Methods Camp

UT Austin, Department of Government

Andrés Cruz and Matt Martin

2023-08-10

Table of contents

Class schedule **6**
Description . . . 6
Course outline . . . 6
Contact info . . . 8

Setup **9**
Installing R and RStudio . . . 9
Setting up for Methods Camp . . . 11

1 Intro to R **12**
1.1 Objects . . . 12
1.2 Vectors and functions . . . 13
1.3 Data frames and lists . . . 16
1.4 Packages . . . 18

2 Tidy data analysis I **20**
2.1 Loading data . . . 20
2.2 Wrangling data with `dplyr` . . . 22
2.2.1 Selecting columns . . . 22
2.2.2 Renaming columns . . . 27
2.2.3 Creating columns . . . 28
2.2.4 Filtering rows . . . 29
2.2.5 Ordering rows . . . 31
2.2.6 Summarizing data . . . 34
2.2.7 Overview . . . 35
2.3 Visualizing data with `ggplot2` . . . 35
2.3.1 Univariate plots: categorical . . . 36
2.3.2 Univariate plots: numerical . . . 38
2.3.3 Bivariate plots . . . 42

3 Matrices **47**
3.1 Introduction . . . 47
3.1.1 Scalars . . . 47
3.1.2 Vectors . . . 47
3.2 Operators . . . 48
3.2.1 Summation . . . 48

3.2.2 Product . . . 49
3.2.3 Basics . . . 50
3.2.4 Structure . . . 51
3.3 Matrix operations . . . 53
3.3.1 Addition and subtraction . . . 53
3.3.2 Scalar multiplication . . . 55
3.3.3 Matrix multiplication . . . 56
3.3.4 Multiplication steps . . . 57
3.4 Properties . . . 60
3.5 Special matrices . . . 60
3.6 Transpose . . . 61
3.7 Inverse . . . 62
3.8 Linear systems and matrices . . . 63
3.9 OLS and matrices . . . 64
3.9.1 Dependent variable . . . 64
3.9.2 Independent variables . . . 64
3.9.3 Linear regression model . . . 65
3.9.4 Estimates . . . 65

4 Tidy data analysis II 69

5 Functions and loops 70
5.1 Basics . . . 70
5.1.1 What is a function? . . . 70
5.1.2 Function machine . . . 70
5.1.3 Visualization . . . 71
5.2 Types of functions . . . 71
5.2.1 Linear functions . . . 71
5.2.2 Quadratic . . . 72
5.2.3 Cubic . . . 72
5.2.4 Polynomial . . . 73
5.2.5 Exponential . . . 73
5.2.6 Trigonometric functions . . . 73
5.3 Logarithms and exponents . . . 74
5.3.1 Logarithms . . . 74
5.3.2 Relationships . . . 74
5.3.3 Basic rules . . . 74
5.3.4 Natural logarithms . . . 75
5.3.5 Definition of e . . . 75
5.4 Functions of functions . . . 76
5.4.1 Basics . . . 76
5.4.2 PMF, PDF, and CDF . . . 76

6 Calculus **77**
6.1 Derviatives . . . 77
6.1.1 Calculating derivatives . . . 77
6.1.2 Notation . . . 78
6.1.3 Special functions . . . 81
6.1.4 Derivatives with addition and substraction . . . 81
6.2 Advanced rules . . . 82
6.2.1 Product rule . . . 82
6.2.2 Quotient rule . . . 82
6.2.3 Chain rule . . . 83
6.2.4 Second derivative . . . 84
6.3 Differentiable and Continuous Functions . . . 85
6.3.1 When is f not differentiable? . . . 86
6.4 Extrema and optimization . . . 86
6.4.1 Extrema . . . 87
6.4.2 Identifying extrema . . . 87
6.4.3 Minimum or maximum? . . . 88
6.4.4 Second derivatives . . . 88
6.4.5 Local vs. Global Extrema . . . 88
6.5 Partial derivatives . . . 89
6.5.1 Application . . . 90
6.6 Integrals . . . 90
6.6.1 Area under a curve . . . 90
6.6.2 Integrals as summation . . . 91
6.6.3 Definite integrals . . . 91
6.6.4 Indefinite integrals . . . 92
6.6.5 Solving definite integrals . . . 92
6.6.6 Constants . . . 92
6.6.7 Rules of integration . . . 93
6.6.8 More rules . . . 93
6.6.9 Solving the problem . . . 94
6.6.10 Integration by parts . . . 95

7 Probability **96**
7.1 What is probability? . . . 96
7.2 Kolmogorov's axioms . . . 96
7.3 Some definitions . . . 97
7.4 Discrete probability . . . 97
7.4.1 Probability Mass Function (PMF) . . . 98
7.4.2 Discrete distribution . . . 98
7.5 Continuous probability . . . 99
7.5.1 Basics . . . 99
7.5.2 Probability Density Function (PDF) . . . 99

7.6 Cumulative Density Function (CDF) . . . 99
7.6.1 Discrete . . . 99
7.6.2 Continuous . . . 100
7.7 Statistics . . . 100
7.7.1 Introduction . . . 100
7.7.2 Univariate statistics . . . 100
7.7.3 Examples of univariate statistics . . . 100
7.7.4 Measures of central tendency . . . 101
7.7.5 Deviations from central tendency . . . 101
7.7.6 Variance . . . 101
7.7.7 Standard deviation . . . 101
7.8 Bivariate statistics . . . 102
7.8.1 Covariance . . . 102
7.8.2 Correlation . . . 102
7.9 Regression . . . 102
7.9.1 Ordinary least squares . . . 102
7.9.2 Residuals . . . 103
7.9.3 Finding the right line . . . 103

8 Simulations **104**

9 Text analysis **105**

10 Wrap up **106**
10.1 Methods at UT . . . 106
10.1.1 Required methods courses . . . 106
10.1.2 Other methods courses . . . 106
10.1.3 More courses . . . 106
10.1.4 Other departments at UT . . . 107
10.1.5 Other resources . . . 107

References **108**

Class schedule

Date	Time	Location
Thurs, Aug. 10	9:00 AM - 4:00 PM	RLP 1.302E
Fri, Aug. 11	9:00 AM - 4:00 PM	RLP 1.302E
Sat, Aug. 12	No class	-
Sun, Aug. 13	No class	-
Mon, Aug. 14	9:00 AM - 4:00 PM	RLP 1.302E
Tues, Aug. 15	9:00 AM - 4:00 PM	RLP 1.302E
Weds, Aug. 16	9:00 AM - 4:00 PM	RLP 1.302E

On class days, we will have a lunch break from 12:00-1:00 PM. We'll also take short breaks periodically during the morning and afternoon sessions as needed.

Description

Welcome to Introduction to Methods for Political Science, aka "Methods Camp"! In the past our incoming students have told us their math skills are rusty and they would like to be better prepared for UT's methods courses. Methods Camp is designed to give everyone a chance to brush up on some skills in preparation for the Stats I and Formal Theory I courses. The other goal of Methods Camp is to allow you to get to know your cohort. We hope that struggling with matrix algebra and the dreaded chain rule will still prove to be a good bonding exercise.

As you can see from the above schedule, we'll be meeting on Thursday, August 10th and Friday, August 11th as well as from Monday, August 14th through Wednesday, August 16th. Classes at UT begin the start of the following week on Monday, August 22nd. Below is a tentaive schedule outlining what will be covered in the class, although we may rearrange things a bit if we find we're going too slowly or too quickly through any of the material.

Course outline

1 Thursday morning: R and RStudio

- Introductions
- RStudio (materials are on the website as zipped RStudio projects)
- Objects (vectors, matrices, data frames)
- Basic functions (`mean()`, `length()`, etc.)

2 Thursday afternoon: `tidyverse` basics I

- Packages: installation and loading (including the tidyverse)
- Data wrangling with dplyr (basic verbs, including the new `.by =` syntax)
- Data visualization basics with `ggplot2`
- Data loading (.csv, .rds, .dta, .xlsx)
- Quarto fundamentals

3 Friday 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

4 Friday afternoon: `tidyverse` basics II

- Data merging and pivoting (`*_join()`, `pivot_*()`)
- Value recoding (`if_else()`, `case_when()`)
- Missing values
- Data visualization extensions: facets, text annotations

5 Monday morning: Functions and loops

- Functions
- For-loops and `lapply()`
- Finding R help (help files, effective Googling, ChatGPT)

6 Monday afternoon: Calculus

- Limits (not sure how to teach this in an R-centric way yet, but there must be a way)
- Derivatives (symbolic, numerical, automatic)
- Integrals

7 Tuesday morning: Probability

- Concepts: probability, random variables, etc.
- PMF, PDF, CDF, etc.
- Distributions (binomial, normal; different functions in R and how to use them)
- Expectation and variance

8 Tuesday afternoon: Simulations

- Simulations (ideas, seed setting, etc.)
- Sampling
- Bootstrapping

9 Wednesday morning: Text analysis

- String manipulation with `stringr`
- Simple text analysis (counts, tf-idf, etc.) with `tidytext` and visualization

10 Wednesday afternoon: Wrap-up

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

Contact info

If you have any questions during or outside of methods camp, you can contact Andrés at andres.cruz@utexas.edu and Matt at mjmartin@utexas.edu.

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, follow the instructions below):

1. Download and install R from the official website, CRAN. Click on "Download R for <Windows/Mac>" 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 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.

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

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

- `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…

[1] 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!

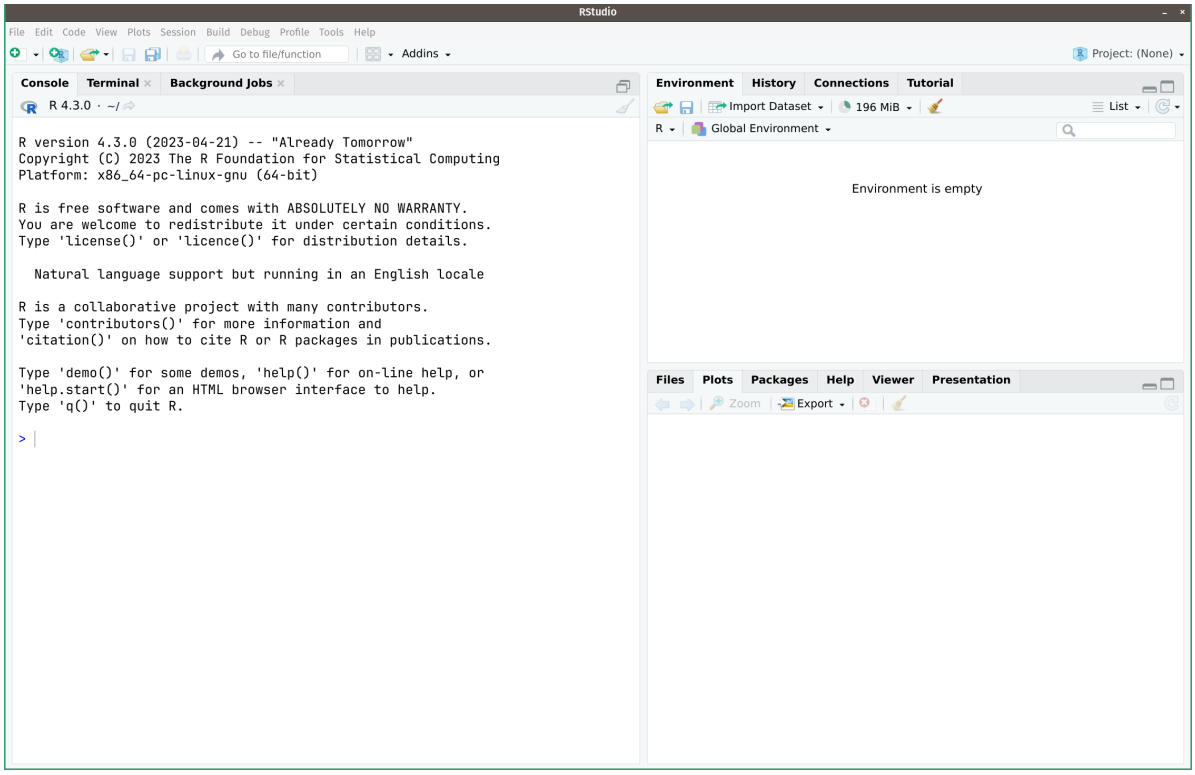


Figure 1: How RStudio looks after a clean installation.

Setting up for Methods Camp

All materials for Methods Camp are both on this website and an RStudio project. An RStudio project is simply a folder where one keeps scripts, datasets, and other files needed for a data analysis project.

You can download our RStudio project here, as a .zip compressed file. On MacOS, the file will be uncompressed automatically. On Windows, you should do `Right click > Extract all`.

> **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/` on your computer. Navigate to the `methodscamp.Rproj` file within it and open it. RStudio should open the project right away. You should see `methodscamp` on the top-right of RStudio—this indicates that you are working in our RStudio project.

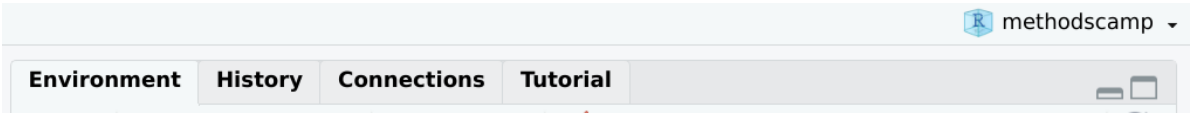


Figure 2: How the bottom-right corner of RStudio looks after opening our project.

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,[2] which contains both code and explanations (you can also read in the next chapter of this website).

[2] 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 ^ 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.

> **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
```

```
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
```

```
my_object2
```

```
[1] 20
```

You can delete objects with the `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 <- c(6, -11, my_object, 0, 20)
```

```
my_vector
```

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

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
```

```
median(my_vector)
```

```
[1] 6
```

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

```
[1] 11.53256
```

```
sum(my_vector)
```

```
[1] 25
```

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

```
[1] -11
```

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

```
[1] 20
```

```r
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:

```r
my_mean <- mean(my_vector)
```

```r
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*:

```r
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`).

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

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

> **Tip**
>
> 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
```

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`.

> **Exercise**
>
> Examine the help file of the `log()` function. How can we compute the 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")
```

```
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 <- data.frame(fruit = my_character_vector,
                            calories_per_100g = c(52, 47, 30, 89),
                            water_per_100g = c(85.6, 86.8, 91.4, 74.9))
```

```
my_data_frame
```

```
      fruit calories_per_100g water_per_100g
1     Apple                52           85.6
2    Orange                47           86.8
3 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
```

> **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:

```r
my_list <- list(my_object, my_vector, my_data_frame)
```

```r
my_list
```

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

[[2]]
[1]   6 -11  10   0  20

[[3]]
       fruit calories_per_100g water_per_100g
1      Apple                52           85.6
2     Orange                47           86.8
3 Watermelon                30           91.4
4     Banana                89           74.9
```

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

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

```
[1] 10
```

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.

```r
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.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

> **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     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

> **Tip**
>
> 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 transformation and wrangling, 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")
```

```
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
```

```
# 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 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`

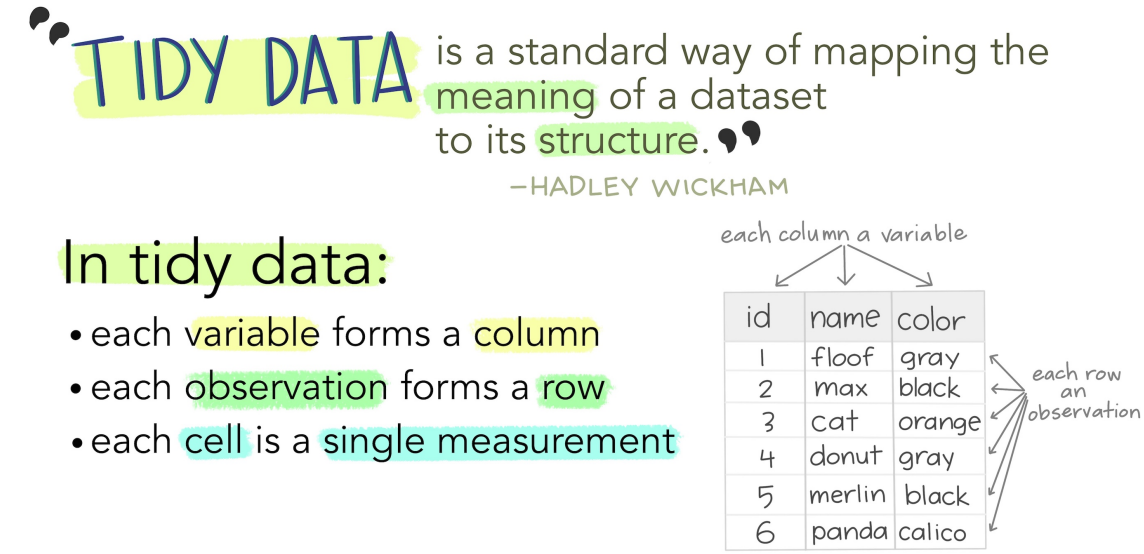
We often need to modify data to conduct our analyses, e.g., creating columns, filtering rows, etc. In the `tidyverse`, these operations are conducted with multiple *verbs*, which we will review now.

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
```



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

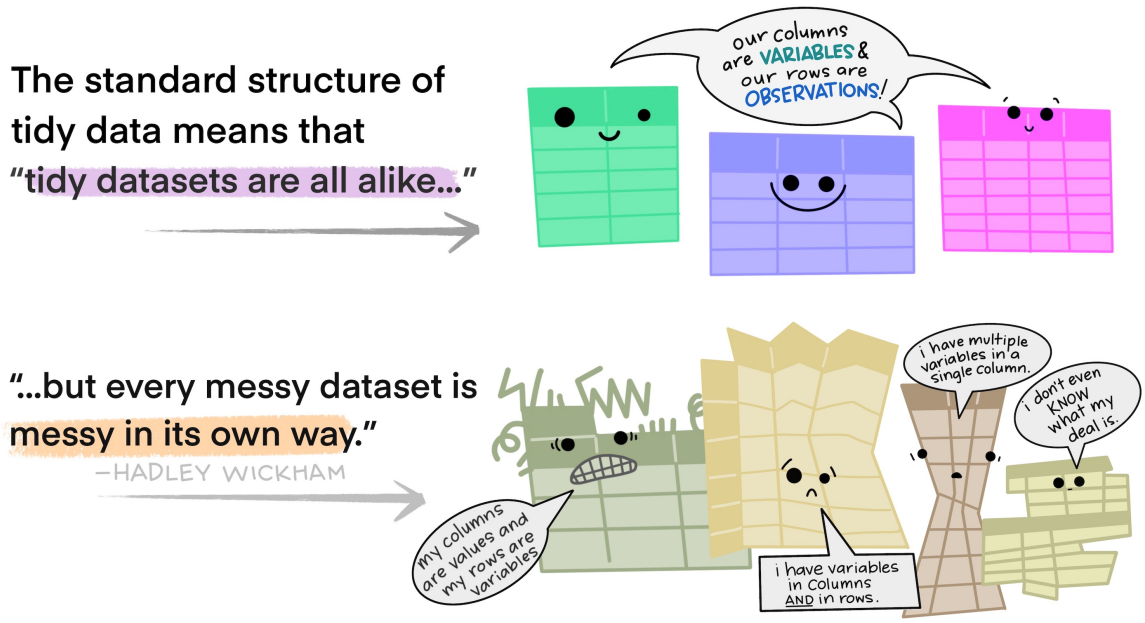


Figure 2.1: Source: Illustrations from the Openscapes blog *Tidy Data for reproducibility, efficiency, and collaboration* by Julia Lowndes and Allison Horst.

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  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
```

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

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

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

```
# A tibble: 122 x 3
   last_name  agree 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 0.203      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>     <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 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
   <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 0.203      0.294
# i 112 more rows
```

> **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> <chr>     <dbl>      <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           121      0.893           0.560
 5 B001230  Baldwin    WI    D           128      0.227           0.510
 6 B001236  Boozman    AR    R           129      0.915           0.851
 7 B001243  Blackburn  TN    R           131      0.885           0.889
 8 B001261  Barrasso   WY    R           129      0.891           0.895
 9 B001267  Bennet     CO    D           121      0.273           0.417
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      0.203           0.294
# 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:

```
trump_scores |>
  select(last_name, agree, agree_pred) |> # select just for clarity
  mutate(pct_agree = 100 * agree,
         pct_agree_pred = 100 * agree_pred)
```

```
# 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      0.787      90.6           78.7
 3 Brown      0.258      0.642      25.8           64.2
 4 Burr       0.893      0.560      89.3           56.0
 5 Baldwin    0.227      0.510      22.7           51.0
 6 Boozman    0.915      0.851      91.5           85.1
 7 Blackburn  0.885      0.889      88.5           88.9
 8 Barrasso   0.891      0.895      89.1           89.5
 9 Bennet     0.273      0.417      27.3           41.7
10 Blumenthal 0.203      0.294      20.3           29.4
# 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:

```r
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>          <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:

```r
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      0.318      -11.4
 9 C001070  Casey      PA    D           129 0.287      0.508        0.724
10 C001088  Coons      DE    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)
`|`	"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    R           118 0.890      0.856         26.0
 2 B000575  Blunt     MO    R           128 0.906      0.787         18.6
 3 B001135  Burr      NC    R           121 0.893      0.560          3.66
 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    R           129 0.891      0.895         46.3
 7 B001310  Braun     IN    R            44 0.909      0.713         19.2
 8 C000567  Cochran   MS    R            68 0.971      0.830         17.8
 9 C000880  Crapo     ID    R           125 0.904      0.870         31.8
10 C001035  Collins   ME    R           129 0.651      0.441         -2.96
# i 59 more rows
```

```
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    R           129 0.922      0.659         9.00
2 C001098  Cruz      TX    R           126 0.921      0.663         9.00
3 F000062  Feinstein CA    D           128 0.242      0.201       -30.1
4 H001075  Harris    CA    D           116 0.164      0.209       -30.1
```

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

```
# A tibble: 1 x 8
  bioguide last_name state party num_votes agree agree_pred margin_trump
  <chr>    <chr>     <chr> <chr>     <dbl> <dbl>      <dbl>        <dbl>
1 M001183  Manchin   WV    D           129 0.504      0.893         42.2
```

> 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      TN    R             2 0         0.115         26.0
 3 L000570  Luján        NM    D           186 0.124     0.243         -8.21
 4 G000555  Gillibrand   NY    D           121 0.124     0.242        -22.5
 5 M001176  Merkley      OR    D           129 0.155     0.323        -11.0
 6 W000817  Warren       MA    D           116 0.155     0.216        -27.2
 7 B001288  Booker       NJ    D           119 0.160     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      NM    D           186 0.124      0.243        -8.21
 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     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     MA    D           127 0.165      0.213       -27.2
 9 W000779  Wyden      OR    D           129 0.186      0.323       -11.0
10 B001277  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  Cochran   MS    R            68 0.971      0.830        17.8
 3 H000338  Hatch     UT    R            84 0.964      0.825        18.1
 4 M001197  McSally   AZ    R           136 0.949      0.562         3.55
 5 P000612  Perdue    GA    R           119 0.941      0.606         5.16
 6 C001096  Cramer    ND    R           135 0.941      0.908        35.7
 7 R000307  Roberts   KS    R           127 0.937      0.818        20.6
 8 C001056  Cornyn    TX    R           129 0.922      0.659         9.00
 9 H001061  Hoeven    ND    R           129 0.922      0.883        35.7
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    OR    D           129 0.155      0.323       -11.0
 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     MA    D           127 0.165      0.213       -27.2
 9 W000779  Wyden      OR    D           129 0.186      0.323       -11.0
10 B001277  Blumenthal CT    D           128 0.203      0.294       -13.6
# i 105 more rows
```

> **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:

```
trump_scores |>
  summarize(mean_agree = mean(agree),
            mean_agree_pred = mean(agree_pred))
```

```
# A tibble: 1 x 2
  mean_agree mean_agree_pred
       <dbl>           <dbl>
1      0.592           0.572
```

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

```
trump_scores |>
  filter(num_votes >= 5) |> # to filter out senators with few data points
  summarize(max_agree = max(agree),
            min_agree = min(agree))
```

```
# A tibble: 1 x 2
  max_agree min_agree
      <dbl>     <dbl>
1         1     0.124
```

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

```
trump_scores |>
  filter(num_votes >= 5) |> # to filter out senators with few data points
  summarize(mean_agree = mean(agree),
            max_agree = max(agree),
```

```
            min_agree = min(agree),
            .by = party) # to group by party
```

```
# A tibble: 2 x 4
  party mean_agree max_agree min_agree
  <chr>      <dbl>     <dbl>     <dbl>
1 R          0.876     1         0.651
2 D          0.272     0.548     0.124
```

> **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
`mutate()`	Creating columns
`filter()`	Filtering rows
`arrange()`	Ordering rows
`summarize()`	Summarizing data
`summarize(…, .by = )`	Summarizing data (by group)

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:

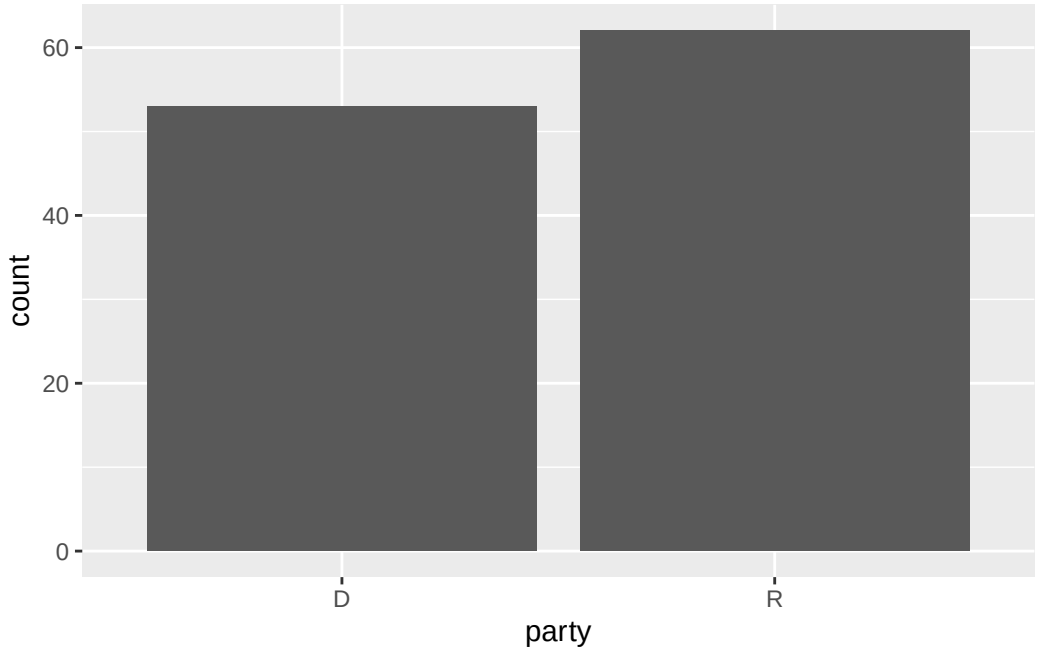
- A `ggplot()` command with a `data =` argument specifying a data frame and a `mapping = aes()` argument specifying "aesthetic mappings," i.e., how we want to use the columns in the data frame in the plot (for example, in the x-axis, as color, etc.).
- "geoms," such as `geom_bar()` or `geom_point()`, 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:

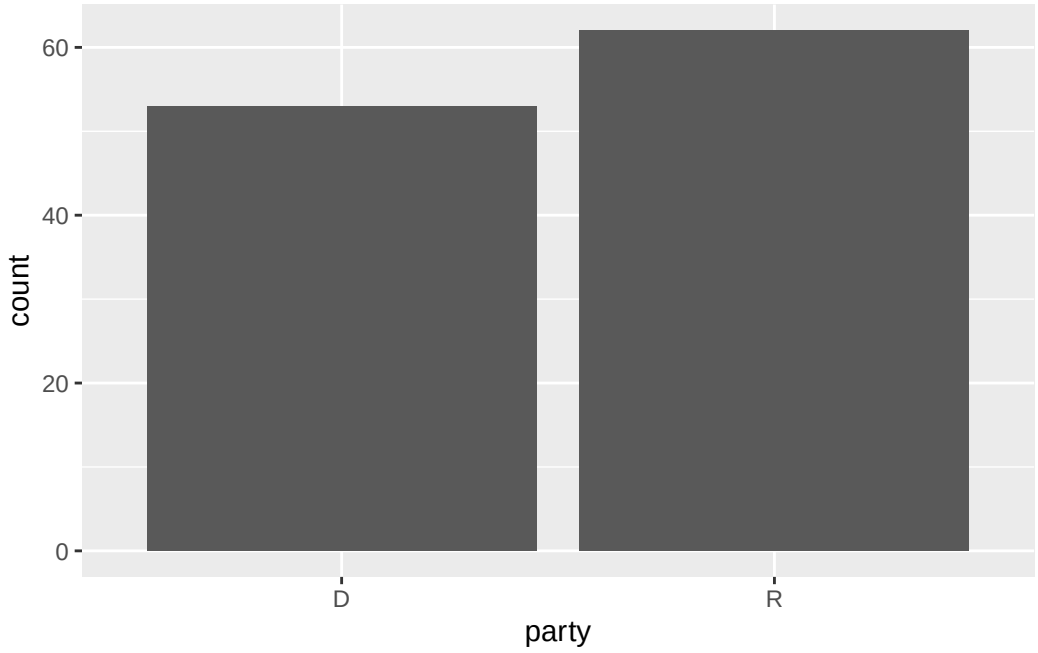
```
ggplot(data = trump_scores_ss, mapping = aes(x = party)) +
  geom_bar()
```



Tip

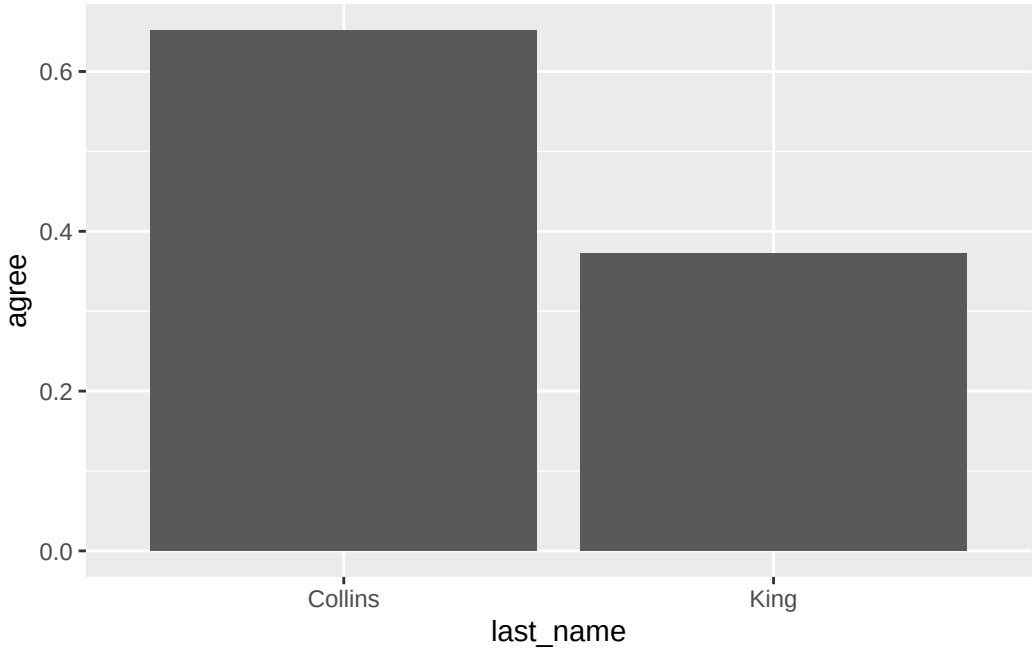
As with any other function, we can drop the argument names if we specify the argument values in order. This is common in `ggplot2` code:

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



Notice how `geom_bar()` 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 `geom_col()` geom specifying both `x` and `y` 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()
```



Exercise

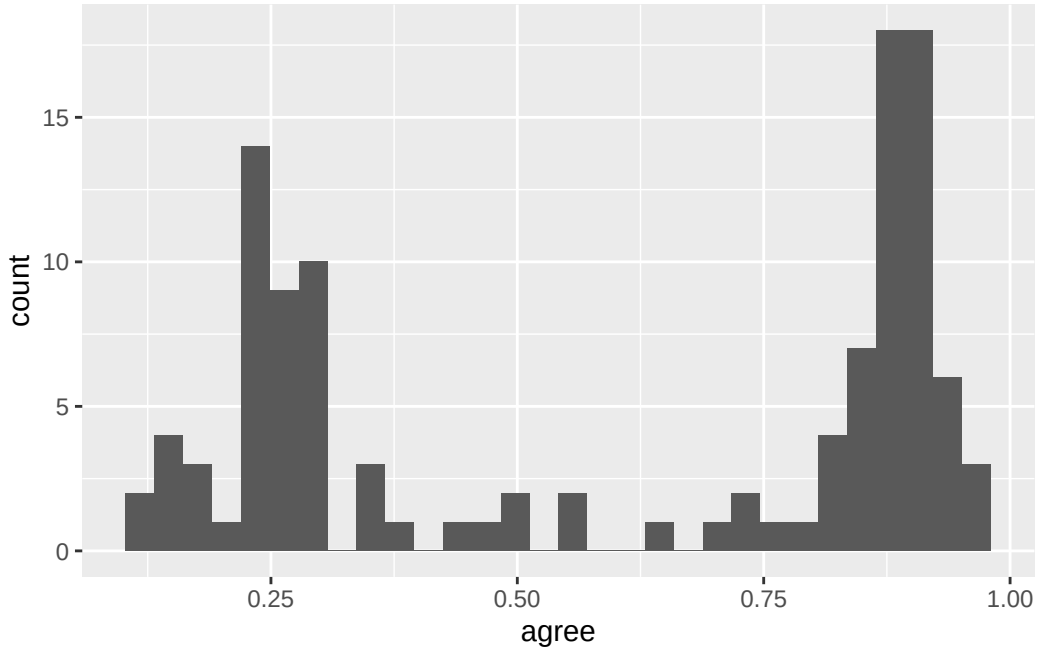
Draw a column plot with the agreement with Trump of Bernie Sanders and Ted Cruz. What happens if you use `last_name` as the `y` aesthetic mapping and `agree` in the `x` aesthetic mapping? Your code:

2.3.2 Univariate plots: numerical

We can draw a histogram with `geom_histogram()`:

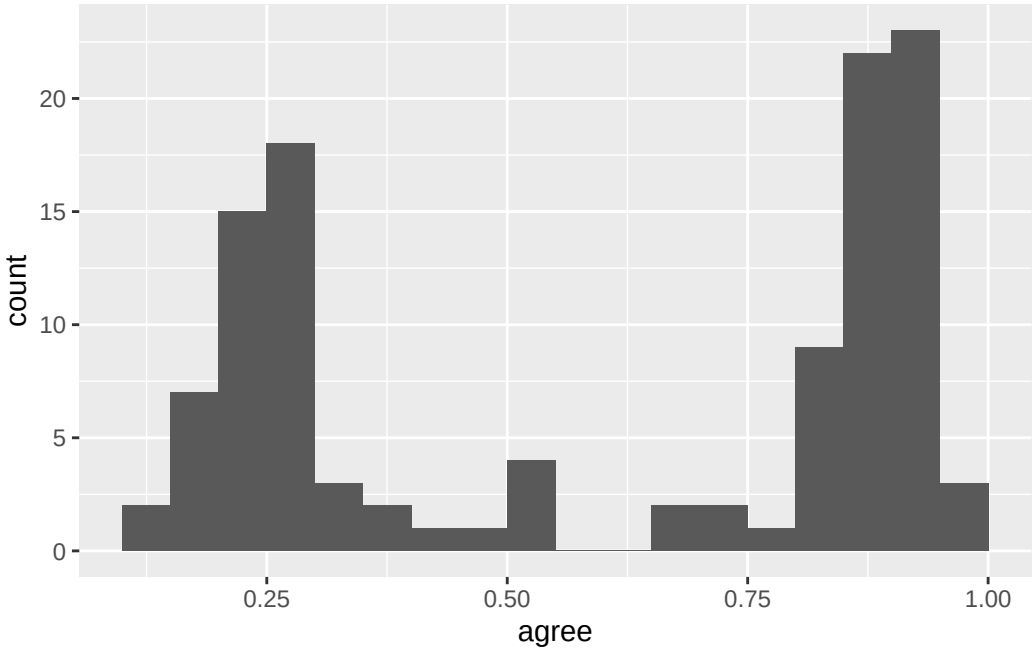
```
ggplot(trump_scores_ss, aes(x = agree)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



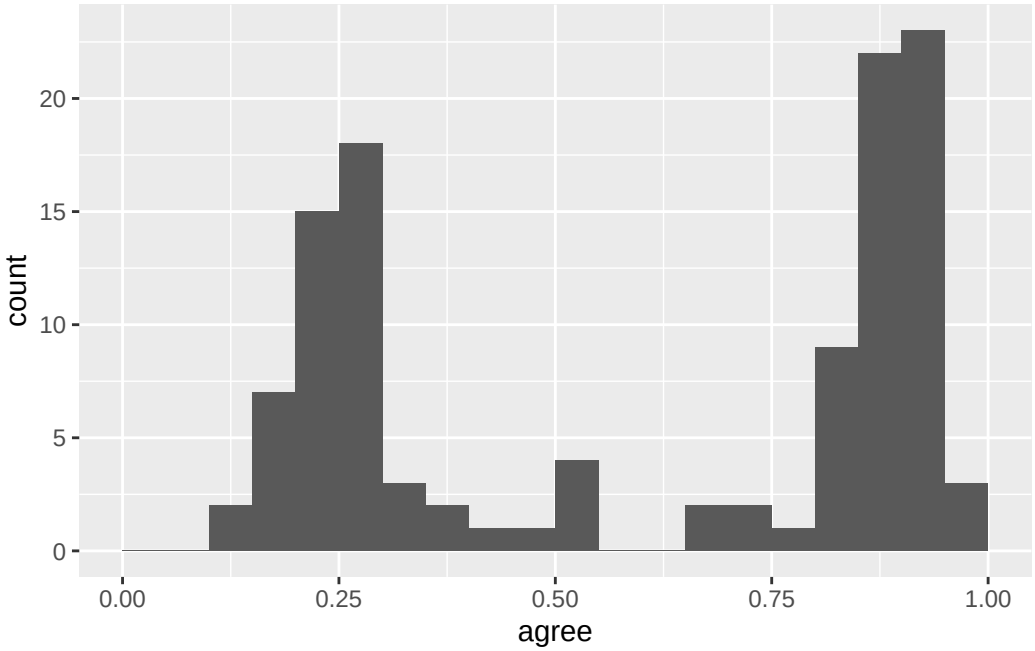
Notice the warning message above. It's telling us that, by default, `geom_histogram()` will draw 30 bins. Sometimes we want to modify this behavior. The following code has some common options for `geom_histogram()` and their explanations:

```
ggplot(trump_scores_ss, aes(x = agree)) +
  geom_histogram(binwidth = 0.05,   # draw bins every 0.05 jumps in x
                 boundary = 0,      # don't shift bins to integers
                 closed   = "left") # close bins on the left
```



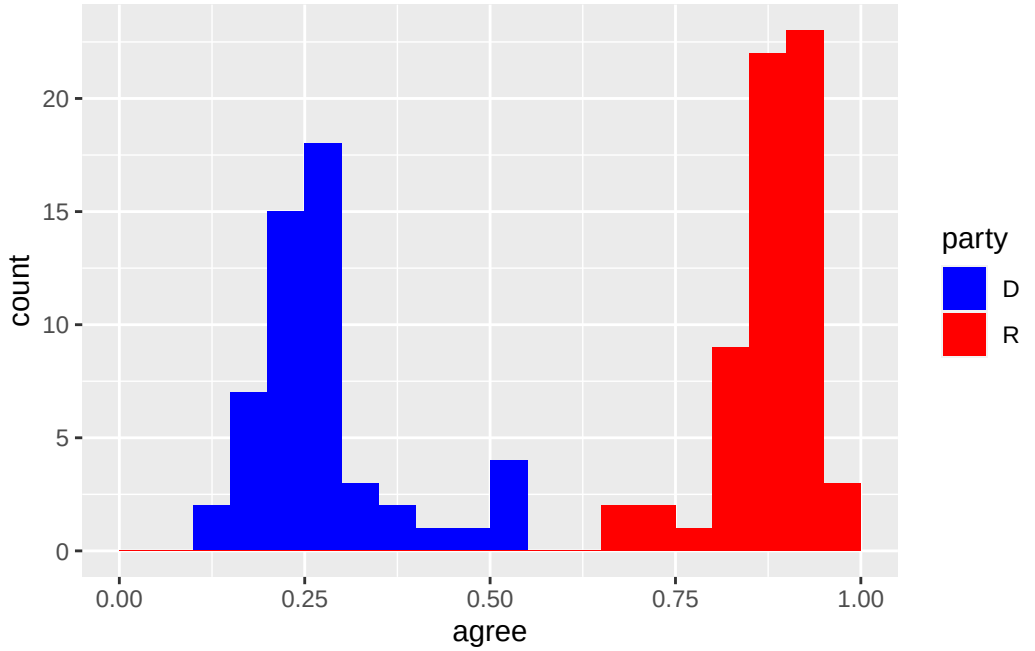
Sometimes we want to manually alter a scale. Here we use . For example, to make the x-axis go from 0 to 1:

```
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!

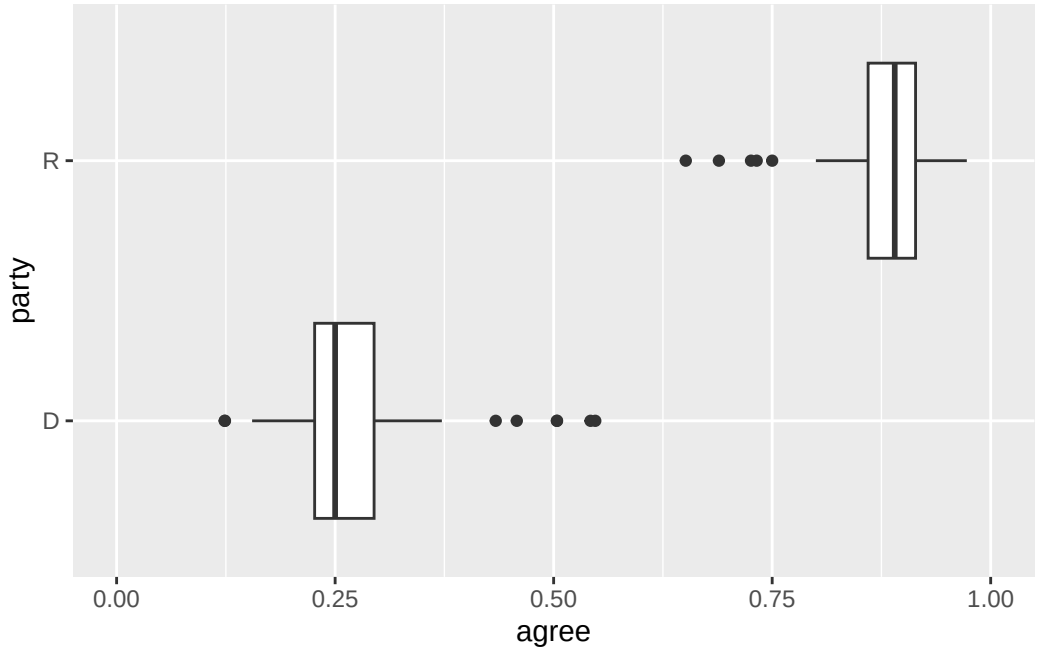
```
ggplot(trump_scores_ss, aes(x = agree, fill = party)) +
  geom_histogram(binwidth = 0.05, boundary = 0, closed   = "left") +
  scale_x_continuous(limits = c(0, 1)) +
  # change default colors:
  scale_fill_manual(values = c("D" = "blue", "R" = "red"))
```



2.3.3 Bivariate plots

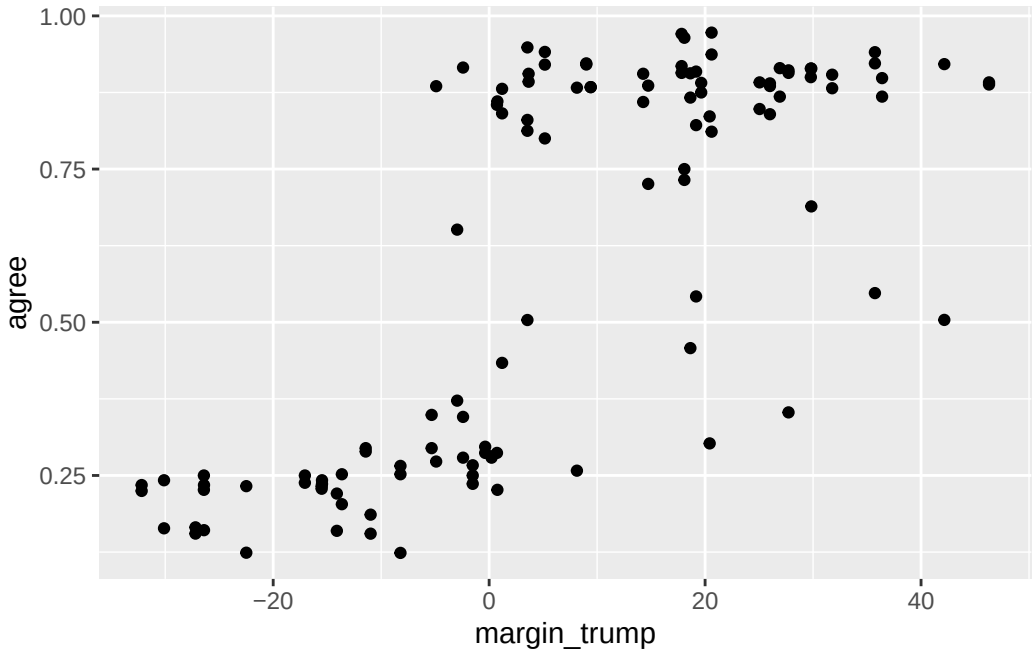
Another common bivariate plot for categorical and numerical variables is the grouped box plot:

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



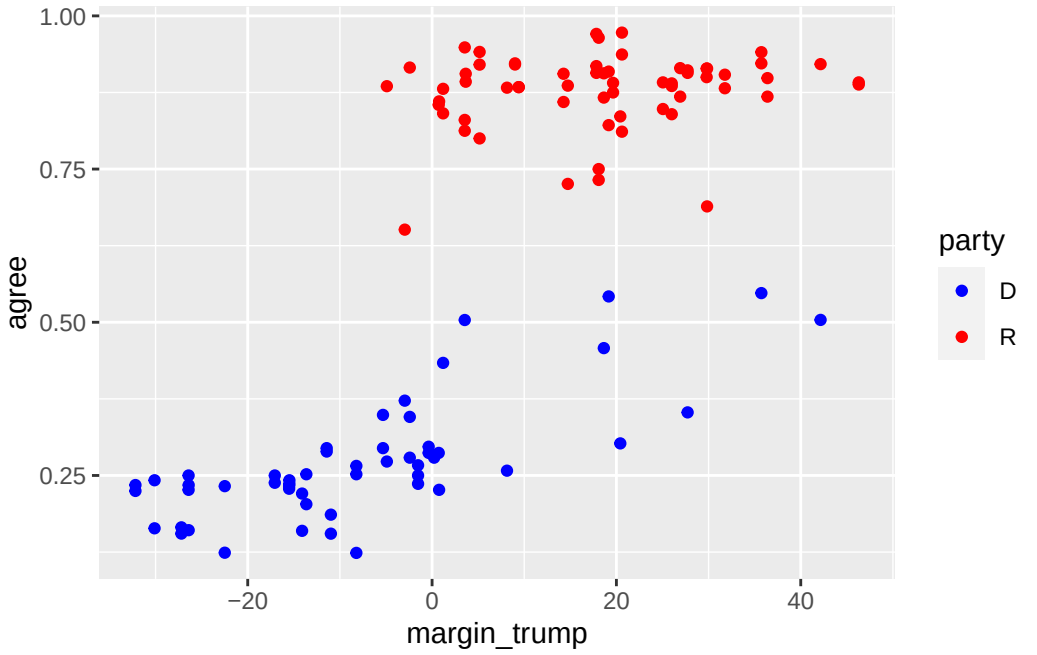
For bivariate plots of numerical variables, scatter plots are made with `geom_point()`:

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



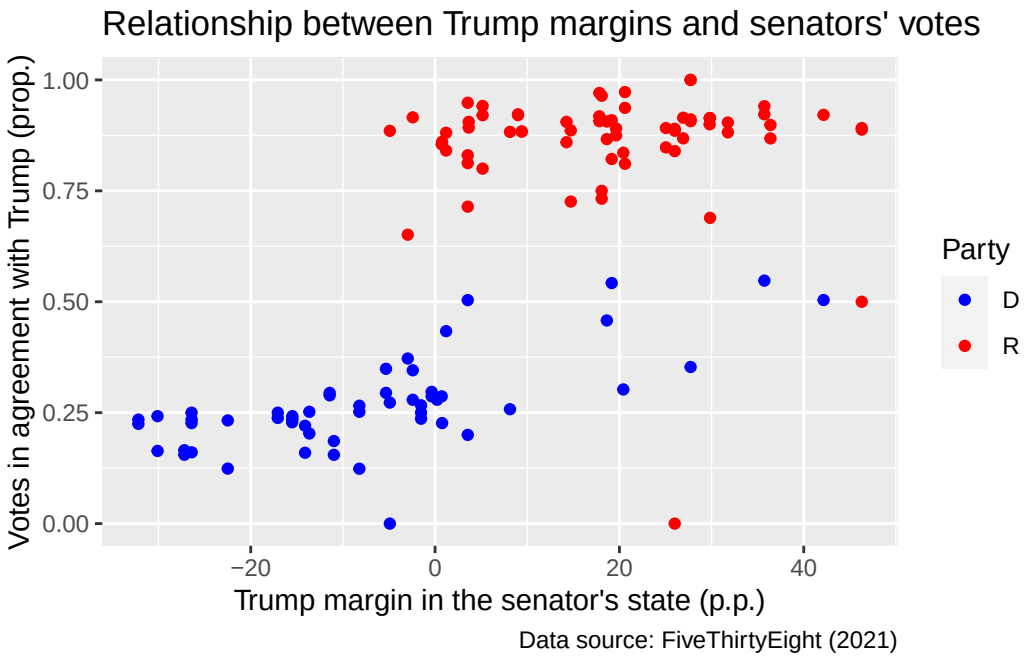
We can add the `color` aesthetic mapping to add a third variable:

```
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
Votes in agreement with Trump (prop.)
1.00
0.75
0.50
0.25
0.00
−20
0
20
40
Trump margin in the senator's state (p.p.)
Data source: FiveThirtyEight (2021)
Party
D
R

3 Matrices

3.1 Introduction

3.1.1 Scalars

- One number (for example, 12) is referred to as a scalar.
- Each scalar in a matrix is an *element* of that matrix.

$$\begin{bmatrix} 12 \end{bmatrix}$$

> **Note**
>
> This is also called a 1 x 1 ("one by one") matrix.

3.1.2 Vectors

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

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

- Since this is a column of numbers, we cleverly refer to it as a *column vector*.
- Here is another example of a vector:

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

- This, in contrast, is called a *row vector*.

3.1.2.1 Vector construction in RStudio

- Vectors are very easy to construct in RStudio.
- Here are two examples:

```
# Making a simple column vector
col_vector <- matrix(1:12)
col_vector
```

```
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
[11,]   11
[12,]   12
```

```
# Making a simple row vector
row_vector <- 15:23
row_vector
```

```
[1] 15 16 17 18 19 20 21 22 23
```

3.2 Operators

3.2.1 Summation

- The summation operator $\sum$ lets us perform an operation on a sequence of numbers, which is often but not always a vector.

$$x = \begin{bmatrix} 12 & 7 & -2 & 0 & 1 \end{bmatrix}$$

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

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

- Then, we do the following math:

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

3.2.2 Product

- The product operator $\prod$ can also perform operations over a sequence of elements in a vector.

$$z = \begin{bmatrix} 5 & -3 & 5 & 1 \end{bmatrix}$$

- We can then calculate the calculate the product of the four elements in the vector, which is expressed as follows:

$$\prod_{i=1}^{4} z_i$$

- Then, we do the following math:

$$5 \times -3 \times 5 \times 1 = -75$$

Matrices

3.2.2.1 Operators in RStudio

```
# Summation
mat <- c(5,-3,5,1)
sum(mat)
```

```
[1] 8
```

```
# Product
prod(mat)
```

```
[1] -75
```

3.2.3 Basics

- We can append vectors together to form a matrix:

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

- 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 $3x3$ matrix.
- We often use capital letters (sometimes **bold-faced**) to represent matrices.

3.2.3.1 Appending in RStudio

- We use `rbind()` to create a matrix, in which each vector will be a row.
- If we use `cbind()`, then each vector will be a column.

```
# Create some vectors
vec_1 <- 1:4
vec_2 <- 5:8
vec_3 <- 9:12
vec_4 <- 13:16

# rbind
rbind_mat <- rbind(vec_1, vec_2, vec_3, vec_4)
rbind_mat
```

```
      [,1] [,2] [,3] [,4]
vec_1    1    2    3    4
vec_2    5    6    7    8
vec_3    9   10   11   12
vec_4   13   14   15   16
```

```r
# cbind
cbind_mat <- cbind(vec_1, vec_2, vec_3, vec_4)
cbind_mat
```

```
     vec_1 vec_2 vec_3 vec_4
[1,]     1     5     9    13
[2,]     2     6    10    14
[3,]     3     7    11    15
[4,]     4     8    12    16
```

3.2.4 Structure

- How do we refer to specific elements of the matrix?
- Matrix A is an $m \times n$ matrix where $m = n = 3$
- 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.

> Reminder
>
> 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).

3.2.4.1 Constructing matrices in RStudio

- Matrices are very easy to construct in RStudio.
- The basic syntax is `matrix(data, nrow, ncol, byrow, dimnames)`.
  - `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.
  - `dimname` is the names assigned to the rows and columns.
- Here are some examples:

```r
# Elements are arranged sequentially by row.
M <- matrix(c(1:12), nrow = 4, byrow = TRUE)
print(M)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

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

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```r
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(1:12), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

print(P)
```

```
     col1 col2 col3
row1    1    2    3
row2    4    5    6
row3    7    8    9
row4   10   11   12
```

```r
# Access the element at 1st row and 3rd column.
print(P[1,3])
```

```
[1] 3
```

```
# Access the element at 4th row and 2nd column.
print(P[4,2])
```

```
[1] 11
```

3.3 Matrix operations

3.3.1 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:

$$A \pm B = C$$

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

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

3.3.1.1 Add and subtract in RStudio

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

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
```

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

```
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
```

```
# Add the matrices.
result1 <- matrix1 + matrix2
cat("Result of addition","\n")
```

```
Result of addition
```

```
print(result1)
```

```
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
```

```
# Subtract the matrices
result2 <- matrix1 - matrix2
cat("Result of subtraction","\n")
```

```
Result of subtraction
```

```
print(result2)
```

```
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

Practice

$$A = \begin{bmatrix} 1 & 4 & 2 \\ -2 & -1 & 0 \\ 0 & -1 & 3 \end{bmatrix}$$

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

Calculate $A + B$

Practice

$$A = \begin{bmatrix} 6 & -2 & 8 & 12 \\ 4 & 42 & 8 & -6 \\ -14 & 5 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 18 & 42 & 3 & 7 \\ 0 & -42 & 15 & 4 \\ -7 & 0 & 21 & -18 \end{bmatrix}$$

Calculate $A - B$

3.3.2 Scalar multiplication

- Scalar multiplication is very intuitive.
- As we know, a scalar is a single number, or a 1 x 1 matrix.
- We multiply each value in the matrix by the scalar to perform this operation.
- This is expressed as follows:

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

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

3.3.2.1 Scalar multiplication in RStudio

- All we need to do is take an established matrix and multiply it by some scalar.

```
matrix1
```

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
```

```
matrix1*3
```

```
     [,1] [,2] [,3]
[1,]    9   -3    6
[2,]   27   12   18
```

Practice

$$A = \begin{bmatrix} 1 & 4 & 2 \\ 8 & -1 & 3 \\ 0 & -2 & 3 \end{bmatrix}$$

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

Calculate $2 \times A$ and $-3 \times B$

3.3.3 Matrix multiplication

- 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?

$$B = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} \quad M = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 2 & 4 \\ 2 & 3 & 2 \end{bmatrix} \quad L = \begin{bmatrix} 6 & 5 & -1 \\ 1 & 4 & 3 \end{bmatrix}$$

> **Warning**
>
> When multiplying matrices, *order matters*.

- Why can't we multiply in the opposite order?

3.3.4 Multiplication steps

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

> **Note**
>
> 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).

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

3.3.4.1 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
Wood	100	150
Cloth	270	420
Feathers	130	195

- 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**
>
> 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

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

- 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 3 x 2 Cost matrix (C) times the 2 x 1 Quantity matrix (Q) yields the 3 x 1 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.

3.3.4.2 Matrix multiplication in RStudio

- We must make sure the matrices are conformable, as we do for our manual calculations.
- Then, we can multiply them together without issue.
- When multiplying together matrices in RStudio, remember to use the `%*%` operator, otherwise you will receive an error message.

> **Warning**
>
> If you have a missing value or `NA` in one of the matrices you are trying to multiply, you will have NAs in your resulting matrix.

```
# Creating matrices
m <- matrix(1:8, nrow=2) #
n <- matrix(8:15, nrow=4)
m
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

```
n
```

```
     [,1] [,2]
[1,]    8   12
[2,]    9   13
[3,]   10   14
[4,]   11   15
```

```
# Multiplying matrices using operator
m%*%n
```

```
     [,1] [,2]
[1,]  162  226
[2,]  200  280
```

3.4 Properties

- Addition and subtraction:
  - Associative: $(A \pm B) \pm C = A \pm (B \pm C)$
  - Communicative: $A \pm B = B \pm A$
- Multiplication:
  - $AB \neq BA$
  - $A(BC) = (AB)C$
  - $A(B + C) = AB + AC$
  - $(A + B)C = AC + BC$

3.5 Special matrices

Square matrix:

- The **diagonal** of a square 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. Only a square matrix has a diagonal.

- The **trace** of a matrix is simply the sum of the diagonal elements of the matrix. So, then, a matrix must be square to have a trace.

Diagonal matrix:

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

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.

Identity matrix:

- The **identity matrix** is a scalar matrix with all of the diagonal elements equal to one.
- All of the off-diagonal elements are equal to zero.
- The capital letter I is reserved for the identity matrix.

3.6 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").

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

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

3.6.0.1 Transpose in RStudio

- We use `t()` to get the transpose.

```
# Construct our matrix
J <- matrix(c(4,5,3,0,7,-2), nrow = 3, byrow = TRUE,)
J
```

```
     [,1] [,2]
[1,]    4    5
[2,]    3    0
[3,]    7   -2
```

```
t(J)
```

```
     [,1] [,2] [,3]
[1,]    4    3    7
[2,]    5    0   -2
```

3.7 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 RStudio for.

3.7.0.1 Examples

- We use the `solve()` function to calculate the inverse of a matrix.

```
# Create 3 different vectors
# using combine method.
a1 <- c(3, 2, 5)
a2 <- c(2, 3, 2)
a3 <- c(5, 2, 4)

# bind the three vectors into a matrix
# using rbind() which is basically
# row-wise binding.
A <- rbind(a1, a2, a3)

# print the original matrix
print(A)
```

```
   [,1] [,2] [,3]
a1    3    2    5
a2    2    3    2
a3    5    2    4
```

```
# Use the solve() function to calculate the inverse.
T1 <- solve(A)

# print the inverse of the matrix
print(T1)
```

```
             a1          a2         a3
[1,] -0.29629630 -0.07407407  0.4074074
[2,] -0.07407407  0.48148148 -0.1481481
[3,]  0.40740741 -0.14814815 -0.1851852
```

3.8 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.

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

3.9 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*.

3.9.1 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.

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

3.9.2 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:

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

- x_{ij} is the i-th observation of the j-th independent variable.

3.9.3 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:

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

3.9.4 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, that is our OLS estimators.
- Remember, the OLS method is to choose $\hat{\beta}$ such that the sum of squared residuals ("SSR") is minimized.

3.9.4.1 Example in RStudio

- 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[,1])
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.

```r
# create two separate matrices for IVs
X1 <- as.matrix(cars_df[,4])
X2 <- as.matrix(cars_df[,6])

# create constant column

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

```
      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
[10,]        1 123 3.440
[11,]        1 123 3.440
[12,]        1 180 4.070
[13,]        1 180 3.730
[14,]        1 180 3.780
[15,]        1 205 5.250
[16,]        1 215 5.424
[17,]        1 230 5.345
[18,]        1  66 2.200
[19,]        1  52 1.615
```

```
[20,]        1  65 1.835
[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
[29,]        1 264 3.170
[30,]        1 175 2.770
[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
```

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

4 Tidy data analysis II

5 Functions and loops

5.1 Basics

5.1.1 What is a function?

- Anything that takes input(s) and gives one defined output.
- They assign a unique value in its range (y values) for each value in its domain (x values).
- In math, this usually looks something like $f(x) = 3x + 4$.
  - x is the *argument* that the function takes.
  - For any x, multiply x by 3 and then add 4
  - Alternative but equivalent notation: $y = 3x + 4$
  - y is "a function of" x, so $y = f(x)$
- We describe functions with both equations and graphs.

5.1.2 Function machine

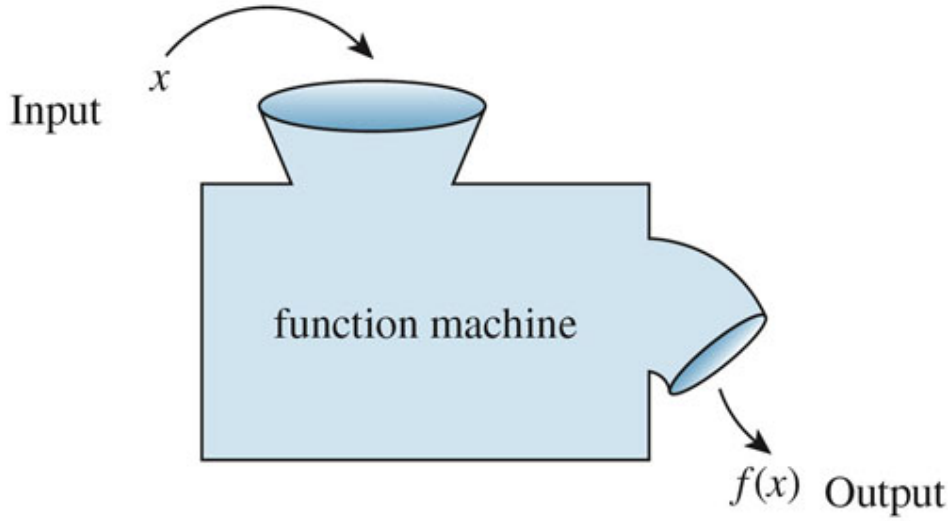
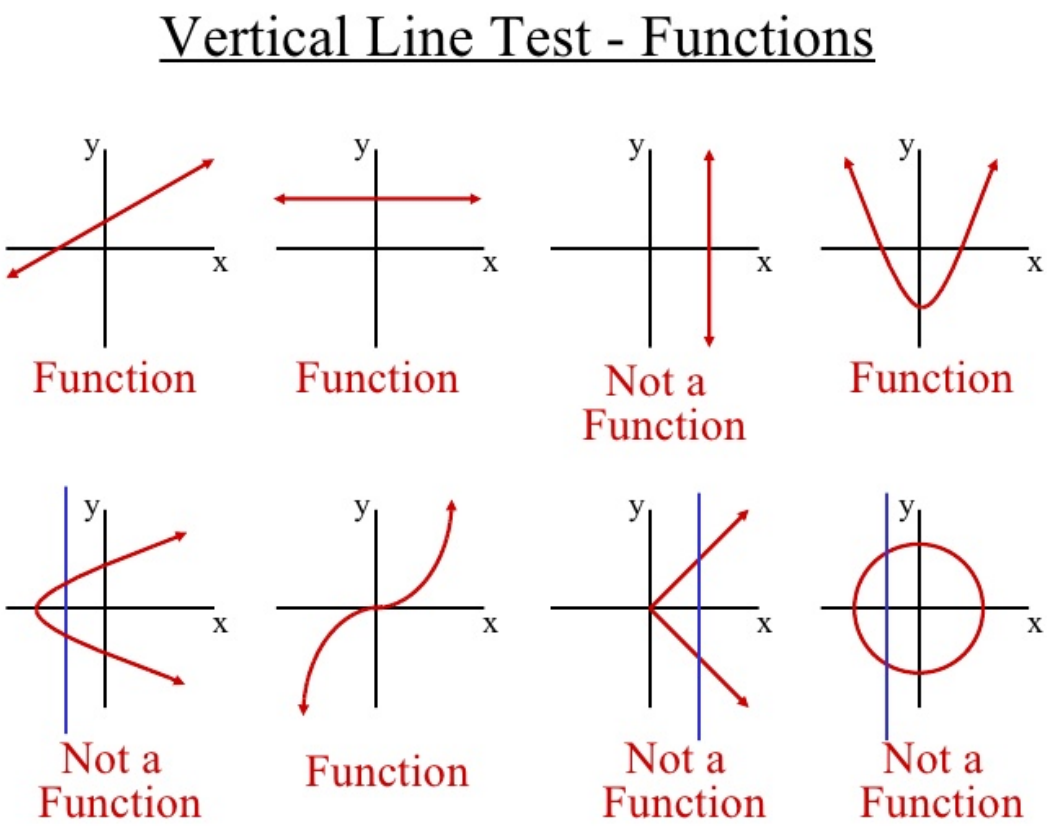


Figure 5.1: Function machine

5.1.3 Visualization

When graphed, we can't draw vertical line through a function. Why not?



5.2 Types of functions

5.2.1 Linear functions

- We can easily make a function that describes a line.

$$y = mx + b$$

- m is the slope (for every one unit increase in x, y increases m units).

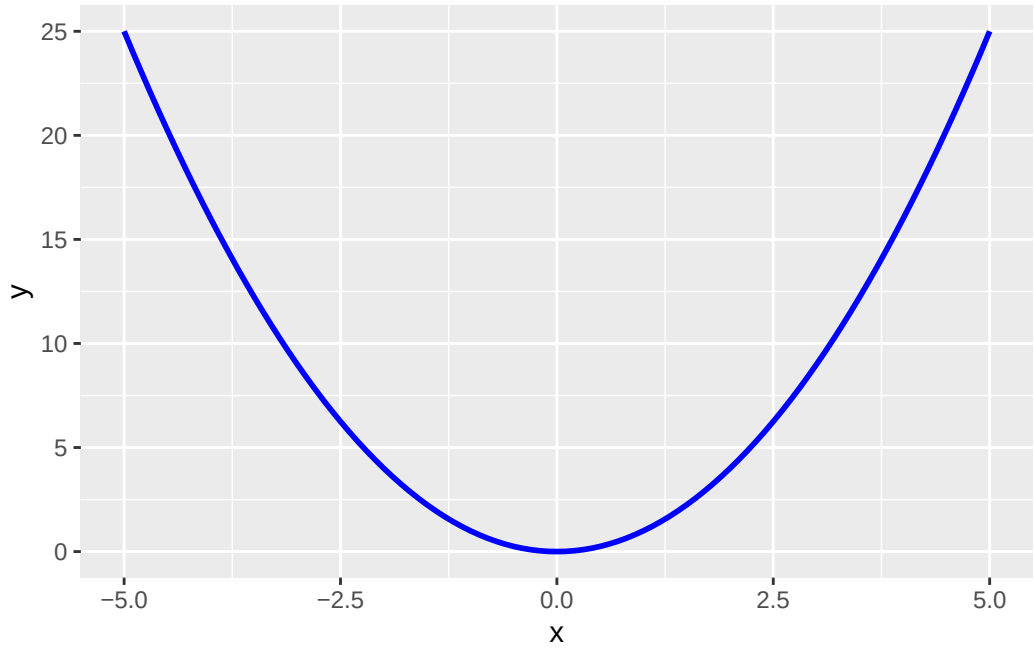
- b is the y-intercept: the value of y when $x = 0$.
- More generally, $y = a + bx$ - a is the intercept and b is the slope.

5.2.2 Quadratic

- These lines have one curve.

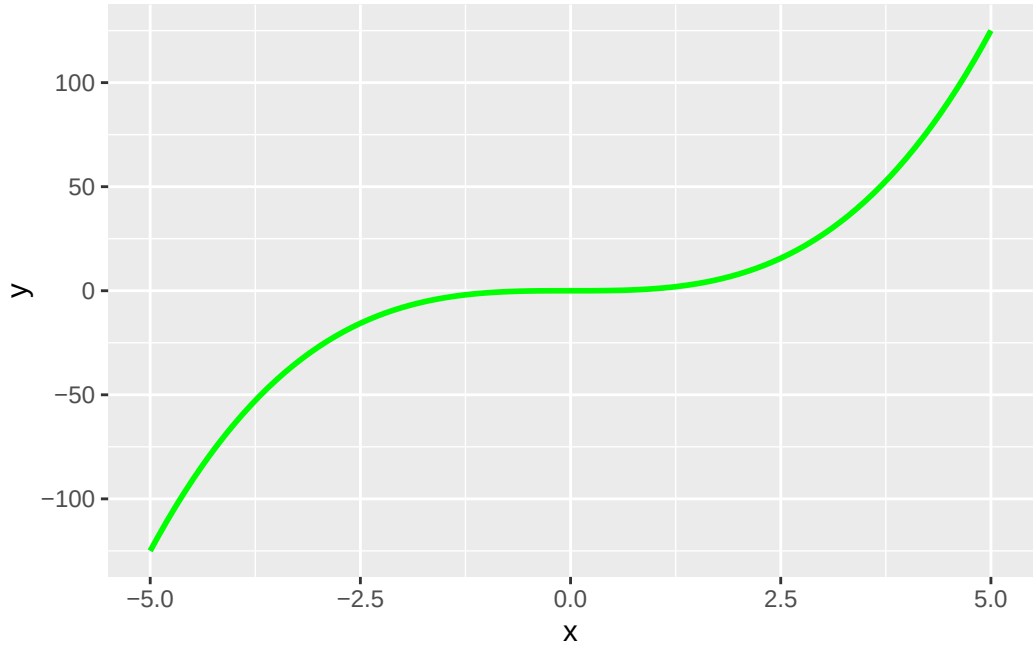
$$y = ax^2 + bx + c$$

- a, b, and c don't have well-defined meanings here.
- If a is negative, the function opens downward; if a is positive,it opens upward.
- Note that x^2 always returns positive values.



5.2.3 Cubic

- These lines (generally) have two curves (inflection points).
- $y = ax^3 + bx^2 + cx + d$
- a, b, c, and d don't have well-defined meanings here.



5.2.4 Polynomial

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

- These functions have (maximum) $n - 1$ changes in direction (turning points). - They also have (maximum) n x-intercepts. - They can be made arbitrarily precise.

5.2.5 Exponential

$$y = ab^x$$

or

$$f(x) = ab^x$$

- Here our independent variable, or input (x), is the exponent.

5.2.6 Trigonometric functions

- These functions include sine, cosine, and tangent.
- They are interesting (to some), but not usually useful for social science.

5.3 Logarithms and exponents

5.3.1 Logarithms

- Logarithms are basically the opposite (inverse) of exponents.
- They ask how many times you must raise the base to get x.
- $log_a(b) = x$ is asking "a raised to what power x gives b?
- $\log_3(81) = 4$ because $3^4 = 81$
- Logarithms can be undefined.
- The base cannot be 0, 1, or negative.

5.3.2 Relationships

If,

$$log_a x = b$$

then,

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

and

$$x = a^b$$

5.3.3 Basic rules

$$\frac{\log_x n}{\log_x m} = \log_m n$$

$$\log_x(ab) = \log_x a + \log_x b$$

$$\log_x \left(\frac{a}{b}\right) = \log_x a - \log_x b$$

$$\log_x a^b = b \log_x a$$

$$\log_x 1 = 0$$

$$log_x x = 1$$

$$m^{\log_m(a)} = a$$

5.3.4 Natural logarithms

- We most often use natural logarithms.
- This means $\log_e(x)$, often written $\ln(x)$.
- $e \approx 2.7183$.
- ln(x) and its exponent opposite, e^x, have nice properties when we hit calculus.

5.3.5 Definition 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.

Practice

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

$$log_{10}(1000)$$

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

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

$$ln(1)$$

$$ln(e^2)$$

$$ln(5e)$$

5.4 Functions of functions

5.4.1 Basics

- Functions can take other functions as arguments.
- This means that outside function takes output of inside function as its input.
- This is typically written as $f(g(x))$.
- 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.
- We write this $(x-3)^2$, NOT x^2-3.

5.4.2 PMF, PDF, and CDF

- PMF - probability mass function
  - This gives the probability that a discrete random variable is exactly equal to some value.
- PDF - probability density function
  - This gives the probability that a continuous random variable falls within a particular range of values.
- CDF - cumulative distribution function
  - This gives the probability that a random variable X takes a value less than or equal to x.

6 Calculus

- Calculus is about dealing with infinitesimal values.
- We are going to focus on two big ideas:
  - Derivatives
  - Integrals

6.1 Derviatives

- "Derivative" is just a fancy term for slope.
- Slope is the rate of change $\frac{\delta y}{\delta x}$ or $\frac{dy}{dx}$.
- Specifically, the derivative is the *instantaneous* rate of change.
- We need slope for our statistics, which are all about fitting lines.
- We also need slope for taking maxima and minima.
- The equation for a line is $y = mx + b$. What is its slope?

6.1.1 Calculating derivatives

- Slope is rise over run, which is $\dfrac{f(x + \Delta x) - f(x)}{\Delta x}$
- To see why, consider the slope of a line connecting two points:

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

- We can define $x_2 = x_1 + \Delta x$ (or equivalently $\Delta x = x_2 - x_1$)

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

- As we've seen, for a curve, we need to be infinitely close for our line's defining points, yielding

$$\lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- This gives us this instantaneous slope (rate of change) of a function at every point on its domain. The above equation is the definition of the derivative.

6.1.2 Notation

- $\frac{d}{dx}f(x)$ is read "The derivative of f of x with respect to x."
  - You can also say "The instantaneous rate of change in f of x with respect to x."
- Lagrange's prime notation: $f'(x)$ (read: "f prime x") is the derivative of $f(x)$.
- If $y = f(x)$, $\frac{dy}{dx}$ is "The derivative of y with respect to x".
  - The variable with respect to which we're differentiating is the one that appears in the denominator.

> **Warning**
>
> Do not try to cancel out the d's, no matter how tempting it is.

Examples

- What is $\frac{d(x^2)}{dx}$?
  - x^2
  - $2x^{2-1}$ -$2x$
- What is $\frac{d(4x^3)}{dx}$?
  - $4x^3$
  - $4 * 3x^{3-1}$
  - $12x^2$

Practice

- Take the derivative of each of the following:

$$x^3$$

$$3x^2$$

$$60x^{11}$$

$$x$$

$$\frac{4}{x^2}$$

$$9\sqrt{x}$$

$$6x^{5/2}$$

$$11,596,232$$

Practice

- Evaluate the derivatives at $x = 2$ and $x = -1$

$$x^3$$

$$3x^2$$

$$60x^{11}$$

$$x$$

$$\frac{4}{x^2}$$

$$9\sqrt{x}$$

$$6x^{5/2}$$

$$11,596,232$$

Practice

- Take the derivative of each of the following:

$$x^3$$

$$3x^2$$

$$60x^{11}$$

$$x$$

$$\frac{4}{x^2}$$

$$9\sqrt{x}$$

$$6x^{5/2}$$

$$11,596,232$$

- Evaluate the derivatives at $x = 2$ and $x = -1$

$$x^3$$

$$3x^2$$

$$60x^{11}$$

$$x$$

$$\frac{4}{x^2}$$

$$9\sqrt{x}$$

$$6x^{5/2}$$

$$11,596,232$$

6.1.3 Special functions

- A few functions have particular rules:

$$\frac{d(ln(x))}{dx} = \frac{1}{x}$$

$$\frac{d(log_b(x))}{dx} = \frac{1}{x * ln(b)}$$

$$\frac{d(e^x)}{dx} = e^x$$

$$\frac{d(a^x)}{dx} = a^x ln(a)$$

$$\frac{dy}{dx}c = 0$$

$$\frac{d(x^x)}{dx} = x^x(1 + ln(x))$$

6.1.4 Derivatives with addition and substraction

- This is perhaps the rasiest rule to remember:

$$\frac{d(f(x) \pm g(x))}{dx} = f'(x) \pm g'(x)$$

Practice

- Take the derivative of each of the following:

$$x^2 + x + 5$$

$$x^4 - 4x^3 + 5x^2 + 8x - 6$$

$$3x^5 - 6x^2$$

$$5x^2 + 8\sqrt{x} - \frac{1}{x}$$

$$ln(x) + 5e^x - 4x^3$$

6.2 Advanced rules

6.2.1 Product rule

- This rule is more complicated:

$$\frac{d(f(x) \times g(x))}{dx} = f'(x)g(x) + g'(x)f(x)$$

- Example:

$$2x \times 3x$$
$$f(x) = 2x$$
$$g(x) = 3x$$
$$f'(x) = 2$$
$$g'(x) = 3$$
$$f'(x)g(x) + g'(x)f(x) = 2 \times 3x + 3 \times 2x$$
$$\frac{d(2x \times 3x)}{dx} = 6x + 6x = 12x$$

Practice

- Take the derivative of each of these:

$$x^3 * x$$
$$e^x * x^2$$
$$ln(x) * x^{-3}$$

Reminder!

$$\frac{d(f(x) * g(x))}{dx} = f'(x)g(x) + g'(x)f(x)$$

6.2.2 Quotient rule

$$\frac{d\frac{f(x)}{g(x)}}{dx} = \frac{f'(x)g(x) - g'(x)f(x)}{[g(x)]^2}$$

- If you're having trouble with this, just apply the product rule to the following:

$$\frac{d[f(x) * g^{-1}(x)]}{dx}$$

Reminder!

$$\frac{d\frac{f(x)}{g(x)}}{dx} = \frac{f'(x)g(x) - g'(x)f(x)}{[g(x)]^2}$$

6.2.3 Chain rule

$$\frac{d[f(g(x))]}{dx} = f'(g(x)) * g'(x)$$

- Let's take the derivative of a function of a function:

$$\frac{d[ln(x^2)]}{dx}$$

$$f(x) = ln(x)$$

$$g(x) = x^2$$

$$f'(x) = \frac{1}{x}$$

$$g'(x) = 2x$$

$$\frac{1}{x^2} * 2x = \frac{2}{x}$$

Practice

- Take the derivative of each of the following:

$$(3x^4 - 8)^2$$

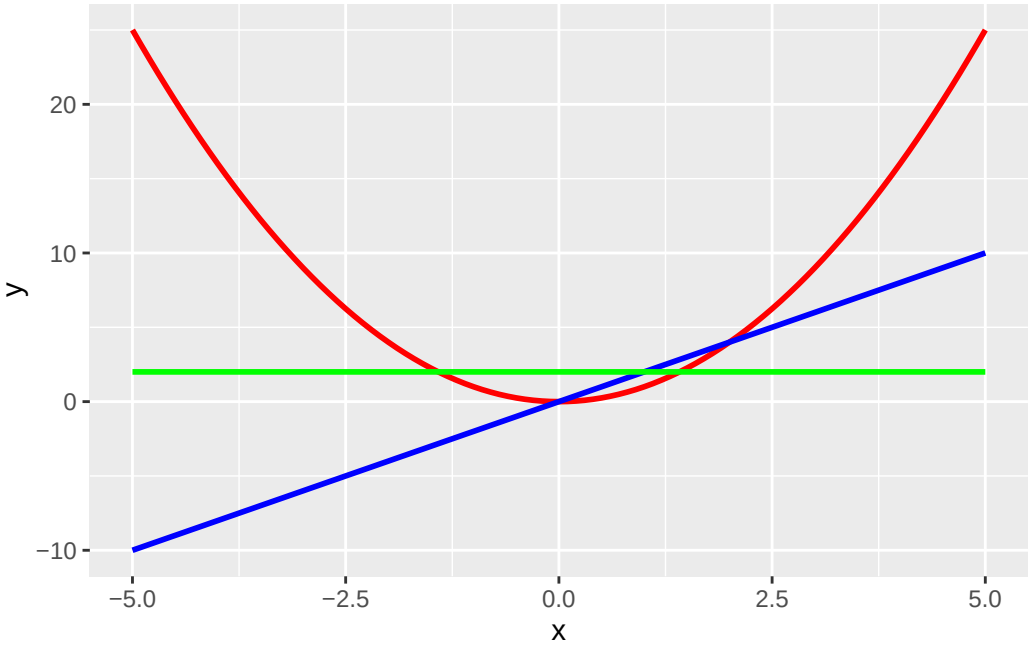
$$e^{x^2}$$

Reminder!

$$\frac{d(f(g(x))}{dx} = f'(g(x)) * g'(x)$$

6.2.4 Second derivative

- Same process as taking single derivative, except input for second derivative is output from first.
- 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.
- Example: driving a car.
  - $f(x)$ = distance traveled at time x
  - $f'(x)$ = speed at time x
  - $f''(x)$ = acceleration at time x
- Let's graph $f(x) = x^2$, $f'(x)$, and $f''(x)$.



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

- First, we take the first derivative:

$$f'(x^4) = 4x^3$$

- Then we use that output to take the second derivative:

$$f''(x^4) = f'(4x^3) = 12x^2$$

Practice

Take the second derivative of the following functions:

$$x^5$$

$$6x^2$$

$$4ln(x)$$

$$3x$$

$$4x^{3/2}$$

6.3 Differentiable and Continuous Functions

- 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:

$$\lim_{x \to a} f(x) = f(a)$$

- Continuity requires 3 conditions to hold:
  - $f(a)$ is defined (a is in the domain of f)
  - $\lim_{x \to a} f(x)$ exists
  - $\lim_{x \to a} f(x) = f(a)$ (the value of f equals the limit of f at a)
- Differentiable:
  - If $f'(x)$ exists, f is differentiable at x.
  - If f is differentiable at every point of an open interval I, f is differentiable on I.
  - Graph must have a (non-vertical) tangent line at each point, be relatively smooth, and not contain any breaks, bends, or cusps.
- 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.

6.3.1 When is f not differentiable?

When does $f'(x)$ not exist?

- When the function is discontinuous at that point.
  - Jump or break in the graph.
- There are different slopes approaching the point from the left and from the right.
  - Corner point
- When the graph of the function has a vertical tangent line at that point.
  - Cusp
  - Vertical inflection point

6.4 Extrema and optimization

Optimization lets us find the minimum or maximum value a function takes.

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

6.4.1 Extrema

- Informally, a maximum is just the highest value a function takes, and a minimum is the lowest value.
- Easy to identify extrema (maxima or minima) intuitively by looking at a graph of the function.
  - Maxima are high points ("peaks")
  - Minima are low points ("valleys")
- Extrema can be *local* or *global*.

6.4.2 Identifying extrema

- The derivative of a function gives the rate of change.
- When the derivative is zero (or fails to exist), 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.
- We'll start by identifying points where this is the case ("critical points" or "stationary points").

> **Note**
>
> A point where $f'(x) = 0$ or $f'(x)$ does not exist is called a *critical point* (or *stationary point*). 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"). 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).

$$f'(x) = 0$$

- This gives us the first-order condition (FOC).

6.4.3 Minimum or maximum?

BUT we don't know if we've found a maximum or minimum, or even if we've found an extremum or just an inflection point.

6.4.4 Second derivatives

- The second derivative gives us the rate of change of the rate of change of the original function. It tells us whether the slope is getting larger or smaller.

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

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 use the Second-Order Condition (SOC).

6.4.5 Local vs. Global Extrema

- To find the minimum/maximum on some interval, compare the local min/max to the value of the function at the interval's endpoints.
- To find the global minimum/maximum, check the function's limits as it approaches $+\infty$ and $-\infty$.
- Extreme value theorem: if a real-valued function f is continuous on the closed interval [a,b], then f must attain a (global) maximum and a (global) minimum.

6.5 Partial derivatives

- We can take the derivative with respect to different variables.
- For a function $fy = (x, z) = xz$, we might want to know how the function changes with x:

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

- We treat all other variables as constants and take derivative with respect to the variable of interest (here x).

How do we take a partial derivative?

Treat all other variables as *constants* and take *derivative* with respect to the variable of interest.

From our earlier example:

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

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

Why? Because the partial derivative of xz with respect to x treats z as a constant.

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

6.5.1 Application

- $\frac{\partial(x^2y+xy^2-x)}{\partial x}$
- We apply the addition rule to take the derivative of each term with respect to x.
- $\frac{\partial(x^2y)}{\partial x}+\frac{\partial(xy^2)}{\partial x}+\frac{\partial(-x)}{\partial x}$
- $2xy + y^2 - 1$

- $\frac{\partial(x^2y+xy^2-x)}{\partial y}$
- We apply the addition rule to take the derivative of each term with respect to y
- $\frac{\partial(x^2y)}{\partial y}+\frac{\partial(xy^2)}{\partial y}+\frac{\partial(-x)}{\partial y}$
- $x^2 + 2xy$

Practice

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

$$3xy - x$$
$$ln(xy)$$
$$x^3 + y^3 + x^4y^4$$
$$e^{xy}$$

6.6 Integrals

6.6.1 Area under a curve

Often we want to find the area under a curve. - Net effect of change - Cumulative density functions (CDFs) - Expected values and utilities

Sometimes this is easy. What's the area under the curve between $x = -1$ and $x = 1$ for this function?

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

"Hint

We can draw this and look at the graph. Remember:

$$area = \ell * w$$

Sometimes (usually) finding the area under a curve is harder. But this is basically the question behind integration.

6.6.2 Integrals as summation

You're familiar with summation notation.

$$\sum_{i=1}^{n} i$$

But this only works when we have discrete values to add. When we need to add continuously, we have to use something else. Specifically, integrals.

6.6.3 Definite integrals

Let's say we have a function

$$y = x^2$$

And we want to find the area under the curve from $x = 0$ to $x = 1$. To find the area we're interested in here, we can use the definite integral.

Generally speaking, the notation looks like this:

$$\int_{x=a}^{b} f(x), dx$$

Here a is the lower limit of integration, b is the upper limit of integration, our function $f(x)$ is our integrand, and x is our variable of integration.

For our question, we're looking for

$$\int_{x=0}^{1} f(x)dx$$

Which will give us a real number denoting the area under the curve of our function ($y = x^2$) between $x = 0$ and $x = 1$.

If f is continuous on $[a, b]$ or bounded on $[a, b]$ with a finite number of discontinuities, then f is integrable on $[a, b]$.

6.6.4 Indefinite integrals

The *indefinite* integral, or *anti-derivative*, $F(x)$ 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 $f(x)$.

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

This process is called anti-differentiation, or indefinite integration.

While the definite integral gives us a real number (the total area under a curve), the indefinite integral gives us a function.

We need the concept of indefinite integrals to help us solve definite integrals.

6.6.5 Solving definite integrals

$$\int_a^b f(x)\ dx = F(b) - F(a) = F(x)\Big|_a^b$$

6.6.6 Constants

> **Note**
>
> C in the following slides 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* $f(x)$.

6.6.7 Rules of integration

$$\int_a^a f(x)\ dx = 0$$

$$\int_a^b f(x)\ dx = -\int_b^a f(x)dx$$

$$\int a\ dx = ax + C \text{ where } a \text{ is a constant}$$

$$\int af(x)dx = a\int f(x)\ dx \text{ where } a \text{ is a constant}$$

6.6.8 More rules

$$\int (f(x) + g(x))\ dx = \int f(x)dx + \int g(x)dx$$

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

$$\int x^{-1} dx = \ln|x| + C$$

6.6.9 Solving the problem

Remember our function $y = x^2$ and our goal of finding the area under the curve from $x = 0$ to $x = 1$.

- Find the indefinite integral, $F(x)$
  - $\int x^2 \, dx$
  - $\frac{x^3}{3} + C$

- 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 $F(1) - F(0)$
  - $\frac{1}{3} - 0 = \frac{1}{3}$

Practice — indefinite integrals

$$\int x^2 \, dx$$

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

$$\int x \, dx$$

$$\int 3x^2 + 2x - 7 \, dx$$

$$\int \frac{2}{x} \, dx$$

Practice — definite integrals

$$\int_1^7 x^2 \, dx$$

$$\int_1^{10} 3x^2 \, dx$$

$$\int_7^7 x \, dx$$

$$\int_1^5 3x^2 + 2x - 7 \, dx$$

$$\int_1^e \frac{2}{x} \, dx$$

6.6.10 Integration by parts

- What if we want to integrate the product of two functions? How to evaluate $\int [f(x)g(x)]dx$?
- There's a formula for that: integration by parts.
- We can derive the formula from the product rule for derivatives, which you already know.

$$\frac{d(f(x)g(x))}{d(x)} = f^{'}(x)g(x) + g^{'}(x)f(x)$$

$$\int \frac{d(f(x)g(x))}{d(x)}dx = \int [f^{'}(x)g(x) + g^{'}(x)f(x)]dx$$

$$f(x)g(x) = \int f^{'}(x)g(x)dx + \int g^{'}(x)f(x)dx$$

- Note that we can't just plug in our two original functions into this formula. We have some work to do first.
- Board example:

$$\int x\sqrt{x}dx$$

7 Probability

7.1 What is probability?

- Frequency with which an event occurs.
  - Typically:

$$Pr(A) = P(A) = \pi(A) = \frac{\text{Number of ways an event can occur}}{\text{Total number of possible outcomes}}$$

- Probability predicts real-world events using theoretical quantities.
  - Formally, it assigns a likelihood of occurrence to each event in sample space
  - We use the probability space triplet (Ω, S, P), which are the sample space, event space, and probability mapping, respectively.
- We can consider probability as a function that maps $\Omega \to \mathbb{R}$.
- We can conceive it in terms of relative frequency or subjective belief.

7.2 Kolmogorov's axioms

- $Pr(S_i) \in \mathbb{R},\ 1 \geq Pr(S_i) \geq 0 \qquad \forall S_i \in S$
  - Where S is the event space, S_i are events.
  - Probabilities must be non-negative.
- $Pr(\Omega) = 1$
  - Where Ω is the sample space.
  - Something has to happen.
  - Probabilities sum/integrate to 1.
- $Pr\left(\bigcup_{i=1}^{\infty} S_i\right) = \sum_{i=1}^{\infty} Pr(S_i) \iff Pr(S_i \cap S_j) = 0 \ \ \forall i \neq j$
  - The probability of disjoint (mutually exclusive) sets is equal to the sum of their individual probabilities.

7.3 Some definitions

- **Random variable**: a variable whose value is determined by the outcome of a random process.
  - Sometimes also called a *stochastic variable*.
  - May be discrete or continuous.
- **Distribution** (of a random variable): the set of values the variable might take.
  - Probability mass function / probability density function defines the probability with which each value occurs.
  - Always sums / integrates to 1.
- **Realization** (of a random variable): a particular value taken by the variable.

- **Population**: the entire set of objects (people, cases, etc.) in which we are interested.
  - Often denoted N.
- **Sample**: a subset of the population we can observe, from which we try to make generalizations about the population.
  - Often denoted n.
- **Frequency distribution**: a count of how often a variable takes each of its possible values.
  - The number of members of a sample that take each value of a variable.
- **Independent random variables**: two variables are statistically independent if the value of one does not affect the value of the other.
  - Formally, $Pr(A \cap B) = Pr(A)Pr(B)$

7.4 Discrete probability

- 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.

7.4.1 Probability Mass Function (PMF)

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

- $0 \leq f(x_i) \leq 1$
  - 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 Discrete distribution

- 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, $Bernoulli(\pi)$.
  - π represents the probability of success.

- Drawing a specific card from a deck:

$$Pr(y = \text{ace of spades}) = \frac{1}{52}$$

- Drawing any card with a specific value from a deck:

$$Pr(y = ace) = \frac{4}{52}$$

- Getting a specific value on two dice rolls:

$$Pr(y = 8) = \frac{5}{36}$$

- We can express the probability mass function in tabular format or in a graph.

7.5 Continuous probability

- 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.
- Even if all events are equally likely, $\frac{1}{\infty} = 0$

7.5.1 Basics

- The domain may not span -∞ to ∞.
  - Even space between 0 and 1 is infinite.
- The domain is defined as the area under the probability density function.
- Two common examples are the uniform and bell curves.

7.5.2 Probability Density Function (PDF)

- Similar to PMF from before, but for continuous variables.
- Gives the probability a value falls within a particular interval
  - $P[a \leq X \leq b] = \int_a^b f(x)\,dx$
  - Total area under the curve is 1.
  - $P(a < X < b)$ is the area under the curve between a and b (where $b > a$).

7.6 Cumulative Density Function (CDF)

7.6.1 Discrete

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

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

- For discrete CDFs, that means summing up over all values.
- What is the probability of rolling a 6 or lower with two dice? $F(6) = ?$

7.6.2 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.

7.7 Statistics

7.7.1 Introduction

- While probability allows us to make predictions about events using distributions, statistics uses events to make estimates about distributions and variables.
- It is the process of learning from data.
- A statistic is a summary of data, capturing some theoretically-relevant quantity.
- Broad categories of numerical and categorical.

7.7.2 Univariate statistics

- These measure a single variable.
- Readily expressed in graphical form.
- Common examples:
  - Central tendency (mean, median, and mode)
  - Variance

7.7.3 Examples of univariate statistics

- The mean ($\bar{x}$) is calculated by summing the data, then dividing by the number of observations:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- The median is found by ordering the observations from highest to lowest and finding the one in the middle.
- The mode is the most common number.

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

- What are the mean, median, and mode of x?

7.7.4 Measures of central tendency

- Mean balances values on either side.
- Median balances observations on either side.
- Mode finds the most typical observation.
- Which is the best? Like most of what you'll learn in statistics, it depends.

7.7.5 Deviations from central tendency

- Consider two data sets:

$$x = \begin{bmatrix} 1 & 1.5 & 2 & 2.5 & 5.5 & 8.5 & 9 & 9.5 & 10 \end{bmatrix}$$

$$y = \begin{bmatrix} 4.5 & 4.8 & 5 & 5.3 & 5.5 & 5.7 & 6 & 6.2 & 6.5 \end{bmatrix}$$

- What is the mean of each?
- What is the median of each?
- Are they similar distributions?

7.7.6 Variance

- We use variance to measure the spread of a single variable.
- Formally defined as the squared deviation from the mean (μ).
- For discrete random variables, it is written $Var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$
- For continuous random variables, it is written $Var(x) = \sigma^2 = \int (x-\mu)^2 f(x)\ dx$

7.7.7 Standard deviation

- Sometimes variance doesn't make sense, either mathematically or conceptually.
  - Not always clear how to interpret "squared deviation from the mean."
- Instead, will frequently see standard deviation, which is square root of variance.
- It is written σ.

7.8 Bivariate statistics

7.8.1 Covariance

- While measures of central tendency and variance/standard deviation provide useful summaries of a single variable, they don't provide insights into relationships between variables.
- For that, we need bivariate statistics.
- Most common and straightforward is covariance.

- Colloquially, can think of covariance as measure of linear deviation from mean.
- When values from one variable are above their mean, are values from the other above or below their mean?
- Put another way, if I told you the value of x was high, would you expect values of y to be high or low?
- Formally, it is written as:

$$cov(X, Y) = E(X - E(X))(Y - E(Y)) = E(XY) - E(X)E(Y)$$

- It is important to note that the magnitude is meaningless; only the direction is interpretable.

7.8.2 Correlation

- Correlation is a normalized measure of covariance
- It is calculated as:

$$\rho_{X,Y} = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

- It varies between -1 and 1.
- What is correlation of two independent variables?

7.9 Regression

7.9.1 Ordinary least squares

- Ordinary least squares regression (OLS) is probably the most widely-used model in political science.
- It is all about drawing a line through data.

- This allows us to evaluate the relationship (the association) between x on y.
- The dependent variable, y, must be continuous, generally speaking.
- The main question is which line to draw.

Line and equation ($\widehat{y_i} = \widehat{\beta}_0 + \widehat{\beta}_1 x_i$) on board

7.9.2 Residuals

- In basically any set of data, no line can pass through every point (observation).
- We will always have make some error in predicting values.
- The error between the line and some point is referred to as the residual.
- If we refer to our predicted value as $\widehat{y}$, then we can calculate the residual for each observation with the following equation:

$$e_i = y_i - \widehat{y}_i$$

7.9.3 Finding the right line

- OLS determines the "best" line by minimizing the sum of squared residuals.
- Plug in all the values for the slope amd intercept and calculate the sum of squared residuals for these infinity combinations.
- That is a lot of work.
- The best solution turns out to be calculus.
- We want to minimize the sum of squared residuals with respect to our β's.

8 Simulations

9 Text analysis

10 Wrap up

10.1 Methods at UT

10.1.1 Required methods courses

- Scope and Methods of Political Science
  - Statistics I (Statistics/linear regression)
- Statistics II (Linear regression and more)
- Statistics III (Maximum likelihood estimation)
  - Only required if your major field is methods

10.1.2 Other methods courses

- **Statistics/econometrics:**
  - Bayesian Statistics
  - Causal Inference
  - Math Methods for Political Analysis
  - Time Series and Panel Data
  - Panel and Multilevel Analysis

10.1.3 More courses

- **Formal Theory**
  - Intro to Formal Political Analysis
  - Formal Political Analysis II
  - Formal Theories of International Relations
- **Everything else**
  - Conceptualization and Measurement
  - Experimental Methods in Political Science
  - Qualitative Methods

  - Network Analysis
  - Seminar in Field Experiments

10.1.4 Other departments at UT

You can also take courses through the Economics, Mathematics, or Statistics (Statistics and Data Science) departments.

- M.S. in Statistics

Software and Topic Short Courses - R, Python, Stata, etc.

- More info here.

10.1.5 Other resources

Summer programs at UT:

- Short courses in statistics
  - Department sometimes offers scholarships to cover part of the cost.

Summer programs outside UT:

- ICPSR (Inter-university Consortium for Political and Social Research)
  - Ann Arbor, Michigan
- EITM (Empirical Implications of Theoretical Models)
  - Houston and other locations (Michigan, Duke, Berkeley, Emory)
- IQMR (Institute for Qualitative and Multi-Method Research)
  - Syracuse, NY

References