your computer
- Please make sure you're opening RStudio and not R



Questions?

Writing R code

Writing R code

At its core, R can function as a (very fancy) calculator, allowing you to perform basic mathematical operations. Here are some common operators:

Mathematical Operators

- **Addition**: `+`
- **Subtraction**: `-`
- **Multiplication**: `*`
- **Division**: `/`
- **Exponentiation**: `^`

Comparison

- **Equal**: `==`
- **Not Equal**: `!=`
- **Greater than** `>`
- **Less Than** `<`

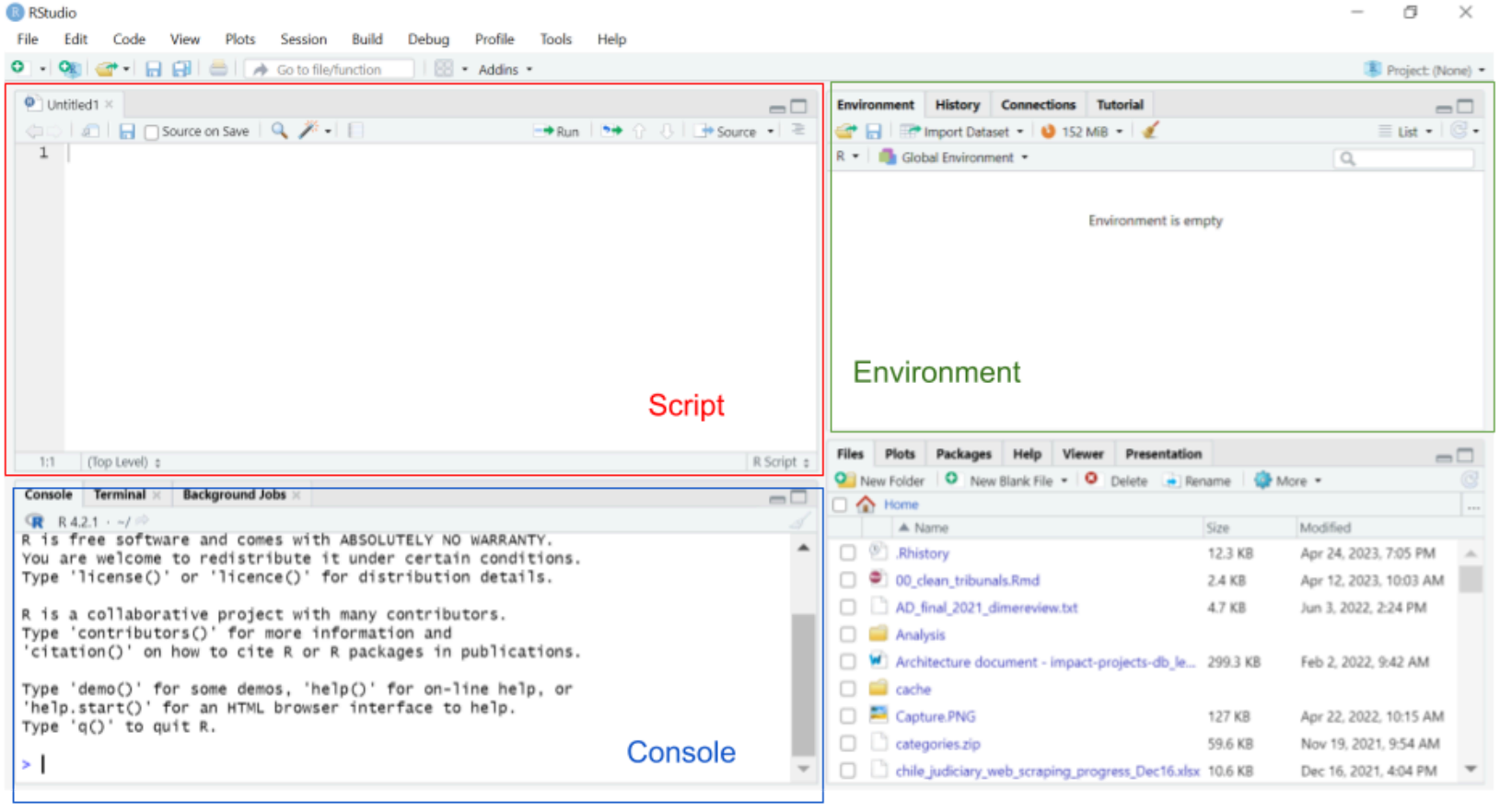
Logical Operators

- **AND**: &
- **OR**: |

And many more ... you can see this list with a lot of examples.

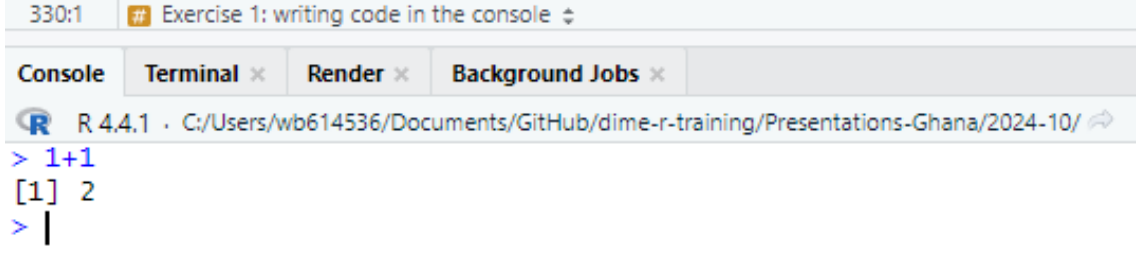
RStudio interface

RStudio has different elements, each with a specific use.



Exercise 1: writing code in the console

1. Write the following code in the console of RStudio
    - `1 + 1`
2. Press Enter to run the code



Writing R code

R objects

- Remember we also mentioned the environment panel? that's where R keeps track of objects
- You can think of R objects as saving information, for example simple numbers or just plain text.
    - A single number can be an object
    - A word can be an object
    - Even an entire data file can be an object
- We create objects in R with the arrow operator (`<-`)
- After an object is created, we can refer to it using its name:

```
x1 <- 100 + 50
```

```
x1
```

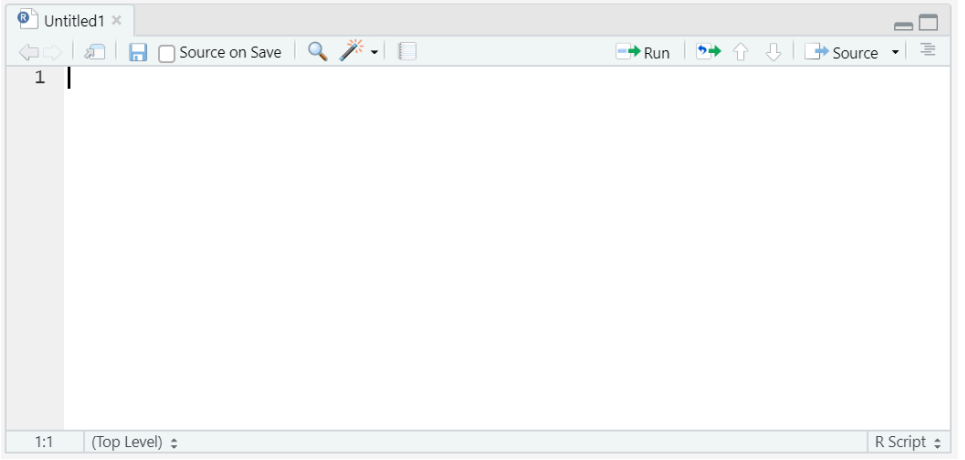
```
## [1] 150
```

Writing R code

Exercise 2: writing a short script and create objects

1- Write or copy the following text into the script section of RStudio

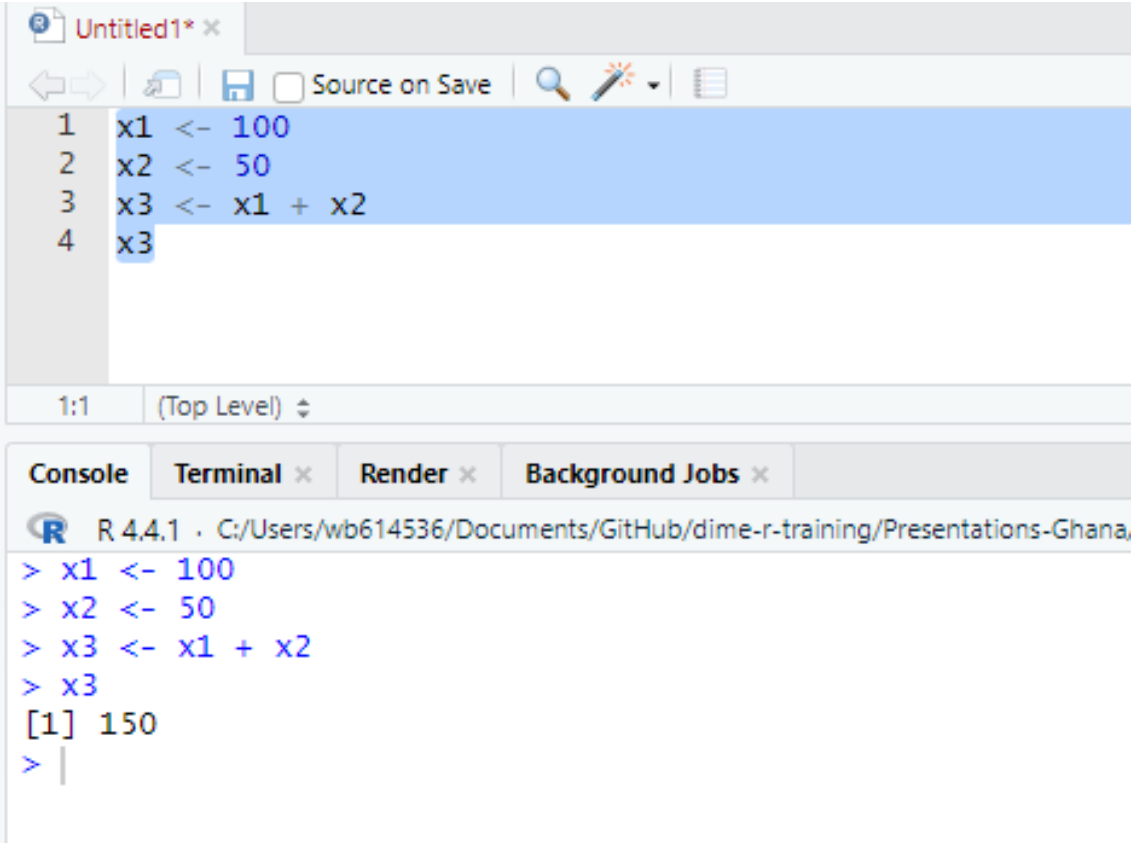
```
x1 <- 100
x2 <- 50
x3 <- x1 + x2
x3
```



2- Select the text you introduced with your mouse

3- Press "Run"

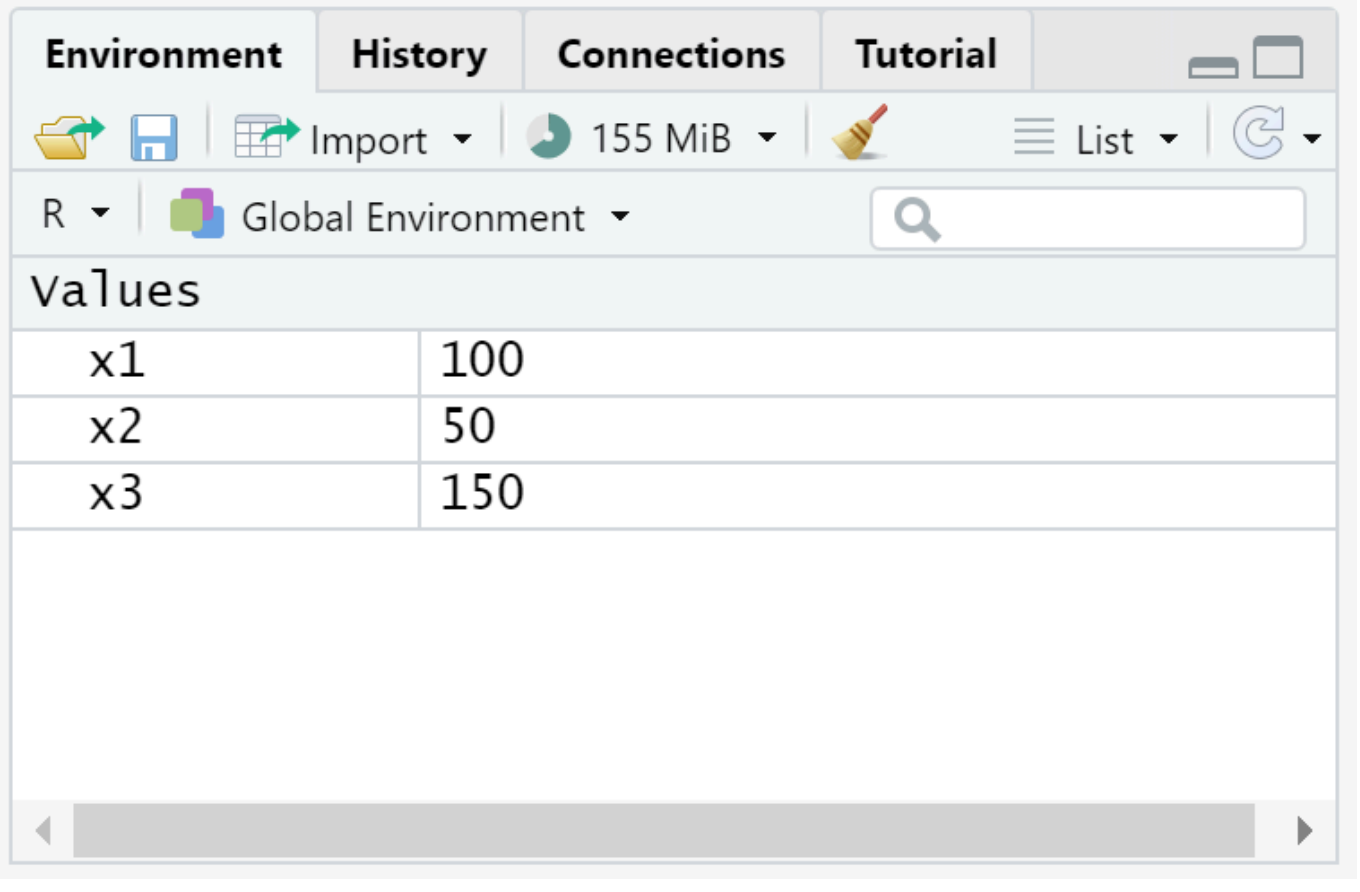
Writing R code



Writing R code

Creating objects in R

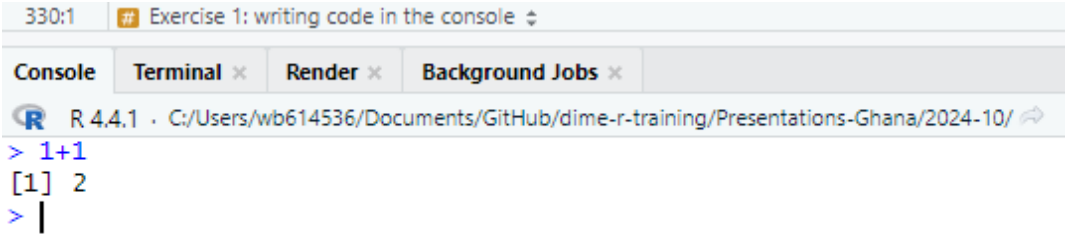
- After any objects are created, they will show in the environment panel



Writing R code

R scripts

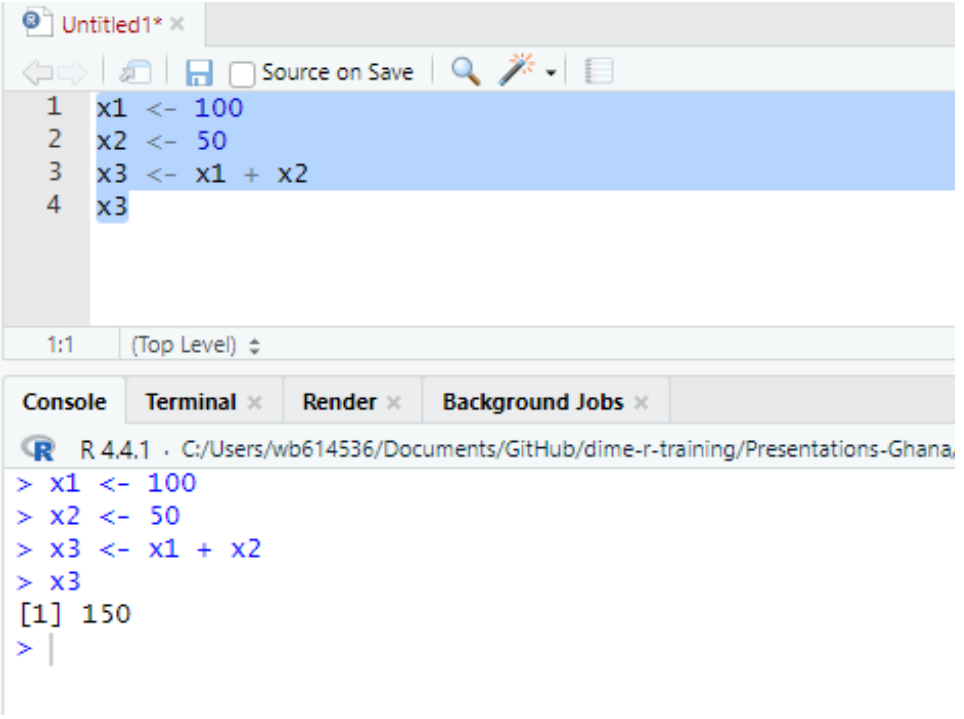
- Writing and running code from the console will execute it immediately



Writing R code

R scripts

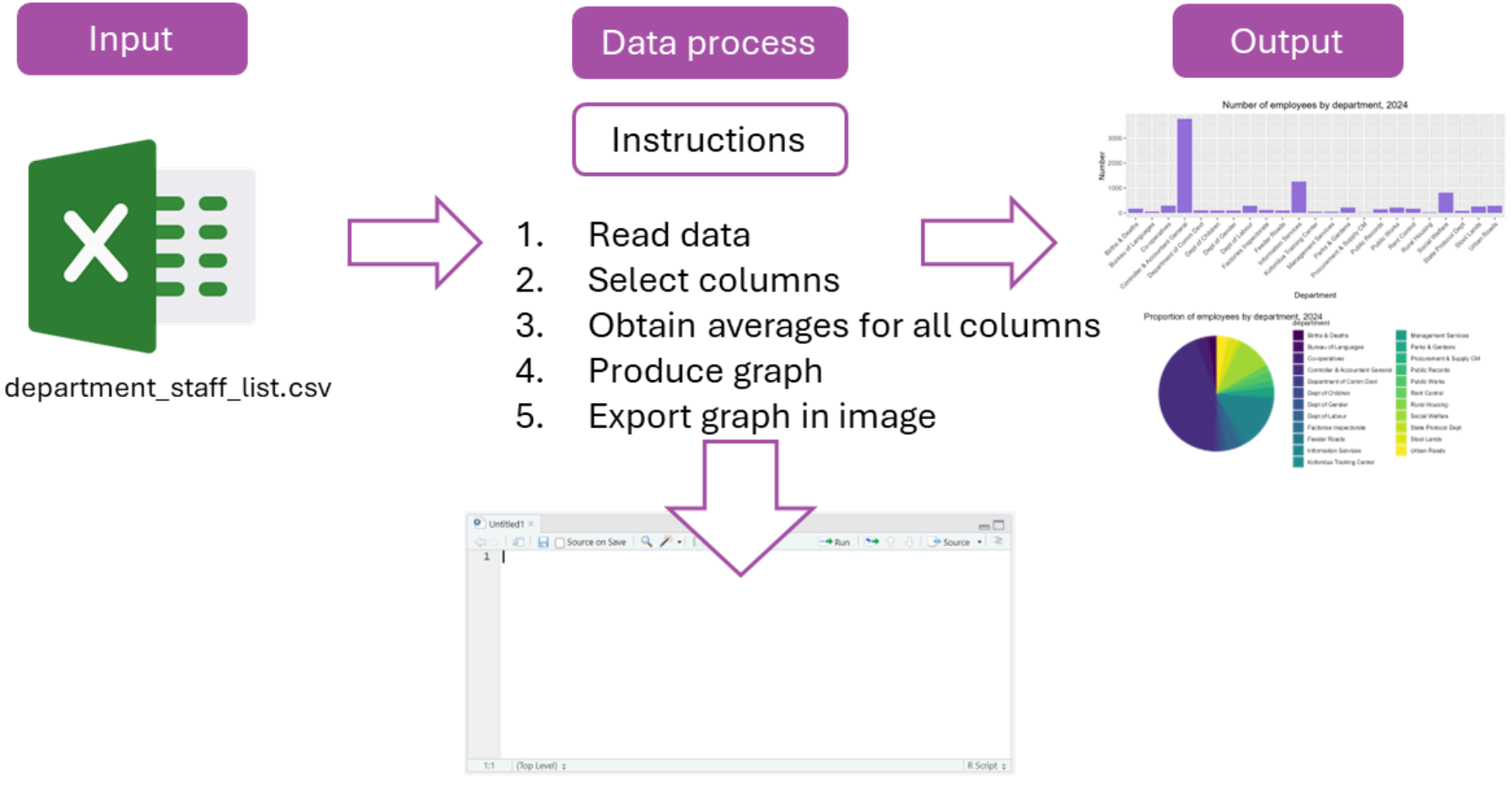
- Writing and running code from the console will execute it immediately
- Writing code in the script panel allow us to write multiple lines of code and execute them later
    - Each line is executed in order
    - The line and the results will show in the console
- **Important:** for the rest of the training, remember to always introduce your code in the script (and not in the console) so you can keep record of what you did



Writing R code

R scripts

- In other words: scripts contain the instructions you give to your computer when doing Government analytics



Object Types

Object Types

What Are Object Types?

- **R objects** are containers for data, and they come in different **types**.
- The type of an object determines:
  - What **operations** can be performed on it.
  - How R will treat it during analysis.

Examples of common object types:

1. **Numeric**: Used for numbers, such as `100` or `3.14`.Can be used for mathematical operations like addition, subtraction, etc.
2. **Character**: Represents text or words, like `"Hello"` or `"Region A"`.
3. **Vector**: A collection of values stored in a single dimension, like a list or a column in Excel. Example: A list of ages: `c(25, 30, 35, 40)`.
4. **Dataframe**: A collection of rows and columns, similar to a table or spreadsheet.Each column can have its own type (e.g., numeric, character).

Object Types

Let's Explore Object Types

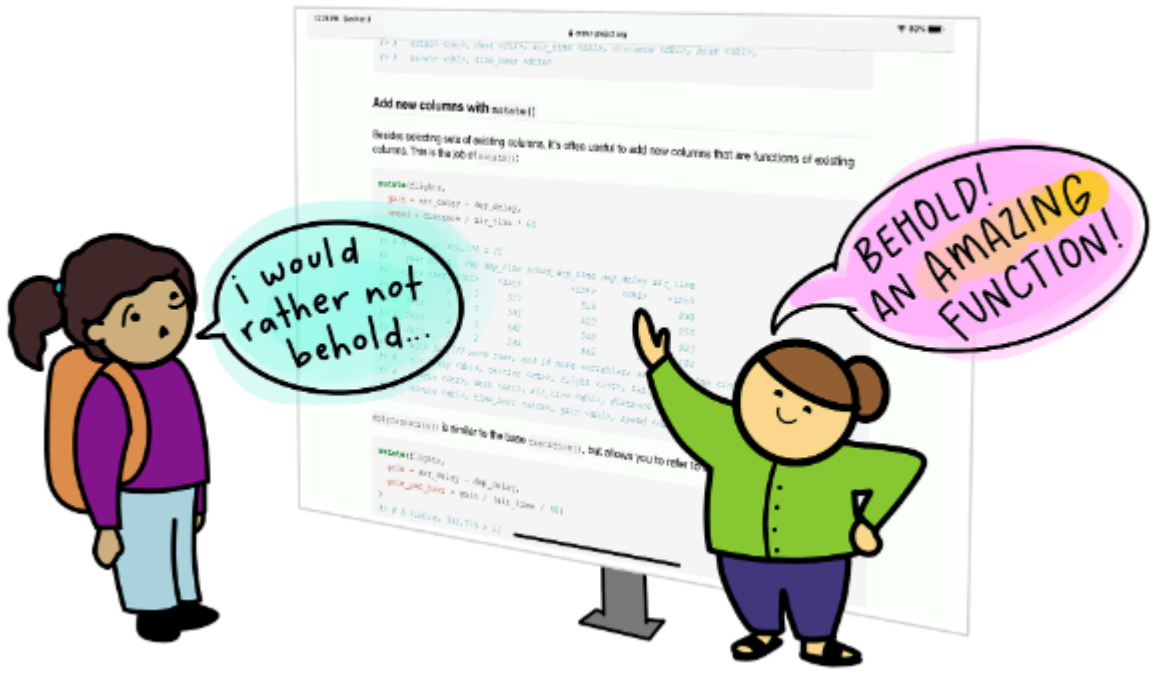
Create objects in RStudio:

```
num_object <- 42                # Numeric

char_object <- "Learning R"     # Character

vector_object <- c(1, 2, 3, 4) # Vector

df_object <- data.frame(
  Name = c("Alice", "Bob"),    # Dataframe
  Age = c(25, 30)
)
```

- Which object do you think would be most common in the work that you do?
- We will discuss more about the last type, but first we need to introduce one last concept.

Functions in R

Functions in R



Functions in R

- Functions are how we apply operations to objects in R
- You can think of functions as little machines that (in most cases) process some kind of **input** and create an **output**.
- Input is everything that goes into a function (arguments and values)

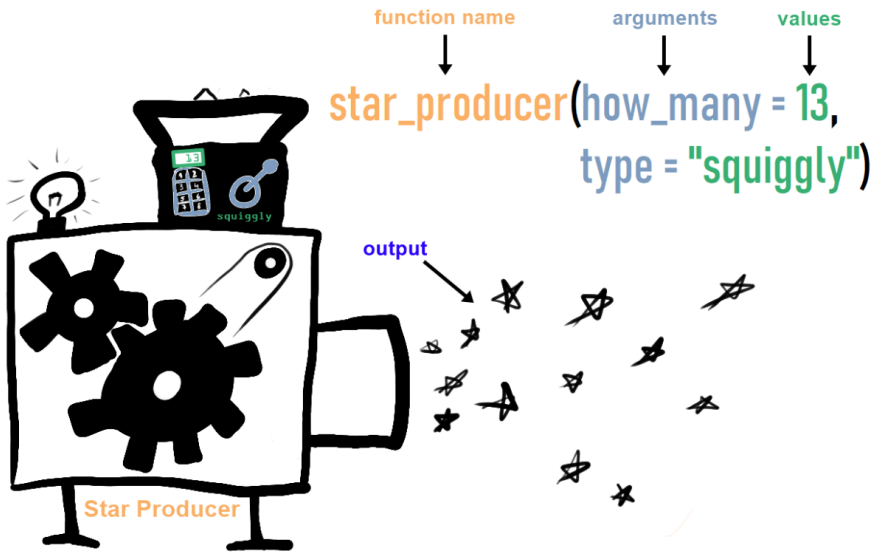
Let's take a look at an example: the star producer!

The Star Producer

Let's consider the following function (that does not exist unfortunately): **A star_producer!**

This little machine creates tiny hand-drawn stars depending on some input. It takes two arguments:

- `how_many` tells the machine how many stars to produce.
- `type` tells the machine how the stars should look like (in this case the machine only supports "squiggly" stars).



Functions in R

Getting `?help`

How do we know what function takes what kind of arguments?

Within R, you can always run the code:

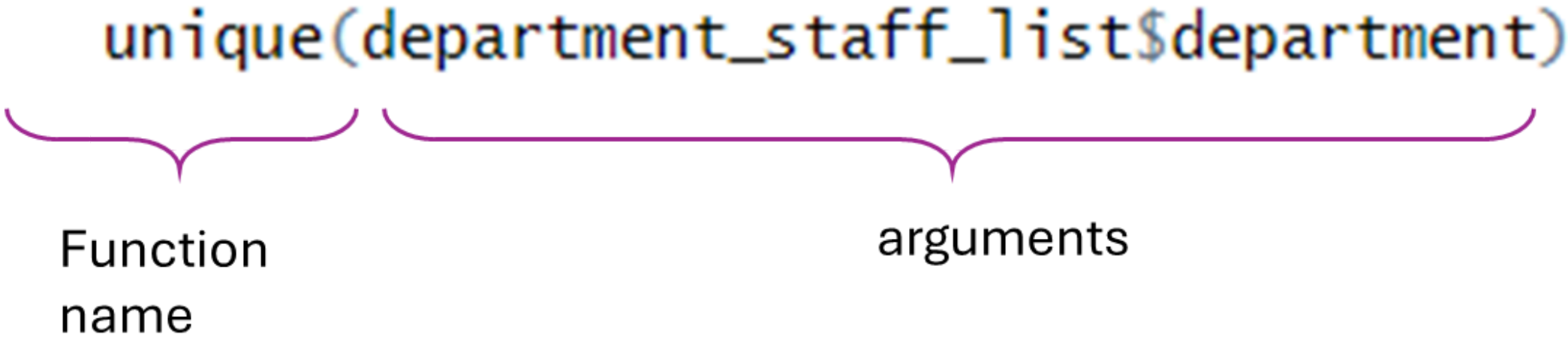
```
?function_name
```

This will open up the documentation for the function, which explains how to use it.

Googling the function name (adding "R" or "rstats" to the search) will also often bring up relevant documentation or examples!

Functions in R

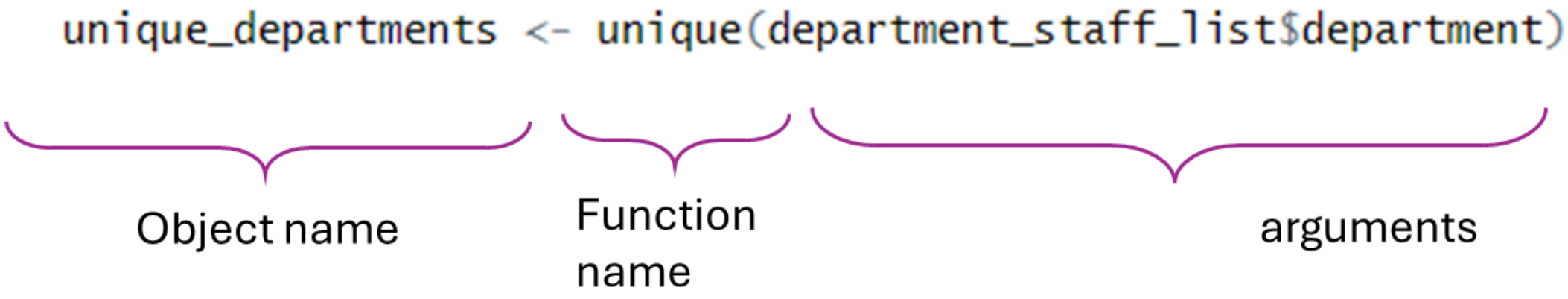
Now in a real function ...



- **Function name:** the name we use to call a function. It goes before the parentheses
- **Arguments:** inputs and specifications for the function to be applied.
    - Arguments go inside the parentheses
    - The first argument is the object you apply the function on

Functions in R

- The results of a function can always be stored in an object with the arrow operator (`<-`)



- The results of a function will only be printed in the console if you don't store them

Functions in R

Okay now let's try it!

Exercise 3: Using a function `sum()`

1. Compute the sum of the numbers 1 to 10 and store it in the object sum_example:

```
sum_example <- sum(c(1:10))
```

1. Print the stored result with `print(sum_example)`

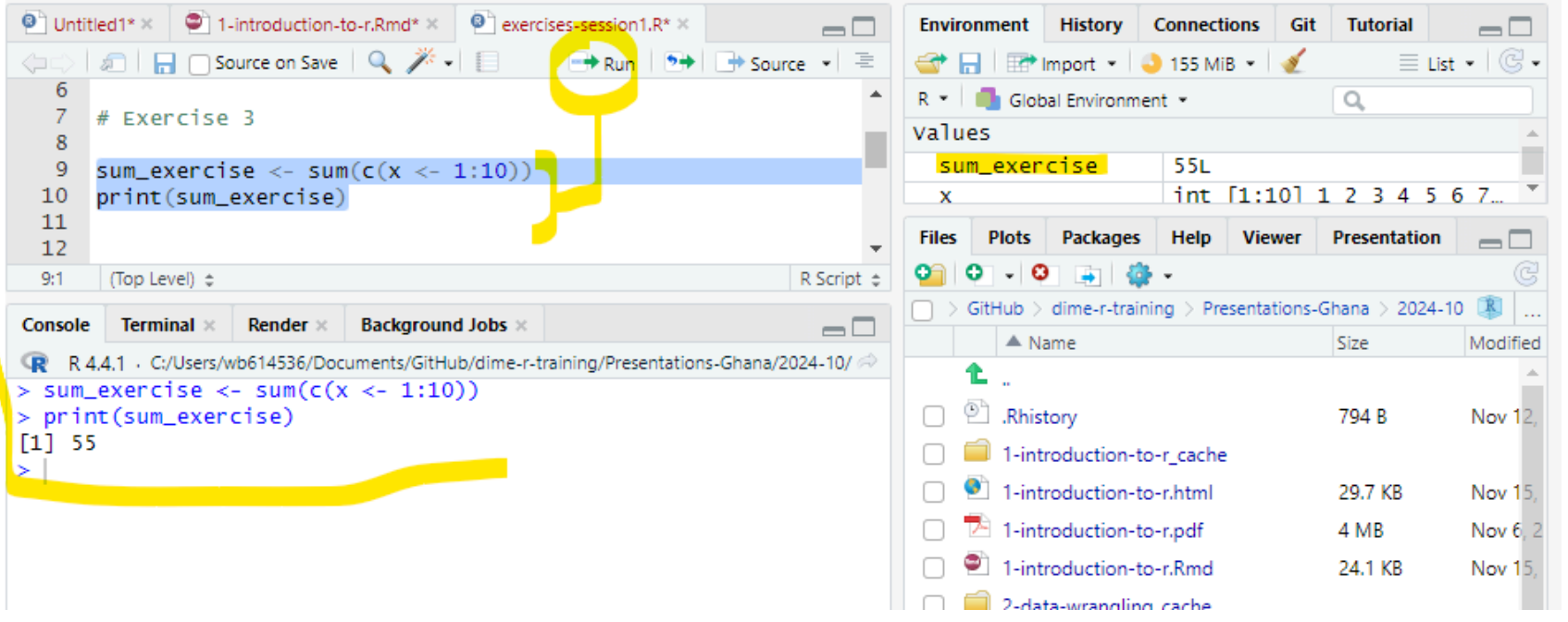
```
print(sum_example)
```

```
## [1] 55
```

❎ In Excel: This is as when you have a column of numbers in Excel and want to calculate their total

Functions in R

Note that this code is both creating a new object (with `sum_example <- sum(c(1:10))`) and printing the result in the console (with `print(sum_example)`)



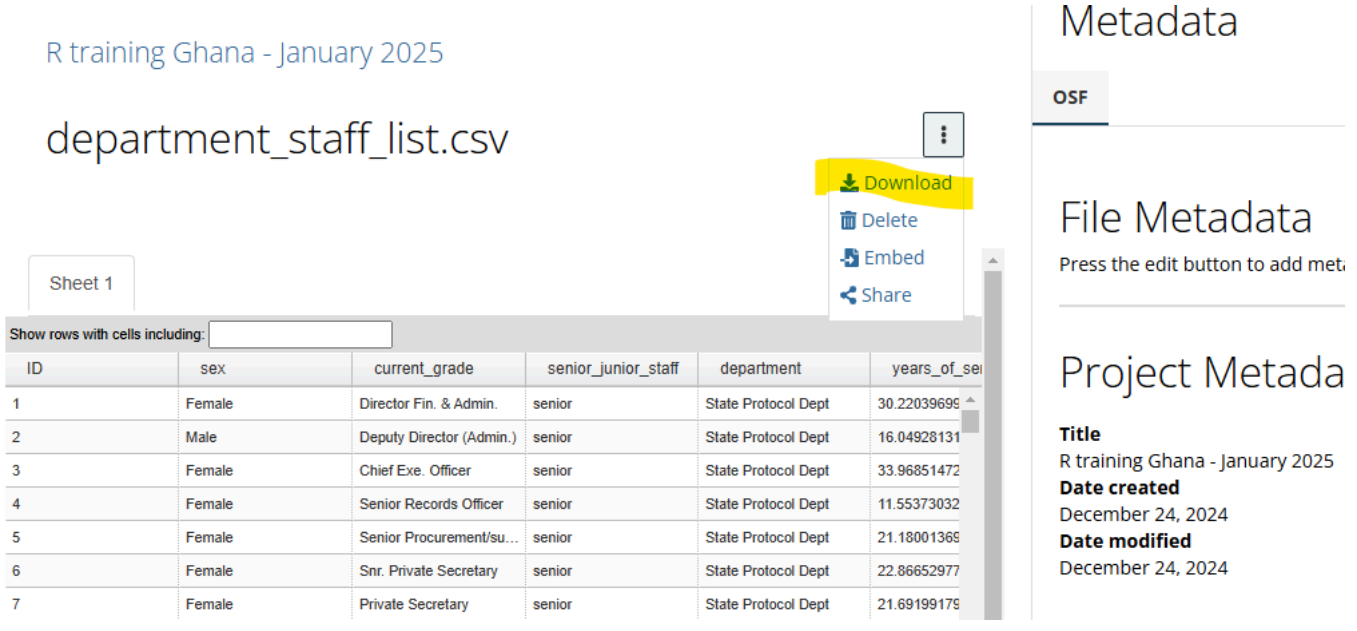
Writing R code

- Now we know how to use RStudio to write R code and produce scripts.
- We also know about objects and functions.
- We haven't still introduced the data to our Government analytics. That comes next

Data in R

Exercise 4: Loading data into R

1.- Go to this page: https://osf.io/chdgj and download the file `department_staff_list.csv`

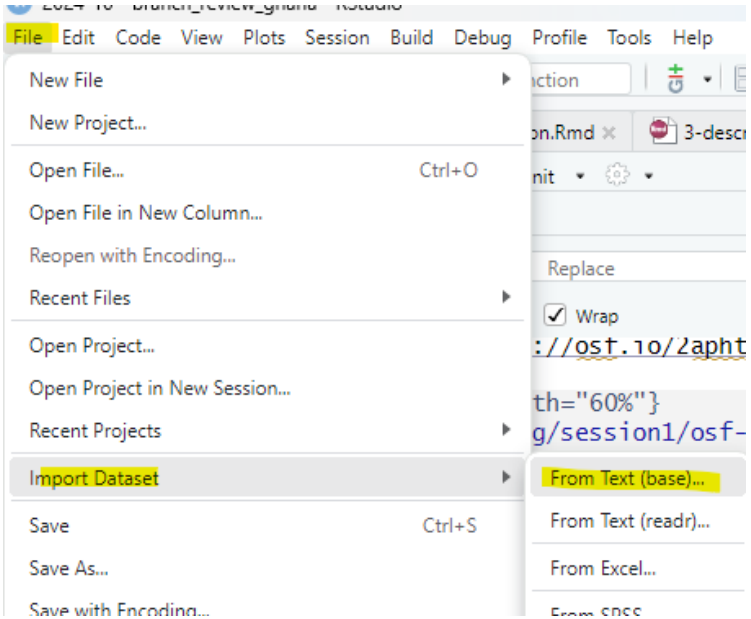


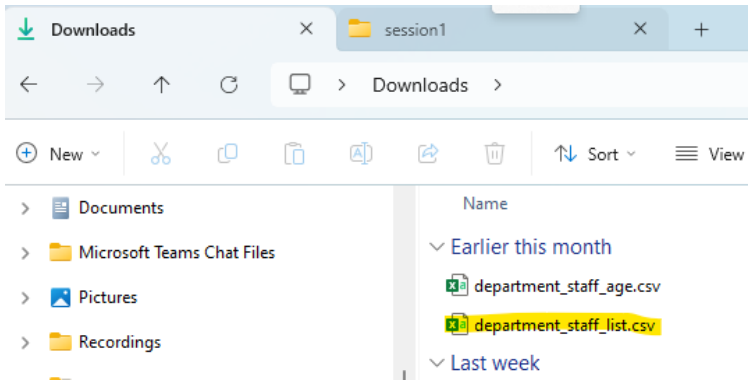
Exercise 4: Loading data into R

There are different ways of importing data to R, one is using the point and click. Let's start with that one.

2.- In RStudio, go to `File` > `Import Dataset` > `From Text (base)` and select the file `department_staff_list.csv`

- If you don't know where the file is, check in your `Downloads` folder



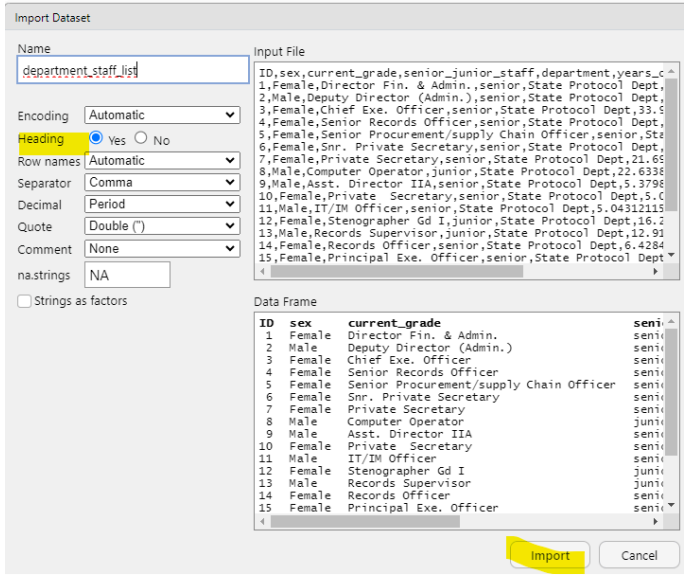


Exercise 4: Loading data into R

3 - Make sure to select `Heading` > `Yes` in the next window

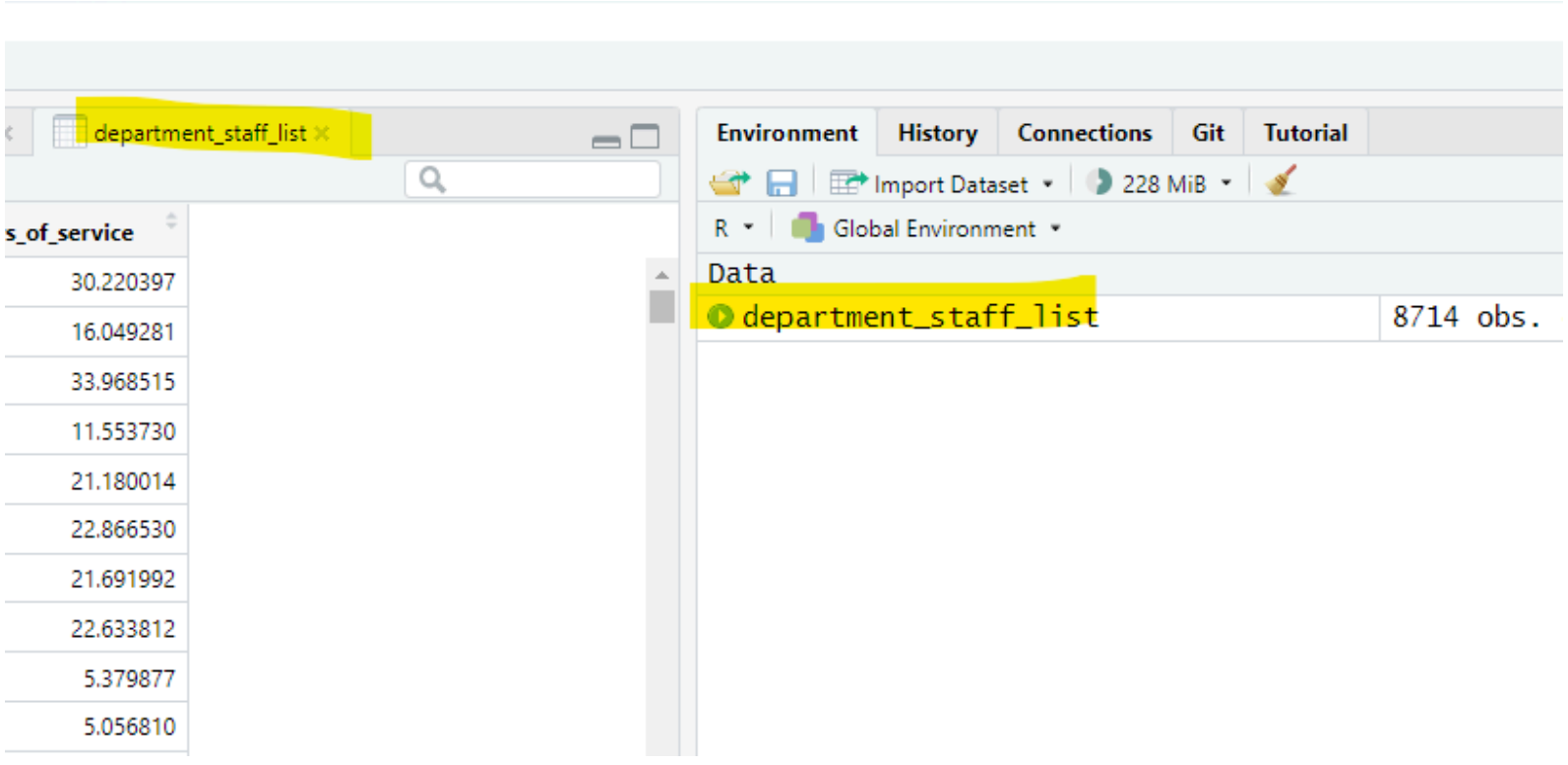
4 - Select `Import`

5 - You will see that the second way to read it by code (using functions), and is what R is doing for you in the background.



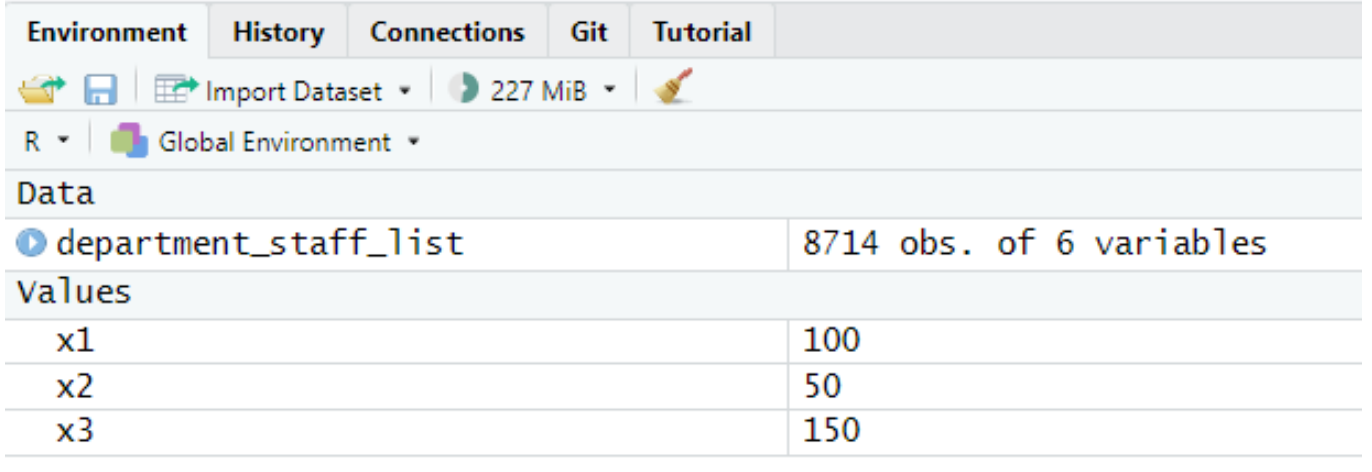
Data in R

- If you did this correctly, you will note that a viewer of the data now appears in RStudio
- You will also note that this appeared as a new object in your environment



Data in R

- Remember we mentioned **dataframe** objects before? For R, `department_staff_list` is an object just like `x1`, `x2`, or `x3`
- The difference is that `department_staff_list` is not a single number like `x1`, but a collection of numeric values similar to an Excel spreadsheet. In R, this type of objects are called **dataframes**
- From now, we will refer to data loaded into R as **dataframes**



Data in R

- Since dataframes are also objects, we can refer to them with their names (exm: `department_staff_list.csv`)
- We'll see an example of that in the next exercise

A note about this dataframe

- Understanding the data you use is very important. For this training, `department_staff_list` is a dataframe from your department.
- Let's use another function to see what's in there

```
glimpse(department_staff_list)
```

```
## Rows: 8,754
## Columns: 7
## $ sex                    <chr> "Female", "Male", "Female", "Female", "Femal…
## $ current_grade          <chr> "Director Fin. & Admin.", "Deputy Director (…
## $ date_of_birth          <dttm> 1964-09-25, 1983-02-19, 1965-06-15, 1974-04…
## $ date_of_first_appointment <dttm> 1994-08-31, 2008-11-01, 1990-12-01, 2013-05…
## $ senior_junior_staff    <chr> "senior", "senior", "senior", "senior", "sen…
## $ department             <chr> "State Protocol Dept", "State Protocol Dept"…
## $ years_of_service       <dbl> 30.209446, 16.038330, 33.957563, 11.542779, …
```

Exercise 5: Using our data

Imagine you want to quickly find out all the distinct departments listed in your staff dataset. In Excel, you might manually scroll or use 'Remove Duplicates.' In R, you can use the unique() function for this purpose.

1. Use the following code to find all the unique departments in `department_staff_list`:

```
unique_departments <- unique(department_staff_list$department)
```

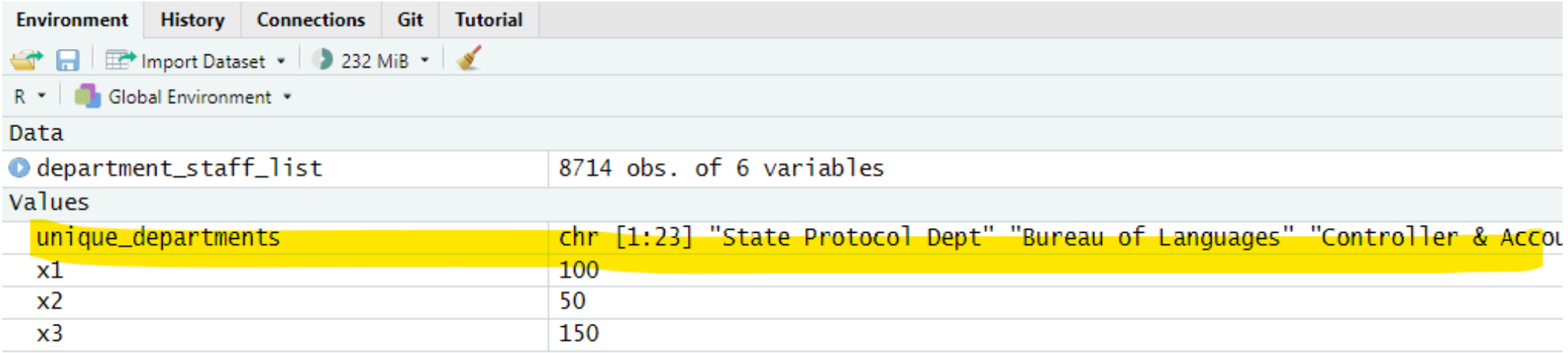
- Note that $ is used to access the `department` column from the dataset.
- Note that we are using the arrow operator (`<-`) to store the result

2.\ Use `print(unique_departments)` to display the unique departments:

```
print(unique_departments)
```

```
##  [1] "State Protocol Dept"             "Bureau of Languages"
##  [3] "Controller & Accountant General" "Stool Lands"
##  [5] "Co-operatives"                   "Factories Inspectorate"
##  [7] "Dept of Labour"                  "Procurement & Supply CM"
##  [9] "Management Services"             "Public Records"
```

Data in R


Environment
History
Connections
Git
Tutorial
Import Dataset
232 MiB
R
Global Environment
Data
department_staff_list
8714 obs. of 6 variables
Values
unique_departments
chr [1:23] "State Protocol Dept" "Bureau of Languages" "Controller & Accou
x1
100
x2
50
x3
150

Storing results in R

There is an important difference between using `<-` and not using it

- Not using `<-` **simply displays the result** in the console. The input dataframe will remain unchanged and the result **will not be stored**

```
unique(department_staff_list$department)
```

```
##  [1] "State Protocol Dept"             "Bureau of Languages"
##  [3] "Controller & Accountant General" "Stool Lands"
##  [5] "Co-operatives"                   "Factories Inspectorate"
##  [7] "Dept of Labour"                  "Procurement & Supply CM"
##  [9] "Management Services"             "Public Records"
## [11] "Public Works"                    "Rural Housing"
## [13] "Parks & Gardens"                 "Births & Deaths"
## [15] "Department of Comm Devt"         "Feeder Roads"
## [17] "Urban Roads"                     "Koforidua Training Center"
## [19] "Dept of Children"                "Dept of Gender"
## [21] "Social Welfare"                  "Information Services"
## [23] "Rent Control"
```

Storing results in R

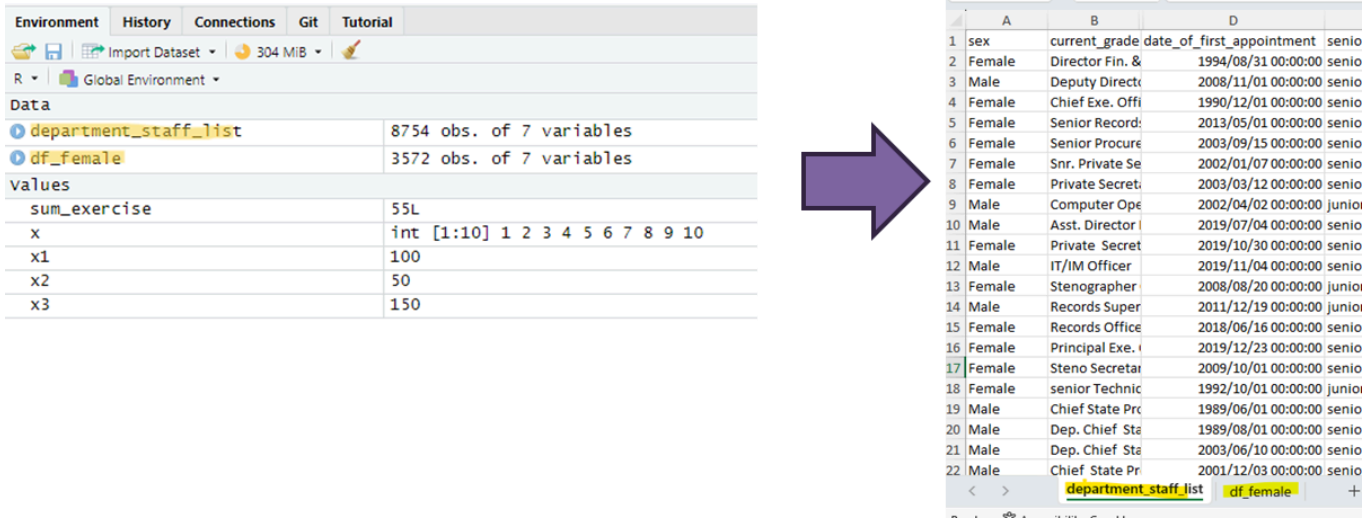
- Using `<-` tells R that we want to **store the result in a new object**, which is the object at the left side of the arrow. This time the result will not be printed in the console but the new object will show in the environment panel

```
unique_departments <- unique(department_staff_list$department)
```

Data in R

- R can store multiple dataframes in the environment. This is analogous to having different spreadsheets in the same ❎ Excel window

- Always remember that dataframes are just objects in R. R differentiates which dataframe the code refers to with the dataframe name



Questions?

Wrapping up

Wrapping up

Add code comments!

- Every line of code that starts with the pound symbol (`#`) will be ignored when R executes the code
- This means that you can add any clarifying comment with `#`. These are called **code comments**
- It's always a good practice to add code comments for yourself to later remember what the code is doing or to explain your code to others if you'll share it

Wrapping up

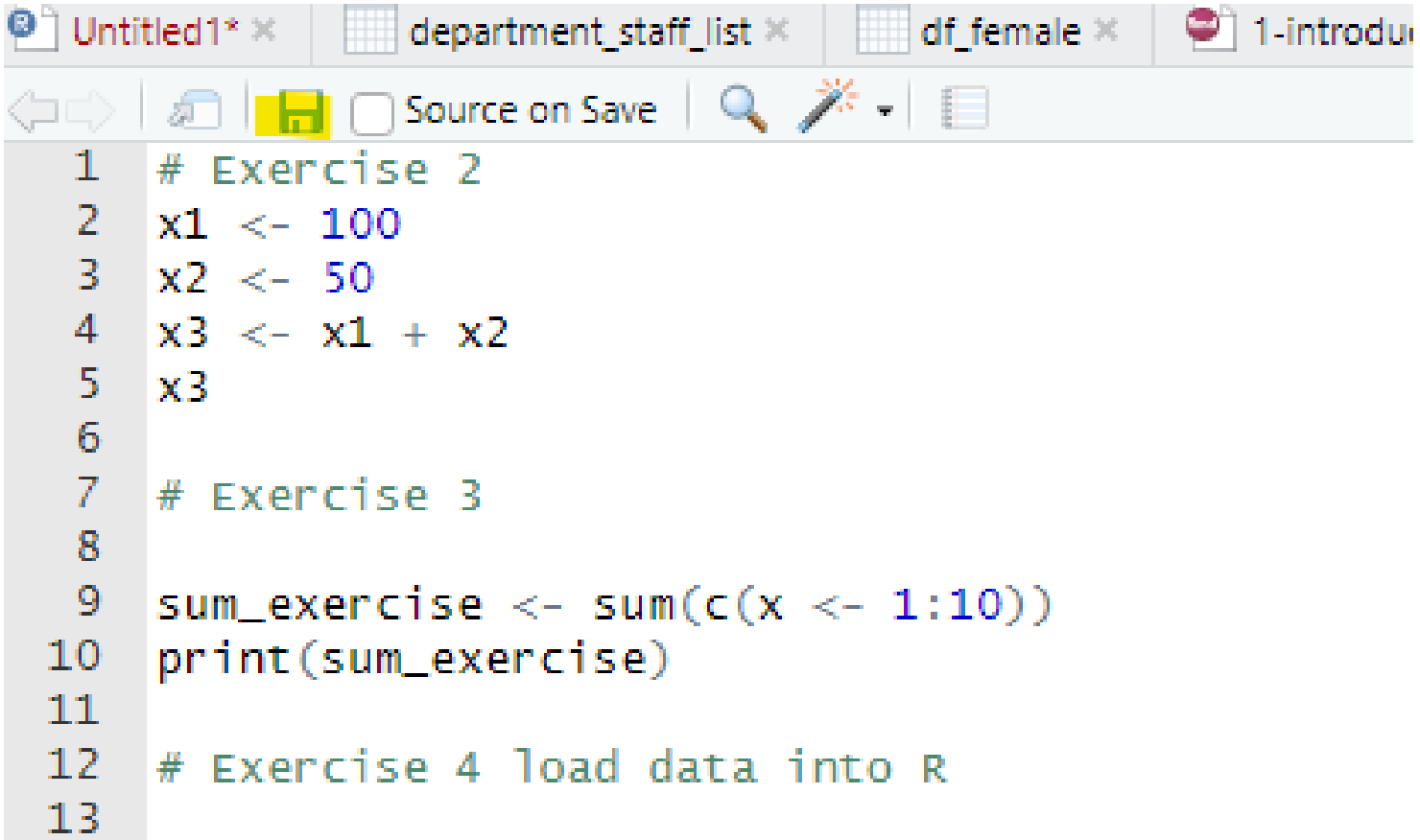
- Try adding code comments to your script so you will remember which part corresponds to each exercise

```r
# Exercise 1: writing code in the console.

# Exercise 2: creating an object
x1 <- 100
x2 <- 50
x3 <- x1 + x2
x3

# Exercise 3: Using a function sum()

sum_exercise <- sum(c(x <- 1:10))
print(sum_exercise)

# Exercise 4: load data into R
# In the training we followed the point and click approach the steps are:
# file> import from text base> select the file
department_staff_list <- read.csv("data/department_staff_list.csv")

# Exercise 5: find the unique deparments in our list

unique_departments <- unique(department_staff_list$department)
```

Wrapping up

Always save your work!

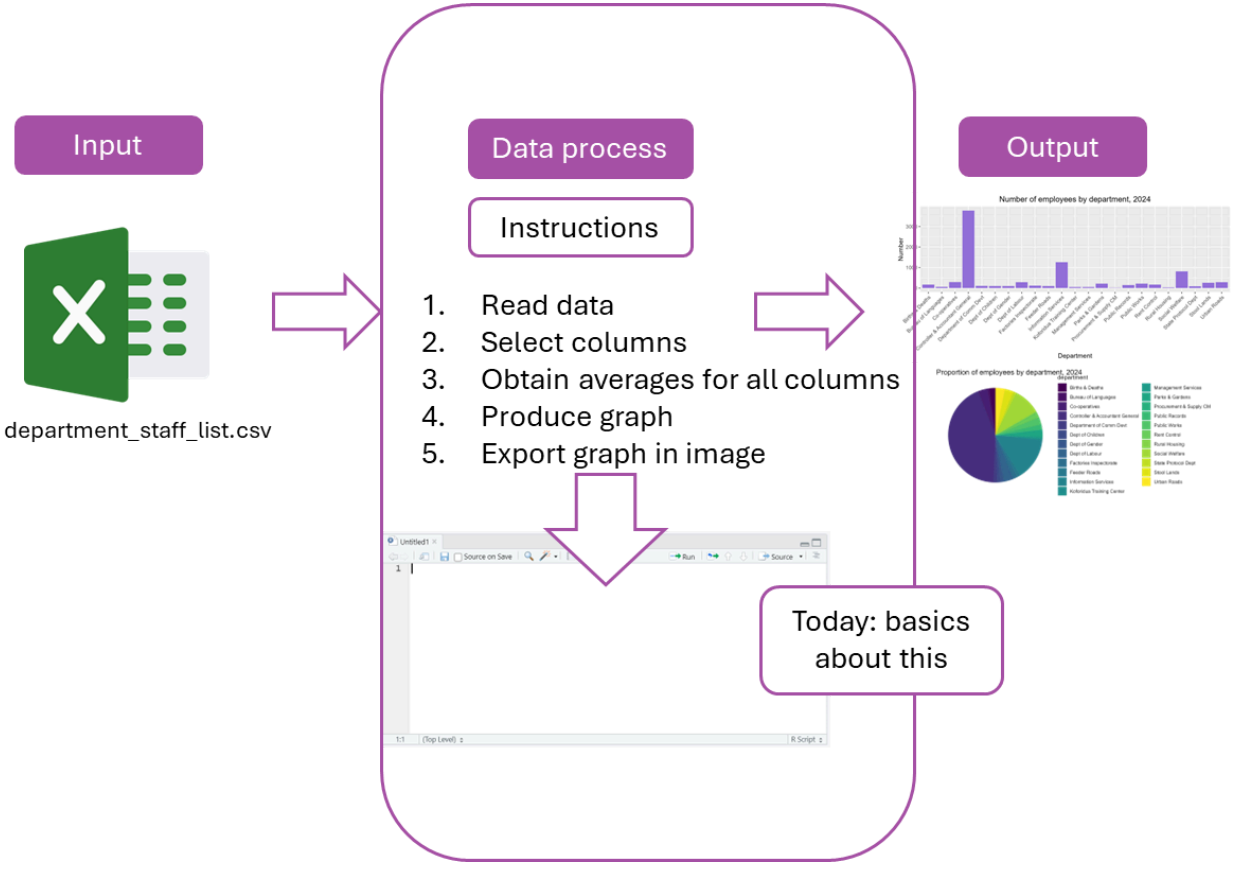
- Click the floppy disk icon to save your work
- Select a location for your file and remember where you're saving it



Wrapping up

This session

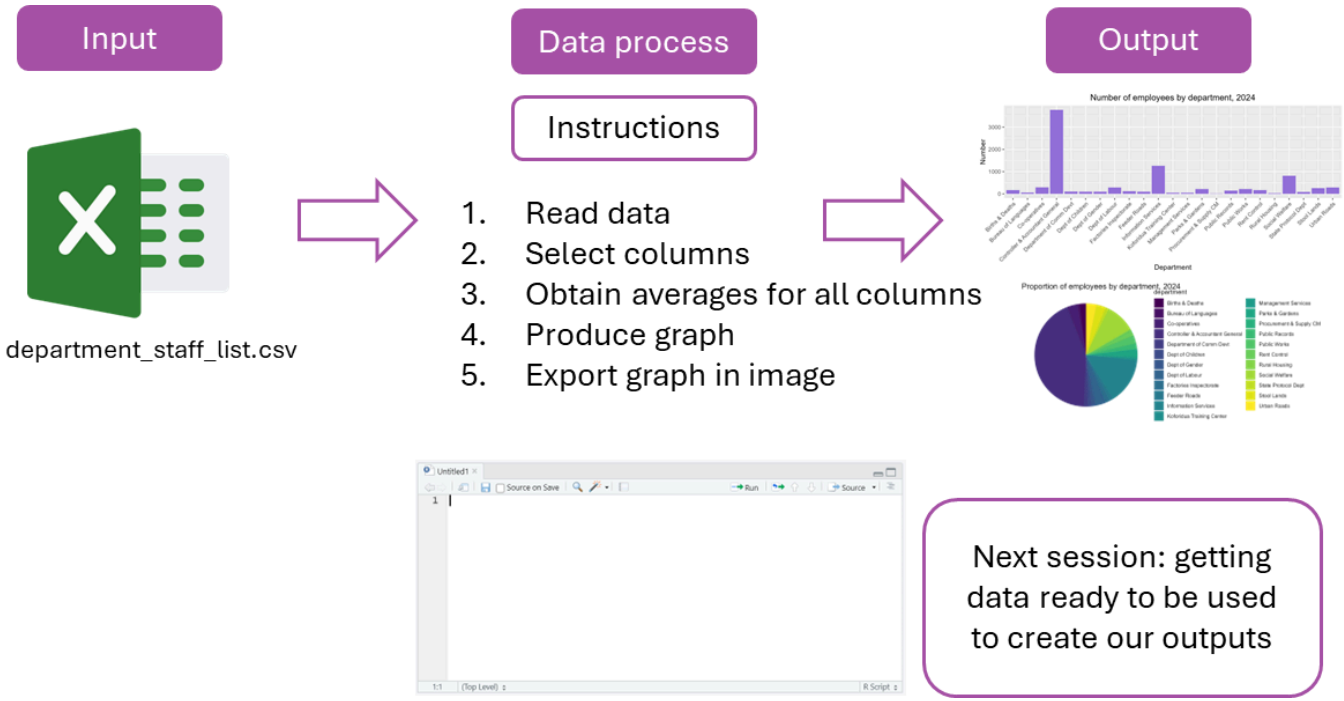
This first session focused on the basics for writing R code



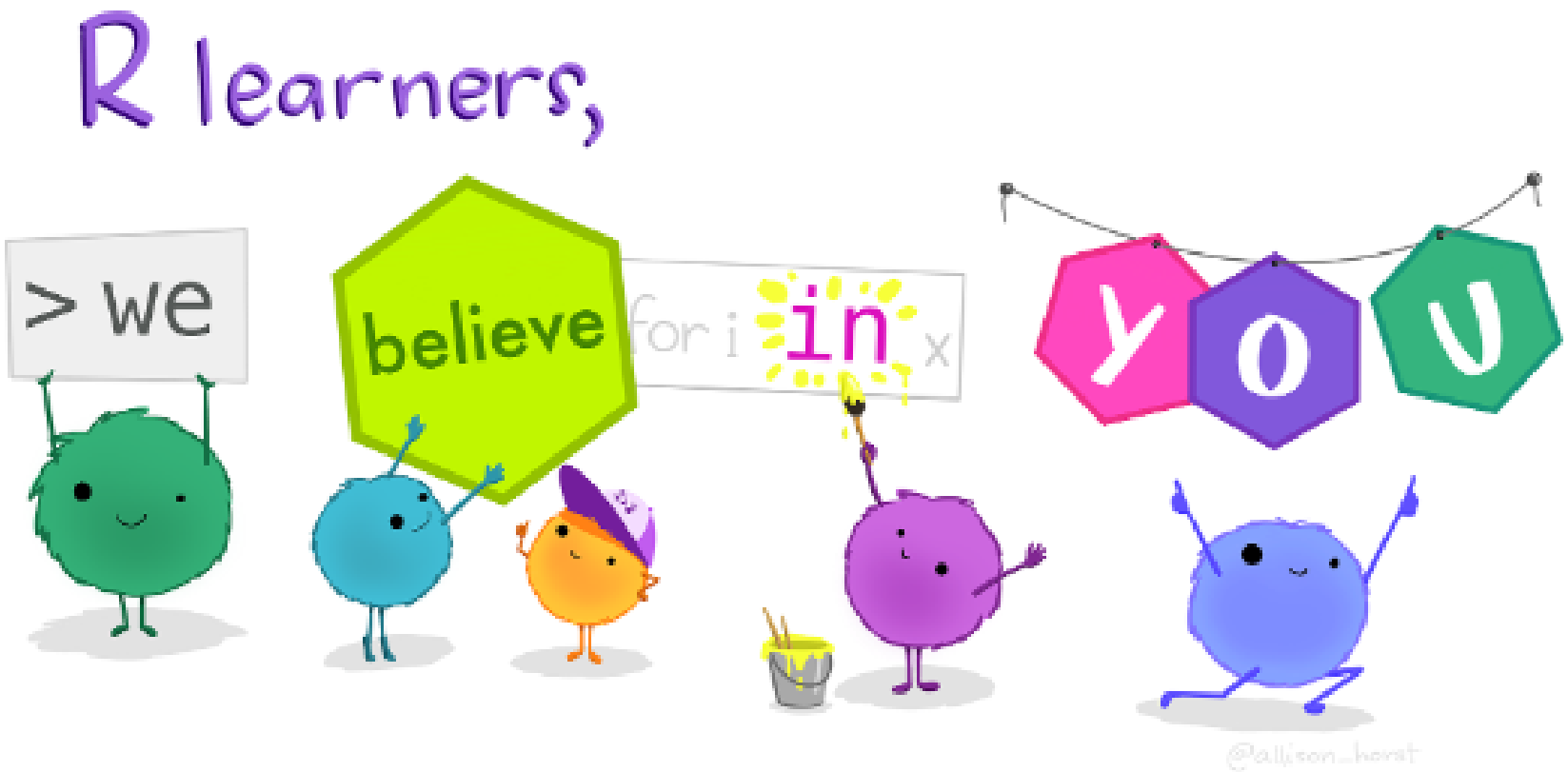
Wrapping up

Next session

In the next session we will learn how to get data ready to be exported as outputs



Thanks! // ¡Gracias! // Obrigado!



Appendix

Appendix

Object Types: character strings

- Character strings are collections of alphanumeric characters usually representing words or texts, or just characters in general

```
s1 <- "Hello World"
print(s1)
```

```
## [1] "Hello World"
```

- Strings characters are **always enclosed in quotes** (`" "`)
- They are usually referred to as just **strings**

Appendix

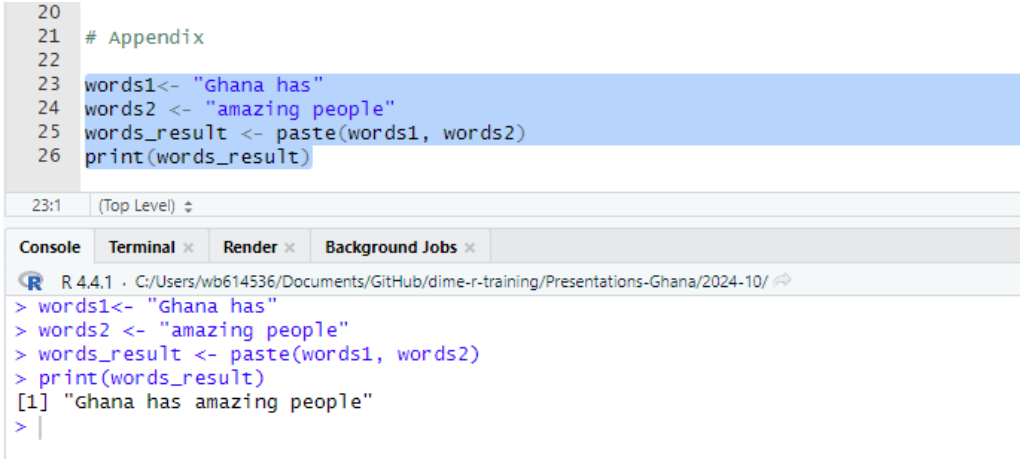
Exercise: create and operate character strings

1. Create a character string object with the words `"Ghana has"` and name it `words1`
2. Create a second string with the words `"amazing people"` and name it `words2`
    - Don't forget to use `<-` to create the string objects
    - Remember to include the quotes: `" "`
3. Use the following code to concatenate `words1` and `words2`, save the result in `words_result`, and print it:

```
words_result <- paste(words1, words2)
print(words_result)
```

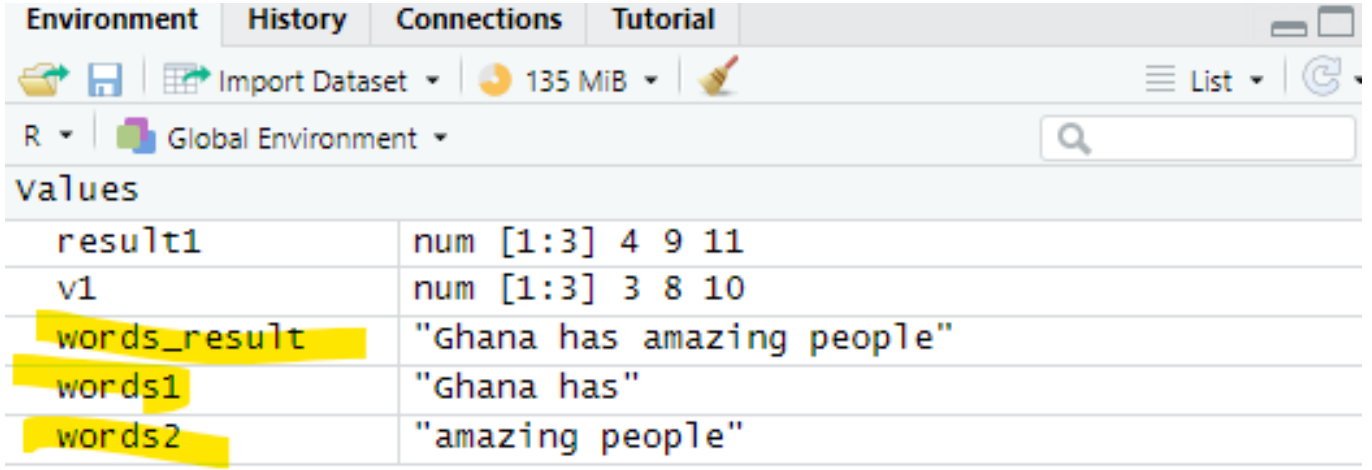
Appendix

Object Types: character strings



Appendix

Object Types: character strings



Appendix

Object Types: Vectors

- Vectors are a collection of values **with a single dimension**, instead of being organized in rows and columns as dataframes
- You can think of a vector in R as a single column in an Excel spreadsheet or an R dataframe
- You can create vectors with the function `c()`, the vector elements are separated by commas

```
my_vector <- c(4, 8, 2, 5)
```

Appendix

Exercise: Create vectors

1- Create a vector with the elements 3, 8, and 10 and name it `v1`:

```
v1 <- c(3, 8, 10)
```

2- create a vector named `result1` with the sum of `v1` + 1.

```
result1 <- v1+1
```

3- Print `v1` and observe the results

```
print(result1)
```

```
## [1]  4  9 11
```

Appendix

Exercise: Create vectors

```
11
12  # Exercise 5
13  v1 <- c(3, 8, 10)
14  result1 <- v1 + 1
15  print(result1)
16
```

17:1 (Top Level)

Console | Terminal | Render | Background Jobs

R 4.4.1 · C:/Users/wb614536/Documents/GitHub/dime-r-training/Presentations-Ghana/2024-

```
> # Exercise 5
> v1 <- c(3, 8, 10)
> result1 <- v1 + 1
> print(result1)
[1]  4  9 11
>
```

Appendix

Exercise: Create vectors