Starting with R and RStudio

A Workshop for Absolute Beginners

Plan for today

1. Installation
2. RStudio Layout
3. Code Basics
4. Objects
5. Packages
6. Importing Data
7. Descriptive Statistics
8. Distributions (Plots)

What you need for today…

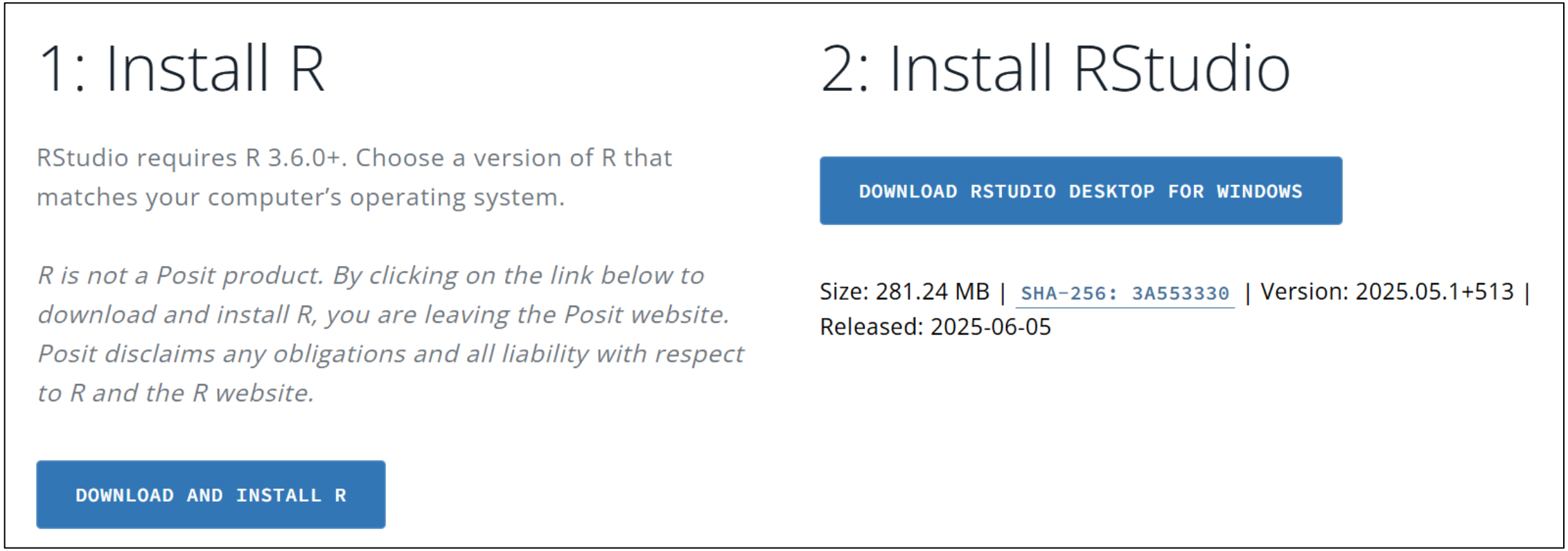
- Access to RStudio (installation details on the next slide).

- **Three data files:**
  - Access via Psychology Staff Moodle Page >

1. Installation

Installation

- www.posit.co/download/rstudio-desktop/



R

- R is a programming language.
- RStudio is the interface you write the language into.
- **Both** need to be installed.
- We only really need to work in RStudio.
  - What's the difference?

What's the difference?

RStudio®

You? (and me!)

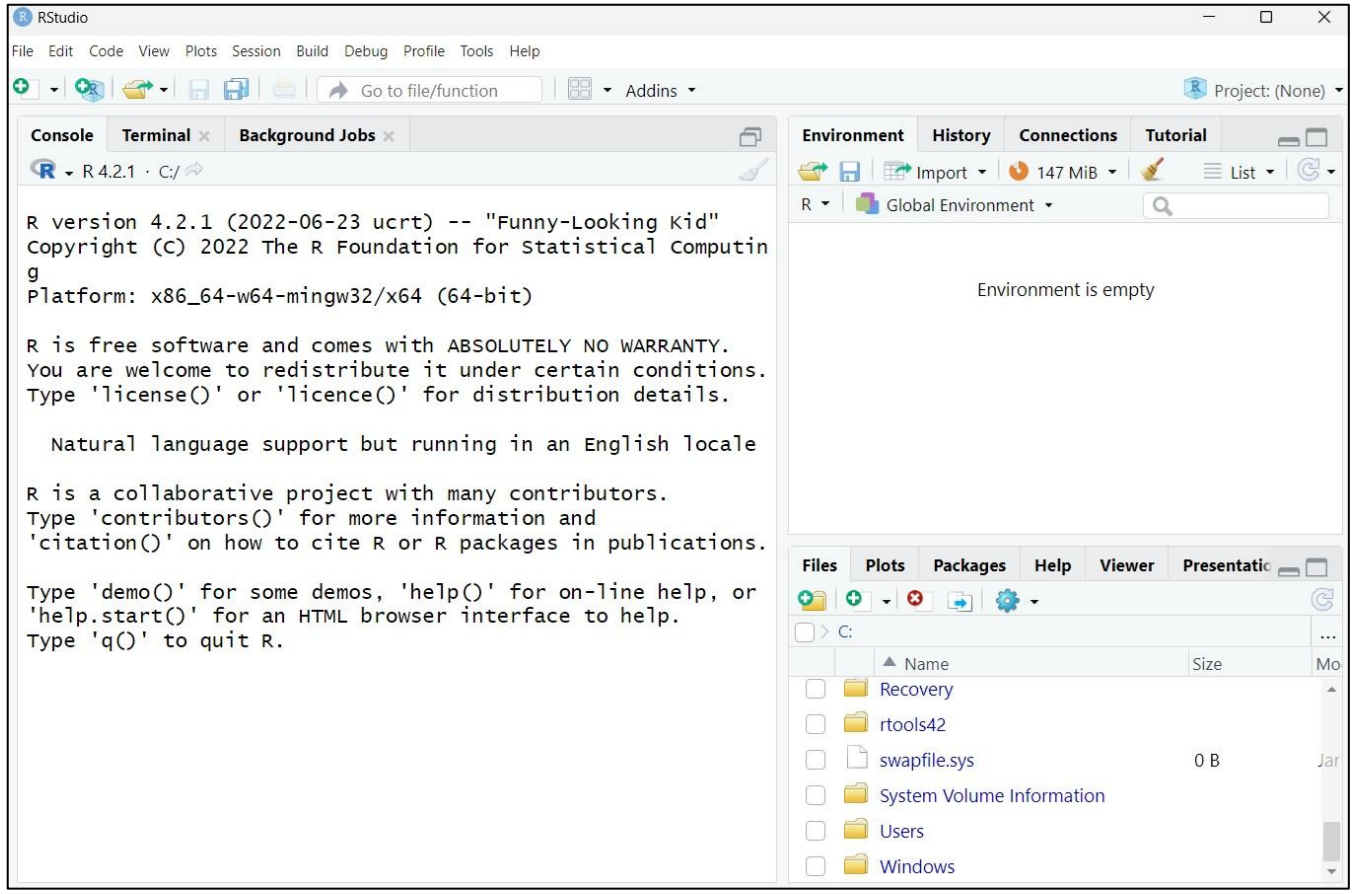
Learn to drive the car.

Not become the mechanic!

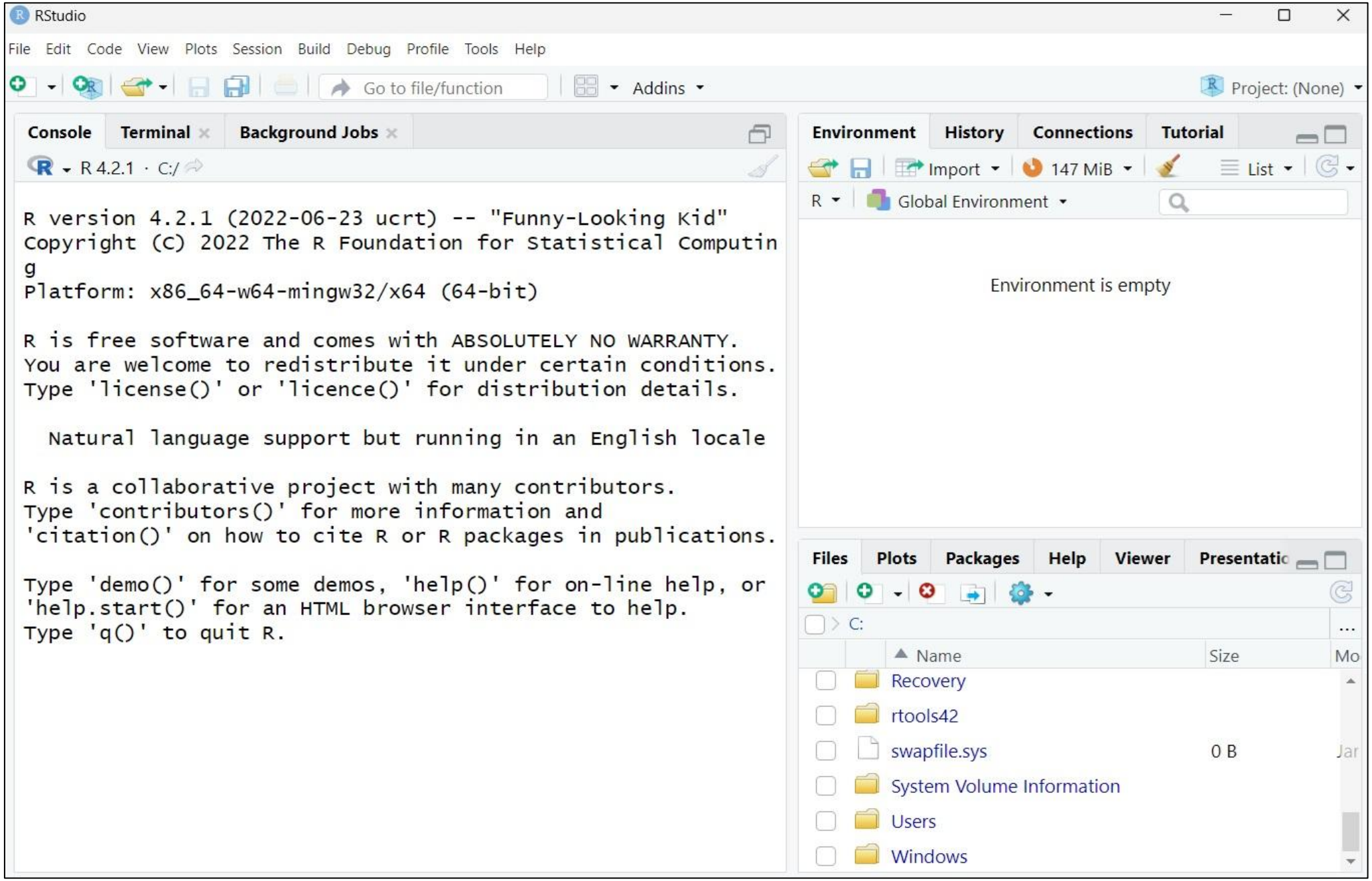
Installation

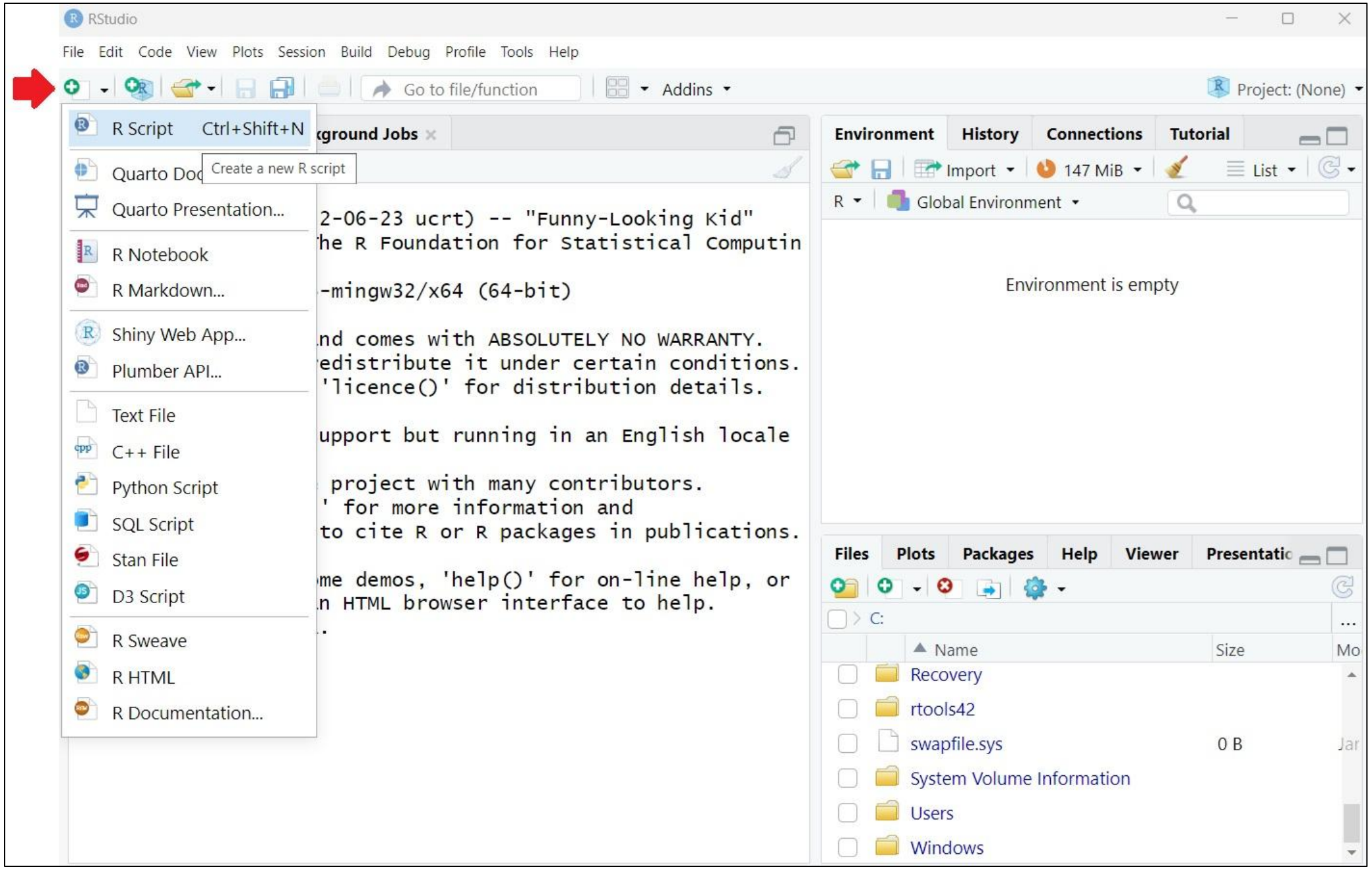
- Once **both** are installed…only get into the car.

- Open RStudio…



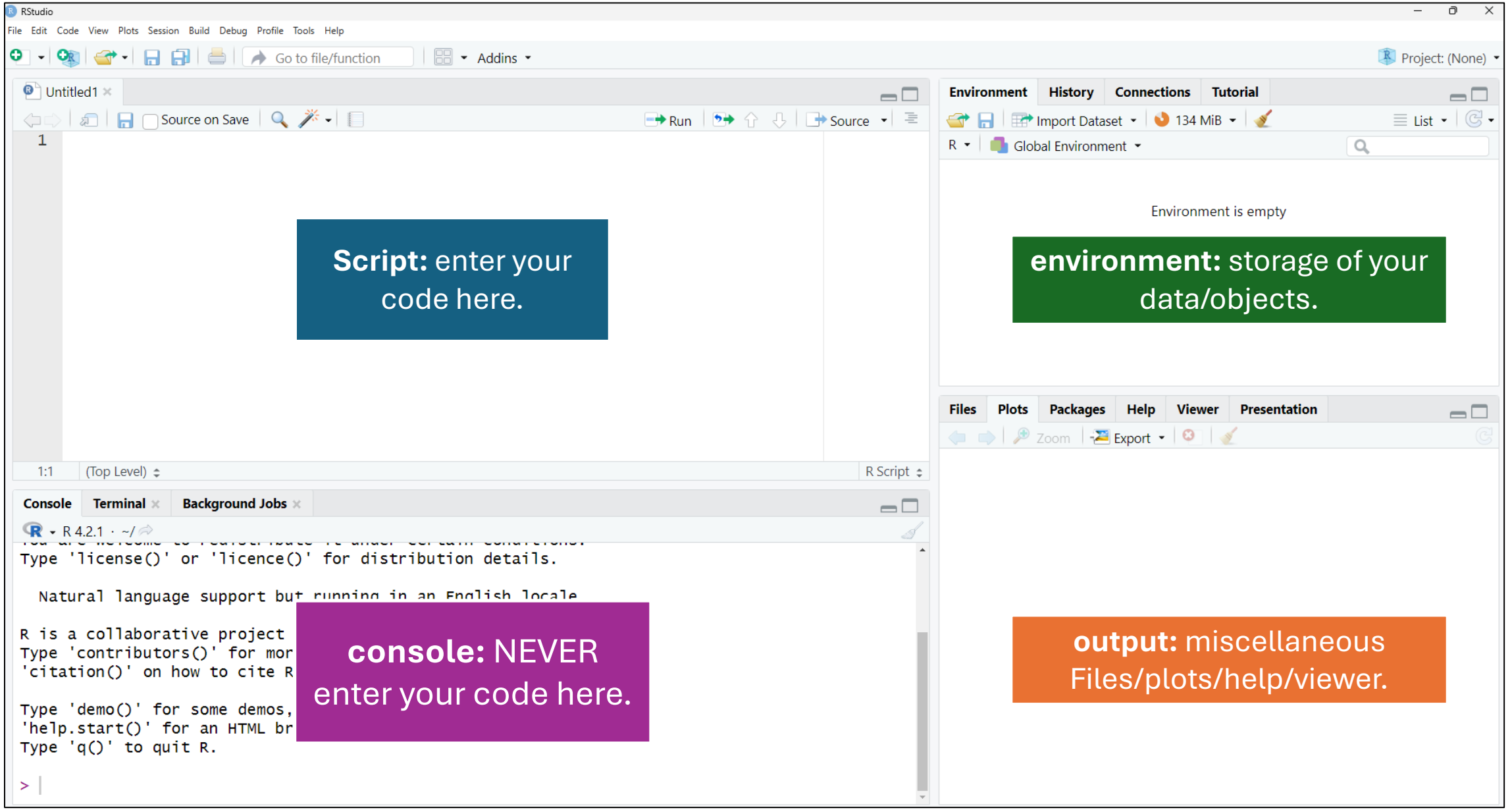
2. RStudio Layout


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console
Terminal
Background Jobs
R 4.2.1 · C:/
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.
Natural language support but running in an English locale
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.
Environment
History
Connections
Tutorial
Import
147 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentatic
C:
Name
Size
Mo
Recovery
rtools42
swapfile.sys
0 B
Jar
System Volume Information
Users
Windows


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
R Script Ctrl+Shift+N
Quarto Doc
Create a new R script
Quarto Presentation...
R Notebook
R Markdown...
Shiny Web App...
Plumber API...
Text File
C++ File
Python Script
SQL Script
Stan File
D3 Script
R Sweave
R HTML
R Documentation...
kground Jobs
2-06-23 ucrt) -- "Funny-Looking Kid"
he R Foundation for Statistical Computin
-mingw32/x64 (64-bit)
nd comes with ABSOLUTELY NO WARRANTY.
edistribute it under certain conditions.
'licence()' for distribution details.
upport but running in an English locale
project with many contributors.
' for more information and
to cite R or R packages in publications.
me demos, 'help()' for on-line help, or
n HTML browser interface to help.
Environment
History
Connections
Tutorial
Import
147 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentatic
C:
Name
Size
Recovery
rtools42
swapfile.sys
0 B
System Volume Information
Users
Windows

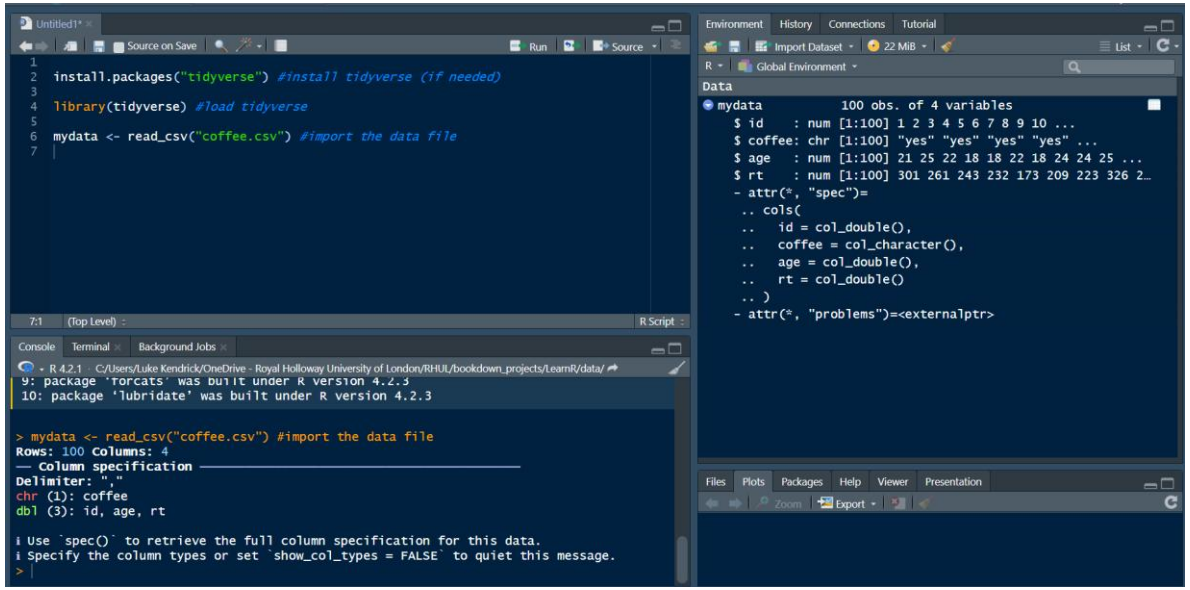
R Studio Layout

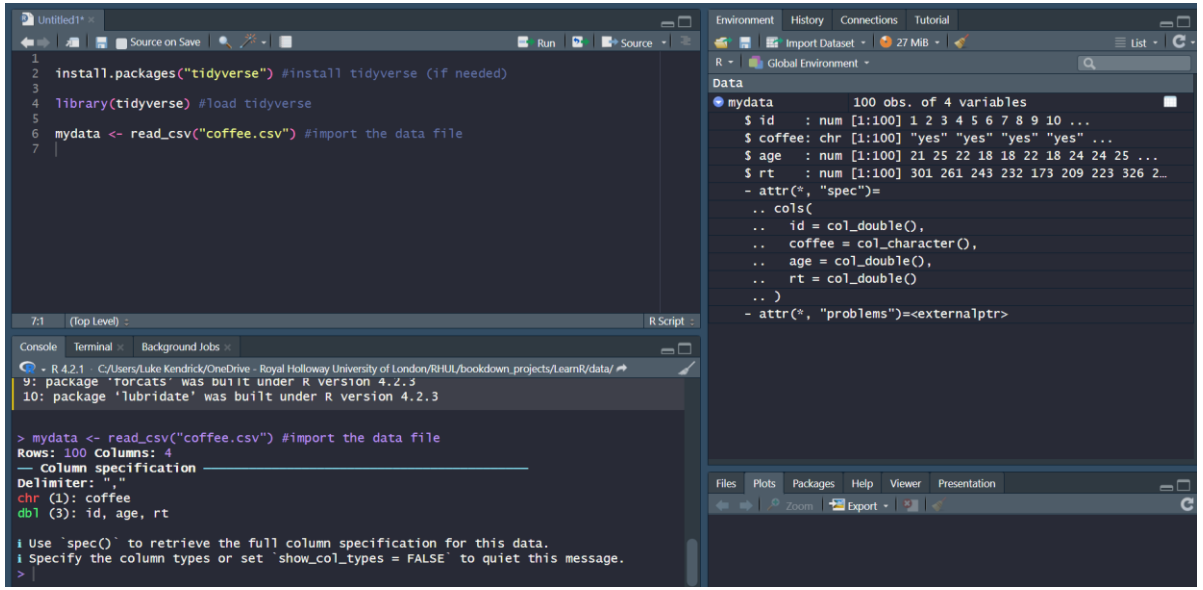
You need **four** panels before you start working!



Other tips

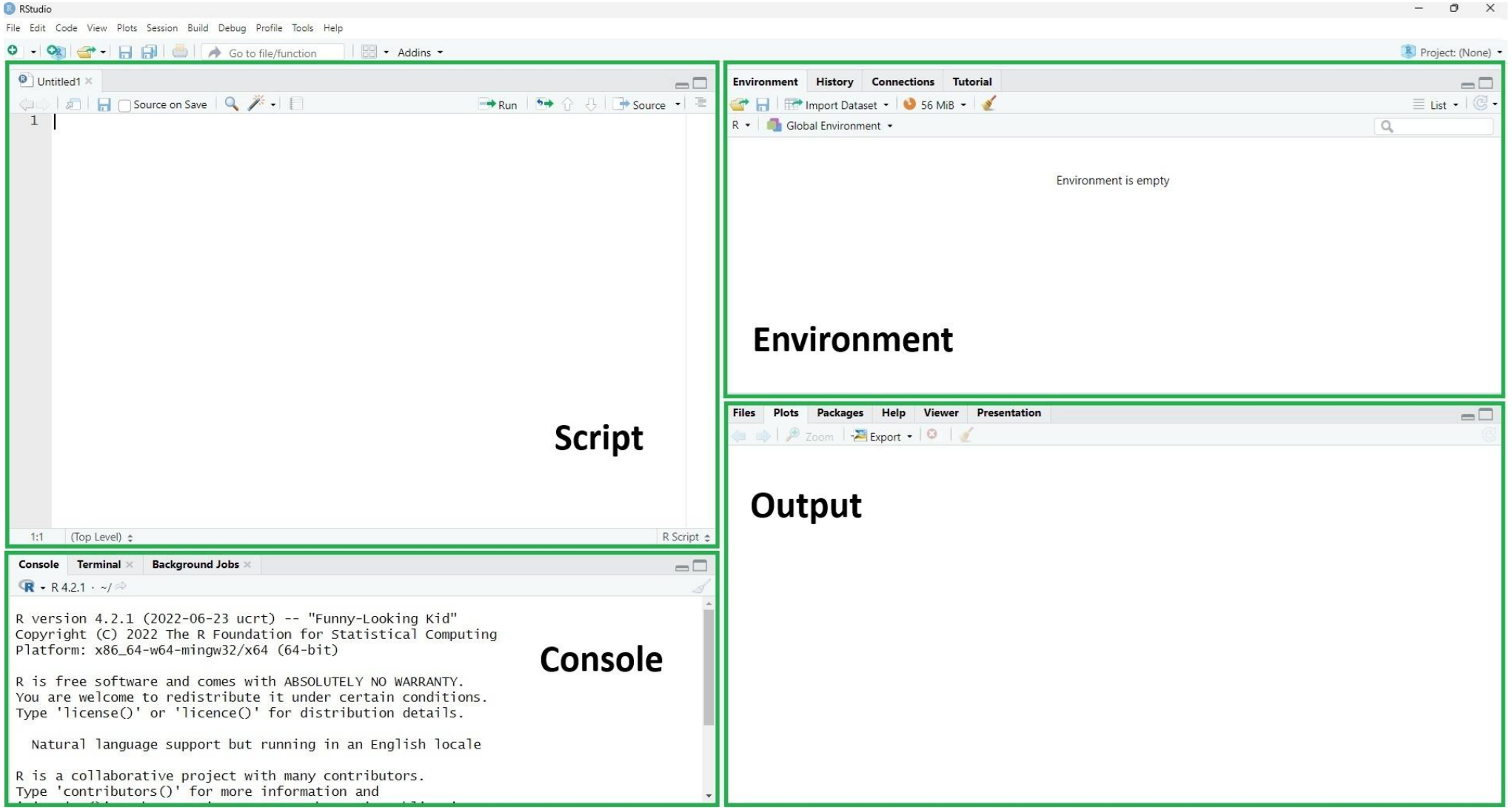
- Theme: Tools -> Global Options -> Appearance -> Editor Theme



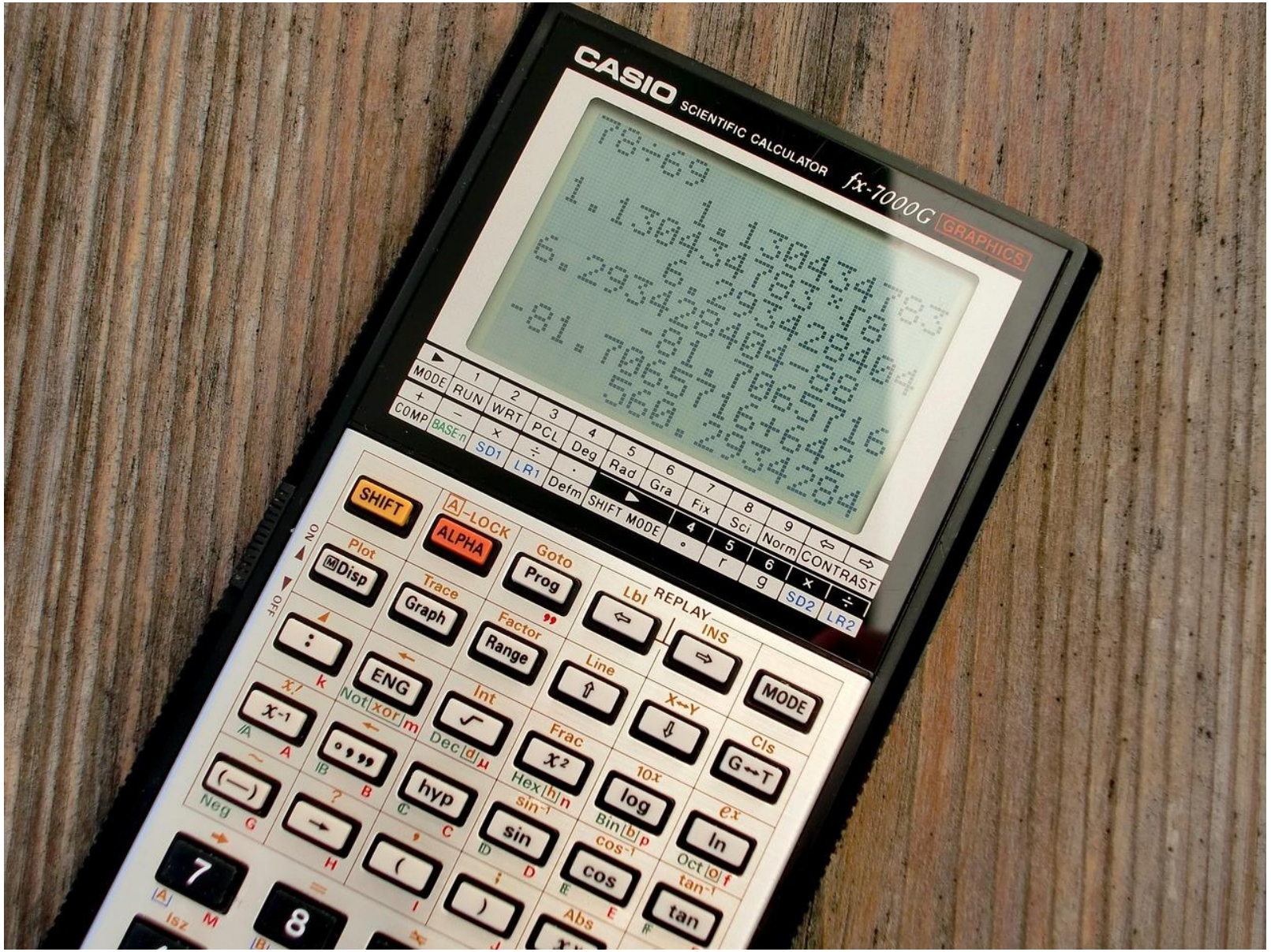


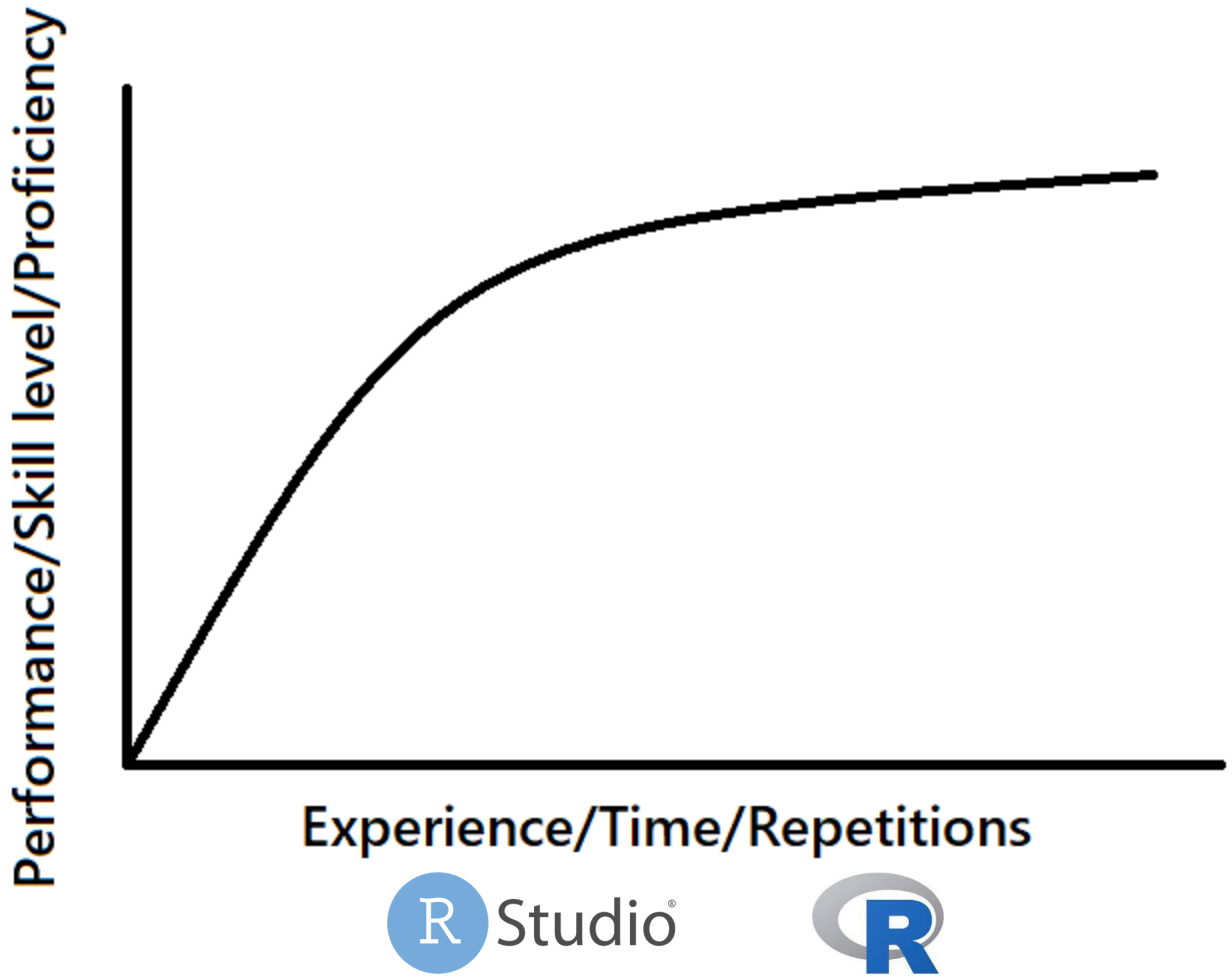
- Pane layout: Tools -> Global Options -> Pane Layout
  - [probably best once more advanced, move things to personal preference]

Summary


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Script
Environment History Connections Tutorial
Import Dataset
56 MiB
List
R
Global Environment
Environment is empty
Environment
Files Plots Packages Help Viewer Presentation
Zoom
Export
Output
Console Terminal Background Jobs
R 4.2.1 · ~/
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.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
Console

3. Code Basics


CASIO
SCIENTIFIC CALCULATOR
fx-7000G
GRAPHICS
78÷69
1.130434783
6.293428404
-81.7065716
560.2934284
MODE
RUN
WRT
PCL
Deg
Rad
Gra
Fix
Sci
Norm
CONTRAST
COMP
BASE-n
SD1
LR1
Defm
SHIFT MODE
SD2
LR2
SHIFT
ALPHA
A-LOCK
Plot
Disp
Goto
Prog
Trace
Graph
Factor
Range
Lbl
REPLAY
INS
MODE
ENG
Line
Int
Frac
Cls
G↔T
hyp
log
sin
cos
ln
tan
Neg
Abs
7
8


Performance/Skill level/Proficiency
Experience/Time/Repetitions
RStudio
R

Code Basics

- Think of code as a set of instructions.

- Some can be short others require multiple steps.

- Let's try some very basic instructions to learn about how R and RStudio works…

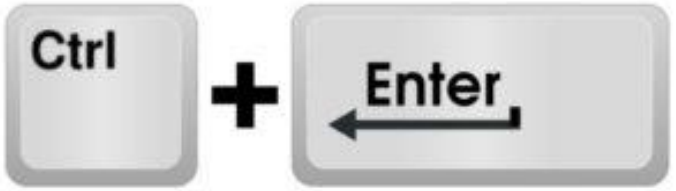
- Enter the following into the script (top left) panel:

```
date()
```

Run

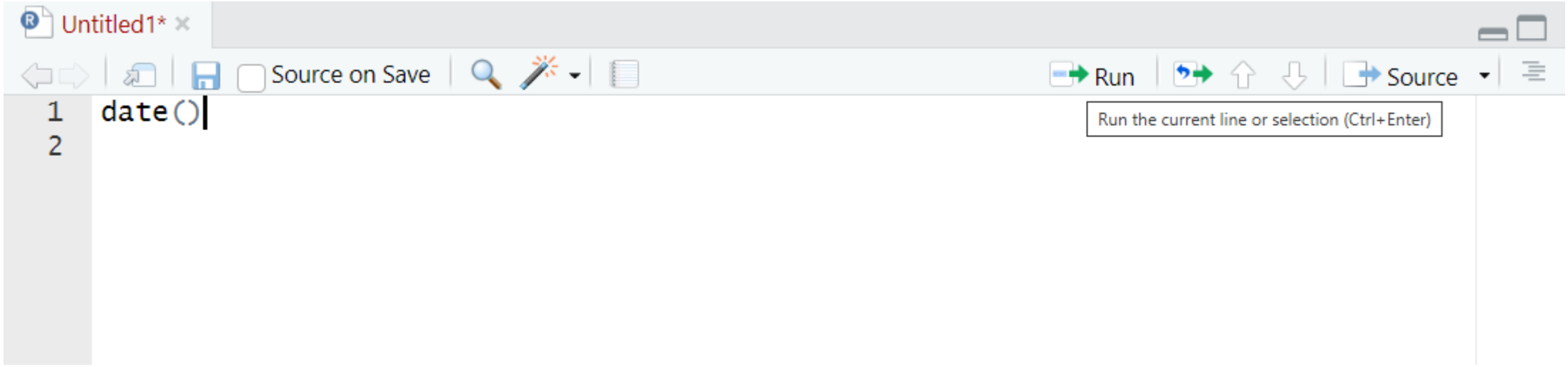
Ensure cursor is at the start or end of the line of code…

- Control + Enter (Windows)
- Command + Enter (Mac)



Run

- Alternative:
- Click “Run” at the top of the script.
- This is the “point and click” approach. [try to use your keyboard rather than mouse]



Code Basics

```
20+80
```

Add

```
2561*13
```

Multiply

```
37/4
```

Divide

```
230-62.8
```

Subtract

Functions and Arguments

- A function is a specific piece of code that will perform a specific task.

```
date()
```

- You can learn about a function at any time using `?`

```
?date()
```

- Check the bottom right panel.

Functions and Arguments

- An argument is information that you pass into a function.

```
function(argument)
```

- Now try a real function with a basic argument:

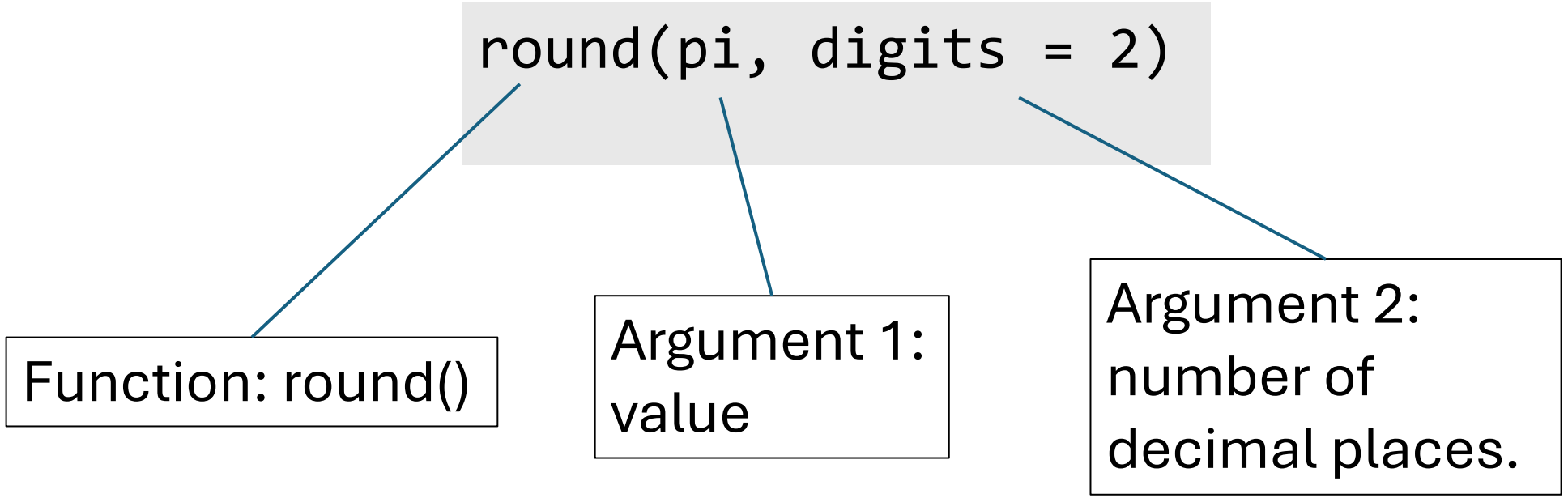
```
round(3.14159265359)
```

- Save time:

```
round(pi)
```

Functions and Arguments

- We can add **multiple arguments.**



Code is very particular…

- Try the following function:
  - NOTE: use a capital `R` to spell `Round()`

```
Round(pi)
```

```
> Round(pi)
Error in Round(pi) : could not find function "Round"
```

- R code is case sensitive: round() is **NOT** the same as Round().

- Best practice is to generally avoid using capital letters!

# #

- “How am I supposed to remember every bit of code and what it does?!”

- #Comments!

- In R we can use `#` to annotate code.

```
round(pi, digits = 2) #round pi to 2dp
```

- Anything after a hashtag will not be run by R.

Top Tips

1. Code is very particular and is case sensitive!
   - Most errors early on are due to typos.

2. Use # to make notes throughout to annotate your code.
   - Reminder to your future self but also anyone else who uses your code.

4. Objects

Creating an object

- We can assign any information or data set to an object.

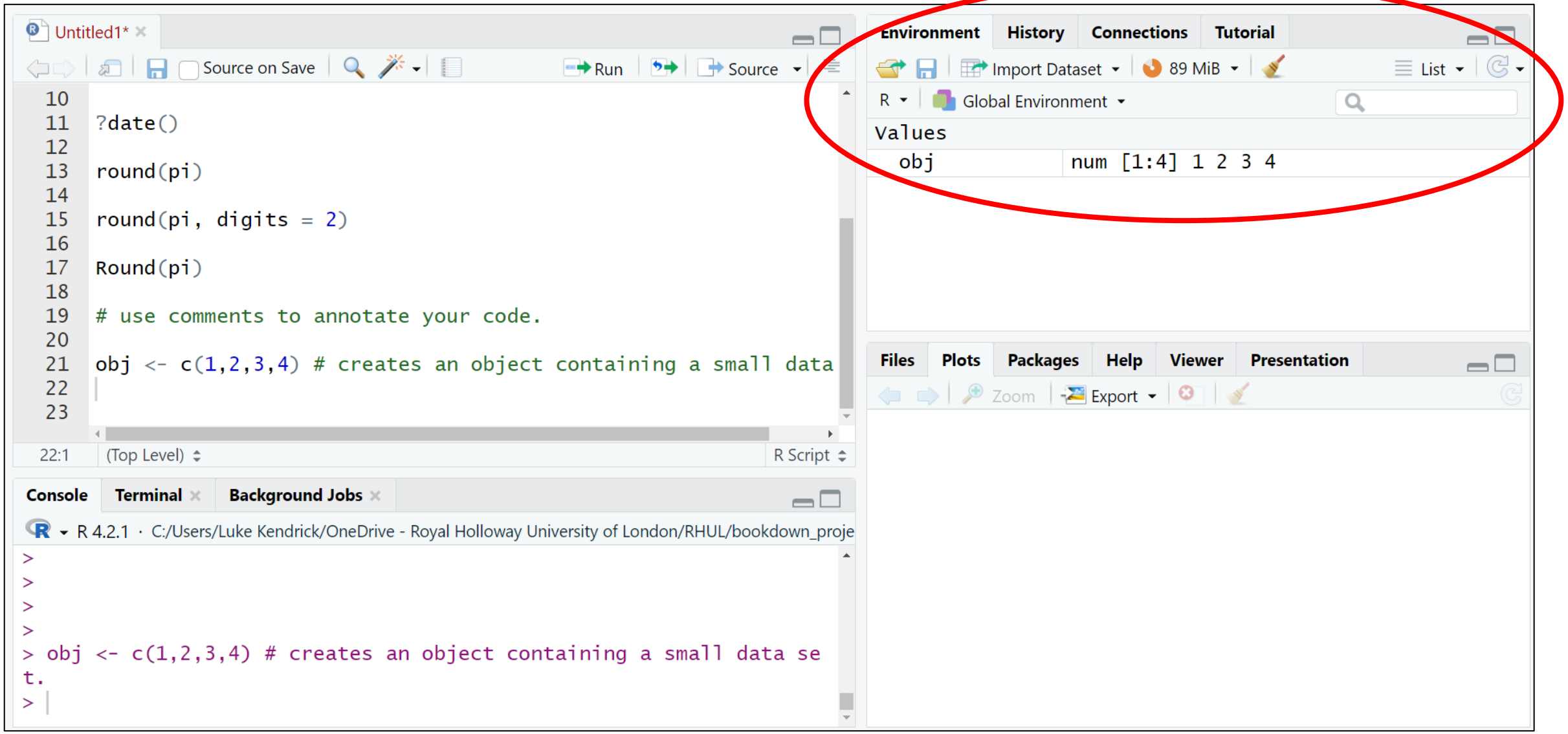
- Objects are temporarily stored in the environment (top right panel).

```
obj <- c(1, 2, 3, 4)
```

Code explanation:

- obj: the name of our object.
- <-: an arrow that is called an assignment operator. Anything to the right of it will be saved as that object.
- c(): a function that allows us to create a list. The list contains a data set with numbers 1-4.

Check the environment panel



Creating an object

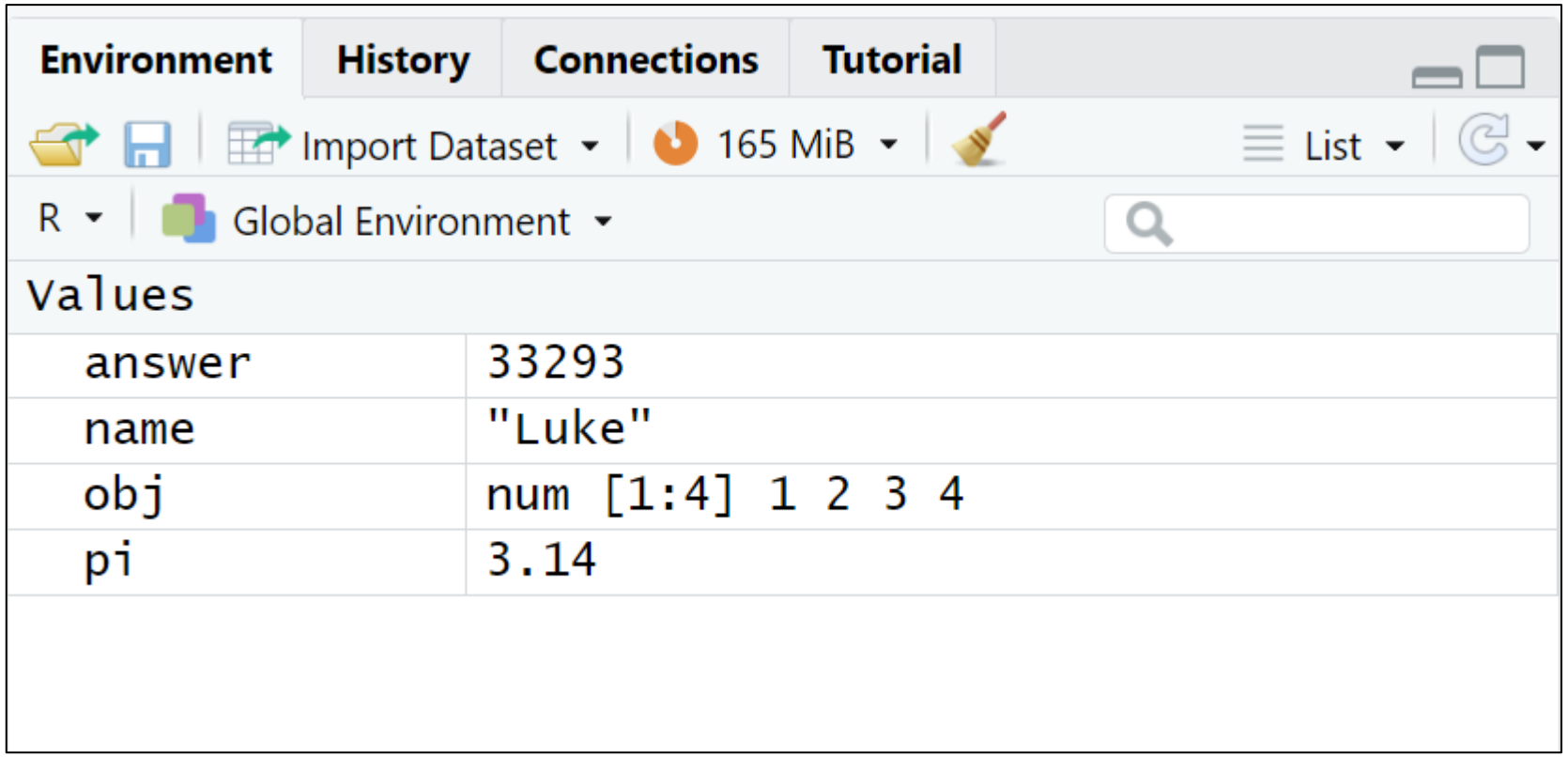
- Create more objects.

```
name <- “YOUR NAME” #use quote marks.

answer <- 2561*13

pi <- round(pi, digits = 2)
```

All objects are stored in the environment



Think of the environment as RStudio's working memory.

Print objects to the console

- Run the following and check the console (bottom left panel).

```
print(obj)
print(name)
print(answer)
print(pi)
```

```
print(Obj)
```

- Remember caps! `obj` is not the same as `Obj`.

Keep object names short

- We tend to repeatedly use objects during data analysis.
  - E.g., a data set saved as an object.

- Keep object names short and simple but informative.

- Long object names are unhelpful:

```
What_is_my_first_name <- “Luke”
```

- Keep it short:

```
name <- “Luke”
```

Remove objects.

- Sometimes we want to remove an object from the environment:

```
rm(name) # will remove the object called “name”.
```

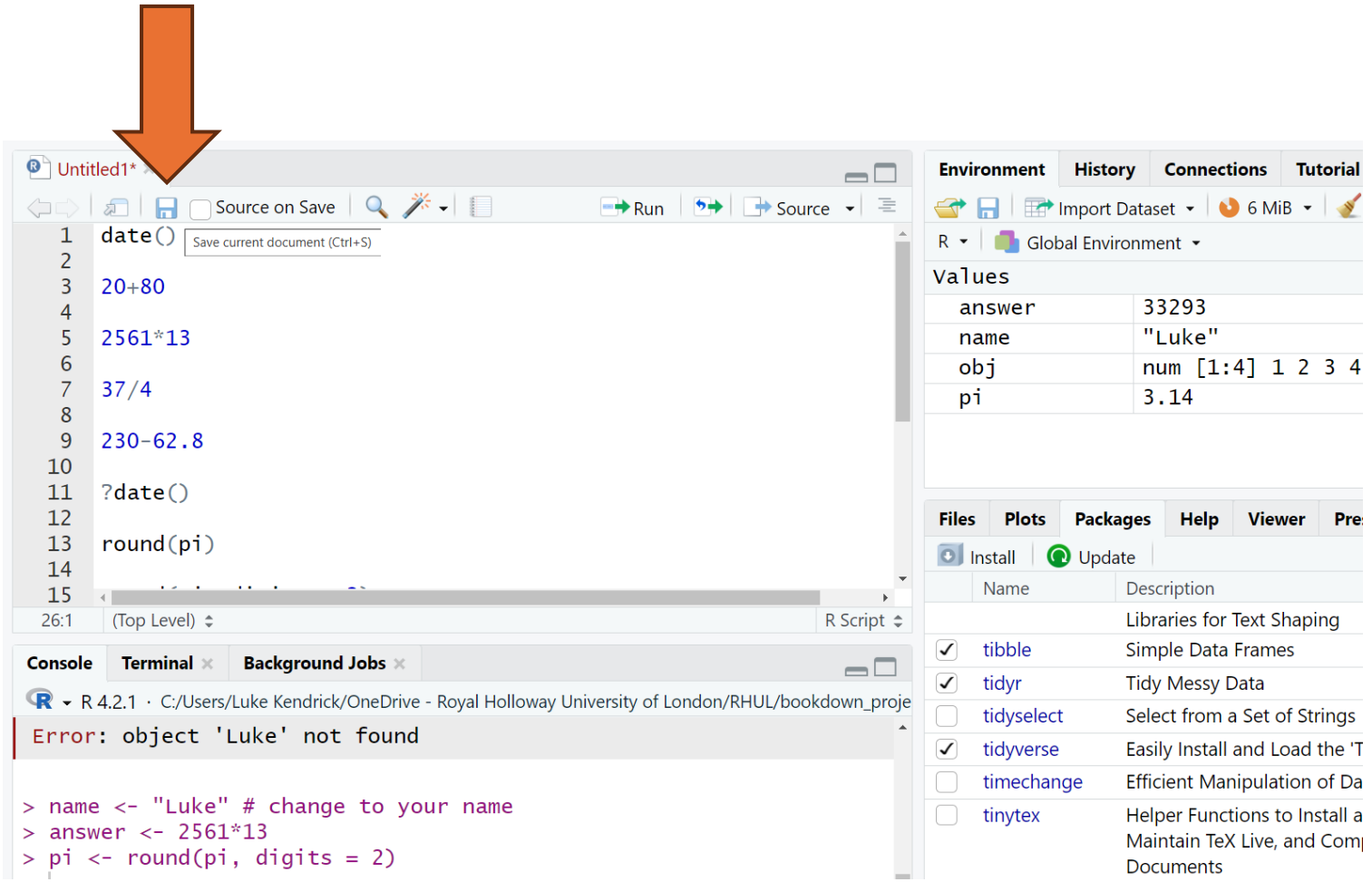
- Start fresh and remove ALL objects:

```
Rm(list = ls()) # clears everything from the env.
```

Save your script!

- Now might be a good time to save your script!

- Perhaps create a new folder on your computer to save it in.



5. Packages

What are packages?

- Many “base” functions exist in RStudio.
  - E.g., date() and round()
- Some functions are part of packages.
- Base RStudio is the “basic subscription” but with packages we will use “add-ons”.
  - Or think of packages like an “app” installed on your phone.
- Welcome to the Tidyverse…

Installing and loading a package

- Let's install Tidyverse **(IF NEEDED):**

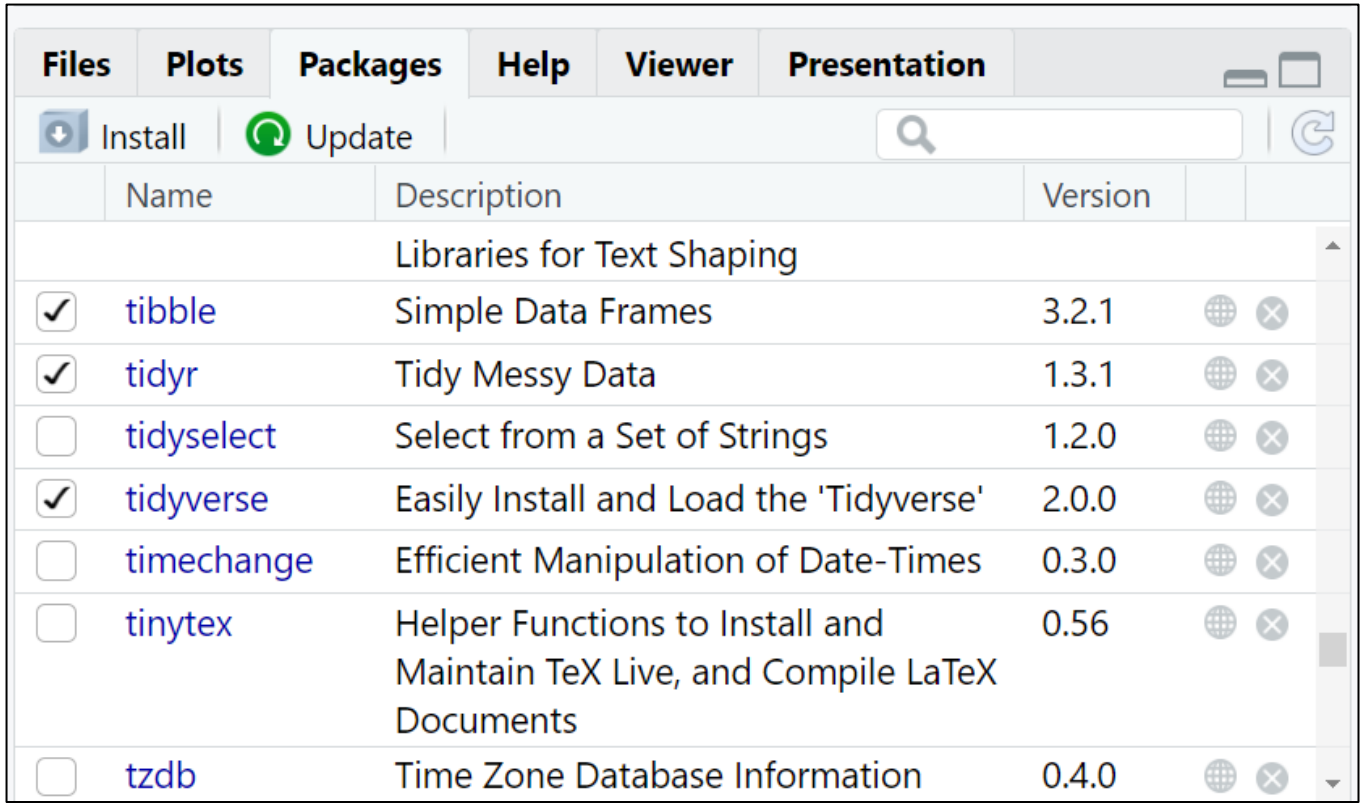
```
install.packages("tidyverse") #installs tidyverse
```

- Loading a package:

```
library(tidyverse) #will load the package
```

Check packages

- Use packages tab in bottom right panel to check all installed packages.
- A tick means the package has been loaded and ready to use.



6. Importing Data

Working directory

- The working directory is the folder on your computer that RStudio will look to find any files.

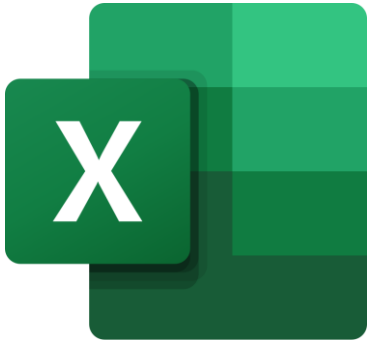
- It is also the place RStudio can save or export files to.

- The way to set-up folder depends on personal preference!

- Download the data set and make a note where you save it.
  - **Maybe save it in the folder you created earlier to save your script!**
  - No need to open it in excel as we will import it to RStudio.

A note on data files

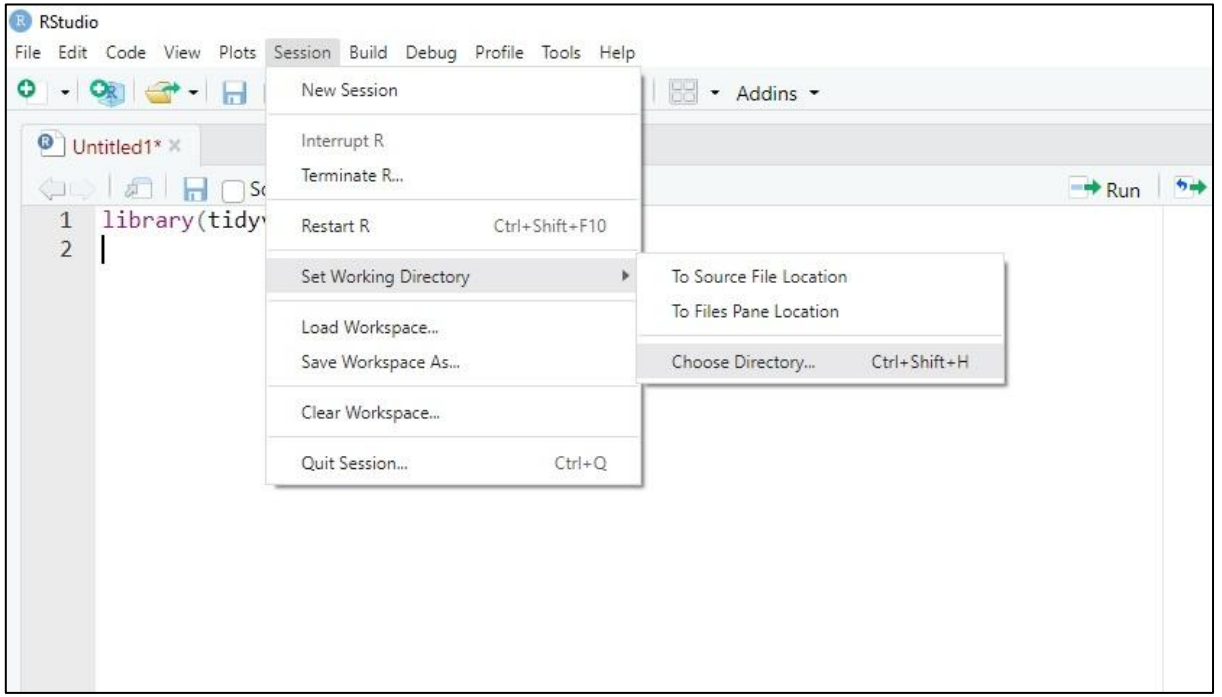
- Raw data can be saved into an excel spreadsheet.



- But **must** be saved as a .csv file
  - “comma separated values”.
- RStudio can handle different formats, even SPSS files!
  - https://support.posit.co/hc/en-us/articles/218611977-Importing-Data-with-the-RStudio-IDE
- Students have been taught to use .csv files.

Set the working directory

- Session -> Set Working Directory -> Choose Directory.
- Choose the **folder** your data is saved to.

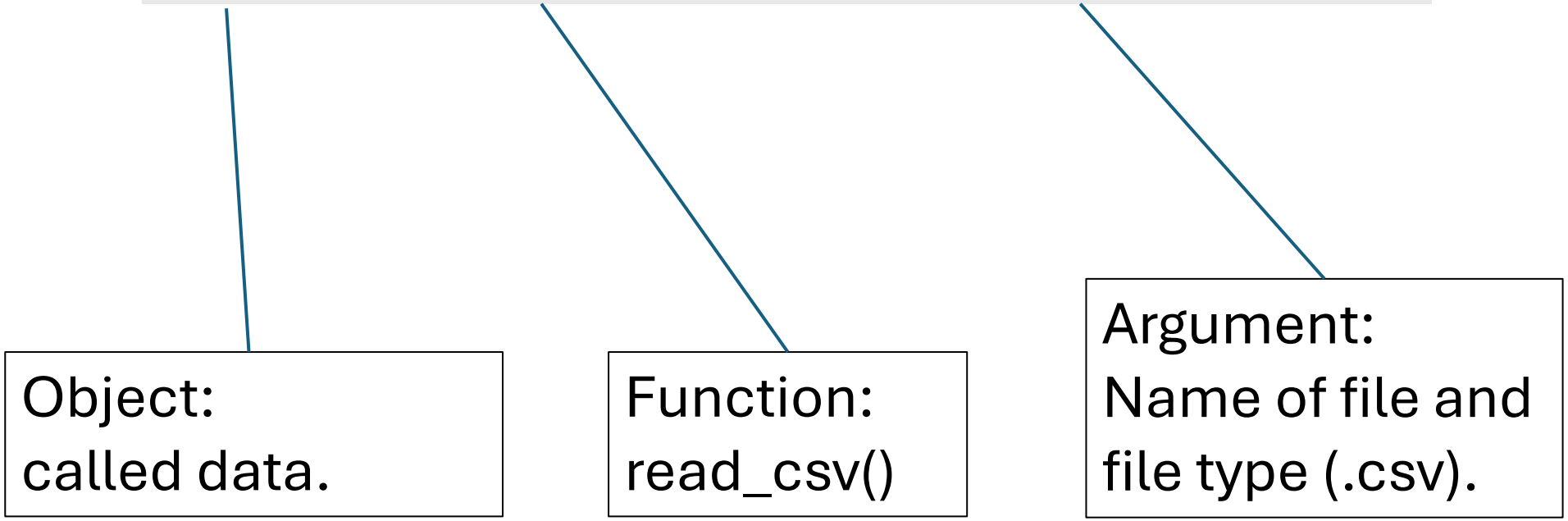


Importing data

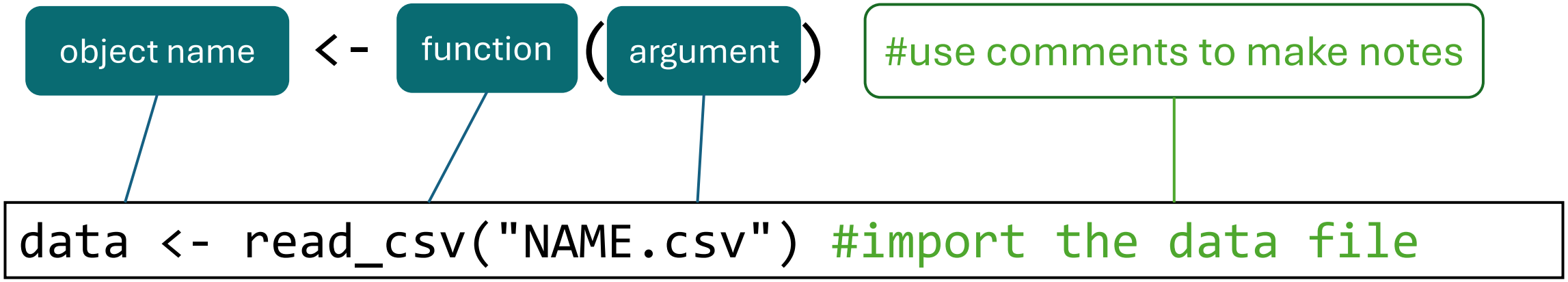
- The quickest way to import data is using code.

- Students were taught to use the read_csv() function@

```
data <- read_csv("NAME OF FILE".csv)
```



Objects, Functions, and Arguments



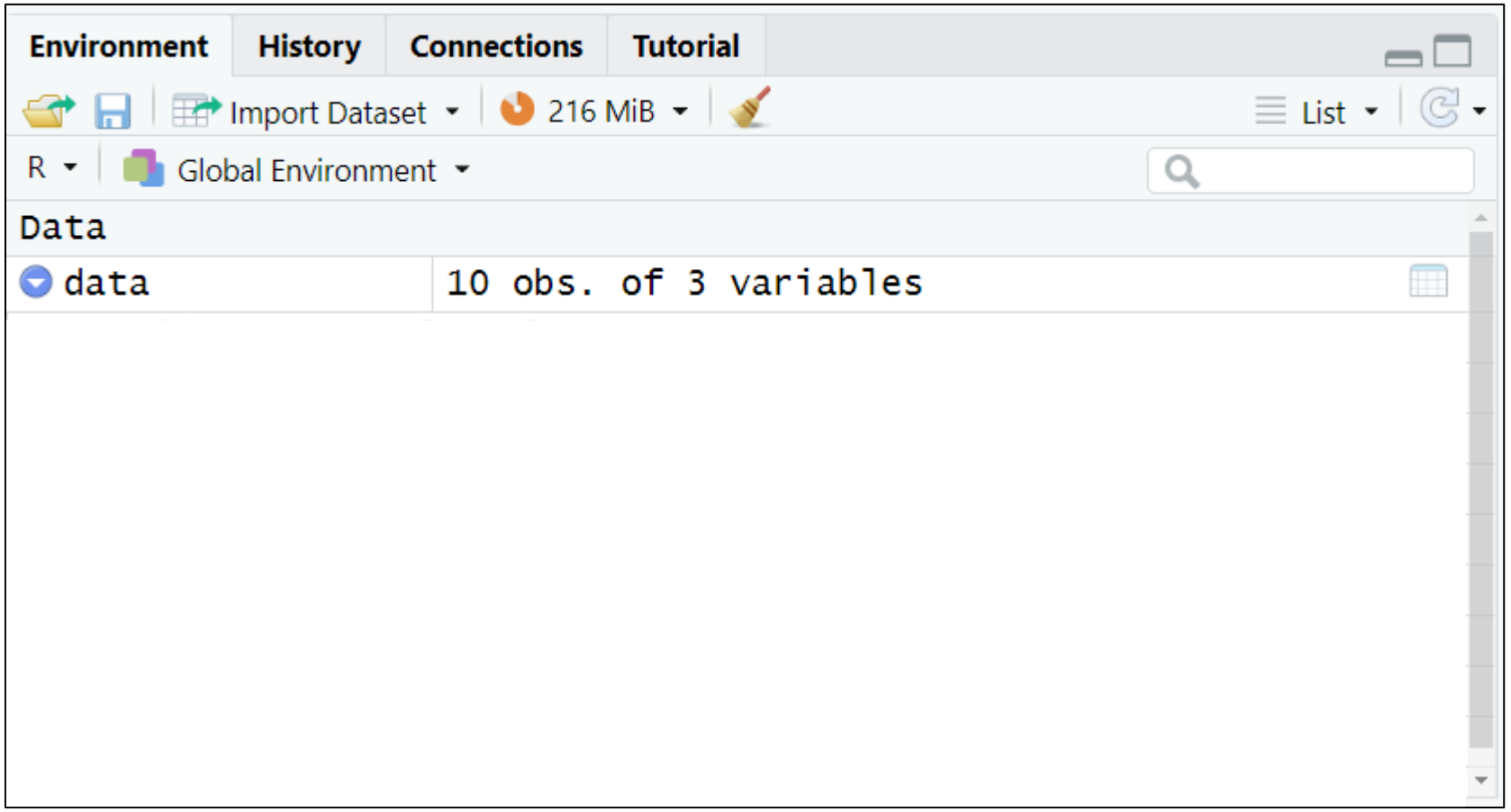

```
data <- read_csv("coffee.csv") #import the data file
```

- Creates an object called “data”.
- Uses the function “read_csv()”.
- Includes the argument which is the name of the .csv data file in quote marks.
- A file called coffee.csv saved in the working directory will be imported…

Check the environment panel

Clicky click the blue symbol!

- This will expand the object to show the characteristics of the data set.



Check your data

- Let's install Tidyverse **(IF NEEDED):**

```
print(data) # print data to the console
```

- Loading a package:

```
view(data) # open data in a new tab
```

7. Descriptive Statistics

Import and explore the new data set

```
rm(data) # will remove the object called “data”.
```

```
data <- read_csv(“sleep.csv”) # new data set
```

```
head(data) #view the first few rows
summary(data) #quick summary of the data set
names(data) #check variable names
```

count() and pipe %>%

```
data %>%
     count(condition)
```

What is %>% ?

- This is a pipe.
- It essentially means “and then”.

The code above reads as:

“take our data” … “and then”

“count the observations in each condition”

Means and Standard Deviations

```
descr <- data %>%
     summarise(mean_age = mean(age),
               sd_age = sd(age),
               mean_change = mean(change))
```

Anything in **orange** is an object or variable from our data set

Anything in **blue** is a label I have created for the output.

`desc` is a new object I have created for the output (short for descriptives).

- We can view this object using the code below:

```
view(desc)
```

summarise() function

- This function allows us to calculate summary statistics.

- Here is a list of key statistics you might want to use in the future (remember to change NULL to the name of your variable):
  - mean(NULL)
  - sd(NULL)
  - median(NULL): will calculate the median.
  - min(NULL) or max(NULL): will provide the minimum and maximum values respectively.
  - n(): will provide the sample size or count of observations. No need to add anything in the parentheses.
  - var(NULL): will calculate the variance.

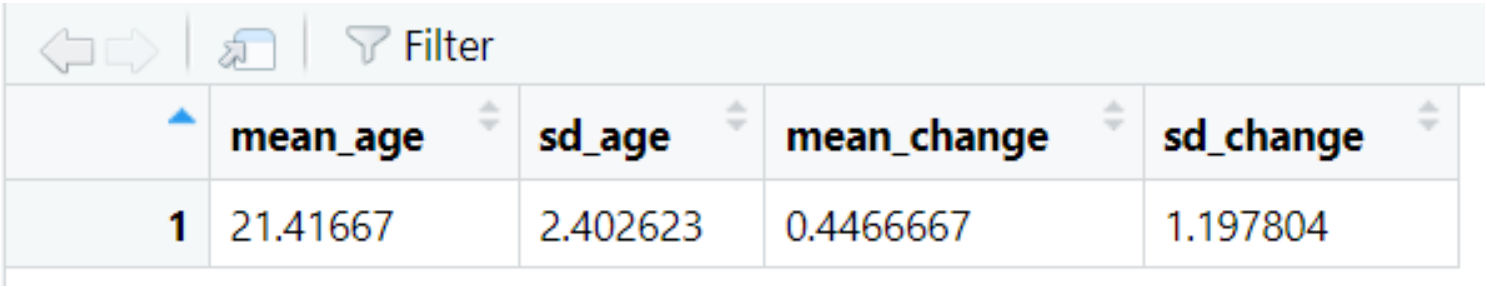
Summarise() function

- Amend your code so it calculates the standard deviation for "change". *HINT: add a line of code for sd_change*.

```
descr <- data %>%
     summarise(mean_age = mean(age),
               sd_age = sd(age),
               mean_change = mean(change))
```

```
descr <- data %>%
     summarise(mean_age = mean(age),
               sd_age = sd(age),
               mean_change = mean(change),
                                  )
```

view(desc)



Filter

	mean_age	sd_age	mean_change	sd_change
1	21.41667	2.402623	0.4466667	1.197804

- This will tell us about the **overall** age and sleep change…
- …but what about splitting by group?

group_by() function

group_by()

```
descr <- data %>%
     group_by(condition) %>%
     summarise(mean_age = mean(age),
               sd_age = sd(age),
               mean_change = mean(change)
               sd_change = sd(change))
```

- Add group by to **split** the data by a variable, in this case “condition”.

- It **MUST** come before the summary statistics are calculated.
  - There is no point in splitting the data after the statistics were calculated.

view(desc)

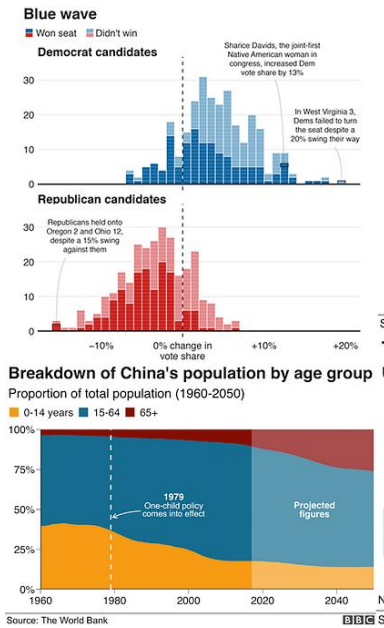
Filter

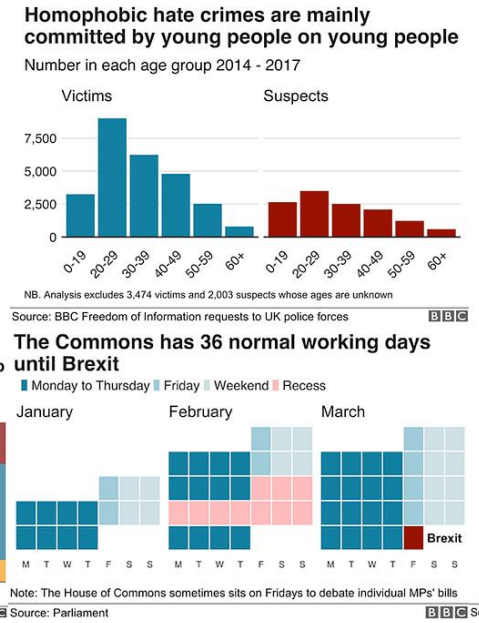
	condition	mean_age	sd_age	mean_change	sd_change
1	control	21.55	2.723678	-0.15	1.0932809
2	int_drug	21.50	2.328315	0.29	1.2030225
3	int_edu	21.20	2.238420	1.20	0.9026277

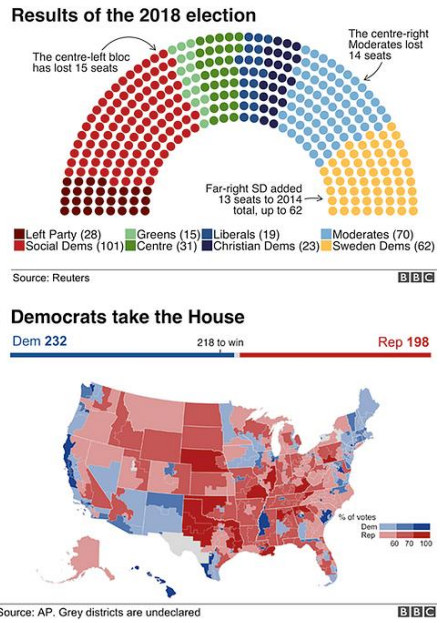
8. Distributions

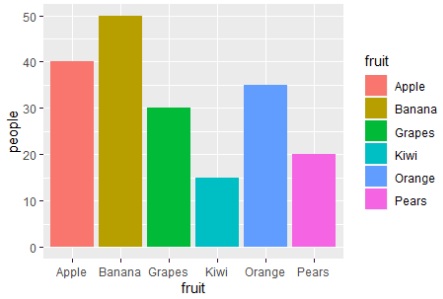
ggplot

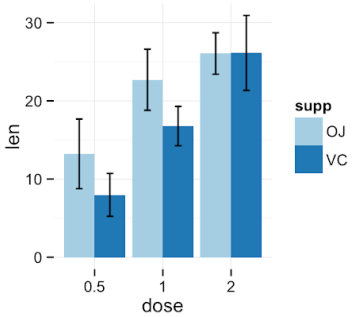
- ggplot is a package that automatically comes with the tidyverse package.
- ggplot is a very powerful data visualisation package, that could merit its own session! (and is beyond the scope of today).

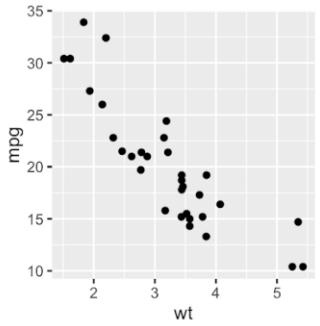


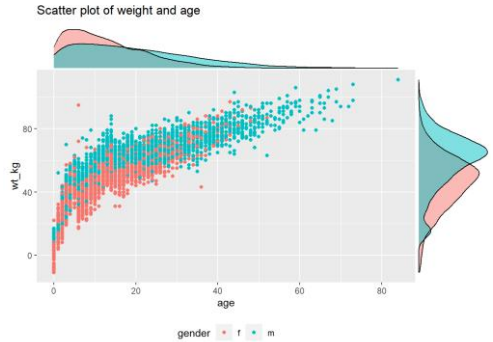












Histogram

```
ggplot(data, aes(x = change, fill = condition)) +
     geom_histogram(colour = "black")
```

- Add a facet_wrap() to overlay the histograms.

```
ggplot(data, aes(x = change, fill = condition)) +
     geom_histogram(colour = "black") +
     facet_wrap(~ condition)
```

Anything in orange needs to match our data set (e.g., the object the data is saved as and the variable names)

Density Plot

```
ggplot(data, aes(x = change, fill = condition)) +
     geom_density(alpha = .5)
```

- Add a facet_wrap() to overlay the plots.

```
ggplot(data, aes(x = change, fill = condition)) +
     geom_density(alpha = .5) +
     facet_wrap(~ condition)
```

Anything in orange needs to match our data set (e.g., the object the data is saved as and the variable names)

Boxplot

```
ggplot(data, aes(x = condition, y = change)) +
     geom_boxplot(width = .4)
```

- Tidy it up by adding theme_classic()

```
ggplot(data, aes(x = condition, y = change)) +
     geom_boxplot(width = .4) +
     theme_classic()
```

Anything in orange needs to match our data set (e.g., the object the data is saved as and the variable names)

9. Wide-form to Long-form Data

Long Data

- The code used today requires long form data.
- Each row should be a **single** observation.

Wide Format

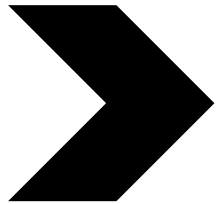
Team	Goals	Shots	Passes
A	7	28	221
B	11	34	234
C	5	19	567
D	12	47	324

Long Format

Team	Variable	Value
A	Goals	7
A	Shots	28
A	Passes	221
B	Goals	11
B	Shots	34
B	Passes	234
C	Goals	5
C	Shots	19
C	Passes	567
D	Goals	12
D	Shots	47
D	Passes	324

Wide-form to Long-form

id	pre	post
1	35	33
2	42	44
3	26	27
4	32	31
5	17	19
6	43	41
7	33	36
8	22	15
9	32	36
10	18	15



id	time_point	sleep_score
1	pre	35
1	post	33
2	pre	42
2	post	44
3	pre	26
3	post	27
4	pre	32
4	post	31
5	pre	17
5	post	19

Pivot Longer

- Repeated measures data tends to be in wide form, rather than long.

- Download the “intervention.csv” data file and convert it to long form.

```
Wide_data <- read_csv(“intervention.csv”)

view(wide_data)
```

- View the data and notice it is not in long format.

Pivot Longer

```
names(wide_data)

long_data <- wide_data %>%
    pivot_longer(cols = c(pre, post),
    names_to = "time_point",
    values_to = "sleep_score")
```

1. Start with your wide-format dataset called wide_data and assign the reshaped version to a new object called long_data.
2. Select the columns that contain repeated measures: pre and post sleep scores to be gathered into a single column.
3. The names of the original columns (pre, post) will become values in a new column called "time_point".
4. The numeric values from pre and post columns will go into a new column called "sleep_score".

Finally…

- Use the long_data to calculate:
  - The mean age in the sample.
  - The standard deviation for age.
  - The mean sleep score.
  - The standard deviation of sleep score.

- Re-use and amend the same code from earlier, but amend the variable names:
  - *Hint:* use names(long_data) and summary(long_data) to check the variable names.

Ongoing Support

- R Training: Correlation and Regression
- R Training: t-Tests and ANOVAs

- **Psychology Final Year Projects: 25/26**
- Staff support package including one-to-one support.
  - Separately for staff and students.
- Access to PS2010 (via guest access).
- E-mail me ☺ Luke.Kendrick@rhul.ac.uk

- Re-learn the basics here: luke-kendrick.github.io/startr
- PS2010 codebook (for 24-25): luke-kendrick.github.io/ps2010-code-book

Other (External) Resources

- https://learningstatisticswithr.com/book/
- https://cran.r-project.org/doc/manuals/r-release/R-intro.html
- https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf