# **Methods Camp**

**UT** Austin, Department of Government

Meiying Xu — Joel Yew

August 2025

# Table of contents

CI	ass so	chedule		6
	Desc	cription	L	6
	Cou	rse outl	line	6
	Con	tact info	io	8
	Ack	nowledg	gements	8
	Mat	erials fr	rom previous editions	8
Se	etup			9
	-	alling R	R and RStudio	9
	Sett	ing up f	for Methods Camp	11
1	Intro	o to R		12
_	1.1	Object	ts	
	1.2		rs and functions	
	1.3		frames and lists	
	1.4		ges	
2	Tidv	, data a	analysis I	20
_	2.1		ng data	_
	2.2		gling data with dplyr	-
		2.2.1	Selecting columns	
		2.2.2	Renaming columns	
		2.2.3	Creating columns	
		2.2.4	Filtering rows	
		2.2.5	Ordering rows	
		2.2.6	Summarizing data	
		2.2.7	Overview	
	2.3		lizing data with ggplot2	
		2.3.1	Univariate plots: categorical	
		2.3.2	Univariate plots: numerical	
		2.3.3	Bivariate plots	
3	Fun	ctions		50
•	3.1	Basics	3	
	0.1	3.1.1	What is a function?	
		3 1 2		<b>F</b> 0

	3.2	Functions in R
	3.3	Common types of functions
		3.3.1 Linear functions
		3.3.2 Quadratic functions
		3.3.3 Cubic functions
		3.3.4 Polynomial functions
		3.3.5 Exponential functions
	3.4	Logarithms and exponents
		3.4.1 Logarithms
		3.4.2 Relationships
		3.4.3 Basic rules
		3.4.4 Natural logarithms
		3.4.5 Illustration of $e$
		3.4.6 Logarithms in R
	3.5	Composite functions (functions of functions)
4	Calc	ulus 67
	4.1	Derivatives
		4.1.1 Rules of differentiation
		4.1.2 Higher-order derivatives
		4.1.3 Partial derivatives
		4.1.4 Differentiability of functions
		4.1.5 How do computers calculate derivatives?
	4.2	Optimization
		4.2.1 Extrema
		4.2.2 Critical points and the First-Order Condition
		4.2.3 Second-Order Condition
		4.2.4 Local or global extrema?
	4.3	Integrals
		4.3.1 Integrals are about infinitesimals too
		4.3.2 Indefinite integrals as antiderivatives
		4.3.3 Solving definite integrals
		4.3.4 Rules of integration
		4.3.5 Solving the problem
_		
5		rices 86
	5.1	Introduction
		5.1.1 Scalars
		5.1.2 Vectors
	5.2	Operators
		5.2.1 Summation
		5.2.2 Product

	5.3	Matrices
		5.3.1 Basics
		5.3.2 Structure
	5.4	Matrix operations
		5.4.1 Addition and subtraction
		5.4.2 Scalar multiplication
		5.4.3 Matrix multiplication
		5.4.4 Properties of operations
	5.5	Special matrices
	5.6	Transpose
	5.7	Inverse
	5.8	Linear systems and matrices
	5.9	OLS and matrices
		5.9.1 Dependent variable
		5.9.2 Independent variables
		5.9.3 Linear regression model
		5.9.4 Estimates
6	Tids	data analysis II 108
U	6.1	Loading data in different formats
	0.1	6.1.1 CSV and R data files
		6.1.2 Excel data files
		6.1.3 Stata and SPSS data files
		6.1.4 Our data for this session
	6.2	Recoding variables
	6.3	Missing values
	6.4	Pivoting data
	6.5	Merging datasets
		6.5.1 Sanity checks
	6.6	Plotting extensions: trend graphs, facets, and customization
		6.6.1 Themes
7		pability 134
		What is probability?
	7.2	Definitions and properties of probability
	7.3	Random variables and probability distributions
		7.3.1 Discrete random variables and probability distributions
		7.3.2 Continuous random variables and probability distributions 136
	7.4	Functions describing probability distributions
		7.4.1 Probability Mass Function (PMF) – Discrete Variables
		7.4.2 Probability Density Function (PDF) – Continuous Variables 137
		7.4.3 Cumulative Density Function (CDF)

	7.5	Comm	non types of probability distributions	140
		7.5.1	Binomial distribution	140
		7.5.2	Uniform distribution	142
		7.5.3	Normal distribution	143
8	Stat	tistics a	and simulations	145
	8.1	Rando	om sampling	145
		8.1.1	Random sampling from theoretical distributions	145
		8.1.2	Random sampling from data	147
	8.2	Statist	tics	150
	8.3	Simula	ations	151
		8.3.1	Loops	152
		8.3.2	An example simulation	153
		8.3.3	Another example simulation: bootstrapping	154
R	eferer	ices		156

### Class schedule

Date	Time	Location
Mon, Aug. 18	1:00 PM - 5:30 PM	RLP 2.606
Tue, Aug. 19	1:00 PM - 5:30 PM	RLP 2.606
Wed, Aug. 20	9:00 AM - 12:00 PM	RLP1.302D
	1:00 PM - 4:30 PM	BAT 5.108
Thu, Aug. 21	9:00 AM - 12:00 PM	RLP 3.106
	1:00 AM - 4:30 PM	RLP 3.106

We'll take short breaks periodically during the morning and afternoon sessions as needed.

### **Description**

Welcome to Introduction to Methods for Political Science, aka "Methods Camp"! Methods Camp is designed to give everyone a chance to brush up on some skills in preparation for the introductory Statistics and Formal Theory courses. The other goal of Methods Camp is to allow you to get to know your cohort. We hope that matrix algebra and the chain rule will still prove to be good bonding exercises!

As you can see from the above schedule, we'll be meeting on Monday, August 18th, through Thursday, August 21st. Classes at UT begin the start of the following week on Monday, August 25th. Below is a tentative schedule outlining what will be covered in the class, although we may rearrange things if we find we're going too slowly or too quickly through the material.

### Course outline

1 Monday afternoon: Intro to R

- Introductions
- R and RStudio: basics
- Objects (vectors, matrices, data frames, etc.)
- Basic functions (mean(), length(), etc.)

• Packages: installation and loading (including the tidyverse)

### 2 Monday afternoon: Tidy data analysis I

- Tidy data
- Data wrangling with dplyr
- Data visualization basics with ggplot2

### 3 Tuesday morning: Functions

- Definitions
- Functions in R
- Common types of functions
- Logarithms and exponents
- Composite functions

### 4 Tuesday afternoon: Calculus

- Derivatives
- Optimization
- Integrals

### 5 Wednesday morning: Matrices

- Matrices
- Systems of linear equations
- Matrix operations (multiplication, transpose, inverse, determinant)
- Solving systems of linear equations in matrix form (and why that's cool)
- Introduction to OLS

### 6 Wednesday afternoon: Tidy data analysis II

- Loading data in different formats (.csv, R, Excel, Stata, SPSS)
- Recoding values (if\_else(), case\_when())
- Handling missing values
- Pivoting data
- Merging data
- Plotting extensions (trend graphs, facets, customization)

### 7 Thursday morning: Probability

- Probability: basic concepts
- Random variables, probability distributions, and their properties
- Common probability distributions

### 8 Thursday afternoon: Statistics and simulations

• Statistics: basic concepts

• Random sampling and loops in R

• Simulation example: bootstrapping

### 9 Thursday afternoon: Wrap-up

- Project management fundamentals
- Self-study resources and materials
- Other software (Overleaf, Zotero, etc.)
- Methods resources at UT

### Contact info

If you have any questions during or outside of the methods camp, you can contact us via email. Or if you are curious about our research, you can also check out our respective websites and Twitter accounts (or should we say X...):

• Meiying Xu: xu.meiying@utexas.edu

• Joel Yew: joel.yew@utexas.edu

### **Acknowledgements**

We thank previous Methods Camp instructors for their accumulated experience and materials, upon which we have based ours upon. UT Gov Prof. Max Goplerud gave us amazing feedback for this iteration of Methods Camp (2025). All errors remain our own (and will hopefully be fixed with your help!).

### Materials from previous editions

- 2024: co-taught by Andrés Cruz and Meiying Xu.
- 2023: co-taught by Andrés Cruz and Matt Martin.

## Setup

### Installing R and RStudio

R is a programming language optimized for statistics and data analysis. Most people use R from RStudio, a graphical user interface (GUI) that includes a file pane, a graphics pane, and other goodies. Both R and RStudio are open source, i.e., free as in beer and free as in freedom!

Your first steps should be to install R and RStudio, in that order (if you have installed these programs before, make sure that your versions are up-to-date—if they are not, simply follow the instructions below to re-install them):

- 1. Download and install R from the official website, CRAN. Click on "Download R for <Windows/MacOS>" and follow the instructions. If you have a Mac, make sure to select the version appropriate for your system (Apple Silicon for newer M1/M2/M3 Macs and Intel for older Macs).
- 2. Download and install RStudio from the official website. Scroll down and select the installer for your operating system (most likely the .exe for Windows 10/11 or the .dmg for macOS 12+).

After these two steps, you can open RStudio in your system, as you would with any program. You should see something like this:

### Note for Windows users

While the installation steps above should be enough for most tasks, we also suggest that Windows users install RTools (click on the "Rtools44 installer" link at the middle of the package to get the .exe file). Rtools is needed on Windows to install some advanced packages, so it is a good idea to have it on your system.

That's it for the installation! We also *strongly* recommend that you change a couple of RStudio's default settings.<sup>1</sup> You can change settings by clicking on Tools > Global Options in the menubar. Here are our recommendations:

<sup>&</sup>lt;sup>1</sup>The idea behind these settings (or at least the first two) is to force R to start from scratch with each new session. No lingering objects from previous coding sessions avoids misunderstandings and helps with reproducibility!



Figure 1: How RStudio looks after a clean installation.

- General > Uncheck "Restore .RData into workspace at startup"
- General > Save workspace to .RData on Exit > Select "Never"
- Code > Check "Use native pipe operator"
- Tools > Global Options > Appearance to change to a dark theme, if you want! Pros: better for night sessions, hacker vibes...

### **Setting up for Methods Camp**

All materials for Methods Camp are both on this website and available as RStudio projects for you to execute locally. An RStudio project is simply a folder where one keeps scripts, datasets, and other files needed for a data analysis project.



### 🔔 Warning

Make sure to properly unzip the materials. Double-clicking the .zip file on most Windows systems will not unzip the folder—you must do Right click > Extract all.

You should now have a folder called methodscamp\_part1/ on your computer. Navigate to the methodscamp part1. Rproj file within it and open it. RStudio should open the project right away. You should see methodscamp\_part1 on the top-right of RStudio—this indicates that you are working in our RStudio project.



Figure 2: How the bottom-right corner of RStudio looks after opening our project.

That's all for setup! We can now start coding. After opening our RStudio project, we'll begin by opening the  $01_r_{intro.qmd}$  file from the "Files" panel, in the bottom-right portion of RStudio. This is a Quarto document, which contains both code and explanations (you can also read the materials in the next chapter of this website).

<sup>&</sup>lt;sup>2</sup>Perhaps you have used R Markdown before. Quarto is the next iteration of R Markdown, and is both more flexible and more powerful!

## 1 Intro to R

In Quarto documents like this one, we can write comments by just using plain text. In contrast, code needs to be within *code blocks*, like the one below. To execute a code block, you can click on the little "Play" button or press Cmd/Ctrl + Shift + Enter when your keyboard is hovering the code block.

2 + 2

[1] 4

That was our first R command, a simple math operation. Of course, we can also do more complex arithmetic:

12345  $^{\circ}$  2 / (200 + 25 - 6 \* 2) # this is an inline comment, see the leading "#"

[1] 715488.4

In order to create a code block, you can press Cmd/Ctrl + Alt + i or click on the little green "+C" icon on top of the script.

i Exercise

Create your own code block below and run a math operation.

### 1.1 Objects

A huge part of R is working with *objects*. Let's see how they work:

my\_object <- 10 # opt/alt + minus sign will make the arrow</pre>

my\_object # to print the value of an object, just call its name

[1] 10

We can now use this object in our operations:

```
2 ^ my_object
```

[1] 1024

Or even create another object out of it:

```
my_object2 <- my_object * 2</pre>
```

```
my_object2
```

[1] 20

You can delete objects with the  ${\tt rm}()$  function (for "remove"):

```
rm(my_object2)
```

### 1.2 Vectors and functions

Objects can be of different types. One of the most useful ones is the *vector*, which holds a series of values. To create one manually, we can use the c() function (for "combine"):

```
my\_vector \leftarrow c(6, -11, my\_object, 0, 20)
```

my\_vector

[1] 6 -11 10 0 20

One can also define vectors by sequences:

```
3:10
[1]
    3 4 5 6 7 8 9 10
We can use square brackets to retrieve parts of vectors:
my_vector[4] # fourth element
[1] 0
my_vector[1:2] # first two elements
[1]
      6 -11
Let's check out some basic functions we can use with numbers and numeric vectors:
sqrt(my_object) # squared root
[1] 3.162278
log(my_object) # logarithm (natural by default)
[1] 2.302585
abs(-5) # absolute value
[1] 5
mean(my_vector)
[1] 5
```

[1] 6

median(my\_vector)

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

[1] 11.53256

```
sum(my_vector)
```

[1] 25

```
min(my_vector) # minimum value
```

[1] -11

```
max(my_vector) # maximum value
```

[1] 20

```
length(my_vector) # length (number of elements)
```

[1] 5

Notice that if we wanted to save any of these results for later, we would need to assign them:

```
my_mean <- mean(my_vector)</pre>
```

```
my_mean
```

[1] 5

These functions are quite simple: they take one object and do one operation. A lot of functions are a bit more complex—they take multiple objects or take options. For example, see the sort() function, which by default sorts a vector *increasingly*:

```
sort(my_vector)
```

```
[1] -11 0 6 10 20
```

If we instead want to sort our vector *decreasingly*, we can use the **decreasing** = TRUE argument (T also works as an abbreviation for TRUE).

```
sort(my_vector, decreasing = TRUE)
```

[1] 20 10 6 0 -11



If you use the argument values in order, you can avoid writing the argument names (see below). This is sometimes useful, but can also lead to confusing code—use it with caution.

```
sort(my_vector, T)
[1] 20 10 6 0 -11
```

A useful function to create vectors in sequence is **seq()**. Notice its arguments:

```
seq(from = 30, to = 100, by = 5)
```

```
[1] 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
```

To check the arguments of a function, you can examine its help file: look the function up on the "Help" panel on RStudio or use a command like the following: ?sort.

### i Exercise

Examine the help file of the log() function. How can we compute the base-10 logarithm of my\_object? Your code:

Other than numeric vectors, character vectors are also useful:

```
my_character_vector <- c("Apple", "Orange", "Watermelon", "Banana")</pre>
```

```
my_character_vector[3]
```

[1] "Watermelon"

```
nchar(my_character_vector) # count number of characters
```

[1] 5 6 10 6

### 1.3 Data frames and lists

Another useful object type is the *data frame*. Data frames can store multiple vectors in a tabular format. We can manually create one with the data.frame() function:

### my\_data\_frame

	fruit	calories_per_100g	water_per_100g
1	Apple	52	85.6
2	Orange	47	86.8
3	${\tt Watermelon}$	30	91.4
4	Banana	89	74.9

Now we have a little 4x3 data frame of fruits with their calorie counts and water composition. We gathered the nutritional information from the USDA (2019).

We can use the data\_frame\$column construct to access the vectors within the data frame:

```
mean(my_data_frame$calories_per_100g)
```

[1] 54.5

### i Exercise

Obtain the maximum value of water content per 100g in the data. Your code:

Some useful commands to learn attributes of our data frame:

```
dim(my_data_frame)
```

[1] 4 3

```
nrow(my_data_frame)
```

[1] 4

### names(my\_data\_frame) # column names

```
[1] "fruit" "calories_per_100g" "water_per_100g"
```

We will learn much more about data frames in our next module on data analysis.

After talking about vectors and data frames, the last object type that we will cover is the *list*. Lists are super flexible objects that can contain just about anything:

```
my_list <- list(my_object, my_vector, my_data_frame)</pre>
```

```
[[1]]
[1] 10
[[2]]
[1]
      6 -11 10
                   0 20
[[3]]
       fruit calories_per_100g water_per_100g
1
       Apple
                             52
2
                             47
                                           86.8
      Orange
3 Watermelon
                             30
                                           91.4
      Banana
                             89
                                           74.9
```

To retrieve the elements of a list, we need to use double square brackets:

```
my_list[[1]]
```

#### [1] 10

my\_list

Lists are sometimes useful due to their flexibility, but are much less common in routine data analysis compared to vectors or data frames.

### 1.4 Packages

The R community has developed thousands of packages, which are specialized collections of functions, datasets, and other resources. To install one, you should use the install.packages() command. Below we will install the tidyverse package, a suite for data analysis that we will use in the next modules. You just need to install packages once, and then they will be available system-wide.

```
install.packages("tidyverse") # this can take a couple of minutes
```

If you want to use an installed package in your script, you must load it with the library() function. Some packages, as shown below, will print descriptive messages once loaded.

### library(tidyverse)

```
-- Attaching core tidyverse packages ------
                                                  ----- tidyverse 2.0.0 --
v dplyr
            1.1.4
                      v readr
                                  2.1.5
v forcats
            1.0.0
                                  1.5.1
                      v stringr
v ggplot2
            3.5.2
                      v tibble
                                  3.3.0
v lubridate 1.9.4
                      v tidyr
                                  1.3.1
v purrr
            1.1.0
               ------ tidyverse_conflicts() --
-- Conflicts --
x dplyr::filter() masks stats::filter()
x dplyr::lag()
                  masks stats::lag()
i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/</a>) to force all conflicts to become
```

### ⚠ Warning

Remember that install.packages("package") needs to be executed just once, while library(package) needs to be in each script in which you plan to use the package. In general, never include install.packages("package") as part of your scripts or Quarto documents!

## 2 Tidy data analysis I

The tidyverse is a suite of packages that streamline data analysis in R. After installing the tidyverse with install.packages("tidyverse") (see the previous module), you can load it with:

### library(tidyverse)

```
-- Attaching core tidyverse packages -----
                                                    ----- tidyverse 2.0.0 --
v dplyr
            1.1.4
                      v readr
                                  2.1.5
v forcats
            1.0.0
                                  1.5.1
                      v stringr
v ggplot2
            3.5.2
                      v tibble
                                  3.3.0
                      v tidyr
                                  1.3.1
v lubridate 1.9.4
v purrr
            1.1.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()
                  masks stats::lag()
i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/</a>) to force all conflicts to become
```



Upon loading, the tidyverse prints a message like the one above. Notice that multiple packages (the constituent elements of the "suite") are actually loaded. For instance, dplyr and tidyr help with data wrangling and transformation, while ggplot2 allows us to draw plots. In most cases, one just loads the tidyverse and forgets about these details, as the constituent packages work together nicely.

Throughout this module, we will use tidyverse functions to load, wrangle, and visualize real data.

### 2.1 Loading data

Throughout this module we will work with a dataset of senators during the Trump presidency, which was adapted from FiveThirtyEight (2021).

We have stored the dataset in .csv format under the data/ subfolder. Loading it into R is simple (notice that we need to assign it to an object):

```
trump_scores <- read_csv("data/trump_scores_538.csv")</pre>
```

Rows: 122 Columns: 8
-- Column specification -----Delimiter: ","

chr (4): bioguide, last\_name, state, party

dbl (4): num\_votes, agree, agree\_pred, margin\_trump

- i Use `spec()` to retrieve the full column specification for this data.
- i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

#### trump\_scores

# 1	A tibble:	122 x 8						
	bioguide	last_name	${\tt state}$	party	${\tt num\_votes}$	agree	agree_pred	margin_trump
	<chr></chr>	<chr></chr>	<chr></chr>	<chr></chr>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>
1	A000360	Alexander	TN	R	118	0.890	0.856	26.0
2	B000575	Blunt	MO	R	128	0.906	0.787	18.6
3	B000944	Brown	OH	D	128	0.258	0.642	8.13
4	B001135	Burr	NC	R	121	0.893	0.560	3.66
5	B001230	Baldwin	WI	D	128	0.227	0.510	0.764
6	B001236	Boozman	AR	R	129	0.915	0.851	26.9
7	B001243	Blackburn	TN	R	131	0.885	0.889	26.0
8	B001261	Barrasso	WY	R	129	0.891	0.895	46.3
9	B001267	Bennet	CO	D	121	0.273	0.417	-4.91
10	B001277	Blumenthal	CT	D	128	0.203	0.294	-13.6
# :	# i 112 more rows							

Let's review the dataset's columns:

- bioguide: A unique ID for each politician, from the Congress Bioguide.
- last\_name
- state
- party
- num\_votes: Number of votes for which data was available.
- agree: Proportion (0-1) of votes in which the senator voted in agreement with Trump.
- agree\_pred: Predicted proportion of vote agreement, calculated using Trump's margin (see next variable).

• margin\_trump: Margin of victory (percentage points) of Trump in the senator's state.

We can inspect our data by using the interface above. An alternative is to run the command View(trump\_scores) or click on the object in RStudio's environment panel (in the top-right section).

Do you have any questions about the data?

By the way, the tidyverse works amazingly with *tidy data*. If you can get your data to this format (and we will see ways to do this), your life will be much easier:

### 2.2 Wrangling data with dplyr

We often need to modify data to conduct our analyses, e.g., creating columns, filtering rows, etc. In the tidyverse, these operations are conducted with multiple *verbs*, which we will review now.

### 2.2.1 Selecting columns

We can select specific columns in our dataset with the select() function. All dplyr wrangling verbs take a data frame as their first argument—in this case, the columns we want to select are the other arguments.

```
select(trump_scores, last_name, party)
```

```
# A tibble: 122 x 2
   last name party
   <chr>>
              <chr>>
 1 Alexander
              R
2 Blunt
              R
3 Brown
              D
4 Burr
              R
5 Baldwin
              D
6 Boozman
              R.
7 Blackburn
              R
8 Barrasso
              R.
9 Bennet
              D
10 Blumenthal D
# i 112 more rows
```



-HADLEY WICKHAM

# In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

6	each col	umn a v	ariable \	
	id	name	color	
	1	floof	gray	each row
	2	max	black	← an
	3	cat	orange	Dobservation
	4	donut	gray	2//
	5	merlin	black	4/
	6	panda	calico	4

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10



(a) Source: Illustrations from the Openscapes blog *Tidy Data for reproducibility, efficiency, and collaboration* by Julia Lowndes and Allison Horst.

This is a good moment to talk about "pipes." Notice how the code below produces the same output as the one above, but with a slightly different syntax. Pipes (|>) "kick" the object on the left of the pipe to the first argument of the function on the right. One can read pipes as "then," so the code below can be read as "take trump\_scores, then select the columns last\_name and party." Pipes are very useful to *chain multiple operations*, as we will see in a moment.

```
trump_scores |>
  select(last_name, party)
```

```
# A tibble: 122 x 2
   last_name party
   <chr>
              <chr>
1 Alexander
2 Blunt
              R
3 Brown
              D
4 Burr
              R
5 Baldwin
              D
6 Boozman
              R
7 Blackburn R
8 Barrasso
              R
9 Bennet
10 Blumenthal D
# i 112 more rows
```

### **?** Tip

You can insert a pipe with the Cmd/Ctrl + Shift + M shortcut. If you have not changed the default RStudio settings, an "old" pipe (%>%) might appear. While most of the functionality is the same, the |> "new" pipes are more readable and don't need any extra packages (to use %>% you need the tidyverse or one of its packages). You can change this RStudio option in Tools > Global Options > Code > Use native pipe operator. Make sure to check the other suggested settings in our Setup module!

Going back to selecting columns, you can select ranges:

```
trump_scores |>
select(bioguide:party)
```

```
# A tibble: 122 x 4
bioguide last_name state party
```

```
<chr>
            <chr>
                       <chr> <chr>
1 A000360 Alexander
                             R.
                      TN
2 B000575
           Blunt
                       MO
                             R.
3 B000944 Brown
                       OH
                             D
4 B001135 Burr
                       NC
                             R
5 B001230 Baldwin
                       WΙ
                             D
6 B001236 Boozman
                       AR
                             R
7 B001243 Blackburn TN
                             R
8 B001261 Barrasso
                             R
                       WY
9 B001267 Bennet
                       CO
                             D
10 B001277 Blumenthal CT
                             D
# i 112 more rows
```

You can also **de**select columns using a minus sign:

```
trump_scores |>
select(-last_name)
```

```
# A tibble: 122 x 7
   bioguide state party num_votes agree agree_pred margin_trump
   <chr>>
            <chr> <chr>
                             <dbl> <dbl>
                                                            <dbl>
                                              <dbl>
1 A000360
                                                           26.0
            TN
                  R
                               118 0.890
                                              0.856
2 B000575
            MO
                  R
                               128 0.906
                                              0.787
                                                           18.6
3 B000944
            OH
                  D
                               128 0.258
                                              0.642
                                                            8.13
4 B001135
           NC
                  R
                               121 0.893
                                              0.560
                                                            3.66
5 B001230 WI
                               128 0.227
                                                            0.764
                  D
                                              0.510
6 B001236
                               129 0.915
                                                           26.9
           AR
                  R
                                              0.851
7 B001243
                                                           26.0
            TN
                  R
                               131 0.885
                                              0.889
8 B001261
            WY
                  R
                               129 0.891
                                              0.895
                                                           46.3
9 B001267
                  D
                               121 0.273
                                              0.417
                                                           -4.91
10 B001277 CT
                               128 0.203
                                              0.294
                                                          -13.6
# i 112 more rows
```

And use a few helper functions, like matches():

```
trump_scores |>
select(last_name, matches("agree"))
```

1	Alexander	0.890	0.856	
2	Blunt	0.906	0.787	
3	Brown	0.258	0.642	
4	Burr	0.893	0.560	
5	Baldwin	0.227	0.510	
6	Boozman	0.915	0.851	
7	Blackburn	0.885	0.889	
8	Barrasso	0.891	0.895	
9	Bennet	0.273	0.417	
10 Blumenthal 0.203 0.294				
# i 112 more rows				

Or everything(), which we usually use to reorder columns:

```
trump_scores |>
select(last_name, everything())
```

### # A tibble: 122 x 8

	last_name	bioguide	state	party	num_votes	agree	agree_pred	margin_trump
	<chr></chr>	<chr></chr>	<chr></chr>	<chr></chr>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>
1	Alexander	A000360	TN	R	118	0.890	0.856	26.0
2	Blunt	B000575	MO	R	128	0.906	0.787	18.6
3	Brown	B000944	OH	D	128	0.258	0.642	8.13
4	Burr	B001135	NC	R	121	0.893	0.560	3.66
5	Baldwin	B001230	WI	D	128	0.227	0.510	0.764
6	Boozman	B001236	AR	R	129	0.915	0.851	26.9
7	Blackburn	B001243	TN	R	131	0.885	0.889	26.0
8	Barrasso	B001261	WY	R	129	0.891	0.895	46.3
9	Bennet	B001267	CO	D	121	0.273	0.417	-4.91
10	${\tt Blumenthal}$	B001277	CT	D	128	0.203	0.294	-13.6

# i 112 more rows

### **?** Tip

Notice that all these commands have not edited our existent objects—they have just printed the requested outputs to the screen. In order to modify objects, you need to use the assignment operator (<-). For example:

```
trump_scores_reduced <- trump_scores |>
select(last_name, matches("agree"))
```

```
trump_scores_reduced
# A tibble: 122 x 3
   last_name agree agree_pred
              <dbl>
   <chr>
                          <dbl>
 1 Alexander 0.890
                         0.856
 2 Blunt
              0.906
                         0.787
                         0.642
 3 Brown
              0.258
 4 Burr
              0.893
                         0.560
 5 Baldwin
              0.227
                         0.510
 6 Boozman
              0.915
                         0.851
 7 Blackburn 0.885
                         0.889
 8 Barrasso
              0.891
                         0.895
 9 Bennet
              0.273
                         0.417
10 Blumenthal 0.203
                         0.294
# i 112 more rows
```

### i Exercise

Select the variables last\_name, party, num\_votes, and agree from the data frame. Your code:

### 2.2.2 Renaming columns

We can use the rename() function to rename columns, with the syntax new\_name = old\_name. For example:

```
trump_scores |>
  rename(prop_agree = agree, prop_agree_pred = agree_pred)
```

```
# A tibble: 122 x 8
  bioguide last_name state party num_votes prop_agree prop_agree_pred
            <chr>
                       <chr> <chr>
                                       <dbl>
   <chr>
                                                  <dbl>
                                                                   <dbl>
 1 A000360 Alexander
                      TN
                             R
                                         118
                                                  0.890
                                                                  0.856
2 B000575 Blunt
                       MO
                             R
                                         128
                                                  0.906
                                                                  0.787
3 B000944 Brown
                       OH
                             D
                                         128
                                                  0.258
                                                                  0.642
4 B001135 Burr
                       NC
                             R
                                                  0.893
                                                                  0.560
                                         121
5 B001230 Baldwin
                       WΙ
                             D
                                         128
                                                  0.227
                                                                  0.510
6 B001236 Boozman
                       AR
                             R
                                         129
                                                  0.915
                                                                  0.851
7 B001243 Blackburn TN
                                         131
                                                  0.885
                                                                  0.889
```

```
8 B001261 Barrasso
                                          129
                                                   0.891
                                                                    0.895
                       WY
                             R
9 B001267 Bennet
                       CO
                                                   0.273
                                                                    0.417
                             D
                                          121
10 B001277 Blumenthal CT
                             D
                                          128
                                                   0.203
                                                                    0.294
# i 112 more rows
# i 1 more variable: margin_trump <dbl>
```

This is a good occasion to show how pipes allow us to chain operations. How do we read the following code out loud? (Remember that pipes are read as "then").

```
trump_scores |>
  select(last_name, matches("agree")) |>
  rename(prop_agree = agree, prop_agree_pred = agree_pred)
```

#### # A tibble: 122 x 3 last\_name prop\_agree prop\_agree\_pred <chr> <dbl> <dbl> 1 Alexander 0.890 0.856 2 Blunt 0.906 0.787 3 Brown 0.258 0.642 4 Burr 0.893 0.560 5 Baldwin 0.227 0.510 6 Boozman 0.915 0.851 7 Blackburn 0.885 0.889 8 Barrasso 0.891 0.895 9 Bennet 0.273 0.417 10 Blumenthal 0.294 0.203 # i 112 more rows

### 2.2.3 Creating columns

It is common to want to create columns, based on existing ones. We can use mutate() to do so. For example, we could want our main variables of interest in terms of percentages instead of proportions:

#### # A tibble: 122 x 5 last\_name agree agree\_pred pct\_agree pct\_agree\_pred <chr> <dbl><dbl><dbl> <dbl> 1 Alexander 0.890 0.856 89.0 85.6 2 Blunt 0.906 78.7 0.787 90.6 3 Brown 0.258 25.8 64.2 0.642 4 Burr 0.893 0.560 89.3 56.0 5 Baldwin 0.227 0.510 22.7 51.0 6 Boozman 0.915 85.1 0.851 91.5 7 Blackburn 0.885 0.889 88.5 88.9 8 Barrasso 89.1 89.5 0.891 0.895 9 Bennet 0.273 27.3 41.7 0.417 29.4 10 Blumenthal 0.203 0.294 20.3 # i 112 more rows

We can also use multiple columns for creating a new one. For example, let's retrieve the total number of votes in which the senator agreed with Trump:

```
trump_scores |>
select(last_name, num_votes, agree) |> # select just for clarity
mutate(num_votes_agree = num_votes * agree)
```

#### # A tibble: 122 x 4

	last_name	num_votes	agree	num_votes_agree
	<chr></chr>	<dbl></dbl>	<dbl></dbl>	<dbl></dbl>
1	Alexander	118	0.890	105
2	Blunt	128	0.906	116
3	Brown	128	0.258	33
4	Burr	121	0.893	108
5	Baldwin	128	0.227	29
6	Boozman	129	0.915	118
7	Blackburn	131	0.885	116
8	Barrasso	129	0.891	115
9	Bennet	121	0.273	33.0
10	Blumenthal	128	0.203	26

# i 112 more rows

### 2.2.4 Filtering rows

Another common operation is to filter rows based on logical conditions. We can do so with the filter() function. For example, we can filter to only get Democrats:

```
trump_scores |>
  filter(party == "D")
```

```
# A tibble: 55 x 8
  bioguide last_name
                       state party num_votes agree agree_pred margin_trump
  <chr>
            <chr>
                       <chr> <chr>
                                        <dbl> <dbl>
                                                         <dbl>
                                                                      <dbl>
1 B000944
            Brown
                       OH
                             D
                                          128 0.258
                                                         0.642
                                                                      8.13
2 B001230
            Baldwin
                       WI
                             D
                                          128 0.227
                                                         0.510
                                                                      0.764
3 B001267
            Bennet
                       CO
                             D
                                          121 0.273
                                                         0.417
                                                                      -4.91
4 B001277 Blumenthal CT
                             D
                                          128 0.203
                                                         0.294
                                                                    -13.6
5 B001288 Booker
                       NJ
                             D
                                          119 0.160
                                                         0.290
                                                                    -14.1
6 C000127 Cantwell
                       WA
                             D
                                          128 0.242
                                                         0.276
                                                                    -15.5
7 C000141 Cardin
                       MD
                             D
                                          128 0.25
                                                         0.209
                                                                    -26.4
8 C000174 Carper
                       DE
                             D
                                          129 0.295
                                                                    -11.4
                                                         0.318
9 C001070
            Casey
                       PA
                             D
                                          129 0.287
                                                         0.508
                                                                       0.724
10 C001088 Coons
                       DΕ
                             D
                                          128 0.289
                                                         0.319
                                                                    -11.4
# i 45 more rows
```

Notice that == here is a *logical operator*, read as "is equal to." So our full chain of operations says the following: take trump\_scores, then filter it to get rows where party is equal to "D".

There are other logical operators:

Logical operator	Meaning
==	"is equal to"
!=	"is not equal to"
>	"is greater than"
<	"is less than"
>=	"is greater than or equal to"
<=	"is less than or equal to"
%in%	"is contained in"
&	"and" (intersection)
1	"or" (union)

Let's see a couple of other examples.

```
trump_scores |>
filter(agree > 0.5)
```

#### # A tibble: 69 x 8 bioguide last\_name state party num\_votes agree agree\_pred margin\_trump <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> 1 A000360 Alexander TN 118 0.890 0.856 26.0 R 2 B000575 Blunt MO R 128 0.906 0.787 18.6 3 B001135 Burr NC 121 0.893 0.560 3.66 R 4 B001236 Boozman AR R 129 0.915 0.851 26.9 5 B001243 Blackburn TN R 131 0.885 0.889 26.0 6 B001261 Barrasso WY 129 0.891 0.895 46.3 R 7 B001310 Braun ΙN R 44 0.909 0.713 19.2 8 C000567 Cochran 68 0.971 17.8 MSR 0.830 9 C000880 Crapo ID 125 0.904 31.8 R 0.870 -2.96 10 C001035 Collins MER 129 0.651 0.441

trump\_scores |>
filter(state %in% c("CA", "TX"))

```
# A tibble: 4 x 8
 bioguide last_name state party num_votes agree agree_pred margin_trump
  <chr>>
           <chr>
                     <chr> <chr>
                                     <dbl> <dbl>
                                                       <dbl>
                                                                    <dbl>
1 C001056
          Cornyn
                     TX
                                       129 0.922
                                                       0.659
                                                                     9.00
                           R
2 C001098 Cruz
                     TX
                                       126 0.921
                                                                     9.00
                           R
                                                       0.663
3 F000062 Feinstein CA
                                        128 0.242
                           D
                                                       0.201
                                                                   -30.1
4 H001075 Harris
                                        116 0.164
                                                                   -30.1
                     CA
                           D
                                                       0.209
```

```
trump_scores |>
filter(state == "WV" & party == "D")
```

# A tibble: 1 x 8

# i 59 more rows

```
bioguide last_name state party num_votes agree agree_pred margin_trump <chr> <chr> <chr> <chr> <chr> <chr> 0.504 0.893 42.2
```

#### i Exercise

- 1. Add a new column to the data frame, called diff\_agree, which subtracts agree and agree\_pred. How would you create abs\_diff\_agree, defined as the absolute value of diff\_agree? Your code:
- 2. Filter the data frame to only get senators for which we have information on fewer

than (or equal to) five votes. Your code:

3. Filter the data frame to only get Democrats who agreed with Trump in at least 30% of votes. Your code:

### 2.2.5 Ordering rows

The arrange() function allows us to order rows according to values. For example, let's order based on the agree variable:

```
trump_scores |>
arrange(agree)
```

```
# A tibble: 122 x 8
  bioguide last_name
                         state party num_votes agree agree_pred margin_trump
   <chr>
            <chr>
                         <chr> <chr>
                                          <dbl> <dbl>
                                                           <dbl>
                                                                         <dbl>
1 H000273 Hickenlooper CO
                               D
                                              2 0
                                                          0.0302
                                                                         -4.91
2 H000601 Hagerty
                                              2 0
                         TN
                               R
                                                          0.115
                                                                         26.0
3 L000570 Luján
                         NM
                               D
                                            186 0.124
                                                          0.243
                                                                         -8.21
                                                                        -22.5
4 G000555 Gillibrand
                         NY
                                            121 0.124
                                                          0.242
                               D
5 M001176 Merkley
                         OR
                               D
                                            129 0.155
                                                          0.323
                                                                        -11.0
6 W000817
           Warren
                               D
                                            116 0.155
                                                          0.216
                                                                        -27.2
                         MA
                                            119 0.160
7 B001288 Booker
                         NJ
                               D
                                                          0.290
                                                                        -14.1
8 S000033 Sanders
                         VT
                               D
                                            112 0.161
                                                          0.221
                                                                        -26.4
9 H001075 Harris
                         CA
                               D
                                            116 0.164
                                                          0.209
                                                                        -30.1
10 M000133 Markey
                         MA
                               D
                                            127 0.165
                                                          0.213
                                                                        -27.2
# i 112 more rows
```

Maybe we only want senators with more than a few data points. Remember that we can chain operations:

```
trump_scores |>
  filter(num_votes >= 10) |>
  arrange(agree)
```

```
# A tibble: 115 x 8
  bioguide last_name state party num_votes agree agree_pred margin_trump
   <chr>
            <chr>
                       <chr> <chr>
                                       <dbl> <dbl>
                                                         <dbl>
                                                                      <dbl>
1 L000570 Luján
                                         186 0.124
                                                         0.243
                                                                      -8.21
                       NM
                             D
2 G000555 Gillibrand NY
                             D
                                         121 0.124
                                                         0.242
                                                                     -22.5
```

3	M001176	Merkley	OR	D	129 0.155	0.323	-11.0
4	W000817	Warren	MΑ	D	116 0.155	0.216	-27.2
5	B001288	Booker	NJ	D	119 0.160	0.290	-14.1
6	S000033	Sanders	VT	D	112 0.161	0.221	-26.4
7	H001075	Harris	CA	D	116 0.164	0.209	-30.1
8	M000133	Markey	MΑ	D	127 0.165	0.213	-27.2
9	W000779	Wyden	OR	D	129 0.186	0.323	-11.0
10	B001277	${\tt Blumenthal}$	CT	D	128 0.203	0.294	-13.6
# i 105 more rows							

By default, arrange() uses increasing order (like sort()). To use decreasing order, add a minus sign:

```
trump_scores |>
  filter(num_votes >= 10) |>
  arrange(-agree)
```

```
# A tibble: 115 x 8
   bioguide last_name state party num_votes agree agree_pred margin_trump
   <chr>
            <chr>
                       <chr> <chr>
                                        <dbl> <dbl>
                                                          <dbl>
                                                                        <dbl>
 1 M001198
            Marshall
                       KS
                             R
                                          183 0.973
                                                          0.933
                                                                        20.6
2 C000567
                                                                        17.8
            Cochran
                       MS
                             R
                                           68 0.971
                                                          0.830
3 H000338
            Hatch
                       UT
                             R
                                           84 0.964
                                                          0.825
                                                                        18.1
            McSally
4 M001197
                       ΑZ
                             R
                                          136 0.949
                                                          0.562
                                                                         3.55
5 P000612 Perdue
                                          119 0.941
                                                          0.606
                                                                         5.16
                       GA
                             R
6 C001096
                       ND
                                                                        35.7
            Cramer
                             R
                                          135 0.941
                                                          0.908
7 R000307
                       KS
                                          127 0.937
                                                                        20.6
            Roberts
                             R
                                                          0.818
8 C001056
            Cornyn
                       TX
                             R
                                          129 0.922
                                                          0.659
                                                                         9.00
                                                                        35.7
                       ND
                                          129 0.922
9 H001061
            Hoeven
                             R
                                                          0.883
10 C001047
            Capito
                       WV
                             R
                                          127 0.921
                                                          0.896
                                                                        42.2
# i 105 more rows
```

You can also order rows by more than one variable. What this does is to order by the first variable, and resolve any ties by ordering by the second variable (and so forth if you have more than two ordering variables). For example, let's first order our data frame by party, and then within party order by agreement with Trump:

```
trump_scores |>
  filter(num_votes >= 10) |>
  arrange(party, agree)
```

```
# A tibble: 115 x 8
  bioguide last_name state party num_votes agree agree_pred margin_trump
  <chr>
            <chr>
                       <chr> <chr>
                                       <dbl> <dbl>
                                                        <dbl>
                                                                     dbl>
 1 L000570 Luján
                       NM
                             D
                                         186 0.124
                                                        0.243
                                                                     -8.21
2 G000555 Gillibrand NY
                             D
                                         121 0.124
                                                        0.242
                                                                    -22.5
3 M001176 Merkley
                             D
                                         129 0.155
                                                        0.323
                                                                    -11.0
                       OR
4 W000817 Warren
                       MA
                             D
                                         116 0.155
                                                        0.216
                                                                    -27.2
5 B001288 Booker
                       NJ
                             D
                                         119 0.160
                                                        0.290
                                                                    -14.1
6 S000033 Sanders
                       VT
                             D
                                         112 0.161
                                                        0.221
                                                                    -26.4
7 H001075 Harris
                       CA
                             D
                                         116 0.164
                                                        0.209
                                                                    -30.1
8 M000133 Markey
                                         127 0.165
                                                                    -27.2
                       MA
                             D
                                                        0.213
9 W000779 Wyden
                             D
                                         129 0.186
                                                        0.323
                                                                    -11.0
                       0R
                                                                    -13.6
10 B001277
           Blumenthal CT
                             D
                                         128 0.203
                                                        0.294
# i 105 more rows
```

### i Exercise

Arrange the data by diff\_pred, the difference between agreement and predicted agreement with Trump. (You should have code on how to create this variable from the last exercise). Your code:

### 2.2.6 Summarizing data

dplyr makes summarizing data a breeze using the summarize() function:

To make summaries, we can use any function that takes a vector and returns one value. Another example:

Grouped summaries allow us to disaggregate summaries according to other variables (usually categorical):

### i Exercise

Obtain the maximum absolute difference in agreement with Trump (the abs\_diff\_agree variable from before) for each party.

### 2.2.7 Overview

Function	Purpose
select()	Select columns
rename()	Rename columns
<pre>mutate()</pre>	Creating columns
filter()	Filtering rows
arrange()	Ordering rows
<pre>summarize()</pre>	Summarizing data
summarize(, .by = )	Summarizing data (by groups)

### 2.3 Visualizing data with ggplot2

ggplot2 is the package in charge of data visualization in the tidyverse. It is extremely flexible and allows us to draw bar plots, box plots, histograms, scatter plots, and many other types of plots (see examples at R Charts).

Throughout this module we will use a subset of our data frame, which only includes senators with more than a few data points:

```
trump_scores_ss <- trump_scores |>
filter(num_votes >= 10)
```

The ggplot2 syntax provides a unifying interface (the "grammar of graphics" or "gg") for drawing all different types of plots. One draws plots by adding different "layers," and the core code always includes the following:

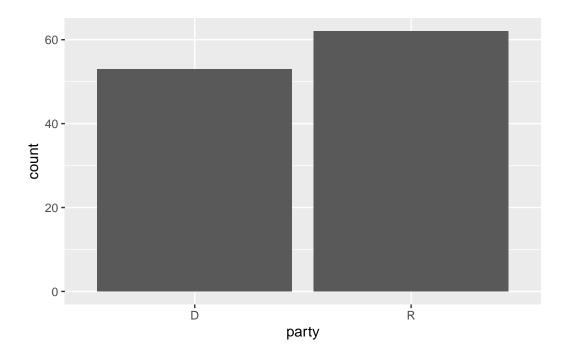
- A ggplot() command with a data = argument specifying a data frame and a mapping = aes() argument specifying "aesthetic mappings," i.e., how we want to use the columns in the data frame in the plot (for example, in the x-axis, as color, etc.).
- "geoms," such as geom\_bar() or geom\_point(), specifying what to draw on the plot.

So all ggplot2 commands will have at least three elements: data, aesthetic mappings, and geoms.

### 2.3.1 Univariate plots: categorical

Let's see an example of a bar plot with a categorical variable:

```
ggplot(data = trump_scores_ss, mapping = aes(x = party)) +
geom_bar()
```



# **?** Tip

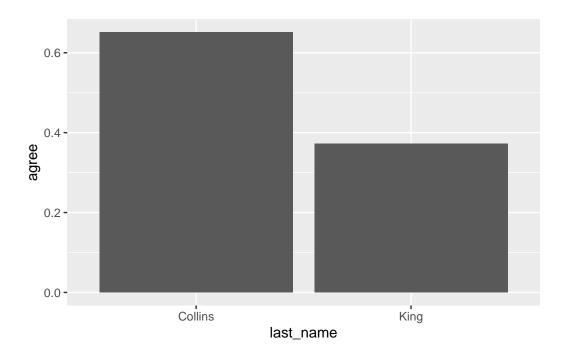
As with any other function, we can drop the argument names if we specify the argument values in order. This is common in ggplot2 code:

```
ggplot(trump_scores_ss, aes(x = party)) +
  geom_bar()
```



Notice how <code>geom\_bar()</code> automatically computes the number of observations in each category for us. Sometimes we want to use numbers in our data frame as part of a bar plot. Here we can use the <code>geom\_col()</code> geom specifying both <code>x</code> and <code>y</code> aesthetic mappings, in which is sometimes called a "column plot:"

```
ggplot(trump_scores_ss |> filter(state == "ME"),
    aes(x = last_name, y = agree)) +
    geom_col()
```

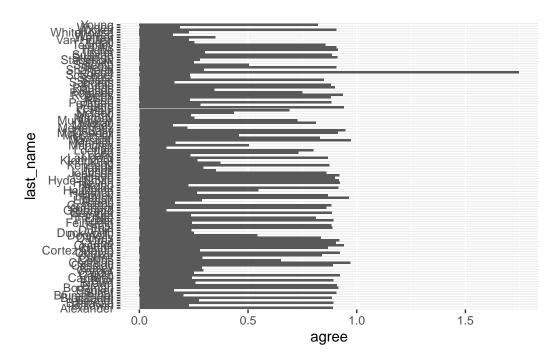


### i Exercise

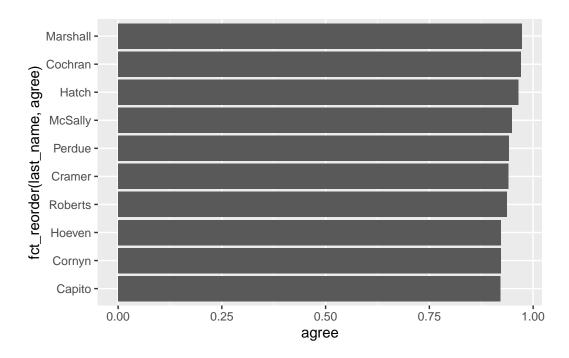
Draw a column plot with the agreement with Trump of Bernie Sanders and Ted Cruz. What happens if you use  $last_name$  as the y aesthetic mapping and agree in the x aesthetic mapping? Your code:

A common use of geom\_col() is to create "ranking plots." For example, who are the senators with highest agreement with Trump? We can start with something like this:

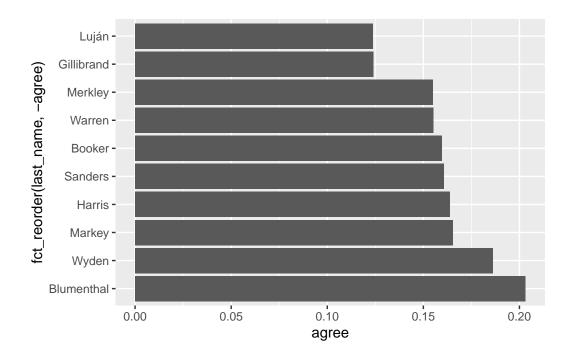
```
ggplot(trump_scores_ss,
    aes(x = agree, y = last_name)) +
    geom_col()
```



We might want to (1) select the top 10 observations and (2) order the bars according to the agree values. We can do these operations with slice\_max() and fct\_reorder(), as shown below:



We can also plot the senators with the *lowest* agreement with Trump using slice\_min() and fct\_reorder() with a minus sign in the ordering variable:

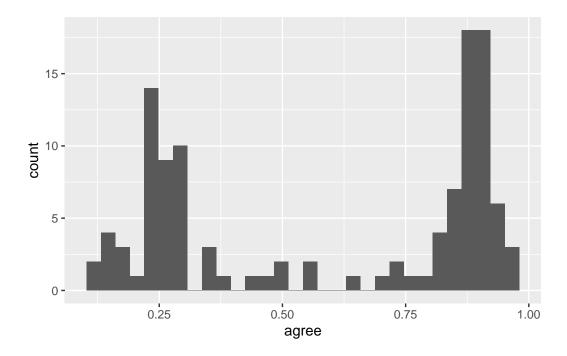


## 2.3.2 Univariate plots: numerical

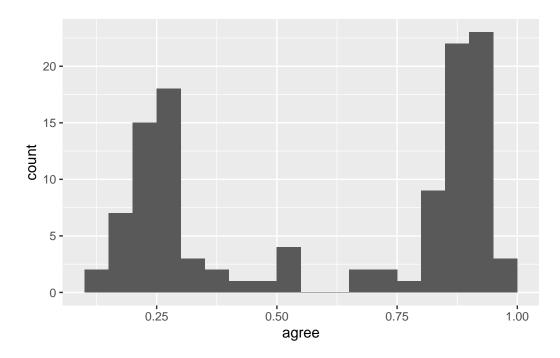
We can draw a histogram with geom\_histogram():

```
ggplot(trump_scores_ss, aes(x = agree)) +
  geom_histogram()
```

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

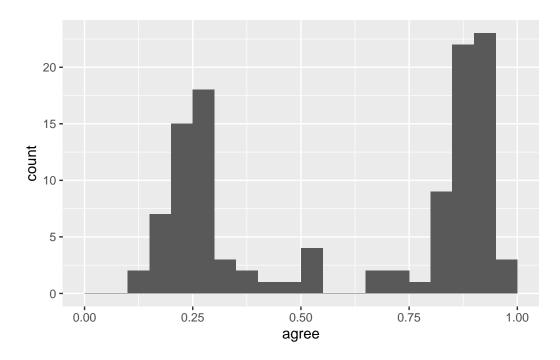


Notice the warning message above. It's telling us that, by default, geom\_histogram() will draw 30 bins. Sometimes we want to modify this behavior. The following code has some common options for geom\_histogram() and their explanations:



Sometimes we want to manually alter a scale. This is accomplished with the scale\_\*() family of ggplot2 functions. Here we use the scale\_x\_continuous() function to make the x-axis go from 0 to 1:

```
ggplot(trump_scores_ss, aes(x = agree)) +
  geom_histogram(binwidth = 0.05, boundary = 0, closed = "left") +
  scale_x_continuous(limits = c(0, 1))
```



Adding the fill aesthetic mapping to a histogram will divide it according to a categorical variable. This is actually a bivariate plot!

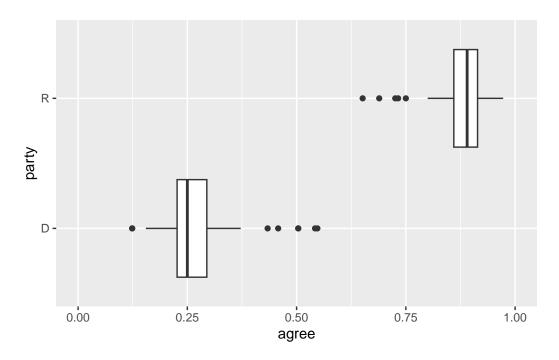
```
ggplot(trump_scores_ss, aes(x = agree, fill = party)) +
  geom_histogram(binwidth = 0.05, boundary = 0, closed = "left") +
  scale_x_continuous(limits = c(0, 1)) +
  # change default colors:
  scale_fill_manual(values = c("D" = "blue", "R" = "red"))
```



## 2.3.3 Bivariate plots

Another common bivariate plot for categorical and numerical variables is the grouped box plot:

```
ggplot(trump_scores_ss, aes(x = agree, y = party)) +
  geom_boxplot() +
  scale_x_continuous(limits = c(0, 1)) # same change as before
```



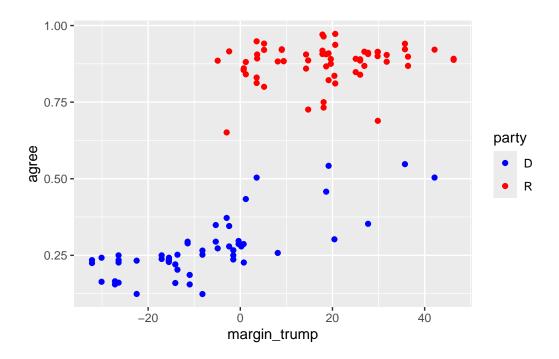
For bivariate plots of numerical variables, scatter plots are made with geom\_point():

```
ggplot(trump_scores_ss, aes(x = margin_trump, y = agree)) +
  geom_point()
```



We can add the color aesthetic mapping to add a third variable:

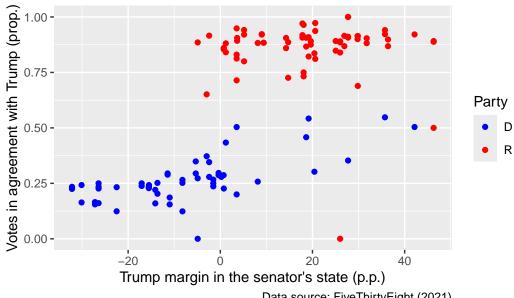
```
ggplot(trump_scores_ss, aes(x = margin_trump, y = agree, color = party)) +
  geom_point() +
  scale_color_manual(values = c("D" = "blue", "R" = "red"))
```



Let's finish our plot with the labs() function, which allows us to add labels to our aesthetic mappings, as well as titles and notes:

```
ggplot(trump_scores, aes(x = margin_trump, y = agree, color = party)) +
  geom_point() +
  scale_color_manual(values = c("D" = "blue", "R" = "red")) +
  labs(x = "Trump margin in the senator's state (p.p.)",
        y = "Votes in agreement with Trump (prop.)",
        color = "Party",
        title = "Relationship between Trump margins and senators' votes",
        caption = "Data source: FiveThirtyEight (2021)")
```

# Relationship between Trump margins and senators' votes



Data source: FiveThirtyEight (2021)

We will review a few more customization options, including text labels and facets, in a subsequent module.

# 3 Functions

### 3.1 Basics

#### 3.1.1 What is a function?

Informally, a function is anything that takes input(s) and gives one defined output. There are always three main parts:

- The input (x values, or each value in the domain)
- The relationship of interest
- The output (y values, or a unique value in the range)

#### Note

" $f(x) = \dots$  is the classic notation for writing a function, but we can also use" $y = \dots$ ". This is because y is a function of x, so y = f(x).

Let's take a look at an example and break down the structure:

$$f(x) = 3x + 4$$

- x is the *input* (some value) that the function takes.
- For any x, we multiply by three and add 4, which is the *relationship*.
- Finally, f(x) or y is the unique result, or the *output*.

The most common name to give a function is, predictably, "f", but we can have other names such as "g" or "h". The choice is yours.

#### Important

When reading out loud, we say "[name of function] of x equals [relationship]. For example,  $f(x) = x^2$  is referred to as for x equals x squared."



Figure 3.1: Function machine. Source: Bill Bailey on Wikimedia Commons.

#### 3.1.2 Vertical line test

## **i** Exercise

When graphed, vertical lines cannot touch functions at more than one point. Why? Which of the following represent functions?

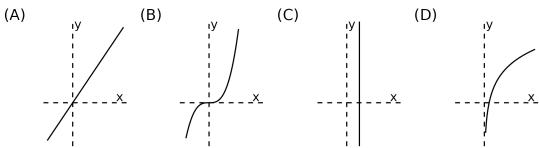




Figure 3.2: Vertical line test: examples.

### 3.2 Functions in R

Often we need to create our own functions in R. To build them: we use the keyword function alongside the following syntax: function\_name <- function(argumentnames){ operation}

- function\_name: the name of the function, that will be stored as an object in the R environment. Make the name concise and memorable!
- function(argumentnames): the inputs of the function.
- { operation }: a set of commands that are run in a predefined order every time we call the function.

For example, we can create a function that multiplies a number by 2:

```
mult_by_two <- function(x){x * 2}</pre>
```

```
mult_by_two(x = 5) # we can also omit the argument name (x =)
```

[1] 10

If the function body works for vectors, our custom function will do too:

```
mult_by_two(1:10)
```

```
[1] 2 4 6 8 10 12 14 16 18 20
```

We can also automate more complicated tasks such as calculating the area of a circle from its radius:

```
circ_area_r <- function(r){
   pi * r ^ 2
}
circ_area_r(r = 3)</pre>
```

[1] 28.27433

#### i Exercise

Create a function that calculates the area of a circle *from its diameter*. So your\_function(d = 6) should yield the same result as the example above. Your code:

Functions can take more than one argument/input. In a silly example, let's generalize our first function:

```
mult_by <- function(x, mult){x * mult}</pre>
```

```
mult_by(x = 1:5, mult = 10)
```

[1] 10 20 30 40 50

```
mult_by(1:5, mult = 10)

[1] 10 20 30 40 50

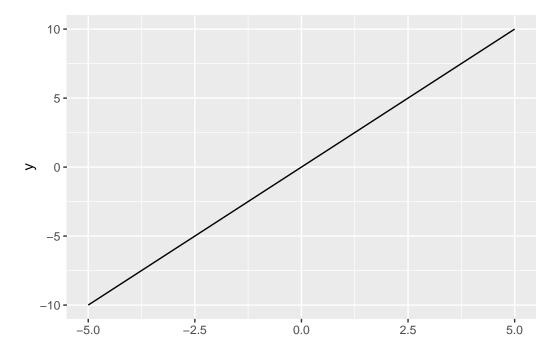
mult_by(1:5, 10)

[1] 10 20 30 40 50

To graph a function, we'll use our friend ggplot2 and stat_function():
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr 1.1.4
                  v readr
                               2.1.5
v forcats 1.0.0
                  v stringr
                               1.5.1
v ggplot2 3.5.2 v tibble
                               3.3.0
                  v tidyr
v lubridate 1.9.4
                               1.3.1
v purrr
           1.1.0
-- Conflicts ------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()
                masks stats::lag()
i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/</a>) to force all conflicts to become
```



User-defined functions have endless possibilities! We encourage you to get creative and try to automate new tasks when possible, especially if they are repetitive.



# 3.3 Common types of functions

#### 3.3.1 Linear functions

$$y = mx + b$$

Linear functions are those whose graph is a straight line (in two dimensions).

- m is the slope, or the rate of change (common interpretation: for every one unit increase in x, y increases m units).
- b is the y intercept, or the constant term (the value of y when x = 0).

Below is a graph of the function y = 3x + 4:

ggplot() + stat\_function(fun = function(x){3 \* x + 4}, # we don't need to create an object 
$$xlim = c(-5, 5)$$
)



### 3.3.2 Quadratic functions

$$y = ax^2 + bx + c$$

Quadratic functions take "U" forms. If a is positive, it is a regular "U" shape. If a is negative, it is an "inverted U" shape.

Note that  $x^2$  always returns positive values (or zero).

Below is a graph of the function  $y = x^2$ :



## **i** Exercise

Social scientists commonly use linear or quadratic functions as theoretical simplifications of social phenomena. Can you give any examples?

#### i Exercise

Graph the function  $y = x^2 + 2x - 10$ , i.e., a quadratic function with a = 1, b = 2, and c = -10.

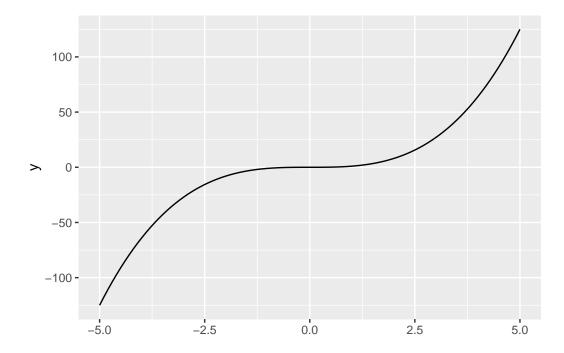
Next, try switching up these values and the xlim = argument. How do they each alter the function (and plot)?

#### 3.3.3 Cubic functions

$$y = ax^3 + bx^2 + cx + d$$

These lines (generally) have two curves (inflection points).

Below is a graph of the function  $y = x^3$ :



#### i Exercise

We'll briefly introduce Desmos, an online graphing calculator. Use Desmos to graph the following function  $y = 1x^3 + 1x^2 + 1x + 1$ . What happens when you change the a, b, c, and d parameters?

#### 3.3.4 Polynomial functions

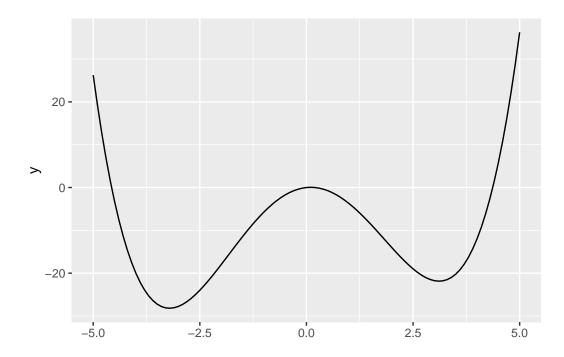
$$y = ax^n + bx^{n-1} + \dots + c$$

These functions have (a maximum of) n-1 changes in direction (turning points). They also have (a maximum of) n x-intercepts.

High-order polynomials can be made arbitrarily precise!

Below is a graph of the function  $y = \frac{1}{4}x^4 - 5x^2 + x$ .

```
ggplot() +
stat_function(fun = function(x){1/4 * x ^4 - 5 * x ^2 + x},
xlim = c(-5, 5))
```



## 3.3.5 Exponential functions

$$y = ab^x$$

Here our input (x), is the exponent.

Below is a graph of the function  $y = 2^x$ :



### i Exercise

Exponential growth appears quite frequently social science theories. Which variables can be theorized to have exponential growth over time?

# 3.4 Logarithms and exponents

## 3.4.1 Logarithms

Logarithms are the opposite/inverse of exponents. They ask how many times you must raise the base to get x.

So  $log_a(b)=x$  is asking "a raised to what power x gives b?" For example,  $\log_3(81)=4$  because



Warning

Logarithms are undefined if the base is  $\leq 0$  (at least in the real numbers).

### 3.4.2 Relationships

$$log_a x = b$$

then,

$$a^{log_a x} = a^b$$

and

$$x = a^b$$

### 3.4.3 Basic rules

- Change of Base rule:  $\frac{\log_x n}{\log_x m} = \log_m n$
- Product Rule:  $\log_x(ab) = \log_x a + \log_x b$
- Quotient Rule:  $\log_x\left(\frac{a}{b}\right) = \log_x a \log_x b$
- Power Rule:  $\log_x a^b = b \log_x a$
- Logarithm of 1:  $\log_x 1 = 0$
- Logarithm of the Base:  $log_x x = 1$
- Exponential Identity:  $m^{\log_m(a)} = a$

## 3.4.4 Natural logarithms

- We most often use natural logarithms for our purposes.
- This means  $log_e(x)$ , which is usually written as ln(x).

### ! Important

 $e \approx 2.7183$ .

• ln(x) and its exponent opposite,  $e^x$ , have nice properties when we perform calculus.

#### **3.4.5** Illustration of e

Imagine you invest \$1 in a bank and receive 100% interest for one year, and the bank pays you back once a year:

$$(1+1)^1 = 2$$

.

When it pays you twice a year with compound interest:

$$(1+1/2)^2 = 2.25$$

If it pays you three times a year:

$$(1+1/3)^3 = 2.37...$$

What will happen when the bank pays you once a month? Once a day?

$$(1+\frac{1}{n})^n$$

However, there is limit to what you can get.

$$\lim_{n \to \infty} (1 + \frac{1}{n})^n = 2.7183... = e$$

For any interest rate k and number of times the bank pays you t:

$$\lim_{n\to\infty}(1+\frac{k}{n})^{nt}=e^{kt}$$

e is important for defining exponential growth. Since  $ln(e^x) = x$ , the natural logarithm helps us turn exponential functions into linear ones.

#### i Exercise

Solve the problems below, simplifying as much as you can.

$$log_{10}(1000)$$

$$log_2(\frac{8}{32})$$

$$10^{log_{10}(300)}$$

 $ln(e^2)$ ln(5e)

## 3.4.6 Logarithms in R

By default, R's log() function computes natural logarithms:

```
log(100)
```

[1] 4.60517

We can change this behavior with the base = argument:

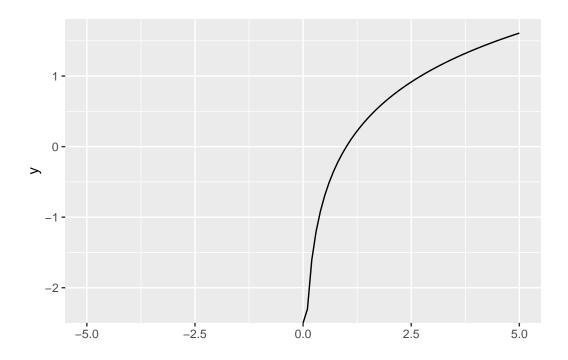
```
log(100, base = 10)
```

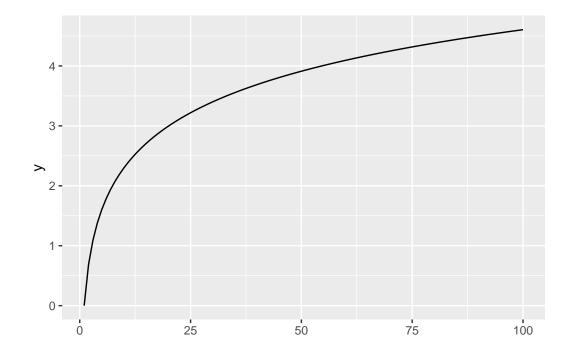
[1] 2

We can also plot logarithms. Remember that  $ln(x) \forall x < 0$  is undefined (at least in the real numbers), and ggplot2 displays a nice warning letting us know!

Warning in log(x): NaNs produced

Warning: Removed 50 rows containing missing values or values outside the scale range (`geom\_function()`).





# 3.5 Composite functions (functions of functions)

Functions can take other functions as inputs, e.g., f(g(x)). This means that the outside function takes the output of the inside function as its input.

Say we have the exterior function

$$f(x) = x^2$$

and the interior function

$$g(x) = x - 3$$

Then if we want f(g(x)), we would subtract 3 from any input, and then square the result or

$$f(g(x)) = (x-3)^2$$



⚠ Warning

We write this as  $(x-3)^2$ , not  $x^2-3!$ 

R can handle this just fine:

```
f \leftarrow function(x)\{x^2\}
g \leftarrow function(x)\{x - 3\}
```

f(g(5))

[1] 4

Here we can also use pipes to make this code more readable (imagine if we were chaining multiple functions...). Remember that pipes can be inserted with the Cmd/Ctrl + Shift + M shortcut.

```
# compute g(5), THEN f() of that
g(5) |> f()
```

[1] 4

# **i** Exercise

Compute g(f(5)) using the definitions above. First do it manually, and then check your answer with R.

# 4 Calculus

In this section we'll focus on three big ideas from calculus: derivatives, optimization, and integrals.

#### 4.1 Derivatives

Derivatives are about (instantaneous) rate of change.

"In the fall of 1972 President Nixon announced that the rate of increase of inflation was decreasing. This was the first time a sitting president used the third derivative to advance his case for reelection" (Rossi 1996)

Let's dissect what Nixon might have said:

Inflation's [first derivative, of prices] rate of increase [second derivative] is going down [third derivative].

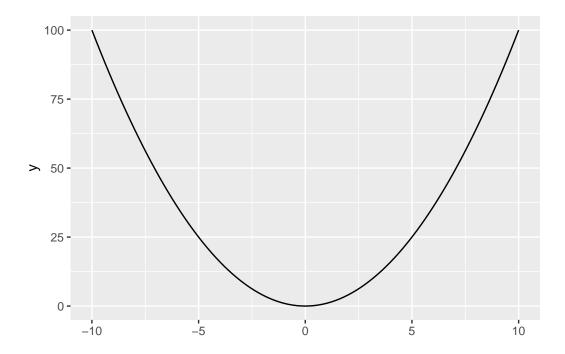
A more graphical way to think about a derivatives is as a *slope*. Let's consider a linear function of the form y = 2x:



We can imagine any political variables in the x- and y-axes. What is the rate of change? In other words, what is the derivative? Remember that we can calculate the slope with:

$$m = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Now consider another slightly more complicated function, a quadratic one,  $y = x^2$ :



What happens when we apply our slope function?

## **i** Exercise

- 1) Use the slope formula to calculate the rate of change between 5 and 6.
- 2) Use the slope formula to calculate the rate of change between 5 and 5.5.
- 3) Use the slope formula to calculate the rate of change between 5 and 5.1.

Takeaway: here the derivative depends on the value of x. It is actually 2x.

Differential calculus is about finding these derivatives in a more straightforward manner! We can generalize our slope formula as follows:

$$m = \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}$$

The point is that when  $\Delta x$  is arbitrarily small, we'll get our rate of change. Formalizing this:

$$\lim_{\Delta x \rightarrow 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x} = \frac{d}{dx} f(x) = \frac{dy}{dx} = f'(x)$$

A few points on notation:

- $\frac{d}{dx}f(x)$  is read "The derivative of f of x with respect to x."
  - The variable with respect to which we're differentiating is the one that appears in the bottom (in the case above, this is x).

### ⚠ Warning

While the above looks like a fraction, it's really not. Do not try to cancel out the ds!

• f'(x) (read: "f prime x") is the derivative of f(x). This is a more compact form to refer to derivatives when you have defined f(x) elsewhere.

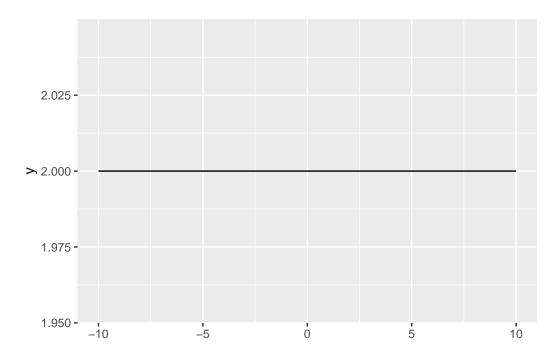
#### 4.1.1 Rules of differentiation

How to compute derivatives? Sometimes you can try a bunch of numbers and get at the answer. Sometimes you can use the limit-based formula above, if you know a few properties of limits. But in most cases you will either use software (more on this later) or the **rules of differentiation**, which we will cover now.

Constant rule: (c)' = 0.

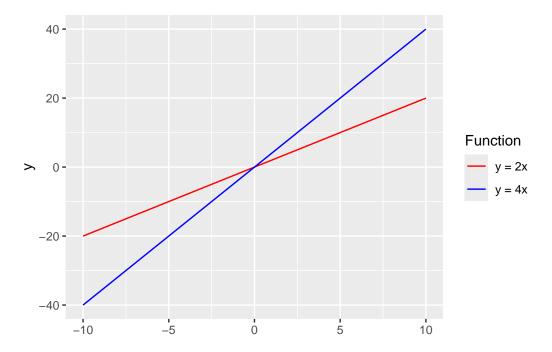
There is no change in a constant:

```
ggplot() +
stat_function(fun = function(x){2}, xlim = c(-10, 10))
```



Coefficient rule:  $(c \cdot f(x))' = c \cdot f'(x)$ .

```
ggplot() +
  stat_function(fun = function(x){2 * x}, xlim = c(-10, 10), aes(color = "y = 2x")) +
  stat_function(fun = function(x){4 * x}, xlim = c(-10, 10), aes(color = "y = 4x")) +
  scale_color_manual("Function", values = c("red", "blue"))
```



Sum/difference rule:  $(f(x) \pm g(x))' = f'(x) \pm g'(x)$ .

The two rules above give us that the derivative is a linear operator.

**Power rule:**  $(x^n)' = nx^{(n-1)}$ 

Remember when we wanted to calculate the derivative of  $y=x^2$  above? We can use the power rule, with n=2:  $nx^{(n-1)}=2x^{(2-1)}=2x$ . Let's try out  $\frac{d}{dx}4x^3$  and  $\frac{d}{dx}(x^2+2x)$  on the board.

# i Exercise

Use the differentiation rules we have covered so far to calculate the derivatives of y with respect to x of the following functions:

- 1)  $y = 2x^2 + 10$ 2)  $y = 5x^4 \frac{2}{3}x^3$ 3)  $y = 9\sqrt{x}$ 4)  $y = \frac{4}{x^2}$ 5)  $y = ax^3 + b$ , where a and b are constants.
- 6)  $y = \frac{2w}{5}$

Exponent and logarithm rules:

$$(c^x)' = c^x \cdot ln(c), \quad \forall x > 0$$
  
 $(e^x)' = e^x$ 

$$\begin{split} (log_a(x))' &= \frac{1}{x \cdot ln(a)}, \quad \forall x > 0 \\ (ln(x))' &= \frac{1}{x}, \qquad \quad \forall x > 0 \end{split}$$

We saw previously how Euler's number (e) arises from compound interest. The properties above make it very useful in a lot of calculus applications!

### i Exercise

Compute the following:

1)  $\frac{d}{dx}(10e^x)$ 2)  $\frac{d}{dx}(ln(x) - \frac{e^2}{3})$ 

Now we'll get to a couple of more advanced (and powerful) rules.

• Product rule: (f(x)g(x))' = f'(x)g(x) + g'(x)f(x)

• Quotient rule:  $(\frac{f(x)}{g(x)})' = \frac{f'(x)g(x) + g'(x)f(x)}{[g(x)]^2}$ 

• Chain rule:  $(f(g(x))' = f'(g(x)) \cdot g'(x)$ 

Let's calculate  $\frac{d}{dx}(3\cdot ln(x)\cdot x^2)$  on the board.

Let's compute  $\frac{d}{dx}(e^{x^2})$  on the board.

# i Exercise

Use the differentiation rules we have covered so far to calculate the derivatives of y with respect to x of the following functions:

### 4.1.2 Higher-order derivatives

We saw how politicians can refer to higher-order derivatives. To compute them, you simply "pass the outputs," starting from the lowest order and going up.

The second derivative tells us whether the slope of a function is increasing, decreasing, or staying the same at any point x on the function's domain. For example, when driving a car:

- f(x) = distance traveled at time x
- f'(x) = speed at time x
- f''(x) = acceleration at time x

Let's compute the following second derivative:

$$f''(x^4) = \frac{d^2(x^4)}{dx^2}$$

- First, we take the first derivative:  $f'(x^4) = 4x^3$
- Then we use that output to take the second derivative:  $f''(x^4) = f'(4x^3) = 12x^2$
- We can keep going... for example, the third derivative:

$$f'''(x^4) = f'(12x^2) = 24x$$

#### i Exercise

Compute the following:

- 1)  $\frac{d^3}{dx^3}(x^5)$
- 2)  $f''(4x^{3/2})$
- 3)  $f''(4 \cdot ln(x))$

#### 4.1.3 Partial derivatives

For a function f(x, z), we might want to know how the function changes with respect to x. We call this a partial derivative:

$$\frac{\partial}{\partial_x} f(x, z) = \frac{\partial_y}{\partial_x} = \partial_x f$$

To obtain a partial derivative, we treat all other variables as constants and take the derivative with respect to the variable of interest (here x). For example:

$$y = f(x, z) = xz$$
$$\frac{\partial_y}{\partial_x} = z$$

What is 
$$\frac{\partial_y}{\partial_z}$$
?

Let's solve  $\frac{\partial (x^2y+xy^2-x)}{\partial x}$  and  $\frac{\partial (x^2y+xy^2-x)}{\partial u}$  on the board.

# Example

Let's say that y is how much I like a movie, d is how many dogs a movie has, and e is how many explosions a movie has. I claim that how much I like a movie can be expressed by a function of the type y = f(d, e). Evaluate the following situations:

- 1. I like dogs and I don't care about action. So I believe that the true relationship is  $y = f(d, e) = 3 \cdot d$ . What is  $\frac{\partial_y}{\partial_d}$ , and how can we interpret it?
- 2. I like dogs and I like action. So I believe that the true relationship is y = f(d, e) = $3 \cdot d + 1 \cdot e$ . What is  $\frac{\partial_y}{\partial x}$ , and how can we interpret it?
- 3. I like dogs and I like action. But I definitely don't like them together—I don't want the dogs to be in danger! So I believe that the true relationship is y = f(d, e) = $3 \cdot d + 1 \cdot e - 10 \cdot d \cdot e$ . What is  $\frac{\partial_y}{\partial_x}$ , and how can we interpret it?

# i Exercise

Take the partial derivative with respect to x and with respect to z of the following functions. What would the notation for each look like?

- 1) y = 3xz x2)  $x^3 + z^3 + x^4z^4$ 3)  $e^{xz}$

# 4.1.4 Differentiability of functions

Not all functions are differentiable at every point of their domains!

An important concept here is whether functions are **continuous** at a point:

- Informally: A function is continuous at a point if its graph has no holes or breaks at that point
- Formally: A function is continuous at a point a if:  $\lim_{x\to a} f(x) = f(a)$

When is a function **differentiable** at a point?

- If a function is differentiable at a point, it is also continuous at that point.
- If a function is continuous at a point, it is *not* necessarily differentiable at that point.
  - Impossible to calculate derivative at sharp turns, cusps, or vertical tangents.

```
\begin{split} & \text{ggplot()} + \\ & \text{stat\_function(fun = function(x)\{abs(x) + 2\}, xlim = c(-4, 4),} \\ & \text{aes(color = "y = |x| + 2"))} + \\ & \text{stat\_function(fun = function(x)\{sqrt(abs(x)) + 1\}, xlim = c(-4, 4),} \\ & \text{aes(color = "y = }\sqrt{(|x|) + 1"))} + \\ & \text{stat\_function(fun = function(x)\{sign(x) * abs(x)^(1 / 3)\}, xlim = c(-4, 4),} \\ & \text{aes(color = "y = }\sqrt{x"))} + \\ & \text{scale\_colour\_manual("Function", values = c("red", "blue", "black"))} + \\ & \text{labs(title = "Examples of functions that are not differentiable at x=0")} \end{split}
```

# Examples of functions that are not differentiable at x=0



Informally, functions need to be continuous and reasonably smooth to be differentiable.

### 4.1.5 How do computers calculate derivatives?

In quite a few statistics and machine learning problems, computers need to compute derivatives of arbitrarily complex functions, perhaps millions of times. How do they do it? (see Baydin et al. 2018 for discussion of these three approaches)

- Symbolic differentiation: automatically combine the rules of differentiation (power rule, product rule, etc.). It is what math solvers use, e.g., WolframAlpha or (presumably) Symbolab.
- Numerical differentiation: infer the derivative by computing the function at different sample values (like we did with  $y = x^2$  before. This is what, for example, R's optim() function does behind the scenes.

• Automatic differentiation: track how every function is constructed from (differentiable) elementary computer operations (e.g., binary arithmetic), and get the result using the chain rule. Implemented in the torch R package, the TensorFlow, PyTorch, and JAX Python libraries, and the ReverseDiff.jl and Zygote.jl Julia packages.

Figure 4.1: An example of computing the gradient of an esoteric function using Zygote.jl (from its documentation)

# 4.2 Optimization

Optimization allows us to find the minimum or maximum values (or extrema) a function takes. It has many applications in the social sciences:

- Formal theory: utility maximization, continuous choices
- Ordinary Least Squares (OLS): Focuses on *minimizing* the squared errors between observed data and model-estimated values
- Maximum Likelihood Estimation (MLE): Focuses on *maximizing* a likelihood function, given observed values.

#### 4.2.1 Extrema

On extrema: informally, a maximum is just the highest value a function takes, and a minimum is the lowest value.

In some situations, it can be easy to identify extrema intuitively by looking at a graph of the function.

- Maxima are high points ("peaks")
- Minima are low points ("valleys")

We can use derivatives (rates of change!) to get at extrema.

# 4.2.2 Critical points and the First-Order Condition

At critical points (or stationary points), the derivative is zero or fails to exist. At these, the function has usually reached a (local) maximum or minimum.

- At a maximum, the function must be increasing before the point and decreasing after it.
- At a minimum, the function must be decreasing before the point and increasing after it.



#### Warning

Local extrema occur at critical points, but not all critical points are extrema. For instance, sometimes the graph is changing between concave and convex ("inflection points"). Or sometimes the function is not differentiable at that point for other reasons.

We can find the local maxima and/or minima of a function by taking the derivative, setting it equal to zero, and solving for x (or whatever variable). This gives us the First-Order Condition (FOC).

$$FOC: f'(x) = 0$$

#### 4.2.3 Second-Order Condition

Notice that after this we only know that there is a critical point. **BUT** we don't know if we've found a maximum or minimum, or even if we've found an extremum.

To determine whether a we are seeing a (local) maximum or minimum, we can use the **Second Derivative Test:** 

• Start by identifying f''(x)

- Substitute in the stationary points  $(x^*)$  identified from the FOC.
  - $-f''(x^*) > 0$  we have a local minimum
  - $-f''(x^*) < 0$  we have a local maximum
  - $-f''(x^*) = 0$  we (may) have an inflection point need to calculate higher-order derivatives (don't worry about this now)

Collectively these give us the **Second-Order Condition** (SOC).

Let's do this procedure and obtain the FOC and SOC for  $y = \frac{1}{2}x^3 + 3x^2 - 2$  on the board. What do we learn? Compare this with the plot of the function on Desmos.

### 4.2.4 Local or global extrema?

Now when it comes to knowing whether extrema are local or global:

- Here we use the **Extreme value theorem**, which states that if a real-valued function is continuous on a closed and bounded (i.e., finite) interval, the function must have a global minimum and a global minimum on that interval at least once. Importantly, in this situation the global extrema exist, and **they are either at the local extrema or at the boundaries** (where we cannot even find critical points).
- So to find the minimum/maximum on some interval, compare the local min/max to the value of the function at the interval's endpoints. So, e.g., if the interval is  $(-\infty, +\infty)$ , check the function's limits as it approaches  $-\infty$  and  $+\infty$ .

Let's try this last step for our example above,  $y = \frac{1}{2}x^3 + 3x^2 - 2$ , to get the global extrema in the entire domain.

# i Exercise

Identify the global extrema of the function  $\frac{x^3}{3} - \frac{3}{2}x^2 - 10x$  in the interval [-6, 6].

# 4.3 Integrals

Informally, we can think of integrals as the flip side of derivatives.

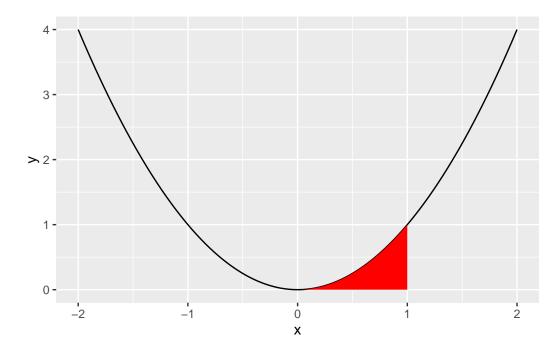
We can motivate integrals as a way of finding the area under a curve. Sometimes finding the area is easy. What's the area under the curve between x = -1 and x = 1 for this function?

$$f(x) = \begin{cases} \frac{1}{3} & \text{for } x \in [0, 3] \\ 0 & \text{otherwise} \end{cases}$$

Normally, finding the area under a curve is much harder. But this is basically the question behind integration.

### 4.3.1 Integrals are about infinitesimals too

Let's say we have a function  $y = x^2$  And we want to find the area under the curve from x = 0 to x = 1. How would we do this?



One way to approximate this area is by drawing narrow rectangles that cover the area in red. Let's draw this on the board.

Our approximation is rough, but it gets better and better the narrower the rectangles are:

$$Area = \lim_{\Delta x \to 0} \sum_{i}^{n} f(x) \cdot \Delta x$$

, where  $\Delta x$  is the width of the rectangles and n is their number.

This is actually one way to define the **definite integral**,  $\int_a^b f(x)dx$  (also known as the Riemann integral). We'll learn how to compute these in a few moments.

#### 4.3.2 Indefinite integrals as antiderivatives

The **indefinite integral**, also known as the **antiderivative**, F(x) is the inverse of the function f'(x).

$$F(x) = \int f(x) \ dx$$

This means if you take the derivative of F(x), you wind up back at f(x).

$$F' = f$$
 or  $\frac{dF(x)}{dx} = f(x)$ 

For example, what is the antiderivative for a constant function f(x) = 1? Is there just one? (this example comes from Moore and Siegel, 2013, p. 137).

This process is called *anti-differentiation*. We can use this concept to help us solve definite integrals!

### 4.3.3 Solving definite integrals

One way to calculate definite integrals, known as the "fundamental theorem of calculus," is shown below:

$$\int_{a}^{b} f(x) dx = F(b) - F(a) = F(x) \bigg|_{a}^{b}$$

First we determine the antiderivative (indefinite integral) of f(x) (and represent it F(x)), substitute the upper limit first and then the lower limit one by one, and subtract the results in order.

# ⚠ Warning

C in the following definitions and rules is the called the "constant of integration." We need to add it when we define all antiderivatives (integrals) of a function because the anti-derivative "undoes" the derivative.

Remember that the derivative of any constant is zero. So if we find an integral F(x) whose derivative is f(x), adding (or subtracting) any constant will give us another integral F(x) + C whose derivative is also f(x).

# 4.3.4 Rules of integration

Many of the rules of integetration have counterparts in differentiation.

Coefficient rule: 
$$\int cf(x) dx = c \int f(x) dx$$

Sum/difference rule: 
$$\int (f(x) \pm g(x)) dx = \int f(x) dx \pm \int g(x) dx$$

Constant rule: 
$$\int c dx = cx + C$$

Power rule: 
$$\int x^n dx = \frac{x^{n+1}}{n+1} + C$$
  $\forall n \neq -1$ 

Reciprocal rule: 
$$\int \frac{1}{x} dx = \ln(x) + C$$

Exponent and logarithm rules:

$$\int e^x dx = e^x + C$$

$$\int c^x dx = \frac{c^x}{\ln(c)} + C$$

$$\int \ln(x) dx = x \cdot \ln(x) - x + C$$

$$\int \log_c(x) dx = \frac{x \cdot \log_c(x) - x}{\log_c(x)} + C$$

The final two rules are analog to the product rule and the chain rule:

Integration by parts: 
$$\int f(x)g'(x) dx = f(x)g(x) - \int f'(x)g(x) dx$$

Integration by substitution:

1. Have 
$$\int f(g(x))g'(x) dx$$

2. Set 
$$u=g(x)$$

3. Compute 
$$\int f(u) du$$

4. Replace u for 
$$g(x)$$

Let's do an example on the board:  $\int e^{x^2} 2x \, dx$ .

# 4.3.5 Solving the problem

Remember our function  $y = x^2$  and our goal of finding the area under the curve from x = 0to x = 1. We can describe this problem as  $\int_0^1 x^2 dx$ 

Find the indefinite integral, F(x):

$$\int x^2 dx = \frac{x^3}{3} + C$$

Now we'll use the fundamental theory of calculus. Evaluate at our lowest and highest points, F(0) and F(1):

• 
$$F(0) = 0$$

• 
$$F(1) = \frac{1}{3}$$

• Technically 0 + C and  $\frac{1}{3} + C$ , but the C's will fall out in the next step

Calculate F(1) - F(0)

$$\frac{1}{3} - 0 = \frac{1}{3}$$

# i Exercise

Solve the following indefinite integrals:  $1. \int x^2 \ dx$   $2. \int 3x^2 \ dx$   $3. \int x \ dx$ 

1. 
$$\int x^2 dx$$

$$2. \int 3x^2 \ dx$$

3. 
$$\int x \ dx$$

4. 
$$\int (3x^2 + 2x - 7) dx$$
  
5.  $\int \frac{2}{x} dx$ 

$$5. \int \frac{2}{x} dx$$

And solve the following definite integrals:

$$1. \int_1^7 x^2 \ dx$$

2. 
$$\int_{1}^{10} 3x^2 dx$$

$$3. \int_{7}^{7} x \ dx$$

$$5. \int_1^e \frac{2}{x} dx$$

# 5 Matrices

Matrices are rectangular collections of numbers. In this module we will introduce them and review some basic operators, to then introduce a sneak peek of why matrices are useful (and cool).

# 5.1 Introduction

#### 5.1.1 Scalars

One number (for example, 12) is referred to as a scalar.

$$a = 12$$

#### 5.1.2 Vectors

We can put several scalars together to make a vector. Here is an example:

$$\vec{b} = \begin{bmatrix} 12\\14\\15 \end{bmatrix}$$

Since this is a column of numbers, we cleverly refer to it as a column vector.

Here is another example of a vector, this time represented as a row vector:

$$\vec{c} = \begin{bmatrix} 12 & 14 & 15 \end{bmatrix}$$

Column vectors are possibly more common and useful, but we sometimes write things down using row vectors to

Vectors are fairly easy to construct in R. As we saw before, we can use the c() function to combine elements:

```
c(5, 25, -2, 1)
```

[1] 5 25 -2 1



#### Warning

Remember that the code above does not *create* any objects. To do so, you'd need to use the assignment operator (<-):

```
vector_example <- c(5, 25, -2, 1)
vector_example
```

Or we can also create vectors from sequences with the : operator or the seq() function:

10:20

[1] 10 11 12 13 14 15 16 17 18 19 20

```
seq(from = 3, to = 27, by = 3)
```

[1] 3 6 9 12 15 18 21 24 27

# 5.2 Operators

#### 5.2.1 Summation

The summation operator  $\sum$  (i.e., the uppercase Sigma letter) lets us perform an operation on a sequence of numbers, which is often but not always a vector.

$$\vec{d} = \begin{bmatrix} 12 & 7 & -2 & 3 & -1 \end{bmatrix}$$

We can then calculate the sum of the first three elements of the vector, which is expressed as follows:

$$\sum_{i=1}^{3} d_i$$

Then we do the following math:

$$12 + 7 + (-2) = 17$$

It is also common to use n in the superscript to indicate that we want to sum all elements:

$$\sum_{i=1}^{n} d_i = 12 + 7 + (-2) + 3 + (-1) = 19$$

We can perform these operations using the sum() function in R:

$$vector_d \leftarrow c(12, 7, -2, 3, -1)$$

sum(vector\_d[1:3])

[1] 17

sum(vector\_d)

[1] 19

#### 5.2.2 Product

The product operator  $\prod$  (i.e., the uppercase Pi letter) can also perform operations over a sequence of elements in a vector. Recall our previous vector:

$$\vec{d} = \begin{bmatrix} 12 & 7 & -2 & 3 & 1 \end{bmatrix}$$

We might want to calculate the product of all its elements, which is expressed as follows:

$$\prod_{i=1}^n d_i = 12 \cdot 7 \cdot (-2) \cdot 3 \cdot (-1) = 504$$

In R, we can compute products using the prod() function:

prod(vector\_d)

[1] 504

# i Exercise

Get the product of the first three elements of vector d. Write the notation by hand and use R to obtain the number.

### 5.3 Matrices

#### **5.3.1** Basics

We can append vectors together to form a matrix:

$$A = \begin{bmatrix} 12 & 14 & 15 \\ 115 & 22 & 127 \\ 193 & 29 & 219 \end{bmatrix}$$

The number of rows and columns of a matrix constitute the *dimensions* of the matrix. The first number is the number of rows ("r") and the second number is the number of columns ("c") in the matrix.

# ! Important

Find a way to remember "r x c" permanently. The order of the dimensions never changes.

Matrix A above, for example, is a  $3 \times 3$  matrix. Sometimes we'd refer to it as  $A_{3\times 3}$ .



It is common to use capital letters (sometimes **bold-faced**) to represent matrices. In contrast, vectors are usually represented with either bold lowercase letters or lowercase letters with an arrow on top (e.g.,  $\vec{v}$ ).

#### Constructing matrices in R

There are different ways to create matrices in R. One of the simplest is via rbind() or cbind(), which paste vectors together (either by rows or by columns):

```
# Create some vectors
vector1 <- 1:4
vector2 <- 5:8</pre>
```

```
vector3 <- 9:12
vector4 <- 13:16</pre>
```

```
# Using rbind(), each vector will be a row
rbind_mat <- rbind(vector1, vector2, vector3, vector4)
rbind_mat</pre>
```

```
[,1] [,2] [,3] [,4]
                2
                      3
vector1
                     7
           5
                6
vector2
                           8
vector3
           9
               10
                    11
                          12
vector4 13
               14
                     15
                          16
```

```
# Using cbind(), each vector will be a column
cbind_mat <- cbind(vector1, vector2, vector3, vector4)
cbind_mat</pre>
```

#### vector1 vector2 vector3 vector4 [1,] 9 5 13 [2,] 2 10 14 7 [3,] 3 11 15 [4,]8 12 4 16

An alternative is to use to properly named matrix() function. The basic syntax is matrix(data, nrow, ncol, byrow):

- data is the input vector which becomes the data elements of the matrix.
- nrow is the number of rows to be created.
- ncol is the number of columns to be created.
- byrow is a logical clue. If TRUE then the input vector elements are arranged by row. By default (FALSE), elements are arranged by column.

Let's see some examples:

```
# Elements are arranged sequentially by row.
M <- matrix(c(1:12), nrow = 4, byrow = T)
M</pre>
```

```
[2,] 4 5 6
[3,] 7 8 9
[4,] 10 11 12
```

```
# Elements are arranged sequentially by column (byrow = F by default). N <- matrix(c(1:12), nrow = 4) N
```

#### 5.3.2 Structure

How do we refer to specific elements of the matrix? For example, matrix A is an  $m \times n$  matrix where m = n = 3. This is sometimes called a *square matrix*.

More generally, matrix B is an  $m \times n$  matrix where the elements look like this:

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{mn} \end{bmatrix}$$

Thus  $b_{23}$  refers to the second unit down and third across. More generally, we refer to row indices as i and to column indices as j.

In R, we can access a matrix's elements using square brackets:

```
# In matrix N, access the element at 1st row and 3rd column. N[1,3]
```

[1] 9

```
# In matrix N, access the element at 4th row and 2nd column. N[4,2]
```

[1] 8



When trying to identify a specific element, the first subscript is the element's row and the second subscript is the element's column (always in that order).

# ⚠ Warning

In R, indexing is 1-based, meaning that the first element of a vector, matrix, or any other data structure is accessed with index 1. In other programming tools such as Python, indexing is 0-based, meaning that the first element of a list, array, or any other data structure is accessed with index 0.

```
# Create a 2x2 matrix in R
matrix_A <- matrix(1:4, nrow = 2, ncol = 2)

# Access the element in the first row, first column
element <- matrix_A[1, 1]  # This will return 1

# Create a list in Python
vector = [10, 20, 30, 40]

# Access the first element
first_element = vector[0]  # This will return 10
import numpy as np

# Create a 2x2 matrix in Python with NumPy
matrix_A = np.array([[1, 2], [3, 4]])

# Access the element in the first row, first column
element = matrix_A[0, 0]  # This will return 1</pre>
```

# 5.4 Matrix operations

#### 5.4.1 Addition and subtraction

- Addition and subtraction are straightforward operations.
- Matrices must have *exactly* the same dimensions for both of these operations.
- We add or subtract each element with the corresponding element from the other matrix.

• This is expressed as follows:

$$A \pm B = C$$

$$c_{ij} = a_{ij} \pm b_{ij} \ \forall i,j$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \pm \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & a_{13} \pm b_{13} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & a_{23} \pm b_{23} \\ a_{31} \pm b_{31} & a_{32} \pm b_{32} & a_{33} \pm b_{33} \end{bmatrix}$$

#### Addition and subtraction in R

We start by creating two 2x3 matrices:

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 9 & 4 & 6 \end{bmatrix}$$

# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
matrix1

And

$$B = \begin{bmatrix} 5 & 2 & 0 \\ 9 & 3 & 4 \end{bmatrix}$$

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2) matrix2

We can simply use the + and - operators for addition and substraction:

#### matrix1 + matrix2

#### matrix1 - matrix2

# i Exercise

(Use code for one of these and do the other one by hand!)

1) Calculate A + B

$$A = \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 1 \\ 2 & -1 \end{bmatrix}$$

2) Calculate A - B

$$A = \begin{bmatrix} 6 & -2 & 8 & 12 \\ 4 & 42 & 8 & -6 \end{bmatrix}$$

$$B = \begin{bmatrix} 18 & 42 & 3 & 7 \\ 0 & -42 & 15 & 4 \end{bmatrix}$$

# 5.4.2 Scalar multiplication

Scalar multiplication is very intuitive. As we know, a scalar is a single number. We multiply each value in the matrix by the scalar to perform this operation.

Formally, this is expressed as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$cA = \begin{bmatrix} ca_{11} & ca_{12} & ca_{13} \\ ca_{21} & ca_{22} & ca_{23} \\ ca_{31} & ca_{32} & ca_{33} \end{bmatrix}$$

In R, all we need to do is take an established matrix and multiply it by some scalar:

# matrix1 from our previous example
matrix1

matrix1 \* 3

# i Exercise

Calculate  $2 \times A$  and  $-3 \times B$ . Again, do one by hand and the other one using R.

$$A = \begin{bmatrix} 1 & 4 & 8 \\ 0 & -1 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} -15 & 1 & 5\\ 2 & -42 & 0\\ 7 & 1 & 6 \end{bmatrix}$$

# 5.4.3 Matrix multiplication

- Multiplying matrices is slightly trickier than multiplying scalars.
- Two matrices must be *conformable* for them to be multiplied together. This means that the number of columns in the first matrix equals the number of rows in the second.
- When multiplying  $A \times B$ , if A is  $m \times n$ , B must have n rows.

# Important

The conformability requirement never changes. Before multiplying anything, check to make sure the matrices are indeed conformable.

• The resulting matrix will have the same number of rows as the first matrix and the number of columns in the second. For example, if A is  $i \times k$  and B is  $k \times j$ , then  $A \times B$ will be  $i \times j$ .

Which of the following can we multiply? What will be the dimensions of the resulting matrix?

$$B_{4\times 1} = \begin{bmatrix} 2\\3\\4\\1 \end{bmatrix} M_{3\times 3} = \begin{bmatrix} 1 & 0 & 2\\1 & 2 & 4\\2 & 3 & 2 \end{bmatrix} L_{2\times 3} = \begin{bmatrix} 6 & 5 & -1\\1 & 4 & 3 \end{bmatrix}$$

The only valid multiplication based on the provided matrices is  $L \times M$ , which results in a  $2 \times 3$ matrix.

Why can't we multiply in the opposite order?

The non-commutative property of matrix multiplication is a fundamental aspect in matrix algebra. The multiplication of matrices is sensitive to the order in which the matrices are multiplied due to the requirements of dimensional compatibility, the resulting dimensions, and the computation process itself.



#### Warning

When multiplying matrices, order matters. Even if multiplication is possible in both directions, in general  $AB \neq BA$ .

#### Multiplication steps

• Multiply each row by each column, summing up each pair of multiplied terms.



This is sometimes to referred to as the "dot product," where we multiply matching members, then sum up.

• The element in position ij is the sum of the products of elements in the ith row of the first matrix (A) and the corresponding elements in the jth column of the second matrix (B).

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

#### Example

Suppose a company manufactures two kinds of furniture: chairs and sofas.

- A chair costs \$100 for wood, \$270 for cloth, and \$130 for feathers.
- Each sofa costs \$150 for wood, \$420 for cloth, and \$195 for feathers.

Chair	Sofa
100	150
270	420
130	195
	100 270

The same information about unit cost (C) can be presented as a matrix.

$$C = \begin{bmatrix} 100 & 150 \\ 270 & 420 \\ 130 & 195 \end{bmatrix}$$

Note that each of the three rows of this 3 x 2 matrix represents a material (wood, cloth, or feathers), and each of the two columns represents a product (chair or coach). The elements are the unit cost (in USD).

Now, suppose that the company will produce 45 chairs and 30 sofas this month. This production quantity can be represented in the following table, and also as a 2 x 1 matrix (Q):

Product	Quantity
Chair	45
Sofa	30

$$Q = \begin{bmatrix} 45 \\ 30 \end{bmatrix}$$

What will be the company's total cost? The "total expenditure" is equal to the "unit cost" times the "production quantity" (the number of units).

The total expenditure (E) for each material this month is calculated by multiplying these two matrices.

$$E = CQ = \begin{bmatrix} 100 & 150 \\ 270 & 420 \\ 130 & 195 \end{bmatrix} \begin{bmatrix} 45 \\ 30 \end{bmatrix} = \begin{bmatrix} (100)(45) + (150)(30) \\ (270)(45) + (420)(30) \\ (130)(45) + (195)(30) \end{bmatrix} = \begin{bmatrix} 9,000 \\ 24,750 \\ 11,700 \end{bmatrix}$$

Multiplying the 3x2 Cost matrix (C) times the 2x1 Quantity matrix (Q) yields the 3x1 Expenditure matrix (E).

As a result of this matrix multiplication, we determine that this month the company will incur expenditures of:

- \$9,000 for wood
- \$24,750 for cloth
- \$11,700 for feathers.

#### Matrix multiplication in R

Before attempting matrix multiplication, we must make sure the matrices are conformable (as we do for our manual calculations).

Then we can multiply our matrices together using the %\*% operator.

```
C <- matrix(c(100, 270, 130, 150, 420, 195), nrow = 3)
C</pre>
```

```
[,1] [,2]
[1,] 100 150
[2,] 270 420
[3,] 130 195
```

```
Q <- matrix(c(45, 30), nrow = 2)
Q
```

[,1]

[1,] 45

[2,] 30

C %\*% Q

[,1]

[1,] 9000

[2,] 24750

[3,] 11700

⚠ Warning

If you have a missing value or NA in one of the matrices you are trying to multiply (something we will discuss in further detail in the next module), you will have NAs in your resulting matrix.

# 5.4.4 Properties of operations

• Addition and subtraction:

- Associative:  $(A \pm B) \pm C = A \pm (B \pm C)$ 

– Communicative:  $A \pm B = B \pm A$ 

• Multiplication:

 $-AB \neq BA$ 

-A(BC) = (AB)C

-A(B+C) = AB + AC

- (A+B)C = AC + BC

# 5.5 Special matrices

#### Square matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- In a square matrix, the number of rows equals the number of columns (m = n):
- The diagonal of a matrix is a set of numbers consisting of the elements on the line from the upper-left-hand to the lower-right-hand corner of the matrix, as in d(A)=[1,5,9]. Diagonals are particularly useful in square matrices.
- The trace of a matrix, denoted as tr(A), is the sum of the diagonal elements of the matrix. tr(A) = 1 + 5 + 9 = 15

### Diagonal matrix:

• In a diagonal matrix, all of the elements of the matrix that are not on the diagonal are equal to zero.

$$D = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

#### Scalar matrix:

• A scalar matrix is a diagonal matrix where the diagonal elements are all equal to each other. In other words, we're really only concerned with one scalar (or element) held in the diagonal.

$$S = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

#### Identity matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- The identity matrix is a scalar matrix with all of the diagonal elements equal to one.
- Remember that, as with all diagonal matrices, the off-diagonal elements are equal to zero.

• The capital letter I is reserved for the identity matrix. For convenience, a 3x3 identity matrix can be denoted as  $I_3$ .

# 5.6 Transpose

The transpose is the original matrix with the rows and the columns interchanged.

The notation is either J' ("J prime") or  $J^T$  ("J transpose").

$$J = \begin{bmatrix} 4 & 5 \\ 3 & 0 \\ 7 & -2 \end{bmatrix}$$

$$J' = J^T = \begin{bmatrix} 4 & 3 & 7 \\ 5 & 0 & -2 \end{bmatrix}$$

In R, we use t() to get the transpose.

$$J \leftarrow matrix(c(4, 3, 7, 5, 0, -2), ncol = 2)$$

t(J)

# 5.7 Inverse

- Just like a number has a reciprocal, a matrix has an inverse.
- When we multiply a matrix by its inverse we get the identity matrix (which is like "1" for matrices).

$$A \times A^{-1} = I$$

• The inverse of A is  $A^{-1}$  only when:

$$AA^{-1} = A^{-1}A = I$$

• Sometimes there is no inverse at all.

### Note

For now, don't worry about calculating the inverse of a matrix manually. This is the type of task we use R for.

• In R, we use the solve() function to calculate the inverse of a matrix:

#### solve(A)

# 5.8 Linear systems and matrices

- A system of equations can be represented by an augmented matrix.
- System of equations:

$$3x + 6y = 12$$

$$5x + 10y = 25$$

• In an augmented matrix, each row represents one equation in the system and each column represents a variable or the constant terms.

$$\begin{bmatrix} 3 & 6 & 12 \\ 5 & 10 & 25 \end{bmatrix}$$

# 5.9 OLS and matrices

- We can use the logic above to calculate estimates for our ordinary least squares (OLS) models.
- OLS is a linear regression technique used to find the best-fitting line for a set of data points (observations) by minimizing the residuals (the differences between the observed and predicted values).
- We minimize the sum of the squared errors.

### 5.9.1 Dependent variable

- Suppose, for example, we have a sample consisting of n observations.
- The dependent variable is denoted as an  $n \times 1$  column vector.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

# 5.9.2 Independent variables

- Suppose there are k independent variables and a constant term, meaning k+1 columns and n rows.
- We can represent these variables as an  $n \times (k+1)$  matrix, expressed as follows:

$$X = \begin{bmatrix} 1 & x_{11} & \dots & x_{1k} \\ 1 & x_{21} & \dots & x_{2k} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_{n1} & \dots & x_{nk} \end{bmatrix}$$

•  $x_{ii}$  is the *i*-th observation of the *j*-th independent variable.

### 5.9.3 Linear regression model

- Let's say we have 173 observations (n = 173) and 2 IVs (k = 3).
- This can be expressed as the following linear equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

• In matrix form, we have:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{1173} & x_{2173} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_{173} \end{bmatrix}$$

• All 173 equations can be represented by:

$$y = X\beta + \epsilon$$

#### 5.9.4 Estimates

• Without getting too much into the mechanics, we can calculate our coefficient estimates with matrix algebra using the following equation:

$$\hat{\beta} = (X'X)^{-1}X'Y$$

- Read aloud, we say "X prime X inverse, X prime Y".
- The little hat on our beta  $(\hat{\beta})$  signifies that these are estimates.
- Remember, the OLS method is to choose  $\hat{\beta}$  such that the sum of squared residuals ("SSR") is minimized.

#### **5.9.4.1** Example in R

• We will load the mtcars data set (our favorite) for this example, which contains data about many different car models.

#### cars\_df <- mtcars

• Now, we want to estimate the association between hp (horsepower) and wt (weight), our independent variables, and mpg (miles per gallon), our dependent variable.

• First, we transform our dependent variable into a matrix, using the as.matrix function and specifying the column of the mtcars data set to create a column vector of our observed values for the DV.

```
Y <- as.matrix(cars_df$mpg)
Y
```

```
[,1]
 [1,] 21.0
 [2,] 21.0
 [3,] 22.8
 [4,] 21.4
 [5,] 18.7
 [6,] 18.1
 [7,] 14.3
 [8,] 24.4
 [9,] 22.8
[10,] 19.2
[11,] 17.8
[12,] 16.4
[13,] 17.3
[14,] 15.2
[15,] 10.4
[16,] 10.4
[17,] 14.7
[18,] 32.4
[19,] 30.4
[20,] 33.9
[21,] 21.5
[22,] 15.5
[23,] 15.2
[24,] 13.3
[25,] 19.2
[26,] 27.3
[27,] 26.0
[28,] 30.4
[29,] 15.8
[30,] 19.7
[31,] 15.0
[32,] 21.4
```

• Next, we do the same thing for our independent variables of interest, and our constant.

```
# create two separate matrices for IVs
X1 <- as.matrix(cars_df$hp)
X2 <- as.matrix(cars_df$wt)

# create constant column

# bind them altogether into one matrix
constant <- rep(1, nrow(cars_df))
X <- cbind(constant, X1, X2)
X</pre>
```

#### constant

```
[1,]
            1 110 2.620
 [2,]
            1 110 2.875
 [3,]
           1 93 2.320
 [4,]
            1 110 3.215
 [5,]
            1 175 3.440
 [6,]
           1 105 3.460
 [7,]
           1 245 3.570
 [8,]
            1 62 3.190
[9,]
           1 95 3.150
           1 123 3.440
[10,]
[11,]
           1 123 3.440
[12,]
            1 180 4.070
[13,]
            1 180 3.730
[14,]
            1 180 3.780
            1 205 5.250
[15,]
[16,]
            1 215 5.424
            1 230 5.345
[17,]
[18,]
            1 66 2.200
[19,]
            1 52 1.615
            1 65 1.835
[20,]
[21,]
            1 97 2.465
[22,]
            1 150 3.520
[23,]
            1 150 3.435
[24,]
            1 245 3.840
[25,]
            1 175 3.845
[26,]
            1 66 1.935
[27,]
            1 91 2.140
[28,]
           1 113 1.513
            1 264 3.170
[29,]
            1 175 2.770
[30,]
```

```
[31,] 1 335 3.570
[32,] 1 109 2.780
```

• Next, we calculate X'X, X'Y, and  $(X'X)^{-1}$ .

Don't forget to use **%\*%** for matrix multiplication!

```
# X prime X
XpX <- t(X) %*% X

# X prime X inverse
XpXinv <- solve(XpX)

# X prime Y
XpY <- t(X) %*% Y

# beta coefficient estimates
bhat <- XpXinv %*% XpY
bhat</pre>
```

```
[,1]
constant 37.22727012
-0.03177295
-3.87783074
```

# 6 Tidy data analysis II

In this session, we'll cover a few more advanced topics related to data wrangling. Again we'll use the tidyverse:

```
library(tidyverse)
```

```
----- tidyverse 2.0.0 --
-- Attaching core tidyverse packages ----
v dplyr
            1.1.4
                      v readr
                                  2.1.5
v forcats
            1.0.0
                      v stringr
                                   1.5.1
                                  3.3.0
v ggplot2
            3.5.2
                      v tibble
v lubridate 1.9.4
                      v tidyr
                                   1.3.1
v purrr
            1.1.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()
                  masks stats::lag()
i Use the conflicted package (<a href="http://conflicted.r-lib.org/">http://conflicted.r-lib.org/</a>) to force all conflicts to become
```

# 6.1 Loading data in different formats.

In this module we will use cross-national data from the Quality of Government (QoG) project (Dahlberg et al., 2023).

Notice how in the data/ folder we have multiple versions of the same dataset (a subset of the QOG basic dataset): .csv (comma-separated values), .rds (R), .xlsx (Excel), .dta (Stata), and .sav (SPSS).

#### 6.1.1 CSV and R data files

We can use the read\_csv() and read\_rds() functions from the tidyverse<sup>1</sup> to read the .csv and .rds (R) data files:

<sup>&</sup>lt;sup>1</sup>Technically, the read\_csv() and read\_rds() functions come from readr, one of the tidyverse constituent packages.

```
qog_csv <- read_csv("data/sample_qog_bas_ts_jan23.csv")

Rows: 1085 Columns: 8
-- Column specification -------
Delimiter: ","
chr (4): cname, ccodealp, region, ht_colonial
dbl (4): year, wdi_pop, vdem_polyarchy, vdem_corr

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.</pre>
```

```
qog_rds <- read_rds("data/sample_qog_bas_ts_jan23.rds")</pre>
```

For reading files from other software (Excel, Stata, or SPSS), we need to load additional packages. Luckily, they are automatically installed when one installs the tidyverse.

### 6.1.2 Excel data files

For Excel files (.xls or .xlsx files), the readxl package has a handy read\_excel() function.

```
library(readxl)
qog_excel <- read_excel("data/sample_qog_bas_ts_jan23.xlsx")</pre>
```



Useful arguments of the read\_excel() function include sheet =, which reads particular sheets (specified via their positions or sheet names), and range =, which extracts a particular cell range (e.g., 'A5:E25').

#### 6.1.3 Stata and SPSS data files

To load files from Stata (.dta) or SPSS (.spss), one needs the haven package and its properly-named read\_stata() and read\_spss() functions:

```
library(haven)
qog_stata <- read_stata("data/sample_qog_bas_ts_jan23.dta")
qog_spss <- read_spss("data/sample_qog_bas_ts_jan23.sav")</pre>
```



Datasets from Stata and SPSS can have additional properties, like variable labels and special types of missing values. To learn more about this, check out the "Labelled data" chapter from Danny Smith's  $Survey\ Research\ Datasets\ and\ R\ (2020)$ .

### 6.1.4 Our data for this session

We will rename one of our objects to qog:

```
qog <- qog_csv
qog</pre>
```

```
# A tibble: 1,085 x 8
                         year region wdi_pop vdem_polyarchy vdem_corr ht_colonial
   cname
              ccodealp
   <chr>
               <chr>
                        <dbl> <chr>
                                        <dbl>
                                                        <dbl>
                                                                   <dbl> <chr>
                         1990 Carib~
 1 Antigua a~ ATG
                                        63328
                                                           NA
                                                                      NA British
2 Antigua a~ ATG
                         1991 Carib~
                                        63634
                                                           NA
                                                                      NA British
3 Antigua a~ ATG
                         1992 Carib~
                                        64659
                                                           NA
                                                                      NA British
4 Antigua a~ ATG
                                                                      NA British
                         1993 Carib~
                                        65834
                                                           NA
5 Antigua a~ ATG
                         1994 Carib~
                                        67072
                                                           NA
                                                                      NA British
6 Antigua a~ ATG
                         1995 Carib~
                                        68398
                                                           NA
                                                                      NA British
7 Antigua a~ ATG
                         1996 Carib~
                                        69798
                                                           NA
                                                                      NA British
8 Antigua a~ ATG
                         1997 Carib~
                                        71218
                                                           NA
                                                                      NA British
9 Antigua a~ ATG
                         1998 Carib~
                                        72572
                                                           NA
                                                                      NA British
10 Antigua a~ ATG
                         1999 Carib~
                                        73821
                                                           NA
                                                                      NA British
# i 1,075 more rows
```

This dataset is a small sample of QOG, which contains data for countries in the Americas from 1990 to 2020. The observational unit is thus country-year. You can access the full codebook online. The variables are as follows:

Variable	Description			
cname	Country name			
ccodealp	Country code (ISO-3 character convention)			
year	Year			
region	Region (following legacy WDI convention). Added to QOG by			
	us.			
wdi_pop	Total population, from the World Development Indicators			
vdem_polyarchy	V-Dem's polyarchy index (electoral democracy)			

Variable	Description		
vdem_corr	V-Dem's corruption index		
${ t ht\_colonial}$	Former colonial ruler		

## 6.2 Recoding variables

Take a look at the ht\_colonial variable. We can do a simple tabulation with count():

```
qog |>
count(ht_colonial)
```

```
# A tibble: 6 x 2
 ht_colonial
                       n
  <chr>
                   <int>
1 British
                     372
2 Dutch
                      31
3 French
                      31
4 Never colonized
                      62
5 Portuguese
                      31
6 Spanish
                     558
```



We might want to recode this variable. For instance, we could create a dummy/binary variable for whether the country was a British colony. We can do this with if\_else(), which works with logical conditions:

Instead of a numeric classification (0 and 1), we could use characters:

if\_else() is great for binary recoding. But sometimes we want to create more than two
categories. We can use case\_when():

```
qog |>
    # syntax is condition ~ value
mutate(cat_col = case_when(
    ht_colonial == "British" ~ "British",
    ht_colonial == "Spanish" ~ "Spanish",
    .default = "Other" # what to do in all other cases
)) |>
count(cat_col)
```

The .default = argument in case when() can also be used to leave the variable as-is for non-specified cases. For example, let's combine Portuguese and Spanish colonies:

```
qog |>
  # syntax is condition ~ value
  mutate(cat_col = case_when(
   ht_colonial %in% c("Spanish", "Portuguese") ~ "Spanish/Portuguese",
    .default = ht_colonial # what to do in all other cases
  )) |>
  count(cat_col)
```

```
# A tibble: 5 x 2
 cat_col
                          n
  <chr>
                      <int>
1 British
                        372
2 Dutch
                         31
3 French
                         31
4 Never colonized
5 Spanish/Portuguese
                        589
```

### i Exercise

- 1. Create a dummy variable, d\_large\_pop, for whether the country-year has a population of more than 1 million. Then compute its mean. Your code:
- 2. Which countries are recorded as "Never colonized"? Change their values to other reasonable codings and compute a tabulation with count(). Your code:

## 6.3 Missing values

Missing values are commonplace in real datasets. In R, missing values are a special type of value in vectors, denoted as NA.



### ⚠ Warning

The special value NA is different from the character value "NA". For example, notice that a numeric vector can have NAs, while it obviously cannot hold the character value "NA":

```
c(5, 4.6, NA, 8)
[1] 5.0 4.6 NA 8.0
```

A quick way to check for missing values in small datasets is with the summary() function:

### summary(qog)

```
ccodealp
                                                          region
   cname
                                            year
                   Length: 1085
                                       Min.
                                                       Length: 1085
Length: 1085
                                               :1990
                   Class : character
Class : character
                                       1st Qu.:1997
                                                       Class : character
Mode :character
                   Mode :character
                                       Median:2005
                                                       Mode :character
                                       Mean
                                               :2005
                                       3rd Qu.:2013
                                       Max.
                                               :2020
                    vdem_polyarchy
                                                        ht_colonial
   wdi_pop
                                        vdem_corr
            40542
                    Min. :0.0710
                                             :0.0260
                                                        Length: 1085
Min.
                                      Min.
1st Qu.:
                    1st Qu.:0.5570
                                                        Class : character
           389131
                                      1st Qu.:0.1890
Median: 5687744
                    Median :0.7030
                                      Median :0.5550
                                                        Mode : character
Mean
       : 25004057
                    Mean
                            :0.6569
                                      Mean
                                              :0.4922
3rd Qu.: 16195902
                    3rd Qu.:0.8030
                                      3rd Qu.:0.7540
       :331501080
Max.
                    Max.
                            :0.9160
                                      Max.
                                              :0.9630
                    NA's
                            :248
                                      NA's
                                              :248
```

Notice that we have missingness in the vdem\_polyarchy and vdem\_corr variables. We might want to filter the dataset to see which observations are in this situation:

```
qog |>
  filter(vdem_polyarchy == NA | vdem_corr == NA)

# A tibble: 0 x 8
# i 8 variables: cname <chr>, ccodealp <chr>, year <dbl>, region <chr>,
# wdi_pop <dbl>, vdem_polyarchy <dbl>, vdem_corr <dbl>, ht_colonial <chr>>
```

But the code above doesn't work! To refer to missing values in logical conditions, we cannot use == NA. Instead, we need to use the is.na() function:

```
qog |>
  filter(is.na(vdem_polyarchy) | is.na(vdem_corr))
```

1	Antigua a~	ATG	1990 (	Carib~	63328	NA	NA	British			
2	Antigua a~	ATG	1991 (	Carib~	63634	NA	NA	British			
3	Antigua a~	ATG	1992 (	Carib~	64659	NA	NA	British			
4	Antigua a~	ATG	1993 (	Carib~	65834	NA	NA	British			
5	Antigua a~	ATG	1994 (	Carib~	67072	NA	NA	British			
6	Antigua a~	ATG	1995 (	Carib~	68398	NA	NA	British			
7	Antigua a~	ATG	1996 (	Carib~	69798	NA	NA	British			
8	Antigua a~	ATG	1997 (	Carib~	71218	NA	NA	British			
9	Antigua a~	ATG	1998 (	Carib~	72572	NA	NA	British			
10	Antigua a~	ATG	1999 (	Carib~	73821	NA	NA	British			
# i 238 more rows											

Notice that, in most R functions, missing values are "contagious." This means that any missing value will contaminate the operation and carry over to the results. For example:

Sometimes we'd like to perform our operations even in the presence of missing values, simply excluding them. Most basic R functions have an na.rm = argument to do this:

### i Exercise

Calculate the median value of the corruption variable for each region (i.e., perform a grouped summary). Your code:

## 6.4 Pivoting data

We will now load another time-series cross-sectional dataset, but in a slightly different format. It's adapted from the World Bank's World Development Indicators (WDI) (2023) and records gross domestic product at purchasing power parity (GDP PPP).

```
gdp <- read_excel("data/wdi_gdp_ppp.xlsx")
gdp</pre>
```

```
# A tibble: 266 x 35
   country_name
                       country_code
                                       1990
                                                 `1991`
                                                          1992
                                                                    `1993`
                                                                             1994
   <chr>
                        <chr>
                                        <dbl>
                                                  <dbl>
                                                           <dbl>
                                                                    <dbl>
                                                                              <dbl>
                                                         2.32e 9
                                      2.03e 9
                                               2.19e 9
1 Aruba
                       ABW
                                                                  2.48e 9
                                                                           2.69e 9
2 Africa Eastern and~ AFE
                                      9.41e11
                                               9.42e11
                                                         9.23e11
                                                                 9.19e11
                                                                           9.35e11
3 Afghanistan
                        AFG
                                     NA
                                              NA
                                                        NA
                                                                 NA
                                                                          NA
4 Africa Western and~ AFW
                                      5.76e11
                                               5.84e11
                                                         5.98e11
                                                                  5.92e11
                                                                           5.91e11
5 Angola
                       AGO
                                      6.85e10
                                               6.92e10
                                                         6.52e10
                                                                  4.95e10
                                                                           5.02e10
                                                                           1.26e10
6 Albania
                       ALB
                                      1.59e10
                                               1.14e10
                                                         1.06e10
                                                                  1.16e10
7 Andorra
                       AND
                                     NA
                                              NA
                                                        NA
                                                                 NA
                                                                          NA
8 Arab World
                       ARB
                                      2.19e12
                                               2.25e12
                                                         2.35e12
                                                                  2.41e12
                                                                           2.48e12
                                               2.03e11
9 United Arab Emirat~ ARE
                                      2.01e11
                                                         2.10e11
                                                                  2.12e11
                                                                           2.27e11
10 Argentina
                       ARG
                                      4.61e11 5.04e11 5.43e11 5.88e11 6.22e11
# i 256 more rows
# i 28 more variables: `1995` <dbl>, `1996` <dbl>, `1997` <dbl>, `1998` <dbl>,
    `1999` <dbl>, `2000` <dbl>, `2001` <dbl>, `2002` <dbl>, `2003` <dbl>,
#
    `2004` <dbl>, `2005` <dbl>, `2006` <dbl>, `2007` <dbl>, `2008` <dbl>,
    `2009` <dbl>, `2010` <dbl>, `2011` <dbl>, `2012` <dbl>, `2013` <dbl>,
#
#
    `2014` <dbl>, `2015` <dbl>, `2016` <dbl>, `2017` <dbl>, `2018` <dbl>,
#
    `2019` <dbl>, `2020` <dbl>, `2021` <dbl>, `2022` <dbl>
```

Note how the information is recorded differently. Here columns are not variables, but years. We call datasets like this one **wide**, in contrast to the **long** datasets we have seen before. In general, R and the **tidyverse** work much nicer with long datasets. Luckily, the **tidyr** package of the **tidyverse** makes it easy to convert datasets between these two formats.

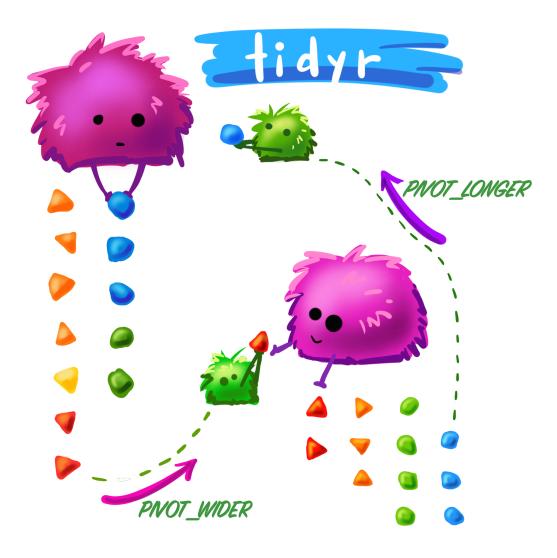


Figure 6.1: Source: Illustration by Allison Horst, adapted by Peter Higgins.

We will use the pivot\_longer() function:

```
names_transform = as.integer) # make sure that years are numeric
gdp_long
```

```
# A tibble: 8,778 x 4
   country_name country_code year wdi_gdp_ppp
   <chr>
                <chr>
                              <int>
                                          <dbl>
1 Aruba
                ABW
                               1990 2025472682.
2 Aruba
                               1991 2186758474.
                ABW
3 Aruba
                ABW
                               1992 2315391348.
4 Aruba
                ABW
                               1993 2484593045.
5 Aruba
                               1994 2688426606.
                ABW
6 Aruba
                ABW
                               1995 2756904694.
7 Aruba
                ABW
                               1996 2789595753.
                               1997 2986175079.
8 Aruba
                ABW
9 Aruba
                ABW
                               1998 3045659222.
10 Aruba
                               1999 3083365758.
                ABW
# i 8,768 more rows
```

Done! This is a much friendlier format to work with. For example, we can now do summaries:

```
gdp_long |>
summarize(mean_gdp_ppp = mean(wdi_gdp_ppp, na.rm = T), .by = country_name)
```

```
# A tibble: 266 x 2
   country_name
                                mean_gdp_ppp
   <chr>
                                       <dbl>
 1 Aruba
                                     3.38e 9
2 Africa Eastern and Southern
                                     1.61e12
3 Afghanistan
                                     5.56e10
4 Africa Western and Central
                                     1.15e12
5 Angola
                                     1.38e11
6 Albania
                                     2.56e10
7 Andorra
                                   NaN
8 Arab World
                                     4.22e12
9 United Arab Emirates
                                     4.29e11
10 Argentina
                                     8.06e11
# i 256 more rows
```

### i Exercise

Convert back gdp\_long to a wide format using pivot\_wider(). Check out the help file using ?pivot\_wider. Your code:

## 6.5 Merging datasets

It is extremely common to want to integrate data from multiple sources. Combining information from two datasets is called *merging* or *joining*.

To do this, we need ID variables in common between the two data sets. Using our QOG and WDI datasets, these variables will be country code (which in this case is shared between the two datasets) and year.



Standardized unit codes (like country codes) are extremely useful when merging data. It's harder than expected for a computer to realize that "Bolivia (Plurinational State of)" and "Bolivia" refer to the same unit. By default, these units will not be matched.<sup>2</sup>

Okay, now to the merging. Imagine we want to add information about GDP to our QOG main dataset. To do so, we can use the left\_join() function, from the tidyverse's dplyr package:

```
qog_plus |>
    # select variables for clarity
    select(cname, ccodealp, year, wdi_pop, wdi_gdp_ppp)
```

```
# A tibble: 1,085 x 5
```

<sup>&</sup>lt;sup>2</sup>There are R packages to deal with these complications. fuzzyjoin matches units by their approximate distance, using some clever algorithms. countrycode allows one to standardize country names and country codes across different conventions.

```
2 Antigua and Barbuda ATG
                                  1991
                                         63634 987701012.
3 Antigua and Barbuda ATG
                                  1992
                                         64659 999143284.
4 Antigua and Barbuda ATG
                                  1993
                                         65834 1051896837.
5 Antigua and Barbuda ATG
                                         67072 1122128908.
                                  1994
6 Antigua and Barbuda ATG
                                  1995
                                         68398 1073208718.
7 Antigua and Barbuda ATG
                                  1996
                                         69798 1144088355.
8 Antigua and Barbuda ATG
                                  1997
                                         71218 1206688391.
9 Antigua and Barbuda ATG
                                  1998
                                         72572 1263778328.
10 Antigua and Barbuda ATG
                                  1999
                                         73821 1310634399.
# i 1,075 more rows
```

## • Tip

Most of the time, you'll want to do a left\_join(), which is great for adding new information to a "base" dataset, without dropping information from the latter. In limited situations, other types of joins can be helpful. To learn more about them, you can read Jenny Bryan's excellent tutorial on dplyr joins.

### i Exercise

There is a dataset on country's CO2 emissions, again from the World Bank (2023), in "data/wdi\_co2.csv". Load the dataset into R and add a new variable with its information, wdi\_co2, to our qog\_plus data frame. Finally, compute the average values of CO2 emissions per capita, by country. Tip: this exercise requires you to do many steps—plan ahead before you start coding! Your code:

### 6.5.1 Sanity checks

Sanity checks are small tests to make sure that your code is doing what you think it's doing. They are especially important in complex operations like joins, but the idea can be extended to pretty much any command.

The tidylog package gives more information about tidyverse operations, and it's an easy/automatic way to check your work:

## library(tidylog)

You can also construct sanity checks manually. For instance, we know that a left join shouldn't modify a data frame's number of rows:

```
nrow(qog) == nrow(qog_plus)
```

[1] TRUE

## 6.6 Plotting extensions: trend graphs, facets, and customization

## i Exercise

Draw a scatterplot with time in the x-axis and democracy scores in the y-axis. Your code:

How can we visualize trends effectively? One alternative is to use a trend graph. Let's start by computing the yearly averages for democracy in the whole region:

```
dem_yearly <- qog |>
  summarize(mean_dem = mean(vdem_polyarchy, na.rm = T), .by = year)
```

summarize: now 31 rows and 2 columns, ungrouped

### dem\_yearly

```
6 1995 0.642
7 1996 0.651
8 1997 0.657
9 1998 0.663
10 1999 0.661
# i 21 more rows
```

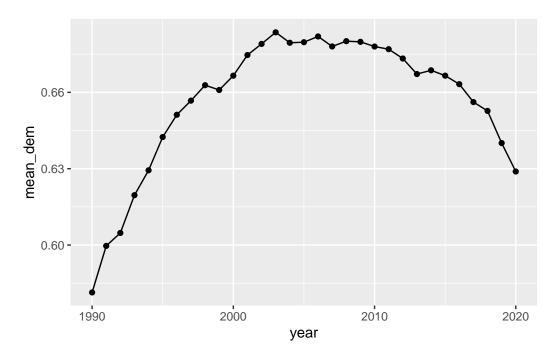
Now we can plot them with a scatterplot:

```
ggplot(dem_yearly, aes(x = year, y = mean_dem)) +
  geom_point()
```

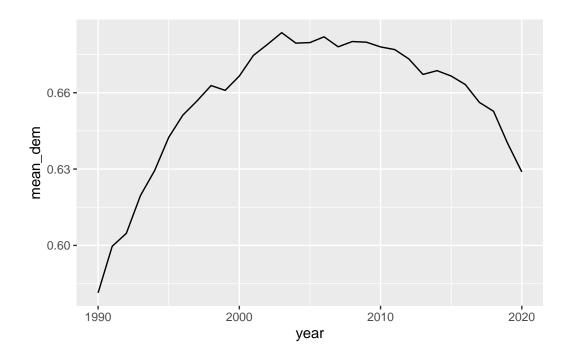


We can add  ${\tt geom\_line()}$  to connect the dots:

```
ggplot(dem_yearly, aes(x = year, y = mean_dem)) +
  geom_point() +
  geom_line()
```



We can, of course, remove to points to only keep the line:

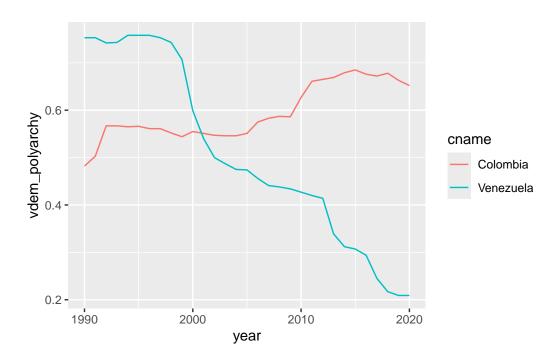


What if we want to plot trends for different countries? We can use the group and color aesthetic mappings (no need to do a summary here! data is already at the country-year level):

```
# filter to only get Colombia and Venezuela
dem_yearly_countries <- qog |>
  filter(ccodealp %in% c("COL", "VEN"))
```

filter: removed 1,023 rows (94%), 62 rows remaining

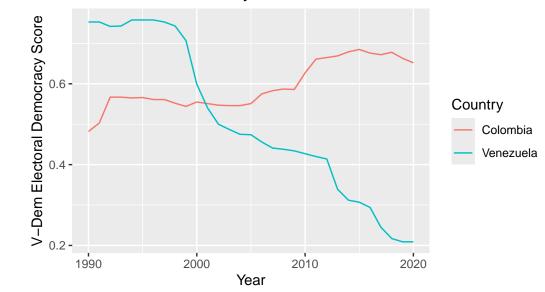
```
ggplot(dem_yearly_countries, aes(x = year, y = vdem_polyarchy, color = cname)) +
  geom_line()
```



Remember that we can use the labs() function to add labels:

```
ggplot(dem_yearly_countries, aes(x = year, y = vdem_polyarchy, color = cname)) +
  geom_line() +
  labs(x = "Year", y = "V-Dem Electoral Democracy Score", color = "Country",
       title = "Evolution of democracy scores in Colombia and Venezuela",
       caption = "Source: V-Dem (Coppedge et al., 2022) in QOG dataset.")
```

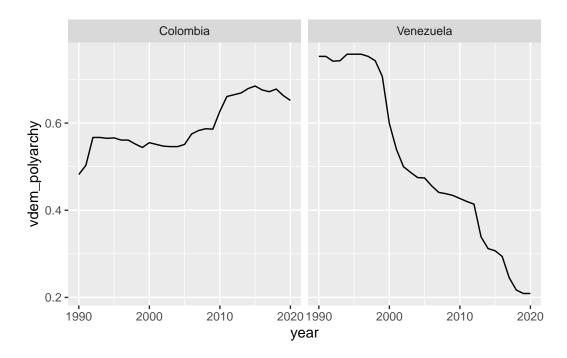
# Evolution of democracy scores in Colombia and Venezuela



Source: V-Dem (Coppedge et al., 2022) in QOG dataset.

Another way to display these trends is by using *facets*, which divide a plot into small boxes according to a categorical variable (no need to add color here):

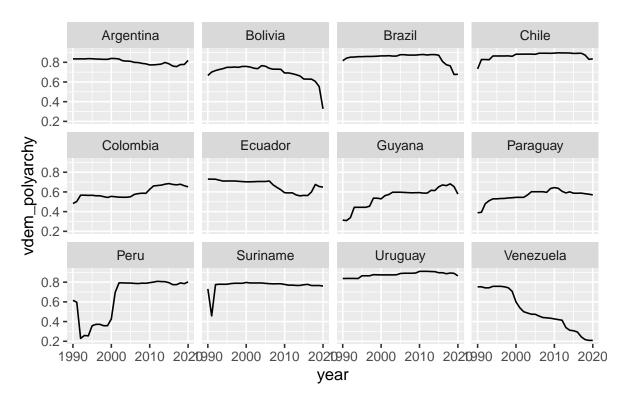
```
ggplot(dem_yearly_countries, aes(x = year, y = vdem_polyarchy)) +
  geom_line() +
  facet_wrap(~cname)
```



Facets are particularly useful for many categories (where the number of distinguishable colors reaches its limit):

```
ggplot(qog |> filter(region == "South America"),
    aes(x = year, y = vdem_polyarchy)) +
    geom_line() +
    facet_wrap(~cname)
```

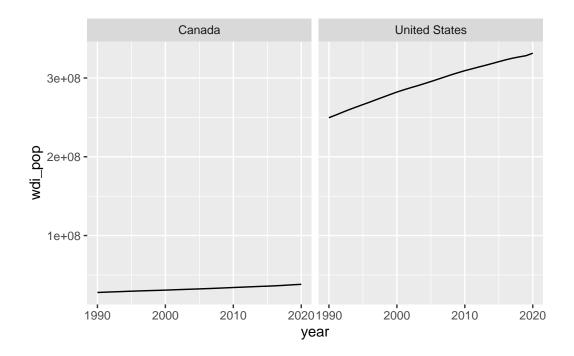
filter: removed 713 rows (66%), 372 rows remaining



With facets, one can control whether each facet picks its own scales or if all facets share the same scale. For example, let's plot the populations of Canada and the US:

```
ggplot(qog |> filter(cname %in% c("Canada", "United States")),
    aes(x = year, y = wdi_pop)) +
    geom_line() +
    facet_wrap(~cname)
```

filter: removed 1,023 rows (94%), 62 rows remaining



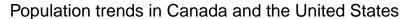
The scales are so disparate that unifying them yields a plot that's hard to interpret. But if we're interested in within-country trends, we can let each facet have its own scale with the scales = argument (which can be "fixed", "free\_x", "free\_y", or "free"):

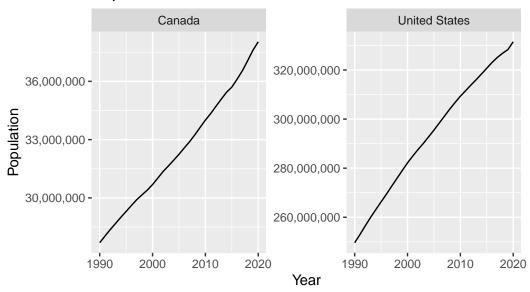
```
ggplot(qog |> filter(cname %in% c("Canada", "United States")),
        aes(x = year, y = wdi_pop)) +
    geom_line() +
   facet_wrap(~cname, scales = "free_y")
```

filter: removed 1,023 rows (94%), 62 rows remaining



This ability to visualize within time trends also makes facets appealing in many situations.





Source: World Development Indicators (World Bank, 2023) in QOG dataset.

While it's impossible for us to review all the customization options you might need, a fantastic reference is the "ggplot2: Elegant Graphics for Data Analysis" book by Hadley Wickham, Danielle Navarro, and Thomas Lin Pedersen.

## i Exercise

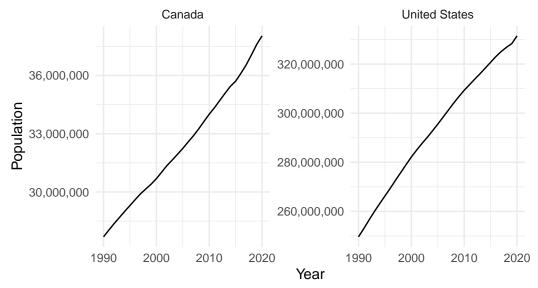
Using your merged dataset from the previous section, plot the trajectories of C02 per capita emissions for the US and Haiti. Use adequate scales.

## **6.6.1 Themes**

We can change the overall aspect of a ggplot2 figure by changing its theme:

```
p +
  theme_minimal()
```

## Population trends in Canada and the United States



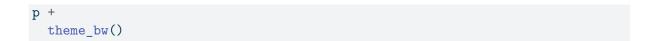
Source: World Development Indicators (World Bank, 2023) in QOG dataset.

p +
 theme\_classic()

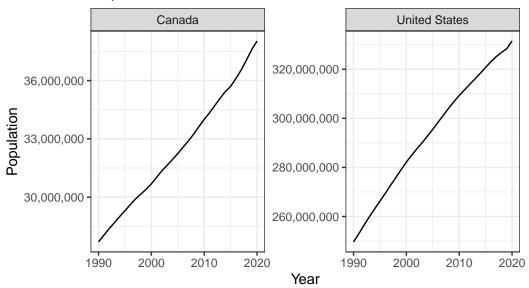
## Population trends in Canada and the United States



Source: World Development Indicators (World Bank, 2023) in QOG dataset.



# Population trends in Canada and the United States

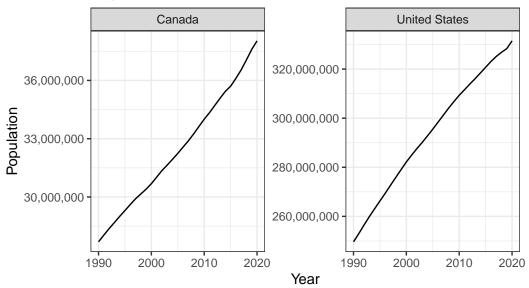


Source: World Development Indicators (World Bank, 2023) in QOG dataset.

If you are going to make multiple plots in a script, you can set the theme at the beginning with theme\_bw():

```
theme_set(theme_bw())
p
```

# Population trends in Canada and the United States



Source: World Development Indicators (World Bank, 2023) in QOG dataset.

# 7 Probability

## 7.1 What is probability?

- Informally, a probability is a number that describes how likely an event is.
  - It is, by definition, between 0 and 1.
  - What is the probability that a fair coin flip will result in heads?
- We can also think of a probability as an outcome's **relative frequency** after repeating an "experiment" many times.<sup>1</sup>
  - In this setting, an experiment is "an action or a set of actions that produce stochastic [random] events of interest" (Imai and Williams 2022, p. 281). Not to confuse with scientific experiments!
  - If we were to flip a million fair coins, what will be the proportion of heads?
- A probability space  $(\Omega, S, P)$  is a formal way to talk about a random process:
  - The sample space  $(\Omega)$  is the set of all possible outcomes.
  - The event space (S) is a collection of events (an event is a subset of  $\Omega$ ).
  - The probability measure (P) is a function that assigns a probability in  $\mathbb{R}$  to every event in S. So  $P: S \to \mathbb{R}$ .
- We can formalize our intuitions with the **probability axioms** (sometimes called Kolmogorov's axioms):
  - $-P(A) > 0, \forall A \in S.$ 
    - \* Probabilities must be non-negative.
  - $-P(\Omega)=1.$ 
    - \* Something has to happen!
    - \* Probabilities sum/integrate to 1.
  - $-P(A \cup B) = P(A) + P(B), \ \forall A, B \in S, \ A \cup B = \emptyset.$ 
    - \* The probability of disjoint (mutually exclusive) events is equal to the sum of their individual probabilities.

<sup>&</sup>lt;sup>1</sup>This is sometimes called the *frequentist* interpretation of probability. There are other possibilities, such as *Bayesian* interpretations of probability, which describe probabilities as degrees of belief.

## 7.2 Definitions and properties of probability

- Joint probability:  $P(A \cap B)$ . The probability that the two events will occur in one realization of the experiment.
- Law of total probability:  $P(A) = P(A \cap B) + P(A \cap B^{C})$ .
- Addition rule:  $P(A \cup B) = P(A) + P(B) P(A \cap B)$ .
- Conditional probability:  $P(A|B) = \frac{P(A \cap B)}{P(B)}$ 
  - the probability of event A occurring given that event B has already occurred is the probability that both A and B occur together devided by the probability that event B occurs
- Bayes theorem:  $P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}$ 
  - -P(A|B): the probability of event A occurring given that B is true
  - -P(B|A): the probability of event B occurring given that A is true.

## 7.3 Random variables and probability distributions

- A random variable is a function  $(X : \Omega \to \mathbb{R})$  of the outcome of a random generative process. Informally, it is a "placeholder" for whatever will be the output of a process we're studying.
- A **probability distribution** describes the probabilities associated with the values of a random variable.
- Random variables (and probability distributions) can be discrete or continuous.

### 7.3.1 Discrete random variables and probability distributions

- A sample space in which there are a (finite or infinite) countable number of outcomes
- Each realization of random process has a discrete probability of occurring.
  - $-f(X=x_i)=P(X=x_i)$  is the probability the variable takes the value  $x_i$ .

### An example

• What's the probability that we'll roll a 3 on one die roll:

$$Pr(y=3) = \frac{1}{6}$$

- If one roll of the die is an "experiment," we can think of a 3 as a "success."
- $Y \sim Bernoulli\left(\frac{1}{6}\right)$
- Fair coins are  $\sim Bernoulli(.5)$ , for example.
- More generally,  $Bernoulli(\pi)$ . We'll talk about other probability distributions soon.
  - $-\pi$  represents the probability of success.

Let's do another example on the board, using the sum of two fair dice.



• Exercise:

What's the probability that the sum of two fair dice equals 7?

### 7.3.2 Continuous random variables and probability distributions

- What happens when our outcome is continuous?
- There are an infinite number of outcomes. This makes the denominator of our fraction difficult to work with.
- The probability of the whole space must equal 1.
- The domain may not span  $-\infty$  to  $\infty$ .
  - Even space between 0 and 1 is infinite!
- Two common examples are the uniform and normal probability distributions, which we will discuss below.

## 7.4 Functions describing probability distributions

## 7.4.1 Probability Mass Function (PMF) – Discrete Variables

Probability of each occurrence encoded in probability mass function (PMF)

•  $0 \le f(x_i) \le 1$ : Probability of any value occurring must be between 0 and 1.

•  $\sum_{x} f(x_i) = 1$ : Probabilities of all values must sum to 1.





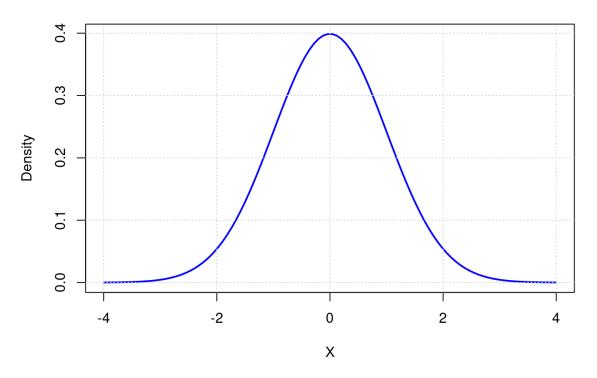
## 7.4.2 Probability Density Function (PDF) – Continuous Variables

- Similar to PMF from before, but for continuous variables.
- Using integration, it gives the probability a value falls within a particular interval

$$P[a \le X \le b] = \int_a^b f(x) \, dx$$

- Total area under the curve is 1.
- P(a < X < b) is the area under the curve between a and b (where b > a).

# Normal Distribution Curve (PDF)



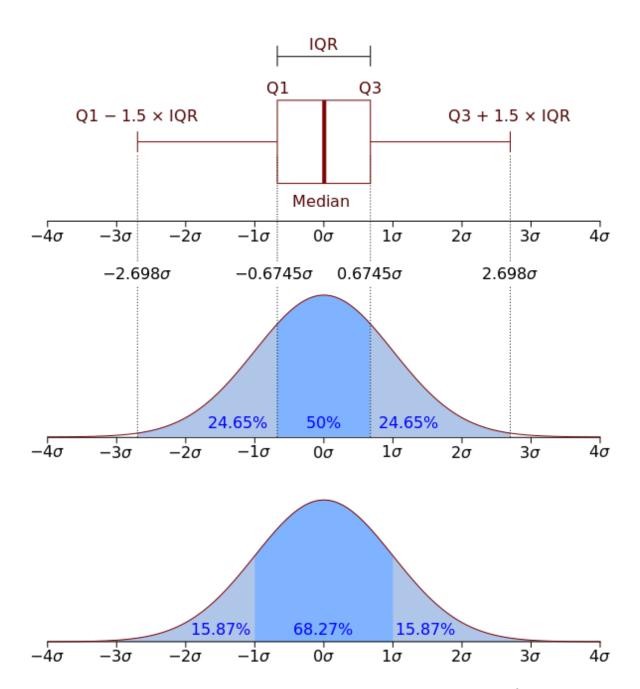


Figure 7.1: Box plot and PDF of a normal distribution  $N(0, \sigma^2)$ 

Source: Wikipedia Commons

## 7.4.3 Cumulative Density Function (CDF)

### **Discrete**

- Cumulatve density function is probability X will take a value of x or lower.
- PDF is written f(x), and CDF is written F'(x).

$$F_X(x) = Pr(X \le x)$$

• For discrete CDFs, that means summing up over all values.



What is the probability of rolling a 6 or lower with two dice? F(6) = ?

### **Continuous**

- We can't sum probabilities for continuous distributions (remember the 0 problem).
- Solution: integration

$$F_Y(y) = \int_{-\infty}^y f(y)dy$$

• Examples of uniform distribution.

## 7.5 Common types of probability distributions

There are many useful probability distributions. In this section we will cover three of the most common ones: the binomial, uniform, and normal distributions.

#### 7.5.1 Binomial distribution

A Binomial distribution is defined as follow:  $X \sim Binomial(n, p)$ 

PMF:

$${n \choose k} p^k (1-p)^{n-k}$$

, where n is the number of trials, p is the probability of success, and k is the number of successes.

Remember that:

$${n \choose k} = \frac{n!}{k!(n-k)!}$$

For example, let's say that voters choose some candidate with probability 0.02. What is the probability of seeing exactly 0 voters of the candidate in a sample of 100 people?

We can compute the PMF of a binomial distribution using R's dbinom() function.

```
dbinom(x = 0, size = 100, prob = 0.02)
```

[1] 0.1326196

```
dbinom(x = 1, size = 100, prob = 0.02)
```

[1] 0.2706522

Similarly, we can compute the CDF using R's pbinom() function:

```
pbinom(q = 0, size = 100, prob = 0.02)
```

[1] 0.1326196

```
pbinom(q = 100, size = 100, prob = 0.02)
```

[1] 1

```
pbinom(q = 1, size = 100, prob = 0.02)
```

[1] 0.4032717

### i Exercise

Compute the probability of seeing between 1 and 10 voters of the candidate in a sample of 100 people.

## 7.5.2 Uniform distribution

A uniform distribution has two parameters: a minimum and a maximum. So  $X \sim U(a,b)$ .

• PDF:

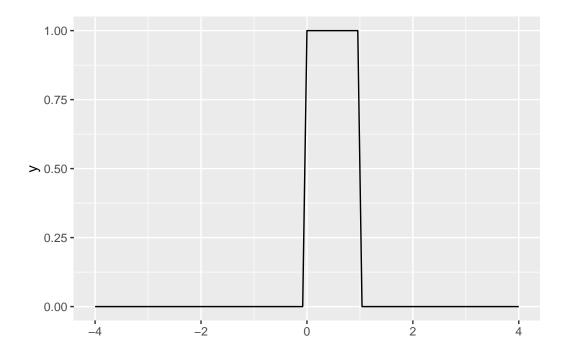
$$\begin{cases} \frac{1}{b-a} &, x \in [a,b] \\ 0 &, \text{ otherwise} \end{cases}$$

• CDF:

$$\begin{cases} 0 & , x < a \\ \frac{x-a}{b-a} & , x \in [a,b] \\ 1 & , x > b \end{cases}$$

In R, dunif() gives the PDF of a uniform distribution. By default, it is  $X \sim U(0,1)$ .

## library(tidyverse)



Meanwhile, punif() evaluates the CDF of a uniform distribution.

$$punif(q = .3)$$

[1] 0.3

## i Exercise

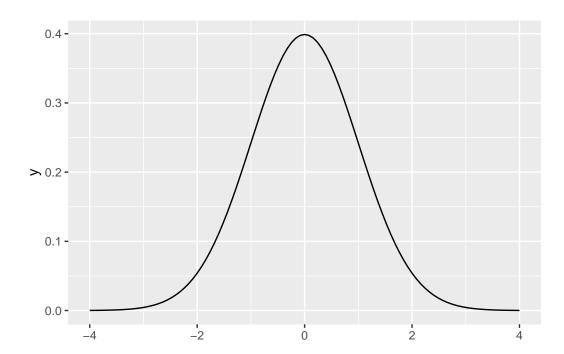
Evaluate the CDF of  $Y \sim U(-2,2)$  at point y=1. Use the formula and punif().

### 7.5.3 Normal distribution

A normal distribution has two parameters: a mean and a standard deviation. So  $X \sim N(\mu, \sigma)$ .

• PDF: 
$$2\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

In R, dnorm() gives us the PDF of a standard normal distribution  $(Z \sim N(0,1))$ :



Like you might expect, pnorm() computes the CDF of a normal distribution (by default, the standard normal).

pnorm(0)

[1] 0.5

pnorm(1) - pnorm(-1)

[1] 0.6826895

## i Exercise

What is the probability of obtaining a value above 1.96 or below -1.96 in a standard normal probability distribution? Hint: use the pnorm() function.

# 8 Statistics and simulations

```
library(tidyverse)
```

# 8.1 Random sampling

Before we jump to statistics and simulations, we'll cover how to do random sampling in R.

## 8.1.1 Random sampling from theoretical distributions

#### Uniform distribution

For the uniform distribution, the arguments specify how many draws we want and the boundaries

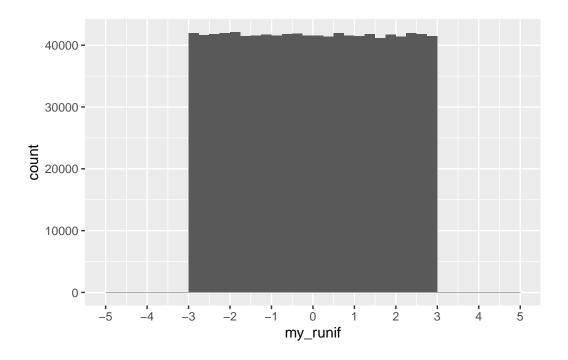
```
runif(n = 20, min = -3, max = 3)

[1] -0.06882837   1.12372959   0.48650395   -2.95095227   -1.24787813   -0.04726624
[7]   1.58911516   -2.09897256   -2.45786544   -1.05644499   1.15522877   0.44173793
[13] -2.59666444   2.55737530   -2.98601484   -2.16931064   -1.57130710   -2.09080377
[19]   2.34671277   -2.10343098
```

When we draw a million times from the distribution, we can then plot it and see that it does look as we would expect:

```
set.seed(123)
my_runif <- runif(n = 1000000, min = -3, max = 3)

ggplot(data.frame(my_runif), aes(x = my_runif)) +
   geom_histogram(binwidth = 0.25, boundary = 0, closed = "right") +
   scale_x_continuous(breaks = seq(-5, 5, 1), limits = c(-5, 5))</pre>
```



#### **Binomial distribution**

For the binomial distribution, we can specify the number of draws, how many trials each draw will have, and the probability of success.

For instance, we can ask R to do the following twenty times: flip a fair coin one hundred times, and count the number of tails.

```
rbinom(n = 20, size = 100, prob = 0.5)
```

[1] 48 45 54 50 58 50 42 58 48 57 53 49 52 51 49 40 57 53 52 41

With prob = , we can implement unfair coins:

```
rbinom(n = 20, size = 100, prob = 0.9)
```

[1] 88 87 93 95 93 92 91 94 87 91 90 92 93 89 90 95 91 90 86 88

#### Normal distribution

For the Normal or Gaussian distribution, we specify the number of draws, the mean, and standard deviation:

```
rnorm(n = 20, mean = 0, sd = 1)

[1] 1.10455864  0.06386693 -1.59684275  1.86298270 -0.90428935 -1.55158044
[7] 1.27986282 -0.32420495 -0.70015076  2.17271578  0.89778913 -0.01338538
[13] -0.74074395  0.36772316 -0.66453402 -1.11498344 -1.15067439 -0.55098894
[19]  0.10503154 -0.27183645
```

### i Exercise

Compute and plot my\_rnorm, a vector with 10,000 draws from a Normal distribution X with mean equal to -10 and standard deviation equal to 2  $(X \sim N(-10,2))$ . You can recycle code!

### 8.1.2 Random sampling from data

In this section we will work with good ol' mtcars, one of R's most notable default datasets. We'll assign it to an object so it shows in our Environment pane:

```
my_mtcars <- mtcars</pre>
```



Default datasets such as mtcars and iris are useful because they are available to everyone, and once you become familiar with them, you can start thinking about the code instead of the intricacies of the data. These qualities also make default datasets ideal for building reproducible examples (see Wickham 2014)

We can use the function sample() to obtain random values from a vector. The size = argument specifies how many values we want. For example, let's get one random value of the "mpg" column:

```
sample(my_mtcars$mpg, size = 1)
```

[1] 24.4

Every time we run this command, we can get a different result:

```
sample(my_mtcars$mpg, size = 1)
```

[1] 14.7

```
sample(my_mtcars$mpg, size = 1)
```

[1] 15.5

In some occasions we do want to get the same result consistently after running some random process multiple times. In this case, we *set a seed*, which takes advantage of R's pseudo-random number generator capabilities. No matter how many times we run the following code block, the result will be the same:

```
set.seed(123)
sample(my_mtcars$mpg, size = 1)
```

[1] 15

Sampling with replacement means that we can get the same value multiple times. For example:

```
set.seed(12)
sample(c("Banana", "Apple", "Orange"), size = 3, replace = T)
```

```
[1] "Apple" "Apple" "Orange"
```

```
sample(my_mtcars$mpg, size = 100, replace = T)
```

```
[1] 26.0 15.2 18.7 18.7 30.4 21.0 24.4 26.0 32.4 15.8 32.4 19.2 18.1 16.4 19.2 [16] 27.3 14.3 10.4 17.3 13.3 21.4 13.3 19.2 24.4 15.0 27.3 17.8 15.2 15.8 14.3 [31] 19.7 16.4 18.7 15.8 19.2 21.0 14.3 15.2 14.3 27.3 21.4 33.9 33.9 21.4 30.4 [46] 33.9 21.4 17.3 17.3 10.4 26.0 18.7 15.2 30.4 10.4 10.4 15.5 14.3 26.0 17.3 [61] 33.9 26.0 24.4 18.7 30.4 32.4 21.5 30.4 15.2 27.3 13.3 17.3 21.4 24.4 13.3 [76] 22.8 33.9 13.3 21.5 14.3 19.2 30.4 24.4 26.0 15.8 10.4 24.4 14.3 15.2 10.4 [91] 19.2 21.0 16.4 19.2 24.4 19.7 18.7 10.4 18.7 17.8
```

In order to sample not from a vector but from a data frame's rows, we can use the slice\_sample() function from dplyr:

```
my_mtcars |>
  slice_sample(n = 2) # a number of rows
```

```
mpg cyl disp hp drat wt qsec vs am gear carb
Dodge Challenger 15.5 8 318 150 2.76 3.52 16.87 0 0 3 2
Datsun 710 22.8 4 108 93 3.85 2.32 18.61 1 1 4 1
```

```
my_mtcars |>
slice_sample(prop = 0.5) # a proportion of rows
```

```
mpg cyl disp hp drat
                                                 qsec vs am gear carb
                                              wt
Toyota Corolla
                          4 71.1 65 4.22 1.835 19.90
                    33.9
                                                        1
                                                           1
                    19.7
                          6 145.0 175 3.62 2.770 15.50
                                                                     6
Ferrari Dino
                                                           1
                                                                5
                                                                     3
Merc 450SE
                    16.4
                          8 275.8 180 3.07 4.070 17.40
Hornet Sportabout
                   18.7
                          8 360.0 175 3.15 3.440 17.02 0
                                                                     2
Maserati Bora
                   15.0
                          8 301.0 335 3.54 3.570 14.60
                                                                5
                                                                     8
                                                        0
                                                          1
Datsun 710
                    22.8
                          4 108.0 93 3.85 2.320 18.61
                                                          1
                                                                4
                                                                     1
                                                        1
                          8 351.0 264 4.22 3.170 14.50
Ford Pantera L
                    15.8
                                                       0
                                                          1
                                                                5
                    15.5
                          8 318.0 150 2.76 3.520 16.87
                                                                     2
Dodge Challenger
                                                        0 0
                                                                3
                    19.2
                          6 167.6 123 3.92 3.440 18.30
                                                                     4
Merc 280
                                                           0
                                                                4
Lincoln Continental 10.4
                          8 460.0 215 3.00 5.424 17.82
Valiant
                   18.1
                          6 225.0 105 2.76 3.460 20.22
                                                                3
                                                        1
                                                           0
                                                                     1
Fiat 128
                   32.4 4 78.7 66 4.08 2.200 19.47
                                                        1 1
                                                                4
                                                                     1
Mazda RX4 Wag
                   21.0
                          6 160.0 110 3.90 2.875 17.02 0 1
                                                                4
                                                                     4
Merc 240D
                          4 146.7 62 3.69 3.190 20.00
                                                                     2
                   24.4
                                                       1 0
                                                                4
Camaro Z28
                   13.3
                          8 350.0 245 3.73 3.840 15.41
                                                                     4
                                                        0 0
                                                                3
Cadillac Fleetwood 10.4
                          8 472.0 205 2.93 5.250 17.98 0 0
                                                                3
                                                                     4
```

Again, we can also use seeds here to ensure that we'll get the same result each time:

```
set.seed(123)
my_mtcars |>
slice_sample(prop = 0.5)
```

```
mpg cyl disp hp drat
                                            wt qsec vs am gear carb
Maserati Bora
                         8 301.0 335 3.54 3.570 14.60
                  15.0
Cadillac Fleetwood 10.4
                         8 472.0 205 2.93 5.250 17.98 0 0
                                                                  4
Honda Civic
                  30.4
                        4 75.7 52 4.93 1.615 18.52 1 1
                                                                  2
Merc 450SLC
                  15.2
                         8 275.8 180 3.07 3.780 18.00 0 0
                                                                  3
                                                             3
Datsun 710
                  22.8
                         4 108.0 93 3.85 2.320 18.61 1 1
```

```
Merc 280
                   19.2
                           6 167.6 123 3.92 3.440 18.30
                                                                       4
Fiat 128
                   32.4
                           4 78.7 66 4.08 2.200 19.47
                                                                       1
Dodge Challenger
                   15.5
                          8 318.0 150 2.76 3.520 16.87
                                                                  3
                                                                       2
Merc 280C
                   17.8
                           6 167.6 123 3.92 3.440 18.90
                                                                  4
                                                                       4
                                                                       2
Hornet Sportabout
                   18.7
                           8 360.0 175 3.15 3.440 17.02
                                                                  3
                                   65 4.22 1.835 19.90
Toyota Corolla
                   33.9
                           4 71.1
                                                                       1
Ford Pantera L
                   15.8
                          8 351.0 264 4.22 3.170 14.50
                                                                       4
AMC Javelin
                   15.2
                          8 304.0 150 3.15 3.435 17.30
                                                                       2
Ferrari Dino
                           6 145.0 175 3.62 2.770 15.50
                                                                  5
                                                                       6
                   19.7
                                                                       2
Merc 230
                   22.8
                           4 140.8 95 3.92 3.150 22.90
                                                                  4
                           4 95.1 113 3.77 1.513 16.90
                                                                       2
                   30.4
                                                                  5
Lotus Europa
```

### i Exercise

Use slice\_sample() to sample 32 rows from mtcars with replacement.

### 8.2 Statistics

The problems considered by probability and statistics are inverse to each other. In probability theory we consider some underlying process which has some randomness or uncertainty modeled by random variables, and we figure out what happens. In statistics we observe something that has happened, and try to figure out what underlying process would explain those observations. (quote attributed to Persi Diaconis)

- In statistics we try to learn about a **data-generating process** (DGP) using our observed data. Examples: surveys, GDP statistics.
- Usually we are restrained to **samples**, while our DGPs of interest are **population-based**.
  - So we use random sampling or refer to superpopulations as a way to justify how the data we observe can approximate the population.
- Statistics has two main targets:
  - Estimation: how we find a reasonable guess of an unknown property (parameter) of a DGP
  - **Inference**: how we describe uncertainty about our estimate
- We use an **estimator**  $\hat{\theta}(\cdot)$ , which is a function that summarizes data, as a guess about a parameter  $\theta$ . A guess generated by an estimator in a given sample is called an **estimate**.

### i Exercise

Suppose there's a uniform distribution  $X \sim U(0, \text{unknown})$  out there.<sup>1</sup> We want to use a sample from this distribution (let's say of n=30 observations) to estimate the unknown upper bound.

Discuss: would the sample maximum be a good estimator? Why or why not?

- Theoretical statistics is all about finding "good" estimators. A few properties of good estimators:
  - **Unbiasedness**: Across multiple random samples, an unbiased estimator gets the right answer on average.
  - Low variance: Across multiple random samples, a low-variance estimator is more concentrated around the true parameter.
  - BUT it's sometimes hard to get both unbiasedness and low variance. So we have to make sacrifices. We usually quantify this via the mean squared error:  $MSE = bias^2 + variance$ . Comparing two estimators, the one with the lowest MSE is said to be more **efficient**.
  - Consistency: A consistent estimator converges in probability to the true value. "If we had enough data, the probability that our estimate would be far from the truth would be close to zero" (Aronow and Miller 2019, p. 105).
- Applied statistics is about using these techniques reasonably in messy real-world situations...

### 8.3 Simulations

- In simulations, we generate fake data following standard procedures. Why?
  - To better understand how our estimators work in different settings (the methods reason)
  - To get insights about complex processes with many moving parts (the substantive reason) (let's talk about gerrymandering).

#### i Exercise

Simulate drawing an n=30 random sample from a  $X \sim U(0, 10)$  distribution and take its maximum value.

<sup>&</sup>lt;sup>1</sup>This is a fascinating distribution with a rich history, and it is used in many statistical textbooks (Whittinghill and Hogg, 2001). It is thoroughly covered in the UT SDS Mathematical Statistics sequence.

## 8.3.1 Loops

Loops allow us to repeat operations in R. The most common construct is the for-loop:

```
for (i in 1:10){
  print(i)
}
```

- [1] 1
- [1] 2
- [1] 3
- [1] 4
- [1] 5
- [1] 6
- [1] 7
- [1] 8
- [1] 9
- [1] 10

It's common to perform operations at each step and save the results. We typically create an empty object and "fill it in" at each step:

```
results <- double(10)
for (i in 1:10){
  results[i] <- i ^ 2
}</pre>
```

#### results

[1] 1 4 9 16 25 36 49 64 81 100

```
i Functional loops

Another way to do loops is with the *apply() family of functions:

sapply(1:10, function(x){x ^ 2})

[1] 1 4 9 16 25 36 49 64 81 100
```

We talked about loops and various extensions in one of our methods workshops last year: Speedy R.

### 8.3.2 An example simulation

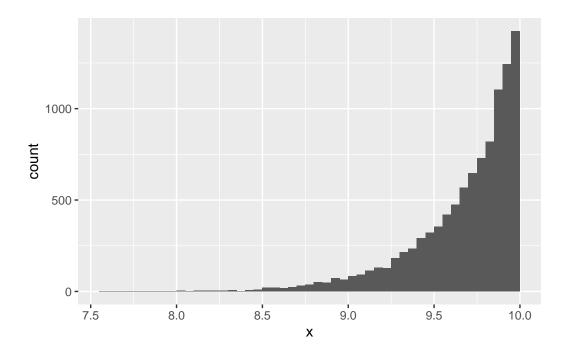
We will simulate our exercise from above 10,000 times:

Now we can analyze our simulated estimates:

```
mean(simulated_estimates)
```

[1] 9.677225

```
ggplot(data.frame(x = simulated_estimates), aes(x = x)) +
geom_histogram(binwidth = 0.05, boundary = 0, closed = "right")
```



### i Exercise

We just simulated to evaluate the sample maximum as an estimator. Modify the code above to evaluate the following two estimators:

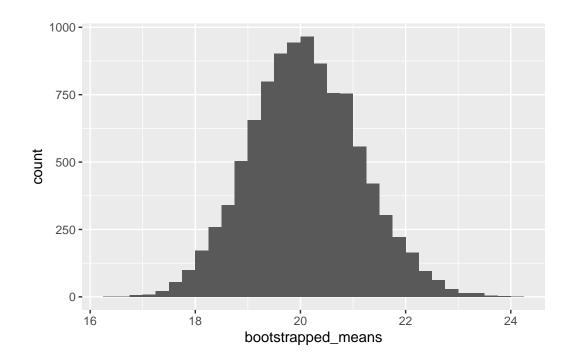
- (sample maximum)  $\cdot \frac{(n+1)}{n}$
- $2 \cdot (\text{sample mean})$

### 8.3.3 Another example simulation: bootstrapping

Bootstrap (and its relatives) is one way in which we can do *inference*, i.e., assess uncertainty. (We'll go through the intuition on the board.)

```
# set seed an number of simulations
set.seed(1)
k <- 10000
bootstrapped_means <- double(k)
for (i in 1:k){
  m <- my_mtcars |> slice_sample(prop = 1, replace = T)
  bootstrapped_means[i] <- mean(m$mpg)
}</pre>
```

```
ggplot(data.frame(bootstrapped_means), aes(x = bootstrapped_means)) +
geom_histogram(binwidth = 0.25, boundary = 0, closed = "right")
```



# References

- Arel-Bundock, Vincent, Nils Enevoldsen, and CJ Yetman. 2018. "Countrycode: An r Package to Convert Country Names and Country Codes." *Journal of Open Source Software* 3 (28): 848. https://doi.org/10.21105/joss.00848.
- Aronow, Peter M, and Benjamin T Miller. 2019. Foundations of Agnostic Statistics. Cambridge University Press.
- Bank, World. 2023. "World Bank Open Data." https://data.worldbank.org/.
- Baydin, Atılım Günes, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. "Automatic Differentiation in Machine Learning: A Survey." The Journal of Machine Learning Research 18 (1): 5595–5637.
- Coppedge, Michael, John Gerring, Carl Henrik Knutsen, Staffan I. Lindberg, Jan Teorell, David Altman, Michael Bernhard, et al. 2022. "V-Dem Codebook V12." Varieties of Democracy (V-Dem) Project. https://www.v-dem.net/dsarchive.html.
- Dahlberg, Stefan, Aksen Sundström, Sören Holmberg, Bo Rothstein, Natalia Alvarado Pachon, Cem Mert Dalli, and Yente Meijers. 2023. "The Quality of Government Basic Dataset, Version Jan23." University of Gothenburg: The Quality of Government Institute. https://www.gu.se/en/quality-government doi:10.18157/qogbasjan23.
- FiveThirtyEight. 2021. "Tracking Congress In The Age Of Trump [Dataset]." https://projects.fivethirtyeight.com/congress-trump-score/.
- Imai, Kosuke, and Nora Webb Williams. 2022. Quantitative Social Science: An Introduction in Tidyverse. Princeton; Oxford: Princeton University Press.
- Moore, Will H., and David A. Siegel. 2013. A Mathematics Course for Political and Social Research. Princeton, NJ: Princeton University Pres.
- Pontin, Jason. 2007. "Oppenheimer's Ghost." MIT Technology Review, October 15, 2007. https://www.technologyreview.com/2007/10/15/223531/oppenheimers-ghost-3/.
- Robinson, David. 2020. Fuzzyjoin: Join Tables Together on Inexact Matching. https://github.com/dgrtwo/fuzzyjoin.
- Rossi, Hugo. 1996. "Mathematics Is an Edifice, Not a Toolbox." *Notices of the AMS* 43 (10): 1108.
- Smith, Danny. 2020. Survey Research Datasets and R. https://socialresearchcentre.github.io/r\_survey\_datasets/.
- U. S. Department of Agriculture [USDA], Agricultural Research Service. 2019. "Department of Agricultural Research Service." https://fdc.nal.usda.gov/.
- Whittinghill, Dexter C, and Robert V Hogg. 2001. "A Little Uniform Density with Big Instructional Potential." *Journal of Statistics Education* 9 (2).

Wickham, Hadley. 2014. "Tidy Data." Journal of Statistical Software 59 (10). https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, Danielle Navarro, and Thomas Lin Pedersen. 2023. *Ggplot2: Elegant Graphics for Data Analysis*. 3rd ed. https://ggplot2-book.org/.