

# **Data Journalism With R and the Tidyverse**

Matt Waite

1/18/24

# Table of contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                              | <b>5</b>  |
| 1.1      | Modern data journalism . . . . .                 | 6         |
| 1.2      | Installations . . . . .                          | 6         |
| 1.3      | About this book . . . . .                        | 7         |
| 1.4      | What we'll cover . . . . .                       | 7         |
| <b>2</b> | <b>Public records</b>                            | <b>9</b>  |
| 2.1      | Federal law . . . . .                            | 9         |
| 2.2      | State law . . . . .                              | 10        |
| <b>3</b> | <b>R basics</b>                                  | <b>11</b> |
| 3.1      | Adding libraries, part 1 . . . . .               | 12        |
| 3.2      | Adding libraries, part 2 . . . . .               | 13        |
| <b>4</b> | <b>Data journalism in the age of replication</b> | <b>14</b> |
| 4.1      | The stylebook . . . . .                          | 16        |
| 4.2      | Replication . . . . .                            | 16        |
| 4.3      | Goodbye Excel? . . . . .                         | 18        |
| 4.4      | "Receptivity ... is high" . . . . .              | 20        |
| 4.5      | Replication in notebooks . . . . .               | 20        |
| <b>5</b> | <b>Data, structures and types</b>                | <b>23</b> |
| 5.1      | Rows and columns . . . . .                       | 23        |
| 5.2      | Types . . . . .                                  | 24        |
| 5.3      | A simple way to get data . . . . .               | 25        |
| 5.4      | Cleaning the data . . . . .                      | 26        |
| <b>6</b> | <b>Aggregates</b>                                | <b>28</b> |
| 6.1      | Importing data . . . . .                         | 29        |
| 6.2      | Group by and count . . . . .                     | 31        |
| 6.3      | Other aggregates: Mean and median . . . . .      | 34        |
| 6.4      | Even more aggregates . . . . .                   | 36        |
| <b>7</b> | <b>Mutating data</b>                             | <b>38</b> |
| 7.1      | Another use of mutate . . . . .                  | 41        |

|           |   |            |
|-----------|---|------------|
| <b>8</b>  | <b>Working with dates</b>   | <b>43</b>  |
| 8.1       | The hard way . . . . .  | 43         |
| <b>9</b>  | <b>Filters and selections</b>   | <b>48</b>  |
| 9.1       | Combining filters . . . . .   | 50         |
| <b>10</b> | <b>Data Cleaning Part I: Data smells</b>                              | <b>52</b>  |
| 10.1      | Wrong Type . . . . .  | 53         |
| 10.2      | Missing Data . . . . .  | 54         |
| 10.3      | Gaps in data . . . . .  | 55         |
| 10.4      | Internal inconsistency . . . . .                                      | 56         |
| 10.5      | A Shortcut: Summary . . . . .   | 57         |
| <b>11</b> | <b>Data Cleaning Part II: Janitor</b>                                 | <b>58</b>  |
| 11.1      | Cleaning headers . . . . .  | 58         |
| 11.2      | Duplicates . . . . .  | 61         |
| 11.3      | Inconsistency . . . . .   | 62         |
| <b>12</b> | <b>Data Cleaning Part III: Open Refine</b>                            | <b>65</b>  |
| 12.1      | Refinr, Open Refine in R . . . . .                                    | 65         |
| 12.2      | More complex issues . . . . .   | 68         |
| 12.3      | Manually cleaning data with Open Refine . . . . .                     | 71         |
| <b>13</b> | <b>Cleaning Data Part IV: PDFs</b>                                    | <b>77</b>  |
| 13.1      | When it looks good, but goes wrong . . . . .                          | 77         |
| 13.2      | When it works well. . . . .   | 81         |
| 13.3      | Cleaning up the data in R . . . . .                                   | 82         |
| <b>14</b> | <b>Combining and joining</b>  | <b>85</b>  |
| 14.1      | Combining data . . . . .  | 85         |
| 14.2      | Joining data . . . . .  | 88         |
| <b>15</b> | <b>Scraping data with Rvest</b>                                       | <b>94</b>  |
| 15.1      | A more difficult example . . . . .                                    | 96         |
| <b>16</b> | <b>Advanced rvest</b>   | <b>100</b> |
| <b>17</b> | <b>Visualizing your data for reporting</b>                            | <b>104</b> |
| 17.1      | Bar charts . . . . .  | 105        |
| 17.2      | Line charts . . . . .   | 108        |
| <b>18</b> | <b>Writing with numbers</b>   | <b>113</b> |
| 18.1      | How to write about numbers without overwhelming with numbers. . . . . | 113        |
| 18.2      | When exact numbers matter . . . . .                                   | 114        |

|   |            |
|---|------------|
| 18.3 An example . . . . .                                     | 114        |
| <b>19 Ethics in data journalism</b>                           | <b>116</b> |
| 19.1 Problems . . . . .                                       | 116        |
| 19.2 The Googlebot . . . . .                                  | 117        |
| 19.3 Data Lifetimes . . . . .                                 | 118        |
| 19.4 You Are a Data Provider . . . . .                        | 119        |
| 19.5 Ethical Data . . . . .                                   | 120        |
| <b>20 Installations</b>                                       | <b>122</b> |
| 20.1 Part 1: Update and patch your operating system . . . . . | 122        |
| 20.2 Part 2: Install R and R Studio . . . . .                 | 127        |
| 20.3 Part 3: Installing R libraries . . . . .                 | 129        |
| 20.4 Part 4: Install Slack . . . . .                          | 130        |
| 20.5 Part 5: Install the tutorials . . . . .                  | 131        |

# 1 Introduction

If you were at all paying attention in pre-college science classes, you have probably seen this equation:

$$d = rt \text{ or } \text{distance} = \text{rate} \times \text{time}$$

In English, that says we can know how far something has travelled if we know how fast it's going and for how long. If we multiply the rate by the time, we'll get the distance.

If you remember just a bit about algebra, you know we can move these things around. If we know two of them, we can figure out the third. So, for instance, if we know the distance and we know the time, we can use algebra to divide the distance by the time to get the rate.

$$d/t = r \text{ or } \text{distance}/\text{time} = \text{rate}$$

In 2012, the South Florida Sun Sentinel found a story in this formula.

People were dying on South Florida tollways in terrible car accidents. What made these different from other car fatal car accidents that happen every day in the US? Police officers driving way too fast were causing them.

But do police regularly speed on tollways or were there just a few random and fatal exceptions?

Thanks to Florida's public records laws, the Sun Sentinel got records from the toll transponders in police cars in south Florida. The transponders recorded when a car went through a given place. And then it would do it again. And again.

Given that those places are fixed – they're toll plazas – and they had the time it took to go from one toll plaza to another, they had the distance and the time.

[It took high school algebra to find how fast police officers were driving. And the results were shocking.](#)

Twenty percent of police officers had exceeded 90 miles per hour on toll roads. In a 13-month period, officers drove between 90 and 110 mph more than 5,000 times. And these were just instances found on toll roads. Not all roads have tolls.

The story was a stunning find, and the newspaper documented case after case of police officers violating the law and escaping punishment. And, in 2013, they won the Pulitzer Prize for Public Service.

All with simple high school algebra.

## 1.1 Modern data journalism

It's a single word in a single job description, but a BuzzFeed job posting in 2017 is another indicator in what could be a profound shift in how data journalism is both practiced and taught.

“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.

And then there’s this:

**“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.”**

This is not the data journalism of 20 years ago. When it started, it was a small group of people in newsrooms using spreadsheets and databases. Data journalism now encompasses programming for all kinds of purposes, product development, user interface design, data visualization and graphics on top of more traditional skills like analyzing data and writing stories.

In this book, you’ll get a taste of modern data journalism through 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. They might seem like two different sides of the brain – mutually exclusive skills. They aren’t. I’m confident you’ll see programming is a creative endeavor and storytelling can be analytical.

Combining them together has the power to change policy, expose injustice and deeply inform.

## 1.2 Installations

This book is all in the R statistical language. To follow along, you’ll do the following:

1. Install the R language on your computer. Go to the [R Project website](#), click download R and select a mirror closest to your location. Then download the version for your computer.

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 in your R Studio. When you see that, you'll know what to do.

## 1.3 About this book

This book is the collection of class materials for the author's Data Journalism class at the University of Nebraska-Lincoln's College of Journalism and Mass Communications. 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 author is copyright Matt Waite 2024.
- 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 Bookdown – is [open sourced on Github](#). Pull requests welcomed!

## 1.4 What we'll cover

- Public records and open data
- R Basics
- Replication
- Data basics and structures
- Aggregates
- Mutating
- Working with dates
- Filters
- Cleaning I: Data smells
- Cleaning II: Janitor
- Cleaning III: Open Refine
- Cleaning IV: Pulling Data from PDFs

- Joins
- Basic data scraping
- Intermediate 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
- Text analysis
- Advanced analysis: Correlations and regressions
- Advanced analysis: Logistic regression
- Writing with and about data
- Data journalism ethics



## 2 Public records

Public records are the lifeblood of investigative reporting. They carry their own philosophical framework, in a manner of speaking.

- Sunlight is the best disinfectant. Corruption hides in the shadows.
- You paid for it with your taxes. It should be yours (with exceptions).
- Journalism with a capital J is about holding the powerful accountable for their actions.

Keeping those things in mind as you navigate public records is helpful.

### 2.1 Federal law

Your access to public records and public meetings is a matter of the law. As a journalist, it is your job to know this law better than most lawyers. Which law applies depends on which branch of government you are asking.

The Federal Government is covered by the Freedom of Information Act, or FOIA. FOIA is not a universal term. Do not use it if you are not talking to a federal agency. FOIA is a beacon of openness to the world. FOIA is deeply flawed and frustrating.

Why?

- There is no real timetable with FOIA. Requests can take months, even years.
- As a journalist, you can ask that your request be expedited.
- Guess what? That requires review. More delays.
- Exemptions are broad. National security, personal privacy, often overused.
- Denied? You can appeal. More delays.

The law was enacted in 1966, but it's still poorly understood by most federal employees, if not outright flouted by political appointees. Lawsuits are common.

Post 9/11, the Bush administration rolled back many agency rules. Obama ordered a "presumption of openness" but followed it with some of the most restrictive policies ever seen. The Trump Administration, similar to the Obama administration, claims to be the most transparent administration, but has steadily removed records from open access and broadly denied access to records.

Result? FOIA is in trouble.

[SPJ is a good resource.](#)

## 2.2 State law

States are – generally – more open than the federal government. The distance between the government and the governed is smaller. Some states, like Florida and Texas, are very open. Others, like Virginia and Pennsylvania, are not.

Nebraska? Somewhere in the middle. Better than some, worse than others.

Under Nebraska law:

- You are entitled to see and copy public records.
- They can charge you a fee for those copies.
- They do not have to give you records in a format different from what they keep them in.

With a written request, Nebraska public officials have **four days** to respond. If Nebraska public officials deny your request, they must do so **in writing** specifying their reasons. If it will take more than four days, they must tell you, in writing, why and how long it will take.

[The Reporters Committee For Freedom of the Press is a good resource.](#)

Please and thank you will get you more records than any lawyer or well written request. When requesting data, you are going to scare the press office and you are going to confuse the agency lawyer. Request to have their data person on the phone.

**Be. Nice.**

Hunting for records is like any other kind of reporting – you have to do research. You have to ask questions. What records do you keep? For how long?

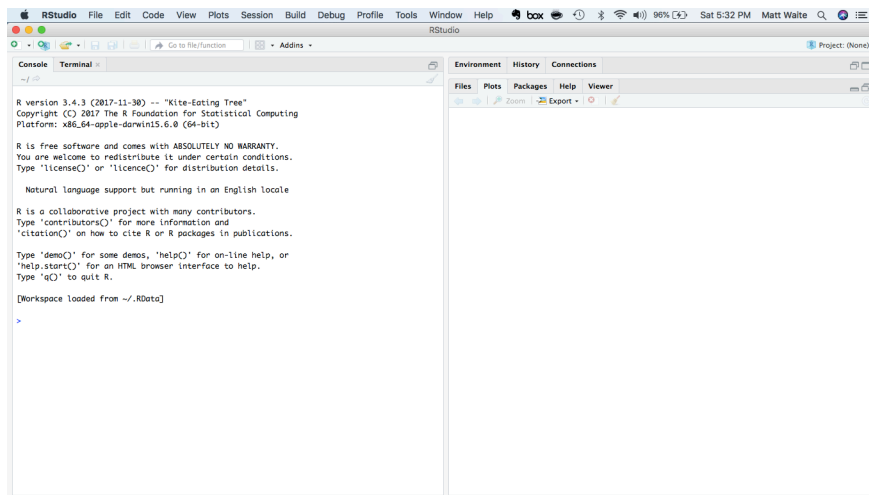
A good source of info? Records retention schedules, often required by law or administrative rule at an agency. [Nebraska's is particularly good.](#)

For an example, let's say we wanted to look at nursing homes in Nebraska, because they're closing down quickly. If we just wanted a data set of licensed nursing homes in the state, our options online are ... not good. The Department of Health and Human Services [publishes a list online](#), but it's a horribly formatted PDF.

But what does the agency keep? Looking at [DHHS's public health departments records retention schedule](#), you'll find a lot more. And that opens doors for public records requests.

## 3 R basics

R is a programming language, one specifically geared toward statistical analysis. Like all programming languages, it has certain built-in functions and you can interact with it in multiple ways. The first, and most basic, is the console.



Think of the console like talking directly to R. It's direct, but it has some drawbacks and some quirks we'll get into later. For now, try typing this into the console and hit enter:

```
2+2
```

```
[1] 4
```

Congrats, you've run some code. It's not very complex, and you knew the answer before hand, but you get the idea. We can compute things. We can also store things. **In programming languages, these are called variables.** We can assign things to variables using `<-`. And then we can do things with them. **The `<-` is called an assignment operator.**

Try this in your console.

```
number <- 2
```

```
number * number
```

```
[1] 4
```

Now assign a different number to the variable `number`. Try run `number * number` again. Get what you expected?

We can have as many variables as we can name. **We can even reuse them (but be careful you know you're doing that or you'll introduce errors).** Try this in your console.

```
firstnumber <- 1
secondnumber <- 2

(firstnumber + secondnumber) * secondnumber
```

```
[1] 6
```

**We can store anything in a variable.** A whole table. An array of numbers. A single word. A whole book. All the books of the 18th century. They're really powerful. We'll explore them at length.

### 3.1 Adding libraries, part 1

The real strength of any given programming language is the external libraries that power it. The base language can do a lot, but it's the external libraries that solve many specific problems – even making the base language easier to use.

For this class, we're going to need several external libraries.

The first library we're going to use is called Swirl. So in the console, type `install.packages('swirl')` and hit enter. That installs swirl.

Now, to use the library, type `library(swirl)` and hit enter. That loads swirl. Then type `swirl()` and hit enter. Now you're running swirl. Follow the directions on the screen. When you are asked, you want to install course 1 R Programming: The basics of programming in R. Then, when asked, you want to do option 1, R Programming, in that course.

When you are finished with the course – it will take just a few minutes – type 0 to exit (it will not be clear that's what you do when you are done).

## 3.2 Adding libraries, part 2

We'll mostly use two libraries for analysis – `dplyr` and `ggplot2`. To get them, and several other useful libraries, we can install a single collection of libraries called the tidyverse. Type this into your console: `install.packages('tidyverse')`

**NOTE:** This is a pattern. You should always install libraries in the console.

Then, to help us with learning and replication, we're going to use R Notebooks. So we need to install that library. Type this into your console: `install.packages('rmarkdown')`

## 4 Data journalism in the age of replication

It's a single word in a single job description, but a BuzzFeed job posting in 2017 is another indicator in what could be a profound shift in how data journalism is both practiced and taught.

“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.

And then there's this:

**“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.”**

The word you're seeing more and more of? Reproducible. And it started in earnest this summer when data journalism crossed a major threshold in American journalism: It now has its own section in the Associated Press Stylebook.

“Data journalism has become a staple of reporting across beats and platforms,” the Data Journalism section of the Stylebook opens. “The ability to analyze quantitative information and present conclusions in an engaging and accurate way is no longer the domain of specialists alone.”

The AP's Data Journalism section discusses how to request data and in what format, guidelines for scraping data from websites with automation, the ethics of using leaked or hacked data and other topics long part of data journalism conference talks.

But the third page of the section contains perhaps the most profound commandment: **“As a general rule, all assertions in a story based on data analysis should be reproducible. The methodology description in the story or accompanying materials should provide a road map to replicate the analysis.”**

Reproducible research – replication – is a cornerstone of scientific inquiry. Researchers across a range of academic disciplines use methods to find new knowledge and publish it in peer reviewed journals. And, when it works, other researchers take that knowledge and try it with their own samples in their own locations. Replication studies exist to take something from an Interesting Finding to a Theory and beyond.

It doesn't always work.

Replication studies aren't funded at nearly the level as new research. And, to the alarm of many, scores of studies can't be replicated by others. Researchers across disciplines are finding that when their original studies are replicated, flaws are found, or the effects found aren't as strong as the original. Because of this, academics across a number of disciplines have written about a replication crisis in their respective fields, particularly psychology, social science and medical research.

In Chapter 1 of the *New Precision Journalism*, Phil Meyer wrote that "we journalists would be wrong less often if we adapted to our own use some of the research tools of the social scientists."

Meyer would go on to write about how computers pouring over datasets too large to crunch by hand had changed social science from a discipline with "a few data and a lot of interpretation" into a much more meaningful and powerful area of study. If journalists could become comfortable with data and some basic statistics, they too could harness this power.

"It used to be said that journalism is history in a hurry," Meyer wrote. "The argument of this book is that to cope with the acceleration of social change in today's world, journalism must become social science in a hurry."

He wrote that in 1971. It might as well have been yesterday.

Journalism doesn't have a history of replication, but the concerns about credibility are substantially greater. Trust in media is at an all time low and shows no signs of improving. While the politics of the day have quite a bit to do with this mistrust of media, being more transparent about what journalists do can't hurt.

The AP's commandment that *Thou Must Replicate Your Findings* could, if taken seriously by the news business, have substantial impacts on how data journalism gets done in newsrooms and how data journalism gets taught, both at professional conferences and universities.

How? Two ways.

- The predominant way that data journalism gets done in a newsroom is through simple tools like Microsoft Excel or Google Sheets. Those simple tools, on their own, lack significant logging functions, meaning journalists will have to maintain detailed logs of what they did so any analysis can be replicated.
- The predominant way that data journalism gets taught – both in professional settings and at most universities – doesn't deal with replication at all. The tools and the training stress *Getting Things Done* – an entirely logical focus for a deadline driven business. The choices of tools – like spreadsheets – are made to get from data to story as quick as possible, without frightening away math and tech phobic students.

If the AP’s replication rules are to be followed, journalism needs to become much more serious about the tools and techniques used to do data journalism. The days of Point and Click tools to do Quick and Dirty analysis that get published are dying. The days of formal methods using documented steps are here.

## 4.1 The stylebook

Troy Thibodeaux, the editor of the AP’s data journalism team, said the stylebook entry started when the data team found themselves answering the same questions over and over. With a grant from the Knight Foundation, the team began to document their own standards and turn that into a stylebook section.

From the beginning, they had a fairly clear idea of what they wanted to do – think through a project and ask what the frequently asked questions are that came up. It was not going to be a soup-to-nuts guide to how to do a data project.

When the section came out, eyebrows went up on the replication parts, surprising Thibodeaux.

“From our perspective, this is a core value for us,” he said. “Just for our own benefit, we need to be able to have someone give us a second set of eyes. We benefit from that every day. We catch things for each other.”

Thibodeaux said the AP data team has two audiences when it comes to replication – they have the readers of the work, and members of the collective who may want to do their own work with the data.

“This is something that’s essential to the way we work,” he said. “And it’s important in terms of transparency and credibility going forward. We thought it would be kind of unexceptionable.”

## 4.2 Replication

Meyer, now 86, said he’s delighted to see replication up for discussion now, but warned that we shouldn’t take it too far.

“Making the analysis replicable was something I worried about from the very beginning,” he wrote in an email. So much so that in 1967, after publishing stories from his landmark survey after the Detroit riots, he shipped the data and backup materials about it to a social science data repository at the University of North Carolina.



And, in doing so, he opened the door to others replicating his results. One scholar attempted to find fault with Meyer's analysis by slicing the data ever thinner until the differences weren't significant – gaming the analysis to criticize the stories.

Meyer believes replication is vitally important, but doesn't believe it should take on the trappings of science replication, where newsrooms take their own samples or re-survey a community. That would be prohibitively expensive.

But journalists should be sharing their data and analysis steps. And it doesn't need to be complicated, he said.

"Replication is a theoretical standard, not a requirement that every investigator duplicate his or her own work for every project," he said. "Giving enough information in the report to enable another investigator to follow in your footsteps is enough. Just telling enough to make replication possible will build confidence."

But as simple as that sounds, it's not so simple. Ask social scientists.

Andrew Gelman, a professor of statistics and political science and director of the Applied Statistics Center at Columbia University, wrote in the journal *CHANCE* in February that difficulties with replication in empirical research are pervasive.

"When an outsider requests data from a published paper, the authors will typically not post or send their data files and code, but instead will point to their sources, so replicators have to figure out exactly what to do from there," Gelman wrote. "End-to-end replicability is not the norm, even among scholars who actively advocate for the principles of open science."

So goes science, so goes journalism.

Until a recent set of exceptions, journalists rarely shared data. The "nerd box" – a sidebar story that explains how a news organization did what they did – is a term that first appeared on NICAR-L, a email listserv of data journalists, in the 1990s.

It was a form born in print.

As newsrooms adapted to the internet, some news organizations began linking to their data sources if they were online. Often, the data used in stories were obtained through records requests. Sometimes, reporters created the data themselves.

Journalism, more explicitly than science, is a competitive business. There have been arguments that nerd boxes and downloadable links give too much away to competitors.

Enter the AP Stylebook.

The AP Stylebook argues explicitly for both internal and external replication. Externally, they argue that the **"methodology description in the story or accompanying materials should provide a road map to replicate the analysis"**, meaning someone else could do the replication post publication.

Internally, the AP Stylebook says: **“If at all possible, an editor or another reporter should attempt to reproduce the results of the analysis and confirm all findings before publication.”**

There are two problems here.

First is that journalism, unlike science, has no history of replication. There is no Scientific Method for stories. There is no Research Methods class taught at every journalism school, at least not where it comes to writing stories. And, beyond that, journalism school isn’t a requirement to get into the news business. In other words, journalism lacks the standards other disciplines have.

The second problem is, in many ways, worse: Except for the largest newsrooms, most news organizations lack editors who could replicate the analysis. Many don’t have a second person who would know what to do.

Not having a second set of eyes in a newsroom is a problem, Thibodeaux acknowledges. Having a data journalism team “is an incredible luxury” at the AP, he said, and their rule is nothing goes on the wire without a second set of eyes.

Thibodeaux, for his part, wants to see fewer “lone nerds in the corner” – it’s too much pressure. That person gets too much credibility from people who don’t understand what they do, and they get too much blame when a mistake is made.

So what would replication look like in a newsroom? What does this mean for how newsrooms do data journalism on deadline? And what does this mean for how data journalism is being taught, particularly at a time when only half of accredited journalism programs teach any data journalism at all?

Are we walking ourselves into our own replication crisis?

## 4.3 Goodbye Excel?

For decades, Excel has been the gateway drug for data journalists, the Swiss Army knife of data tools, the One Tool You Can’t Live Without. Investigative Reporters and Editors, an organization that trains investigative journalists, have built large amounts of their curricula around Excel. Of the journalism schools that teach data journalism, most of them begin and end with spreadsheets.

The Stylebook says at a minimum, today’s data journalists should keep a log that details:

- The source of the data, making sure to work on a copy of the data and not the original file.
- Data dictionaries or any other supporting documentation of the data.

- **“Description of all steps required to transform the data and perform the analysis.”**

The trouble with Excel is, unless you are keeping meticulous notes on what steps you are taking, there’s no way to keep track. Many data journalists will copy and paste the values of a formula over the formula itself to prevent Excel from fouling up cell references when moving data around – a practical step that also cuts off another path to being able to replicate the results.

An increasing number of data journalists are switching to tools like analysis notebooks, which use languages like Python and R, to document their work. The notebooks, generally speaking, allow a data journalist to mix code and explanation in the same document.

Combined with online sharing tools like GitHub, analysis notebooks seem to solve the problem of replication. But the number using them is small compared to those using spreadsheets. Recent examples of news organizations using analysis notebooks include the [Los Angeles Times](#), the [New York Times](#), [FiveThirtyEight](#), and [Buzzfeed](#).

Peter Aldous, a data journalist at BuzzFeed recently published a story about how the online news site used machine learning to find airplanes being used to spy on people in American cities. Published with the story is the code Aldous used to build his case.

“I think of it this way: As a journalist, I don’t like to simply trust what people tell me. Sometimes sources lie. Sometimes they’re just mistaken. So I like to verify what I’m told,” he wrote in an email. “By the same token, why should someone reading one of my articles believe my conclusions, if I don’t provide the evidence that explains how I reached them?”

The methodology document, associated code and source data took Aldous a few hours to create. The story, from the initial data work through the reporting required to make sense of it all, took a year. Aldous said there wasn’t a discussion about if the methodology would be published because it was assumed – “it’s written into our DNA at BuzzFeed News.”

“My background is in science journalism, and before that (way back in the 1980s) in science,” Aldous said. “In science, there’s been a shift from descriptive methods sections to publishing data and analysis code for reproducible research. And I think we’re seeing a similar shift in data journalism. Simply saying what you’ve done is not as powerful as providing the means for others to repeat and build on your work.”

Thibodeaux said that what BuzzFeed and others do with analysis notebooks and code repositories that include their data is “lovely.”

“That to me is the shining city on the hill,” Thibodeaux said. “We’re not going to get there, and I don’t think we have to for every story and every use case, and I don’t think it’s necessarily practical for every person working with data to get to that point.”

There’s a wide spectrum of approaches that still gets journalists to the essence of what the stylebook is trying to do, Thibodeaux said. There are many tools, many strategies, and the

AP isn't going to advocate for any single one of them, he said. They're just arguing for transparency and replicability, even if that means doing more work.

"There's a certain burden that comes with transparency," he said. "And I think we have to accept that burden."

The question, Thibodeaux said, is what is sufficient? What's enough transparency? What does someone need for replicability?

"Maybe we do have to set a higher standard – the more critical the analysis is to the story, and the more complex that analysis is, that's going to push the bar on what is a sufficient methodology statement," he said. "And it could end up being a whole code repo in order to just say, this isn't black magic, here's how we got it if you're so interested."

## 4.4 "Receptivity ... is high"

Though written almost half a century ago, Meyer foresaw how data journalism was going to arrive in the newsroom.

"For the new methods to gain currency in journalism, two things must happen," he wrote. "Editors must feel the need strongly enough to develop the in-house capacity for systematic research ... The second need, of course, is for the editors to be able to find the talent to fill this need."

Meyer optimistically wrote that journalism schools were prepared to provide that talent – they were not then, and only small handful are now – but students were unlikely to be drawn to these new skills if they didn't see a chance to use those skills in their careers.

It's taken 45 years, but we are now at this point.

"The potential for receptivity, especially among the younger generation of newspaper managers, is high," Meyer wrote.

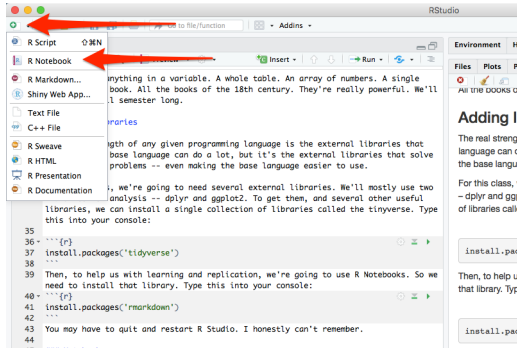
## 4.5 Replication in notebooks

For our purposes in this book, replication requires two things from you, the student: What and why. What is this piece of code doing, and why are you doing that here and now? What lead you to this place? That you can copy and paste code from this book or the internet is not impressive. What is necessary for learning is that you know what a piece of code is doing a thing and why you want to do that thing here.

In an R Notebook, there are two blocks: A block that uses markdown, which has no special notation, and a code block. The code blocks can run multiple languages inside R Studio, including Python, a general purpose scripting language; and SQL, or Structured Query Language, the language of databases.

For the rest of the class, we're going to be working in notebooks. In notebooks, you will both run your code and explain each step, much as I am doing here.

To start a notebook, you click on the green plus in the top left corner and go down to R Notebook. Do that now.



You will see that the notebook adds a lot of text for you. It tells you how to work in notebooks – and you should read it. The most important parts are these:

To add text, simply type. To add code you can click on the *Insert* button on the toolbar or by pressing *Cmd+Option+I* on Mac or *Ctl+Alt+I* on Windows.

Highlight all that text and delete it. You should have a blank document. This document is called a R Markdown file – it's a special form of text, one that you can style, and one you can include R in the middle of it. Markdown is a simple markup format that you can use to create documents. So first things first, let's give our notebook a big headline. Add this:

**# My awesome notebook**

Now, under that, without any markup, just type This is my awesome notebook.

Under that, you can make text bold by writing It is **\*\*really\*\*** awesome.

If you want it italics, just do this on the next line: No, it's *\_really\_* awesome. I swear.

To see what it looks like without the markup, click the Preview or Knit button in the toolbar. That will turn your notebook into a webpage, with the formatting included.

Throughout this book, we're going to use this markdown to explain what we are doing and, more importantly, why we are doing it. Explaining your thinking is a vital part of understanding what you are doing.

That explanation, plus the code, is the real power of notebooks. To add a block of code, follow the instructions from above: click on the *Insert* button on the toolbar or by pressing *Cmd+Option+I* on Mac or *Ctl+Alt+I* on Windows.

In that window, use some of the code from above and add two numbers together. To see it run, click the green triangle on the right. That runs the chunk. You should see the answer to your addition problem.

And that, just that, is the foundation you need to start this book.

## 5 Data, structures and types

Data are everywhere (and data is plural of datum, thus the use of are in that statement). It surrounds you. Every time you use your phone, you are creating data. Lots of it. Your online life. Any time you buy something. It's everywhere. News, like life, is no different. Modernity is drowning in data, and more comes along all the time.

In news, and in this class, we'll be dealing largely with two kinds of data: **event level data** and **summary data**. It's not hard to envision event level data. A car accident. A crime. A fire. They are the events that make up the whole. Combine them together – summarize them – and you'll have some notion of how the year went. What we usually see is summary data – who wants to scroll through 365 days of crime data to figure out if crime was up or down?

To start with, we need to understand the shape of data.

### 5.1 Rows and columns

Data, oversimplifying it a bit, is information organized. Generally speaking, it's organized into rows and columns. Rows, generally, are individual elements. A crime. A county. An accident. Columns, generally, are components of the data, sometimes called variables. So if each row is a crime, the first column might be the type. The second is the date and time. The third is the location. And so on.

| G  | Date       | Opp                       | W/L | Column |    |    | FG% | 3P    | 3PA | 3P% |       |
|----|------------|---------------------------|-----|--------|----|----|-----|-------|-----|-----|-------|
| 1  | 2018-11-06 | Mississippi Valley State  | W   | 106    | 37 | 38 | 69  | 0.551 | 15  | 37  | 0.405 |
| 2  | 2018-11-11 | Southeastern Louisiana    | W   | 87     | 35 | 31 | 55  | 0.564 | 8   | 25  | 0.32  |
| 3  | 2018-11-14 | Seton Hall                | W   | 80     | 57 | 26 | 63  | 0.413 | 8   | 23  | 0.348 |
| 4  | 2018-11-19 | N Missouri State          | W   | 85     | 62 | 31 | 61  | 0.508 | 13  | 33  | 0.394 |
| 5  | 2018-11-20 | N Texas Tech              | L   | 52     | 70 | 17 | 48  | 0.354 | 5   | 23  | 0.217 |
| 6  | 2018-11-24 | Western Illinois          | W   | 73     | 49 | 29 | 63  | 0.46  | 4   | 20  | 0.2   |
| 7  | 2018-11-26 | @ Clemson                 | W   | 68     | 66 | 26 | 55  | 0.473 | 7   | 22  | 0.318 |
| 8  | 2018-12-02 | Rows                      | W   | 75     | 60 | 22 | 46  | 0.478 | 6   | 14  | 0.429 |
| 9  | 2018-12-05 | @ Minnesota               | L   | 78     | 85 | 28 | 61  | 0.459 | 7   | 21  | 0.333 |
| 10 | 2018-12-08 | Creighton                 | W   | 94     | 75 | 32 | 60  | 0.533 | 14  | 27  | 0.519 |
| 11 | 2018-12-16 | N Oklahoma State          | W   | 79     | 56 | 25 | 53  | 0.472 | 8   | 18  | 0.444 |
| 12 | 2018-12-22 | Cal State Fullerton       | W   | 86     | 62 | 26 | 59  | 0.441 | 10  | 22  | 0.455 |
| 13 | 2018-12-29 | Southwest Minnesota State | W   | 79     | 38 | 29 | 66  | 0.439 | 11  | 24  | 0.458 |
| 14 | 2019-01-02 | @ Maryland                | L   | 72     | 74 | 24 | 51  | 0.471 | 9   | 22  | 0.409 |
| 15 | 2019-01-06 | @ Iowa                    | L   | 84     | 93 | 30 | 65  | 0.462 | 4   | 23  | 0.174 |
| 16 | 2019-01-10 | Penn State                | W   | 70     | 64 | 27 | 55  | 0.491 | 9   | 23  | 0.391 |
| 17 | 2019-01-14 | @ Indiana                 | W   | 66     | 51 | 27 | 60  | 0.45  | 8   | 24  | 0.333 |
| 18 | 2019-01-17 | Michigan State            | L   | 64     | 70 | 22 | 67  | 0.328 | 5   | 26  | 0.192 |
| 19 | 2019-01-21 | @ Rutgers                 | L   | 69     | 76 | 25 | 60  | 0.417 | 9   | 22  | 0.409 |

One of the critical components of data analysis, especially for beginners, is having a mental picture of your data. What does each row mean? What does each column in each row signify? How many rows do you have? How many columns?

EXERCISE: I love orange Skittles. What are my chances of getting more orange Skittles than other colors in a fun sized packet? Each person in the class must track their package and everyone else using a spreadsheet. What differences between sheets emerge? What similarities?

## 5.2 Types

There are scores of data types in the world, and R has them. In this class, we're primarily going to be dealing with data frames, and each element of our data frames will have a data type.

Typically, they'll be one of four types of data:

- Numeric: a number, like the number of car accidents in a year or the number of journalism majors.
- Character: Text, like a name, a county, a state.
- Date: Fully formed dates – 2019-01-01 – have a special date type. Elements of a date, like a year (ex. 2019) are not technically dates, so they'll appear as numeric data types.



- Logical: Rare(ish), but every now and then we'll have a data type that's Yes or No, True or False, etc.

**Question:** Is a zip code a number? Is a jersey number a number? Trick question, because the answer is no. Numbers are things we do math on. If the thing you want is not something you're going to do math on – can you add two phone numbers together? – then make it a character type. If you don't, most every software system on the planet will drop leading zeros. For example, every zip code in Boston starts with 0. If you record that as a number, your zip code will become a four digit number, which isn't a zip code anymore.

## 5.3 A simple way to get data

The hardest part of doing data journalism is often getting the data. In news, there's scores of organizations and agencies collecting data, and zero standards on how it's being collected.

If we're lucky – huge IF in news – the data we want is in a downloadable format. If we're a little less lucky, there's a way to get the data on the web. And maybe that data is in a simple table. If so, we can pull that data directly into Google Sheets.

The Lincoln Police Department publishes a daily summary of calls. Some days – like when it snows – that data becomes news. So let's pretend that it snowed today and we need to see how many accidents the Lincoln Police responded to and what percentage of their call load that represents.

Open a browser and go to the [LPD's log page](#). Now, in a new tab, log into Google Docs/Drive and open a new spreadsheet. In the first cell of the first row, copy and paste this formula in:

```
=importHTML("http://cjis.lincoln.ne.gov/~lpd/cfstoday.htm","table",1)
```

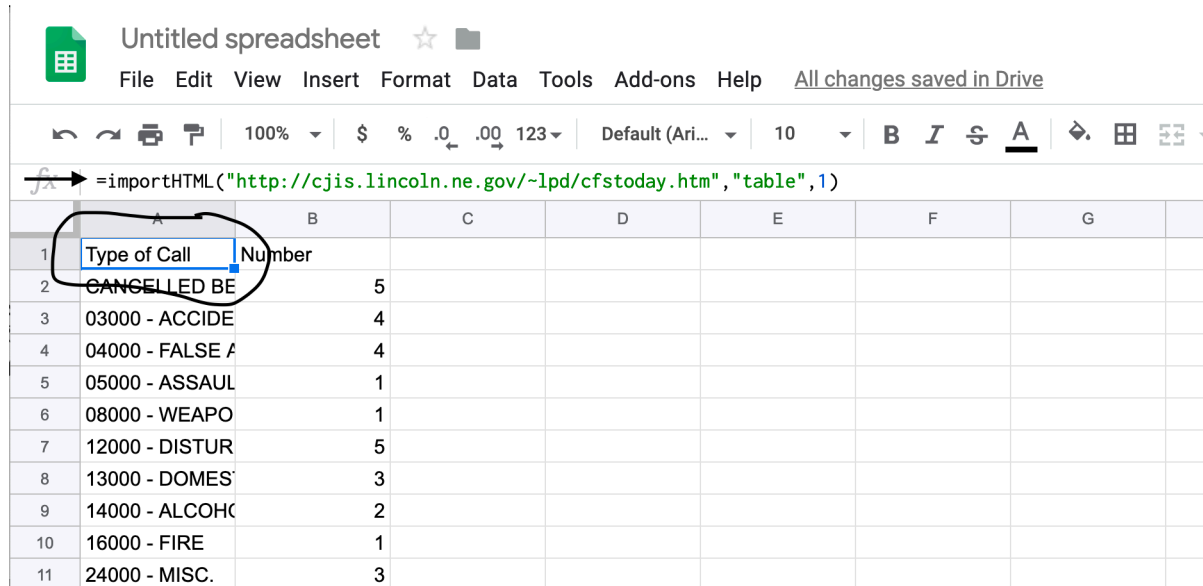
This is **function**, with three **inputs**. The function is `importHTML` and the three inputs in order are the url of the page, the HTML tag we're after (a

tag in our case) and the number of the tag you're after. So our function says go to the LPD page and get the first table tag you find. Fortunately for us, there's only one.

If your version worked right, you've got the data from that page in a spreadsheet.

## 5.4 Cleaning the data

The first thing we need to do is recognize that we don't have data, really. We have the results of a formula. You can tell by putting your cursor on that field, where you'll see the formula again. This is where you'd look:



|    | A               | B      | C | D | E | F | G |
|----|-----------------|--------|---|---|---|---|---|
| 1  | Type of Call    | Number |   |   |   |   |   |
| 2  | CANCELLED BE    | 5      |   |   |   |   |   |
| 3  | 03000 - ACCIDE  | 4      |   |   |   |   |   |
| 4  | 04000 - FALSE A | 4      |   |   |   |   |   |
| 5  | 05000 - ASSAUL  | 1      |   |   |   |   |   |
| 6  | 08000 - WEAPO   | 1      |   |   |   |   |   |
| 7  | 12000 - DISTUR  | 5      |   |   |   |   |   |
| 8  | 13000 - DOMES   | 3      |   |   |   |   |   |
| 9  | 14000 - ALCOHC  | 2      |   |   |   |   |   |
| 10 | 16000 - FIRE    | 1      |   |   |   |   |   |
| 11 | 24000 - MISC.   | 3      |   |   |   |   |   |

The solution is easy:

Edit > Select All or type command/control A Edit > Copy or type command/control C Edit > Paste Special > Values Only or type command/control shift V

You can verify that it worked by looking in that same row 1 column A, where you'll see the formula is gone.

Untitled spreadsheet ☆

File Edit View Insert Format Data Tools Add-ons Help [All changes saved in Drive](#)

100% \$ % .0 .00 123 Default (Ari... 10 B I S A

Type of Call

|    | A               | B      | C | D | E | F | G |
|----|-----------------|--------|---|---|---|---|---|
| 1  | Type of Call    | Number |   |   |   |   |   |
| 2  | CANCELLED BE    | 5      |   |   |   |   |   |
| 3  | 03000 - ACCIDE  | 4      |   |   |   |   |   |
| 4  | 04000 - FALSE A | 4      |   |   |   |   |   |
| 5  | 05000 - ASSAUL  | 1      |   |   |   |   |   |
| 6  | 08000 - WEAPO   | 1      |   |   |   |   |   |
| 7  | 12000 - DISTUR  | 5      |   |   |   |   |   |
| 8  | 13000 - DOMES   | 3      |   |   |   |   |   |
| 9  | 14000 - ALCOHC  | 2      |   |   |   |   |   |
| 10 | 16000 - FIRE    | 1      |   |   |   |   |   |
| 11 | 24000 - MISC.   | 3      |   |   |   |   |   |
| 12 | 27000 - NARCO   | 5      |   |   |   |   |   |

Now you have data, but look closely. At the bottom of the data, you have the total number of calls. More often than not, and particularly the deeper into this book we go, you want to delete that. So click on the number next to that Total Calls line to highlight it and go up to Edit > Delete Row XX where XX is the row number.

After you've done that, you can export it for use in R. Go to File > Download as > Comma Separated Values.

## 6 Aggregates

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

Base R does a lot, but there are a mountain of external libraries that do things to make R better/easier/more fully featured. We already installed the tidyverse – or you should have if you followed the instructions for the last assignment – which isn't exactly a library, but a collection of libraries. Together, they make up the tidyverse. Individually, they are extraordinarily useful for what they do. We can load them all at once using the tidyverse name, or we can load them individually. Let's start with individually.

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. Good practice – one I'm going to insist on – is that you put all your library steps at the top of your notebooks.

That code looks like this:

```
library(readr)
```

To load them both, you need to do this:

```
library(readr)
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

```
filter, lag
```

The following objects are masked from 'package:base':

```
intersect, setdiff, setequal, union
```

But, because those two libraries – and several others that we’re going to use over the course of this class – are so commonly used, there’s a shortcut to loading all of the libraries we’ll need:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v forcats   1.0.0      v stringr   1.5.0
v ggplot2   3.4.4      v tibble    3.2.1
v lubridate 1.9.2      v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

You can keep doing that for as many libraries as you need. I’ve seen notebooks with 10 or more library imports.

## 6.1 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 data from a csv file – a comma-separated values file.

The CSV file we’re going to read from is a [Nebraska Game and Parks Commission dataset](#) of confirmed mountain lion sightings in Nebraska. There are, on occasion, fierce debates about mountain lions and if they should be hunted in Nebraska. This dataset can tell us some interesting things about that debate.

So step 1 is to import the data. The code looks *something* like this, but hold off copying it just yet:

```
mountainlions <- read_csv("~/Documents/Data/mountainlions.csv")
```

Let’s unpack that.

The first part – mountainlions – is the name of your variable. A variable is just a name of a thing. In this case, our variable is a dataframe, which is R’s way of storing data. We can call

this whatever we want. I always want to name dataframes after what is in it. In this case, we're going to import a dataset of mountain lion sightings from the Nebraska Game and Parks Commission. **Variable names, by convention are one word all lower case.** You can end a variable with a number, but you can't start one with a number.

The `<-` bit, you'll recall from the basics, 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." So we're creating an empty vessel called `mountainlions` and stuffing all this data into it.

The `read_csv` bits are pretty obvious, except for one thing. What happens in the quote marks is the path to the data. In there, I have to tell R where it will find the data.

The easiest thing to do, if you are confused about how to find your data, is to put your data in the same folder as as your notebook (you'll have to save that notebook first). If you do that, then you just need to put the name of the file in there (`mountainlions.csv`).

In my case, the file path I've got starts with a `~` character. That's a shortcut for my home directory. It's the same on your computer. Your home directory is where your Documents, Desktop and Downloads directories are. I've got a folder called Documents in my home directory, and in there is a folder called Data that has the file called `mountainlions.csv` in it. Thus, `~/Documents/Data/mountainlions.csv`

Some people – insane people – leave the data in their downloads folder. The data path then would be `~/Downloads/nameofthedatafilehere.csv` on PC or Mac.

A quick way to find your data file? The tab key. If you start your code `dataframenamehere <- read_csv("")` and after typing the first quote mark you hit tab, it will show you the files in the folder you are in. With that, you can start to narrow in on where you need to go.

**So what you put in your import step will be different from mine.** Your first task is to import the data. Here's mine. Use the tab key to find your data file and get the correct path.

```
mountainlions <- read_csv("data/mountainlions.csv")
```

```
Rows: 393 Columns: 4
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (3): Cofirm Type, COUNTY, Date
```

```
dbl (1): ID
```

```
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.
```

Now we can inspect the data we imported. What does it look like? To do that, we use `head(mountainlions)` to **show the headers and the first six rows of data**. If we wanted to see them all, we could just simply enter `mountainlions` and run it.

To get the number of records in our dataset, we run `nrow(mountainlions)`

```
head(mountainlions)
```

```
# A tibble: 6 x 4
  ID `Cofirm Type` COUNTY      Date
<dbl> <chr>      <chr>      <chr>
1     1 Track      Dawes      9/14/91
2     2 Mortality  Sioux     11/10/91
3     3 Mortality  Scotts Bluff 4/21/96
4     4 Mortality  Sioux     5/9/99
5     5 Mortality  Box Butte   9/29/99
6     6 Track      Scotts Bluff 11/12/99
```

```
nrow(mountainlions)
```

```
[1] 393
```

## 6.2 Group by and count

So what if we wanted to know how many mountain lion sightings there were in each county?

To do that by hand, we'd have to take each of the 393 records and sort them into a pile. We'd put them in groups and then count them.

`dplyr` has a group by function in it that does just this. A massive amount of data analysis involves grouping like things together and then doing simple things like counting them, or averaging them together. So it's a good place to start.

So to do this, we'll take our dataset and we'll introduce a new operator: `|>`. The best way to read that operator, in my opinion, is to interpret that as "and then do this."

We're going to establish a pattern that will come up again and again throughout this book: `data |> function`. 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(FIELD NAME)`  
`|> summarize(VARIABLE NAME = AGGREGATE FUNCTION(FIELD NAME))`

Here's the code:

```
mountainlions |>
  group_by(COUNTY) |>
  summarise(
    total = n()
  )
```

```
# A tibble: 42 x 2
  COUNTY      total
  <chr>      <int>
1 Banner         6
2 Blaine         3
3 Box Butte      4
4 Brown        15
5 Buffalo        3
6 Cedar          1
7 Cherry        30
8 Custer         8
9 Dakota         3
10 Dawes       111
# i 32 more rows
```

So let's walk through that. We start with our dataset – `mountainlions` – 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 `COUNTY`, which you could get from looking at `head(mountainlions)`. After we group the data, we need to count them up. In `dplyr`, we use `summarize` [which can do 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 counties: `total = n()`, says create a new field, called `total` and set it equal to `n()`, which might look weird, but it's common in stats. The number of things in a dataset? Statisticians call in `n`. There are `n` number of incidents in this dataset. So `n()` is a function that counts the number of things there are.

And 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 `|>` and use `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 mountain lion sightings, we need to sort it in descending order. That looks like this:

```
mountainlions |>
  group_by(COUNTY) |>
```



```

summarise(
  count = n()
) |> arrange(desc(count))

```

```

# A tibble: 42 x 2
  COUNTY      count
  <chr>      <int>
1 Dawes      111
2 Sioux       52
3 Sheridan   35
4 Cherry     30
5 Scotts Bluff 26
6 Keya Paha  20
7 Brown      15
8 Rock        11
9 Lincoln     10
10 Custer      8
# i 32 more rows

```

We can, if we want, group by more than one thing. So how are these sightings being confirmed? To do that, we can group by County and “Cofirm Type”, which is how the state misspelled Confirm. But note something in this example below:

```

mountainlions |>
  group_by(COUNTY, `Cofirm Type`) |>
  summarise(
    count = n()
  ) |> arrange(desc(count))

```

``summarise()`` has grouped output by 'COUNTY'. You can override using the ``.groups`` argument.

```

# A tibble: 93 x 3
# Groups:   COUNTY [42]
  COUNTY      `Cofirm Type`      count
  <chr>      <chr>          <int>
1 Dawes      Trail Camera Photo    41
2 Sioux      Trail Camera Photo    40
3 Dawes      Track                 19
4 Keya Paha  Trail Camera Photo    18

```

```

5 Cherry          Trail Camera Photo    17
6 Dawes           Mortality              17
7 Sheridan        Trail Camera Photo    16
8 Dawes           Photo                  13
9 Dawes           DNA                    11
10 Scotts Bluff Trail Camera Photo      11
# i 83 more rows

```

See it? When you have a field name that has two words, `readr` wraps it in back ticks, which is next to the 1 key on your keyboard. You can figure out which fields have back ticks around it by looking at the output of `readr`. Pay attention to that, because it's coming up again in the next section and will be a part of your homework.

## 6.3 Other aggregates: Mean and median

In the last example, we grouped some data together and counted it up, but there's so much more you can do. You can do multiple measures in a single step as well.

Let's look at some [salary data from the University of Nebraska](#).

```
salaries <- read_csv("data/nusalaries1819.csv")
```

```
Rows: 13039 Columns: 7
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (4): Employee, Position, Campus, Department
```

```
num (3): Budgeted Annual Salary, Salary from State Aided Funds, Salary from ...
```

```
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.
```

```
head(salaries)
```

```
# A tibble: 6 x 7
```

|   | Employee          | Position | Campus | Department | Budgeted Annual Sala~1 |
|---|-------------------|----------|--------|------------|------------------------|
|   | <chr>             | <chr>    | <chr>  | <chr>      | <dbl>                  |
| 1 | Abbey, Bryce M    | Associa~ | UNK    | Kinesiolo~ | 61276                  |
| 2 | Abbott, Frances M | Staff S~ | UNL    | FM&P Faci~ | 37318                  |
| 3 | Abboud, Cheryl A  | Adminis~ | UNMC   | Surgery-U~ | 76400                  |

```

4 Abdalla, Maher Y          Asst Pr~ UNMC   Pathology~      74774
5 Abdelkarim, Ahmed Mohamed A~ Post-Do~ UNMC   Surgery-T~     43516
6 Abdel-Monem, Tarik L      Researc~ UNL    Public Po~     58502
# i abbreviated name: 1: `Budgeted Annual Salary`
# i 2 more variables: `Salary from State Aided Funds` <dbl>,
#   `Salary from Other Funds` <dbl>

```

In summarize, we can calculate any number of measures. Here, we'll use R's built in mean and median functions to calculate ... well, you get the idea.

```

salaries |>
  summarise(
    count = n(),
    mean_salary = mean(`Budgeted Annual Salary`),
    median_salary = median(`Budgeted Annual Salary`)
  )

```

```

# A tibble: 1 x 3
  count mean_salary median_salary
  <int>      <dbl>      <dbl>
1 13039      62065.      51343

```

So there's 13,039 employees in the database, spread across four campuses plus the system office. The mean or average salary is about \$62,000, but the median salary is slightly more than \$51,000.

Why?

Let's let sort help us.

```

salaries |> arrange(desc(`Budgeted Annual Salary`))

```

```

# A tibble: 13,039 x 7
  Employee          Position      Campus Department Budgeted Annual Sala~1
  <chr>             <chr>      <chr>   <chr>      <dbl>
1 Frost, Scott A    Head Coach-Foo~ UNL     Athletics  5000000
2 Miles, Timothy S  Head Coach-Bas~ UNL     Athletics  2375000
3 Moos, William H   Athletic Direc~ UNL     Athletics  1000000
4 Gold, Jeffrey P   Chancellor      UNMC    Office of~  853338
5 Chinander, Erik J Assistant Coac~ UNL     Athletics  800000
6 Walters, Troy M   Assistant Coac~ UNL     Athletics  700000

```

```

7 Cook, John G          Head Coach-Vol~ UNL    Athletics          675000
8 Williams, Amy M       Head Coach-Wom~ UNL    Athletics          626750
9 Bounds, Hank M        President      UNCA    Office of~          540000
10 Austin Jr, Gregory D Assistant Coac~ UNL    Athletics          475000
# i 13,029 more rows
# i abbreviated name: 1: `Budgeted Annual Salary`
# i 2 more variables: `Salary from State Aided Funds` <dbl>,
#   `Salary from Other Funds` <dbl>

```

Oh, right. In this dataset, the university pays a football coach \$5 million. Extremes influence averages, not medians, and now you have your answer.

So when choosing a measure of the middle, you have to ask yourself – could I have extremes? Because a median won't be sensitive to extremes. It will be the point at which half the numbers are above and half are below. The average or mean will be a measure of the middle, but if you have a bunch of low paid people and then one football coach, the average will be wildly skewed. Here, because there's so few highly paid football coaches compared to people who make a normal salary, the number is only slightly skewed in the grand scheme, but skewed nonetheless.

## 6.4 Even more aggregates

There's a ton of things we can do in summarize – we'll work with more of them as the course progresses – but here's a few other questions you can ask.

Which department on campus has the highest wage bill? And what is the highest and lowest salary in the department? And how wide is the spread between salaries? We can find that with `sum` to add up the salaries to get the total wage bill, `min` to find the minimum salary, `max` to find the maximum salary and `sd` to find the standard deviation in the numbers.

```

salaries |>
  group_by(Campus, Department) |>
  summarize(
    total = sum(`Budgeted Annual Salary`),
    avgsalary = mean(`Budgeted Annual Salary`),
    minsalary = min(`Budgeted Annual Salary`),
    maxsalary = max(`Budgeted Annual Salary`),
    stdev = sd(`Budgeted Annual Salary`)) |> arrange(desc(total))

```

``summarise()`` has grouped output by 'Campus'. You can override using the ``.groups`` argument.

```

# A tibble: 804 x 7
# Groups:   Campus [5]
  Campus Department      total avgsalary minsalary maxsalary stdev
  <chr>   <chr>          <dbl>     <dbl>     <dbl>     <dbl> <dbl>
1 UNL    Athletics        3.56e7  118508.    12925    5000000 3.33e5
2 UNMC   Pathology/Microbiology 1.36e7   63158.     1994    186925 3.41e4
3 UNL    Agronomy & Horticulture 8.98e6   66496.     5000    208156 4.01e4
4 UNMC   Anesthesiology        7.90e6   78237.    10000    245174 3.59e4
5 UNL    School of Natural Resourc~ 6.86e6   65995.     2400    194254 3.28e4
6 UNL    College of Law         6.70e6   77953.     1000    326400 7.23e4
7 UNL    University Television   6.44e6   55542.    16500    221954 2.75e4
8 UNL    University Libraries    6.27e6   51390.     1200    215917 2.68e4
9 UNMC   Pharmacology/Exp Neurosci~ 6.24e6   58911.     2118    248139 4.29e4
10 UNMC   CON-Omaha Division      6.11e6   78304.     3000    172522 4.48e4
# i 794 more rows

```

So again, no surprise, the UNL athletic department has the single largest wage bill at nearly \$36 million. The average salary in the department is \$118,508 – more than double the univeristy as a whole, again thanks to Scott Frost’s paycheck.

## 7 Mutating data

One of the most common data analysis techniques is to look at change over time. The most common way of comparing change over time is through percent change. The math behind calculating percent change is very simple, and you should know it off the top of your head. The easy way to remember it is:

$$(\text{new} - \text{old}) / \text{old}$$

Or new minus old divided by old. Your new number minus the old number, the result of which is divided by the old number. To do that in R, we can use `dplyr` and `mutate` to calculate new metrics in a new field using existing fields of data.

So first we'll import the tidyverse so we can read in our data and begin to work with it.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

Now we'll import a common and [simple dataset of county population estimates](#) from the US Census Bureau. Each year, the Census Bureau publishes estimates for states and counties. This one has every county in the US. A common question: who are the winners and losers?

```
population <- read_csv('data/countypopulations.csv')
```

```
Rows: 3142 Columns: 13
```

```
-- Column specification -----
```

```
Delimiter: ","
chr (2): STNAME, CTYNAME
dbl (11): CENSUS2010POP, ESTIMATESBASE2010, POPESTIMATE2010, POPESTIMATE2011...
```

```
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.
```

The code to calculate percent change 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. So in this case, we’re trying to do (new-old)/old, but we’re doing it with fields.

If we look at what we got when we imported the data, you’ll see there’s `POPESTIMATE2018` as the new data, and we’ll use `POPESTIMATE2017` as the old data. So we’re looking at one year. Then, to help us, we’ll use `arrange` again to sort it, so we get the fastest growing county over one year.

```
population |> mutate(
  change = (POPESTIMATE2018 - POPESTIMATE2017)/POPESTIMATE2017
)
```

```
# A tibble: 3,142 x 14
  STNAME CTYNAME CENSUS2010POP ESTIMATESBASE2010 POPESTIMATE2010
  <chr>   <chr>         <dbl>         <dbl>         <dbl>
1 Alabama Autauga County      54571          54574          54754
2 Alabama Baldwin County    182265         182264         183111
3 Alabama Barbour County     27457          27457          27330
4 Alabama Bibb County        22915          22920          22872
5 Alabama Blount County      57322          57321          57373
6 Alabama Bullock County     10914          10911          10878
7 Alabama Butler County      20947          20943          20942
8 Alabama Calhoun County     118572         118594         118477
9 Alabama Chambers County    34215          34171          34122
10 Alabama Cherokee County   25989          25989          25974
# i 3,132 more rows
# i 9 more variables: POPESTIMATE2011 <dbl>, POPESTIMATE2012 <dbl>,
# POPESTIMATE2013 <dbl>, POPESTIMATE2014 <dbl>, POPESTIMATE2015 <dbl>,
# POPESTIMATE2016 <dbl>, POPESTIMATE2017 <dbl>, POPESTIMATE2018 <dbl>,
# change <dbl>
```

Click the black arrow pointing right and you’ll see, way out on the right, your change column. But what do you see right away? 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.

```
population |> mutate(
  change = ((POPESTIMATE2018 - POPESTIMATE2017)/POPESTIMATE2017)*100
)
```

```
# A tibble: 3,142 x 14
```

|    | STNAME  | CTYNAME         | CENSUS2010POP | ESTIMATESBASE2010 | POPESTIMATE2010 |
|----|---------|-----------------|---------------|-------------------|-----------------|
|    | <chr>   | <chr>           | <dbl>         | <dbl>             | <dbl>           |
| 1  | Alabama | Autauga County  | 54571         | 54574             | 54754           |
| 2  | Alabama | Baldwin County  | 182265        | 182264            | 183111          |
| 3  | Alabama | Barbour County  | 27457         | 27457             | 27330           |
| 4  | Alabama | Bibb County     | 22915         | 22920             | 22872           |
| 5  | Alabama | Blount County   | 57322         | 57321             | 57373           |
| 6  | Alabama | Bullock County  | 10914         | 10911             | 10878           |
| 7  | Alabama | Butler County   | 20947         | 20943             | 20942           |
| 8  | Alabama | Calhoun County  | 118572        | 118594            | 118477          |
| 9  | Alabama | Chambers County | 34215         | 34171             | 34122           |
| 10 | Alabama | Cherokee County | 25989         | 25989             | 25974           |

```
# i 3,132 more rows
```

```
# i 9 more variables: POPESTIMATE2011 <dbl>, POPESTIMATE2012 <dbl>,
# POPESTIMATE2013 <dbl>, POPESTIMATE2014 <dbl>, POPESTIMATE2015 <dbl>,
# POPESTIMATE2016 <dbl>, POPESTIMATE2017 <dbl>, POPESTIMATE2018 <dbl>,
# change <dbl>
```

Now, does this ordering do anything for us? No. Let's fix that with arrange.

```
population |> mutate(
  change = ((POPESTIMATE2018 - POPESTIMATE2017)/POPESTIMATE2017)*100
) |> arrange(desc(change))
```

```
# A tibble: 3,142 x 14
```

|   | STNAME         | CTYNAME        | CENSUS2010POP | ESTIMATESBASE2010 | POPESTIMATE2010 |
|---|----------------|----------------|---------------|-------------------|-----------------|
|   | <chr>          | <chr>          | <dbl>         | <dbl>             | <dbl>           |
| 1 | Texas          | Loving County  | 82            | 82                | 84              |
| 2 | Colorado       | San Juan Coun~ | 699           | 699               | 708             |
| 3 | North Dakota   | McKenzie Coun~ | 6360          | 6359              | 6411            |
| 4 | Kentucky       | Lee County     | 7887          | 7887              | 7718            |
| 5 | North Dakota   | Williams Coun~ | 22398         | 22399             | 22588           |
| 6 | Texas          | Comal County   | 108472        | 108485            | 109270          |
| 7 | Texas          | Kenedy County  | 416           | 413               | 417             |
| 8 | Texas          | Kaufman County | 103350        | 103363            | 103890          |
| 9 | North Carolina | Brunswick Cou~ | 107431        | 107429            | 108065          |



```

10 Florida          Walton County          55043          55043          55211
# i 3,132 more rows
# i 9 more variables: POPESTIMATE2011 <dbl>, POPESTIMATE2012 <dbl>,
#   POPESTIMATE2013 <dbl>, POPESTIMATE2014 <dbl>, POPESTIMATE2015 <dbl>,
#   POPESTIMATE2016 <dbl>, POPESTIMATE2017 <dbl>, POPESTIMATE2018 <dbl>,
#   change <dbl>

```

So who had the most growth last year from the year before? Is everyone moving to Loving County, Texas? Or is it small changes in a small county? Also, note North Dakota showing up twice in the top 10.

## 7.1 Another use of mutate

Note in our data we have separate State and County name fields. If we were publishing this, we wouldn't want that.

So how can we fix that? Mutate! And a new function to combine text together called `paste`. Paste allows us to merge fields together easily with a separator. In our case, we want to combine the county name and the state name with a comma and a space between them.

```

population |>
  mutate(
    change = ((POPESTIMATE2018 - POPESTIMATE2017)/POPESTIMATE2017)*100,
    location = paste(CTYNAME, STNAME, sep=", ") |>
    arrange(desc(change))

```

```

# A tibble: 3,142 x 15
  STNAME      CTYNAME      CENSUS2010POP ESTIMATESBASE2010 POPESTIMATE2010
  <chr>      <chr>      <dbl>          <dbl>          <dbl>
1 Texas      Loving County      82             82             84
2 Colorado   San Juan Coun~    699            699            708
3 North Dakota McKenzie Coun~   6360           6359           6411
4 Kentucky   Lee County        7887           7887           7718
5 North Dakota Williams Coun~   22398          22399          22588
6 Texas      Comal County     108472          108485          109270
7 Texas      Kenedy County      416            413            417
8 Texas      Kaufman County    103350          103363          103890
9 North Carolina Brunswick Cou~ 107431          107429          108065
10 Florida    Walton County     55043          55043          55211
# i 3,132 more rows
# i 10 more variables: POPESTIMATE2011 <dbl>, POPESTIMATE2012 <dbl>,

```

```
# POPESTIMATE2013 <dbl>, POPESTIMATE2014 <dbl>, POPESTIMATE2015 <dbl>,  
# POPESTIMATE2016 <dbl>, POPESTIMATE2017 <dbl>, POPESTIMATE2018 <dbl>,  
# change <dbl>, location <chr>
```

EXERCISE: What happens when you sort it in ascending order? Delete the desc part in arrange and see what happens. How would you describe this list?

## 8 Working with dates

One of the most frustrating things in data is working with dates. Everyone has a different opinion on how to record them, and every software package on the planet has to sort it out. Dealing with it can be a little ... confusing. And every dataset has something new to throw at you. So consider this an introduction.

We're going to do this two ways. First I'm going to show you how to use base R to solve a tricky problem. And then we'll use a library called `lubridate` to solve a more common and less tricky problem. And then we'll use a new library to solve most of the common problems before they start.

### 8.1 The hard way

First, we'll import `tidyverse` like we always do.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

We're going to use a dataset of [parking tickets at UNL](#). If we do this the old way – using `read.csv` – this is what we get:

```
tickets <- read.csv("data/tickets.csv")
head(tickets)
```

|   | Citation | Date                | Location        | Violation                    |
|---|----------|---------------------|-----------------|------------------------------|
| 1 | 15078429 | 2012-04-02 07:15:00 | North Stadium   | Expired Meter                |
| 2 | 24048318 | 2012-04-02 07:22:00 | Housing         | No Valid Permit Displayed    |
| 3 | 24048320 | 2012-04-02 07:26:00 | 14th & W Street | No Valid Permit Displayed    |
| 4 | 15078430 | 2012-04-02 07:36:00 | Champions Club  | Parking in Unauthorized Area |
| 5 | 18074937 | 2012-04-02 07:39:00 | Sandoz          | Expired Meter                |
| 6 | 18074938 | 2012-04-02 07:40:00 | Sandoz          | Expired Meter                |

Note the date is a factor, not a date. We have to fix that. There's a lot of ways to fix dates. The base R way is to use formatting. The code is ... a little odd ... but it's useful to know if you get a really odd date format. What you are doing is essentially parsing the date into it's component parts then reassembling it into a date using formatting.

```
newtickets <- tickets |> mutate(
  CleanDate = as.POSIXct(Date, format="%Y-%m-%d %H:%M:%S")
)

head(newtickets)
```

|   | Citation | Date                | Location        | Violation                    |
|---|----------|---------------------|-----------------|------------------------------|
| 1 | 15078429 | 2012-04-02 07:15:00 | North Stadium   | Expired Meter                |
| 2 | 24048318 | 2012-04-02 07:22:00 | Housing         | No Valid Permit Displayed    |
| 3 | 24048320 | 2012-04-02 07:26:00 | 14th & W Street | No Valid Permit Displayed    |
| 4 | 15078430 | 2012-04-02 07:36:00 | Champions Club  | Parking in Unauthorized Area |
| 5 | 18074937 | 2012-04-02 07:39:00 | Sandoz          | Expired Meter                |
| 6 | 18074938 | 2012-04-02 07:40:00 | Sandoz          | Expired Meter                |

|   | CleanDate           |
|---|---------------------|
| 1 | 2012-04-02 07:15:00 |
| 2 | 2012-04-02 07:22:00 |
| 3 | 2012-04-02 07:26:00 |
| 4 | 2012-04-02 07:36:00 |
| 5 | 2012-04-02 07:39:00 |
| 6 | 2012-04-02 07:40:00 |

CleanDate is now a special date format that includes times.

You can almost read the code that created it: The format of the date is %Y, which means a four digit year DASH %m or two digit month DASH %d or two digit day SPACE %H or two digit hour COLON %M or two digit minute COLON %S or two digit second. You can remix that as you need. If you had a date that was 20021212 then you would do `format="%Y%m%d"` and so on.

There is a [library called lubridate](#) that can parse some common date problems. If it's not already installed, just run `install.packages('lubridate')`

```
library(lubridate)
```

Lubridate can handle this tickets data easier with one of it's many functions. The functions parse dates given a basic pattern. In this case, our data is in a very common pattern of year month date hours minutes seconds. Lubridate has a function called `ymd_hms`.

```
lubridatetickets <- tickets |> mutate(  
  CleanDate = ymd_hms(Date)  
)  
  
head(lubridatetickets)
```

|   | Citation   | Date                | Location        | Violation                    |
|---|------------|---------------------|-----------------|------------------------------|
| 1 | 15078429   | 2012-04-02 07:15:00 | North Stadium   | Expired Meter                |
| 2 | 24048318   | 2012-04-02 07:22:00 | Housing         | No Valid Permit Displayed    |
| 3 | 24048320   | 2012-04-02 07:26:00 | 14th & W Street | No Valid Permit Displayed    |
| 4 | 15078430   | 2012-04-02 07:36:00 | Champions Club  | Parking in Unauthorized Area |
| 5 | 18074937   | 2012-04-02 07:39:00 | Sandoz          | Expired Meter                |
| 6 | 18074938   | 2012-04-02 07:40:00 | Sandoz          | Expired Meter                |
|   | CleanDate  |                     |                 |                              |
| 1 | 2012-04-02 | 07:15:00            |                 |                              |
| 2 | 2012-04-02 | 07:22:00            |                 |                              |
| 3 | 2012-04-02 | 07:26:00            |                 |                              |
| 4 | 2012-04-02 | 07:36:00            |                 |                              |
| 5 | 2012-04-02 | 07:39:00            |                 |                              |
| 6 | 2012-04-02 | 07:40:00            |                 |                              |

That's less code and less weirdness, so that's good.

But to get clean data, I've installed a library and created a new field so I can now start to work with my dates. That seems like a lot, but don't think your data will always be perfect and you won't have to do these things.

Still, there's got to be a better way. And there is.

Fortunately, `readr` anticipates some date formattings and can automatically handle this (indeed it uses lubridate under the hood). The change in your code? You just use `read_csv` instead of `read.csv`

```
tickets <- read_csv("data/tickets.csv")
```

```

Rows: 161315 Columns: 4
-- Column specification -----
Delimiter: ","
chr (3): Citation, Location, Violation
dtm (1): Date

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.

```

```
head(tickets)
```

```

# A tibble: 6 x 4
  Citation Date          Location      Violation
  <chr>      <dtm>          <chr>      <chr>
1 15078429 2012-04-02 07:15:00 North Stadium Expired Meter
2 24048318 2012-04-02 07:22:00 Housing      No Valid Permit Displayed
3 24048320 2012-04-02 07:26:00 14th & W Street No Valid Permit Displayed
4 15078430 2012-04-02 07:36:00 Champions Club Parking in Unauthorized Area
5 18074937 2012-04-02 07:39:00 Sandoz       Expired Meter
6 18074938 2012-04-02 07:40:00 Sandoz       Expired Meter

```

And just like that, the dates are formatted correctly.

But you're not done with lubridate yet. It has some interesting pieces parts we'll use elsewhere.

What's a question you might have about parking tickets on campus involving dates?

How about what month are the most tickets issued? We could use formatting to create a Month field but that would group all the Aprils ever together. We could create a year and a month together, but that would give us an invalid date object and that would create problems later. Lubridate has something called a floor date that we can use.

So to follow along here, we're going to use mutate to create a month field, group by to lump them together, summarize to count them up and arrange to order them. We're just chaining things together.

```

tickets |>
  mutate(Month = floor_date(Date, "month")) |>
  group_by(Month) |>
  summarise(total = n()) |>
  arrange(desc(total))

```

```
# A tibble: 56 x 2
  Month                total
  <dtm>                <int>
1 2014-10-01 00:00:00  5177
2 2015-04-01 00:00:00  4913
3 2014-09-01 00:00:00  4645
4 2015-09-01 00:00:00  4541
5 2015-10-01 00:00:00  4403
6 2015-03-01 00:00:00  4392
7 2016-02-01 00:00:00  4314
8 2016-09-01 00:00:00  4221
9 2016-03-01 00:00:00  4194
10 2012-10-01 00:00:00  4173
# i 46 more rows
```

So the most tickets in this dataset were issued in September of 2014. April of 2015 was second. Then two Septembers and an October.

Any guesses why those months?

I'll give you a hint. It involves 90,000 people gathering in a big building on campus in the fall and one day in April or late March every spring.

## 9 Filters and selections

More often than not, we have more data than we want. Sometimes we need to be rid of that data. In `dplyr`, there's two ways to go about this: filtering and selecting.

**Filtering creates a subset of the data based on criteria.** All records where the count is greater than 10. All records that match "Nebraska". Something like that. **Filtering works with rows – when we filter, we get fewer rows back than we start with.**

**Selecting simply returns only the fields named.** So if you only want to see Year and County, you select those fields. When you look at your data again, you'll have two columns. If you try to use one of your columns that you had before you used select, you'll get an error. **Selecting works with columns. You will have the same number of records when you are done, but fewer columns of data to work with.**

Let's work with the [salaries data from the University of Nebraska](#). It has data from all NU campuses, but only one of them is our campus, so let's filter out everyone else.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
```

```
-- Conflicts ----- tidyverse_conflicts() --
```

```
x dplyr::filter() masks stats::filter()
```

```
x dplyr::lag()     masks stats::lag()
```

```
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
salaries <- read_csv("data/nusalaries1819.csv")
```

```
Rows: 13039 Columns: 7
```

```
-- Column specification -----
```

```
Delimiter: ","
```



```
chr (4): Employee, Position, Campus, Department
num (3): Budgeted Annual Salary, Salary from State Aided Funds, Salary from ...
```

```
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.
```

The data we want to filter on is in `Campus`. So we're going to use `filter` and something called a comparison operator. We need to filter all records equal to UNL. The comparison operators in R, like most programming languages, are `==` for equal to, `!=` for not equal to, `>` for greater than, `>=` for greater than or equal to and so on.

**Be careful: `=` is not `==` and `=` is not “equal to”. `=` is an assignment operator in most languages – how things get named.**

```
unl <- salaries |> filter(Campus == "UNL")

head(unl)
```

```
# A tibble: 6 x 7
  Employee          Position      Campus Department Budgeted Annual Sala~1
  <chr>             <chr>      <chr>  <chr>          <dbl>
1 Abbott, Frances M Staff Secy III UNL    FM&P Faci~    37318
2 Abdel-Monem, Tarik L Research Specia~ UNL    Public Po~    58502
3 Abel, Marco       Chairperson  UNL    English        64470
4 Abel, Marco       Professor    UNL    English        39647
5 Abel, Rick A      Control Systems~ UNL    FM&P Buil~    57178
6 Abendroth, Curtis L Asst Dir Facili~ UNL    Housing F~    79037
# i abbreviated name: 1: `Budgeted Annual Salary`
# i 2 more variables: `Salary from State Aided Funds` <dbl>,
#   `Salary from Other Funds` <dbl>
```

And just like that, we have just UNL, which we can verify looking at the head, the first six rows.

We also have more data than we might want. For example, the salary data is only in the Budgeted Annual Salary column. The other two salary fields are useless detail.

To simplify our dataset, we can use `select`.

```
selected_unl <- unl |> select(Employee, Position, Campus, `Budgeted Annual Salary`)

head(selected_unl)
```

```
# A tibble: 6 x 4
  Employee          Position      Campus Budgeted Annual Sala~1
  <chr>             <chr>          <chr>          <dbl>
1 Abbott, Frances M Staff Secy III UNL             37318
2 Abdel-Monem, Tarik L Research Specialist UNL             58502
3 Abel, Marco       Chairperson    UNL             64470
4 Abel, Marco       Professor      UNL             39647
5 Abel, Rick A      Control Systems Tech/Alarm~ UNL             57178
6 Abendroth, Curtis L Asst Dir Facilities Mgt/Ma~ UNL             79037
# i abbreviated name: 1: `Budgeted Annual Salary`
```

And now we only have four columns of data for whatever salary analysis we might want to do.

## 9.1 Combining filters

So let's say we wanted to know how many full professors make more than \$100,000. We can do this a number of ways. The first is we can chain together a whole lot of filters.

```
profs <- salaries |> filter(Campus == "UNL") |> filter(Position == "Professor") |> filter(
  nrow(profs)
```

```
[1] 312
```

So that gives us 312 full professors – that's the top rank of tenured professors – who make more than \$100,000. But that's silly and repetitive, no? We can do better using boolean operators – AND and OR. In this case, AND is `&` and OR is `|`.

The difference? With AND, all three things must be true to be included. With OR, any of those three things can be true and it will be included. An assistant professor making \$100k at UNO will get included because they make \$100k. One of the conditions is true.

Here's the difference.

```
andprofs <- salaries |> filter(Campus == "UNL" & Position == "Professor" & `Budgeted Annual Salary` > 100000)
nrow(andprofs)
```

```
[1] 312
```

So AND gives us the same answer we got before. What does OR give us?

```
orprofs <- salaries |> filter(Campus == "UNL" | Position == "Professor" | `Budgeted Annual  
nrow(orprofs)
```

```
[1] 7248
```

So there's 7,248 unique people in the NU system who are at UNL (6,079 to be exact), are full Professors (1,086 of them), or make more than \$100,000 (1,604) of them. Included in that list? Football coach Scott Frost, who makes ... ahem ... more than \$100,000. A smidge more.

# 10 Data Cleaning Part I: Data smells

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](#).

Let's look at some campus parking ticket data. You can [get it here](#).

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](#). The common mistakes we're looking for are:

- Missing data
- 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.

But with several of these data smells, we can do them first, before we do anything else.

## 10.1 Wrong Type

First, let's look at **Wrong Type Of Data**. We can sniff that out by looking at the output of `readr`

```
library(tidyverse)

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
  explicit

tickets <- read_csv("data/tickets.csv")

Rows: 161315 Columns: 4
-- Column specification -----
Delimiter: ","
chr  (3): Citation, Location, Violation
dtm  (1): Date

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.
```

Right away, we see there's 104,265 parsing errors. Why? Look closely. The Citation number that `readr` interprets from the first rows comes in at a number. But 56,000 rows in, those citation numbers start having letters in them, and letters are not numbers.

The cheap way to fix this is to change the `guess_max` parameter of `readr` to just use more than a few rows to guess the column types. It'll go a little slower, but it'll fix the problem.

```
tickets <- read_csv("data/tickets.csv", guess_max = 60000)
```

```

Rows: 161315 Columns: 4
-- Column specification -----
Delimiter: ","
chr  (3): Citation, Location, Violation
dtm  (1): Date

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.

```

For this, things seem to be good. `Date` appears to be in date format, things that aren't numbers appear to be text. That's a good start.

## 10.2 Missing Data

The second smell we can find in code is **missing data**. We can do that through a series of Group By and Count steps.

```

tickets |> group_by(Location) |> tally()

# A tibble: 247 x 2
  Location          n
  <chr>          <int>
1 1001 Y Street    508
2 10th & Q Street  303
3 10th & U Street  222
4 1101 Y Street    83
5 11th & Y Street   38
6 1235 Military Road  33
7 1320 Q           1
8 13th & R Lot   4918
9 14th & Avery Lot 1601
10 14th & Avery Parking Garage 2494
# i 237 more rows

```

What we're looking for here are blanks: Tickets without a location. Typically, those will appear first or last, depending on several things, so it's worth checking the front and back of our data.

What about ticket type?

```
tickets |> group_by(Violation) |> tally()
```

```
# A tibble: 25 x 2
  Violation          n
  <chr>          <int>
1 Damage Property      25
2 Displaying Altered Permit 23280
3 Displaying Counterfeit Permit 18
4 Displaying Stolen Permit 4
5 Expired Meter      45072
6 Failure to Pay[on exit] 251
7 Failure to Reg. Veh to Permit 53
8 Failure to Register Veh w/ UNL 113
9 False Lost/Stolen Permit Rept 927
10 Falsify Permit Application 3
# i 15 more rows
```

None here either, so that's good. It means our tickets will always have a location and a violation type.

## 10.3 Gaps in data

Let's now look at **gaps in data**. It's been my experience that gaps in data often have to do with time, so let's first look at ticket dates, so we can see if there's any big jumps in data. You'd expect the numbers to change, but not by huge amounts. Huge change would indicate, more often than not, that the data is missing. Let's start with Date. If we're going to work with dates, we should have lubridate handy for `floor_date`.

```
library(lubridate)
```

```
tickets |> group_by(floor_date(Date, "month")) |> tally()
```

```
# A tibble: 56 x 2
  `floor_date(Date, "month")`    n
  <dtm>          <int>
1 2012-04-01 00:00:00      3473
2 2012-05-01 00:00:00      2572
3 2012-06-01 00:00:00      2478
```

```

4 2012-07-01 00:00:00      2134
5 2012-08-01 00:00:00      3774
6 2012-09-01 00:00:00      4138
7 2012-10-01 00:00:00      4173
8 2012-11-01 00:00:00      3504
9 2012-12-01 00:00:00      1593
10 2013-01-01 00:00:00      3078
# i 46 more rows

```

First thing to notice: our data starts in April 2012. So 2012 isn't a complete year. Then, scroll through. Look at December 2013 - March 2014. The number of tickets drops to about 10 percent of normal. That's ... odd. And then let's look at the end - November 2016. So not a complete year in 2016 either.

## 10.4 Internal inconsistency

Any time you are going to focus on something, you should check it for consistency inside the data set. So let's pretend the large number of Displaying Altered Permit tickets caught your attention and you want to do a story about tens of thousands of students being caught altering their parking permits to reduce their costs parking on campus. Good story right? Before you go calling the parking office for comment, I'd check that data first.

```

tickets |> filter(Violation == "Displaying Altered Permit") |> group_by(floor_date(Date, "
# A tibble: 29 x 2
  `floor_date(Date, "month")`      n
  <dtm>                        <int>
1 2012-04-01 00:00:00            1072
2 2012-05-01 00:00:00            1283
3 2012-06-01 00:00:00            1324
4 2012-07-01 00:00:00            1357
5 2012-08-01 00:00:00            2249
6 2012-09-01 00:00:00            1797
7 2012-10-01 00:00:00            1588
8 2012-11-01 00:00:00            1183
9 2012-12-01 00:00:00             458
10 2013-01-01 00:00:00            1132
# i 19 more rows

```



So this charge exists when our data starts, but scroll forward: In October 2013, there's 1,081 tickets written. A month later, only 121. A month after that? 1. And then one sporadically for three more years.

Something major changed. What is it? That's why you are a reporter. Go ask. But we know our data doesn't support the story we started with.

And that's what Data Smells are designed to do: stop you from going down a bad path.

## 10.5 A Shortcut: Summary

One quick way to get some data smells is to use R's built in summary function. What summary does is run summary statistics on each column of your dataset. Some of the output is ... underwhelming ... but some is really useful. For example, looking at min and max for dates can point to bad data there. Min and max will also show you out of range numbers – numbers far too big or small to make sense.

The syntax is simple.

```
summary(tickets)
```

| Citation         | Date                           | Location         |
|------------------|--------------------------------|------------------|
| Length:161315    | Min. :2012-04-02 07:15:00.00   | Length:161315    |
| Class :character | 1st Qu.:2013-05-06 09:42:30.00 | Class :character |
| Mode :character  | Median :2014-10-17 12:03:00.00 | Mode :character  |
|                  | Mean :2014-08-13 19:36:52.66   |                  |
|                  | 3rd Qu.:2015-10-08 07:31:30.00 |                  |
|                  | Max. :2016-11-03 13:59:19.00   |                  |
| Violation        |                                |                  |
| Length:161315    |                                |                  |
| Class :character |                                |                  |
| Mode :character  |                                |                  |

In this case, the output doesn't do much for us except dates. Looking at the min and max will tell us if we have any out of range dates. In this case, we do not.

# 11 Data Cleaning Part II: Janitor

The bane of every data analyst's existence is data cleaning.

Every developer, every data system, every agency, the all have opinions about how data gets collected. Some decisions make sense from the outside. Some decisions are based entirely on internal politics: who is creating the data, how they are creating it, why they are creating it. Is it automated? Is it manual? Are data normalized? Are there free form fields where users can just type into or does the system restrict them to choices?

Your question – what you want to do with the data – is almost never part of that equation.

So cleaning data is the process of fixing issues in your data so you can answer the questions you want to answer. Unfortunately, there's no template here. There's no checklist. It's just a big bag of tricks that eventually runs out and you'll be left fixing individual issues by hand, if it's really bad.

But let's start simple. There are certain things that need we can start with that will make our lives easier. We'll slowly make it harder as we dig deeper.

## 11.1 Cleaning headers

One of the first places we can start with cleaning data is cleaning the headers. Every system has their own way of recording headers, and every developer has their own thoughts of what a good idea is within it. R is most happy when headers are one word, lower case, without special characters. If you've noticed `readr` output with backticks around headers like Incident Date, it's because of the space. Headers that start with numbers or are just a number – 2002 – also get backticks in `readr`.

There is an external library in R called `janitor` that makes fixing headers trivially simple. You can install it by running `install.packages("janitor")` in your console.

Load libraries like we always do.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
library(janitor)
```

Attaching package: 'janitor'

The following objects are masked from 'package:stats':

```
chisq.test, fisher.test
```

Let's load a new dataset – the [list of active inmates in the Nebraska prison system](#).

```
inmates <- read_csv("data/activeinmates.csv")
```

New names:

```
* `FIRST NAME` -> `FIRST NAME...3`
* `MIDDLE NAME` -> `MIDDLE NAME...4`
* `NAME EXTENSION` -> `NAME EXTENSION...5`
* `FIRST NAME` -> `FIRST NAME...7`
* `MIDDLE NAME` -> `MIDDLE NAME...8`
* `NAME EXTENSION` -> `NAME EXTENSION...9`
* `` -> `...31`
* `` -> `...32`
```

Warning: One or more parsing issues, call `problems()` on your data frame for details, e.g.:

```
dat <- vroom(...)
problems(dat)
```

Rows: 7674 Columns: 32

```
-- Column specification -----  
Delimiter: ","  
chr (22): COMMITTED LAST NAME, FIRST NAME...3, MIDDLE NAME...4, NAME EXTENSI...  
dbl (6): ID NUMBER, MIN MONTH, MIN DAY, MAX MONTH, MAX DAY, GOOD TIME LAW  
lgl (4): NAME EXTENSION...9, GUN CLAUSE, ...31, ...32
```

```
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.
```

From the output of `readr`, you can see all kinds of problems from the get go. Two columns are missing names entirely. Three columns repeat – first name, middle name and name extension. And many field names have spaces or other not-allowed characters. Not to mention: All of them are in ALL CAPS.

Janitor makes this easy to fix. How easy? This easy.

```
inmates |> clean_names()
```

```
# A tibble: 7,674 x 32
```

|    | id_number | committed_last_name | first_name_3 | middle_name_4 | name_extension_5 |
|----|-----------|---------------------|--------------|---------------|------------------|
|    | <dbl>     | <chr>               | <chr>        | <chr>         | <chr>            |
| 1  | 6145      | KANE                | THOMAS       | <NA>          | <NA>             |
| 2  | 20841     | ARNOLD              | WILLIAM      | L             | <NA>             |
| 3  | 25324     | WALKER              | RICHARD      | T             | <NA>             |
| 4  | 25565     | ALVAREZ             | THOMAS       | A             | <NA>             |
| 5  | 26103     | ADAMS               | BRIAN        | J             | <NA>             |
| 6  | 26547     | KIRBY               | RONALD       | EUGENE        | <NA>             |
| 7  | 27471     | NIEMANN             | MAX          | A             | <NA>             |
| 8  | 27666     | ORTIZ               | LAWRENCE     | <NA>          | <NA>             |
| 9  | 27767     | POINDEXTER          | EDWARD       | <NA>          | <NA>             |
| 10 | 27778     | DITTRICH            | LADDIE       | F             | <NA>             |

```
# i 7,664 more rows
```

```
# i 27 more variables: legal_last_name <chr>, first_name_7 <chr>,  
# middle_name_8 <chr>, name_extension_9 <lgl>, date_of_birth <chr>,  
# race_desc <chr>, gender <chr>, facility <chr>,  
# current_sentence_pardoned_or_commuted_date <chr>, gun_clause <lgl>,  
# sentence_begin_date <chr>, min_term_year <chr>, min_month <dbl>,  
# min_day <dbl>, max_term_year <chr>, max_month <dbl>, max_day <dbl>, ...
```

Just like that, all lower case, all one word, no backticks necessary to confuse our code later on.

Another thing janitor does well is to make it easy to drop empty columns. Remember the two unnamed columns in the data? Turns out they're unnamed because there's nothing in them. Nada. Blank. We could use `select` but janitor reduces the labor involved there.

First, let's see how many columns we have.

```
inmates |> ncol()
```

```
[1] 32
```

And by using `remove_empty("cols")`, how many do we get rid of?

```
inmates |> clean_names() |> remove_empty("cols") |> ncol()
```

```
[1] 28
```

So this tells us there's three completely empty columns in our data. So why keep them around.

So we can run all of this together and get a dataset with useful columns and clean headers.

```
inmates |> clean_names() |> remove_empty("cols") -> clean_headers_inmates
```

## 11.2 Duplicates

One of the most difficult problems to fix in data is duplicates in the data. They can creep in with bad joins, bad data entry practices, mistakes – all kinds of reasons. One trick is determining if a duplicate is indeed a duplicate.

So the question is, do we have any inmates repeated? Anyone in prison twice?

Here we'll use a function called `get_dupes`. And we'll use the inmate's last name, first name and date of birth. The likelihood that someone has the same name and date of birth is very small, so if there are no duplicates, we should get zero records returned.

```
clean_headers_inmates |> get_dupes(committed_last_name, first_name_3, date_of_birth)
```

```
# A tibble: 2 x 29
  committed_last_name first_name_3 date_of_birth dupe_count id_number
  <chr>                <chr>        <chr>          <int>    <dbl>
1 WALLACE              PAMELA      7/11/1966         2     99240
2 WALLACE              PAMELA      7/11/1966         2     99955
# i 24 more variables: middle_name_4 <chr>, name_extension_5 <chr>,
#   legal_last_name <chr>, first_name_7 <chr>, middle_name_8 <chr>,
#   race_desc <chr>, gender <chr>, facility <chr>,
#   current_sentence_pardoned_or_commuted_date <chr>,
#   sentence_begin_date <chr>, min_term_year <chr>, min_month <dbl>,
#   min_day <dbl>, max_term_year <chr>, max_month <dbl>, max_day <dbl>,
#   parole_eligibility_date <chr>, earliest_possible_release_date <chr>, ...
```

Uh oh. We get two Pamela Wallaces born on the same day in 1966. But is it a duplicate record? Look closely. Two different `id_numbers`. In two different facilities on two different dates. Two different sentencing dates. Is it a duplicate record or the same person entering the system two different times?

## 11.3 Inconsistency

Janitor also has some handy tools for our data smells. One is called `tabyl`, which creates a table of unique records in a single field.

So does the Department of Corrections record gender consistently? `tabyl` will tell us and will tell us a little bit about the data.

```
clean_headers_inmates |> tabyl(gender)
```

```
gender    n    percent
FEMALE  732 0.09538702
MALE   6942 0.90461298
```

So the Department of Corrections clearly doesn't buy into more modern sentiments about gender, but they are at least consistent. Every inmate has a gender – no NAs – and note that 90 percent of inmates are men.

How about race?

```
clean_headers_inmates |> tabyl(race_desc)
```

| race_desc        | n    | percent      | valid_percent |
|------------------|------|--------------|---------------|
| ASIAN            | 61   | 0.0079489184 | 0.0079520271  |
| BLACK            | 2037 | 0.2654417514 | 0.2655455612  |
| HISPANIC         | 1059 | 0.1379984363 | 0.1380524052  |
| NATIVE AMERICAN  | 349  | 0.0454782382 | 0.0454960240  |
| OTHER            | 56   | 0.0072973677 | 0.0073002216  |
| PACIFIC ISLANDER | 7    | 0.0009121710 | 0.0009125277  |
| WHITE            | 4102 | 0.5345321866 | 0.5347412332  |
| <NA>             | 3    | 0.0003909304 | NA            |

Three people do not have a race – and according to the Census Bureau, Hispanic is not a race, it’s an ethnicity – but otherwise, it looks solid. There’s very little in the way of inconsistency.

How about what facilities they are in?

```
clean_headers_inmates |> tabyl(facility)
```

| facility                       | n    | percent    | valid_percent |
|--------------------------------|------|------------|---------------|
| COMMUNITY CORRECTIONS-LINCOLN  | 1276 | 0.16627574 | 0.16887242    |
| COMMUNITY CORRECTIONS-OMAHA    | 368  | 0.04795413 | 0.04870302    |
| DIAGNOSTIC & EVALUATION CENTER | 778  | 0.10138129 | 0.10296453    |
| LINCOLN CORRECTIONAL CENTER    | 571  | 0.07440709 | 0.07556908    |
| NEBRASKA CORR CENTER FOR WOMEN | 480  | 0.06254887 | 0.06352567    |
| NEBRASKA CORR YOUTH FACILTY    | 83   | 0.01081574 | 0.01098465    |
| NEBRASKA STATE PENITENTIARY    | 1588 | 0.20693250 | 0.21016411    |
| OMAHA CORRECTIONAL CENTER      | 1059 | 0.13799844 | 0.14015352    |
| TECUMSEH STATE COR INSTITUTION | 1104 | 0.14386239 | 0.14610905    |
| WORK ETHIC CAMP                | 249  | 0.03244722 | 0.03295394    |
| <NA>                           | 118  | 0.01537660 | NA            |

Not sure how I feel about 118 inmates not having a facility. That’s probably worth investigating. At least one, I know about – it lists the inmate as having escaped in the 1960s and never found. Not sure about the others.

But sometimes, NAs are not bad data. Sometimes they’re just NA. Let’s look at `inst_release_type` or how inmates were released.

```
clean_headers_inmates |> tabyl(inst_release_type)
```

| inst_release_type    | n    | percent      | valid_percent |
|----------------------|------|--------------|---------------|
| DISCRETIONARY PAROLE | 1391 | 0.1812614021 | 0.5506730008  |

|                  |      |              |              |
|------------------|------|--------------|--------------|
| ESCAPE           | 51   | 0.0066458170 | 0.0201900238 |
| MANDATORY PAROLE | 2    | 0.0002606203 | 0.0007917656 |
| RE-PAROLE        | 3    | 0.0003909304 | 0.0011876485 |
| RELEASED TO PRS  | 1079 | 0.1406046390 | 0.4271575614 |
| <NA>             | 5148 | 0.6708365911 | NA           |

By far the largest group here is NA. Why is that? They haven't been released yet. They're still in prison.



## 12 Data Cleaning Part III: Open Refine

Gather 'round kids and let me tell you a tale about your author. In college, your author got involved in a project where he mapped crime in the city, looking specifically in the neighborhoods surrounding campus. This was in the mid 1990s. Computers were under powered. Tools were pretty primitive. I was given a database of nearly 50,000 calls for service.

And then I learned that addresses were not stored in a standard way. However the officer wrote it down, that's how it was recorded.

What did that mean?

It meant the Lincoln Police Department came up with dozens of ways to say a single place. And since the mapping software needed the addressed to be in a specific form, I had to fix them. For example, I will go to my grave knowing that Lincoln High School's street address is 2229 J Street. Police officers wrote down LHS, L.H.S., Lincoln HS, Lincoln H.S., LHS (J Street), 2229 J, 2229 J ST, St., Street and on and on and on. That one was relatively easy. The local convenience store chain, with 8 locations around the city, was harder. I had to use the patrol district to locate them.

It took me four months to clean up more than 30,000 unique addresses and map them.

I tell you this because if I had Open Refine, it would have taken me a week, not four months. Every time I talk about Open Refine, I remember this, and I get mad.

We're going to explore two ways into Open Refine: Through R, and through Open Refine itself.

### 12.1 Refinr, Open Refine in R

What is Open Refine?

Open Refine is a series of tools – algorithms – that find small differences in text and helps you fix them quickly. How Open Refine finds those small differences is through something called clustering. The algorithms behind clustering are not exclusive to Open Refine, so they can be used elsewhere.

Enter `refinr`, a package that contains the same clustering algorithms as Open Refine but all within R. Go ahead and install it if you haven't already by opening the console and running `install.packages("refinr")`. Then we can load libraries as we do.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2

-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
library(refinr)
library(janitor)
```

Attaching package: 'janitor'

The following objects are masked from 'package:stats':

`chisq.test`, `fisher.test`

Let's load a simple dataset where we know there's a simple problem. Let's load the dataset of mountainlion sightings.

```
mountainlions <- read_csv("https://mattwaite.github.io/datajournalismfiles/mountainlions.csv")
```

Rows: 393 Columns: 4

```
-- Column specification -----
Delimiter: ","
chr (3): Cofirm Type, COUNTY, Date
dbl (1): ID
```

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.

The issue in this dataset, if you look carefully, is that there's two Sheridan counties – a Sheridan and a sheridan.

```
mountainlions |> group_by(COUNTY) |> tally()
```

```
# A tibble: 42 x 2
  COUNTY      n
  <chr>    <int>
1 Banner      6
2 Blaine      3
3 Box Butte   4
4 Brown     15
5 Buffalo     3
6 Cedar       1
7 Cherry     30
8 Custer      8
9 Dakota      3
10 Dawes    111
# i 32 more rows
```

The first merging technique we'll try is the `key_collision_merge`. The key collision merge function takes each string and extracts the key parts of it. It then puts every key in a bin based on the keys matching. So in this case, it finds sheridan and Sheridan and recognizes that the keys match, and since Sheridan is more common, it uses that one.

One rule you should follow: **do not overwrite your original fields**. Always work on a copy. If you overwrite your original field, how will you know if it did the right thing? How can you compare it to your original data? To follow this, I'm going to mutate a new field called `CleanCounty` and put the results of key collision merge there.

Then, to show it worked, I'll do the same group and count.

```
mountainlions |>
  mutate(CleanCounty = key_collision_merge(COUNTY)) |>
  group_by(CleanCounty) |> tally()
```

```
# A tibble: 41 x 2
  CleanCounty      n
  <chr>    <int>
1 Banner      6
2 Blaine      3
3 Box Butte   4
```

```

4 Brown          15
5 Buffalo        3
6 Cedar          1
7 Cherry         30
8 Custer         8
9 Dakota         3
10 Dawes         111
# i 31 more rows

```

And just like that, instead of 35 and 2 in two different Sheridan counties, we have 37 in one Sheridan County.

## 12.2 More complex issues

Let's load a [dataset of the charges Nebraska prison inmates](#) were convicted of, which is why they're in prison. We'll also use `janitor`'s `clean_names` function to give us usable headers.

```
charges <- read_csv("data/charges.csv") |> clean_names()
```

```

Rows: 19226 Columns: 14
-- Column specification -----
Delimiter: ","
chr (9): OFFENSE MINIMUM YEAR OR TERM, OFFENSE MAXIMUM YEAR OR TERM, OFFENSE...
dbl (5): ID NUMBER, MINIMUM MONTH, MINIMUM DAY, MAXIMUM MONTH, MAXIMUM DAY

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.

```

The problematic – and among the most interesting – fields in this dataset is the name of the charges. What is the most common charge keeping someone in prison?

I'm not going to run the list here because it's long – thousands of lines long. You should run it yourself:

```
charges |> tabyl(offense_arrest_desc)
```

You'll see right away that there's problems. There's dozens upon dozens of charges that are the same thing, just slightly different. There's 4003 unique charges, and many of them are duplicates.

```
charges |> group_by(offense_arrest_desc) |> tally(sort=TRUE) |> nrow()
```

[1] 4003

So how does `key_collision_merge` do with this?

```
charges |>
  mutate(
    clean_charges = key_collision_merge(offense_arrest_desc)
  ) |>
  group_by(clean_charges) |>
  tally() |>
  nrow()
```

[1] 3420

Cuts down the duplicates by 583. But since the charges are often multiple words, we should try using `n_gram_merge`, which looks at multiple words.

Here's an example using sensible defaults for weighting – unfortunately [the documentation](#) doesn't do much to explain what they are.

```
charges |>
  mutate(
    clean_charges = n_gram_merge(
      offense_arrest_desc, weight = c(d = 0.2, i = 0.2, s = 1, t = 1))) |>
  group_by(clean_charges) |>
  tally() |>
  nrow()
```

[1] 3036

Cuts it down by almost 1000. That seems pretty good. Here's a different method, using a method that turns words into phonetic spellings called `soundex`.

```
charges |>
  mutate(
    clean_charges = n_gram_merge(
      offense_arrest_desc, method = "soundex", useBytes = TRUE
```

```

    )) |>
  group_by(clean_charges) |>
  tally() |>
  nrow()

```

[1] 2688

Cut it down by almost 1400!

BUT.

Are they right?

We have no idea. Let's look at the first 30 rows.

```

charges |>
  mutate(
    clean_charges = n_gram_merge(
      offense_arrest_desc, method = "soundex", useBytes = TRUE
    ) |>
    filter(clean_charges != offense_arrest_desc) |> select(offense_arrest_desc, clean_charges)

```

# A tibble: 30 x 2

| offense_arrest_desc<br><chr>     | clean_charges<br><chr>         |
|----------------------------------|--------------------------------|
| 1 ASSLT WI INFLICT BODILY INJURY | ASSLT W/I INFLCT BODILY INJURY |
| 2 STAB W/I TO KILL WOUND OR MAIM | STAB W/I KILL, WOUND, OR MAIM  |
| 3 USE OF WEAPON TO COMMIT FELONY | USE WEAPON TO COMMIT FELONY    |
| 4 3RD DEGREE ASSAULT ON OFFICER  | ASSAULT ON OFFICER 3RD DEGREE  |
| 5 3RD DEGREE ASSAULT ON OFFICER  | ASSAULT ON OFFICER 3RD DEGREE  |
| 6 POSS CONTROLLED SUBSTANCE      | POSSESS CONTROLLED SUBSTANCE   |
| 7 MANUF/DIST CONT SUBST - MARIJ. | DISTRIBUTION OF C/S-MARIJUANA  |
| 8 POSS DEADLY WEAP BY FELON      | POSSESS DEADLY WEAPON BY FELON |
| 9 POSS FIREARM BY FELON          | POS FIREARM BY FELON           |
| 10 3RD DGR ASSAULT ON AN OFFICER | ASSAULT ON AN OFFICER 3RD DEGR |

# i 20 more rows

If you look carefully, you'll see a lot of success here. But look at line 23. The charge is theft by taking \$0-500. The clean version? Theft by taking \$5000. That's a big difference, and a bad miss.

So can we trust automated data cleaning?

This note from the documentation is exceedingly important:

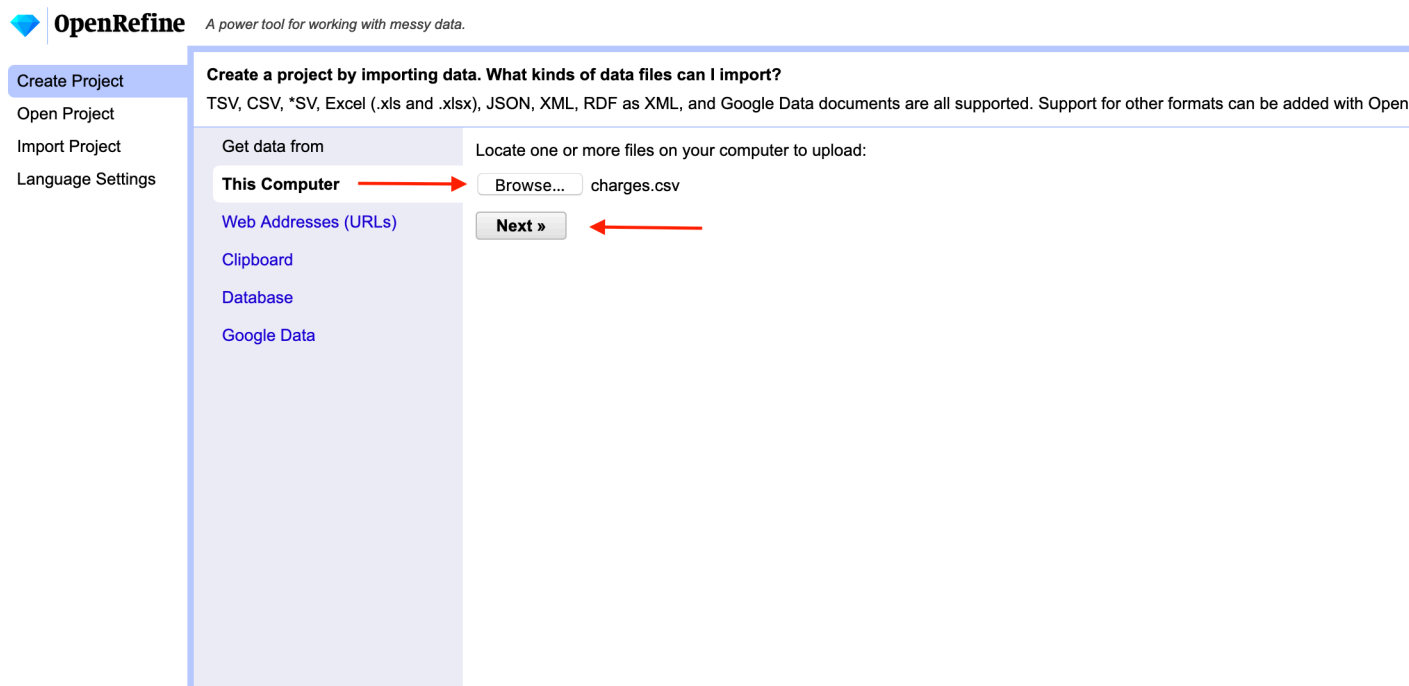
This package is NOT meant to replace OpenRefine for every use case. For situations in which merging accuracy is the most important consideration, OpenRefine is preferable. Since the merging steps in refinr are automated, there will usually be more false positive merges, versus manually selecting clusters to merge in OpenRefine.

## 12.3 Manually cleaning data with Open Refine

Open Refine is free software. [You should download and install it.](#) Refinr is great for quick things on smaller datasets that you can check to make sure it's not up to any mischief. For bigger datasets, Open Refine is the way to go. And it has a lot more tools than refinr does (by design, but still).

After you install it, run it. Open Refine works in the browser, and the app spins up a small web server visible only on your computer to interact with it. A browser will pop up automatically.

You first have to import your data into a project.



After your data is loaded into the app, you'll get a screen to look over what the data looks like. On the top right corner, you'll see a button to create the project.

Create Project

Open Project

Import Project

Language Settings

« Start Over

Configure Parsing Options

Project name c

|     | ID NUMBER | OFFENSE | MINIMUM YEAR OR TERM | MINIMUM MONTH | MINIMUM DAY | OFFENSE MAXIMUM YEAR OR TERM | MAXIMUM MONTH | MAXIMUM |
|-----|-----------|---------|----------------------|---------------|-------------|------------------------------|---------------|---------|
| 1.  | 6145      | 1       |                      | 0             | 0           | 3                            | 0             | 0       |
| 2.  | 20841     | 10      |                      | 0             | 0           | LFE                          |               |         |
| 3.  | 20841     | 10      |                      | 0             | 0           | LFE                          |               |         |
| 4.  | 25324     | 10      |                      | 0             | 0           | LFE                          |               |         |
| 5.  | 25565     | LFE     |                      |               |             | LFE                          |               |         |
| 6.  | 26103     | LFE     |                      |               |             | LFE                          |               |         |
| 7.  | 26547     | LFE     |                      |               |             | LFE                          |               |         |
| 8.  | 27471     | 20      |                      | 0             | 0           | LFE                          |               |         |
| 9.  | 27666     | LFE     |                      |               |             | LFE                          |               |         |
| 10. | 27666     | 10      |                      | 0             | 0           | 15                           | 0             | 0       |
| 11. | 27666     | 6       |                      | 0             | 0           | 10                           | 0             | 0       |
| 12. | 27767     | LFE     |                      |               |             | LFE                          |               |         |
| 13. | 27778     | 2       |                      | 0             | 0           | 3                            | 0             | 0       |
| 14. | 27778     | 80      |                      | 0             | 0           | LFE                          |               |         |
| 15. | 27778     | 0       |                      | 0             | 0           | 1                            | 0             | 0       |
| 16. | 27778     | 5       |                      | 0             | 0           | 5                            | 0             | 0       |
| 17. | 29123     | LFE     |                      |               |             | LFE                          |               |         |
| 18. | 29138     | 10      |                      | 0             | 0           | LFE                          |               |         |
| 19. | 29257     | 10      |                      | 0             | 0           | LFE                          |               |         |
| 20. | 29257     | 1       |                      | 0             | 0           | 10                           | 0             | 0       |
| 21. | 29267     | 10      |                      | 0             | 0           | 30                           | 0             | 0       |
| 22. | 29267     | 10      |                      | 0             | 0           | 30                           | 0             | 0       |
| 23. | 29267     | 10      |                      | 0             | 0           | 30                           | 0             | 0       |

Parse data as

CSV / TSV / separator-based files

Line-based text files

Fixed-width field text files

PC-Axis text files

JSON files

MARC files

JSON-LD files

Character encoding

Columns are separated by

☒ commas (CSV)
 ☐ tabs (TSV)
 ☐ custom: ,

Escape special characters with \

☐ Column names (comma separated):

☐ Ignore first
 ☒ Parse next
 ☐ Discard initial
 ☐ Load at most
 ☒ Use character
 ☐ Parse cell text in numbers, dates

The real power in Open Refine is in faceting. In our case, we're specifically going to use text faceting. Next to the OFFENSE ARREST DESC header, click the down arrow, then facet, then text facet.



Extensions: v

« first < previous 1 - 10 next »

| MUM DAY | OFFENSE ARRE             | FELONY MSDMNR           | OFFENSE TYPE | OFFENSE ATTEI |
|---------|--------------------------|-------------------------|--------------|---------------|
|         | Facet                    | Text facet              |              |               |
|         | Text filter              | Numeric facet           |              |               |
|         | Edit cells               | Timeline facet          |              |               |
|         | Edit column              | Scatterplot facet       |              |               |
|         | Transpose                | Custom text facet...    |              |               |
|         | Sort...                  | Custom Numeric Facet... |              |               |
|         | View                     | Customized facets       |              |               |
|         | Reconcile                | FELONY                  | *            |               |
|         | MURDER 2ND DEGREE        | FELONY                  | *            |               |
|         | MURDER 1ST DEGREE        | FELONY                  | *            |               |
|         | INMATE DETAININE ANOTHER | FELONY                  | A            |               |

After that, a new box will appear on the left. It tells us how many unique offenses are there: 4,082. And, there's a button on the right of the box that says Cluster. Click that.

**OFFENSE ARREST DESC** [change](#)

4082 choices Sort by: **name** count [Cluster](#)

PANDERING 1

1ST DEG ASSAULT/VOP 1

1ST DEG ASSLT ON OFFICER 2

1ST DEG CRIMINAL TRESPASS 1

1ST DEG DOMESTIC ASSAULT 1

1ST DEG DOMESTIC ASSAULT/VOP 1 [edit](#) [include](#)

1ST DEG FALSE IMPRISONMENT-VOP 1

1ST DEG FORCIBLE SEX ASSULT 1

1ST DEG FORCIBLE SEX'L

The default clustering algorithm used is key collision, using the fingerprint function. This is the same method we used with Sheridan County above.

At the top, you'll see which method was used, and how many clusters that algorithm identified. Then, below that, you can see what those clusters are. Then, using human judgement, you can say if you agree with the cluster. If you do, click the merge checkbox. When it merges, the new result will be what it says in New Cell Value. Most often, that's the row with the most common result.

Cluster & Edit column "OFFENSE ARREST DESC"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. [Find out more](#)

Methodkey collision

Keying Functionfingerprint

303 clusters found

| Cluster Size | Row Count | Values in Cluster  | Merge?                   | New Cell Value           |
|--------------|-----------|--|--------------------------|--------------------------|
| 5            | 65        | <ul style="list-style-type: none"> <li>• POS CNTRL SUB - METH (27 rows)</li> <li>• POS CNTRL SUB (METH) (14 rows)</li> <li>• POS CNTRL SUB -METH (13 rows)</li> <li>• POS CNTRL SUB METH (10 rows)</li> <li>• POS CNTRL SUB- METH (1 rows)</li> </ul>  | <input type="checkbox"/> | POS CNTRL SUB - METH     |
| 5            | 9         | <ul style="list-style-type: none"> <li>• POSSESS C/S METH VOP (4 rows)</li> <li>• VOP POSSESS C/S METH (2 rows)</li> <li>• POSSESS C/S - METH VOP (1 rows)</li> <li>• POSSESS C/S- METH VOP (1 rows)</li> <li>• POSSESS CS (METH) VOP (1 rows)</li> </ul>                                    | <input type="checkbox"/> | POSSESS C/S METH VOP     |
| 5            | 34        | <ul style="list-style-type: none"> <li>• THEFT BY RECEIVING \$5000+ (29 rows)</li> <li>• THEFT BY RECEIVING \$5000 + (2 rows)</li> <li>• THEFT BY RECEIVING \$5,000 &gt; (1 rows)</li> <li>• THEFT BY RECEIVING \$5000 (1 rows)</li> <li>• THEFT BY RECEIVING \$5000&gt; (1 rows)</li> </ul> | <input type="checkbox"/> | THEFT BY RECEIVING \$500 |
| 5            | 388       | <ul style="list-style-type: none"> <li>• ASSAULT 2ND DEGREE (370 rows)</li> <li>• ASSAULT 2ND DEGREE (11 rows)</li> <li>• 2ND DEGREE ASSAULT (3 rows)</li> <li>• ASSAULT 2ND DEGREE (3 rows)</li> <li>• ASSAULT - 2ND DEGREE (1 rows)</li> </ul>   | <input type="checkbox"/> | ASSAULT 2ND DEGREE       |

# Choices in Cluster

# Rows in Cluster

Average Length of Choices

Length Variance of Choices

Select All

Unselect All

Export Clusters

Merge Selected & Re-Cluster

Merge Selected & Close

Close

Now begins the fun part: You have to look at all 303 clusters found and decide if they are indeed valid. The key collision method is very good, and very conservative. You'll find that most of them are usually valid.

When you're done, click Merge Selected and Re-Cluster.

If any new clusters come up, evaluate them. Repeat until either no clusters come up or the clusters that do come up are ones you reject.

Now. Try a new method. Rinse and repeat. You'll keep doing this, and if the dataset is reasonably clean, you'll find the end.

If it's not, it'll go on forever.

## Cluster & Edit column "OFFENSE ARREST DESC"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. [Find out more.](#)

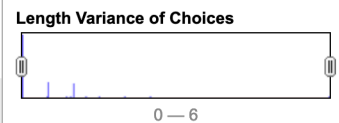
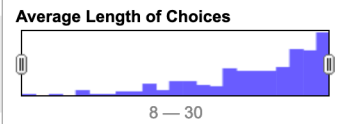
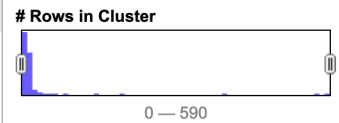
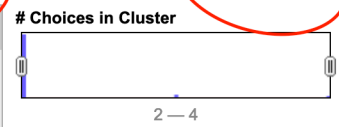
Method key collision

Keying Function ngram-fingerprint

Ngram Size 2

265 clusters found

| Cluster Size | Row Count | Values in Cluster   | Merge?                   | New Cell Value         |
|--------------|-----------|---|--------------------------|------------------------|
| 4            | 38        | <ul style="list-style-type: none"> <li>DEL/DSP/DST/MAN/POS METH (34 rows)</li> <li>DEL/DSP/DST/MAN/POS-METH (2 rows)</li> <li>DEL/DSP/DST/MAN POS METH (1 rows)</li> <li>DEL/DSP/DST/MAN POS-METH (1 rows)</li> </ul> | <input type="checkbox"/> | DEL/DSP/DST/MAN/POS MI |
| 4            | 81        | <ul style="list-style-type: none"> <li>POS CNTRL SUB-METH/VOP (63 rows)</li> <li>POS CNTRL SUB (METH) VOP (9 rows)</li> <li>POS CNTRL SUB METH-VOP (5 rows)</li> <li>POS CNTRL SUB-METH VOP (4 rows)</li> </ul>       | <input type="checkbox"/> | POS CNTRL SUB-METH/VC  |
| 3            | 53        | <ul style="list-style-type: none"> <li>POSSESS C/S METH (49 rows)</li> <li>POSSESS C/S-METH (3 rows)</li> <li>POSSES CS - METH (1 rows)</li> </ul>  | <input type="checkbox"/> | POSSESS C/S METH       |
| 3            | 7         | <ul style="list-style-type: none"> <li>DOMESTIC ASSAULT 3RD DEG VOP (3 rows)</li> <li>DOMESTIC ASSAULT 3RD DEG/VOP (3 rows)</li> <li>DOMESTIC ASSAULT-3RD DEG VOP (1 rows)</li> </ul>                                 | <input type="checkbox"/> | DOMESTIC ASSAULT 3RD I |
| 3            | 50        | <ul style="list-style-type: none"> <li>DUI 4TH OFFENSE (48 rows)</li> <li>DUI 4TH OFFENSE (1 rows)</li> <li>DUI-4TH OFFENSE (1 rows)</li> </ul>   | <input type="checkbox"/> | DUI 4TH OFFENSE        |



Select All Unselect All

Export Clusters

Merge Selected & Re-Cluster

Merge Selected & Close

Close

**Cluster & Edit column "OFFENSE ARREST DESC"**

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. [Find out more.](#)

Method: nearest neighbor levenshtein Radius: 1.0 Block Chars: 6 211 clusters found

| Cluster Size | Row Count | Values in Cluster  | Merge?                              | New Cell Value           |
|--------------|-----------|--|-------------------------------------|--------------------------|
| 4            | 65        | <ul style="list-style-type: none"> <li>DUI 4TH OFFENSE (50 rows)</li> <li>DUI 5TH OFFENSE (12 rows)</li> <li>DWI 4TH OFFENSE (2 rows)</li> <li>DUI 6TH OFFENSE (1 rows)</li> </ul>         | <input type="checkbox"/>            | DUI 4TH OFFENSE          |
| 3            | 4         | <ul style="list-style-type: none"> <li>MAN/DIST/POSS W/I DIST METH (2 rows)</li> <li>MAN/DIST/POS W/I DIST METH (1 rows)</li> <li>MAN/DST/POS W/I DIST METH (1 rows)</li> </ul>            | <input checked="" type="checkbox"/> | MAN/DIST/POSS W/I DIST I |
| 3            | 12        | <ul style="list-style-type: none"> <li>POS CNTRL SUB-CLONAZEPAM (10 rows)</li> <li>POS CNTRL SUB-CLONAZEPAN (1 rows)</li> <li>POS CNTRL SUB-CLONZEPAM (1 rows)</li> </ul>                  | <input checked="" type="checkbox"/> | POS CNTRL SUB-CLONAZE    |
| 3            | 10        | <ul style="list-style-type: none"> <li>POSS DEADLY WEAP BY PROH PRSON (5 rows)</li> <li>POSS DEADLY WEAP BY PROH PRSN (4 rows)</li> <li>POSS DEADLY WEAP BY PROHB PRSN (1 rows)</li> </ul> | <input checked="" type="checkbox"/> | POSS DEADLY WEAP BY P    |
| 3            | 21        | <ul style="list-style-type: none"> <li>POS CNTRL SUB-ALPRAZOLAM (15 rows)</li> <li>POSS CNTRL SUB-ALPRAZOLAM (5 rows)</li> <li>POS CNTRL SUB-ALZRAZOLAM (1 rows)</li> </ul>                | <input checked="" type="checkbox"/> | POS CNTRL SUB-ALPRAZC    |

Select All Unselect All Export Clusters Merge Selected & Re-Cluster Merge Selected & Close Close

# Choices in Cluster

# Rows in Cluster

Average Length of Choices

Length Variance of Choices

A question for all data analysts – if the dataset is bad enough, can it ever be cleaned?

There's no good answer. You have to find it yourself.

## 13 Cleaning Data Part IV: PDFs

The next circle of Hell on the Dante’s Inferno of Data Journalism is the PDF. Governments everywhere love the PDF and publish all kinds of records in a PDF. The problem is a PDF isn’t a data format – it’s a middle finger, saying I’ve Got Your Accountability Right Here, Pal.

It’s so ridiculous that there’s a constellation of tools that do nothing more than try to harvest tables out of PDFs. There are online services like [CometDocs](#) where you can upload your PDF and point and click your way into an Excel file. There are mobile device apps that take a picture of a table and convert it into a spreadsheet. But one of the best is a tool called [Tabula](#). It was build by journalists for journalists.

There is a version of Tabula that will run inside of R – a library called Tabulizer – but the truth is I’m having the hardest time installing it on my machine, which leads me to believe that trying to install it across a classroom of various machines would be disasterous. The standalone version works just fine.

Unfortunately, harvesting tables from PDFs with Tabula is an exercise in getting your hopes up, only to have them dashed. We’ll start with an example.

### 13.1 When it looks good, but goes wrong

Every year, the University of Nebraska-Lincoln publishes dozens of PDFs that give you interesting demographic information about students, the faculty and a variety of other things. But all of it – every little bit of it – is in a PDF. And most of them are designed to look “nice” not convey data. Even when they do very obviously look like they came from a spreadsheet – like someone printed the spreadsheet to a PDF so they could put it on the web – it doesn’t work.

A perfect example of this is the data showing the [breakdown of students by degree, major, race and sex](#). Open it up, it looks like a spreadsheet but in a PDF.

Total Headcount Enrollment: by College, Major, Degree, Race and Gender (Table 100)  
Fall 2018

| College                                | Major Name                                     | Major 1 | Degree Id | American Indian/Alaska Native |        | Asian |        | Black-Non Hispanic |        | Hispanic |        | Non-resident Alien |        | Pacific Islander |        |
|--|--|---------|-----------|-------------------------------|--------|-------|--------|--------------------|--------|----------|--------|--------------------|--------|------------------|--------|
|  |  |         |           | Male                          | Female | Male  | Female | Male               | Female | Male     | Female | Male               | Female | Male             | Female |
| College of Agri Sci and Natl Resources | Agribusiness                                   | ABUS    | B1AB      |                               |        |       |        | 1                  |        | 2        |        | 1                  |        |                  |        |
|  | Agricultural & Env Sci Comm                    | AESC    | B1SC      |                               |        |       |        |                    |        | 2        |        |                    | 1      |                  |        |
|  | Agricultural Economics                         | AECN    | B1AE      |                               |        |       |        |                    |        | 1        | 2      | 3                  | 2      |                  |        |
|  | Agricultural Education                         | AEDU    | B1ED      |                               |        |       |        |                    |        |          | 1      |                    | 1      |                  |        |
|  | Agronomy                                       | AGRO    | B1AG      |                               |        |       |        | 2                  |        | 2        |        | 2                  | 5      | 1                |        |
|  | Animal Science                                 | ASCI    | B1AS      |                               |        |       | 4      | 1                  | 1      | 2        | 12     |                    |        |                  |        |
|  | Applied Climate Science                        | APCS    | B1AC      |                               |        |       |        |                    |        | 1        |        |                    | 1      |                  |        |
|  | Applied Science                                | APSC    | B1AP      | 1                             |        |       |        |                    | 1      |          |        |                    |        |                  |        |
|  | Biochemistry                                   | BIOC    | B1BC      |                               | 1      | 3     | 5      | 1                  |        | 2        | 2      | 3                  | 3      |                  |        |
|  | Environmental Restoration Sci                  | ENRS    | B1ER      |                               |        |       |        |                    | 1      |          | 2      | 2                  | 2      |                  |        |
|  | Environmental Studies                          | ENVR    | B1ES      |                               |        |       |        |                    |        | 2        | 5      | 1                  | 1      |                  |        |
|  | Fisheries and Wildlife                         | FWL     | B1FW      | 1                             |        |       | 1      |                    | 2      | 5        | 10     |                    |        |                  |        |
|  | Food Science and Technology                    | FDST    | B1FS      |                               |        | 3     | 1      | 1                  |        | 1        | 1      | 22                 | 44     |                  |        |
|  | Food Tech for Companion Animal                 | FTCA    | B1FC      |                               |        |       |        |                    |        |          |        |                    |        |                  |        |
|  | Forensic Science                               | FORS    | B1FO      |                               |        | 1     | 3      |                    | 6      | 2        | 9      | 3                  |        |                  |        |
|  | General - CASNR                                | EXPS    | NDEG      |                               |        |       |        |                    |        |          |        |                    |        |                  |        |
|  | Grassland Ecology & Management                 | GECM    | B1GE      |                               |        |       |        |                    |        |          |        |                    |        |                  |        |
|  | Grazing Livestock Systems                      | GRLS    | B1GL      |                               |        |       |        |                    |        |          |        |                    |        |                  |        |
|  | HRTM-(Ecotourism/Parks & Recreation Options .. | HRTA    | BAHR      |                               |        | 1     | 1      |                    |        |          | 2      | 1                  |        |                  |        |
|  | Horticulture                                   | HORT    | B1HO      |                               |        |       |        |                    |        | 2        | 3      |                    | 2      |                  |        |
|  | Insect Science                                 | INSC    | B1IS      |                               |        |       |        |                    |        |          | 1      | 1                  |        |                  |        |
|  | Integrated Science                             | ITSC    | B1IN      |                               |        |       |        |                    |        |          |        | 99                 | 56     |                  |        |
|  | Mechanized Systems Management                  | MSYM    | B1MS      |                               |        | 2     |        |                    |        | 3        | 1      | 1                  |        |                  |        |

[Download and install Tabula](#). Tabula works much the same way as Open Refine does – it works in the browser by spinning up a small webserver in your computer.

When Tabula opens, you click browse to find the PDF on your computer somewhere, and then click import.

After it imports, click autodetect tables. You'll see red boxes appear around what Tabula believes are the tables. You'll see it does a pretty good job at this.

1.

Total Headcount Enrollment: by College, Major, Degree, Race and Gender (Table 100)  
Fall 2018

| College                                      | Major Name                                      | Major ID | Degree ID | American Indian/Alaska Native |        | Asian |        | Black/African American |        | Hispanic/Latino |        | Non-resident Alien |        | Pacific Islander |        | Two or More Races |        | Unknown |        | White |
|--|---|----------|-----------|-------------------------------|--------|-------|--------|------------------------|--------|-----------------|--------|--------------------|--------|------------------|--------|-------------------|--------|---------|--------|-------|
|  |   |          |           | Male                          | Female | Male  | Female | Male                   | Female | Male            | Female | Male               | Female | Male             | Female | Male              | Female | Male    | Female | Male  |
| College of Agriculture and Natural Resources | Agriculture                                     | AGUS     | AGAD      |                               |        |       |        | 1                      |        | 2               |        |                    |        |                  |        |                   |        | 1       |        | 1     |
|  | Agriculture & City Sci. Comm.                   | AGUS     | AGSC      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Agricultural Economics                          | AGUS     | AGAE      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Agricultural Education                          | AGUS     | AGED      |                               |        |       |        |                        |        | 1               |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Agriculture                                     | AGUS     | AGAD      |                               |        |       |        | 2                      |        | 2               |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Animal Science                                  | AGUS     | AGAS      |                               |        |       |        | 4                      | 1      | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Applied Climate Science                         | APCS     | AGAC      |                               |        |       |        |                        |        | 1               |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Applied Science                                 | APSC     | AGAP      | 1                             |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Biochemistry                                    | BISC     | BISC      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Environmental Restoration Sci                   | ENRS     | AGER      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Environmental Studies                           | ENRS     | AGES      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Fisheries and Wildlife                          | FWSL     | AGFW      | 1                             |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Food Science and Technology                     | FSTT     | AGFS      |                               |        |       |        |                        |        | 1               | 1      | 1                  | 1      |                  |        |                   |        |         |        | 1     |
|  | Food Tech for Companion Animal                  | FTCA     | AGFC      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Forensic Science                                | FORS     | AGFO      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | General - CAGNR                                 | ENRS     | AGGR      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Grassland Ecology & Management                  | GEOM     | AGGE      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Grazing Livestock Systems                       | GLSLS    | AGGL      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | HRTM (Distraction/Partis Misconception Systems) | HRTM     | AGHM      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Horticulture                                    | HORT     | AGHO      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Insect Science                                  | ISCS     | AGIS      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Integrated Science                              | ITSC     | AGIN      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Mechanical Systems Management                   | MSDM     | AGMS      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Microbiology                                    | MBIO     | AGMB      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Natural Resources & Ecosystems                  | NRES     | AGNR      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Park Golf Management                            | PGM      | AGPG      | 1                             |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Plant Biology                                   | PBIO     | AGPB      |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Agricultural Sci (Ornith)                   | PAOS     | AGPO      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Plant Sci & Tech (Ornith)                   | PPST     | AGPT      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Turfgrass & Landscape Mgmt (Ornith)         | PTLM     | AGPT      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Veterinary Medicine                         | PVET     | AGVM      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Turfgrass and Landscape Mgmt                    | TGMT     | AGTM      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Veterinary Medicine                             | VMD      | AGVM      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Veterinary Science                              | VSDS     | AGVS      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Veterinary Technology                           | VTET     | AGVT      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
| Total  |   |          |           | 1                             | 1      | 1     | 1      | 1                      | 1      | 1               | 1      | 1                  | 1      | 1                | 1      | 1                 | 1      | 1       | 1      | 1     |
| College of Architecture                      | Architectural Studies                           | ARCH     | ARCH      |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Architecture                                    | ARCH     | ARCH      | 1                             |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Interior Design                                 | IDRS     | IDRS      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Landscape Architecture                          | LARC     | LARC      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Architecture                                | PAAC     | ARCH      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Pre-Interior Design                             | PIID     | ARCH      |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
| Total  |   |          |           | 1                             | 1      | 1     | 1      | 1                      | 1      | 1               | 1      | 1                  | 1      | 1                | 1      | 1                 | 1      | 1       | 1      | 1     |
| College of Arts and Sciences                 | Actuarial Science                               | AACT     | BA        |                               |        |       |        |                        |        |                 |        |                    |        |                  |        |                   |        |         |        | 1     |
|  | Anthropology                                    | ANTH     | BA        |                               |        |       |        |                        |        | 2               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Biology   | BIOL     | BA        |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Biochemistry                                    | BBIO     | BS        |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Biological Sciences                             | BIOS     | BA        |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Chemistry                                       | CHRM     | BS        |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |
|  | Chemical Engineering                            | CEME     | BA        |                               |        |       |        |                        |        | 1               | 1      |                    |        |                  |        |                   |        |         |        | 1     |

If you like what you see, click Preview and Export Extracted Data.

And here's where it all starts to go wrong.

You'll see at first it looks good – you get a reasonable representation of the table. But look at the first line.

## Preview of Extracted Tabular Data

|  |         |           |             |                      |             |                    |             |                    |                  |                   |
|--|---------|-----------|-------------|----------------------|-------------|--------------------|-------------|--------------------|------------------|-------------------|
|  |         |           |             | American             |             |                    |             |                    |                  |                   |
|  |         |           |             |                      |             |                    |             |                    |                  |                   |
|  |         |           |             | Indian/Alaska Native | Asian       | Black-Non Hispanic | Hispanic    | Non-resident Alien | Pacific Islander | Two or More Races |
| Colege Major Name                          | Major 1 | Degree Id | Male Female | Male Female          | Male Female | Male Female        | Male Female | Male Female        | Male Female      | Male Female       |
| Colege of Agri Sci and Agribusiness        | ABUS    | B1AB      |             |                      |             | 1                  | 2           | 1                  |                  |                   |
| Natl Resources Agricultural & Env Sci Comm | AESC    | B1SC      |             |                      |             |                    | 2           | 1                  |                  | 1                 |
| Agricultural Economics                     | AECN    | B1AE      |             |                      |             |                    | 1 2         | 3 2                |                  | 3 1               |
| Agricultural Education                     | AEDU    | B1ED      |             |                      |             |                    | 1           | 1                  |                  | 1 4               |

First, the misspelling of college is disturbing. Did a university document misspell it? No. Which means Tabula is reading two ls as one. That's ... not good.

Second, notice how the College of Agri Sci and Natl Resources, which was in it's own column before have been merged, somewhat inartfully, into the first column of major names. There is no major Colege of Agri Sci and Agribusiness. Same with Natl Resources Agricultural & Env Sci Comm. Those aren't things.

Note the empty column between Major1 and DegreeId.

Now scroll down some more.



|  |      |      |   |   |   |   |   |    |   |   |   |   |
|--|------|------|---|---|---|---|---|----|---|---|---|---|
| Colege of Architecture Architectural Studies | ARCS | BSD  |   | 1 | 3 | 1 |   | 12 | 3 | 8 | 5 |   |
| Architecture                                 | ARCH | MARC | 1 | 2 | 1 | 1 | 1 | 2  | 2 | 4 | 1 | 2 |
| Interior Design                              | IDES | BSD  |   |   |   |   | 1 |    |   |   | 1 |   |
| Landscape Architecture                       | LAND | BLA  |   |   |   |   |   | 2  |   | 3 | 1 | 1 |
| Pre-Architecture                             | PARC | NDEG |   | 1 | 1 | 1 | 1 | 5  | 3 | 3 |   | 1 |
| Pre-Interior Design                          | PINT | NDEG |   | 1 |   |   |   |    | 2 | 1 | 2 |   |
| Pre-Landscape Architecture                   | PLAR | NDEG |   |   |   | 1 |   |    |   |   |   |   |

|                    |                     |       |    |    |    |   |  |   |   |    |    |   |
|--------------------|---------------------|-------|----|----|----|---|--|---|---|----|----|---|
| Colege of Arts and | Actuarial Science   | AACTS | BA |    |    |   |  |   |   | 1  |    |   |
| Sciences           |                     |       | BS |    |    |   |  | 2 | 1 | 19 | 19 | 1 |
|                    | Anthropology        | ANTH  | BA |    |    |   |  | 2 | 4 | 1  |    |   |
|                    |                     |       | BS |    |    |   |  |   | 1 |    |    | 1 |
|                    | Biochemistry        | ABIOC | BS | 19 | 13 | 1 |  | 4 | 3 | 5  | 13 | 3 |
|                    | Biological Sciences | BIOS  | BA |    |    |   |  |   |   |    |    |   |

Notice Architecture has the same merging of college name and first major problems as the first one does, but note the blank column is missing.

Look at Arts and Sciences. Arts and Sciences are now in their own column, as the data shows, but there's now empty names that shouldn't be. What are those?

In short, it's a mess.

Here's the sad truth: THIS IS PRETTY GOOD. Open it in a spreadsheet and a little copying and pasting work while double checking the right names line up with the right rows and you're in business. As converted PDFs, this isn't bad.

It beats typing it out.

## 13.2 When it works well.

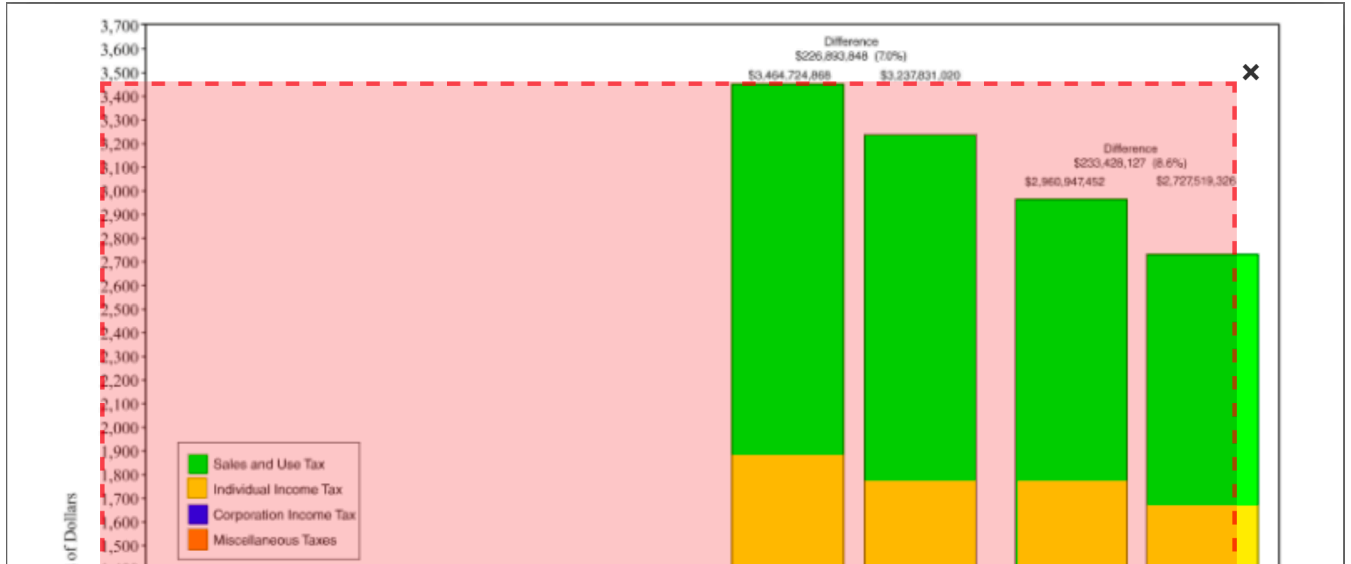
Each month, the Nebraska Department of Revenue releases the monthly tax receipts of the state, and forecasts into the future what tax receipts might be in the near future. They do this

for planning purposes – the Legislature needs to know how much money the state may have when the new budget is put into place so they know how much money they have to spend.

The announcement comes in a [press release](#). Each press release includes a table showing the current number, the predicted number, and difference. Of course it's a PDF.

Let's look at the most recent month as of this writing: [January 2020](#). Download it, open it in Tabula and hit Autodetect tables.

You'll note it finds no tables on the first page. Which is good, because there aren't any. Let's look at the third page. It finds a table, but is it one?



Let's hit the X in the top right on that one.

That leaves page 2. It finds two tables there. Let's just grab the first. Hit X on the second and click to preview the extracted data.

This looks good. So let's export it to a csv.

## 13.3 Cleaning up the data in R

The good news is that we have data we don't have to retype. The bad news is, it's hardly in importable shape.

Let's load libraries.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2

-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

To import this, we need one row of headers. We have three. And we need headers that make sense.

We can spell these out in the import step. First, we'll use `skip` to skip the first three lines. Then we'll spell out the column names by hand in a `col_names` bit. Here's how it looks.

```
receipts <- read_csv("data/tabula-General_Fund_Receipts_January_2020.csv", skip = 3, col_n
```

```
Rows: 7 Columns: 9
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (9): Month, TotalActualNetReceipts, TotalProjectedNetReceipts, Differenc...
```

```
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.
```

Now we have a harder part.

The columns come in as character columns. Why? Because the state puts commas and \$ and % in them, which R does not interpret as anything except text. So we need to get rid of them. We can mutate columns and use a function called `gsub` that finds a string and replaces it with something. So in our case, we're going to `gsub(",", "", fieldname)`. The unfortunate part is we have a lot of columns and a lot of fixes. So this is going to require a lot of code. It is repetitive, though, so we can copy and paste and adjust with most of it.

At the end, we need to use a function called `mutate_at` and convert the columns that aren't text into numbers.

And one last thing: If we do many months of this, we should note which report this comes from. We can do this with `mutate` as well.

Here's what that looks like:

```

receipts |> mutate(
  TotalActualNetReceipts = gsub(",", "", TotalActualNetReceipts),
  TotalActualNetReceipts = gsub("\\$", "", TotalActualNetReceipts),
  TotalProjectedNetReceipts = gsub(",", "", TotalProjectedNetReceipts),
  TotalProjectedNetReceipts = gsub("\\$", "", TotalProjectedNetReceipts),
  Difference = gsub(",", "", Difference),
  Difference = gsub("\\$", "", Difference),
  PercentDifference = gsub("%", "", PercentDifference),
  CumulativeActualNetReceipts = gsub(",", "", CumulativeActualNetReceipts),
  CumulativeActualNetReceipts = gsub("\\$", "", CumulativeActualNetReceipts),
  CumulativeProjectedNetReceipts = gsub(",", "", CumulativeProjectedNetReceipts),
  CumulativeProjectedNetReceipts = gsub("\\$", "", CumulativeProjectedNetReceipts),
  CumulativeDifference = gsub(",", "", CumulativeDifference),
  CumulativeDifference = gsub("\\$", "", CumulativeDifference),
  CumulativePercentDifference = gsub("%", "", CumulativePercentDifference)
) |> mutate_at(vars(-Month), as.numeric) |> mutate(ReportMonth = "January 2020")

```

# A tibble: 7 x 10

|   | Month<br><chr> | TotalActualNetReceipts<br><dbl> | TotalProjectedNetReceipts<br><dbl> | Difference<br><dbl> |
|---|----------------|---------------------------------|------------------------------------|---------------------|
| 1 | July           | 284883132                       | 271473079                          | 13410054            |
| 2 | August         | 462019974                       | 440504016                          | 21515958            |
| 3 | September      | 551908013                       | 510286143                          | 41621870            |
| 4 | October        | 289723434                       | 266204529                          | 23518905            |
| 5 | November       | 431787603                       | 404934524                          | 26853079            |
| 6 | December       | 472926836                       | 421455999                          | 51470837            |
| 7 | January        | 467698460                       | 412661036                          | 55037424            |

```

# i 6 more variables: PercentDifference <dbl>,
#   CumulativeActualNetReceipts <dbl>, CumulativeProjectedNetReceipts <dbl>,
#   CumulativeDifference <dbl>, CumulativePercentDifference <dbl>,
#   ReportMonth <chr>

```

We can now reuse this with other months after we harvest the data out of it.

## 14 Combining and joining

Often, as data journalists, we're looking at data across time or at data stored in multiple tables. And to do that, we need to often need to merge that data together.

Depending on what we have, we may just need to stack data on top of each other to make new data. If we have 2019 data and 2018 data and we want that to be one file, we stack them. If we have a dataset of cows in counties and a dataset of populations in county, we're going to join those two together on the county – the common element.

Let's explore.

### 14.1 Combining data

In the last assignment, we harvested data out of PDFs. Let's reuse what we did there and merge three months of reports from the Department of Revenue together. For mine, I have [January 2020](#), [December 2019](#), and [November 2019](#).

Let's do what we need to import them properly. Unlike the last example, I've merged it all into one step for each of the three datasets.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```

receiptsJan20 <- read_csv("data/tabula-General_Fund_Receipts_January_2020.csv", skip = 3,
  TotalActualNetReceipts = gsub(",", "", TotalActualNetReceipts),
  TotalActualNetReceipts = gsub("\\$", "", TotalActualNetReceipts),
  TotalProjectedNetReceipts = gsub(",", "", TotalProjectedNetReceipts),
  TotalProjectedNetReceipts = gsub("\\$", "", TotalProjectedNetReceipts),
  Difference = gsub(",", "", Difference),
  Difference = gsub("\\$", "", Difference),
  PercentDifference = gsub("%", "", PercentDifference),
  CumulativeActualNetReceipts = gsub(",", "", CumulativeActualNetReceipts),
  CumulativeActualNetReceipts = gsub("\\$", "", CumulativeActualNetReceipts),
  CumulativeProjectedNetReceipts = gsub(",", "", CumulativeProjectedNetReceipts),
  CumulativeProjectedNetReceipts = gsub("\\$", "", CumulativeProjectedNetReceipts),
  CumulativeDifference = gsub(",", "", CumulativeDifference),
  CumulativeDifference = gsub("\\$", "", CumulativeDifference),
  CumulativePercentDifference = gsub("%", "", CumulativePercentDifference)
) |> mutate_at(vars(-Month), as.numeric) |> mutate(ReportMonth = "January 2020")

```

Rows: 7 Columns: 9

```

-- Column specification -----
Delimiter: ","
chr (9): Month, TotalActualNetReceipts, TotalProjectedNetReceipts, Differenc...

```

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.

```

receiptsDec19 <- read_csv("data/tabula-General_Fund_Receipts_December_2019.csv", skip = 3,
  TotalActualNetReceipts = gsub(",", "", TotalActualNetReceipts),
  TotalActualNetReceipts = gsub("\\$", "", TotalActualNetReceipts),
  TotalProjectedNetReceipts = gsub(",", "", TotalProjectedNetReceipts),
  TotalProjectedNetReceipts = gsub("\\$", "", TotalProjectedNetReceipts),
  Difference = gsub(",", "", Difference),
  Difference = gsub("\\$", "", Difference),
  PercentDifference = gsub("%", "", PercentDifference),
  CumulativeActualNetReceipts = gsub(",", "", CumulativeActualNetReceipts),
  CumulativeActualNetReceipts = gsub("\\$", "", CumulativeActualNetReceipts),
  CumulativeProjectedNetReceipts = gsub(",", "", CumulativeProjectedNetReceipts),
  CumulativeProjectedNetReceipts = gsub("\\$", "", CumulativeProjectedNetReceipts),
  CumulativeDifference = gsub(",", "", CumulativeDifference),
  CumulativeDifference = gsub("\\$", "", CumulativeDifference),
  CumulativePercentDifference = gsub("%", "", CumulativePercentDifference)
) |> mutate_at(vars(-Month), as.numeric) |> mutate(ReportMonth = "December 2019")

```

Rows: 6 Columns: 9

-- Column specification -----

Delimiter: ","

chr (9): Month, TotalActualNetReceipts, TotalProjectedNetReceipts, Differenc...

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.

```
receiptsNov19 <- read_csv("data/tabula-General_Fund_Receipts_November_12-13-2019.csv", skip = 1)
TotalActualNetReceipts = gsub(",", "", TotalActualNetReceipts),
TotalActualNetReceipts = gsub("\\$", "", TotalActualNetReceipts),
TotalProjectedNetReceipts = gsub(",", "", TotalProjectedNetReceipts),
TotalProjectedNetReceipts = gsub("\\$", "", TotalProjectedNetReceipts),
Difference = gsub(",", "", Difference),
Difference = gsub("\\$", "", Difference),
PercentDifference = gsub("%", "", PercentDifference),
CumulativeActualNetReceipts = gsub(",", "", CumulativeActualNetReceipts),
CumulativeActualNetReceipts = gsub("\\$", "", CumulativeActualNetReceipts),
CumulativeProjectedNetReceipts = gsub(",", "", CumulativeProjectedNetReceipts),
CumulativeProjectedNetReceipts = gsub("\\$", "", CumulativeProjectedNetReceipts),
CumulativeDifference = gsub(",", "", CumulativeDifference),
CumulativeDifference = gsub("\\$", "", CumulativeDifference),
CumulativePercentDifference = gsub("%", "", CumulativePercentDifference)
) |> mutate_at(vars(-Month), as.numeric) |> mutate(ReportMonth = "November 2019")
```

Rows: 5 Columns: 9

-- Column specification -----

Delimiter: ","

chr (9): Month, TotalActualNetReceipts, TotalProjectedNetReceipts, Differenc...

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.

All three of these datasets have the same number of columns, all with the same names, so if we want to merge them together to compare them over time, we need to stack them together. The verb here, in R, is `rbind`. The good news about `rbind` is that it is very simple. The bad news – you can only merge two things at a time.

Since we have three things, we're going to do this in steps. First, we'll create a dataframe that will hold it all and we'll populate it with two of our revenue dataframes `rbinded` together. Then, we'll overwrite our new dataframe with the results of that dataframe with the third revenue dataframe.

```

predictions1 <- rbind(receiptsJan20, receiptsDec19)

predictions2 <- rbind(predictions1, receiptsNov19)

predictions2

# A tibble: 18 x 10
  Month      TotalActualNetReceipts TotalProjectedNetReceipts Difference
<chr>          <dbl>          <dbl>          <dbl>
1 July            284883132            271473079    13410054
2 August           462019974            440504016    21515958
3 September        551908013            510286143    41621870
4 October          289723434            266204529    23518905
5 November         431787603            404934524    26853079
6 December         472926836            421455999    51470837
7 January          467698460            412661036    55037424
8 July            284883132            271473079    13410054
9 August           462019974            440504016    21515958
10 September       551908013            510286143    41621870
11 October         289723434            266204529    23518905
12 November        431787603            404934524    26853079
13 December        472926836            421455999    51470837
14 July            284883132            271473079    13410054
15 August           462019974            440504016    21515958
16 September       551908013            510286143    41621870
17 October         289723434            266204529    23518905
18 November        431787603            404934524    26853079
# i 6 more variables: PercentDifference <dbl>,
#   CumulativeActualNetReceipts <dbl>, CumulativeProjectedNetReceipts <dbl>,
#   CumulativeDifference <dbl>, CumulativePercentDifference <dbl>,
#   ReportMonth <chr>

```

And boom, like that, we have 18 rows of data instead of three dataframes of 5, 6, and 7 respectively.

## 14.2 Joining data

More difficult is when you have two separate tables that are connected by a common element or elements.



Let's return to our fatal accident data. In reality, the Fatality Analysis Reporting System data has 27 tables in it – everything from details about the damage to the paperwork done.

Let's just merge two of them and just for the state of Nebraska – download [the accidents](#) and [the people](#).

Often, when talking about relational data files like this, there's substantial amounts of documentation that go with it to tell you how these things are related and what codes mean. [The FARS data is no different](#). You should open it and click on the PERSON Data File.

ST\_CASE should be used to merge the Person data file with the Accident data file for a set of all motorists and non-motorists.

So that's what we're going to do.

```
accidents <- read_csv("data/neaccidents.csv")
```

```
Rows: 201 Columns: 52
-- Column specification -----
Delimiter: ","
chr (3): TWAY_ID, TWAY_ID2, RAIL
dbl (49): STATE, ST_CASE, VE_TOTAL, VE_FORMS, PVH_INVL, PEDS, PERNOTMVIT, PE...

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.
```

```
persons <- read_csv("data/nepersons.csv")
```

```
Rows: 553 Columns: 62
-- Column specification -----
Delimiter: ","
dbl (62): STATE, ST_CASE, VE_FORMS, VEH_NO, PER_NO, STR_VEH, COUNTY, DAY, MO...

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.
```

First, notice something in the environment about your dataset: there are 201 accidents but 553 persons. That means there's not quite 3 people involved in every accident on average between drivers and passengers. Some are single car, single person crashes. Some involve a lot of people.

To put these two tables together, we need to use something called a join. There are different kinds of join. It's better if you think of two tables sitting next to each other. A `left_join` takes all the records from the left table and only the records that match in the right one. A `right_join` does the same thing. An `inner_join` takes only the records where they are equal. There's one other join – a `full_join` which returns all rows of both, regardless of if there's a match – but I've never once had a use for a full join.

In the PERSON Data File documentation, we see that column that connects these two tables together is the `ST_CASE` column.

So we can do this join multiple ways and get a similar result. We can put the person file on the left and the accident on the right and use a left join to get them all together. And we use `by=` to join by the right column. And to avoid rendering hundreds of rows of data, I'm going to count the rows at the end. The reason I'm going this is important: **Rule 1 in joining data is having an idea of what you are expecting to get.** So with a left join with people on the left, I have 553 people, so I expect to get 553 rows when I'm done.

```
persons |> left_join(accidents, by="ST_CASE") |> nrow()
```

```
[1] 553
```

Remove the `nrow` and run it again for yourself. See how there are several columns that end with `.X`? That means they're duplicates. There's a solid chance they are the same in both tables. By default, `dplyr` will do a "natural" join, where it'll match all the matching columns in both tables. So if we take out the `by`, it'll use all the common columns between the tables. That may not be right – our documentation says `ST_CASE` is how they are related – but let's try it. If it works, we should get 553 rows.

```
persons |> left_join(accidents)
```

Joining with ``by = join_by(STATE, ST_CASE, VE_FORMS, COUNTY, DAY, MONTH, HOUR, MINUTE, RUR_URB, FUNC_SYS, HARM_EV, MAN_COLL, SCH_BUS)``

```
# A tibble: 553 x 101
```

|   | STATE | ST_CASE | VE_FORMS | VEH_NO | PER_NO | STR_VEH | COUNTY | DAY   | MONTH | HOUR  | MINUTE |
|---|-------|---------|----------|--------|--------|---------|--------|-------|-------|-------|--------|
|   | <dbl> | <dbl>   | <dbl>    | <dbl>  | <dbl>  | <dbl>   | <dbl>  | <dbl> | <dbl> | <dbl> | <dbl>  |
| 1 | 31    | 310001  | 2        | 1      | 1      | 0       | 55     | 1     | 1     | 0     | 20     |
| 2 | 31    | 310001  | 2        | 2      | 1      | 0       | 55     | 1     | 1     | 0     | 20     |
| 3 | 31    | 310002  | 1        | 1      | 1      | 0       | 157    | 7     | 1     | 22    | 0      |
| 4 | 31    | 310003  | 1        | 0      | 1      | 1       | 55     | 10    | 1     | 6     | 57     |
| 5 | 31    | 310003  | 1        | 1      | 1      | 0       | 55     | 10    | 1     | 6     | 57     |

```

6    31  310004      2    1    1      0    79    10    1    16    55
7    31  310004      2    2    1      0    79    10    1    16    55
8    31  310005      2    1    1      0   119    10    1     4    30
9    31  310005      2    2    1      0   119    10    1     4    30
10   31  310006      1    0    1      1   109    11    1     6    25
# i 543 more rows
# i 90 more variables: RUR_URB <dbl>, FUNC_SYS <dbl>, HARM_EV <dbl>,
#   MAN_COLL <dbl>, SCH_BUS <dbl>, MAKE <dbl>, MAK_MOD <dbl>, BODY_TYP <dbl>,
#   MOD_YEAR <dbl>, TOW_VEH <dbl>, SPEC_USE <dbl>, EMER_USE <dbl>,
#   ROLLOVER <dbl>, IMPACT1 <dbl>, FIRE_EXP <dbl>, AGE <dbl>, SEX <dbl>,
#   PER_TYP <dbl>, INJ_SEV <dbl>, SEAT_POS <dbl>, REST_USE <dbl>,
#   REST_MIS <dbl>, AIR_BAG <dbl>, EJECTION <dbl>, EJ_PATH <dbl>, ...

```

So instead of just one column, it used 13. And we got the same answer. And we don't have any columns with .X after it anymore. So we're good to move forward.

Let's save our joined data to a new dataframe.

```
personaccidents <- persons |> left_join(accidents)
```

Joining with `by = join\_by(STATE, ST\_CASE, VE\_FORMS, COUNTY, DAY, MONTH, HOUR, MINUTE, RUR\_URB, FUNC\_SYS, HARM\_EV, MAN\_COLL, SCH\_BUS)`

Now, with our joined data, we can answer questions that come from both datasets. So what if we looked at median age of drivers who died broken down by what kind of roadway the accident happened on? We can do this now because the accident data has the roadway information and the age and who was driving and what type of injury they sustained comes from the person table.

We get this by using filters followed by a group by and summarize. In the data documentation linked above, look in the PERSON Data File to get the appropriate filters. In this case, we want PER\_TYPE of 1 (the driver) and an INJ\_SEV of 4, which means death. In the ACCIDENT Data File section, we learn it's the ROUTE we want to group by.

```

personaccidents |>
  filter(PER_TYP == 1) |>
  filter(INJ_SEV == 4) |>
  group_by(ROUTE) |>
  summarize(
    count = n(),
    avgage = mean(AGE),
    medage = median(AGE))

```

```
# A tibble: 5 x 4
  ROUTE count avgage medage
  <dbl> <int> <dbl> <dbl>
1     1    15  40.5    33
2     2    51  53.1    51
3     3    31  42.3    42
4     4    37  37.3    36
5     6    18  40.4    35
```

According to our query, 15 accidents happened on interstates, and the median age of those was the lowest of all at 33. The most accidents were on US Highways, which makes sense because there's a lot more lane miles of US Highways than Interstates in Nebraska and pretty much every other state. But the second most common is county roads. And the median age of drivers there was quite low at 36.

Let's break this down one more step. What if we added RUR\_URB – if the accident happened in rural or urban place. A common feeling in a rural state like Nebraska is that urban interstate is a ribbon of insanity. But is it?

```
personaccidents |>
  filter(PER_TYP == 1) |>
  filter(INJ_SEV == 4) |>
  group_by(RUR_URB, ROUTE) |>
  summarize(
    count = n(),
    avgage = mean(AGE),
    medage = median(AGE))
```

`summarise()` has grouped output by 'RUR\_URB'. You can override using the `groups` argument.

```
# A tibble: 8 x 5
# Groups:   RUR_URB [2]
  RUR_URB ROUTE count avgage medage
  <dbl> <dbl> <int> <dbl> <dbl>
1     1     1     12  45.3  42.5
2     1     2     41  52.5  51
3     1     3     27  41.9  42
4     1     4     37  37.3  36
5     2     1      3  21.3  21
6     2     2     10  55.3  54.5
7     2     3      4   45   35
```

8            2            6            18            40.4            35

In 2018, only 3 of the 15 deaths on interestates were in urban areas. All the rest were in rural areas. And all 37 drivers who died in accidents on county roads were in rural areas. Most of the driver deaths on US Highways were in rural places as well.

## 15 Scraping data with Rvest

Sometimes, governments put data online on a page or in a searchable database. And when you ask them for a copy of the data underneath the website, they say no. Why? Because they have a website. That's it. That's their reason. We don't have to give you the data because we've already given you the data, never mind that they haven't. One of the most powerful tools you can learn as a data journalist is how to scrape data. Scraping is the process of programming a computer to act like a browser, go to a website, inject the HTML from that website and turn it into data.

The degree of difficulty here goes from Easy to So Hard You Want To Throw Your Laptop Out A Window. And the curve between the two steep. You can learn how to scrape Easy in a day. The hard ones take months to years of programming experience.

So.

Let's try an easy one.

We're going to use a library called `rvest`, which you can install it the same way we've done all installs: go to the console and `install.packages("rvest")`.

We'll load them first:

```
library(rvest)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter()          masks stats::filter()
x readr::guess_encoding() masks rvest::guess_encoding()
x dplyr::lag()             masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

The first thing we need to do is define a URL. What URL are we going to scrape? This is where paying attention to URLs pays off. Some search urls are addressable – meaning you can copy the url of your search and go to it again and again. Or is the search term invisible?

Let's take an example from the Nebraska Legislature. The Legislature publishes a daily agenda that tells you what bills will be debated on the floor that day. Here's [an example from Feb. 3](#). Go that page. You'll see multiple sections – a reports section, and the General File section. General File is the first stage of floor debate. Those are the bills to be debated.

Let's grab them with rvest.

First, we create a url variable and set it equal to that url.

```
url <- "https://nebraskalegislature.gov/calendar/agenda.php?day=2020-02-03"
```

Now we're going to do a handful of things at once. We're going to take that url, pass it to a `read_html` command, which does what you think it does. We're then going to search that HTML for a specific node, the node that contains our data.

The most difficult part of scraping data from any website is knowing what exact HTML tag you need to grab. In this case, we want a `<table>` tag that has all of our data table in it. But how do you tell R which one that is? Well, it's easy, once you know what to do. But it's not simple. So I've made a short video to show you how to find it.

Using that same trick on the Legislature page, we find the table with those general file bills and it's in something called `agenda_table_4464`. With that, we can turn it into a table.

```
agenda <- url |>
  read_html() |>
  html_nodes(xpath = '//*[@id="agenda_table_4464"]') |>
  html_table()
```

After doing that, looking at the environment for your agenda. And you'll see ... not a dataframe. You'll see a list. With one thing in it. That one thing? A dataframe. So we need to grab that first element. We get it by doing this:

```
agenda <- agenda[[1]]
```

And now we can take a look:

```
agenda
```

```
# A tibble: 10 x 3
  Document
```

```
Introducer Description
```

|    | <chr>                           | <chr>       | <chr>       |
|----|---------------------------------|-------------|-------------|
| 1  | "Currently or Pendingon Floor\n | ~ Bolz      | Provide a ~ |
| 2  | "Currently or Pendingon Floor\n | ~ Kolterman | Adopt the ~ |
| 3  | "Currently or Pendingon Floor\n | ~ Kolterman | Appropriat~ |
| 4  | "Currently or Pendingon Floor\n | ~ Bolz      | Change eli~ |
| 5  | "Currently or Pendingon Floor\n | ~ Kolterman | Change pro~ |
| 6  | "Currently or Pendingon Floor\n | ~ Kolterman | Appropriat~ |
| 7  | "Currently or Pendingon Floor\n | ~ Dorn      | Change pro~ |
| 8  | "Currently or Pendingon Floor\n | ~ Wishart   | Change pro~ |
| 9  | "Currently or Pendingon Floor\n | ~ McDonnell | Change pro~ |
| 10 | "Currently or Pendingon Floor\n | ~ Vargas    | Change pro~ |

And as you can see, it's ... not perfect. But we can fix that with a little gsub magic.

```
agenda |> mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document))
```

```
# A tibble: 10 x 3
```

|    | Document<br><chr> | Introducer<br><chr> | Description<br><chr> |
|----|-------------------|---------------------|----------------------|
| 1  | "                 | ~ Bolz              | Provide a ~          |
| 2  | "                 | ~ Kolterman         | Adopt the ~          |
| 3  | "                 | ~ Kolterman         | Appropriat~          |
| 4  | "                 | ~ Bolz              | Change eli~          |
| 5  | "                 | ~ Kolterman         | Change pro~          |
| 6  | "                 | ~ Kolterman         | Appropriat~          |
| 7  | "                 | ~ Dorn              | Change pro~          |
| 8  | "                 | ~ Wishart           | Change pro~          |
| 9  | "                 | ~ McDonnell         | Change pro~          |
| 10 | "                 | ~ Vargas            | Change pro~          |

## 15.1 A more difficult example

The Nebraska Legislature, with its unique unicameral or one house structure, does some things a little differently than other legislatures. Example: Bills have to go through three rounds of debate before getting passed. There's General File (round 1), Select File (round 2), and Final Reading (round 3).

So what does a day where they do more than general file look like? [Like this](#).

How do we scrape that?



```
harderurl <- "https://nebraskalegislature.gov/calendar/agenda.php?day=2020-02-21"
```

```
harderagenda <- harderurl |>
  read_html() |>
  html_nodes("table") |>
  html_table()
```

You can see that `harderagenda` has a list of four instead of a list of one. We can see each item in the list has a dataframe. We can see them individually. Here's the first:

```
harderagenda[[1]]
```

```
# A tibble: 4 x 3
  Document                               Introducer Description
  <chr>                                <chr>      <chr>
1 "Currently or Pendingon Floor\n"    ~ Kolterman Define the~
2 "Currently or Pendingon Floor\n"    ~ Geist      Change and~
3 "Currently or Pendingon Floor\n"    ~ Chambers  Change pro~
4 "Currently or Pendingon Floor\n"    ~ Gragert    Provide fo~
```

Here's the second:

```
harderagenda[[2]]
```

```
# A tibble: 4 x 3
  Document                               Introducer Description
  <chr>                                <chr>      <chr>
1 "Currently or Pendingon Floor\n"    ~ Crawford  Change the~
2 "Currently or Pendingon Floor\n"    ~ Lindstrom  Change pro~
3 "Currently or Pendingon Floor\n"    ~ Quick      Change the~
4 "Currently or Pendingon Floor\n"    ~ Hunt       Adopt the ~
```

So we can merge those together no problem, but how would we know what stage each bill is at?

Look at the page – we can see that the bills are separated by a big headline like “SELECT FILE: 2020 PRIORITY BILLS”. To separate these, we need to grab those and then add them to each bill using `mutate`.

Here's how we grab them:

```
labels <- harderurl |>
  read_html() |>
  html_nodes(".card-header") |>
  html_text()
```

Another list. If you look at the first, it's at the top of the page with no bills. Here's the second:

```
labels[[2]]
```

```
[1] "\n                                SELECT FILE:  2020 PRIORITY BILLS\n"
```

So we know can see there's some garbage in there we want to clean out. We can use a new library called `stringr` to trim the excess spaces and `gsub` to strip the newline character: `\n`.

```
harderagenda[[1]] |>
  mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document)) |>
  mutate(Stage = labels[[2]]) |>
  mutate(Stage = gsub("\n", "", Stage)) |>
  mutate(Stage = str_trim(Stage, side = "both"))
```

# A tibble: 4 x 4

|   | Document | Introducer  | Description | Stage |
|---|----------|-------------|-------------|-------|
|   | <chr>    | <chr>       | <chr>       | <chr> |
| 1 | "        | ~ Kolterman | Define the~ | SELE~ |
| 2 | "        | ~ Geist     | Change and~ | SELE~ |
| 3 | "        | ~ Chambers  | Change pro~ | SELE~ |
| 4 | "        | ~ Gragert   | Provide fo~ | SELE~ |

Now it's just a matter grinding through the items in the list.

NOTE: This is grossly inefficient and very manual. And, we'd have to change this for every day we want to scrape. As such, this is not the "right" way to do this. We'll cover that in the next chapter.

```
harderagenda1 <- harderagenda[[1]] |>
  mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document)) |>
  mutate(Stage = labels[[2]]) |> mutate(Stage = gsub("\n", "", Stage)) |>
  mutate(Stage = str_trim(Stage, side = "both"))
```

```
harderagenda2 <- harderagenda[[2]] |>
  mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document)) |>
  mutate(Stage = labels[[3]]) |> mutate(Stage = gsub("\n", "", Stage)) |>
  mutate(Stage = str_trim(Stage, side = "both"))
```

```
harderagenda3 <- harderagenda[[3]] |>
  mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document)) |>
  mutate(Stage = labels[[4]]) |> mutate(Stage = gsub("\n", "", Stage)) |>
  mutate(Stage = str_trim(Stage, side = "both"))
```

```
harderagenda4 <- harderagenda[[4]] |>
  mutate(Document = gsub("Currently or Pendingon Floor\n ", "", Document)) |>
  mutate(Stage = labels[[5]]) |> mutate(Stage = gsub("\n", "", Stage)) |>
  mutate(Stage = str_trim(Stage, side = "both"))
```

Now we merge:

```
largeragenda <- rbind(harderagenda1, harderagenda2)
largeragenda <- rbind(largeragenda, harderagenda3)
largeragenda <- rbind(largeragenda, harderagenda4)
```

And now we have a dataset of all bills and what stage they're at for that day.

```
largeragenda
```

```
# A tibble: 10 x 4
```

|    | Document<br><chr> | Introducer<br><chr> | Description<br><chr> | Stage<br><chr> |
|----|-------------------|---------------------|----------------------|----------------|
| 1  | "                 | ~ Kolterman         | Define the~          | "SEL~          |
| 2  | "                 | ~ Geist             | Change and~          | "SEL~          |
| 3  | "                 | ~ Chambers          | Change pro~          | "SEL~          |
| 4  | "                 | ~ Gragert           | Provide fo~          | "SEL~          |
| 5  | "                 | ~ Crawford          | Change the~          | "GEN~          |
| 6  | "                 | ~ Lindstrom         | Change pro~          | "GEN~          |
| 7  | "                 | ~ Quick             | Change the~          | "GEN~          |
| 8  | "                 | ~ Hunt              | Adopt the ~          | "GEN~          |
| 9  | "                 | ~ Agricultu~        | Adopt the ~          | "GEN~          |
| 10 | "                 | ~ Howard            | Congratula~          | "LEG~          |

## 16 Advanced rvest

Continuing a discussion from the last chapter, this is an example of when it goes from Easy to Moderately Difficult.

In the last exercise, we scraped bills out of one day of floor activity in the Nebraska Legislature. What if we wanted all of them? Let's say we wanted to keep a running scoreboard of who has introduced the most bills that have seen the floor?

Here's how to use some programming knowledge with R to grab all days and merge them together. First we start with libraries, as we always do.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

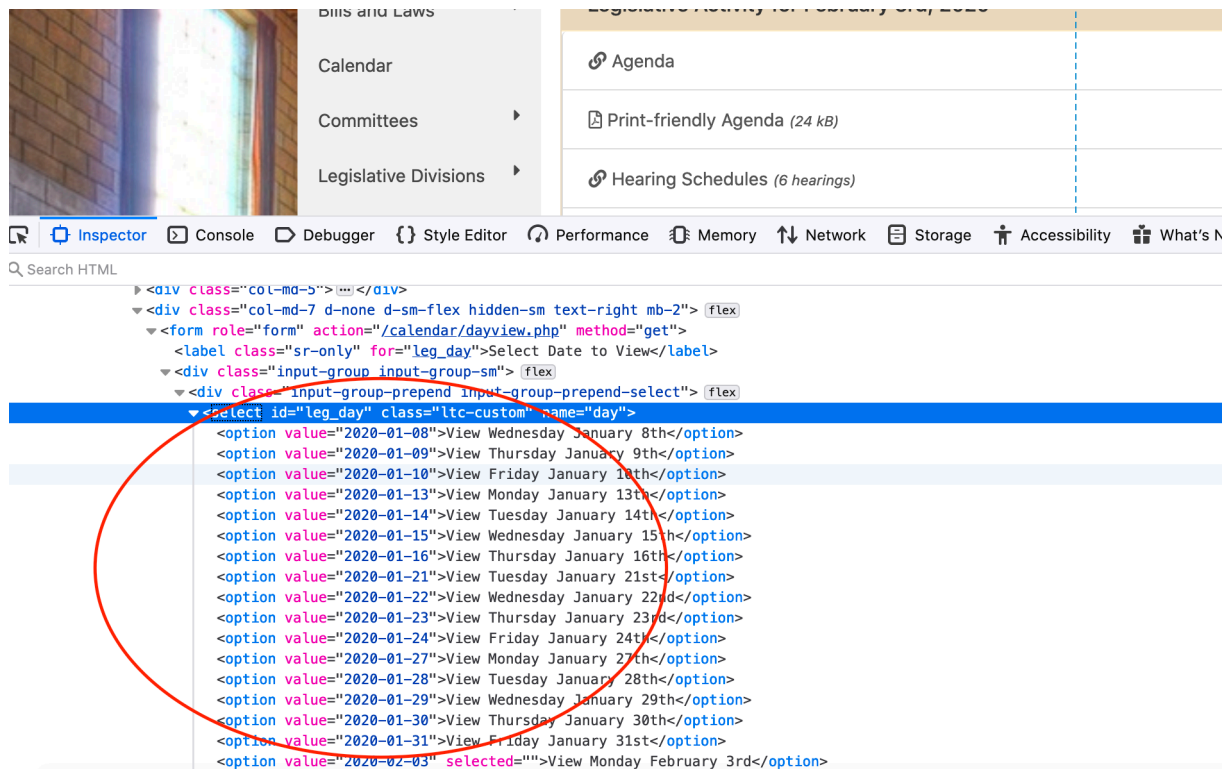
```
library(rvest)
```

Attaching package: 'rvest'

The following object is masked from 'package:readr':

```
guess_encoding
```

Now we'll start the trickery. So we need to know what days the legislature is in session. Fortunately, the legislature tells us that in a drop down menu. Inspect [the dropdown menu of this page](#). See the list?



If it's HTML in the page, we can grab it, so let's do that. We start with the URL to any day, really.

```
url <- "https://nebraskalegislature.gov/calendar/dayview.php?day=2020-02-03"
```

I'm going to create a thing called a calendar and store the list of option values in there. So when I'm done, I'll have a list of dates.

```
calendar <- url |>
  read_html() |>
  html_nodes(xpath = '//*[@id="leg_day"]') |>
  html_nodes("option") |>
  html_attrs()
```

```
calendar[1][1]
```

```
[[1]]  
NULL
```

Now this part gets tough, but if you follow, it's logical. It's step by step, really.

First, I noticed at the top of the agenda pages is a link to the daily agenda in csv format. Convenient, that.



If you look at that url, it looks like this:

```
https://nebraskalegislature.gov/calendar/agenda.php?day=2020-01-14&print=csv
```

If we can change the date in that url to each day of the session, we'd get a csv file for each day of the session that we can merge together.

Let's break down the steps to do that.

1. I need a place to store this. A lazy way? Just import a day, filter everything out of it, and I've got an empty table I can `rbind` stuff into.
2. I need to set up a loop. For each day in the calendar, do some stuff.
3. I need to clean out the word selected (an HTML thing) from one of the dates.
4. I need to just scrape days that have business pending on the floor. So I can skip the first few days and I can skip any days into the future.
5. I need to take my date from calendar and add them to the url. Then I need to take that agenda url and add the csv parts.
6. Then I can use `read_csv` like we have been to read the CSV url directly.
7. Then I just `rbind` my new csv to the one I created.
8. This isn't going to work perfectly every time, so I'm going to add some `tryCatch` statements that say try this, and if it doesn't work, do nothing.

9. Then, because I'm a decent fellow, I'm going to pause my query for three seconds to give their servers a break.

Let's run it.

```
allbills <- read_csv("https://nebraskalegislature.gov/calendar/agenda.php?day=2020-01-14&p=1")

for (i in calendar){
  date <- i[1]
  checkdate <- as.Date(date)

  if (checkdate <= Sys.Date() & checkdate > as.Date("2020-01-13")) {

    agendaurl <- paste("https://nebraskalegislature.gov/calendar/agenda.php?day=", i, sep="")
    csvurl <- paste(agendaurl, "&print=csv", sep="")
    tryCatch(
      agenda <- read_csv(csvurl, col_types = cols()) |> mutate(Date = checkdate),
      error = function(e){NA})
    tryCatch(
      allbills <- rbind(allbills, agenda),
      error = function(e){NA})
    Sys.sleep(3)
  }
}
```

And after that, I have a dataset of entries of bills on the floor. So if I want to see who has had the most bills on the floor – including repeats – I could answer that now.

```
allbills |> group_by(Introducer) |> tally(sort=TRUE) |> na.omit() |> top_n(5)
```

Selecting by n

```
# A tibble: 0 x 2
# i 2 variables: Introducer <chr>, n <int>
```

Senator Kolterman, collect your prize.

A note about advanced scraping – every site is different. Every time you want to scrape a site, you'll be puzzling over different problems. But the steps remain the same: find a pattern, exploit it, clean the data on the fly and put it into a place to store it.

## 17 Visualizing your data for reporting

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 it's own. And not every story needs a feat of programming and art. Sometimes, you can help yourself and your story by just creating a quick chart.

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.

Let's revisit some data we've used in the past and turn it into charts.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.2      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.0
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.2      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

The dataset we'll use is [the mountainlion data](#) we looked at in Chapter 6.

```
mountainlions <- read_csv("data/mountainlions.csv")
```

```
Rows: 393 Columns: 4
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (3): Cofirm Type, COUNTY, Date
```

```
dbl (1): ID
```



- 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.

## 17.1 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, compared to the next, 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.

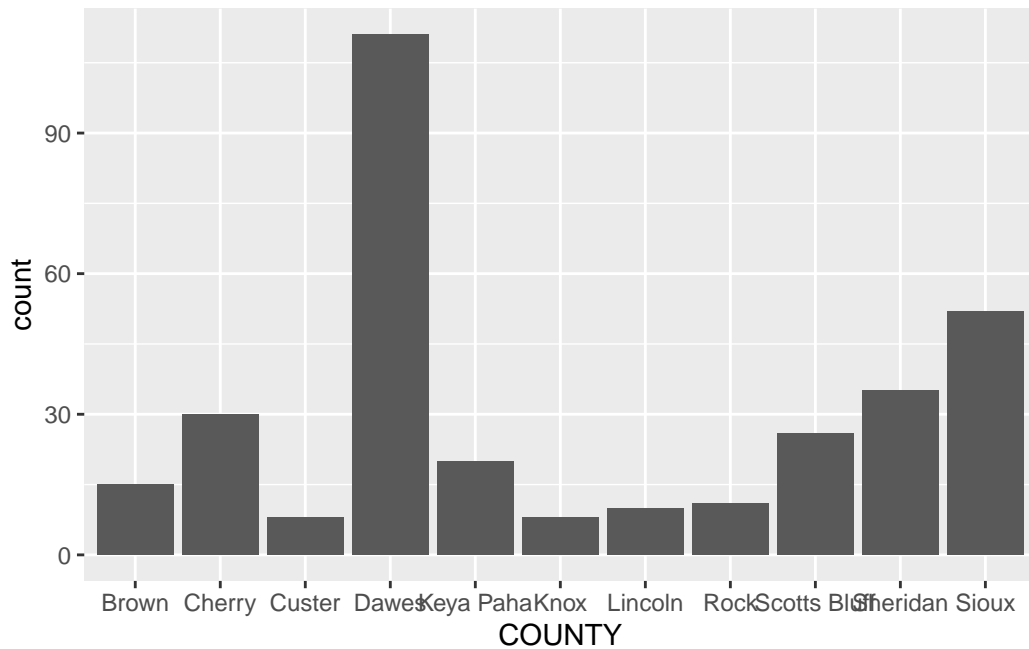
So what does the chart of the top 10 counties with the most mountain sightings look like?

First, we'll create a dataframe of those top 10, called `topsightings`.

```
mountainlions |>
  group_by(COUNTY) |>
  summarise(
    total = n()
  ) |> top_n(10, total) -> topsightings
```

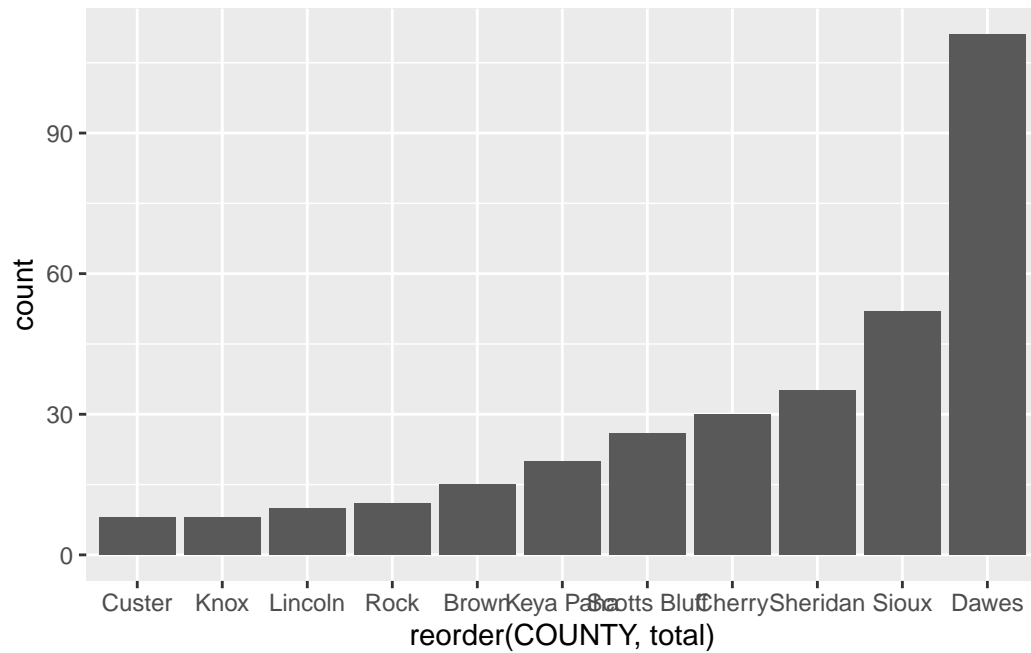
Now `ggplot`. The first thing we do with `ggplot` is invoke it, which creates the canvas. In `ggplot`, we work with geometries – the shape that the data will take – and aesthetics – the data that will take shape. In a bar chart, we first pass in the data to the geometry, then set the aesthetic. We tell `ggplot` what the x value is – in a bar chart, that's almost always your grouping variable. Then we tell it the weight of the bar – the number that will set the height.

```
ggplot() + geom_bar(data=topsightings, aes(x=COUNTY, weight=total))
```



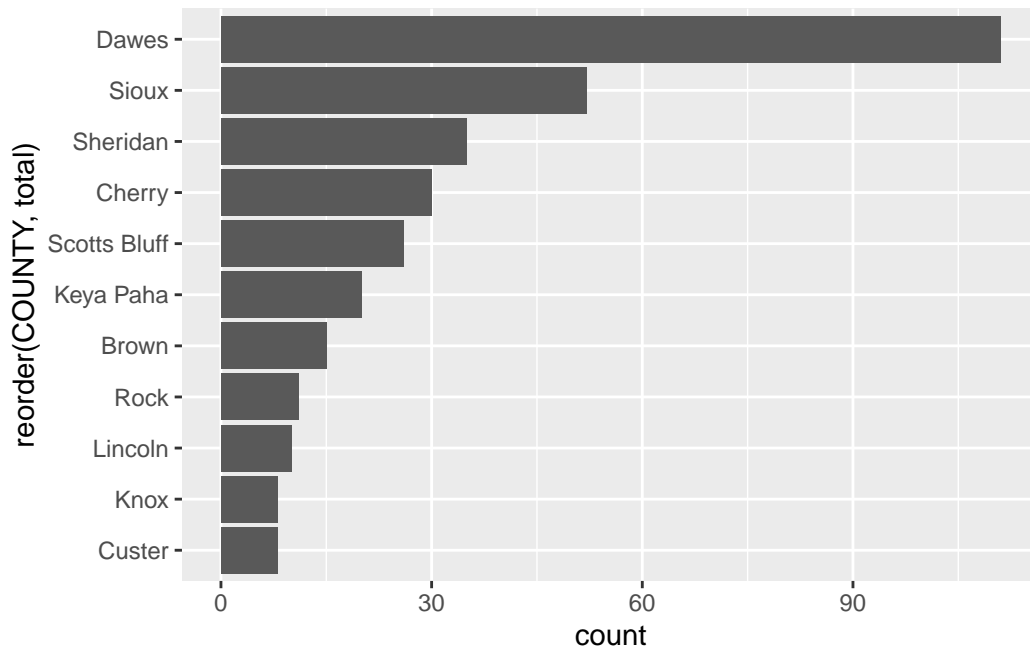
The bars look good, but the order makes no sense. In ggplot, we use `reorder`, and we reorder the x value based on the weight, like this:

```
ggplot() + geom_bar(data=topsightings, aes(x=reorder(COUNTY, total), weight=total))
```



Better, but it looks ... not great on the bottom. We can fix that by flipping the coordinates.

```
ggplot() + geom_bar(data=topsightings, aes(x=reorder(COUNTY, total), weight=total)) + coord_flip()
```



Art? No. Tells you the story? Yep. And for reporting purposes, that's enough.

## 17.2 Line charts

Line charts show change over time. It works the much the same as a bar chart, code wise, but instead of a weight, it uses a y. And if you have more than one group in your data, it takes a group element.

The secret to knowing if you have a line chart is if you have a date. The secret to making a line chart is your x value is almost always a date.

To make this easier, I've created a new version of the [county population estimates data that it formatted for charting](#). Let's import it:

```
populationestimates <- read_csv("data/countypopulationslong.csv")
```

```
Rows: 28278 Columns: 6
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr  (4): STNAME, CTYNAME, Year, Name
```

```
dbl  (1): Population
```

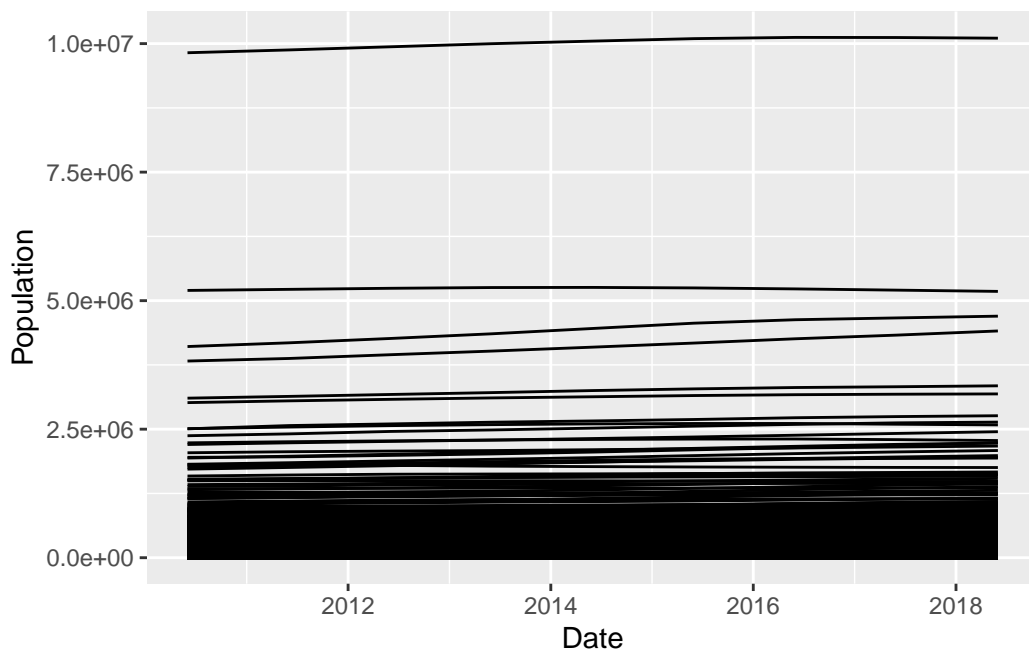
```
date (1): Date
```

```
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.
```

As you can see, I've switched the data from the years going wide to the right to each line being one county, one year. And, I've added a date column, which is the estimates date.

Now, if we tried to make a line chart of all 3,142 counties, we'd get a mess. But, it's the first mistake people make in creating a line chart, so let's do that.

```
ggplot() + geom_line(data=populationestimates, aes(x=Date, y=Population, group=Name))
```



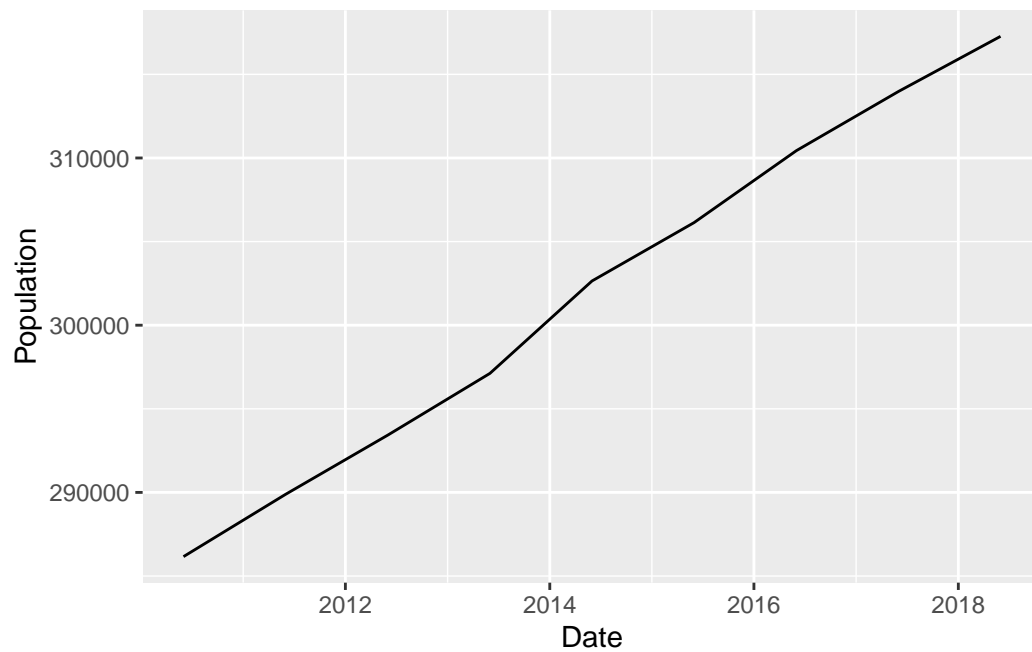
And what do we learn from this? There's one very, very big county, some less big counties, and a ton of smaller counties. So many we can't see Them, and because the numbers are so big, any changes are dwarfed.

So let's thin the herd here. How about we just look at Lancaster County, Nebraska.

```
lancaster <- populationestimates |> filter(Name == "Lancaster County, Nebraska")
```

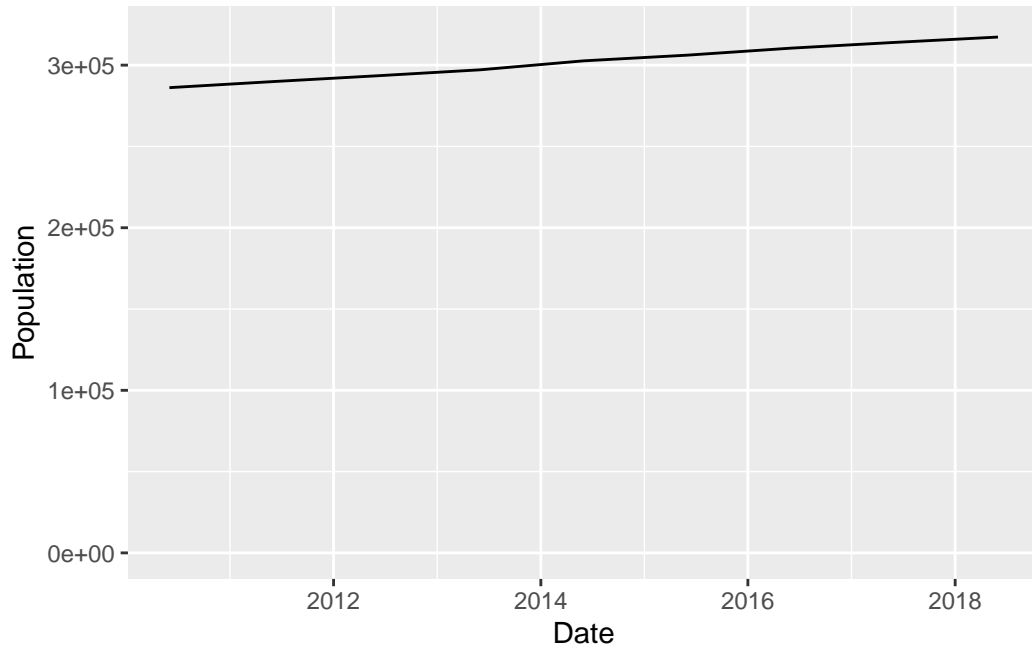
Now let's chart it.

```
ggplot() + geom_line(data=lancaster, aes(x=Date, y=Population, group=Name))
```



Growing like gangbusters, right? Well, not exactly. Note the y axis doesn't start at 0.

```
ggplot() + geom_line(data=lancaster, aes(x=Date, y=Population, group=Name)) + scale_y_cont
```



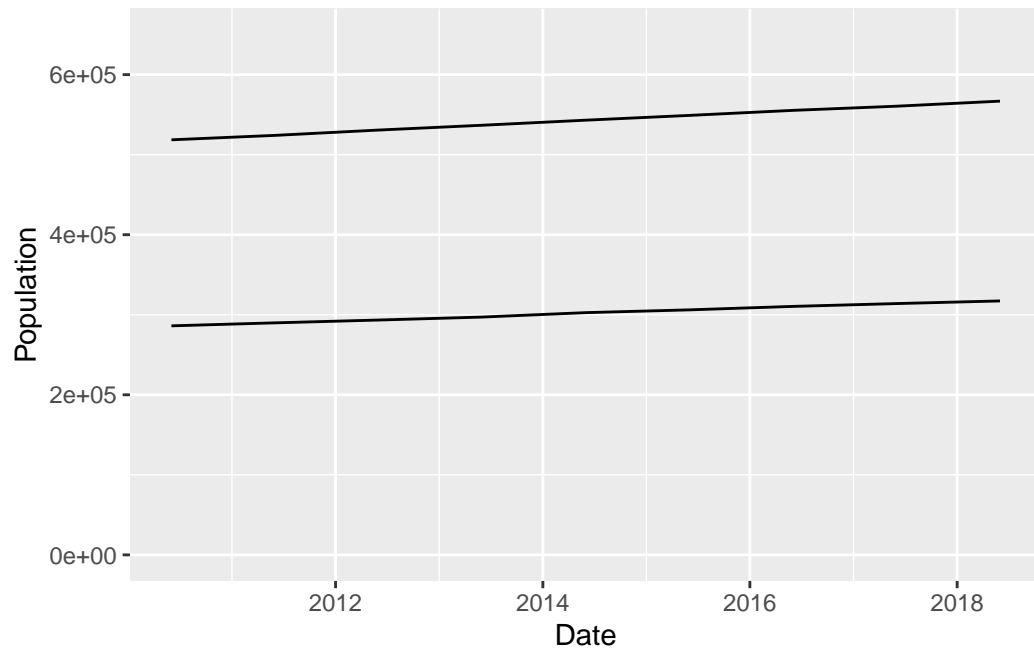
More accurate.

But how does that compare to Omaha? Geoms work in layers. We can just add Omaha. First, we need a dataframe with Douglas County in it.

```
douglas <- populationestimates |> filter(Name == "Douglas County, Nebraska")
```

With that, we just add another geom\_line. We will also need to adjust our y axis limits to expand to fit Omaha.

```
ggplot() + geom_line(data=lancaster, aes(x=Date, y=Population, group=Name)) + geom_line(da
```



So both counties are growing, but from the line we can see that Omaha is growing just ever so slightly faster. No accounting for taste.



## 18 Writing with numbers

The number one sin of all early career data journalist is to get really, really, really attached to the analysis you've done and include every number you find.

Don't do that.

Numbers tell you what. Numbers rarely tell you why. What question has driven most people since they were three years old? Why. The very first thing to do is realize that is the purpose of reporting. You've done the analysis to determine the what. Now go do the reporting to do the why. Or as an old editor of mine used to say "Now go do that reporting shit you do."

The trick to writing a numbers story is to frame your story around people. Sometimes, your lead can be a number, if that number is compelling. Often, your lead is a person, a person who is one of the numbers you are writing about.

Tell their story. Briefly. Then, let us hear from them. Let them speak about what it is you are writing about.

Then come the numbers.

### 18.1 How to write about numbers without overwhelming with numbers.

Writing complex stories is often a battle against that complexity. You don't want to overwhelm. You want to simplify where you can. The first place you can do that is only use exact numbers where an exact number is called for.

Where you can, do the following:

- Using ratios instead of percents
- Often, it's better to put it in counts of 10. 6 of 10, 4 of 10. It's easy to translate that from a percentage to a ratio.
- But be careful when your number is 45 percent. Is that 4 in 10 or 5 in 10?
- If a ratio doesn't make sense, round. There's 287,401 people in Lincoln, according to the Census Bureau. It's easier, and no less accurate, to say there's more than 287,000 people in Lincoln.

**A critical question your writing should answer: As compared to what?**

How does this compare to the average? The state? The nation? The top? The bottom?

One of the most damning numbers in the series of stories Craig Pittman and I wrote that became the book [Paving Paradise](#) was comparing approvals and denials.

We were looking at the US Army Corps of Engineers and their permitting program. We were able to get a dataset of just a few years of permits that was relatively clean. From that, we were able to count the number of times the corps had said yes to a developer to wipe out wetlands the law protected and how many times they said no.

They said yes 12,000 times. They said no once.

That one time? Someone wanted to build an eco-lodge in the Everglades. Literally. Almost every acre of the property was wetlands. So in order to build it, the developer would have to fill in the very thing they were going to try to bring people into. The corps said no.

## **18.2 When exact numbers matter**

Sometimes ratios and rounding are not appropriate.

This is being written in the days of the coronavirus. Case counts are where an exact number is called for. You don't say that there are more than 70 cases in Lancaster County on the day this was written. You specify. It's 75.

You don't say almost 30 deaths. It's 28.

Where this also comes into play is any time there are deaths: Do not round bodies.

## **18.3 An example**

[Read this story from USA Today and the Arizona Republic](#). Notice first that the top sets up a conflict: People say one thing, and that thing is not true.

No one could have anticipated such a catastrophe, people said. The fire's speed was unprecedented, the ferocity unimaginable, the devastation unpredictable.

Those declarations were simply untrue. Though the toll may be impossible to predict, worst-case fires are a historic and inevitable fact.

The first voice you hear? An expert who studies wildfires.

Phillip Levin, a researcher at the University of Washington and lead scientist for the Nature Conservancy in Washington, puts it this way: “Fire is natural. But the disaster happens because people didn’t know to leave, or couldn’t leave. It didn’t have to happen.”

Then notice how they take what is a complex analysis using geographic information systems, raster analysis, the merging of multiple different datasets together and show that it’s quite simple – the averaging together of pixels on a 1-5 scale.

Then, they compare what they found to a truly massive fire: The Paradise fire that burned 19,000 structures.

Across the West, 526 small communities — more than 10 percent of all places — rank higher.

And that is how it’s done. Simplify, round, ratios: simple metrics, powerful results.

# 19 Ethics in data journalism

[This originally appeared on Open News in March 2013.](#)

In 2009, a senior web editor asked me and another developer a question: could our development group build a new news application for Tampabay.com that displayed a gallery of mug shots? Stories about goofy crimes with strange mug shots were popular with readers. The vision, on the part of management, was a website that would display the mugshots collected every day from publicly available websites by two editors—well paid, professional editors with other responsibilities.

Newsrooms are many things. Alive. Filled with energy. Fueled by stress, coffee and profanity. But they are also idea factories. Day after day, ideas come from everywhere. From reporters on the beat. From editors reading random things. From who knows where. Some of them are brilliant. Some would never work. Most need more people and time than are available. And some are dumber than anyone cares to admit.

We thought this idea was nuts. Why would we pay someone, let alone an editor, to fetch mug shots from the Internet? Couldn't we do that with a scraper?

If only this were the most complex question we would face.

Because given enough time and enough creativity, scraping a mug shot website is easy. You need to recognize a pattern, parse some HTML and gather the pieces you need. At least that's how it should work. Police agencies that put mugs online usually buy software from a vendor. Apparently, those vendors enjoy making horrific, non-standard, broken-in-interesting-and-unique-ways HTML. You'll swear. A lot. But you'll grind it out. And that's part of the fun. Scraping isn't any fun with clean, semantic, valid HTML. And scraping mug shot websites, by that definition, is tons of fun.

The complexity comes when you realize the data you are dealing with represent real people's lives.

## 19.1 Problems

The first problem we faced, long before we actually had data, was that data has a life of its own. Because we were going to put this information in front of a big audience, Google was going to find it. That meant if we used our normal open door policy for the Googlebot, someone's mug

shot was going to be the first record in Google for their name, most likely. It would show up first because most people don't actively cultivate their name on the web for visibility in Google. It would show up first because we know how SEO works and they don't. It would show up first because our site would have more traffic than their site, and so Google would rank us higher.

And that record in Google would exist as long as the URL did. Longer when you consider the cached versions Google keeps.

That was a problem because here are the things we could not know:

- Was this person wrongly arrested?
- Was this person innocent?
- Were the charges dropped against this person?
- Did this person lie about any of their information?

## 19.2 The Googlebot

So it turned out to be very important to know the Googlebot. It's your friend ... until it isn't. We went to our bosses and said words that no one had said to them before: we did not want Google to index these pages. In a news organization, the page view is the coin of the realm. It is — unfortunately — how many things are evaluated when the bosses ask if it was successful or not. So, with that in mind, Google is your friend. Google brings you traffic. Indeed, Google is your single largest referrer of traffic at a news organization, so you want to throw the doors open and make friends with the Googlebot.

But here we were, saying Google wasn't our friend and that we needed to keep the Googlebot out. And, thankfully, our bosses listened to our argument. They too didn't want to be the first result in Google for someone.

So, to make sure we were telling the Googlebot no, we used three lines of defense. We told it no in robots.txt and on individual pages as a meta tag, and we put the most interesting bits of data into a simple JavaScript wrapper that made it hard on the bot if the first two things failed.

The second solution had ramifications beyond the Googlebot. We decided that we were not trying to make a complete copy of the public record. That existed already. If you wanted to look at the actual public records, the sheriff's offices in the area had websites and they were the official keeper of the record. We were making browsing those images easy, but we were not the public record.

That freedom had two consequences: it meant our scrapers could, at a certain point and given a number of failures, just give up on getting a mug. Data entered by humans will be flawed. There will be mistakes. Because of that, our code would have to try and deal with that. Well,

there's an infinite number of ways people can mess things up, so we decided that since we were not going to be an exact copy of the public record, we could deal with the most common failures and dump the rest. During testing, we were getting well over 98% of mugs without having to spend our lives coding for every possible variation of typo.

The second consequence of the decision actually came from the newspapers lawyers. They asked a question that dumbfounded us: How long are you keeping mugs? We never thought about it. Storage was cheap. We just assumed we'd keep them all. But, why should we do that? If we're not a copy of the public record, we don't have to keep them. And, since we didn't know the result of each case, keeping them was really kind of pointless.

So, we asked around: How long does a misdemeanor case take to reach a judgement? The answer we got from various sources was about 60 days. From arrest to adjudication, it took about two months. So, at the 60 day mark, we deleted the data. We had no way of knowing if someone was guilty or innocent, so all of them had to go. We even called the script The Reaper.

We'd later learn that the practical impacts of this were nil. People looked at the day's mugs and moved on. The amount of traffic a mug got after the day of arrest was nearly zero.

## 19.3 Data Lifetimes

The life of your data matters. You have to ask yourself, Is it useful forever? Does it become harmful after a set time? We had to confront the real impact of deleting mugs after 60 days. People share them, potentially lengthening their lifetime long after they've fallen off the homepage. Delete them and that URL goes away.

We couldn't stop people from sharing links on social media—and indeed probably didn't want to stop them from doing it. Heck, we did it while we were building it. We kept IMing URLs to each other. And that's how we realized we had a problem. All our work to minimize the impact on someone wrongly accused of a crime could be damaged by someone sharing a link on Facebook or Twitter.

There's a difference between frictionless and unobstructed sharing and some reasonable constraints.

We couldn't stop people from posting a mug on Facebook, but we didn't have to make it easy and we didn't have to put that mug front and center. So we blocked Facebook from using the mug as the thumbnail image on a shared link. And, after 60 days, the URL to the mug will throw a 404 page not found error. Because it's gone.

We couldn't block Google from memorializing someone's arrest, only to let it live on forever on Facebook.

## 19.4 You Are a Data Provider

The last problem didn't come until months later. And it came in the middle of the night. Two months after we launched, my phone rang at 1 a.m. This is never a good thing. It was my fellow developer, Jeremy Bowers, now with NPR, calling me from a hotel in Washington DC where he was supposed to appear in a wedding the next day. Amazon, which we were using for image hosting, was alerting him that our bandwidth bills had tripled on that day. And our traffic hadn't changed.

What was going on?

After some digging, we found out that another developer had scraped our site—because we were so much easier to scrape than the Sheriff's office sites—and had built a game out of our data called Pick the Perp. There were two problems with this: 1. The game was going viral on Digg (when it was still a thing) and Reddit. It was getting huge traffic. 2. That developer had hotlinked our images. He/she was serving them from our S3 account, which meant we were bearing the costs. And they were going up exponentially by the minute.

What we didn't realize when we launched, and what we figured out after Pick the Perp, was that we had become data provider, in a sense. We had done the hard work of getting the data out of a website and we put it into neat, semantic, easily digestible HTML. If you were after a stream of mugshots, why go through all the hassle of scraping four different sheriff's office's horrible HTML when you could just come get ours easily?

Whoever built Pick the Perp, at least at the time, chose to use our site. But, in doing so, they also chose to hotlink images—use the URL of our S3 bucket, which cost us money—instead of hosting the images themselves.

That was a problem we hadn't considered. People hotlink images all the time. And, until those images are deleted from our system, they'll stay hotlinked somewhere.

Amazon's S3 has a system where you can attach a key to a file that expires after X period of time. In other words, the URL to your image only lasts 15 minutes, or an hour, or however long you decide, before it breaks. It gives you fine grained control over how long someone can use your image URL.

So at 3 a.m., after two hours of pulling our hair out, we figured out how to sync our image keys with our cache refreshes. So every 15 minutes, a url to an image expired and Pick the Perp came crashing down.

While the Pick the Perp example is an easy one—it's never cool to hotlink an image—it does raise an issue to consider. Because you are thinking carefully about how to build your app the right way doesn't mean someone else will. And it doesn't mean they won't just go take your data from your site. So how could you deal with that? Make the data available as a download? Create an API that uses your same ethical constructs? Terms of service? All have pros and cons and are worth talking about before going forward.

## 19.5 Ethical Data

We live in marvelous times. The web offers you no end of tools to make things on the web, to put data from here on there, to make information freely available. But, we're an optimistic lot. Developers want to believe that their software is being used only for good. And most people will use it for good. But, there are times where the data you're working with makes people uncomfortable. Indeed, much of journalism is about making people uncomfortable, publishing things that make people angry, or expose people who don't want to be exposed.

What I want you to think about, before you write a line of code, is what does it mean to put your data on the internet? What could happen, good and bad? What should you do to be responsible about it?

Because it can have consequences.

On Dec. 23, the Journal News in New York published a map of every legal gun permit holder in their home circulation county. It was a public record. They put it into Google Fusion Tables and Google dutifully geocoded the addresses. It was a short distance to publication from there.

Within days, angry gun owners had besieged the newspaper with complaints, saying the paper had given criminals directions to people's houses where they'd find valuable guns to steal. They said the paper had violated their privacy. One outraged gun owner assembled a list of the paper's staff, including their home addresses, telephone numbers, email addresses and other details. The paper hired armed security to stand watch at the paper.

By February, the New York state legislature removed handgun permits from the public record, citing the Journal News as the reason.

There's no end of arguments to be had about this, but the simple fact is this: The reason people were angry was because you could click on a dot on the map and see a name and an address. In Fusion Tables, removing that info window would take two clicks.

Because you can put data on the web does not mean you should put data on the web. And there's a difference between a record being "public" and "in front of a large audience."

So before you write the first line of code, ask these questions:

- This data is public, but is it widely available? And does making it widely available and easy to use change anything?
- Should this data be searchable in a search engine?
- Does this data expose information someone has a reasonable expectation that it would remain at least semi-private?
- Does this data change over time?
- Does this data expire?
- What is my strategy to update or delete data?



- How easy should it be to share this data on social media?
- How should I deal with other people who want this data? API? Bulk download?

Your answers to these questions will guide how you build your app. And hopefully, it'll guide you to better decisions about how to build an app with ethics in mind.

## 20 Installations

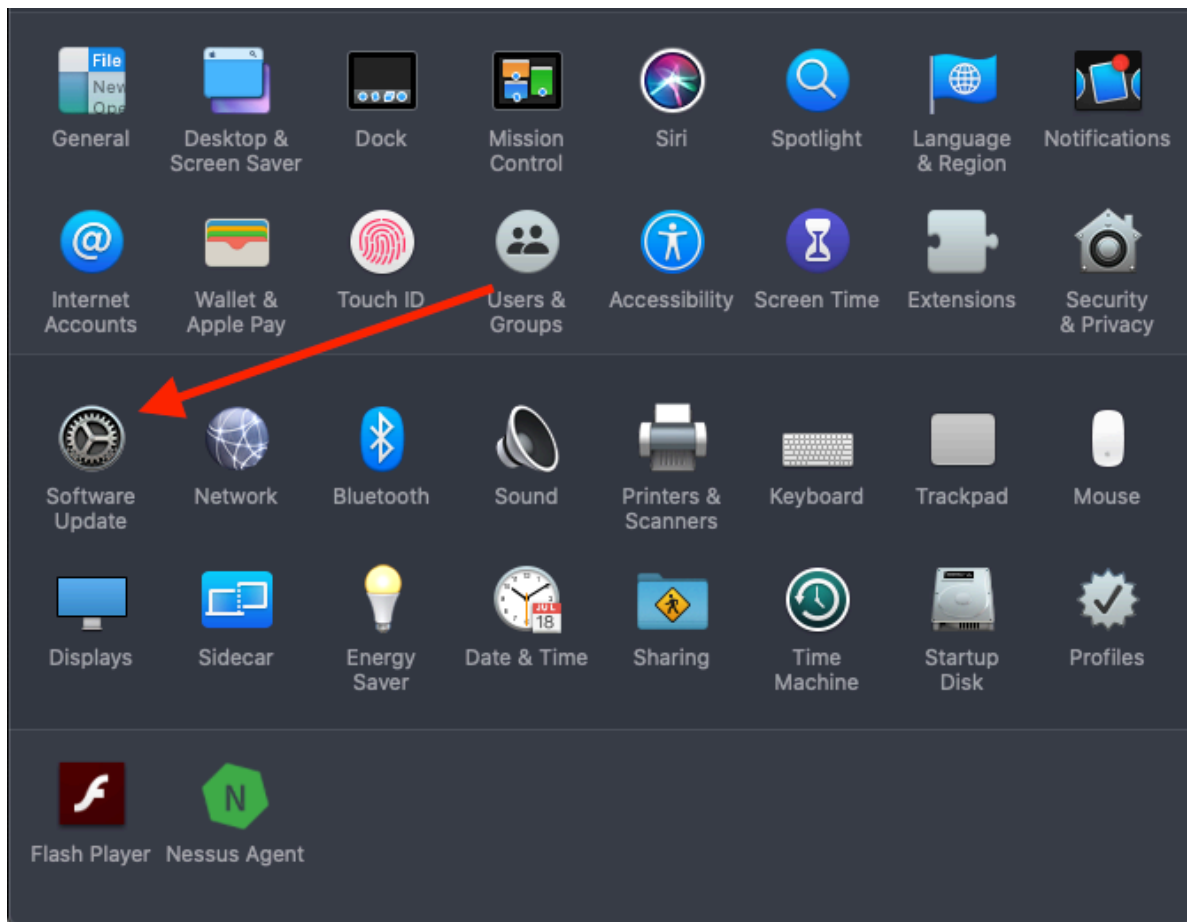
You're going to do things most of you aren't used to doing with your computer in this class. In order to do that, you need to clean up your computer. I've seen what your computer looks like. It's disgusting.

### 20.1 Part 1: Update and patch your operating system

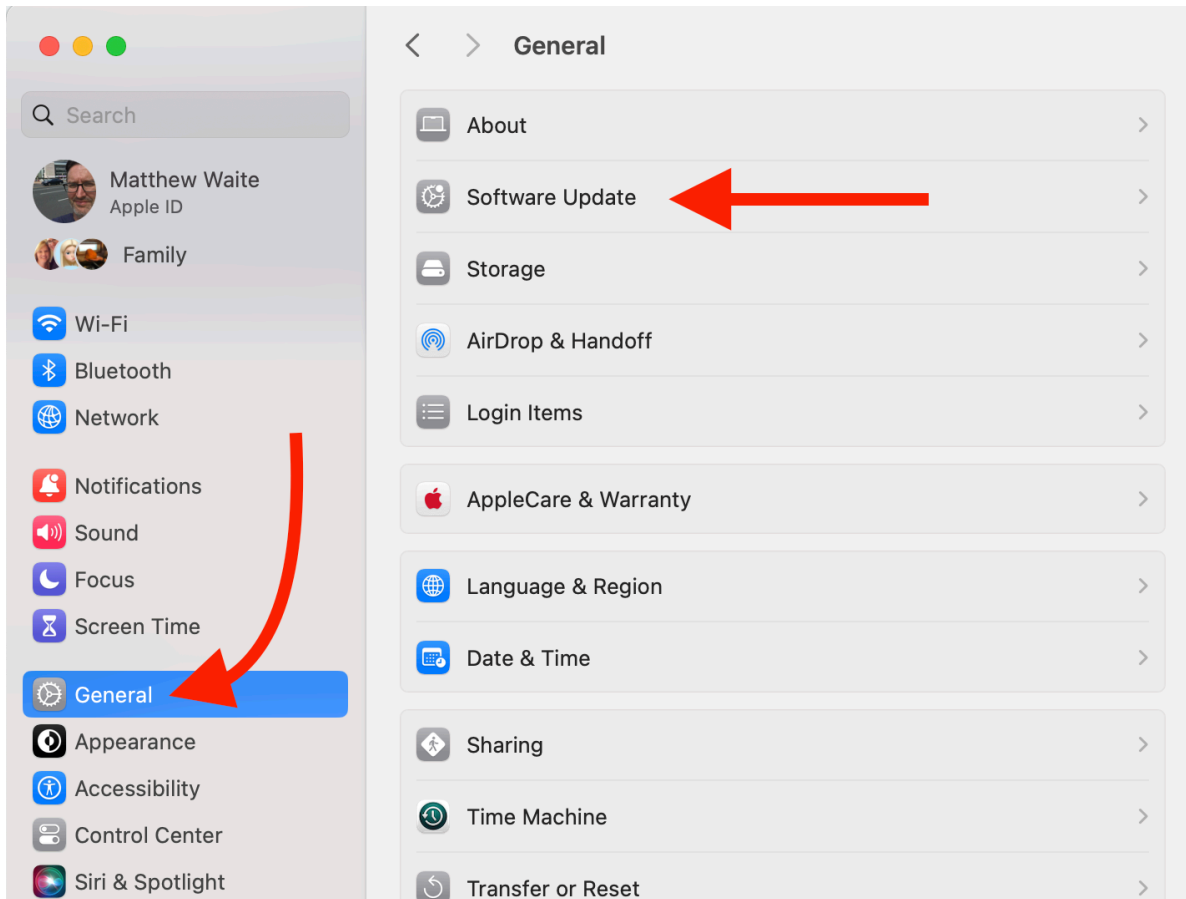
**On a Mac:**

1. Open System Preferences.

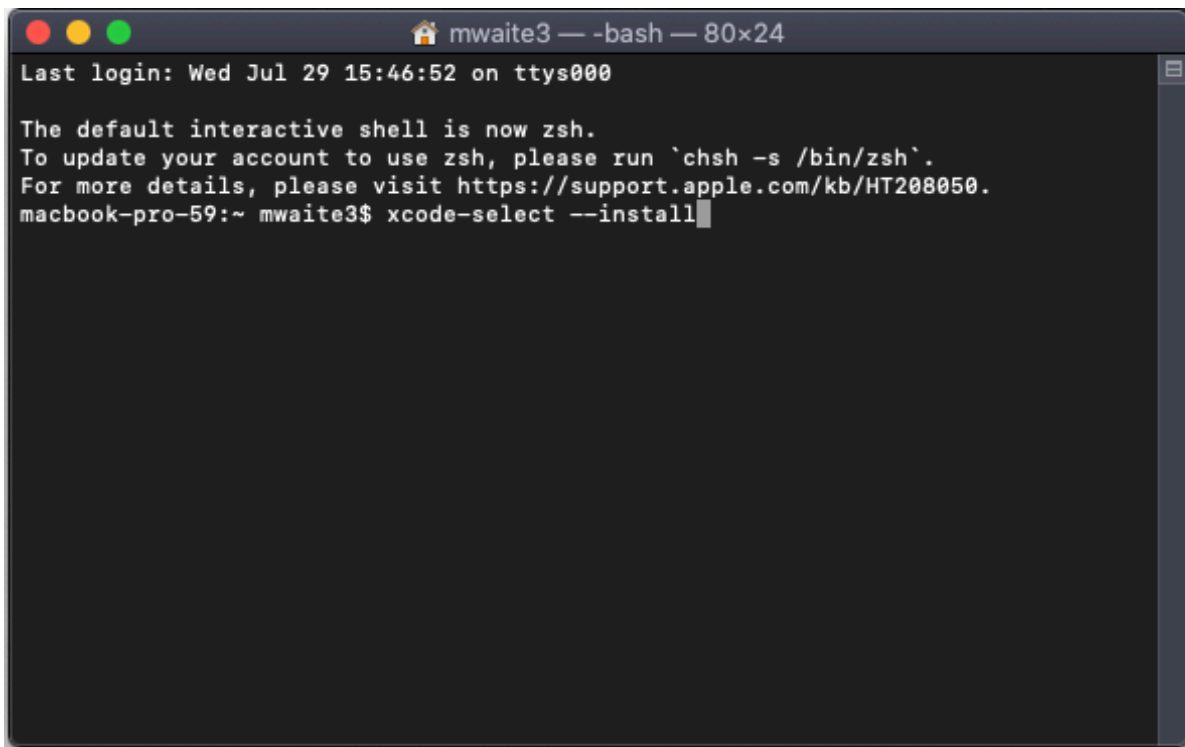
Depending on how old your Mac OS is, you might see this:



Or you might see this:



2. Check and see if you have the latest version of the Mac OS installed. If your computer says “Your Mac is up to date”, then you’re good to go, regardless of what comes next.
3. If you aren’t on Sonoma and you can update to it, you should do it. This will take some time – hours, so don’t do it when you need your laptop – but it’s important for you and your computer to stay up to date on operating systems.
4. When you’re done, make sure you click the Automatically keep my Mac up to date box and install those updates regularly. **Don’t ignore them. Don’t snooze them. Install them.**
5. With an up-to-date operating system, now install the command line tools. To do this, click on the magnifying glass in the top right of the screen and type terminal. Hit enter – the first entry is the terminal app.
6. In the terminal app, type `xcode-select --install` and hit enter. Let it run.

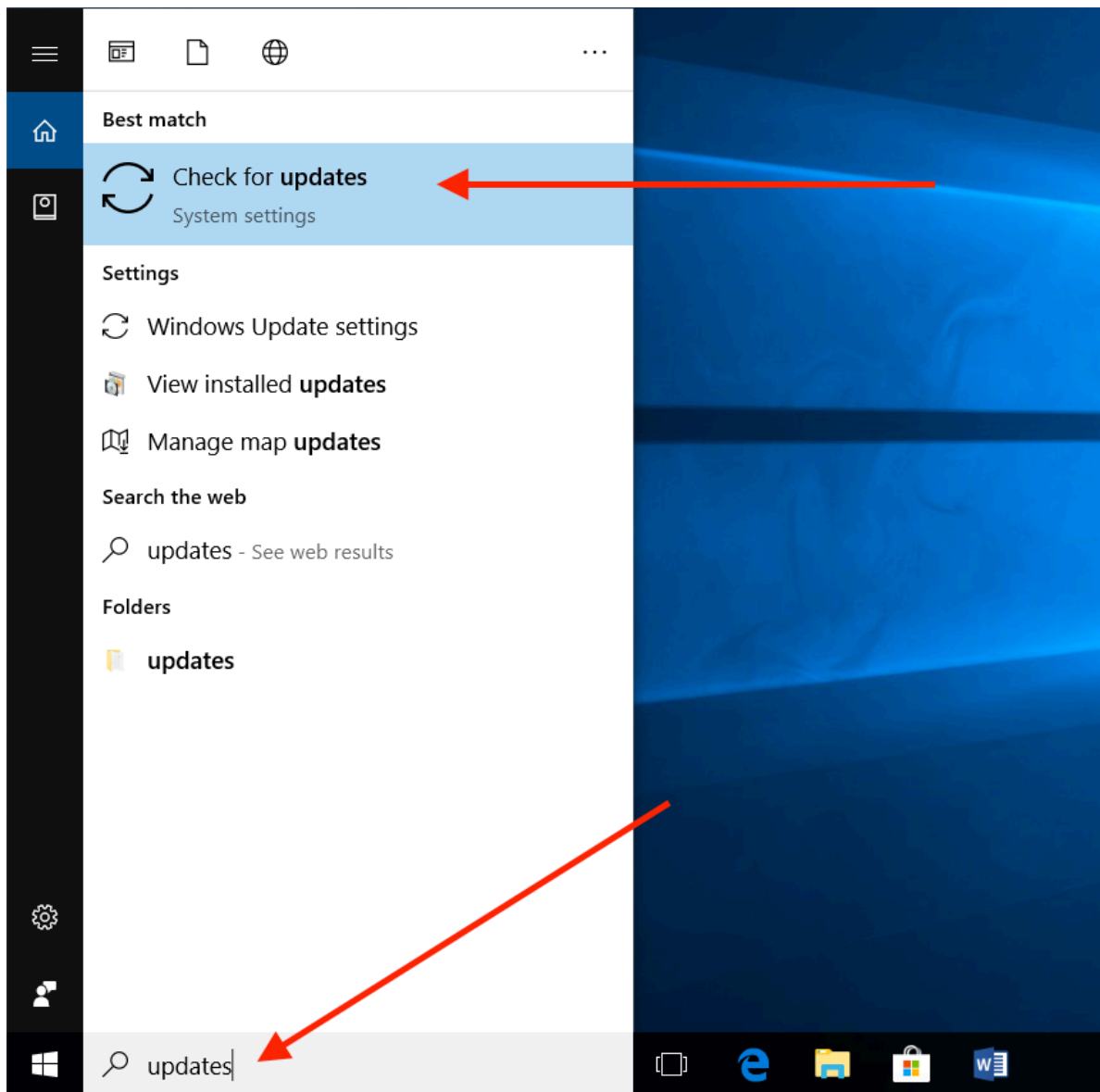
A terminal window with a dark background and light gray text. The window title bar shows three colored circles (red, yellow, green) on the left, a home icon followed by 'mwaite3' and '-bash' in the center, and '80x24' on the right. The terminal content shows a login message, a notification about switching to zsh, and a command being executed.

```
mwaite3 — -bash — 80x24
Last login: Wed Jul 29 15:46:52 on ttys000

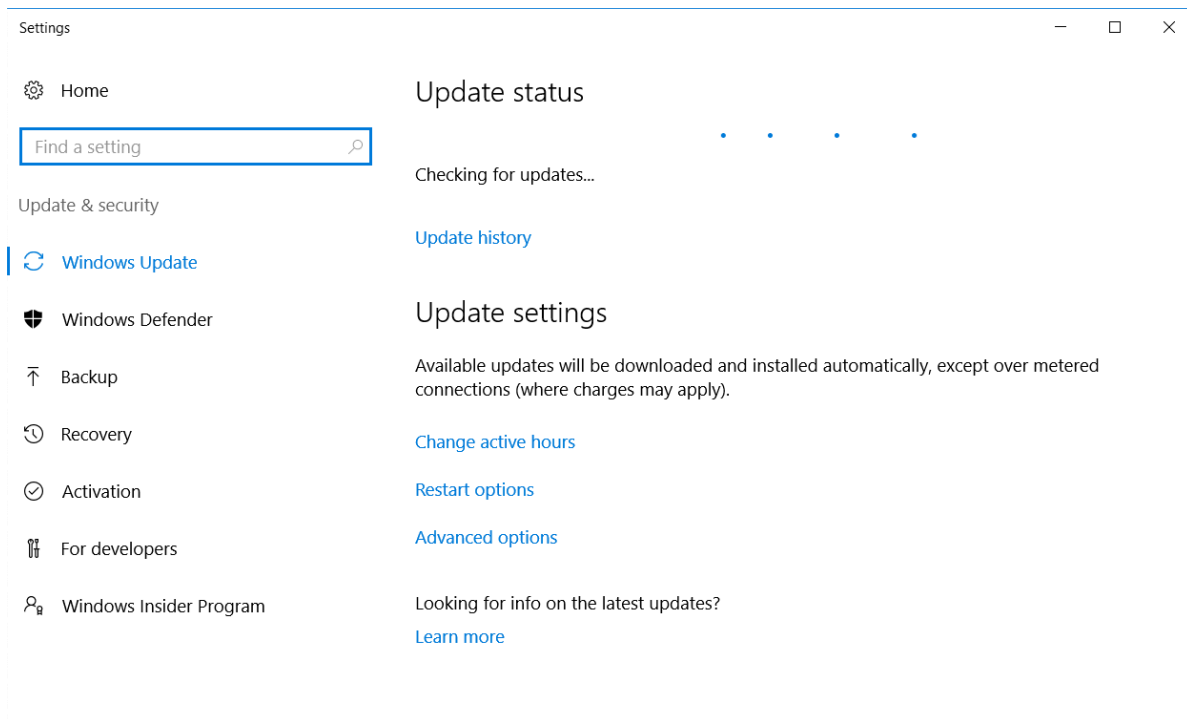
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
macbook-pro-59:~ mwaite3$ xcode-select --install
```

**On Windows:**

1. Type Updates into the Cortana search then click Check for updates



2. After the search for updates completes, apply any that you have. Depending on if you'd done this recently or if you have automatic updates set, this might take a long time or go very quickly.



3. When you're done, make sure you set up automatic updates for your Windows machine and install those updates regularly. Don't ignore them. Don't snooze them. Install them.

## 20.2 Part 2: Install R and R Studio

1. Go [here](#). Go to Step 1 and click Download and Install R
  - If you're on a Mac, click on Download R for MacOS. If you have a newer Mac with an M1/M2/M3 chip, you want the arm64 version. If you're on an older Mac with an Intel chip, you want the X86\_64 version.
  - If you're on Windows, install the base package *AND* install Rtools. When either downloads, run the executable and accept the defaults and license agreement.

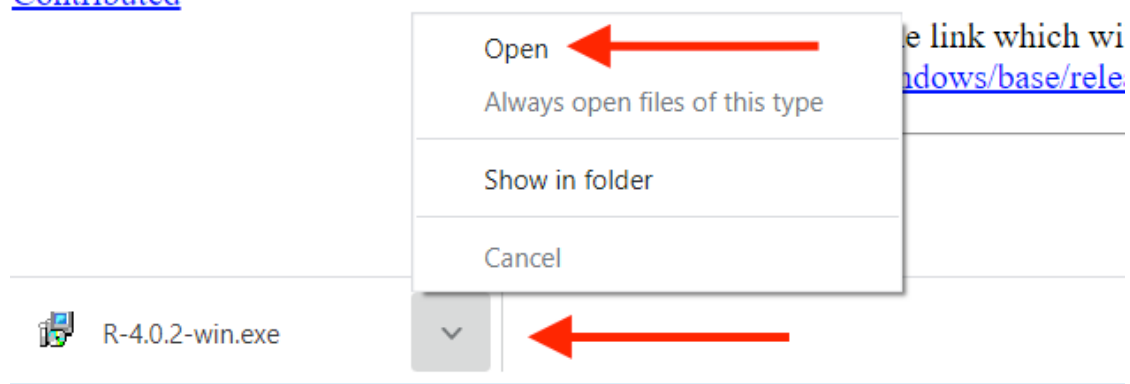
[Documentation](#)

[Manuals](#)

[FAQs](#)

[Contributed](#)

- Features to this release are incorporated
- A build of the development version (v [snapshot build](#)).
- [Previous releases](#)



2. Go back to [here](#). Go to Step 2 and click R Studio Desktop for your version.

**Mac users:**

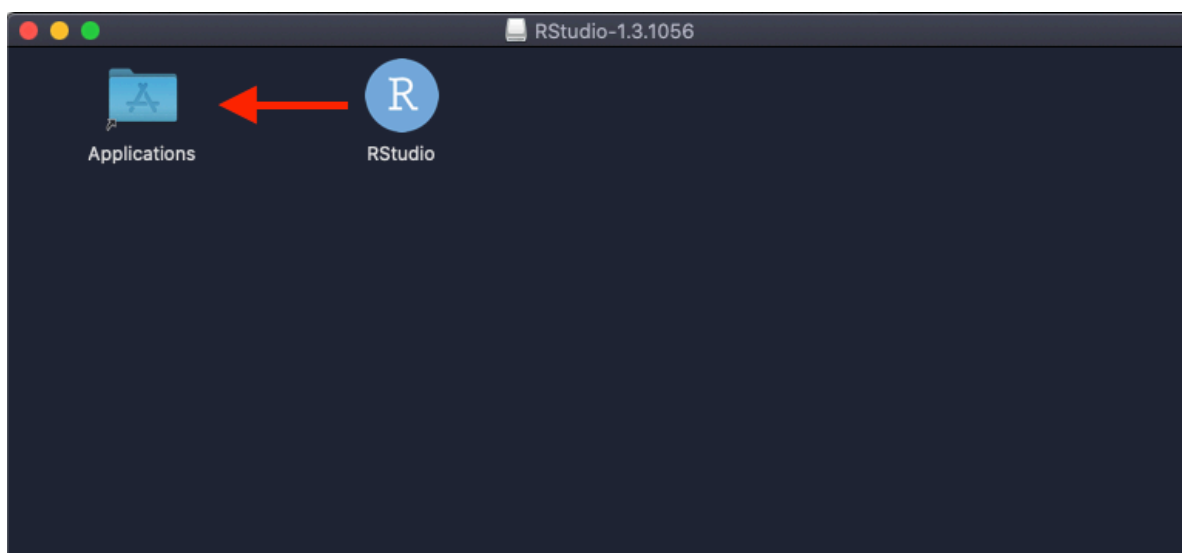


Figure 20.1: Make sure you drag the R Studio icon into the Applications folder icon.

**Windows users:**

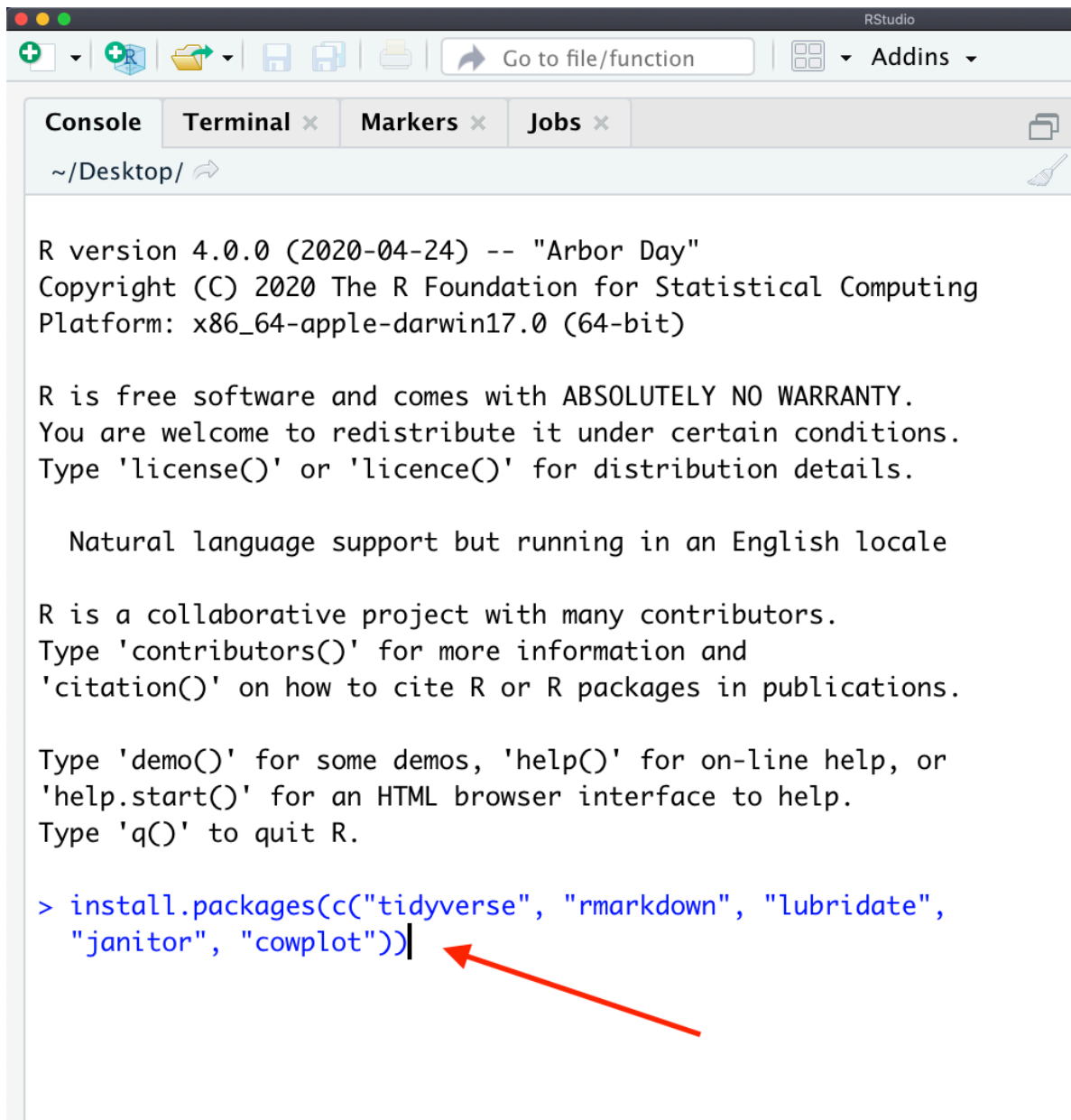
You can find it by typing RStudio into the Cortana search.



## 20.3 Part 3: Installing R libraries

1. Open R Studio. It should show the Console view by default. We'll talk a lot more about the console later.
2. Copy and paste this into the console and hit enter:

```
install.packages(c("tidyverse", "rmarkdown", "lubridate", "janitor", "learnr",  
"remotes", "devtools", "waffle", "ggrepel", "ggbeeswarm", "ggbump", "ggalt",  
"ggtext", "rvest"))
```



The screenshot shows the RStudio interface with the console pane active. The console displays the standard R startup message, including the version (4.0.0), copyright (2020 The R Foundation), and platform (x86\_64-apple-darwin17.0). It also includes a disclaimer about the warranty and instructions on how to use R. At the bottom of the console, a blue prompt character is followed by the command to install several packages: `install.packages(c("tidyverse", "rmarkdown", "lubridate", "janitor", "cowplot"))`. A red arrow points to the closing parenthesis of this command.

```
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

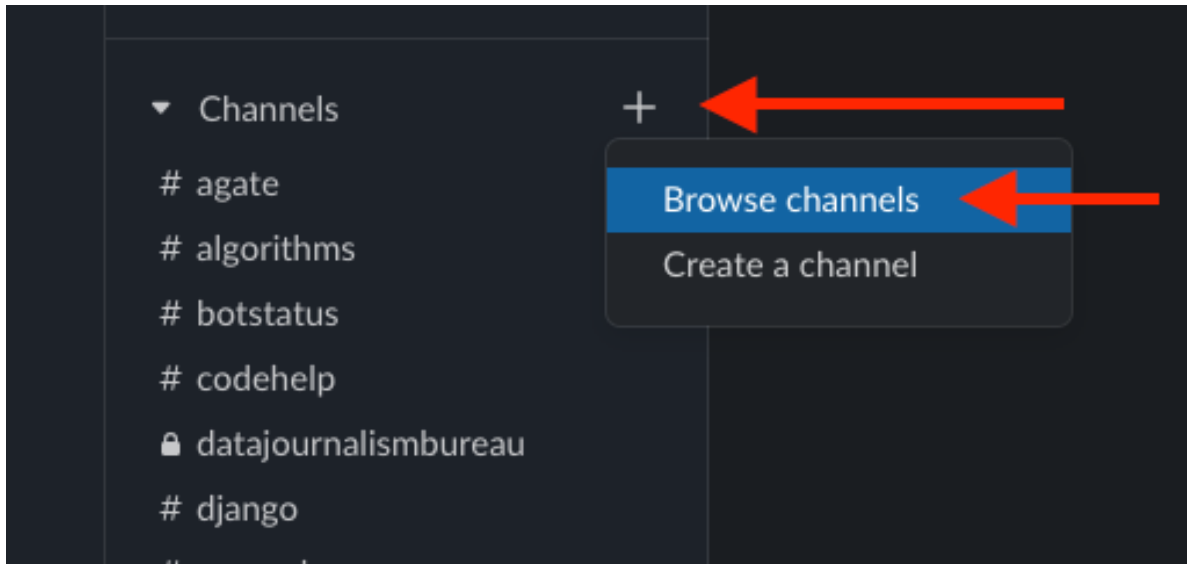
> install.packages(c("tidyverse", "rmarkdown", "lubridate",
  "janitor", "cowplot"))
```

## 20.4 Part 4: Install Slack

1. Install [Slack](#) on your computer and your phone (you can find Slack in whatever app store you use). The reason I want it on both is because you are going to ask me for help with code via Slack. **Do not use screenshots unless specifically asked.** I want you to copy and paste your code. You can't do that on a phone. So you need the desktop

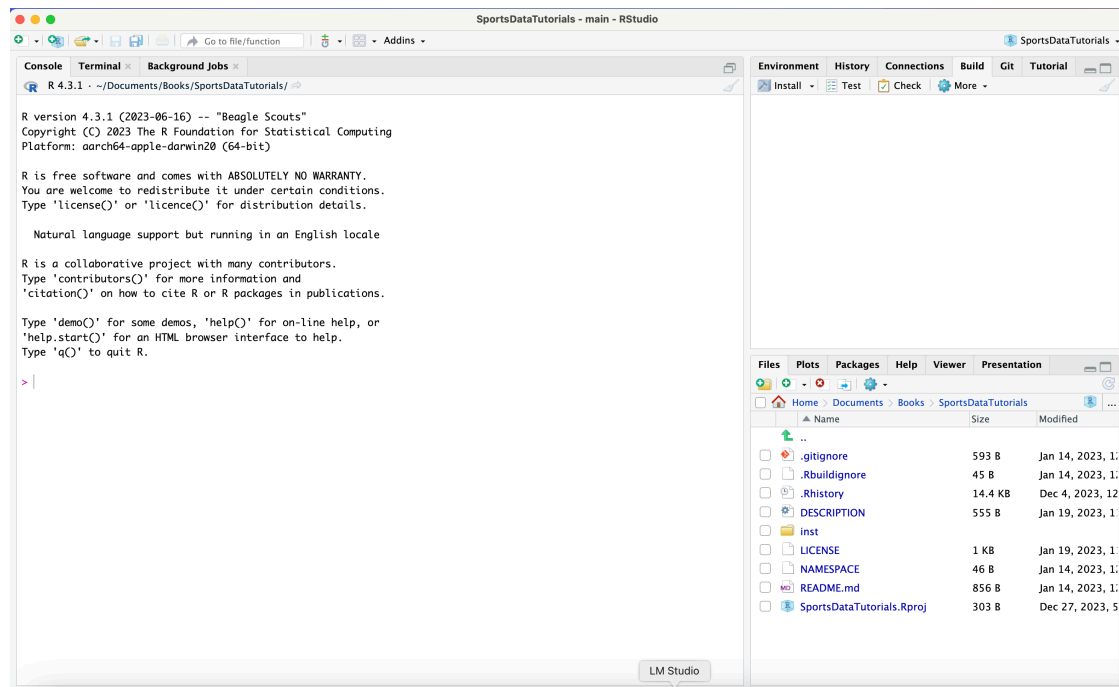
version. But I can usually solve your problem within a few minutes if you respond right away, and I know that you have your phone on you and are checking it. So the desktop version is for work, the phone version is for notifications.

2. Email me the address you want connected to Slack. Use one you'll actually check.
3. When you get the Slack invitation email, log in to the class slack via the apps, **not the website**.
4. Add the #r channel for general help I'll send to everyone in the channel and, if you want, the #jobstuff channel for news about jobs I come across.

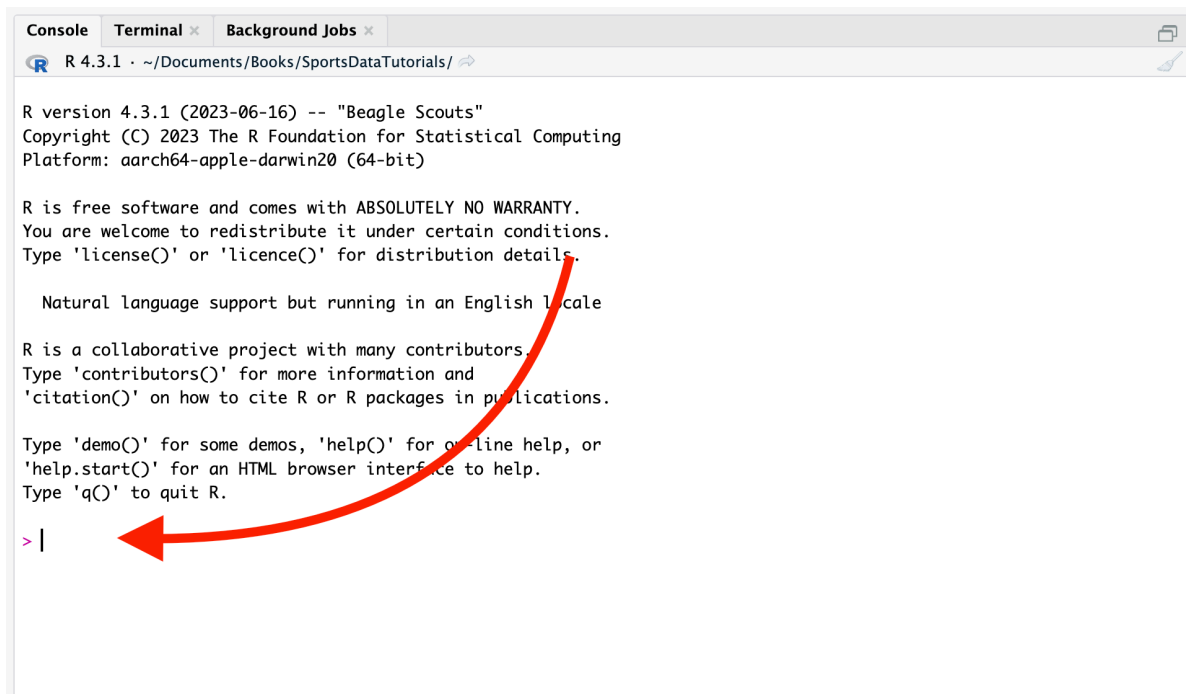


## 20.5 Part 5: Install the tutorials

To get the tutorials, do the following.



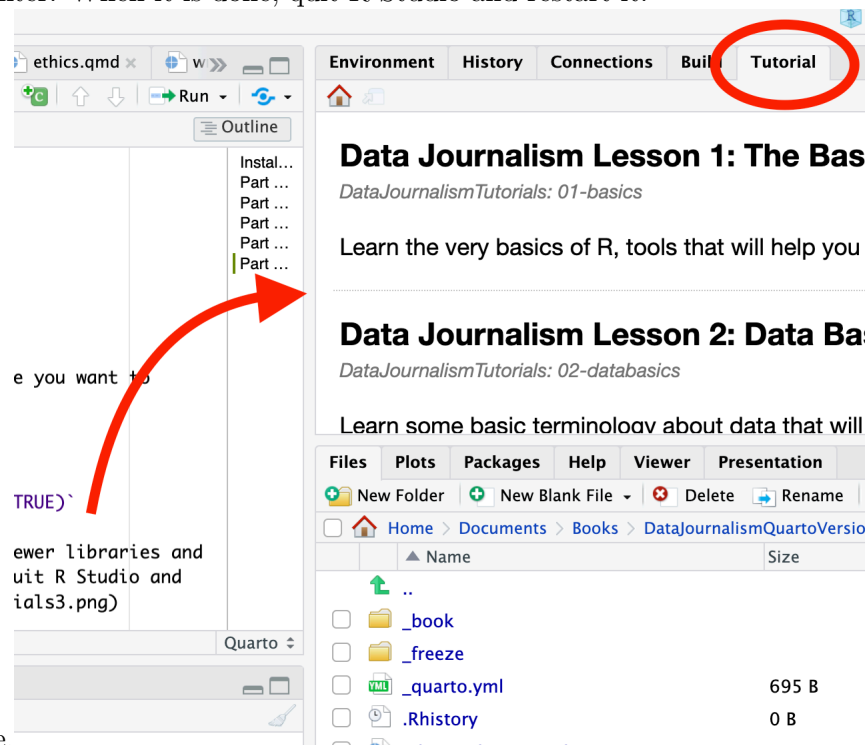
1. Open R Studio.
2. R Studio defaults to the console view. This is good, This is where you want to be.



3. In the console, enter the following:

```
devtools::install_github("mattwaite/DataJournalismTutorials", force=TRUE)
```

4. You should see some automated output. If you are told there are newer libraries and asked if you want to install them, just hit enter. When it is done, quit R Studio and restart it.



This is what it will look like when done.