Data Journalism with R and the Tidyverse

Code, data and visuals for storytellers

Matt Waite & Sarah Cohen (original authors); updated by Sean Mussenden, Rob We

8/29/2022

Table of contents

1 Introduction

Welcome to data journalism. The main goal of this course is to expand your ability to report and tell stories using data. You will use these tools to discover trends in data, like what Rachell Sanchez-Smith found with the sharp jump in COVID-19 cases in Arkansas children. You will learn how to create and publish graphics and maps. It's hard work but it is a lot of fun and very rewarding.

We have some basic goals for you to reach in this class. By the end of the semester, we want you to have basic proficiency and independence with data analysis. We want you to be able to write about data clearly, using the Associated Press style as a benchmark. We want you to be able to find and download a dataset, clean it up, visualize it.

The skills you will learn in the coming weeks are in high demand in journalism and beyond. Examine this BuzzFeed job description from 2017:

"We're looking for someone with a passion for news and a commitment to using data to find amazing, important stories — both quick hits and deeper analyses that drive conversations," the posting seeking a data journalist says. It goes on to list five things BuzzFeed is looking for: Excellent collaborator, clear writer, deep statistical understanding, knowledge of obtaining and restructuring data.

"You should have a strong command of at least one toolset that (a) allows for filtering, joining, pivoting, and aggregating tabular data, and (b) enables reproducible workflows."

You will learn these skills in this book. You'll get a taste of modern data journalism through Google Sheets and programming in R, a statistics language. You'll be challenged to think programmatically while thinking about a story you can tell to readers in a way that they'll want to read. Combining them together has the power to change policy, expose injustice and deeply inform.

1.1 Installations

This book begins with a basic review of Google Sheets and then shifts to the R statistical language. To follow along, you'll do the following:

1. Install the R language on your computer. Go to the this website, click download R based on your operating system. If that link somehow doesn't work, check R Project website and find a different location.
2. Install R Studio Desktop. The free version is great.

Going forward, you'll see passages like this:

```
install.packages("tidyverse")
```

That is code that you'll need to run common software packages in your R Studio.

1.2 About this book

This book is the collection of class materials compiled by various data journalism professors around the country: Matt Waite at the University of Nebraska-Lincoln's College of Journalism and Mass Communications and Sarah Cohen of Arizona State University. This version was edited by Derek Willis, Sean Mussenden and Rob Wells at the University of Maryland Philip Merrill College of Journalism.

There's some things you should know about it:

- It is free for students.
- The topics will remain the same but the text is going to be constantly tinkered with.
- What is the work of the authors is copyright Matt Waite 2020, Sarah Cohen 2022 and Derek Willis, Sean Mussenden and Rob Wells 2022.
- The text is Attribution-NonCommercial-ShareAlike 4.0 International Creative Commons licensed. That means you can share it and change it, but only if you share your changes with the same license and it cannot be used for commercial purposes. I'm not making money on this so you can't either.
- As such, the whole book – authored in Quarto – in its original form is open sourced on Github. Pull requests welcomed!

1.3 What we'll cover

- Spreadsheets
- R Basics
- Replication, Data Diary
- Data basics and structures
- Aggregates
- Mutating

- Working with dates
- Filters
- Data cleaning techniques, Janitor
- Pulling Data from PDFs
- Joins
- Basic data scraping
- Getting data from APIs: Census
- Visualizing for reporting: Basics
- Visualizing for reporting: Publishing
- Geographic data basics
- Geographic queries
- Geographic visualization
- Text analysis basics
- Writing with and about data
- Data journalism ethics

2 Learn a new way to read

Getting started in data journalism often feels as if you've left the newsroom and entered the land of statistics, computer programming and data science. This chapter will help you start seeing data reporting in a new way, by learning how to study great works of the craft as a writer rather than a reader.

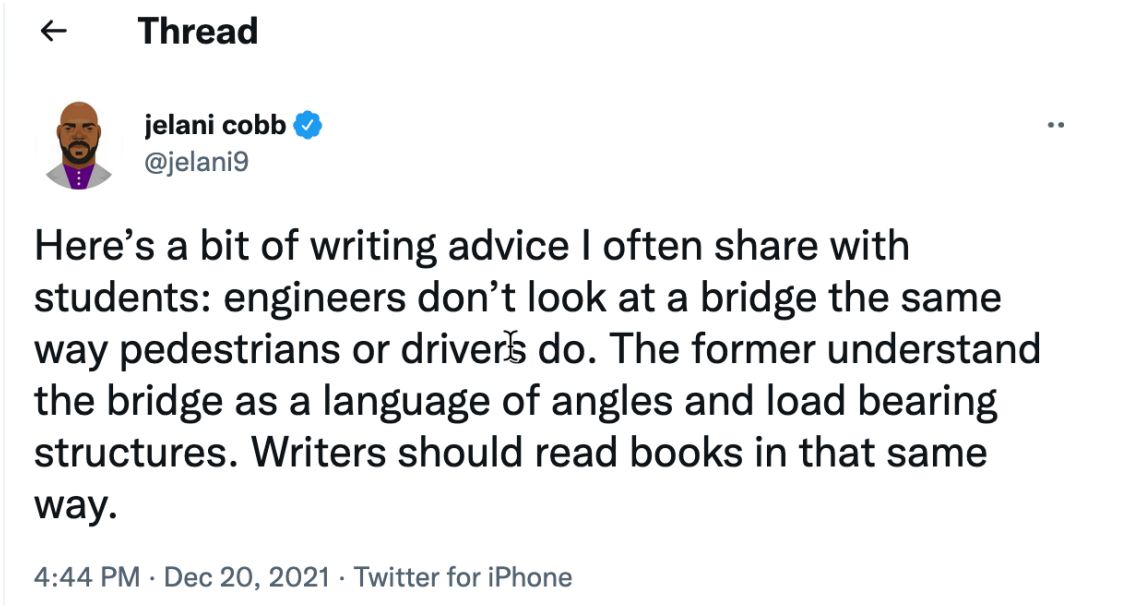


Figure 2.1: jelani cobb

Jelani Cobb tweeted, "an engineer doesn't look at a bridge the same way pedestrians or drivers do." They see it as a "language of angles and load bearing structures." We just see a bridge. While he was referring to long-form writing, reporting with data can also be learned by example – if you spend enough time with the examples.

Almost all good writers and reporters try to learn from exemplary work. I know more than one reporter who studies prize-winning journalism to hone their craft. This site will have plenty of examples, but you should stay on the lookout for others.

2.1 Read like a reporter

Try to approach data or empirical reporting as a reporter first, and a consumer second. The goal is to triangulate how the story was discovered, reported and constructed. You'll want to think about why *this* story, told this way, at this time, was considered newsworthy enough to publish when another approach on the same topic might not have been.

What were the questions?

In data journalism, we often start with a tip, or a hypothesis. Sometimes it's a simple question. Walt Bogdanich of The New York Times is renowned for seeing stories around every corner. Bogdanich has said that the prize-winning story "A Disability Epidemic Among a Railroad's Retirees" came from a simple question he had when railway workers went on strike over pension benefits – how much were they worth? The story led to an FBI investigation and arrests, along with pension reform at the largest commuter rail in the country.

The hypothesis for some stories might be more directed. In 2021, the Howard Center for Investigative Journalism at ASU published "Little victims everywhere", a set of stories on the lack of justice for survivors of child sexual assault on Native American reservations. That story came after previous reporters for the center analyzed data from the Justice Department showing that the FBI dropped most of the cases it investigated, and the Justice Department then only prosecuted about half of the matters referred to it by investigators. The hypothesis was that they were rarely pursued because federal prosecutors – usually focused on immigration, white collar crime and drugs – weren't as prepared to pursue violent crime in Indian Country.

When studying a data-driven investigation, try to imagine what the reporters were trying to prove or disprove, and what they used to do it. In journalism, we rely on a mixture of quantitative and qualitative methods. It's not enough to prove the "numbers" or have the statistical evidence. That is just the beginning of the story. We are supposed to ground-truth them with the stories of actual people and places.

Go beyond the numbers

It's easy to focus on the numbers or statistics that make up the key findings, or the reason for the story. Some reporters make the mistake of thinking all of the numbers came from the same place – a rarity in most long-form investigations. Instead, the sources have been woven together and are a mix of original research and research done by others. Try to pay attention to any sourcing done in the piece. Sometimes, it will tell you that the analysis was original. Other times it's more subtle.

But don't just look at the statistics being reported in the story. In many (most?) investigations, some of the key people, places or time elements come directly from a database.

Sarah Cohen at Arizona State University analyzed the Paycheck Protection Program loan data for ProPublica and found a handful of sketchy-looking records from a single county in coastal New Jersey. It turned out to be a pretty good story.

Often, the place that a reporter visits is determined by examples found in data. In this story on rural development funds, all of the examples came from an analysis of the database. Once the data gave us a good lead, the reporters examined press releases and other easy-to-get sources before calling and visiting the recipients or towns.

2.2 Reading tips

You'll get better at reading investigations and data-driven work over time, but for now, remember to go beyond the obvious:

- Where might the reporters have found their key examples, and what made them good characters or illustrations of the larger issue? Could they have come from the data?
- What do you think came first – a narrative single example that was broadened by data (naively, qualitative method), or a big idea that was illustrated with characters (quantitative method)?
- What records were used? Were they public records, leaks, or proprietary data?
- What methods did they use? Did they do their own testing, use statistical analysis, or geographic methods? You won't always know, but look for a methodology section or a description alongside each story.
- How might you localize or adapt these methods to find your own stories?
- Pick out the key findings (usually in the nut graf or in a series of bullets after the opening chapter): are they controversial? How might they have been derived? What might have been the investigative hypothesis? Have they given critics their due and tried to falsify their own work?
- How effective is the writing and presentation of the story? What makes it compelling journalism rather than a dry study? How might you have done it differently? Is a video story better told in text, or would a text story have made a good documentary? Are the visual elements well integrated? Does the writing draw you in and keep you reading? Think about structure, story length, entry points and graphics all working together.
- Are you convinced? Are there holes or questions that didn't get addressed?

2.3 Analyze data for story, not study

As journalists we'll often be using data, social science methods and even interviewing differently than true experts. We're seeking stories, not studies. Recognizing news in data is one of the hardest skills for less experienced reporters new to data journalism. This list of potential newsworthy data points is adapted from Paul Bradshaw's "Data Journalism Heist".

LAST UPDATED ON 2015-06-10

- Compare the claims of powerful people and institutions against facts – the classic investigative approach.
- Report on *unexpected* highs and lows (of change, or of some other characteristic)

- Look for outliers – individual values that buck a trend seen in the rest
- Verify or bust some myths
- Find signs of distress, happiness or dishonesty or any other emotion.
- Uncover *new* or *under-reported* long-term trends.
- Find data suggesting your area is *the same* or *different* than most others of its kind.

Bradshaw also did a recent study of data journalism pieces: "Here are the angles journalists use most often to tell the stories in data", in Online Journalism Blog. I'm not sure I agree, only because he's looking mainly at visualizations rather than stories, but they're worth considering.

3 Newsroom math

Jo Craven McGinty, then of The New York Times, used simple rates and ratios to discover that a 6-story brick New Jersey hospital was the most expensive in the nation. In 2012, Bayonne Medical Center "charged the highest amounts in the country for nearly one-quarter of the most common hospital treatments," the Times story said.

To do this story, McGinty only needed to know the number of the procedures reported to the government and the total amount each hospital charged. Dividing those to find an average price, then ranking the most common procedures, led to this surprising result.

3.1 Why numbers?

Using averages, percentages and percent change is the bread and butter of data journalism, leading to stories ranging from home price comparisons to school reports and crime trends. It may have been charming at one time for reporters to announce that they didn't "do" math, but no longer. Instead, it is now an announcement that the reporter can only do some of the job. You will never be able to tackle complicated, in-depth stories without reviewing basic math.

The good news is that most of the math and statistics you need in a newsroom isn't nearly as difficult as high school algebra. You learned it somewhere around the 4th grade. You then had a decade to forget it before deciding you didn't like math. But mastering this most basic arithmetic again is a requirement in the modern age.

In working with typical newsroom math, you will need to learn how to:

- Overcome your fear of numbers
- Integrate numbers into your reporting
- Routinely compute averages, differences and rates
- Simplify and select the right numbers for your story

While this chapter covers general tips, you can find specific instructions for typical newsroom math in this Appendix A, an except from Sarah Cohen's outstanding book, Numbers in the Newsroom. It's worth getting your own copy if you don't already have one.

3.2 Overcoming your fear of math

When we learned to read, we got used to the idea that 26 letters in American English could be assembled into units that we understand without thinking – words, sentences, paragraphs and books. We never got the same comfort level with 10 digits, and neither did our audience.

Think of your own reaction to seeing a page of words. Now imagine it as a page of numbers.

Instead, picture the number "five". It's easy. It might be fingers or it might be a team on a basketball court. But it's simple to understand.

Now picture the number 275 million. It's hard. Unfortunately, 275 billion isn't much harder, even though it's magnitudes larger. (A million seconds goes by in about 11 days but you may not have been alive for a billion seconds – about 36 years.)

The easiest way to get used to some numbers is to learn ways to cut them down to size by calculating rates, ratios or percentages. In your analysis, keep an eye out for the simplest *accurate* way to characterize the numbers you want to use. "Characterize" is the important word here – it's not usually necessary to be overly precise so long as your story doesn't hinge on a nuanced reading of small differences. (And is anything that depends on that news? It may not be.)

Here's one example of putting huge numbers in perspective. Pay attention to what you really can picture - it's probably the $21 equivalent.

The Chicago hedge fund billionaire Kenneth C. Griffin, for example, earns about $68.5 million a month after taxes, according to court filings made by his wife in their divorce. He has given a total of $300,000 to groups backing Republican presidential candidates. That is a huge sum on its face, yet is the equivalent of only $21.17 for a typical American household, according to Congressional Budget Office data on after-tax income. *"Buying Power", Nicholas Confessore, Sarah Cohen and Karen Yourish, The New York Times, October 2015*

Originally the reporters had written it even more simply, but editors found the facts so unbelievable that they wanted give readers a chance to do the math themselves. That's reasonable, but here's an even simpler way to say it: "earned nearly $1 billion after taxes…He has given $300,000 to groups backing candidates, the equivalent of a dinner at Olive Garden for the typical American family , based on Congressional Budget Office income data." (And yes, the reporter checked the price for an Olive Garden meal at the time for four people.)

3.3 Put math in its place

For journalists, numbers – or facts – make up the third leg of a stool supported by human stories or anecdotes , and insightful comment from experts. They serve us in three ways:

- ***As summaries***. Almost by definition, a number counts something, averages something, or otherwise summarizes something. Sometimes, it does a good job, as in the average height of Americans. Sometimes it does a terrible job, as in the average income of Americans. Try to find summaries that accurately characterize the real world.
- ***As opinions***. Sometimes it's an opinion derived after years of impartial study. Sometimes it's an opinion tinged with partisan or selective choices of facts. Use them accordingly.
- ***As guesses***. Sometimes it's a good guess, sometimes it's an off-the-cuff guess. And sometimes it's a hopeful guess. Even when everything is presumably counted many times, it's still a (very nearly accurate) guess. Yes, the "audits" of presidential election results in several states in 2021 found a handful of errors – not a meaningful number, but a few just the same.

Once you find the humanity in your numbers, by cutting them down to size and relegating them to their proper role, you'll find yourself less fearful. You'll be able to characterize what you've learned rather than numb your readers with every number in your notebook. You may even find that finding facts on your own is fun.

3.4 Going further

Tipsheets

- Steve Doig's "Math Crib Sheet"
- Appendix A: Common newsroom math, adapted from drafts of the book *Numbers in the Newsroom*, by Sarah Cohen.
- A viral Twitter thread:

4 Census Data

We will be using data from the U.S. Census for many of these exercises. Here's a quick rundown of the origins and inner-workings of this important dataset.

Each decade, the Census Bureau counts every person living in the United States and the five U.S. territories. This is known as the Decennial Census and it is used to apportion seats in the U.S. House of Representatives, among other things.

In addition, the Census Bureau conducts ongoing survey of communities, known as the American Community Survey or ACS that provides more timely information about social, economic, housing, and demographic data every year. You can find information on housing, small business ownership, population profiles and much more. The ACS uses an annual sample size of about 3.5 million addresses and is collecting information daily.

Congressional lawmakers look at the ACS data to determine distribution of federal spending, among other things. Read more about census data here

Here's something important to know about ACS results: Data are pooled across a calendar year. So the ACS numbers reflect data collected over a period of time. By contrast, the Decennial Census is a single point-in-time count of the population.

We will be using ACS data to examine trends in wealth in Baltimore neighborhoods because this survey provides detail not available yet in the Decennial Census.

You can access Census Data through multiple ways. The U.S. Census Bureau offers data.census.gov. Using this site requires some training and patience. Here is a good place to start, a presentation the Census Bureau staff made to the Investigative Reporters and Editors conference in 2019.

Another useful resource is censusreporter.org, a site not affiliated with the Census Bureau that's designed to make it easier to navigate and retrieve the ACS data.

When we use R, we will use a software library called tidycensus that makes it very easy to retrieve Census data from the Census API, or application programming interface, basically a raw data feed optimized for R, python and similar programs. Stay tuned on that later this semester.

Data journalist Paul Overberg, now with The Wall Street Journal, compiled this useful guide about terminology when dealing with Census data.

5 Spreadsheets

Introduction

Some people consider using spreadsheets the table stakes for getting into data journalism. It's relatively easy to see what you're doing and you can easily share your work with your colleagues. In fact, pieces of the Pulitzer-Prize winning COVID-19 coverage from The New York Times was compiled using an elaborate and highly tuned set of Google spreadsheets with dozens of contributors.

This guide uses Google Sheets, which allows students to do the exercises regardless of their computer operating system, Mac, Windows or Linux. The exercises can be easily adapted to Microsoft Excel, which can handle larger datasets and has more options for pivot tables and other more advanced functions. However, Google Sheets have their own advanced functions for scraping websites or importing non-tabular file formats like JSON.

Most of the screen shots and instructions are created with a MacOS Monterey. Some come from earlier Mac versions, but are largely the same now. Windows users should replace any instructions for using the CMD- key with the CTL- key. There is a table that compares keystrokes for Apple desktops, laptops and Windows machines for Excel at the bottom of Spreadsheet Refresher

Tutorials

Spreadsheets are used in almost every workplace in America. This section covers most of what you need in the newsroom, which is a different set of skills than in other businesses.

- Spreadsheet Refresher : Start over with good habits
- Sorting and filtering to find stories : The first step of interviewing data
- Grouping with pivot tables: Aggregating, and the super power of spreadsheets
- Spreadsheet Formulas: Percents, sums, and other basic computations used in newsrooms.

6 Spreadsheet Refresher

Spreadsheets are everywhere, so it's worth re-learning how to use them well. Reporters usually use spreadsheets in three ways:

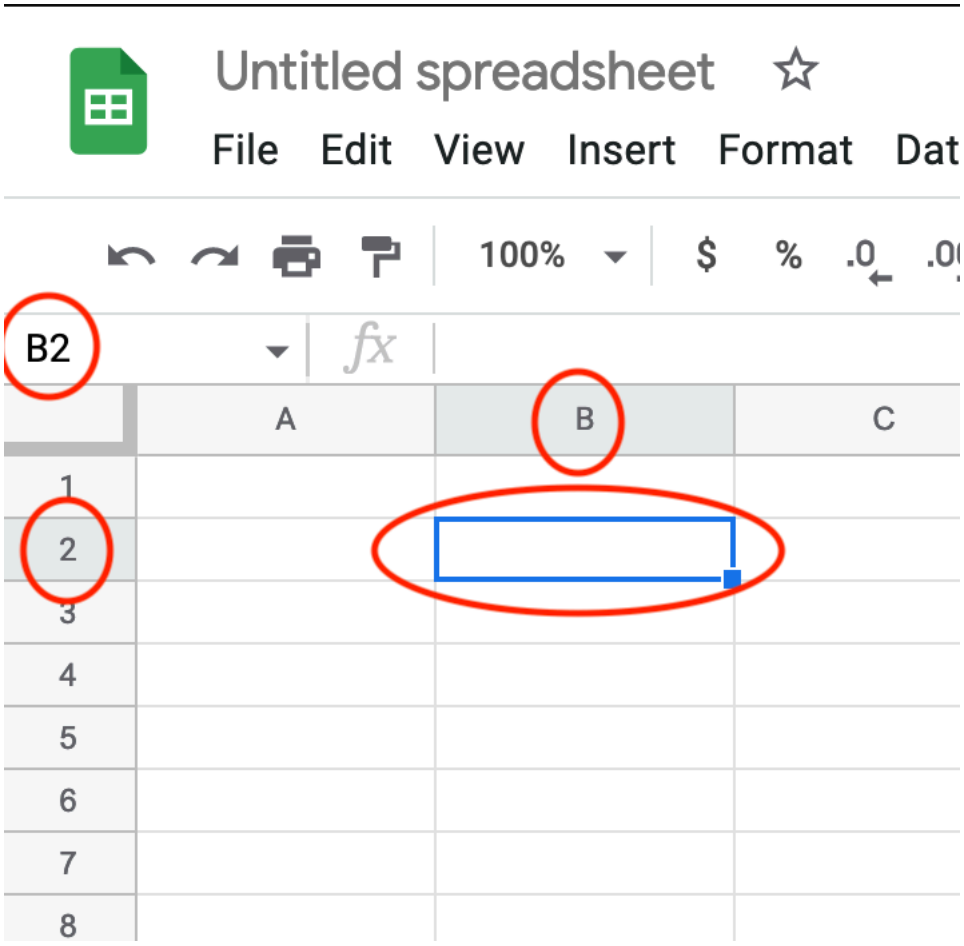
- To create original databases of events for sorting, filtering and counting. Examples include a long-running court case; the details of each opioid death in a city; a list of police shootings and their documents; or even a list of your own public records requests or contact log.
- To use data created by others for fast, simple analysis and data cleanup. Many government agencies provide their information in spreadsheet form, but they often require some rejiggering before you can use them.
- To perform simple, straightforward analysis on data and share with team members. This is becoming less common as more reporters learn programming languages, but it's still common in newsrooms to share data, especially through Google Sheets.

This guide will Google Sheets since the program is available to anyone regardless of operating system. Google Sheets are easy to share for reporting teams.

Some reporters flinch at typing in 30 or 100 entries into a spreadsheet. You shouldn't. If you learn to take notes in a structured way, you'll always be able to find and verify your work. If you try to calculate a sum of 30 numbers on a calculator, you'll have to type them all in at least twice anyway. Also, getting used to these easy tasks on a spreadsheet keeps your muscles trained for when you need to do more.

6.1 Re-learning Excel from the ground up

The spreadsheet grid

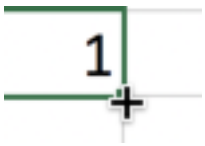


::: {.pl-2rem .float-right .col-md-4}

When you start up a spreadsheet, you'll see letters across the top and numbers down the side. If you ever played Battleship, you'll recognize the idea – every little square, or cell, is referenced by the intersection of its column letter and row number:

B2 is the cell that is currently active. You can tell because it's outlined in the sheet and it's shown on the upper left corner.

Mouse shapes



The Copy Tool, or the thin black cross. When you see this, you'll copy anything that's selected. This can be good or bad.

The Evil Hand. If you use this symbol, you will MOVE the selection to a new location. This is very rarely a good idea or something you intend.

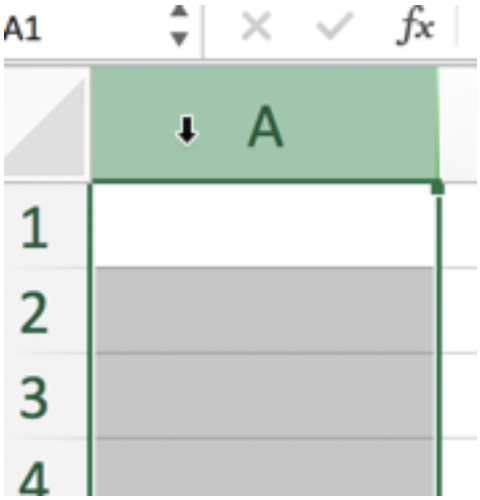
Selecting cells and ranges

Spreadsheets act only on the cells or regions you have selected. If you begin typing, you'll start entering information into the currently selected cell.

To select: Hold the cursor over the cell and click ONCE – *not twice*. Check the formula bar to make sure you've selected what you think you've got. You can also look at the bottom right of your spreadsheet for more information.

You'll often work with *ranges* of cells in formulas. These are defined by the corners of the area you want to work on – often a column of information. In the example below, the range is A1:C6, with the ":" referring to the word "through".

To select a group of cells and act on them all at once: Hover the cursor over one corner, click ONCE and drag to the diagonal corner. Make sure the Evil Hand is nowhere to be seen. The entire area will be shaded in except for the currently selected cell. Look at the upper right corner to see how many rows and columns you selected.



To select a column or row : Hover the cursor over the letter at the top of the column. For a row, hover it over the row number in the margin

Reading the screen

The areas of the spreadsheet have different visual clues, and learning to read them will make your life much easier.

This image shows some key areas on the screen when you're just viewing the sheet:

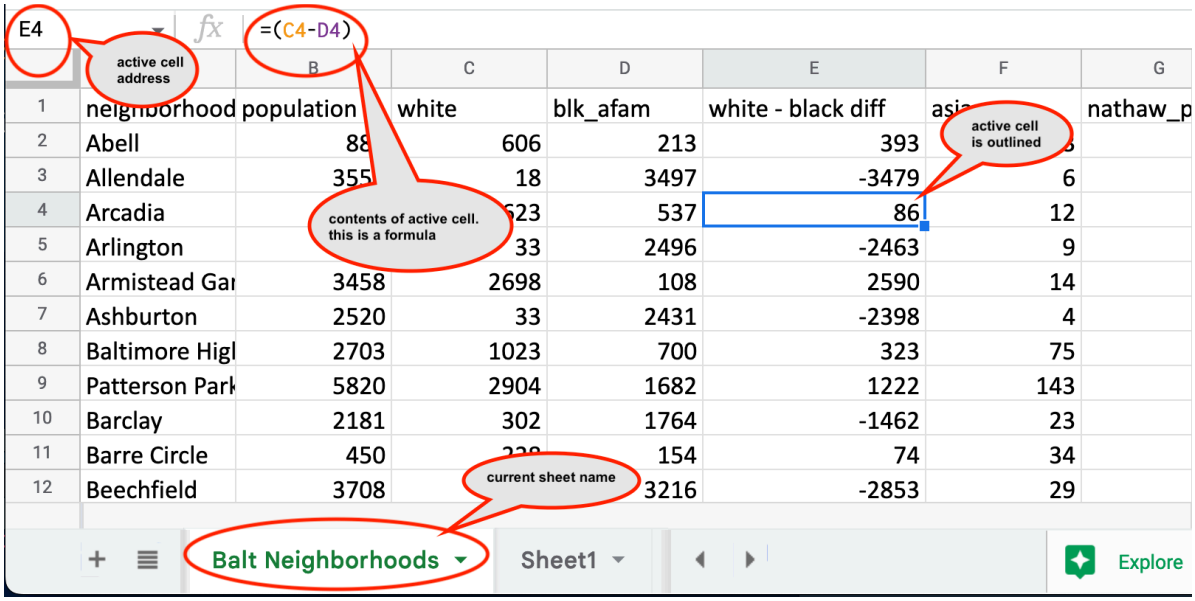
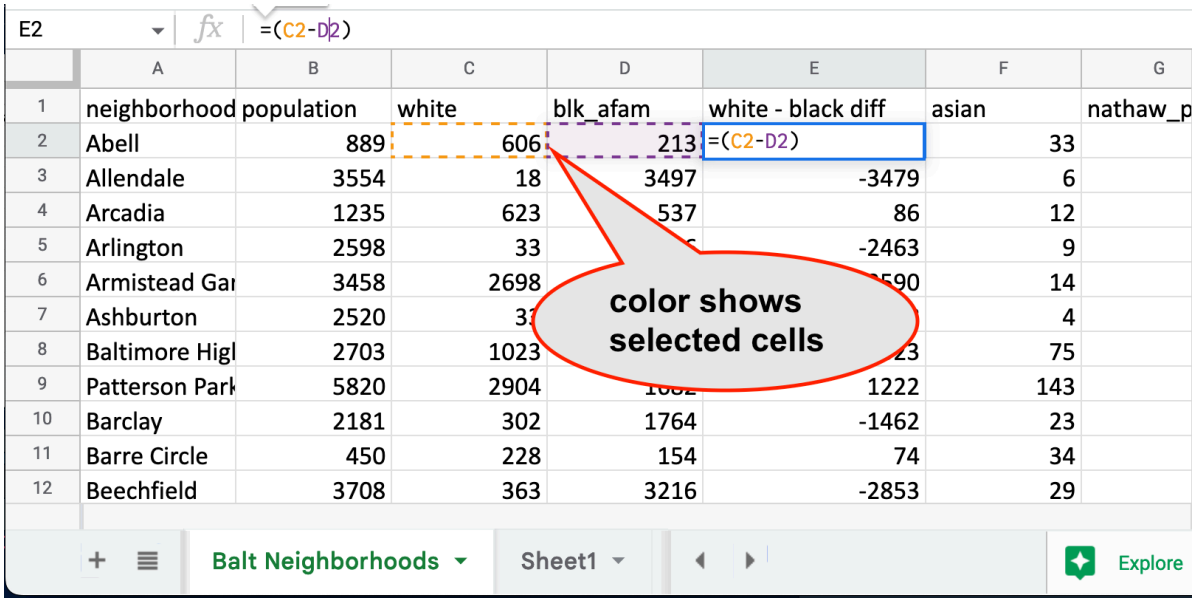


Figure 6.1: ready

This is how it changes when you're editing



	A	B	C	D	E	F	G
1	neighborhood	population	white	blk_afam	white - black diff	asian	nathaw_p
2	Abell	889	606	213	=(C2-D2)	33	
3	Allendale	3554	18	3497	-3479	6	
4	Arcadia	1235	623	537	86	12	
5	Arlington	2598	33		-2463	9	
6	Armistead Gar	3458	2698		590	14	
7	Ashburton	2520	3			4	
8	Baltimore Higl	2703	1023		23	75	
9	Patterson Park	5820	2904		1222	143	
10	Barclay	2181	302	1764	-1462	23	
11	Barre Circle	450	228	154	74	34	
12	Beechfield	3708	363	3216	-2853	29	

Figure 6.2: editing

Entering data

Select the cell and start typing. The information you type won't be locked into the cell until you hit the Return / Enter key, or move your selection to another cell. Hit "Escape" to cancel the entry.

You can't do a lot of things while you're editing, so if you have a lot of greyed out menu items, look at your formula bar to see if you are still editing a cell.

If you're having trouble getting to a menu item or seeing the result of your work, try hitting "Escape" and try again. You may not have actually entered the information into the sheet.

Locking in headings

As your spreadsheet grows vertically with more rows, you'll want to be able to see the top all the time. When it grows horizontally with more columns, you'll probably want to see columns in the left, such as names. This is called "Freezing Panes" – you freeze part of the page so it stays in place when you move around.

Select View in the menu, then Freeze, then the number of rows to freeze. Select 1 row. As you now scroll down the sheet, your headings will remain but you can see the data as you move deeper into the sheet.

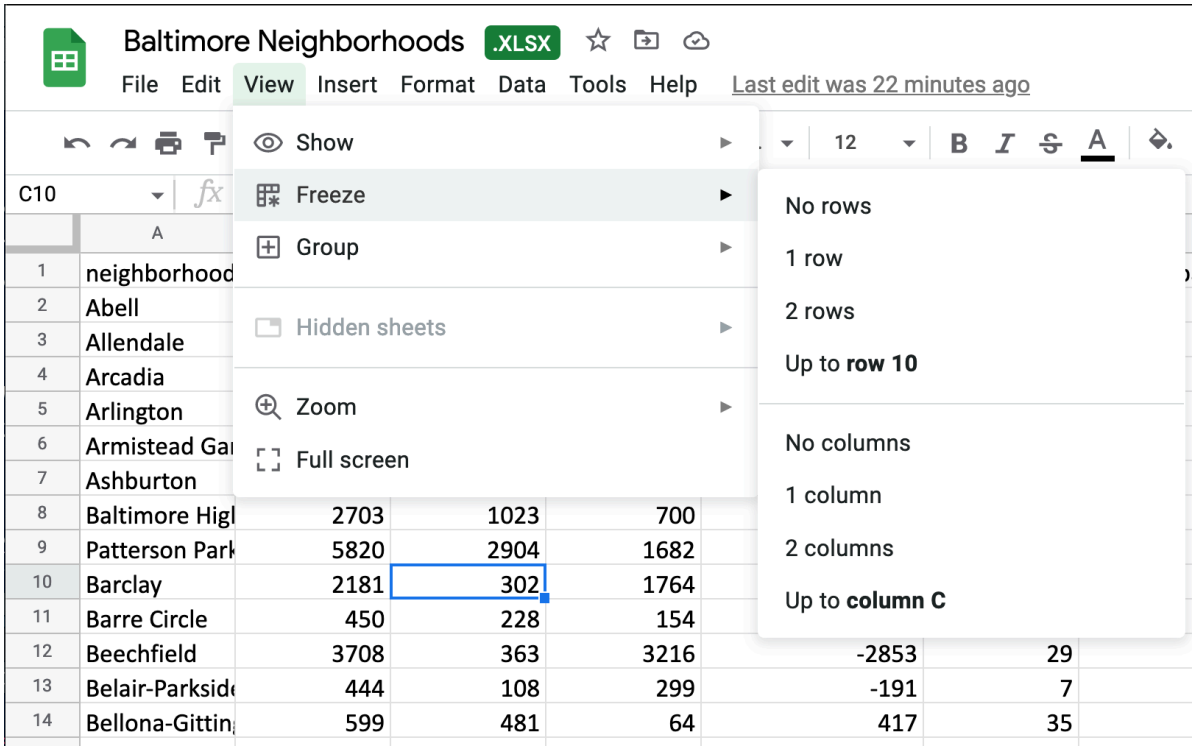


Figure 6.3: freeze panes

Formatting tricks

- Use the buttons or the format dialog box to make numbers easier to read.
- If a column is filled with a lot of text, select Format, then Wrapping from the menu to wrap text. This means that when you double-click to widen a column, it will get taller, not wider. This is good when you need to save valuable real estate on the screen.

6.2 Getting started with a dataset

SLOW DOWN! Don't do anything until you understand what you have in front of you and can predict what your next mouse click will do to it.

Most data we encounter was created by someone else for some purpose other than ours. This means that you can't assume anything. It may not be complete. It may be inaccurate. It may mean something completely different than it appears at first blush.

First steps

- Document where you got the spreadsheet and how you can get back to the original. Create a new tab (click the + sign in the lower left), name it Data Dictionary, copy the URL of your source data and any other notes about it. Make this your regular practice. It will save time and stress on deadline.
- Read anything you can about what it contains. Look for documentation that comes with the data.
- Save the original into a safe place with its original name and metadata. Work on a copy.
- If the spreadsheet shows #### instead of words or numbers, widen your columns. If it shows 7E-14 or something like that, format them as numbers, not "General".
- Check your corners – look at the top left and bottom right. Is the data all in one area? Are there footnotes or other non-data sections mixed in? We're going to want to fix that later.

Interview your data

Headings

The most fraught part of data reporting is understanding what each *column* actually means. These often have cryptic, bureaucratic names. You may need to go back to the source of the data to be sure you actually understand them.

If your data doesn't have any headings, that's going to be your first priority. In effect, you'll need to build what we call a *data dictionary* or *record layout* if one hasn't been provided. Many reporters create these as a page in a dataset.

Unit of analysis

A *unit of analysis* refers to the items that are listed in the rows of your dataset. Ideally, every row should be at the same unit of analysis – a person, an inspection, or a city, for example. Summaries should be separated by a blank row, or moved to a different sheet. Think of this as the noun you'd use to describe every row.

Row numbers

The data was probably given to you in some sort of natural sort order. Different computer systems sort differently – some are case-sensitive, others are not. It may depend on when and where the data as created! The order of the data may even depend on a column you don't have. If you don't do something now, you'll never be able to get back to the original order, which could have meaning for both the agency and for fact-checking.

6.3 Video walkthrough

These first steps, along with adding an ID row, are shown here. You can follow along with the same dataset.

Getting started with Google Sheets

[Getting started with Google Sheets](https://www.youtube.com/embed/1hGoYzmkhfc)

6.4 Keyboard shortcuts

Google Sheets keyboard shortcuts can be found in the menu: Help, then Keyboard Shortcuts.

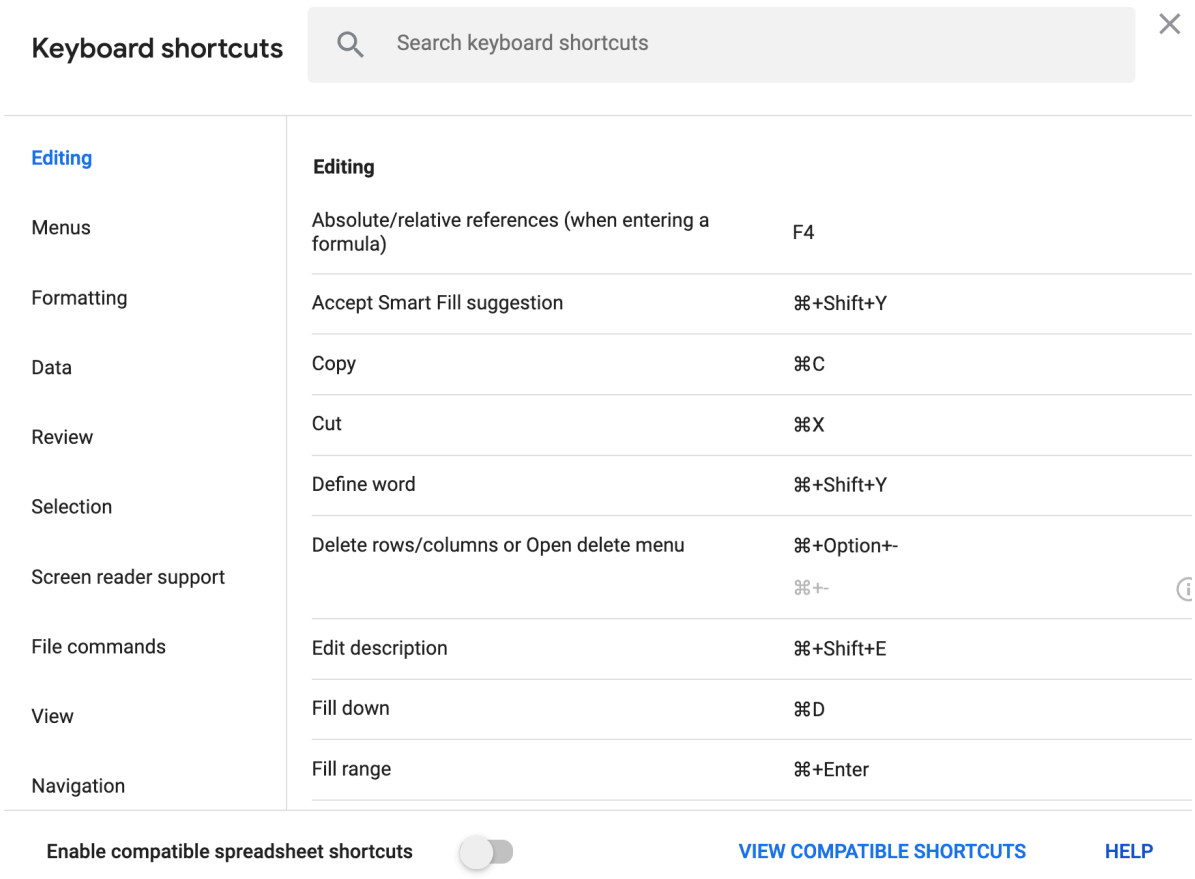


Figure 6.4: Keyboard shortcuts

7 Sorting and filtering to find stories

7.1 A sorting miracle

After police in Ferguson, Mo., killed Michael Brown in 2014, advocates and journalists began examining the racial and ethnic gaps between police departments and the communities they served.

The New York Times found a 7-year-old survey conducted by the Justice Department that allowed it to compare the data for major cities in a standalone graphic that it published later that year.

When newer data reflecting departments' makeup in 2012 was released a year later, Matt Apuzzo and Sarah Cohen hoped it would show some differences. It didn't. So we were left trying to find news in the data that was clearly of public interest.

Cohen matched up the demographics of police departments with their cities and then started sorting, filtering and Googling. Could there be news in the outliers on the list? Which departments most closely represented their communities? Which ones had unusually large gaps?

Cohen quickly stumbled on telling anecdote to frame the story: Inkster, Mich. had one of the least representative departments in the country, and had recently hired a new police chief to help mend the department's fraught relationship with its largely African-American community. Where had he come from? Selma, Ala., one of the most representative police departments in the nation. Interviews with the chief, William T. Riley III, suggested one reason for some cities' disparities: there was no state or federal money to pay for training new police officers.

The story, "Police Chiefs, Looking to Diversity Forces, Face Structural Hurdles" helped explain the persistent gap between the makeup of police in some areas and the communities they served.

7.2 Sorting and filtering as a reporting tool

Sorting and filtering can:

- Narrow your focus to specific items that you want to examine in your story.

Figure 7.1: Chief William T. Riley III. Credit: Laura McDermott for The New York Times

- Show you rows containing the highest and lowest values of any column. That can be news or it can be errors or other problems with the data.
- Let you answer quick "how many?" questions, with a count of the rows that match your criteria. (In the next lesson, you'll see that pivot tables, or group-by queries, are much more powerful for this in most cases.)

7.3 Example data

Data from the Washington Post police shootings database for use in this tutorial - Documentation from the Post's github site :::

- The data for this and several other chapters is the Washington Post's public data collection of police shootings in the U.S. It includes the nation's best guess about each fatal police shooting since 2015. There are a couple of caveats:
- It excludes deadly police interactions other than shooting a firarem at the suspect. Any strangulation, car crashes, Tasers without guns or other methods are excluded.
- It is based primarily on news reports and the results public records requests so it often contains the story as told by police. We know that many of those reports are sugar-coated at best, and lies at worst.
- The Post says this is a list of fatal shootings, but doesn't say what happens if more than one person is killed. The 2019 shooting of D'Angelo Brown & Megan Rivera in West Memphis is shown as two rows[1] in the data even though it was one event. So each row might be considered a shooting "victim", a "suspect" or a shooting "fatality" rather than a "shooting".

The screenshots in this tutorial may not match exactly to what you get on the Washington Post data. This tutorial used data current to Aug. 3, 2022.

It's a good example set for us because it's been used as the basis of many stories, it has at least one of each *data type* that we plan to deal with in Google Sheets, and it is well documented on the Post's github site.

7.4 Get the data into Google Sheets

- Download the [police shooting data from the Washington Post](https://github.com/washingtonpost/data-police-shootings/releases/download/v0.1/fatal-police-shootings-data.csv
- Open Google Sheets. File | Import | Upload | Select the downloaded file "fatal-police-shootings-data.csv" . After it uploads, select the green "Import Data" button.

[1]Finding these is something that's pretty hard in a spreadsheet but will be really easy in R.

•

7.5 Understanding data types

When you open the spreadsheet, the first thing to notice is its *granularity*. Unlike Census or budget spreadsheets, this is a list capturing specific characteristics of each fatality Each column has the same *type* of data from top to bottom. Those types are:

- **Text**. Text or "character" columns can come in long or short form. When they are standardized (the values can contain only one of a small list of values), they're called "categorical". If they're more free-form, they're might be called "free text". The computer doesn't know the difference, but you should. The Post data has examples of both. In spreadsheets, text is left-justified (they move toward the left of the cell and will line up vertically at the beginning)
- **Numbers**. These are pure numbers with no commas, dollar signs or other embellishments. In Google Sheets, as we'll see in the computing section, these can be formatted to *look* like numbers we care about , but underneath they're just numbers. Adding up a column of numbers that has a word in it or has missing values will just be ignored in Google Sheets. It will trip up most other languages. These are right-justified, so the last digit is always lined up vertically.
- **Logical**: This is a subset of text. It can take one of only two values – yes or no, true or false. There is no "maybe".
- **Date and times**: These are actual dates on the calendar, which have magical properties. Underneath, they are a number. In Google Sheets, that number is the number of days since Jan. 1, 1900.[2] They can also have time attached to them, which in Google Sheets is a fraction of a day. What this means is that the number 44,536.5 is really Dec. 6, 2021 at noon. In Google Sheets, you use a format to tell the spreadsheet how you want to see the date or time, just the way you look at dollar values with commas and symbols. (If you get a spreadsheet with a lot of dates of 1/1/1900, it means there is a 0 in that column, which is sometimes a fill-in for "I don't know.")

Here's a picture of a date that is shown in a variety of formats.

Unformatted	Formatted values					
As a number	"Short date"	"Long date"	Time	Date & mil. time	Month	Day of the week
44540.87431	12/10/21	Friday, December 10, 2021	8:59:00 PM	12/10/21 20:59	Dec. 2021	Friday

Figure 7.2: date formats

[2]Each language deals with dates and times a little differently. We'll see how R does it later on. But just know that dates can be tricky because of these differences and time is even more tricky

All of these are the same, underlying value – the number at the left. Notice that all of these are right-justified.

This means that when you see "Friday, December 10", the computer sees 44540.87431. When you put the dates in order, they won't be alphabetized with all of the Fridays shown together. Instead, they'll be arranged by the actual date and time.

It also means that you can compute 911 response times even when it crosses midnight, or or compute the someone's age today given a date of birth. Keeping actual calendar dates in your data will give it much more power than just having the words. (Google Sheets uses the 1st of the month as a stand-in for an actual date when all you know is the month and year.)

7.5.1 Sorting rows

Sorting means rearranging the rows of a data table into a different order. Some reporters take a conceptual shortcut and call this "sorting columns". That thinking will only get you into trouble – it lets you forget that you want to keep the rows in tact while changing the order in which you see them. In fact, in other languages it's called "order by" or "arrange" by one or more columns – a much clearer way to think of it.

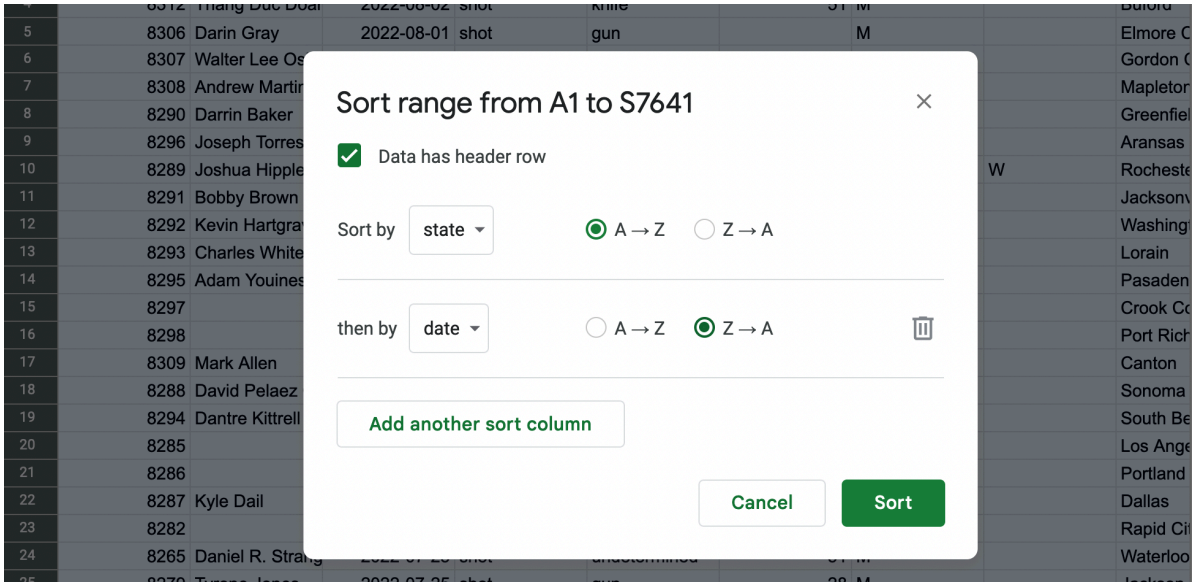
In Google Sheets, look for the sort options under the Data tab at the top of your screen. In this case, sorting from oldest to newest gives you a list of the fatalities in chronological order, including the time of day.

To sort your data:

- Make a copy of your data. Left click on the "fatal-police-shootings-data" tab, select Duplicate
- Select your data by clicking the box above Row 1 and to the left of Column A
- Select Data | Sort Range | Advanced Range Sorting Options
- Click "Data has header row" and then select date from the Sort by dialog box. Select Z –> A
- Select Sort

Adding fields to the sort

Adding more columns to the sort box tells Google Sheets what to do when the first one is the same or tied. For example, sorting first by state then by date gives you a list that shows all of the events by state in sequence:



7.5.2 Filtering

Filtering means picking out only some of the rows you want to see based on a criteria you select in a column. Think of it as casting a fishing net – the more filters you add, the fewer fish will be caught.

To activate filters in Google Sheets, from the Menu:

- Data | Filter Views | Create a New Filter View
- You'll see little triangles next to the column headings.

Click the "armed" heading. You will see options for various weapons. All are selected by default with a check mark. To select just "ax", click on clear and then select "ax." The sheet how is filtered to just weapons using an ax. To remove the filter, repeat the steps and "select all" and the entire sheet is displayed again.

Each filter you select adds more conditions, narrowing your net.

To find fatalities that involved a firearm with a Taser, use the drop-down menu under `manner_of_death` select "shot and Tasered".

This method works for small-ish and simple-ish columns. If your column has more than 10,000 different entries, such as names or addresses, only the first 10,000 will be considered. We only caught these for stories when someone did a fact-check using a different method of filtering. If your column has a lot of distinct entries, use option that says "Choose One", and then use the "Contains" option. Better yet, don't use filtering for counting things at all.

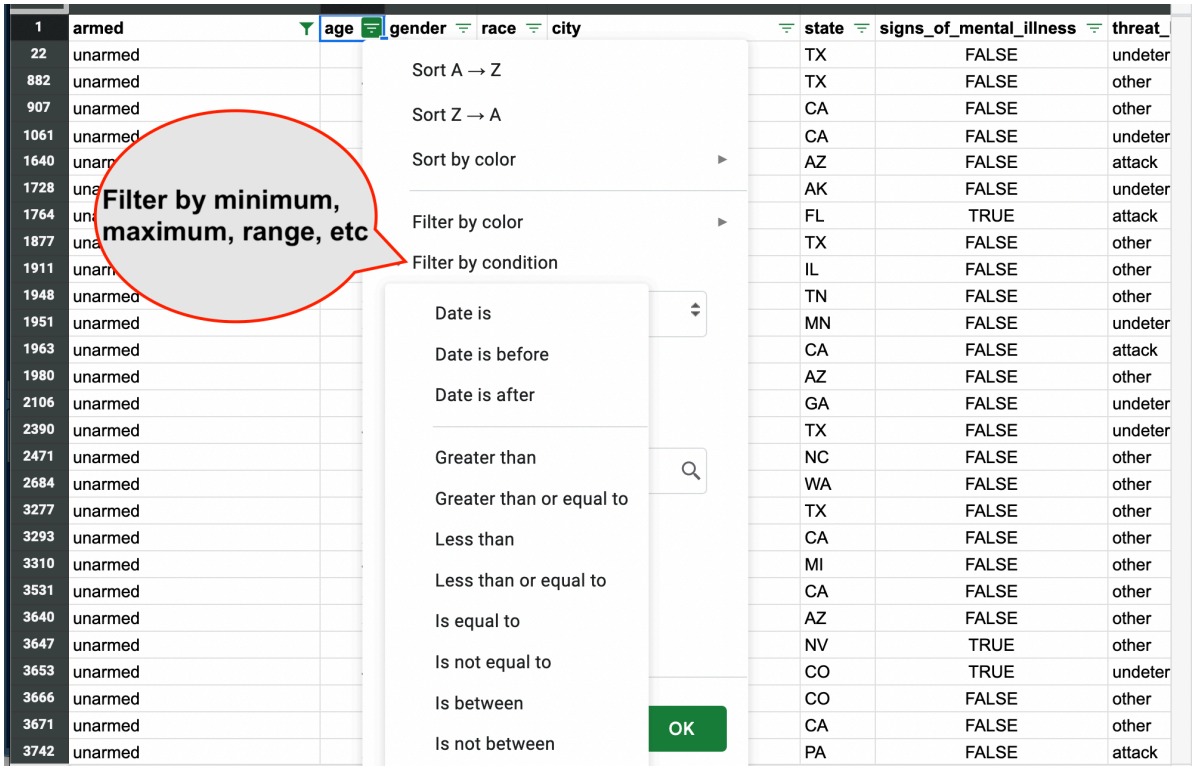
Add more filters to narrow down your list of cases even more. For example, the New York Times ran a series of stories in 2021 about unarmed people shot by police. One story was about those who were fleeing by car. Here's one way to get a preliminary list of those cases:

1. Remove any filter you already have on.
2. Turn on the filters again if you turned them off.
3. Choose "unarmed" under `armed` and "car" under `flee`.

(Of course, the Times didn't stop there in trying to find more cases and teasing out more of them from this and other data. But this is a start.)

Different kinds of filters

There are several filter options. You can filter by various conditions. For numerical data, you can set a minimum or maximum value or a range of values. This is useful for dates to specify a certain time period. For text, you can filter if a word contains a few letters, useful to capture spelling variations.



8 Spreadsheet Formulas

The quick review of math in Google Sheets uses the City of Baltimore's 2022 budget, compared with previous years.

Get the Google Sheet to follow along

You should get into the habit of creating unique identifiers, checking your corners and looking for documentation before you ever start working with a spreadsheet. These habits were covered in Replication and the data diary and on an Excel refresher .

8.1 Formulas in spreadsheets

Every formula begins with the equals sign (=). Rather than the values you want to work with in the formula, you'll use *references* to other cells in the sheet.

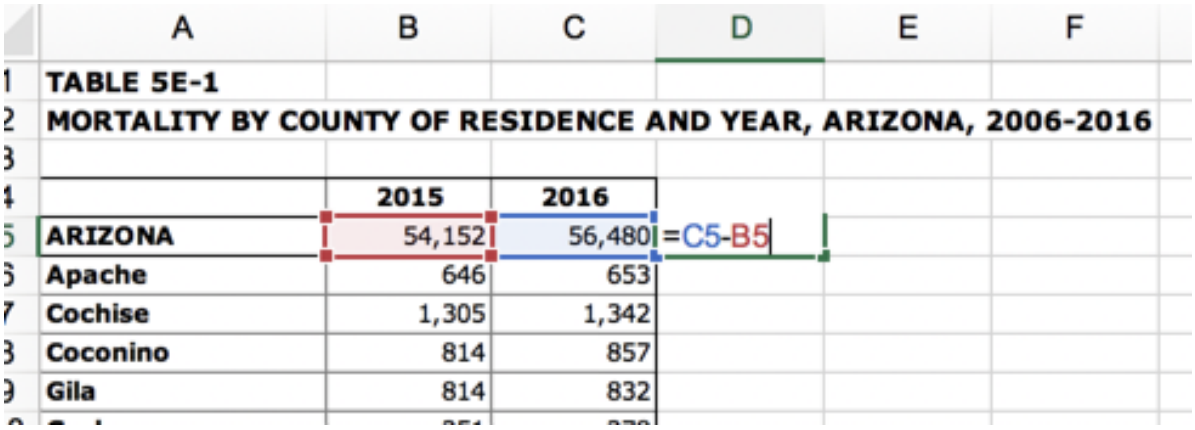
The easiest formulas are simple arithmetic: adding, subtracting, multiplying and dividing two or more cells. You'll just use simple operators to do this:

operator	symbol	example
addition	+	=A2+B2
subtraction	-	=A2-B2
multiplication	*	=A2*B2
division	/	=A2/B2

Here's what a spreadsheet looks like while editing some simple arithmetic:

The other kind of formula is a *function*. A function is a command that has a name, and requires *arguments* – usually the cell addresses or the range of addresses that it will act on. Every programming language has functions built in and many have extensions, or packages or libraries, that add even more as users find things they want to do more efficiently. You begin using a function the same way you begin a formula – with an = sign. Here are three common functions that create summary statistics for the numbers contained in a *range* of addresses. A range is a set of cells defined by its corner cell address: the top left through the bottom right.

You'll usually use them on a single column at a time.



	A	B	C	D	E	F
1	TABLE 5E-1					
2	MORTALITY BY COUNTY OF RESIDENCE AND YEAR, ARIZONA, 2006-2016					
3						
4		2015	2016			
5	ARIZONA	54,152	56,480	=C5-B5		
6	Apache	646	653			
7	Cochise	1,305	1,342			
8	Coconino	814	857			
9	Gila	814	832			

Figure 8.1: formula

Formula	What it does
=SUM(start:finish)	Adds up the numbers between `start` and `finish`
=AVERAGE(start:finish)	Computes the mean of the numbers
=MEDIAN(start:finish)	Derives the median of the numbers

…where "start" means the first cell you want to include, and finish means the last cell. Use the cell address of the first number you want to include , a colon, then the cell address of the last number you want to include. You can also select them while you're editing the formula.

Here's an example of adding up all of the rows in a list by county:

8.2 Common spreadsheet arithmetic

The budget document shows three years' of data: The actual spending in the fiscal year that ended in 2016; the spending that was estimated for the end of fiscal year 2017; and the proposed spending for fiscal year 2018. The first page of the document shows these amounts for broad spending categories.

You may want to widen the columns and format the numbers before you start:

8.2.1 Check the government's math with SUM

Our first job is to make sure the government has provided us data that adds up. To do that, we'll SUM all of the departments' spending.

To add up the numbers from FY 2020 Actual, enter the following formula in cell C16, just below the number provided by the government:

```
=SUM(C2:C13)
and hit the enter key
```


Copy that formula to the right. Notice how the formula changes the addresses that it is using as you move to the right – it's adjusted them to refer to the current column.

	A	B	C
1		year_2015	year_2016
2	Apache	646	653
3	Cochise	1,305	1,342
4	Coconino	814	857
5	Gila	814	832
6	Graham	251	278
7	Greenlee	69	52
8	La Paz	254	270
9	Maricopa	28,945	30,311
10	Mohave	3,024	3,181
11	Navajo	907	1,010
12	Pima	9,241	9,492
13	Pinal	2,968	2,991
14	Santa Cruz	294	301
15	Yavapai	2,918	2,955
16	Yuma	1,427	1,506
17	Unknown	275	449
18			
19		=SUM(B2:B17)	

Figure 8.2: formula

8.2.2 Change in spending

The increase or decrease in projected spending from 2017 to 2018 is just the difference between the two values, beginning in cell F3

```
new-old, or  =E2-D2
```

When you copy it down, note how the references to each row also adjusted. In line 3, it's E3-D3, and so on. Excel and other spreadsheets assume that, most of the time, you want these kinds of adjustments to be made.

8.2.3 Percent change

We can't tell the *rate* of growth for each department until we calculate the percent change from one year to another. Now that we already have the change, the percent change is easy. The formula is:

```
( new - old ) / old

.. or just scream "NOO"
```

The new-old is already in column F, so all that's left is to divide again. In grade school, you also had to move the decimal place over two spots, since the concept of percent change is "out of 100". Excel formats will do that for you.

Remember, it's always (new-old)/old , **NOT** the big one minus the little one. Doing it correctly, the answer could be negative, meaning the value fell.

B	C	D	E	F	G
	Act 2016	Est 2017	Budget 2018	change from 2017	percent chan
rnment	$103,679	$111,601	$129,653	$18,052	=F2/D2
	$874,560	$930,155	$1,032,609	$102,454	
ce	$53,367	$61,104	$63,272	$2,168	
on	$541,951	$605,421	$650,439	$45,018	
levelopment	$176,318	$207,237	$239,428	$32,191	
nrichment	$243,029	$259,893	$268,072	$8,179	
al services	$378,827	$429,960	$450,858	$20,898	

Figure 8.3: formula

When you're done, you can format the answer as a percentage to get it into whole numbers.

It's also worth comparing the picture you get by looking at raw numbers vs. percentages. It's instructive that federal grants are up 308%.

8.2.4 Parts of a whole: percent of total

We'd also like to know what portion of the total spending is eaten up by each department. To do that, we need the percent of total.

In our case, let's use the total that the government gave us. In practice, you'd have to decide what to do if your figures didn't match those provided by officials. You can't assume that the total is wrong – you could be missing a category, or there could be a mistake in one of the line items.

The formula for percent of total is:

```
category / total
```

Here's a good trick with spreadsheets when you need to divide against a fixed total. You don't have to type in each formula one by one, though. Instead, you'll use anchors, known in spreadsheets as "absolute references". Think of a dollar sign as an anchor or stickpin, holding down the location of part of your formula. If you put the stickpin before the letter in the formula, it holds the column in place. If you put it before the number, it holds the row in place. If you put it in both places, it holds the cell in place.

In this case, we want to see what percentage property taxes, income taxes, etc. are of the total revenues in FY22, which is $4,331,049,486 (cell E14). Let's figure it out.

- Left click column F, insert column to the left. Name it Pct of Total
- Create formula to divide property taxes into total revenues: =(E2/E14)
- Modify forumula so it will anchor to the E14 as you move down the spreadsheet =(E2/E14)
- Copy formula down to F13

8.3 While we're at it: two kinds of averages

Although it doesn't make a lot of sense in this context, we'll go ahead and calculate the *average* or *mean* size of each department, and then calculate the *median* size.

Simple average, or mean

A simple average, also known as the mean, is skewed toward very high or very low values. Its formula is

```
sum of pieces / # of pieces that were summed
```

But in Google Sheets, all we need is the word AVERAGE:

```
=AVERAGE(C2:C9)
```

Median

In Google Sheets, you can get the median of a list of numbers by just using the formula, MEDIAN()

```
= MEDIAN(C2:C9)
```

Doing simple calclutions like this on data that is provided to you by the government lets you ask better questions when you get an interview, and may even convince officials to talk with you. There's a big difference between asking them to tell you what the budget numbers are, and asking them to explain specific results!

8.4 FAQs

Should I use average or median?

It depends. Averages are easier to explain but can be misleading. Usually, if they're very different, median will be a better representation of the typical person, city or department. Averages in these cases are more like totals.

My percents are small numbers with decimal points

Use the format as a % button to move the decimal point over two places and insert the percentage symbol.

9 Grouping with pivot tables

In the wake of a police shooting in 2016, reporter Mitch Smith obtained a list of traffic stops from the St. Anthony Police Department in Minnesota. He was writing a story on Philandro Castile's death and was running out of time. He wanted to answer a simple question: Were minority motorists more likely to be stopped in St. Anthony than whites?

Rob Gebeloff made a quick pivot table to answer the question. That night, Smith wrote:

> In each of the three small suburbs patrolled by the St. Anthony police, less than 10 percent of the population is black. But data released by the city on Tuesday showed that a far higher percentage of the people ticketed or arrested by St. Anthony officers were African-American.
>
> Last year, around 19 percent of those cited by St. Anthony police were black, as were roughly 41 percent of people arrested by the department, a review of the city's data showed. Those percentages do not include the large number of defendants whose race was unknown.

Summarizing a list of items in a spreadsheet is done using pivot tables. In other languages, it's considered "aggregating" or "grouping and summarizing". Think of pivot tables and grouping as answering the questions, "How many?" and "How much?" They are particularly powerful when your question also has the words "the most" or the "the least" or "of each". Some examples:

- Which *Zip Code* had *the most* crimes?
- What *month* had *the least* total rainfall?
- *How much* did *each candidate* raise last quarter?
- In playing cards, *how many* of *each suit* do I have in my hand?
- On average, are *Cronkite students taller or shorter* than in other schools?

Confusing grouping with sorting or arranging

Many reporters confuse this summarization with "sorting". One reason is that this is how we express the concept in plain language: "I want to sort Skittles by color."

But in data analysis, sorting and and grouping are very different things. *Sorting* involves shuffling a table's rows into some order based on the values in a column. In other languages,

this is called *arranging* or *ordering*, much clearer concepts. *Grouping*, which is what pivot tables do, is a path to aggregating and computing summary statistics such as a count (the number of items), sum (how much they add up to), or average for category. It means "make piles and compute statistics for each one."

When to use filter vs. pivot tables

Something that trips up beginners is a desire to see details and totals at the same time, which is more difficult than it sounds.

A filter is used to *display* your selected items as a list. You'll get to see all of the detail and every column. As a convenience, Google Sheets shows you how many items are in that filtered list. That's great when you want to just look at them, or get more information about them. For instance, if you had a list of crimes by ZIP code, you might just want to see the list in your neighborhood – where, exactly, were they? When did they happen? Was it at night or the morning? What crimes happened on which blocks?

A pivot table is used when you *just want to see summaries* – does my ZIP code have more crime than others? Are robberies more common than car theft in my Zip code, and how does that compare to others?

In practice, you'll go back and forth between summary and detail. They're both important, just different.

9.1 Tutorial

::: {.alert .alert-info .opacity-2} We will continue to use data from the Washington Post police shootings database for this tutorial.

First, let's modify this spreadsheet to include the descriptions of race and ethnicities: A for Asian, B for Black, etc.

- Select Column I, city, and insert a new column to the left. Name it race_ethicity
- Create a filter. Select race, filter for A
- Type Asian in Column I. Copy Asian down the entire column so every A in column H corresponds with Asian in Column I
- Repeat: B = Black. H = Hispanic. W = White, non-Hispanic, N= Native American, O=Other, blanks=Unknown.

Follow this video to see the process.

Setting up the pivot table

From the main menu on Google Sheets, choose *Insert*, then *Pivot table*, then New sheet.

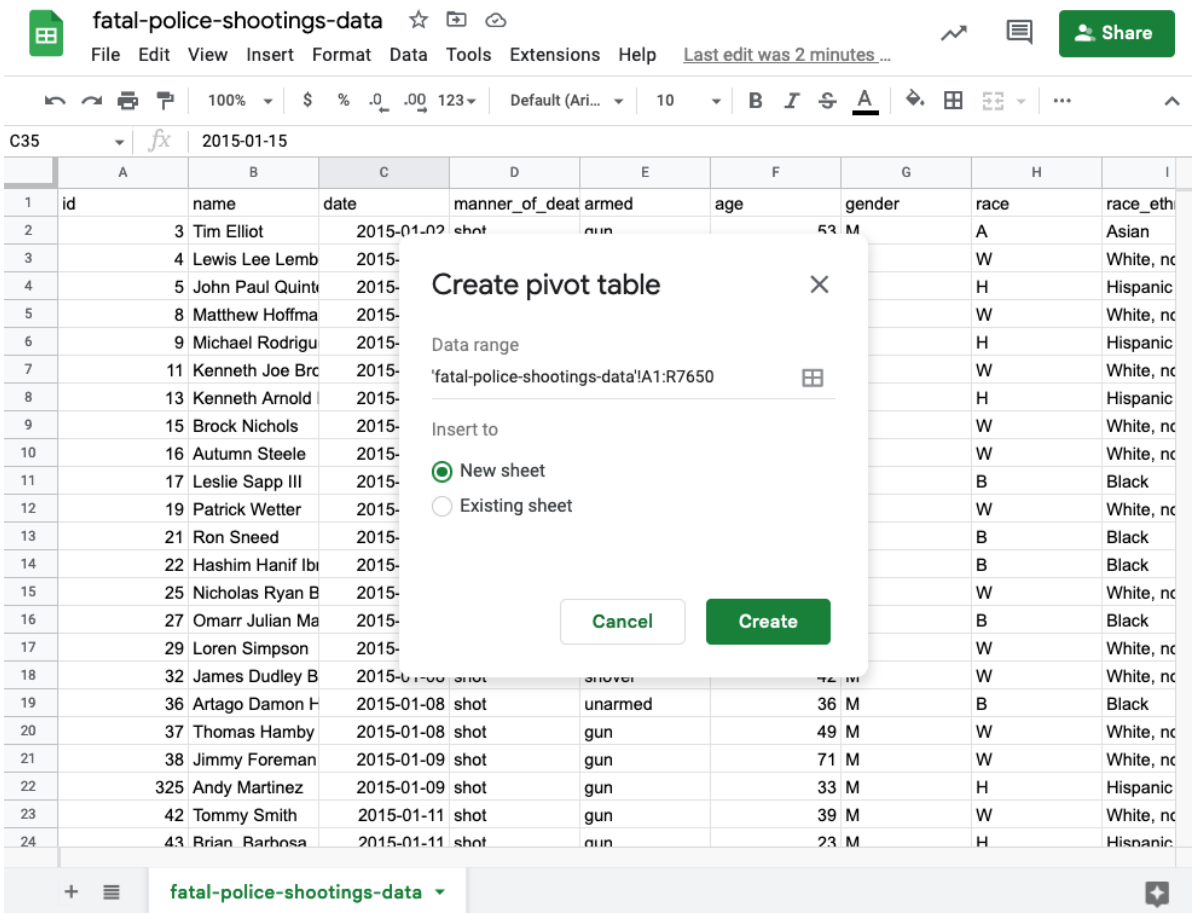


Figure 9.1: insert menu

Next, you will see the Pivot Table editor. Here's what it looks like:

Counting , or "how many"?

For Rows, select Add and then race_ethnicity. For values, select Add and then state. You will now see all of the race and ethnicity totaled.

We're totalling on state because it's good to have something that's always filled out into the Values area (`state` is a safe one in this data).

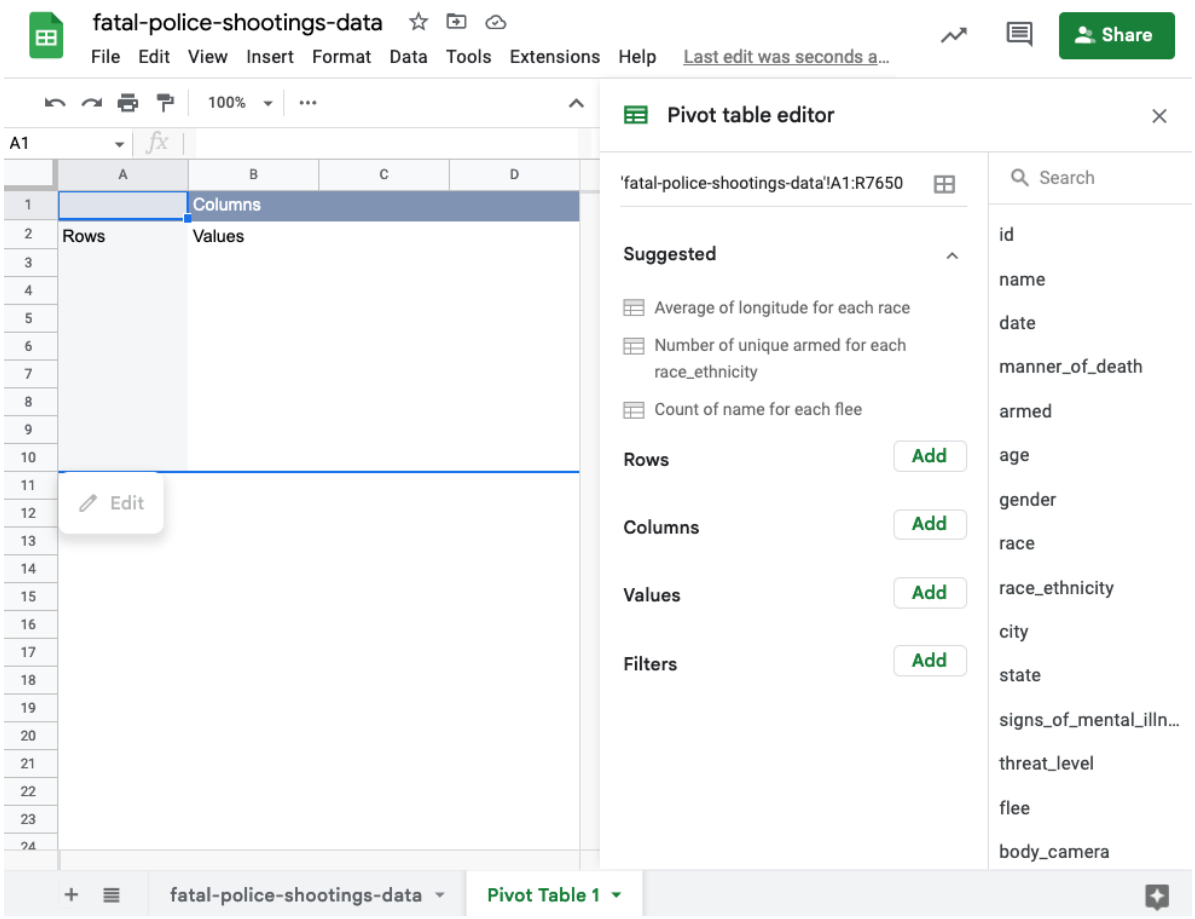
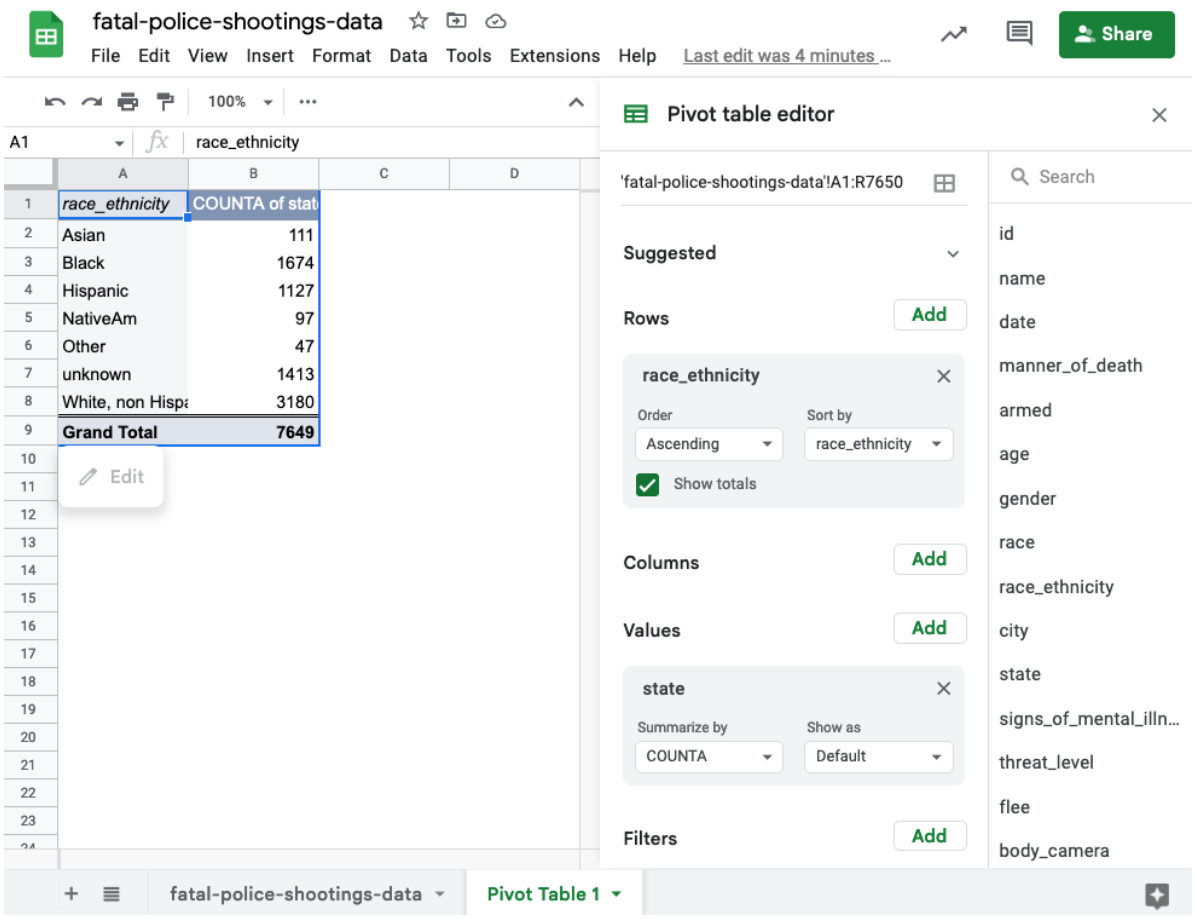


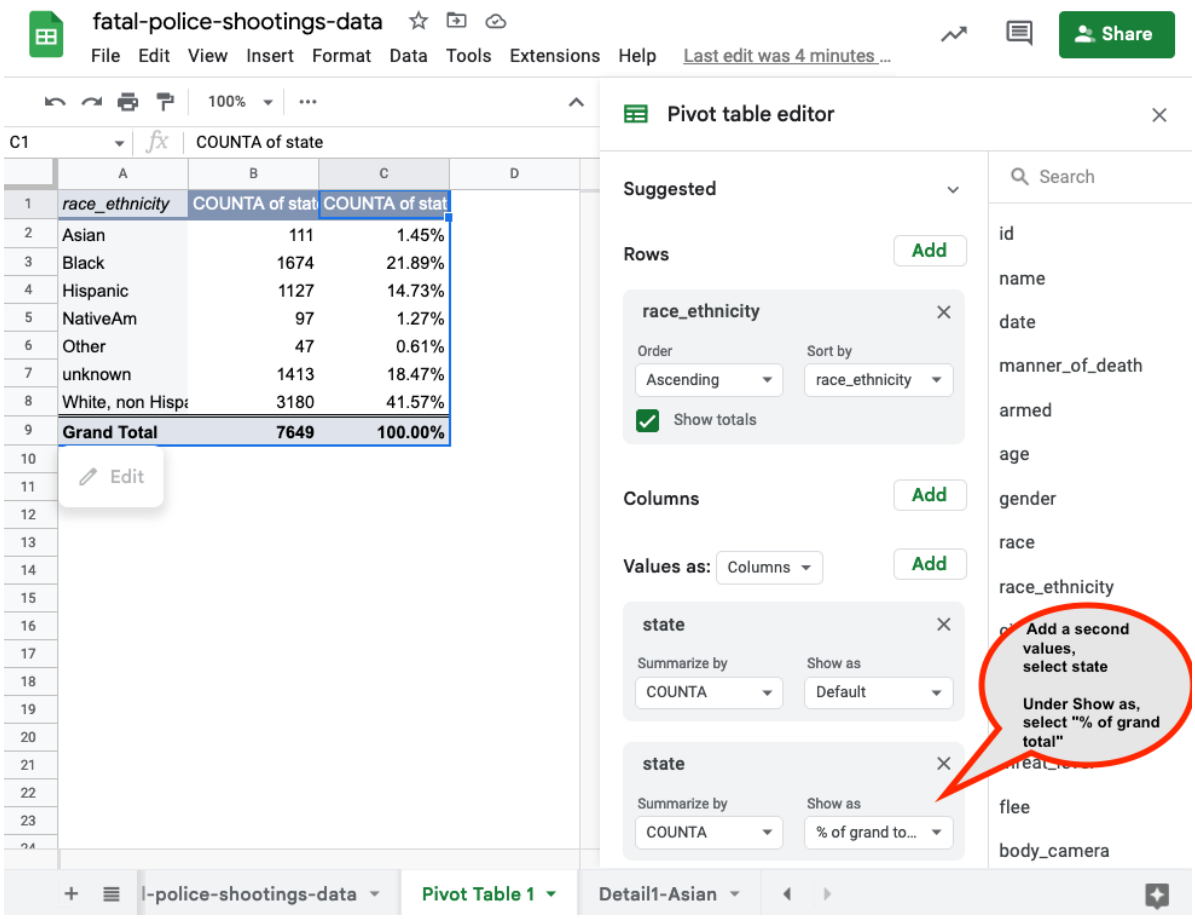
Figure 9.2: pivot menu



Percents of total

It's hard to compare raw numbers unless they're really small. Instead, we'd like to know what *percent* of fatalities by ethnicity. To get a "Percent of Column total", do the following:

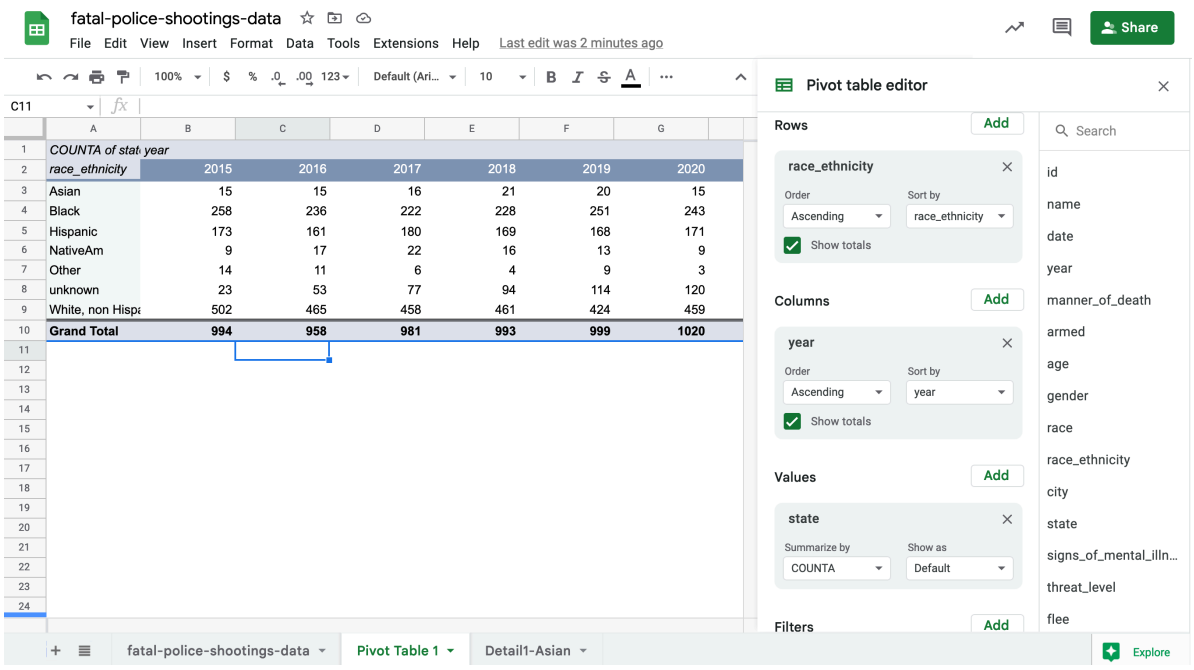
- Add a second values, select state
- Under Show as, select "% of grand total"



More variables

Suppose you'd like to see the number of fatalities by year, with the years across the top and the `ethnicity` down the sides. Add a year variable to columns

- Remove the percent of total column
- Select Columns, then year



COUNTA of state	year					
race_ethnicity	2015	2016	2017	2018	2019	2020
Asian	15	15	16	21	20	15
Black	258	236	222	228	251	243
Hispanic	173	161	180	169	168	171
NativeAm	9	17	22	16	13	9
Other	14	11	6	4	9	3
unknown	23	53	77	94	114	120
White, non Hispa	502	465	458	461	424	459
Grand Total	**994**	**958**	**981**	**993**	**999**	**1020**

Even more variables

Say you wanted to see each city's total shootings by year. Which one had the most last year, and which one had the most overall?

This is actually really hard in a pivot table, because there are cities with the same names in different states. It means you'd need to have a pivot table with TWO columns down the side, and one across the top. Here's an attempt at solving the problems:

- First, Rows, add state
- Rows, add city
- Values, add state, CountA is the default
- Columns, add year

The problems is we can't sort by the combination of city and state. But it does help answer the question on some level.

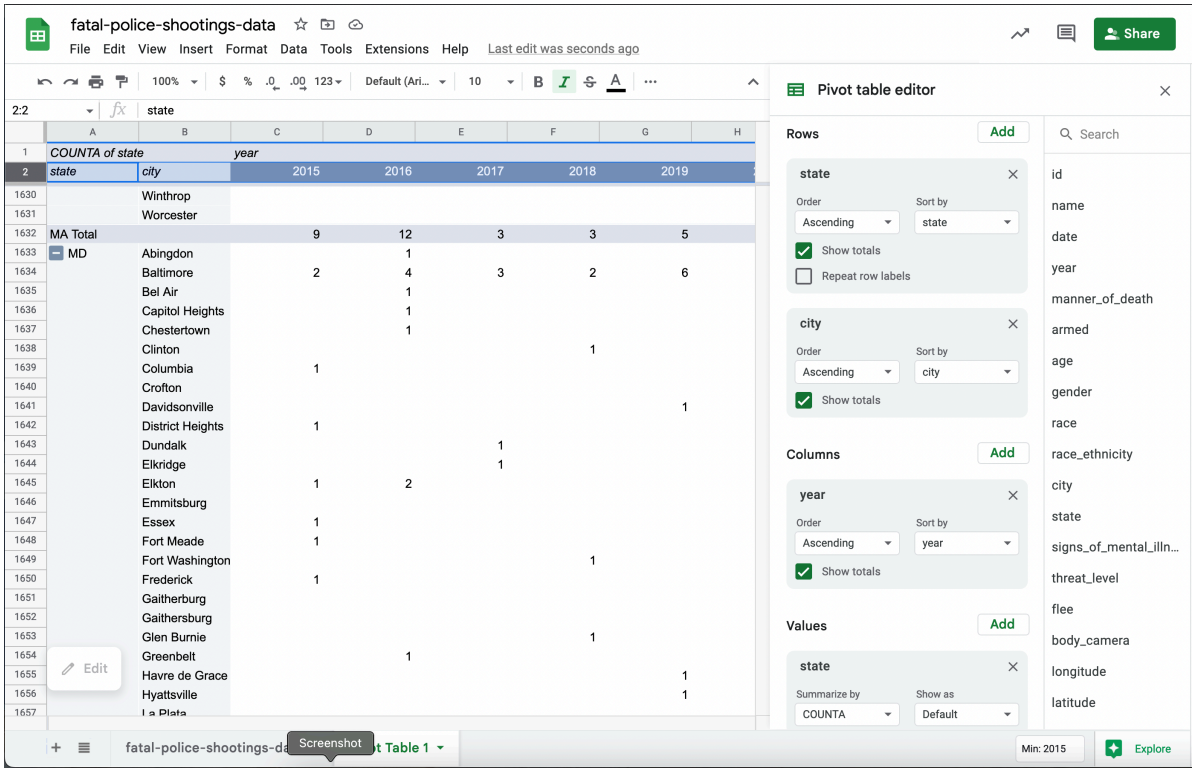


Figure 9.3: More Variables!

9.2 FAQ

I have too many columns

If you want two sets of statistics – say, number of fatalities and percent of fatalities – across the top, it can get very wide and confusing very quickly. One alternative is to change it into more of a vertical rectangle by dragging the "Values" element from the columns to the rows on the right. (This only shows up when you have two calculations being made.)

I want to sort by percents, not numbers

You can't.

Things aren't adding up

You have to be super careful about which column you use to Count things – it has to always be filled out (there can't be any blanks). Go through the filters and find one that doesnt have (Blanks) at the bottom to be sure.

Its a crazy number!

You might have dragged a numeric column into the "Values" area. Check to see if it says "Count" or "Sum". Change it to "Count" if it has something else on it, unless you wanted to add up that column.

This is so frustrating - I can't get what I want

Right? It's time to go to a programming language!

10 Cleaning data with Google Sheets

This chapter will demonstrate some of the bedrock data cleaning skills using Google Sheets, techniques that can be used in Excel and other spreadsheets. We will normalize and clean data by deleting rows, stripping whitespace, making characters lowercase or uppercase. In addition, you will learn to split text to columns, a very handy tool for splitting up dates.

Make a copy of your data before cleaning)

We will use a version of the Washington Post police shooting data to conduct these exercises.

10.0.1 Text to columns

We want to split up the date field into day, month and year. Currently, the format is 2015-01-02. Luckily, the fields all share a common separator, a hyphen, and we can ask Google Sheets to split all according to the hyphen. Other common separators are commas and spaces.

First steps when modifying data: make a backup copy! - Left click on the tab "Police Shootings to Clean" - Select duplicate - Rename "Copy of Police Shootings to Clean" to "Original Police Shootings to Clean." Do not touch this version.

Time to split text to columns. I am extra paranoid (for good reason) and so I always duplicate a date field before modifying it. Duplicate the date column (click on Column C, left click, copy, then Insert column to left, select new blank Column C and paste), save the copy as date-original.

- Select date column
- Select Data | Split text to columns
- See a dialog box: Separator. Select Custom and type in a dash - and enter. You now have the date field chopped up to year, month and day. Rename column E for month and column f for day.

10.0.2 Normalizing

Scroll down the race_ethnicity column and you will see a number of different categories for the same thing: white, White, non Hispanic and Black, African Am. To see all the variations of categorical variables, create a filter and check the different variables

This presents a big problem when you are trying to group and summarize based on these variable names. See this chart

We see white totals 44 and White, non Hispanic total 3,136. We want those to be together – the total is 3,180 – because they are the same thing. Also note that African Am totals 29 and Black totals 1,645, and we would want to combine those as well.

Let's fix it!

Before changing any data, let's work with a copy of the column. - Select race_ethnicity (Column k), left click, copy - Left click on Column K, insert column to right, paste - Rename as race_ethnicity2

Renaming variables. We will rename all "white" as "White, non Hispanic" - Filter race_ethnicity (Column K) to white - in race_ethnicity2, write "White, non Hispanic" in the first column and copy down the list

See how this process works

10.0.3 Lowercase or Uppercase character conversion

Create a filter and notice two variations on Native American: NativeAm and nativeam. You can resolve these differences easily by converting all to Upper or Lower case text using the =UPPER or =LOWER functions.

- To convert NativeAm to lower case, filter on race_ethnicity (Column K) for NativeAm.
- In race_ethnicity2 (Column L), insert a blank column, and type the function =LOWER(K67) and hit enter.

The result should be nativeam as the first entry in race_ethnicity2.

See this example

10.0.4 White space

One obnoxious feature of spreadsheet data is the invisible "white space" or hidden character or carriage returns that can impede your ability to group and summarize variables. Look at the age column. See how some numbers are flush left while most are flush right. The flush left data has hidden white space. You can fix this by clicking on individual cells and deleting the space around the number or you can do it with a function.

- Select age (Column H), left click on Column H, insert column to right, rename as age2
- In cell I2, type =TRIM(H2) and enter. Copy the formula down.

Note how all of the values have been normalized.

These are some of the basic go-to tools for data cleaning in Google Sheets, which can be adapted to Excel, R and other programming languages.

11 Getting started with R and RStudio

In this chapter

- Install R , RStudio
- The power of packages, especially the `tidyverse`
- Set up defaults for reporting with data
- Issue your first R commands
- Work in projects
- Relax!

This is probably your first introduction to coding. Don't be worried. With effort, much of what reporters do in coding can be learned in a few weeks.

Like most reporters, I learned the coding that I know (which isn't a lot) because I wanted to get a story done. In our class, we are not trying to become a programmer or social scientist. We're working on stories.

You saw during the pivot table lesson that spreadsheets have limits. We couldn't easily get the city with the most police shootings because we would have had to put both city and state into the pivot table. A median is missing from pivot tables entirely. It's easy to lose track of where you are and what you did. That's the reason to learn some coding – there is something we want to know that isn't very easy to get in other ways.

All programming languages have one thing in common: You write instructions, called algorithms, and the program executes your statements in order. It means you can do more complicated work in computer programming than in point-and-click parts of Excel. It can also scale – you can repeat your instructions millions of times, tweak it a little, and re-run the program without messing anything else up. The computer won't mind. Really.

Writing code can also be self-documenting. You can largely eliminate those painstaking Excel data diaries and replace them with documents that explain your work as you go. You'll still need to record information about your interviews and decisions, but you'll no longer have to write down every mouse click.

If you're nervous about getting started with programming, take look at the Appendix: A gentle introduction to programming and Jesse Lecy's "Learning how to Learn."

R or Python?

If you ask a data scientist or technologist which language you should learn first, you'll start a heated debate often between advocates of R, Python, Javascript , SQL, Julia and others. Ask the same question of a data journalist and the answer will be: "Choose one that is free and that your colleagues use so you can get help." For our purposes, it really doesn't matter – any of the standard languages will do.

The only rule I would make is to try to stick to your first language for a little while before deciding you want to learn a different one. It would be like trying to learn Portuguese and Spanish at the same time, when you know neither one to begin with. They're related, but very different.

Employers who hire data reporters usually don't care which programming language you know because it's relatively easy to learn another once you're comfortable with the concepts and good data journalism habits. In a few cases, such as the Associated Press, R is preferred.

11.1 Install R and RStudio

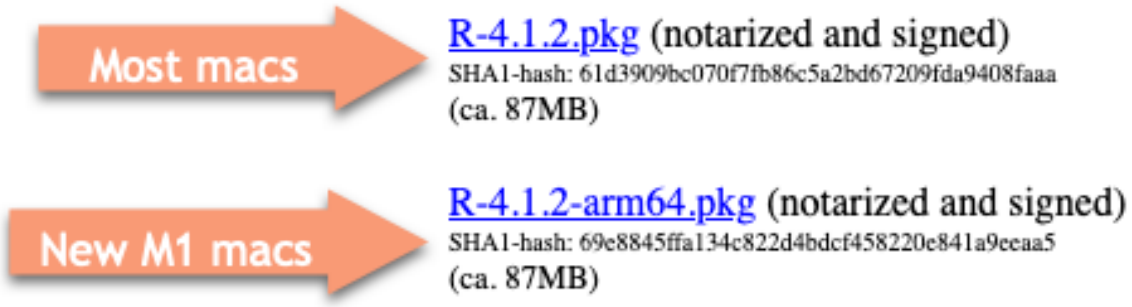
- R is the programming language itself, and has to be installed first
- RStudio is a software program that makes it easier to interact with the programming language. Install it second.
- Packages are sets of programs written by volunteers and data scientists that perform specialized jobs more easily that working with the "base" R language. A package must be installed once on your computer, then invoked to use them in a program.

Follow this interactive tutorial on installing R, RStudio and the tidyverse on your computer:

https://learnr-examples.shinyapps.io/ex-setup-r/#section-welcome .

There are two differences between the video and today:

- The tidyverse will take much longer to finish installation. It has a lot to do and often looks like it's stalled.
- There are two versions of R for Mac users: The traditional one and the one for the new M1 chip on the latest machines. Choose the one that matches your machine by checking the apple in the upper left and looking at "About this Mac". It will say "Apple M1" as the processor if you have it.



11.2 Unlocking packages and the tidyverse

The real power of R comes with packages. Packages are bundles of programs that others have found useful to extend the base R language. R is almost useless without them. There are more than 10,000 packages available for R, each doing a special job.

If you followed along with the tutorial, the last thing you did was install a "package" called the `tidyverse`. Almost everything we do from now on depends on that step.

The tidyverse packages up a whole set of other packages that are designed to work together smoothly with similar grammar and syntax. It's particularly useful for the kind of work reporters do – importing, cleaning and analyzing data that we get from others and can't control how it's structured.

We'll be working almost exclusively within the tidyverse in this course. I strongly suggest that when you Google for help, put the word "tidyverse" somewhere in your query. Otherwise, you may get answers that look inscrutable and unfamiliar.

The tidyverse is the brainchild of Hadley Wickham, a statistician from New Zealand, who famously identified **tidy data** principles. He's currently the chief data scientist for RStudio in Houston.

11.3 Set up RStudio for data reporting

Staying organized is one of the challenges of data reporting – you're constantly re-downloading and re-jiggering your analysis and it's easy to get your material separated. This setup helps

ensure that you always know where to find your work and can move it to another comptuer seamlessly.

Before you start, decide on a folder you'll use to store all of your R work. Within my Documents folder, I created a sub-folder called `data-class`. It will make this guide a little easier if you do the same thing, especially if you're not very familiar with using directories and folders.

Start up RStudio once you've made your folder. Make sure you start up RStudio (not the R language) by searching for it in Spotlight or in the Search bar in Windows. Here's what they look like:

Get to the Preferences (under the RStudio menu item on a Mac) and make sure it looks like this in the General tab:

(I've turned OFF all of the options to restore anything when you start up RStudio and set up a default working directory by browsing to the one I just created.)

Under the R Markdown options, make sure that the box called "Execute setup chunk automatically" is checked.

11.4 The screen

This is what your screen probably looks like:

The Console

The Console is where you can type commands and interact directly with the programming language. Think of it as a very powerful calculator at first. One reason to use it is to install packages.

If you followed the installation demo, you've already used the console to install one package. (Go back and do that part now if you skipped it.) Install a few more that will be useful in this course.

Copy these commands one a a time, and paste them into the Console, then hit Return/Enter to execute the command.

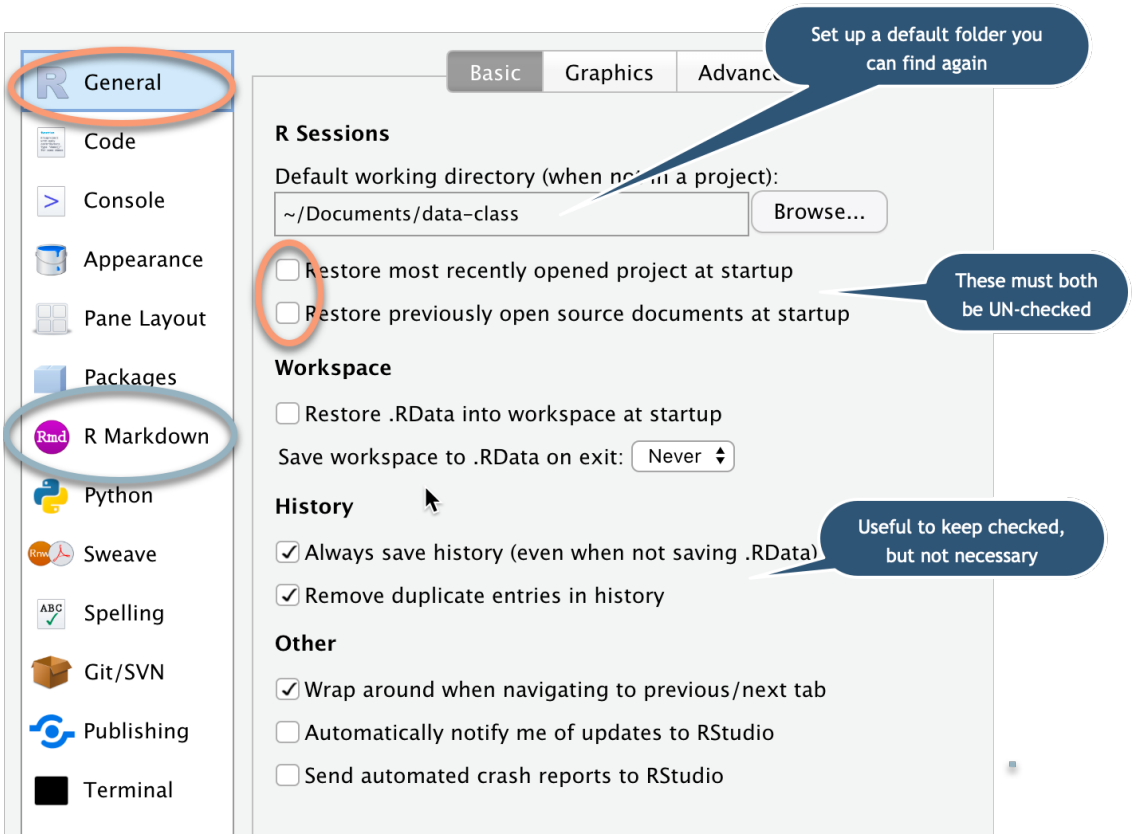


Figure 11.1: see below

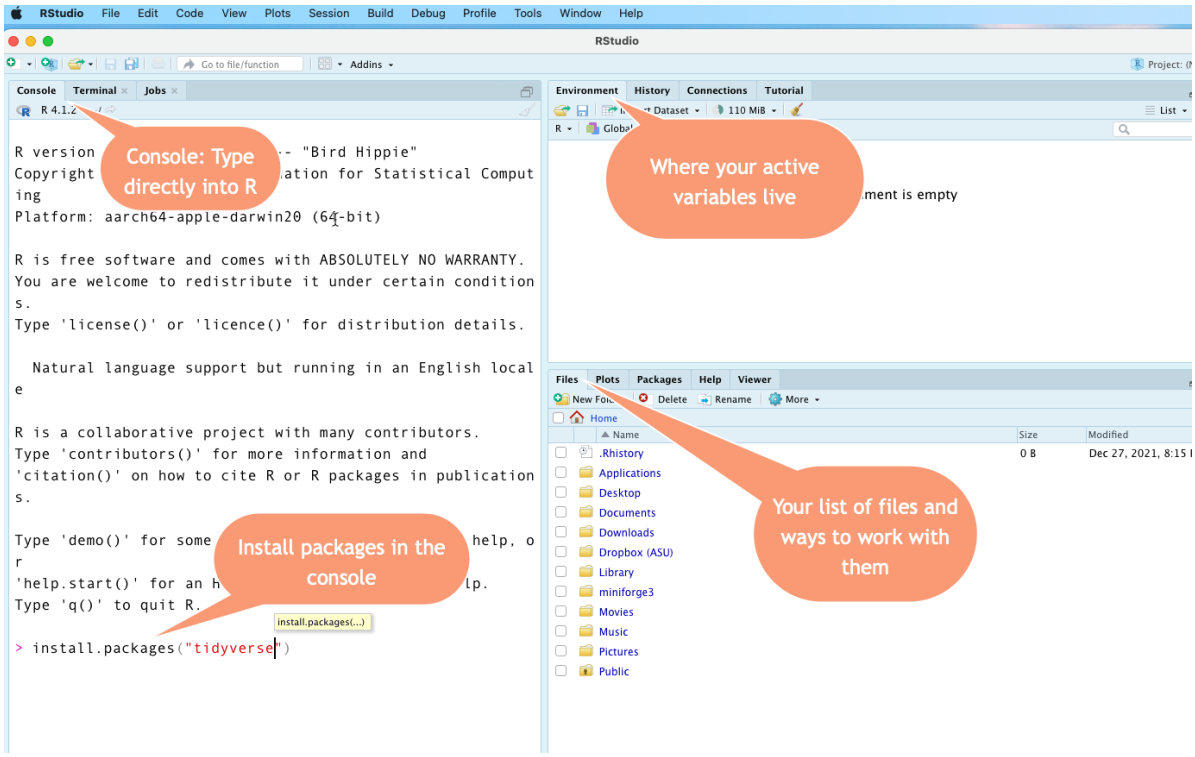


Figure 11.2: console

```
install.packages("janitor")
install.packages("rmarkdown")
install.packages("skimr")
install.packages("swirl")
```

These package names should all be in quotes. We'll be installing other packages later in this guide, but for now that is everything you need.

Files tab

We won't be using many of the tabs in the lower right, but the Files tab can help you if you're having trouble navigating your work. Under the More button, you can choose "Go to working directory", since that's where R thinks you've parked all of your work. This can be confusing in R, which is why we'll be working in "projects" that bundle up all of your work in one place.

Environment

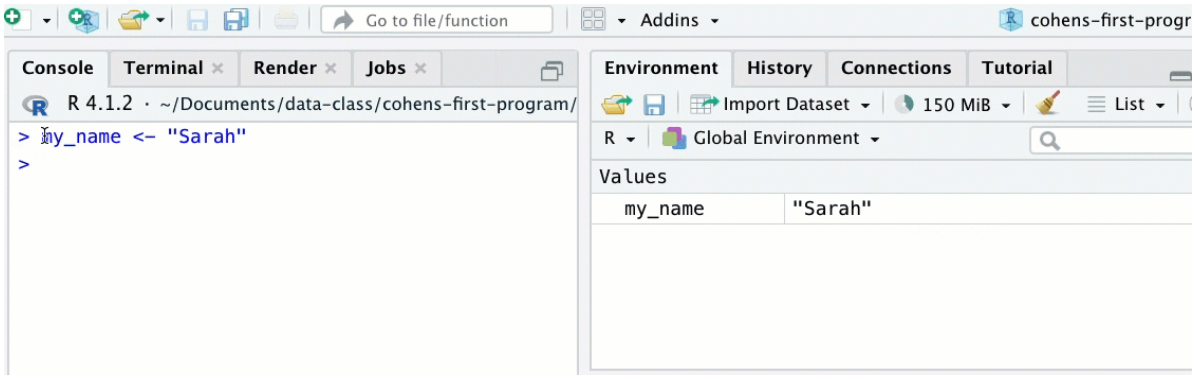
The upper right screen is the Environment, which is where your active *variables* live. A variable is a named thing. It might be a word, a list of words or numbers, or a data frame (spreadsheet). Anything that you want to use has to be listed in that environment before you can reference it. This will make more sense later.

Typing into the console

When you type this: `5+5` after the > prompt, you'll get this back after you press Return/Enter: `[1] 10`

When you type this: `"Merrill"` (with quotes) after the > prompt, you'll get this back: `[1] "Merrill"`

To create a new variable, you'll use the *assignment* operator `<-` (two characters : A less than sign and a hyphen). Here is how I would create the variable called `my_name` (lower case, no spaces). Notice how it appears in the Environment after being created. Then I can print it by typing the name of the variable instead of the letters of my name in quotes:



The console remembers your commands, but you have to type them one at a time and it will forget them when you leave for the day. That's why we're going to work in programs called R Markdown documents most of the time.

11.5 Working directory

Here's how to set your working directory so you can keep your files organized. First, you should have created a folder within the Documents folder called `data_class`. Run the following command to see if R Studio is pointing to that folder

```
getwd()
```

```
[1] "/Users/smussend/Desktop/git_repos/data_journalism_interactive_textbook/03_tutorials/qmd_
```

If it is not, you can navigate R Studio to that folder using the Files tab in the lower right corner window. Once you find your data_class folder, then select the Set Working Directory option under the More menu.

You can also set the path programmatically using setwd() which means set working directory. Just find the path directory using Finder – directions are here – and copy that link to this command.

```
setwd("/Users/YOURNAME/Documents/data_class")
```

11.6 Interactive R tutorial

One of the packages you installed earler was called `swirl`. Invoke it now by typing `library(swirl)` into the Console. You can follow the instructions from there. Don't bother going beyond the first chapter – it's more geared at other kinds of jobs than ours.

Relax by Silwia Bartyzel via Unsplash

11.7 Relax!

You're all set up and we're ready to start programming. Congratulate yourself - everything is new, nothing is intuitive and the screen is intimidating. You've come a long way.

11.8 Other resources

Sharon Machlis' Practical R for Mass Communications and Journalism has an intro to R and RStudio in chapters 2.3 through 2.6

Ben Stenhaug created a fast-paced video introducing the RStudio interface. Don't worry too much about what some of it means just yet – just see if you can get used to the different parts of the screen.

12 Loading and Analyzing Data

This tutorial, based on a 2022 presentation for the National Institute for Computer Assisted Reporting (NICAR), will show you the essential workflow for any data analysis project in R. You will learn to import data, explore it, use basic commands to sort and filter and build summary tables. You will build a basic data visualization from your work – all in R code.

You will be executing commands in this document that are contained in "chunks," which are separate from the text and contain live R code. **Click the green arrow at the right on line 6** and run the help.start() command

```
help.start()
```

In the bottom right window of R Studio, you will see a Help window that displays basic help commands for the program.

12.0.1 Install software to grab data

```
Tidyverse: Eight separate software packages to perform
data import, tidying, manipulation, visualisation, and           programming

Rio: Easy importing features
Janitor: Data cleaning
```

You should have installed tidyverse already. If not, then delete the hashtag in front of install.packages("tidyverse") and run the code chunk at line 22.

```
#install.packages("tidyverse")
install.packages("rio")
install.packages("janitor")
```

Remember, package installation usually is a one-time thing on your hard drive. But when you need to load the software libraries each time you start a script. Libraries are bits of software you will have to load each time into R to make things run.

```
library(tidyverse)
library(rio)
library(janitor)
```

Check to see what's installed by clicking on "Packages" tab in File Manager, lower right pane

12.0.2 Data

We will work with a dataset of MediaSalaries that I cleaned and modified slightly in March 2022 for this tutorial. Some of the detail has been removed so we can make calculations. This MediaSalaries sheet was a crowdsourced project involving reporters worldwide to share information about salaries and benefits. Open this file in Google Sheets

```
1) Select Salaries tab

IRE Old School: Four Corners Test!
13 Columns
1658 Rows

Numberic data in Salary, Years Experience
Mixed string data in Gender Identity / Ethnicity, Job duties
```

12.0.3 Import Data

```
We'll now load this data into R. You can load spreadsheets from the Internet as follows:
```

```
MediaBucks <- rio::import("https://docs.google.com/spreadsheets/d/1jkbQFwIdaWv8K0OAd6Wq7Zx
```

What happened? Look at the table

```
View(MediaBucks)
```

What happened?

R grabbed the spreadsheet from the folder
We told R to grab the first sheet, RealMediaSalaries2
R created a dataframe called MediaBucks
basics of R: <- is known as an "assignment operator."
It means: "Make the object named to the left equal to the output of the code to the right."

12.0.4 Explore Data

Click the green arrow code chunk to get the answers below.

How many rows?

```
nrow(MediaBucks)
```

```
[1] 1658
```

How many columns?

```
ncol(MediaBucks)
```

```
[1] 13
```

Dimensions: Gives number rows, then columns

```
dim(MediaBucks)
```

```
[1] 1658   13
```

Names of your columns

```
colnames(MediaBucks)
```

```
 [1] "TITLE"                     "COMPANY"
 [3] "Salary"                    "Salary_Details"
 [5] "Salary_Details2"           "Race"
 [7] "Gender"                    "YEARS_EXPERIENCE"
 [9] "LOCATION"                  "JOB_DUTIES"
[11] "PREV_SALARIES_TITLES"      "SALARY_original"
[13] "Gender_Ethnicity_Original"
```

OR

```
names(MediaBucks)
```

```
 [1] "TITLE"                     "COMPANY"
 [3] "Salary"                    "Salary_Details"
 [5] "Salary_Details2"           "Race"
 [7] "Gender"                    "YEARS_EXPERIENCE"
 [9] "LOCATION"                  "JOB_DUTIES"
[11] "PREV_SALARIES_TITLES"      "SALARY_original"
[13] "Gender_Ethnicity_Original"
```

Check data types

```
str(MediaBucks)
```

```
'data.frame':   1658 obs. of  13 variables:
 $ TITLE                   : chr  "Photojournalist" "Staff Writer" "Staff reporter" "Reporte
 $ COMPANY                 : chr  "College Student Newspaper" "Paxton Media" "Schurz Commun
 $ Salary                  : num  0 12 12 13.2 14.9 ...
 $ Salary_Details          : chr  "" "hour" "hour" "hour" ...
 $ Salary_Details2         : chr  "" "anhour" "-2012" "~32,000/year" ...
 $ Race                    : chr  "asian" "white" "white" "white" ...
 $ Gender                  : chr  "male" "female" "female" "female" ...
 $ YEARS_EXPERIENCE        : chr  "3" "8" "2" "3" ...
 $ LOCATION                : chr  "" "Georgia" "Central Kentucky" "Houston" ...
 $ JOB_DUTIES              : chr  "Photo" "Write and cover news stories, photograph news, wr
 $ PREV_SALARIES_TITLES    : chr  "" "" "" "" ...
 $ SALARY_original         : chr  "0" "$12 an hour" "$12/hour (2012)" "13.25/hour ~ 32,000/y
 $ Gender_Ethnicity_Original: chr  "Cis male asian" "White female" "cis white female" "Cis wh
```

Let's look at the first six rows

```
head(MediaBucks)
```

```
           TITLE                    COMPANY Salary Salary_Details
1 Photojournalist  College Student Newspaper   0.00
2    Staff Writer               Paxton Media  12.00          hour
3  Staff reporter Schurz Communications Inc.  12.00          hour
4        Reporter          Hearst Newspapers  13.25          hour
5  Staff Reporter           Ogden Newspapers  14.91          hour
6   Desk Assitant                       KTLA  15.00          hour
  Salary_Details2     Race Gender    YEARS_EXPERIENCE        LOCATION
1                    asian   male                   3
2          anhour    white female                   8         Georgia
```

```
3           -2012    white female                   2 Central Kentucky
4    ~32,000/year    white female                   3          Houston
5         perhour    white   male 3+ years experience             Utah
6                 noanswer female                   2      Los Angeles

1
2
3
4
5
6 Organize daily field drives and retrieves file videos\nUse FTP system to send/receive vide
  PREV_SALARIES_TITLES           SALARY_original Gender_Ethnicity_Original
1                                              0            Cis male asian
2                                  $12 an hour              White female
3                                $12/hour (2012)          cis white female
4                     13.25/hour ~ 32,000/year           Cis white female
5                                $14.91 per hour            cis white male
6                                       $15/hour                cis female
```

Here is a quick way to view the range of your data

```
summary(MediaBucks$Salary)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      0   42000   60000   64194   78000  770000       4
```

Size and scope

```
sum(MediaBucks$Salary, na.rm=TRUE)
```

```
[1] 106177432
```

$106 million! for 1,658 journalists

Context: NYT earnings in 2020 = $100 m Facebook profit for one day: $114 million (Q42021=$10.3B)

average

```
mean(MediaBucks$Salary, na.rm=TRUE)
```

```
[1] 64194.34
```

Distribution

```
quantile(MediaBucks$Salary, c(0.1, 0.2, 0.3, 0.4,0.5, 0.6, 0.7, 0.8, 0.9), na.rm=TRUE)
```

```
  10%   20%   30%   40%   50%   60%   70%   80%   90%
28720 39000 45000 52000 60000 65000 75000 84000 101700
```

```
quantile(MediaBucks$Salary, c(0.25, 0.50, 0.75, 0.9, 0.99), na.rm=TRUE)
```

```
  25%   50%   75%   90%   99%
42000 60000 78000 101700 200000
```

12.0.5 Navigation Tips

```
Shortcut Commands
Tab - Autocomplete
In Console Window (lower left)
--Control (or Command) + UP arrow - last lines run
Control (or Command) + Enter - Runs current or selected lines of code in the top left box of
Shift + Control (or Command) +P - Reruns previous region code
```

12.0.6 Dplyr

```
dplyr has many tools for data analysis
select Choose which columns to include
filter Filter the data
arrange Sort the data, by size for continuous variables, by date, or alphabetically
group_by Group the data by a categorical variable
```

Build a simple summary table by Gender

```
MediaBucks %>%
  select(Gender, Salary) %>%
  group_by(Gender) %>%
  summarize(Total = sum(Salary, na.rm=TRUE))
```

```
# A tibble: 4 x 2
  Gender       Total
  <chr>        <dbl>
1 female   63198034.
2 male     35201242.
3 na        3061718.
4 noanswer  4716438.
```

What is the sample size?

```
MediaBucks %>%
  count(Gender) %>%
  arrange(desc(n))
```

```
    Gender    n
1   female 1025
2     male  503
3 noanswer   71
4       na   59
```

Better idea: Check Averages!

Build a simple summary table by Gender

```
MediaBucks %>%
  select(Gender, Salary) %>%
  group_by(Gender) %>%
  summarize(Avg_Salary = mean(Salary, na.rm=TRUE))
```

```
# A tibble: 4 x 2
  Gender   Avg_Salary
  <chr>         <dbl>
1 female       61717.
2 male         69983.
3 na           52788.
4 noanswer     68354.
```

Quick filter out hourly workers

```
MediaSalary <- MediaBucks %>%
  filter(Salary >= 1000)
```

Just give me a list of the top 10 salaries and companies: use slice_max. slice_max and slice_min are features in the Dplyr library (part of Tidyverse) that produce quick summary tables. See what else you can do with the slice commands.

```
MediaBucks %>%
  select(COMPANY, Salary) %>%
  slice_max(Salary, n = 10)
```

```
                COMPANY Salary
1     Tribune Publishing 770000
2              The Onion 550008
3  G/O Media (The Onion) 500000
4  G/O Media (The Onion) 500000
5               VoxMedia 400000
6             ProPublica 395000
7    Digital_First_Media 375000
8          The Intercept 368249
9           NewYorkTimes 350000
10             LBI Media 291200
```

Questions:

```
1: View the range of your data
2: Number of rows
3: Number of rows cut with filter
```

12.0.7 Find Your News Organization

Filter

```
WSJ <- subset(MediaBucks, COMPANY=="WallStreetJournal")
```

```
summary(WSJ$Salary)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     38   41000   51100   64275   75750  236000
```

Using Wildcards

```
Journal <- subset(MediaBucks, grepl("?Journal", COMPANY))
```

```
Bloom <- subset(MediaBucks, grepl("?Bloomberg", COMPANY))
```

More Tables

Build a table with several companies of your choice

```
BigBoys <- filter(MediaSalary, COMPANY %in% c("NewYorkTimes", "WallStreetJournal", "Bloomb
```

Table with just reporter salaries

```
Reporters <- subset(MediaBucks, grepl("?reporter", TITLE))
summary(Reporters$Salary)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     12   35000   50625   57077   67250  230504
```

Questions:

```
1: Who is making $230,504 as a reporter???
2: Make a table for editors, figure out medians.
3: Find highest paid editor. Resent them.
4: Make a table for any position involving data
```

Table with Black reporters at Wall Street Journal

```
WSJ_Black <- MediaBucks %>% filter(Race =="black", COMPANY == "WallStreetJournal")
```

Build a simple summary table by Race

```
Race <- MediaBucks %>%
  select(Race, Salary) %>%
  group_by(Race) %>%
  summarize(Avg_Salary = mean(Salary, na.rm=TRUE)) %>%
  arrange(desc(Avg_Salary))
Race
```

```
# A tibble: 13 x 2
   Race       Avg_Salary
   <chr>           <dbl>
 1 chinese        76167.
 2 australian     75000
 3 african        73485.
 4 hispanic       70481.
 5 black          69371.
 6 poc            68061.
 7 asian          66427.
 8 noanswer       64925.
 9 latina         64806.
10 white          63025.
11 mixed          55756.
12 mideastern     51833.
13 native         50000
```

Wait! What are the totals by race?

```
MediaBucks %>%
  count(Race) %>%
  arrange(desc(n))
```

```
         Race    n
1       white 1094
2    noanswer  192
3         poc  115
4       asian   77
5       black   57
6    hispanic   53
7      latina   23
8       mixed   22
9     african   13
10 mideastern    7
11    chinese    3
12 australian    1
13     native    1
```

Advanced: Build a summary table and count by race

```
MediaBucks %>%
  select(Race, Salary) %>%
 group_by(Race) %>%
  summarize(Total=n(),
            Avg = mean(Salary, na.rm=TRUE)) %>%
  arrange(desc(Total))
```

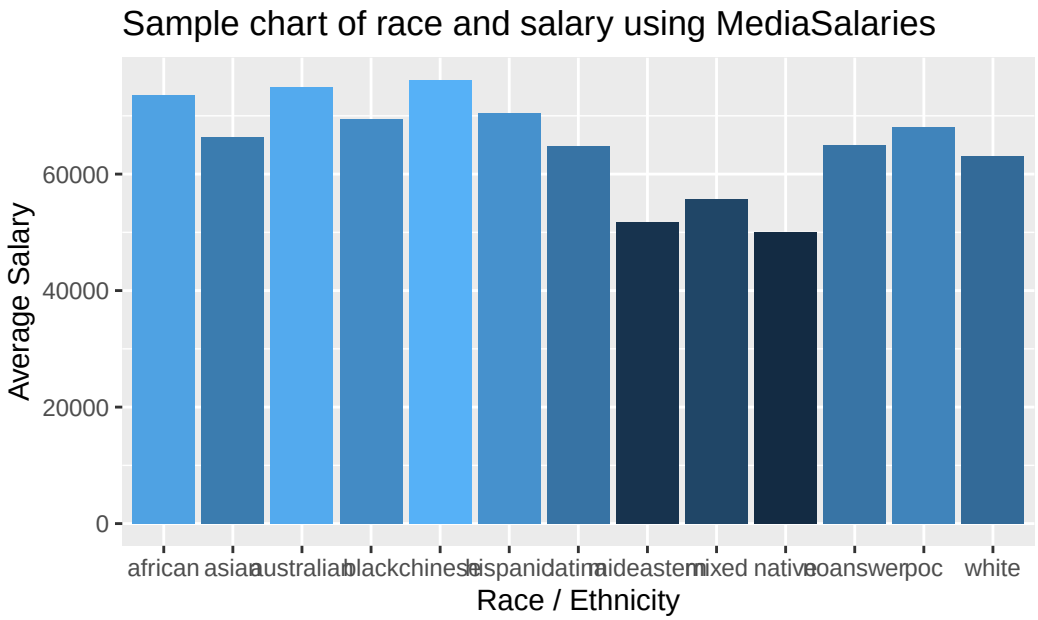
```
# A tibble: 13 x 3
   Race       Total    Avg
   <chr>      <int>  <dbl>
 1 white       1094 63025.
 2 noanswer     192 64925.
 3 poc          115 68061.
 4 asian         77 66427.
 5 black         57 69371.
 6 hispanic      53 70481.
 7 latina        23 64806.
 8 mixed         22 55756.
 9 african       13 73485.
10 mideastern     7 51833.
11 chinese        3 76167.
12 australian     1 75000
13 native         1 50000
```

```
#details: https://stackoverflow.com/questions/36183601/average-and-count-with-aggregation-
```

12.0.8 Visualize

Let's make a simple chart of our salaries by race.

```
Race %>%
ggplot(aes(x = Race, y = Avg_Salary, fill = Avg_Salary)) +
  geom_col(position = "dodge") +
  theme(legend.position = "none") +
  labs(title = "Sample chart of race and salary using MediaSalaries",
       caption = "1658 records, sample data. Graphic by Rob Wells, 8/11/2022",
       y="Average Salary",
       x="Race / Ethnicity")
```



This is a basic chart using ggplot. To break down the code: - These lines: Race %>% ggplot(aes(x = Race, y = Avg_Salary, fill = Avg_Salary)) + – Uses the Race table, calls the ggplot program, assigns the x axis to Race, y axis to Avg_Salary and fills the color according to Avg_Salary - These lines: geom_col(position = "dodge") + theme(legend.position = "none") + – creates a chart of columns, and removes a legend box. - These lines: labs(title = Assign the headline and captions.

12.0.9 What You Have Learned So Far

```
How to navigate in R studio
How to install libraries and packages
How to import a .xlsx file into R
How to obtain summary statistics (summary)
How to build basic tables from a dataset
How to conduct filter queries from a dataset
```

12.0.10 Questions

```
1: Build a table for NewYorkTimes employees, and determine median salary of NewYorkTimes empl
```

```
#your answer here
```

```
2: Identify title, gender and race of the highest paid position at NYT
```

```
#your answer here
```

```
3: Search for Bloomberg,  check median salary, compare to NYT results above.
```

```
#your answer here
```

12.0.11 Tutorials

Excellent book by Sharon Machlis https://www.routledge.com/Practical-R-for-Mass-Communication-and-Journalism/Machlis/p/book/9781138726918

First five chapters are free on her website. My recommendation: buy the book. https://www.machlis.com/R4Journalists/

All Cheat Sheets https://www.rstudio.com/resources/cheatsheets/

MaryJo Webster tutorials http://mjwebster.github.io/DataJ/R_tutorials/opiate_deaths.nb.html https://github.com/mjwebster/R_tutorials/blob/master/Using_R.Rmd

Aldhous' R tutorial http://paldhous.github.io/NICAR/2018/r-analysis.html

Ron Campbell Lecture https://github.com/roncampbell/NICAR2018/blob/master/Intro%20to%20R.md

Excellent Tutorial Spelling out Excel and Comparable Commands in R https://trendct.org/2015/06/12/r-for-beginners-how-to-transition-from-excel-to-r/ https://docs.google.com/presentation/d/1O0eFLypJLP-PAC63Ghq2QURAnhFo6Dxc7nGt4y_l90s/edit#slide=id.g1bc441664e_0_59

Andrew Ba Tran first Data Analysis Steps Using R https://docs.google.com/presentation/d/1O0eFLypJLP-PAC63Ghq2QURAnhFo6Dxc7nGt4y_l90s/edit#slide=id.p

Charts https://www.rdocumentation.org/packages/ggplot2/versions/1.0.1/topics/geom_bar http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/

Base R Cheat Sheet https://www.povertyactionlab.org/sites/default/files/r-cheat-sheet.pdf

13 Using GitHub

GitHub is a platform for managing and storing files, data and code built atop Git, a popular open source version control software. GitHub accounts are free and it's easy to get started. The one prerequisite is that you have Git installed on your local computer. There are installers for Mac, Windows and Linux.

13.1 How It Works

Version control is based on the ideas that you want to keep track of changes you make to a collection of files and that multiple people can work together without getting in each other's way or having to do things in a set order. For individual users, it's great for making sure that you always have your work.

GitHub users work in what are known as repositories on their local computers and also *push* changes to a remote repository located on GitHub. That remote repository is key: if you lose your computer, you can fetch a version of your files from GitHub. If you want to work with someone else on the same files, you can each have a local copy, push changes to GitHub and then pull each others' changes back to your local computers.

So, like Microsoft Word's track changes but with a remote backup and multiple editors.

13.2 Getting Started

After installing Git and signing up for a GitHub account, download and install GitHub Desktop. It will have you sign into your GitHub account and then you'll have access to any existing repositories. If you don't have any, that's fine! You can make one locally.

GitHub has good documentation for working in the Desktop app, and while the emphasis in this book will be on using GitHub for version control, it also supports recording issues (read: problems or questions) with your files, contributing to projects that aren't yours and more.

13.3 Video overview

This YouTube video from Coder Coder provides a nice overview of Git, GitHub and GitHub Desktop.

13.4 Advanced Use

Although our focus is on the GitHub Desktop app, you can use Git and GitHub from your computer's command line interface, and GitHub has a purpose-built command line client, too. GitHub can also serve as a publishing platform for many types of files, and entire websites are hosted on GitHub Pages.

14 Mutuating, Aggregates, Filters

In this chapter, you will learn additional core data skills that allow you to filter, summarize and append new calculations to datasets. The skills in this chapter, once mastered, will transform your ability to analyze data in ways that are difficult in spreadsheets.

14.0.1 R Libraries Background

R is a statistical programming language that is purpose built for data analysis.

There is a basic R program, which geeks call Base R, and it does a lot. We will be bringing in additional software programs known as libraries that do things to make R really operate better and easier.

The two libraries we are going to need for this assignment are `readr` and `dplyr`. The library `readr` reads different types of data in. For this assignment, we're going to read in csv data or Comma Separated Values data. That's data that has a comma between each column of data. Then we're going to use `dplyr` to analyze it.

To use a library, you need to import it. Put all your library steps at the top of your notebooks. Both of these libraries are contained in the Tidyverse suite of eight essential software packages. So just load tidyverse and dplyr and readr are ready to go.

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

14.1 Importing data

The first thing we need to do is get some data to work with. We do that by reading it in. In our case, we're going to read a datatable from an "CSV" file, a stripped down version of a spreadsheet you might open in a program like Google Sheets, in which each column is separated by a comma.

So step 1 is to import the data. The code to import the data looks like this:

```
baltcity_income<- read_csv("assets/data/baltcity_income_clean.csv") %>%
as.data.frame()
```

Let's unpack that.

The first part – **baltcity_income** – is the name of a variable.

A **variable** is just a name that we'll use to refer to some more complex thing. In this case, the more complex thing is the data we're importing into R that will be stored as a **dataframe**, which is one way R stores data.

We can call this variable whatever we want. The variable name doesn't matter, technically. We could use any word. Generally we want variable names to be descriptive, hence **baltcity_income**. It's good to keep variable names lower case and one word but two or more words need to be connected by an underscore. You can't start a variable with a number.

The `<-` is the **variable assignment operator**. It's how we know we're assigning something to a word. Think of the arrow as saying "Take everything on the right of this arrow and stuff it into the thing on the left."

read_csv() is a function, one that only works when we've loaded the tidyverse. A **function** is a little bit of computer code that takes in information and follows a series of pre-determined steps and spits it back out. A recipe to make pizza is a kind of function. We might call it **make_pizza()**.

The function does one thing. It takes a preset collection of ingredients – flour, water, oil, cheese, tomato, salt – and passes them through each step outlined in a recipe, in order. Things like: mix flour and water and oil, knead, let it sit, roll it out, put tomato sauce and cheese on it, bake it in an oven, then take it out.

The output of our **make pizza()** function is a finished pie.

We'll make use of a lot of pre-written functions from the tidyverse and other packages, and even write some of our own. Back to this line of code:

```
baltcity_income<- read_csv("assets/data/baltcity_income_clean.csv") %>%
as.data.frame()
```

Inside of the **read_csv()** function, we've put the name of the file we want to load. Things we put inside of function, to customize what the function does, are called **arguments**. And

lastly, we added a **pipe operator** %>% (shift + cntl + M) that adds another command to turn the imported data into a data frame with **as.data.frame()**. The pipe operator - %>% - basically tells R to "and then do this."

Here is the entire command in a code chunk. Run it by clicking the green arrow to the right below.

```
baltcity_income <- read_csv("assets/data/baltcity_income_clean.csv")
```

In this data set, each row represents a Census district, and each column represents a feature of that district: its location, the median household income in 2010, 2016, 2020, the neighborhood identifier and geographic coordinators.

After loading the data, it's a good idea to get a sense of its shape. What does it look like? There are several ways we can examine it.

By looking in the R Studio environment window, we can see the number of rows (called "obs.", which is short for observations), and the number of columns (called variables). We can double click on the dataframe name in the environment window, and explore it like a spreadsheet.

There are several useful functions for getting a sense of the dataset right in our markdown document.

If we run `glimpse(baltcity_income)`, it will give us a list of the columns, the data type for each column and and the first few values for each column.

```
glimpse(baltcity_income)
```

```
Rows: 200
Columns: 6
$ Neighborhood <chr> "Canton", "Patterson Park North & East", "Canton", "Canto~
$ x2010        <dbl> 75938, 58409, 77841, 70313, 86157, 60439, 47131, 59236, 2~
$ x2016        <dbl> 100985, 96171, 116875, 97153, 85986, 79063, 64643, 100778~
$ x2020        <dbl> 128839, 130357, 151389, 114946, 98194, 95536, 86125, 1018~
$ Census       <dbl> 101.00, 102.00, 103.00, 104.00, 105.00, 201.00, 202.00, 2~
$ GEOID        <dbl> 24510010100, 24510010200, 24510010300, 24510010400, 24510~
```

If we type `head(baltcity_income)`, it will print out the columns and the first six rows of data.

```
head(baltcity_income)
```

```
# A tibble: 6 x 6
  Neighborhood                x2010  x2016  x2020 Census       GEOID
  <chr>                       <dbl>  <dbl>  <dbl>  <dbl>       <dbl>
1 Canton                      75938 100985 128839    101 24510010100
2 Patterson Park North & East 58409  96171 130357    102 24510010200
3 Canton                      77841 116875 151389    103 24510010300
4 Canton                      70313  97153 114946    104 24510010400
5 Fells Point                 86157  85986  98194    105 24510010500
6 Fells Point                 60439  79063  95536    201 24510020100
```

We can also click on the data name in the R Studio environment window to explore it interactively.

14.1.1 Group by and count

There is some overlap among the Census tracts, which typically are groups of betweem 1,200 to 8,000 people, and neighborhoods, which can be much larger. Let's figure out how many neighborhoods are represented in this dataset.

`dplyr` has a group by function that compiles things together and then produces simple summaries by counting things, or averaging them together. It's a good place to start.

The first step of every analysis starts with the data being used. Then we apply functions to the data.

In our case, the pattern that you'll use many, many times is: `data %>% group_by(COLUMN NAME) %>% summarize(VARIABLE NAME = AGGREGATE FUNCTION(COLUMN NAME))`

In our dataset, the column with neighborhood identifier is called "Neighborhood." Neighborhoods overlap with Census tracts.

Here's the code to count the number of census tracts in each neighborhood:

```
baltcity_income %>%
  group_by(Neighborhood) %>%
  summarise(
    count_tracts = n()
  )
```

```
# A tibble: 56 x 2
   Neighborhood                      count_tracts
   <chr>                                    <int>
 1 Allendale/Irvington/S. Hilton                6
```

```
 2 Beechfield/Ten Hills/West Hills              3
 3 Belair-Edison                                4
 4 Brooklyn/Curtis Bay/Hawkins Point            4
 5 Canton                                       3
 6 Cedonia/Frankford                            5
 7 Cherry Hill                                  3
 8 Chinquapin Park/Belvedere                    2
 9 Claremont/Armistead                          4
10 Clifton-Berea                                5
# ... with 46 more rows
```

So let's walk through that.

We start with our dataset – `baltcity_incomes` – and then we tell it to group the data by a given field in the data. In this case, we wanted to group together all the counties, signified by the field name `Neighborhood`, which you could get from using the glimpse() function. After we group the data, we need to count them up.

In dplyr, we use the `summarize()` function, which can do alot more than just count things.

Inside the parentheses in summarize, we set up the summaries we want. In this case, we just want a count of the number of loans for each county grouping. The line of code `count_tracts = n(),` says create a new field, called `count_tracts` and set it equal to `n()`. `n()` is a function that counts the number of rows or records in each group. Why the letter n? The letter n is a common symbol used to denote a count of something.

When we run that, we get a list of counties with a count next to them. But it's not in any order.

So we'll add another "and then do this" symbol – %>% – and use a new function called `arrange()`. Arrange does what you think it does – it arranges data in order. By default, it's in ascending order – smallest to largest. But if we want to know the county with the most loans, we need to sort it in descending order. That looks like this:

```
baltcity_income %>%
  group_by(Neighborhood) %>%
  summarise(
    count_tracts = n()
  ) %>%
  arrange(desc(count_tracts))
```

```
# A tibble: 56 x 2
   Neighborhood                          count_tracts
   <chr>                                        <int>
```

```
 1 Southwest Baltimore                          8
 2 Allendale/Irvington/S. Hilton                6
 3 Medfield/Hampden/Woodberry/Remington         6
 4 Sandtown-Winchester/Harlem Park              6
 5 Cedonia/Frankford                            5
 6 Clifton-Berea                                5
 7 Greater Charles Village/Barclay              5
 8 Greater Rosemont                             5
 9 Harford/Echodale                             5
10 Inner Harbor/Federal Hill                    5
# ... with 46 more rows
```

The Census data contains a column detailing the neighborhood. It has associated the Census tracts to neighborhood names. This dataset may have several neighborhood values since Census tracts are a smaller unit of measurement.

Southwest Baltimore neighborhood is spread out over 8 census tracts, more than any other neighborhood.

Here's the code to determine the count the number of census tracts in each neighborhood:

```
baltcity_income %>%
   summarise(
    count_tracts = n()
  )
```

```
# A tibble: 1 x 1
  count_tracts
         <int>
1          200
```

14.2 Interviewing Your Data: Min, Max, Mean, Medians

What is the typical median income? What about the highest and lowest median incomes in the city? For that, we can use the `min()` and `max()` functions.

```
baltcity_income %>%
  select(Neighborhood, x2010, x2016, x2020, Census) %>%
  summarise(
    count_tracts = n(),
    x2020_median = median(x2020, na.rm=TRUE),
```

```
    min_2020 = min(x2020, na.rm=TRUE),
    max_2020 = max(x2020, na.rm=TRUE)
  )
```

```
# A tibble: 1 x 4
  count_tracts x2020_median min_2020 max_2020
         <int>        <dbl>    <dbl>    <dbl>
1          200        49875    13559   199531
```

Here we see the typical median household income is `$49,875` by census tract for Baltimore City in 2020 (see result for x2020 median). The lowest median income was `$13,559` and the highest was `$199,531`. From another Census analysis, we know hat citywide, the median household income was `$52,164` for 2016-2020. The **`na.rm=TRUE`** argument lets R knock out any empty rows from the calculation.

Here's another quick way to determine the distribution of a particular column of data

```
summary(baltcity_income$x2020)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  13559   35702   49875   56311   64372  199531       5
```

This tells us the minimum, maximum, median, average (mean), and the first and third quartile, as well as rows with no values.

14.2.1 Filters: Extracting Needles from Haystacks

Where are these rich and poor places? Let's filter for the lowest value, $13,559, and find out where it is

```
baltcity_income %>%
  #temp code - remove later
  as.data.frame() %>%
  select(Neighborhood, x2020) %>%
  filter(x2020 ==13559)
```

```
         Neighborhood x2020
1 Upton/Druid Heights 13559
```

It is part of the Upton/Druid Heights neighborhood in West Baltimore.

We can stack filters using the Or connector: | It's above the enter key on a Mac keyboard

```
baltcity_income %>%
  select(Neighborhood, x2020) %>%
  filter((x2020 ==13559) | (x2020==199531))
```

```
# A tibble: 2 x 2
  Neighborhood                       x2020
  <chr>                              <dbl>
1 Upton/Druid Heights                13559
2 North Baltimore/Guilford/Homeland 199531
```

Now we know in one report the wealthiest neighborhood, North Baltimore/Guilford/Homeland, and the poorest, Upton/Druid Heights.

Read this for more details about logical operators.

Let's filter for the wealthy neighborhoods, all above $100,000

```
baltcity_income %>%
  select(Neighborhood, x2020) %>%
  filter(x2020 > 100000) %>%
  arrange(desc(x2020))
```

```
# A tibble: 17 x 2
   Neighborhood                       x2020
   <chr>                              <dbl>
 1 North Baltimore/Guilford/Homeland 199531
 2 North Baltimore/Guilford/Homeland 195353
 3 Greater Roland Park/Poplar Hill   155605
 4 South Baltimore                   151659
 5 Canton                            151389
 6 Greater Roland Park/Poplar Hill   151146
 7 Inner Harbor/Federal Hill         133333
 8 Highlandtown                      130769
 9 Patterson Park North & East       130357
10 Canton                            128839
11 South Baltimore                   121685
12 Inner Harbor/Federal Hill         120729
13 Canton                            114946
```

```
14 Mount Washington/Coldspring       109688
15 Highlandtown                      107438
16 Inner Harbor/Federal Hill         107344
17 Fells Point                       101815
```

Using the summarise function, we can figure out an average value on a column. In this case, we're going to average all of the median income values by census tract.

```
baltcity_income %>%
  select(Neighborhood, x2020, Census) %>%
    summarise(
    count_tracts = n(),
    x2020_avg = mean(x2020, na.rm=TRUE))
```

```
# A tibble: 1 x 2
  count_tracts x2020_avg
         <int>     <dbl>
1          200    56311.
```

In the example above, we created a new summary value called x2020_avg that holds the result of the math, the average of the entire x2020 column of median incomes.

14.2.2 Other summarization methods: mean, median, min and max

Here's another trick, pulling out the minimum and maximum values

```
baltcity_income %>%
  select(Neighborhood, x2020, Census) %>%
    summarise(
    count_tracts = n(),
    min_2020 = min(x2020, na.rm=TRUE),
    max_2020 = max(x2020, na.rm=TRUE))
```

```
# A tibble: 1 x 3
  count_tracts min_2020 max_2020
         <int>    <dbl>    <dbl>
1          200    13559   199531
```

To kick it up a notch, here's the same idea but with averages and medians for the three years in our data: 2010, 2016, 2020.

```
baltcity_income %>%
  select(Neighborhood, x2010, x2016, x2020, Census) %>%
    summarise(
    count_tracts = n(),
    x2020_median = median(x2020, na.rm=TRUE),
    x2020_avg = mean(x2020, na.rm=TRUE),
    x2016_median = median(x2016, na.rm=TRUE),
    x2016_avg = mean(x2016, na.rm=TRUE),
    x2010_median = median(x2010, na.rm=TRUE),
    x2010_avg = mean(x2010, na.rm=TRUE))
```

```
# A tibble: 1 x 7
  count_tracts x2020_median x2020_avg x2016_median x2016_avg x2010_med~1 x2010~2
         <int>        <dbl>     <dbl>        <dbl>     <dbl>       <dbl>   <dbl>
1          200        49875    56311.        39583    46744.      37080.  41923.
# ... with abbreviated variable names 1: x2010_median, 2: x2010_avg
```

Use the right diamond at `x2016_median` to see columns 5-7.

14.2.3 Using sum

There's much more we can to summarize each group. Let's pull in another dataset and summarize by group.

```
#loading 2020 and 2010 Baltimore City population by race
baltcity_race <- read_csv("assets/data/baltcity_race_8_13.csv") %>%
  as.data.frame()
```

Let's say we wanted to know the total population by white people in Baltimore? For that, we could use the `sum()` function to add up all of the population in the column "x2020_white". We put the column we want to total – "amount" – inside the sum() function `sum(amount)`. Note that we can simply add a new summarize function here, keeping our count_loans field in our output table.

This abbreviated slice of Census data contains columns detailing the population by race in Census tracts. There is the x2020_total which provides the full population, then x2020_white, x2020_black, x2020_hispanic. We omitted Asians and Pacific islanders and people identifying with more than one race for simplicity in this example.

Here we can select a race variable and summarize it.

```
baltcity_race %>%
  select(x2020_white, x2020_black) %>%
  summarize(
    white_total = sum(x2020_white, na.rm = TRUE),
    black_total = sum(x2020_black, na.rm = TRUE)
  )
```

```
  white_total black_total
1      178996      375002
```

Pre-Lab Question: We know the median income for Baltimore City (I just told you a few paragraphs ago). Construct a filter for all census tracts below the citywide median household income for 2020. Count them. What percentage of the city's census tracts are below the median? Put that in code too. Draft a tweet with your findings.

Answer this question in English: Write in Elms

```
`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJjaGFwdGVyIiwiYm9va
```
```

15 Mutating data

Often the data will prompt questions that lack immediate answers. We're continuing to work with the Baltimore median income data and we want to know how incomes changed over time by census tract and the percentage change.

To do that in R, we can use `dplyr` and `mutate` to calculate new metrics in a new field using existing fields of data. That's the essence of `mutate` - using the data you have to answer a new question.

So first we'll import the tidyverse so we can work with it.

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

We'll import a dataset of median household income in Baltimore for the 2010, 2016 and 2020 surveys. The data is in the data folder in this chapter's pre-lab directory. We'll use this to explore ways to create new information from existing data.

```
baltcity_income <- read_csv("assets/data/baltcity_income_clean.csv") %>%
  as.data.frame()

#working on this with your laptop, uncomment and use this code below
#baltcity_income<- read_csv("baltcity_income_clean.csv") %>%
#  as.data.frame()
```

One main question involves how the median income changed from 2010 to 2020. First, let's add a column called `diff_2010_2020` to see how median income changed for each census tract. The code is pretty simple. Remember, with `summarize`, we used `n()` to count things. With

`mutate`, we use very similar syntax to calculate a new value – a new column of data – using other values in our dataset.

```
baltcity_income %>%
  select(Census, Neighborhood, x2010, x2020) %>%
    mutate(Diff_Income = (x2020-x2010))
```

	Census	Neighborhood	x2010	x2020	Diff_Income
1	101.00	Canton	75938	128839	52901
2	102.00	Patterson Park North & East	58409	130357	71948
3	103.00	Canton	77841	151389	73548
4	104.00	Canton	70313	114946	44633
5	105.00	Fells Point	86157	98194	12037
6	201.00	Fells Point	60439	95536	35097
7	202.00	Fells Point	47131	86125	38994
8	203.00	Fells Point	59236	101815	42579
9	301.00	Harbor East/Little Italy	21932	23083	1151
10	302.00	Harbor East/Little Italy	63885	98254	34369
11	401.00	Downtown/Seton Hill	45948	62131	16183
12	402.00	Downtown/Seton Hill	13229	43333	30104
13	601.00	Patterson Park North & East	46822	56652	9830
14	602.00	Patterson Park North & East	51403	70313	18910
15	603.00	Patterson Park North & East	53813	84318	30505
16	604.00	Oldtown/Middle East	35139	53472	18333
17	701.00	Madison/East End	37701	45602	7901
18	702.00	Madison/East End	35667	31379	-4288
19	703.00	Madison/East End	15000	35000	20000
20	704.00	Oldtown/Middle East	14527	27321	12794
21	801.01	Belair-Edison	49688	65417	15729
22	801.02	Belair-Edison	35594	30000	-5594
23	802.00	Clifton-Berea	24200	38289	14089
24	803.01	Clifton-Berea	32024	45446	13422
25	803.02	Clifton-Berea	24423	41298	16875
26	804.00	Clifton-Berea	23893	34567	10674
27	805.00	Clifton-Berea	35223	28203	-7020
28	806.00	Greenmount East	26726	35221	8495
29	807.00	Greenmount East	20679	40885	20206
30	808.00	Oldtown/Middle East	35532	48137	12605
31	901.00	Greater Govans	36444	52988	16544
32	902.00	Northwood	67500	82609	15109
33	903.00	The Waverlies	43824	63523	19699
34	904.00	The Waverlies	26490	30685	4195

35	905.00	The Waverlies	36507	43727	7220
36	906.00	Midway/Coldstream	36408	50048	13640
37	907.00	Midway/Coldstream	27330	30964	3634
38	908.00	Midway/Coldstream	35473	31250	-4223
39	909.00	Greenmount East	19056	19524	468
40	1001.00	Greenmount East	28929	30913	1984
41	1002.00	Oldtown/Middle East	9862	19172	9310
42	1003.00	Unassigned--Jail	NA	NA	NA
43	1101.00	Midtown	38797	53407	14610
44	1102.00	Midtown	32563	62703	30140
45	1201.00	North Baltimore/Guilford/Homeland	56205	72349	16144
46	1202.01	Greater Charles Village/Barclay	54000	81393	27393
47	1202.02	Greater Charles Village/Barclay	30179	36563	6384
48	1203.00	Greater Charles Village/Barclay	38281	55074	16793
49	1204.00	Greater Charles Village/Barclay	39375	50034	10659
50	1205.00	Midtown	37500	55170	17670
51	1206.00	Greater Charles Village/Barclay	23488	36221	12733
52	1207.00	Medfield/Hampden/Woodberry/Remington	39730	61653	21923
53	1301.00	Penn North/Reservoir Hill	21271	25835	4564
54	1302.00	Penn North/Reservoir Hill	30255	57457	27202
55	1303.00	Penn North/Reservoir Hill	32273	43380	11107
56	1304.00	Penn North/Reservoir Hill	32500	30868	-1632
57	1306.00	Medfield/Hampden/Woodberry/Remington	53654	91635	37981
58	1307.00	Medfield/Hampden/Woodberry/Remington	48634	72648	24014
59	1308.03	Medfield/Hampden/Woodberry/Remington	51548	62188	10640
60	1308.04	Medfield/Hampden/Woodberry/Remington	48207	62054	13847
61	1308.05	Mount Washington/Coldspring	53984	53750	-234
62	1308.06	Medfield/Hampden/Woodberry/Remington	48750	95142	46392
63	1401.00	Midtown	32917	57292	24375
64	1402.00	Upton/Druid Heights	16213	23152	6939
65	1403.00	Upton/Druid Heights	21607	41913	20306
66	1501.00	Sandtown-Winchester/Harlem Park	18097	26989	8892
67	1502.00	Sandtown-Winchester/Harlem Park	25224	40644	15420
68	1503.00	Greater Rosemont	32289	35476	3187
69	1504.00	Greater Mondawmin	32632	33702	1070
70	1505.00	Greater Mondawmin	34609	33967	-642
71	1506.00	Greater Rosemont	29631	31250	1619
72	1507.01	Greater Mondawmin	38125	51944	13819
73	1507.02	Greater Mondawmin	48942	50150	1208
74	1508.00	Forest Park/Walbrook	37199	39337	2138
75	1509.00	Forest Park/Walbrook	37440	51250	13810
76	1510.00	Dorchester/Ashburton	35056	45870	10814
77	1511.00	Dorchester/Ashburton	45649	58843	13194

78	1512.00	Southern Park Heights	12386	22632	10246
79	1513.00	Southern Park Heights	33579	33866	287
80	1601.00	Sandtown-Winchester/Harlem Park	19583	27969	8386
81	1602.00	Sandtown-Winchester/Harlem Park	25417	24848	-569
82	1603.00	Sandtown-Winchester/Harlem Park	22292	13963	-8329
83	1604.00	Sandtown-Winchester/Harlem Park	27813	23036	-4777
84	1605.00	Greater Rosemont	24556	37819	13263
85	1606.00	Greater Rosemont	27128	38482	11354
86	1607.00	Greater Rosemont	29472	39959	10487
87	1608.01	Edmondson Village	41549	53543	11994
88	1608.02	Edmondson Village	29355	41932	12577
89	1701.00	Downtown/Seton Hill	30197	35582	5385
90	1702.00	Upton/Druid Heights	9412	13559	4147
91	1703.00	Upton/Druid Heights	17892	18912	1020
92	1801.00	Poppleton/The Terraces/Hollins Market	18429	NA	NA
93	1802.00	Poppleton/The Terraces/Hollins Market	16585	NA	NA
94	1803.00	Poppleton/The Terraces/Hollins Market	37879	64125	26246
95	1901.00	Southwest Baltimore	22893	24559	1666
96	1902.00	Southwest Baltimore	43214	52738	9524
97	1903.00	Southwest Baltimore	17237	14269	-2968
98	2001.00	Southwest Baltimore	32214	38203	5989
99	2002.00	Southwest Baltimore	30061	27740	-2321
100	2003.00	Southwest Baltimore	19063	21953	2890
101	2004.00	Southwest Baltimore	28324	31975	3651
102	2005.00	Southwest Baltimore	30239	34306	4067
103	2006.00	Allendale/Irvington/S. Hilton	29813	39076	9263
104	2007.01	Allendale/Irvington/S. Hilton	34556	33592	-964
105	2007.02	Allendale/Irvington/S. Hilton	27305	43734	16429
106	2008.00	Allendale/Irvington/S. Hilton	32711	49875	17164
107	2101.00	Washington Village/Pigtown	48857	76368	27511
108	2102.00	Washington Village/Pigtown	46319	36661	-9658
109	2201.00	Inner Harbor/Federal Hill	59259	80305	21046
110	2301.00	Inner Harbor/Federal Hill	62222	98125	35903
111	2302.00	Inner Harbor/Federal Hill	80521	107344	26823
112	2303.00	South Baltimore	53194	92841	39647
113	2401.00	South Baltimore	87619	151659	64040
114	2402.00	Inner Harbor/Federal Hill	92500	133333	40833
115	2403.00	Inner Harbor/Federal Hill	89211	120729	31518
116	2404.00	South Baltimore	72122	121685	49563
117	2501.01	Beechfield/Ten Hills/West Hills	52098	60463	8365
118	2501.02	Allendale/Irvington/S. Hilton	36960	53854	16894
119	2501.03	Morrell Park/Violetville	36957	36555	-402
120	2502.03	Cherry Hill	29747	33902	4155

121	2502.04	Cherry Hill	12384	14349	1965
122	2502.05	Westport/Mount Winans/Lakeland	47738	38779	-8959
123	2502.06	Morrell Park/Violetville	52674	66900	14226
124	2502.07	Cherry Hill	23310	41518	18208
125	2503.01	Westport/Mount Winans/Lakeland	27414	24877	-2537
126	2503.03	Morrell Park/Violetville	42007	40230	-1777
127	2504.01	Brooklyn/Curtis Bay/Hawkins Point	34401	32019	-2382
128	2504.02	Brooklyn/Curtis Bay/Hawkins Point	31250	37059	5809
129	2505.00	Brooklyn/Curtis Bay/Hawkins Point	32738	30526	-2212
130	2506.00	Brooklyn/Curtis Bay/Hawkins Point	NA	NA	NA
131	2601.01	Cedonia/Frankford	46789	68983	22194
132	2601.02	Cedonia/Frankford	48783	60313	11530
133	2602.01	Cedonia/Frankford	33936	44093	10157
134	2602.02	Cedonia/Frankford	31241	39282	8041
135	2602.03	Cedonia/Frankford	32615	46250	13635
136	2603.01	Belair-Edison	42347	52117	9770
137	2603.02	Belair-Edison	42727	50000	7273
138	2603.03	Claremont/Armistead	30192	NA	NA
139	2604.01	Claremont/Armistead	31392	48661	17269
140	2604.02	Claremont/Armistead	42938	39000	-3938
141	2604.03	Claremont/Armistead	32373	35282	2909
142	2604.04	Orangeville/East Highlandtown	35708	54375	18667
143	2605.01	Orangeville/East Highlandtown	42298	64706	22408
144	2606.04	Southeastern	20710	21977	1267
145	2606.05	Southeastern	33545	44794	11249
146	2607.00	Orangeville/East Highlandtown	36901	80000	43099
147	2608.00	Highlandtown	35200	48850	13650
148	2609.00	Highlandtown	68571	107438	38867
149	2610.00	Patterson Park North & East	36343	60893	24550
150	2611.00	Highlandtown	83207	130769	47562
151	2701.01	Lauraville	57326	80875	23549
152	2701.02	Lauraville	54250	58020	3770
153	2702.00	Lauraville	60128	80788	20660
154	2703.01	Lauraville	54623	80655	26032
155	2703.02	Lauraville	68088	63929	-4159
156	2704.01	Hamilton	51641	64620	12979
157	2704.02	Hamilton	53125	84556	31431
158	2705.01	Harford/Echodale	59516	66604	7088
159	2705.02	Hamilton	53493	71694	18201
160	2706.00	Harford/Echodale	66914	67625	711
161	2707.01	Harford/Echodale	28618	37591	8973
162	2707.02	Harford/Echodale	39205	56963	17758
163	2707.03	Harford/Echodale	63521	88750	25229

164	2708.01	Loch Raven	51415	58995	7580
165	2708.02	Loch Raven	46181	55753	9572
166	2708.03	Loch Raven	42234	52272	10038
167	2708.04	Chinquapin Park/Belvedere	41094	59583	18489
168	2708.05	Chinquapin Park/Belvedere	46875	59763	12888
169	2709.01	Northwood	51023	47260	-3763
170	2709.02	Northwood	48555	58269	9714
171	2709.03	Northwood	54607	52083	-2524
172	2710.01	Greater Govans	33203	35822	2619
173	2710.02	Greater Govans	38500	40281	1781
174	2711.01	North Baltimore/Guilford/Homeland	42994	55859	12865
175	2711.02	North Baltimore/Guilford/Homeland	99300	199531	100231
176	2712.00	North Baltimore/Guilford/Homeland	133548	195353	61805
177	2713.00	Greater Roland Park/Poplar Hill	111250	155605	44355
178	2714.00	Greater Roland Park/Poplar Hill	97891	151146	53255
179	2715.01	Mount Washington/Coldspring	88929	109688	20759
180	2715.03	Greater Roland Park/Poplar Hill	79861	62500	-17361
181	2716.00	Southern Park Heights	43265	34832	-8433
182	2717.00	Pimlico/Arlington/Hilltop	29069	36484	7415
183	2718.01	Pimlico/Arlington/Hilltop	24802	25278	476
184	2718.02	Pimlico/Arlington/Hilltop	31392	40256	8864
185	2719.00	Glen-Fallstaff	52308	54866	2558
186	2720.03	Cross-Country/Cheswolde	70942	57543	-13399
187	2720.04	Cross-Country/Cheswolde	49567	66148	16581
188	2720.05	Cross-Country/Cheswolde	49757	72500	22743
189	2720.06	Glen-Fallstaff	24859	34019	9160
190	2720.07	Glen-Fallstaff	35903	41728	5825
191	2801.01	Glen-Fallstaff	46576	47610	1034
192	2801.02	Howard Park/West Arlington	33938	43618	9680
193	2802.00	Howard Park/West Arlington	45047	56458	11411
194	2803.01	Dickeyville/Franklintown	33707	47366	13659
195	2803.02	Forest Park/Walbrook	33864	56004	22140
196	2804.01	Beechfield/Ten Hills/West Hills	44954	47192	2238
197	2804.02	Edmondson Village	52784	41179	-11605
198	2804.03	Beechfield/Ten Hills/West Hills	51435	57582	6147
199	2804.04	Allendale/Irvington/S. Hilton	43779	40946	-2833
200	2805.00	Oldtown/Middle East	11278	14304	3026

Now we've got our `Diff_Income` column. Let's provide some context and use a percentage change calculation to compare the gains. We'll add a new column, `Diff_Pct_2020`. Remember the percentage change calculation is (New-Old)/Old

```r
baltcity_income %>%
  select(Census, Neighborhood, x2010, x2020) %>%
    mutate(Diff_Income = (x2020-x2010)) %>%
    mutate(Diff_Pct_2020 = (x2020-x2010)/x2010)
```

	Census	Neighborhood	x2010	x2020	Diff_Income
1	101.00	Canton	75938	128839	52901
2	102.00	Patterson Park North & East	58409	130357	71948
3	103.00	Canton	77841	151389	73548
4	104.00	Canton	70313	114946	44633
5	105.00	Fells Point	86157	98194	12037
6	201.00	Fells Point	60439	95536	35097
7	202.00	Fells Point	47131	86125	38994
8	203.00	Fells Point	59236	101815	42579
9	301.00	Harbor East/Little Italy	21932	23083	1151
10	302.00	Harbor East/Little Italy	63885	98254	34369
11	401.00	Downtown/Seton Hill	45948	62131	16183
12	402.00	Downtown/Seton Hill	13229	43333	30104
13	601.00	Patterson Park North & East	46822	56652	9830
14	602.00	Patterson Park North & East	51403	70313	18910
15	603.00	Patterson Park North & East	53813	84318	30505
16	604.00	Oldtown/Middle East	35139	53472	18333
17	701.00	Madison/East End	37701	45602	7901
18	702.00	Madison/East End	35667	31379	-4288
19	703.00	Madison/East End	15000	35000	20000
20	704.00	Oldtown/Middle East	14527	27321	12794
21	801.01	Belair-Edison	49688	65417	15729
22	801.02	Belair-Edison	35594	30000	-5594
23	802.00	Clifton-Berea	24200	38289	14089
24	803.01	Clifton-Berea	32024	45446	13422
25	803.02	Clifton-Berea	24423	41298	16875
26	804.00	Clifton-Berea	23893	34567	10674
27	805.00	Clifton-Berea	35223	28203	-7020
28	806.00	Greenmount East	26726	35221	8495
29	807.00	Greenmount East	20679	40885	20206
30	808.00	Oldtown/Middle East	35532	48137	12605
31	901.00	Greater Govans	36444	52988	16544
32	902.00	Northwood	67500	82609	15109
33	903.00	The Waverlies	43824	63523	19699
34	904.00	The Waverlies	26490	30685	4195
35	905.00	The Waverlies	36507	43727	7220
36	906.00	Midway/Coldstream	36408	50048	13640

37	907.00	Midway/Coldstream	27330	30964	3634
38	908.00	Midway/Coldstream	35473	31250	-4223
39	909.00	Greenmount East	19056	19524	468
40	1001.00	Greenmount East	28929	30913	1984
41	1002.00	Oldtown/Middle East	9862	19172	9310
42	1003.00	Unassigned--Jail	NA	NA	NA
43	1101.00	Midtown	38797	53407	14610
44	1102.00	Midtown	32563	62703	30140
45	1201.00	North Baltimore/Guilford/Homeland	56205	72349	16144
46	1202.01	Greater Charles Village/Barclay	54000	81393	27393
47	1202.02	Greater Charles Village/Barclay	30179	36563	6384
48	1203.00	Greater Charles Village/Barclay	38281	55074	16793
49	1204.00	Greater Charles Village/Barclay	39375	50034	10659
50	1205.00	Midtown	37500	55170	17670
51	1206.00	Greater Charles Village/Barclay	23488	36221	12733
52	1207.00	Medfield/Hampden/Woodberry/Remington	39730	61653	21923
53	1301.00	Penn North/Reservoir Hill	21271	25835	4564
54	1302.00	Penn North/Reservoir Hill	30255	57457	27202
55	1303.00	Penn North/Reservoir Hill	32273	43380	11107
56	1304.00	Penn North/Reservoir Hill	32500	30868	-1632
57	1306.00	Medfield/Hampden/Woodberry/Remington	53654	91635	37981
58	1307.00	Medfield/Hampden/Woodberry/Remington	48634	72648	24014
59	1308.03	Medfield/Hampden/Woodberry/Remington	51548	62188	10640
60	1308.04	Medfield/Hampden/Woodberry/Remington	48207	62054	13847
61	1308.05	Mount Washington/Coldspring	53984	53750	-234
62	1308.06	Medfield/Hampden/Woodberry/Remington	48750	95142	46392
63	1401.00	Midtown	32917	57292	24375
64	1402.00	Upton/Druid Heights	16213	23152	6939
65	1403.00	Upton/Druid Heights	21607	41913	20306
66	1501.00	Sandtown-Winchester/Harlem Park	18097	26989	8892
67	1502.00	Sandtown-Winchester/Harlem Park	25224	40644	15420
68	1503.00	Greater Rosemont	32289	35476	3187
69	1504.00	Greater Mondawmin	32632	33702	1070
70	1505.00	Greater Mondawmin	34609	33967	-642
71	1506.00	Greater Rosemont	29631	31250	1619
72	1507.01	Greater Mondawmin	38125	51944	13819
73	1507.02	Greater Mondawmin	48942	50150	1208
74	1508.00	Forest Park/Walbrook	37199	39337	2138
75	1509.00	Forest Park/Walbrook	37440	51250	13810
76	1510.00	Dorchester/Ashburton	35056	45870	10814
77	1511.00	Dorchester/Ashburton	45649	58843	13194
78	1512.00	Southern Park Heights	12386	22632	10246
79	1513.00	Southern Park Heights	33579	33866	287

80	1601.00	Sandtown-Winchester/Harlem Park	19583	27969	8386
81	1602.00	Sandtown-Winchester/Harlem Park	25417	24848	-569
82	1603.00	Sandtown-Winchester/Harlem Park	22292	13963	-8329
83	1604.00	Sandtown-Winchester/Harlem Park	27813	23036	-4777
84	1605.00	Greater Rosemont	24556	37819	13263
85	1606.00	Greater Rosemont	27128	38482	11354
86	1607.00	Greater Rosemont	29472	39959	10487
87	1608.01	Edmondson Village	41549	53543	11994
88	1608.02	Edmondson Village	29355	41932	12577
89	1701.00	Downtown/Seton Hill	30197	35582	5385
90	1702.00	Upton/Druid Heights	9412	13559	4147
91	1703.00	Upton/Druid Heights	17892	18912	1020
92	1801.00	Poppleton/The Terraces/Hollins Market	18429	NA	NA
93	1802.00	Poppleton/The Terraces/Hollins Market	16585	NA	NA
94	1803.00	Poppleton/The Terraces/Hollins Market	37879	64125	26246
95	1901.00	Southwest Baltimore	22893	24559	1666
96	1902.00	Southwest Baltimore	43214	52738	9524
97	1903.00	Southwest Baltimore	17237	14269	-2968
98	2001.00	Southwest Baltimore	32214	38203	5989
99	2002.00	Southwest Baltimore	30061	27740	-2321
100	2003.00	Southwest Baltimore	19063	21953	2890
101	2004.00	Southwest Baltimore	28324	31975	3651
102	2005.00	Southwest Baltimore	30239	34306	4067
103	2006.00	Allendale/Irvington/S. Hilton	29813	39076	9263
104	2007.01	Allendale/Irvington/S. Hilton	34556	33592	-964
105	2007.02	Allendale/Irvington/S. Hilton	27305	43734	16429
106	2008.00	Allendale/Irvington/S. Hilton	32711	49875	17164
107	2101.00	Washington Village/Pigtown	48857	76368	27511
108	2102.00	Washington Village/Pigtown	46319	36661	-9658
109	2201.00	Inner Harbor/Federal Hill	59259	80305	21046
110	2301.00	Inner Harbor/Federal Hill	62222	98125	35903
111	2302.00	Inner Harbor/Federal Hill	80521	107344	26823
112	2303.00	South Baltimore	53194	92841	39647
113	2401.00	South Baltimore	87619	151659	64040
114	2402.00	Inner Harbor/Federal Hill	92500	133333	40833
115	2403.00	Inner Harbor/Federal Hill	89211	120729	31518
116	2404.00	South Baltimore	72122	121685	49563
117	2501.01	Beechfield/Ten Hills/West Hills	52098	60463	8365
118	2501.02	Allendale/Irvington/S. Hilton	36960	53854	16894
119	2501.03	Morrell Park/Violetville	36957	36555	-402
120	2502.03	Cherry Hill	29747	33902	4155
121	2502.04	Cherry Hill	12384	14349	1965
122	2502.05	Westport/Mount Winans/Lakeland	47738	38779	-8959

123	2502.06	Morrell Park/Violetville	52674	66900	14226
124	2502.07	Cherry Hill	23310	41518	18208
125	2503.01	Westport/Mount Winans/Lakeland	27414	24877	-2537
126	2503.03	Morrell Park/Violetville	42007	40230	-1777
127	2504.01	Brooklyn/Curtis Bay/Hawkins Point	34401	32019	-2382
128	2504.02	Brooklyn/Curtis Bay/Hawkins Point	31250	37059	5809
129	2505.00	Brooklyn/Curtis Bay/Hawkins Point	32738	30526	-2212
130	2506.00	Brooklyn/Curtis Bay/Hawkins Point	NA	NA	NA
131	2601.01	Cedonia/Frankford	46789	68983	22194
132	2601.02	Cedonia/Frankford	48783	60313	11530
133	2602.01	Cedonia/Frankford	33936	44093	10157
134	2602.02	Cedonia/Frankford	31241	39282	8041
135	2602.03	Cedonia/Frankford	32615	46250	13635
136	2603.01	Belair-Edison	42347	52117	9770
137	2603.02	Belair-Edison	42727	50000	7273
138	2603.03	Claremont/Armistead	30192	NA	NA
139	2604.01	Claremont/Armistead	31392	48661	17269
140	2604.02	Claremont/Armistead	42938	39000	-3938
141	2604.03	Claremont/Armistead	32373	35282	2909
142	2604.04	Orangeville/East Highlandtown	35708	54375	18667
143	2605.01	Orangeville/East Highlandtown	42298	64706	22408
144	2606.04	Southeastern	20710	21977	1267
145	2606.05	Southeastern	33545	44794	11249
146	2607.00	Orangeville/East Highlandtown	36901	80000	43099
147	2608.00	Highlandtown	35200	48850	13650
148	2609.00	Highlandtown	68571	107438	38867
149	2610.00	Patterson Park North & East	36343	60893	24550
150	2611.00	Highlandtown	83207	130769	47562
151	2701.01	Lauraville	57326	80875	23549
152	2701.02	Lauraville	54250	58020	3770
153	2702.00	Lauraville	60128	80788	20660
154	2703.01	Lauraville	54623	80655	26032
155	2703.02	Lauraville	68088	63929	-4159
156	2704.01	Hamilton	51641	64620	12979
157	2704.02	Hamilton	53125	84556	31431
158	2705.01	Harford/Echodale	59516	66604	7088
159	2705.02	Hamilton	53493	71694	18201
160	2706.00	Harford/Echodale	66914	67625	711
161	2707.01	Harford/Echodale	28618	37591	8973
162	2707.02	Harford/Echodale	39205	56963	17758
163	2707.03	Harford/Echodale	63521	88750	25229
164	2708.01	Loch Raven	51415	58995	7580
165	2708.02	Loch Raven	46181	55753	9572

166	2708.03	Loch Raven	42234	52272	10038	
167	2708.04	Chinquapin Park/Belvedere	41094	59583	18489	
168	2708.05	Chinquapin Park/Belvedere	46875	59763	12888	
169	2709.01	Northwood	51023	47260	-3763	
170	2709.02	Northwood	48555	58269	9714	
171	2709.03	Northwood	54607	52083	-2524	
172	2710.01	Greater Govans	33203	35822	2619	
173	2710.02	Greater Govans	38500	40281	1781	
174	2711.01	North Baltimore/Guilford/Homeland	42994	55859	12865	
175	2711.02	North Baltimore/Guilford/Homeland	99300	199531	100231	
176	2712.00	North Baltimore/Guilford/Homeland	133548	195353	61805	
177	2713.00	Greater Roland Park/Poplar Hill	111250	155605	44355	
178	2714.00	Greater Roland Park/Poplar Hill	97891	151146	53255	
179	2715.01	Mount Washington/Coldspring	88929	109688	20759	
180	2715.03	Greater Roland Park/Poplar Hill	79861	62500	-17361	
181	2716.00	Southern Park Heights	43265	34832	-8433	
182	2717.00	Pimlico/Arlington/Hilltop	29069	36484	7415	
183	2718.01	Pimlico/Arlington/Hilltop	24802	25278	476	
184	2718.02	Pimlico/Arlington/Hilltop	31392	40256	8864	
185	2719.00	Glen-Fallstaff	52308	54866	2558	
186	2720.03	Cross-Country/Cheswolde	70942	57543	-13399	
187	2720.04	Cross-Country/Cheswolde	49567	66148	16581	
188	2720.05	Cross-Country/Cheswolde	49757	72500	22743	
189	2720.06	Glen-Fallstaff	24859	34019	9160	
190	2720.07	Glen-Fallstaff	35903	41728	5825	
191	2801.01	Glen-Fallstaff	46576	47610	1034	
192	2801.02	Howard Park/West Arlington	33938	43618	9680	
193	2802.00	Howard Park/West Arlington	45047	56458	11411	
194	2803.01	Dickeyville/Franklintown	33707	47366	13659	
195	2803.02	Forest Park/Walbrook	33864	56004	22140	
196	2804.01	Beechfield/Ten Hills/West Hills	44954	47192	2238	
197	2804.02	Edmondson Village	52784	41179	-11605	
198	2804.03	Beechfield/Ten Hills/West Hills	51435	57582	6147	
199	2804.04	Allendale/Irvington/S. Hilton	43779	40946	-2833	
200	2805.00	Oldtown/Middle East	11278	14304	3026	

	Diff_Pct_2020
1	0.696634096
2	1.231796470
3	0.944849116
4	0.634775930
5	0.139710064
6	0.580701203
7	0.827353546

```
8     0.718802755
9     0.052480394
10    0.537982312
11    0.352202490
12    2.275606622
13    0.209944043
14    0.367877361
15    0.566870459
16    0.521727995
17    0.209570038
18   -0.120223175
19    1.333333333
20    0.880704894
21    0.316555305
22   -0.157161319
23    0.582190083
24    0.419123158
25    0.690947058
26    0.446741724
27   -0.199301593
28    0.317855272
29    0.977126554
30    0.354750647
31    0.453956756
32    0.223837037
33    0.449502556
34    0.158361646
35    0.197770291
36    0.374642936
37    0.132967435
38   -0.119048290
39    0.024559194
40    0.068581700
41    0.944027581
42             NA
43    0.376575508
44    0.925590394
45    0.287234232
46    0.507277778
47    0.211537824
48    0.438677151
49    0.270704762
50    0.471200000
```

```
51    0.542106608
52    0.551799648
53    0.214564430
54    0.899091059
55    0.344157655
56   -0.050215385
57    0.707887576
58    0.493769791
59    0.206409560
60    0.287240442
61   -0.004334618
62    0.951630769
63    0.740498830
64    0.427989885
65    0.939788032
66    0.491352158
67    0.611322550
68    0.098702344
69    0.032789899
70   -0.018550088
71    0.054638723
72    0.362465574
73    0.024682277
74    0.057474663
75    0.368856838
76    0.308477864
77    0.289031523
78    0.827224285
79    0.008547009
80    0.428228566
81   -0.022386592
82   -0.373631796
83   -0.171754216
84    0.540112396
85    0.418534356
86    0.355829262
87    0.288671207
88    0.428444899
89    0.178328973
90    0.440607735
91    0.057008719
92             NA
93             NA
```

```
94    0.692890520
95    0.072773337
96    0.220391540
97   -0.172187736
98    0.185912957
99   -0.077209674
100   0.151602581
101   0.128901285
102   0.134495188
103   0.310703384
104  -0.027896747
105   0.601684673
106   0.524716456
107   0.563092290
108  -0.208510546
109   0.355152804
110   0.577014561
111   0.333118069
112   0.745328420
113   0.730891702
114   0.441437838
115   0.353297239
116   0.687210560
117   0.160562786
118   0.457088745
119  -0.010877506
120   0.139677951
121   0.158672481
122  -0.187670200
123   0.270076318
124   0.781123981
125  -0.092543956
126  -0.042302473
127  -0.069242173
128   0.185888000
129  -0.067566742
130            NA
131   0.474342260
132   0.236352828
133   0.299298680
134   0.257386127
135   0.418059175
136   0.230712919
```

```
137   0.170220235
138            NA
139   0.550108308
140  -0.091713634
141   0.089858833
142   0.522768007
143   0.529765001
144   0.061178175
145   0.335340587
146   1.167962928
147   0.387784091
148   0.566813959
149   0.675508351
150   0.571610562
151   0.410790915
152   0.069493088
153   0.343600319
154   0.476575801
155  -0.061082716
156   0.251331307
157   0.591642353
158   0.119094025
159   0.340250126
160   0.010625579
161   0.313543923
162   0.452952430
163   0.397175737
164   0.147427793
165   0.207271389
166   0.237675806
167   0.449919696
168   0.274944000
169  -0.073751053
170   0.200061786
171  -0.046221180
172   0.078878415
173   0.046259740
174   0.299227799
175   1.009375629
176   0.462792404
177   0.398696629
178   0.544023455
179   0.233433413
```

```
180  -0.217390215
181  -0.194915058
182   0.255082734
183   0.019192001
184   0.282364934
185   0.048902654
186  -0.188872600
187   0.334516916
188   0.457081416
189   0.368478217
190   0.162242710
191   0.022200275
192   0.285226000
193   0.253313206
194   0.405227401
195   0.653791637
196   0.049784224
197  -0.219858290
198   0.119510061
199  -0.064711391
200   0.268309984
```

Look at `Diff_Pct_2020`. Do those numbers look like we expect them to? No. They're a decimal expressed as a percentage. So let's fix that by multiplying by 100. We're also rounding the result to two digits from nine so it looks cleaner

```
baltcity_income %>%
  select(Census, Neighborhood, x2010, x2020) %>%
    mutate(Diff_Income = (x2020-x2010)) %>%
    mutate(Diff_Pct_2020 = round((x2020-x2010)/x2010*100,2))
```

```
     Census                          Neighborhood  x2010  x2020 Diff_Income
1    101.00                                Canton  75938 128839       52901
2    102.00          Patterson Park North & East  58409 130357       71948
3    103.00                                Canton  77841 151389       73548
4    104.00                                Canton  70313 114946       44633
5    105.00                           Fells Point  86157  98194       12037
6    201.00                           Fells Point  60439  95536       35097
7    202.00                           Fells Point  47131  86125       38994
8    203.00                           Fells Point  59236 101815       42579
9    301.00              Harbor East/Little Italy  21932  23083        1151
10   302.00              Harbor East/Little Italy  63885  98254       34369
```

11	401.00	Downtown/Seton Hill	45948	62131	16183
12	402.00	Downtown/Seton Hill	13229	43333	30104
13	601.00	Patterson Park North & East	46822	56652	9830
14	602.00	Patterson Park North & East	51403	70313	18910
15	603.00	Patterson Park North & East	53813	84318	30505
16	604.00	Oldtown/Middle East	35139	53472	18333
17	701.00	Madison/East End	37701	45602	7901
18	702.00	Madison/East End	35667	31379	-4288
19	703.00	Madison/East End	15000	35000	20000
20	704.00	Oldtown/Middle East	14527	27321	12794
21	801.01	Belair-Edison	49688	65417	15729
22	801.02	Belair-Edison	35594	30000	-5594
23	802.00	Clifton-Berea	24200	38289	14089
24	803.01	Clifton-Berea	32024	45446	13422
25	803.02	Clifton-Berea	24423	41298	16875
26	804.00	Clifton-Berea	23893	34567	10674
27	805.00	Clifton-Berea	35223	28203	-7020
28	806.00	Greenmount East	26726	35221	8495
29	807.00	Greenmount East	20679	40885	20206
30	808.00	Oldtown/Middle East	35532	48137	12605
31	901.00	Greater Govans	36444	52988	16544
32	902.00	Northwood	67500	82609	15109
33	903.00	The Waverlies	43824	63523	19699
34	904.00	The Waverlies	26490	30685	4195
35	905.00	The Waverlies	36507	43727	7220
36	906.00	Midway/Coldstream	36408	50048	13640
37	907.00	Midway/Coldstream	27330	30964	3634
38	908.00	Midway/Coldstream	35473	31250	-4223
39	909.00	Greenmount East	19056	19524	468
40	1001.00	Greenmount East	28929	30913	1984
41	1002.00	Oldtown/Middle East	9862	19172	9310
42	1003.00	Unassigned--Jail	NA	NA	NA
43	1101.00	Midtown	38797	53407	14610
44	1102.00	Midtown	32563	62703	30140
45	1201.00	North Baltimore/Guilford/Homeland	56205	72349	16144
46	1202.01	Greater Charles Village/Barclay	54000	81393	27393
47	1202.02	Greater Charles Village/Barclay	30179	36563	6384
48	1203.00	Greater Charles Village/Barclay	38281	55074	16793
49	1204.00	Greater Charles Village/Barclay	39375	50034	10659
50	1205.00	Midtown	37500	55170	17670
51	1206.00	Greater Charles Village/Barclay	23488	36221	12733
52	1207.00	Medfield/Hampden/Woodberry/Remington	39730	61653	21923
53	1301.00	Penn North/Reservoir Hill	21271	25835	4564

54	1302.00	Penn North/Reservoir Hill	30255	57457	27202
55	1303.00	Penn North/Reservoir Hill	32273	43380	11107
56	1304.00	Penn North/Reservoir Hill	32500	30868	-1632
57	1306.00	Medfield/Hampden/Woodberry/Remington	53654	91635	37981
58	1307.00	Medfield/Hampden/Woodberry/Remington	48634	72648	24014
59	1308.03	Medfield/Hampden/Woodberry/Remington	51548	62188	10640
60	1308.04	Medfield/Hampden/Woodberry/Remington	48207	62054	13847
61	1308.05	Mount Washington/Coldspring	53984	53750	-234
62	1308.06	Medfield/Hampden/Woodberry/Remington	48750	95142	46392
63	1401.00	Midtown	32917	57292	24375
64	1402.00	Upton/Druid Heights	16213	23152	6939
65	1403.00	Upton/Druid Heights	21607	41913	20306
66	1501.00	Sandtown-Winchester/Harlem Park	18097	26989	8892
67	1502.00	Sandtown-Winchester/Harlem Park	25224	40644	15420
68	1503.00	Greater Rosemont	32289	35476	3187
69	1504.00	Greater Mondawmin	32632	33702	1070
70	1505.00	Greater Mondawmin	34609	33967	-642
71	1506.00	Greater Rosemont	29631	31250	1619
72	1507.01	Greater Mondawmin	38125	51944	13819
73	1507.02	Greater Mondawmin	48942	50150	1208
74	1508.00	Forest Park/Walbrook	37199	39337	2138
75	1509.00	Forest Park/Walbrook	37440	51250	13810
76	1510.00	Dorchester/Ashburton	35056	45870	10814
77	1511.00	Dorchester/Ashburton	45649	58843	13194
78	1512.00	Southern Park Heights	12386	22632	10246
79	1513.00	Southern Park Heights	33579	33866	287
80	1601.00	Sandtown-Winchester/Harlem Park	19583	27969	8386
81	1602.00	Sandtown-Winchester/Harlem Park	25417	24848	-569
82	1603.00	Sandtown-Winchester/Harlem Park	22292	13963	-8329
83	1604.00	Sandtown-Winchester/Harlem Park	27813	23036	-4777
84	1605.00	Greater Rosemont	24556	37819	13263
85	1606.00	Greater Rosemont	27128	38482	11354
86	1607.00	Greater Rosemont	29472	39959	10487
87	1608.01	Edmondson Village	41549	53543	11994
88	1608.02	Edmondson Village	29355	41932	12577
89	1701.00	Downtown/Seton Hill	30197	35582	5385
90	1702.00	Upton/Druid Heights	9412	13559	4147
91	1703.00	Upton/Druid Heights	17892	18912	1020
92	1801.00	Poppleton/The Terraces/Hollins Market	18429	NA	NA
93	1802.00	Poppleton/The Terraces/Hollins Market	16585	NA	NA
94	1803.00	Poppleton/The Terraces/Hollins Market	37879	64125	26246
95	1901.00	Southwest Baltimore	22893	24559	1666
96	1902.00	Southwest Baltimore	43214	52738	9524

97	1903.00	Southwest Baltimore	17237	14269	-2968
98	2001.00	Southwest Baltimore	32214	38203	5989
99	2002.00	Southwest Baltimore	30061	27740	-2321
100	2003.00	Southwest Baltimore	19063	21953	2890
101	2004.00	Southwest Baltimore	28324	31975	3651
102	2005.00	Southwest Baltimore	30239	34306	4067
103	2006.00	Allendale/Irvington/S. Hilton	29813	39076	9263
104	2007.01	Allendale/Irvington/S. Hilton	34556	33592	-964
105	2007.02	Allendale/Irvington/S. Hilton	27305	43734	16429
106	2008.00	Allendale/Irvington/S. Hilton	32711	49875	17164
107	2101.00	Washington Village/Pigtown	48857	76368	27511
108	2102.00	Washington Village/Pigtown	46319	36661	-9658
109	2201.00	Inner Harbor/Federal Hill	59259	80305	21046
110	2301.00	Inner Harbor/Federal Hill	62222	98125	35903
111	2302.00	Inner Harbor/Federal Hill	80521	107344	26823
112	2303.00	South Baltimore	53194	92841	39647
113	2401.00	South Baltimore	87619	151659	64040
114	2402.00	Inner Harbor/Federal Hill	92500	133333	40833
115	2403.00	Inner Harbor/Federal Hill	89211	120729	31518
116	2404.00	South Baltimore	72122	121685	49563
117	2501.01	Beechfield/Ten Hills/West Hills	52098	60463	8365
118	2501.02	Allendale/Irvington/S. Hilton	36960	53854	16894
119	2501.03	Morrell Park/Violetville	36957	36555	-402
120	2502.03	Cherry Hill	29747	33902	4155
121	2502.04	Cherry Hill	12384	14349	1965
122	2502.05	Westport/Mount Winans/Lakeland	47738	38779	-8959
123	2502.06	Morrell Park/Violetville	52674	66900	14226
124	2502.07	Cherry Hill	23310	41518	18208
125	2503.01	Westport/Mount Winans/Lakeland	27414	24877	-2537
126	2503.03	Morrell Park/Violetville	42007	40230	-1777
127	2504.01	Brooklyn/Curtis Bay/Hawkins Point	34401	32019	-2382
128	2504.02	Brooklyn/Curtis Bay/Hawkins Point	31250	37059	5809
129	2505.00	Brooklyn/Curtis Bay/Hawkins Point	32738	30526	-2212
130	2506.00	Brooklyn/Curtis Bay/Hawkins Point	NA	NA	NA
131	2601.01	Cedonia/Frankford	46789	68983	22194
132	2601.02	Cedonia/Frankford	48783	60313	11530
133	2602.01	Cedonia/Frankford	33936	44093	10157
134	2602.02	Cedonia/Frankford	31241	39282	8041
135	2602.03	Cedonia/Frankford	32615	46250	13635
136	2603.01	Belair-Edison	42347	52117	9770
137	2603.02	Belair-Edison	42727	50000	7273
138	2603.03	Claremont/Armistead	30192	NA	NA
139	2604.01	Claremont/Armistead	31392	48661	17269

140	2604.02	Claremont/Armistead	42938	39000	-3938
141	2604.03	Claremont/Armistead	32373	35282	2909
142	2604.04	Orangeville/East Highlandtown	35708	54375	18667
143	2605.01	Orangeville/East Highlandtown	42298	64706	22408
144	2606.04	Southeastern	20710	21977	1267
145	2606.05	Southeastern	33545	44794	11249
146	2607.00	Orangeville/East Highlandtown	36901	80000	43099
147	2608.00	Highlandtown	35200	48850	13650
148	2609.00	Highlandtown	68571	107438	38867
149	2610.00	Patterson Park North & East	36343	60893	24550
150	2611.00	Highlandtown	83207	130769	47562
151	2701.01	Lauraville	57326	80875	23549
152	2701.02	Lauraville	54250	58020	3770
153	2702.00	Lauraville	60128	80788	20660
154	2703.01	Lauraville	54623	80655	26032
155	2703.02	Lauraville	68088	63929	-4159
156	2704.01	Hamilton	51641	64620	12979
157	2704.02	Hamilton	53125	84556	31431
158	2705.01	Harford/Echodale	59516	66604	7088
159	2705.02	Hamilton	53493	71694	18201
160	2706.00	Harford/Echodale	66914	67625	711
161	2707.01	Harford/Echodale	28618	37591	8973
162	2707.02	Harford/Echodale	39205	56963	17758
163	2707.03	Harford/Echodale	63521	88750	25229
164	2708.01	Loch Raven	51415	58995	7580
165	2708.02	Loch Raven	46181	55753	9572
166	2708.03	Loch Raven	42234	52272	10038
167	2708.04	Chinquapin Park/Belvedere	41094	59583	18489
168	2708.05	Chinquapin Park/Belvedere	46875	59763	12888
169	2709.01	Northwood	51023	47260	-3763
170	2709.02	Northwood	48555	58269	9714
171	2709.03	Northwood	54607	52083	-2524
172	2710.01	Greater Govans	33203	35822	2619
173	2710.02	Greater Govans	38500	40281	1781
174	2711.01	North Baltimore/Guilford/Homeland	42994	55859	12865
175	2711.02	North Baltimore/Guilford/Homeland	99300	199531	100231
176	2712.00	North Baltimore/Guilford/Homeland	133548	195353	61805
177	2713.00	Greater Roland Park/Poplar Hill	111250	155605	44355
178	2714.00	Greater Roland Park/Poplar Hill	97891	151146	53255
179	2715.01	Mount Washington/Coldspring	88929	109688	20759
180	2715.03	Greater Roland Park/Poplar Hill	79861	62500	-17361
181	2716.00	Southern Park Heights	43265	34832	-8433
182	2717.00	Pimlico/Arlington/Hilltop	29069	36484	7415

183	2718.01	Pimlico/Arlington/Hilltop	24802	25278	476
184	2718.02	Pimlico/Arlington/Hilltop	31392	40256	8864
185	2719.00	Glen-Fallstaff	52308	54866	2558
186	2720.03	Cross-Country/Cheswolde	70942	57543	-13399
187	2720.04	Cross-Country/Cheswolde	49567	66148	16581
188	2720.05	Cross-Country/Cheswolde	49757	72500	22743
189	2720.06	Glen-Fallstaff	24859	34019	9160
190	2720.07	Glen-Fallstaff	35903	41728	5825
191	2801.01	Glen-Fallstaff	46576	47610	1034
192	2801.02	Howard Park/West Arlington	33938	43618	9680
193	2802.00	Howard Park/West Arlington	45047	56458	11411
194	2803.01	Dickeyville/Franklintown	33707	47366	13659
195	2803.02	Forest Park/Walbrook	33864	56004	22140
196	2804.01	Beechfield/Ten Hills/West Hills	44954	47192	2238
197	2804.02	Edmondson Village	52784	41179	-11605
198	2804.03	Beechfield/Ten Hills/West Hills	51435	57582	6147
199	2804.04	Allendale/Irvington/S. Hilton	43779	40946	-2833
200	2805.00	Oldtown/Middle East	11278	14304	3026

	Diff_Pct_2020
1	69.66
2	123.18
3	94.48
4	63.48
5	13.97
6	58.07
7	82.74
8	71.88
9	5.25
10	53.80
11	35.22
12	227.56
13	20.99
14	36.79
15	56.69
16	52.17
17	20.96
18	-12.02
19	133.33
20	88.07
21	31.66
22	-15.72
23	58.22
24	41.91

```
25          69.09
26          44.67
27         -19.93
28          31.79
29          97.71
30          35.48
31          45.40
32          22.38
33          44.95
34          15.84
35          19.78
36          37.46
37          13.30
38         -11.90
39           2.46
40           6.86
41          94.40
42             NA
43          37.66
44          92.56
45          28.72
46          50.73
47          21.15
48          43.87
49          27.07
50          47.12
51          54.21
52          55.18
53          21.46
54          89.91
55          34.42
56          -5.02
57          70.79
58          49.38
59          20.64
60          28.72
61          -0.43
62          95.16
63          74.05
64          42.80
65          93.98
66          49.14
67          61.13
```

68	9.87
69	3.28
70	-1.86
71	5.46
72	36.25
73	2.47
74	5.75
75	36.89
76	30.85
77	28.90
78	82.72
79	0.85
80	42.82
81	-2.24
82	-37.36
83	-17.18
84	54.01
85	41.85
86	35.58
87	28.87
88	42.84
89	17.83
90	44.06
91	5.70
92	NA
93	NA
94	69.29
95	7.28
96	22.04
97	-17.22
98	18.59
99	-7.72
100	15.16
101	12.89
102	13.45
103	31.07
104	-2.79
105	60.17
106	52.47
107	56.31
108	-20.85
109	35.52
110	57.70

```
111         33.31
112         74.53
113         73.09
114         44.14
115         35.33
116         68.72
117         16.06
118         45.71
119         -1.09
120         13.97
121         15.87
122        -18.77
123         27.01
124         78.11
125         -9.25
126         -4.23
127         -6.92
128         18.59
129         -6.76
130            NA
131         47.43
132         23.64
133         29.93
134         25.74
135         41.81
136         23.07
137         17.02
138            NA
139         55.01
140         -9.17
141          8.99
142         52.28
143         52.98
144          6.12
145         33.53
146        116.80
147         38.78
148         56.68
149         67.55
150         57.16
151         41.08
152          6.95
153         34.36
```

154	47.66
155	-6.11
156	25.13
157	59.16
158	11.91
159	34.03
160	1.06
161	31.35
162	45.30
163	39.72
164	14.74
165	20.73
166	23.77
167	44.99
168	27.49
169	-7.38
170	20.01
171	-4.62
172	7.89
173	4.63
174	29.92
175	100.94
176	46.28
177	39.87
178	54.40
179	23.34
180	-21.74
181	-19.49
182	25.51
183	1.92
184	28.24
185	4.89
186	-18.89
187	33.45
188	45.71
189	36.85
190	16.22
191	2.22
192	28.52
193	25.33
194	40.52
195	65.38
196	4.98

```
197        -21.99
198         11.95
199         -6.47
200         26.83
```

Now, let's fix the ordering with arrange and sort descending on the percentage change column.

```
baltcity_income %>%
  select(Census, Neighborhood, x2010, x2020) %>%
    mutate(Diff_Income = (x2020-x2010)) %>%
    mutate(Diff_Pct_2020 = round((x2020-x2010)/x2010*100,2)) %>%
    arrange(desc(Diff_Pct_2020))
```

```
     Census                          Neighborhood  x2010  x2020 Diff_Income
1    402.00                   Downtown/Seton Hill  13229  43333       30104
2    703.00                      Madison/East End  15000  35000       20000
3    102.00           Patterson Park North & East  58409 130357       71948
4   2607.00         Orangeville/East Highlandtown  36901  80000       43099
5   2711.02     North Baltimore/Guilford/Homeland  99300 199531      100231
6    807.00                      Greenmount East  20679  40885       20206
7   1308.06  Medfield/Hampden/Woodberry/Remington  48750  95142       46392
8    103.00                               Canton  77841 151389       73548
9   1002.00                  Oldtown/Middle East   9862  19172        9310
10  1403.00                  Upton/Druid Heights  21607  41913       20306
11  1102.00                              Midtown  32563  62703       30140
12  1302.00            Penn North/Reservoir Hill  30255  57457       27202
13   704.00                  Oldtown/Middle East  14527  27321       12794
14   202.00                          Fells Point  47131  86125       38994
15  1512.00                Southern Park Heights  12386  22632       10246
16  2502.07                          Cherry Hill  23310  41518       18208
17  2303.00                      South Baltimore  53194  92841       39647
18  1401.00                              Midtown  32917  57292       24375
19  2401.00                      South Baltimore  87619 151659       64040
20   203.00                          Fells Point  59236 101815       42579
21  1306.00  Medfield/Hampden/Woodberry/Remington  53654  91635       37981
22   101.00                               Canton  75938 128839       52901
23  1803.00 Poppleton/The Terraces/Hollins Market  37879  64125       26246
24   803.02                        Clifton-Berea  24423  41298       16875
25  2404.00                      South Baltimore  72122 121685       49563
26  2610.00           Patterson Park North & East  36343  60893       24550
27  2803.02                 Forest Park/Walbrook  33864  56004       22140
```

28	104.00	Canton	70313	114946	44633
29	1502.00	Sandtown-Winchester/Harlem Park	25224	40644	15420
30	2007.02	Allendale/Irvington/S. Hilton	27305	43734	16429
31	2704.02	Hamilton	53125	84556	31431
32	802.00	Clifton-Berea	24200	38289	14089
33	201.00	Fells Point	60439	95536	35097
34	2301.00	Inner Harbor/Federal Hill	62222	98125	35903
35	2611.00	Highlandtown	83207	130769	47562
36	603.00	Patterson Park North & East	53813	84318	30505
37	2609.00	Highlandtown	68571	107438	38867
38	2101.00	Washington Village/Pigtown	48857	76368	27511
39	1207.00	Medfield/Hampden/Woodberry/Remington	39730	61653	21923
40	2604.01	Claremont/Armistead	31392	48661	17269
41	2714.00	Greater Roland Park/Poplar Hill	97891	151146	53255
42	1206.00	Greater Charles Village/Barclay	23488	36221	12733
43	1605.00	Greater Rosemont	24556	37819	13263
44	302.00	Harbor East/Little Italy	63885	98254	34369
45	2605.01	Orangeville/East Highlandtown	42298	64706	22408
46	2008.00	Allendale/Irvington/S. Hilton	32711	49875	17164
47	2604.04	Orangeville/East Highlandtown	35708	54375	18667
48	604.00	Oldtown/Middle East	35139	53472	18333
49	1202.01	Greater Charles Village/Barclay	54000	81393	27393
50	1307.00	Medfield/Hampden/Woodberry/Remington	48634	72648	24014
51	1501.00	Sandtown-Winchester/Harlem Park	18097	26989	8892
52	2703.01	Lauraville	54623	80655	26032
53	2601.01	Cedonia/Frankford	46789	68983	22194
54	1205.00	Midtown	37500	55170	17670
55	2712.00	North Baltimore/Guilford/Homeland	133548	195353	61805
56	2501.02	Allendale/Irvington/S. Hilton	36960	53854	16894
57	2720.05	Cross-Country/Cheswolde	49757	72500	22743
58	901.00	Greater Govans	36444	52988	16544
59	2707.02	Harford/Echodale	39205	56963	17758
60	2708.04	Chinquapin Park/Belvedere	41094	59583	18489
61	903.00	The Waverlies	43824	63523	19699
62	804.00	Clifton-Berea	23893	34567	10674
63	2402.00	Inner Harbor/Federal Hill	92500	133333	40833
64	1702.00	Upton/Druid Heights	9412	13559	4147
65	1203.00	Greater Charles Village/Barclay	38281	55074	16793
66	1608.02	Edmondson Village	29355	41932	12577
67	1601.00	Sandtown-Winchester/Harlem Park	19583	27969	8386
68	1402.00	Upton/Druid Heights	16213	23152	6939
69	803.01	Clifton-Berea	32024	45446	13422
70	1606.00	Greater Rosemont	27128	38482	11354

71	2602.03	Cedonia/Frankford	32615	46250	13635
72	2701.01	Lauraville	57326	80875	23549
73	2803.01	Dickeyville/Franklintown	33707	47366	13659
74	2713.00	Greater Roland Park/Poplar Hill	111250	155605	44355
75	2707.03	Harford/Echodale	63521	88750	25229
76	2608.00	Highlandtown	35200	48850	13650
77	1101.00	Midtown	38797	53407	14610
78	906.00	Midway/Coldstream	36408	50048	13640
79	1509.00	Forest Park/Walbrook	37440	51250	13810
80	2720.06	Glen-Fallstaff	24859	34019	9160
81	602.00	Patterson Park North & East	51403	70313	18910
82	1507.01	Greater Mondawmin	38125	51944	13819
83	1607.00	Greater Rosemont	29472	39959	10487
84	2201.00	Inner Harbor/Federal Hill	59259	80305	21046
85	808.00	Oldtown/Middle East	35532	48137	12605
86	2403.00	Inner Harbor/Federal Hill	89211	120729	31518
87	401.00	Downtown/Seton Hill	45948	62131	16183
88	1303.00	Penn North/Reservoir Hill	32273	43380	11107
89	2702.00	Lauraville	60128	80788	20660
90	2705.02	Hamilton	53493	71694	18201
91	2606.05	Southeastern	33545	44794	11249
92	2720.04	Cross-Country/Cheswolde	49567	66148	16581
93	2302.00	Inner Harbor/Federal Hill	80521	107344	26823
94	806.00	Greenmount East	26726	35221	8495
95	801.01	Belair-Edison	49688	65417	15729
96	2707.01	Harford/Echodale	28618	37591	8973
97	2006.00	Allendale/Irvington/S. Hilton	29813	39076	9263
98	1510.00	Dorchester/Ashburton	35056	45870	10814
99	2602.01	Cedonia/Frankford	33936	44093	10157
100	2711.01	North Baltimore/Guilford/Homeland	42994	55859	12865
101	1511.00	Dorchester/Ashburton	45649	58843	13194
102	1608.01	Edmondson Village	41549	53543	11994
103	1201.00	North Baltimore/Guilford/Homeland	56205	72349	16144
104	1308.04	Medfield/Hampden/Woodberry/Remington	48207	62054	13847
105	2801.02	Howard Park/West Arlington	33938	43618	9680
106	2718.02	Pimlico/Arlington/Hilltop	31392	40256	8864
107	2708.05	Chinquapin Park/Belvedere	46875	59763	12888
108	1204.00	Greater Charles Village/Barclay	39375	50034	10659
109	2502.06	Morrell Park/Violetville	52674	66900	14226
110	2805.00	Oldtown/Middle East	11278	14304	3026
111	2602.02	Cedonia/Frankford	31241	39282	8041
112	2717.00	Pimlico/Arlington/Hilltop	29069	36484	7415
113	2802.00	Howard Park/West Arlington	45047	56458	11411

114	2704.01	Hamilton	51641	64620	12979
115	2708.03	Loch Raven	42234	52272	10038
116	2601.02	Cedonia/Frankford	48783	60313	11530
117	2715.01	Mount Washington/Coldspring	88929	109688	20759
118	2603.01	Belair-Edison	42347	52117	9770
119	902.00	Northwood	67500	82609	15109
120	1902.00	Southwest Baltimore	43214	52738	9524
121	1301.00	Penn North/Reservoir Hill	21271	25835	4564
122	1202.02	Greater Charles Village/Barclay	30179	36563	6384
123	601.00	Patterson Park North & East	46822	56652	9830
124	701.00	Madison/East End	37701	45602	7901
125	2708.02	Loch Raven	46181	55753	9572
126	1308.03	Medfield/Hampden/Woodberry/Remington	51548	62188	10640
127	2709.02	Northwood	48555	58269	9714
128	905.00	The Waverlies	36507	43727	7220
129	2001.00	Southwest Baltimore	32214	38203	5989
130	2504.02	Brooklyn/Curtis Bay/Hawkins Point	31250	37059	5809
131	1701.00	Downtown/Seton Hill	30197	35582	5385
132	2603.02	Belair-Edison	42727	50000	7273
133	2720.07	Glen-Fallstaff	35903	41728	5825
134	2501.01	Beechfield/Ten Hills/West Hills	52098	60463	8365
135	2502.04	Cherry Hill	12384	14349	1965
136	904.00	The Waverlies	26490	30685	4195
137	2003.00	Southwest Baltimore	19063	21953	2890
138	2708.01	Loch Raven	51415	58995	7580
139	105.00	Fells Point	86157	98194	12037
140	2502.03	Cherry Hill	29747	33902	4155
141	2005.00	Southwest Baltimore	30239	34306	4067
142	907.00	Midway/Coldstream	27330	30964	3634
143	2004.00	Southwest Baltimore	28324	31975	3651
144	2804.03	Beechfield/Ten Hills/West Hills	51435	57582	6147
145	2705.01	Harford/Echodale	59516	66604	7088
146	1503.00	Greater Rosemont	32289	35476	3187
147	2604.03	Claremont/Armistead	32373	35282	2909
148	2710.01	Greater Govans	33203	35822	2619
149	1901.00	Southwest Baltimore	22893	24559	1666
150	2701.02	Lauraville	54250	58020	3770
151	1001.00	Greenmount East	28929	30913	1984
152	2606.04	Southeastern	20710	21977	1267
153	1508.00	Forest Park/Walbrook	37199	39337	2138
154	1703.00	Upton/Druid Heights	17892	18912	1020
155	1506.00	Greater Rosemont	29631	31250	1619
156	301.00	Harbor East/Little Italy	21932	23083	1151

157	2804.01	Beechfield/Ten Hills/West Hills	44954	47192	2238
158	2719.00	Glen-Fallstaff	52308	54866	2558
159	2710.02	Greater Govans	38500	40281	1781
160	1504.00	Greater Mondawmin	32632	33702	1070
161	1507.02	Greater Mondawmin	48942	50150	1208
162	909.00	Greenmount East	19056	19524	468
163	2801.01	Glen-Fallstaff	46576	47610	1034
164	2718.01	Pimlico/Arlington/Hilltop	24802	25278	476
165	2706.00	Harford/Echodale	66914	67625	711
166	1513.00	Southern Park Heights	33579	33866	287
167	1308.05	Mount Washington/Coldspring	53984	53750	-234
168	2501.03	Morrell Park/Violetville	36957	36555	-402
169	1505.00	Greater Mondawmin	34609	33967	-642
170	1602.00	Sandtown-Winchester/Harlem Park	25417	24848	-569
171	2007.01	Allendale/Irvington/S. Hilton	34556	33592	-964
172	2503.03	Morrell Park/Violetville	42007	40230	-1777
173	2709.03	Northwood	54607	52083	-2524
174	1304.00	Penn North/Reservoir Hill	32500	30868	-1632
175	2703.02	Lauraville	68088	63929	-4159
176	2804.04	Allendale/Irvington/S. Hilton	43779	40946	-2833
177	2505.00	Brooklyn/Curtis Bay/Hawkins Point	32738	30526	-2212
178	2504.01	Brooklyn/Curtis Bay/Hawkins Point	34401	32019	-2382
179	2709.01	Northwood	51023	47260	-3763
180	2002.00	Southwest Baltimore	30061	27740	-2321
181	2604.02	Claremont/Armistead	42938	39000	-3938
182	2503.01	Westport/Mount Winans/Lakeland	27414	24877	-2537
183	908.00	Midway/Coldstream	35473	31250	-4223
184	702.00	Madison/East End	35667	31379	-4288
185	801.02	Belair-Edison	35594	30000	-5594
186	1604.00	Sandtown-Winchester/Harlem Park	27813	23036	-4777
187	1903.00	Southwest Baltimore	17237	14269	-2968
188	2502.05	Westport/Mount Winans/Lakeland	47738	38779	-8959
189	2720.03	Cross-Country/Cheswolde	70942	57543	-13399
190	2716.00	Southern Park Heights	43265	34832	-8433
191	805.00	Clifton-Berea	35223	28203	-7020
192	2102.00	Washington Village/Pigtown	46319	36661	-9658
193	2715.03	Greater Roland Park/Poplar Hill	79861	62500	-17361
194	2804.02	Edmondson Village	52784	41179	-11605
195	1603.00	Sandtown-Winchester/Harlem Park	22292	13963	-8329
196	1003.00	Unassigned--Jail	NA	NA	NA
197	1801.00	Poppleton/The Terraces/Hollins Market	18429	NA	NA
198	1802.00	Poppleton/The Terraces/Hollins Market	16585	NA	NA
199	2506.00	Brooklyn/Curtis Bay/Hawkins Point	NA	NA	NA

```
200 2603.03                 Claremont/Armistead  30192     NA          NA
    Diff_Pct_2020
1          227.56
2          133.33
3          123.18
4          116.80
5          100.94
6           97.71
7           95.16
8           94.48
9           94.40
10          93.98
11          92.56
12          89.91
13          88.07
14          82.74
15          82.72
16          78.11
17          74.53
18          74.05
19          73.09
20          71.88
21          70.79
22          69.66
23          69.29
24          69.09
25          68.72
26          67.55
27          65.38
28          63.48
29          61.13
30          60.17
31          59.16
32          58.22
33          58.07
34          57.70
35          57.16
36          56.69
37          56.68
38          56.31
39          55.18
40          55.01
41          54.40
```

42	54.21
43	54.01
44	53.80
45	52.98
46	52.47
47	52.28
48	52.17
49	50.73
50	49.38
51	49.14
52	47.66
53	47.43
54	47.12
55	46.28
56	45.71
57	45.71
58	45.40
59	45.30
60	44.99
61	44.95
62	44.67
63	44.14
64	44.06
65	43.87
66	42.84
67	42.82
68	42.80
69	41.91
70	41.85
71	41.81
72	41.08
73	40.52
74	39.87
75	39.72
76	38.78
77	37.66
78	37.46
79	36.89
80	36.85
81	36.79
82	36.25
83	35.58
84	35.52

85	35.48
86	35.33
87	35.22
88	34.42
89	34.36
90	34.03
91	33.53
92	33.45
93	33.31
94	31.79
95	31.66
96	31.35
97	31.07
98	30.85
99	29.93
100	29.92
101	28.90
102	28.87
103	28.72
104	28.72
105	28.52
106	28.24
107	27.49
108	27.07
109	27.01
110	26.83
111	25.74
112	25.51
113	25.33
114	25.13
115	23.77
116	23.64
117	23.34
118	23.07
119	22.38
120	22.04
121	21.46
122	21.15
123	20.99
124	20.96
125	20.73
126	20.64
127	20.01

128	19.78
129	18.59
130	18.59
131	17.83
132	17.02
133	16.22
134	16.06
135	15.87
136	15.84
137	15.16
138	14.74
139	13.97
140	13.97
141	13.45
142	13.30
143	12.89
144	11.95
145	11.91
146	9.87
147	8.99
148	7.89
149	7.28
150	6.95
151	6.86
152	6.12
153	5.75
154	5.70
155	5.46
156	5.25
157	4.98
158	4.89
159	4.63
160	3.28
161	2.47
162	2.46
163	2.22
164	1.92
165	1.06
166	0.85
167	-0.43
168	-1.09
169	-1.86
170	-2.24

```
171         -2.79
172         -4.23
173         -4.62
174         -5.02
175         -6.11
176         -6.47
177         -6.76
178         -6.92
179         -7.38
180         -7.72
181         -9.17
182         -9.25
183        -11.90
184        -12.02
185        -15.72
186        -17.18
187        -17.22
188        -18.77
189        -18.89
190        -19.49
191        -19.93
192        -20.85
193        -21.74
194        -21.99
195        -37.36
196            NA
197            NA
198            NA
199            NA
200            NA
```

So we know who the winners since the descending sort by `Diff_Pct_2020` shows us Census tracts by the highest percentage first. Now, which areas suffered the biggest declines in median income? We will copy the previous code and alter the arrange so it is ascending from the lowest value.

```
baltcity_income %>%
  select(Census, Neighborhood, x2010, x2020) %>%
    mutate(Diff_Income = (x2020-x2010)) %>%
    mutate(Diff_Pct_2020 = round((x2020-x2010)/x2010*100,2)) %>%
    arrange(Diff_Pct_2020)
```

	Census	Neighborhood	x2010	x2020	Diff_Income
1	1603.00	Sandtown-Winchester/Harlem Park	22292	13963	-8329
2	2804.02	Edmondson Village	52784	41179	-11605
3	2715.03	Greater Roland Park/Poplar Hill	79861	62500	-17361
4	2102.00	Washington Village/Pigtown	46319	36661	-9658
5	805.00	Clifton-Berea	35223	28203	-7020
6	2716.00	Southern Park Heights	43265	34832	-8433
7	2720.03	Cross-Country/Cheswolde	70942	57543	-13399
8	2502.05	Westport/Mount Winans/Lakeland	47738	38779	-8959
9	1903.00	Southwest Baltimore	17237	14269	-2968
10	1604.00	Sandtown-Winchester/Harlem Park	27813	23036	-4777
11	801.02	Belair-Edison	35594	30000	-5594
12	702.00	Madison/East End	35667	31379	-4288
13	908.00	Midway/Coldstream	35473	31250	-4223
14	2503.01	Westport/Mount Winans/Lakeland	27414	24877	-2537
15	2604.02	Claremont/Armistead	42938	39000	-3938
16	2002.00	Southwest Baltimore	30061	27740	-2321
17	2709.01	Northwood	51023	47260	-3763
18	2504.01	Brooklyn/Curtis Bay/Hawkins Point	34401	32019	-2382
19	2505.00	Brooklyn/Curtis Bay/Hawkins Point	32738	30526	-2212
20	2804.04	Allendale/Irvington/S. Hilton	43779	40946	-2833
21	2703.02	Lauraville	68088	63929	-4159
22	1304.00	Penn North/Reservoir Hill	32500	30868	-1632
23	2709.03	Northwood	54607	52083	-2524
24	2503.03	Morrell Park/Violetville	42007	40230	-1777
25	2007.01	Allendale/Irvington/S. Hilton	34556	33592	-964
26	1602.00	Sandtown-Winchester/Harlem Park	25417	24848	-569
27	1505.00	Greater Mondawmin	34609	33967	-642
28	2501.03	Morrell Park/Violetville	36957	36555	-402
29	1308.05	Mount Washington/Coldspring	53984	53750	-234
30	1513.00	Southern Park Heights	33579	33866	287
31	2706.00	Harford/Echodale	66914	67625	711
32	2718.01	Pimlico/Arlington/Hilltop	24802	25278	476
33	2801.01	Glen-Fallstaff	46576	47610	1034
34	909.00	Greenmount East	19056	19524	468
35	1507.02	Greater Mondawmin	48942	50150	1208
36	1504.00	Greater Mondawmin	32632	33702	1070
37	2710.02	Greater Govans	38500	40281	1781
38	2719.00	Glen-Fallstaff	52308	54866	2558
39	2804.01	Beechfield/Ten Hills/West Hills	44954	47192	2238
40	301.00	Harbor East/Little Italy	21932	23083	1151
41	1506.00	Greater Rosemont	29631	31250	1619
42	1703.00	Upton/Druid Heights	17892	18912	1020

43	1508.00	Forest Park/Walbrook	37199	39337	2138
44	2606.04	Southeastern	20710	21977	1267
45	1001.00	Greenmount East	28929	30913	1984
46	2701.02	Lauraville	54250	58020	3770
47	1901.00	Southwest Baltimore	22893	24559	1666
48	2710.01	Greater Govans	33203	35822	2619
49	2604.03	Claremont/Armistead	32373	35282	2909
50	1503.00	Greater Rosemont	32289	35476	3187
51	2705.01	Harford/Echodale	59516	66604	7088
52	2804.03	Beechfield/Ten Hills/West Hills	51435	57582	6147
53	2004.00	Southwest Baltimore	28324	31975	3651
54	907.00	Midway/Coldstream	27330	30964	3634
55	2005.00	Southwest Baltimore	30239	34306	4067
56	105.00	Fells Point	86157	98194	12037
57	2502.03	Cherry Hill	29747	33902	4155
58	2708.01	Loch Raven	51415	58995	7580
59	2003.00	Southwest Baltimore	19063	21953	2890
60	904.00	The Waverlies	26490	30685	4195
61	2502.04	Cherry Hill	12384	14349	1965
62	2501.01	Beechfield/Ten Hills/West Hills	52098	60463	8365
63	2720.07	Glen-Fallstaff	35903	41728	5825
64	2603.02	Belair-Edison	42727	50000	7273
65	1701.00	Downtown/Seton Hill	30197	35582	5385
66	2001.00	Southwest Baltimore	32214	38203	5989
67	2504.02	Brooklyn/Curtis Bay/Hawkins Point	31250	37059	5809
68	905.00	The Waverlies	36507	43727	7220
69	2709.02	Northwood	48555	58269	9714
70	1308.03	Medfield/Hampden/Woodberry/Remington	51548	62188	10640
71	2708.02	Loch Raven	46181	55753	9572
72	701.00	Madison/East End	37701	45602	7901
73	601.00	Patterson Park North & East	46822	56652	9830
74	1202.02	Greater Charles Village/Barclay	30179	36563	6384
75	1301.00	Penn North/Reservoir Hill	21271	25835	4564
76	1902.00	Southwest Baltimore	43214	52738	9524
77	902.00	Northwood	67500	82609	15109
78	2603.01	Belair-Edison	42347	52117	9770
79	2715.01	Mount Washington/Coldspring	88929	109688	20759
80	2601.02	Cedonia/Frankford	48783	60313	11530
81	2708.03	Loch Raven	42234	52272	10038
82	2704.01	Hamilton	51641	64620	12979
83	2802.00	Howard Park/West Arlington	45047	56458	11411
84	2717.00	Pimlico/Arlington/Hilltop	29069	36484	7415
85	2602.02	Cedonia/Frankford	31241	39282	8041

86	2805.00	Oldtown/Middle East	11278	14304	3026
87	2502.06	Morrell Park/Violetville	52674	66900	14226
88	1204.00	Greater Charles Village/Barclay	39375	50034	10659
89	2708.05	Chinquapin Park/Belvedere	46875	59763	12888
90	2718.02	Pimlico/Arlington/Hilltop	31392	40256	8864
91	2801.02	Howard Park/West Arlington	33938	43618	9680
92	1201.00	North Baltimore/Guilford/Homeland	56205	72349	16144
93	1308.04	Medfield/Hampden/Woodberry/Remington	48207	62054	13847
94	1608.01	Edmondson Village	41549	53543	11994
95	1511.00	Dorchester/Ashburton	45649	58843	13194
96	2711.01	North Baltimore/Guilford/Homeland	42994	55859	12865
97	2602.01	Cedonia/Frankford	33936	44093	10157
98	1510.00	Dorchester/Ashburton	35056	45870	10814
99	2006.00	Allendale/Irvington/S. Hilton	29813	39076	9263
100	2707.01	Harford/Echodale	28618	37591	8973
101	801.01	Belair-Edison	49688	65417	15729
102	806.00	Greenmount East	26726	35221	8495
103	2302.00	Inner Harbor/Federal Hill	80521	107344	26823
104	2720.04	Cross-Country/Cheswolde	49567	66148	16581
105	2606.05	Southeastern	33545	44794	11249
106	2705.02	Hamilton	53493	71694	18201
107	2702.00	Lauraville	60128	80788	20660
108	1303.00	Penn North/Reservoir Hill	32273	43380	11107
109	401.00	Downtown/Seton Hill	45948	62131	16183
110	2403.00	Inner Harbor/Federal Hill	89211	120729	31518
111	808.00	Oldtown/Middle East	35532	48137	12605
112	2201.00	Inner Harbor/Federal Hill	59259	80305	21046
113	1607.00	Greater Rosemont	29472	39959	10487
114	1507.01	Greater Mondawmin	38125	51944	13819
115	602.00	Patterson Park North & East	51403	70313	18910
116	2720.06	Glen-Fallstaff	24859	34019	9160
117	1509.00	Forest Park/Walbrook	37440	51250	13810
118	906.00	Midway/Coldstream	36408	50048	13640
119	1101.00	Midtown	38797	53407	14610
120	2608.00	Highlandtown	35200	48850	13650
121	2707.03	Harford/Echodale	63521	88750	25229
122	2713.00	Greater Roland Park/Poplar Hill	111250	155605	44355
123	2803.01	Dickeyville/Franklintown	33707	47366	13659
124	2701.01	Lauraville	57326	80875	23549
125	2602.03	Cedonia/Frankford	32615	46250	13635
126	1606.00	Greater Rosemont	27128	38482	11354
127	803.01	Clifton-Berea	32024	45446	13422
128	1402.00	Upton/Druid Heights	16213	23152	6939

129	1601.00	Sandtown-Winchester/Harlem Park	19583	27969	8386
130	1608.02	Edmondson Village	29355	41932	12577
131	1203.00	Greater Charles Village/Barclay	38281	55074	16793
132	1702.00	Upton/Druid Heights	9412	13559	4147
133	2402.00	Inner Harbor/Federal Hill	92500	133333	40833
134	804.00	Clifton-Berea	23893	34567	10674
135	903.00	The Waverlies	43824	63523	19699
136	2708.04	Chinquapin Park/Belvedere	41094	59583	18489
137	2707.02	Harford/Echodale	39205	56963	17758
138	901.00	Greater Govans	36444	52988	16544
139	2501.02	Allendale/Irvington/S. Hilton	36960	53854	16894
140	2720.05	Cross-Country/Cheswolde	49757	72500	22743
141	2712.00	North Baltimore/Guilford/Homeland	133548	195353	61805
142	1205.00	Midtown	37500	55170	17670
143	2601.01	Cedonia/Frankford	46789	68983	22194
144	2703.01	Lauraville	54623	80655	26032
145	1501.00	Sandtown-Winchester/Harlem Park	18097	26989	8892
146	1307.00	Medfield/Hampden/Woodberry/Remington	48634	72648	24014
147	1202.01	Greater Charles Village/Barclay	54000	81393	27393
148	604.00	Oldtown/Middle East	35139	53472	18333
149	2604.04	Orangeville/East Highlandtown	35708	54375	18667
150	2008.00	Allendale/Irvington/S. Hilton	32711	49875	17164
151	2605.01	Orangeville/East Highlandtown	42298	64706	22408
152	302.00	Harbor East/Little Italy	63885	98254	34369
153	1605.00	Greater Rosemont	24556	37819	13263
154	1206.00	Greater Charles Village/Barclay	23488	36221	12733
155	2714.00	Greater Roland Park/Poplar Hill	97891	151146	53255
156	2604.01	Claremont/Armistead	31392	48661	17269
157	1207.00	Medfield/Hampden/Woodberry/Remington	39730	61653	21923
158	2101.00	Washington Village/Pigtown	48857	76368	27511
159	2609.00	Highlandtown	68571	107438	38867
160	603.00	Patterson Park North & East	53813	84318	30505
161	2611.00	Highlandtown	83207	130769	47562
162	2301.00	Inner Harbor/Federal Hill	62222	98125	35903
163	201.00	Fells Point	60439	95536	35097
164	802.00	Clifton-Berea	24200	38289	14089
165	2704.02	Hamilton	53125	84556	31431
166	2007.02	Allendale/Irvington/S. Hilton	27305	43734	16429
167	1502.00	Sandtown-Winchester/Harlem Park	25224	40644	15420
168	104.00	Canton	70313	114946	44633
169	2803.02	Forest Park/Walbrook	33864	56004	22140
170	2610.00	Patterson Park North & East	36343	60893	24550
171	2404.00	South Baltimore	72122	121685	49563

172	803.02	Clifton-Berea	24423	41298	16875	
173	1803.00	Poppleton/The Terraces/Hollins Market	37879	64125	26246	
174	101.00	Canton	75938	128839	52901	
175	1306.00	Medfield/Hampden/Woodberry/Remington	53654	91635	37981	
176	203.00	Fells Point	59236	101815	42579	
177	2401.00	South Baltimore	87619	151659	64040	
178	1401.00	Midtown	32917	57292	24375	
179	2303.00	South Baltimore	53194	92841	39647	
180	2502.07	Cherry Hill	23310	41518	18208	
181	1512.00	Southern Park Heights	12386	22632	10246	
182	202.00	Fells Point	47131	86125	38994	
183	704.00	Oldtown/Middle East	14527	27321	12794	
184	1302.00	Penn North/Reservoir Hill	30255	57457	27202	
185	1102.00	Midtown	32563	62703	30140	
186	1403.00	Upton/Druid Heights	21607	41913	20306	
187	1002.00	Oldtown/Middle East	9862	19172	9310	
188	103.00	Canton	77841	151389	73548	
189	1308.06	Medfield/Hampden/Woodberry/Remington	48750	95142	46392	
190	807.00	Greenmount East	20679	40885	20206	
191	2711.02	North Baltimore/Guilford/Homeland	99300	199531	100231	
192	2607.00	Orangeville/East Highlandtown	36901	80000	43099	
193	102.00	Patterson Park North & East	58409	130357	71948	
194	703.00	Madison/East End	15000	35000	20000	
195	402.00	Downtown/Seton Hill	13229	43333	30104	
196	1003.00	Unassigned--Jail	NA	NA	NA	
197	1801.00	Poppleton/The Terraces/Hollins Market	18429	NA	NA	
198	1802.00	Poppleton/The Terraces/Hollins Market	16585	NA	NA	
199	2506.00	Brooklyn/Curtis Bay/Hawkins Point	NA	NA	NA	
200	2603.03	Claremont/Armistead	30192	NA	NA	

	Diff_Pct_2020
1	-37.36
2	-21.99
3	-21.74
4	-20.85
5	-19.93
6	-19.49
7	-18.89
8	-18.77
9	-17.22
10	-17.18
11	-15.72
12	-12.02
13	-11.90

14	-9.25
15	-9.17
16	-7.72
17	-7.38
18	-6.92
19	-6.76
20	-6.47
21	-6.11
22	-5.02
23	-4.62
24	-4.23
25	-2.79
26	-2.24
27	-1.86
28	-1.09
29	-0.43
30	0.85
31	1.06
32	1.92
33	2.22
34	2.46
35	2.47
36	3.28
37	4.63
38	4.89
39	4.98
40	5.25
41	5.46
42	5.70
43	5.75
44	6.12
45	6.86
46	6.95
47	7.28
48	7.89
49	8.99
50	9.87
51	11.91
52	11.95
53	12.89
54	13.30
55	13.45
56	13.97

57	13.97
58	14.74
59	15.16
60	15.84
61	15.87
62	16.06
63	16.22
64	17.02
65	17.83
66	18.59
67	18.59
68	19.78
69	20.01
70	20.64
71	20.73
72	20.96
73	20.99
74	21.15
75	21.46
76	22.04
77	22.38
78	23.07
79	23.34
80	23.64
81	23.77
82	25.13
83	25.33
84	25.51
85	25.74
86	26.83
87	27.01
88	27.07
89	27.49
90	28.24
91	28.52
92	28.72
93	28.72
94	28.87
95	28.90
96	29.92
97	29.93
98	30.85
99	31.07

100	31.35
101	31.66
102	31.79
103	33.31
104	33.45
105	33.53
106	34.03
107	34.36
108	34.42
109	35.22
110	35.33
111	35.48
112	35.52
113	35.58
114	36.25
115	36.79
116	36.85
117	36.89
118	37.46
119	37.66
120	38.78
121	39.72
122	39.87
123	40.52
124	41.08
125	41.81
126	41.85
127	41.91
128	42.80
129	42.82
130	42.84
131	43.87
132	44.06
133	44.14
134	44.67
135	44.95
136	44.99
137	45.30
138	45.40
139	45.71
140	45.71
141	46.28
142	47.12

143	47.43
144	47.66
145	49.14
146	49.38
147	50.73
148	52.17
149	52.28
150	52.47
151	52.98
152	53.80
153	54.01
154	54.21
155	54.40
156	55.01
157	55.18
158	56.31
159	56.68
160	56.69
161	57.16
162	57.70
163	58.07
164	58.22
165	59.16
166	60.17
167	61.13
168	63.48
169	65.38
170	67.55
171	68.72
172	69.09
173	69.29
174	69.66
175	70.79
176	71.88
177	73.09
178	74.05
179	74.53
180	78.11
181	82.72
182	82.74
183	88.07
184	89.91
185	92.56

```
186         93.98
187         94.40
188         94.48
189         95.16
190         97.71
191        100.94
192        116.80
193        123.18
194        133.33
195        227.56
196            NA
197            NA
198            NA
199            NA
200            NA
```

15.0.1 Using mutate to clean data

For this example, we'll examine a dataset of loans small businesses obtained to stay afloat during the Covid-19 pandemic. It's called the PPP loan database which stands for Paycheck Protection Program loans.

```
#ppp_applications_md_small <- read_csv("assets/data/ppp_applications_md.csv") %>%
#  select(name, amount, city, servicing_lender_state)

#write_csv(ppp_applications_md_small,"assets/data/ppp_applications_md_small.csv")

maryland_ppp <- read_csv("assets/data/ppp_applications_md_small.csv")
```

```
Rows: 195869 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (3): name, city, servicing_lender_state
dbl (1): amount

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.
```

```
#working on this with your laptop, uncomment and use this code below
#maryland_ppp <- read.csv("ppp_applications_md.csv")
```

Take a look at the `city` column in our data.

```
#View(maryland_ppp)
```

You'll notice that there's a mix of styles: "Baltimore" and "BALTIMORE" for example. R will think those are two different cities, and that will mean that any aggregates we create based on city won't be accurate.

So how can we fix that? Mutate is not just for math! And a function called `str_to_upper` that will convert a character column into all uppercase.

```
maryland_ppp %>%
  mutate(
    upper_city = str_to_upper(city)
) %>%
   select(city, upper_city, name, amount)
```

```
# A tibble: 195,869 x 4
   city             upper_city       name                                amount
   <chr>            <chr>            <chr>                                <dbl>
 1 Columbia         COLUMBIA         BAYWOOD HOTELS INC.                 2   e6
 2 Lanham           LANHAM           PARTNERS ELECTRIC SERVICE INC.      1.09e6
 3 Bethesda         BETHESDA         RP3 LLC                             7.88e5
 4 Chevy Chase      CHEVY CHASE      SUSHI KO CHEVY CHASE LLC.           6.56e5
 5 Capitol Heights  CAPITOL HEIGHTS  VILLAGE ACADEMY OF MARYLAND         4.70e5
 6 Brentwood        BRENTWOOD        GOSHEN HOUSE TRADING LLC            3.46e5
 7 Tracys Landing   TRACYS LANDING   WEAVER BOAT WORKS                   3.31e5
 8 Essex            ESSEX            BOBS OVERHEAD DOOR REPAIR AND SERVI~ 3.18e5
 9 Annapolis        ANNAPOLIS        PARENTS FOR MONTESSORI EDUCATION INC 2.84e5
10 Prince Frederick PRINCE FREDERICK HARVEST PRINCE FREDERICK LLC        2.42e5
# ... with 195,859 more rows
```

Notice we kept the original data in `city` and transformed it in a new column, `upper_city`. Always a good practice to keep your original data intact in case you make a coding mistake or need to change the original for reference.

We could do the same thing with the `address` column in order to standardize that for analysis, too.

Think of how easy this is compared to a similar data cleaning lesson in Google Sheets.

15.0.2 A more powerful use

Mutate is even more useful when combined with some additional functions. Let's say you want to know if the servicing lender is located in Maryland or outside the state. There are three possible answers:

1. The lender is in Maryland
2. The lender is outside Maryland
3. The data doesn't tell us (`servicing_lender_state` is blank or NA)

We can create a new column that accounts for these possibilities and populate it using mutate and `case_when`, which is like an if/else statement but for more than two options.

```
maryland_with_in_out <- maryland_ppp %>%
  mutate(
    in_out = case_when(
        servicing_lender_state == 'NA' ~ "NA",
        servicing_lender_state == 'MD' ~ "IN",
        servicing_lender_state != 'MD' ~ "OUT"
      )
  )

maryland_with_in_out %>%
  select(name, amount, servicing_lender_state, in_out)
```

```
# A tibble: 195,869 x 4
   name                                    amount servicing_lender_~1 in_out
   <chr>                                    <dbl> <chr>               <chr>
 1 BAYWOOD HOTELS INC.                    2000000  MD                  IN
 2 PARTNERS ELECTRIC SERVICE INC.         1086257  NY                  OUT
 3 RP3 LLC                                 787500  VA                  OUT
 4 SUSHI KO CHEVY CHASE LLC.               655732  NY                  OUT
 5 VILLAGE ACADEMY OF MARYLAND             469800  WV                  OUT
 6 GOSHEN HOUSE TRADING LLC                346500  NJ                  OUT
 7 WEAVER BOAT WORKS                       330700  MD                  IN
 8 BOBS OVERHEAD DOOR REPAIR AND SERVICE INC  317504  MD                  IN
 9 PARENTS FOR MONTESSORI EDUCATION INC    284185  MD                  IN
10 HARVEST PRINCE FREDERICK LLC            242094. MO                  OUT
# ... with 195,859 more rows, and abbreviated variable name
#   1: servicing_lender_state
```

We can then use our new `in_out` column in group_by statements to make summarizing easier.

In this case there are no Maryland loans where `servicing_lender_state` has a value of 'NA', but you should never assume that will be the case for a dataset. If you know that the only options are the lender is in Maryland or is outside it, you can rewrite the previous code as an if/else statement:

```
maryland_with_in_out <- maryland_ppp %>%
  mutate(
    in_out = if_else(
        servicing_lender_state == 'MD', "IN", "OUT"
      )
  )

maryland_with_in_out %>%
  select(name, amount, servicing_lender_state, in_out)
```

```
# A tibble: 195,869 x 4
   name                                      amount servicing_lender_~1 in_out
   <chr>                                      <dbl> <chr>               <chr>
 1 BAYWOOD HOTELS INC.                      2000000  MD                  IN
 2 PARTNERS ELECTRIC SERVICE INC.           1086257  NY                  OUT
 3 RP3 LLC                                   787500  VA                  OUT
 4 SUSHI KO CHEVY CHASE LLC.                 655732  NY                  OUT
 5 VILLAGE ACADEMY OF MARYLAND               469800  WV                  OUT
 6 GOSHEN HOUSE TRADING LLC                  346500  NJ                  OUT
 7 WEAVER BOAT WORKS                         330700  MD                  IN
 8 BOBS OVERHEAD DOOR REPAIR AND SERVICE INC 317504  MD                  IN
 9 PARENTS FOR MONTESSORI EDUCATION INC      284185  MD                  IN
10 HARVEST PRINCE FREDERICK LLC              242094. MO                  OUT
# ... with 195,859 more rows, and abbreviated variable name
#   1: servicing_lender_state
```

Mutate is an essential tool to make your data more useful and allows you to ask more questions.

16 Pre-Lab Questions:

Based on the skills you have learned above, use this dataset of Maryland cities 'assets/data/city_md_income.csv' (it is included in your prelab folder)

And answer the following questions:

Question #1:

1) Calculate the difference in median income from 2010 to 2020 for all places listed. Calculate the percentage change.

Which city had the highest absolute change? Which had the highest percentage change?

Produce a table with the top 25 places ranked by the highest percentage change.

Produce a second table with the top 25 places ranked by the lowest or negative percentage change.

Question #2: Answer this question in English: Write in Elms

2) Building on the previous lesson on filtering, describe how you would determine the median value of the percentage change and produce a table with the 10 neighborhoods with the highest change and the 10 with the lowest change.

17 Visualizing your data for reporting

17.0.1 Why to use visualization

Before you launch into trying to chart or map your data, take a minute to think about the many roles that static and interactive graphic elements play in your journalism.

Sometimes, you can help yourself and your story by just creating a quick chart, which helps you see patterns in the data that wouldn't otherwise surface. In the reporting phase, visualizations can:

- Help you identify themes and questions for the rest of your reporting
- Identify outliers – good stories, or perhaps errors, in your data
- Help you find typical examples
- Show you holes in your reporting

Visualizations also play multiple roles in publishing:

- Illustrate a point made in a story in a more compelling way
- Remove unnecessarily technical information from prose
- Allow exploration, provide transparency about your reporting to readers

Visualizing data is becoming a much greater part of journalism. Large news organizations are creating graphics desks that create complex visuals with data to inform the public about important events. To do it well is a course on its own. And not every story needs a feat of programming and art.

Good news: one of the best libraries for visualizing data is in the tidyverse and it's pretty simple to make simple charts quickly with just a little bit of code. It's called ggplot2.

Let's revisit some data we've used in the past and turn it into charts. First, let's load libraries. When we load the tidyverse, we get ggplot2.

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
```

```
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ---------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

17.0.2 311 Call Data

The dataset we'll use involves the 311 calls for service to the San Francisco Police Department. The 311 calls are for general non-emergency issues ranging from blocked sidewalks to graffiti to homeless concerns. Details are here If you're curious, here is the original dataset: 3,048,797 records

This tutorial uses a subset of this data [1]

```
SF <- read_csv("assets/data/SF_311.csv")
```

Clean column names, Process dates

The imported data recognized the date as a character text, which means you cannot run an analysis to group by years, months or days. The code below did four things - 1: `janitor::clean_names` normalized the column names - 2: `as.Date(SF$call_date)` converted text into date fields - 3: `lubridate::mdy(SF$call_date)` converted the date into a format to extract the dates in detail using lubridate, an important software library for date analysis - 4: `lubridate::year(SF$call_date2)` extracted the year into a new column.

```
#This cleans column names
SF <- janitor::clean_names(SF)
#This processes dates for analysis
SF$call_date2 <- as.Date(SF$call_date)
SF$call_date2 <- lubridate::mdy(SF$call_date)
#This creates a new column for year
SF$year <- lubridate::year(SF$call_date2)
```

Check our data. Note that `call_date2` is a `<date>` field.

```
glimpse(SF)
```

```
Rows: 157,237
```

[1] The Calls for Service were filtered as follows: CONTAINS homeless, 915, 919, 920: Downloaded 157,237 records 3/31/16 to 11/30/2019. This is 5.1% of all calls in the broader database. File renamed to: SF_311.xlsx

```
Columns: 16
$ crime_id                  <dbl> 190040497, 190200644, 190213959, 190271011, 1~
$ original_crime_type_name <chr> "919", "Homeless Complaint", "Homeless Compla~
$ report_date               <chr> "1/4/19", "1/20/19", "1/21/19", "1/27/19", "1~
$ call_date                 <chr> "1/4/19", "1/20/19", "1/21/19", "1/27/19", "1~
$ offense_date              <chr> "1/4/19", "1/20/19", "1/21/19", "1/27/19", "1~
$ call_time                 <time> 06:58:00, 06:19:00, 22:42:00, 09:28:00, 13:2~
$ call_date_time            <chr> "1/4/19 6:58", "1/20/19 6:19", "1/21/19 22:42~
$ disposition               <chr> "HAN", "HAN", "ADV", "HAN", "HAN", "HAN", "HA~
$ address                   <chr> "400 Block Of Jones St", "8th And Market", "M~
$ city                      <chr> "San Francisco", NA, "San Francisco", "San Fr~
$ state                     <chr> "CA", "CA", "CA", "CA", "CA", "CA", "CA", "CA~
$ agency_id                 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ address_type              <chr> "Premise Address", "Geo-Override", "Intersect~
$ common_location           <chr> NA, NA, NA, NA, NA, NA, "Jose Coronado Plgd, ~
$ call_date2                <date> 2019-01-04, 2019-01-20, 2019-01-21, 2019-01-~
$ year                      <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 201~
```

17.0.3 Bar charts

The first kind of chart we'll create is a simple bar chart.

It's a chart designed to show differences between things – the magnitude of one thing, compared to the next thing, and the next, and the next.

So if we have thing, like a county, or a state, or a group name, and then a count of that group, we can make a bar chart.

So what does the chart the years of the 311 call data look like?

First, we'll do a basic count of the years. There are just four years in the dataset, from 2016-2019. We built a simple table called `years`.

```
years <- SF %>%
   count(year) %>%
   group_by(year)

years
```

```
# A tibble: 4 x 2
# Groups:   year [4]
   year     n
  <dbl> <int>
```

```
1  2016 37802
2  2017 49523
3  2018 36057
4  2019 33855
```

Now let's create a bar chart using ggplot.

We need to tell ggplot what kind of chart to make.

In ggplot, we work with two key concepts called geometries (abbreivated frequently as **geom**) and asthetics (abbreviated as **aes**).

Geometries are the shape of the data: line charts, bar charts, scatterplots, histograms, pie charts, etc.

Asesthetics help ggplot know what component of our data to visualize – why we'll visualize values from one column instead of another.

In a bar chart, we first pass in the data to the geometry, then set the aesthetic.

In the codeblock below, we've added a new function, geom_bar().

Using geom_bar() – as opposed to geom_line() – says we're making a bar chart.

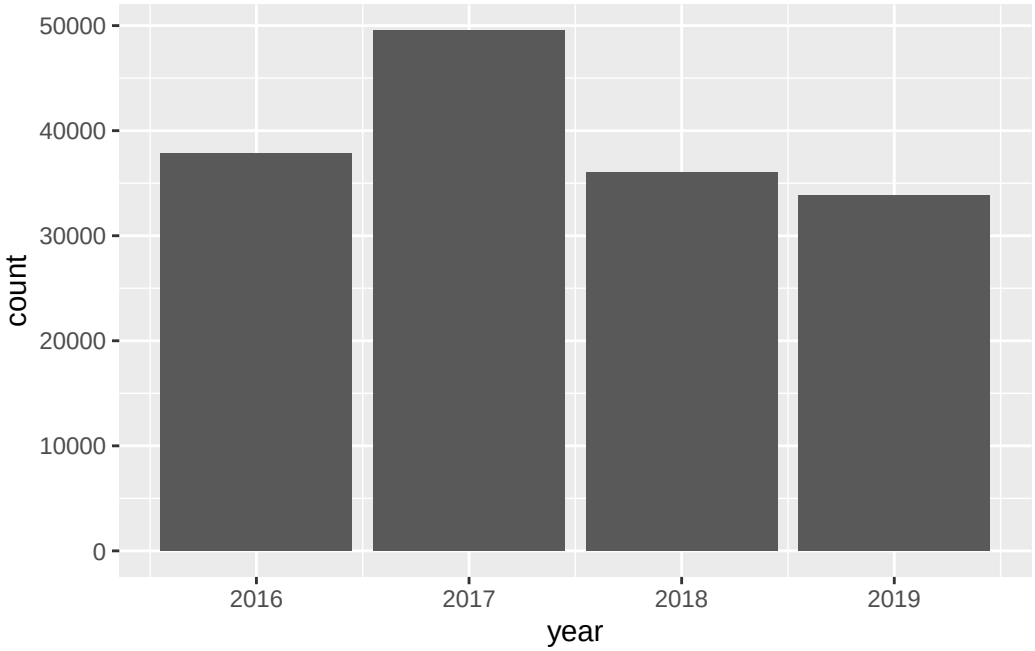
Inside of that function, the asthetic, aes, says which columns to use in drawing the chart.

We're setting the values on the x axis (horizontal) to be the `year`. We set weight to `n` or the total calls per year, and it uses that value to "weight" or set the height of each bar.

One quirk here with ggplot.

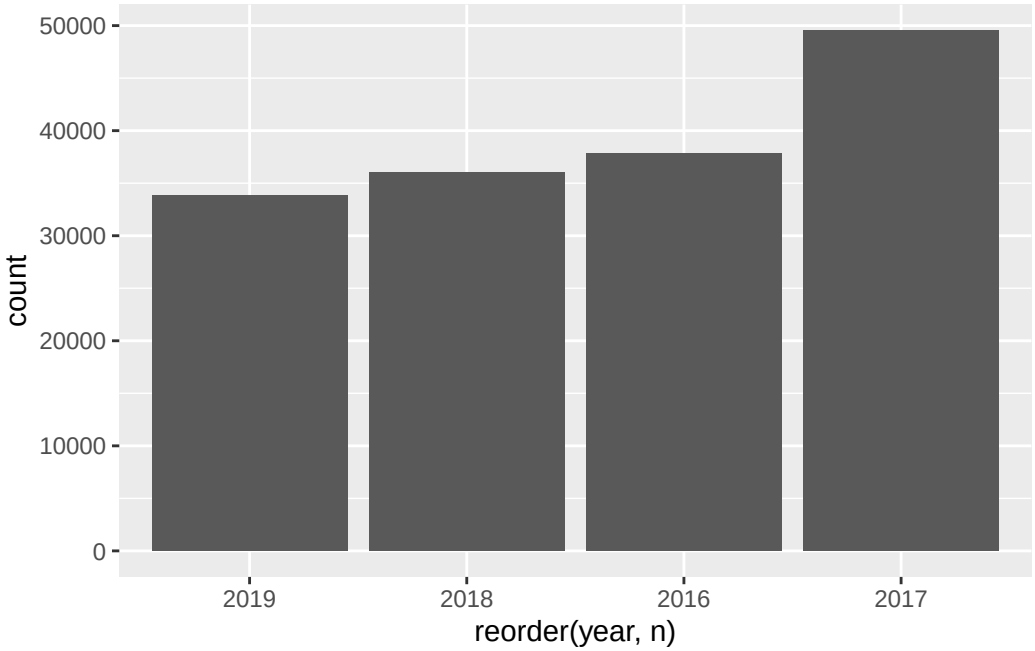
After we've invoked the ggplot() function, you'll notice we're using a + symbol. It means the same thing as %>% – "and then do this". It's just a quirk of ggplot() that after you invoke the ggplot() function, you use + instead of %>%. It makes no sense to me either, just something to live with.

```
years %>%
  ggplot() +
  geom_bar(aes(x=year, weight=n))
```



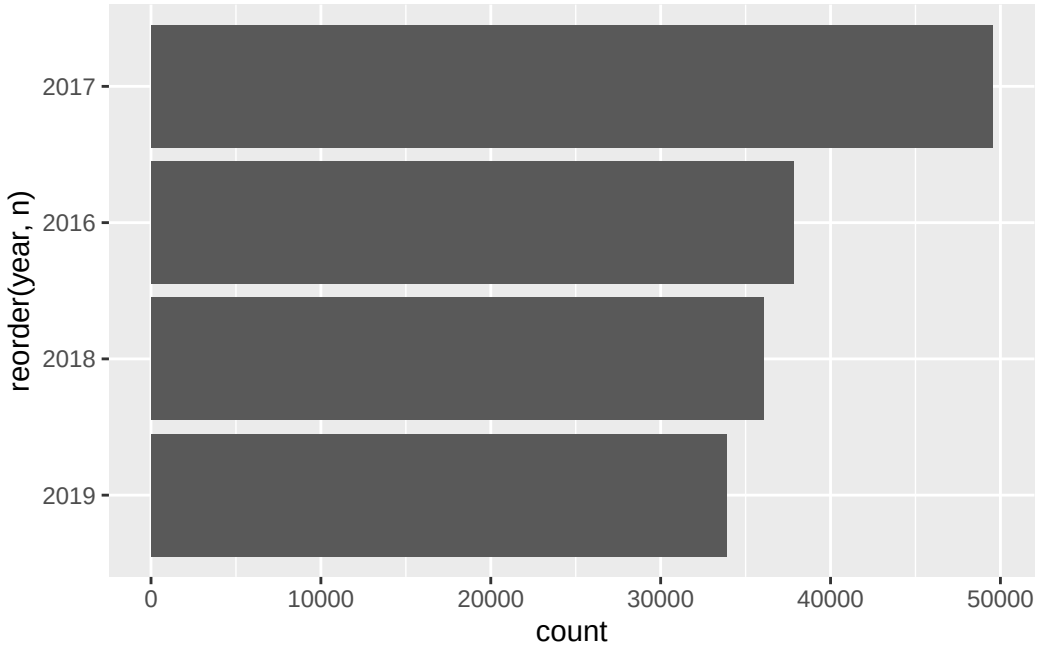
This is a very basic chart. But it's hard to derive much meaning from this chart. We can order the bars from highest to lowest by count of calls per year. We can fix that by using the reorder() function to do just that:

```
years %>%
  ggplot() +
  geom_bar(aes(x=reorder(year, n), weight=n))
```



This gives us another view of the data. But the bottom labels can be flipped if we wanted to avoid overlapping names. We can fix that by flipping it from a vertical bar chart (also called a column chart) to a horizontal one. coord_flip() does that for you.

```
years %>%
  ggplot() +
  geom_bar(aes(x=reorder(year, n), weight=n)) +
  coord_flip()
```



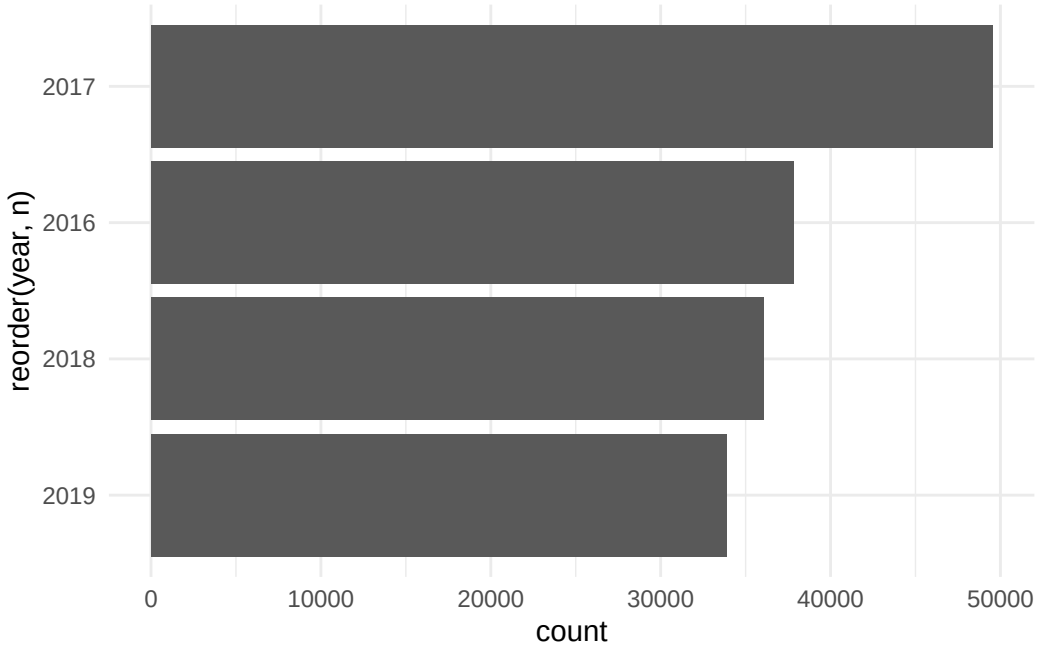
Notice how this clearly shows a trend about the decline in 311 calls per year.

We're mainly going to use these charts to help us in reporting, so style isn't that important.

We can pretty up these charts for publication with some more code. To style the chart, we can change or even modify the "theme", a kind of skin that makes the chart look better.

Here I'm changing the theme slightly to remove the gray background with one of ggplot's built in themes, theme_minimal()

```
years %>%
  ggplot() +
  geom_bar(aes(x=reorder(year, n), weight=n)) +
  coord_flip() +
  theme_minimal()
```



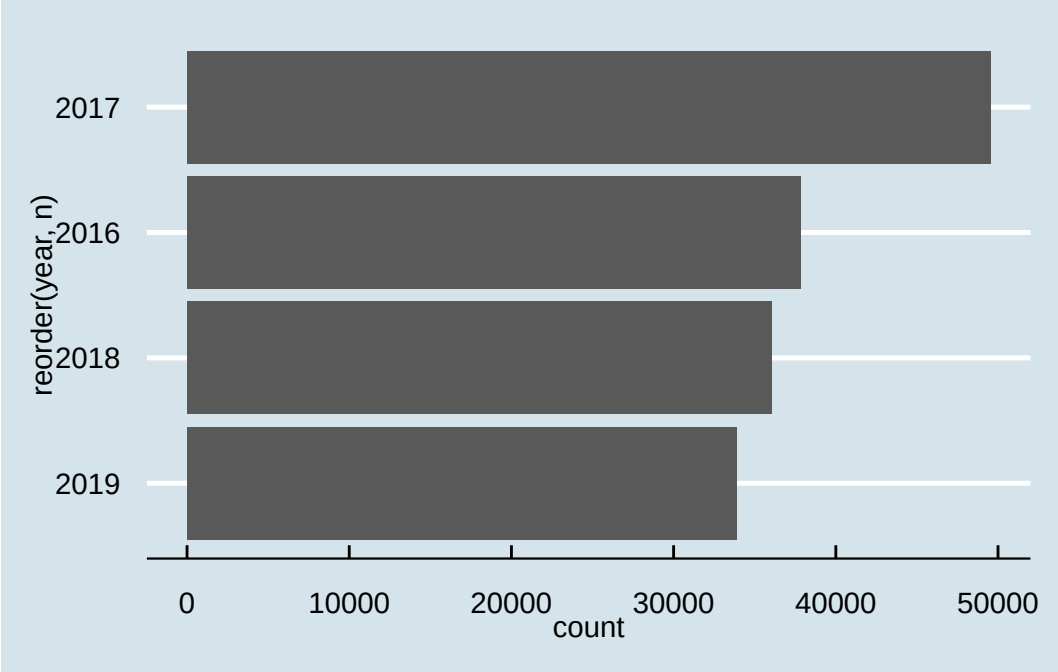
The ggplot universe is pretty big, and lots of people have made and released cool themes for you to use. Want to make your graphics look kind of like The Economist's graphics? There's a theme for that.

First, you have to install and load a package that contains lots of extra themes, called ggthemes.

```
#install.packages('ggthemes')
library(ggthemes)
```

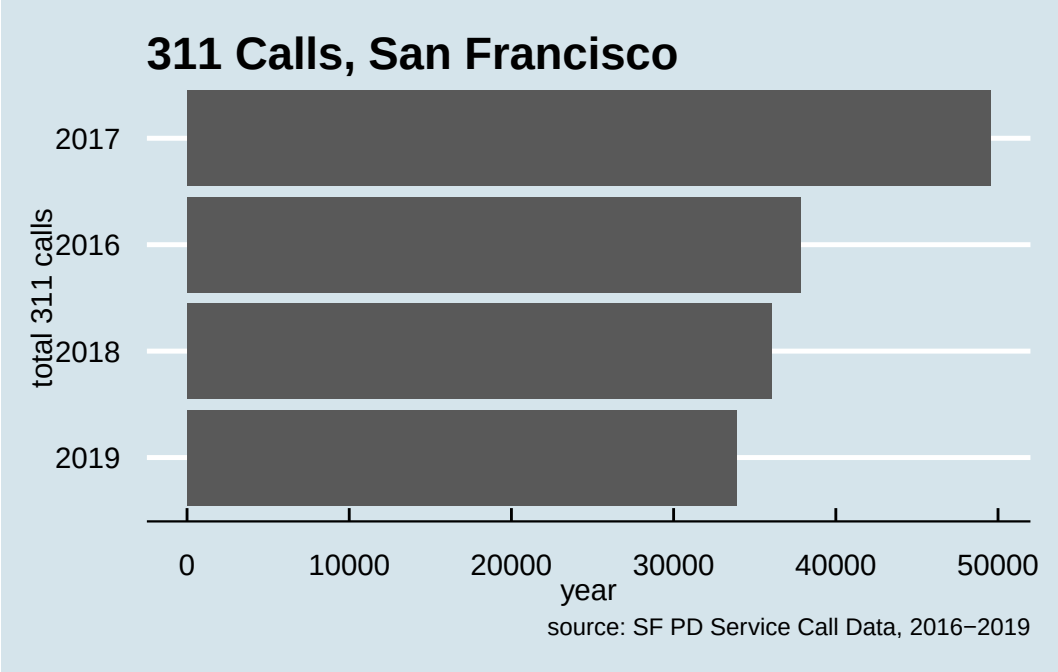
And now we'll apply the economist theme from that package with theme_economist()

```
years %>%
  ggplot() +
  geom_bar(aes(x=reorder(year, n), weight=n)) +
  coord_flip() +
  theme_minimal() +
  theme_economist()
```



Those axis titles are kind of a mess. Let's change "count" on the x axis to "total 311 calls" and change "reorder(year, n)" to "year". And while we're at it, let's add a basic title, "311 Calls for Service By Year, San Francisco" and a source as a caption. We'll use a new function, labs(), which is short for labels.

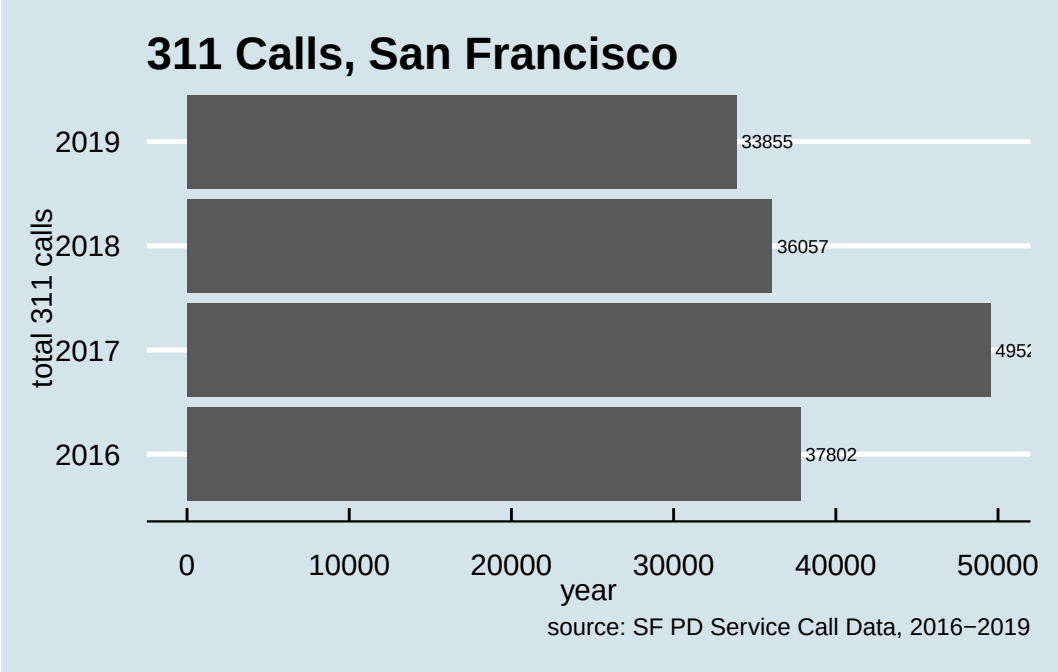
```
years %>%
  ggplot() +
  geom_bar(aes(x=reorder(year, n), weight=n)) +
  coord_flip() +
  theme_minimal() +
  theme_economist() +
  labs(
    title="311 Calls, San Francisco",
    x = "total 311 calls",
    y = "year",
    caption = "source: SF PD Service Call Data, 2016-2019")
```



Not super pretty, but good enough to show an editor to help them understand the conclusions you reached with your data analysis.

With geom_text, we can add numbers to the bars

```
years %>%
  ggplot(aes(x=year, y=n, weight=n)) +
  coord_flip() +
  theme_economist() +
  geom_col()+
  geom_text(aes(label=n), hjust = -.1, size = 2.5) +
    labs(
    title="311 Calls, San Francisco",
    x = "total 311 calls",
    y = "year",
    caption = "source: SF PD Service Call Data, 2016-2019")
```

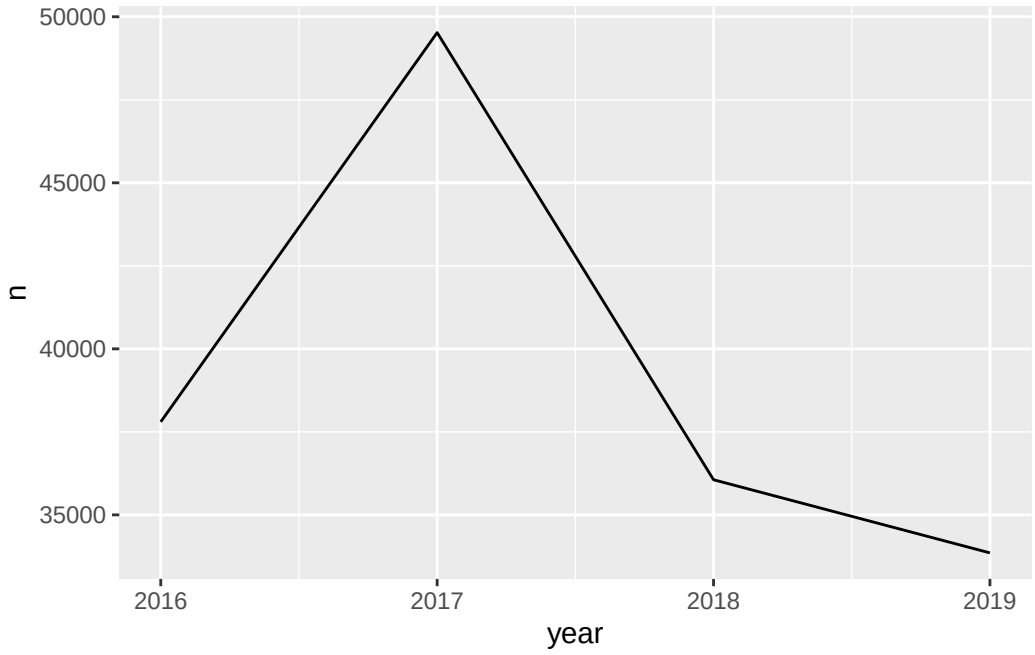


17.1 Line charts

Let's look at how to make another common chart type that will help you understand patterns in your data.

Line charts can show change over time. It works much the same as a bar chart, code wise, but instead of a weight, it uses a y. We'll put the year on the x axis and count of loans, or `n`, on the y axis.

```
years %>%
  ggplot() +
  geom_line(aes(x=year, y=n))
```



It's not super pretty, but there's an obvious pattern, a bog peak in 2017.

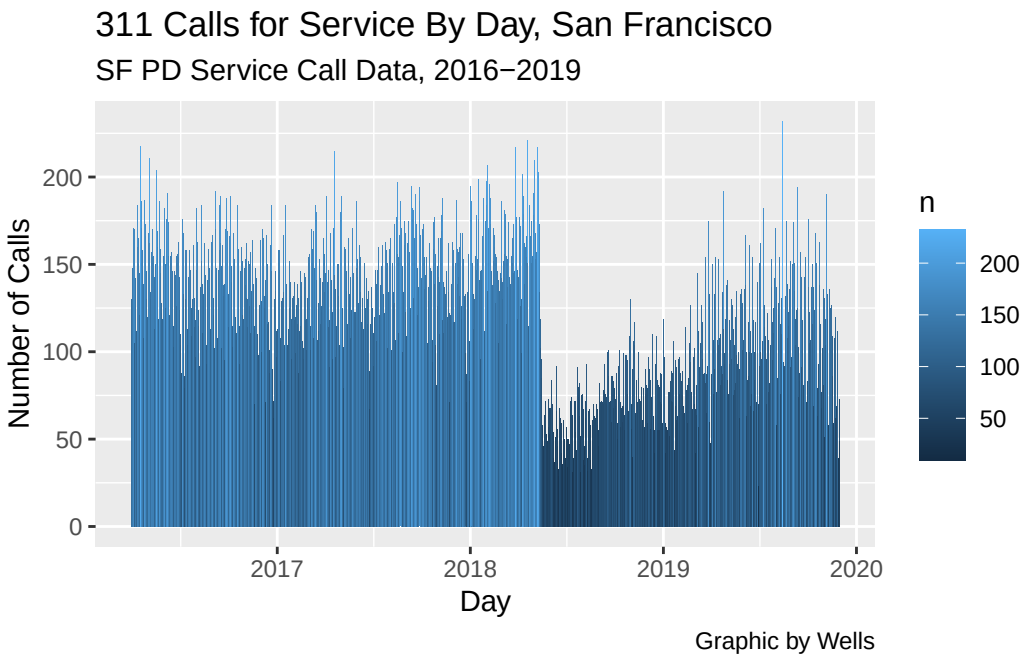
Let's build another table with some detail by day.

```
days <- SF %>%
  count(call_date2)
days
```

```
# A tibble: 1,340 x 2
   call_date2     n
   <date>     <int>
 1 2016-03-31    10
 2 2016-04-01   130
 3 2016-04-02   119
 4 2016-04-03   148
 5 2016-04-04   171
 6 2016-04-05   170
 7 2016-04-06   170
 8 2016-04-07   105
 9 2016-04-08   142
10 2016-04-09    95
# ... with 1,330 more rows
```

Here is the chart by day

```
days %>%
  ggplot(aes(x = call_date2, y = n, fill = n)) +
  geom_bar(stat = "identity") +
  labs(title = "311 Calls for Service By Day, San Francisco",
       subtitle = "SF PD Service Call Data, 2016-2019",
       caption = "Graphic by Wells",
       y="Number of Calls",
       x="Day")
```



By charting this, we can quickly see a pattern that can help guide our reporting: why was there such a huge drop in mid-2018?

Let's build a line chart for just a few variables based on year and disposition: Citations of people, Cancelled Calls and Admonished people on the scene.

```
dispo <- SF %>%
  filter(disposition == c("CAN", "CIT", "ADM"))
```

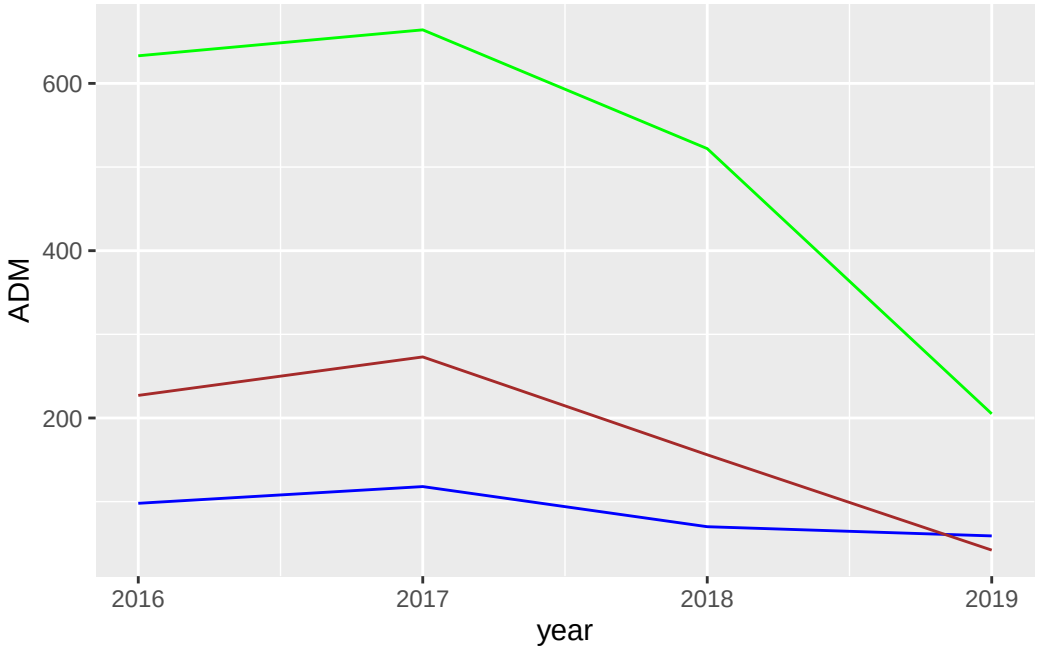
```
Warning in disposition == c("CAN", "CIT", "ADM"): longer object length is not a
multiple of shorter object length
```

```
dispo <- dispo %>%
  select(year, disposition) %>%
  group_by(year) %>%
  count(disposition)

dispo_pivot <- dispo %>%
  pivot_wider(names_from = "disposition", values_from = "n") %>%
  as.data.frame()
```

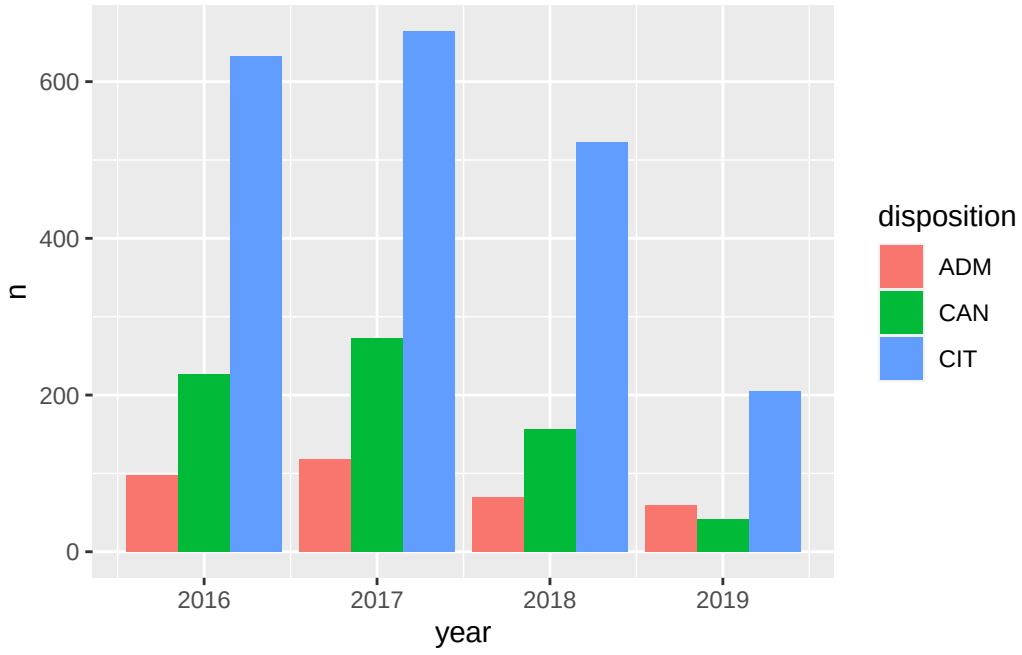
With this dataframe, we will now use separate lines per column to graph the results

```
ggplot(dispo_pivot, aes(x=year)) +
  geom_line(aes(y=ADM), color = "blue") +
  geom_line(aes(y=CAN), color = "brown") +
  geom_line(aes(y=CIT), color = "green")
```



Or you can plot them with side-by-side columns. Use dispo and plot the columns by type of disposition. Each individual variable in disposition is assigned its own column and color. The years are plotted on the x axis.

```
ggplot(dispo, aes(year, n, fill = disposition)) +
  geom_col(position = "dodge")
```



We're just scratching the surface of what ggplot can do, and chart types. There's so much more you can do, so many other chart types you can make. But the basics we've shown here will get you started.

18 Pre-Lab Question 1

Which year had the most arrests? The arrest code is ARR. Create a new dataframe that filters for arrests and totals them by year. Call this arrest.

```
#your code here
```

Then create a bar chart that colors the bars by arrest total. Include a headline and caption

```
#your code here
```

19 Pre-Lab Question 2

What are the most common resolutions for 311 calls? Build a table called all_dispo that counts all of the dispositions Filter to the top 12 most common dispositions.

And here are the disposition codes, in case you're curious: https://github.com/profrobwells/Guest_Lectures/blo

```
# your code here
```

Create a bar chart of all_dispo and include a headline and caption

```
# your code here
```

NERD ZONE # Pre-Lab Question 3 Take the chart above, sort the bars in descending order

```
# your code here
```

20 Joins

In this chapter:

- A `join` combines two or more tables (data frames) by column.
- joining to data frames requires exact matches on one or more columns. Close matches don't count.
- Use codes in one data frame to "look up" information in another, and attach it to a row, such as the translation of codes to words or demographics of a Census tract.
- Many public records databases come with "lookup tables" or "code sheets". Make sure to ask for the codes AND their translations in a data request.
- Reporters don't always stick to matchmaking the way the database designers intended. "Enterprise" joins are those that let you find needles in a haystack, such as bus drivers with a history of DUIs.
- Matching one data frame against an entirely different one will always produce errors. You can minimize the *kind* of error you fear most – false positives or false negatives – but you likely will have to report out your findings on the ground.

20.1 Join basics

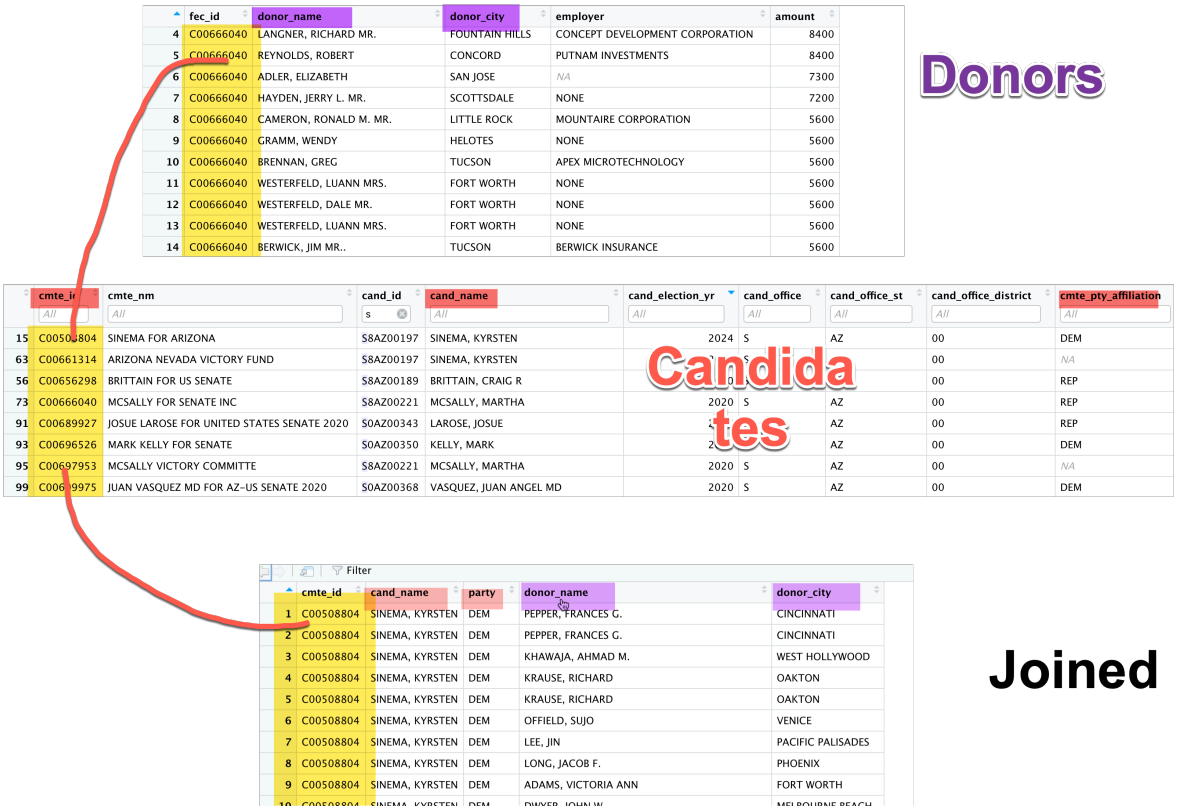
"Join" in computer programming mean adding columns to a data frame by combining it with another data frame or table. Reporters use it in many ways, some intended by the people who made the data source, and some not.

From now on, we'll start using the term "table" instead of "data frame", since we can talk about several different ones at the same time.

Many databases are created expecting you to join tables (data frames) because it's a more efficient way to store and work with large databases. This is what's called a "relational database", and they're everywhere.

Here's an example, using campaign finance information. The Federal Elections Commission distributes campaign contribution in *related* tables, each referring to a different noun. One table lists donations, the other lists candidates and other political action committees. They link together using a common code:

The reason to do this is that you never have to worry that any changes to the candidate information – the treasurer, the address or the office sought – carries over to the donation. It's



	fec_id	donor_name	donor_city	employer	amount
4	C00666040	LANGNER, RICHARD MR.	FOUNTAIN HILLS	CONCEPT DEVELOPMENT CORPORATION	8400
5	C00666040	REYNOLDS, ROBERT	CONCORD	PUTNAM INVESTMENTS	8400
6	C00666040	ADLER, ELIZABETH	SAN JOSE	NA	7300
7	C00666040	HAYDEN, JERRY L. MR.	SCOTTSDALE	NONE	7200
8	C00666040	CAMERON, RONALD M. MR.	LITTLE ROCK	MOUNTAIRE CORPORATION	5600
9	C00666040	GRAMM, WENDY	HELOTES	NONE	5600
10	C00666040	BRENNAN, GREG	TUCSON	APEX MICROTECHNOLOGY	5600
11	C00666040	WESTERFELD, LUANN MRS.	FORT WORTH	NONE	5600
12	C00666040	WESTERFELD, DALE MR.	FORT WORTH	NONE	5600
13	C00666040	WESTERFELD, LUANN MRS.	FORT WORTH	NONE	5600
14	C00666040	BERWICK, JIM MR..	TUCSON	BERWICK INSURANCE	5600

	cmte_id	cmte_nm	cand_id	cand_name	cand_election_yr	cand_office	cand_office_st	cand_office_district	cmte_pty_affiliation
	All	All	s	All	All	All	All	All	All
15	C00508804	SINEMA FOR ARIZONA	S8AZ00197	SINEMA, KYRSTEN	2024	S	AZ	00	DEM
63	C00661314	ARIZONA NEVADA VICTORY FUND	S8AZ00197	SINEMA, KYRSTEN	[illegible]	S	[illegible]	00	NA
56	C00656298	BRITTAIN FOR US SENATE	S8AZ00189	BRITTAIN, CRAIG R	[illegible]	[illegible]	[illegible]	00	REP
73	C00666040	MCSALLY FOR SENATE INC	S8AZ00221	MCSALLY, MARTHA	2020	S	AZ	00	REP
91	C00689927	JOSUE LAROSE FOR UNITED STATES SENATE 2020	S0AZ00343	LAROSE, JOSUE	[illegible]	[illegible]	AZ	00	REP
93	C00696526	MARK KELLY FOR SENATE	S0AZ00350	KELLY, MARK	[illegible]	[illegible]	AZ	00	DEM
95	C00697953	MCSALLY VICTORY COMMITTE	S8AZ00221	MCSALLY, MARTHA	2020	S	AZ	00	NA
99	C00699975	JUAN VASQUEZ MD FOR AZ-US SENATE 2020	S0AZ00368	VASQUEZ, JUAN ANGEL MD	2020	S	AZ	00	DEM

Filter

	cmte_id	cand_name	party	donor_name	donor_city
1	C00508804	SINEMA, KYRSTEN	DEM	PEPPER, FRANCES G.	CINCINNATI
2	C00508804	SINEMA, KYRSTEN	DEM	PEPPER, FRANCES G.	CINCINNATI
3	C00508804	SINEMA, KYRSTEN	DEM	KHAWAJA, AHMAD M.	WEST HOLLYWOOD
4	C00508804	SINEMA, KYRSTEN	DEM	KRAUSE, RICHARD	OAKTON
5	C00508804	SINEMA, KYRSTEN	DEM	KRAUSE, RICHARD	OAKTON
6	C00508804	SINEMA, KYRSTEN	DEM	OFFIELD, SUJO	VENICE
7	C00508804	SINEMA, KYRSTEN	DEM	LEE, JIN	PACIFIC PALISADES
8	C00508804	SINEMA, KYRSTEN	DEM	LONG, JACOB F.	PHOENIX
9	C00508804	SINEMA, KYRSTEN	DEM	ADAMS, VICTORIA ANN	FORT WORTH

Figure 20.1: Campaign finance join

only listed once in the candidate table. Most large databases are constructed this way. For example:

- Your school records are held using your student ID, which means that your address and home email only needs to be changed once, not in every class or in every account you have with the school.
- Inspection records, such as those for restaurants, hospitals, housing code violations and workplace safety, typically have at least *three* tables: The establishment (like a restaurant or a workplace), an inspection (an event on a date), and a violation (something that they found). They're linked together using establishment ID's.
- A court database usually has many types of records: A master case record links to information on charges, defendants, lawyers, sentences and court hearings.

Each table, then, is described using a different noun – candidates or contributions; defendants or cases; students or courses. This conforms to the **tidy data** principle that different types of information are stored in different tables.

20.2 Matchmaking with joins

The political data type of join described above is often referred to as a "lookup table". You'll match codes to their meanings in a way that was intended by the people who made the database. But there are other ways reporters use joins:

"Enterprise" joins

Journalists have taken to calling a specific kind of join "enterprise", referring to the enterprising reporters who do this. Here, you'll look for needles in a haystack. Some of the most famous data journalism investigations relied on joining two databases that started from completely different sources, such as:

- Bus drivers who had DUI citations
- Donors to a governor who got contracts from the state
- Day care workers with criminal records
- Small businesses that have defaulted on government backed loans that got a PPP loan anyway.

When you match these kinds of datasets, you will always have some error. You always have to report out any suspected matches, so they are time consuming stories.

In the mid-2000s, when some politicians insisted that dead people were voting and proposed measures to restrict registration, almost every regional news organization sent reporters on futile hunts for the dead voters. They got lists of people on the voter rolls, then lists of people

who had died through the Social Security Death Index or local death certificates. I never met anyone who found a single actual dead voter, but months of reporter-hours were spent tracking down each lead.

It's very common for two people to have the same name in a city. In fact, it's common to have two people at the same home with the same name – they've just left off "Jr." and "Sr." in the database. In this case, you'll find matches that you shouldn't. These are false positives, or Type I errors in statistics.

Also, we rarely get dates of birth or Social Security Numbers in public records, so we have to join by name and sometimes location. If someone has moved, sometimes uses a nickname, or the government has recorded the spelling incorrectly, the join will fail – you'll miss some of the possible matches. This is very common with company names, which can change with mergers and other changes in management, and can be listed in many different ways.

These are false negatives, or Type II errors in statistics.[1]

In different contexts, you'll want to minimize different kinds of errors. For example, if you are looking for something extremely rare, and you want to examine every possible case – like a child sex offender working in a day care center – you might choose to make a "loose" match and get lots of false positives, which you can check. If you want to limit your reporting only to the most promising leads, you'll be willing to live with missing some cases in order to be more sure of the joins you find.

You'll see stories of this kind write around the lack of precision – they'll often say, "we verified x cases of…." rather than pretend that they know of them all.

Find cases with interesting characteristics

You'll often want to learn more about a geographic area's demographics or voting habits or other characteristics, and match it to other data. Sometimes it's simple: Find the demographics of counties that switched from Trump to Biden as a way to isolate places you might want to visit. Another example from voting might be to find the precinct that has the highest percentage of Latino citizens in the county, then match that precinct against the voter registration rolls to get a list of people you might want to interview on election day. In these instances, the `join` is used for a `filter`.

This is also common when you have data by zip code or some other geography, and you want to find clusters of interesting potential stories, such as PPP loans in minority neighborhoods.

[1] I remember them by thinking of the boy who cried wolf. When the village came running and there was no wolf, it was a Type I error, or false positive ; when the village ignored the boy and there was a wolf, it was a Type II error, or false negative.

Summarize data against another dataset

The previous examples all result in lists of potential story people or places. If you use join on summarized data, you can characterize a broad range of activity across new columns. Simplified, this is how you can write that more PPP money went to predominantly white neighborhoods than those that were majority Black.

20.2.1 Types of joins

There are several types of joins, each connecting two tables in a slightly different matter. The most important ones are:

- inner_join
- left_join
- anti_join

We'll explore how each of these work in example code below:

20.2.2 Load libraries

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

20.2.3 Load data

We're going to load four tables to demonstrate different types of joins. All contain information from the U.S. Census 2019 American Community Survey for Maryland counties. Load and examine them.

```
###
# Total population for each Maryland county
# County identified by GEOID (a 5-digit code), not name
###

maryland_county_population <- read_rds("assets/data/maryland_county_population.rds")
```

```
###
# A lookup table that shows the name of each Maryland county, paired with GEOID
###


maryland_county_lookup_table <- read_rds("assets/data/maryland_county_lookup_table.rds")

###
# Total population for each Maryland county, EXCEPT for Prince George's County
# County identified by GEOID (a 5-digit code), not name
###

maryland_county_population_no_pg <- read_rds("assets/data/maryland_county_population_no_pg

###
# Display the tables
###

maryland_county_population
```

```
# A tibble: 24 x 2
   geoid total_pop
   <chr>     <dbl>
 1 24001     71002
 2 24003    575421
 3 24005    828193
 4 24009     92094
 5 24011     33260
 6 24013    168233
 7 24015    102889
 8 24017    161448
 9 24019     31994
10 24021    255955
# ... with 14 more rows
```

```
maryland_county_lookup_table
```

```
# A tibble: 24 x 2
   geoid name
   <chr> <chr>
```

```
 1 24001 Allegany County, Maryland
 2 24003 Anne Arundel County, Maryland
 3 24005 Baltimore County, Maryland
 4 24009 Calvert County, Maryland
 5 24011 Caroline County, Maryland
 6 24013 Carroll County, Maryland
 7 24015 Cecil County, Maryland
 8 24017 Charles County, Maryland
 9 24019 Dorchester County, Maryland
10 24021 Frederick County, Maryland
# ... with 14 more rows
```

```
maryland_county_population_no_pg
```

```
# A tibble: 23 x 2
   geoid total_pop
   <chr>     <dbl>
 1 24001     71002
 2 24003    575421
 3 24005    828193
 4 24009     92094
 5 24011     33260
 6 24013    168233
 7 24015    102889
 8 24017    161448
 9 24019     31994
10 24021    255955
# ... with 13 more rows
```

20.2.4 Inner Join

The table `maryland_county_population` has two columns, one with a `geoid` code representing each Maryland county (and Baltimore city), and another with total population. The `maryland_county_lookup_table` is a classic example of a lookup table. It contains the same `geoid` representing each Maryland county, plus a name column with the proper name.

We are going to join `maryland_county_population`, using the one column both share: `geoid`. We'll use an `inner_join`. The code below says: "Connect the two tables on the `geoid` column. If a given `geoid` exists in both tables, include it in our new table. If it isn't in both, don't include it."

```
updated_maryland_county_population <- maryland_county_lookup_table %>%
  inner_join(maryland_county_population, by="geoid")

updated_maryland_county_population
```

```
# A tibble: 24 x 3
   geoid name                           total_pop
   <chr> <chr>                              <dbl>
 1 24001 Allegany County, Maryland         71002
 2 24003 Anne Arundel County, Maryland    575421
 3 24005 Baltimore County, Maryland       828193
 4 24009 Calvert County, Maryland          92094
 5 24011 Caroline County, Maryland         33260
 6 24013 Carroll County, Maryland         168233
 7 24015 Cecil County, Maryland           102889
 8 24017 Charles County, Maryland         161448
 9 24019 Dorchester County, Maryland       31994
10 24021 Frederick County, Maryland       255955
# ... with 14 more rows
```

Because both tables contained the same 24 counties, our new table has 24 counties as well.

(In this case, they have the same column name, but they don't have to. As long as they CONTAIN the same thing, they can have different names. You can also join using more than one column).

What happens when they aren't identical?

To illustrate what happens, let's run the same code, but this time use population table that contains every Maryland county EXCEPT for Prince George's County. This table only has 23 rows. The lookup table still has 24.

```
# A tibble: 23 x 3
   geoid name                           total_pop
   <chr> <chr>                              <dbl>
 1 24001 Allegany County, Maryland         71002
 2 24003 Anne Arundel County, Maryland    575421
 3 24005 Baltimore County, Maryland       828193
 4 24009 Calvert County, Maryland          92094
 5 24011 Caroline County, Maryland         33260
 6 24013 Carroll County, Maryland         168233
 7 24015 Cecil County, Maryland           102889
```

```
 8 24017 Charles County, Maryland         161448
 9 24019 Dorchester County, Maryland       31994
10 24021 Frederick County, Maryland       255955
# ... with 13 more rows
```

When we join, we get only 23 rows. One table had every Maryland county. The other had every county but Prince George's. When we inner join, only values that both tables have in common are returned. Thus, we get a table without P.G.

This gif from https://github.com/gadenbuie/tidyexplain shows what happens with inner_joins.

20.2.5 Left Joins

Let's stick with our mismatched tables to illustrate what happens with another type of join, a left join (and it's cousin, the right join). The code is the same, but we've swapped `left_join` for `inner_join`. Remember, our population table is missing P.G. county, but the lookup table has it.

```
updated_maryland_county_population_no_pg <- maryland_county_lookup_table %>%
  left_join(maryland_county_population_no_pg, by="geoid")

updated_maryland_county_population_no_pg
```

```
# A tibble: 24 x 3
   geoid name                           total_pop
   <chr> <chr>                              <dbl>
 1 24001 Allegany County, Maryland         71002
 2 24003 Anne Arundel County, Maryland    575421
 3 24005 Baltimore County, Maryland       828193
 4 24009 Calvert County, Maryland          92094
 5 24011 Caroline County, Maryland         33260
 6 24013 Carroll County, Maryland         168233
 7 24015 Cecil County, Maryland           102889
 8 24017 Charles County, Maryland         161448
 9 24019 Dorchester County, Maryland       31994
10 24021 Frederick County, Maryland       255955
# ... with 14 more rows
```

What happens? Everywhere there's a match on GEOID, the population value is returned.

But look closely at Prince George's County. Its value is NA.

A `left_join` says "Return every single value from our first table – in this case the lookup table, with 24 rows. Where there's a match with the population table, make the match. If there is no match, just put NA."

Unlike the `inner_join`, the `left_join` returned P.G. county despite it's lack of a match.

This gif from https://github.com/gadenbuie/tidyexplain shows what happens with left_joins.

A note: `left_join` has a close cousin called the `right_join` which isn't used very often, and we won't delve into here. It's basically the same thing, it just starts from the second table, not the first.

20.3 Anti joins

An `anti_join` is useful for seeing what values exist in one table that are missing from another table. It comes in handy during data cleaning and when writing more advanced functions. The code below says: "Connect the two tables, but only show me rows that exist in the lookup table, but DO NOT exist in the population table."

```
updated_maryland_county_population_no_pg <- maryland_county_lookup_table %>%
  anti_join(maryland_county_population_no_pg, by="geoid")

updated_maryland_county_population_no_pg
```

```
# A tibble: 1 x 2
  geoid name
  <chr> <chr>
1 24033 Prince George's County, Maryland
```

This gif from https://github.com/gadenbuie/tidyexplain shows what happens with anti_joins.

20.4 Joining risks

There are lots of risks in joining tables that you created yourself, or that were created outside a big relational database system. Keep an eye on the number of rows returned every time that you join – you should know what to expect.

20.4.1 Double counting with joins

We won't go into this in depth, but just be aware it's easy to double-count rows when you join. Here's a made-up example, in which a zip code is on the border and is in two counties:

Say you want to use some data on zip codes :

zip code	county	info
21012	PG	some data
21012	Baltimore	some more data

and match it to a list of restaurants in a zip code:

zip code	restaurant name
21012	My favorite restaurant
21012	My second-favorite restaurant

When you match these on zip code, you'll get **4** rows:

zip code	county	info	restaurant name
21012	PG	some data	My favorite restaurant
21012	Baltimore	some more data	My favorite restaurant
21012	PG	some data	My second-favorite restaurant
21012	Baltimore	some more data	My second-favority restaurant

Now, every time you try to count restaurants, these two will be double-counted.

In computing, this is called a "many-to-many" relationship – there are many rows of zip codes and many rows of restaurants. In journalism, we call it spaghetti. It's usually an unintended mess.

Here's a gif that shows the double counting in action.

20.4.2 Losing rows with joins

The opposite can occur if you aren't careful and there are items you want to keep that are missing in your reference table. That's what happened in the immunization data above for the seven schools that I couldn't find.

20.4.3 Merging or Binding

Joins are one way to combine two data sets, using shared values in one or more columns. Binding two data sets together is another way.

Let's suppose we have two tables with population information for Maryland counties, each with the same columns: geoid, name and total population.

- One has information for Maryland's 10 largest counties by population
- The other has information for every other Maryland county.

```
###
# Total population for each top-10 largest Maryland county
# County identified by GEOID (a 5-digit code) and name
###

maryland_large_county_population <- read_rds("assets/data/maryland_large_county_population


###
# Total population for all other Maryland counties
# County identified by GEOID (a 5-digit code) and name
###
maryland_small_county_population <- read_rds("assets/data/maryland_small_county_population


# Display
maryland_large_county_population
```

```
# A tibble: 10 x 3
   geoid name                                total_pop
   <chr> <chr>                                   <dbl>
 1 24031 Montgomery County, Maryland           1047661
 2 24033 Prince George's County, Maryland       910551
 3 24005 Baltimore County, Maryland             828193
 4 24510 Baltimore city, Maryland               602274
 5 24003 Anne Arundel County, Maryland          575421
 6 24027 Howard County, Maryland                322407
 7 24021 Frederick County, Maryland             255955
 8 24025 Harford County, Maryland               253736
 9 24013 Carroll County, Maryland               168233
10 24017 Charles County, Maryland               161448
```

```
maryland_small_county_population
```

```
# A tibble: 14 x 3
   geoid name                           total_pop
   <chr> <chr>                              <dbl>
 1 24029 Kent County, Maryland              19456
 2 24039 Somerset County, Maryland          25699
 3 24023 Garrett County, Maryland           29155
 4 24019 Dorchester County, Maryland        31994
 5 24011 Caroline County, Maryland          33260
 6 24041 Talbot County, Maryland            37087
 7 24035 Queen Anne's County, Maryland      50163
 8 24047 Worcester County, Maryland         51967
 9 24001 Allegany County, Maryland          71002
10 24009 Calvert County, Maryland           92094
11 24015 Cecil County, Maryland            102889
12 24045 Wicomico County, Maryland         103222
13 24037 St. Mary's County, Maryland       113182
14 24043 Washington County, Maryland       150575
```

If we want to combine these tables into one dataset with every county's information, a join isn't the way to go. For that, we'd use `bind_rows()`. Thick of bind_rows as stacking two tables on top of each other. This code gives us 24 rows.

```
all_county_population <- maryland_small_county_population %>%
  bind_rows(maryland_large_county_population)

all_county_population
```

```
# A tibble: 24 x 3
   geoid name                           total_pop
   <chr> <chr>                              <dbl>
 1 24029 Kent County, Maryland              19456
 2 24039 Somerset County, Maryland          25699
 3 24023 Garrett County, Maryland           29155
 4 24019 Dorchester County, Maryland        31994
 5 24011 Caroline County, Maryland          33260
 6 24041 Talbot County, Maryland            37087
 7 24035 Queen Anne's County, Maryland      50163
 8 24047 Worcester County, Maryland         51967
```

```
 9 24001 Allegany County, Maryland         71002
10 24009 Calvert County, Maryland          92094
# ... with 14 more rows
```

Question #1: Answer this question in English: Write in Elms

You have a dataframe that contains information on the population of each Maryland county, structured like this example (the column headers and one example row):

geoid | name | total_pop 24029 | Kent County, Maryland | 19456

You have another dataframe with a count of large employers (over 10000 workers) by county in Maryland, structured like this example (the column headers and one example row):

name | number_of_large employers Kent County | 15

You want to join these two dataframes to answer the question "which Maryland county has the highest number of large employees per person?"

What do you think will happen when you attempt to "inner_join" these two tables?

20.5 Resources

- The "Relational data" chapter in the R for Data Science textbook has details on exactly how a complex data set might fit together.
- An example using a superheroes dataset, from Stat 545 at the University of British Columbia

21 Data Cleaning

Any time you are given a dataset from anyone, you should immediately be suspicious. Is this data what I think it is? Does it include what I expect? Is there anything I need to know about it? Will it produce the information I expect?

One of the first things you should do is give it the smell test.

Failure to give data the smell test can lead you to miss stories and get your butt kicked on a competitive story.

With data smells, we're trying to find common mistakes in data. For more on data smells, read the GitHub wiki post that started it all. Some common data smells are:

- Missing data or missing values
- Gaps in data
- Wrong type of data
- Outliers
- Sharp curves
- Conflicting information within a dataset
- Conflicting information across datasets
- Wrongly derived data
- Internal inconsistency
- External inconsistency
- Wrong spatial data
- Unusable data, including non-standard abbreviations, ambiguous data, extraneous data, inconsistent data

Not all of these data smells are detectable in code. You may have to ask people about the data. You may have to compare it to another dataset yourself. Does the agency that uses the data produce reports from the data? Does your analysis match those reports? That will expose wrongly derived data, or wrong units, or mistakes you made with inclusion or exclusion.

This chapter will take us through the first three, and look at common solutions.

21.1 Wrong Type

First, let's look at **Wrong Type Of Data**.

Load the tidyverse.

```
# Remove scientific notation
options(scipen=999)
# Load the tidyverse
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.2      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Then lets load a dataframe of Census population statistics, one row per state per year between the period of 2015 and 2020, with overall totals and totals by Census race categories. I have intentionally introduced some flaws that we'll have to clean.

```
state_population_dirty <- read_rds("assets/data/state_population_dirty.rds")

glimpse(state_population_dirty)
```

```
Rows: 255
Columns: 11
$ geoid                          <chr> "01", "01", "01", "01", "01", "02", "02",~
$ state                          <chr> "Alabama", "Alabama", "Alabama", "Alabama~
$ year                           <dbl> 2015, 2016, 2017, 2018, 2020, 2015, 2016,~
$ total_pop                      <chr> "4830620", "4841164", "4850771", "4864680~
$ white_alone_pop                <dbl> 3325464, 3325037, 3317702, 3317453, 33028~
$ black_alone_pop                <dbl> NA, NA, NA, NA, NA, 25022, 24443, 23702, ~
$ amer_ind_ak_native_alone_pop   <dbl> 23850, 23919, 25098, 25576, 24764, 101313~
$ asian_alone_pop                <dbl> 59599, 60744, 62815, 64609, 67909, 42921,~
$ native_hi_alone_pop            <dbl> 2439, 2008, 2213, 2182, 2042, 8841, 8862,~
$ other_race_alone_pop           <dbl> 61078, 61991, 66942, 70055, 74996, 9273, ~
$ two_or_more_races_pop          <dbl> 81646, 85412, 88834, 91619, 119322, 61755~
```

Let's sort the dataframe by total population to identify the state with the largest population in 2020.

```
state_population_dirty %>%
  filter(year == 2020) %>%
  arrange(desc(total_pop))
```

```
# A tibble: 51 x 11
   geoid state      year total~1 white~2 black~3 amer_~4 asian~5 nativ~6 other~7
   <chr> <chr>     <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 26    Michigan   2020 9973907 7735902 1360149   50035  316844    3117  131580
 2 10    Delaware   2020 967679   652587  212795    3560   38528     705   21937
 3 34    New Jers~  2020 8885418 5820147 1189681   22288  857873    3156  564662
 4 46    South Da~  2020 879336   735228   18836   74975   12413     544    7320
 5 51    Virginia   2020 8509358 5643436 1620649   22553  570398    5994  240542
 6 38    North Da~  2020 760394   651470   23959   39165   11979    1004    8875
 7 53    Washingt~  2020 7512465 5523881  290245   91766  662902   51117  360578
 8 02    Alaska     2020 736990   466961   23894  107298   47289   10485   12231
 9 04    Arizona    2020 7174064 5292498  325105  311014  239190   14633  492027
10 11    District~  2020 701974   288306  318631    2438   28762     328   33764
# ... with 41 more rows, 1 more variable: two_or_more_races_pop <dbl>, and
#   abbreviated variable names 1: total_pop, 2: white_alone_pop,
#   3: black_alone_pop, 4: amer_ind_ak_native_alone_pop, 5: asian_alone_pop,
#   6: native_hi_alone_pop, 7: other_race_alone_pop
```

Something seems off. The largest U.S. states – New York, California, Texas – aren't on this list. It's topped by Michigan, with 9.9 million people, followed by Delaware with 967,000, followed by New Jersey with 8.8 million. It's not treating the values in this column as numbers, but sorting them … alphabetically, in a sense.

Let's use glimpse to figure out why.

```
glimpse(state_population_dirty)
```

```
Rows: 255
Columns: 11
$ geoid                          <chr> "01", "01", "01", "01", "01", "02", "02",~
$ state                          <chr> "Alabama", "Alabama", "Alabama", "Alabama~
$ year                           <dbl> 2015, 2016, 2017, 2018, 2020, 2015, 2016,~
$ total_pop                      <chr> "4830620", "4841164", "4850771", "4864680~
$ white_alone_pop                <dbl> 3325464, 3325037, 3317702, 3317453, 33028~
```

```
$ black_alone_pop              <dbl> NA, NA, NA, NA, NA, 25022, 24443, 23702, ~
$ amer_ind_ak_native_alone_pop <dbl> 23850, 23919, 25098, 25576, 24764, 101313~
$ asian_alone_pop              <dbl> 59599, 60744, 62815, 64609, 67909, 42921,~
$ native_hi_alone_pop          <dbl> 2439, 2008, 2213, 2182, 2042, 8841, 8862,~
$ other_race_alone_pop         <dbl> 61078, 61991, 66942, 70055, 74996, 9273, ~
$ two_or_more_races_pop        <dbl> 81646, 85412, 88834, 91619, 119322, 61755~
```

Here we can see the column name, sample values, and the data type. Most of the number columns in this dataframe are stored as numbers ("dbl"). But not the value in total_pop. It's stored as "character". R is interpreting as a text string. To sort properly, we'll need to change it. Let's update the dataframe by mutating that column to change the data type to numeric.

```
state_population_dirty <- state_population_dirty %>%
  mutate(total_pop = as.numeric(total_pop))

glimpse(state_population_dirty)
```

```
Rows: 255
Columns: 11
$ geoid                        <chr> "01", "01", "01", "01", "01", "02", "02",~
$ state                        <chr> "Alabama", "Alabama", "Alabama", "Alabama~
$ year                         <dbl> 2015, 2016, 2017, 2018, 2020, 2015, 2016,~
$ total_pop                    <dbl> 4830620, 4841164, 4850771, 4864680, 48931~
$ white_alone_pop              <dbl> 3325464, 3325037, 3317702, 3317453, 33028~
$ black_alone_pop              <dbl> NA, NA, NA, NA, NA, 25022, 24443, 23702, ~
$ amer_ind_ak_native_alone_pop <dbl> 23850, 23919, 25098, 25576, 24764, 101313~
$ asian_alone_pop              <dbl> 59599, 60744, 62815, 64609, 67909, 42921,~
$ native_hi_alone_pop          <dbl> 2439, 2008, 2213, 2182, 2042, 8841, 8862,~
$ other_race_alone_pop         <dbl> 61078, 61991, 66942, 70055, 74996, 9273, ~
$ two_or_more_races_pop        <dbl> 81646, 85412, 88834, 91619, 119322, 61755~
```

Now when we sort, it works.

```
state_population_dirty %>%
  filter(year == 2020) %>%
  arrange(desc(total_pop))
```

```
# A tibble: 51 x 11
   geoid state      year total~1 white~2 black~3 amer_~4 asian~5 nativ~6 other~7
```

```
   <chr> <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 06    Californ~  2020  3.93e7  2.21e7 2250962  311629 5834312  149636 5623747
 2 48    Texas      2020  2.86e7  1.98e7 3464424  137921 1415664   25328 1788398
 3 12    Florida    2020  2.12e7  1.52e7 3381061   55655  590668   13339  699596
 4 36    New York   2020  1.95e7  1.22e7 3002401   76535 1674216    9376 1670723
 5 42    Pennsylv~  2020  1.28e7  1.02e7 1419582   20798  449320    4268  312888
 6 17    Illinois   2020  1.27e7  8.87e6 1796660   33972  709567    5196  757150
 7 39    Ohio       2020  1.17e7  9.39e6 1442655   20442  268527    3907  129717
 8 13    Georgia    2020  1.05e7  6.02e6 3319844   34962  434603    7127  306609
 9 37    North Ca~  2020  1.04e7  7.02e6 2217522  120272  308958    7368  334295
10 26    Michigan   2020  9.97e6  7.74e6 1360149   50035  316844    3117  131580
# ... with 41 more rows, 1 more variable: two_or_more_races_pop <dbl>, and
#   abbreviated variable names 1: total_pop, 2: white_alone_pop,
#   3: black_alone_pop, 4: amer_ind_ak_native_alone_pop, 5: asian_alone_pop,
#   6: native_hi_alone_pop, 7: other_race_alone_pop
```

21.2 Missing Data

The second smell we can find in code is **missing data**.

Let's try to calculate the total Black alone population for the U.S.

```
state_population_dirty %>%
  filter(year == 2020) %>%
  summarise(
    total_us_black_alone_population = sum(black_alone_pop)
  )
```

```
# A tibble: 1 x 1
  total_us_black_alone_population
                            <dbl>
1                              NA
```

We get an NA value, which isn't correct. Let's examine the values in that column to see why.

```
state_population_dirty
```

```
# A tibble: 255 x 11
   geoid state    year total_pop white~1 black~2 amer_~3 asian~4 nativ~5 other~6
   <chr> <chr>   <dbl>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```
 1 01    Alabama  2015   4830620 3325464      NA   23850   59599    2439   61078
 2 01    Alabama  2016   4841164 3325037      NA   23919   60744    2008   61991
 3 01    Alabama  2017   4850771 3317702      NA   25098   62815    2213   66942
 4 01    Alabama  2018   4864680 3317453      NA   25576   64609    2182   70055
 5 01    Alabama  2020   4893186 3302834      NA   24764   67909    2042   74996
 6 02    Alaska   2015    733375  484250   25022  101313   42921    8841    9273
 7 02    Alaska   2016    736855  483518   24443  103574   44218    8862    9900
 8 02    Alaska   2017    738565  481971   23702  104995   45604    9075   10505
 9 02    Alaska   2018    738516  478834   24129  106660   46556    8849   11027
10 02    Alaska   2020    736990  466961   23894  107298   47289   10485   12231
# ... with 245 more rows, 1 more variable: two_or_more_races_pop <dbl>, and
#   abbreviated variable names 1: white_alone_pop, 2: black_alone_pop,
#   3: amer_ind_ak_native_alone_pop, 4: asian_alone_pop,
#   5: native_hi_alone_pop, 6: other_race_alone_pop
```

Ah, we see a missing value for Alabama. Because of that value, the summarise calculation won't work. We could tell R to ignore NA values when doing the calculation, by adding na.rm=TRUE to the sum function.

```
state_population_dirty %>%
  filter(year == 2020) %>%
  summarise(
    total_us_black_alone_population = sum(black_alone_pop, na.rm=TRUE)
  )
```

```
# A tibble: 1 x 1
  total_us_black_alone_population
                            <dbl>
1                        39926065
```

But! That's not the right answer. We want to know the total for the U.S. including Alabama in 2020. So, we need to fix that value. We look up the value on the Census website, and determine it's 1,301,319 for 2020. So let's update the column. This uses a function called case_when() with mutate().

Here's how to interpret what the function below says.

"Overwrite the black_alone_pop column with new values in one of the rows, and the old values for every other row. If the state column equals Alabama AND (the &) the year column equals 2020 THEN (the tilde or ~) put the value 1301319. In any other case (any other state, or Alabama in any other year than 2020), THEN keep the value that currently exists in the black_alone_pop column.

```r
state_population_dirty <- state_population_dirty %>%
  mutate(black_alone_pop = case_when(
    state == "Alabama" & year == 2020 ~ 1301319,
    TRUE ~ black_alone_pop
  ))

state_population_dirty
```

```
# A tibble: 255 x 11
   geoid state    year total_pop white~1 black~2 amer_~3 asian~4 nativ~5 other~6
   <chr> <chr>   <dbl>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 01    Alabama  2015   4830620 3325464      NA   23850   59599    2439   61078
 2 01    Alabama  2016   4841164 3325037      NA   23919   60744    2008   61991
 3 01    Alabama  2017   4850771 3317702      NA   25098   62815    2213   66942
 4 01    Alabama  2018   4864680 3317453      NA   25576   64609    2182   70055
 5 01    Alabama  2020   4893186 3302834 1301319   24764   67909    2042   74996
 6 02    Alaska   2015    733375  484250   25022  101313   42921    8841    9273
 7 02    Alaska   2016    736855  483518   24443  103574   44218    8862    9900
 8 02    Alaska   2017    738565  481971   23702  104995   45604    9075   10505
 9 02    Alaska   2018    738516  478834   24129  106660   46556    8849   11027
10 02    Alaska   2020    736990  466961   23894  107298   47289   10485   12231
# ... with 245 more rows, 1 more variable: two_or_more_races_pop <dbl>, and
#   abbreviated variable names 1: white_alone_pop, 2: black_alone_pop,
#   3: amer_ind_ak_native_alone_pop, 4: asian_alone_pop,
#   5: native_hi_alone_pop, 6: other_race_alone_pop
```

Now when we run our summarization code, it works.

```r
state_population_dirty %>%
  filter(year == 2020) %>%
  summarise(
    total_us_black_alone_population = sum(black_alone_pop)
  )
```

```
# A tibble: 1 x 1
  total_us_black_alone_population
                            <dbl>
1                        41227384
```

21.3 Gaps in data

Let's now look at **gaps in data**. One type of gap in data has to do with time.

Let's calculate the average white alone population in Maryland over the period represented in the data, which is 2015 to 2020.

```
state_population_dirty %>%
  filter(state == "Maryland") %>%
  summarise(
    mean_white_alone_pop = mean(white_alone_pop)
  )
```

```
# A tibble: 1 x 1
  mean_white_alone_pop
                 <dbl>
1             3373558.
```

Does this represent the average white population from 2015 to 2020? Let's take a closer look at the years represented in the data.

```
state_population_dirty %>%
  filter(state == "Maryland") %>%
  select(year)
```

```
# A tibble: 5 x 1
   year
  <dbl>
1  2015
2  2016
3  2017
4  2018
5  2020
```

We have 2015, 2016, 2017, 2018 and 2020. We can't accurately say this represents the average over this period without 2019. So let's add it, using a function called add_row() to put in the correct value 3,343,003. Note that it adds a row for 2019.

```
state_population_dirty %>%
  filter(state == "Maryland") %>%
```

```
    select(state,year,white_alone_pop) %>%
    add_row(
      state = "Maryland",
      year = 2019,
      white_alone_pop = 3343003
    )
```

```
# A tibble: 6 x 3
  state     year white_alone_pop
  <chr>    <dbl>           <dbl>
1 Maryland  2015         3416107
2 Maryland  2016         3408240
3 Maryland  2017         3395212
4 Maryland  2018         3373181
5 Maryland  2020         3275048
6 Maryland  2019         3343003
```

Now, when we take the average for Maryland, it's accurate.

```
 state_population_dirty %>%
    filter(state == "Maryland") %>%
    select(state,year,white_alone_pop) %>%
    add_row(
      state = "Maryland",
      year = 2019,
      white_alone_pop = 3343003
    ) %>%
    summarise(
      mean_white_alone_pop = mean(white_alone_pop)
    )
```

```
# A tibble: 1 x 1
  mean_white_alone_pop
                 <dbl>
1             3368465.
```

(APPENDIX) Appendix

Newsroom numbers cheat sheet

(Selected and edited from drafts of "Numbers in the Newsroom", 2nd edition, 2014, by Sarah Cohen. Full book available from IRE)

Virtually every number we use must be compared with something – another time, another place or a total. This newsroom math guide helps you put numbers into perspective using three devices: "Per" something, change, and averages.

These are arithmetic problems – formulas many of us learned sometime around the fourth grade. Unfortunately, we had another decade to forget them before we got into reporting – then got caught in the trap of math phobia – before these tools became second nature.

The good news is that most news stories don't depend on fancy math. Master this arithmetic again, and you'll be able to tackle most reporting jobs.

NOTE TO READERS ON MOBILE DEVICES: This is going to look incomprehensible. Don't even try until you get to a bigger screen.

The PERS: Fractions, rates, percents and per capita

You can usually simplify your story if you can re-jigger your numbers into a rate, a ratio or a percentage. "One out of four" is a fraction, or a rate. "Forty percent" is another ratio or rate. And 235 deaths per 100,000 people is another.

Percents and fractions are used to scale of very large or very small numbers while putting them into perspective.

Rates are also used to level the playing field – they compare two items that have a different base.

When you see a lot of numbers in copy, examine them to see if a simple rate – "one of four" or 25 percent – would simplify your story.

Fractions and percents

Repeat this: "Percents are fractions. Fractions are percents." Remembering this all the time will keep you focused on the key element of percentages: They're ratios, or rates, expressed as a fraction of 100.

Figuring a percent:

Step 1: Know your base. Think of the words "out of." It's the total of all the groups.

Step 2: Divide the category you care about by the base.
Remember that a fraction sign (/) means "divided by" (÷).

Step 3: Move the decimal point two places to the right (or multiply by 100) to get the rate per hundred, or percent.

Step 4: Round the answer to no more than one decimal place. Better yet, look for an easier fraction your readers will understand.

Formula

Step 1: Total = The base

Step 2: (Category / Total) = Proportion

Step 3: Proportion x 100 = Percent

Step 4: Round and simplify.

Example

If 58 people say they will vote in an upcoming election and 92 say they won't, this is how to compute the percent of people who claim they will vote:

Step 1: Base = number of people asked = 92 + 58 = 150

Step 2: Rate = 58 out of 150 = 58/150 = .386666..

Step 3: Percent = .38666… x 100 = 38.666666….

Step 4: Round and simplify: = nearly 40 percent

From fractions to percents and back

You probably know that 1 out of 4 is one-quarter, and that it's also 25 percent. But you may not know how to get from one to another.

From fractions to percents:

$1/4 = 1 \div 4 = 0.25$. Move the decimal place over two places, or multiply by 100, to get 25%

From percents to fractions:

1. Write your percent as a fraction: 25/100
2. Try to find a "least common denominator:" 25 in this case goes into both the top and the bottom. You might want to round off either number to come out to a simple denominator.
3. Simplify: (25 / 25) / (100 / 25) = 1 / 4

To get "One out of " numbers:

1. Express your percentage as a proportion by dividing by 100, so 25% is 0.25.
2. Now divide one by that number: 1 / .25 = 4, so your answer is one-fourth.

Tip for spreadsheet users: Excel allows you to format a number as a fraction or a percent. Play around with formats to see how the number is most easily described.

Rates and per capita

As with percentages, per person or per capita rates are used to level the playing field.

They're often used when you need to compare two dissimilar places or events: Crimes in cities with different populations, deaths from various diseases or Gross Domestic Product across countries.

Rates also are often used with very big or very small numbers to change them into something we can understand.

Sometimes, though, a rate makes things more complicated, especially when events are rare and there is a consensus that they shouldn't ever happen. Some examples include the 32 crashes attributed to GM's faulty ignition switch, or the 64 deaths that the Centers for Disease Control associated with pharmacy compounding errors in 2012.

One rule of thumb is to use raw numbers when they are under 100, and revert to some kind of fraction or rate when they grow bigger.

Rates for large numbers

A raw per-person figure is an average and should usually be used with very big numbers.

A Gross Domestic Product of $17 trillion is hard to digest. So we reduce it to a number we can understand. If we divide it by 317 million, we get about $54,000 for every man, woman and child in the country. It doesn't mean that each person earned $54,000 – in fact, almost half of all families earned less than that altogether at this writing. Instead, it includes all of the income that is generated by companies as well as people.

But the device turns an incomprehensible number into something we can picture. It also helps if we want to compare countries – it levels the playing field by adjusting for the size of the country.

Rates for small numbers - crime, death, and other rare events

Rates such as 23 per 1,000 people or something like it – are the same as percentages, but you multiply by something bigger than 100 or move the decimal place further to the right. Use these for very small numbers.

If 2.5 million people die in this country every year, then the percentage of people who die is a really small number: 0.789 per 100, or percent.

A number that little is hard to digest. So experts up the ante and express the figure as 789 deaths per 100,000 people.

Be careful about rates based on very small numbers. One example is the number of police shootings per 100,000 people. Most police departments in the country are very small and are more likely to serve only about 5,000 people. This means that just one shooting in the department can lift them from one of the lowest rates in the nation to one of the highest. Expect rates based on very small numbers to be unstable and potentially misleading.[1]

Figuring a rate

Step 1: Choose your base. This is often difficult. In reporting on fatalities by make of car, should you use the number of cars on the road, the number sold, or the total miles driven each year? You'll have to decide.

Step 2: Divide the number you care about by the base. Choosing the numerator can also be tricky. Going back to the automobile fatality example, would you use the total number of deaths or the number of driver deaths? Take a hint using other reports you see on the topic. Experts have often come to an informal agreement about what the most telling number is.

Step 3: Multiply by a nice round number, such as 1,000, 100,000 or 1 million.

[1]Note the disclaimer in this story on police shootings by The Washington Post, in which changes in the rates of police shootings may just be random.

Step 4: Round the answer and simplify.

Formula

Step 1: Choose the base, or "total"

Step 2: (Category / Total) = Proportion or Rate

Step 3: Proportion x 1,000 = Rate per thousand

Step 4: Round to zero decimal places

Example

According to the FBI Crime in the United States for 2012, there were 13,000 violent and property crimes in Pittsburgh out of a population of 312,000. There were 8,870 crimes in Tucson out of a population of 531,000. Figuring a rate per thousand residents lets you compare the two cities:

Pittsburgh	Tucson
Step 1: Base= 312,000 people Step 2: 13,000 crimes / 312,000 people = 0.041 Step 3: 0.041 x 1,000 = 41 crimes per thousand	Step 1: Base = 531,000 people Step 2: 8,870 / 531,000 people = 0.017 Step 3: .017 * 1,000 = 17 crimes per thousand

So the crime rate for Pittsburgh was nearly 2 1/2 times that of Tucson that year , or 47/17 = 2.4

Selecting your multiplier

Some people feel that changing their multiplier from 100 to something bigger is cheating.

After all, a 0.2 percent rate becomes a big number – 200 – when you change the base from 100 to 100,000!

In practice, though, there's nothing magical about using a base of 100 (or percent). Instead, use the number that makes sense for the comparison you're making.

- Choose a round number – 1,000, 1 million or 100,000.

- Choose the same number that the experts use: Crimes per 1,000 people, deaths per 100,000, or crashes per million miles driven, for example.
- Choose a base that will give you an easy way to express it to your readers. This is one that results in a number generally between 1 and 1,000 or so.
- Try to avoid using an outrageously large base. For instance, avoid expressing a local number in terms of 1 million people. Only a handful of cities have more than a million people.
- Keep the same base throughout your story. Don't shift from 100,000 to 1,000 in crime statistics, for instance, when you move from murders to total crime rates.

You will often have to balance these rules of thumb against each other to come up with a compromise that allows you to write gracefully while keeping the sense of scale appropriate for the comparisons you're making.

Measuring change

We often write about change or difference, usually as a difference between place or time.

Simple differences

A simple difference is just the result of subtracting one number from another. If you are measuring differences in time, it's the newer number minus the older number.

One time to use a simple difference is when the number is understandable without any calculations. Prices of common household goods, salaries and home prices are examples of numbers that needn't always be put into perspective using percentage changes.

In the end, we work in news. That means that sometimes you'll use a raw number when it's more newsworthy. This doesn't necessarily mean the number is more alarming – just more meaningful.

Figuring a difference:

Subtract the older number from the newer number.

This is not the same as subtracting the little number from the big number.

If a number has fallen you get a negative number. If a number has risen you get a positive number.

Formula

New – Old.

Example An executive made $2.4 million last year. She made $2.9 million this year.

Her raise was: $2.9 – $2.4 = 0.5 million, or $500,000, or half a million dollars.

Percent change / Percent difference

The most butchered form of newsroom math is the percent difference, or the percent change.

Part of the problem is that some folks have found five or six different ways to compute them. Unfortunately, only two of them work every time. I'll show you both because sometimes – especially when you want to compare rates – one is easier than the other.

Note that these methods work whether or not the number is going up or going down. If the number has fallen, you'll get a negative answer. If the number has risen, you'll get a positive one. And it still comes out right if the increase is bigger than 100 percent.[2]

In practice, I use Method 1 when I'm working in spreadsheets because I can look at the simple difference in one column and then use it in the formula for the percentage difference. I use Method 2 when I want to compare to percent changes to one another or when working with annual rates.

Method 1: Subtract then divide

Method 2: Divide then subtract

Figuring a percent change

Step 1: Get the simple difference between the numbers by subtracting the older number from the newer number. It doesn't matter which one is bigger!

Step 2: Divide the answer by the older number.

Step 3: Multiply by 100, or move the decimal point two places to the right.

Step 4: Round off and simplify.

Step 1: Get the proportion of the new number compared to the old number. This is the same as the percent of total above, except the old number is the base.

Step 2: Subtract 1 from that ratio

Step 3: Multiply by 100, or move the decimal point two places to the right.

Step 4: Round off and simplify.

Formula

[2]It's impossible for a number to fall more than 100 percent. That would mean it went below zero and then no formula works. There's no good way to show a percent change when a figure like annual company earnings goes from profit to loss.

Step 1: New – Old = Difference

Step 2: Difference / Old = Decimal answer

Step 3: Decimal x 100 = percentage difference

Step 4: Round off.

Step 1: New / Old = Ratio

Step 2: Ratio - 1 = Decimal answer

Step 3: Decimal x 100 = percentage difference

Step 4: Round off.

Example

An executive made $2.4 million last year. They made $2.9 million in this year.

Step 1: Difference = 2.9 – 2.4 = 0.5

Step 2: Difference / Original number = 0.5 / 2.4 = .208

Step 3: Move the decimal point = 20.8%

Step 4: Round off and simplify: 21% = 21 / 100 = about 20 / 100 = or about one-fifth.

Step 1: Ratio = 2.9/2.4 = 1.208

Step 2: Decimal answer = 1.208 - 1 = .208

Step 3: Move the decimal point = 20.8%

Step 4: Round off and simplify: 21% = 21 / 100 = about 20 / 100 = or about one-fifth.

So the executive got a raise equivalent to one-fifth of their original salary.

Reversing or predicting a percent change

Remember that a number can grow many times, but it can only fall 100 percent to zero. This is a rough concept until you think it through. If you double a price of $20, increasing it by 100%, it's $40. If you triple it, it's $60. But if you reduce the $40 back to $20, it's a 50 percent drop, to one-half the level, not a 100 percent decrease. In other words, percent changes can't be reversed.

This means that the ads claiming you'll use three times less detergent or a food contains three times less salt are wrong and impossible. What they probably mean is that it would be three times as much if you used the other brand or ate the other food, or the brand is one third as much. Here are two ways this makes a difference:

You need two of three numbers to reverse or predict a percent change:

- Where the number started

- Where it ended
- What the percent change would (or will) be

Any two of those will give you what you need. It's easiest if we use Method 2 above to get there. The example above assumes you know where it started and where it ended. Here's how to do it if you you have either of the other two:

Where it starts and the percent change

Example: You started with $100 and it grew by a total of 12%. Or, you started with $100 and it fell by a total of 12% (the percent change was -12%)

Step 1: Convert the percent change to a ratio by moving the decimal place back : .12 (up) or -.12 (down)

Step 2: Add 1, resulting in 1.12 (up) or .88 (down)

Step 3: Multiply the beginning number by that amount : $100 x 1.12 = $112 (up), or $100 x .88 = $88 (down)

Where it ends and the total percent change

For example, say your house is worth $330,000, and it had appreciated by a total of 15% over the past few years. Here's how to figure out where it started:

Step 1: Convert the percent change to a decimal, as above: 0.15

Step 2: Divide the current value by that amount = $330 / .15 = $287

This isn't intuitive, but it differs because you're starting from a bigger base. 15 percent of 330 isn't the same as 15 percent of 287.

Going further with percents and rates

There are three common problems in changes and rates you will probably encounter that aren't part of this guide. You should get help or look it up when these situations come up: [3]

1. Relative risk: That's the technical term for dividing two percentages. If the mortgage denial rate for Black homeowners was 10 percent, and the denial rate of white homeowners was 5 percent, it means that Black homeowners are twice as likely to be denied a loan. This can be used with both rates and with changes.
2. Annual rates: When you know that something has grown, say, 2 percent a year for 10 years, it's not the same thing as 20 percent. You have to annualize it.

[3]They are part of the "Numbers in the Newsroom" book from which this guide is derived.

3. Adjusting for inflation: Comparing values across two points in time – especially today – means putting them on the same footing. Generally, you want to convert old values to their buying power today. For example, it's hard to compare salaries for teachers today with those 50 years ago, because our money isn't worth as much today.

Average and typical values

Averages[4] are just summaries. If a quote sums up an event, or an anecdote sums up a person using their actions instead of words, an average sums up a human condition of some kind – money, congestion, death or disease – in a single number.

Choosing your average carefully or deciding there may be another number or method to sum up a situation can mean the difference between accurately and inaccurately describing your story.

Understanding different kinds of "measures of central tendency" – what they tell us and what they don't – is the first thing you learn in basic statistics classes. If an it doesn't describe your data well, it's not very productive to move forward into many other kinds of analysis.

Trying to compare populations over time is particularly tricky using averages because of giant demographic shifts. Between the Baby Boom and the Millennials came what some people call the Baby Bust. Getting average spending on education, for example, across these generations is really misleading – it will boom, then bust, them boom again and no one number will describe that pattern.[5]

Two types of averages are reviewed here. Consult an introductory statistics book if your work depends on an average.

The average or mean

A "mean" is what people mean when they say the word "average".
It's most descriptive when it summarizes numbers that don't vary too much at either the top or bottom ends. These averages will often be misleading when they refer to items measured in dollar amount like incomes, housing costs and the like.

Figuring a simple average or mean

Step 1: Add up a list of numbers.

Step 2: Divide the answer by the number of numbers you've added up.

[4]I'm using the term "average" freely here. Technically, a simple average and median are measures of central tendency, but I'll treat them as different types of averages for simplicity sake.

[5]This is sort of an example of "Simpson's paradox" in that an average hides meaningful trends among sub-populations.

Formula

Step 1: Sum of numbers

Step 2: Sum / Count of numbers

For spreadsheet users: =AVERAGE(list of numbers)

Example

Here are six home prices on a block:

```
$275,000          $1,200,000
$275,000            $500,000
$200,000            $395,000
```

Step 1: 275 + 275 + 200 + 1,200 + 500 + 395 = 2,845 or $2,845,000

Step 2: $2,845,000 / 5 = $569,000.

So the average home price is more than all but one on the list.

The median

Medians are often used to summarize the value of things measured in dollars, especially home prices and incomes. They are not sensitive to one or two unusually high or low values the way the average in the previous example is.

But it's harder to get a median because you need a list of all values. For example, if you know the total income of a metropolitan area and the number of people in that area, you can compute the average – or per capita income – but not the median.

One way to express the median is to call it the "typical" value. Another way is to say that it's the "middle" value.

Figuring a median:

Step 1: List all of your numbers in order, beginning with the lowest and ending with the highest.

Step 2: Count how many numbers you have and divide by two.

Step 3: Add 0.5. If that comes out to a whole number (like 13), count up the list that many values.

If it's not (like 12.5), take the average of the two numbers surrounding the number. [6]

[6] In statistical programs like R, there are various ways to specify how to deal with medians when there are ties like this. This is the most common way, but it may not be the way your program handles it.

In other words, this is the closest you can get to the middle of the list. This is a sorting and counting job, not a calculator job.

In a spreadsheet, use the =MEDIAN() function.

Example:

Step 1:

The same list, but listed from lowest to highest, with an extra expensive home

1. $200,000
2. $275,000
3. $275,000
4. $395,000
5. $500,000
6. $1,200,000

Step 2: 6/2 = 3

Step 3: 3 + .5 = 3.5

Step 4: Average the 3rd and 4th items on the list: (275 + 395) / 2 = $335,000

As a rule of thumb, the median will be more telling than the average when they're very different as in this example. But the word "median" sounds very technical to some readers and the average encompasses all of the values in a list, so we use it when they're not too different.