

# 1 - Introduction to statistical programming in R

## R labs - Manage Successful Field Research

---

The World Bank | [WB Github](#)

June 2025

# Before we begin

Go to [http://bit.ly/msfr25\\_materials](http://bit.ly/msfr25_materials) and download  
**1-intro-R.pdf**

# Table of contents

1. About the training
2. Statistical programming
3. Writing R code
4. Data in R
5. Functions in R
6. R libraries
7. File paths
8. Piping

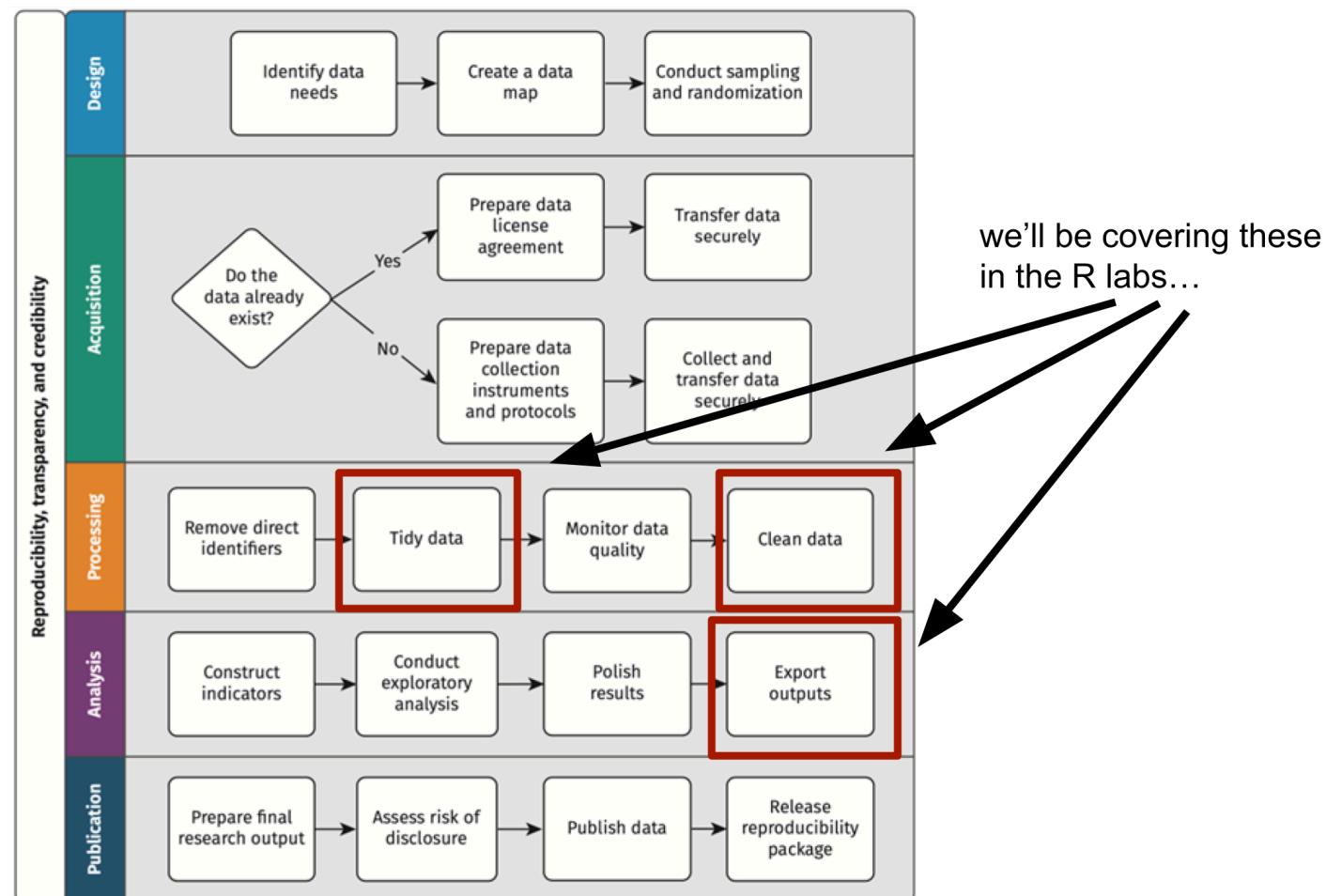
# About the R lab sessions of MSFR

---

# About the R lab sessions of MSFR

- This is an **introduction** to statistical programming and reproducible research in R
- Content is focused on usual tasks for field research

# About the R lab sessions of MSFR



# Statistical programming

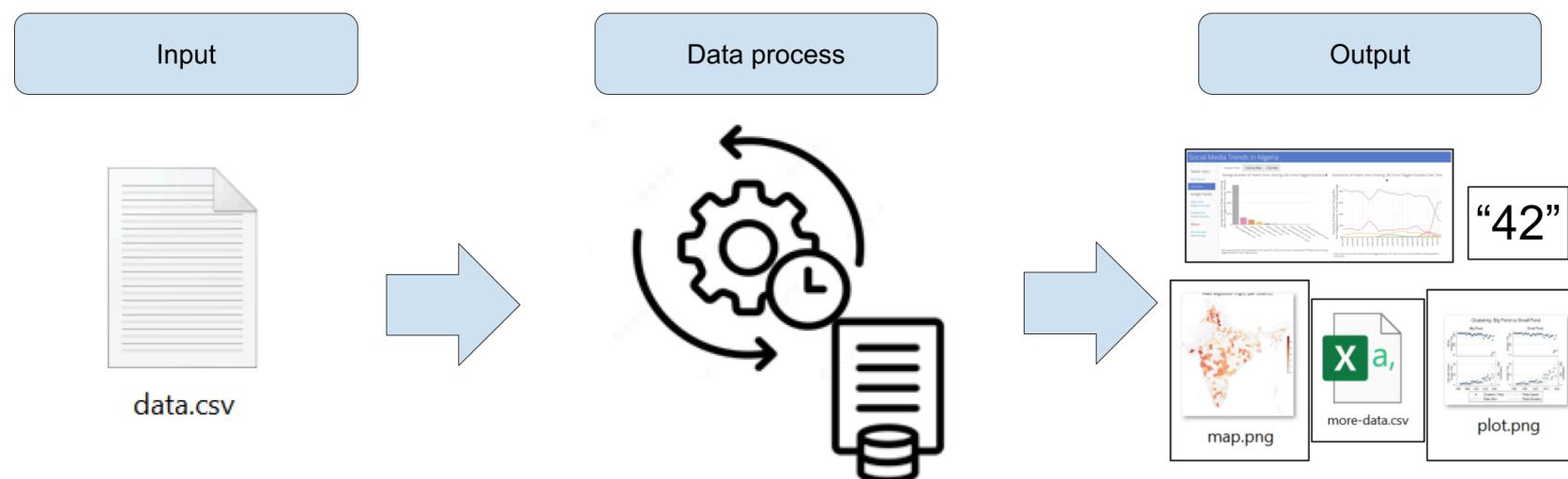
---

# Statistical programming

## Data work in development research

More generally, for any of the processes in development research, we'll call data work everything that:

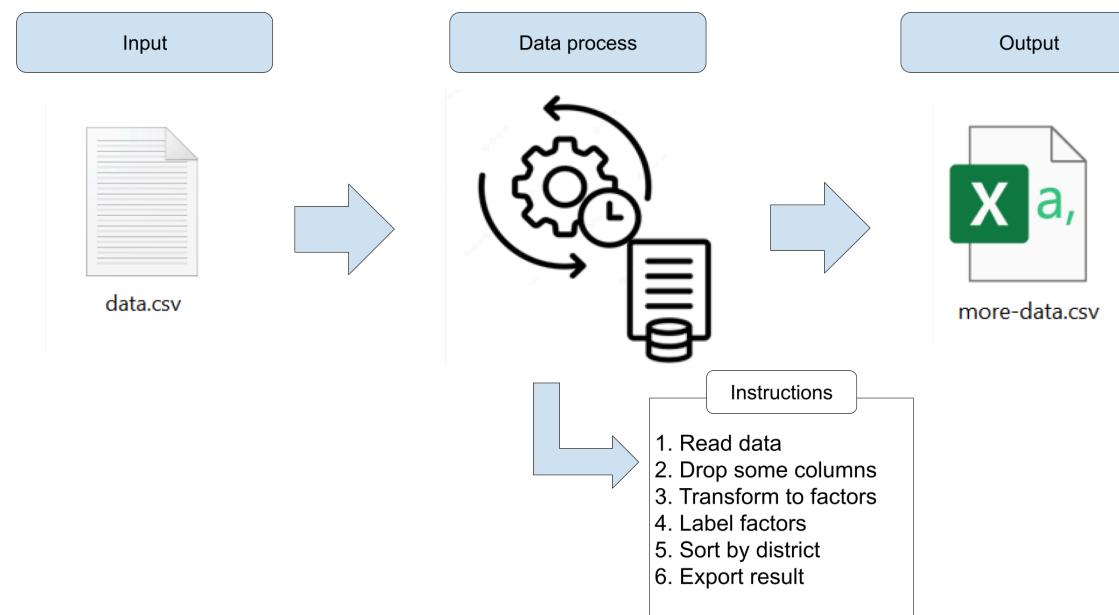
1. Starts with a data input
2. Runs some process with the data
3. Produces an output with the result



# Statistical programming

## So what do we mean by statistical programming?

- Programming consists of producing instructions for a computer to do something
- In the context of data work, that "something" is statistical analysis, mathematical operations, or data transformations
- Hence, statistical programming consists of producing instructions so our computers will conduct statistical analysis on data
- From now on, we're going to refer to these instructions for data processes as **code**



# Statistical programming

## Reproducibility

In science, the formal definition of reproducibility was provided by Bollen et al. (2015):

*the ability [...] to duplicate the results of a prior study using the same materials and procedures as were used by the original investigator*

this translates to data work as:

**The ability to reproduce outputs using the same code and data inputs.**

This is also usually referred to as *computational reproducibility*.

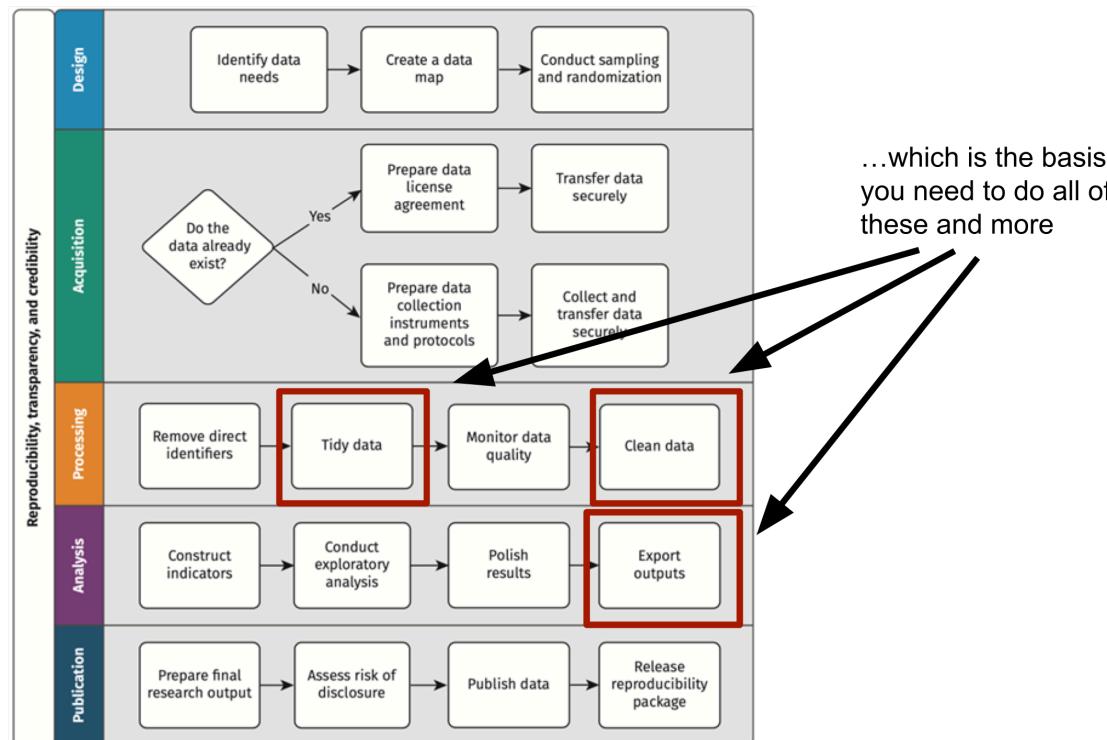
# Writing R code

---

# Writing R code

## How to write R code?

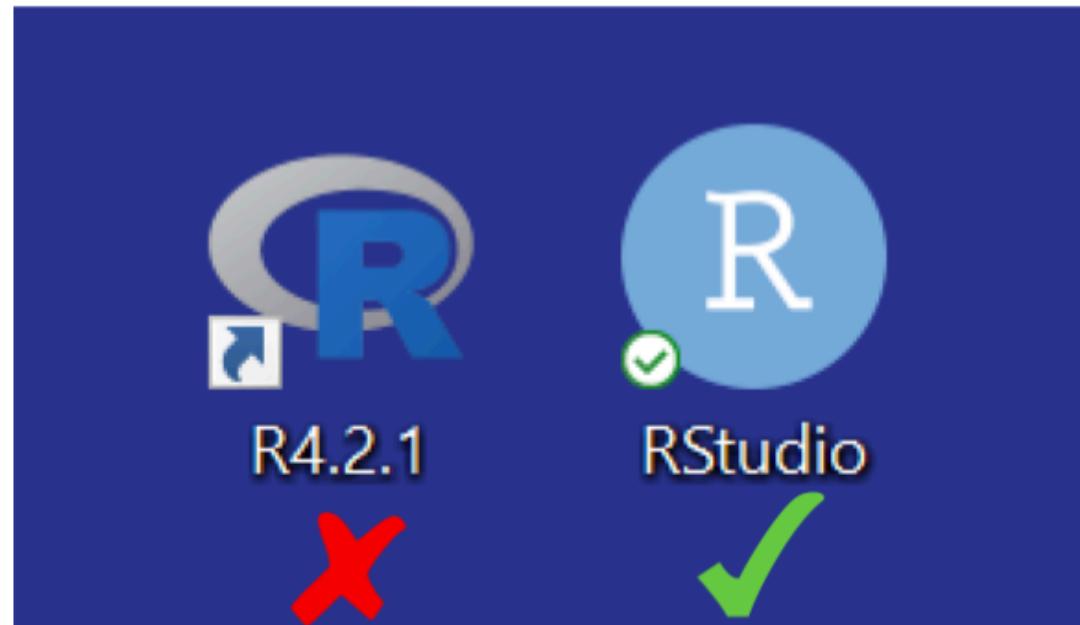
The rest of this session focuses on the basics of writing R code...



# Writing R code

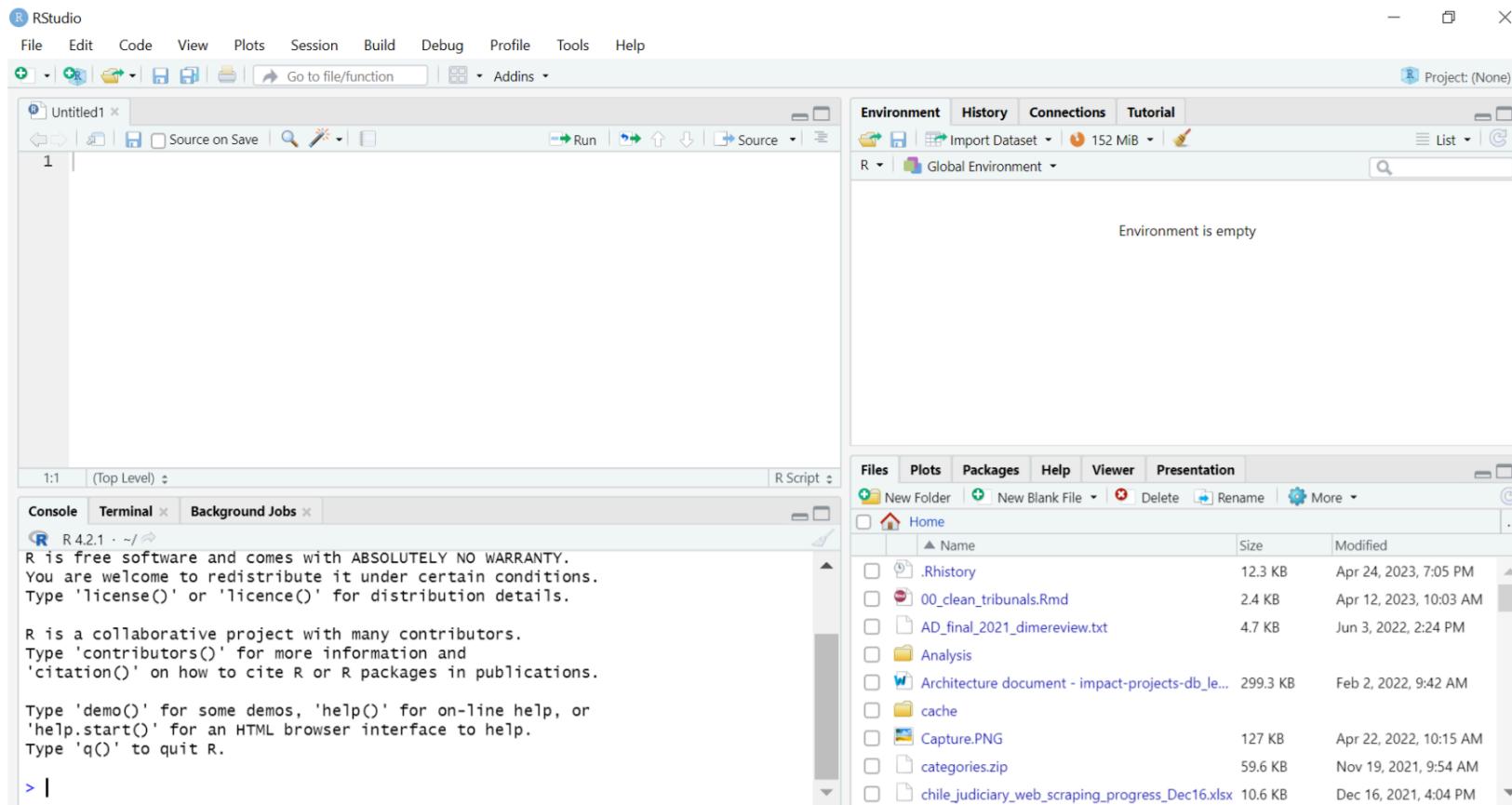
## How to write R code?

- We'll use RStudio to write R code in this training
- Now open RStudio in your computer
- Please make sure you're opening RStudio and not R



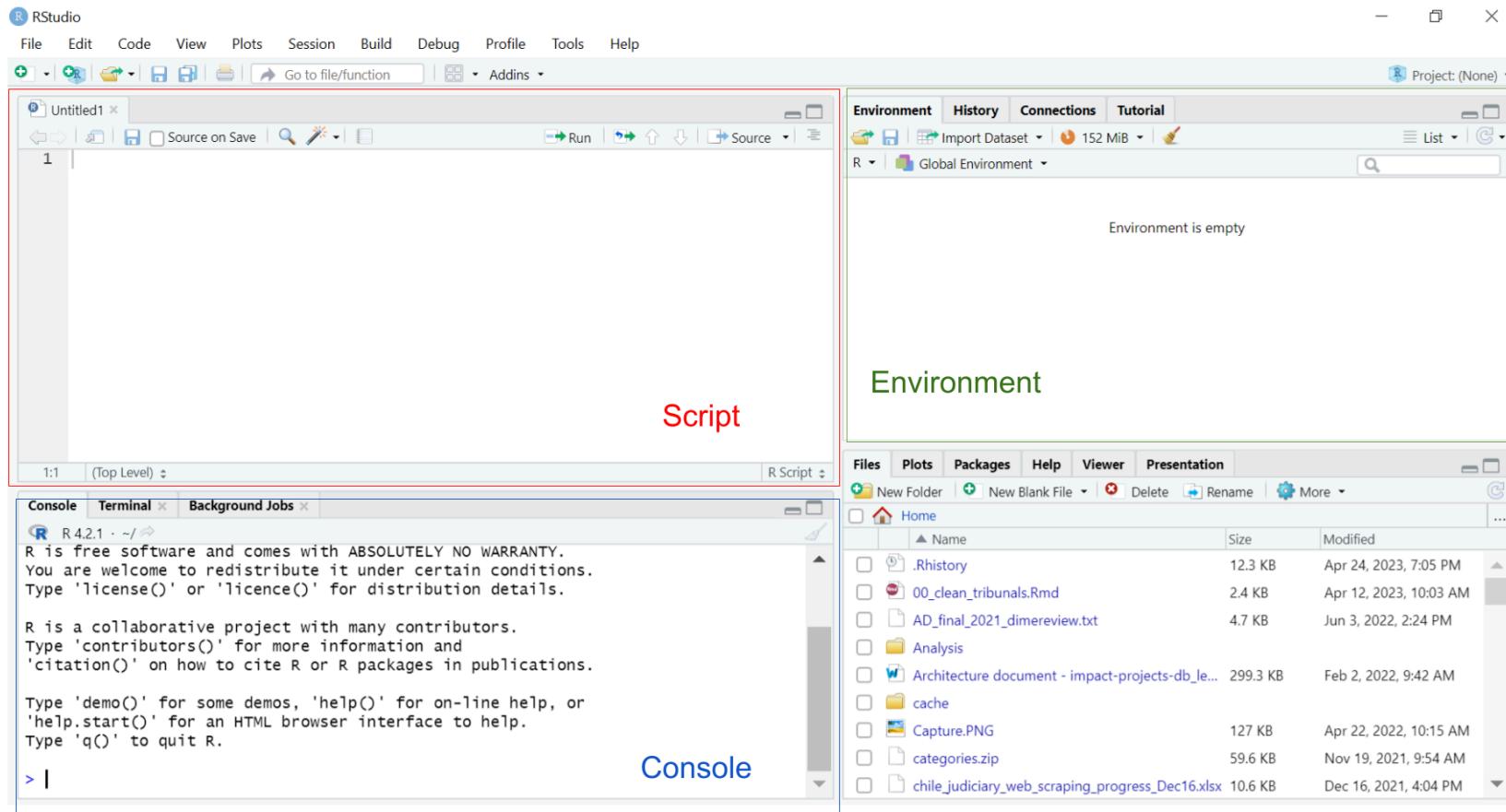
# Writing R code

## RStudio interface



# Writing R code

## RStudio interface



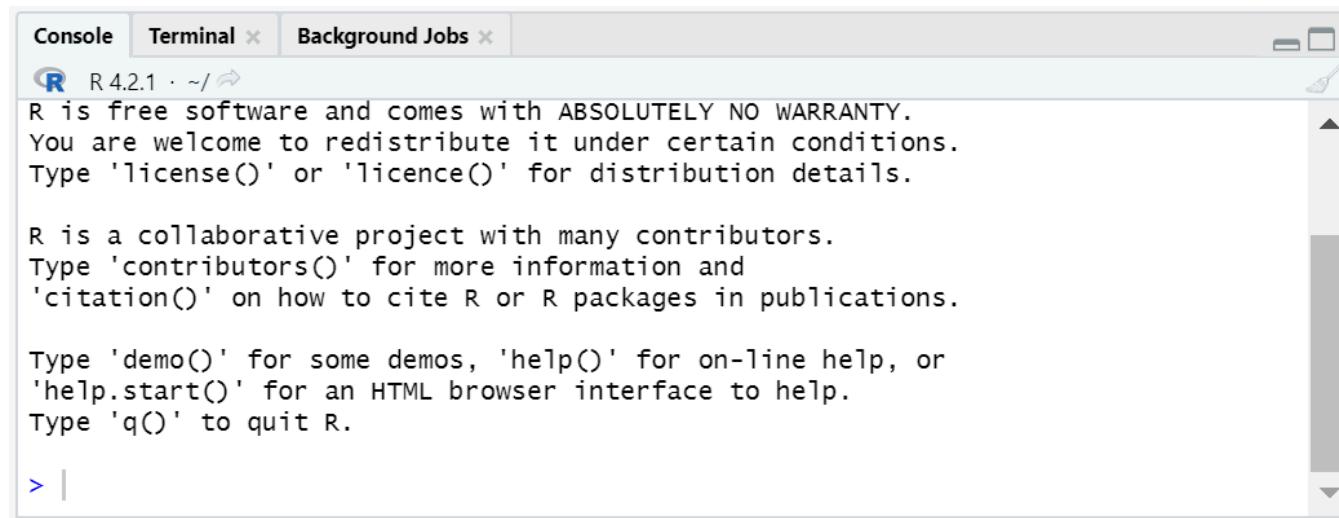
# Writing R code

## Exercise 1: writing code in the console

1. Write the following code in the console of RStudio

- o `print("hello world")`
- o Make sure to include the quotes: `" "`

2. Press Enter to run the code

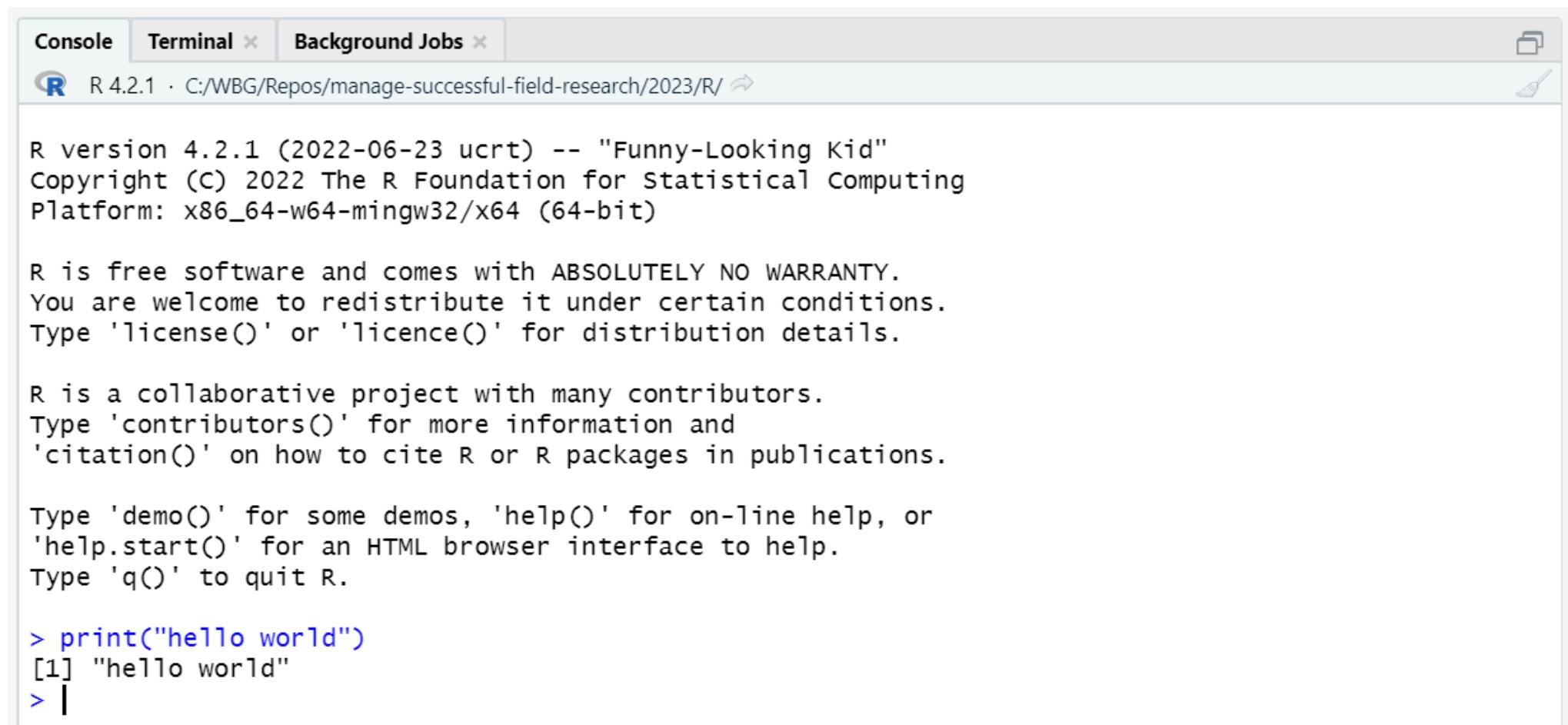


The screenshot shows the RStudio interface with the 'Console' tab selected. The title bar indicates 'R 4.2.1 · ~/'. The console window displays the standard R startup message:

```
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.
```

A blue cursor arrow is visible at the bottom left of the console area, indicating where the user can type their own code.

# Writing R code



The screenshot shows the RStudio interface with the 'Console' tab selected. The title bar indicates 'R 4.2.1 · C:/WBG/Repos/manage-successful-field-research/2023/R/'. The console window displays the standard R startup message, followed by a user command and its output.

```
R version 4.2.1 (2022-06-23 ucrt) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> print("hello world")
[1] "hello world"
> |
```

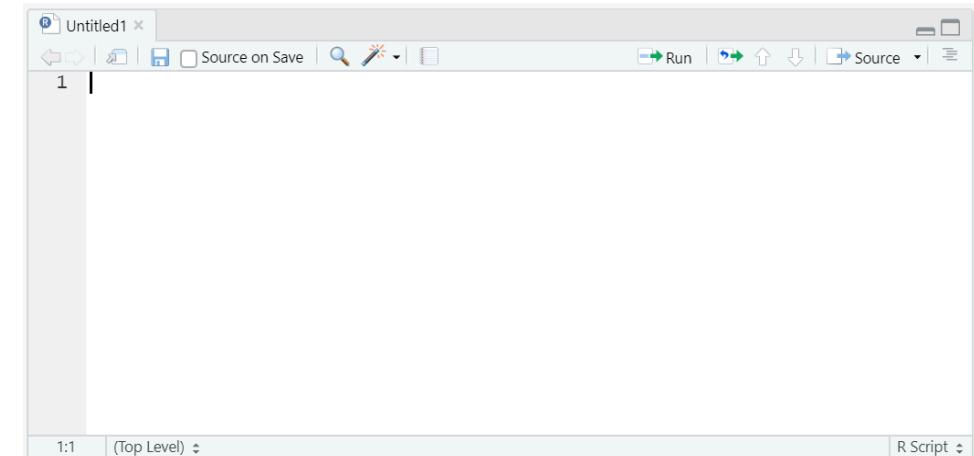
# Writing R code

## Exercise 2: writing a short script

1. Open a new script with **File** >> **New File** >> **R Script**

2. Write or copy the following text into the script

```
print("hello")
print("bonjour")
print("hola")
print("bom dia")
```



1. Select the text you introduced with your mouse
2. Press "Run"

# 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

The screenshot shows the RStudio interface. The top panel is a script editor titled "Untitled1\*" containing four lines of R code:

```
1 print("hello")
2 print("bonjour")
3 print("hola")
4 print("bom dia")
```

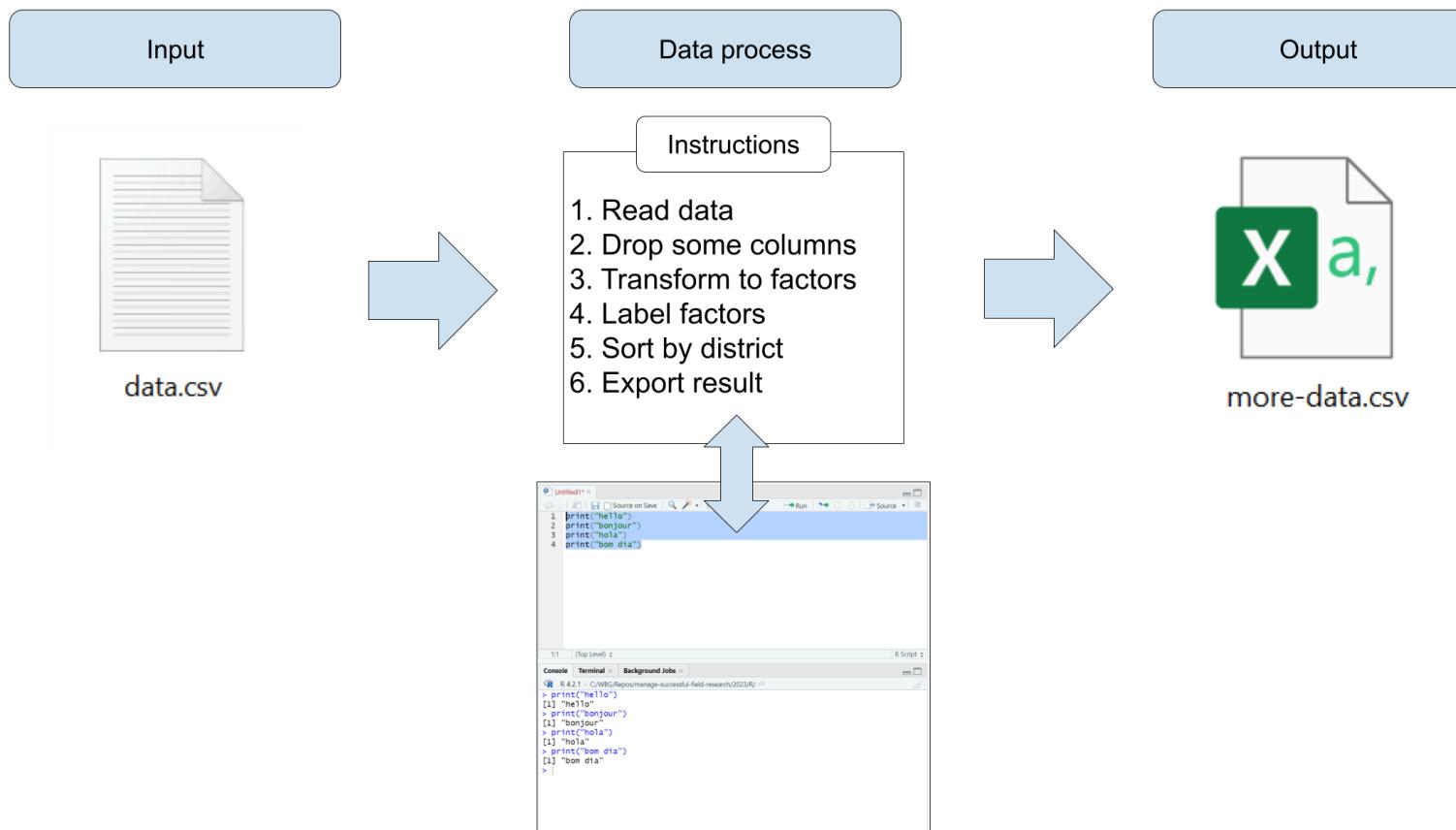
The bottom panel is a console window showing the execution of the script. It displays the R logo, the version (R 4.2.1), the working directory (C:/WBG/Repos/manage-successful-field-research/2023/R/), and the output of each printed line:

```
> print("hello")
[1] "hello"
> print("bonjour")
[1] "bonjour"
> print("hola")
[1] "hola"
> print("bom dia")
[1] "bom dia"
> |
```

# Writing R code

## R scripts

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



# Writing R code

## Creating objects in R

- Remember we also mentioned the environment panel? that's where R keeps track of objects
- Objects are representations of data that currently exist in R's memory
  - 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 (`<-`)
- Example: creating an object called `x`

```
x <- 10
```

# Writing R code

## Creating objects in R

- After an object is created, we can refer to it using its name

```
x <- 10
```

```
print(x)
```

```
## [1] 10
```

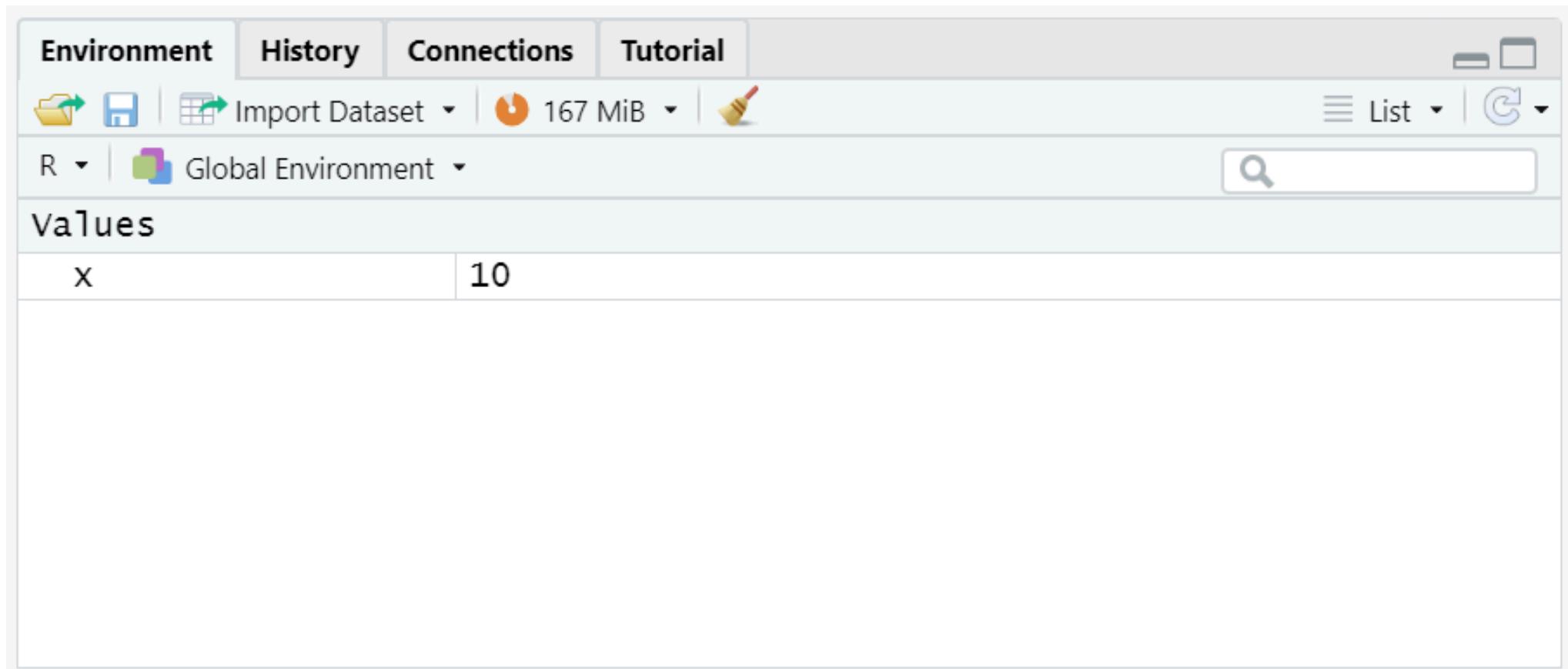
```
print(x + 5)
```

```
## [1] 15
```

# Writing R code

## Creating objects in R

- After an object is created, it will show in the environment panel



# Writing R code

## Exercise 3: operations with objects

1. Create an object called `x1` in your script and assign to it the value 50
2. Create an object called `x2` and make it equal to 15
3. Create an object called `x3` and make it the sum of `x1` and `x2`
4. Print `x3`. It should show the value 65

```
x1 <- 50
x2 <- 15
x3 <- x1 + x2
print(x3)
```

# Writing R code

The screenshot shows the RStudio IDE interface with the following components:

- Script Editor (Top Left):** An untitled R script containing the following code:

```
1 print("hello")
2 print("bonjour")
3 print("hola")
4 print("bom dia")
5
6 x1 <- 50
7 x2 <- 15
8 x3 <- x1 + x2
9 print(x3)
```
- Global Environment (Top Right):** A table showing variable values:

Values	
x	10
x1	50
x2	15
x3	65
- File Browser (Bottom Right):** A file tree view showing the directory structure:

```
C: > WBG > Repos > manage-successful-field-research > 2023 > R
```

Name	Size	Modified
..	0 B	May 2, 2023, 10:23 PM
.Rhistory	218 B	May 2, 2023, 11:22 PM
Code	121 B	May 2, 2023, 3:31 PM
Data		
Presentations		
R.Rproj		
README.md		
- Console (Bottom Left):** Shows the R session history:

```
R 4.2.1 · C:/WBG/Repos/manage-successful-field-research/2023/R/ 
> x1 <- 50
> x2 <- 15
> x3 <- x1 + x2
> print(x3)
[1] 65
>
```

# Writing R code

- These are the very basics of writing R code. You'll become more familiarized with it in this and the next sessions
- Now we know how to use RStudio to write R code and produce scripts
- However, we haven't still introduced the data to our data work. That comes next

# Data in R

---

# Data in R

## Exercise 4: Loading data into R

1. Click on this link: [http://bit.ly/msfr25\\_materials](http://bit.ly/msfr25_materials) and download the file `treat_status.csv`
2. In RStudio, go to `File > Import Dataset > From Text (base)` and select the file `treat_status.csv`
  - If you don't know where the file is, check in your `Downloads` folder
3. Make sure to select `Heading > Yes` in the next window
4. Select `Import`

The screenshot shows the 'Import Dataset' dialog in RStudio. The 'Name' field is set to 'treat\_status'. The 'Input File' pane displays the contents of 'treat\_status.csv', which contains three columns: 'vid', 'Treatment', and 'district'. The 'Data Frame' pane shows the same data as a structured table.

**Import Dataset**

Name: treat\_status

Encoding: Automatic

Heading: Yes

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double ("")

Comment: None

na.strings: NA

Strings as factors

**Input File**

	vid	Treatment	district
1	1,0	District B	
2	2,1	District B	
3	3,1	District B	
4	4,0	District B	
5	5,0	District B	
6	6,1	District A	
7	7,1	District B	
8	8,0	District A	
9	9,0	District B	
10	10,1	District C	
11	11,0	District C	
12	12,0	District C	
13	13,0	District B	
14	14,0	District B	
15	15,0	District A	
16	16,0	District A	
17	17,0	District B	
18	18,1	District C	

**Data Frame**

vid	Treatment	district
1	0	District B
2	1	District B
3	1	District B
4	0	District B
5	0	District B
6	1	District A
7	1	District B
8	0	District A
9	0	District B
10	1	District C
11	0	District C
12	0	District C
13	0	District B
14	0	District B
15	0	District A
16	0	District A
17	0	District B

# Data in R

- If you did this correctly, you will note that a viewer of the data now appears in RStudio
- You can click on the **x** next to `treat_status` to return to the script
- To open the viewer again, use the code: `View(treat_status)` (notice the uppercase "V")

The screenshot shows the RStudio interface with a data viewer window open. The window title is 'treat\_status'. The viewer displays a table with 9 rows and 3 columns. The columns are labeled 'vid', 'Treatment', and 'district'. The data shows 5 entries from District B and 4 entries from District A. The 'Treatment' column has values 1, 2, 3, 4, 5, 6, 7, 8, and 9 respectively. The 'district' column has values 'District B' for the first five rows and 'District A' for the last four. The table includes a header row with column names and row numbers. A status bar at the bottom indicates 'Showing 1 to 9 of 80 entries, 3 total columns'.

	vid	Treatment	district
1	1	0	District B
2	2	1	District B
3	3	1	District B
4	4	0	District B
5	5	0	District B
6	6	1	District A
7	7	1	District B
8	8	0	District A
9	9	0	District B

Showing 1 to 9 of 80 entries, 3 total columns

# Data in R

- Additionally, you will now see an object named `treat_status` in your environment

The screenshot shows the RStudio interface with the 'Environment' tab selected. In the Global Environment pane, there is one object listed:

treat_status	80 obs. of 3 variables
values	
x1	50
x2	15
x3	65

The 'treat\_status' object is highlighted with a yellow background.

# Data in R

- Remember we mentioned objects before? For R, `treat_status` is an object just like `x1`, `x2`, or `x3`
- The difference is that `treat_status` 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**

The screenshot shows the RStudio interface with the 'Environment' tab selected. The top bar includes tabs for Environment, History, Connections, and Tutorial, along with various icons for file operations and search. Below the tabs, there are dropdown menus for 'R' and 'Global Environment'. A search bar is also present. The main area displays a data frame named 'treat\_status' with the following details:  
- Data: 80 obs. of 3 variables  
- Values:

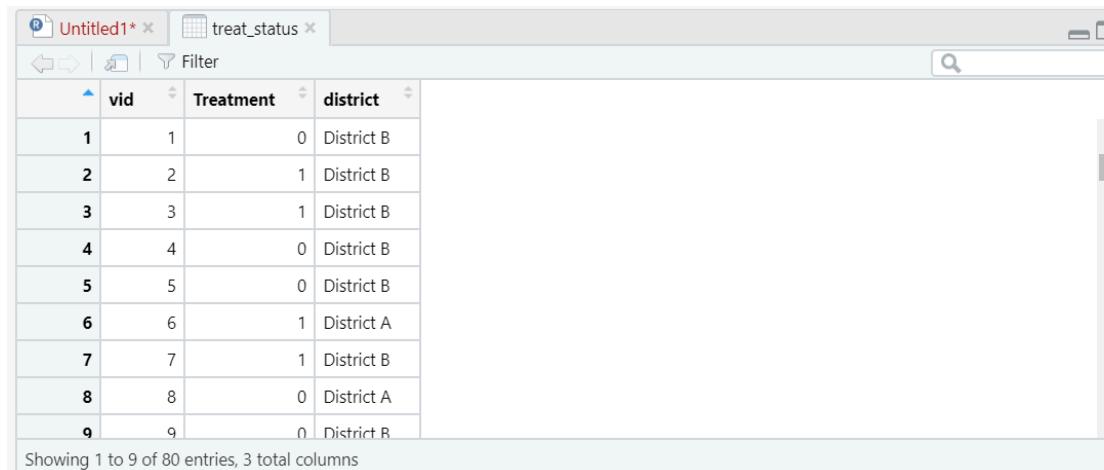
x1	50
x2	15
x3	65

# Data in R

## A note about this dataframe

Knowing the data you use is extremely important. For the lab sessions, `treat_status` is a dataframe with data about villages, their treatment status in the project, and the districts they are located.

- Variable (column) `vid` is the village identifier
- `Treatment` is a dummy with information about the village treatment status
- `district` is the name of the village district



	vid	Treatment	district
1	1	0	District B
2	2	1	District B
3	3	1	District B
4	4	0	District B
5	5	0	District B
6	6	1	District A
7	7	1	District B
8	8	0	District A
9	9	0	District B

Showing 1 to 9 of 80 entries, 3 total columns

## Exercise 5: Understanding data in R

1. Use the following code to subset (filter) `treat_status` and leave only the villages in the treatment group:

```
subset(treat_status, Treatment == 1)
```

- Note that the `T` in `Treatment` needs to be **uppercase** because that's how the variable is named in the dataframe
- Note that there are **two equal signs** in the condition, not one

2. Use `View(treat_status)` to visualize the dataframe again and see how it changed (note the uppercase "V")

# Data in R

Does anything look out of place?

vid	Treatment	district
1	0	District B
2	1	District B
3	1	District B
4	0	District B
5	0	District B
6	1	District A
7	1	District B
8	0	District A
9	0	District B

Showing 1 to 9 of 80 entries, 3 total columns

# Data in R

- Indeed, the dataframe `treat_status` didn't change
- That is because we didn't use the arrow operator (`<-`) to store the result in an object
- Instead, R only printed the result in the console for us (and nothing else)

The screenshot shows the RStudio interface with the following components:

- Code Editor:** Shows an R script with the following code:

```
1 print("hello")
2 print("bonjour")
3 print("hola")
4 print("bom dia")
5
6 x1 <- 50
7 x2 <- 15
8 x3 <- x1 + x2
9 print(x3)
10
11 subset(treat_status, Treatment == 1)
12 View(treat_status)
```

The line `subset(treat_status, Treatment == 1)` is highlighted with a red box.
- Run Button:** A red box highlights the "Run" button in the toolbar.
- Environment Tab:** Shows the `treat_status` data frame with 80 observations and 3 variables. The values for `x1`, `x2`, and `x3` are listed as 50, 15, and 65 respectively. This section is also highlighted with a red box.
- Console Tab:** Shows the output of the code execution:

```
62 64 1 District B
63 65 1 District C
65 67 1 District A
66 68 1 District A
68 70 1 District A
72 74 1 District B
75 77 1 District A
76 78 1 District B
79 81 1 District C
80 82 1 District A
```

This section is highlighted with a red box.
- Message:** Below the environment pane, the text "No changes here" is displayed.
- File Explorer:** Shows the project structure with files like `.Rhistory`, `00_clean_tribunals.Rmd`, `AD_final_2021_dimereview.txt`, `Analysis`, and `Architecture document - impact-pr...`.

## Exercise 6: Operating with data and storing the result

- Subset the dataframe again and save the result into a new dataframe called `treated_villages`. You can use this code for that:

```
treated_villages <- subset(treat_status, Treatment == 1)
```

- Use `View(treated_villages)` to visualize the result (notice the uppercase "V")

# Data in R

Now the resulting dataframe looks correct!

	vid	Treatment	district
2	2	1	District B
3	3	1	District B
6	6	1	District A
7	7	1	District B
10	10	1	District C
18	18	1	District C
20	20	1	District B

Showing 1 to 7 of 40 entries, 3 total columns

# Data in R

Note that this time R didn't print the resulting dataframe in the console, it only showed the code we were running. Also, now the new dataframe appears in the environment.

The screenshot shows the RStudio interface with the following components:

- Code Editor:** Displays an R script with the following code:

```
1 print("bonjour")
2 print("hola")
3 print("bom dia")
4
5 x1 <- 50
6 x2 <- 15
7 x3 <- x1 + x2
8 print(x3)
9
10 subset(treat_status, Treatment == 1)
11 View(treat_status)
12
13 treated_villages <- subset(treat_status, Treatment == 1)
14 View(treated_villages)
15
```
- Run Button:** A red box highlights the "Run" button in the toolbar.
- Console:** Shows the command `> treated_villages <- subset(treat_status, Treatment == 1)` and its response `>`.
- Environment Tab:** Shows the Global Environment pane with the following data:

Data	treat_status    80 obs. of 3 variables treated_villag... 40 obs. of 3 variables
Values	x1                50 x2                15 x3                65
- File Explorer:** Shows the file structure in the "Files" tab.

# Data in R

- R can store multiple dataframes in the environment. This is analogous to having different spreadsheets in the same Excel window
- Remember that dataframes are just objects in R. R differentiates which dataframe the code refers to with the dataframe name

The screenshot shows the RStudio interface with the 'Environment' tab selected. The top bar includes tabs for Environment, History, Connections, Tutorial, and several icons for file operations like Import, Save, and Undo. Below the tabs, there's a dropdown for 'R' and 'Global Environment'. A search bar is also present. The main area is divided into sections: 'Data' and 'values'. Under 'Data', there are two entries: 'treat\_status' and 'treated\_villag...', both described as containing 80 observations and 3 variables. Each entry has a blue play icon and a grid icon. Under 'values', there is a table with three rows: x1 (50), x2 (15), and x3 (65). At the bottom of the interface, there are navigation arrows.

# Functions in R

---

# Functions in R

- Functions are how we apply operations to objects in R
- We have used a few functions in the previous exercises. For example, `subset()` and `View()` are functions
- Everything that has a name plus parentheses is a function in R

```
subset(treat_status, Treatment == 1)
```

# Functions in R

Functions have the following elements:

```
subset(treat_status, Treatment == 1)
```

function name                                      arguments

- **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 and are separated by commas
  - 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 (`<-`)

```
treated_villages <- subset(treat_status, Treatment == 1)
```

- As we saw in exercise 5, the results of a function will only be printed in the console if you don't store them

# R libraries

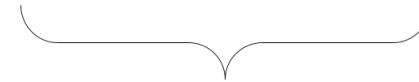
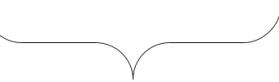
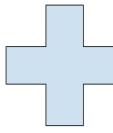
---

# R libraries

- Installing R in your computer gives you access to its basic functions
- Additionally, you can also install libraries. Libraries are packages of additional R functions that allow you to do:
  - Operations that basic R functions don't do (example: geospatial data analysis)
  - Operations that basic R functions do, but easier (example: data cleaning)
- Libraries are also called **packages**. Both terms refer to the same in R work

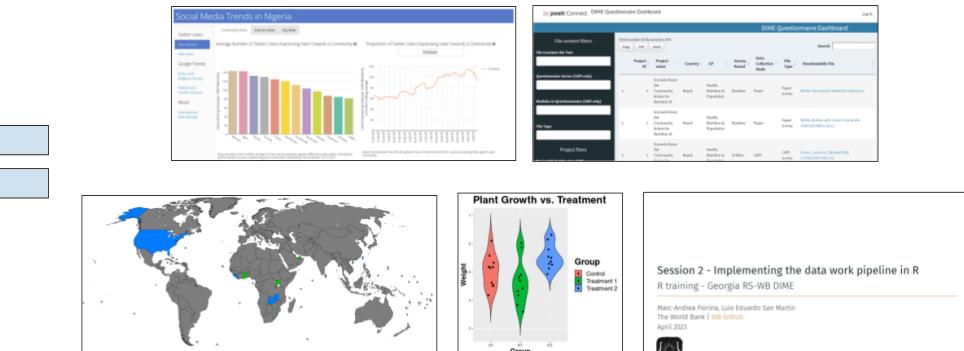
# R libraries

In a nutshell:



Basic R

Libraries



Enhanced R capabilities

# R libraries

We'll install two libraries that we'll use extensively in the labs: `haven` and `here`

## Exercise 7: Installing libraries

1. Install the libraries with `install.packages()`:

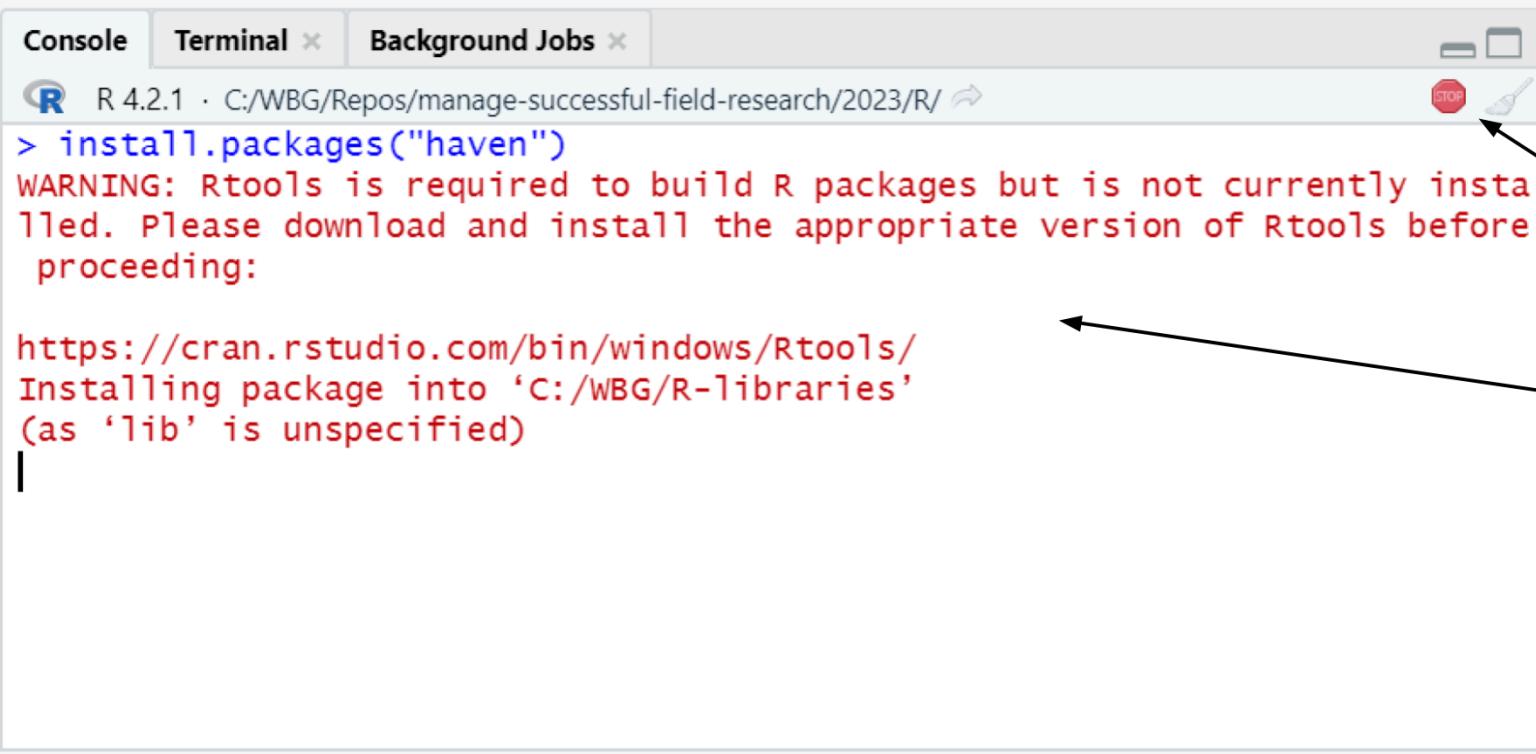
- Run in the console: `install.packages("here")`
- Run in the console: `install.packages("haven")`
- Note the quotes (" ") in the package names
- Introduce this code in the console, not the script panel



A screenshot of the RStudio interface showing the Console tab selected. The title bar indicates "Console", "Terminal x", and "Background Jobs x". The status bar shows "R 4.2.1 · C:/WBG/Repos/manage-successful-field-research/2023/R/" and a small icon. In the main console area, the command `> install.packages("haven")` is typed in blue, indicating it is being entered or has been run.

# R libraries

## Installing libraries



The screenshot shows the RStudio interface with the 'Console' tab selected. The code entered is `> install.packages("haven")`. A red warning message follows: **WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceeding:** <https://cran.rstudio.com/bin/windows/Rtools/>. The message continues: **Installing package into ‘C:/WBG/R-libraries’ (as ‘lib’ is unspecified)**. In the top right corner of the console window, there is a red 'STOP' button with a white icon and a grey 'PAUSE' button with a white icon.

the STOP sign means  
that the code is still  
running, just wait until  
it finishes

note that this is  
not an error

# R libraries

Now that the libraries are installed, we only need to load them to start using the functions they have.

## Exercise 8: Loading libraries

1. Load `haven` with: `library(haven)`

2. Load `here` with `library(here)`

3. Load `dplyr` with `library(dplyr)`

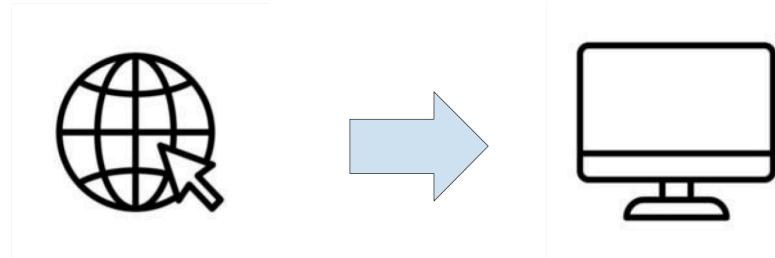
- Run this code **from the script panel** this time, not the console
- Notice that we don't use quotes in the library names this time
- If loading `dplyr` fails, install it with  
`install.packages("dplyr")`



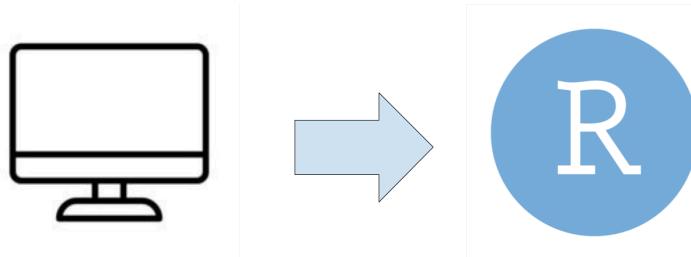
The screenshot shows the RStudio interface with the 'Console' tab selected. The session starts with the R logo and 'R 4.2.1 · C:/WBG/Repos/manage-successful-field-research/2023/R/'. The command `> library(haven)` is entered, followed by a warning message: `Warning message:  
package 'haven' was built under R version 4.2.2`. A cursor is visible at the bottom of the console window.

# R libraries

- Library installation:



- Library loading:



- You install R libraries only once in your computer
- You load libraries every time you open a new RStudio window (only load the libraries you will use)

# File paths

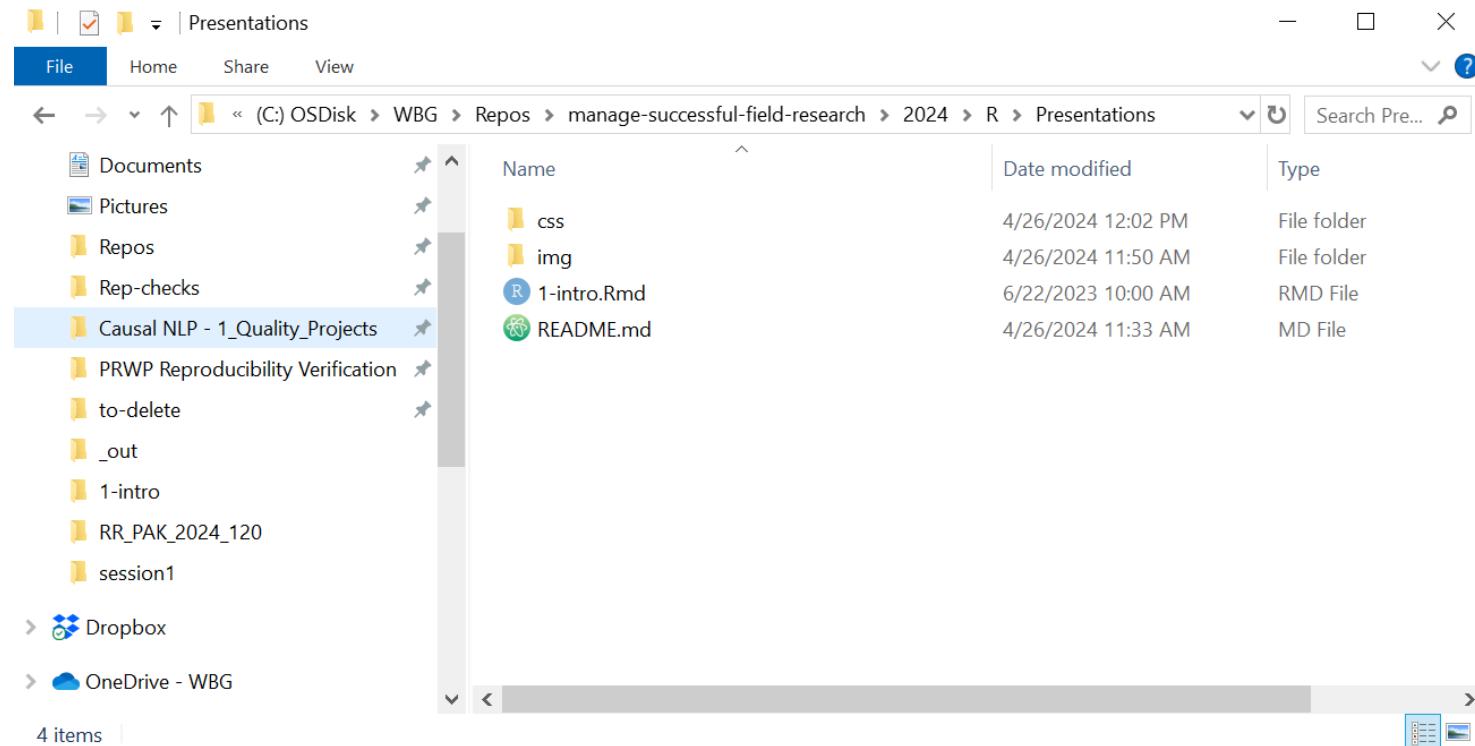
---

# File paths

File paths are the location of a file or folder in your computer. For example:

```
C:/WBG/Repos/manage-successful-field-research/2025/R/Presentations/1-intro.Rmd
```

is the location of this presentation file in my computer.



# File paths

- File paths are important because they define the route from which your computer reads input data or exports outputs
- Remember when you manually loaded `treat_status.csv` in exercise 4? you were using a file path implicitly. This is how the command for loading the data was implicitly applied in my computer:

```
treat_status <- read.csv("C:/Users/wb532468/Downloads/treat_status.csv")
```

- This will look different for your computer because the path to your `Downloads` folder is not going to be the same

# File paths

## Relative and absolute file paths

- Absolute file paths define a location starting from a disk in Windows (`C:/`) or the folder `/Users/` in Mac.
  - For example: `C:/Users/wb532468/Downloads/treat_status.csv`
- Relative file paths define the location of a file relative to an "anchor" location.
  - For example: `Downloads/treat_status.csv` is the relative path of `treat_status.csv` with respect of `C:/Users/wb532468`

# File paths

## Relative and absolute file paths

- You can use both in R
- Relative file paths will work with respect of the current working directory where R is located. You can check it with the function `getwd()`

```
getwd()
```

```
## [1] "C:/WBG/repos/manage-successful-field-research/2025/R/Presentations"
```

- In Windows, the default working directory when starting R is the Documents folder

# File paths

## Relative and absolute file paths

- The current working directory will always be the "anchor" for relative paths in R
- You can change it with the function `setwd()`, as in:

```
setwd("C:/path/that/only/exists/in/my/computer")
```

- However, we absolutely recommend not to do it
- Why? Because having code that sets a working directory with a path that only exists in your computer will break reproducibility

# File paths

## RStudio projects and `here`

- Instead, you should use RStudio projects and the library `here`
- RStudio projects automatically sets the working directory to a "project" folder, so you can use it as anchor to work with relative paths that will work for everyone else that uses the same folder structure
- This is crucial because it allows smooth interoperability between different computers where the exact path to the project root directory differs
- This will become clearer with next two exercises

# File paths

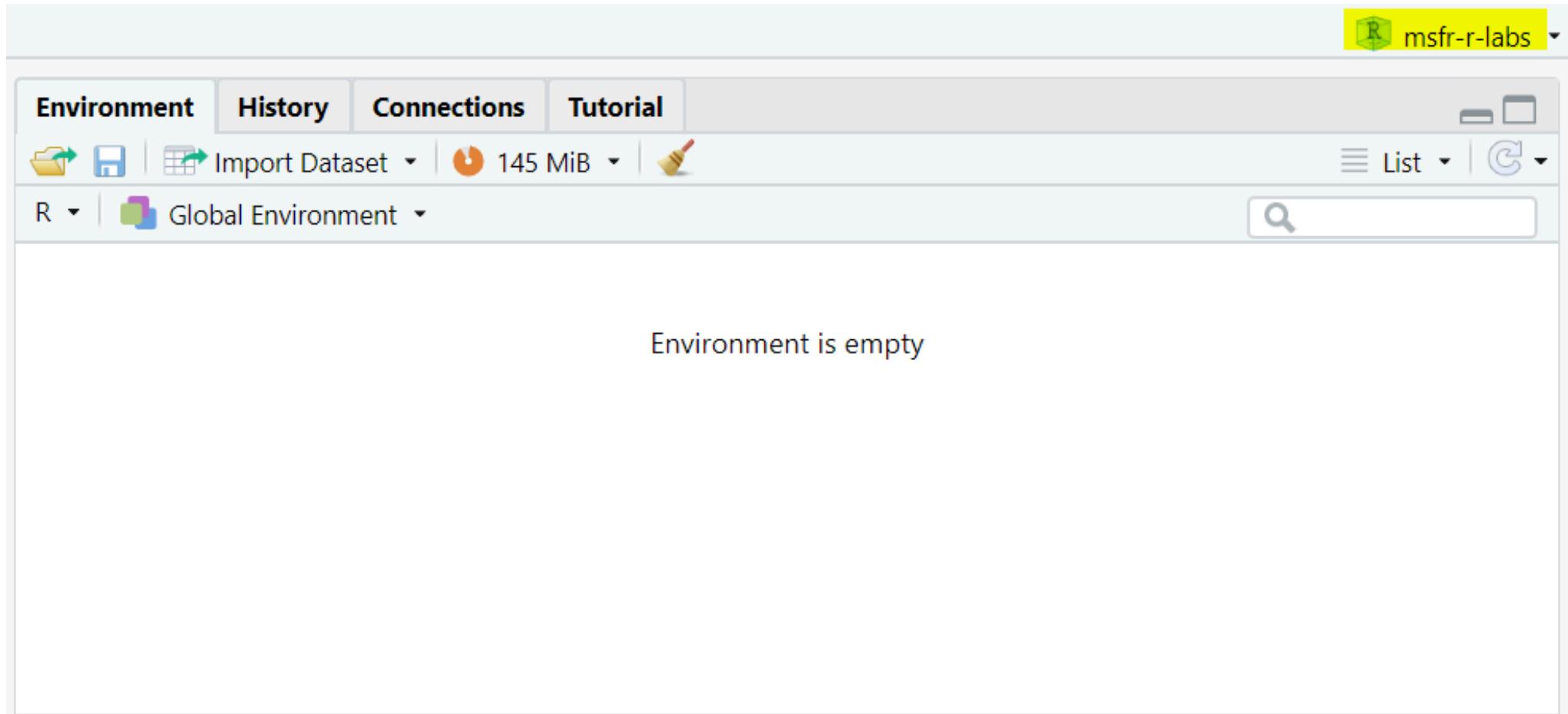
Save your current script by clicking in the floppy disk icon **before starting this exercise.**

## Exercise 9

1. Create a folder named `msfr-r-labs` in your preferred location in your computer
2. Go to [http://bit.ly/msfr25\\_materials](http://bit.ly/msfr25_materials) and download the file: `DataWork.zip`
3. Unzip `DataWork.zip` in the folder `msfr-r-labs`
4. On RStudio, select `File > New Project...`
5. Select `Existing Directory`
6. Browse to the location of `msfr-r-labs` and select `Create Project`
7. RStudio will restart after this

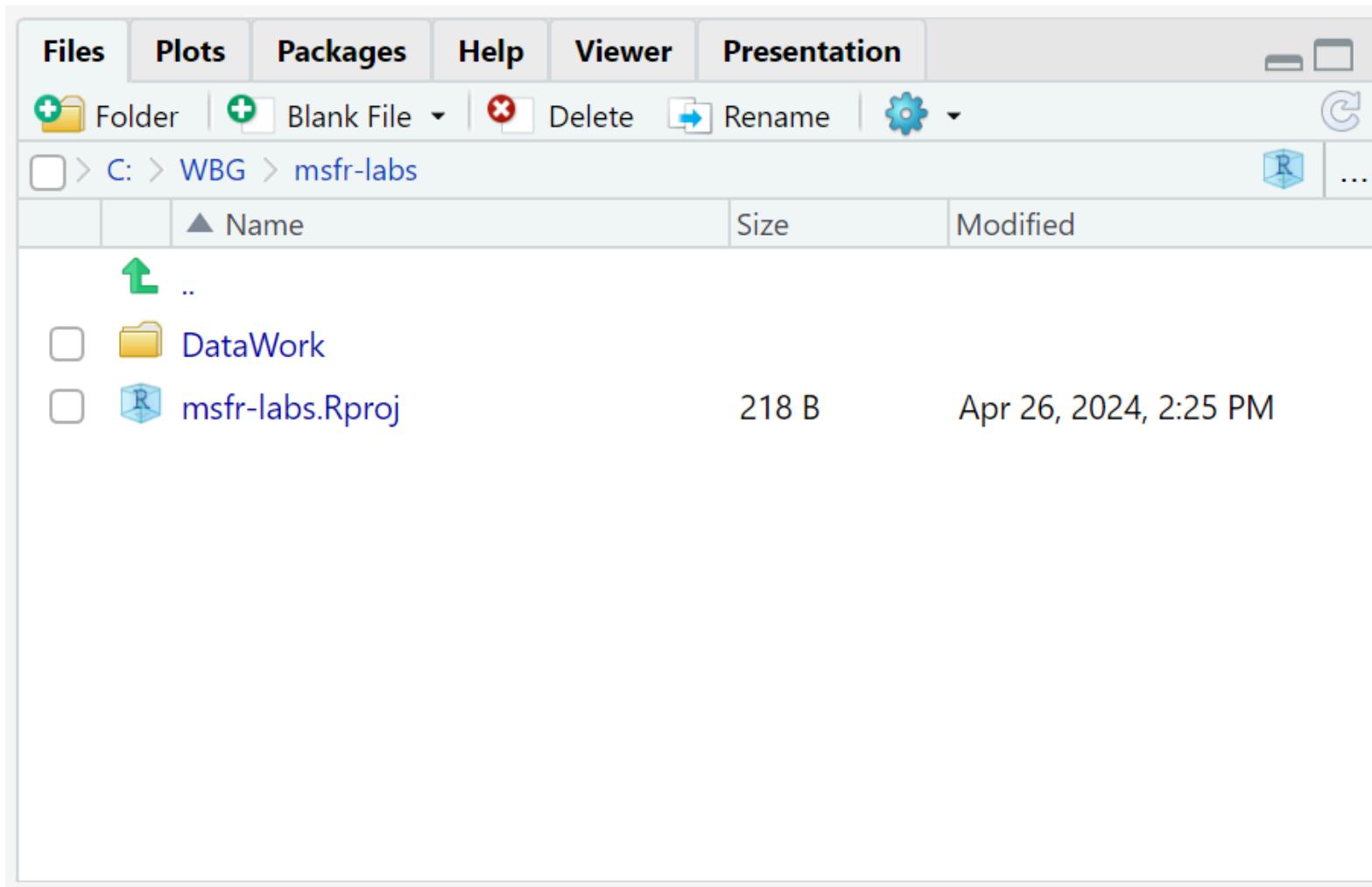
# File paths

## RStudio projects



# File paths

## RStudio projects



# File paths

## Using `here`

- `here` locates files relative to your project root
- It uses the root project directory to build paths to files easily
- It allows for interoperability between different computers where the absolute path to the same file is not the same

# File paths

## Using `here`

- Now you'll be able to use `here()` to point the location of every file relative to your project root
  - For example, to load a `csv` file located in: `C:/WBG/project-root-name/data/raw/data-file.csv`, you should use:

```
path <- here("data", "raw", "data-file.csv")
df <- read.csv(path)
```

- Notes:
  - Your project root is the directory that contains the `.Rproj` file
  - The arguments of `here` are the consecutive subfolders where your file is located. The final argument is the file name **with the file extension**
  - The result of `here()` is an absolute path that points to a file or folder location in your computer

# File paths

## Exercise 10

1. Start a new script with `File > New File > R Script`

2. Load the libraries `here` and `haven` with `library()`

3. Create an object called `path` that will be a file path to `DataWork/data/raw/TZA_CCT_baseline.dta`. Use `here()` for this:

```
here("DataWork", "Data", "Raw", "TZA_CCT_baseline.dta")
```

4. Use `read_stata()` to load the file and assign it to an object with `<-`. The argument of `read_stata()` should be the object `path`.

```
# Loading libraries
library(here)
library(haven)

# Importing data
path <- here("DataWork", "Data", "Raw", "TZA_CCT_baseline.dta")
baseline <- read_stata(path)
```

# File paths

```
# Loading libraries
library(here)
library(haven)

# Importing data
path <- here("DataWork", "data", "raw", "TZA_CCT_baseline.dta")
baseline <- read_stata(path)
```

Some notes:

- The lines that start with `#` are code comments. R doesn't run these lines. We're using them to create small sections in the script
- It's usual for R scripts to start with a section loading libraries and a next section loading data

# File paths

```
# Loading libraries
library(here)
library(haven)

# Importing data
path <- here("DataWork", "data", "raw", "TZA_CCT_baseline.dta")
baseline <- read_stata(path)
```

Some more notes:

- `read_stata()` is a function from the library `haven`. You can only use it after loading `haven` with `library(haven)` (we did this already in exercise 9)
- The same happens with `here()`, which is a function from the library `here`. You can only use `here()` after having run `library(here)` (exercise 9)
- Remember to assign the dataframe you're reading to an object. You can call it `baseline` as we did before

# File paths

If you did the exercise correctly, you should see `baseline` listed in the environment panel

The screenshot shows the RStudio interface with the 'Environment' tab selected. The top bar includes tabs for 'Environment', 'History', 'Connections', and 'Tutorial', along with various icons for file operations like 'Import Dataset' and '188 MiB'. Below the tabs, the 'Global Environment' dropdown is set to 'R'. A search bar is also present. The main area displays the 'Data' section, which lists the 'baseline' dataset. The 'baseline' entry shows '1760 obs. of 55 variables'. Under the 'values' section, the 'path' variable is listed as 'C:/WBG/msfr-labs/Datawork/data/raw/TZA\_CCT\_baseline.csv'. The entire window has a light gray background.

# Piping

---

# Piping

## Functions inside functions

- R allows you to use functions inside functions. For example, this will show the summary of the logarithm of the dataframe `baseline`

```
summary(log(baseline$food_cons))
```

- In this case, R is taking the *output* (result) of `log(baseline$food_cons)` as the *input* of `summary()`
- Also, note that the operator `$` is subsetting the column `food_cons` from the dataframe `baseline`

# Piping

## Functions inside functions

- This is convenient because it saves us from having to store intermediate results into objects frequently, keeping our R environment cleaner
- However, it also has the big downside of making code convoluted and unclear, as in the example below

```
round(mean(log(baseline$food_cons)), 2)
```

- Piping helps keeping code clear in cases like this

# Piping

- The pipe operator is this: `%>%`
- Pipes are used between objects or the results of functions, as in: `log(baseline$food_cons) %>% summary()`
- The meaning of the operator is:

Take the output (or object) of the function at the left and pass it as the first argument of the function at the right

In short:

- `x %>% function()` is the same as `function(x)`
- `x %>% function1() %>% function2()` is the same as `function2(function1(x))`

# Piping

Hence, these are equivalent:

```
# The long way, with intermediate objects:  
log_food_cons <- log(baseline$food_cons)  
summary(log_food_cons)  
  
# Shorter, but code is less clear:  
summary(log(baseline$food_cons))  
  
# With pipes, clearer and with no intermediate objects:  
baseline$food_cons %>% log() %>% summary()  
  
# Note that you can also concatenate pipes and include line breaks:  
baseline$food_cons %>%  
  log() %>%  
  summary()
```

Pipes are part of the libraries included in the tidyverse, such as `dplyr`. When you loaded `dplyr` in exercise 9, the pipes became available for use.

# Piping

## Exercise 11: Load data with pipes

We'll load the same data we loaded manually in Exercise 4, this time with code.

1. Use the functions `here()` and `read.csv()` with pipes between them (`%>%`) to read the file

`DataWork/Data/Raw/treat_status.csv`. Name the dataframe `treatment`. Some hints:

- **Very important:** always remember to save the data read into a dataframe object with `<-`, otherwise R will only print the dataframe and it will not be stored in the environment
- Remember that the arguments of `here` are the names of the folders in the path to the file, with quotes (" "). The last argument is the file name with the file extension, as in: `here("DataWork", "Data", "Raw", "treat_status.csv")`

2. The solution is this:

```
treatment <- here("DataWork", "Data", "Raw", "treat_status.csv") %>% read.csv()
```

# Piping

```
# Adding line breaks for code clarity:  
treatment <-  
  here("DataWork", "Data", "Raw", "treat_status.csv") %>%  
  read.csv()
```

The screenshot shows the RStudio interface with the 'Environment' tab selected. The global environment contains two datasets: 'baseline' and 'treatment'. The 'treatment' dataset is highlighted with a yellow box. Its details are shown below:

baseline	1760 obs. of 55 variables
treatment	80 obs. of 3 variables

Below the datasets, there is a 'values' section with a 'path' entry:

path	"C:/WBG/msfr-labs/Datawork/data/raw/TZA_CCT_basel...
------	--

Thanks! Gracias!

---