

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

```
1 ggplot(data = gapminder,  
2         aes(x = gdpPercap,  
3             y = lifeExp,  
4             color = continent))+  
5   geom_point(alpha=0.3)+  
6   geom_smooth(method = "lm")+  
7   scale_x_log10(breaks=c(1000,10000, 100000),  
8                 label=scales::dollar)+  
9   labs(x = "GDP/Capita",  
10        y = "Life Expectancy (Years)")+  
11  facet_wrap(~continent)+
```



Or This

Input

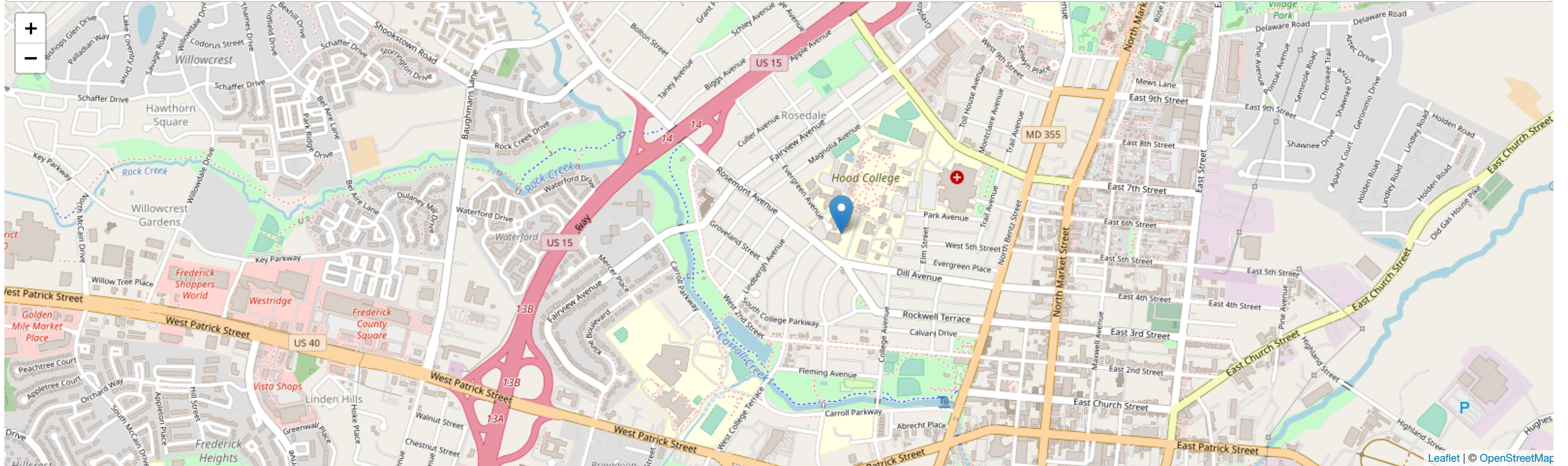
Output

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



Or This

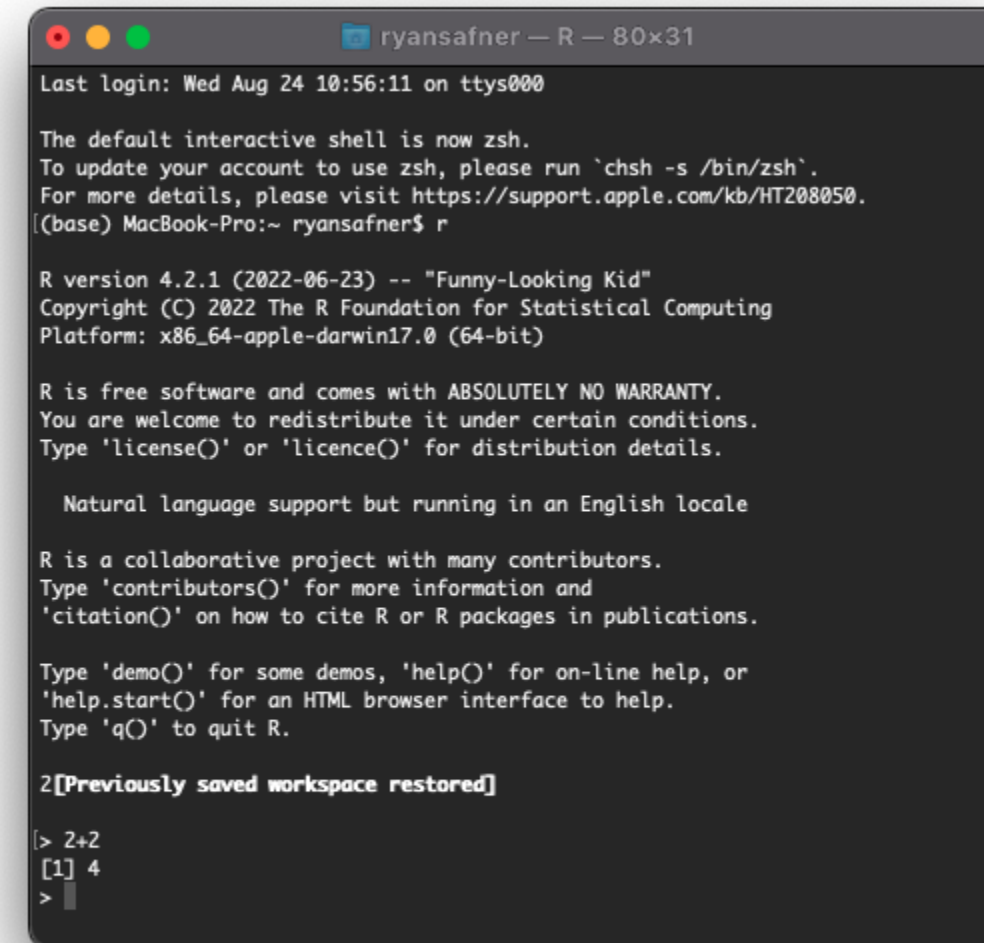
```
1 library(leaflet)
2 leaflet() %>%
3   addTiles() %>%
4   addMarkers(lng = -77.420, lat = 39.421,
5             popup = "Rosenstock Hall, Hood College")
```



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**



```
ryansafner — R — 80x31
Last login: Wed Aug 24 10:56:11 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) MacBook-Pro:~ ryansafner$ r

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (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.

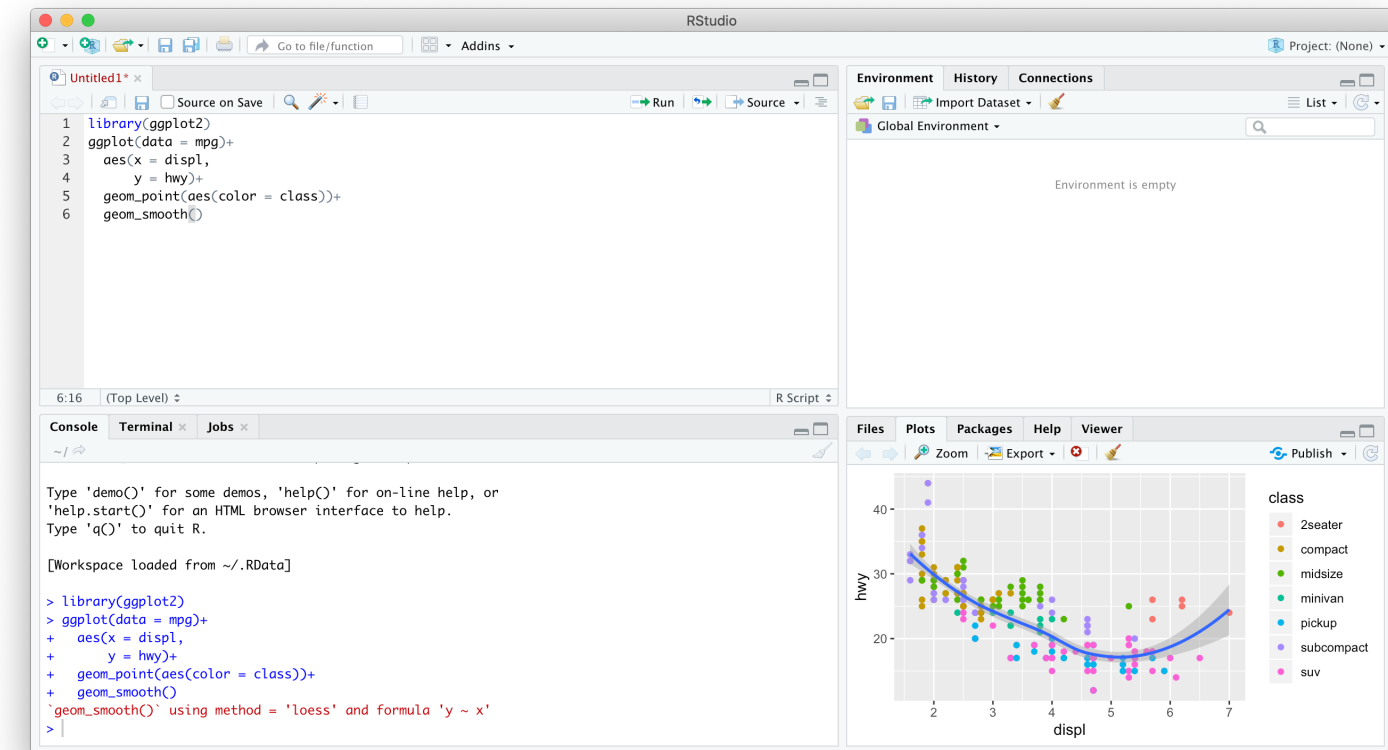
2[Previously saved workspace restored]

> 2+2
[1] 4
>
```

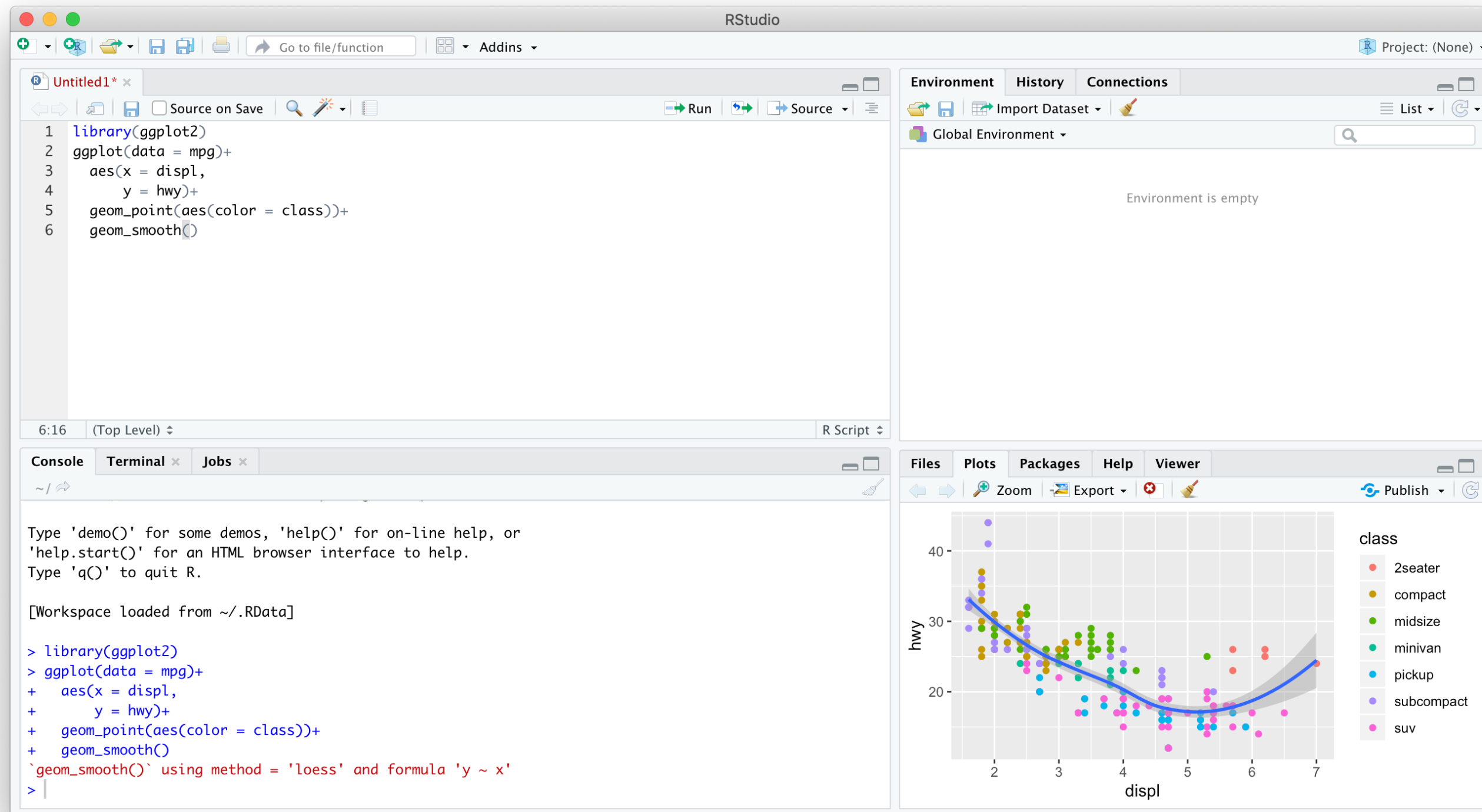


R and R Studio

- **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**

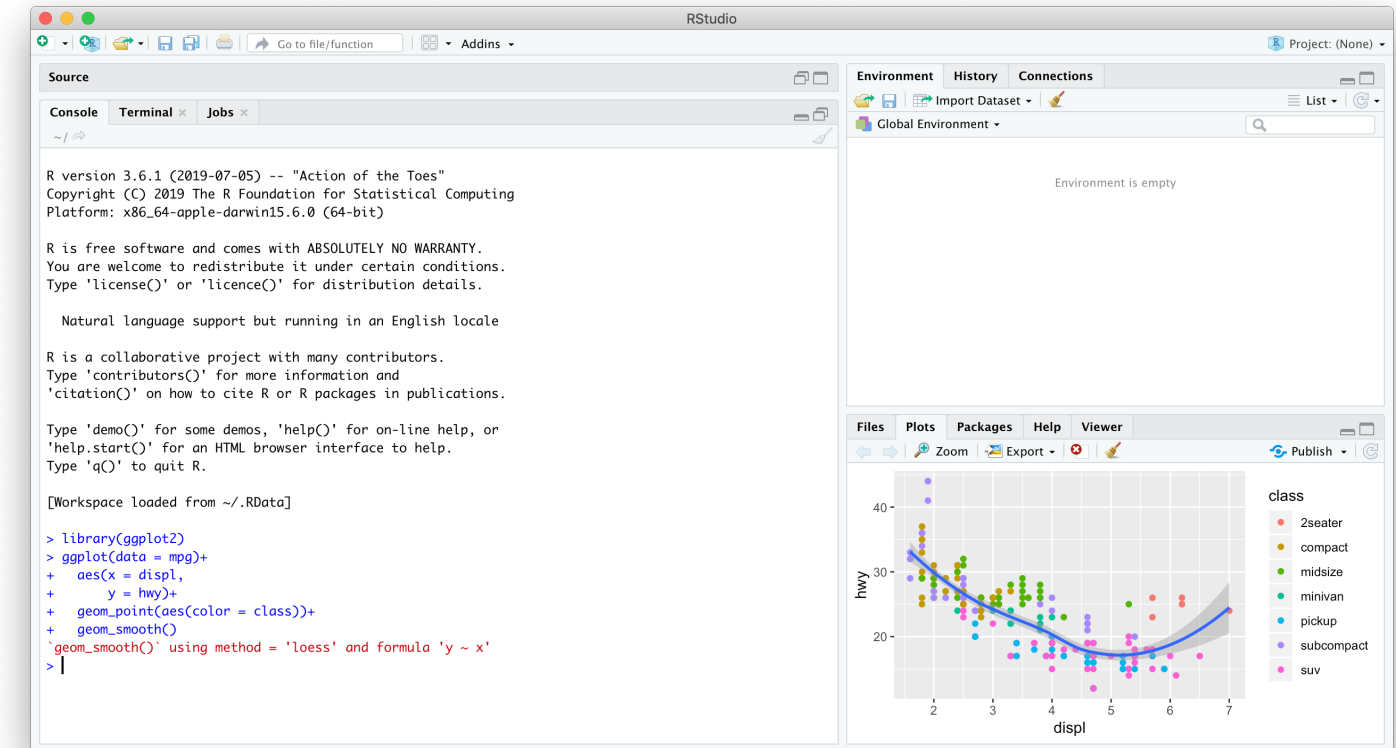


R Studio — Four Panes



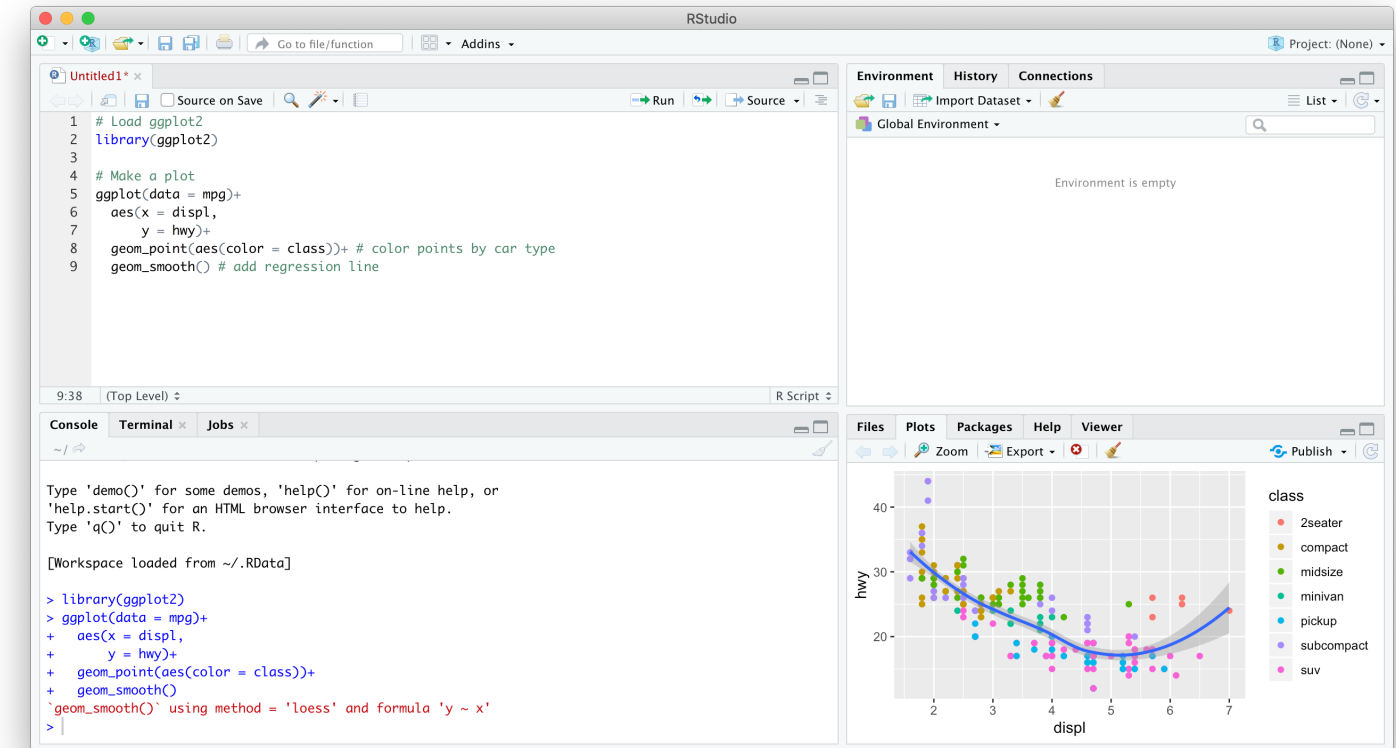
Ways to Use R Studio: Using the Console

- 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

The screenshot shows the R Studio interface with a Quarto document open. The editor window displays the following content:

```
---
title: "Untitled"
format: html
editor: visual
---
```

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
{r}
1 + 1
```

You can add options to executable code like this

```
{r}
#| echo: false
2 * 2
```

The `echo: false` option disables the printing of code (only output is displayed).

The right sidebar shows the **Environment** pane (empty) and the **Files** pane (showing project files):

Name	Size	Modified
..		
.Rhistory	0 B	Aug 24, 2022, 11:07 AM
project.Rproj	205 B	Aug 24, 2022, 11:07 AM



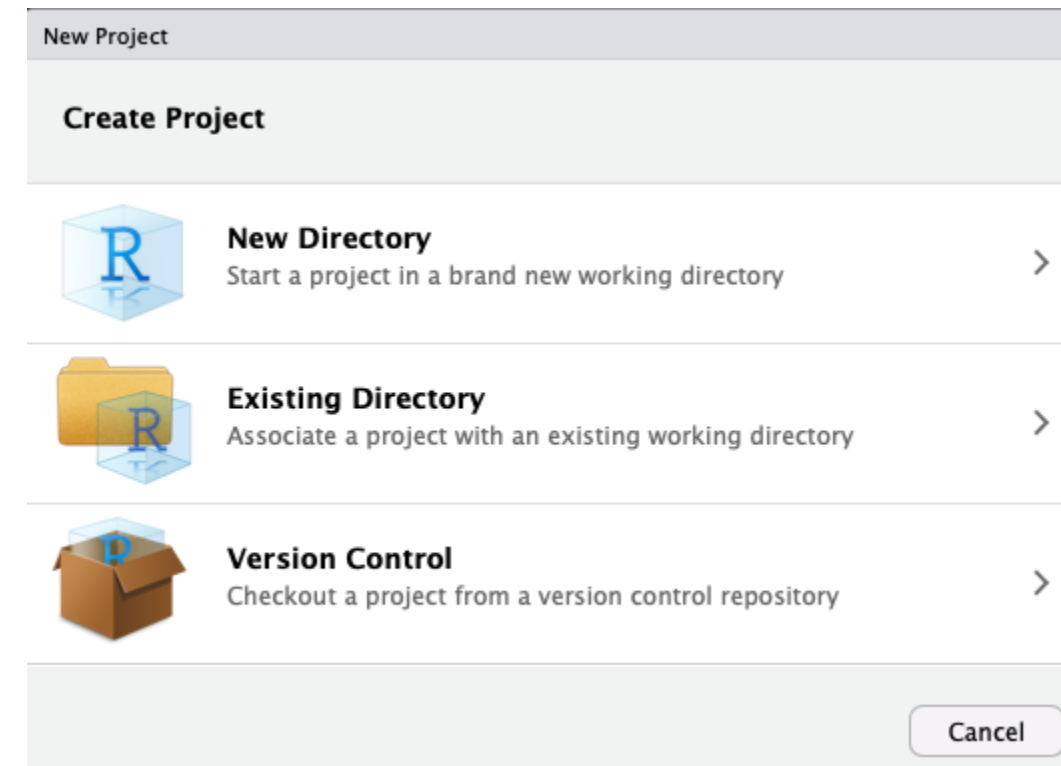
Getting to Know Your Computer

- 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



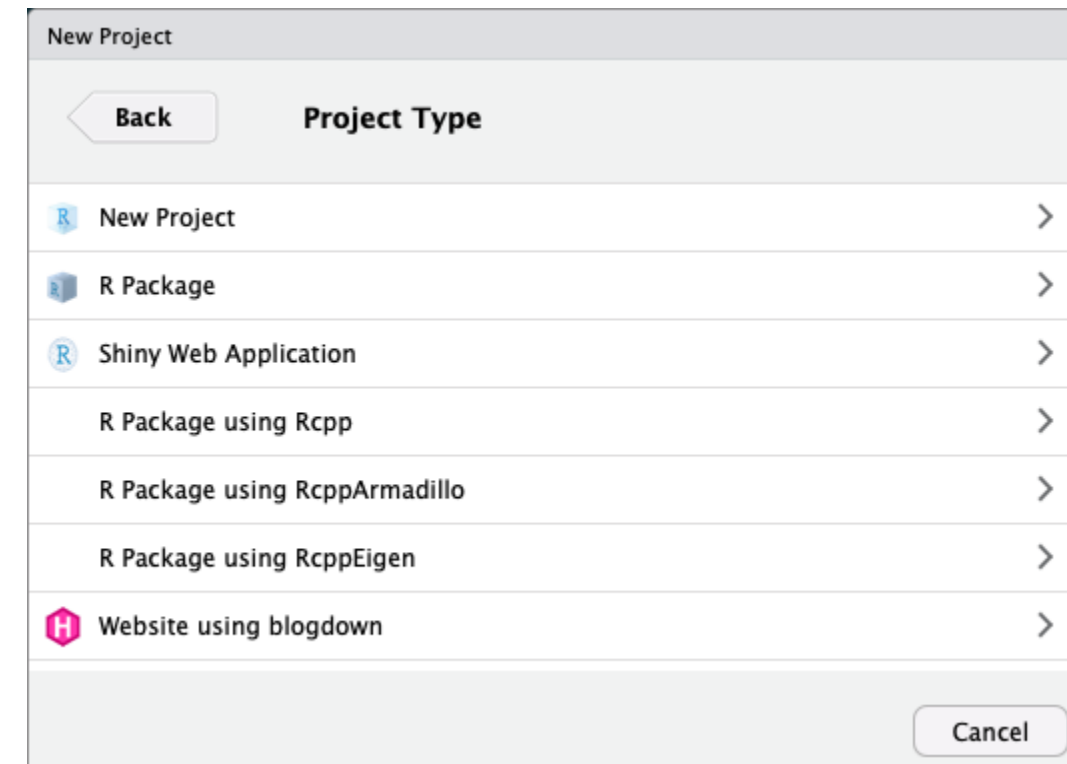
Avoid this Hassle with R Projects

- 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



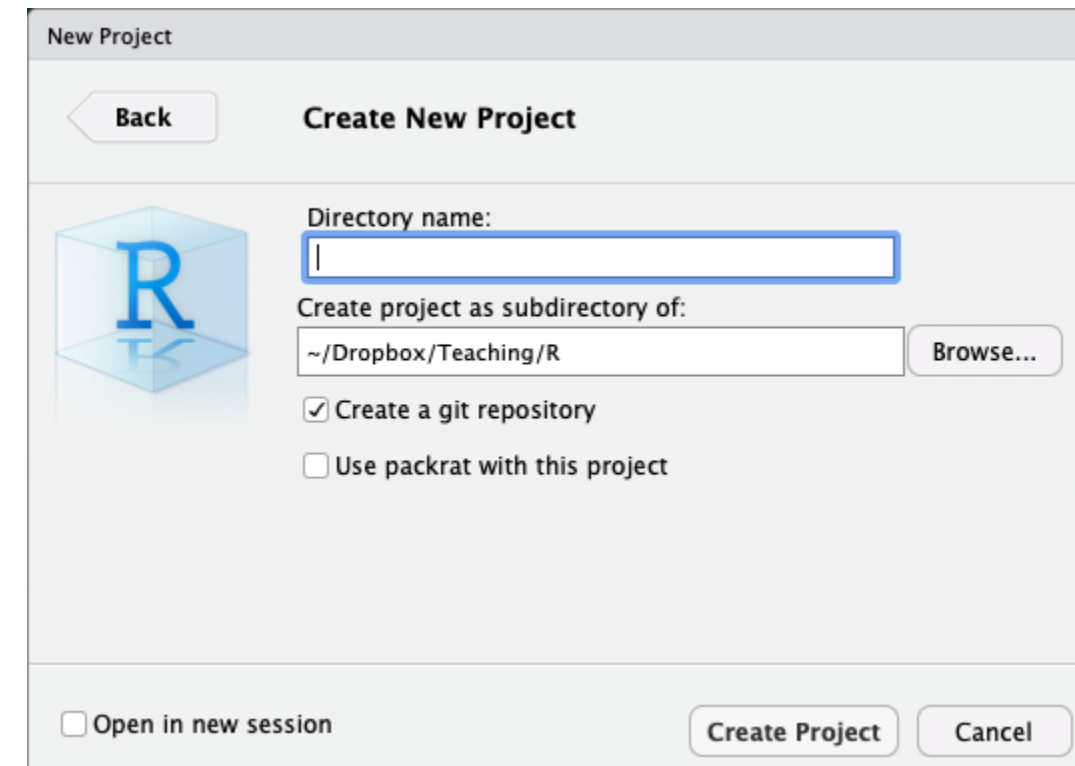
Avoid this Hassle with R Projects

- 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...

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.

— Jesse Mostipak (@kierisi) **August 18, 2017**

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.

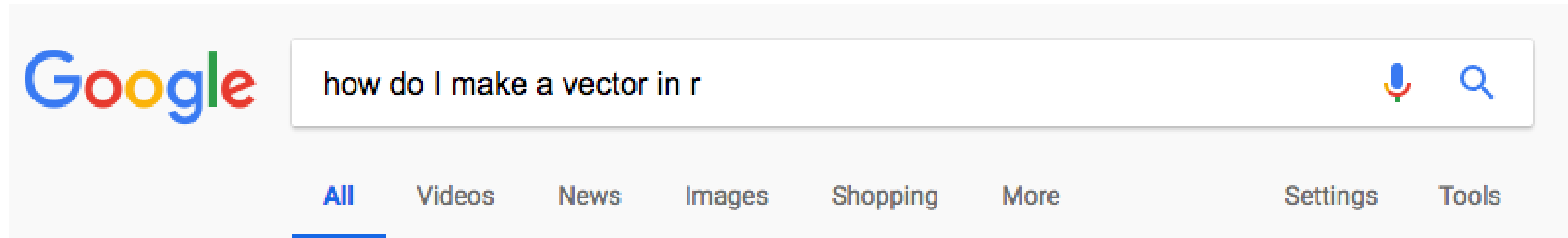
— Katie Mack (@AstroKatie) **December 8, 2018**



...and Sucking



Say Hello To My Little Friend



About 395,000,000 results (0.60 seconds)

R Vector: Create, Modify and Access Vector Elements - DataMentor

<https://www.datamentor.io/r-programming/vector> ▼

In this article, you'll learn about **vector** in R programming. You'll learn to **create** them, access their elements using different methods, and modify them in your program. **Vector** is a basic data structure in R. It contains element of the same type.

Vector | R Tutorial

www.r-tutor.com/r-introduction/vector ▼

An R tutorial on the concept of **vectors** in R. Discuss how to **create vectors** of numeric, logical and character string data types.

2. Basic Data Types — R Tutorial - Cyclismo

<https://www.cyclismo.org/tutorial/R/types.html> ▼

We look at some of the ways that R can store and organize data. This is a ... You can **create** a list (also called a "**vector**") using the **c** command: `> a <- c(1,2,3,4,5) > ...`



Say Hello to My Better Friend

Search

results found containing **how do I make a vector** tagged with **r**

[r] how do I make a vector

search

500 results

- relevance
- newest
- votes
- active

R is a free, open-source programming language and software environment for statistical computing, bioinformatics, visualization and general computing. Provide minimal, reproducible, representative example(s) with your questions. Use `dput()` for data and specify all non-base packages with library ...

[Learn more...](#) [Top users](#) [Synonyms \(2\)](#) [r jobs](#)

2

votes

2

answers

Q: How do I make a specific factor in a vector have a higher level than every other factor?

Given **a vector** for which "b" will always be an element of, **how do I make** "b" have **a** higher level than all the other factors (without reordering the other factors relative to each other)? For example ... , but **how do I make** it so `levels(df$x) = "c","d","b"` In other words, **I** want "b" to always show up last. ...

r

asked Dec 26 '13 by Ben



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
```

The name of the function, and the library it is in.

mean {base}

R Documentation
Arithmetic Mean

What it does.

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

x

An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.

trim

the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

na.rm

a logical value indicating whether NA values should be stripped before the computation proceeds.

...

further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.
If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

[Package base version 3.4.3 [Index](#)]

Visit the package's Index page to look for Demos and Vignettes detailing how it works.

The function's name, and in the parentheses the named arguments it expects, in the order it expects them. If an argument has a default value, it is shown. Arguments without default values (e.g. `x`) must be provided by you.

The ellipsis allows other arguments to be passed to and from the function.

Other related functions

Self-contained examples that you can run at the console. These may use built-in datasets or other R functions.

From Kieran Healy, **Data Visualization**.



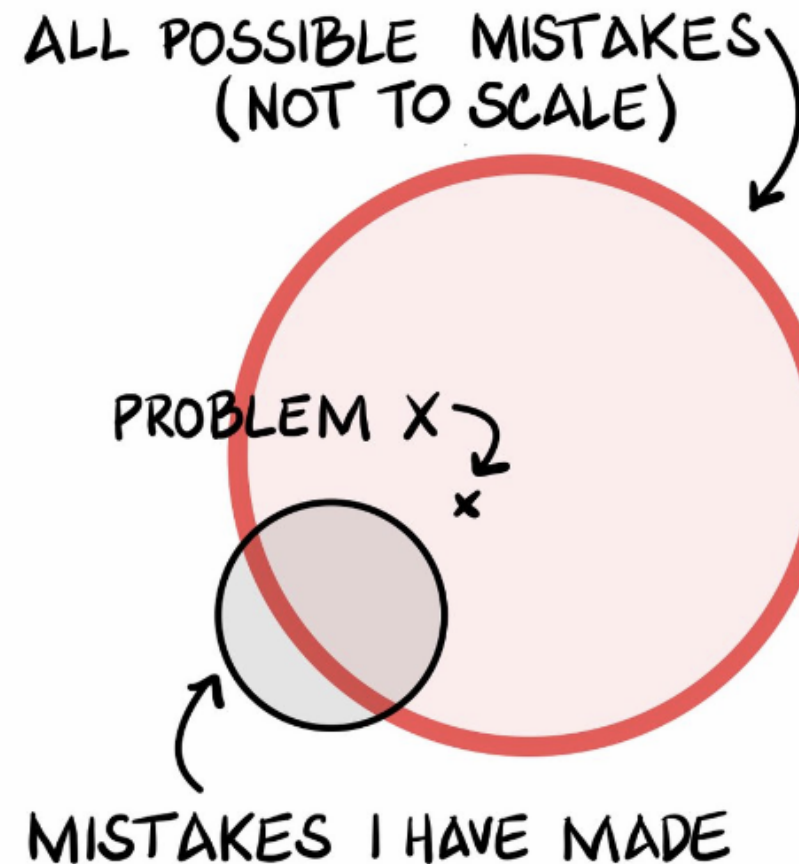


I've Failed More Times Than You

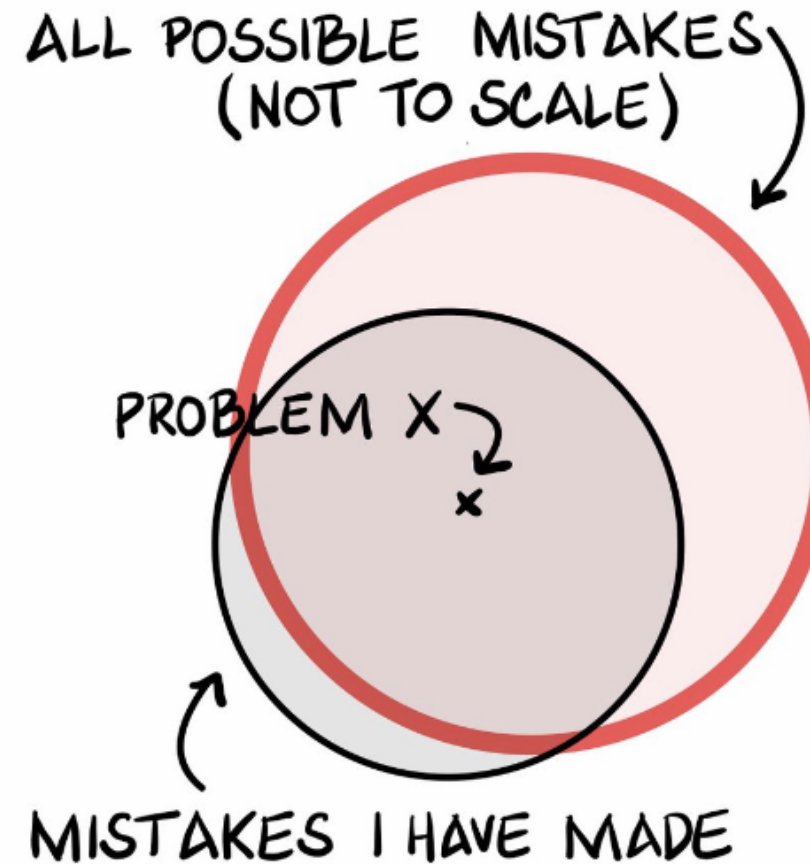
©Victoria Grinberg

WHY YOUR PHD ADVISOR CAN SOLVE YOUR PROBLEM SO "EASILY"

EARLY CAREER



MID CAREER



(YES, A GOOD RESEARCHER WILL MAKE MISTAKES OUTSIDE OF
WHAT HAS BEEN CONSIDERED POSSIBLE PREVIOUSLY)



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
 - DO NOT USE SPACES! You've seen webpages intended to be called `my webpage in html` turned into `http://my%20webpage%20in%20html.html`

```
1 i_use_underscores
2 some.people.use.snake.case
3 othersUseCamelCase
```



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:

```
1 > 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
```

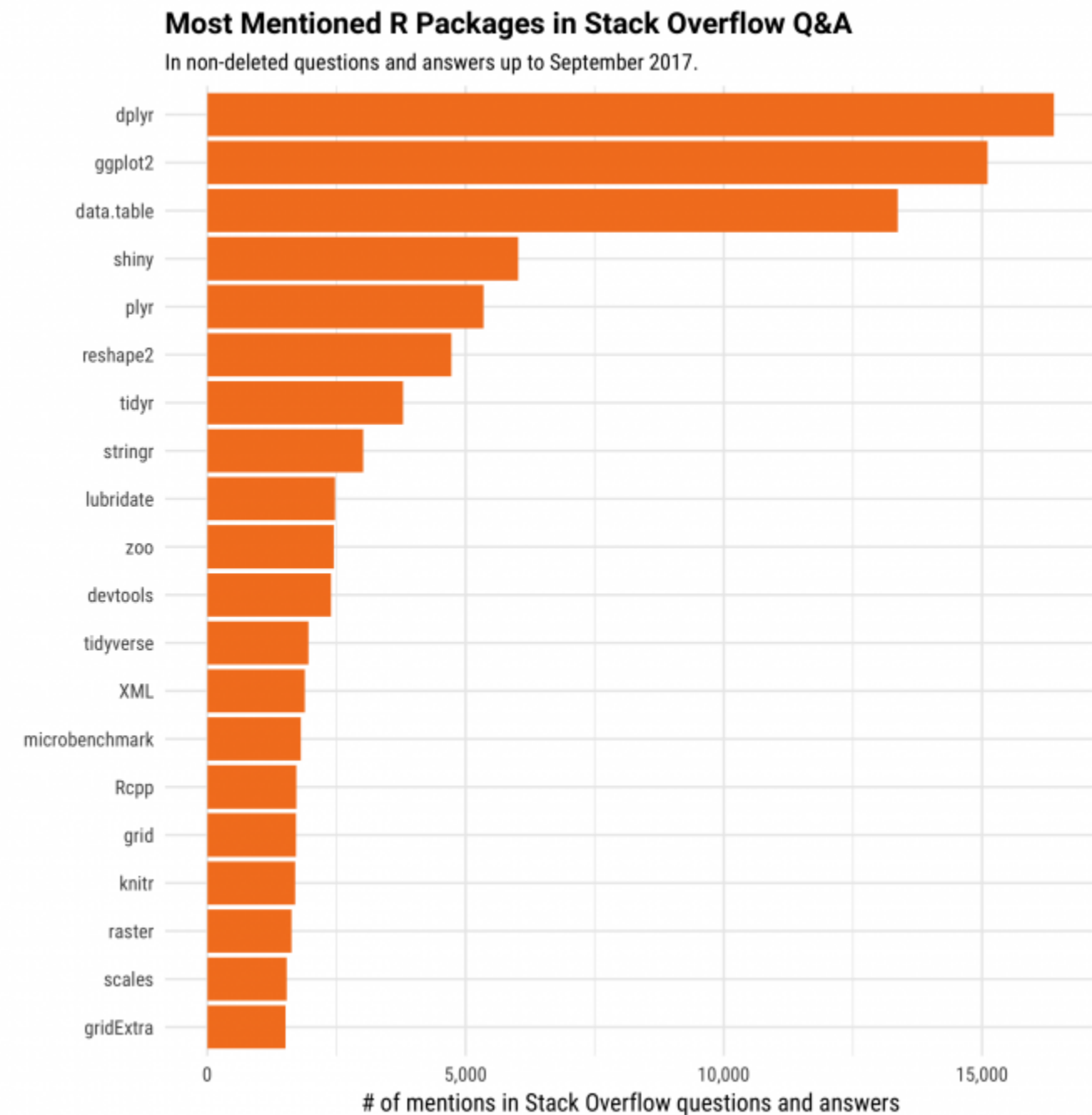
```
1 pi/2
```

```
[1] 1.570796
```



Packages and Libraries

- Since R is open source, users contribute **packages**
 - 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_name")`



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)`

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

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

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

```
[1] 15
```

```
1 # find the mean
2 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!

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

```
[1] 1.581139
```

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

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

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

```
[1] 1.58
```



Types of R Objects

Numeric

- `numeric` objects are just numbers¹
- Can be mathematically manipulated

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

```
[1] 5
```

```
1 x*y
```

```
[1] 6
```



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 indicators¹
- 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 to²
 - `&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 equal to blue?
```

```
[1] FALSE
```

```
1 z > 1 & z < 12 # test is z > 1 AND z < 12?
```

```
[1] TRUE
```

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

```
[1] TRUE
```

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

2. One `=` assigns a value (like `<`). Two `==` evaluates a conditional statement!



Factor

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

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

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

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

```
1 # create a vector named vec
2 vec <- c(1, "orange", 83.5, pi)
3
4 # look at vec
5 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
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47



Dataframes II

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

```
1 # make two vectors
2 fruits = c("apple", "orange", "pear", "kiwi", "pineapple")
3 numbers = c(3.3, 2.0, 6.1, 7.5, 4.2)
4
5 # combine into dataframe
6 df = data.frame(fruits, numbers)
7
8 # do it all in one step (note the order)
9 df = data.frame(fruits=c("apple", "orange", "pear", "kiwi", "pineapple"),
10                numbers=c(3.3, 2.0, 6.1, 7.5, 4.2))
11
```

	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

```
1 my_vector
```

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

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

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

```
2 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 class¹

```
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 statistics¹ by column (variable) with `summary()`

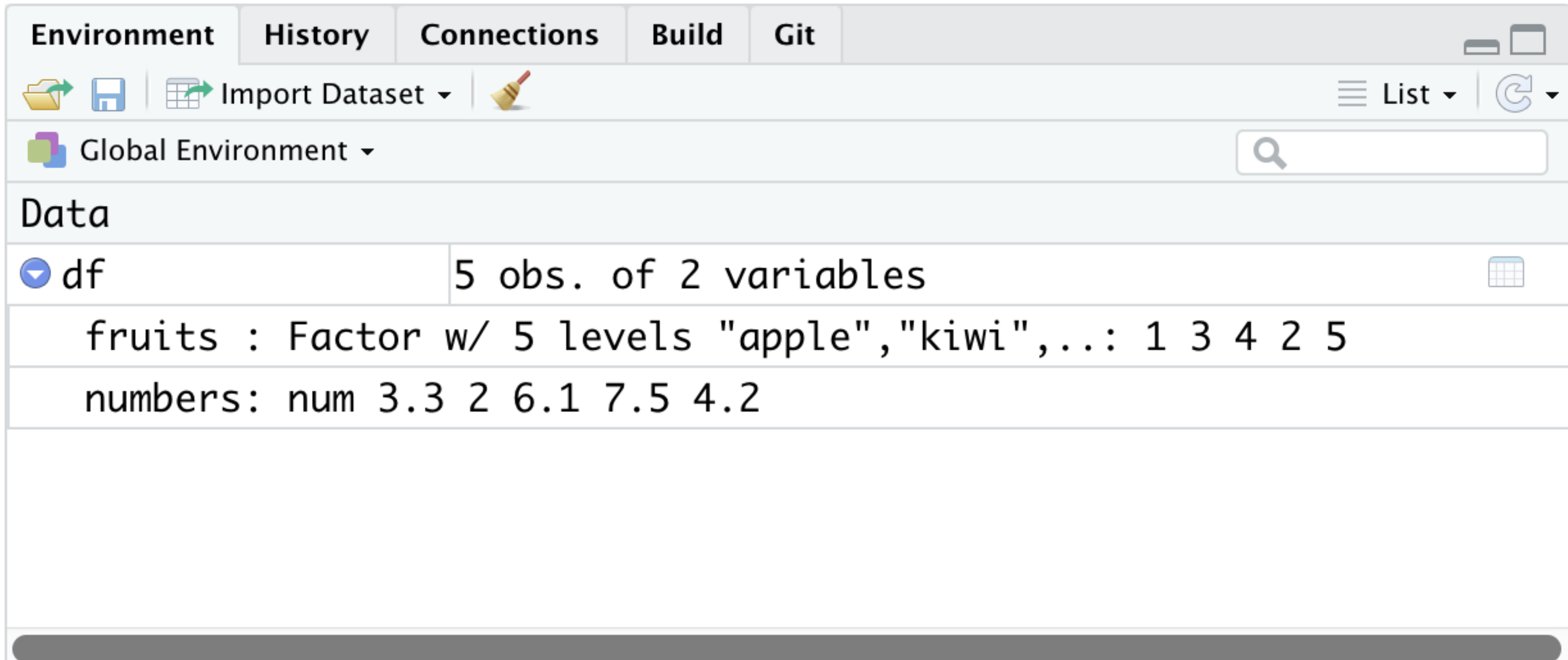
```
1 summary(df)
```

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



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



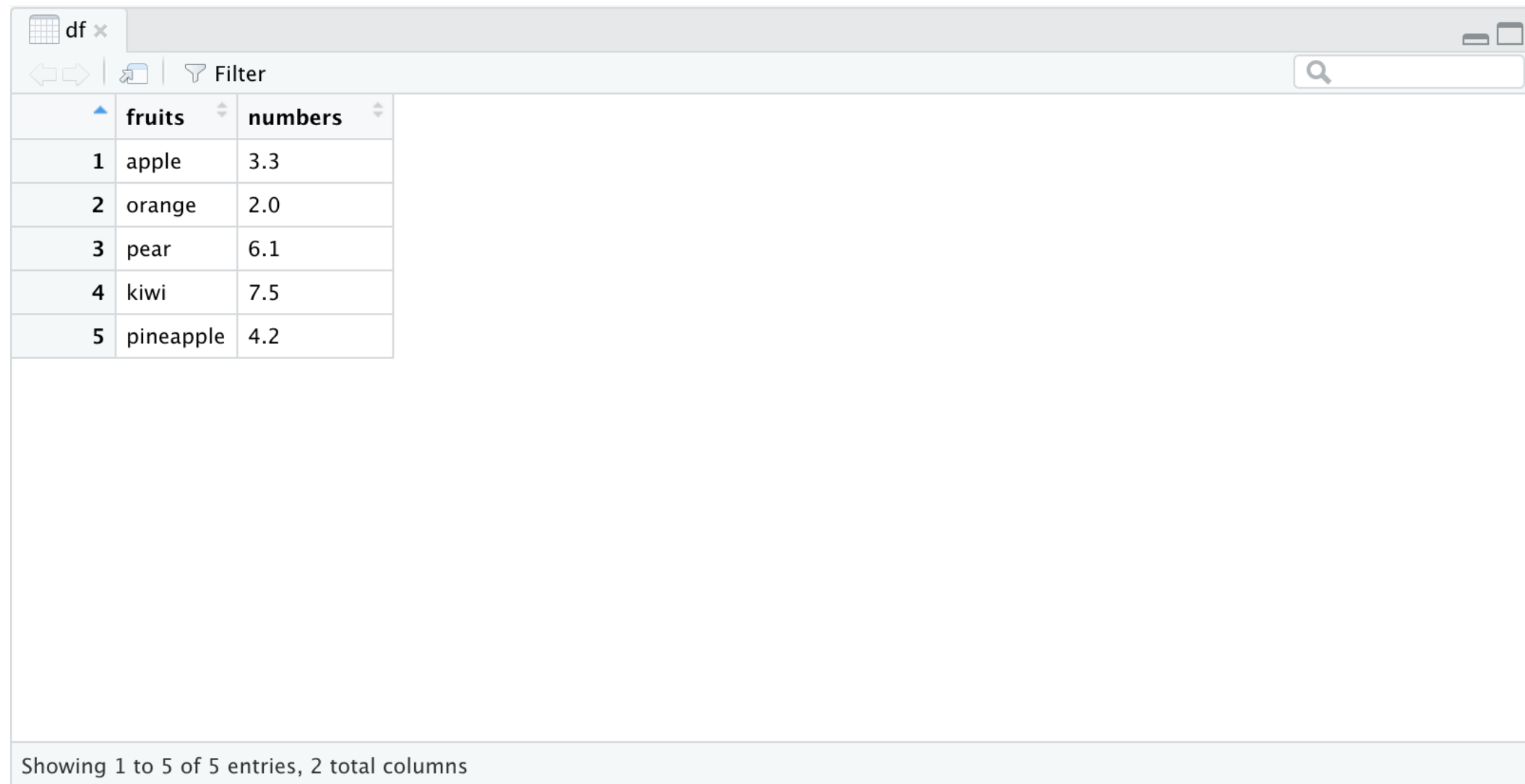
The screenshot shows the RStudio Environment pane. At the top, there are tabs for 'Environment', 'History', 'Connections', 'Build', and 'Git'. Below the tabs is a toolbar with icons for file operations and a search bar. The main area displays the 'Global Environment' with a search bar. Under the 'Data' section, the object 'df' is listed with a blue arrow icon. To the right of 'df' is the text '5 obs. of 2 variables'. Below this, the structure of the dataframe is shown: 'fruits : Factor w/ 5 levels "apple","kiwi",...: 1 3 4 2 5' and 'numbers: num 3.3 2 6.1 7.5 4.2'.

Object	Class	Size
df	data.frame	5 obs. of 2 variables

```
fruits : Factor w/ 5 levels "apple","kiwi",...: 1 3 4 2 5
numbers: num 3.3 2 6.1 7.5 4.2
```

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



The screenshot shows a RStudio data frame viewer for an object named 'df'. The viewer has a header bar with a grid icon, the name 'df', and a close button. Below the header is a toolbar with navigation arrows, a filter icon, and a search box. The main area displays a table with two columns: 'fruits' and 'numbers'. The table contains five rows of data. At the bottom, a status bar indicates 'Showing 1 to 5 of 5 entries, 2 total columns'.

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

Showing 1 to 5 of 5 entries, 2 total columns



Functions Again I

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

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

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

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

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

```
1 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 elements
```

```
[1] 4
```

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

```
[1] 21
```

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

```
[1] 10
```

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

```
[1] 2
```

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

```
[1] 5.25
```

```
1 median(my_vector) # median of all elements
```

```
[1] 4.5
```

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

```
[1] 11.58333
```

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

```
[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

```
1 > 2+(2*3  
2 +
```

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

