1.2 — Meet R

ECON 480 • Econometrics • Fall 2022

Dr. Ryan Safner
Associate Professor of Economics

safner@hood.edu
ryansafner/metricsF22
metricsF22.classes.ryansafner.com

Data Science

- **You go into data analysis with the tools you know, not the tools you need**
- The next 2-3 weeks are all about giving you the tools you need
  - Admittedly, a bit before you know what you need them *for*
- We will extend them as we learn specific models

R

- **Free** and **open source**
- A very large community
  - Written by statisticians for statistics
  - Most packages are written for R first
- Can handle virtually any data format
- Makes replication easy
- Can integrate into documents (with R `markdown`)
- R is a *language* so it can do *everything*
  - A good stepping stone to learning other languages like *Python*

Excel (or Stata) Can't Do This

Code | Output

```
ggplot(data = gapminder,
       aes(x = gdpPercap,
           y = lifeExp,
           color = continent))+
  geom_point(alpha=0.3)+
  geom_smooth(method = "lm")+
    scale_x_log10(breaks=c(1000,10000, 100000),
                  label=scales::dollar)+
    labs(x = "GDP/Capita",
         y = "Life Expectancy (Years)")+
  facet_wrap(~continent)+
```

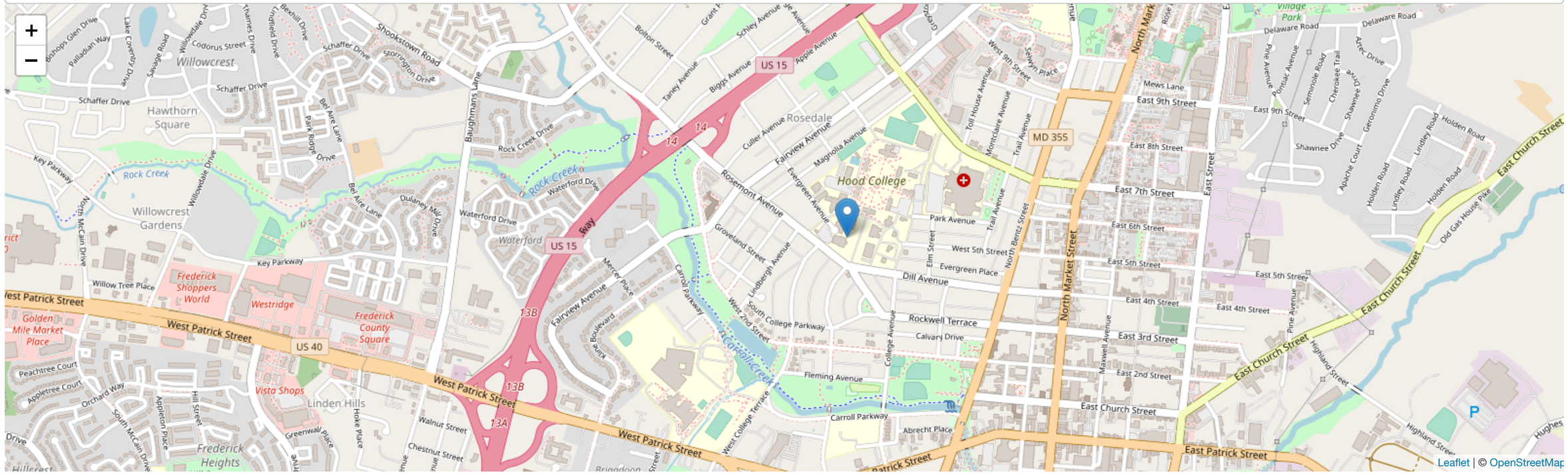
Or This

Input | Output

The average GDP per capita is `r dollar(mean(gapminder$gdpPercap))` with a standard deviation of `r dollar(sd(gapminder$gdpPercap))`.

Or This

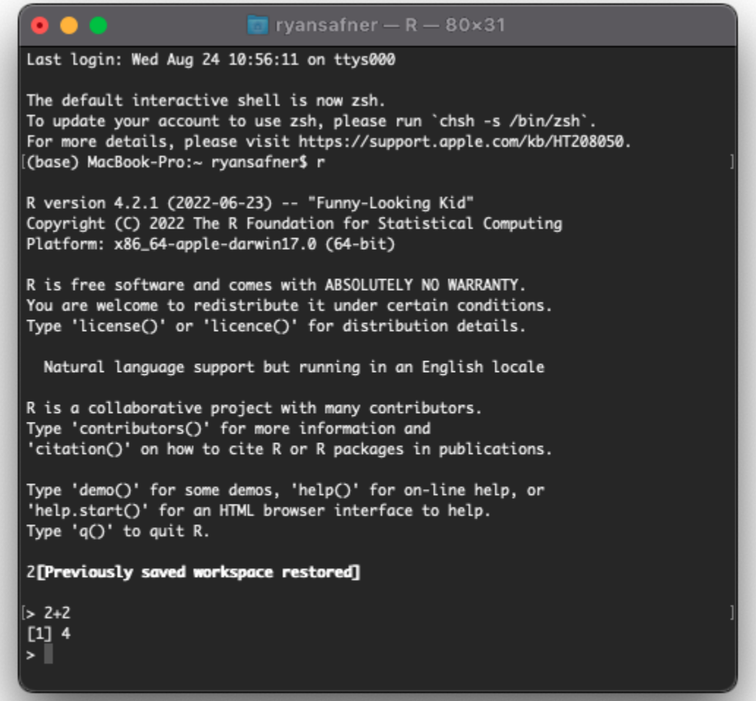
```
1  library(leaflet)
2  leaflet() %>%
3    addTiles() %>%
4    addMarkers(lng = -77.420, lat = 39.421,
5               popup = "Rosenstock Hall, Hood College")
```



Meet R and R Studio

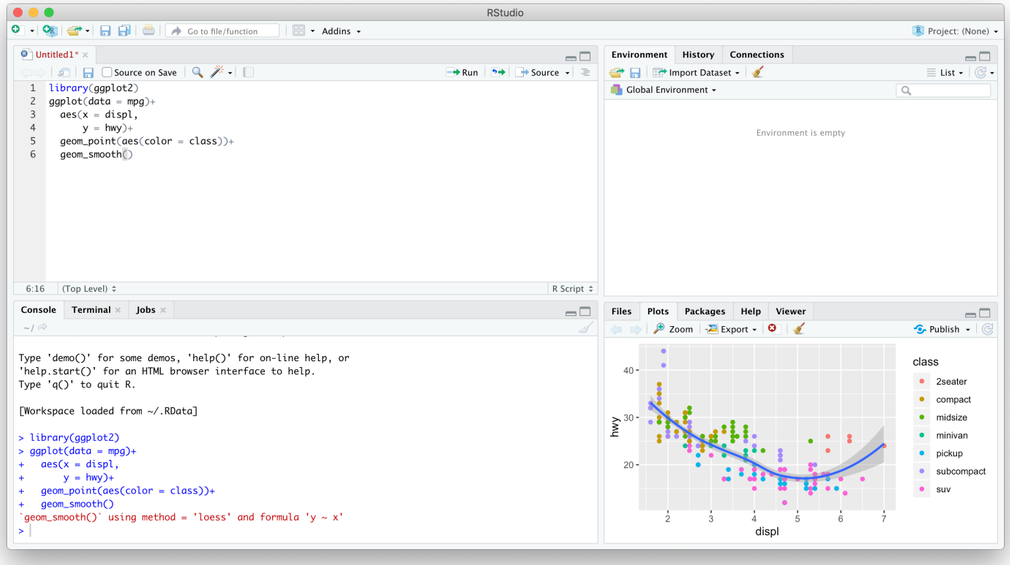
R and R Studio

- **R** is the programming language that executes commands
- Could run this from your computer's shell
  - On Windows: **Command prompt**
  - On Mac/Linux: **Terminal**



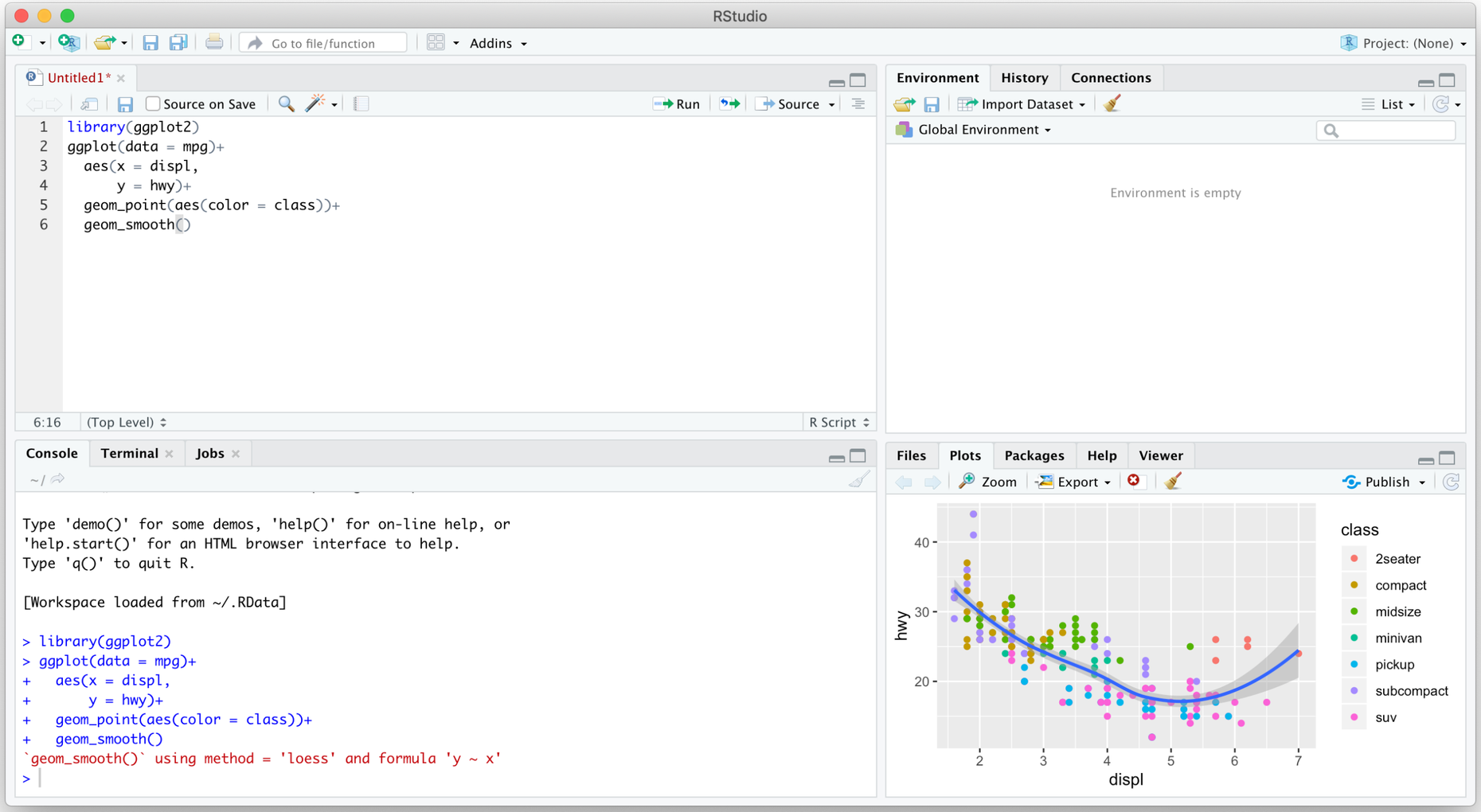
R and R Studio

- **R Studio**[1] is an **integrated development environment** (IDE) that makes your coding life a lot easier
  - Write code in scripts
  - Execute individual commands & scripts
  - Auto-complete, highlight syntax
  - View data, objects, and plots
  - Get help and documentation on commands and functions
  - Integrate code into documents with `Quarto`



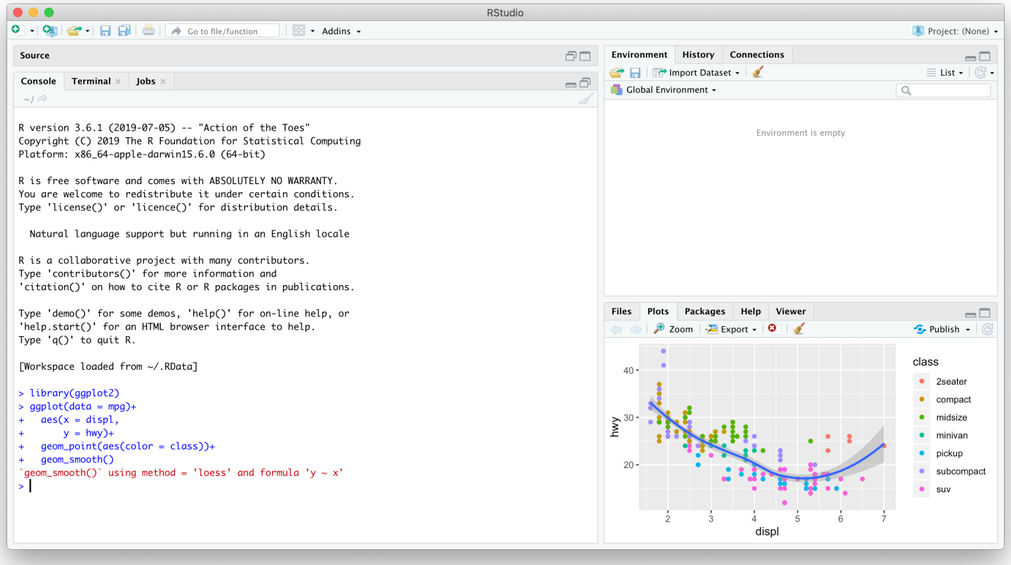
1. The company R Studio recently announced they will be rebranding later this fall as **Posit**

R Studio — Four Panes



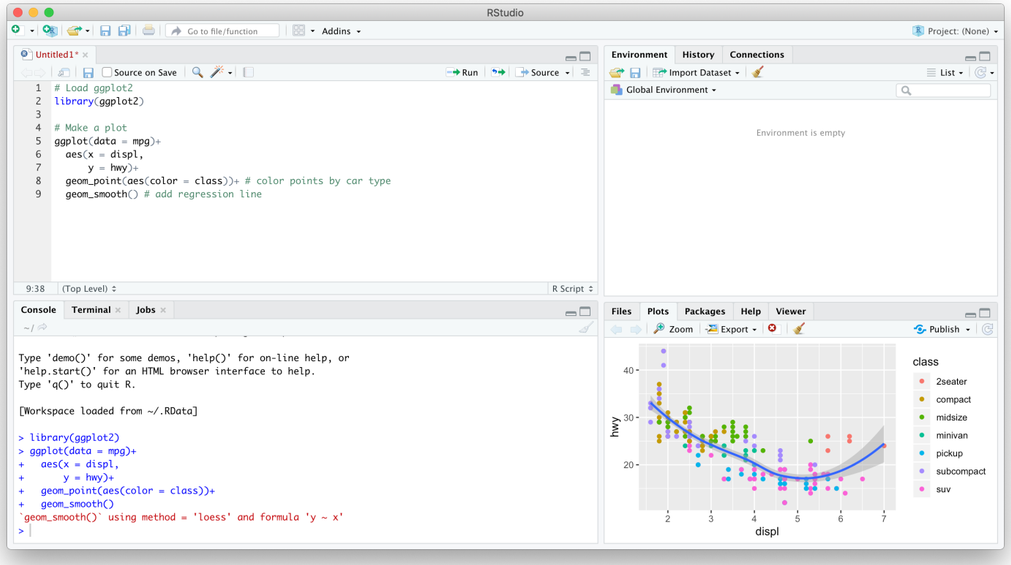
Ways to Use R Studio: Using the Console

- Type individual commands into the console pane (bottom left)
- Great for testing individual commands to see what happens
- Not saved! Not reproducible! Not recommended!

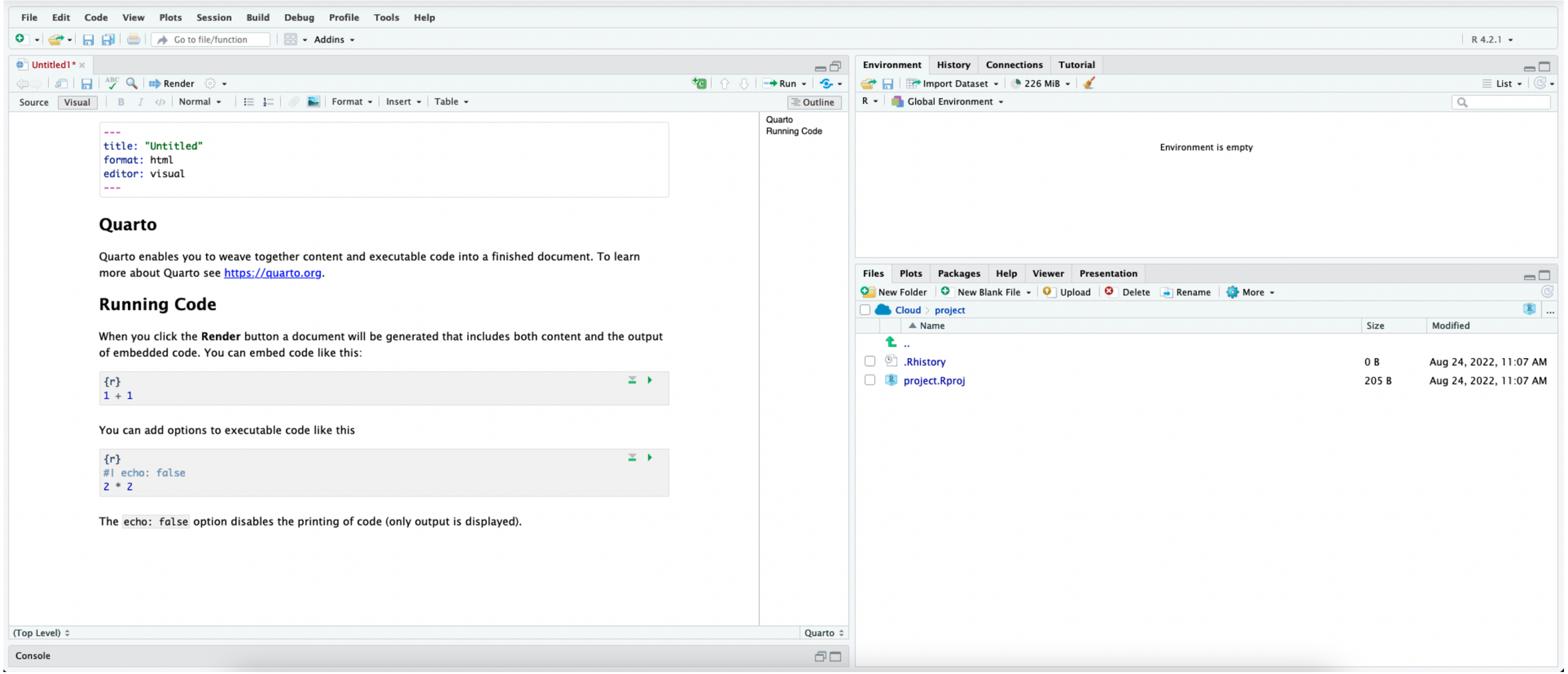


Ways to Use R Studio: Writing a .R Script

- Source pane is a text-editor
- Make .R files: all input commands in a single script
- Comment with #
- Can run any or all of script at once
- Can save, reproduce, and send to others!



Ways to Use R Studio: Quarto Documents



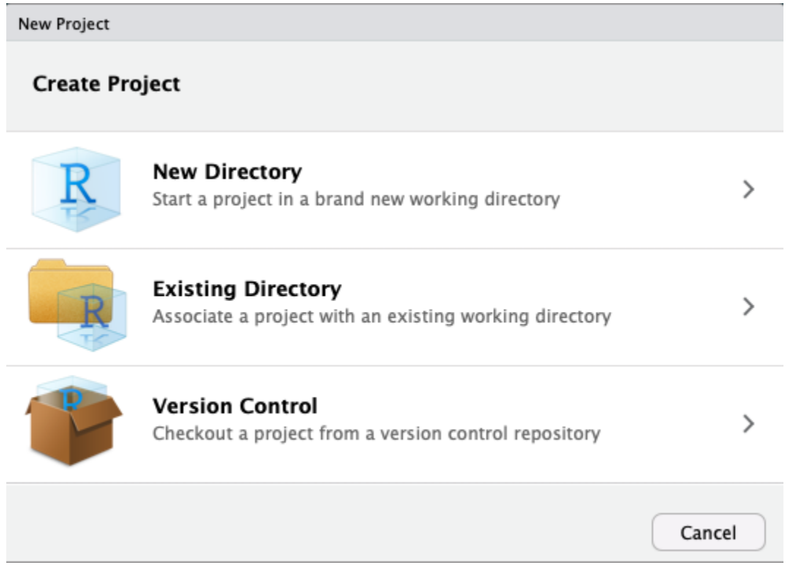
Getting to Know Your Computer

- R assumes a default (often inconvenient) **"working directory"** on your computer
  - The first place it looks to `open` or `save` files
- Find out where R this is with `getwd()`
- Change it with `setwd(path/to/folder)`[1]

1. Note the path is OS-specific. For Windows it might be `C:/Documents/`. For Mac it is often your username folder.

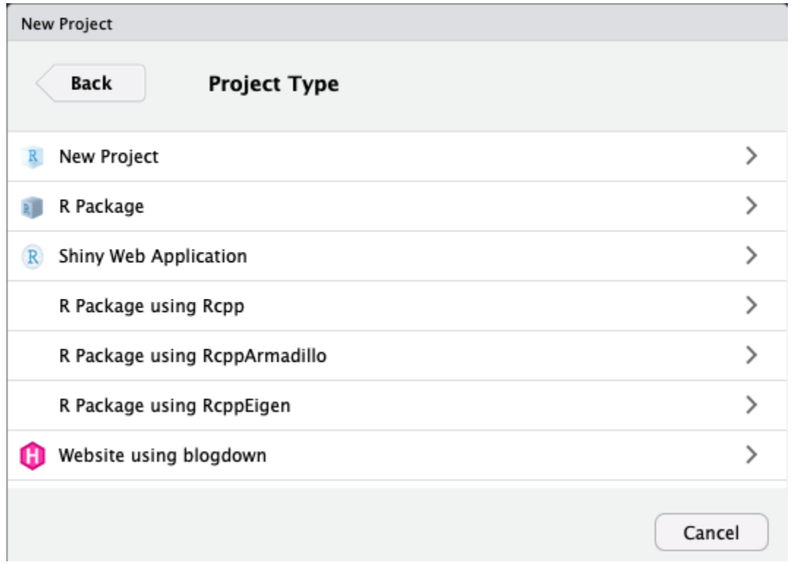
Avoid this Hassle with R Projects

- A `R Project` is a way of systematically organizing your R history, working directory, and related files in a single, self-contained directory
- Can easily be sent to others who can reproduce your work easily
- Connects well with version control software like GitHub
- Can open multiple projects in multiple windows



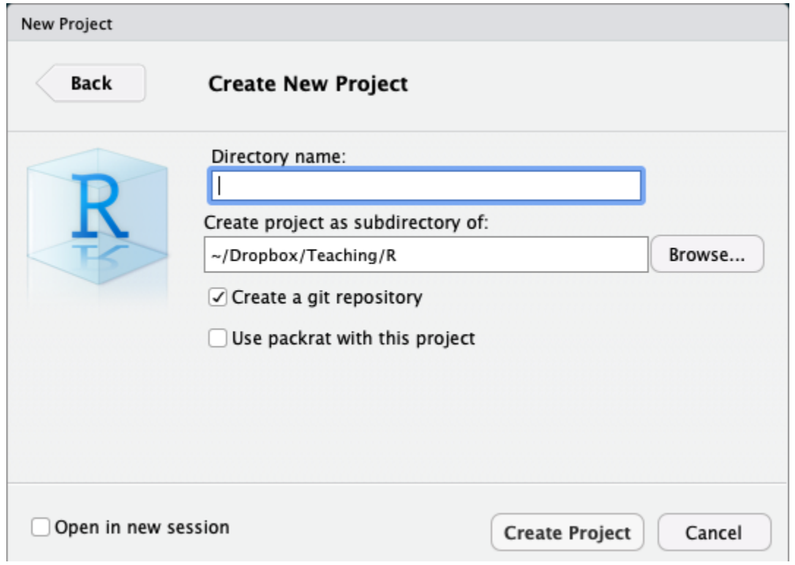
Avoid this Hassle with R Projects

- In almost all cases, you simply want a `New Project`
- For more advanced uses, your project can be an `R Package` or a `Shiny Web Application`
- If you have other packages that create templates installed (as I do, in the previous image), they will also show up as options



Avoid this Hassle with R Projects

- Enter a name for the project in the top field
  - Also creates a folder on your computer with the name you enter into the field
- Choose the location of the folder on your computer
- Depending on if you have other packages or utilities installed (such as `git`, see below!), there may be additional options, do not check them unless you know what you are doing
- Bottom left checkbox allows you to open a new instance (window) of `R` just for this project (and keep existing windows open)

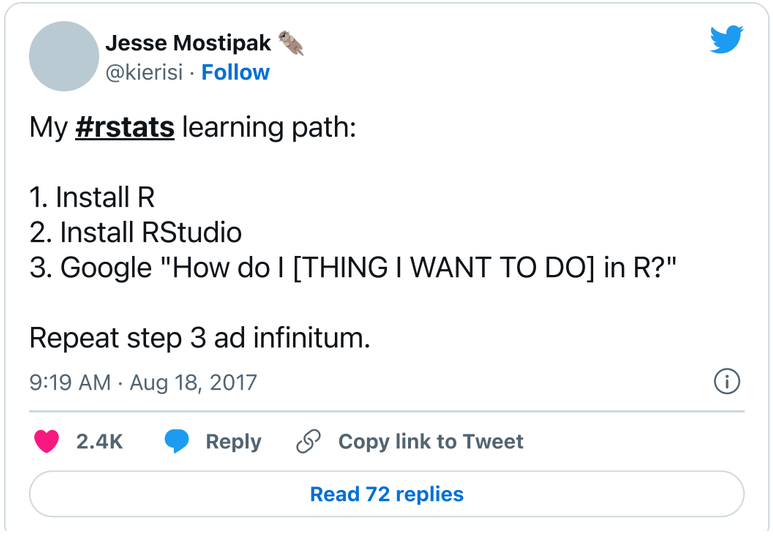


An Intro to Coding

Learning...

- **You don't "*learn R*", you learn *how to do things in R***
- In order to do learn this, you need to learn *how to search for what you want to do*

Learning...

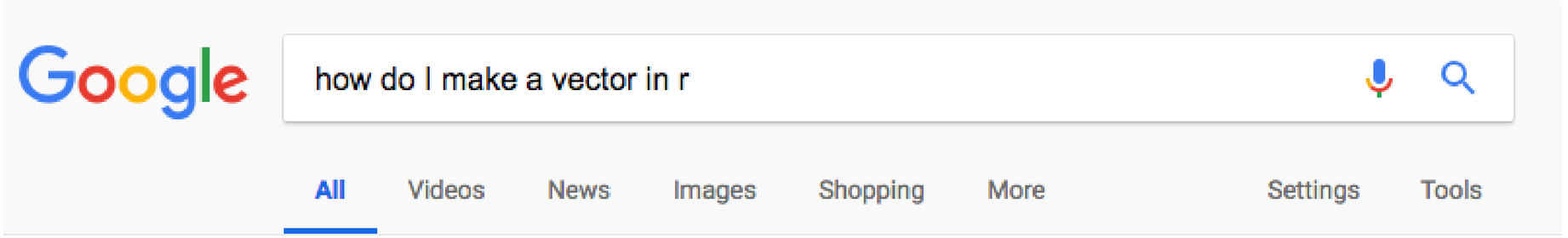

Jesse Mostipak
@kierisi · Follow
My #rstats learning path:
1. Install R
2. Install RStudio
3. Google "How do I [THING I WANT TO DO] in R?"
Repeat step 3 ad infinitum.
9:19 AM · Aug 18, 2017
2.4K
Reply
Copy link to Tweet
Read 72 replies


Katie Mack
@AstroKatie · Follow
A surprisingly large part of having expertise in a topic is not so much knowing everything about it but learning the language and sources well enough to be extremely efficient in google searches.
11:34 AM · Dec 8, 2018
15.1K
Reply
Copy link to Tweet
Read 185 replies

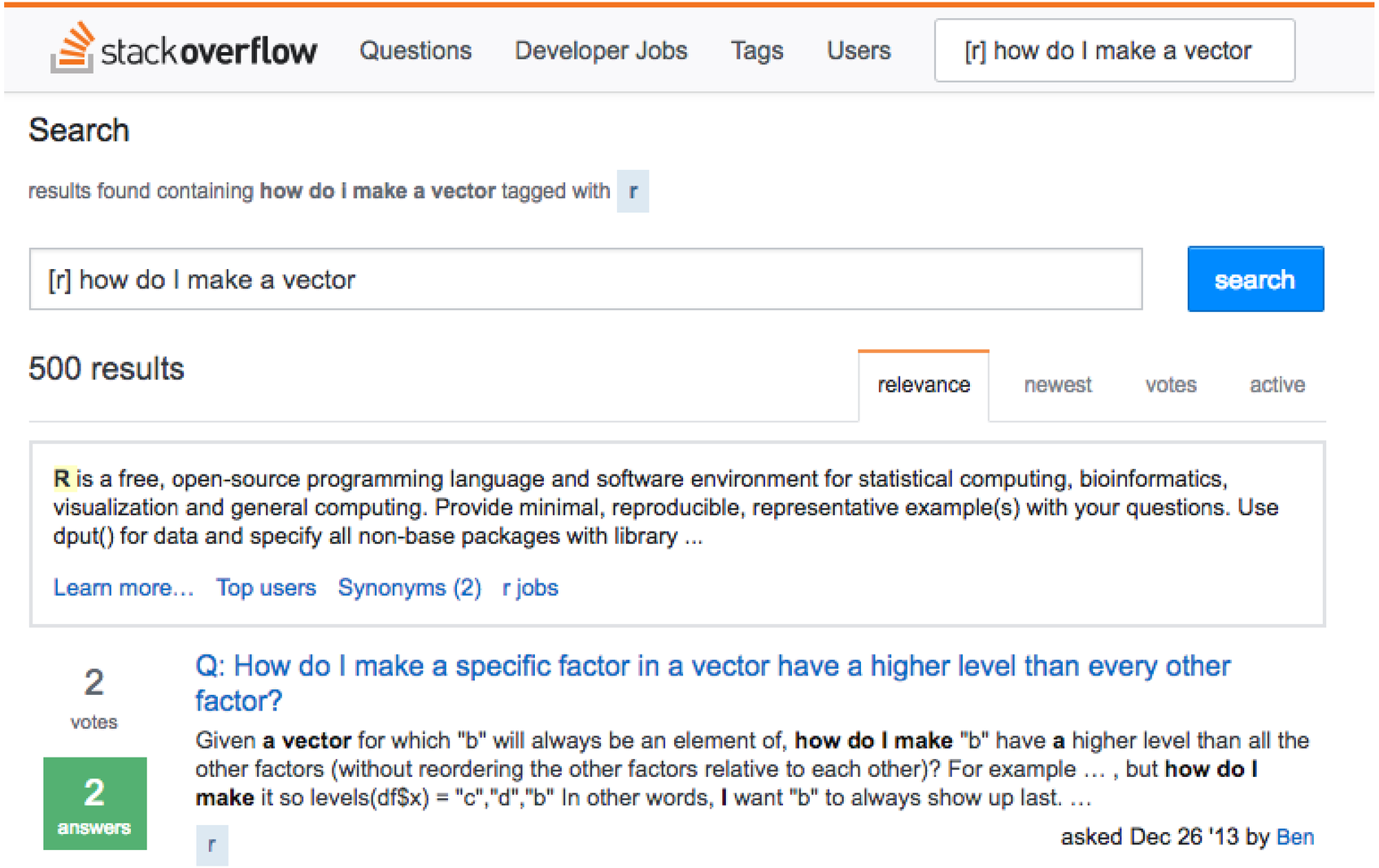
...and Sucking


Dude, sucking at something is the first step
towards being sort of good at something

Say Hello To My Little Friend



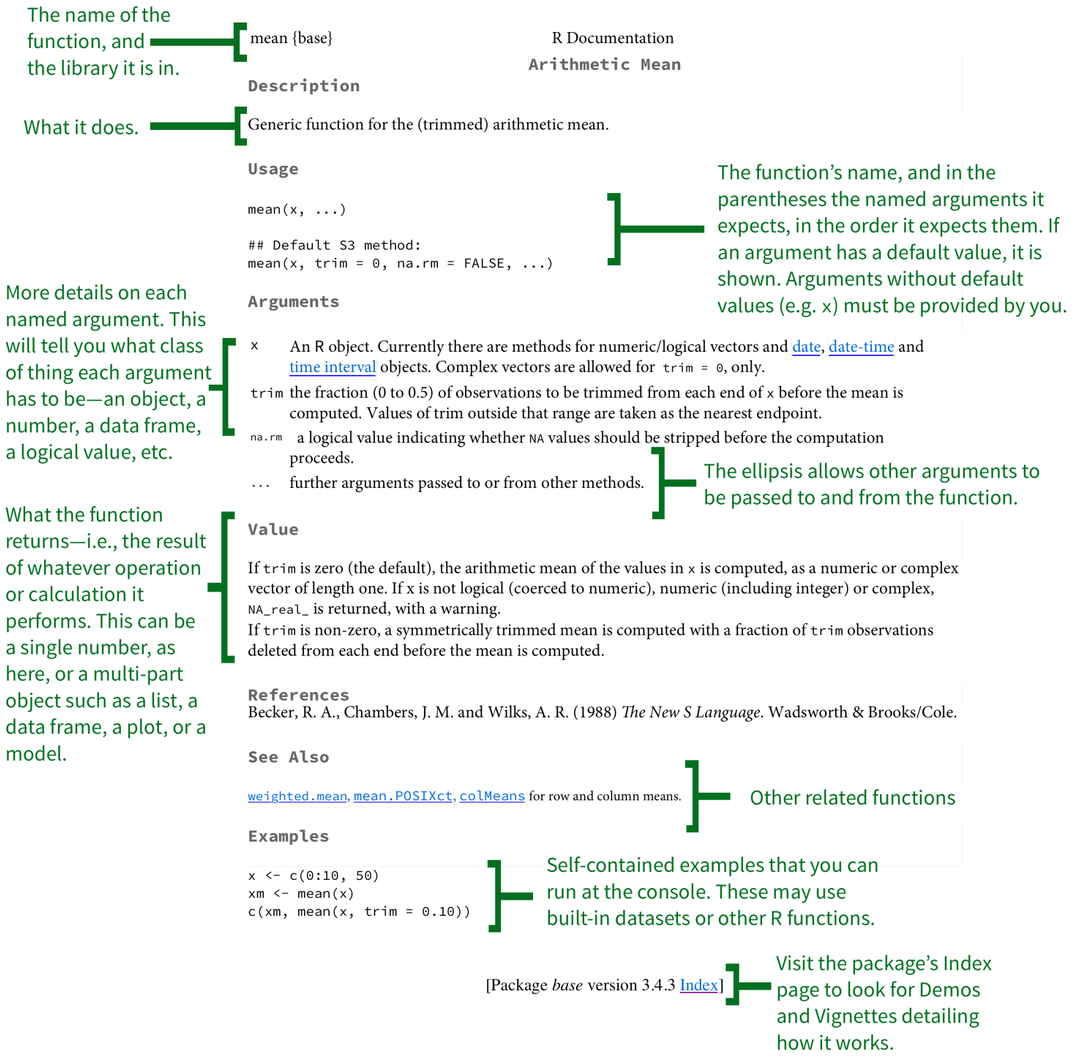
Say Hello to My Better Friend



R Is Helpful Too!

- Type `help(function_name)` or `?(function_name)` to get documentation on a function

```
1 help(mean)
2
3 ?mean() # does the same thing
```



From Kieran Healy, Data Visualization.

I've Failed More Times Than You

Tips for Writing Code

- Comment, comment, comment!
- The hashtag # starts a comment, R will ignore everything on the rest of that line

- Save often!
  - Write scripts that save the commands that did what you wanted (and comment them!)
  - Better yet, use a version control system like Git (I may cover this later)

Style and Naming

- Once we start writing longer blocks of code, it helps to have a consistent (and human-readable!) style
- I follow this style guide (you are not required to)[1]
- Naming objects and files will become important[.magenta[‡]]
  - DO NOT USE SPACES! You've seen seen webpages intended to be called `my webpage in html` turned into `http://my%20webpage%20in%20html.html`

1 Consider your folders on your computer as well

Simple Commands

- You'll have to get used to the fact that you are coding in commands to execute
- Start with the easiest: simple math operators and calculations:

```
> 2+2
```

```
[1] 4
```

- Note that R will ask for **input** with `>` and give you **output** starting with `[1]`

Simple Commands

- We can start using more fancy commands

```
1 2^3
```

```
[1] 8
```

```
1 sqrt(25)
```

```
[1] 5
```

```
1 log(6)
```

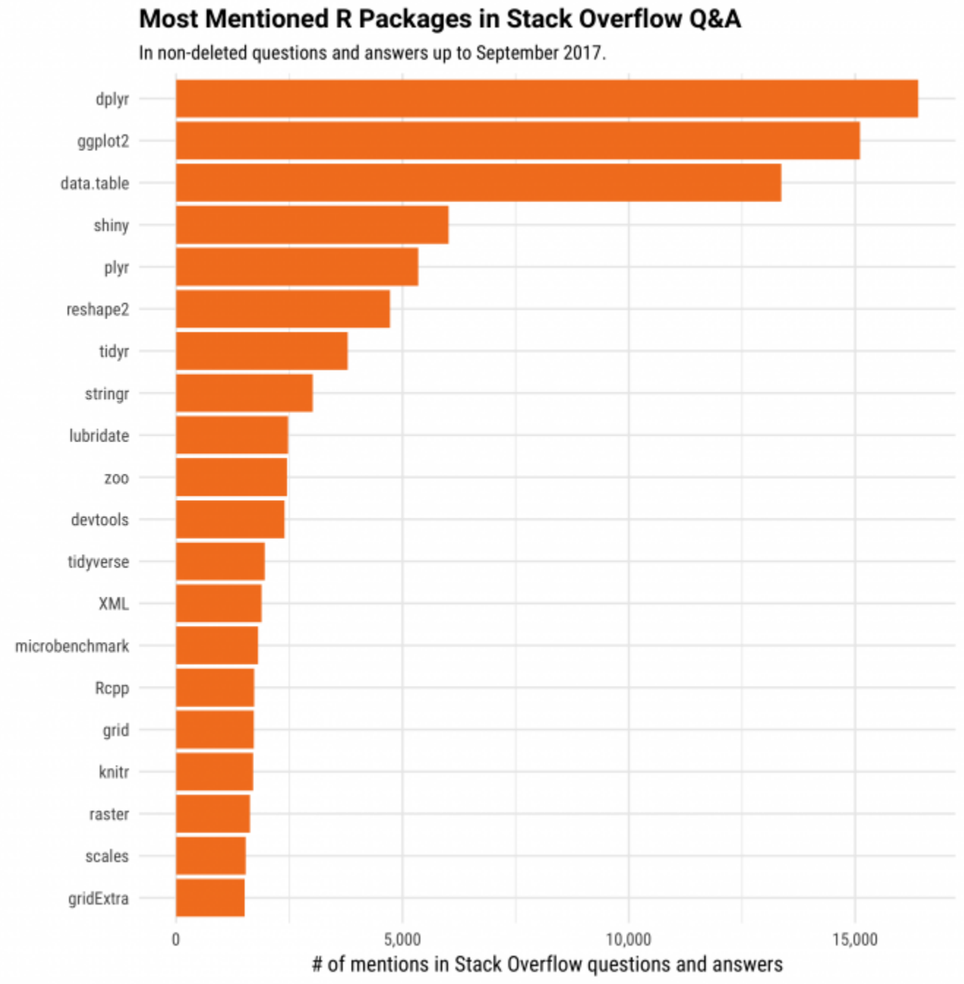
```
[1] 1.791759
```

```
1 pi/2
```

```
[1] 1.570796
```


Packages and Libraries

- Since R is open source, users contribute **packag**
  - Really it's just users writing custom functions saving them for others to use
- Load packages with `library()`
  - e.g. `library("package_name")`
- If you don't have a package, you must first `install.packages()`[1]
  - e.g. `install.packages("package_na`



1. Yes, note the plural, even if it's just for one package

Objects & Functions

- R is an **object-oriented** programming language, meaning you will always be:

1. creating `objects`
   - assign values to an object with `=` (or `<-`)[1]
2. running `functions` on `objects`
   - syntax: `my_function(my_object)`

```
# make an object
my_object = c(1,2,3,4,5)

# look at it
my_object
```

```
[1] 1 2 3 4 5
```

```
# find the sum
sum(my_object)
```

```
[1] 15
```

```
# find the mean
mean(my_object)
```

```
[1] 3
```


1. You can read this as "gets"

More About Functions

- Functions have **arguments**, the input(s)
- Some functions may have multiple arguments
- The argument of a function can be *another* function!

```
# find the sd
sd(my_object)
```

```
[1] 1.581139
```

```
# round everything in my object
round(my_object,2)
```

```
[1] 1 2 3 4 5
```

```
# round the sd to two decimals
round(sd(my_object),2)
```

```
[1] 1.58
```


Types of R Objects

Numeric

- `numeric` objects are just numbers[1]
- Can be mathematically manipulated

```
1 x <- 2
2 y <- 3
3 x+y
```

```
[1] 5
```

```
1 x*y
```

```
[1] 6
```


1 If you want to get technical, R calls these `integer` (for whole numbers) or `double` if there are decimal values.

Character

- `character` objects are “strings” of text **contained inside quote marks**
- Can contain spaces, so long as contained within quote marks

```
1 name <- "Ryan Safner"
2 address <- "Hood College"
3
4 name
```

```
[1] "Ryan Safner"
```

```
1 address
```

```
[1] "Hood College"
```

Logical

- `logical` objects are **boolean/binary** `TRUE` or `FALSE` indicators[1]
- Used a lot to evaluate **conditionals**:
  - `>`, `<`: greater than, less than
  - `>=`, `<=`: greater than or equal to, less than or equal to
  - `==`, `!=`: is equal to, is not equal to[2]
  - `&in&` : is a member of the set of ($\in$)
  - `&`: "AND"
  - `|`: "OR"

```
1 z = 10 # set z equal to 10
2
3 z==10 # test is z equal to 10?
```

```
[1] TRUE
```

```
1 "red"=="blue" # test is red equa
```

```
[1] FALSE
```

```
1 z > 1 & z < 12 # test is z > 1 A
```

```
[1] TRUE
```

```
1 z <= 1 | z==10 # test is z >= 1
```

```
[1] TRUE
```

1. Technically, under the hood, R is actually storing them as `numeric`: `1 = TRUE`, `0 = FALSE`!

Factor

- `factor` objects contain **categorical** data - membership in mutually exclusive groups
- Look like `character` strings, behave more like `logical`s, but with *more than two options*

```
 [1] senior    senior    senior    freshman  freshman  junior    freshman
 [8] senior    freshman  sophomore
Levels: freshman sophomore junior senior
```

- We'll make much more extensive use of them later

```
 [1] senior    senior    senior    freshman  freshman  junior    freshman
 [8] senior    freshman  sophomore
Levels: freshman < sophomore < junior < senior
```

Data Structures

Vectors

- `vector` the simplest type of object, just a collection of elements
  - All elements must be the same data type!
- Make a vector using the **combine/concatenate** `c()` function

```
# create a vector named vec
vec <- c(1,"orange", 83.5, pi)

# look at vec
vec
```

```
[1] "1"                 "orange"
"83.5"
"3.14159265358979"
```

Dataframes I

- `data.frame` or `tibble`: what we'll always be using; think like a **"spreadsheet"**:
  - Each **column** is a vector (variable) of data all the same type
  - Each **row** is an observation (pair of values for all variables)

```
1 library(ggplot2)
2 diamonds
```

```
# A tibble: 53,940 × 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Dataframes II

- Dataframes are really just combinations of (column) vectors
- You can make data frames by combinining named vectors with `data.frame()` or creating each column/vector in each argument

```
# make two vectors
fruits = c("apple","orange","pea
numbers = c(3.3,2.0,6.1,7.5,4.2)

# combine into dataframe
df = data.frame(fruits,numbers)

# do it all in one step (note th
df = data.frame(fruits=c("apple"
                numbers=c(3.3,2.0

```

```
     fruits numbers
1     apple     3.3
2    orange     2.0
3      pear     6.1
4      kiwi     7.5
5 pineapple     4.2
```

Working with Objects

Objects: Storing, Viewing, and Overwriting

- We want to store things in objects to run functions on them later
- Recall, any object is created with the assignment operator `=` or `<-`

```
1  my_vector = c(1,2,3,4,5)
```

- R will not give any output after an assignment

Objects: Storing, Viewing, and Overwriting

- *View* an object (and list its contents) by typing its name

```
my_vector
```

```
[1] 1 2 3 4 5
```

- objects maintain their values until they are assigned different values that will *overwrite* the object

```
my_vector = c(2,7,9,1,5)
my_vector
```

```
[1] 2 7 9 1 5
```


Objects: Checking and Changing Classes

- Check what type of object something is with `class()`

```
1 class("six")
```

```
[1] "character"
```

```
1 class(6)
```

```
[1] "numeric"
```

- Can also use logical tests of `is.()`

```
1 is.numeric("six")
```

```
[1] FALSE
```

```
1 is.character("six")
```

```
[1] TRUE
```

Objects: Checking and Changing Classes

- Convert objects from one class to another with `as.object_class()`
  - Pay attention: you can't convert non-numbers to `numeric`, etc!

```
1 as.character(6)
```

```
[1] "6"
```

```
1 as.numeric("six")
```

```
[1] NA
```


Objects: Different Classes and Coercion I

- Different types of objects have different rules about mixing classes
- Vectors can *not* contain different types of data
  - Different types of data will be "**coerced**" into the lowest-common denominator type of object

```
1 mixed_vector = c(pi, 12, "apple", 6.32)
2 class(mixed_vector)
```

```
[1] "character"
```

```
1 mixed_vector
```

```
[1] "3.14159265358979" "12"                "apple"             "6.32"
```

Objects: Different Classes and Coercion II

- Data frames can have columns with different types of data, so long as all the elements in each column are the same class[1]

```
1 df
```

```
     fruits numbers
1     apple     3.3
2    orange     2.0
3      pear     6.1
4      kiwi     7.5
5 pineapple     4.2
```

```
1 class(df$fruits)
```

```
[1] "character"
```

```
1 class(df$numbers)
```

```
[1] "numeric"
```


1. Remember each column in a data frame is a vector!

More on Dataframes I

- Learn more about a data frame with the `str()` command to view its structure

```
1 class(df)
```

```
[1] "data.frame"
```

```
1 str(df)
```

```
'data.frame':   5 obs. of  2 variables:
 $ fruits : chr  "apple" "orange" "pear" "kiwi" ...
 $ numbers: num  3.3 2 6.1 7.5 4.2
```

More on Dataframes II

- Take a look at the first 5 (or `n`) rows with `head()`

```
1 head(df)
```

```
     fruits numbers
1     apple     3.3
2    orange     2.0
3      pear     6.1
4      kiwi     7.5
5 pineapple     4.2
```

```
1 head(df, n=2)
```

```
  fruits numbers
1  apple     3.3
2 orange     2.0
```

More on Dataframes III

Get summary statistics[1] by column (variable) with `summary()`

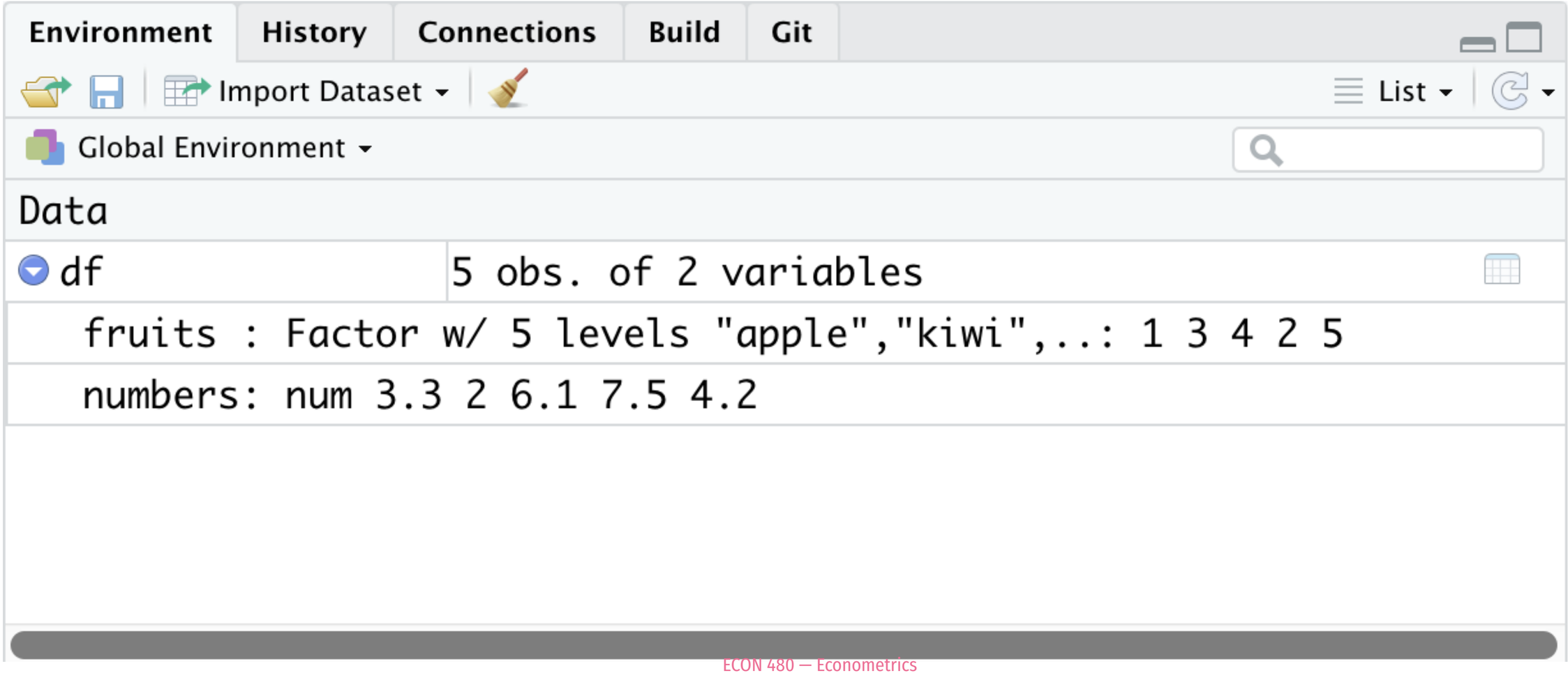
```
1 summary(df)
```

```
   fruits              numbers
Length:5           Min.   :2.00
Class :character   1st Qu.:3.30
Mode  :character   Median :4.20
                   Mean   :4.62
                   3rd Qu.:6.10
                   Max.   :7.50
```


1. For `numeric` data only; a frequency table is displayed for `character` or `factor` data.

More on Dataframes IV

- Note, once you save an object, it shows up in the **Environment Pane** in the upper right window
- Click the blue arrow button in front of the object for some more information



More on Dataframes V

- `data.frame` objects can be viewed in their own panel by clicking on the name of the object in the environment pane
- Note you cannot edit anything in this pane, it is for viewing only

Functions Again I

- Functions in R are **vectorized**, meaning running a function on a vector applies it to *each* element

```
my_vector = c(2,4,5,10) # create object called my_vector
my_vector # look at it
```

```
[1]  2  4  5 10
```

```
my_vector+4 # add 4 to all elements of my_vector
```

```
[1]  6  8  9 14
```

```
my_vector^2 # square all elements of my_vector
```

```
[1]   4  16  25 100
```

Functions Again II

- But often we want to run functions on vectors that *aggregate* to a result (e.g. a statistic):

```
1 length(my_vector) # how many ele
```

```
[1] 4
```

```
1 sum(my_vector) # add all element
```

```
[1] 21
```

```
1 max(my_vector) # find largest el
```

```
[1] 10
```

```
1 min(my_vector) # find smallest e
```

```
[1] 2
```

```
1 mean(my_vector) # mean of all el
```

```
[1] 5.25
```

```
1 median(my_vector) # median of al
```

```
[1] 4.5
```

```
1 var(my_vector) # variance of obj
```

```
[1] 11.58333
```

```
1 sd(my_vector) # standard deviati
```

```
[1] 3.40343
```


Some Common Errors

- If you make a coding error (e.g. forget to close a parenthesis), R might show a `+` sign waiting for you to finish the command

```
> 2+(2*3
+
```

- Either finish the command– e.g. add `)`–or hit `Esc` to cancel