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 and download

1-intro-R.pdf

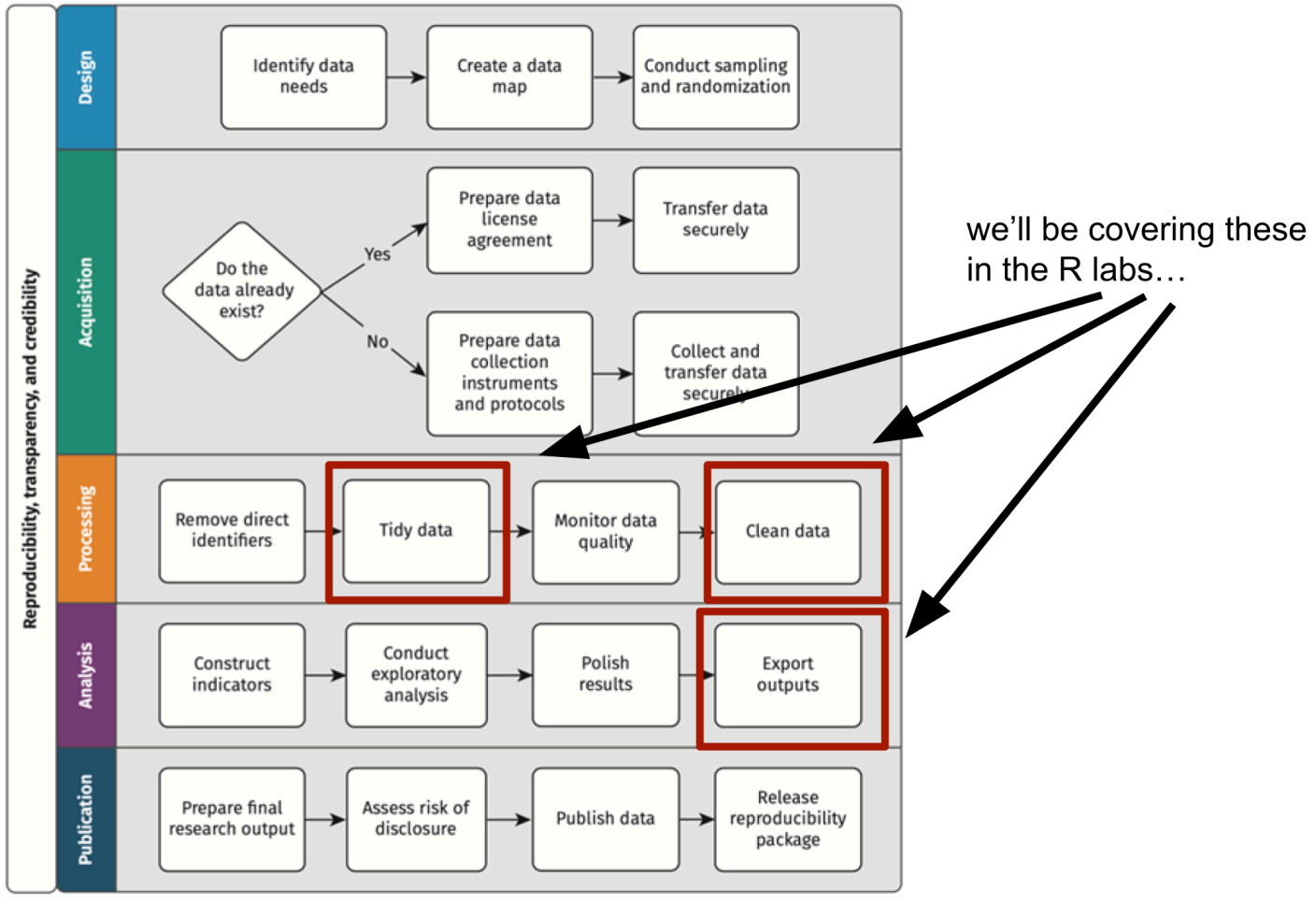
Table of contents

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

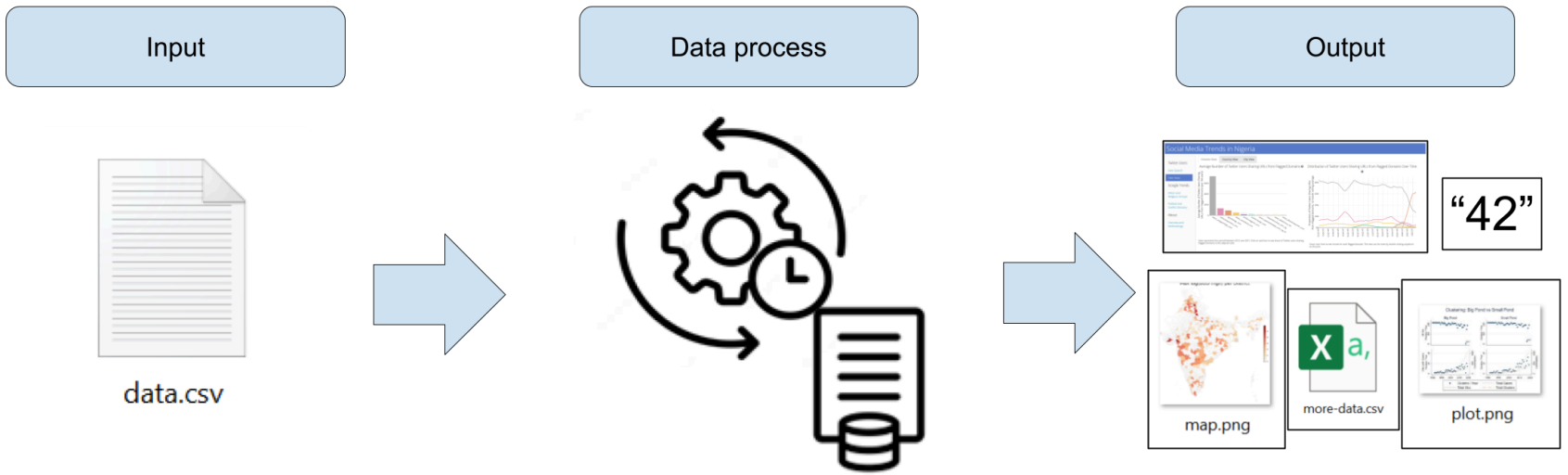


Statistical programming

Data work in development research

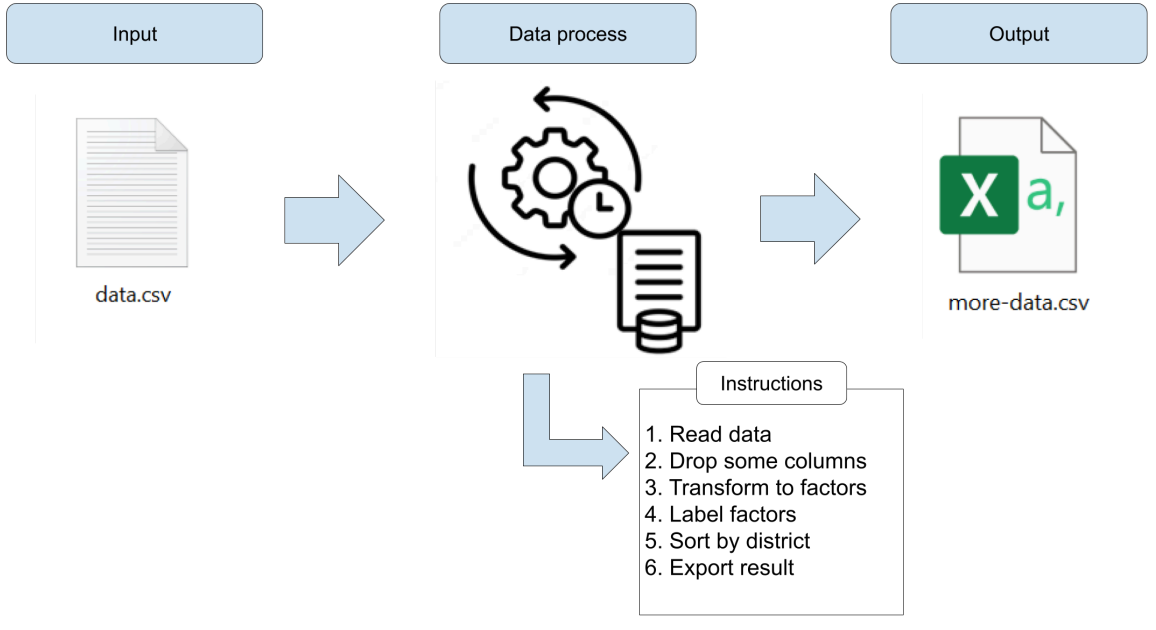
More generally, for any of the processes in development research, we'll call data work everything that:

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



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**



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.

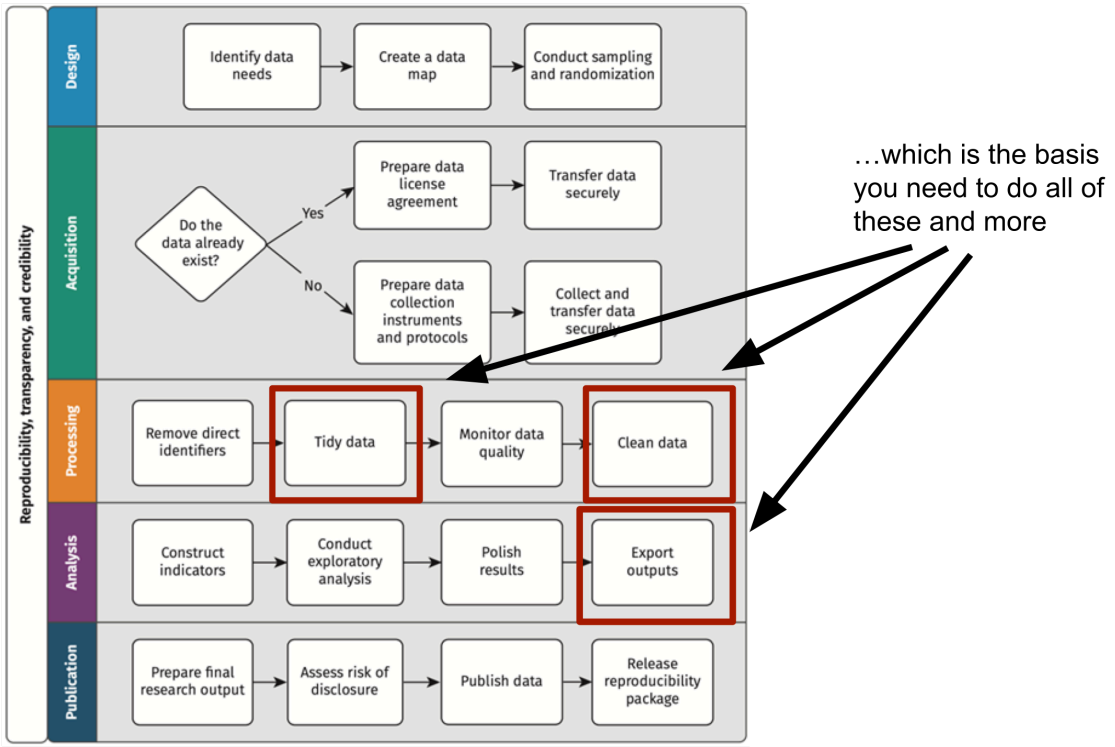
This is also usually referred to as *computational reproducibility*.

Writing R code

Writing R code

How to write R code?

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



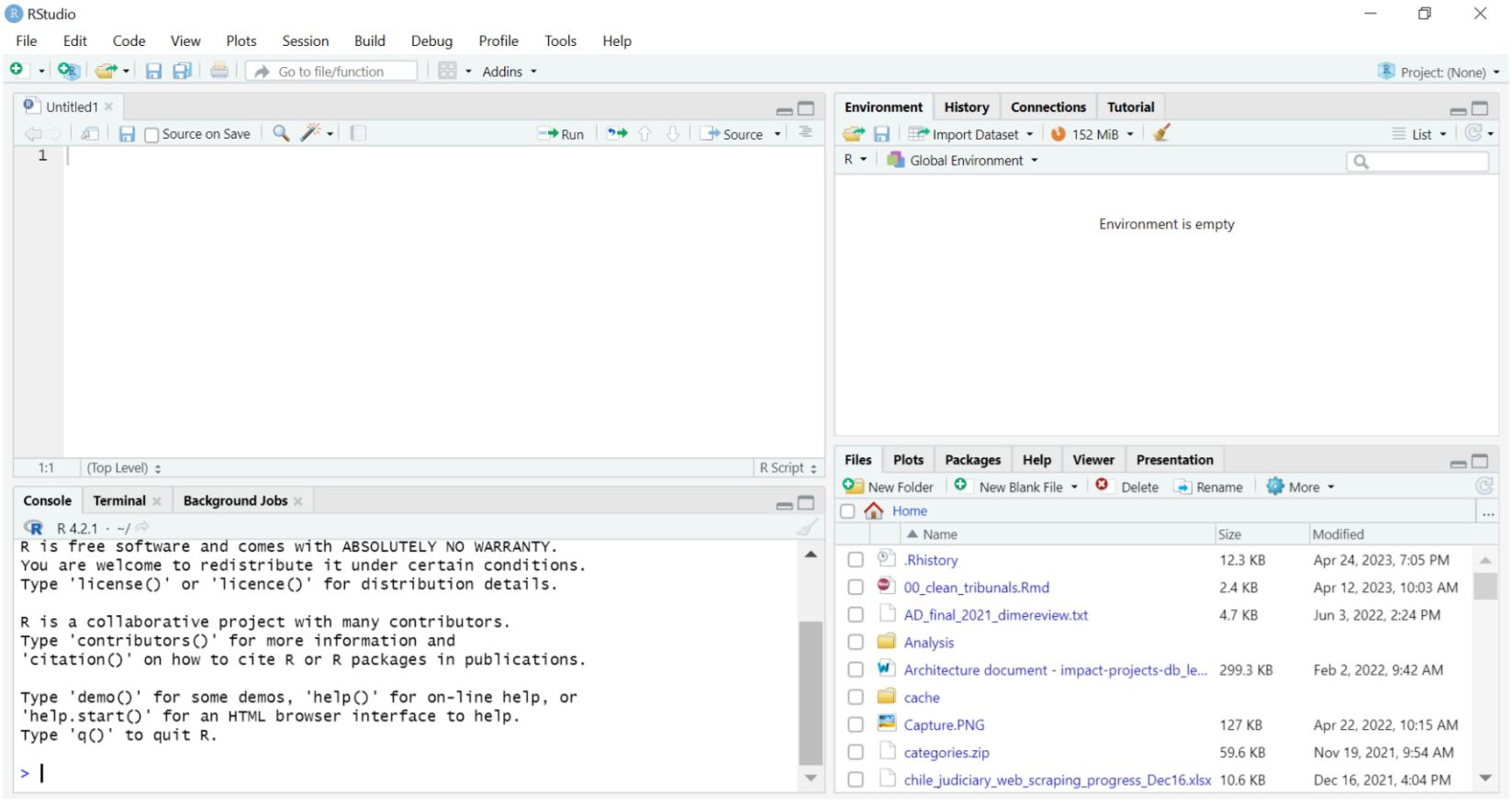
Writing R code

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

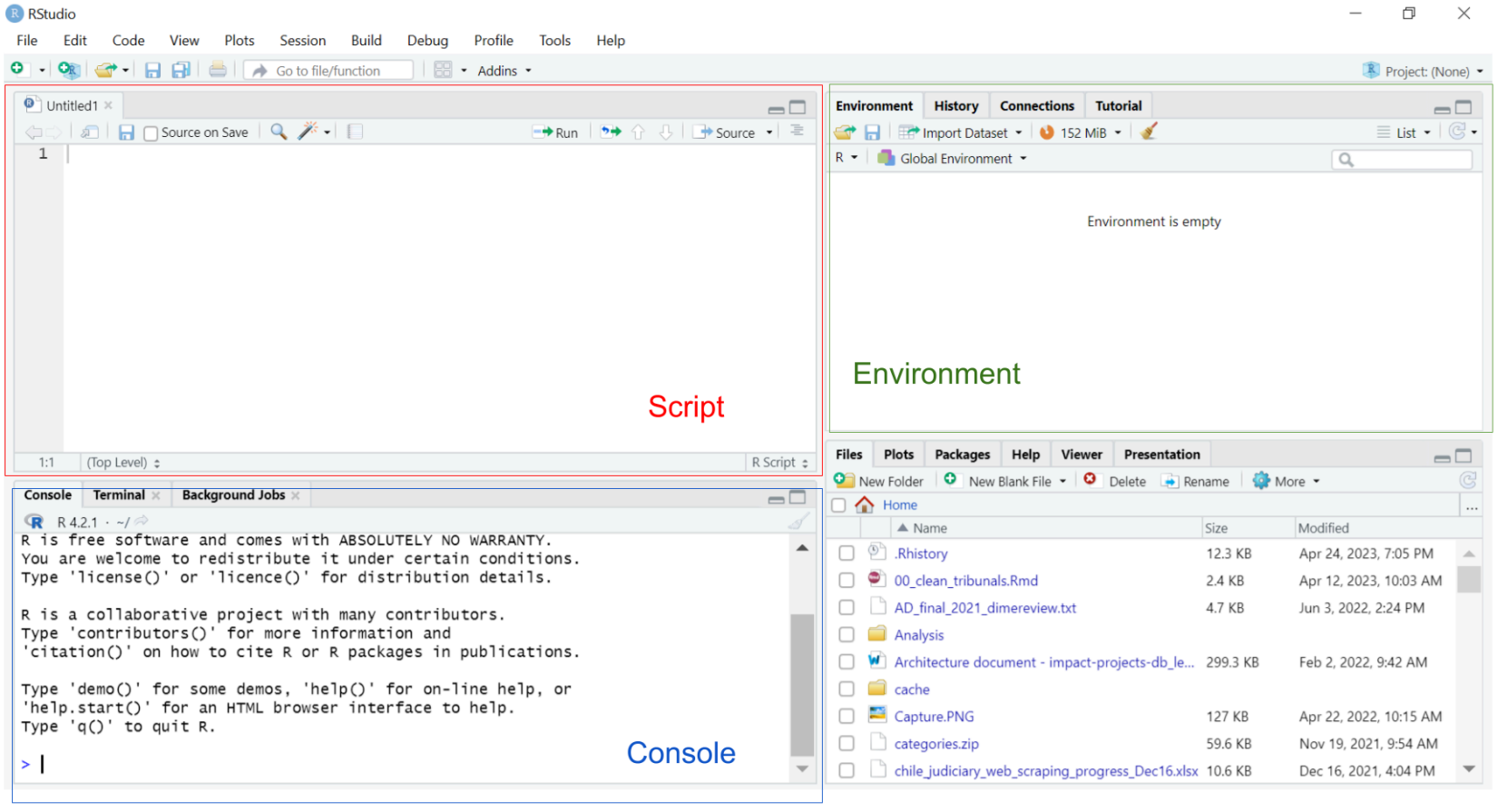
Writing R code

RStudio interface



Writing R code

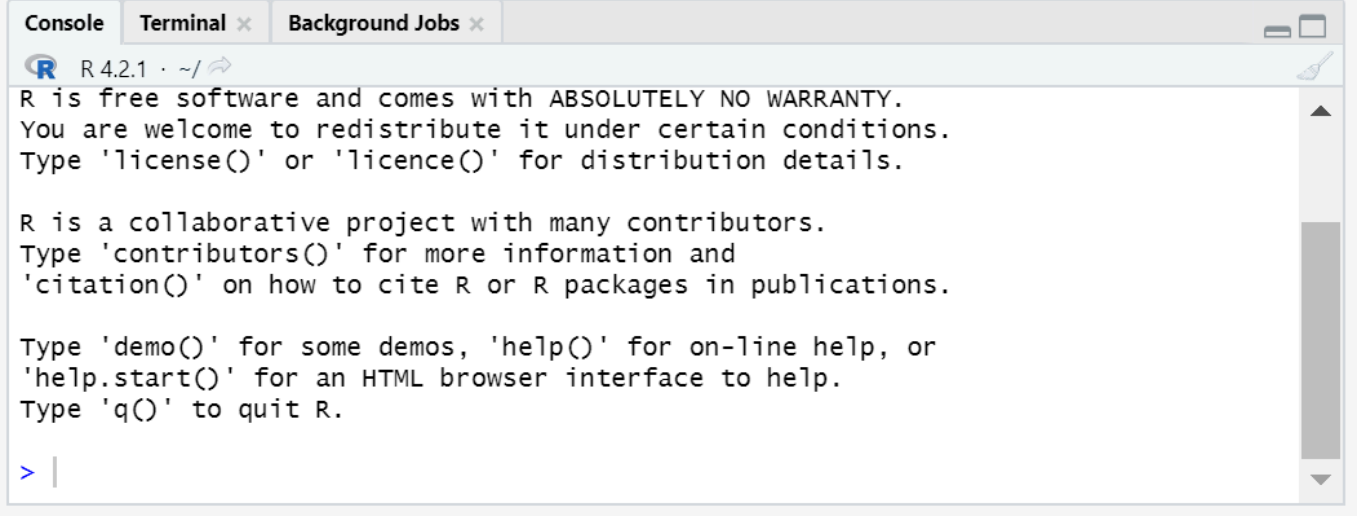
RStudio interface



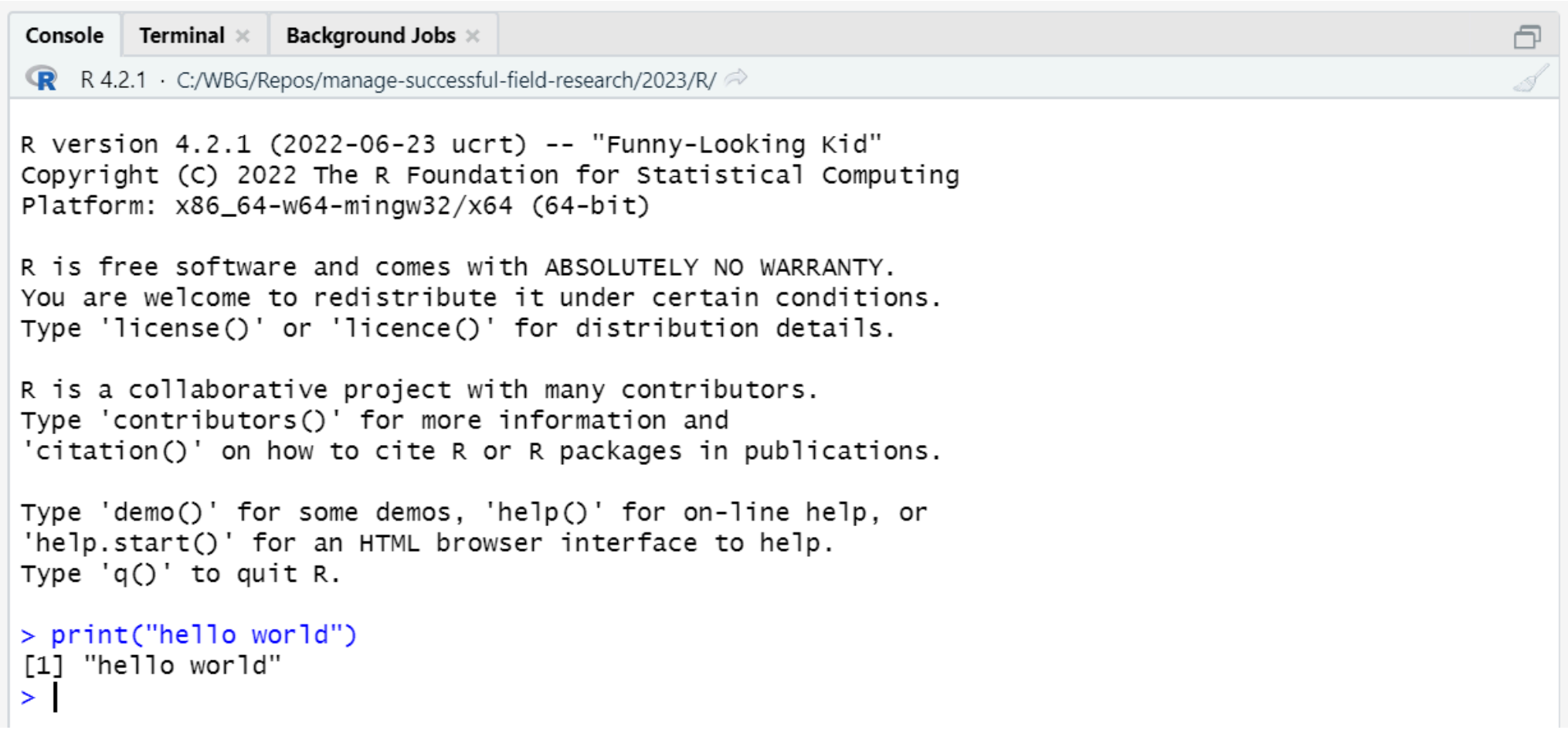
Writing R code

Exercise 1: writing code in the console

1. Write the following code in the console of RStudio
    - `print("hello world")`
    - Make sure to include the quotes: `" "`
2. Press Enter to run the code



Writing R code



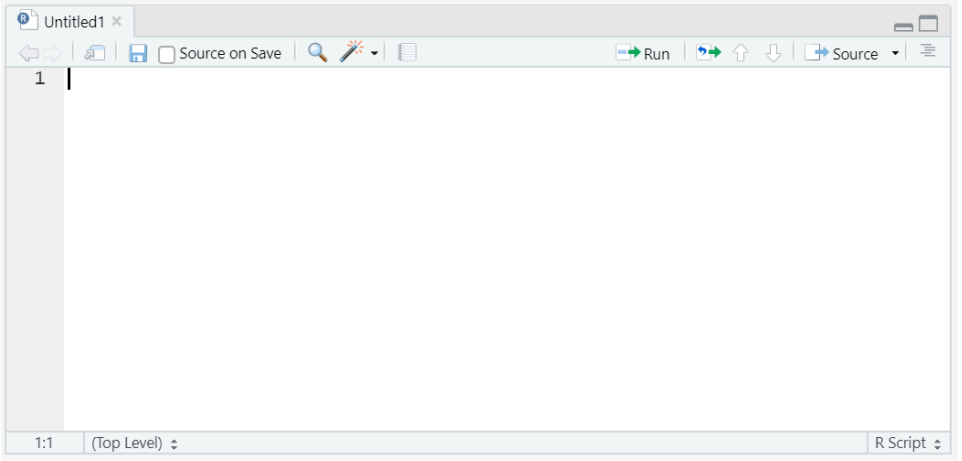
Writing R code

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")
```



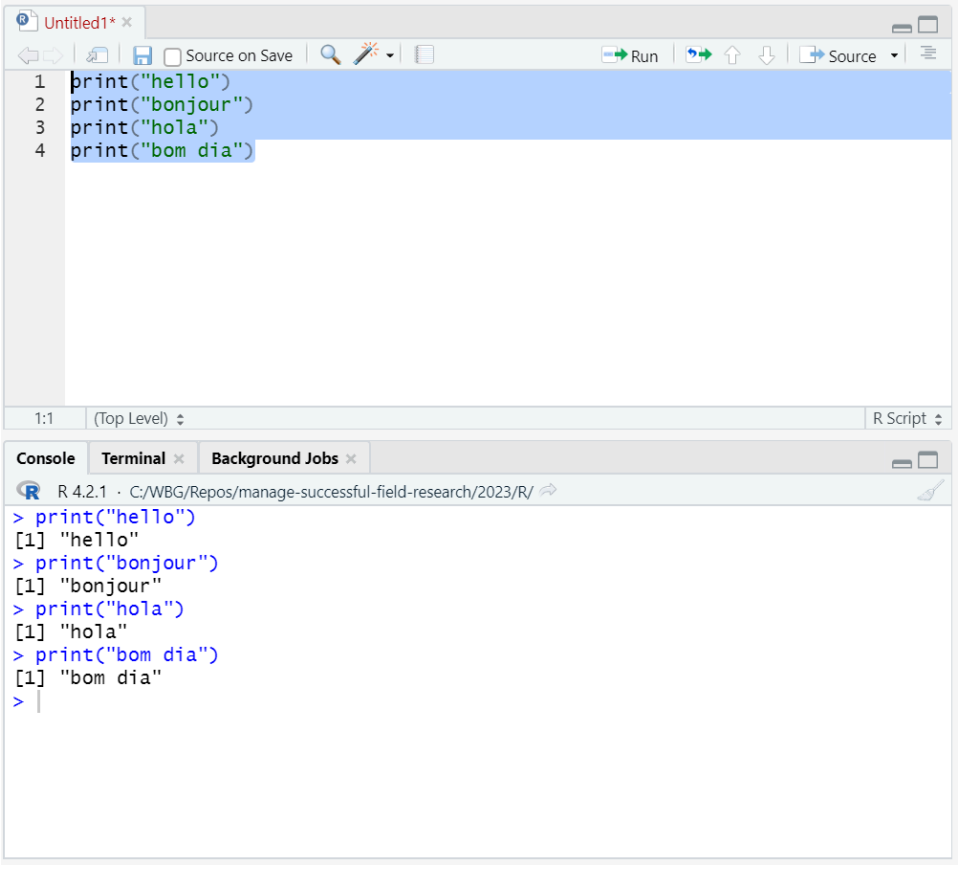
1. Select the text you introduced with your mouse

2. Press "Run"

Writing R code

R scripts

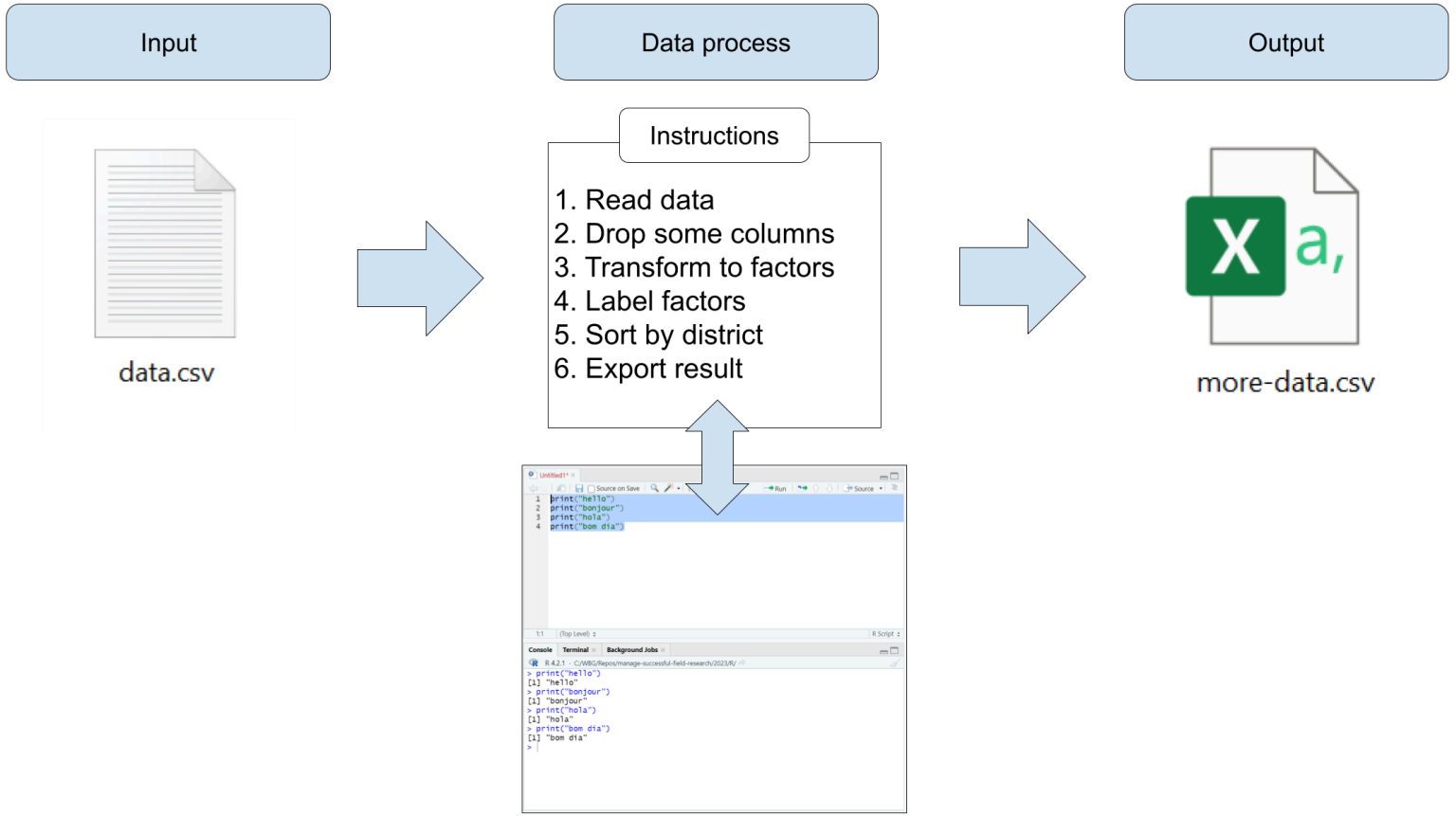
- Writing and running code from the console will execute it immediately
- Writing code in the script panel allow us to write multiple lines of code and execute them later
    - Each line is executed in order
    - The line and the results will show in the console
- **Important:** for the rest of the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



Writing R code

R scripts

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



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
```

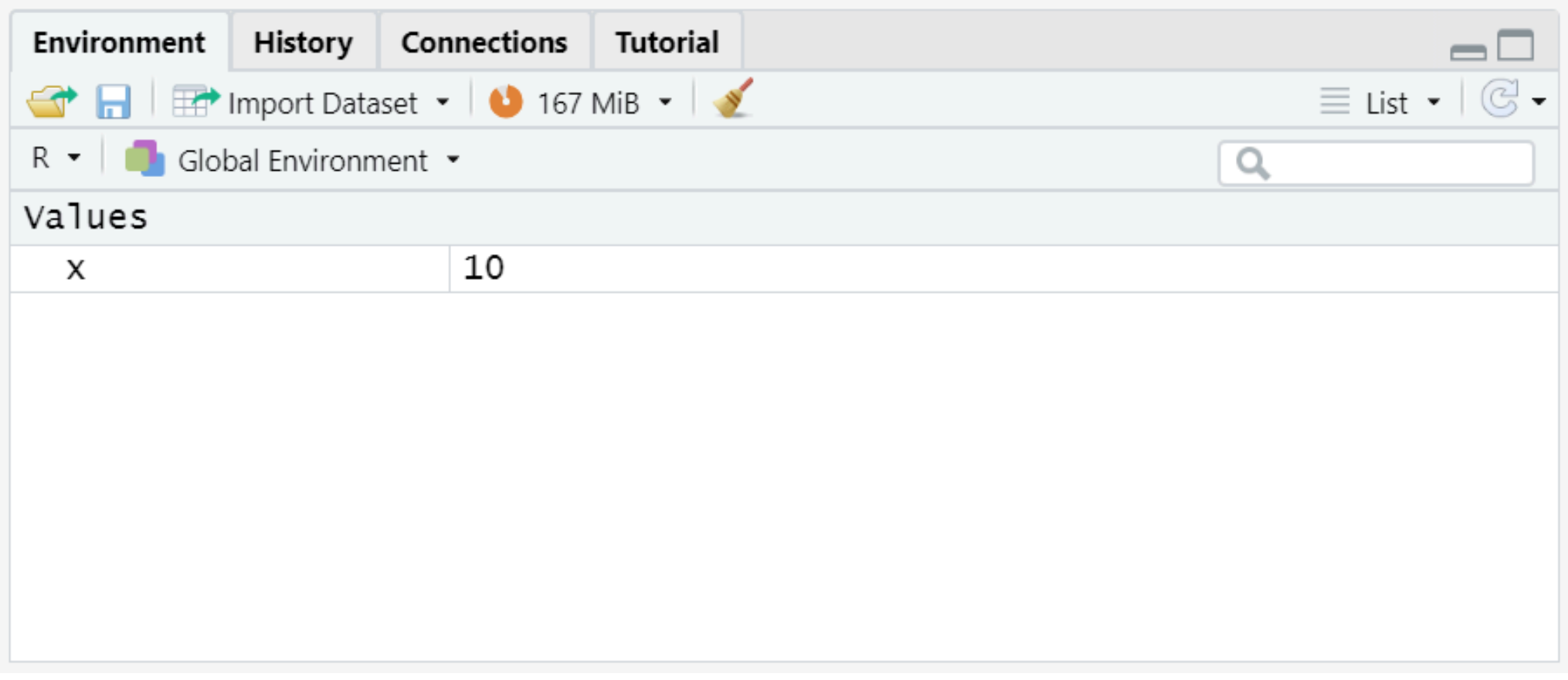
```
print(x + 5)
```

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

Writing R code

Creating objects in R

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

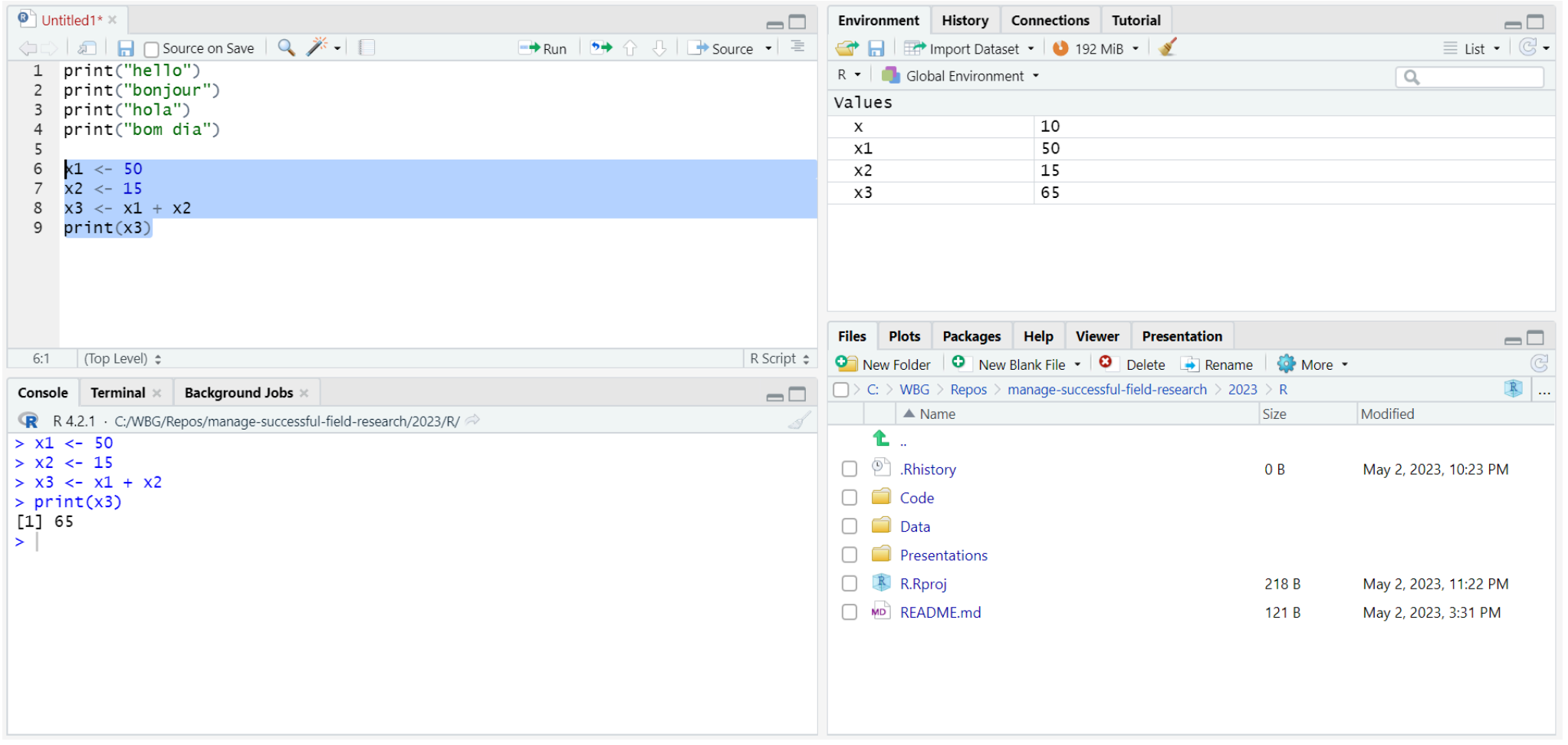


Exercise 3: operations with objects

1. Create an object called `x1` in your script and assign to it the value of 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



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

Exercise 4: Loading data into R

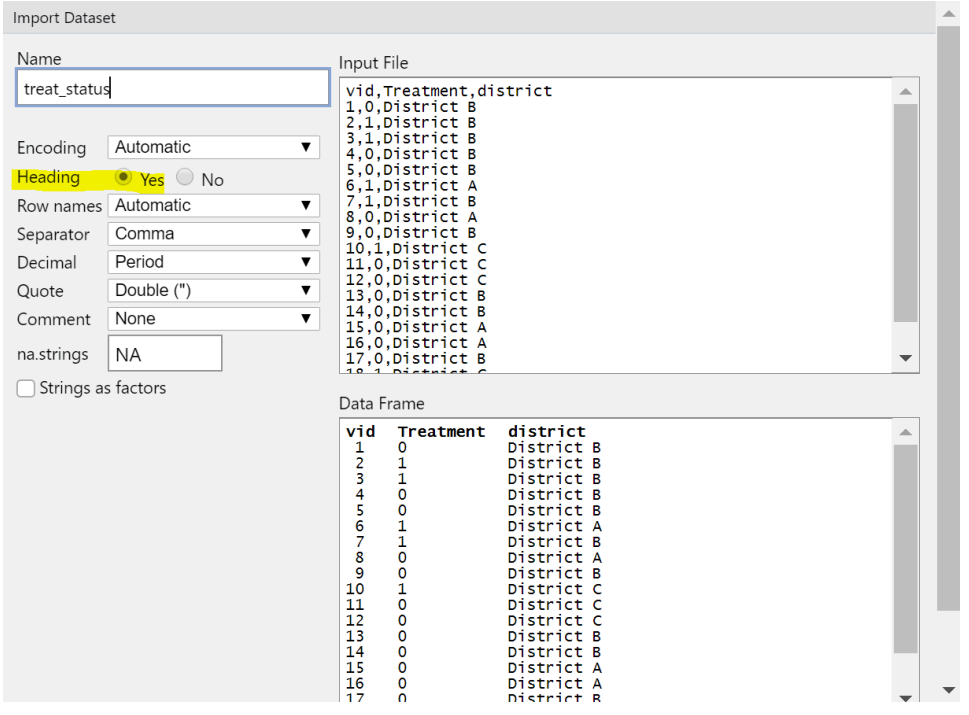
1. Click on this link: http://bit.ly/msfr25_materials and download the file `treat_status.csv`

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`



Data in R

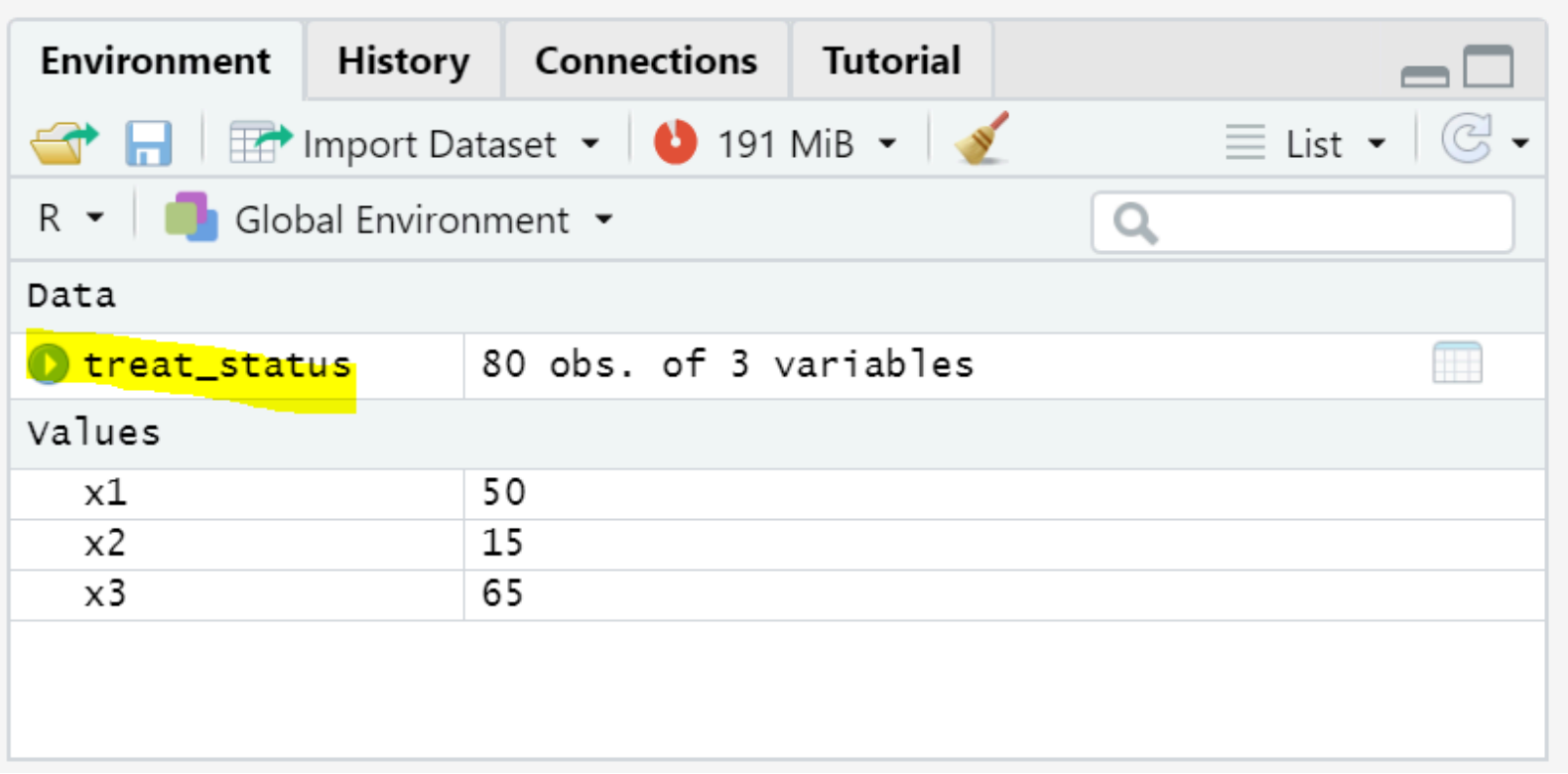
- If you did this correctly, you will note that a viewer of the data now appears in RStudio
- You can click on the `x` next to `treat_status` to return to the script
- To open the viewer again, use the code: `View(treat_status)` (notice the uppercase "V")

	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

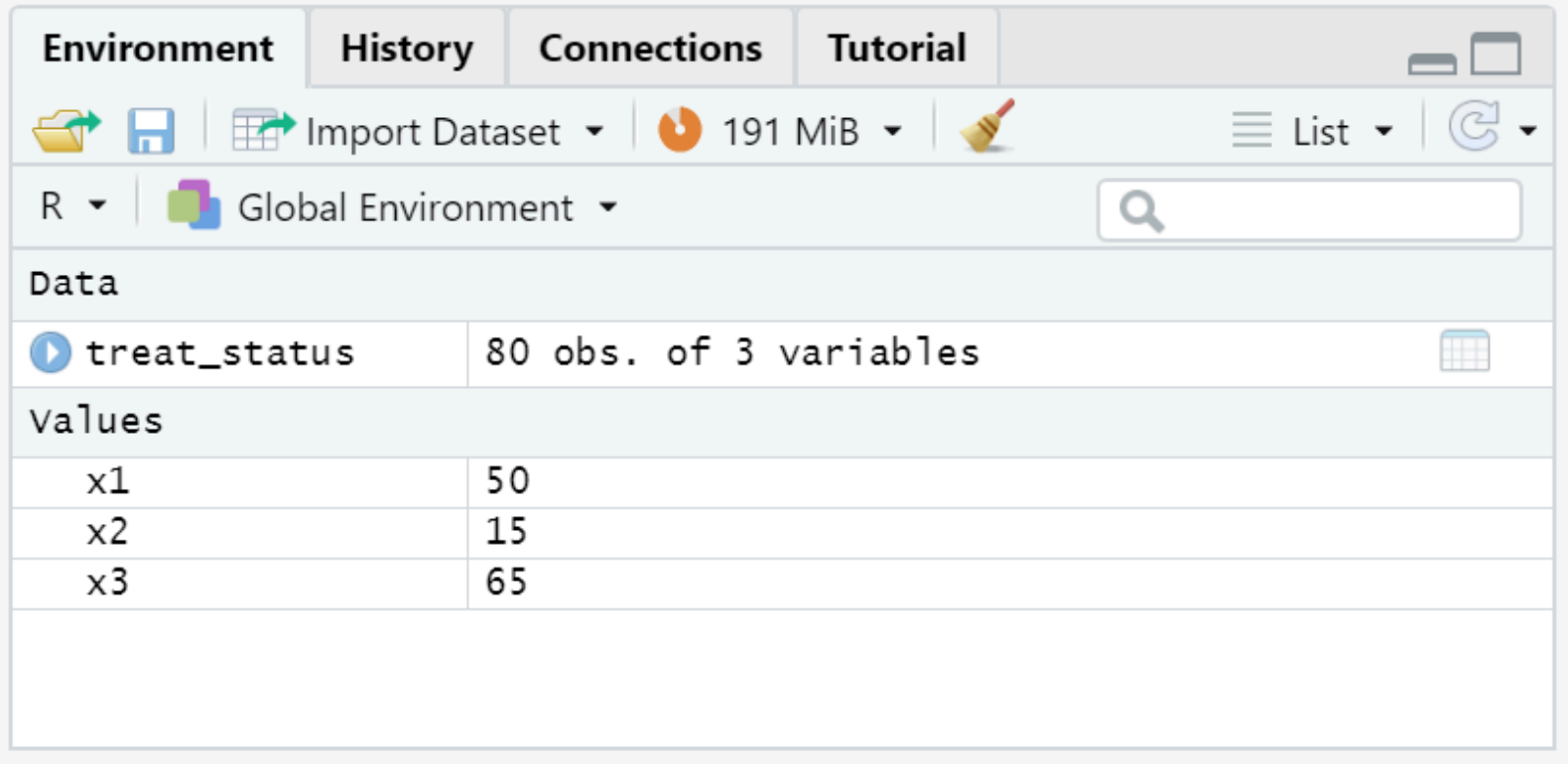
Data in R

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



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**



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?

Untitled1* | treat_status

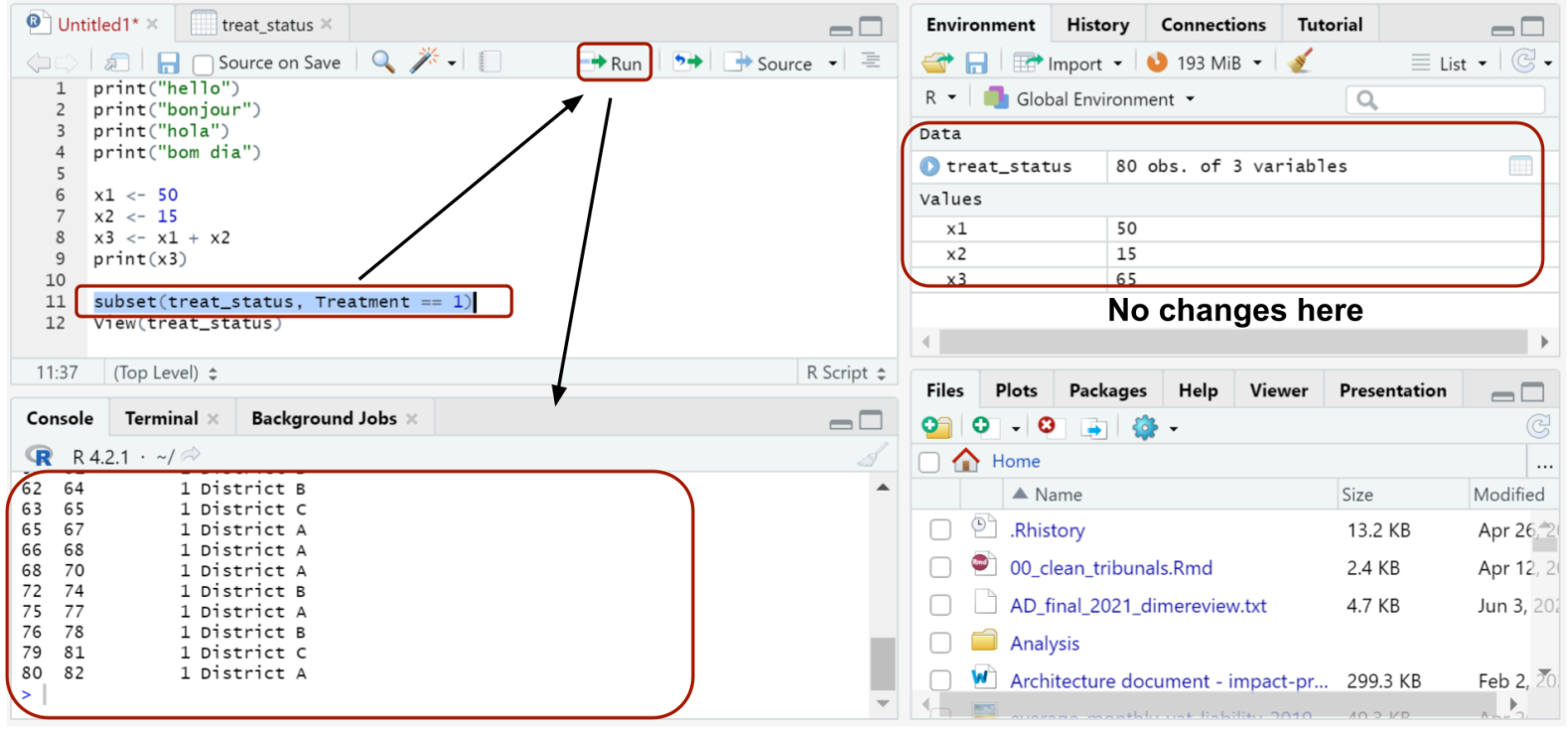
Filter

	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

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)



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!

Untitled1* | treated_villages

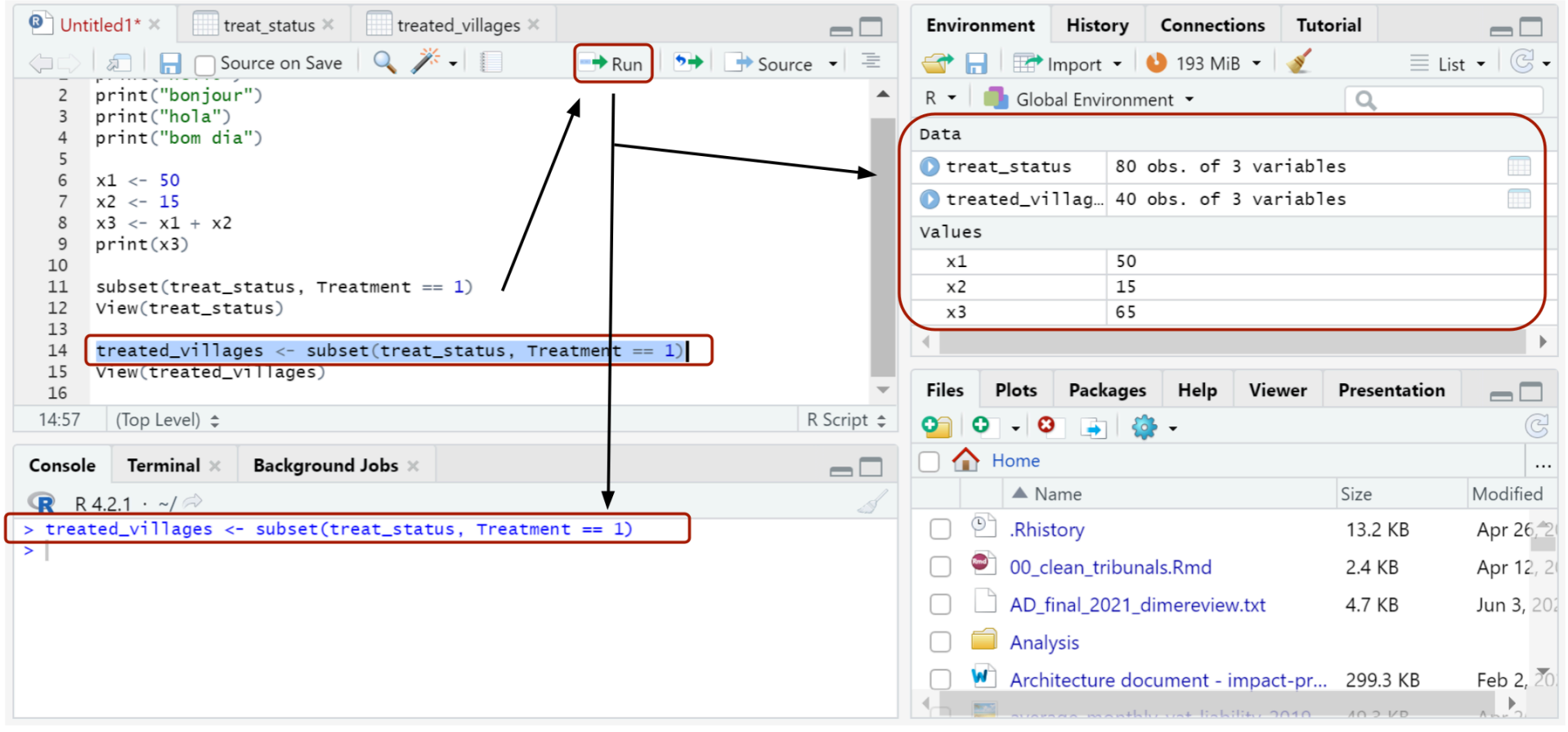
Filter

	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

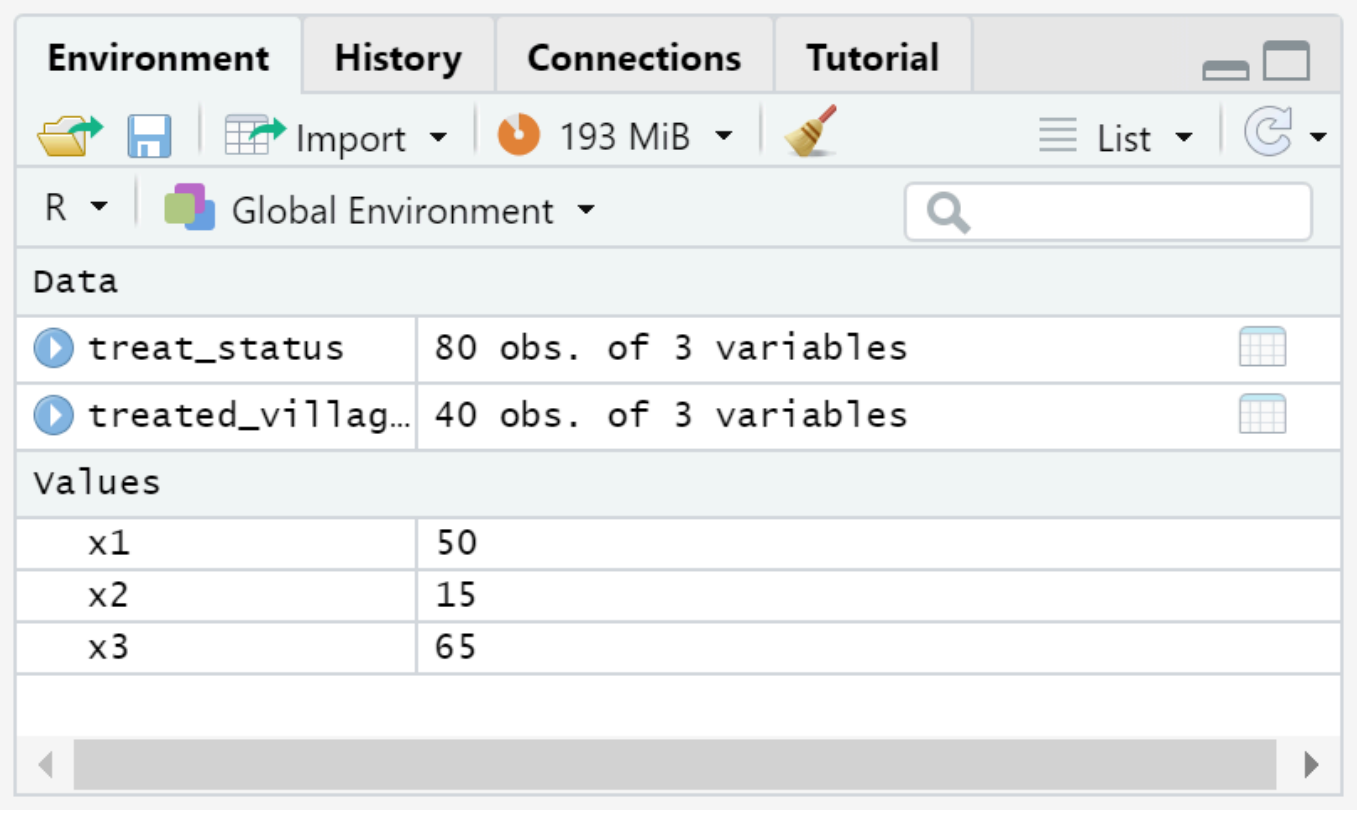
Data in R

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.



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



Functions in R

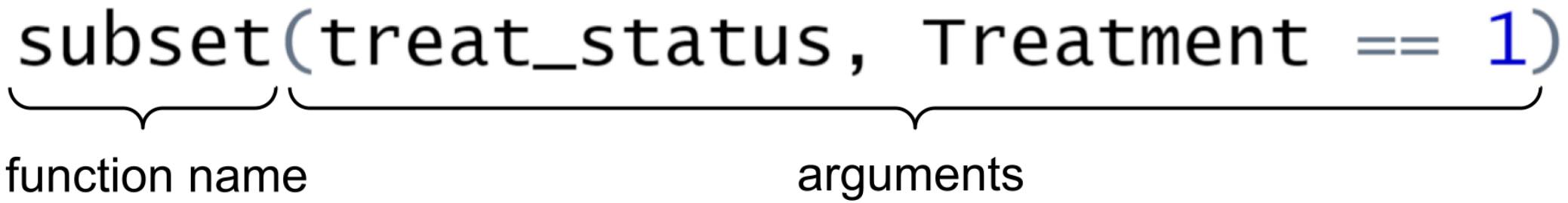
Functions in R

- 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:



- **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

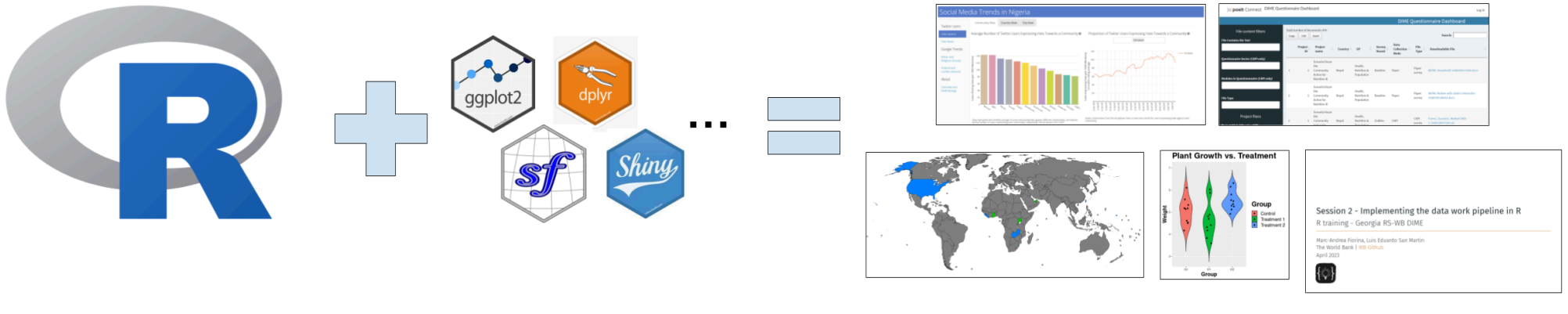
R libraries

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

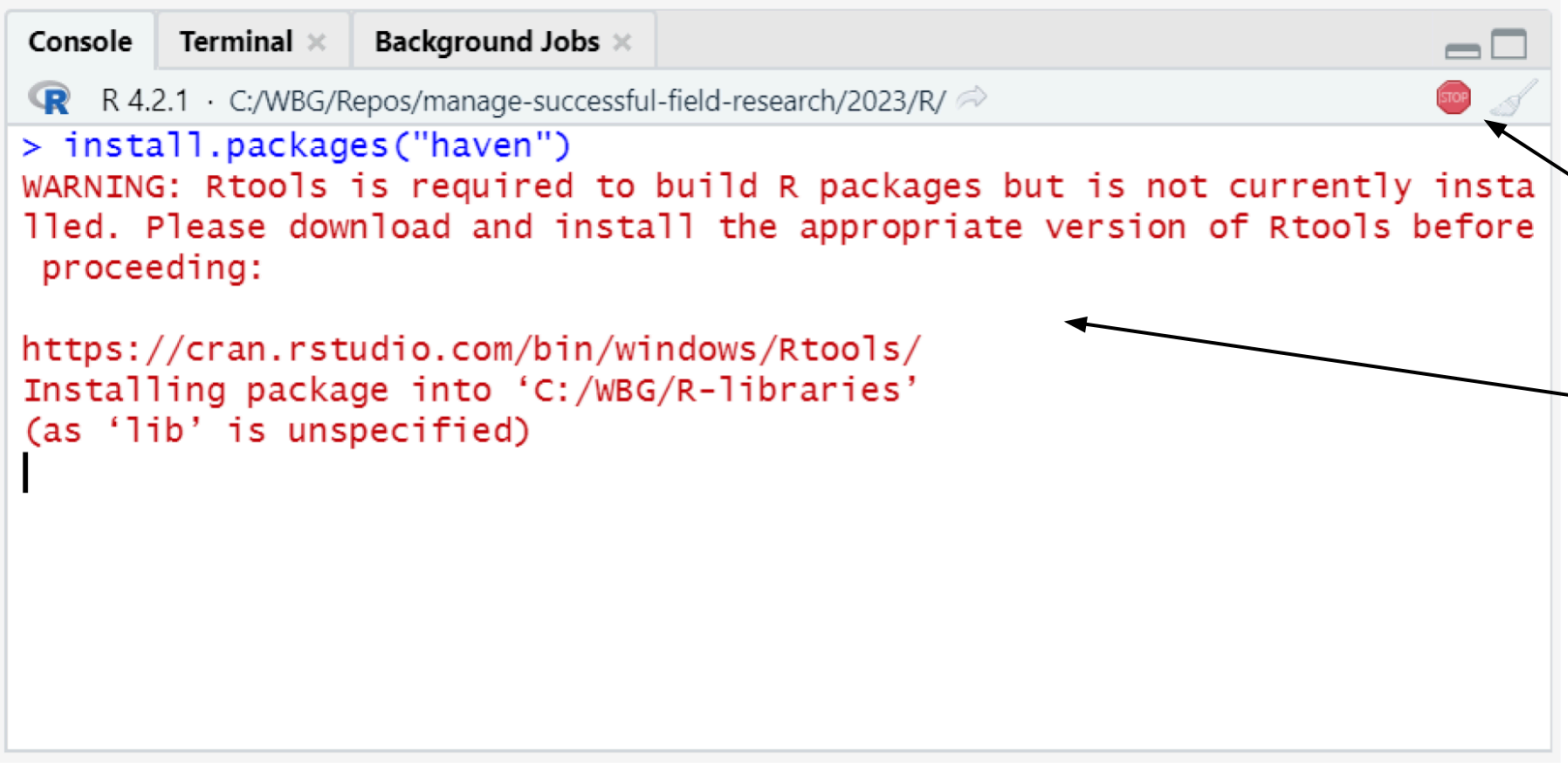
We'll install two libraries that we'll use extensively in the labs: `haven` and `here`

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

R libraries

Installing libraries

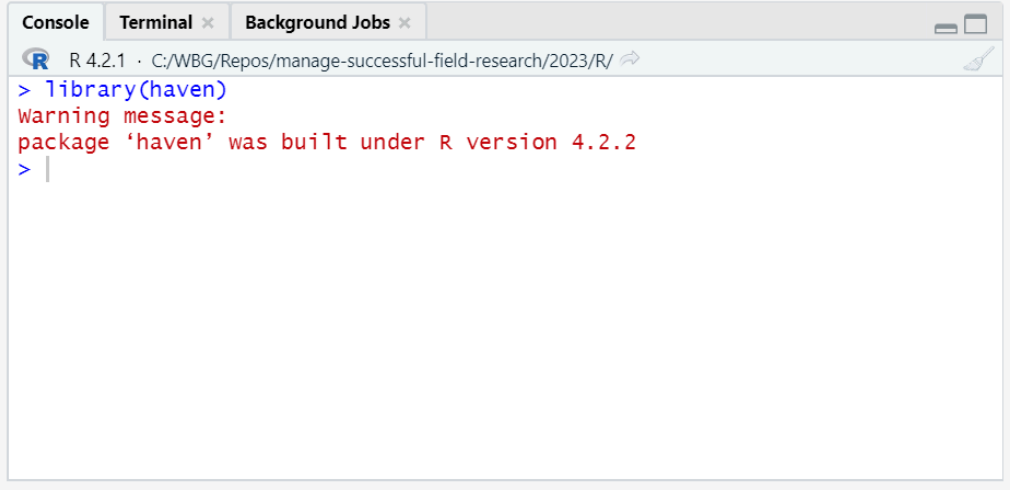


R libraries

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

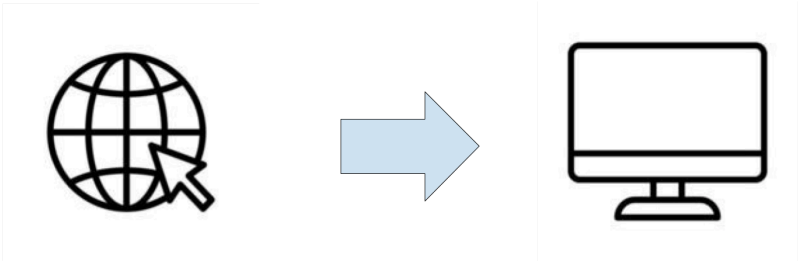
Exercise 8: Loading libraries

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")`

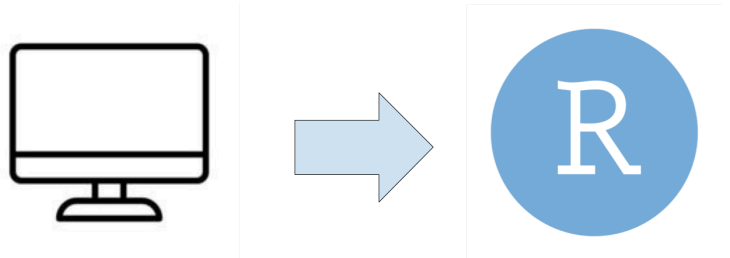


R libraries

- 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)

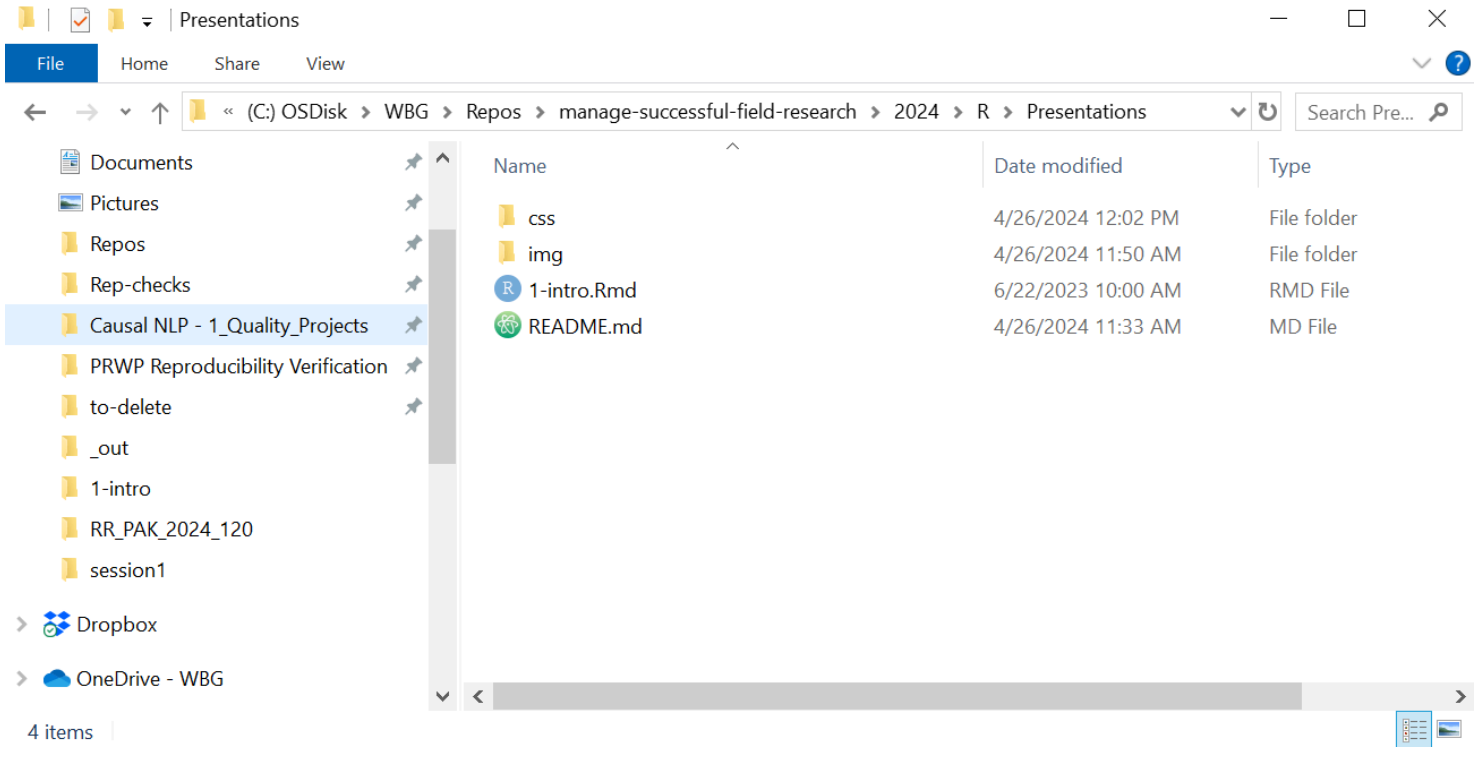
File paths

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

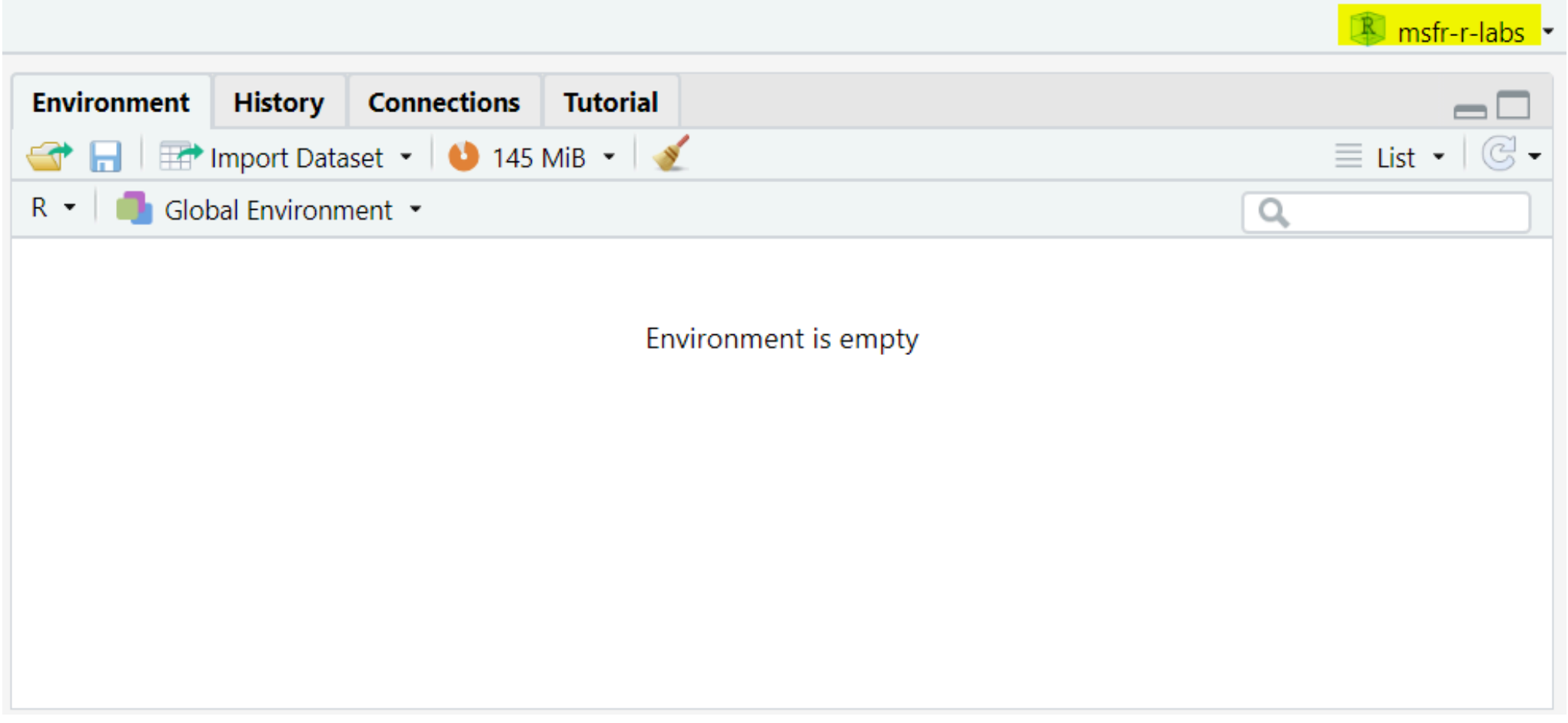
Save your current script by clicking in the floppy disk icon **before starting this exercise**.

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 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

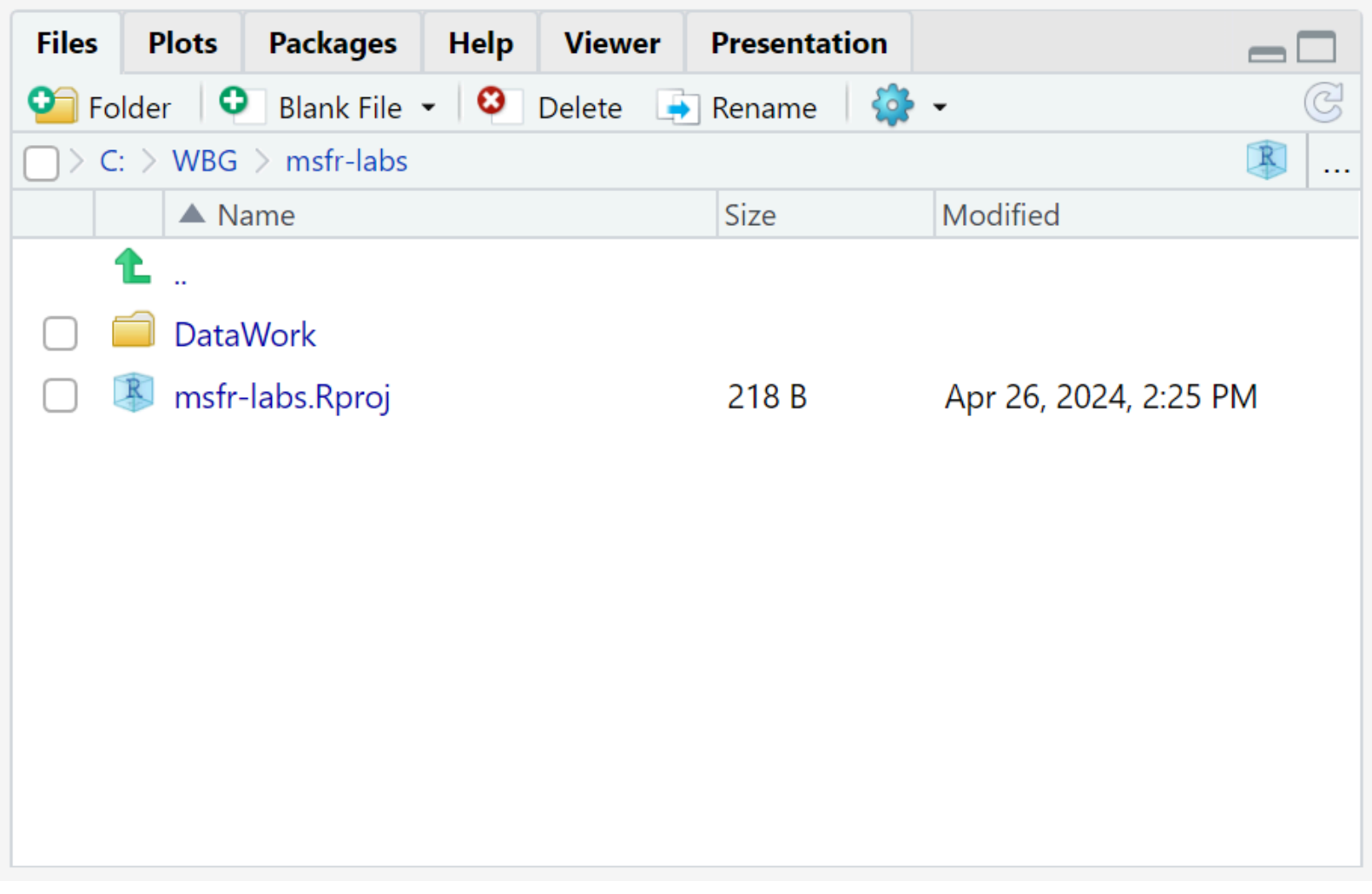
File paths

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 at 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

```r
# 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 scrips to start with a section loading libraries and a next section loading data

File paths

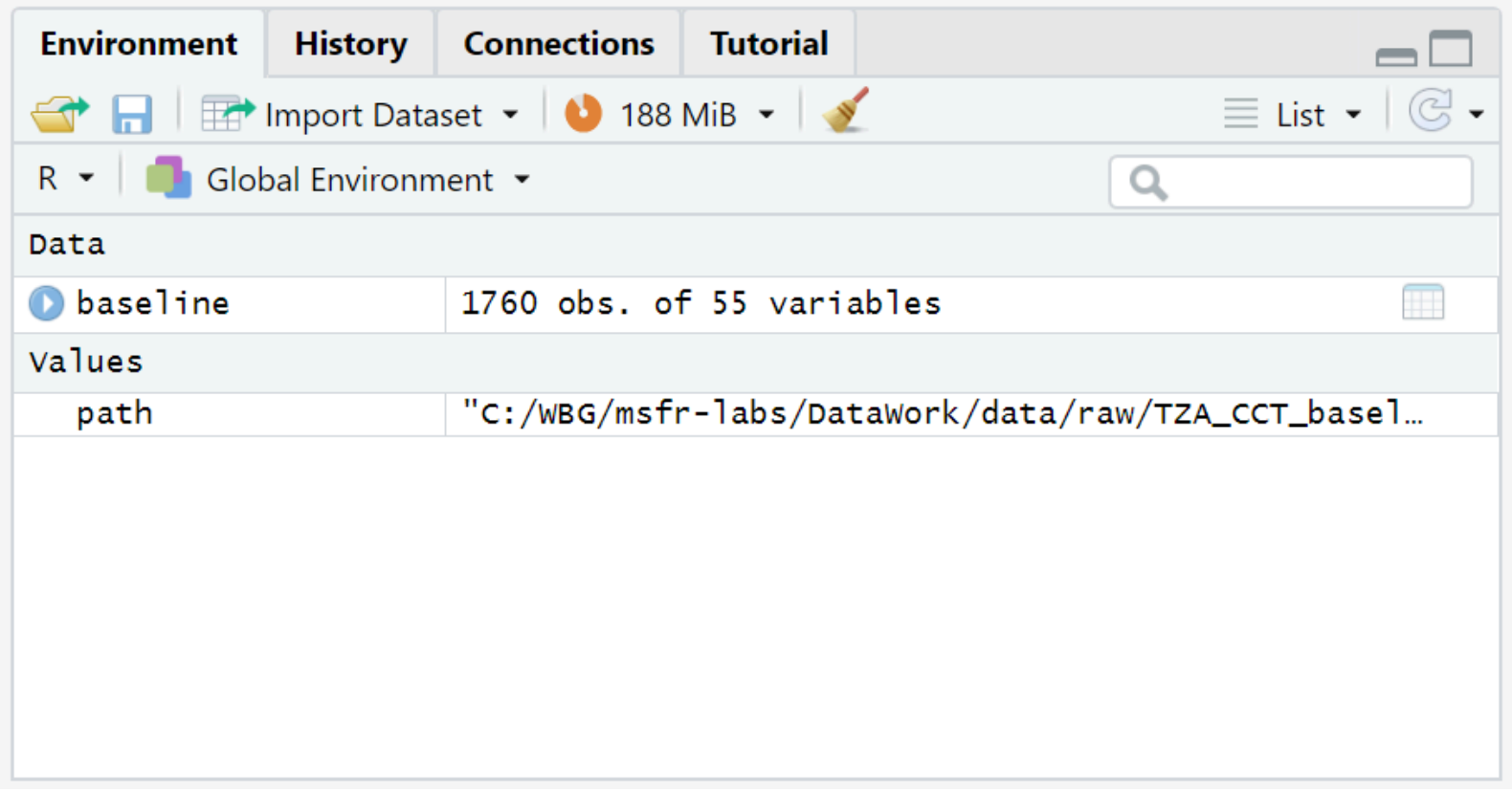
```
# Loading libraries
library(here)
library(haven)

# 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



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.

Exercise 11: Load data with pipes

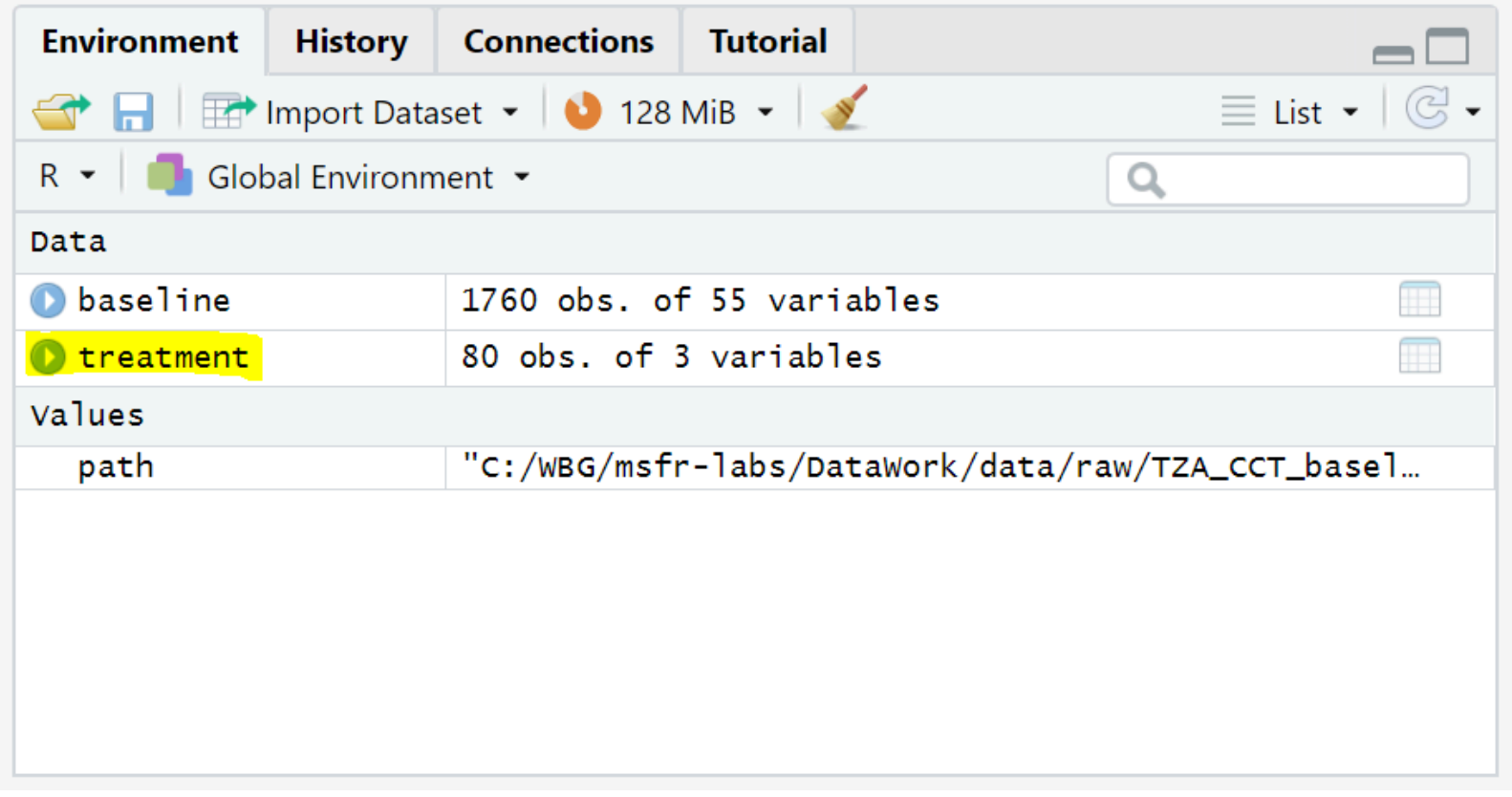
We'll load the same data we loaded manually in Exercise 4, this time with code.

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()
```



Thanks! Gracias!