

1.2 — Meet R

ECON 480 • Econometrics • Fall 2021

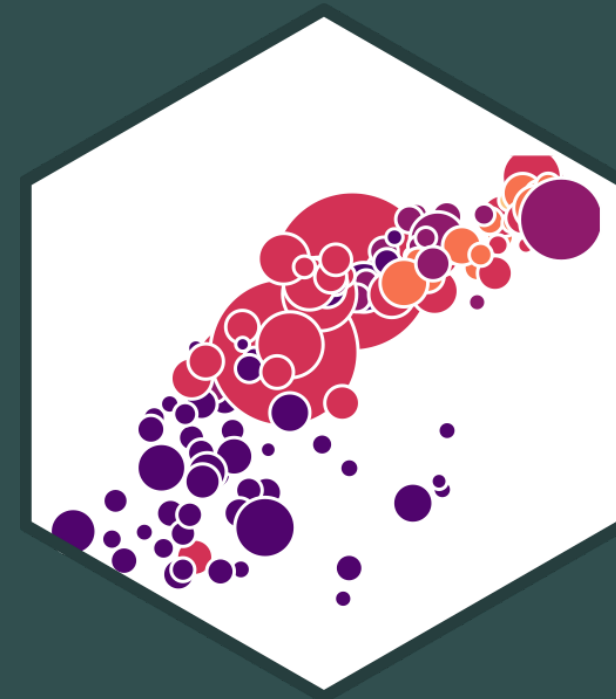
Ryan Safner

Assistant Professor of Economics

✉ safner@hood.edu

🔗 ryansafner/metricsF21

🌐 metricsF21.classes.ryansafner.com



Outline



[Meet R and R Studio](#)

[Ways to Use R](#)

[Coding Basics](#)

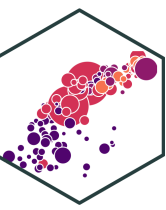
[Types of R Objects](#)

[Data Structures](#)

[Working with Objects](#)


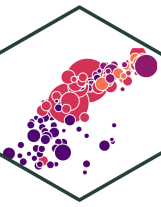
[Working with Data](#)

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

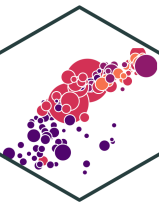
Why Not Excel? I



spreadsheets:
a dystopian moonscape
of unrecorded
user actions

— Gordon Shotwell

Why Not Excel? II



The workbook you opened contains automatic links to information in another workbook.

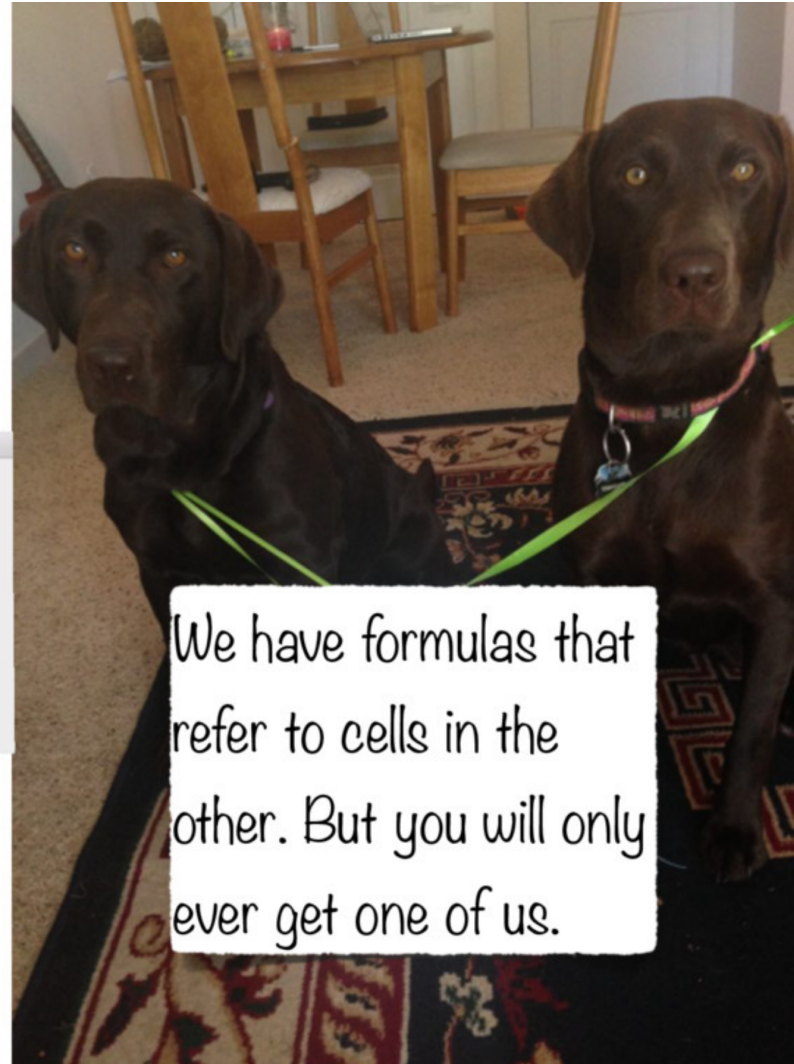
Do you want to update this workbook with changes made to the other workbook?

- To update all linked information, click Update. You must have access to all of the linked workbooks.
- To keep the existing information, click Ignore Links.
- To open your workbook and receive more options to which links get updated, click Edit Links.

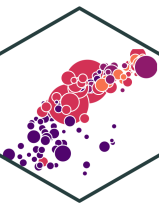
Edit Links

Update

Ignore Links



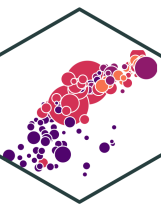
Why Use R?



- **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*

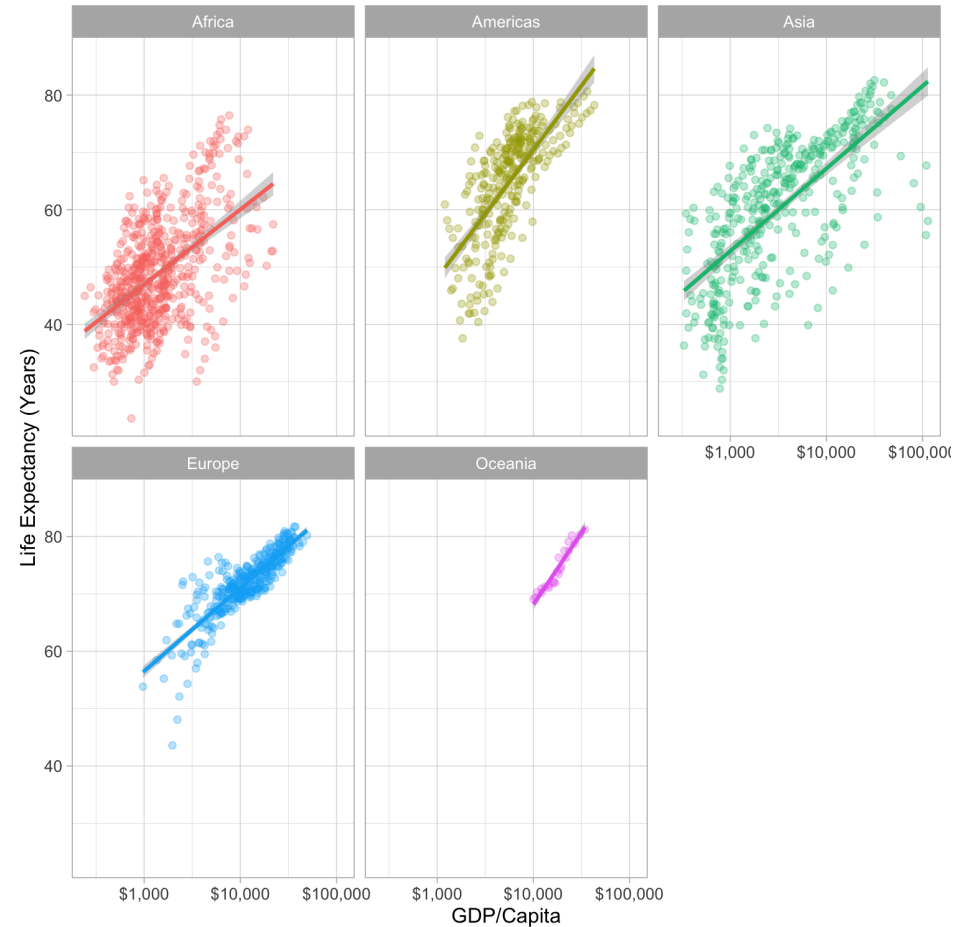


Excel and Stata Can't Do This (In Slides)

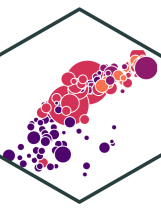


```
library("gapminder")
library("tidyverse")

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,
                        label=scales::dollar
  labs(x = "GDP/Capita",
       y = "Life Expectancy (Years)"
  facet_wrap(~continent)+
  guides(color = F)+
  theme_light()
```



Or This: Execute R Code Inside Your Documents



Code

```
The average GDP per capita is $ `r`  
round(mean(gapminder$gdpPerCap),2)`  
with a standard deviation of $ `r`  
round(sd(gapminder$gdpPerCap),2)`  
.
```

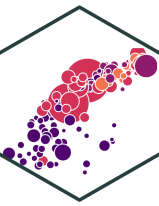
Output

The average GDP per capita is \$7215.33 with a standard deviation of \$9857.45.



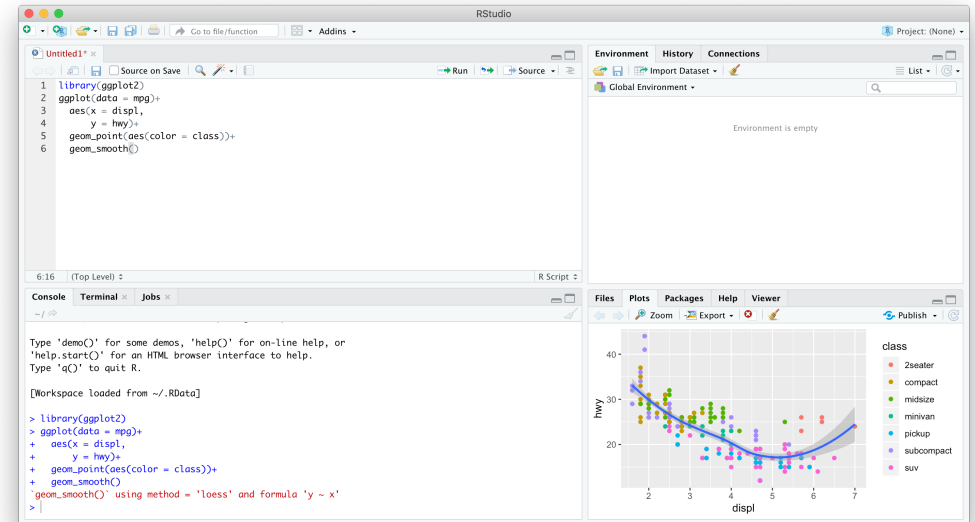
Meet R and R Studio

R and R Studio I



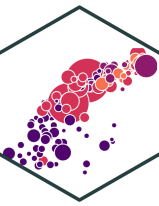
- **R** is the programming language that executes commands
- **R Studio** is an integrated development environment (IDE) that makes your coding life a lot easier
 - Write code in scripts
 - Execute individual commands or entire scripts
 - Auto-complete, highlight syntax
 - View data, objects, and plots
 - Get help and documentation on commands and functions
 - Integrate code into documents with **R**

Markdown

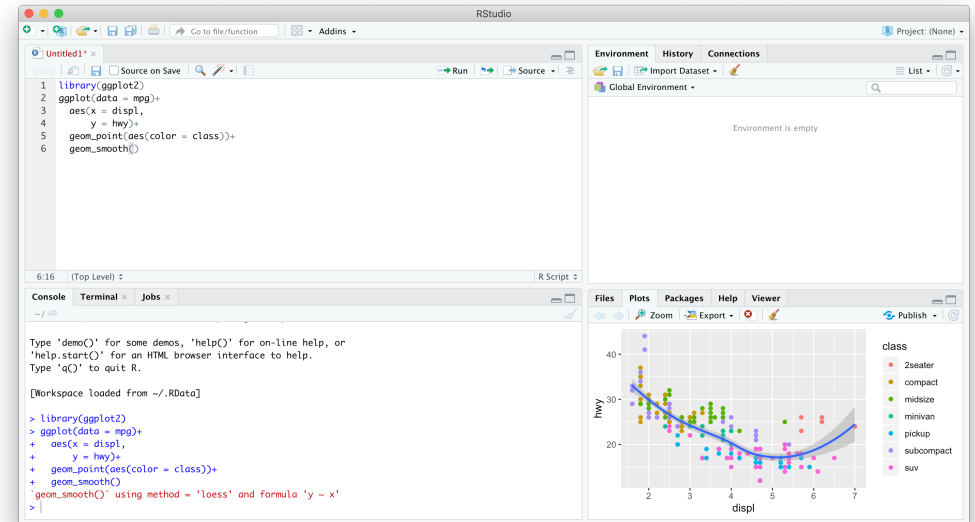


R Studio

R and R Studio II

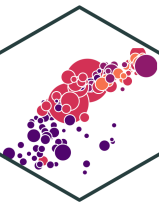


- **R** is like your car's engine, **R Studio** is the dashboard
- You will do everything in **R Studio**
- **R** itself is just a command language (you could run it in your computer's shell/terminal/command prompt)



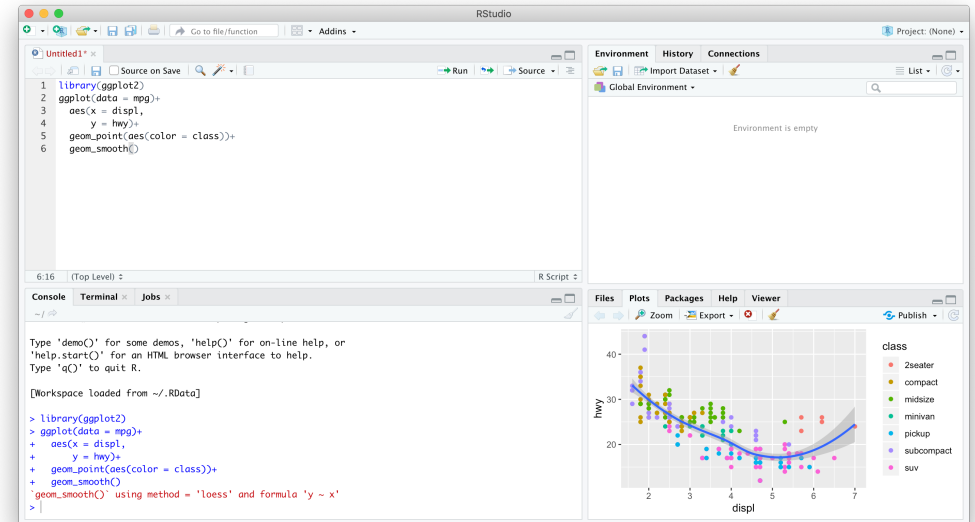
R Studio

R and R Studio III



R Studio has 4 window panes:

1. **Source**[†]: a text editor for documents, R scripts, etc.
2. **Console**: type in commands to run
3. **Browser**: view files, plots, help, etc
4. **Environment**: view created objects, command history, version control

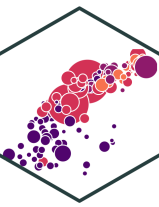


R Studio

[†]May not be immediately visible until you create new files.

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*



Jesse Mostipak
@kierisi



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 💬 72 🔗 Copy link to Tweet

[Tweet your reply](#)



Katie Mack ✓
@AstroKatie

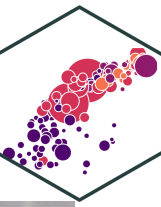


A surprisingly large part of having expertise in a topic is not so much knowing everything about it but learning the language and sources well enough to be extremely efficient in google searches.

11:24 AM · Dec 8, 2018



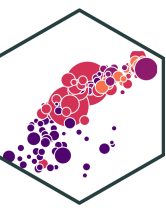
...and Sucking





Ways to Use R

1. Using the Console



- Type individual commands into the console window
- Great for testing individual commands to see what happens
- Not saved! Not reproducible! Not recommended!

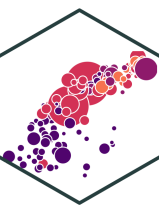
```
2+2
```

```
## [1] 4
```

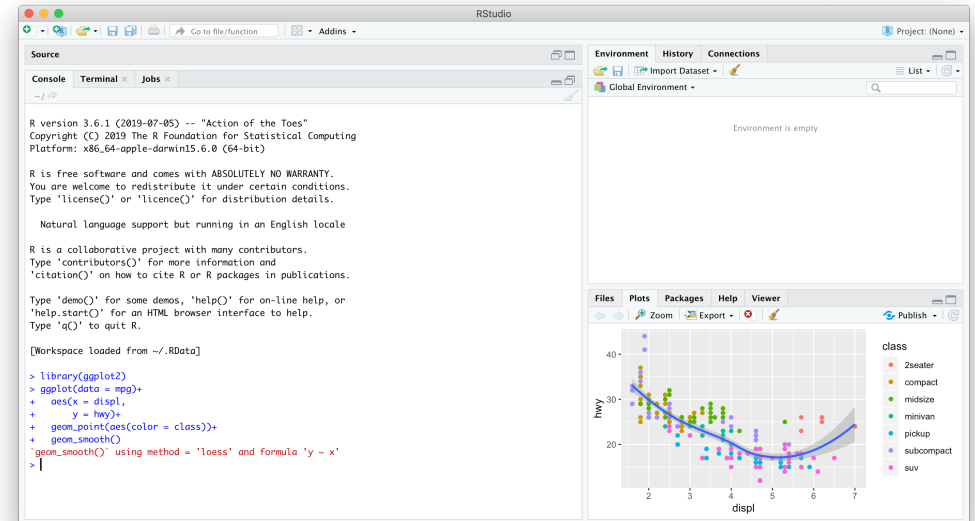
```
summary(mpg$hwy)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  12.00   18.00   24.00   23.44   27.00   44.00
```

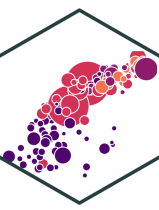

1. Using the Console



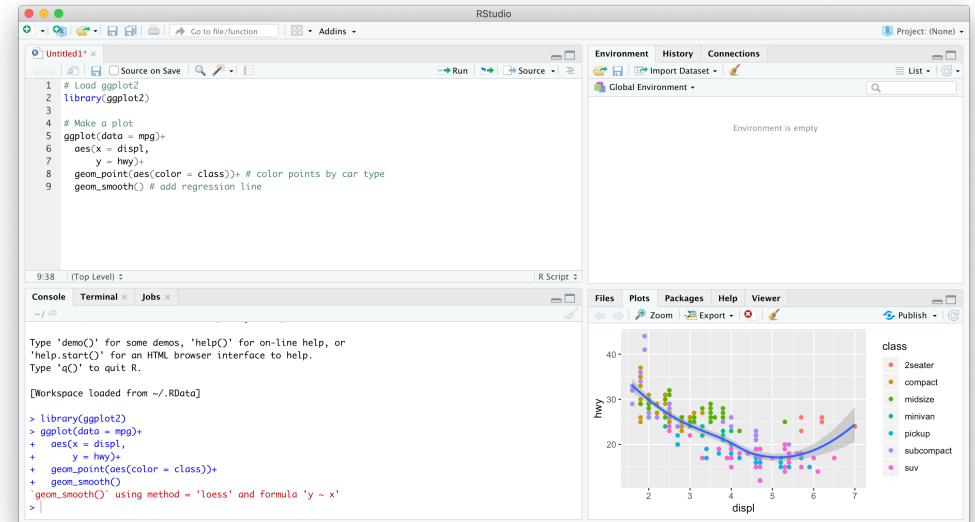
- Type individual commands into the console window
- Great for testing individual commands to see what happens
- Not saved! Not reproducible! Not recommended!



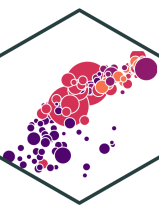
2. Writing an R Script



- 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!



3. Using Markdown

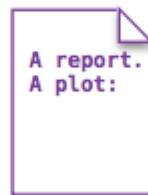


- A later lecture: [R Markdown](#), a simple markup language to write documents in
 - **Optional**, but many students have enjoyed it and use it well beyond this class!
- Can integrate text, [R](#) code, figures, citations & bibliographies in a *single* plain-text file & output into a variety of formats: PDF, webpage, slides, Word doc, etc.

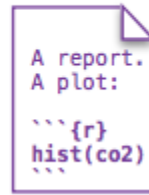
i. Open - Open a file that uses the .Rmd extension.



ii. Write - Write content with the easy to use R Markdown syntax



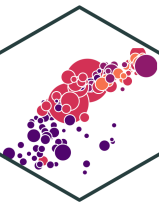
iii. Embed - Embed R code that creates output to include in the report



iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.



For Today

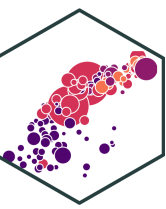


- Practicing typing at the Command line/Console
- Learning different commands and objects relevant for data analysis
- Saving and running `.R` scripts
- Later: `R markdown`, literate programming, workflow management
- **Today may seem a bit overwhelming**
 - You don't need to know or internalize all of this today
 - Use this as a reference to come back to over the semester



Coding Basics

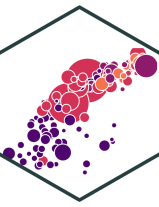
Getting to Know Your Computer



- 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)`[†]
- Soon I'll show you better ways where you won't ever have to worry about this

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

Coding

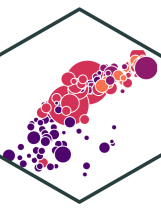


Hadley Wickham

Chief Scientist, R Studio

"There's an implied contract between you and R: it will do the tedious computation for you, but in return, you must be completely precise in your instructions. Typos matter. Case matters." - [R for Data Science, Ch. 4](#)

Say Hello to My Little Friend



Google

how do I make a vector in r

All Videos News Images Shopping More Settings Tools

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

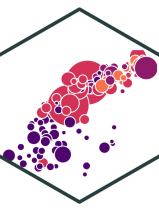
An R tutorial on the concept of **vectors** in R. Discuss how to **create vectors** of numeric, logical and character string data types.

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



stackoverflow

Questions

Developer Jobs

Tags

Users

[r] how do I make a vector

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

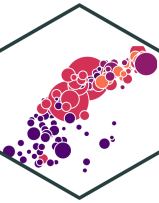
Q: How do I make a specific factor in a vector have a higher level than every other factor?

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](#)

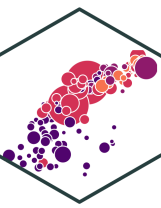
R Is Helpful Too!



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

From Kieran Healy's excellent (free online!) [book on Data Visualization](#).

Tips for Writing Code

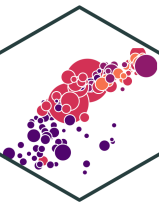


- Comment, comment, comment!
- The hashtag `#` starts a comment, R will ignore everything on the rest of that line

```
# Run regression of y on x, save as reg1  
reg1<-lm(y~x, data=data) #runs regression  
summary(reg1$coefficients) #prints coefficients
```

- 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 hope to cover this later)

Style and Naming



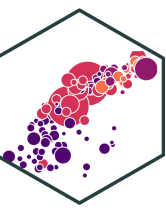
- Once we start writing longer blocks of code, it helps to have a consistent (and human-readable!) style
- I follow [this style guide](#) (you are not required to)[†]
- Naming objects and files will become important[‡]
 - DO NOT USE SPACES! You've seen webpages intended to be called `my webpage in html`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

```
i_use_underscores
some.people.use.snake.case
othersUseCamelCase
```

[†] Also described in [today's course notes page](#) and the course [resources](#).

[‡] Consider your folders on your computer as well...

Coding Basics



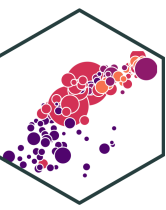
- You'll have to get used to the fact that you are coding in commands to execute
- Start with the easiest: simple math operators and calculations:

```
> 2+2
```

```
## [1] 4
```

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

Coding Basics II



- We can start using more fancy commands

```
2^3
```

```
## [1] 8
```

```
sqrt(25)
```

```
## [1] 5
```

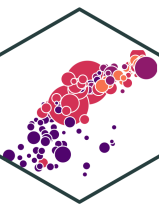
```
log(6)
```

```
## [1] 1.791759
```

```
pi/2
```

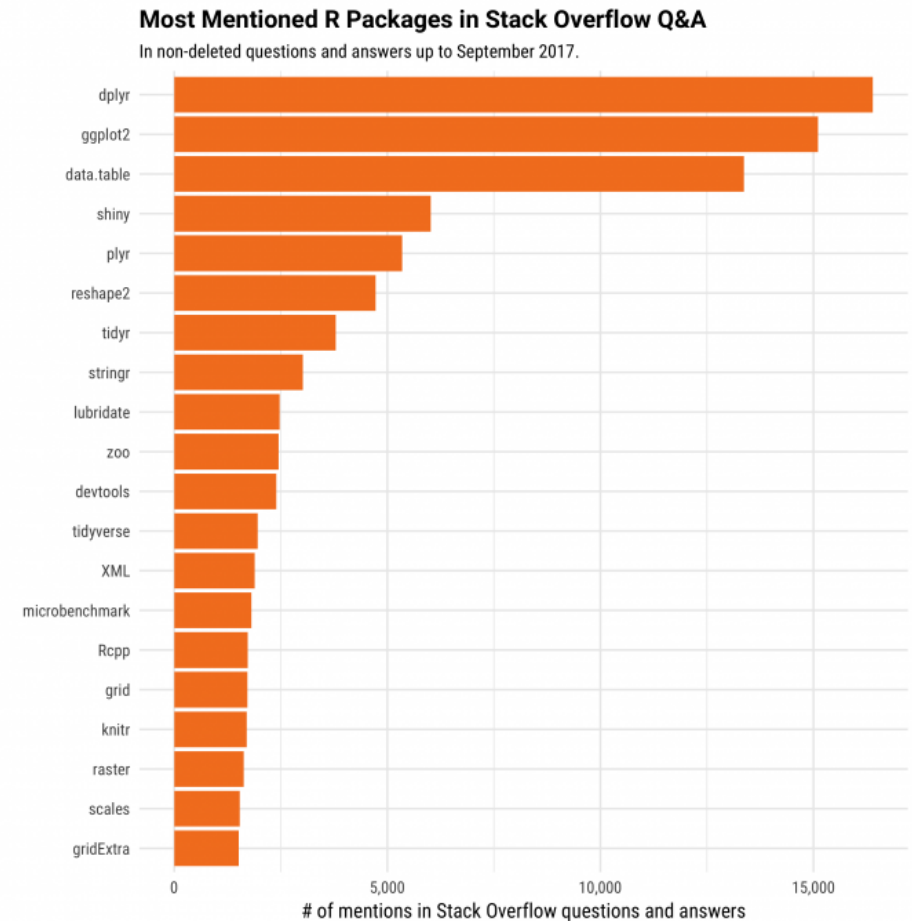
```
## [1] 1.570796
```

Packages

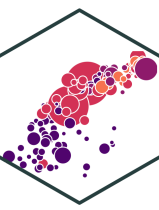


- Since R is open source, users contribute **packages**
 - Really it's just users writing custom functions and saving them for others to use
- Load packages with `library()`
 - e.g. `library("package_name")`
- If you don't have a package, you must first `install.packages()`[†]
 - e.g. `install.packages("package_name")`

[†] Yes, note the plural, even if it's just for one package!



R: Objects and Functions



- R is an **object-oriented** programming language
- 99% of the time, you will be:

1. creating `objects`

- assign values to an object with `=` (or `<-`)

2. running `functions` on `objects`

- syntax:

```
function_name(object_name)
```

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

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

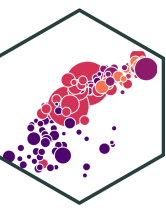
```
# find the sum  
sum(my_object)
```

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

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

```
## [1] 3
```


R: Objects and Functions II



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

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

```
## [1] 1.581139
```

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

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

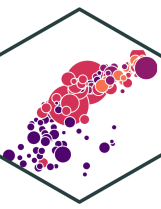
```
# round the sd to two decimals  
round(sd(my_object),2)
```

```
## [1] 1.58
```



Types of R Objects

Numeric



- **Numeric** objects are just numbers[†]
- Can be mathematically manipulated

```
x = 2  
y = 3  
x+y
```

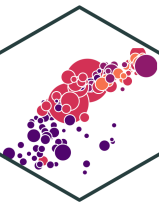
```
## [1] 5
```

```
x*y
```

```
## [1] 6
```

[†] If you want to get technical, R may call these `integer` or `double` if there are decimal values.

Character



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

```
name = "Ryan Safner"  
address = "Washington D.C."
```

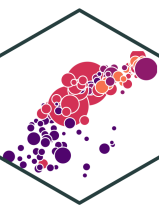
```
name
```

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

```
address
```

```
## [1] "Washington D.C."
```

Logical



- **Logical** objects are **binary** TRUE or FALSE indicators
- 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[†]
 - `&in`: is a member of the set of (\in)
 - `&`: "AND"
 - `|`: "OR"

[†] One `=` *assigns* a value (like `<-`).

Two `==` *evaluate* a conditional statement.

```
z = 10 # set z equal to 10
```

```
z==10 # is z equal to 10?
```

```
## [1] TRUE
```

```
"red"=="blue" # is red equal to blue?
```

```
## [1] FALSE
```

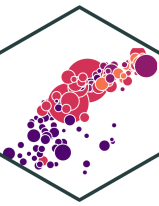
```
z > 1 & z < 12 # is z > 1 AND < 12?
```

```
## [1] TRUE
```

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

```
## [1] TRUE
```

Factor



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

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

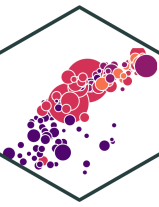
- We'll make much more extensive use of them later

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



Data Structures

Vectors



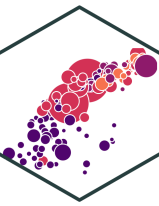
- **Vector**: the simplest type of object, just a collection of elements
- Make a vector using the **combine** `c()` function

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

```
# look at vec  
vec
```

```
## [1] "1"          "orange"      "83.5"        "3.141592653589793"
```


Data Frames I



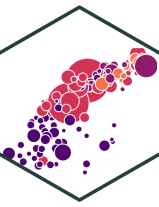
- `Data.frame`: what we'll be using almost always
- Think like a “spreadsheet”
- Each **column** is a vector (variable)
- Each **row** is an observation (pair of values for all variables)

```
library("ggplot2")
```

```
diamonds
```

```
## # A tibble: 53,940 × 10
##   carat cut      color clarity depth table price     x     y
##   <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
## 2  0.21 Premium  E     SI1     59.8    61    326  3.89  3.84  2
## 3  0.23 Good     E     VS1     56.9    65    327  4.05  4.07  2
## 4  0.29 Premium  I     VS2     62.4    58    334  4.2   4.23  2
## 5  0.31 Good     J     SI2     63.3    58    335  4.34  4.35  2
## 6  0.24 Very Good J     VVS2    62.8    57    336  3.94  3.96  2
## 7  0.24 Very Good I     VVS1    62.3    57    336  3.95  3.98  2
## 8  0.26 Very Good H     SI1     61.9    55    337  4.07  4.11  2
## 9  0.22 Fair     E     VS2     65.1    61    337  3.87  3.78  2
## 10 0.23 Very Good H     VS1     59.4    61    338  4     4.05  2
## # ... with 53,930 more rows
```

Data Frames 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

```
# make two vectors
fruits = c("apple", "orange", "pear", "kiwi", "pine
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 the = instead of
df = data.frame(fruits=c("apple", "orange", "pear
                    numbers=c(3.3, 2.0, 6.1, 7.5, 4.2))

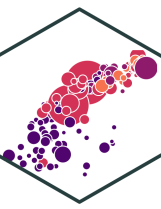
# look at it
df
```

```
##      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

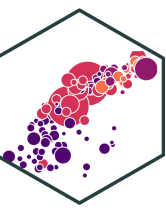


- We want to store things in objects to run functions on them later
- Recall, any object is created with the assignment operator `=` or `<-`

```
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
```

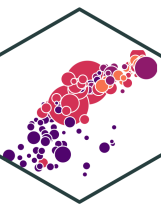
```
## [1] 1 2 3 4 5
```

- 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()`

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

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

```
class(6)
```

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

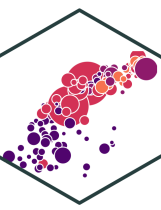
- Can also use logical tests of `is.()`

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

```
## [1] FALSE
```

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

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!

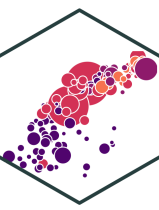
```
as.character(6)
```

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

```
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

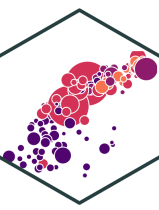
```
mixed_vector = c(pi, 12, "apple", 6.32)
class(mixed_vector)
```

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

```
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[†]

```
df
```

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

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

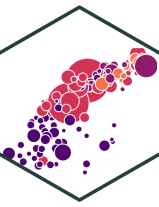
```
## [1] "character"
```

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

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

[†]Remember each column in a data frame is a vector!

More on Data Frames I



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

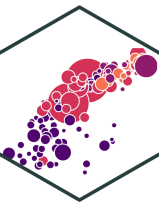
```
class(df)
```

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

```
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 Data Frames II



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

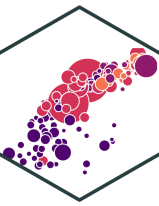
```
head(df)
```

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

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

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

More on Data Frames III



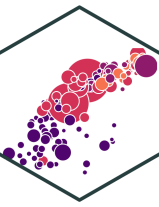
- Get summary statistics[†] by column (variable) with `summary()`

```
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
```

[†] For `numeric` data only; a frequency table is displayed for `character` or `factor` data

More on Data Frames IV

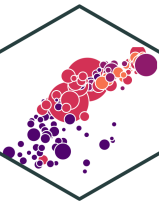


- Note, once you save an object, it shows up in the **Environment Pane** in the upper right window
- Click the blue arrow button in front of the object for some more information

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 is titled 'Global Environment' and contains a section labeled 'Data'. Under 'Data', there is an object named 'df' with a blue arrow icon to its left. To the right of 'df' is the text '5 obs. of 2 variables'. Below this, the structure of the data frame is displayed: 'fruits : Factor w/ 5 levels "apple","kiwi",...: 1 3 4 2 5' and 'numbers: num 3.3 2 6.1 7.5 4.2'.

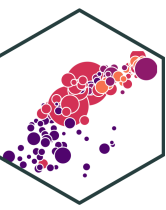
Environment	History	Connections	Build	Git
Global Environment				
Data				
df	5 obs. of 2 variables			
fruits : Factor w/ 5 levels "apple","kiwi",...: 1 3 4 2 5				
numbers: num 3.3 2 6.1 7.5 4.2				

More on Data Frames 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
```

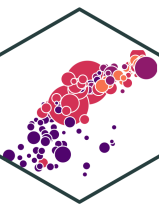
```
my_vector+4 # add 4 to all elements of my_vector
```

```
## [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):

```
length(my_vector) # how many elements?
```

```
## [1] 4
```

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

```
## [1] 21
```

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

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

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

```
.pull-right[
```

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

```
## [1] 5.25
```

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

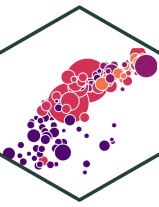
```
## [1] 4.5
```

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

```
## [1] 11.58333
```

```
sd(my_vector) # standard deviation of object
```


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
```

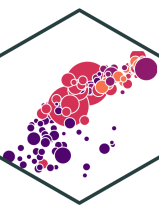
```
+
```

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



Working with Data

Indexing and Subsetting I



```
mtcars
```

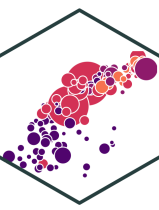
```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02
## Datsun 710      22.8   4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4   6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3   8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40
```

The `mtcars` dataset is automatically built in with `R`.

- Each element in a data frame is **indexed** by referring to its row and column: `df[r, c]`
- To select elements by row and column (“**subset**”), type in the row(s) and/or column(s) to select
 - Leaving `r` or `c` blank selects *all* rows or columns
 - Select multiple values with `c()`[†]
 - Select a range of values with `:`
 - Don't forget the comma between `r` and `c`!

[†] You can also “negate” values, selecting everything **except** for values with a `-` in front of them.

Indexing and Subsetting II



```
mtcars
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Datsun 710     22.8  4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive 21.4  6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7  8 360.0 175 3.15 3.440 17.02
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3  8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4  4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8  4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2  6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8  6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4  8 275.8 180 3.07 4.070 17.40
```

Subset by Row (Observations)

```
mtcars[1,] # first row
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4  21    6  160 110   3.9 2.62 16.46
```

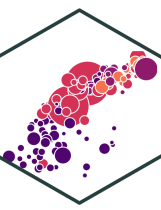
```
mtcars[c(1,3,4),] # first, third, and fourth rows
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4    21.0  6  160 110 3.90 2.620 16.46
## Datsun 710   22.8  4  108  93 3.85 2.320 18.61
## Hornet 4 Drive 21.4  6  258 110 3.08 3.215 19.44
```

```
mtcars[1:3,] # first through third rows
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4    21.0  6  160 110 3.90 2.620 16.46
## Mazda RX4 Wag 21.0  6  160 110 3.90 2.875 17.02
## Datsun 710   22.8  4  108  93 3.85 2.320 18.61
```

Indexing and Subsetting III



```
mtcars
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Datsun 710      22.8  4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4  6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7  8 360.0 175 3.15 3.440 17.02
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3  8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4  4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8  4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2  6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8  6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4  8 275.8 180 3.07 4.070 17.40
```

Subset by Column (Variable)

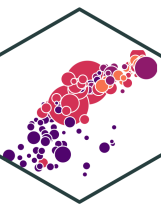
```
mtcars[,6] # select column 6
```

```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440
```

```
mtcars[,2:4] # select columns 2 through 4
```

```
##           cyl  disp  hp
## Mazda RX4      6 160.0 110
## Mazda RX4 Wag  6 160.0 110
## Datsun 710      4 108.0  93
## Hornet 4 Drive  6 258.0 110
## Hornet Sportabout 8 360.0 175
## Valiant        6 225.0 105
## Duster 360     8 360.0 245
## Merc 240D      4 146.7  62
## Merc 230       4 140.8  95
## Merc 280       6 167.6 123
## Merc 280C      6 167.6 123
## Merc 450SE     8 275.8 180
```

Indexing and Subsetting IV



```
mtcars
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4   6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3   8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40
```

Subset by Column (Variable)

- Alternatively, double brackets `[[]]` selects a column by position

```
mtcars[[6]] # select sixth column (wt)
```

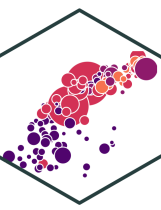
```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440
```

- Can select columns **by name** with `$`

```
mtcars$wt # does the same thing!
```

```
## [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440
```

Indexing and Subsetting V



```
mtcars
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Datsun 710     22.8  4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4  6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7  8 360.0 175 3.15 3.440 17.02
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3  8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4  4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8  4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2  6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8  6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4  8 275.8 180 3.07 4.070 17.40
```

- Select observations (rows) that meet logical (conditional) criteria

Subset by a Condition

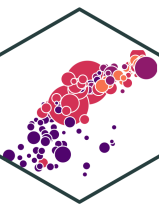
```
mtcars[mtcars$wt>4,] # select all obs with wt>4
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Merc 450SE 16.4   8 275.8 180 3.07 4.07 17.4
```

```
mtcars[mtcars$cyl==6,] # select all obs with exactly 6 cyl
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Hornet 4 Drive  21.4  6 258.0 110 3.08 3.215 19.44
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
## Merc 280       19.2  6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8  6 167.6 123 3.92 3.440 18.90
```

Indexing and Subsetting V



```
mtcars
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Datsun 710     22.8  4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4  6 258.0 110 3.08 3.215 19.44
## Hornet Sportabout 18.7  8 360.0 175 3.15 3.440 17.02
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
## Duster 360     14.3  8 360.0 245 3.21 3.570 15.84
## Merc 240D      24.4  4 146.7  62 3.69 3.190 20.00
## Merc 230       22.8  4 140.8  95 3.92 3.150 22.90
## Merc 280       19.2  6 167.6 123 3.92 3.440 18.30
## Merc 280C      17.8  6 167.6 123 3.92 3.440 18.90
## Merc 450SE     16.4  8 275.8 180 3.07 4.070 17.40
```

- Select observations (rows) that meet logical (conditional) criteria

Subset by a Condition

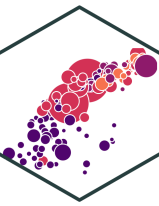
```
mtcars[mtcars$wt>2 & mtcars$wt<3,] # obs where 2<wt<3
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6  160 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6  160 110 3.90 2.875 17.02
## Datsun 710     22.8  4  108  93 3.85 2.320 18.61
```

```
mtcars[mtcars$cyl==4 | mtcars$cyl==6,] # obs with 4 OR 6 cyl
```

```
##           mpg cyl  disp  hp drat   wt  qsec
## Mazda RX4      21.0  6 160.0 110 3.90 2.620 16.46
## Mazda RX4 Wag  21.0  6 160.0 110 3.90 2.875 17.02
## Datsun 710     22.8  4 108.0  93 3.85 2.320 18.61
## Hornet 4 Drive  21.4  6 258.0 110 3.08 3.215 19.44
## Valiant        18.1  6 225.0 105 2.76 3.460 20.22
```


What's To Come



- Next class: data visualization with `ggplot2`
- And then: data wrangling with `tidyverse`
- And then: literate programming and workflow management with `R Markdown`, `R Projects`, maybe `git`
- Finally: onto statistics and econometric theory!