Telling Stories With Data

Rohan Alexander

2021-01-19

Contents

Part I

Essentials

Chapter 1

Introduction

These notes are being actively developed.

If you have comments or suggestions, or find mistakes, then please don't hesitate to get in touch.

Version: 0.0.0.9000.

1.1 Welcome

Hi, I'm Rohan Alexander. You can find out more about me here. These are notes that I wrote to support my teaching at the University of Toronto across the Faculty of Information and the Department of Statistical Sciences. The focus is on using quantitative methods to tell stories with data.

1.2 Structure

The parts of these notes are:

- Essentials
  - Hello world
  - R essentials
  - Workflow
- Communicate
  - Graphs, tables and text
  - Maps
  - Websites
  - Shiny
- Hunting and gathering
  - Sampling and survey essentials
  - APIs, scraping, PDFs and text

  - RCTs, A/B testing
  - Implementing surveys
  - Open data
  - Other
- Clean
  - Cleaning and preparing
  - Storage and retrieval
  - Dissemination and protection
- Model
  - Exploratory data analysis
  - Regression essentials
  - Matching and difference-in-differences
  - Instrumental variables
  - Regression discontinuity design
  - Poll of polls
  - Multilevel regression with post-stratification
  - Text
- Other
  - Cloud
  - Deploy

1.3 On telling stories

Like many parents, when our child was born, one of the first things that my wife and I did regularly was read stories to him. In doing so we carried on a tradition that has occurred for millennia. Myths, fables, and fairy tales can be seen and heard all around us. Not only are they entertaining but they enable us to easily learn something about the world. While 'The Very Hungry Caterpillar' may seem quite far from the world of quantitative analysis, there are similarities. Both are trying to tell the reader a story.

When conducting quantitative analysis we are trying to tell the reader a story that will convince them of something. It may be as exciting as predicting elections, as banal as increasing internet advertising click rates by 0.01 per cent, as serious as finding the cause of some disease, or as fun as forecasting the winner of a basketball game. In any case the key elements are the same. When writing fiction Wikipedia suggests there are five key elements: character, plot, setting, theme, and style. When we are conducting quantitative analysis we have analogous concerns:

1. What is the data? Who generated it and how?
2. What is the data trying to say? How can we let it say this?
3. What is the broader context surrounding the data? Where and when was it generated? Could other data have been generated?
4. What are we hoping others will see from this data?
5. How can you convince them of this?

In the past, certain elements of telling stories with quantitative data were easier. For instance, experimental design has a long and robust tradition within traditional applications such as agricultural and medical sciences, physics, and chemistry. Student's t-distribution was identified by a chemist, William Sealy Gosset, who was working at Guinness and needed to assess the quality of the beer (**?**)! It would have been possible for him to randomly sample the beer and change one aspect at a time. Indeed, many of the fundamental statistical methods that we use today were developed in an agricultural setting. In the settings for which they were developed it was typically possible to establish control groups, randomize, and easily deal with any ethical concerns. In such a setting any subsequent story that is told with the resulting data is likely to be fairly convincing.

Unfortunately, such a set-up is rarely possible in modern applied statistics applications. On the other hand, there are many aspects that are easier today. For instance, we have well-developed statistical techniques, easier access to larger datasets, and open source statistical languages such as R. But the lack of ability to conduct traditional experiments means that we must turn to other aspects in order to tell a reader a convincing story about our data. These other aspects allow us to tell convincing stories even in the absence of a traditional experimental set-up.

1.4 Telling stories with data

The aim of these notes is to equip you with everything you need to be able to write short(ish), technical, memos, that convince a reader of the story you are telling. These notes encourage research-based, independent learning. This means that you should develop your own questions and answer them to the extent that you can. We focus on methods that can provide convincing stories even when we cannot conduct traditional experiments. Importantly, these approaches do not rely on 'big data' (which is widely known by practitioners to not be a panacea (**?**)), but instead on better using the data that are available. The purpose of the notes is to allow you to tell convincing stories using data and quantitative analysis. They blend theory and case studies to equip you to with practical skills, a sophisticated workflow, and an appreciation for how more-advanced methods build on what is covered here.

Data science is multi-disciplinary. It takes the 'best' bits from fields such as statistics, data visualisation, programming, and experimental design (to name a few). As such, data science projects require a blend of these skills. These are hands-on notes in which you will learn these skills by conducting research projects using real-world data. This means that you will:

- obtain and clean relevant datasets;
- develop your own research questions;
- use statistical techniques to answer those questions; and

- communicate your results in a meaningful way.

These notes were developed in collaboration with professional data scientists as well as academics from a variety of fields. They are designed around approaches that are used extensively in academia, government, and industry. Furthermore, they include many aspects, such as data cleaning and communication, that are critical, but rarely taught. However, these notes do not contain everything that you need. Your learning must be 'active' when using these notes because that is the way you will continue to learn through the rest of your life and career. You need to seek out additional information, critically evaluate it, and apply it to your situation.

The workflow that we follow in these notes is:

1. Research question development.
2. Data collection.
3. Data cleaning.
4. Exploratory data analysis.
5. Statistical modelling.
6. Evaluation.
7. Communication.
8. Reproduce.

All of these aspects are critical to being able to convince a reader of your story. Your ability to convince them of your story depends on the quality of all aspects of your workflow.

If we were to expand on this workflow then we roughly get the chapters that are covered in these notes, although they are re-ordered as necessary. From the first chapter we will have a workflow (make a graph then write about it convincingly) that allows us to tell a convincing (albeit likely basic) story. In each subsequent chapter we add aspects and depth to our workflow that will allow us to speak with increasing sophistication and credibility.

This workflow also aligns nicely with the skills that are sought in data scientists. For instance, Mango Solutions, a UK data science consultancy, describes 'the six core capabilities of data scientists' as: 1. communicator; 2. data-wrangler; 3. programmer; 4. technologist; 5. modeller; and 6. visualiser (**?**).

These notes are also designed to enable you to build a portfolio of work that you could show to a potential employer. This is arguably the most important thing that you should be doing. (**?**, p. 55) describe a portfolio as 'a set of data science projects that you can show to people so they can see what kind of data science work you can do'. They describe this as a 'step [that] can really help you be successful'.

1.4.1 Software

The software that we use in these notes is R (**?**). This language was chosen because it is open-source, widely used, general enough to cover the entire workflow, yet specific enough to have plenty of the tools that we need for statistical analysis built in. We do not assume that you have used R before, and so another reason for selecting R for these notes is the community of R users which is, in general, especially welcoming of new-comers and there are a lot of great beginner-friendly materials available.

If you don't have a programming language, then R is a great one to start with. If you have a preferred programming language already, then it wouldn't hurt to pick up R as well. That said, if you have a good reason to prefer another open source programming language (for instance you use Python daily at work) then you may wish to stick with that. However, all examples in these notes are in R.

Please download R and R Studio onto your own computer. You can download R for free here: http://cran.utstat.utoronto.ca/, and you can download R Studio Desktop for free here: https://rstudio.com/products/rstudio/download/#download.

Please also create an account on R Studio Cloud: https://rstudio.cloud/. This will allow you to run R in the cloud, which will be helpful when we are getting started.

1.4.2 Assumed background

These notes assume familiarity with first-year statistics. For instance, if you have a taken a course or two where you covered hypothesis testing and similar concepts then that should be enough. That said, enthusiasm and interest can take you pretty far, so if you've got those then don't worry about too much else.

1.4.3 Structure

These notes are structured around a fairly dense 12-week course. Each chapter contains a list of required reading, as well as a list of recommended reading for those who are interested in the topic and want a starting place for further exploration. All chapters contain a summary of the key concepts and skills that are developed in that chapter. Code and technical chapters additionally contain a list of the main packages and functions that are used in the chapter. Many of the chapters also have a pre-quiz. This is a short quiz that you should complete after doing the required readings, but before going through the chapter to test your knowledge. After completing the chapter, you should go back through the lists and the pre-quiz to make sure that you understand each aspect.

There are problem sets throughout these notes. These are opportunities for you to conduct your own research on a topic that is of interest to you. Although the initial problem sets require you to use data from the Toronto Open Data

Portal (https://open.toronto.ca/), after those first few you are able to use any appropriate dataset. Although open-ended research may be new to you, the extent to which you are able to develop your own questions, use quantitative methods to explore them, and communicate your story to a reader, is the true measure of the success of these notes.

1.5 Acknowledgements

Many people gave generously of their time, code, and data to help develop these notes.

Thank you to Monica Alexander, Michael Chong, and Sharla Gelfand for allowing their code to be used.

Thank you to Kelly Lyons, Hareem Naveed, and Periklis Andritsos for helpful comments.

These notes have greatly benefited from the notes and teaching materials of others that are freely available online, especially:

- Chris Bail's *Text as Data*;
- Andrew Heiss's *Program Evaluation for Public Service*;
- Grant McDermott's *Data Science for Economists*;
- David Mimno's *Text Mining for History and Literature*
- Ed Rubin's *PhD Econometrics (III)* and *Introduction to Econometrics (II)*;

Thank you to the following students who identified specific improvements in these notes: Aaron Miller, Amy Farrow, Cesar Villarreal Guzman, Faria Khandaker, Hong Shi, Mounica Thanam, and Wijdan Tariq.

Finally, thank you to the Winter 2020 and 2021 INF2178 and Fall 2020 Term STA304 students at the University of Toronto, whose feedback greatly improved all aspects.

1.6 Contact

Any comments or suggestions on these notes would be welcomed. You can contact me: rohan.alexander@utoronto.ca.

Chapter 2

Drinking from a fire hose

Required reading

- Barrett, Malcolm, 2021, 'Data science as an atomic habit', 16 January, https://malco.io/2021/01/04/data-science-as-an-atomic-habit/.
- Bryan, Jennifer and Jim Hester, 2020, *What they Forgot to Teach You About R*, Chapters 1 to 5, https://rstats.wtf/debugging-r-code.html.
- Gelfand, Sharla, 2019, 'Cleaning up after the federal election', https://sharla.party/talk/2019-10-24-uoft-brown-bag/.
- Hoa, Karen, 2019, 'This is how AI bias really happens - and why it's so hard to fix, *MIT Technology Review*, 4 February, https://www.technologyreview.com/s/612876/this-is-how-ai-bias-really-happensand-why-its-so-hard-to-fix/.
- Keyes, Os, 2019, 'Counting the Countless', *Real Life*, 8 April, freely available at: https://reallifemag.com/counting-the-countless/.
- Wickham, Hadley, and Garrett Grolemund, 2017, *R for Data Science*, Chapters 3 - 6, 8, 10, 11, 13, 14, 15, and 18, https://r4ds.had.co.nz/.

Required viewing

- Kuriwaki, Shiro, 2020, 'Defining Custom Functions in R', *Vimeo*, 2 February, https://vimeo.com/388825332.
- Register, Yim, 2020, 'Data Science Ethics in 6 Minutes', *YouTube*, 29 December, https://youtu.be/mA4gypAiRYU.

Alternative reading

There are a lot of great alternative 'getting started with R' type materials. Depending on your background and interests you may find some of the following useful:

- Arnold, Taylor, and Lauren Tilton, 2015, *Humanities Data in R*, Springer, Chapters 1 to 5.

- Hall, Megan, 2019, 'An Introduction to R With Hockey Data', https://hockey-graphs.com/2019/12/11/an-introduction-to-r-with-hockey-data/.
- Hanretty, Chris, 2020, 'ConveRt', slides http://chrishanretty.co.uk/conveRt/#1.
- Phillips, Nathaniel D., 2018, *YaRrr! The Pirate's Guide to R*, Chapter 2, https://bookdown.org/ndphillips/YaRrr/started.html.

Recommended reading

- Alexander, Monica, 2019, 'The concentration and uniqueness of baby names in Australia and the US', https://www.monicaalexander.com/posts/2019-20-01-babynames/.
- Hvitfeldt, Emil, 2020, 'Emoji in ggplot2', https://www.hvitfeldt.me/blog/real-emojis-in-ggplot2/.
- Pavlik, Kaylin, 2018, 'Dairy Queen Deserts in Minnesota', https://www.kaylinpavlik.com/dairy-queen-deserts/.
- 'R Studio Cloud Guide', https://rstudio.cloud/learn/guide.
- Scherer, Cédric, 2019, 'Best TidyTuesday 2019', https://cedricscherer.netlify.com/2019/12/30/best-tidytuesday-2019/.
- Silge, Julia, 2019, 'Reordering and facetting for ggplot2', https://juliasilge.com/blog/reorder-within/.
- Smale, David, 2019, 'Happy Days', https://davidsmale.netlify.com/portfolio/happy-days/.

Key libraries

- `ggplot2`
- `tidyverse`

Key concepts/skills/etc

- R is fun and allows you to accomplish really interesting projects.
- But like any language it is a slow path to mastery.
- The way to learn is to start with a really small project in mind, and break down the steps required to achieve it. Look at other people's code to work out how you might deal with the steps. Copy, paste, and modify someone else's code to achieve each step. Don't worry about perfection, just worry about achieving each step. Complete that project and move onto the next project. Rinse and repeat. Each project you'll get a little better.
- Tibbles
- Importing data
- Joining data
- Strings
- Factors
- Dates
- Pivot

Key functions

- `%>%` 'pipe'
- `dplyr::arrange()`
- `dplyr::filter()`
- `dplyr::group_by()`
- `dplyr::mutate()`
- `dplyr::select()`
- `dplyr::summarise()`
- `dplyr::ungroup()`
- `ggplot::facet_wrap()`
- `ggplot::geom_histogram()`
- `class()`
- `dplyr::case_when()`
- `ggplot::facet_wrap()`
- `ggplot::geom_density()`
- `ggplot::geom_histogram()`
- `ggplot::geom_point()`
- `janitor::clean_names()`
- `skimr::skim()`
- `tidyr::pivot_longer()`
- `tidyr::pivot_wider()`

Quiz

1. If I had a dataset with the following columns: `name`, `age` and wanted to focus on `name`, then which verb should I use (pick one)?
   a. `tidyverse::select()`.
   b. `tidyverse::mutate()`.
   c. `tidyverse::filter()`.
   d. `tidyverse::rename()`.
2. If I want to cite R then how do I find a recommended citation (pick one)?
   a. `cite('R')`.
   b. `cite()`.
   c. `citation('R')`.
   d. `citation()`.
3. According to Register, 2020, data decisions affect (pick one)?
   a. Real people.
   b. No one.
   c. Those in the training set.
   d. Those in the test set.
4. In your own words, what is data science?
5. According to Keyes, 2019, what is perhaps a more accurate accurate definition of data science (pick one)?
   a. 'The inhumane reduction of humanity down to what can be counted.';
   b. 'The quantitative analysis of large amounts of data for the purpose of decision-making.';
   c. 'Data science is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights

from many structural and unstructured data.'

6. Imagine that you have a job in which including 'race' as an explanatory variable improves the performance of your model. What types of issues would you consider when deciding whether to include this variable in production? What if the variable was sexuality? Please be sure to refer to Mullainathan, 2019, in your answer.
7. What are three advantages of R? What are three disadvantages?
8. What is R Studio?
    a. An integrated development environment (IDE)
    b. A closed source paid program
    c. A programming language created by Guido van Rossum
    d. A statistical programming language
9. What is R?
    a. A open source statistical programming language
    b. A programming language created by Guido van Rossum
    c. A closed source statistical programming language
    d. An integrated development environment (IDE)
10. Which of the following are not tidyverse verbs (pick one)?
    a. select().
    b. filter().
    c. arrange().
    d. mutate().
    e. visualize().
11. If I wanted to make a new column which verb should I use (pick one)?
    a. select().
    b. filter().
    c. arrange().
    d. mutate().
    e. visualize().
12. If I wanted to focus on particular rows which verb should I use (pick one)?
    a. select().
    b. filter().
    c. arrange().
    d. mutate().
    e. summarise()
13. If I wanted a summary of the data that gave me the mean by sex, which two verbs should I use (pick one)?
    a. summarise().
    b. filter().
    c. arrange().
    d. mutate().
    e. group_by().
14. What are the three key aspects of the grammar of graphics (select all)?
    a. data.
    b. aesthetics.
    c. type.

   d. geom_histogram().
15. What is not one of the four challenges for mitigating bias mentioned in Hao 2019 (pick one)?
   a. Unknown unknowns.
   b. Imperfect processes.
   c. The definitions of fairness.
   d. Lack of social context.
   e. Disinterest given profit considerations.
16. What would be the output of `class("edward")` (pick one)?
   a. "numeric".
   b. "character".
   c. "data.frame".
   d. "vector".
17. How can I simulate 10,000 draws from a normal distribution with a mean of 27 and a standard deviation of 3 (pick one)?
   a. `rnorm(10000, mean = 27, sd = 3)`.
   b. `rnorm(27, mean = 10000, sd = 3)`.
   c. `rnorm(3, mean = 10000, sd = 27)`.
   d. `rnorm(27, mean = 3, sd = 1000)`.

2.1 Hello world

To jump in we will get some data from the wild, make a graph with it, and then use this to tell a story. Some of the code may be a bit unfamiliar to you if it's your first-time using R. It'll all soon be familiar. But the only way to learn how to code is to code. Please try to get this working on your own computer, typing out (not copy/pasting) all the code that you need. It's important and normal to realise that you're going to be bad at this for a while.

> Whenever you're learning a new tool, for a long time, you're going to suck… But the good news is that is typical; that's something that happens to everyone, and it's only temporary.
>
> Hadley Wickham as quoted by **?**.

One of the great things about graphs is that sometimes this is all you need to have a convincing story, as Figure **??** from **?** show.

In this section we are going to focus on making a table and a graph from our data. Although you will be guided thoroughly to achieve this, hopefully by seeing the power of quantitative analysis with R you will be motivated to stick with it when you run into difficulties later on.

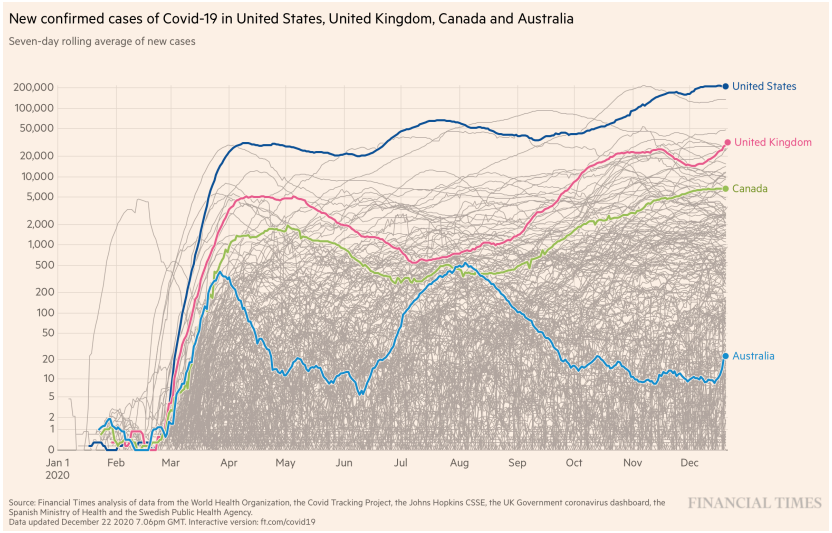


Figure 2.1: New confirmed cases of Covid-19 in United States, United Kingdom, Canada and Australia, as at 22 December 2020.

Figure 2.2: The County Election (1851–52), by George Caleb Bingham (American, 1811 - 1879), as downloaded from https://artvee.com/dl/the-county-election.

2.2 Case study - Canadian elections

2.2.1 Getting started

To get started you should open a new R Markdown file (File -> New File -> R Markdown). As this is our first attempt at using R in the wild, we will just have everything in the one R Markdown document. (In later projects we will move to a more robust set-up.) Then you should create a new R code chunk (keyboard shortcut: Command + Option + I) and add some preamble documentation. I like to specify the purpose of the document, who the author is and their contact details, when the file was written or last updated, and pre-requisites that the file relies on. You may also like to include a license, and list outstanding issues or todos. Remember that in R, lines that start with '#' are comments - they won't run.

```
#### Preamble ####
# Purpose: Read in voting data from the 2019 Canadian Election and output a
# dataset that can be used for analysis.
# Author: Rohan Alexander
# Email: rohan.alexander@utoronto.ca
# Date: 9 January 2019
# Prerequisites: Need the text file from the Canadian elections website
# Issues:
# To do:
```

After this I typically set-up my workspace. This usually involves installing and/or reading in any packages, and possibly updating them. Remember that you only need to install a package once for each computer. But you need to call it every time you want to use it. (Here I've added excessive comments so that you know what is going on and why - in general I wouldn't explain what tidyverse is.)

```
#### Workspace set-up ####
install.packages("tidyverse") # Only need to do this once
install.packages("janitor") # Only need to do this once
install.packages("here")
```

In this case we are going to use `tidyverse` ?, `janitor` ?, and `here` ?.

```
#### Workspace set-up ####
# tidyverse is a collection of packages
# Try ?tidyverse to see more
library(tidyverse) # Calls the tidyverse - need to do this each time.
library(janitor) # janitor helps us clean datasets
library(here) # here helps us to know where files are
# update.packages() # You can uncomment this if you want to update your packages.
```

2.2.2 Get the data

We read in the dataset from the Elections Canada website. We can actually pass a website to the `read_tsv()` function, which saves a lot of time.

```
#### Read in the data ####
# Read in the data using read_tsv from the readr package (part of the tidyverse)
# The '<-' is assigning the output of readr::read_tsv to a object called raw_data.
raw_elections_data <- readr::read_tsv(file = "http://enr.elections.ca/DownloadResults.a
                                      skip = 1)
# There is some debris on the first line so we skip them.
# We have read the data from the Elections Canada website. We may like to save
# it just in case something happens and they move it.
write_csv(raw_elections_data, here("inputs/data/canadian_voting.csv"))
```

(Note that Elections Canada updates this link with the latest elections. When I run this on 31 December 2020, I get the results of a Toronto by-election. While I'll certainly update these notes from time to time, it may be that there's been an election between now and when you run these notes, so your specific results may be slightly different.)

2.2.3 Clean the data

Now we'd like to clean the data so that we can use it.

```
#### Basic cleaning ####
raw_elections_data <- read_csv(here("inputs/data/canadian_voting.csv"))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   `Electoral district number - Numéro de la circonscription` = col_character(),
##   `Electoral district name` = col_character(),
##   `Nom de la circonscription` = col_character(),
##   `Type of results*` = col_character(),
##   `Type de résultats**` = col_character(),
##   `Surname - Nom de famille` = col_character(),
##   `Middle name(s) - Autre(s) prénom(s)` = col_logical(),
##   `Given name - Prénom` = col_character(),
##   `Political affiliation` = col_character(),
##   `Appartenance politique` = col_character(),
##   `Votes obtained - Votes obtenus` = col_double(),
##   `% Votes obtained - Votes obtenus %` = col_double(),
##   `Rejected ballots - Bulletins rejetés***` = col_double(),
##   `Total number of ballots cast - Nombre total de votes déposés` = col_double()
## )
```

```
# If you called the library (as we did) then you don't need to use this set-up
# of janitor::clean_names, you could just use clean_names, but I'm making it
# explicit here, but won't in the future.
cleaned_elections_data <- janitor::clean_names(raw_elections_data)
# One thing to notice for those who have a Stata background is that we just
# overwrote the name - that's fine in R.

# The pipe operator - %>% - pushes the output from one line to be an input to the
# next line.
cleaned_elections_data <-
  cleaned_elections_data %>%
  # Filter to only have certain rows
  filter(type_of_results == "validated") %>%
  # Select only certain columns
  select(electoral_district_number_numero_de_la_circonscription,
         electoral_district_name,
         political_affiliation,
         surname_nom_de_famille,
         percent_votes_obtained_votes_obtenus_percent
         ) %>%
  # Rename the columns to be a bit shorter
  rename(riding_number = electoral_district_number_numero_de_la_circonscription,
         riding = electoral_district_name,
         party = political_affiliation,
         surname = surname_nom_de_famille,
         votes = percent_votes_obtained_votes_obtenus_percent)

head(cleaned_elections_data)
```

```
## # A tibble: 6 x 5
##   riding_number riding         party                     surname     votes
##   <chr>         <chr>          <chr>                     <chr>       <dbl>
## 1 35108         Toronto Centre People's Party - PPC      Bawa          1.1
## 2 35108         Toronto Centre Free Party Canada         Cappelletti   0.3
## 3 35108         Toronto Centre NDP-New Democratic Party  Chang        17
## 4 35108         Toronto Centre Independent               Clarke        0.5
## 5 35108         Toronto Centre Liberal                   Ien          42
## 6 35108         Toronto Centre Libertarian               Komar         0.5
```

Finally we may like to save our cleaned dataset.

```
#### Save ####
readr::write_csv(cleaned_elections_data, "outputs/data/cleaned_elections_data.csv")
```

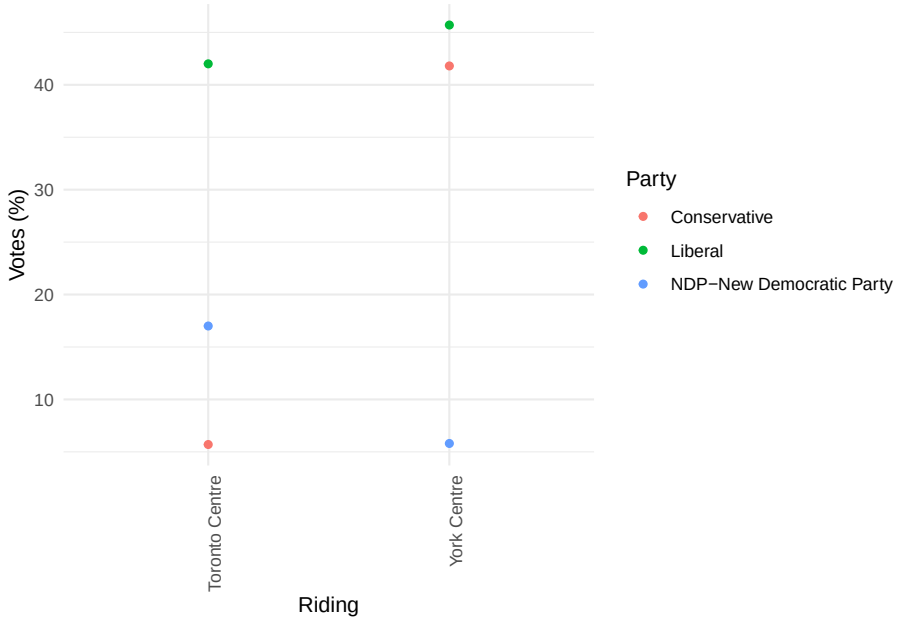
2.2.4 Make a graph

First we need to read in the dataset, we then filter the number of parties to a smaller number.

```
#### Read in the data ####
cleaned_elections_data <-
  readr::read_csv("outputs/data/cleaned_elections_data.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   riding_number = col_double(),
##   riding = col_character(),
##   party = col_character(),
##   surname = col_character(),
##   votes = col_double()
## )
```

```
# Make a graph just considers Toronto riding
cleaned_elections_data %>%
  filter(party %in% c("Bloc Québécois",
                      "Conservative",
                      "Liberal",
                      "NDP-New Democratic Party")
  ) %>%
  ggplot(aes(x = riding, y = votes, color = party)) +
  geom_point() +
  theme_minimal() + # Make the theme neater
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + # Change the angle
  labs(x = "Riding",
       y = "Votes (%)",
       color = "Party")
```




```
# Save the graph
ggsave("outputs/figures/toronto_results.pdf", width = 40, height = 20, units = "cm")
```

2.2.5 Make a table

There are an awful lot of ways to make a table in R. First we'll try the built-in function `summary()`.

```
#### Read in the data ####
cleaned_elections_data <-
  readr::read_csv("outputs/data/cleaned_elections_data.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   riding_number = col_double(),
##   riding = col_character(),
##   party = col_character(),
##   surname = col_character(),
##   votes = col_double()
## )
```

```
#### Make some tables ####
# Try some different default summary table
summary(cleaned_elections_data)
```

```
##  riding_number      riding             party              surname
```

```
##  Min.   :35108   Length:15          Length:15          Length:15
##  1st Qu.:35108   Class :character   Class :character   Class :character
##  Median :35108   Mode  :character   Mode  :character   Mode  :character
##  Mean   :35112
##  3rd Qu.:35118
##  Max.   :35118
##      votes
##  Min.   : 0.20
##  1st Qu.: 0.55
##  Median : 3.60
##  Mean   :13.34
##  3rd Qu.:24.85
##  Max.   :45.70
```

Now we can try a `group_by()` and `summarise()`.

```
# Make our own
cleaned_elections_data %>%
  # Using group_by and summarise means that whatever summary statistics we
  # construct will be on a party basis. We could group_by multiple variables and
  # similarly, we could create a bunch of different other summary statistics.
  group_by(party) %>%
  summarise(min = min(votes),
            mean = mean(votes),
            max = max(votes))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 9 x 4
##   party                       min  mean   max
##   <chr>                     <dbl> <dbl> <dbl>
## 1 Conservative                5.7 23.8   41.8
## 2 Free Party Canada           0.3  0.3    0.3
## 3 Green Party                 2.6 17.7   32.7
## 4 Independent                 0.5  0.55   0.6
## 5 Liberal                    42   43.8   45.7
## 6 Libertarian                 0.5  0.5    0.5
## 7 NDP-New Democratic Party    5.8 11.4   17
## 8 No Affiliation              0.2  0.2    0.2
## 9 People's Party - PPC        1.1  2.35   3.6
```

2.3 Case study - Toronto homelessness

Toronto has a large homeless population, and of course given the pandemic and winter it is critical that there are enough places in shelters. Unfortunately, as we will see in this case study, there are not enough places. However, the one good thing is that we have the data to see this is a problem.

In this case study we are going to use data on the number of people in Toronto shelters to make a graph of usage. This will also introduce us to Tidy Tuesday! In my experience, people are most successful at 'learning R' when they are learning it to achieve something else. If you're in a university course then that might be 'pass the course', but often it's nice to have some other projects. TidyTuesday is a weekly event in which the R community comes together around a dataset and shares code and approaches. You can learn more about it here: https://github.com/rfordatascience/tidytuesday.

2.3.1 Getting started

Again, open a new R Markdown file: (File -> New File -> R Markdown). Update the details so that they reflect your own.

Again, add some top matter with some comments and explanations of the code.

```
#### Preamble ####
# Purpose: Read in Toronto homelessness data and output a graph.
# Author: Rohan Alexander
# Email: rohan.alexander@utoronto.ca
# Date: 22 December 2020
# Prerequisites:
# Issues:
# To do:
```

I want to talk a little about the libraries this time. Libraries are bits of code that other people have written. There are a few common ones that you'll see regularly, especially the `tidyverse`. To use a package we first have to install it and then we need to load it. Jenny Bryan has a wonderful analogy of installing a lightbulb - `install.packages("tidyverse")`. You only need to do this once, but then if you want light then you need to turn on the switch - `library(tidyverse)`. So because we installed everything earlier we won't need to do it again, we can just call the library.

```
#### Workspace set-up ####
library(tidyverse)
```

Given that a lot of people gave up their time to make `R` and the packages, it's important to cite them. Luckily, it's easy to get the information that you need to properly cite them.

```
# To get the citation for R run:
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
```

```
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
```

```
# And to get the citation for a package run that function with the package name. For i
citation('tidyverse')
```

```
##
##   Wickham et al., (2019). Welcome to the tidyverse. Journal of Open
##   Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Welcome to the {tidyverse}},
##     author = {Hadley Wickham and Mara Averick and Jennifer Bryan and Winston Chang a
##     year = {2019},
##     journal = {Journal of Open Source Software},
##     volume = {4},
##     number = {43},
##     pages = {1686},
##     doi = {10.21105/joss.01686},
##   }
```

```
# Again, don't worry too much about the details - we'll get into them later.
```

2.3.2 Get the data

We are going to grab some data that has been made available about Toronto homeless shelters. Again, don't worry too much about the details for now, but what we are saying here, is that there's a CSV that has been made available to us on GitHub and this code downloads it to our own computer. After we download it we can quickly look at the data using `head()`.

```
toronto_shelters <-
  readr::read_csv(
    'https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-12-01/sh
    )
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   id = col_double(),
##   occupancy_date = col_datetime(format = ""),
##   organization_name = col_character(),
##   shelter_name = col_character(),
##   shelter_address = col_character(),
##   shelter_city = col_character(),
##   shelter_province = col_character(),
##   shelter_postal_code = col_character(),
##   facility_name = col_character(),
##   program_name = col_character(),
##   sector = col_character(),
##   occupancy = col_double(),
##   capacity = col_double()
## )
```

```
head(toronto_shelters)
```

```
## # A tibble: 6 x 13
##      id occupancy_date      organization_na~ shelter_name shelter_address
##   <dbl> <dttm>              <chr>            <chr>        <chr>
## 1     1 2017-01-01 00:00:00 COSTI Immigrant~ COSTI Recep~ 100 Lippincott~
## 2     2 2017-01-01 00:00:00 Christie Ossing~ Christie Os~ 973 Lansdowne ~
## 3     3 2017-01-01 00:00:00 Christie Ossing~ Christie Os~ 973 Lansdowne ~
## 4     4 2017-01-01 00:00:00 Christie Refuge~ Christie Re~ 43 Christie St~
## 5     5 2017-01-01 00:00:00 City of Toronto  Birchmount ~ 1673 Kingston ~
## 6     6 2017-01-01 00:00:00 City of Toronto  Birkdale Re~ 1229 Ellesmere~
## # ... with 8 more variables: shelter_city <chr>, shelter_province <chr>,
## #   shelter_postal_code <chr>, facility_name <chr>, program_name <chr>,
## #   sector <chr>, occupancy <dbl>, capacity <dbl>
```

2.3.3 Make a graph

The dataset is on a daily basis for each shelter. What I'd like to do is to get an overall picture of the availability of shelter places in Toronto. This code is based on code by Florence Vallée-Dubois[1] and Lisa Lendway[2].

[1] https://github.com/florencevdubois/MyTidyTuesdays/blob/master/2020.12.01.R

[2] https://github.com/llendway/tidy_tuesday_in_thirty/blob/main/2020_12_01_tidy_tuesday.Rmd

Now, I'd like to first do some data manipulation before we graph it this time. I'm going to introduce a few new functions here that we'll see a lot more soon. Again, don't worry if this doesn't all make sense right now. You're learning a brand-new language! Just try to focus on picking up a word or two and staying motivated!

Finally, because it's Christmas, we can grab a seasonally-appropriate theme from my colleague Liza Bolton.

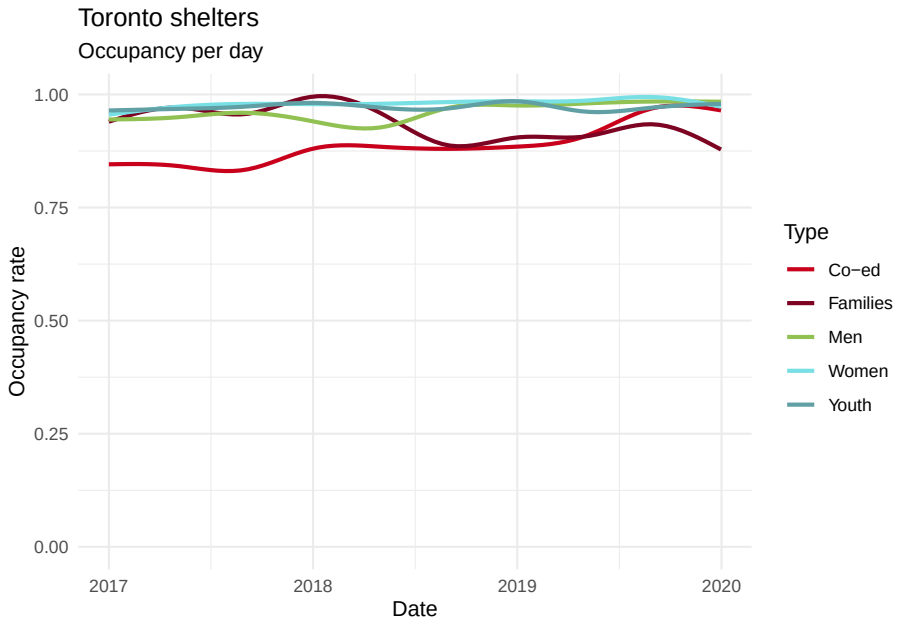
```
# In contrast to the earlier packages, which were located in a central repository
# of packages called CRAN, Liza's is available on her GitHub. Again, don't worry
# about the details for now, it'll all be clarified later.
devtools::install_github("elb0/decemberLB", ref = "main")
```

```
## Skipping install of 'decemberLB' from a github remote, the SHA1 (2cc38a25) has not c
##   Use `force = TRUE` to force installation
```

```
# Once it's installed we call the library as usual.
library(decemberLB)

toronto_shelters %>%
  tidyr::drop_na(occupancy, capacity) %>% # We only want rows that have data
  group_by(occupancy_date, sector) %>% # We want to know the occupancy by date and sec
  summarise(the_sum = sum(occupancy),
            the_capacity = sum(capacity),
            the_usage = the_sum / the_capacity, .groups = 'drop') %>%
  ggplot(aes(x = occupancy_date, y = the_usage, color = sector)) +
  geom_smooth(aes(group = sector), se = FALSE) +
  scale_y_continuous(limits = c(0, NA)) +
  labs(color = "Type",
       x = "Date",
       y = "Occupancy rate",
       title = "Toronto shelters",
       subtitle = "Occupancy per day") +
  theme_minimal() +
  scale_color_december(palette = "xmas")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



2.4 R essentials

This section is the basics of using R. Some of it may not make sense at first, but these are commands that we will come back to throughout these notes. You should initially just go through this chapter quickly, noting aspects that you don't understand. Then start to play around with some of the initial case studies. Then maybe come back to this chapter. That way you will see how the various bits fit into context, and hopefully be more motivated to pick up various aspects. We will come back to everything in this chapter in more detail at some point in these notes.

R is an open source language that is useful for statistical programming

You can download R for free here: http://cran.utstat.utoronto.ca/, and you can download R Studio Desktop for free here: https://rstudio.com/products/rstudio/download/#download.

When you are using R you will run into trouble at some point. To work through that trouble:

1. Look at the help file for the function by putting ? before the function e.g. `?pivot_wider`.
2. Check the class of your data, by `class(data_set$data_column)`.
3. Check for typos.
4. Google the error.
5. Google what you are trying to do.

6. Restart R (`Session -> Restart R and Clear Output`).
7. Try to make a small example and see if you have the same issues.
8. Restart your computer.

The past ten years or so of R have been characterised by the rise of the tidyverse. This is '… an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.' **?**. There are three distinctions here: the original R language, typically referred to as 'base'; the 'tidyverse' which is a collection of packages that build on top of the R language; and other packages.

Pretty much everything that you can do in the tidyverse, you can also do in base. However, as the tidyverse was built especially for modern data science it is usually easier to use the tidyverse, especially when you are setting out. Additionally, pretty much everything that you can do in the tidyverse, you can also do with other packages. However, as the tidyverse is a coherent collection of packages, it is often easier to use the tidyverse, especially when you are setting out. Eventually you will start to see cases where it makes sense to trade-off the convenience and coherence of the tidyverse for some features of base or other packages. Indeed you'll see that at various points in these notes. For instance, the tidyverse can be slow, and so if you need to import thousands of CSVs then it can make sense to switch away from `read_csv()`. That is great and the appropriate use of base and non-tidyverse packages, rather than dogmatic insistence on a solution, is a sign of your development as an applied statistician.

Get started by loading the `tidyverse` package.

```
library(tidyverse)
```

The general workflow that we will use involves:

1. Import
2. Tidy
3. Transforming, descriptive
4. Plot
5. Model
6. Repeat 3/4

People like Keyes have tried to tell us this for a long time, but COVID-19 make it very clear to everyone - most of the data that we use will have humans at the heart of it. It's vitally important that you keep that in mind and grapple with it in everything that you do with R. It can be really easy to forget that almost every point in our dataset is likely a person.

2.5 Social impact

> "We shouldn't have to think about the societal impact of our work because it's hard and other people can do it for us" is a really bad

> argument. I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore. But basically all facial recognition work would not get published if we took Broader Impacts sections seriously. There is almost no upside and enormous downside risk. To be fair though i should have a lot of humility here. For most of grad school I bought in to the myth that science is apolitical and research is objectively moral and good no matter what the subject is.
>
> Joe Redmon, 20 February 2020.

Although the term 'data science' is ubiquitous in academia, industry, and even more generally, it is difficult to define. One deliberately antagonistic definition of data science is '[t]he inhumane reduction of humanity down to what can be counted' (**?**). While purposefully controversial, this definition highlights one reason for the increased demand for data science and quantitative methods over the past decade—individuals and their behaviour are now at the heart of it. Many of the techniques have been around for many decades, but what makes them popular now is this human focus.

Unfortunately, even though much of the work may be focused on individuals, issues of privacy and consent, and ethical concerns more broadly, rarely seem front of mind. While there are some exceptions, in general, even at the same time as claiming that AI, machine learning, and data science are going to revolutionise society, consideration of these types of issues appears to have been largely treated as something that would be nice to have, rather than something that we may like to think of before we embrace the revolution.

For the most part, these are not new issues. In the sciences, there has been considerable recent ethical consideration around CRISPR technology and gene editing, but in an earlier time similar conversations were had, for instance, about Wernher von Braun being allowed to building rockets for the US. In medicine, of course, these concerns have been front-of-mind for some time. Data science seems determined to have its own Tuskegee syphilis experiment moment rather than think about and deal appropriately with these issues, based on the experiences of other fields, before they occur.

That said, there is some evidence that data scientists are beginning to be more concerned about the ethics surrounding the practice. For instance, NeurIPS, the most prestigious machine learning conference, now requires a statement on ethics to accompany all submissions.

> In order to provide a balanced perspective, authors are required to include a statement of the potential broader impact of their work, including its ethical aspects and future societal consequences. Authors should take care to discuss both positive and negative outcomes.
>
> NeurIPS call for papers, as accessed 26 February 2020.

Ethical considerations will be mentioned throughout these notes rather than clumped in one easily ignorable part that can be thrown away after 'ethics week'. The purpose is not to prescriptively rule things in or out, but to provide an opportunity to raise some issues that should be front of mind. The variety of data science applications, the relative youth of the field, and the speed of change, mean that ethical considerations can sometimes be set aside when it comes to data science. This is in contrast to fields such as science, medicine, engineering, and accounting where there is a long history. Nonetheless it can helpful to think through some ethical considerations that you may encounter in the content of a usual data science project.

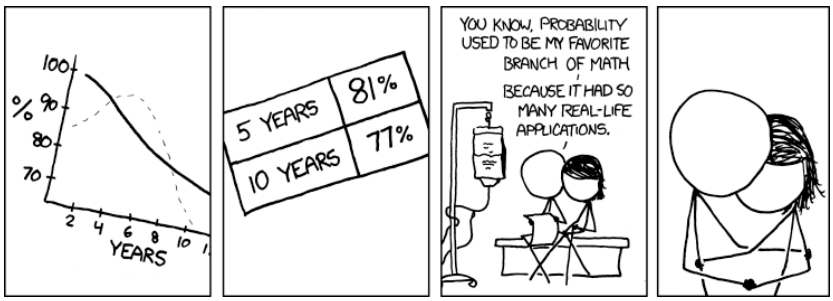


Figure 2.3: Probability, from https://xkcd.com/881/.

2.6 R, R Studio, and R Studio Cloud

My colleague Liza Bolton has a lovely analogy here on the relationship between R and R Studio which I really like. R is like a car engine and R Studio is like the car. Although some of us can use a car engine directly, most of us use a car to interact with the engine.

2.6.1 R

R - https://www.r-project.org/ - is an open source and free programming language that is focused on general statistics. (Free in this context doesn't refer to a price of zero, but instead to 'freedom', but it also does have a price of zero). This is in contrast with a open source programming language that is designed for general purpose, such as Python, or an open source programming language that is focused on probability, such as Stan. It was created by Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand. It is maintained by the R Core Team and changes to this 'base' of code occur methodically and with concern given to a variety of different priorities.

If you are in Canada then you can download R here: http://cran.utstat.utoronto.ca/, if you are in Australia then you can download R here: https://cran.csiro.au/, otherwise you should go here - https://cran.r-

project.org/mirrors.html - and find a location that suits you. (It doesn't really matter where you get it from, it's just that it may be slightly faster to use a closer option.)

Many people build on this stable base, to extend the capabilities of R to better and more quickly suit their needs. They do this by creating packages. Typically, although not always, a package is a collection R code, and this allows you to more easily do things that you want to do. These packages are managed by the Comprehensive R Archive Network (CRAN) - https://cran.r-project.org/, and other repositories. CRAN is built into the download of R that you just got, so you can use it straight away.

If you want to use a package then you need to firstly install it in your computer, and then you need to load it when you want to use it. Di Cook, who is a Professor of Business Analytics at Monash University in Australia, describes this as analogous to a lightbulb: if you want light in your house, first you need to screw in the lightbulb, and you need to turn the switch on. You only need to screw in the lightbulb once per house, but you need to turn the switch on every time you want to use the light.

To install a package on your computer (again, you'll need to do this only once per computer) you use the code:

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

Then when you want to use a package, you need to call it with this code:

```
library(tidyverse)
```

You can open R and use it on your computer. It is primarily designed to be interacted with through the command line. This is how I had to start with R, and it's fine, but it can be useful to have a richer environment than the command line provides. In particular, it can be useful to install an Integrated Development Environment (IDE), which is an application that brings together various bits and pieces that you'll use all the time. The one that we will use is R Studio.

2.6.2 R Studio

R Studio is distinct to R and they are different entities. R Studio builds on top of R to make it easier for you to use R. This is in the same way that you can use the internet from the command line, but most of us use a browser such as Chrome, Firefox, or Safari.

R Studio is free in the sense that you don't pay anything for it. It is also free in the sense of being able to take the code, modify it, and distribute that code provided others are similarly allowed to take your code and modify it and distribute, etc. However, it is important to recognise that R Studio is an entity and so it is possible that in the future the current situation could change.

You can download R Studio here: https://rstudio.com/products/rstudio/download/#download.

When you open R Studio it will look like Figure **??**.

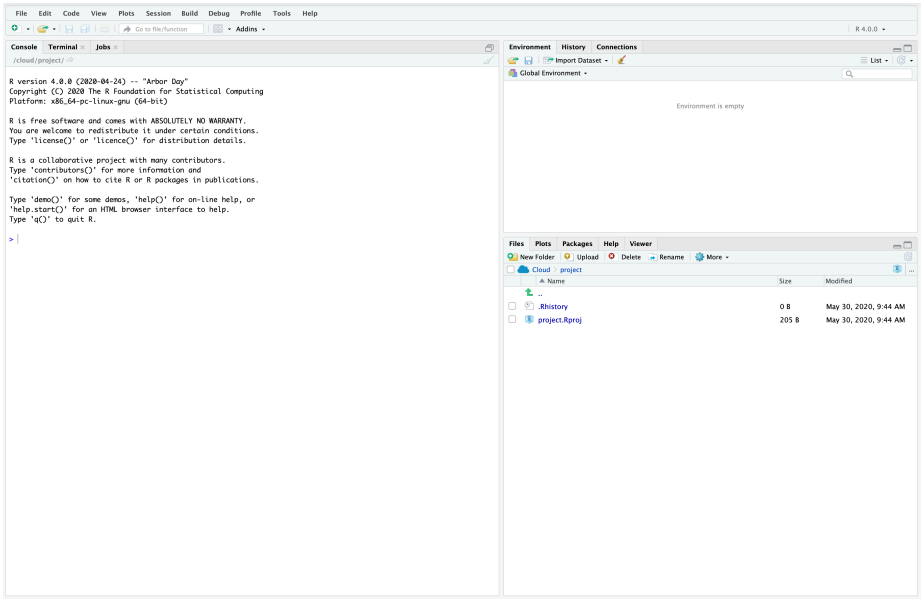


Figure 2.4: Opening R Studio for the first time

The left pane is a console in which you can type and execute R code line by line. Try it with 2+2 by clicking next to the prompt '>' and typing that out then pressing enter. The code that you type should be:

```
2 + 2
```

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

And hopefully you get the same answer printed in the console.

The pane on the top right has information about your environment. For instance, when we create variables a list of their names and some properties will appear there. Try to type the following code, replacing my name with your name, next to the prompt, and again press enter:

```
my_name <- "Rohan"
```

You should notice a new value in the environment pane with the variable name and its value.

The pane in the bottom right is a file manager. At the moment it should just have two files - an R History file and a R Project file. We'll get to what these are later, but for now we will create and save a file.

Type out the following code (don't worry too much about the details for now):

```
saveRDS(object = my_name, file = "my_first_file.rds")
```

And you should see a new '.rds' file in your list of files.

2.6.3 R Studio Cloud

While you can download R Studio to your own computer, initially we will us R Studio Cloud, which is an online version that is provided by R Studio. We will use this so that you can focus on getting comfortable with R and R Studio in an environment that is consistent. This way you don't have to worry about what computer you have or installation permissions while you are still getting used to the basics.

The R Studio Cloud - https://rstudio.cloud/ - is as easy as it gets in terms of moving to the cloud. The trade-off is that it is not very powerful and it is sometimes slow, but for the purposes of the initial sections of these notes that will be fine.

To get started, go to https://rstudio.cloud/ and create an account. If you are going to be a student for a while then it might be worthwhile using a university email account, because although they don't yet charge for it, they will probably start charging soon, but with some luck they will offer education discounts.

Once you have an account and log in, then it should look something like Figure **??**.

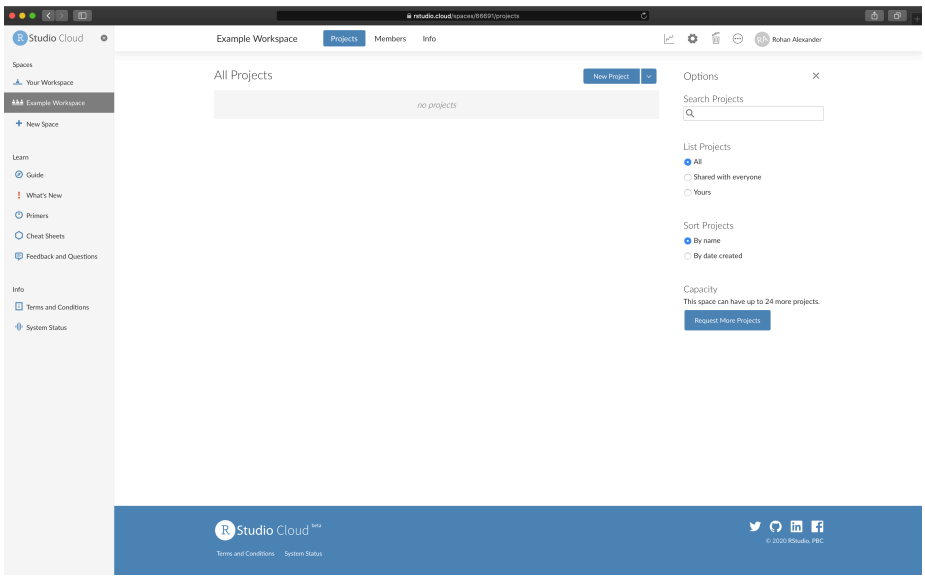


Figure 2.5: Opening R Studio Cloud for the first time

(You'll be in 'Your Workspace', and you won't have a 'Example Workspace'.) From here you should start a 'New Project'. You can give the project a name by clicking on 'Untitled Project' and replacing it. We can now use R Studio in the cloud.

While working line-by-line in the console is fine, it is easier to write out a whole script that can then be executed. We will do this by making an R Script. To do this go to: File -> New File -> R Script, or use the shortcut Command + Shift + N. The console pane will fall to the bottom left and an R Script will open in the top left. Let's write some code that will grab all of the Australian politicians and then construct a small table about the genders of the prime ministers.

(Some of this code won't make sense at this stage, but just type it all out to get in the habit and then run it, by selecting all of the code and clicking 'Run' (or using the keyboard shortcut: Command + Return)

```
# Install the packages that we'll need
install.packages("devtools")
install.packages("tidyverse")
```

```
# Load the packages that we need to use this time
library(devtools)
library(tidyverse)

# Grab the data on Australian politicians
install_github("RohanAlexander/AustralianPpoliticians")

# Make a table of the counts of genders of the prime ministers
AustralianPoliticians::all %>%
  as_tibble() %>%
  count(gender, wasPrimeMinister)
```

```
## # A tibble: 4 x 3
##   gender wasPrimeMinister     n
##   <chr>             <int> <int>
## 1 female                1     1
## 2 female               NA   235
## 3 male                  1    29
## 4 male                 NA  1511
```

You can save your R Script as 'my_first_r_script.R' using File -> Save As (or the keyboard shortcut: Command + S). When you're done your workspace should look something like Figure **??**.

One thing to be aware of is that each R Studio Cloud workspace is essentially a new computer. Because of this, you'll need to install any package that you want to use for each workspace. For instance, before you can use the tidyverse, you need to install.packages("tidyverse"). This is in contrast to when you use your own computer.

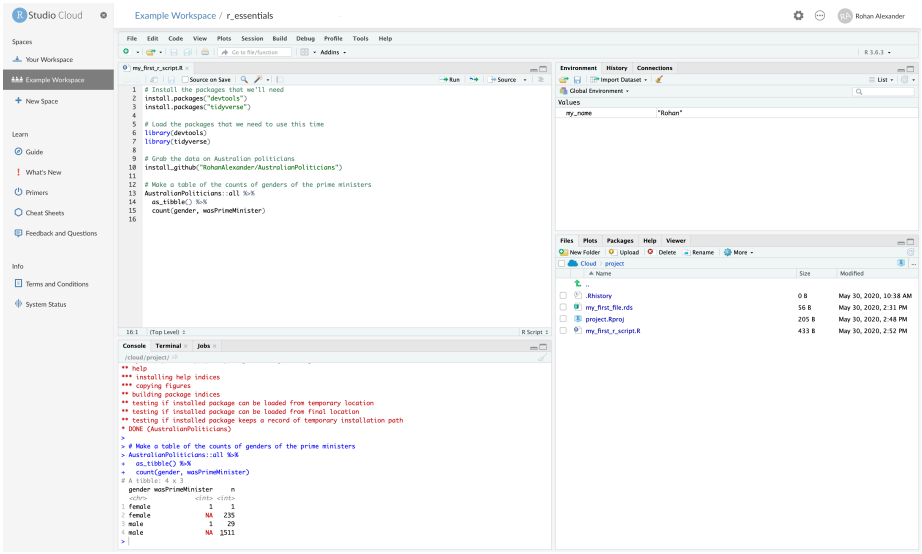


Figure 2.6: After running an R Script

A few final notes on R Studio Cloud for you to keep in the back of your mind:

1) In the Australian politicians example we got our data from the website GitHub, but you can get data into your workspace from your local computer in a variety of ways. One way is to use the 'upload' button in the Files panel.
2) R Studio Cloud allows some degree of collaboration. For instance, you can give someone else access to a workspace that you create. This could be useful for collaborating on an assignment, although it is not quite full featured yet and you cannot both be in the workspace at the same time (in contrast to, say, Google Docs).
3) There are a variety of weaknesses of R Studio Cloud, in particular at the moment there is a 1GB limit on RAM. Additionally, it is still underdeveloped and things break from time to time. The R Studio Community page that is focused on R Studio Cloud can be helpful sometimes: https://community.rstudio.com/c/rstudio-cloud.

2.7 Tidyverse I

Aspects of 'Tidyverse I' were written with Monica Alexander.

One of the key packages that we use in these notes is the `tidyverse` ?. The `tidyverse` is actually a package of packages (i.e. when you install `tidyverse`, you are actually installing a whole bunch of different packages). The key package in the `tidyverse` in terms of manipulating data is `dplyr` ?, and the key package

in the `tidyverse` in terms of creating graphs is `ggplot2` **?**.

In this section we are going to cycle through some essentials from the Tidyverse. You'll come back to the functions in this section regularly.

I want to keep this section self-contained, so let's start by installing the `tidyverse` (again, to use Di Cook's analogy, this is the equivalent of screwing in the light-bulb). If you just did it, then you don't need to do it again.

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

Now we can load the `tidyverse` (again, to use Di Cook's analogy, the equivalent of turning on the light-switch).

```
library(tidyverse)
```

Here we are going to download the data about Australian politicians using the function `read_csv()`.

```
australian_politicians <-
  read_csv(
    file =
      "https://raw.githubusercontent.com/RohanAlexander/telling_stories_with_data/maste
    )
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   birthDate = col_date(format = ""),
##   birthYear = col_double(),
##   deathDate = col_date(format = ""),
##   member = col_double(),
##   senator = col_double(),
##   wasPrimeMinister = col_double()
## )
## i Use `spec()` for the full column specifications.
```

We will now cover the pipe and six functions that are useful to know and that we will use all the time:

- `select()`
- `filter()`
- `arrange()`
- `mutate()`
- `summarise()/summarize()`
- `group_by()`

2.7.1 The pipe

One key tidyverse helper is the 'pipe': `%>%`. Read it as "and then" (keyboard shortcut: Command + Shift + M). This takes the output of a line of code and uses it as an input to the next line of code. You don't have to use it, but it tends to make your code more readable.

The idea of the pipe is that you take your dataset, **and then**, do something to it. In this case, we will look at the first few lines of our dataset by piping `australian_politicians` through to the `head()` function.

```
australian_politicians %>%
  head()
```

```
## # A tibble: 6 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
## 1 Abbott1~ Abbott  Richard Hart~ Richard   <NA>       Abbott, Ri~
## 2 Abbott1~ Abbott  Percy Phipps  Percy     <NA>       Abbott, Pe~
## 3 Abbott1~ Abbott  Macartney     Macartney Mac        Abbott, Mac
## 4 Abbott1~ Abbott  Charles Lydi~ Charles   Aubrey     Abbott, Au~
## 5 Abbott1~ Abbott  Joseph Palmer Joseph    <NA>       Abbott, Jo~
## 6 Abbott1~ Abbott  Anthony John  Anthony   Tony       Abbott, To~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

2.7.2 Selecting

The `select()` function is used to get a particular column of a dataset. For instance, we might like to select the first names column.

```
australian_politicians %>%
  select(firstName) %>%
  head()
```

```
## # A tibble: 6 x 1
##   firstName
##   <chr>
## 1 Richard
## 2 Percy
## 3 Macartney
## 4 Charles
## 5 Joseph
## 6 Anthony
```

In R, there are many ways to do things. Another way to get a particular column of a dataset is to use the dollar sign. This is from base R (as opposed

to `select()` which is from the `tidyverse` package).

```
australian_politicians$firstName %>%
  head()
```

```
## [1] "Richard"   "Percy"     "Macartney" "Charles"   "Joseph"    "Anthony"
```

The two are almost equivalent and differ only in the class of what they return (we'll talk more about class later in the notes).

For the sake of completeness, if you combine `select()` with `pull()` then you will get the same class of output as if you use the dollar sign.

```
australian_politicians %>%
  select(firstName) %>%
  pull() %>%
  head()
```

```
## [1] "Richard"   "Percy"     "Macartney" "Charles"   "Joseph"    "Anthony"
```

You can also use `select` to get rid of columns, by selecting in a negative sense.

```
australian_politicians %>%
  select(-firstName)
```

```
## # A tibble: 1,776 x 19
##    uniqueID surname allOtherNames commonName displayName earlierOrLaterN~ title
##    <chr>    <chr>   <chr>         <chr>      <chr>       <chr>            <chr>
##  1 Abbott1~ Abbott  Richard Hart~ <NA>       Abbott, Ri~ <NA>             <NA>
##  2 Abbott1~ Abbott  Percy Phipps  <NA>       Abbott, Pe~ <NA>             <NA>
##  3 Abbott1~ Abbott  Macartney     Mac        Abbott, Mac <NA>             <NA>
##  4 Abbott1~ Abbott  Charles Lydi~ Aubrey     Abbott, Au~ <NA>             <NA>
##  5 Abbott1~ Abbott  Joseph Palmer <NA>       Abbott, Jo~ <NA>             <NA>
##  6 Abbott1~ Abbott  Anthony John  Tony       Abbott, To~ <NA>             <NA>
##  7 Abel1939 Abel    John Arthur   <NA>       Abel, John  <NA>             <NA>
##  8 Abetz19~ Abetz   Eric          <NA>       Abetz, Eric <NA>             <NA>
##  9 Adams19~ Adams   Judith Anne   <NA>       Adams, Jud~ nee Bird         <NA>
## 10 Adams19~ Adams   Dick Godfrey~ <NA>       Adams, Dick <NA>             <NA>
## # ... with 1,766 more rows, and 12 more variables: gender <chr>,
## #   birthDate <date>, birthYear <dbl>, birthPlace <chr>, deathDate <date>,
## #   member <dbl>, senator <dbl>, wasPrimeMinister <dbl>, wikidataID <chr>,
## #   wikipedia <chr>, adb <chr>, comments <chr>
```

Finally, you can select, based on conditions. For instance, selecting all all of the columns that start with something, for instance, 'birth'.

```
australian_politicians %>%
  select(starts_with("birth"))
```

```
## # A tibble: 1,776 x 3
##    birthDate  birthYear birthPlace
```

```
##    <date>         <dbl> <chr>
##  1 NA              1859 Bendigo
##  2 1869-05-14      1869 Hobart
##  3 1877-07-03      1877 Murrurundi
##  4 1886-01-04      1886 St Leonards
##  5 1891-10-18      1891 North Sydney
##  6 1957-11-04      1957 London
##  7 1939-06-25      1939 Sydney
##  8 1958-01-25      1958 Stuttgart
##  9 1943-04-11      1943 Picton
## 10 1951-04-29      1951 Launceston
## # ... with 1,766 more rows
```

2.7.3 Filtering

The `filter()` function is used to get particular rows from a dataset. For instance, we might like to filter to only politicians that became prime minister.

```
australian_politicians %>%
  filter(wasPrimeMinister == 1)
```

```
## # A tibble: 30 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Abbott1~ Abbott  Anthony John  Anthony   Tony       Abbott, To~
##  2 Barton1~ Barton  Edmund        Edmund    <NA>       Barton, Ed~
##  3 Bruce18~ Bruce   Stanley Melb~ Stanley   <NA>       Bruce, Sta~
##  4 Chifley~ Chifley Joseph Bened~ Joseph    Ben        Chifley, B~
##  5 Cook1860 Cook    Joseph        Joseph    <NA>       Cook, Jose~
##  6 Curtin1~ Curtin  John Joseph ~ John      <NA>       Curtin, Jo~
##  7 Deakin1~ Deakin  Alfred        Alfred    <NA>       Deakin, Al~
##  8 Fadden1~ Fadden  Arthur Willi~ Arthur    Arthur     Fadden, Ar~
##  9 Fisher1~ Fisher  Andrew        Andrew    <NA>       Fisher, An~
## 10 Forde18~ Forde   Francis Mich~ Francis   Frank      Forde, Fra~
## # ... with 20 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

The `filter()` function also accepts two conditions. For instance, we can look at politicians who were prime minister and were named Joseph.

```
australian_politicians %>%
  filter(wasPrimeMinister == 1 & firstName == "Joseph")
```

```
## # A tibble: 3 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
```

```
##   <chr>    <chr>   <chr>          <chr>     <chr>      <chr>
## 1 Chifley~ Chifley Joseph Bened~ Joseph    Ben        Chifley, B~
## 2 Cook1860 Cook    Joseph         Joseph    <NA>       Cook, Jose~
## 3 Lyons18~ Lyons   Joseph Aloys~ Joseph    <NA>       Lyons, Jos~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

We would get the same result if we use a comma instead of an ampersand.

```
australian_politicians %>%
  filter(wasPrimeMinister == 1, firstName == "Joseph")
```

```
## # A tibble: 3 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>          <chr>     <chr>      <chr>
## 1 Chifley~ Chifley Joseph Bened~ Joseph    Ben        Chifley, B~
## 2 Cook1860 Cook    Joseph         Joseph    <NA>       Cook, Jose~
## 3 Lyons18~ Lyons   Joseph Aloys~ Joseph    <NA>       Lyons, Jos~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

Similarly, we can look at politicians who were named Myles or Ruth.

```
australian_politicians %>%
  filter(firstName == "Ruth" | firstName == "Myles")
```

```
## # A tibble: 3 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>          <chr>     <chr>      <chr>
## 1 Coleman~ Coleman Ruth Nancy     Ruth      <NA>       Coleman, R~
## 2 Ferrick~ Ferric~ Myles Aloysi~ Myles     <NA>       Ferricks, ~
## 3 Webber1~ Webber  Ruth Stephan~ Ruth      <NA>       Webber, Ru~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

We can also pipe the results, for instance, pipe from the `filter()` to `select()`

```
australian_politicians %>%
  filter(firstName == "Ruth" | firstName == "Myles") %>%
  select(firstName, surname)
```

```
## # A tibble: 3 x 2
##   firstName surname
```

```
##   <chr>     <chr>
## 1 Ruth      Coleman
## 2 Myles     Ferricks
## 3 Ruth      Webber
```

Finally, we can `filter()` to a particular row number, for instance, in this case row 853.

```
australian_politicians %>%
  filter(row_number() == 853)
```

```
## # A tibble: 1 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
## 1 Jarman1~ Jarman  Alan William  Alan      <NA>       Jarman, Al~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

But there is also a dedicated function to do this, which is `slice()`

```
australian_politicians %>%
  slice(853)
```

```
## # A tibble: 1 x 20
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
## 1 Jarman1~ Jarman  Alan William  Alan      <NA>       Jarman, Al~
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

2.7.4 Arranging

We can change the order of the dataset based on the values in a particular column using the `arrange()` function. For instance, we may like to arrange the data by year of birth.

```
australian_politicians %>%
  arrange(surname)
```

```
## # A tibble: 1,776 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Abbott1~ Abbott  Richard Hart~ Richard   <NA>       Abbott, Ri~
##  2 Abbott1~ Abbott  Percy Phipps  Percy     <NA>       Abbott, Pe~
##  3 Abbott1~ Abbott  Macartney     Macartney Mac        Abbott, Mac
```

```
##  4 Abbott1~ Abbott  Charles Lydi~ Charles   Aubrey     Abbott, Au~
##  5 Abbott1~ Abbott  Joseph Palmer Joseph    <NA>       Abbott, Jo~
##  6 Abbott1~ Abbott  Anthony John  Anthony   Tony       Abbott, To~
##  7 Abel1939 Abel    John Arthur   John      <NA>       Abel, John
##  8 Abetz19~ Abetz   Eric          Eric      <NA>       Abetz, Eric
##  9 Adams19~ Adams   Judith Anne   Judith    <NA>       Adams, Jud~
## 10 Adams19~ Adams   Dick Godfrey~ Dick      <NA>       Adams, Dick
## # ... with 1,766 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

We can also use the `desc()` function to arrange in descending order.

```
australian_politicians %>%
  arrange(desc(surname))
```

```
## # A tibble: 1,776 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Zimmerm~ Zimmer~ Trent Moir    Trent     <NA>       Zimmerman,~
##  2 Zeal1830 Zeal    William Aust~ William   <NA>       Zeal, Will~
##  3 Zappia1~ Zappia  Antonio       Antonio   Tony       Zappia, To~
##  4 Zammit1~ Zammit  Paul John     Paul      <NA>       Zammit, Pa~
##  5 Zakharo~ Zakhar~ Alice Olive   Alice     Olive      Zakharov, ~
##  6 Zahra19~ Zahra   Christian Jo~ Christian <NA>       Zahra, Chr~
##  7 Young19~ Young   Harold Willi~ Harold    <NA>       Young, Har~
##  8 Young19~ Young   Michael Jero~ Michael   Mick       Young, Mick
##  9 Young19~ Young   Terry James   Terry     <NA>       Young, Ter~
## 10 Yates18~ Yates   George Edwin  George    Gunner     Yates, Gun~
## # ... with 1,766 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

We can also arrange based on more than one column.

```
australian_politicians %>%
  arrange(firstName, surname)
```

```
## # A tibble: 1,776 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Blain18~ Blain   Adair Macali~ Adair     <NA>       Blain, Ada~
##  2 Armstro~ Armstr~ Adam Alexand~ Adam      Bill       Armstrong,~
##  3 Bandt19~ Bandt   Adam Paul     Adam      <NA>       Bandt, Adam
```

```
##  4 Dein1889 Dein    Adam Kemball  Adam      Dick       Dein, Dick
##  5 Ridgewa~ Ridgew~ Aden Derek    Aden      <NA>       Ridgeway, ~
##  6 Bennett~ Bennett Adrian Frank  Adrian    <NA>       Bennett, A~
##  7 Gibson1~ Gibson  Adrian        Adrian    <NA>       Gibson, Ad~
##  8 Wynne18~ Wynne   Agar          Agar      <NA>       Wynne, Agar
##  9 Roberts~ Robert~ Agnes Robert~ Agnes     <NA>       Robertson,~
## 10 Bird1906 Bird    Alan Charles  Alan      <NA>       Bird, Alan
## # ... with 1,766 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

We can pipe `arrange()` to another `arrange()`.

```
australian_politicians %>%
  arrange(firstName) %>%
  arrange(surname)
```

```
## # A tibble: 1,776 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Abbott1~ Abbott  Anthony John  Anthony   Tony       Abbott, To~
##  2 Abbott1~ Abbott  Charles Lydi~ Charles   Aubrey     Abbott, Au~
##  3 Abbott1~ Abbott  Joseph Palmer Joseph    <NA>       Abbott, Jo~
##  4 Abbott1~ Abbott  Macartney     Macartney Mac        Abbott, Mac
##  5 Abbott1~ Abbott  Percy Phipps  Percy     <NA>       Abbott, Pe~
##  6 Abbott1~ Abbott  Richard Hart~ Richard   <NA>       Abbott, Ri~
##  7 Abel1939 Abel    John Arthur   John      <NA>       Abel, John
##  8 Abetz19~ Abetz   Eric          Eric      <NA>       Abetz, Eric
##  9 Adams19~ Adams   Dick Godfrey~ Dick      <NA>       Adams, Dick
## 10 Adams19~ Adams   Judith Anne   Judith    <NA>       Adams, Jud~
## # ... with 1,766 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

It is just important to be clear about the precedence of each.

```
australian_politicians %>%
  arrange(surname, firstName)
```

```
## # A tibble: 1,776 x 20
##    uniqueID surname allOtherNames firstName commonName displayName
##    <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
##  1 Abbott1~ Abbott  Anthony John  Anthony   Tony       Abbott, To~
##  2 Abbott1~ Abbott  Charles Lydi~ Charles   Aubrey     Abbott, Au~
```

```
##  3 Abbott1~ Abbott  Joseph Palmer Joseph    <NA>       Abbott, Jo~
##  4 Abbott1~ Abbott  Macartney     Macartney Mac        Abbott, Mac
##  5 Abbott1~ Abbott  Percy Phipps  Percy     <NA>       Abbott, Pe~
##  6 Abbott1~ Abbott  Richard Hart~ Richard   <NA>       Abbott, Ri~
##  7 Abel1939 Abel    John Arthur   John      <NA>       Abel, John
##  8 Abetz19~ Abetz   Eric          Eric      <NA>       Abetz, Eric
##  9 Adams19~ Adams   Dick Godfrey~ Dick      <NA>       Adams, Dick
## 10 Adams19~ Adams   Judith Anne   Judith    <NA>       Adams, Jud~
## # ... with 1,766 more rows, and 14 more variables: earlierOrLaterNames <chr>,
## #   title <chr>, gender <chr>, birthDate <date>, birthYear <dbl>,
## #   birthPlace <chr>, deathDate <date>, member <dbl>, senator <dbl>,
## #   wasPrimeMinister <dbl>, wikidataID <chr>, wikipedia <chr>, adb <chr>,
## #   comments <chr>
```

2.7.5 Grouping

We can group variables using the function `group_by()` and then apply some other function within those groups. For instance, we could arrange by first name within gender, and then get the first three results.

```
australian_politicians %>%
  group_by(gender) %>%
  arrange(firstName) %>%
  slice(1:3)
```

```
## # A tibble: 6 x 20
## # Groups:   gender [2]
##   uniqueID surname allOtherNames firstName commonName displayName
##   <chr>    <chr>   <chr>         <chr>     <chr>      <chr>
## 1 Roberts~ Robert~ Agnes Robert~ Agnes     <NA>       Robertson,~
## 2 MacTier~ MacTie~ Alannah Joan~ Alannah   <NA>       MacTiernan~
## 3 Zakharo~ Zakhar~ Alice Olive   Alice     Olive      Zakharov, ~
## 4 Blain18~ Blain   Adair Macali~ Adair     <NA>       Blain, Ada~
## 5 Armstro~ Armstr~ Adam Alexand~ Adam      Bill       Armstrong,~
## 6 Bandt19~ Bandt   Adam Paul     Adam      <NA>       Bandt, Adam
## # ... with 14 more variables: earlierOrLaterNames <chr>, title <chr>,
## #   gender <chr>, birthDate <date>, birthYear <dbl>, birthPlace <chr>,
## #   deathDate <date>, member <dbl>, senator <dbl>, wasPrimeMinister <dbl>,
## #   wikidataID <chr>, wikipedia <chr>, adb <chr>, comments <chr>
```

2.7.6 Mutating

The `mutate()` function is used to make a new column. For instance, perhaps we want to make a new column that is 1 if a person was a member and a senator and 0 otherwise.

```
australian_politicians <-
  australian_politicians %>%
  mutate(was_both = if_else(member == 1 & senator == 1, 1, 0))

australian_politicians %>%
  select(member, senator, was_both) %>%
  head()
```

```
## # A tibble: 6 x 3
##   member senator was_both
##    <dbl>   <dbl>    <dbl>
## 1      0       1        0
## 2      1       1        1
## 3      0       1        0
## 4      1       0        0
## 5      1       0        0
## 6      1       0        0
```

2.7.7 Summarise

The function `summarise()` is used to create new summary variables. For instance, looking at the maximum of birth year to find who the most recently born politicians are.

```
australian_politicians %>%
  summarise(youngest_politicians_birth_year = max(birthYear, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   youngest_politicians_birth_year
##                             <dbl>
## 1                            1994
```

And we can check that using `arrange()`.

```
australian_politicians %>%
  arrange(-birthYear) %>%
  select(uniqueID, surname, allOtherNames, birthYear) %>%
  slice(1:3)
```

```
## # A tibble: 3 x 4
##   uniqueID       surname     allOtherNames    birthYear
##   <chr>          <chr>       <chr>                <dbl>
## 1 SteeleJohn1994 Steele-John Jordon Alexander      1994
## 2 Chandler1990   Chandler    Claire               1990
## 3 Roy1990        Roy         Wyatt Beau           1990
```

The `summarise()` function is particularly powerful in conjunction with `group_by()`. For instance, let's look at the year of birth of the youngest by

gender.

```
australian_politicians %>%
  group_by(gender) %>%
  summarise(youngest_politician_birth_year = max(birthYear, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   gender youngest_politician_birth_year
##   <chr>                           <dbl>
## 1 female                           1990
## 2 male                             1994
```

Let's look at mean of age at death by gender.

```
australian_politicians %>%
  mutate(days_lived = deathDate - birthDate) %>%
  filter(!is.na(days_lived)) %>%
  group_by(gender) %>%
  summarise(mean_days_lived = round(mean(days_lived), 2)) %>%
  arrange(-mean_days_lived)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   gender mean_days_lived
##   <chr>  <drtn>
## 1 female 28857.30 days
## 2 male   27372.89 days
```

We can use `group_by()` for more than one group for instance, looking again at average number of days lived by gender and by which house.

```
australian_politicians %>%
  mutate(days_lived = deathDate - birthDate) %>%
  filter(!is.na(days_lived)) %>%
  group_by(gender, wasPrimeMinister) %>%
  summarise(mean_days_lived = round(mean(days_lived), 2)) %>%
  arrange(-mean_days_lived)
```

```
## `summarise()` regrouping output by 'gender' (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
## # Groups:   gender [2]
##   gender wasPrimeMinister mean_days_lived
##   <chr>             <dbl> <drtn>
## 1 female               NA 28857.30 days
## 2 male                  1 28446.61 days
## 3 male                 NA 27345.20 days
```

2.7.8 Counting

We can use the function `count()` to create counts by groups. For instance, the number of politicians by gender.

```
australian_politicians %>%
  group_by(gender) %>%
  count()
```

```
## # A tibble: 2 x 2
## # Groups:   gender [2]
##   gender     n
##   <chr>  <int>
## 1 female   236
## 2 male    1540
```

2.7.9 Proportions

Finally, often calculating proportions is a combination of `summarise()` and `mutate()` (and `group_by()`).

Let's calculate the proportion of genders.

Note here, that we needed to `ungroup()` the data before mutating.

```
australian_politicians %>%
  group_by(gender) %>%
  count() %>%
  ungroup() %>%
  mutate(prop = n/(sum(n)))
```

```
## # A tibble: 2 x 3
##   gender     n  prop
##   <chr>  <int> <dbl>
## 1 female   236 0.133
## 2 male    1540 0.867
```

2.8 Base

2.8.1 Class

A class is the broader type of object that something is. For instance, your class is probably 'human', which is itself a 'animal'. Similarly, if we create a number in R we can use `class()` to work out its class, which in this case will be numeric.

```
my_number <- 8
class(my_number)
```

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

Or we could make it a character.

```
my_name <- "rohan"
class(my_name)
```

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

Finally, we can often coerce classes to be something else.

```
my_number_as_character <- as.character(my_number)
class(my_number_as_character)
```

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

There are many ways for your code to not run, but having an issue with the classes is the almost always the first thing to check.

2.8.2 Simulating data

Simulating data is a key skill for statistics. We will use the following functions all the time: `rnorm()`, `sample()`, and `runi()`. Arguably the most important function is `set.seed()`, which we need because while we want our data to be random, we want it to be repeatable.

Let's get 10 observations from the standard normal.

```
set.seed(853)

number_of_observations <- 10

simulated_data <- tibble(person = c(1:number_of_observations),
                         observation = rnorm(number_of_observations,
                                             mean = 0,
                                             sd = 1)
                         )
```

Then let's add 10 draws from the uniform distribution between 0 and 10.

```
simulated_data$another_observation <- runif(number_of_observations,
                                            min = 0,
                                            max = 10)
```

Finally, let's use sample, which allows use to pick from a list of items, to add a favourite colour to each observation.

```
simulated_data$fav_colour <- sample(x = c("blue", " white "),
                                    size = number_of_observations,
                                    replace = TRUE)
```

We set the option `replace` to `TRUE` because we are only choosing between two items, but we want ten outcomes. Depending on the simulation you should

think about whether you need it `TRUE` or `FALSE`. Also, there is another useful option to adjust the probability with which each item is drawn. In particular, the default is that both options are equally likely, but perhaps we might like to have 10 per cent `blue` with 90 per cent `white`. The way to do this is to set the option `prob`. As always with functions, you can find more in the help with `?sample`.

2.8.3 Functions

There are a lot of functions in R, and almost any common task that you might need to do is likely already done. But you will need to write your own functions. The way to do this is to define a function and give it a name. Your function will probably have some inputs (note that these inputs can have default values). Your function will then do something with these inputs and then return something.

```
my_function <- function(some_names) {
  print(some_names)
}

my_function(c("rohan", "monica"))
```

```
## [1] "rohan"  "monica"
```

2.9 ggplot essentials

The `ggplot` package is the plotting package that is part of the `tidyverse` collection of packages.

In a similar way to piping, it works in layers. But instead of using the pipe (`%>%`) ggplot uses `+`.

2.9.1 Main features

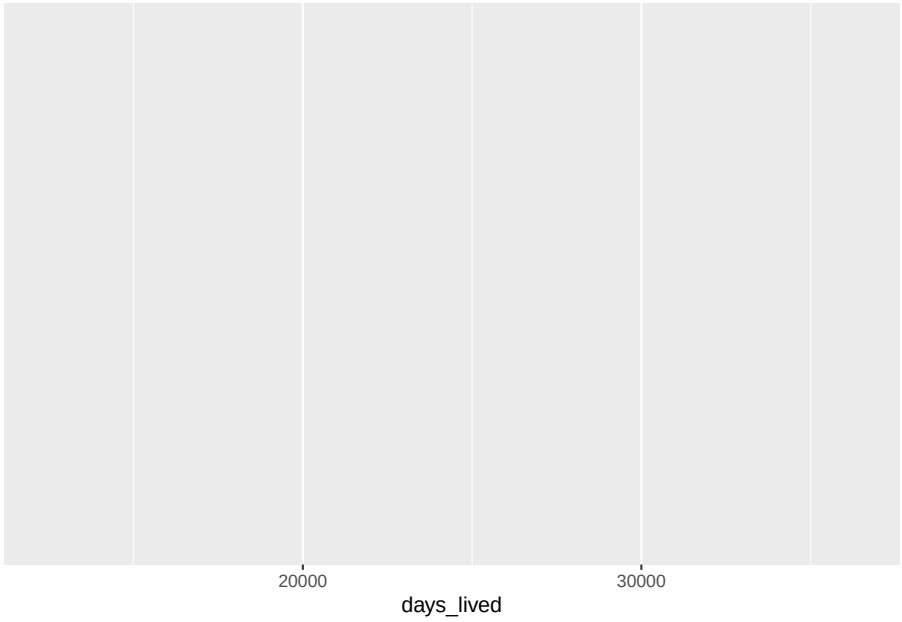
There are three key aspects:

1. data;
2. aesthetics / mapping; and
3. type.

For instances, let's build up a histogram of age of death with increasing complexity.

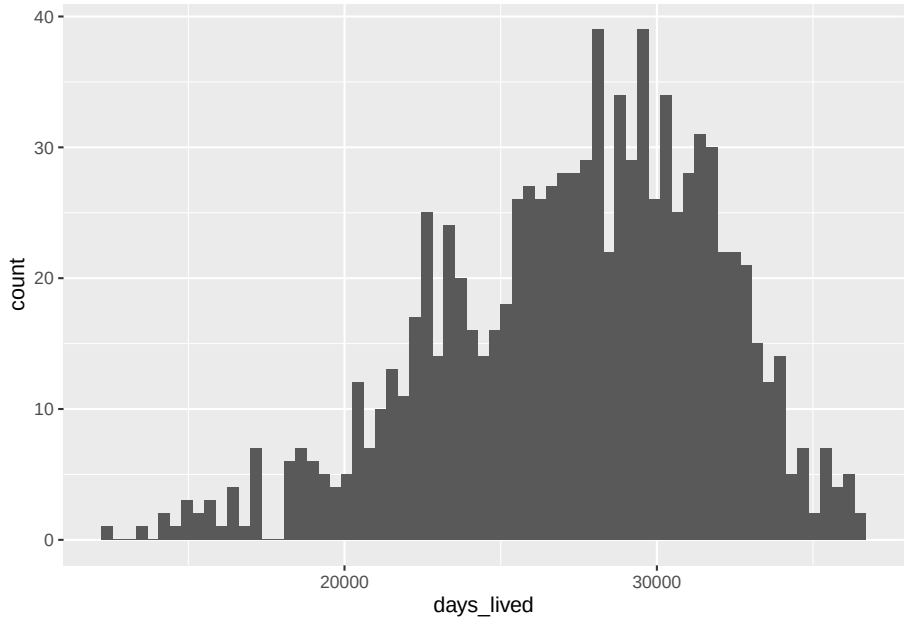
Starts with a grey box:

```
australian_politicians %>%
  mutate(days_lived = as.integer(deathDate - birthDate)) %>%
  filter(!is.na(days_lived)) %>%
  ggplot(mapping = aes(x = days_lived))
```



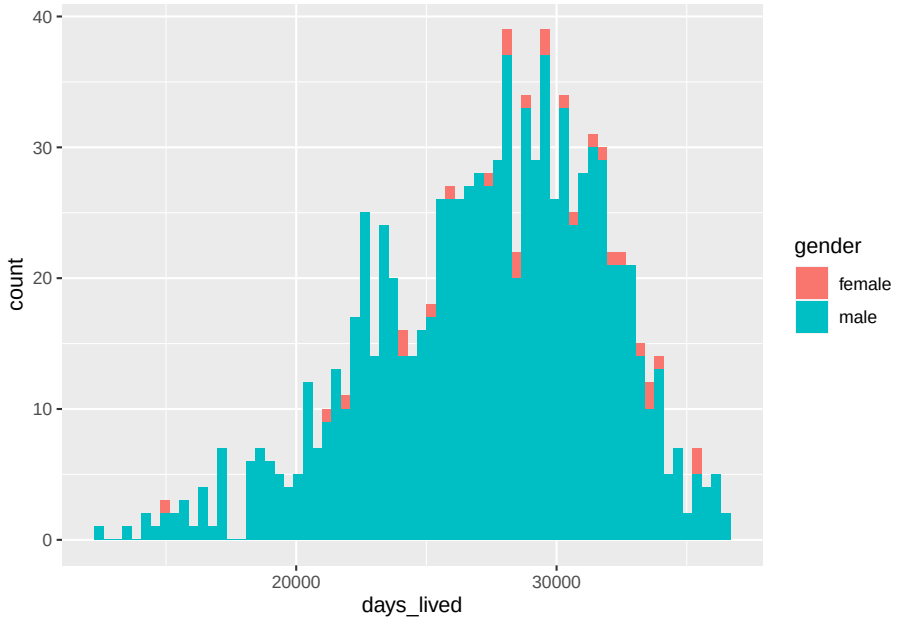
We need to tell it what we want to plot. This is where `geom` comes in

```
australian_politicians %>%
  mutate(days_lived = as.integer(deathDate - birthDate)) %>%
  filter(!is.na(days_lived)) %>%
  ggplot(mapping = aes(x = days_lived)) +
  geom_histogram(binwidth = 365)
```



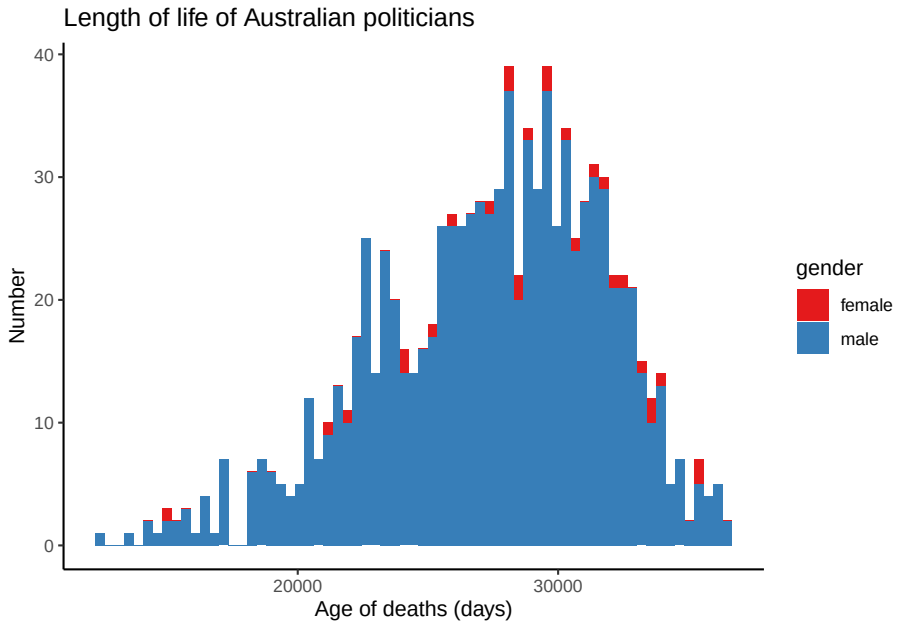
Now let's color the bars by gender, which means adding an aesthetic.

```
australian_politicians %>%
  mutate(days_lived = as.integer(deathDate - birthDate)) %>%
  filter(!is.na(days_lived)) %>%
  ggplot(mapping = aes(x = days_lived, fill = gender)) +
  geom_histogram(binwidth = 365)
```



We can add some labels, change the color, and background.

```
australian_politicians %>%
  mutate(days_lived = as.integer(deathDate - birthDate)) %>%
  filter(!is.na(days_lived)) %>%
  ggplot(mapping = aes(x = days_lived, fill = gender)) +
  geom_histogram(binwidth = 365) +
  labs(title = "Length of life of Australian politicians",
       x = "Age of deaths (days)",
       y = "Number") +
  theme_classic() +
  scale_fill_brewer(palette = "Set1")
```

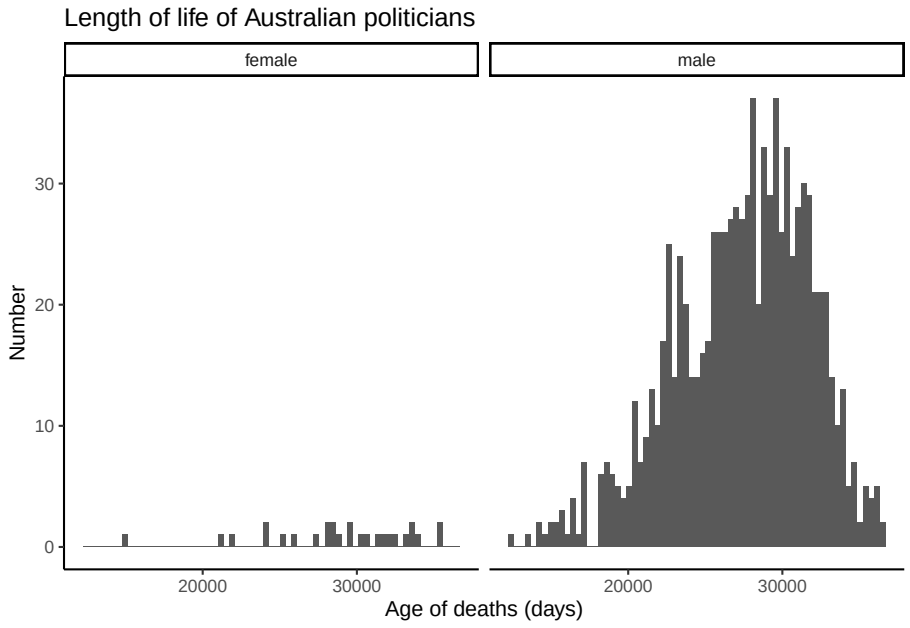


I forget who said this but, '`ggplot` makes it so easy to have nicely labelled axes, there's no real excuse not to'.

2.9.2 Facets

Facets are subplots and are invaluable because they allow you to add another variable to your plot without having to make a 3D plot.

```
australian_politicians %>%
  mutate(days_lived = as.integer(deathDate - birthDate)) %>%
  filter(!is.na(days_lived)) %>%
  ggplot(mapping = aes(x = days_lived)) +
  geom_histogram(binwidth = 365) +
  labs(title = "Length of life of Australian politicians",
       x = "Age of deaths (days)",
       y = "Number") +
  theme_classic() +
  scale_fill_brewer(palette = "Set1") +
  facet_wrap(~gender)
```



2.10 Tidyverse II

2.10.1 Tibbles

A tibble is a data frame, but it is a data frame with some particular changes that make it easier to work with. You should read Chapter 10 of **?** for more detail. The main difference is that compared with a dataframe, a tibble doesn't convert strings to factors, and it prints nicely, including letting you know the class of a column.

You can make a tibble manually if you need, for instance this can be handy for simulating data, but usually we will just import data as a tibble.

```
people <-
  tibble(names = c("rohan", "monica"),
         website = c("rohanalexander.com", "monicaalexander.com"),
         fav_colour = c("blue", " white "),
         )
people
```

```
## # A tibble: 2 x 3
##   names  website             fav_colour
##   <chr>  <chr>               <chr>
## 1 rohan  rohanalexander.com  "blue"
## 2 monica monicaalexander.com " white "
```

2.10.2 Importing data

There are a variety of ways to import data. If you are dealing with CSV files then try `read_csv()` in the first instance. There were examples of that in earlier sections.

2.10.3 Joining data

We can join two datasets together in a variety of ways. The most common join that I use is `left_join()`, where I have one main dataset and I want to join another to it based on some common column names. Here we'll join two datasets based on favourite colour.

```
both <-
  simulated_data %>%
  left_join(people, by = "fav_colour")

both
```

```
## # A tibble: 10 x 6
##    person observation another_observation fav_colour names  website
##     <int>       <dbl>               <dbl> <chr>      <chr>  <chr>
##  1      1     -0.360                9.52  "blue"     rohan  rohanalexander.com
##  2      2     -0.0406               0.586 " white "  monica monicaalexander.com
##  3      3     -1.78                 2.48  "blue"     rohan  rohanalexander.com
##  4      4     -1.12                 5.80  " white "  monica monicaalexander.com
##  5      5     -1.00                 5.26  "blue"     rohan  rohanalexander.com
##  6      6      1.78                 4.09  "blue"     rohan  rohanalexander.com
##  7      7     -1.39                 3.97  "blue"     rohan  rohanalexander.com
##  8      8     -0.497                2.52  " white "  monica monicaalexander.com
##  9      9     -0.558                6.29  "blue"     rohan  rohanalexander.com
## 10     10     -0.824                8.57  "blue"     rohan  rohanalexander.com
```

2.10.4 Strings

We've seen a string earlier, but it is an object that is created with single or double quotes. String manipulation is an entire book in itself, but you should start with the stringr package (**?**).

I'll just cover a few essentials: `stringr::str_detect()`, `stringr::str_replace()`, `stringr::str_squish()`.

```
head(people)
```

```
## # A tibble: 2 x 3
##   names  website             fav_colour
##   <chr>  <chr>               <chr>
## 1 rohan  rohanalexander.com  "blue"
## 2 monica monicaalexander.com " white "
```

```
people <-
  people %>%
  mutate(is_rohan = stringr::str_detect(names, "rohan"),
         make_howlett = stringr::str_replace(website, "alexander", "howlett"),
         fav_colour_trim = stringr::str_squish(fav_colour)
         )

head(people)
```

```
## # A tibble: 2 x 6
##   names  website            fav_colour is_rohan make_howlett     fav_colour_trim
##   <chr>  <chr>              <chr>      <lgl>    <chr>            <chr>
## 1 rohan  rohanalexander.com "blue"     TRUE     rohanhowlett.com blue
## 2 monica monicaalexander.c~ " white "  FALSE    monicahowlett.c~ white
```

2.10.5 Pivot

Datasets tend to be either long or wide. Generally, in the tidyverse, and certainly for ggplot, we need long data. To go from one to the other you can use the `pivot_longer()` and `pivot_wider()` functions.

Let's see an example with some data on whether red team or blue team won a competition in some year.

```
pivot_example_data <-
  tibble(year = c(2019, 2020, 2021),
         blue_team = c(1, 2, 1),
         red_team = c(2, 1, 2))

head(pivot_example_data)
```

```
## # A tibble: 3 x 3
##    year blue_team red_team
##   <dbl>     <dbl>    <dbl>
## 1  2019         1        2
## 2  2020         2        1
## 3  2021         1        2
```

This dataset is in wide format at the moment. To get it into long format, what we'd like is to have a column that specifies the team, and then another that specifies the result. We'll use `tidyr::pivot_longer`.

```
data_pivoted_longer <-
  pivot_example_data %>%
  tidyr::pivot_longer(cols = c("blue_team", "red_team"),
                      names_to = "team",
                      values_to = "position")
```

```
head(data_pivoted_longer)
```

```
## # A tibble: 6 x 3
##    year team      position
##   <dbl> <chr>        <dbl>
## 1  2019 blue_team        1
## 2  2019 red_team         2
## 3  2020 blue_team        2
## 4  2020 red_team         1
## 5  2021 blue_team        1
## 6  2021 red_team         2
```

Occasionally, you'll need to go from long data to wide data. We accomplish this with `tidyr::pivot_wider`.

```
data_pivoted_wider <-
  data_pivoted_longer %>%
  tidyr::pivot_wider(id_cols = c("year", "team"),
                     names_from = "team",
                     values_from = "position")

head(data_pivoted_wider)
```

```
## # A tibble: 3 x 3
##    year blue_team red_team
##   <dbl>     <dbl>    <dbl>
## 1  2019         1        2
## 2  2020         2        1
## 3  2021         1        2
```

2.10.6 Factors

A factor is a string that has an inherent ordering. For instance, the days of the week have an order - Monday, Tuesday, Wednesday,... - which is not alphabetical. Factors feature prominently in base, but they often add more complication than they are worth and so the tidyverse gives them a less prominent role. Nonetheless taking advantage of factors is useful in certain circumstances, for instance when plotting the days of the week we probably want them in the usual ordering than in the alphabetical ordering that would result if we had them as characters. The package that we use to deal with factors is `forcats` (**?**).

Sometimes you will have a character vector and you will want it ordered in a particular way. The default is that a character vector is ordered alphabetically, but you may not want that, for instance, the days of the week would look strange on a graph if they were alphabetically ordered: Friday, Monday, Saturday, Sunday, Thursday, Tuesday, Wednesday!

The way to change the ordering is to change the variable from a character to a factor. I would then use the `forcats` package to specify an ordering by hand. The help page is here: https://forcats.tidyverse.org/reference/fct_relevel.html.

Let's look at a concrete example.

```
my_data <- tibble(all_names = c("Rohan", "Monica", "Edward"))
```

If we plotted this then Edward would be first, because it would be alphabetical. But if instead I want to be first as I am the oldest then we could use `forcats` in the following way.

```
library(forcats) # (BTW you'll probably have to install that one)
library(tidyverse)

my_data <-
  my_data %>%
  mutate(all_names = factor(all_names), # Change to factor
         all_names_releveled = fct_relevel(all_names, "Rohan", "Monica")) # Change the

# Then compare the two
my_data$all_names
```

```
## [1] Rohan  Monica Edward
## Levels: Edward Monica Rohan
```

```
my_data$all_names_releveled
```

```
## [1] Rohan  Monica Edward
## Levels: Rohan Monica Edward
```

2.10.7 Cases

If you need to write a few conditional statements then `case_when` is the way to go.

Let's start with a tibble of dates and pretend that we want to group them into 'pre-1950', '1950-2000', '2000-onwards'

```
case_when_example <-
  tibble(some_dates = c("1909-12-31", "1919-12-31", "1929-12-31", "1939-12-31",
                        "1949-12-31", "1959-12-31", "1969-12-31", "1979-12-31",
                        "1989-12-31", "1999-12-31", "2009-12-31")
         )

case_when_example <-
  case_when_example %>%
  mutate(some_dates = lubridate::ymd(some_dates)
         )
```

```
head(case_when_example)
```

```
## # A tibble: 6 x 1
##   some_dates
##   <date>
## 1 1909-12-31
## 2 1919-12-31
## 3 1929-12-31
## 4 1939-12-31
## 5 1949-12-31
## 6 1959-12-31
```

Now we'll use `dplyr::case_when()` to group these.

```
case_when_example <-
  case_when_example %>%
  mutate(year_group =
           case_when(
             some_dates < lubridate::ymd("1950-01-01") ~ "pre-1950",
             some_dates < lubridate::ymd("2000-01-01") ~ "1950-2000",
             some_dates >= lubridate::ymd("2000-01-01") ~ "2000-onwards",
             TRUE ~ "CHECK ME"
             )
  )

head(case_when_example)
```

```
## # A tibble: 6 x 2
##   some_dates year_group
##   <date>     <chr>
## 1 1909-12-31 pre-1950
## 2 1919-12-31 pre-1950
## 3 1929-12-31 pre-1950
## 4 1939-12-31 pre-1950
## 5 1949-12-31 pre-1950
## 6 1959-12-31 1950-2000
```

We could accomplish this with a series of `if_else` statements, but `case_when` is just a bit cleaner. The only thing to be aware of is that statements are evaluated in order. So as soon as something matches it doesn't continue down the list of conditions. That's why we have that catch-all at the end - if the date doesn't fit any of the earlier conditions then we've got a problem and want to know about it.

Chapter 3

Workflow

Required reading

- Alexander, Monica, 2019, 'Reproducibility in demographic research', https://www.monicaalexander.com/posts/2019-10-20-reproducibility/.
- Bailey, Jack, 2020, 'UK Voting Intention Poll Tracker', https://github.com/jackobailey/poll_tracker.
- Bryan, Jennifer and Jim Hester, 2020, 'What they Forgot to Teach You About R', Chapters 11, 12, and 13, https://rstats.wtf/debugging-r-code.html.
- Bryan, Jenny, 2020, *Happy Git and GitHub for the useR*, Chapters 4, 6, 7, 9, https://happygitwithr.com/.
- Fan, Jean, 'R Style Guide', https://jef.works/R-style-guide/.
- Gelman, Andrew, 2016, 'What has happened down here is the winds have changed', https://statmodeling.stat.columbia.edu/2016/09/21/what-has-happened-down-here-is-the-winds-have-changed/.
- Healy, Kieran, 2020, 'The Kitchen Counter Observatory', 21 May, https://kieranhealy.org/blog/archives/2020/05/21/the-kitchen-counter-observatory/.
- Hill, Alison, 2020, 'How I Teach R Markdown', 28 May, https://alison.rbind.io/post/2020-05-28-how-i-teach-r-markdown/.
- Mostipak, Jesse, 2018, 'So you've been asked to make a reprex', 24 February, https://www.jessemaegan.com/post/so-you-ve-been-asked-to-make-a-reprex/.
- Phillips, Nathaniel D., 2018, *YaRrr! The Pirate's Guide to R*, Chapter 4.3, https://bookdown.org/ndphillips/YaRrr/a-brief-style-guide-commenting-and-spacing.html.
- Taback, Nathan, 2019, 'R Markdown for Class Reports', https://scidesign.github.io/Rmarkdownforclassreports.html.
- Tierney, Nicholas, 2020, *RMarkdown for Scientists*, 9 September, Chapters 7-10, 12-15, https://rmd4sci.njtierney.com.

- Wickham, Hadley, *Advanced R*, Chapter on Style, http://adv-r.had.co.nz/Style.html.
- Wickham, Hadley, and Garrett Grolemund, 2017, *R for Data Science*, Chapter 27, https://r4ds.had.co.nz/.
- Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, Tracy K. Teal, 2017, 'Good enough practices in scientific computing', *PLoS Computational Biology*, 13(6).

Recommended reading

- AirbnbEng, 2016, 'Scaling Knowledge at Airbnb', 25 February, https://medium.com/airbnb-engineering/scaling-knowledge-at-airbnb-875d73eff091.
- Bik, Elisabeth, 2020, 'The Tadpole Paper Mill', *Science Integrity Digest*, 21 February, https://scienceintegritydigest.com/2020/02/21/the-tadpole-paper-mill/.
- Chan, Martin, 2020, 'RStudio Projects and Working Directories: A Beginner's Guide', https://martinctc.github.io/blog/rstudio-projects-and-working-directories-a-beginner's-guide/.
- Daly, Darren, 2020, 'A brief history of medical statistics and its impact on reproducibility', 2 February, https://medium.com/@darren_dahly/a-brief-history-of-medical-statistics-and-its-role-in-reproducibility-23e31082f024.
- Ebert, Philip A. "Bayesian reasoning in avalanche terrain: a theoretical investigation." Journal of Adventure Education and Outdoor Learning 19, no. 1 (2019): 84-95.
- Gertler, Paul, Sebastian Martinez, Patrick Premand, Laura Rawlings, and Christel Vermeersch, 'Impact Evaluation in Practice', Chapters 1, 2.
- Guzey, Alexey, 2020, 'How Life Sciences Actually Work: Findings of a Year-Long Investigation', https://guzey.com/how-life-sciences-actually-work/.
- King, Gary, 'How to Write a Publishable Paper as a Class Project', https://gking.harvard.edu/papers.
- King, Gary, 2006, 'Publication, publication', *PS: Political Science & Politics*, vol. 39, pp. 119-125, https://gking.harvard.edu/files/gking/files/paperspub.pdf.
- Lowndes, Julia S. Stewart, Benjamin D. Best, Courtney Scarborough, Jamie C. Afflerbach, Melanie R. Frazier, Casey C. O'Hara, Ning Jiang, and Benjamin S. Halpern, 2017, 'Our path to better science in less time using open data science tools', Nature ecology & evolution, 1(6).
- Miyakawa, Tsuyoshi, 2020, 'No raw data, no science: another possible source of the reproducibility crisis', *Molecular Brain*, 13, 24. https://doi.org/10.1186/s13041-020-0552-2
- Peek, Ryan, 2020, '10 tips to souping up rmarkdown', 20 February, https://ryanpeek.github.io/2020-02-20-10-tips-to-souping-up-rmarkdown/#S8.
- Riederer, Emily, 2019, 'Resource Round-Up: Reproducible Research Edi-

tion', 29 August, https://emilyriederer.netlify.com/post/resource-round-up-reproducible-research-edition/.
- Riederer, Emily, 2019, 'RMarkdown Driven Development (RmdDD)', 4 May, https://emilyriederer.netlify.com/post/rmarkdown-driven-development/.
- Silver, Nate, 2020, 'We Fixed An Issue With How Our Primary Forecast Was Calculating Candidates' Demographic Strengths', 19 February, https://fivethirtyeight.com/features/we-fixed-a-mistake-in-how-our-primary-forecast-was-calculating-candidates-demographic-strengths/.
- Soetewey, Antoine, 2020, 'Getting started in R markdown', 18 February, https://www.statsandr.com/blog/getting-started-in-r-markdown/.
- Varian, Hal, 'How to Build an Economic Model in Your Spare Time', http://people.ischool.berkeley.edu/~hal/Papers/how.pdf. (This paper was written a while ago for economists, please just ignore the economics specific parts.)
- Wayne, Hillel, 2017, 'How do we trust our science code?', 14 August, https://www.hillelwayne.com/post/how-do-we-trust-science-code/.
- Wayne, Hillel, 2020, 'This is how science happens', 9 March, https://www.hillelwayne.com/post/this-is-how-science-happens/.

Required viewing

- Bryan, Jenny, 'Debugging in R', https://resources.rstudio.com/rstudio-conf-2020/object-of-type-closure-is-not-subsettable-jenny-bryan.

Recommended viewing

- Gelfand, Sharla, 2020, 'Don't repeat yourself, talk to yourself! Repeated reporting in the R universe', 30 January, talk given at rstudio::conf, San Francisco, https://resources.rstudio.com/rstudio-conf-2020/dont-repeat-yourself-talk-to-yourself-repeated-reporting-in-the-r-universe-sharla-gelfand.

Recommended activity

- 'First Contributions' https://github.com/forwards/first-contributions.

Fun song

- The Magnetic Fields, 2016, ''86 How I Failed Ethics', 17 November, https://youtu.be/Hu5dEXZ7DOY (thanks to Paul Hodgetts).

Key concepts/skills/etc

- Restart R often (Session -> Restart R and Clear Output).
- Debugging is a skill, and you will get better at it with time and practice.
- Start with reading the error message.
- Check the class.
- You may get frustrated at times, this is normal.
- There are various tools that can help. Google is your friend.
- Make a small example and try to get the code running on that.

- Cultivating a tenacious mentality may help.
- Writing code that future-you can understand.
- Developing important questions.
- Reproducibility and replicability
- The importance of data and code access
- The importance of version control in a modern scientific workflow.
- The basics of Git and GitHub, as a solo data scientist.

Key GitHub workflow with commands

- Get the latest changes: `git pull`.
- Add your updates: `git add -A`.
- Check on everything: `git status`.
- Commit your changes: `git commit -m "Short description of changes"`.
- Push your changes to GitHub: `git push`.

Quiz

1. What are three features of a good research question (write a paragraph or two)?
2. What is a counterfactual (pick one)?
    a. If-then statements in which the if didn't happen.
    b. If-then statements in which the if happen.
    c. Statements that are either true or false.
    d. Statements that are neither true or false.
3. How do you hide the warnings in a R Markdown R chunk (pick one)?
    a. `echo = FALSE`
    b. `include = FALSE`
    c. `eval = FALSE`
    d. `warning = FALSE`
    e. `message = FALSE`
4. What is a reprex and why is it important to be able to make one (select all)?
    a. A reproducible example that enables your error to be reproduced.
    b. A reproducible example that helps others help you.
    c. A reproducible example during the construction of which you may solve your own problem.
    d. A reproducible example that demonstrates you've actually tried to help yourself.
5. Why are R Projects important (select all)?
    a. They help with reproducibility.
    b. They make it easier to share code.
    c. They make your workspace more organized.
    d. They are all that needs to be done.
6. Consider this sequence: '`git pull`, `git status`, _____, `git status`, `git commit -m "My message"`, `git push`'. What is the missing step (pick one)?

a. `git add -A`.
b. `git status`.
c. `git pull`.
d. `git push`.

3.1 Introduction

> Suppose you have cancer and you have to choose between a black box AI surgeon that cannot explain how it works but has a 90% cure rate and a human surgeon with an 80% cure rate. Do you want the AI surgeon to be illegal?
>
> Geoffrey Hinton, 20 February 2020.
>
> The number one thing to keep in mind about machine learning is that performance is evaluated on samples from one dataset, but the model is used in production on samples that may not necessarily follow the same characteristics… The finance industry has a saying for this: "past performance is no guarantee of future results". Your model scoring X on your test dataset doesn't mean it will perform at level X on the next N situations it encounters in the real world. The future may not be like the past.
>
> So when asking the question, "would you rather use a model that was evaluated as 90% accurate, or a human that was evaluated as 80% accurate", the answer depends on whether your data is typical per the evaluation process. Humans are adaptable, models are not. If significant uncertainty is involved, go with the human. They may have inferior pattern recognition capabilities (versus models trained on enormous amounts of data), but they understand what they do, they can reason about it, and they can improvise when faced with novelty
>
> If every possible situation is known and you want to prioritize scalability and cost-reduction, go with the model. Models exist to encode and operationalize human cognition in well-understood situations. ("well understood" meaning either that it can be explicitly described by a programmer, or that you can amass a dataset that densely samples the distribution of possible situations – which must be static)
>
> François Chollet, 20 February 2020.

If science is about systematically building and organising knowledge in terms of testable explanations and predictions, then data science takes this and focuses on data. Fundamentally data science is still science, and as such, building and organising knowledge is a critical aspect. Being able to do something yourself, once, does not achieve this. Hence, the focus on reproducibility and replicability.

? says 'Research is reproducible if it can be reproduced exactly, given all the materials used in the study.' '[Hence] materials need to be provided!'. ''[M]aterials' usually means data, code and software.' The minimum requirement is to be able to 'reproduce the data, methods and results (including figures, tables)'.

Similarly, from **?** you should have noticed that this has been an issue in other sciences. e.g. psychology. The issue with not being reproducible is that we are not contributing to knowledge. We no longer have any idea was is fact in the field of psychology. (This is coming for other fields too, including the field that I was trained in - economics.) Do this matter? Yes.

Some of the examples that **?** cites (which turned out to be dodgy) don't really matter e.g. ESP or the power pose. It doesn't really matter. But increasingly the same methods are being applied in areas where they do matter e.g. 'nudge' units. Similarly, **?** makes it clear that it's a big problem in data science.

The 'gay face' paper that **?** writes about has not released their dataset. We have no way of knowing what is going on with it. They have found a certain set of results based on that dataset, their methods, and what they did, but we have no way of knowing how much that matters. As **?** says 'the paper itself does some things right. It has a detailed discussion of the limitations of the data and the method'. You must do this in everything that you write, but it is not enough.

Without the data, we don't know what their results speak to as we don't understand how representative the sample is. If the dataset is biased, then that undermines their claims. There's a reason that while initial medical trials are done on mice, etc, eventually human trials are required.

In order to do the study they needed a trained dataset. They trained it using Mechanical Turk. Figure **??** is from **?**.

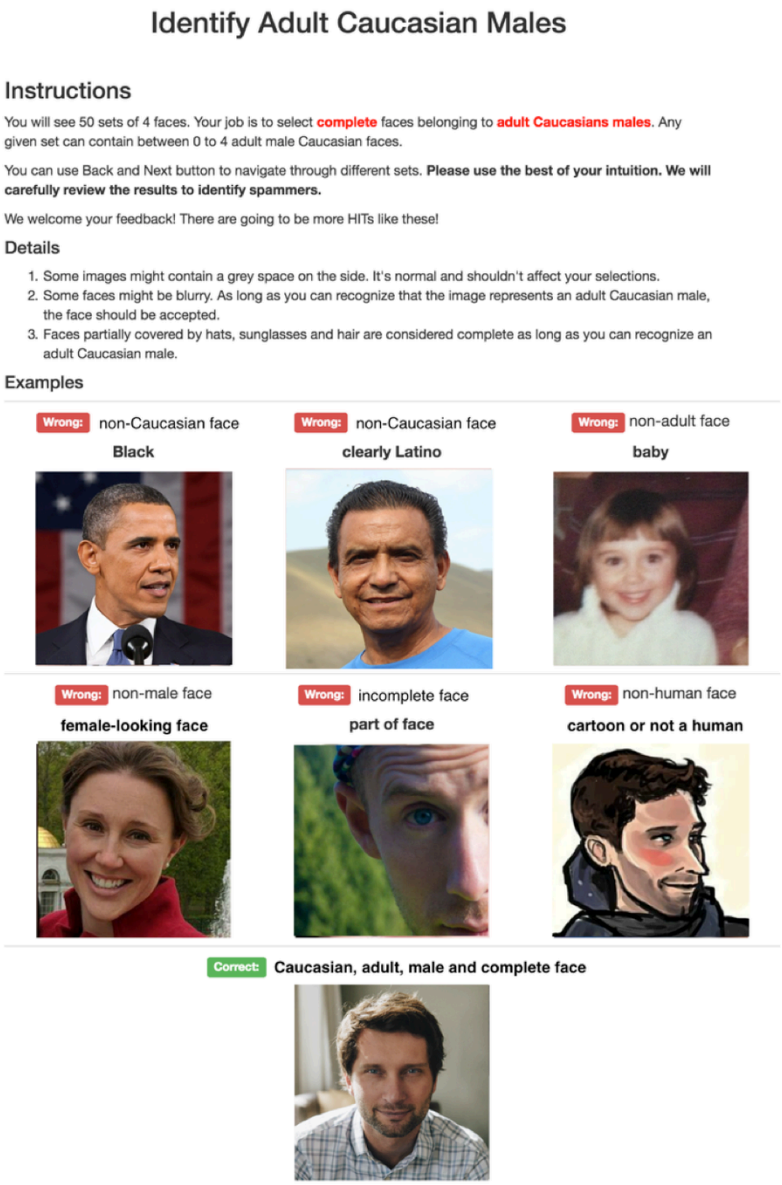
? comments:

> The problems here are legion: Barack Obama is biracial but simply "Black" by American cultural norms. "Clearly Latino" begs the question "to whom?" Latino is an ethnic category, not a racial one: many Latinos already are Caucasian, and increasingly so. By training their workers according to stereotypical American categories, WAK's algorithm can only spit out the garbage they put in.

'WAK's algorithm can only spit out the garbage they put in.' I would encourage you to print out that statement and paste it somewhere that you will see it every time you get a data science result.

What steps can we take to make our work reproducible?

1. Ensure your entire workflow is documented. How did you get the raw data? Can you save the raw data? Will the raw data always be available? Is the raw data available to others? What steps are you taking to trans-



Instructions for workers to do classification piecework on the Amazon Mechanical Turk platform, p. 46.

Figure 3.1: Instructions for workers to do classification piecework on the Amazon Mechanical Turk platform, p. 46. From Mattson.

form the raw data in data that can be analysed? How are you analysing the data? How are you building the report?

2. Try to improve each time. Can you run your entire workflow again? Can 'another person' run your entire workflow again? Can a future you run your entire workflow again? Can a future 'another person' run your entire workflow again? Each of these requirements is increasingly more onerous. We are going to start with worrying about the first. The way we are going to do this is by using R Markdown.

3.2 R Markdown

3.2.1 Getting started

R Markdown is a mark-up language similar to html or LaTeX, in comparison to a WYSIWYG language, such as Word. This means that all of the aspects are consistent, for instance, all 'main headings' will look the same. However it means that use symbols to designate how you would like certain aspects to appear, and it is only when you compile it that you get to see it.

R Markdown is a variant of regular markdown that is specifically designed to allow R code chunks to be included. The advantage is that you can get a 'live' document in which code executes and is then printed to a document. The disadvantage is that it can take a while for the document to compile because all of the code needs to run.

You can create a new R Markdown document within R Studio (File -> New File -> R Markdown Document). Another advantage of R Markdown is that very similar code can compile into a variety of documents, including html pages and PDFs. R Markdown also has default options set up for including a title, author, and date sections.

3.2.2 Basic commands

If you ever need a reminder of the basics of R Markdown then this is built into R Studio (Help -> Markdown Quick Reference). This provides the code for commonly needed commands:

- Emphasis: `*italic*, **bold**, _italic_, __bold__`
- Headers (these need to go on their own line with a line before and after): `# Header 1, ## Header 2, ### Header 3`
- Lists:

```
Unordered List
* Item 1
* Item 2
    + Item 2a
    + Item 2b
```

```
Ordered List
1. Item 1
2. Item 2
3. Item 3
    + Item 3a
    + Item 3b
```

- URLs: Can just include an address: http://example.com, or can include a `[linked phrase](http://example.com)`.
- Basic images can just be included either from the internet: `![alt text](http://example.com/logo.png)` or from a local file: `![alt text](figures/img.png)`.

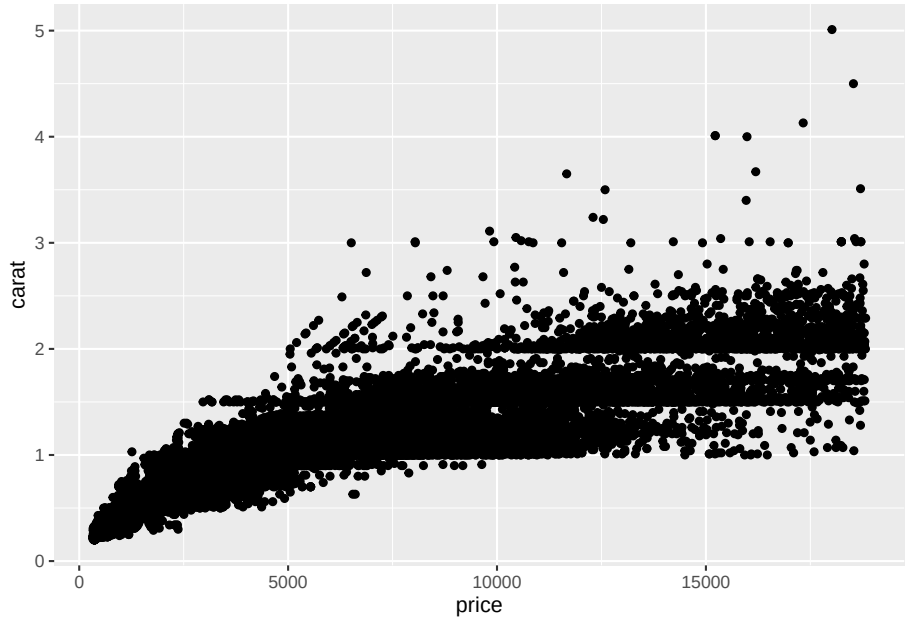
In order to create an actual document, once you have these pieces set up, click 'Knit'.

3.2.3 R chunks

You can include R (and a bunch of other languages) code in code chunks within your R Markdown document. Then when you knit your document, the R code will run and be included in your document.

To create an R chunk start with three backticks and then within curly braces tell markdown that this is an R chunk. Anything inside this chunk will be considered R code and run as such.

```r
library(tidyverse)
ggplot(data = diamonds) +
  geom_point(aes(x = price, y = carat))
```



There are various evaluation options that are available in chunks. You include these by putting a comma after `r` and then specifying any options before the closing curly brace. Helpful options include:

- `echo = FALSE`: run the code and include the output, but don't print the code in the document.
- `include = FALSE`: run the code but don't output anything and don't print the code in the document.
- `eval = FALSE`: don't run the code, and hence don't include the outputs, but do print the code in the document.
- `warning = FALSE`: don't display warnings.
- `message = FALSE`: don't display messages.

3.2.4 Abstracts and PDF outputs

In the default header, you can add a section for a header, so that it would look like this:

```
---
title: My document
author: Rohan Alexander
date: 5 January 2020
output: html_document
abstract: "This is my abstract."
---
```

Similarly, you can change the output from `html_document` to `pdf_document` in

order to produce a PDF. This uses LaTeX in the background so you may need to install a bunch of related packages.

3.2.5 References

You can reference a bibliography by including one in the preamble and then calling it in the text when you need.

```
---
title: My document
author: Rohan Alexander
date: 5 January 2020
output: html_document
abstract: "This is my abstract."
bibliography: bibliography.bib
---
```

You need to make a separate file called `bibliography.bib`. In that you need an entry for the item that you want to reference. R and R packages usually provide this for you for instance, if you run `citation()` then it tells you the entry to put in your bibtex file:

```
@Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2020},
    url = {https://www.R-project.org/},
  }
```

You need to create a unique key that you'll refer to it with in the text. This can be anything if it's unique, but I try to use meaningful ones, so that bibtex entry could become:

```
@Manual{citeR,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2020},
    url = {https://www.R-project.org/},
  }
```

And to cite R you'd then include the following: `@citeR`, which would put the brackets around the year, like this: **?** or `[@citeR]`, which would put the brackets around the whole thing, like this: (**?**).

3.2.6 Cross-references

Finally, it can be useful to cross-reference figures, tables and equations. This makes it easier to refer to them in the text. To do this for a figure you refer to the name of the R chunk that creates/contains the figure. For instance, `(Figure \@ref(fig:billssssss))` will produce: (Figure **??**) as the name of the R chunk is `billssssss` (don't forget to add the `fig` in front of the chunk name.

```
library(palmerpenguins)
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(alpha = 0.8) +
  scale_fill_manual(values = c("darkorange","purple","cyan4"),
                    guide = FALSE) +
  theme_minimal() +
  facet_wrap(~species, ncol = 1) +
  coord_flip()
```

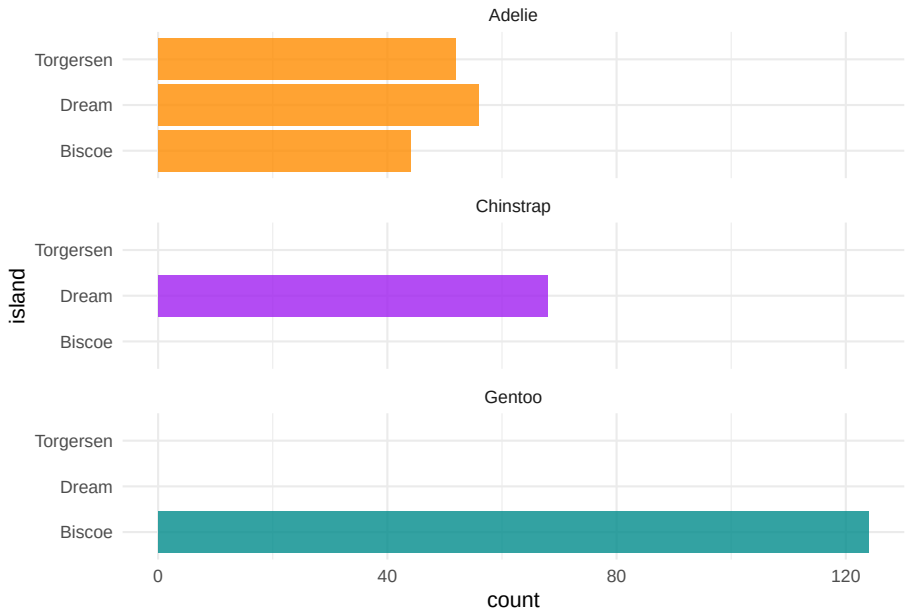


Figure 3.2: More bills of penguins

You can do similar for tables and equations e.g `(Table \@ref(tab:penguinhead))` will produce: (Table **??**) (again, don't forget to ad `tab` in front).

```
penguins %>%
  select(species, bill_length_mm, bill_depth_mm) %>%
  slice(1:5) %>%
```

Table 3.1: A penguin table

species	bill_length_mm	bill_depth_mm
Adelie	39.1	18.7
Adelie	39.5	17.4
Adelie	40.3	18.0
Adelie	NA	NA
Adelie	36.7	19.3

```
knitr::kable(caption = "A penguin table")
```

And finally, you can, and should, cross-reference equations also, but this time you need to add a tag `(\#eq:slope)` and then reference that e.g. use `Equation \@ref(eq:slope).` to produce Equation (??).

```
\begin{equation}
Y = a + b X \label{eq:slope}
\end{equation}
```

$$Y = a + bX \tag{3.1}$$

3.3 R projects

RStudio has the option of creating a project, which allows you to keep all the files (data, analysis, report etc) associated with a particular project together. To create a project, click Click File > New Project, then select empty project, name your project and think about where you want to save it. For example, if you are creating a project for Problem Set 2, you might call it `ps2` and save it in a sub-folder called `PS2` in your INF2178 folder.

Once you have created a project, a new file with the extension `.RProj` will appear in that file. As an example, download the R help folder. Whenever I work on class materials, I open the project file and work from that.

The main advantage of projects is that you don't have to set the working directory or type the whole file path to read in a file (for example, a data file). So instead of reading a csv from `"~/Documents/toronto/teaching/INF2178/data/"` you can just read it in from `data/`.

To meet even the minimal expected level of reproducibility, you must use R projects. You must not use `setwd()` because that ties your work to your computer.

3.4 Git and GitHub

3.4.1 Introduction

Here we introduce Git and GitHub. These are tools that:

a. enhance the reproducibility of your work by making it easier to share your code and data;
b. make it easier to show off your work;
c. improve your workflow by encouraging you to think systematically about your approach; and
d. (although we won't take advantage of this) make it easier to work in teams.

Git is a version control system. One way you might be used to doing version control is to have various versions of your files: `first_go.R`, `first_go-fixed.R`, `first_go-fixed-with-mons-edits.R`. But this soon becomes cumbersome. Some of you may use dates, for instance: `2020-03-12-analysis.R`, `2020-03-13-analysis.R`, `2020-03-14-analysis.R`, etc. While this keeps a record it can be difficult to search if you need to go back - will you really remember the date of some change in a week? How about a month or a year? It gets unwieldy fairly quickly.

Instead of this, Git allows you to only have one version of the file `analysis.R` and it keeps a record of the changes to that file, and a snapshot of that file at a given point in time. When it takes that snapshot is determined by you. When you want Git to take a snapshot you additionally include a message, saying what changed between this snapshot and the last. In that way, there is only ever one version of the file, but the history can be more easily searched.

The issue is that Git was designed for software developers. As such, while it works, it can be a little ungainly for non-developers (Figure **??**).

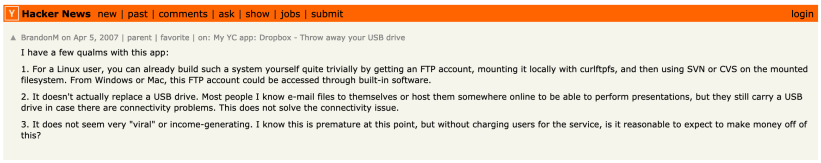


Figure 3.3: An infamous response to the launch of Dropbox in 2007, trivialising the use-case for Dropbox, and while this actually would work, it wouldn't for most of us.

Hence, GitHub, GitLab, and various other companies offer easier-to-use services that build on Git. GitHub used to be the weapon of choice, but they were sold to Microsoft in 2018 and since then other variants such as GitLab have risen in popularity. We will introduce GitHub here because it remains the most popular, and it is built into RStudio, however you should feel free to explore other options.

One of the hardest aspects of Git, and the rest, for me was the terminology. Folders are called 'repos'. Saving is called a 'commit'. You'll get used to it eventually, but just so you know - it's not you, it's Git - feeling confused it entirely normal

These are brief notes and you should refer to Jenny Bryan's book for further detail. Frankly, I can't improve on Bryan's book and I use them regularly myself.

3.4.2 Git

Check if you have Git installed by opening R Studio and then going to the Terminal and typing the following and then return.

```
git --version
```

If you get a version number, then you are done (Figure **??**).

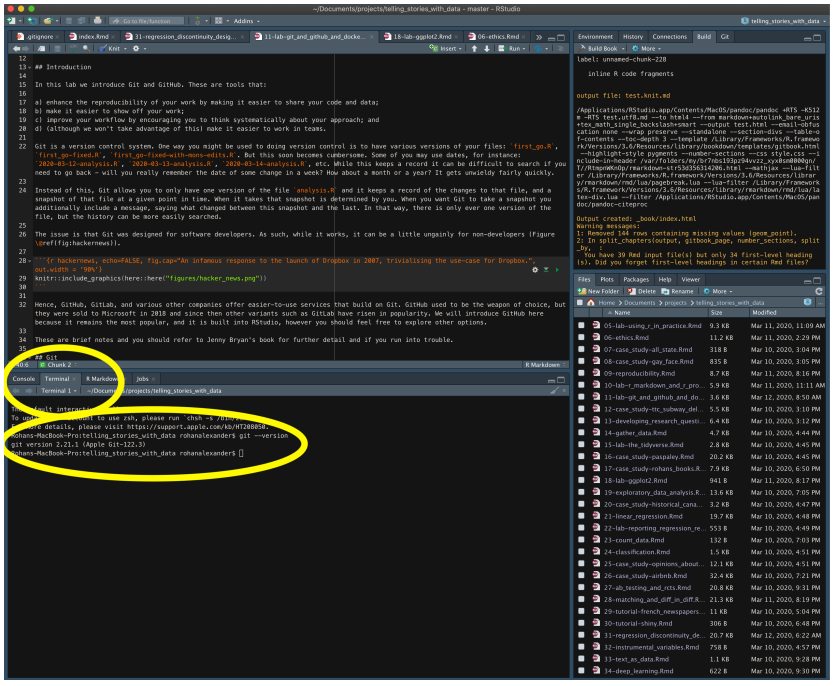


Figure 3.4: How to access the Terminal within R Studio

If you have a Mac then Git should come pre-installed, if you have Windows then there's a chance, and if you have Linux then you probably don't need this guide. If you don't get a version number, then you need to install it. Please go to Chapter 5 of **?** for some instructions based on your operating system.

After you have Git, then you need to tell it your username and email. You need

this because Git adds this information whenever you take a 'snapshot', or to use Git's language whenever you make a commit.

Again, within the Terminal, type the following, replacing the details with yours, and then return after each line.

```
git config --global user.name 'Jane Doe'
git config --global user.email 'jane@example.com'
git config --global --list
```

The details that you enter here will be public (there are various ways to hide your email address if you need to do this and GitHub provides instructions about this).

Again, if you have issues or need more detailed instructions please go to Chapter 7 of **?**.

3.4.3 GitHub

The first step is to create an account on GitHub (https://github.com) (Figure **??**).

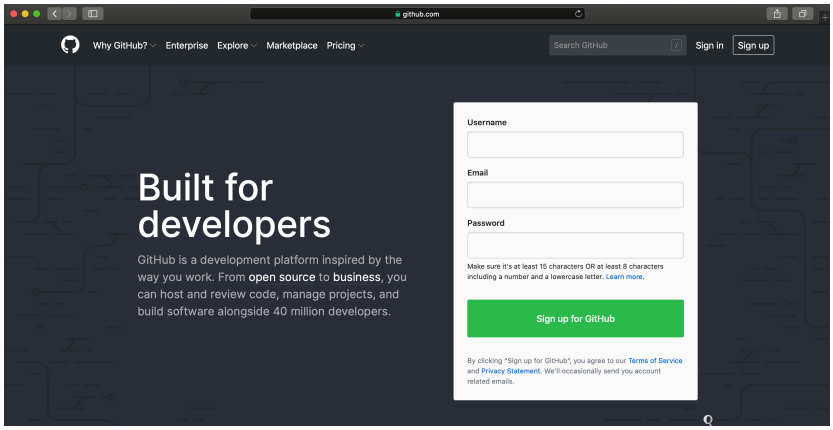


Figure 3.5: Sign up screen at GitHub

GitHub doesn't have the most intuitive user experience in the world, but we are now going to make a new folder (which is called a 'repo' in Git). You are looking for a plus sign in the top right, and then select 'New Repository' (Figure **??**).

At this point you can add a sensible name for your repo. Leave it as public (you can delete it later if you want). And check the box to initialize with a readme. In the 'Add .gitignore' option you can leave it for now, but if you start using GitHub more regularly then you may like to select the R option here. (That

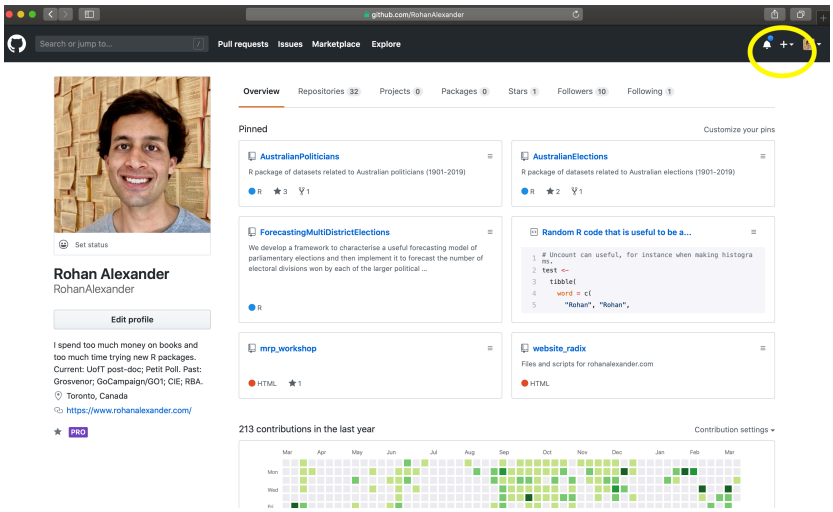


Figure 3.6: Create a new repository

just tells Git to ignore various files.) After that, just click the button to create a new repository (Figure **??**).

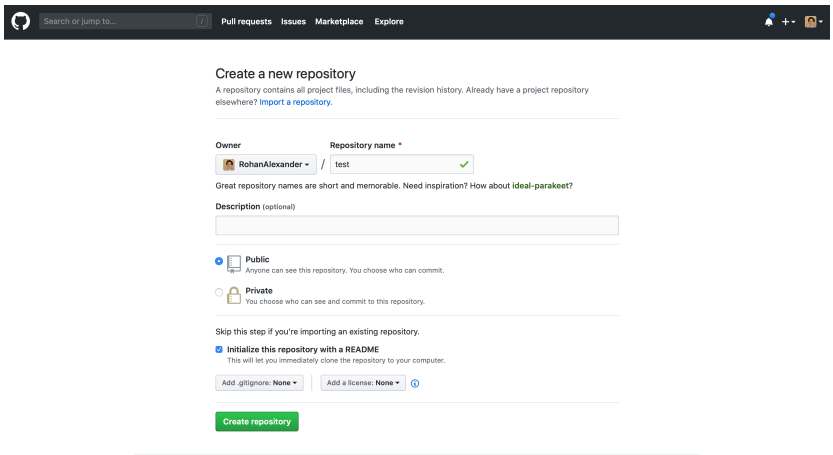


Figure 3.7: Create a new repository, really

You'll now be taken to a screen that is fairly empty, but the details that you need are in the green 'Clone or Download' button, then click the clipboard (Figure **??**).

Now you need to open Terminal, and use `cd` to get to where you want to save the folder, then:

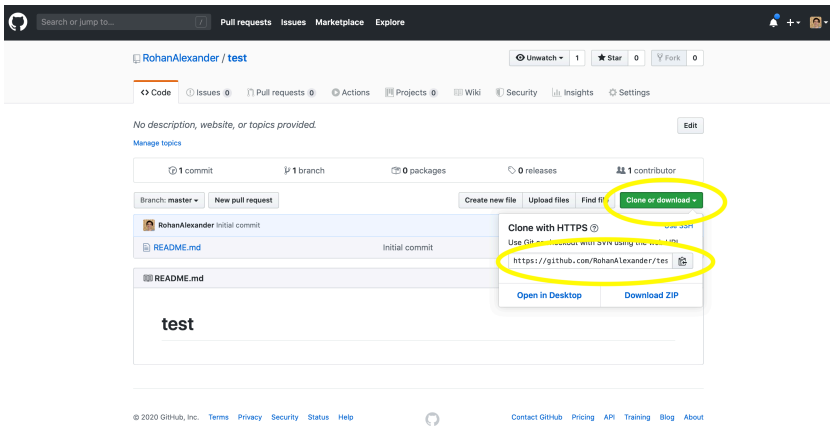


Figure 3.8: Get the details of your new repository

```
git clone https://github.com/RohanAlexander/test.git
```

At this point, a new folder has been created. We can now interact with it.

The first step is almost always to pull the latest changes with `git pull` (this is slightly pointless in this example because it's just us but it's a good habit to get into). We can then make a change to the folder, for instance, update the readme, and then save it as usual. Once this is done, we need to add, commit, and push. As before, use `cd` to navigate to your folder, then `git status` to see if there is anything going on (you should see some reference to the change you made). Then `git add -A` adds the changes to the staging area (this seems pointless, and it is in this context, but this allows you to specify specific files and similar if needed). Then `git status` to check what has happened. Then `git commit -m "Minor update to readme"`, and then `git status` to check on everything, and finally `git push`.

To summarise (assuming you are in the relevant folder):

```
git pull
git status
git add -A
git status
git commit -m "Short commit message"
git status
git push
```

3.4.4 Using Git within RStudio

I promised that GitHub was built into RStudio, but so far we've not really taken advantage of that. The way to do it is to create a new repo in GitHub, and copy the information, as before.

At this point, you open RStudio, select `Files`, `New Project`, `Version Control`, `Git`, and paste the information for the repo. Go through the rest of it, saving the folder somewhere sensible, and clicking 'Open in new session'. This will then create a new folder on your computer which will be a Git folder that is linked to the GitHub repo that you created.

At this point, you'll have a 'Git' tab (Figure **??**).

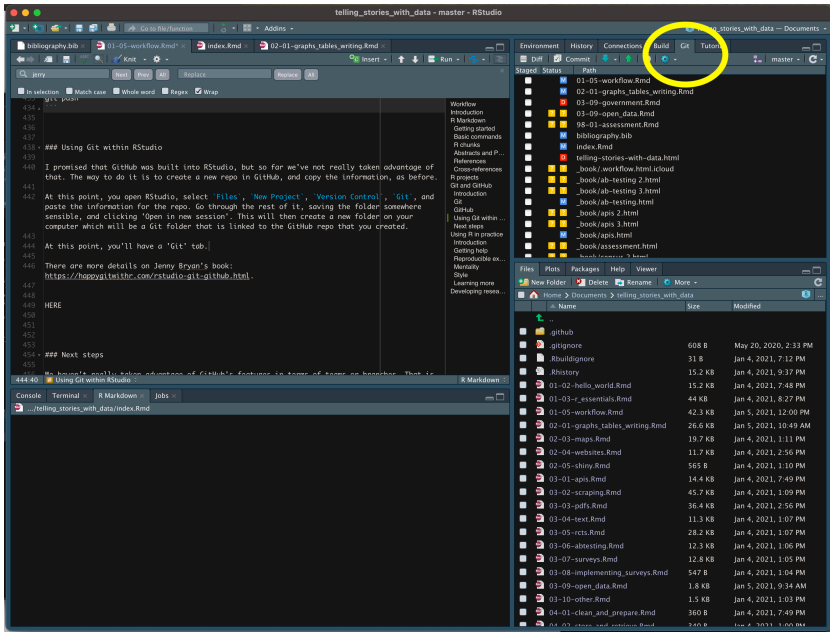


Figure 3.9: The Git pane in R Studio

First pull (click the blue down arrow). Now you want to tick the 'staged' box against the files that you want to commit. Then click 'Commit'. Type a message in the 'Commit message' box and then click 'Commit'. Finally 'Push'. Again, the details are in **?**, especially Chapter 12.

3.4.5 Next steps

We haven't really taken advantage of GitHub's features in terms of teams or branches. That is certainly something that you should get into once you have more confidence with these basics. As always, when it comes to Git for data scientists who use R, you should go to the relevant sections of **?**.

3.5 Using R in practice

3.5.1 Introduction

This section is what do when your code doesn't do what you want, discusses a mindset that may help when doing quantitative analysis with R, and finally, some recommendations around how to write your code.

3.5.2 Getting help

Programming is hard and everyone struggles sometimes. At some point your code won't run or will throw an error. This is normal, and it happens to everyone. It happens to me on a daily, sometimes hourly, basis. Everyone gets frustrated. There are a few steps that are worthwhile taking when this happens:

- Sometimes the error messages in R are useful. Read it carefully and see if there's anything of use in it. At the very least, if you get the same message in the future, hopefully you might remember how you solved the problem this time!
- If you're getting an error then try googling it, (I find it can help to include the term 'R' or 'tidyverse' or the relevant package name).
- If there's a particular function that seems to be giving trouble, have a look at the help file for it. Sometimes you might be putting in the arguments in the wrong order. You can do this with '?function' e.g. for help with select, you would type '`?select`' and then run that line.
- Check the class of the object. Sometimes R is a little fussy and converting the class can help.
- If your code just isn't running, then try searching for what you are trying to do, e.g. 'save PDF of graph in R made using ggplot'. Almost always there are relevant blog posts or Stack Overflow answers that will help.
- Try to restart R and R Studio and load everything again.
- Try to restart your computer.

There are a few small mistakes that I often make and may be worth checking in case you make them too:

- check the class e.g. `class(my_dataset$its_column)` to make sure that is what it should be;
- when you're using ggplot make sure you use '`+`' not '`%>%`'; and
- check whether you are using '.' when you shouldn't be, or vice versa.

It's almost always helpful to take a break and come back the next day.

Asking for help is a skill that you will get better at. Try not to say 'this doesn't work', or 'I tried everything' or 'here's the error message, what do I do?'. In general, it's not possible to help based on that because there are too many possibilities. Instead:

1. Provide a small example of your data, and code, and detail what is going wrong.
2. Document what you have tried so far - what Stack Overflow pages have you looked at and why are they not quite what you're after? What RStudio Community pages have you tried?
3. Be clear about the outcome that you would like.

As the RStudio Community welcome page says 'your job is to make it as easy as possible for others to help you'. To enable that to happen you need to 'create a minimal reproducible example, or reprex for short' You can get more information about this here, but basically what is needed is that you code is reproducible (so include things like `library()`, etc) and that is it minimal - that means making a very simple smaller example and reproducing the error on that small example.

Usually doing this actually allows you to solve your own problem. If it doesn't then it'll allow someone else a fighting chance as being able to help you. I especially recommend the `reprex` package (`?`).

There's almost no chance that you've got a problem that someone hasn't addressed before, it's just a matter of finding the answer! Try to be tenacious with this and learn how to solve your own problems.

3.5.3 Mentality

> (Y)ou are a real, valid, *competent* user and programmer no matter what IDE you develop in or what tools you use to make your work work for you
>
> (L)et's break down the gates, there's enough room for everyone
>
> Sharla Gelfand, 10 March 2020.

I'm a little hesitant to make suggestions with regard to mentality. If you write code, then you're coder regardless of how you do it, what you're using it for, or who you are. But I want to share a few traits that I have found have been useful to cultivate in myself. That said, entirely, whatever works for you is great, so take or leave this section.

- Focused: I've found that having an aim to 'learn R' or something similar tends to be problematic, because there's no real end point to that. Instead I would recommend smaller, more specific goals, such as 'make a histogram about the 2019 Canadian Election with ggplot'. That is something that you can focus on and achieve. With more nebulous goals it becomes easier to get lost on tangents, much more difficult to get help, and I've noticed that people who have nebulous goals seem to give up.
- Curious: I've found that it's useful to just have a go. In general, the worst that happens is that you waste your time and have to give up. You can rarely break something irreparably with code. If you want to know what

happens if you pass a 'vector' instead of a 'dataframe' to `ggplot` then just try it.

- Pragmatic: At the same time, I've found that it's best to try to stick within the bounds of what I know and just make one small change each time. For instance if you're wanting to do some regression, and curious about the `tidymodels` package (`?`) instead of `lm()`. Perhaps you could just use one aspect from the `tidymodels` package initially and then make another change next time. In my opinion ugly code that gets the job done, is better than beautiful code that is never finished.
- Tenacious: This is a balancing act. I always find there are unexpected problems and issues with every project. On the one hand persevering despite these is a good tendency. But on the other hand I've learned that sometimes I need to be prepared to give up on something if it doesn't seem like a break-through is possible. This is where I have found that mentors can be useful as they tend to have a better idea. This is also where planning comes in.
- Planned: I have found it is very useful to plan out what you are going to do. For instance, you may want to make a histogram of the 2019 Canadian Election. I find it useful to plan the steps that are needed and even to sketch out how I might implement each step. For instance, the first step is to get the data. What packages might be useful? Where might the data be? What is our back-up plan for if we can't find the data in that initial spot?
- Done is better than perfect: We all have various perfectionist tendencies to a certain extent, but I recommend that you try to turn them off to a certain extent when it comes to R. In the first instance just try to write code that works, especially in the early days. You can always come back and improve aspects of it. But it is important to actually ship.

3.5.4 Code comments

Comment your code.

There is no one way to write code, especially in R. However, there are some general guidelines that will make it easier for you even if you're just working on your own.

Comment your code.

Comments in R can be added by including the # symbol. The shortcut on Mac is 'Command + Shift + m'. You don't have to put a comment at the start of the line, it can be midway through. In general, you don't need to comment what every aspect of your code is doing but you should comment parts that are not obvious. For instance, if you read in some value then you may like to comment where it is coming from.

Comment your code.

You should comment why you are doing something. What are you trying to achieve?

Comment your code.

You must comment to explain weird things. Like if you're removing some specific row, say row 27, then why are you removing that row? It may seem obvious in the moment, but future-you in six months won't remember.

Comment your code.

I like to break my code into sections. For instance, setting up my workspace, reading in datasets, manipulating and cleaning the dataset, analysing the datasets, and finally producing tables and figures. While it can be difficult to speak generally, I usually separate each of those certainly with comments explaining what is going on, and sometimes into separate files, depending on the length.

Comment your code.

Additionally, at the top of each file I put basic information, such as the purpose of the file, and pre-requisites or dependencies, the date, the author and contact information, and finally and red-flags, bodies, or todos.

Comment your code.

At the very least I recommend something like the following for every R script:

```
#### Preamble ####
# Purpose: Brief sentence about what this script does
# Author: Your name
# Data: The date it was written
# Contact: Add your email
# License: Think about how your code may be used
# Pre-requisites:
# - Maybe you need some data or some other script to have been run?


#### Workspace setup ####
# Don't keep the install.packages line - just comment out if need be
# Load libraries
library(tidyverse)

# Read in the raw data.
raw_data <- readr::read_csv("inputs/data/raw_data.csv")


#### Next section ####
...
```

3.5.5 Learning more

One of the great aspects of R is that there is a friendly community of people who use it. There are a variety of ways that I learn about new tricks, functions, and packages including:

- R Ladies 'is a world-wide organization to promote gender diversity in the R community'. Whether that is your cup of tea or not, they tend to share fantastic material on their twitter feeds: https://twitter.com/rladiesglobal and https://twitter.com/WeAreRLadies (where someone takes over the account for the week, so the enthusiasm never wanes). If you're in Toronto, then the local chapter is: https://rladies.org/canada-rladies/locality/Toronto/ and https://www.meetup.com/rladies-toronto/.
- R Weekly is a weekly newsletter that highlights new things that have happened recently.
- A popular R podcast is Not So Standard Deviations.

Another great way to learn is by exchanging your code with others. Initially, just have them read it and give you feedback about it. But after you get a bit more confident run each other's code. The most efficiently I've ever improved in my R journey has been by having Monica try to run my code.

3.6 Developing research questions

Both qualitative and quantitative approaches have their place, but here we focus on quantitative approaches. (Qualitative research is important as well, and often the most interesting work has a little of both - 'mixed methods'.) This means that we are subject to issues surrounding data quality, scales, measures, sources, etc. We are especially interested in trying to tease out causality.

Broadly there are two ways to go about research:

1) data-first,
2) question-first.

If you get a job somewhere typically you will initially be data-first. This means that you will need to work out the questions that you can reasonably answer with the data available to you. After you show some promise, you may be given the latitude to explore specific questions, possibly even gathering data specifically for that purpose. Contrast this with the example of the Behavioural Insights Team, (**?**, p. 23) who got to design and then carry out experiments given the remit of the entire British government (as they were spun out of the prime minister's office).

When deciding the questions that you can reasonably answer with the data that are available, you need to think about:

1) Theory: Do you have a reasonable expectation that there is something causal that could be determined? Charting the stock market - maybe, but might be better with haruspex because at least that way you have something you could eat. You need a reasonable theory of how x may be affecting y.
2) Importance: There are plenty of trivial questions that you could ask, but it's important to not waste your time. The importance of a question also helps with motivation when you are on your fourth straight week of cleaning data and de-bugging your code. It also (and this becomes important) makes it easier to get talented people to work with you, or similarly to convince people to fund you or allow you to work on this project.
3) Availability: Can you reasonably expect get more data about this research question in the future or is this the extent of the data that could be gathered?
4) Iteration: Is this something that you can come back to and run often or is this a once-off analysis?

The 'FINER framework' as a mnemonic device used in medicine.[1] This framework reminds us to ask questions that are (**?**):

- Feasible: Adequate number of subjects; adequate technical expertise; affordable in time and money; manageable in scope.
- Interesting: Getting the answer intrigues investigator, peers and community.
- Novel: Confirms, refutes or extends previous findings
- Ethical: Amenable to a study that institutional review board will approve.
- Relevant: To scientific knowledge; to clinical and health policy; to future research.

? build on this in terms of developing research questions and recommend 'PICOT':

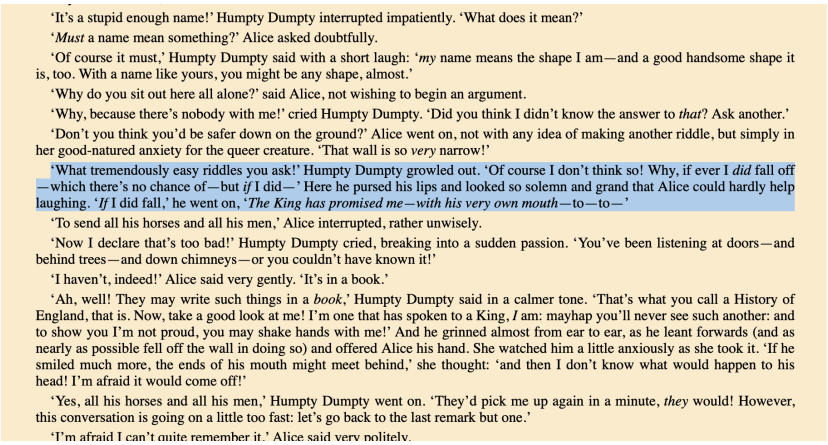
- Population: What specific population are you interested in?
- Intervention: What is your investigational intervention?
- Comparison group: What is the main alternative to compare with the intervention?
- Outcome of interest: What do you intend to accomplish, measure, improve or affect?
- Time: What is the appropriate follow-up time to assess outcome

Often time will be constrained, possibly in interesting ways and these can guide your research. If we are interested in the effect of Trump's tweets on the stock market, then that can be done just by looking at the minutes (milliseconds?) after he tweets. But what if we are interested in the effect of a cancer drug on

[1] Thanks to Aaron Miller for pointing me to this.

long term outcomes? If the effect takes 20 years then we either have to wait a while, or we need to look at people who were treated in 2000, but then we have selection effects and different circumstances to if we give the drug today. Often the only reasonable thing to do is to build a statistical model, but then we need adequate sample sizes, etc.

Usually the creation of a counterfactual is crucial. We'll discuss counterfactuals a lot more later, but briefly, a counterfactual is an if-then statement in which the 'if' is false. Consider the example of Humpty Dumpty from Lewis Carroll's Through the Looking-Glass:



'It's a stupid enough name!' Humpty Dumpty interrupted impatiently. 'What does it mean?'

'*Must* a name mean something?' Alice asked doubtfully.

'Of course it must,' Humpty Dumpty said with a short laugh: '*my* name means the shape I am—and a good handsome shape it is, too. With a name like yours, you might be any shape, almost.'

'Why do you sit out here all alone?' said Alice, not wishing to begin an argument.

'Why, because there's nobody with me!' cried Humpty Dumpty. 'Did you think I didn't know the answer to *that*? Ask another.'

'Don't you think you'd be safer down on the ground?' Alice went on, not with any idea of making another riddle, but simply in her good-natured anxiety for the queer creature. 'That wall is so *very* narrow!'

'What tremendously easy riddles you ask!' Humpty Dumpty growled out. 'Of course I don't think so! Why, if ever I *did* fall off—which there's no chance of—but *if* I did—' Here he pursed his lips and looked so solemn and grand that Alice could hardly help laughing. '*If* I did fall,' he went on, '*The King has promised me—with his very own mouth*—to—to—'

'To send all his horses and all his men,' Alice interrupted, rather unwisely.

'Now I declare that's too bad!' Humpty Dumpty cried, breaking into a sudden passion. 'You've been listening at doors—and behind trees—and down chimneys—or you couldn't have known it!'

'I haven't, indeed!' Alice said very gently. 'It's in a book.'

'Ah, well! They may write such things in a *book*,' Humpty Dumpty said in a calmer tone. 'That's what you call a History of England, that is. Now, take a good look at me! I'm one that has spoken to a King, *I* am: mayhap you'll never see such another: and to show you I'm not proud, you may shake hands with me!' And he grinned almost from ear to ear, as he leant forwards (and as nearly as possible fell off the wall in doing so) and offered Alice his hand. She watched him a little anxiously as she took it. 'If he smiled much more, the ends of his mouth might meet behind,' she thought: 'and then I don't know what would happen to his head! I'm afraid it would come off!'

'Yes, all his horses and all his men,' Humpty Dumpty went on. 'They'd pick me up again in a minute, *they* would! However, this conversation is going on a little too fast: let's go back to the last remark but one.'

'I'm afraid I can't quite remember it,' Alice said very politely.

Figure 3.10: Humpty Dumpty example

Humpty is satisfied with what would happen if he were to fall off, even though he is similarly satisfied that this would never happen. (I won't ruin the story for you.) The comparison group often determines your results e.g. the relationship between VO2 and athletic outcomes, compared with elite athletic outcomes.

Finally, we can often dodge ethics boards in data science, especially once you leave university. Typically, ethics guides from medicine and other fields are focused on ethics boards. But we often don't have those in data science applications. Even if your intentions are unimpeachable, I want to suggest one additional aspect to think about, and that is Bayes theorem:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

(The probability of A given B depends on the probability of B given A, the probability of A, and the probability of B.)

To see why this may be relevant, let's go to the canonical Bayes example: There is some test for a disease that is 99 per cent accurate both ways (that is, if a person actually has the disease there is a 99 per cent chance the test says

positive, and is a person does not have the disease then there is a 99 per cent chance the test says negative). Let's just say that only 0.005 of the population has the disease. Then if we randomly pick someone from the general population then the chance that they have the disease is outstandingly low. This is even if they test positive:

$$\frac{0.99 \times 0.005}{0.99 \times 0.005 + 0.01 \times 0.995} \approx 33.2$$

To see why this may be relevant, consider the example of Google's AI cancer testing (**?**). Basically what they have done is to train a model that can identify breast cancer. They claim 'greater accuracy, fewer false positives, and fewer false negatives than experts'.

I, and many others (**?**), would argue this is probably not where we would want these resources directed at this point. Even when perfectly healthy people go and get screened they tend to find various things that are 'wrong' with them. The issue is that they're perfectly healthy and that we've rarely got a good idea as to whether that aspect that was flagged by the test is a big deal or not.

Given low prevalence in the community, we probably don't want wide-spread use of a particular testing regime that only looks at one aspect (i.e. the mammogram in this case). Bayes rule guides us that the danger caused by the unnecessary 'treatment' would probably outweigh the benefits. The authors of that Google blog post likely have unimpeachable ethics, but they may not understand Bayes rule.

Part II

Communicate

Chapter 4

Static

Required reading

- Healy, Kieran, 2019, *Data Visualization: A Practical Introduction*, Princeton University Press, Chapters 3 and 4, https://socviz.co/.
- Wickham, Hadley, and Garrett Grolemund, 2017, *R for Data Science*, Chapter 28, https://r4ds.had.co.nz/.
- Hodgetts, Paul, 2020, 'The ggfortify Package', 31 December, https://www.hodgettsp.com/posts/r-ggfortify/.
- Zinsser, William, 1976 [2016], *On Writing Well*. (Any edition is fine. This book is first, despite alphabetical order because if you're serious about improving your writing then you should start with this book. It only takes a few hours to read. You'll go onto other books, but start with this one.)
- Zinsser, William, 2009, 'Writing English as a Second Language', Lecture, Columbia Graduate School of Journalism, 11 August, https://theamericanscholar.org/writing-english-as-a-second-language/. (I'm realistic enough to realise that requiring a book, even though I've said it's great and it's short, is a bit of a stretch. If you really don't want to commit to reading the Zinsser, then please at least read this 'crib notes' version of it.)
- Alexander, Monica, 2019, 'The concentration and uniqueness of baby names in Australia and the US', https://www.monicaalexander.com/posts/2019-20-01-babynames/. (Look at how Monica explains concepts, especially the Gini coefficient, in a way that you can understand even if you've never heard of it before.)
- Bronner, Laura, 2020, 'Quant Editing', http://www.laurabronner.com/quant-editing. (Read these points and evaluate your own writing against them. It's fine to not comply with them if you have a good reason, but you need to know that you're not complying with them).
- (The) Economist, 2013, 'Johnson: Those six little rules', Prospero, 29

July 2013, available at: https://www.economist.com/prospero/2013/07/29/johnson-those-six-little-rules.
- Girouard, Dave, 2020, 'A Founder's Guide to Writing Well', First Round Review, 4 August, https://firstround.com/review/a-founders-guide-to-writing-well/.
- Graham, Paul, 2020, 'How to Write Usefully', http://paulgraham.com/useful.html. (Graham is good at writing for a programmer, but if you have a similar background then you may like this.)

Recommended reading

- (The) Economist, 1991 [2014], 'The Economist Style Guide', Twelfth edition. (Any edition is fine. Pick a point or two each day and think about how it related to your own writing.)
- Cochrane, John H., 2005, 'Writing Tips for Ph. D. Students', https://faculty.chicagobooth.edu/john.cochrane/research/papers/phd_paper_writing.pdf. (This is aimed at academic research papers, but parts are still broadly relevant. And if you're going into academia then this is very relevant.)
- Codrey, Laura, 2013, 'Churchill's call for brevity', 17 October, https://blog.nationalarchives.gov.uk/churchills-call-for-brevity/.
- Five Thirty Eight, 2020, Pick almost any article in their sports (https://fivethirtyeight.com/sports/) or politics (https://fivethirtyeight.com/politics/) sections. (The people at 538 write beautifully. Look at how their titles tell you exactly what is going on, or what they found. Look at how nicely their first paragraphs motivates you to read the rest of the article. Why am I reading about BYU basketball when I'm indifferent to both BYU and college basketball? Because that title and first paragraph hooked me.)
- Graham, Paul, 2005, 'Writing, Briefly', http://paulgraham.com/writing44.html.
- Patrick, Cameron, 2019, 'Plotting multiple variables at once using ggplot2 and tidyr', 26 November, https://cameronpatrick.com/post/2019/11/plotting-multiple-variables-ggplot2-tidyr/.
- Patrick, Cameron, 2020, 'Making beautiful bar charts with ggplot', 15 March, https://cameronpatrick.com/post/2020/03/beautiful-bar-charts-ggplot/.
- Shapiro, Jesse M., 'Four Steps to an Applied Micro Paper', https://www.brown.edu/Research/Shapiro/pdfs/foursteps.pdf. (This is mostly recommended for the part about 'the robot' with regard to your data section.)
- Shapiro, Julian, 'Writing Well', https://www.julian.com/guide/write/intro.
- Strunk, William Jr., 1959 [2009] 'The Elements of Style'. (Any edition is fine. Eventually you'll move beyond this, but it's important to know the rules before you break them).
- Vanderplas, Susan, Dianne Cook, and Heike Hofmann, 2020, 'Testing Statistical Charts: What Makes a Good Graph?', *Annual Review of*

Statistics and Its Application, https://www.annualreviews.org/doi/abs/10.1146/annurev-statistics-031219-041252

Examples of well-written papers

- Barron, Alexander TJ, Jenny Huang, Rebecca L. Spang, and Simon DeDeo. "Individuals, institutions, and innovation in the debates of the French Revolution." Proceedings of the National Academy of Sciences 115, no. 18 (2018): 4607-4612.
- Chambliss, Daniel F. "The Mundanity of Excellence: An Ethnographic Report on Stratification and Olympic Swimmers." Sociological Theory 7, no. 1 (1989): 70-86. doi:10.2307/202063.
- Joyner, Michael J. "Modeling: optimal marathon performance on the basis of physiological factors." Journal of Applied Physiology 70, no. 2 (1991): 683-687.
- Kharecha, Pushker A., and James E. Hansen, 2013, 'Prevented mortality and greenhouse gas emissions from historical and projected nuclear power', *Environmental science & technology*, 47, no. 9, pp. 4889-4895.
- Samuel, Arthur L., 1959, 'Some studies in machine learning using the game of checkers', *IBM Journal of research and development*, 3, no. 3, pp. 210-229.
- Wardrop, Robert L., 1995, 'Simpson's paradox and the hot hand in basketball', *The American Statistician*, 49, no. 1, 24-28.

Key concepts/skills/etc

- Show the reader your raw data, or as close as you can come to it.
- Use either `geom_point` or `geom_bar` initially.
- Writing efficiently and effectively is a requirement if you want your work to be convincing.
- Don't waste your reader's time.
- A good title says what the paper is about, a great title says what the paper found.
- For a six-page paper, a good abstract is a three to five sentence paragraph. For a longer paper your abstract can be slightly longer.

Key libraries

- `ggplot`
- `patchwork`

Key functions/etc

- `ggplot::geom_point()`
- `ggplot::geom_bar()`

Quiz

1. I have a dataset that contains measurements of height (in cm) for a sample of 300 penguins, who are either the Adeline or Emperor species. I am interested in visualizing the distribution of heights by species in a graphical

way. Please discuss whether a pie chart is an appropriate type of graph to use. What about a box and whisker plot? Finally, what are some considerations if you made a histogram? [Please write a paragraph or two for each aspect.]

2. Assume the dataset and columns exist. Would this code work? `data %>% ggplot(aes(x = col_one)) %>% geom_point()` (pick one)?
   a. Yes
   b. No
3. If I have categorical data, which geom should I use to plot it (pick one)?
   a. `geom_bar()`
   b. `geom_point()`
   c. `geom_abline()`
   d. `geom_boxplot()`
4. Why are box plots often inappropriate (pick one)?
   a. They hide the full distribution of the data.
   b. They are hard to make.
   c. They are ugly.
   d. The mode is clearly displayed.
5. Which of the following is the best title (pick one)?
   a. "Problem Set 1"
   b. "Unemployment"
   c. "Examining Canada's Unemployment (2010-2020)"
   d. "Canada's Unemployment Increased between 2010 and 2020"

4.1 Introduction

In order to convince someone of your story, your paper must be well-written, well-organized, and easy to follow. It should flow easily from one point to the next. It should have proper sentence structure, spelling, vocabulary, and grammar. Each point should be articulated clearly and completely without being overly verbose. Papers should demonstrate your understanding of the topics you are writing about and your confidence in discussing the terms, techniques and issues that are relevant. References must be included and properly cited because this enhances your credibility.

> People who need to write: founders, VCs, lawyers, software engineers, designers, painters, data scientists, musicians, filmmakers, creative directors, physical trainers, teachers, writers. Learn to write.
>
> Sahil Lavingia.
>
> This is great advice. Writing well has done just as much for me as knowing how to code. I'd add that if you're intimidated by writing, start a blog and write often about something you're interested in. You'll get better. At least that's what I've done for the past 10

years. :)

Vicki Boykis.

This chapter is about writing. By the end of it you will have a better idea of how to write short, detailed, quantitative papers that communicate exactly what you want them to and don't waste the time of your reader.

One critical part of telling stories with *data*, is that it's ultimately the data that has to convince them. You're the medium, but the data are the message. To that end, the easiest way to try to convince someone of your story is to show them the data that allowed you to come to that story. Plot your raw data, or as close to it as possible.

While `ggplot` is a fantastic tool for doing this, there is a lot to that package and so it can be difficult to know where to start. My recommendation is that you start with either a scatter plot or a bar chart. What is critical is that you show the reader your raw data. These notes run through how to do that. It then discusses some more advanced options, but the important thing is that you show the reader your raw data (or as close to it as you can). Students seem to get confused what 'raw' means; I'm using it to refer to as close to the original dataset as possible, so no sums, or averages, etc, if possible. Sometimes your data are too disperse for that or you've got other constraints, so there needs to be an element of manipulation. The main point is that you, at the very least, need to plot the data that you're going to be modelling. If you are dealing with larger datasets then just take a 10/1/0.1/etc per cent sample.

Figure 4.1: Show me the data!

Source: YouTube screenshot.

4.2 Graphs

Graphs are critical to tell a compelling story. And the most important thing with your graphs is to plot your raw data. Again: Plot. Your. Raw. Data.

Let's look at a somewhat fun example from the `datasauRus` package (**?**).

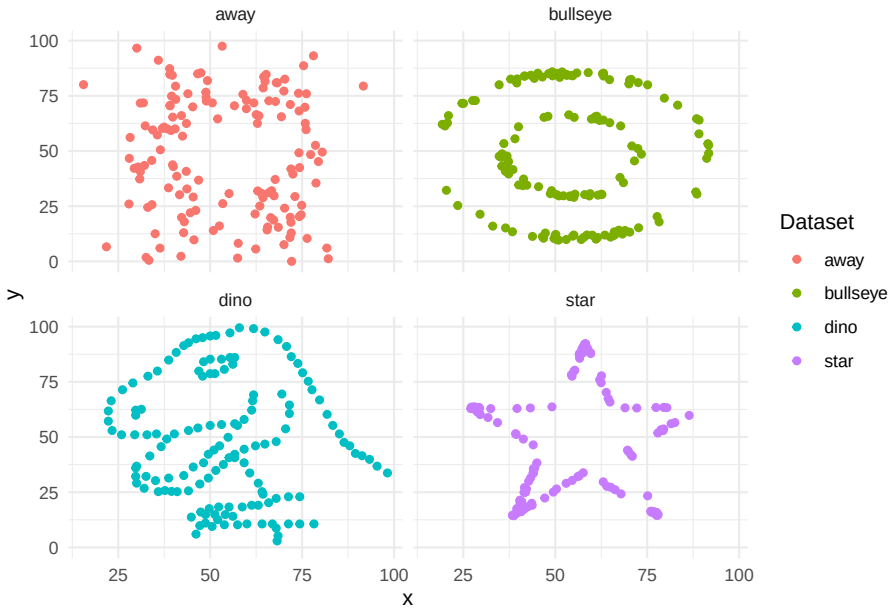
```
library(datasauRus)

# Code from: https://juliasilge.com/blog/datasaurus-multiclass/
datasaurus_dozen %>%
  filter(dataset %in% c("dino", "star", "away", "bullseye")) %>%
  group_by(dataset) %>%
  summarise(across(c(x, y), list(mean = mean, sd = sd)),
    x_y_cor = cor(x, y)
  ) %>%
  ungroup()
```

```
## # A tibble: 4 x 6
##   dataset  x_mean  x_sd y_mean  y_sd x_y_cor
##   <chr>     <dbl> <dbl>  <dbl> <dbl>   <dbl>
## 1 away       54.3  16.8   47.8  26.9 -0.0641
## 2 bullseye   54.3  16.8   47.8  26.9 -0.0686
## 3 dino       54.3  16.8   47.8  26.9 -0.0645
## 4 star       54.3  16.8   47.8  26.9 -0.0630
```

And despite these similarities at a summary statistic level, they're actually very different, well, beasts, when you plot the raw data.

```
datasaurus_dozen %>%
  filter(dataset %in% c("dino", "star", "away", "bullseye")) %>%
  ggplot(aes(x=x, y=y, colour=dataset)) +
  geom_point() +
  theme_minimal() +
  facet_wrap(vars(dataset), nrow = 2, ncol = 2) +
  labs(colour = "Dataset")
```



4.2.1 Bar chart

Bar charts are useful when you have one variable that you want to focus on. Hint: you almost always have one variable that you want to focus on. Hence, you should almost always include at least one (and likely many) bar charts. Bar charts go by a variety of names, depending on their specifics. I recommend the R Studio Data Viz Cheat Sheet.

To get started, let's simulate some data.

```
set.seed(853)

number_of_observation <- 10000

example_data <- tibble(person = c(1:number_of_observation),
                       smoker = sample(x = c("Smoker", "Non-smoker"),
                                       size = number_of_observation,
                                       replace = TRUE),
                       age_died = runif(number_of_observation,
                                        min = 0,
                                        max = 100) %>% round(digits = 0),
                       height = sample(x = c(50:220),
                                       size =  number_of_observation,
                                       replace = TRUE),
                       num_children = sample(x = c(0:5),
                                             size = number_of_observation,
```

```
                                                replace = TRUE,
                                                prob = c(0.1, 0.2, 0.40, 0.15, 0.1, 0.05))
                                )
```

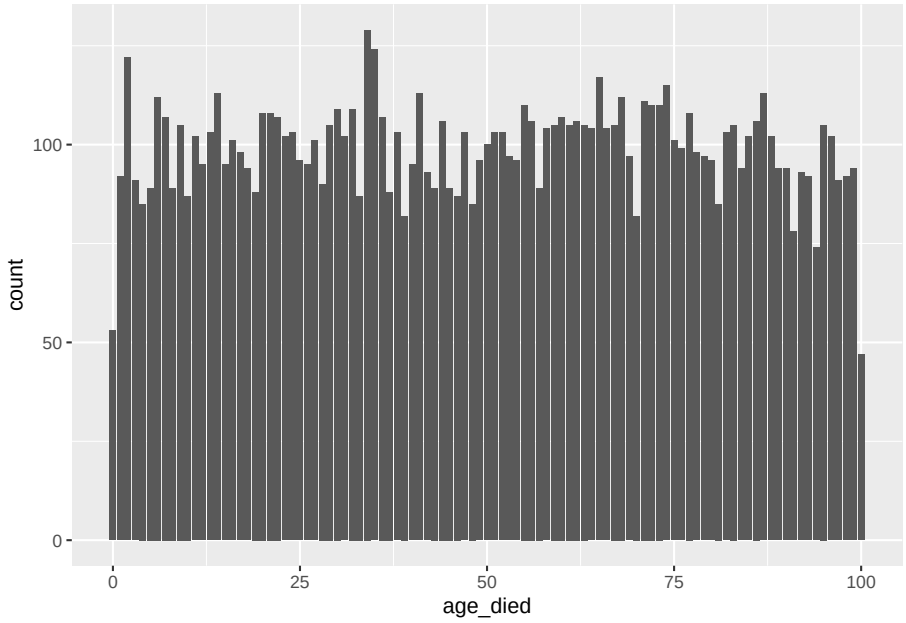
First, let's have a look at the data.

```
head(example_data)
```

```
## # A tibble: 6 x 5
##   person smoker     age_died height num_children
##    <int> <chr>         <dbl>  <int>        <int>
## 1      1 Smoker           55     80            3
## 2      2 Non-smoker       54     78            2
## 3      3 Non-smoker       84    109            1
## 4      4 Smoker           75    114            4
## 5      5 Smoker           32    135            1
## 6      6 Smoker           37    220            0
```

Now let's plot the age distribution. Based on our simulated data, we're expecting a fairly uniform plot.

```
example_data %>%
  ggplot(mapping = aes(x = age_died)) +
  geom_bar()
```



Now let's make it look a little better. There are themes that are built into ggplot, or you can install other themes from other packages, or you can edit

aspects yourself. I'd recommend starting with the `ggthemes` package for some fun ones, but I tend to just use classic or minimal. Remember that you must always refer to your graphs in your text (Figure **??**).

```
example_data %>%
  ggplot(mapping = aes(x = age_died)) +
  geom_bar() +
  theme_minimal() +
  labs(x = "Age died",
       y = "Number",
       title = "Number of people who died at each age",
       caption = "Source: Simulated data.")
```

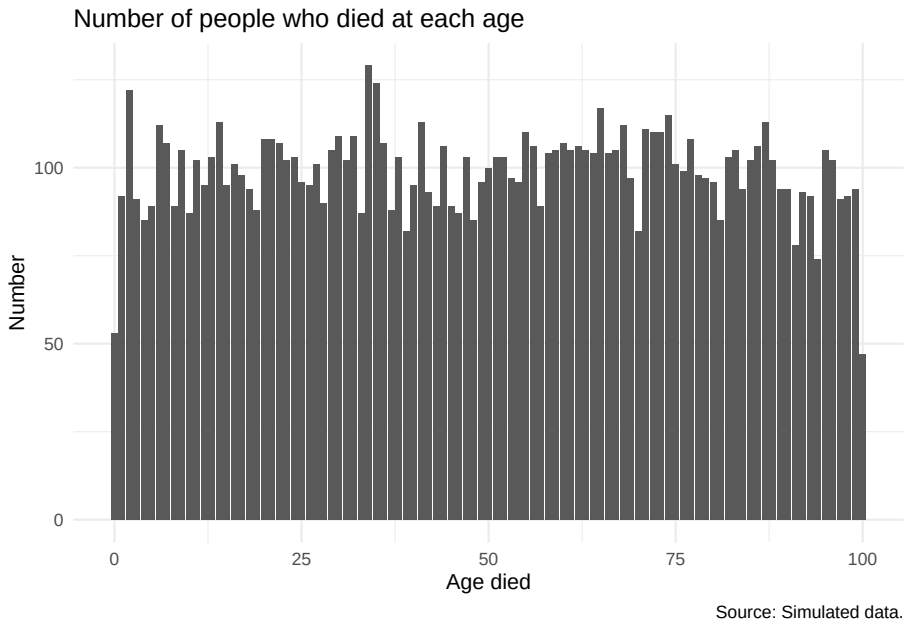


Figure 4.2: Number of people who died at each age

We may want to facet by some variable, in this case whether the person is a smoker (Figure **??**).

```
example_data %>%
  ggplot(mapping = aes(x = age_died)) +
  geom_bar() +
  theme_minimal() +
  facet_wrap(vars(smoker)) +
  labs(x = "Age died",
       y = "Number",
       title = "Number of people who died at each age, by whether they smoke",
```

```
       caption = "Source: Simulated data.")
```

Figure 4.3: Number of people who died at each age, by whether they smoke

Alternatively, we may wish to colour by that instead (Figure **??**). I'll filter to just a handful of age-groups to keep it tractable.

```
example_data %>%
  filter(age_died < 25) %>%
  ggplot(mapping = aes(x = age_died, fill = smoker)) +
  geom_bar(position = "dodge") +
  theme_minimal() +
  labs(x = "Age died",
       y = "Number",
       fill = "Smoker",
       title = "Number of people who died at each age, by whether they smoke",
       caption = "Source: Simulated data.")
```

It's important to recognise that a boxplot hides the full distribution of a variable. Unless you need to communicate the general distribution of many variables at once then you should not use them. The same box plot can apply to very different distributions.

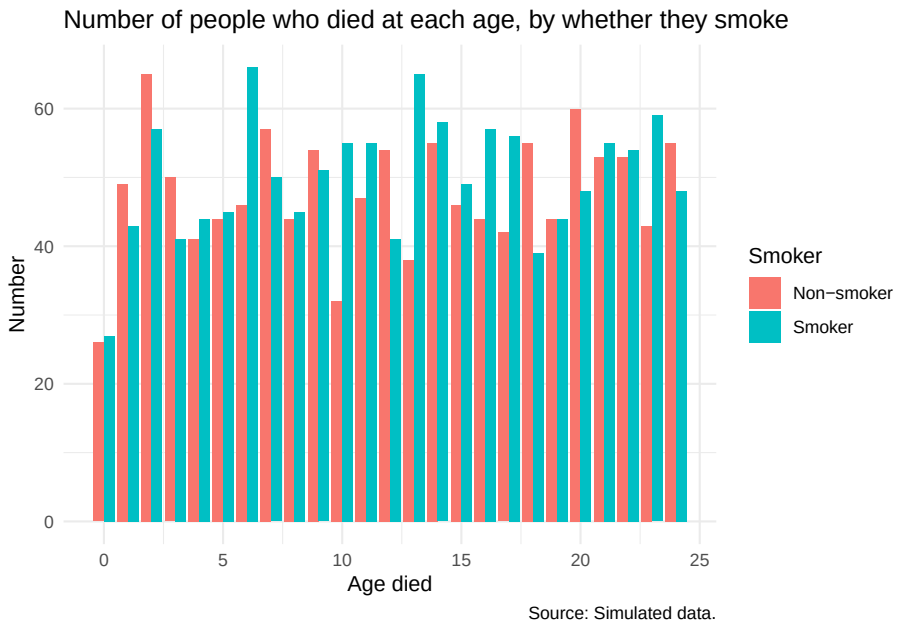


Figure 4.4: Number of people who died at each age, by whether they smoke

4.2.2 Scatter plot

Often, we are also interested in the relationship between two series. We'll do that with a scatter plot. In this case, let's simulate some data, say years of education and income.

```
set.seed(853)

number_of_observation <- 500

scatter_data <-
  tibble(years_of_education = runif(n = number_of_observation, min = 10, max = 25),
         error = rnorm(n= number_of_observation, mean = 0, sd = 10000),
         ) %>%
  mutate(income = years_of_education * 5000 + error,
         income = if_else(income < 0, 0, income))

head(scatter_data)
```

```
## # A tibble: 6 x 3
##   years_of_education   error income
##                <dbl>   <dbl>  <dbl>
## 1               15.4 -13782. 63180.
```

```
## 2              11.8   7977. 66985.
## 3              17.3  -9787. 76498.
## 4              14.7  12999. 86689.
## 5              10.6  -1500. 51302.
## 6              16.1   1911. 82202.
```

Now let's look at income as a function of years of education (Figure **??**).

```
scatter_data %>%
  ggplot(mapping = aes(x = years_of_education, y = income)) +
  geom_point() +
  theme_minimal() +
  labs(x = "Years of education",
       y = "Income",
       title = "Relationship between income and years of education",
       caption = "Source: Simulated data.")
```

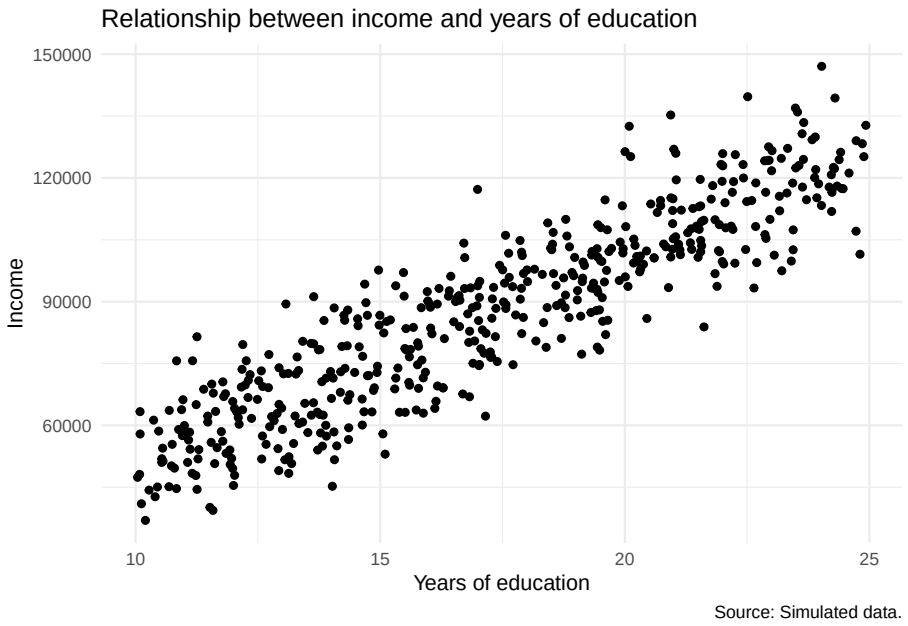


Figure 4.5: Relationship between income and years of education

4.2.3 Other

4.2.3.1 Best fit

If we're interested in quickly adding a line of best fit then, continuing with the earlier income example, we can do that with `geom_smooth()` (Figure **??**).

```
scatter_data %>%
  ggplot(mapping = aes(x = years_of_education, y = income)) +
  geom_point() +
  geom_smooth(method = lm, color = "black") +
  theme_minimal() +
  labs(x = "Years of education",
       y = "Income",
       title = "Relationship between income and years of education",
       caption = "Source: Simulated data.")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

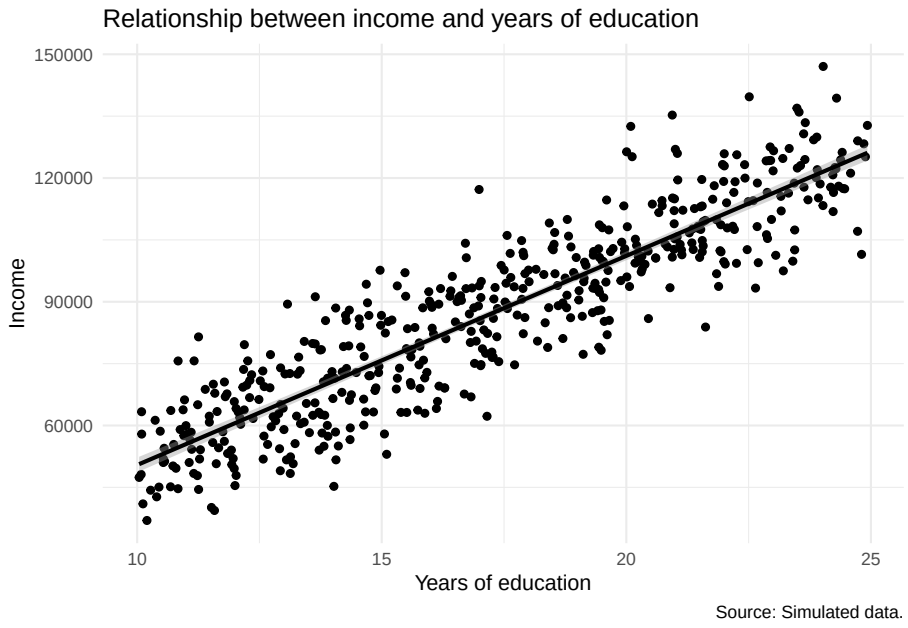


Figure 4.6: Relationship between income and years of education

4.2.3.2 Histogram

If we want to get counts by groups then we may want to use a histogram. Figure **??** shows the counts for our simulated incomes.

```
scatter_data %>%
  ggplot(mapping = aes(x = income)) +
  geom_histogram() +
  theme_minimal() +
  labs(x = "Income",
       y = "Number",
```

```
       title = "Distribution of income",
       caption = "Source: Simulated data.")
```

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

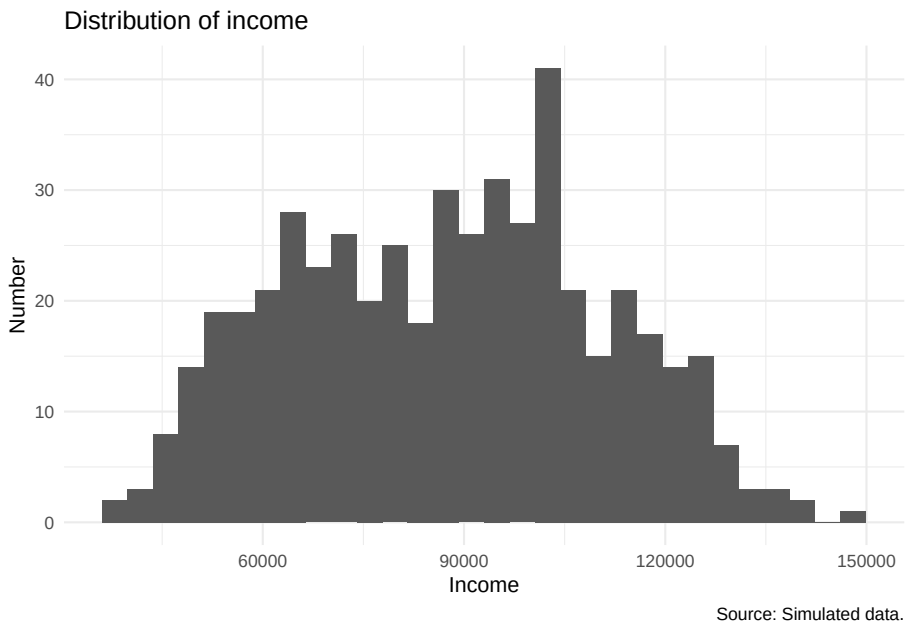


Figure 4.7: Distribution of income

4.2.3.3 Multiple plots

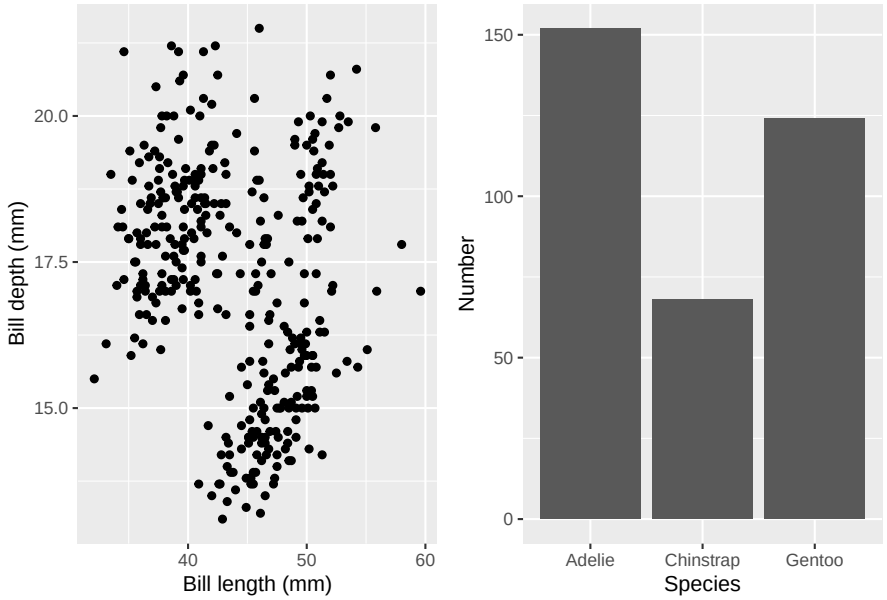
Finally, let's try putting them together. We're going to use the `patchwork` package (**?**) and the `penguins` package for data. Don't forget `install.packages("palmerpenguins")` as this is probably the first time you've used the package.

```
library(patchwork)
library(palmerpenguins)

p1 <-
  ggplot(palmerpenguins::penguins) +
  geom_point(aes(bill_length_mm, bill_depth_mm)) +
  labs(x = "Bill length (mm)",
       y = "Bill depth (mm)")
p2 <-
  ggplot(palmerpenguins::penguins) +
  geom_bar(aes(species)) +
  labs(x = "Species",
```

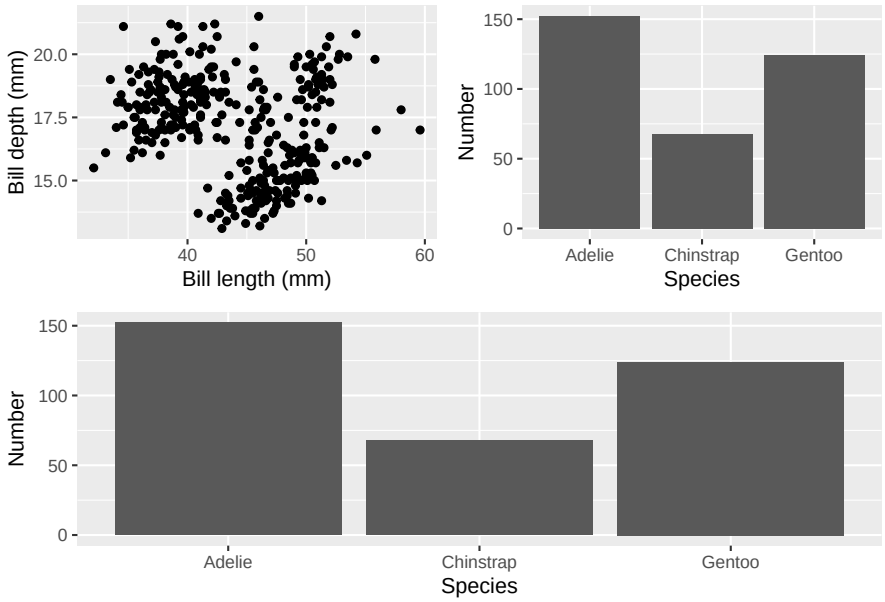
```
       y = "Number")

p1 + p2
```



And we can make things fairly involved fairly quickly.

```
(p1 | p2) /
  p2
```



4.3 Tables

Tables are also critical to tell a compelling story. We may prefer a table to a graph when there are only a few features that we want to focus on. We'll use `knitr::kable()` alongside the 'kableExtra' package and also the `gt` package.

Let's start with the kable package and the summary dinosaur data from earlier.

```
example_data <-
  datasaurus_dozen %>%
  filter(dataset %in% c("dino", "star", "away")) %>%
  group_by(dataset) %>%
  summarize(
    Mean    = mean(x),
    Std_dev = sd(x),
    )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
example_data %>%
  knitr::kable()
```

dataset	Mean	Std_dev
away	54.26610	16.76983
dino	54.26327	16.76514
star	54.26734	16.76896

Table 4.1: My first table.

Dataset	Mean	Standard deviation
away	54.27	16.77
dino	54.26	16.77
star	54.27	16.77

Even the defaults are pretty good, but we can add a few tweaks to make the table better. The first is that this many significant digits is inappropriate, we may also like to add a caption, make the column names consistent, and change the alignment.

```
example_data %>%
  knitr::kable(digits = 2,
               caption = "My first table.",
               col.names = c("Dataset", "Mean", "Standard deviation"),
               align = c('l', 'l', 'l')
               )
```

The ''kableExtra' package builds extra functionality (**?**).

The `gt` package (**?**) is a newer package that brings a lot of exciting features. However, being newer it sometimes has issues with PDF output.

```
library(gt)

example_data %>%
  gt()
```

dataset	Mean	Std_dev
away	54.26610	16.76982
dino	54.26327	16.76514
star	54.26734	16.76896

We could add sub-titles easily.

```
example_data %>%
  gt() %>%
  tab_header(
    title = "Summary stats can be misleading",
    subtitle = "With an example from a dinosaur!"
  )
```

Summary stats can be misleading

With an example from a dinosaur!

dataset	Mean	Std_dev
away	54.26610	16.76982
dino	54.26327	16.76514
star	54.26734	16.76896

One common reason for needing a table is to report regression results. You should consider `gtsummary`, `stargazer`, and `modelsummary`. But at the mo-

ment, my favourite is `modelsummary` (**?**).

```
library(modelsummary)

mod <- lm(y ~ x, datasaurus_dozen)
modelsummary(mod)
```

	Model 1
(Intercept)	53.590
	(2.119)
x	-0.106
	(0.037)
Num.Obs.	1846
R2	0.004
R2 Adj.	0.004
AIC	17383.0
BIC	17399.6
Log.Lik.	-8688.506
F	8.072

4.4 Title, abstract, and introduction

A title is the first opportunity that you have to tell the reader your story. Ideally you will tell the reader exactly what you found. An effective title is critical in order to get your work read when there are other competing priorities. A title doesn't have to be 'cute' to be great.

- Good: 'On the 2019 Canadian Federal Election'. (At least the reader knows what the paper is about.)
- Better: 'The Liberal Party performance in the 2019 Canadian Federal Election'. (At least the reader knows what the paper is about more specifically.)
- Even better: 'The Liberal Party did poorly in rural areas in the 2019 Canadian Federal Election'. (The reader knows what the paper is about.)

You should put your name and the date on the paper because this provides an important context to the paper.

For a six-page paper, a good abstract is a three to five sentence paragraph. For a longer paper your abstract can be slightly longer. The abstract should say: What you did, what you found, and why the reader should care. Each of these should just be a sentence or two, so keep it very high level.

You should then have an introduction that tells the reader everything they need to know. You are not writing a mystery story - tell the reader the most important points in the introduction. For a six-page paper, your introduction

may be two or three paragraphs. Four would likely be too much, but it depends on the context.

Your introduction should set the scene and give the reader some background. For instance, you may like to start of a little broader, to provide some context to your paper. You should then describe how your paper fits into that context. Then give some high-level results - provide more detail than you provided in the abstract, but don't get into the weeds - and finally broadly discuss next steps or glaring weaknesses. With regard to that high-level result: you need to pick one. If you have a bunch of interesting findings, then good for you, but pick one and write your introduction around that. If it's compelling enough then the reader will end up reading all your other interesting findings in the discussion/results sections. Finally, you should highlight the remainder of the paper.

As an example:

> The Canadian Liberal Party has always struggled in rural ridings. In the past 100 years they have never won more than 25 per cent of them. But even by those standards the 2019 Federal Election was a disappointment with the Liberal Party winning only 2 of the 40 rural ridings.
>
> In this paper we look at why the performance of the Liberal Party in this most recent election was so poor. We construct a model in which whether the Liberal Party won the riding is explained by the number of farms in the riding, the average internet connectivity, and the median age. We find that as the median age of a riding increases, the likelihood that a riding was won by the Liberal Party decreases by 14 percentage points. Future work could expand the time horizon that is considered which would allow a more nuanced understanding of these effects.
>
> The remainder of this paper is structured as follows: Section 2 discusses the data, Section 3 discusses the model, Section 4 presents the results, and finally Section 5 discusses our findings and some weaknesses.

The recommended readings provide some lovely examples of titles, abstracts, and introductions. Please take the time to briefly read these papers.

4.5 Figures, tables, equations, and technical terms

Figure and tables are a critical aspect of convincing people of your story. In a graph you can show your data and then let people decide for themselves. And in a table you can more easily summarise your data.

Figures, tables, equations, etc, should be numbered and then referenced in the text e.g. "Figure 1 shows…" and then have Figure 1.

You should make sure that all aspects of your graph are legible. Always label all of the axes. Your graphs should have titles, and the point that you want to communicate should be clear.

If you use a technical term, then it should be briefly explained in plain language for readers who might not be familiar with it. A great example of this is this post by Monica Alexander where she explains the Gini coefficient:

> To look at the concentration of baby names, let's calculate the Gini coefficient for each country, sex and year. The Gini coefficient measures dispersion or inequality among values of a frequency distribution. It can take any value between 0 and 1. In the case of income distributions, a Gini coefficient of 1 would mean one person has all the income. In this case, a Gini coefficient of 1 would mean that all babies have the same name. In contrast, a Gini coefficient of 0 would mean names are evenly distributed across all babies.

4.6 On brevity

The PM's vanishing briefs

Tim Shipman Political Editor

Boris Johnson's aides have been ordered to send him shorter memos, limiting papers to just two sides of A4. Civil servants have also been told to cut the number of documents put into the prime minister's red box to "make sure that he reads them".

The edict emerged after Johnson spent a week away at his Chevening country house. Members of the Downing Street policy unit were told to provide "weekend reading" for the prime minister on keys aspects of policy. But a source said: "They've been told it should be an easy read: no more than four pages, or he's never going to read it. Two pages is preferable."

Another said Johnson's private office and his closest aide, Dominic Cummings, had put a cap on what goes into the red box. An official said: "Box submissions have to be brief if he is going to read it. If they're overly long or overly complex, Dom sends them back with savage comments."

A Whitehall source accused Johnson and Cummings of running "government by ADHD" – attention deficit hyperactivity disorder.

The details emerged as it was revealed that Rishi Sunak, the new chancellor, will use his first budget to move parts of the Treasury to the north of England; that Johnson told Sajid Javid, the former chancellor, last week that he "regrets" his resignation; and that MI5 officers have reduced the amount of intelligence they pass to Priti Patel because they "do not trust" the home secretary.

Reports, page 2 and pages 14-15

Figure 4.8: 'No more than four pages, or he's never going to read it. Two pages is preferable.'

Source: Shipman, Tim, 2020, "The prime minister's vanishing briefs', The Sunday Times, 23 February, available at: https://www.thetimes.co.uk/article/the-prime-ministers-vanishing-briefs-67mt0bg95 via Sarah Nickson.

> Insisting on two page briefs is sensible - not 'government by ADHD'. PM has to be across lots of issues - cannot and should not be across (most of) them in the same depth as secretaries of state. Danger lies in PM trying to take on too much and getting bogged down in

> detail.
>
> This might irk officials who lack a sense of where their issue sits within the PM's list of priorities - or the writing skills to draft a succinct brief. But there'd be very few occasions when a brief to the PM warrants more than two pages.
>
> This is not something peculiar to the current PM - other ministers have raised the same in interviews with @instituteforgov Oliver Letwin complained of 'huge amount of terrible guff, at huge, colossal, humungous length coming from some departments' https://www.instituteforgovernment.org.uk/ministers-reflect/person/oliver-letwin/
>
> Letwin sent briefs back and asked they be re-drafted to one quarter of the length. 'Somewhere along the line the Civil Service had got used to splurge of the meaningless kind' Similarly, Theresa Villiers talked about the civil service's 'frustrating tendency to produce six pages of obscure and rather impenetrable text' and wishes she'd be firmer in sending documents back for re-drafting: https://www.instituteforgovernment.org.uk/ministers-reflect/person/theresa-villiers/
>
> Sarah Nickson, 23 Feb 2020.

Brevity is important. Partly this because you are writing for the reader, not yourself, and your reader has other priorities. But it is also because as the writer it focuses you to consider what your most important points are, how you can best support them, and where your arguments are weakest.

If you don't think that examples from government are persuasive, then please consider Amazon's 2017 Letter to Shareholders, or other statements about Bezos and memo writing, for instance:

> Well structured, narrative text is what we're after rather than just text… The reason writing a 4 page memo is harder than "writing" a 20 page powerpoint is because the narrative structure of a good memo forces better thought and better understanding of what's more important than what, and how things are related.
>
> Jeff Bezos, 9 June 2004.

4.7 Other

Typos and other grammatical mistakes affect the credibility of your claims. If the reader can't trust you to use a spell-checker then why should they trust you to use logistic regression? Microsoft Word has a fantastic spell-checker that is much better than what is available for R Markdown: copy/paste your work into there, look for the red lines and fix them in your R Markdown. Then look for

the green lines and think about if you need to fix them in your R Markdown. If you don't have Word then Google Docs is pretty good and so is Apple's Pages.

A few other general tips that I have stolen from various people including the Reserve Bank of Australia's style guide:

- Think about what you are writing. Aim to write everything as though it were on the front page of the newspaper, because one day it could be.
- Be concise. Remove as many words as possible.
- Be direct. Think about the structure of your story, and identify the key pieces of information and arrange them so that your paper flows logically from one to the next. You should use sub-headings if you need.
- Be precise. For instance, the stock-market didn't improve or worsen, it rose or fell. Distinguish levels from rates of change.
- Be clear.
- Write simply.
- Use short sentence where possible.
- Avoid jargon.

You should break these rules when you need to. But the only way to know whether you need to break a rule is to know the rules in the first instance.

Chapter 5

Maps

- TODO: Add something about geocoding.

Required reading

- Cooley, David, 2020, 'mapdeck', freely available at: https://symbolixau.github.io/mapdeck/index.html.
- Kolb, Jan-Philipp, 2019, 'Using Web Services to Work with Geodata in R', *The R Journal*, 11:2, pages 6-23, freely available at: https://journal.r-project.org/archive/2019/RJ-2019-041/index.html.
- Gabrielle, 2019, 'Visualising spatial data using sf and mapdeck - part one', 4 December, freely available at: https://resources.symbolix.com.au/2019/12/04/mapdeck-1/.
- 'Leaflet for R', freely available at: https://rstudio.github.io/leaflet/.

Required viewing

- Kuriwaki, Shiro, 2020, 'Making maps in R with sf', 1 March, freely available at: https://vimeo.com/394800836.

Recommended reading

- Engel, Claudia A, 2019, *Using Spatial Data with R*, 11 February, Chapter 3 Making Maps in R, freely available at: https://cengel.github.io/R-spatial/mapping.html.
- Lovelace, Robin, Jakub Nowosad, Jannes Muenchow, 2020, *Geocomputation with R*, 29 March, Chapter 8, Making maps with R, freely available at: https://geocompr.robinlovelace.net/adv-map.html.

Key concepts/skills/etc

- Thinking of maps as a (often fiddly, but strangely enjoyable) variant of a usual ggplot.

- Broadening the data that we make available via interactive maps, while still telling a clear story.
- Becoming comfortable with (and excited about) creating both static and interactive maps.

Key libraries

- `ggmap`
- `leaflet`
- `maps`
- `mapdeck`

Key functions/etc

- `add_arc()`
- `addCircleMarkers()`
- `addMarkers()`
- `addTiles()`
- `canada.cities`
- `geom_polygon()`
- `get_stamenmap()`
- `ggmap()`
- `leaflet()`
- `map()`
- `map_data()`
- `mapdeck()`
- `mapdeck_style()`

5.1 Introduction

In many ways maps can be thought of as a fancy graph, where the x-axis is latitude, the y-axis is longitude, and there is some outline or a background image. We are used to this type of set-up, for instance, in a ggplot setting that is quite familiar.

```
ggplot() +
  geom_polygon( # First draw an outline
    data = some_data,
    aes(x = latitude,
        y = longitude,
        group = group
        )) +
  geom_point( # Then add points of interest
    data = some_other_data,
    aes(x = latitude,
        y = longitude)
    )
```

And while there are some small complications, for the most part it is as straightforward as that. The first step is to get some data. And helpfully, there is some geographic data built into ggplot, and there is some other information built into a package called `maps`.

```
library(maps)
library(tidyverse)

canada <- map_data(database = "world", regions = "canada")
canadian_cities <- maps::canada.cities

head(canada)
```

```
##        long      lat group order region    subregion
## 1 -59.78760 43.93960     1     1 Canada Sable Island
## 2 -59.92227 43.90391     1     2 Canada Sable Island
## 3 -60.03775 43.90664     1     3 Canada Sable Island
## 4 -60.11426 43.93911     1     4 Canada Sable Island
## 5 -60.11748 43.95337     1     5 Canada Sable Island
## 6 -59.93604 43.93960     1     6 Canada Sable Island
```

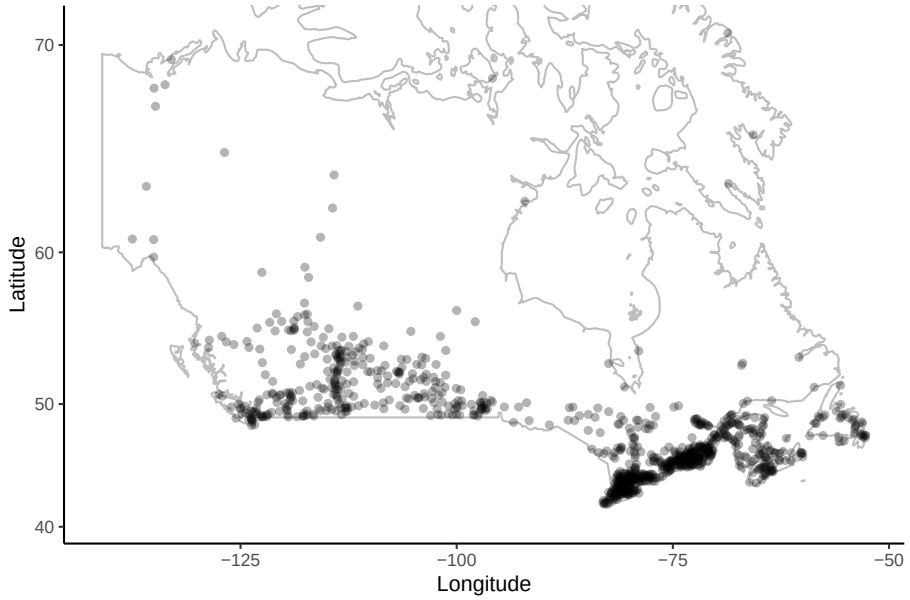
```
head(canadian_cities)
```

```
##            name country.etc    pop   lat    long capital
## 1 Abbotsford BC          BC 157795 49.06 -122.30       0
## 2      Acton ON          ON   8308 43.63  -80.03       0
## 3 Acton Vale QC          QC   5153 45.63  -72.57       0
## 4    Airdrie AB          AB  25863 51.30 -114.02       0
## 5    Aklavik NT          NT    643 68.22 -135.00       0
## 6    Albanel QC          QC   1090 48.87  -72.42       0
```

With that information in hand we can then create a map of Canada that shows the cities with a population over 1,000. (The `geom_polygon()` function within `ggplot` draws shapes, by connecting points within groups. And the `coord_map()` function adjusts for the fact that we are making something that is 2D map to represent something that is 3D.)

```
ggplot() +
  geom_polygon(data = canada,
               aes(x = long,
                   y = lat,
                   group = group),
               fill = "white",
               colour = "grey") +
  coord_map(ylim = c(40, 70)) +
  geom_point(aes(x = canadian_cities$long,
                 y = canadian_cities$lat),
             alpha = 0.3,
```

```
            color = "black") +
  theme_classic() +
  labs(x = "Longitude",
       y = "Latitude")
```




```
# If I'm being honest, this 'simple example' took me six hours to work out. Firstly
# to find Canada and then to find Canadian cities.
```

In this section we will go through two types of maps: static and interactive. Static maps will be useful for printed output, such as a PDF or Word report, or where there is something in particular that you want to illustrate. Interactive maps will be more useful in an online setting, where you want your users to be able to explore the data themselves. (Further, they are a great way to take advantage of having your own website.)

5.2 Static maps

As is often the case with R, there are many different ways to get started create static maps. We've already seen how they can be built using simply ggplot, but here we'll explore one package that has a bunch of functionality built in that will make things easier: `ggmap`.

There are two essential components to a map: 1) some border or background image (also known as a tile); and 2) something of interest within that border or on top of that tile. In `ggmap`, we will use an open source option for our tile,

Stamen Maps (maps.stamen.com), and we will use plot points based on latitude and longitude.

5.2.0.1 Australian polling places

Like Canada, in Australia people go to specific locations, called booths, to vote. These booths have latitudes and longitudes and so we can plot these. One reason we may like to do this is to notice patterns over geographies.

To get started we need to get a tile. We are going to use `ggmap` to get a tile from Stamen Maps, which builds on OpenStreetMap (openstreetmap.org). The main argument to this function is to specify a bounding box. This requires two latitudes - one for the top of the box and one for the bottom of the box - and two longitudes - one for the left of the box and one for the right of the box. (It can be useful to use Google Maps, or an alternative, to find the values of these that you need.) The bounding box provides the coordinates of the edges that you are interested in. In this case I have provided it with coordinates such that it will be centered around Canberra, Australia (our equivalent of Ottawa - a small city that was created for the purposes of being the capital).

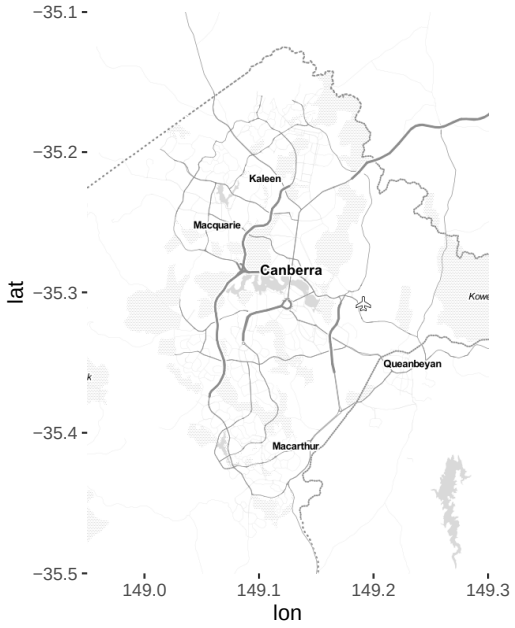
```
library(ggmap)

bbox <- c(left = 148.95, bottom = -35.5, right = 149.3, top = -35.1)
```

Once you have defined the bounding box, then the function `get_stamenmap()` will get the tiles in that area. The number of tiles that it needs to get depends on the zoom, and the type of tiles that it gets depends on the maptype. I've chosen the maptype that I like here - the black and white option - but the helpfile specifies a few others that you may like. At this point you can pass your maps to ggmap and it will plot the tile! It will be actively downloading these tiles, so you need an internet connection.

```
canberra_stamen_map <- get_stamenmap(bbox, zoom = 11, maptype = "toner-lite")

ggmap(canberra_stamen_map)
```



Once we have a map then we can use `ggmap()` to plot it. (That circle in the middle of the map is where the Australian Parliament House is... yes, our parliament is surrounded by circular roads (we call them 'roundabouts'), actually it's surrounded by two of them.)

Now we want to get some data that we will plot on top of our tiles. We will just plot the location of the polling places, based on which 'division' (the Australian equivalent to 'ridings' in Canada) it is. This is available here: https://results.aec.gov.au/20499/Website/Downloads/HouseTppByPollingPlaceDownload-20499.csv. (The Australian Electoral Commission (AEC) is the official government agency that is responsible for elections in Australia.)

```
# Read in the booths data for each year
booths <- readr::read_csv("https://results.aec.gov.au/24310/Website/Downloads/GeneralP
                          skip = 1,
                          guess_max = 10000)

head(booths)
```

```
## # A tibble: 6 x 15
##   State DivisionID DivisionNm PollingPlaceID PollingPlaceTyp~ PollingPlaceNm
##   <chr>      <dbl> <chr>               <dbl>            <dbl> <chr>
## 1 ACT          318 Bean                93925                5 Belconnen BEA~
## 2 ACT          318 Bean                93927                5 BLV Bean PPVC
## 3 ACT          318 Bean                11877                1 Bonython
```

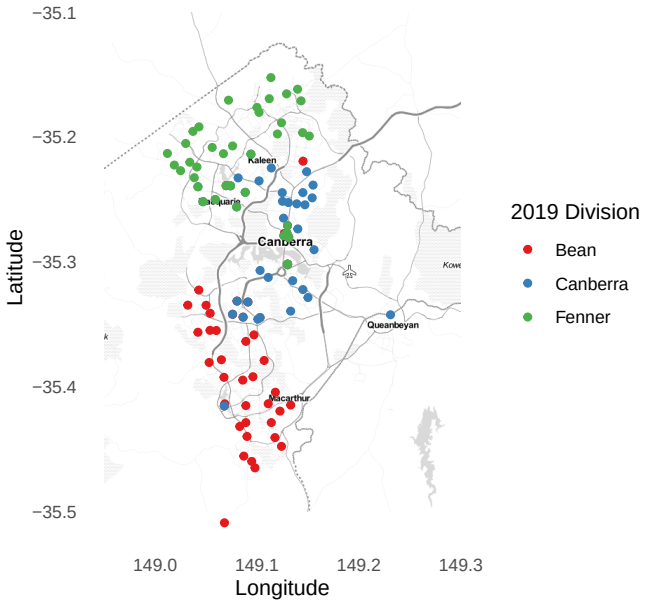
```
## 4 ACT          318 Bean                11452                1 Calwell
## 5 ACT          318 Bean                 8761                1 Chapman
## 6 ACT          318 Bean                 8763                1 Chisholm
## # ... with 9 more variables: PremisesNm <chr>, PremisesAddress1 <chr>,
## #   PremisesAddress2 <chr>, PremisesAddress3 <chr>, PremisesSuburb <chr>,
## #   PremisesStateAb <chr>, PremisesPostCode <chr>, Latitude <dbl>,
## #   Longitude <dbl>
```

This dataset is for the whole of Australia, but as we are just going to plot the area around Canberra we will filter to that and only to booths that are geographic (the AEC has various options for people who are in hospital, or not able to get to a booth, etc, and these are still 'booths' in this dataset).

```
# Reduce the booths data to only rows with that have latitude and longitude
booths_reduced <-
  booths %>%
  filter(State == "ACT") %>%
  select(PollingPlaceID, DivisionNm, Latitude, Longitude) %>%
  filter(!is.na(Longitude)) %>% # Remove rows that don't have a geography
  filter(Longitude < 165) # Remove Norfolk Island
```

Now we can use `ggmap` in the same way as before to plot our underlying tiles, and then build on that using `geom_point()` to add our points of interest.

```
ggmap(canberra_stamen_map,
      extent = "normal",
      maprange = FALSE) +
  geom_point(data = booths_reduced,
             aes(x = Longitude,
                 y = Latitude,
                 colour = DivisionNm),
             ) +
  scale_color_brewer(name = "2019 Division", palette = "Set1") +
  coord_map(projection="mercator",
            xlim=c(attr(map, "bb")$ll.lon, attr(map, "bb")$ur.lon),
            ylim=c(attr(map, "bb")$ll.lat, attr(map, "bb")$ur.lat)) +
  labs(x = "Longitude",
       y = "Latitude") +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



We may like to save the map so that we don't have to draw it every time, and we can do that in the same way as any other graph, using `ggsave()`.

```
ggsave("outputs/figures/map.pdf", width = 20, height = 10, units = "cm")
```

Finally, the reason that I used Stamen Maps and OpenStreetMap is because it is open source, however you can also use Google Maps if you want. This requires you to first register a credit card with Google, and specify a key, but with low usage should be free. The `get_googlemap()` function with `ggmap`, brings some nice features that `get_stamenmap()` does not have. For instance, you can enter a placename and it'll do it's best to find it rather than needing to specify a bounding box.

5.2.0.2 Toronto bike parking

Let's see another example of a static map, this time using Toronto data accessed via the `opendatatoronto` package. The dataset that we are going to plot is available here: https://open.toronto.ca/dataset/street-furniture-bicycle-parking/.

```
# This code is based on code from: https://open.toronto.ca/dataset/street-furniture-bi
library(opendatatoronto)
# (The string identifies the package.)
resources <- list_package_resources("71e6c206-96e1-48f1-8f6f-0e804687e3be")
# In this case there is only one dataset within this resource so just need the first o
raw_data <- filter(resources, row_number()==1) %>% get_resource()
```

```
write_csv(raw_data, "inputs/data/bike_racks.csv")
head(raw_data)
```

Now that we've saved a copy of the data, we can use that one. First we need to clean it up a bit. There are some clear errors in the ADDRESSNUMBERTEXT field, but not too many, so we'll just ignore it.

```
raw_data <- read_csv("inputs/data/bike_racks.csv")
# We'll just focus on the data that we want
bike_data <- tibble(ward = raw_data$WARD,
                    id = raw_data$ID,
                    status = raw_data$STATUS,
                    street_address = paste(raw_data$ADDRESSNUMBERTEXT, raw_data$ADDRESSSTREET),
                    latitude = raw_data$LATITUDE,
                    longitude = raw_data$LONGITUDE)
rm(raw_data)
```

Some of the bike racks were temporary so remove them and also let's just look at the area around the university, which is Ward 11

```
# Only keep ones that still exist
bike_data <-
  bike_data %>%
  filter(status == "Existing") %>%
  select(-status)

bike_data <- bike_data %>%
  filter(ward == 11) %>%
  select(-ward)
```

If you look at the dataset at this point then you'll notice that there is a row for every bike parking spot. But we don't really need to know that, because sometimes there are lots right next to each other. Instead we'd just like the one point (we'll take advantage of this in an interactive graph in a moment). So we want to create a count by address, and then just get one instance per address.

```
bike_data <-
  bike_data %>%
  group_by(street_address) %>%
  mutate(number_of_spots = n(),
         running_total = row_number()
         ) %>%
  ungroup() %>%
  filter(running_total == 1) %>%
  select(-id, -running_total)

head(bike_data)
```

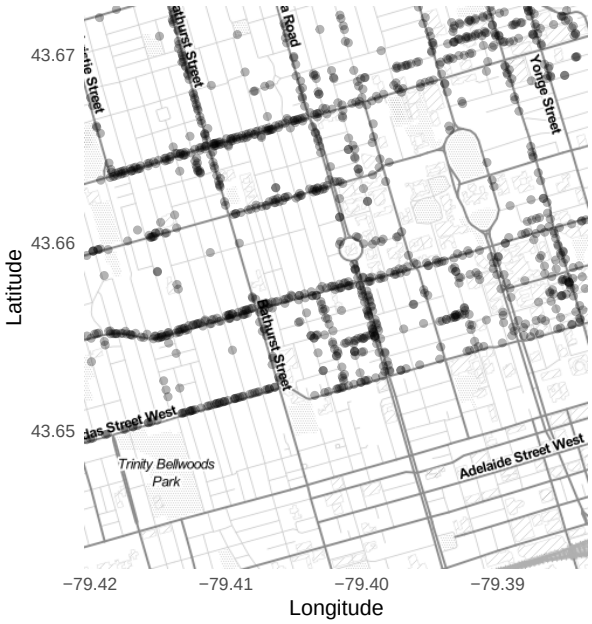
```
## # A tibble: 6 x 4
##   street_address   latitude longitude number_of_spots
##   <chr>               <dbl>     <dbl>           <int>
## 1 8 Kensington Ave     43.7     -79.4               1
## 2 87 Avenue Rd         43.7     -79.4               4
## 3 162 Mc Caul St       43.7     -79.4               1
## 4 147 Baldwin St       43.7     -79.4               2
## 5 888 Yonge St         43.7     -79.4               1
## 6 180 Elizabeth St     43.7     -79.4              10
```

Now we can grab our tile, and add our bike rack data onto it.

```
bbox <- c(left = -79.420390, bottom = 43.642658, right = -79.383354, top = 43.672557)

toronto_stamen_map <- get_stamenmap(bbox, zoom = 14, maptype = "toner-lite")

ggmap(toronto_stamen_map,  maprange = FALSE) +
  geom_point(data = bike_data,
             aes(x = longitude,
                 y = latitude),
             alpha = 0.3
             ) +
  labs(x = "Longitude",
       y = "Latitude") +
  theme_minimal()
```



5.3 Interactive maps

The nice thing about interactive maps is that you can let your users decide what they are interested in. Additionally, if there is a lot of information then you may like to leave it to your users as to selectively focus on what they are interested in. For instance, in the case of Canadian politics, some people will be interested in Toronto ridings, while others will be interested in Manitoba, etc. But it would be difficult to present a map that focuses on both of those, so an interactive map is a great option for allowing users to zoom in on what they want.

5.3.0.1 Leaflet

The `leaflet` package is originally a JavaScript library of the same name that has been brought over to R. It makes it easy to make interactive maps. The basics are fairly similar to the `ggmap` set-up, but of course after that, there are many, many, options.

Let's redo the bike map from earlier, and possibly the interaction will allow us to see what the issue is with the data.

In the same way as a graph in `ggplot` begins with the `ggplot()` function, a map in the `leaflet` package begins with a call to the `leaflet()` function. This allows you to specify data, and a bunch of other options such as width and height. After this, we add 'layers', in the same way that we added them in `ggplot`. The first layer that we'll add is a tile with the function `addTiles()`. In this case, the default is from OpenStreeMap. After that we'll add markers that show the location of each bike parking spot with `addMarkers()`.

```
library(leaflet)

leaflet(data = bike_data) %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng = bike_data$longitude,
             lat = bike_data$latitude,
             popup = bike_data$street_address,
             label = ~as.character(bike_data$number_of_spots))
```

There are two options here that may not be familiar. The first is popup, and this is what happens when you click on the marker. In this example this is giving the address. The second is label, which is what happens when you hover over the marker. In this example it is given the number of spots.

5.3.0.2 COVID-19

Let's have another go, this time with Ontario data on COVID-19.

We can download the latest data from the Ontario Data Catalogue. This is a fast moving situation in which they are likely to make breaking changes to this

dataset. To ensure these notes work, I will save and then use the dataset as at 4 April 2020, but you are able to get the up-to-date dataset using the link and the code.

```
ontario_covid <- read_csv("https://data.ontario.ca/datastore/dump/455fd63b-603d-4608-8
write_csv(ontario_covid, "inputs/data/ontario_covid_2020-04-04.csv")
```

```
ontario_covid <- read_csv("inputs/data/ontario_covid_2020-04-04.csv")
head(ontario_covid)
```

```
## # A tibble: 6 x 14
##   `_id` ROW_ID ACCURATE_EPISODE_D~ Age_Group CLIENT_GENDER CASE_ACQUISITIO~
##   <dbl>  <dbl> <dttm>              <chr>     <chr>         <chr>
## 1     1      1 2020-03-07 00:00:00 40s       MALE          Neither
## 2     2      2 2020-03-08 00:00:00 20s       MALE          Neither
## 3     3      3 2020-03-10 00:00:00 40s       FEMALE        Neither
## 4     4      4 2020-03-11 00:00:00 50s       FEMALE        Neither
## 5     5      5 2020-03-12 00:00:00 30s       FEMALE        Neither
## 6     6      6 2020-03-15 00:00:00 50s       MALE          Neither
## # ... with 8 more variables: OUTCOME1 <chr>, Reporting_PHU <chr>,
## #   Reporting_PHU_Address <chr>, Reporting_PHU_City <chr>,
## #   Reporting_PHU_Postal_Code <chr>, Reporting_PHU_Website <chr>,
## #   Reporting_PHU_Latitude <dbl>, Reporting_PHU_Longitude <dbl>
```

There is a lot of information here, but we'll just plot the number of cases, by the reporting area (health areas). So this isn't the location of the person, but the location of the responsible health unit. Because of this, we'll add a little bit of noise so that the marker for each person can be seen. We do this with `jitter()`.

```
ontario_covid <-
  ontario_covid %>%
  mutate(Reporting_PHU_Latitude = jitter(Reporting_PHU_Latitude, amount = 0.1),
         Reporting_PHU_Longitude = jitter(Reporting_PHU_Longitude, amount = 0.1))
```

We will introduce a different type of marker here, which is circles. This will allow us to use different colours for the outcomes of each case. There are three possible outcomes: the case is resolved, it is not resolved, or it was fatal.

```
library(leaflet)

pal <- colorFactor("Dark2", domain = ontario_covid$OUTCOME1 %>% unique())

leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addCircleMarkers(
    data = ontario_covid,
    lng = ontario_covid$Reporting_PHU_Longitude,
```

```
    lat = ontario_covid$Reporting_PHU_Latitude,
    color = pal(ontario_covid$OUTCOME1),
    popup = paste("<b>Age-group:</b>", as.character(ontario_covid$Age_Group), "<br>",
                  "<b>Gender:</b>", as.character(ontario_covid$CLIENT_GENDER), "<br>",
                  "<b>Acquisition:</b>", as.character(ontario_covid$CASE_ACQUISITIONINFO), "<br>"
                  "<b>Episode date:</b>", as.character(ontario_covid$ACCURATE_EPISODE_DATE), "<br
    ) %>%
  addLegend("bottomright",
            pal = pal,
            values = ontario_covid$OUTCOME1 %>% unique(),
    title = "Case outcome",
    opacity = 1
  )
```

5.3.0.3 mapdeck

Thank you to Shaun Ratcliff for introducing me to mapdeck.

The package `mapdeck` is an R package that is built on top of Mapbox (https://www.mapbox.com). It is based on WebGL, which means that your web browser does a lot of work for you. The nice thing is that because of this, it can do a bunch of things that leaflet struggles with, especially dealing with larger datasets. Mapbox is a full-featured application that many businesses that you may have heard of use: https://www.mapbox.com/showcase. To close out these notes on mapping, I want to briefly touch on `mapdeck`, as it is a newer, but very exciting, package.

To this point we have used stamen maps as our tile, but mapdeck uses mapbox - https://www.mapbox.com/ - and so you need to register and get a token for this. (It's free.) Once you have that token you add it to R using:

```
library(mapdeck)
set_token("asdf") # replace asdf with your token.
mapdeck_tokens()

set_token(test$key)
```

(Don't add it into your script otherwise everyone will be able to take it and use it, especially once you add it to GitHub.)

Then we need some data. Here we're going to just use the example dataset, which is about flights.

```
# Code taken from the example: https://github.com/SymbolixAU/mapdeck
library(mapdeck)

url <- 'https://raw.githubusercontent.com/plotly/datasets/master/2011_february_aa_flight_paths.cs
flights <- read.csv(url)
```

```
flights$info <- paste0("<b>",flights$airport1, " - ", flights$airport2, "</b>")

head(flights)
```

```
##   start_lat start_lon  end_lat    end_lon airline airport1 airport2 cnt
## 1  32.89595 -97.03720 35.04022 -106.60919      AA      DFW      ABQ 444
## 2  41.97960 -87.90446 30.19453  -97.66987      AA      ORD      AUS 166
## 3  32.89595 -97.03720 41.93887  -72.68323      AA      DFW      BDL 162
## 4  18.43942 -66.00183 41.93887  -72.68323      AA      SJU      BDL  56
## 5  32.89595 -97.03720 33.56294  -86.75355      AA      DFW      BHM 168
## 6  25.79325 -80.29056 36.12448  -86.67818      AA      MIA      BNA  56
##               info
## 1 <b>DFW - ABQ</b>
## 2 <b>ORD - AUS</b>
## 3 <b>DFW - BDL</b>
## 4 <b>SJU - BDL</b>
## 5 <b>DFW - BHM</b>
## 6 <b>MIA - BNA</b>
```

Finally, we can call the map. Again, this is just the example in the package's website.

```
mapdeck(style = mapdeck_style('dark')
        ) %>%
  add_arc(
    data = flights
    , origin = c("start_lon", "start_lat")
    , destination = c("end_lon", "end_lat")
    , stroke_from = "airport1"
    , stroke_to = "airport2"
    , tooltip = "info"
    , layer_id = 'arclayer'
  )
```

And this is pretty nice!

Chapter 6

Making a website

6.1 Getting started with Blogdown

Required reading

- Hill, Alison, 2017, 'Up & Running with blogdown', 12 June, freely available at: https://alison.rbind.io/post/2017-06-12-up-and-running-with-blogdown/.
- Salmon, Maëlle, 2020, 'What to know before you adopt Hugo/blogdown', 29 February, freely available at: https://masalmon.eu/2020/02/29/hugo-maintenance/.
- Xie, Yihui, Amber Thomas, and Alison Presmanes Hill, 2020, *blogdown: Creating Websites with R Markdown*', as at 6 February, freely available at: https://bookdown.org/yihui/blogdown/.

Recommended reading

- Hill, Alison, 2019, 'A Spoonful of Hugo: Troubleshooting Your Build', 4 March, freely available at: https://alison.rbind.io/post/2019-03-04-hugo-troubleshooting/.
- De Leon, Desirée, 2019, 'Trying out Blogdown', 4 September, freely available at: http://desiree.rbind.io/post/2019/trying-out-blogdown/.
- Navarro, Danielle, 2018, 'Day 1: Getting started with blogdown', 27 April, freely available at: https://djnavarro.net/post/starting-blogdown/.

Key concepts/skills/etc

- Building a website using blogdown and hugo.

Key libraries

- `blogdown`
- `tidyverse`

6.2 Introduction

These notes were originally developed for a workshop at the ANU delivered in 2017, and published to my website. Thank you to Minhee Chae and Peter Gibbard for helpful comments.

A website is a critical part of communication. If you are searching for a job then it acts as one place to bring everything that you can do together. If you are using R, then you might like a website that makes it easy to share your work. This is where `blogdown` helps.

`blogdown` is a package that allows you to make websites (not just blogs, notwithstanding its name) largely within R Studio. It builds on Hugo, which is a popular tool for making websites. `blogdown` lets you freely and quickly get a website up-and-running. It is easy to add content from time-to-time. It integrates with R Markdown which lets you easily share your work. And the separation of content and styling allows you to relatively quickly change your website's design.

That said, using `blogdown` is more work than Google sites or Squarespace. It requires a little more knowledge than using a basic Wordpress site. And if you want to customise many aspects of your website, or need everything to be 'just so' then `blogdown` may not be for you. `blogdown` is still under active development and various aspects may break in future releases. That said, the investment of time required to set up a `blogdown` website is unlikely to be wasted. Even if `blogdown` were shuttered tomorrow most of the content could be repurposed for a regular Hugo website.

This post is a simplified version of the `blogdown` user-guide and the blog post by Alison Presmanes Hill. It sticks to the basics and doesn't require much decision-making. The purpose is to allow someone without much experience to use `blogdown` to get a website up-and-running. Head to those two resources once you've got a website working and want to dive a bit deeper.

6.3 Foundations

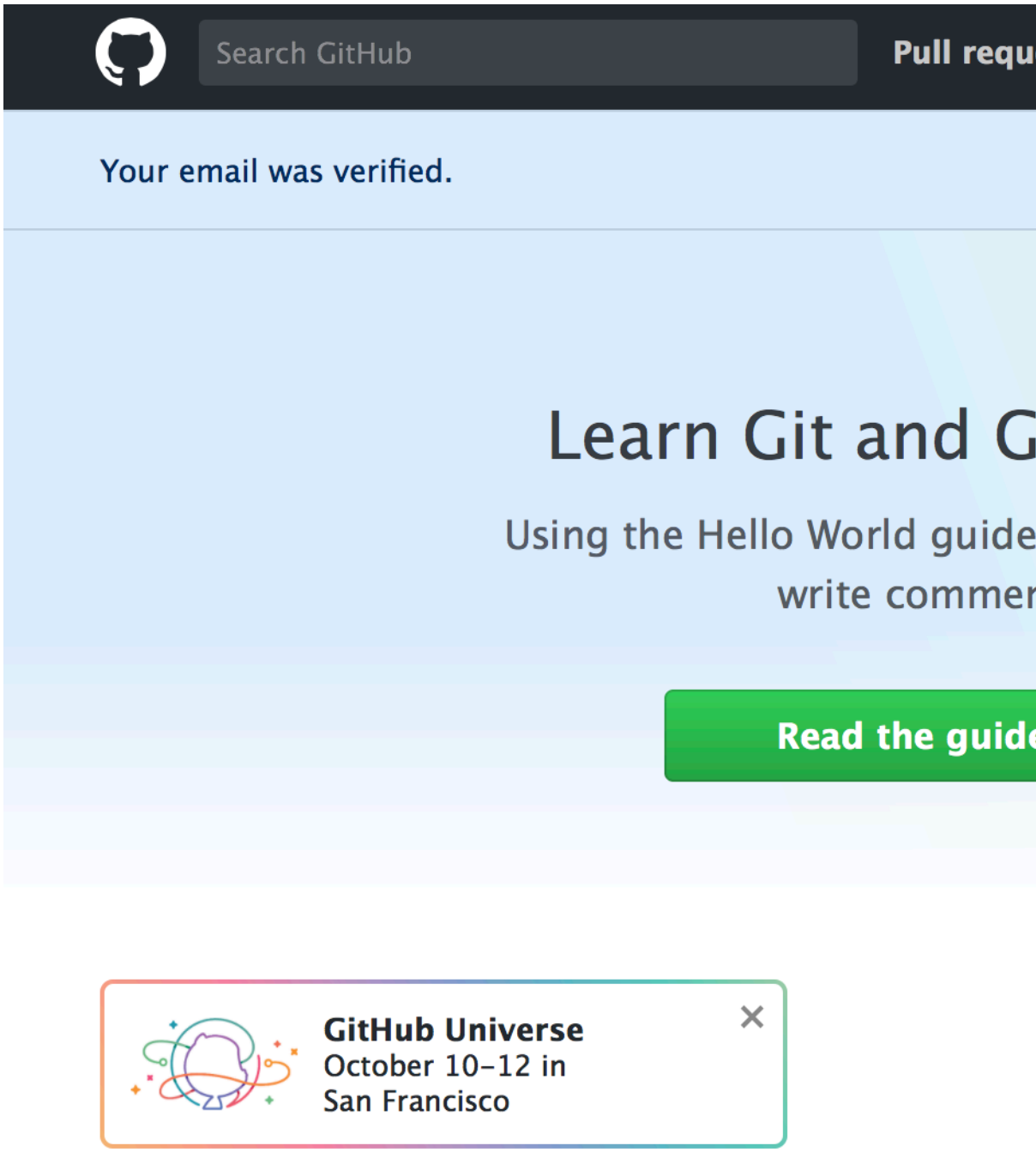
To use `blogdown` you need R and R Studio, but if you have made it this far in the course then you probably know that. We'll install the `blogdown` package then use GitHub to create a new folder where we'll create our website.

First install blogdown:

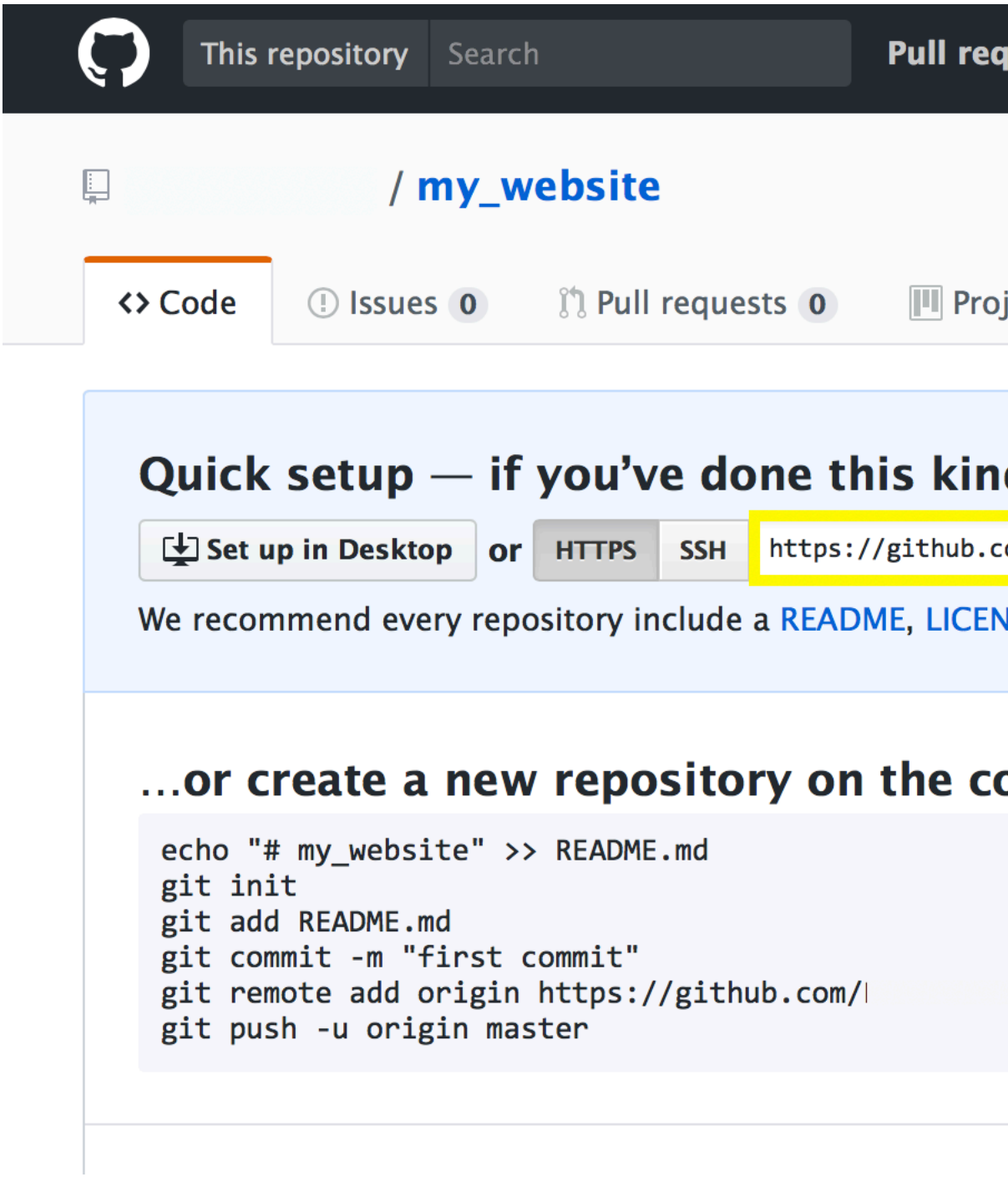
```
install.packages("blogdown")
```

Now we want to create a folder in GitHub (because it will be easier to put your website onto the internet if you have a GitHub account). We have seen GitHub earlier in this notes, but if you didn't do this at that point, then please create a free GitHub account at https://github.com/.

Once you have an account, create a new repository by clicking on the plus and call it 'my_website' (or whatever you want).


Search GitHub
Pull requ
Your email was verified.
Learn Git and G
Using the Hello World guide
write commer
Read the guide
GitHub Universe
October 10–12 in
San Francisco

Don't worry about including a readme or gitignore. Once you get to the 'Quick setup' page, copy the website address.



At this point, we want to get that folder onto our own computer. So open

RStudio, select `Files`, `New Project`, `Version Control`, `Git`, and paste the information for the repo. Go through the rest of it, saving the folder somewhere sensible, and clicking 'Open in new session'. This will then create a new folder on your computer which will be a Git folder that is linked to the GitHub repo that you created.

We can now construct a frame for our website in that folder.

6.4 Build the frame

Open R Studio and install Hugo via the blogdown package with the following code:

```
blogdown::install_hugo()
```

In R Studio create a new project in the folder that you just created 'my_website'. To do this click on: File -> New Project -> Existing Directory. Then navigate to the folder 'my_website'. This will open a new R Studio session. Creating a project just adds a .proj file in the folder that makes it easier to come back to your website later.

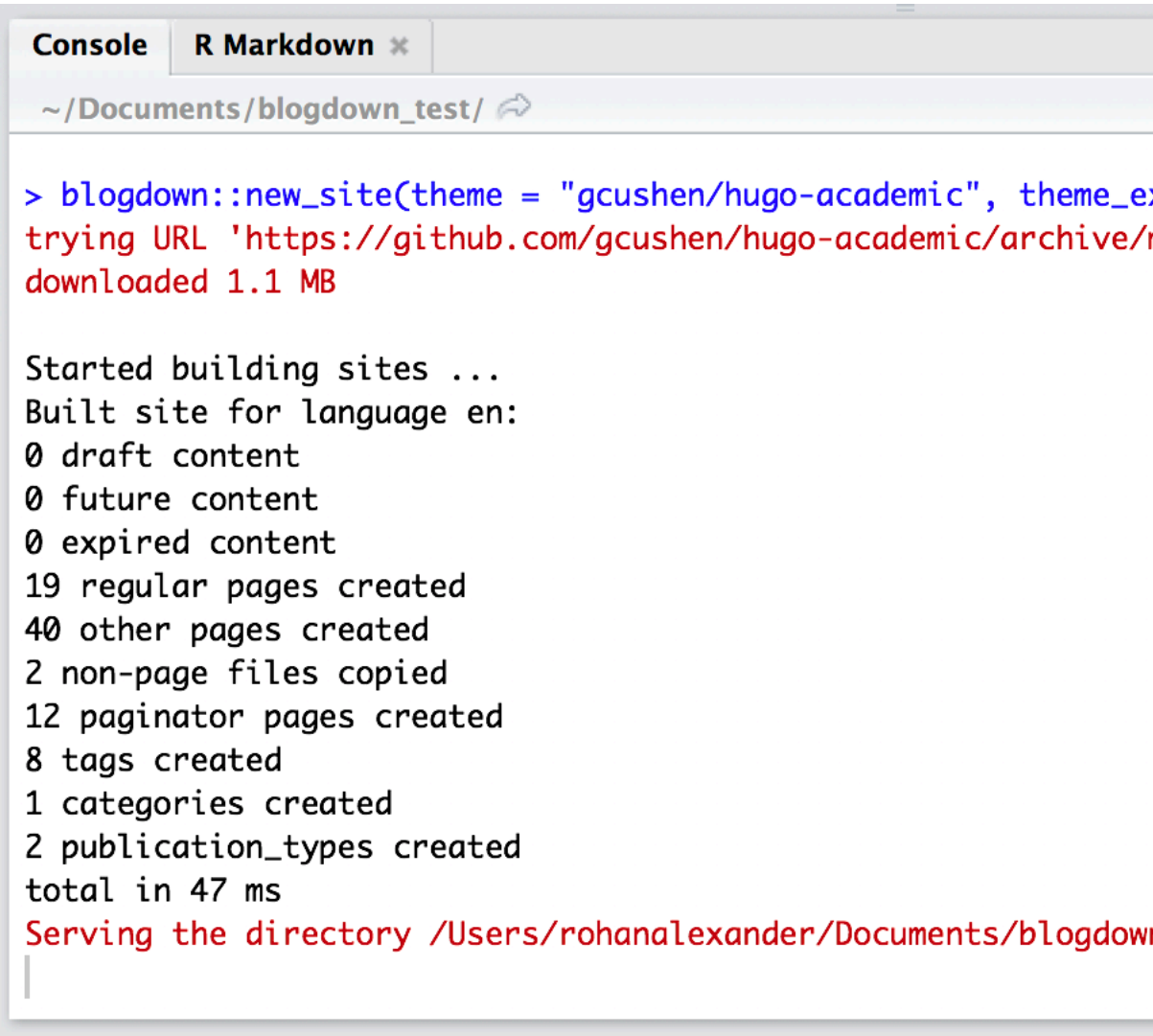
Using that new R Studio session create your website with the following code:

```
blogdown::new_site(theme = "gcushen/hugo-academic", theme_example = TRUE)
```

This will:

- download files into your 'my_website' folder;
- open a R Markdown file that you can close for now; and
- begin serving the site in your R Studio viewer.

The console and viewer of your R Studio session should look like this:



Now that we have a frame, we can add our own content.

6.5 Add content

At this point, the default website is being 'served' locally. This means that changes you make will be reflected in the website that you see in your R Studio Viewer. To see the website in a web browser click the 'show in new window' button on the top left of the Viewer. This is circled in the above image. That will open the website using the address that the R Studio also tells you.

6.5.0.1 Headshot

The first change to make is to update the headshot. In your folder, go to my_website -> static -> img. Replace 'portrait.jpg' with your own square headshot jpg. If you do this correctly then when you go back to your website the image will have updated.

6.5.0.2 Personal details, contacts, and main menu

To update the biography and other details in that first pane, go to File -> Open File in the R Studio menu and open config.toml which is in my_website -> config.toml. This file will either open in a text editor or in R Studio – it doesn't matter which. When you save the file the changes will be reflected in the website.

Search for 'title' or go to line 2. It should say:

```
'title = "Academic"'
```

Change that to:

```
'title = "Your Name"'
```

Search for '[params]' or go to line 21. There you can update parameters such as name, role, and contact details. If you don't want a particular parameter to show up on your website then set it equal to "". (An example of this is on line 33.)

Once you've updated these parameters, search for '[[params.social]]' or go to line 126. There you can update your contact details, such as email, twitter, etc. Just delete or comment out the full four lines if you don't want a particular contact type displayed on your website. You can always add more later.

Finally, search for '[[menu.main]]' or go to line 152. There you can change the menu items that are displayed on the top right of your website. For instance if you don't want a blog then delete or comment out the four lines:

```
[[menu.main]]
  name = "Posts"
  url = "#posts"
  weight = 3
```

If you want to change the order of the items then change the 'weight'. Ascending values from left to right.

6.5.0.3 Biography

In your folder, go to my_website -> content -> home -> about.md. That should open in R Studio or your text editor. Any changes that you save should immediately show up in your website.

Search for '# List your academic interests.' or go to line 12. There you can change your academic interests. If you don't want this to show up on your website then you can just delete or comment out lines 12-18.

Search for '# List your qualifications (such as academic degrees).' or go to line 20. There you can change your academic qualification. If you don't want this to show up on your website then you can just delete or comment out these lines.

The 'year' is a numeric field. If you'd prefer to include duration (e.g. 2013 – 2017), then replace the '2012' with ' "2013 – 2017" ' (the "" are important). Or similarly, if you are expecting a degree then you could replace the 'year' with ' "Expected month year" '.

Search for '# Biography' or go to line 43. There you can add a brief biography.

6.5.0.4 Teaching

Most of the other files in my_website -> content -> home just display content from elsewhere. This is because of the setup of the website. The exception is teaching.md. Open that and edit everything after line 15.

6.5.0.5 Publications

In your folder, go to my_website -> content -> publication. There are two default publications added there. You can edit those and then copy them to add extra publications.

6.5.0.6 Posts

If you want a blog in your website then the content is saved in: my_website -> content -> post. If you don't want a blog then just delete this folder and comment out the posts menu item from my_website -> config.toml file so it doesn't show up in the menu.

Once your website is working, if you want a new blog post, then you can simply use the R Studio menu bar: Tools -> Addins -> New Post.

6.5.0.7 Etc

Go through the different parts and change it as you need.

6.5.0.8 Subsequent editing

To come back to editing your website once you've closed R Studio, go to the 'my_website' folder and then double-click on the Rproj file, 'blogdown_test.Rproj'. That will open a new instance of R Studio.

From there you can type 'blogdown:::serve_site()' into the console to serve your site and then continue editing, or you could use the R Studio menu bar: Tools -> Addins -> Serve Site.

6.6 Making your website public

6.6.0.1 Commit

So far everything has happened on your own computer. The first step to making your website public is to commit these changes to GitHub. To do this open Terminal again and as before use cd and ls to navigate to 'my_website'.

Once there, type each of the following lines (adding your own description) and follow each by 'return'

```
git add -A
git commit -m "DESCRIBE THE CHANGE YOU ARE ADDING"
git push
```

(You may be asked for your GitHub password. Terminal is a bit tricky to type passwords into because you don't know how many characters you've typed, but have a go and follow it by 'return'.)

6.6.0.2 Netlify

There are many ways to make your website public, but one way is to use Netlify. What we are going to do is to link GitHub and Netlify, so that when you make a change in GitHub then Netlify automatically updates. Once you have an account then click "New site from Git" and at that point you can link your GitHub account. Once it has been authorised, then you should select the repo that you want to make public. The publish director is 'public'.

Once this is all specified then you can "Deploy site". Netlify gives you a default address, but you can change this. However it will still have .netlify.com. To get rid of this you need to purchase a domain, and then go through the custom domains setting in Netlify.

Chapter 7

Shiny

TODO: ADD CONTENT

Part III

Hunting and Gathering (Data)

Chapter 8

Gathering data

Required reading

- Cooksey, Brian, 2014, 'An Introduction to APIs', Zapier, 22 April, https://zapier.com/learn/apis/.
- Gelfand, Sharla, 2019, 'Crying @ Sephora', 8 November, https://sharla.party/post/crying-sephora/.
- Luscombe, Alex, 2020, 'Getting your .pdfs into R', 5 August, https://alexluscombe.ca/post/r-pdftools/.
- Luscombe, Alex, 2020, 'Parsing your .pdfs in R', 10 August, https://alexluscombe.ca/post/parsing-pdfs/.
- Silge, Julia and David Robinson, 2020, *Text Mining with R*, Chapters 1, 3, and 6, https://www.tidytextmining.com/.
- Wickham, Hadley, nd, 'Getting started with httr', https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html.
- Wickham, Hadley, 2014, 'rvest: easy web scraping with R', 24 November, https://blog.rstudio.com/2014/11/24/rvest-easy-web-scraping-with-r/.

Recommended reading

- Alexander, Rohan, 2019, 'Gathering and analysing text data', 3 January, https://rohanalexander.com/posts/2019-01-03-gathering-and-analysing-text-data/. (Example of web-scraping.)
- Benoit, Kenneth, 2019, 'Text as data: An overview', 17 July, https://kenbenoit.net/pdfs/28%20Benoit%20Text%20as%20Data%20draft%202.pdf.
- Bolton, Liza, 2019, 'A quick look at museums per capita', 26 March, http://blog.dataembassy.co.nz/museums-per-capita/. (Example of web-scraping.)
- Bryan, Jennifer, and Jim Hester, 2020, 'What They Forgot to Teach You About R', chapter 7, https://rstats.wtf/index.html.

- Clavelle, Tyler, 2017, 'Using R to extract data from web APIs', 5 June, https://www.tylerclavelle.com/code/2017/randapis/.
- Dogucu, Mine, and Mine Çetinkaya-Runde, 2020,'Web Scraping in the Statistics and Data Science Curriculum: Challenges and Opportunities', 6 May. (Walks through some basics.)
- Goldman, Shayna, 2019, 'How Much Do NHL Players Really Make? Part 2: Taxes', https://hockey-graphs.com/2019/01/08/how-much-do-nhl-players-really-make-part-2-taxes/. (Example of web-scraping.)
- Graham, Shawn, 2019, 'Scraping with rvest', 7 November, https://electricarchaeology.ca/2019/11/07/scraping-with-rvest/. (Example of web-scraping.)
- Henze, Martin, 2020, 'Web Scraping with rvest + Astro Throwback', 23 January, https://heads0rtai1s.github.io/2020/01/23/rvest-intro-astro/. (Example of web-scraping.)
- Hudon, Caitlin, 2017, ''Blue Christmas: A data-driven search for the most depressing Christmas song', 22 December, https://caitlinhudon.com/2017/12/22/blue-christmas/.
- James, Gareth, Daniela Witten, Trevor Hastie and Robert Tibshirani, 2017, *An Introduction to Statistical Learning with Applications in R*, Chapter 6, http://faculty.marshall.usc.edu/gareth-james/ISL/.
- Luscombe, Alex, 2020, 'A Gentle Introduction to Tesseract OCR', 3 June, https://alexluscombe.ca/post/ocr-tutorial/.
- Marshall, James, 'HTML Made Really Easy', https://www.jmarshall.com/easy/html/. (Primer on HTML.)
- Marshall, James, 'HTTP Made Really Easy', https://www.jmarshall.com/easy/http/.
- Nakagawara, Ryo, 2020, 'Intro to {polite} Web Scraping of Soccer Data with R!', 14 May, https://ryo-n7.github.io/2020-05-14-webscrape-soccer-data-with-R/. (Introduction to the `polite` package.)
- Pavlik, Kaylin, 2020, 'How do fiber types appear together in yarn blends?', https://www.kaylinpavlik.com/ravelry-yarn-fibers/.
- Silge, Julia, 2017, 'Scraping CRAN with rvest', 5 March, https://juliasilge.com/blog/scraping-cran/. (Example of web-scraping.)
- Silge, Julia, 2018, 'The game is afoot! Topic modeling of Sherlock Holmes stories', 25 January, https://juliasilge.com/blog/sherlock-holmes-stm/.
- Silge, Julia, 2018, 'Training, evaluating, and interpreting topic models', 8 September, https://juliasilge.com/blog/evaluating-stm/.
- Smale, David, 2020, 'Daniel Johnston', https://davidsmale.netlify.com/portfolio/daniel-johnston/.
- Taddy, Matt, 2019, *Business Data Science*, Chapter 8, pp. 231-259.
- Wickham, Hadley, 'Managing Secrets', https://cran.r-project.org/web/packages/httr/vignettes/secrets.html.

Recommended viewing

- D'Agostino McGowan, Lucy, 2020 'Harnessing the Power of the Web via R Clients for Web APIs', talk at ASA Joint Statistical Meeting 2018, https:

//www.lucymcgowan.com/talk/asa_joint_statistical_meeting_2018/.
- Tatman, Rachel, 2018, 'Character Encoding and You', 21 February, https://youtu.be/2U9EHYqc59Y.

Key concepts/skills/etc

- Use APIs where possible because the data provider has specified the data they would like to make available to you, and the conditions under which they are making it available.
- Often R packages have been written to make it easier to use APIs.
- Use R environments to manage your keys.
- Using the verb GET ('a GET request') means providing a URL and the server will return something, using the verb POST (a POST request') means providing some data and the server will deal with that data.
- Cleaning data
- Graphing data to tell a story
- Respectfully scraping data
- Approaching extracting text from PDFs as a workflow.
- Planning what is needed at the start.
- Starting small and then iterating.
- Putting in place checks.
- Gathering text data.
- Preparing text datasets.

Key libraries

- `babynames`
- `broom`
- `dplyr`
- `ggplot2`
- `glmnet`
- `gutenbergr`
- `janitor`
- `jsonlite`
- `pdftools`
- `purrr`
- `rtweet`
- `rvest`
- `spotifyr`
- `stringi`
- `tidymodels`
- `tidytext`
- `tidyverse`
- `usethis`

Key functions/etc

- `augment()`
- `bind_tf_idf()`

- `cast_dtm()`
- `edit_r_environ()`
- `fromJSON()`
- `geom_segment()`
- `get_artist_audio_features()`
- `get_favorites()`
- `get_my_top_artists_or_tracks()`
- `glance()`
- `glmnet()`
- `gutenberg_download()`
- `html_nodes()`
- `html_text()`
- `map_dfr()`
- `pdf_text()`
- `pmap_dfr()`
- `read_csv()`
- `read_html()`
- `safely()`
- `search_tweets()`
- `SelectorGadget`
- `separate()`
- `separate_rows()`
- `str_detect()`
- `str_replace()`
- `str_squish()`
- `tidy()`
- `tokens()`
- `unnest_tokens()`
- `walk2()`

Quiz

1. In your own words, what is an API (write a paragraph or two)?
2. Find two APIs and discuss how you could use them to tell interesting stories (write a paragraph or two for each).
3. Find two APIs that have an R packages written around them. How could you use these to tell interesting stories? (Write a paragraph or two for each.)
4. What is the main argument to the GET method (pick one)?
   a.
   b.
   c.
   d.
5. Name three reasons why we should be respectful when getting scraping data from websites (write a paragraph or two).
6. What features of a website do we typically take advantage of when we parse the code (select all)?

    a.
    b.
    c.
    d.
7. What are three advantages and three disadvantages of scraping compared with using an API (write a paragraph or two)?
8. Which of these are functions from the `purrr` package (select all)?
    a.
    b.
    c.
    d.
9. What are three delimiters that could be useful when trying to bring order to the PDF that you read in as a character vector (write a paragraph or two)?
10. What do I need to put inside "SOMETHING_HERE" if I want to match regular expressions for a full stop i.e. "." (pick one)? (Hint: See the 'strings' cheatsheet.)
    a.
    b.
    c.
    d.
11. Name three reasons for sketching out what you want before starting to try to extract data from a PDF (write a paragraph or two for each).
12. If you are interested in demographic data then what are three checks that you might like to do? What are three if you are interested in economic data such as GDP, interest rates, and exchange rates? (Write an explanation for each.)
13. What does the `purrr` package do (select all)?
    a.
    b.
    c.
    d.
14. Why should we use `safely()` when scraping data (select all)?
    a.
    b.
    c.
    d.

8.1 APIs

8.1.1 Introduction

In everyday language, and for our purposes, an Application Programming Interface (API) is simply a situation in which someone has set up specific files on their computer such that you can follow their instructions to get them. For

instance, when you use a gif on Slack, Slack asks Giphy's server for the appropriate gif, Giphy's server gives that gif to Slack and then Slack inserts it into your chat. The way in which Slack and Giphy interact is determined by Giphy's API. More strictly, an API is just an application that runs on a server that we access using the HTTP protocol.

In our case, we are going to focus on using APIs for gathering data. So I'll tailor the language that I use toward that, and so:

> [a]n API is the tool that makes a website's data digestible for a computer. Through it, a computer can view and edit data, just like a person can by loading pages and submitting forms. (**?**, Chapter 1)

For instance, you could go to Google Maps and then scroll and click and drag to center the map on Canberra, Australia, or you could just paste this into your browser: https://www.google.ca/maps/@-35.2812958,149.1248113,16z. You just used the Google Maps API.[1]

The advantage of using an API is that the data provider specifies exactly the data that they are willing to provide, and the terms under which they will provide it. These terms may include things like rate limits (i.e. how often you can ask for data), and what you can do with the data (e.g. maybe you're not allowed to use it for commercial purposes, or to republish it, or whatever). Additionally, because the API is being provided specifically for you to use it, it is less likely to be subject to unexpected changes. Because of this it is ethically and legally clear that when an API is available you should try to use it.

In this chapter we introduce some APIs in R. In particular, we will first introduce some R packages that wrap around APIs and make it easier to use an API, and we will then deal directly with APIs.

8.1.2 R packages that wrap around APIs

There are a lot of R packages that wrap around APIs making it easier for you to use an API within 'familiar surroundings'. Here, I'll run through some useful and/or fun ones.

8.1.3 Using APIs directly

In this section we introduce GET requests in which we use an API directly. We will use the `httr` package **?**. A GET request tries to obtain some specific data.

You make a GET request, using, `GET()`, which takes a URL as an argument.

TBD

[1] There are at least six great coffee shops shown just in this section of map including: Mocan & Green Grout; The Cupping Room; Barrio Collective Coffee; Lonsdale Street Cafe; Two Before Ten; and Red Brick. There are also two coffee shops that I love but that most wouldn't classify as 'great' including: The Street Theatre Cafe; and the CBE Cafe.

8.2 Case study - rtweet

Twitter is a rich source of text and other data. The Twitter API is the way in which Twitter ask that you interact with Twitter in order to gather these data. The `rtweet` package (**?**) is built around this API and allows us to interact with it in ways that are similar to using any other R package. Initially all you need a regular Twitter account.

Get started by install the library if you need and then calling it.

```
# install.packages('rtweet')
library(rtweet)
library(tidyverse)
```

To get started we need to authorise rtweet. We start that process by calling a function from the package.

```
get_favorites(user = "RohanAlexander")
```

This will open a browser on your computer, and you will then have to log into your regular Twitter account as shown in Figure **??**.

Once that is done we can actually get my favourites and then save them.

```
rohans_favs <- get_favorites("RohanAlexander")

saveRDS(rohans_favs, "dont_push/rohans_favs.rds")
```

And then looking at the most recent favourite, we can see it was when Professor Bolton tweeted about one of the stellar students in ISSC.

```
rohans_favs %>%
  arrange(desc(created_at)) %>%
  slice(1) %>%
  select(screen_name, text)
```

```
## # A tibble: 1 x 2
##   screen_name text
##   <chr>       <chr>
## 1 Liza_Bolton One of our awesome @UofTStatSci students! I  learning about the ~
```

Let's look at who is tweeting about R, using one of the common R hashtags: #rstats. I've removed retweets so that we hopefully get some actual interesting projects.

```
rstats_tweets <- search_tweets(
  q = "#rstats",
  include_rts = FALSE
)

saveRDS(rstats_tweets, "dont_push/rstats_tweets.rds")
```

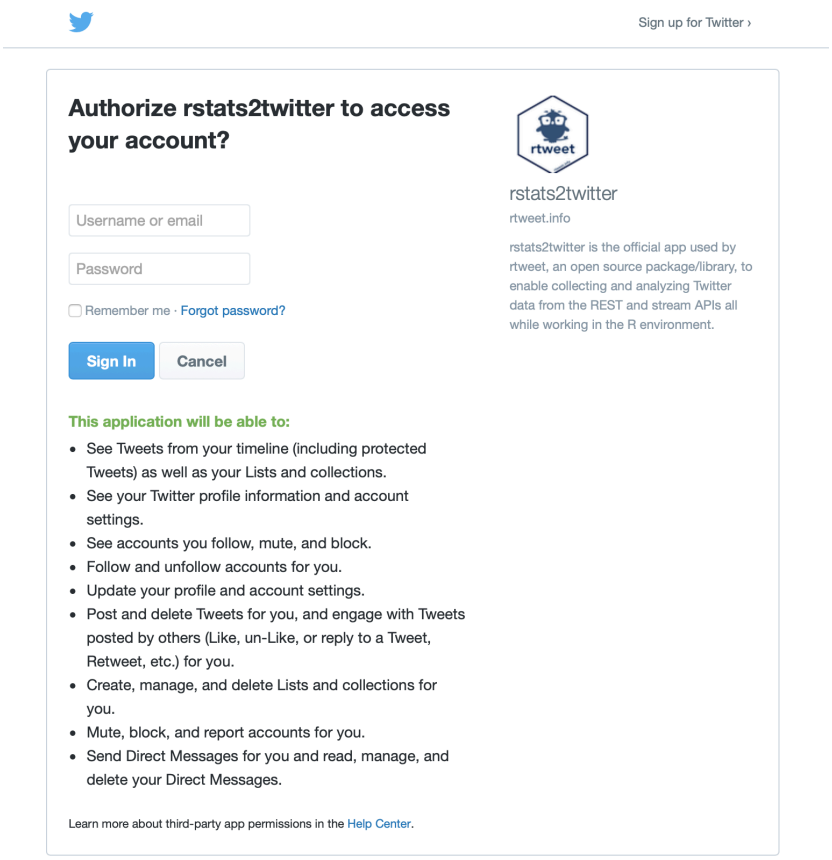


Figure 8.1: rtweet authorisation page

And then have a look at them.

```
names(rstats_tweets)
```

```
##  [1] "user_id"                "status_id"
##  [3] "created_at"             "screen_name"
##  [5] "text"                   "source"
##  [7] "display_text_width"     "reply_to_status_id"
##  [9] "reply_to_user_id"       "reply_to_screen_name"
## [11] "is_quote"               "is_retweet"
## [13] "favorite_count"         "retweet_count"
## [15] "quote_count"            "reply_count"
## [17] "hashtags"               "symbols"
## [19] "urls_url"               "urls_t.co"
## [21] "urls_expanded_url"      "media_url"
## [23] "media_t.co"             "media_expanded_url"
## [25] "media_type"             "ext_media_url"
## [27] "ext_media_t.co"         "ext_media_expanded_url"
## [29] "ext_media_type"         "mentions_user_id"
## [31] "mentions_screen_name"   "lang"
## [33] "quoted_status_id"       "quoted_text"
## [35] "quoted_created_at"      "quoted_source"
## [37] "quoted_favorite_count"  "quoted_retweet_count"
## [39] "quoted_user_id"         "quoted_screen_name"
## [41] "quoted_name"            "quoted_followers_count"
## [43] "quoted_friends_count"   "quoted_statuses_count"
## [45] "quoted_location"        "quoted_description"
## [47] "quoted_verified"        "retweet_status_id"
## [49] "retweet_text"           "retweet_created_at"
## [51] "retweet_source"         "retweet_favorite_count"
## [53] "retweet_retweet_count"  "retweet_user_id"
## [55] "retweet_screen_name"    "retweet_name"
## [57] "retweet_followers_count" "retweet_friends_count"
## [59] "retweet_statuses_count" "retweet_location"
## [61] "retweet_description"    "retweet_verified"
## [63] "place_url"              "place_name"
## [65] "place_full_name"        "place_type"
## [67] "country"                "country_code"
## [69] "geo_coords"             "coords_coords"
## [71] "bbox_coords"            "status_url"
## [73] "name"                   "location"
## [75] "description"            "url"
## [77] "protected"              "followers_count"
## [79] "friends_count"          "listed_count"
## [81] "statuses_count"         "favourites_count"
```

```
## [83] "account_created_at"      "verified"
## [85] "profile_url"             "profile_expanded_url"
## [87] "account_lang"            "profile_banner_url"
## [89] "profile_background_url"  "profile_image_url"
```

```
rstats_tweets %>%
  select(screen_name, text) %>%
  head()
```

```
## # A tibble: 6 x 2
##   screen_name     text
##   <chr>           <chr>
## 1 CRANberriesFeed CRAN updates: gstat hdme opalr tinytest https://t.co/y5W2NTKS~
## 2 CRANberriesFeed New CRAN package FeatureImpCluster with initial version 0.1.2~
## 3 CRANberriesFeed CRAN updates: crfsuite https://t.co/y5W2NTKSXT #rstats
## 4 CRANberriesFeed CRAN updates: DSI https://t.co/y5W2NTKSXT #rstats
## 5 CRANberriesFeed New CRAN package arcos with initial version 1.1 https://t.co/~
## 6 CRANberriesFeed CRAN updates: COVID19 gaiah mosaic smoothSurv https://t.co/y5~
```

There is a bunch of other things that you can do just using a regular user account, and if you're interested then you should try the examples in the `rtweet` package documentation: https://rtweet.info/index.html. But more is available once you register as a developer (https://developer.twitter.com/en/apply-for-access). The Twitter API document is surprisingly readable and you may enjoy some of it: https://developer.twitter.com/en/docs.

When I introduced APIs I said that the 'data provider specifies exactly the data that they are willing to provide…' and we have certainly been able to take advantage of what they provide But I continued '…and the terms under which they will provide it' and here we haven't done our part. In particular, I took some tweets and saved them. If I had pushed these to GitHub then it's possible I may have accidently stored sensitive information if there happened to be some in the tweets. Or if I had taken enough tweets to start to do some reasonable statistical analysis then even if there wasn't sensitive information I may have violated the terms if I had pushed those saved tweets to GitHub. Finally, I linked a Twitter user name, in this case `@Liza_Bolton` with Professor Bolton. I happened to ask her if this was okay, but if I hadn't done that then I would have been violating the Twitter terms of service.

If you use Twitter data, please take a moment to look at the terms: https://developer.twitter.com/en/developer-terms/more-on-restricted-use-cases.

8.3 Case study - spotifyr

For the next example I will introduce the `spotifyr` package (**?**). Again, this is a wrapper that has been developed around an API, in this case the Spotify API.

https://www.rcharlie.com/spotifyr/

```
# devtools::install_github('charlie86/spotifyr')
library(spotifyr)
```

In order to use this account you need a Spotify Developer Account, which you can set-up here: https://developer.spotify.com/dashboard/. That'll have you log in with your Spotify details and then accept their terms (it's worth looking at some of these and I'll follow up on a few below) as in Figure **??**.

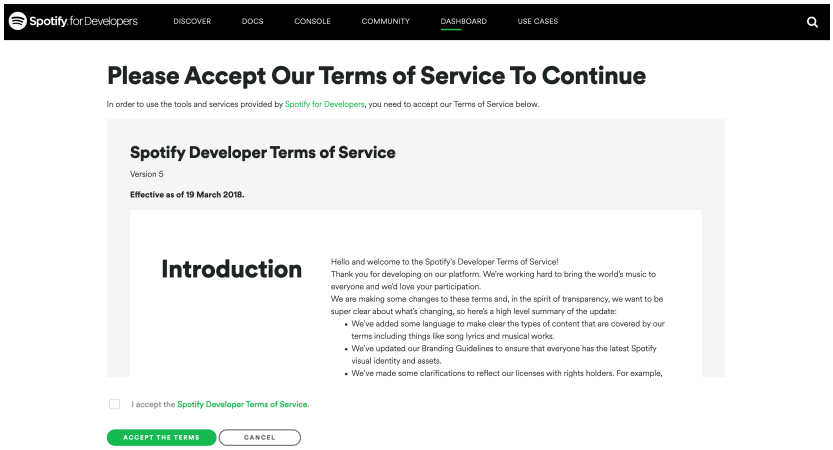


Figure 8.2: rtweet authorisation page

What we need from here is a 'Client ID' and you can just fill out some basic details. In our case we probably 'don't know' what we're building, which means that Spotify requires us to use a non-commercial agreement, which is fine. In order to use the Spotify API we need a Client ID and a Client Secret.

These are things that you want to keep to yourself. There are a variety of ways of keeping this secret, (and my understanding is that a helpful package is on its way) but we'll keep them in our System Environment. In this way, when we push to GitHub they won't be included. To do this we need to be careful about the naming, because `spotifyr` will look in our environment for specifically named keys.

To do this we are going to use the `usethis` package **?**. So if you don't have that then please install it. There is a file called '.Renviron' which we will open and add our secrets to. This file also controls things like your default library location and more information is available at **?** and **?**.

```
usethis::edit_r_environ()
```

When you run that function it will open a file. There you can add your Spotify secrets.

```
SPOTIFY_CLIENT_ID = 'PUT_YOUR_CLIENT_ID_HERE'
SPOTIFY_CLIENT_SECRET = 'PUT_YOUR_SECRET_HERE'
```

Save your '.Renviron' file, and then restart R (Session -> Restart R). You can now draw on that variable when you need.

Some functions that require your secrets as arguments will now just work. For instance, we will get information about Radiohead using `get_artist_audio_features()`. One of the arguments is `authorization`, but as that is set to default to look at the R Environment, we don't need to do anything further.

```
radiohead <- get_artist_audio_features('radiohead')
saveRDS(radiohead, "inputs/radiohead.rds")
```

```
radiohead <- readRDS("inputs/radiohead.rds")

names(radiohead)
```

```
##  [1] "artist_name"                  "artist_id"
##  [3] "album_id"                     "album_type"
##  [5] "album_images"                 "album_release_date"
##  [7] "album_release_year"           "album_release_date_precision"
##  [9] "danceability"                 "energy"
## [11] "key"                          "loudness"
## [13] "mode"                         "speechiness"
## [15] "acousticness"                 "instrumentalness"
## [17] "liveness"                     "valence"
## [19] "tempo"                        "track_id"
## [21] "analysis_url"                 "time_signature"
## [23] "artists"                      "available_markets"
## [25] "disc_number"                  "duration_ms"
## [27] "explicit"                     "track_href"
## [29] "is_local"                     "track_name"
## [31] "track_preview_url"            "track_number"
## [33] "type"                         "track_uri"
## [35] "external_urls.spotify"        "album_name"
## [37] "key_name"                     "mode_name"
## [39] "key_mode"
```

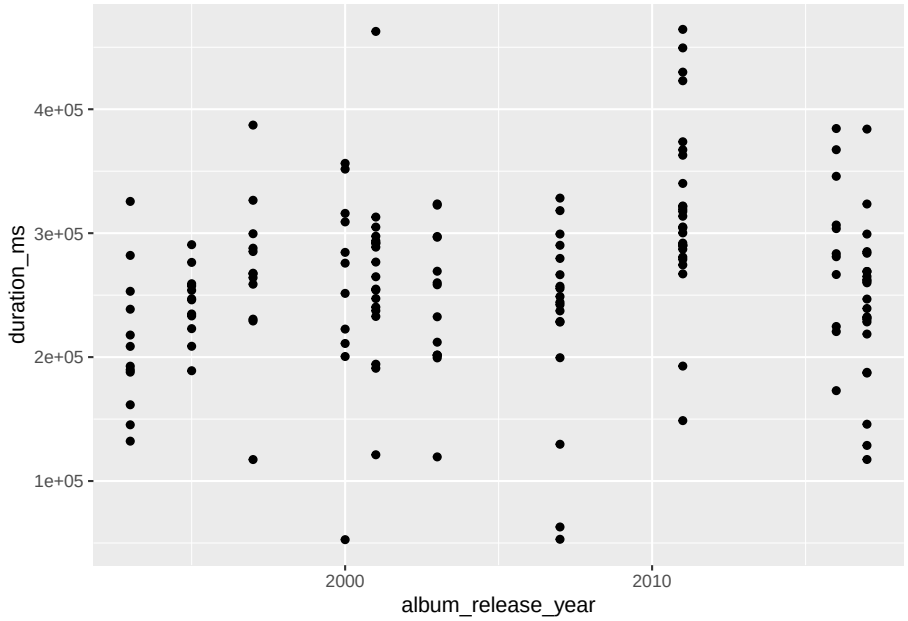
```
radiohead %>%
  select(artist_name, track_name, album_name) %>%
  head()
```

```
##   artist_name                              track_name
## 1   Radiohead                      Airbag - Remastered
## 2   Radiohead           Paranoid Android - Remastered
## 3   Radiohead Subterranean Homesick Alien - Remastered
```

```
## 4   Radiohead     Exit Music (For a Film) - Remastered
## 5   Radiohead                    Let Down - Remastered
## 6   Radiohead                Karma Police - Remastered
##                       album_name
## 1 OK Computer OKNOTOK 1997 2017
## 2 OK Computer OKNOTOK 1997 2017
## 3 OK Computer OKNOTOK 1997 2017
## 4 OK Computer OKNOTOK 1997 2017
## 5 OK Computer OKNOTOK 1997 2017
## 6 OK Computer OKNOTOK 1997 2017
```

Let's just make a quick graph looking at track length over time.

```
radiohead %>%
  ggplot(aes(x = album_release_year, y = duration_ms)) +
  geom_point()
```

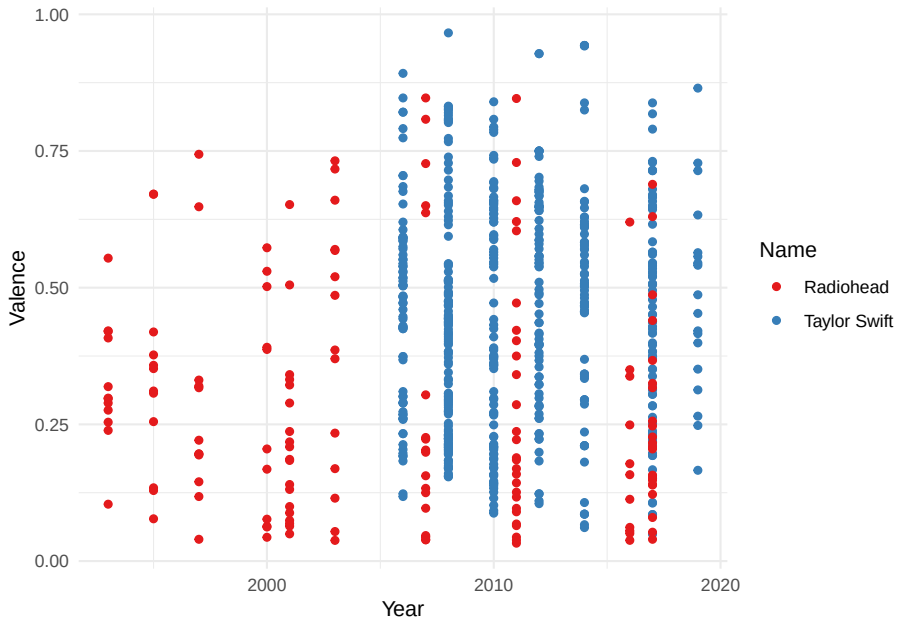


Just because we can, let's settle an argument. I've always said that Radiohead of quite depressing, but they're my wife's favourite band. So let's see how depressing they are. Spotify provides various information about each track, including 'valence', which Spotify define as '(a) measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).' So higher values are happier. Let's compare someone who we know it likely to be happy - Taylor Swift - with Radiohead.

```
swifty <- get_artist_audio_features('taylor swift')
saveRDS(swifty, "inputs/swifty.rds")
```

```
swifty <- readRDS("inputs/swifty.rds")

tibble(name = c(swifty$artist_name, radiohead$artist_name),
       year = c(swifty$album_release_year, radiohead$album_release_year),
       valence = c(swifty$valence, radiohead$valence)
                ) %>%
  ggplot(aes(x = year, y = valence, color = name)) +
  geom_point() +
  theme_minimal() +
  labs(x = "Year",
       y = "Valence",
       color = "Name") +
  scale_color_brewer(palette = "Set1")
```



Finally, for the sake of embarrassment, let's look at our most played artists.

```
top_artists <- get_my_top_artists_or_tracks(type = 'artists', time_range = 'long_term'

saveRDS(top_artists, "inputs/top_artists.rds")
```

```
top_artists <- readRDS("inputs/top_artists.rds")

top_artists %>%
```

```
  select(name, popularity)
```

```
##                   name popularity
## 1            Radiohead         81
## 2  Bombay Bicycle Club         66
## 3                Drake        100
## 4        Glass Animals         74
## 5                JAY-Z         85
## 6        Laura Marling         65
## 7       Sufjan Stevens         75
## 8      Vampire Weekend         73
## 9     Sturgill Simpson         65
## 10          Nick Drake         66
## 11        Dire Straits         78
## 12               Lorde         80
## 13         Marian Hill         65
## 14       José González         68
## 15       Stevie Wonder         79
## 16          Disclosure         82
## 17      Ben Folds Five         52
## 18       Ainslie Wills         40
## 19            Coldplay         89
## 20               alt-J         75
```

So pretty much my wife and I like what everyone else likes, with the exception of Ainslie Wills, who is an Australian and I suspect we used to listen to her when we were homesick.

How amazing that we live in a world that all that information is available with very little effort or cost.

Again, there is a lot more at the package's website: https://www.rcharlie.com/spotifyr/. A very nice little application of the Spotify API using some statistical analysis is **?**.

8.4 Scraping

8.4.1 Introduction

Web-scraping is a way to get data from websites into R. Rather than going to a website ourselves through a browser, we write code that does it for us. This opens up a lot of data to us, but on the other hand, it is not typically data that is being made available for these purposes and so it is important to be respectful of it. While generally not illegal, the specifics with regard to the legality of web-scraping depends on jurisdictions and the specifics of what you're doing, and so it is also important to be mindful of this. And finally, web-scraping imposes a

cost on the website host, and so it is important to reduce this to the extent that it's possible.

That all said, web-scraping is an invaluable source of data. But they are typically a datasets that can be created as a by-product of someone trying to achieve another aim. For instance, a retailer may have a website with their products and their prices. That has not been created deliberately as a source of data, but we can scrape it to create a dataset. As such, the following principals guide my web-scraping.

1. Avoid it. Try to use an API wherever possible.
2. Abide by their desires. Some websites have a file 'robots.txt' that contains information about what they are comfortable with scrapers doing, for instance 'https://www.google.com/robots.txt'. If they have one of these then you should read it and abide by it.
3. Reduce the impact.
    - Firstly, slow down your scraper, for instance, rather than having it visit the website every second, slow it down (using `sys.sleep()`). If you're only after a few hundred files then why not just have it visit once a minute, running in the background overnight?
    - Secondly, consider the timing of when you run the scraper. For instance, if it's a retailer then why not set your script to run from 10pm through to the morning, when fewer customers are likely to need the site? If it's a government website and they have a big monthly release then why not avoid that day?
4. Take only what you need. For instance, don't scrape the entire of Wikipedia if all you need is to know the names of the 10 largest cities in Canada. This reduces the impact on their website and allows you to more easily justify what you are doing.
5. Only scrape once. Save everything as you go so that you don't have to re-collect data. Similarly, once you have the data, you should keep that separate and not modify it. Of course, if you need data over time then you will need to go back, but this is different to needlessly re-scraping a page.
6. Don't republish the pages that you scraped. (This is in contrast to datasets that you create from it.)
7. Take ownership and ask permission if possible. At a minimum level your scripts should have your contact details in them. Depending on the circumstances, it may be worthwhile asking for permission before you scrape.

8.4.2 Getting started

Webscraping is possible by taking advantage of the underlying structure of a webpage. We use patterns in the HTML/CSS to get the data that we want. To look at the underlying HTML/CSS you can either: 1) open a browser, right-click, and choose something like 'Inspect'; or 2) save the website and then open it with a text editor rather than a browser.

HTML/CSS is a markup language comprised of matching tags. So if you want text to be bold then you would use something like:

```
<b>My bold text</b>
```

Similarly, if you want a list then you start and end the list as well as each item.

```
<ul>
  <li>Learn webscraping</li>
  <li>Do data science</li>
  <li>Proft</li>
</ul>
```

When webscraping we will search for these tags.

To get started, this is some HTML/CSS from my website. Let's say that we want to grab my name from it. We can see that the name is in bold, so we want to probably focus on that feature and extract it.

```
website_extract <- "<p>Hi, I'm <b>Rohan</b> Alexander.</p>"
```

We will use the `rvest` package **?**.

```
# install.packages("rvest")
library(rvest)

rohans_data <- read_html(website_extract)

rohans_data
```

```
## {html_document}
## <html>
## [1] <body><p>Hi, I'm <b>Rohan</b> Alexander.</p></body>
```

The language used by `rvest` to look for tags is 'node', so we will focus on bold nodes. By default `html_nodes()` returns the tags as well. So we can focus on the text that they contain, using `html_text()`.

```
rohans_data %>%
  html_nodes("b")
```

```
## {xml_nodeset (1)}
## [1] <b>Rohan</b>
```

```
first_name <-
  rohans_data %>%
  html_nodes("b") %>%
  html_text()

first_name
```

```
## [1] "Rohan"
```

The result is that we learn my first name.

8.5 Case study - Rohan's books

8.5.1 Introduction

In this case study we are going to scrape a list of books that I own, clean it, and look at the distribution of the first letters of author surnames. It is slightly more complicated than the example above, but the underlying approach is the same - download the website, look for the nodes of interest, extract the information, clean it.

8.5.2 Gather

Again, the key library that we are using is the `rvest` library. This makes it easier to download a website, and to then navigate the html to find the aspects that we are interested in. You should create a new project in a new folder (File -> New Project). Within that new folder you should make three new folders: `inputs`, `outputs`, and `scripts`.

In the scripts folder you should write and save a script along these lines. This script loads the libraries that we need, then visits my website, and saves a local copy.

```
#### Contact details ####
# Title: Get data from rohanalexander.com
# Purpose: This script gets data from Rohan's website about the books that he
# owns. It calls his website and then saves the dataset to inputs.
# Author: Rohan Alexander
# Contact: rohan.alexander@utoronto.ca
# Last updated: 20 May 2020


#### Set up workspace ####
library(rvest)
library(tidyverse)


#### Get html ####
rohans_data <- read_html("https://rohanalexander.com/bookshelf.html")
# This takes a website as an input and will read it into R, in the same way that we
# can read a, say, CSV into R.

write_html(rohans_data, "inputs/my_website/raw_data.html")
```

```
# Always save your raw dataset as soon as you get it so that you have a record
# of it. This is the equivalent of, say, write_csv() that we have used earlier.
```

8.5.3 Clean

Now we need to navigate the HTML to get the aspects that we want, and to then put them into some sensible structure. I always try to get the data into a tibble as early as possible. While it's possible to work with the nested data, I move to a tibble so that the usual verbs that I'm used to can be used.

In the scripts folder you should write and save a new R script along these lines. First, we need to add the top matter, read in the libraries and the data that we scraped.

```
#### Contact details ####
# Title: Clean data from rohanaledander.com
# Purpose: This script cleans data that was downloaded in 01-get_data.R.
# Author: Rohan Alexander
# Contact: rohan.alexander@utoronto.ca
# Pre-requisites: Need to have run 01_get_data.R and have saved the data.
# Last updated: 20 May 2020


#### Set up workspace ####
library(tidyverse)
library(rvest)

rohans_data <- read_html("inputs/my_website/raw_data.html")

rohans_data
```

```
## {html_document}
## <html xmlns="http://www.w3.org/1999/xhtml" lang="" xml:lang="">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body>\n\n<!--radix_placeholder_front_matter-->\n\n<script id="distill-fr ...
```

Now we need to identify the data that we are interested in using html tags and convert it to a tibble. If you look at the website, then you should notice that we are likely trying to focus on list items (Figure **??**).

Let's look at the source (Figure **??**).

There's a lot of debris, but scrolling down we eventually get to a list (Figure **??**).

The tag for a list item is 'li', so we modify the earlier code to focus on that and to get the text.

Bookshelf

Inspired by Patrick Collison's version, this is an incomplete (so far) and unordered list of the books that I own. I've been a bit more liberal than Patrick and included books that I've read and would like to own (designated by "-"). If you have recommendations, then please get in touch. I usually only buy books that I like, but bolded books are ones I especially like.

- -"A Little Life", Hanya Yanighara. Recommended by Lauren.
- "The Andromeda Strain", Michael Crichton.
- "Is There Life After Housework", Don Aslett.
  Got given this at the Museum of Clean in Pocatello, Idaho. The museum was surprisingly good!
- "The Chosen", Chaim Potok.
- "The Forsyth Saga", John Galsworthy.
- "Freakonomics", Steven Levitt and Stephen Dubner.
- "Tess of the D'Urbervilles" Thomas Hardy.
- "The Da Vinci Code", Dan Brown.
- "King Charles III", Anthony Holden.
- "The Road Washes Out in Spring", Baron Wormser.
  Stayed at the author's house, so bought some of his books and particularly liked this one about how his family lived off the grid in New England. Liked it even better after living in Amherst for a while.
- "The Terminal Man", Michael Crichton.
- "The Shepherd's Hut", Tim Winton.
  Present from Helen.
- **"Code: The Hidden Language of Computer Hardware and Software", Charles Petzold.**
- "Hitch-22", Christopher Hitchens.
- "Audacity of Hope", Barack Obama.
- "Kennedy", Ted Sorenson.
- "The Son Also Rises", Greg Clark.

Figure 8.3: Some of Rohan's books

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" lang="" xml:lang="">

<head>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale
  <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1"/>
  <meta name="generator" content="distill" />

  <style type="text/css">
  /* Hide doc at startup (prevent jankiness while JS renders/trans
  body {
    visibility: hidden;
  }
  </style>

 <!--radix_placeholder_import_source-->
 <!--/radix_placeholder_import_source-->

```

Figure 8.4: Source code for top of the page

```
<div class="d-article">
<ul>
<li>-“A Little Life”, Hanya Yanighara. Recommended by Lauren.</li
<li>“The Andromeda Strain”, Michael Crichton.</li>
<li>“Is There Life After Housework”, Don Aslett.<br />
Got given this at the Museum of Clean in Pocatello, Idaho. The mu
<li>“The Chosen”, Chaim Potok.</li>
<li>“The Forsyth Saga”, John Galsworthy.</li>
<li>“Freakonomics”, Steven Levitt and Stephen Dubner.</li>
<li>“Tess of the D’Urbervilles”, Thomas Hardy.</li>
<li>“The Da Vinci Code”, Dan Brown.</li>
<li>“King Charles III”, Anthony Holden.</li>
<li>“The Road Washes Out in Spring”, Baron Wormser.<br />
Stayed at the author’s house, so bought some of his books and par
<li>“The Terminal Man”, Michael Crichton.</li>
<li>“The Shepherd’s Hut”, Tim Winton.<br />
Present from Helen.</li>
<li><strong>“Code: The Hidden Language of Computer Hardware and S
<li>“Hitch-22”, Christopher Hitchens.</li>
```

Figure 8.5: Source code for list

```
#### Clean data ####
# Identify the lines that have books on them based on the list html tag
text_data <- rohans_data %>%
  html_nodes("li") %>%
  html_text()

all_books <- tibble(books = text_data)

head(all_books)
```

```
## # A tibble: 6 x 1
##   books
##   <chr>
## 1 "-"A Little Life", Hanya Yanighara. Recommended by Lauren."
## 2 ""The Andromeda Strain", Michael Crichton."
## 3 ""Is There Life After Housework", Don Aslett.\nGot given this at the Museum o~
## 4 ""The Chosen", Chaim Potok."
## 5 ""The Forsyth Saga", John Galsworthy."
## 6 ""Freakonomics", Steven Levitt and Stephen Dubner."
```

We now need to clean the data. First we want to separate the title and the author

```
# All content is just one string, so need to separate title and author
all_books <-
  all_books %>%
  separate(books, into = c("title", "author"), sep = """)

# Remove leading comma and clean up the titles a little
all_books <-
  all_books %>%
  mutate(author = str_remove_all(author, "^, "),
         author = str_squish(author),
         title = str_remove(title, """),
         title = str_remove(title, "^-")
         )

head(all_books)
```

```
## # A tibble: 6 x 2
##   title                  author
##   <chr>                  <chr>
## 1 A Little Life          Hanya Yanighara. Recommended by Lauren.
## 2 The Andromeda Strain   Michael Crichton.
## 3 Is There Life After H~ Don Aslett. Got given this at the Museum of Clean in P~
## 4 The Chosen             Chaim Potok.
## 5 The Forsyth Saga       John Galsworthy.
```

```
## 6 Freakonomics           Steven Levitt and Stephen Dubner.
```

Finally, some specific cleaning is needed.

```
# Some authors have comments after their name, so need to get rid of them, although there are som
# J. K. Rowling.
# M. Mitchell Waldrop.
# David A. Price
all_books <-
  all_books %>%
  mutate(author = str_replace_all(author,
                                  c("J. K. Rowling." = "J K Rowling.",
                                    "M. Mitchell Waldrop." = "M Mitchell Waldrop.",
                                    "David A. Price" = "David A Price")
                                  )
         ) %>%
  separate(author, into = c("author_correct", "throw_away"), sep = "\\.", extra = "drop") %>%
  select(-throw_away) %>%
  rename(author = author_correct)

# Some books have multiple authors, so need to separate them
# One has multiple authors:
# "Daniela Witten, Gareth James, Robert Tibshirani, and Trevor Hastie"
all_books <-
  all_books %>%
  mutate(author = str_replace(author,
                              "Daniela Witten, Gareth James, Robert Tibshirani, and Trevor Hastie
                              "Daniela Witten and Gareth James and Robert Tibshirani and Trevor H
  separate(author, into = c("author_first", "author_second", "author_third", "author_fourth"), se
  pivot_longer(cols = starts_with("author_"),
               names_to = "author_position",
               values_to = "author") %>%
  select(-author_position) %>%
  filter(!is.na(author))

head(all_books)
```

```
## # A tibble: 6 x 2
##   title                          author
##   <chr>                          <chr>
## 1 A Little Life                  Hanya Yanighara
## 2 The Andromeda Strain           Michael Crichton
## 3 Is There Life After Housework Don Aslett
## 4 The Chosen                     Chaim Potok
## 5 The Forsyth Saga               John Galsworthy
## 6 Freakonomics                   Steven Levitt
```

It looks there is some at the end because I have a best of. I'll just get rid of those manually because it's not the focus.

```
all_books <-
  all_books %>%
  slice(1:118)
```

8.5.4 Explore

Finally, just because we have the data now, so we may as well try to do something with it, let's look at the distribution of the first letter of the author names.

```
all_books %>%
  mutate(
    first_letter = str_sub(author, 1, 1)
    ) %>%
  group_by(first_letter) %>%
  count()
```

```
## # A tibble: 21 x 2
## # Groups:   first_letter [21]
##    first_letter     n
##    <chr>        <int>
##  1 ""               1
##  2 "A"              8
##  3 "B"              5
##  4 "C"              4
##  5 "D"             10
##  6 "E"              3
##  7 "F"              1
##  8 "G"             10
##  9 "H"              6
## 10 "I"              1
## # ... with 11 more rows
```

8.6 Case study - Canadian Prime Ministers

8.6.1 Introduction

In this case study we are interested in how long Canadian prime ministers lived, based on the year that they were born. We will scrape data from Wikipedia, clean it, and then make a graph.

The key library that we will use for scraping is `rvest`. This adds a lot of functions that will make life easier. That said, every time you scrape a website things will change. Each scrape will largely be bespoke, even if you can borrow some code from earlier projects that you have completed. It is completely normal

to feel frustrated at times. It helps to begin with an end in mind.

To that end, let's generate some simulated data. Ideally, we want a table that has a row for each prime minister, a column for their name, and a column each for the birth and death years. If they are still alive, then that death year can be empty. We know that birth and death years should be somewhere between 1700 and 1990, and that death year should be larger than birth year. Finally, we also know that the years should be integers, and the names should be characters. So, we want something that looks roughly like this:

```
library(babynames)
library(tidyverse)

simulated_dataset <-
  tibble(prime_minister = sample(x = babynames %>% filter(prop > 0.01) %>%
                                   select(name) %>% unique() %>% unlist(),
                                 size = 10, replace = FALSE),
         birth_year = sample(x = c(1700:1990), size = 10, replace = TRUE),
         years_lived = sample(x = c(50:100), size = 10, replace = TRUE),
         death_year = birth_year + years_lived) %>%
  select(prime_minister, birth_year, death_year, years_lived) %>%
  arrange(birth_year)

head(simulated_dataset)
```

```
## # A tibble: 6 x 4
##   prime_minister birth_year death_year years_lived
##   <chr>               <int>      <int>       <int>
## 1 Cynthia              1707       1777          70
## 2 Joyce                1715       1805          90
## 3 Emily                1716       1801          85
## 4 Jennifer             1731       1813          82
## 5 Frances              1743       1838          95
## 6 Cora                 1768       1823          55
```

One of the advantages of generating a simulated dataset is that if you are working in groups then one person can start making the graph, using the simulated dataset, while the other person gathers the data. In terms of a graph, we want something like Figure **??**.

8.6.2 Gather

We are starting with a question that is of interest, which how long each Canadian prime minister lived. As such, we need to identify a source of data While there are likely to be plenty of data sources that have the births and deaths of each prime minister, we want one that we can trust, and as we are going to be scraping, we want one that has some structure to it. The Wikipedia page (https:

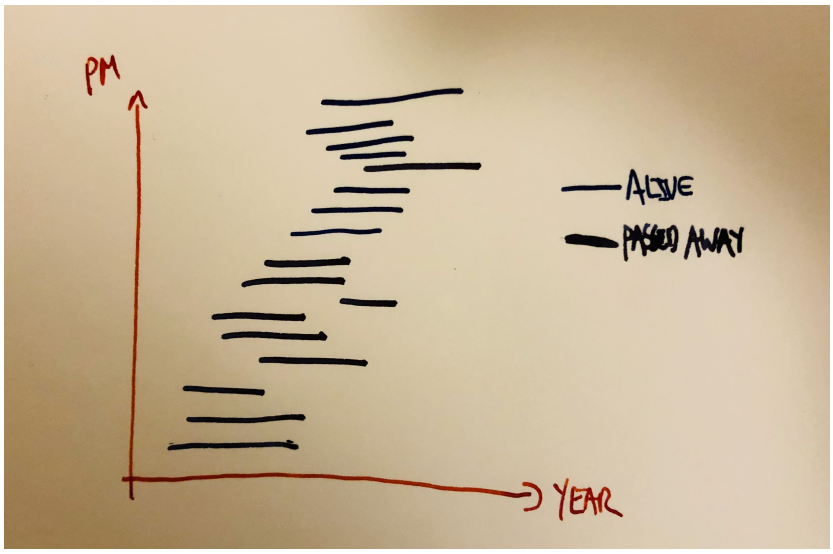


Figure 8.6: Sketch of planned graph.

//en.wikipedia.org/wiki/List_of_prime_ministers_of_Canada) fits both these criteria. As it is a popular page the information is more likely to be correct, and the data are available in a table.

We load the library and then we read in the data from the relevant page. The key function here is `read_html()`, which you can use in the same way as, say, `read_csv()`, except that it takes a html page as an input. Once you call `read_html()` then the page is downloaded to your own computer, and it is usually a good idea to save this, using `write_html()` as it is your raw data. Saving it also means that we don't have to keep visiting the website when we want to start again with our cleaning, and so it is part of being polite. However, it is likely not our property (in the case of Wikipedia, we might be okay), and so you should probably not share it.

```
library(rvest)
```

```
raw_data <- read_html("https://en.wikipedia.org/wiki/List_of_prime_ministers_of_Canada")
write_html(raw_data, "inputs/wiki/pms.html") # Note that we save the file as a html fi
```

8.6.3 Clean

Websites are made up of html, which is a markup language. We are looking for patterns in the mark-up that we can use to help us get closer to the data that we want. This is an iterative process and requires a lot of trial and error. Even simple examples will take time. You can look at the html by using a browser, right clicking, and then selecting `view page source`. Similarly, you could open

the html file using a text editor.

8.6.3.1 By inspection

We are looking for patterns that we can use to select the information that is of interest - names, birth year, and death year. When we look at the html it looks like there is something going on with `<tr>`, and then `<td>` (thanks to Thomas Rosenthal for identifying this). We select those nodes using `html_nodes()`, which takes the tags as an input. If you only want the first one then there is a singular version, `html_node()`.

```
# Read in our saved data
raw_data <- read_html("inputs/wiki/pms.html")

# We can parse tags in order
parse_data_inspection <-
  raw_data %>%
  html_nodes("tr") %>%
  html_nodes("td") %>%
  html_text() # html_text removes any remaining html tags

# But this code does exactly the same thing - the nodes are just pushed into
# the one function call
parse_data_inspection <-
  raw_data %>%
  html_nodes("tr td") %>%
  html_text()

head(parse_data_inspection)
```

```
## [1] "Abbreviation key:"
## [2] "No.: Incumbent number, Min.: Ministry, Refs: References\n"
## [3] "Colour key:"
## [4] "\n\n  Liberal Party of Canada\n \n  Historical Conservative parties (including Liberal-Co
## [5] "Provinces key:"
## [6] "AB: Alberta, BC: British Columbia, MB: Manitoba, NS: Nova Scotia,ON: Ontario, QC: Quebec,
```

At this point our data is in a character vector, we want to convert it to a table, and reduce the data down to just the information that we want. The key that is going to allow us to do this is the fact that there seems to be a blank line (which in html is denoted by `\n`) before the key information that we need. So, once we identify that line then we can filter to just the line below it!

```
parsed_data <-
  tibble(raw_text = parse_data_inspection) %>% # Convert the character vector to a table
  mutate(is_PM = if_else(raw_text == "\n\n", 1, 0), # Look for the blank line that is
         # above the row that we want
         is_PM = lag(is_PM, n = 1)) %>% # Identify the actual row that we want
```

```
  filter(is_PM == 1) # Just get the rows that we want

head(parsed_data)
```

```
## # A tibble: 6 x 2
##   raw_text                                                          is_PM
##   <chr>                                                             <dbl>
## 1 "\nSir John A. MacDonald(1815-1891)MP for Kingston, ON\n"             1
## 2 "\nAlexander Mackenzie(1822-1892)MP for Lambton, ON\n"                1
## 3 "\nSir John A. MacDonald(1815-1891)MP for Victoria, BC until 1882MP for~     1
## 4 "\nSir John Abbott(1821-1893)Senator for Quebec\n"                    1
## 5 "\nSir John Thompson(1845-1894)MP for Antigonish, NS\n"               1
## 6 "\nSir Mackenzie Bowell(1823-1917)Senator for Ontario\n"              1
```

8.6.3.2 Using the selector gadget

If you are comfortable with html then you might be able to see patterns, but one tool that may help is the SelectorGadget: https://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html. This allows you to pick and choose the elements that you want, and then gives you the input to give to `html_nodes()` (Figure ??)

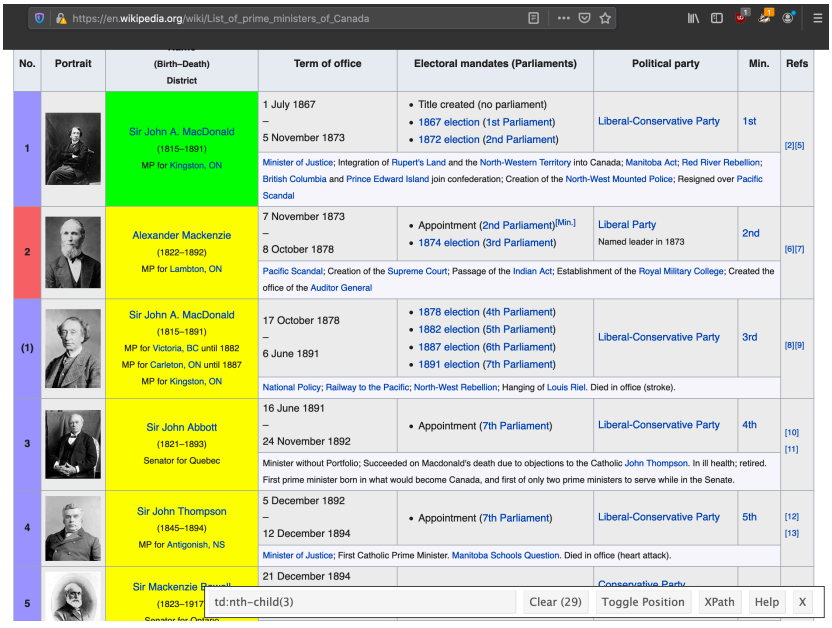


Figure 8.7: Using the Selector Gadget to identify the tag, as at 13 March 2020.

```
# Read in our saved data
raw_data <- read_html("inputs/wiki/pms.html")

# We can parse tags in order
parse_data_selector_gadget <-
  raw_data %>%
  html_nodes("td:nth-child(3)") %>%
  html_text() # html_text removes any remaining html tags

head(parse_data_selector_gadget)
```

```
## [1] "\nSir John A. MacDonald(1815-1891)MP for Kingston, ON\n"
## [2] "\nAlexander Mackenzie(1822-1892)MP for Lambton, ON\n"
## [3] "\nSir John A. MacDonald(1815-1891)MP for Victoria, BC until 1882MP for Carleton, ON until
## [4] "\nSir John Abbott(1821-1893)Senator for Quebec\n"
## [5] "\nSir John Thompson(1845-1894)MP for Antigonish, NS\n"
## [6] "\nSir Mackenzie Bowell(1823-1917)Senator for Ontario\n"
```

In this case there is one prime minister - Robert Borden - who changed party and we would need to filter away that row: `\nUnionist Party\n"`.

8.6.3.3 Clean data

Now that we have the parsed data, we need to clean it to match what we wanted. In particular we want a names column, as well as columns for birth year and death year. We will use `separate()` to take advantage of the fact that it looks like the dates are distinguished by brackets.

```
initial_clean <-
  parsed_data %>%
  separate(raw_text,
           into = c("Name", "not_name"),
           sep = "\\(",
           remove = FALSE) %>% # The remove = FALSE option here means that we
  # keep the original column that we are separating.
  separate(not_name,
           into = c("Date", "all_the_rest"),
           sep = "\\)",
           remove = FALSE)

head(initial_clean)
```

```
## # A tibble: 6 x 6
##   raw_text           Name     not_name          Date  all_the_rest         is_PM
##   <chr>              <chr>    <chr>             <chr> <chr>                <dbl>
## 1 "\nSir John A. Ma~ "\nSir ~ "1815-1891)MP fo~ 1815~ "MP for Kingston, O~     1
## 2 "\nAlexander Mack~ "\nAlex~ "1822-1892)MP fo~ 1822~ "MP for Lambton, ON~     1
```

```
## 3 "\nSir John A. Ma~ "\nSir ~ "1815-1891)MP fo~ 1815~ "MP for Victoria, B~     1
## 4 "\nSir John Abbot~ "\nSir ~ "1821-1893)Senat~ 1821~ "Senator for Quebec~     1
## 5 "\nSir John Thomp~ "\nSir ~ "1845-1894)MP fo~ 1845~ "MP for Antigonish,~     1
## 6 "\nSir Mackenzie ~ "\nSir ~ "1823-1917)Senat~ 1823~ "Senator for Ontari~     1
```

Finally, we need to clean up the columns.

```
cleaned_data <-
  initial_clean %>%
  select(Name, Date) %>%
  separate(Date, into = c("Birth", "Died"), sep = "-", remove = FALSE) %>% # The
  # PMs who have died have their birth and death years separated by a hyphen, but
  # you need to be careful with the hyphen as it seems to be a slightly odd type of
  # hyphen and you need to copy/paste it.
  mutate(Birth = str_remove(Birth, "b. ")) %>% # Alive PMs have slightly different for
  select(-Date) %>%
  mutate(Name = str_remove(Name, "\n")) %>% # Remove some html tags that remain
  mutate_at(vars(Birth, Died), ~as.integer(.)) %>% # Change birth and death to integer
  mutate(Age_at_Death = Died - Birth) %>%  # Add column of the number of years they li
  distinct() # Some of the PMs had two goes at it.

head(cleaned_data)
```

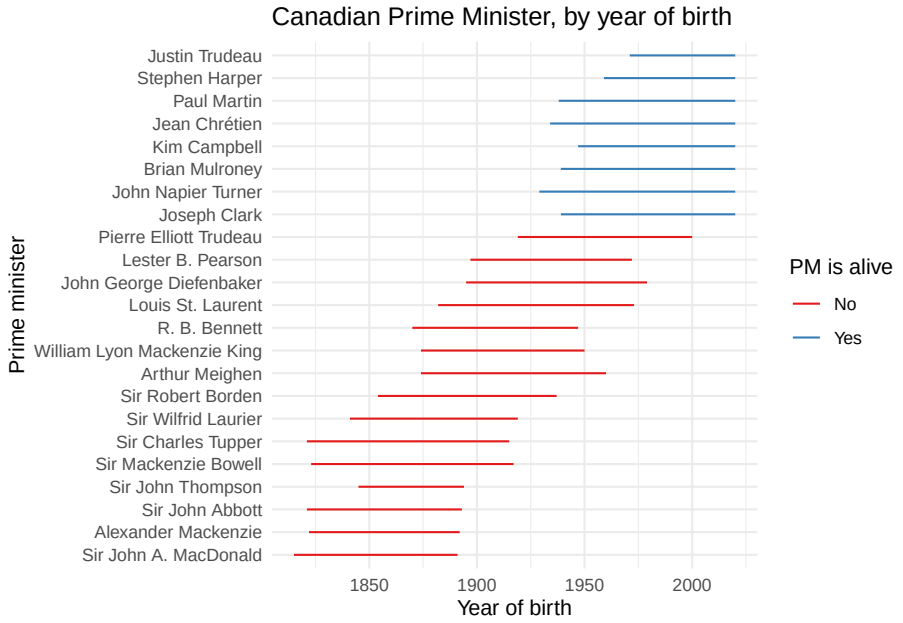
```
## # A tibble: 6 x 4
##   Name                  Birth  Died Age_at_Death
##   <chr>                 <int> <int>        <int>
## 1 Sir John A. MacDonald  1815  1891           76
## 2 Alexander Mackenzie    1822  1892           70
## 3 Sir John Abbott        1821  1893           72
## 4 Sir John Thompson      1845  1894           49
## 5 Sir Mackenzie Bowell   1823  1917           94
## 6 Sir Charles Tupper     1821  1915           94
```

8.6.4 Explore

At this point we'd like to make a graph that illustrates how long each prime minister lived. If they are still alive then we would like them to run to the end, but we would like to colour them differently.

```
cleaned_data %>%
  mutate(still_alive = if_else(is.na(Died), "Yes", "No"),
         Died = if_else(is.na(Died), as.integer(2020), Died)) %>%
  mutate(Name = as_factor(Name)) %>%
  ggplot(aes(x = Birth,
             xend = Died,
             y = Name,
             yend = Name,
```

```
            color = still_alive)) +
  geom_segment() +
  labs(x = "Year of birth",
       y = "Prime minister",
       color = "PM is alive",
       title = "Canadian Prime Minister, by year of birth") +
  theme_minimal() +
  scale_color_brewer(palette = "Set1")
```



8.7 PDFs

8.7.1 Introduction

In contrast to an API, a PDF is usually only produced for human (not computer) consumption. The nice thing about PDFs is that they are static and constant. And it is nice that they make data available at all. But the trade-off is that:

- It is not overly useful to do larger-scale statistical analysis.
- We don't know how the PDF was put together so we don't know whether we can trust it.
- We can't manipulate the data to get results that we are interested in.

Indeed, sometimes governments publish data as PDFs because they don't actually want you to be able to analyse it! Being able to get data from PDFs opens up a large number of dataset for you, some of which we'll see in this chapter.

There are two important aspects to keep in mind when approaching a PDF with a mind to extracting data from it:

1. Begin with an end in mind. Planning and then literally sketching out what you want from a final dataset/graph/paper stops you wasting time and keeps you focused.
2. Start simple, then iterate. The quickest way to make a complicated model is often to first build a simple model and then complicate it. Start with just trying to get one page of the PDF working or even just one line. Then iterate from there.

In this chapter we start by walking through several examples and then go through three case studies of varying difficulty.

8.7.2 Getting started

Figure **??** is a PDF that consists of just the first sentence from Jane Eyre taken from Project Gutenberg **?**.

We will use the package `pdftools` **?** to get the text in this one page PDF into R.

```
# install.packages("pdftools")
library(pdftools)
library(tidyverse)

first_example <- pdftools::pdf_text("inputs/pdfs/first_example.pdf")

first_example
```

```
## [1] "There was no possibility of taking a walk that day."
```

```
class(first_example)
```

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

We can see that the PDF has been correctly read in, as a character vector.

We will now try a slightly more complicated example that consists of the first few paragraphs of Jane Eyre (Figure **??**). Also notice that now we have the chapter heading as well.

We use the same function as before.

```
second_example <- pdftools::pdf_text("inputs/pdfs/second_example.pdf")

second_example
```

```
## [1] "CHAPTER I\nThere was no possibility of taking a walk that day. We had been wanc
something lighter, franker, more natural, as it were-she really must exclude me from\nr
```

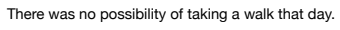


Figure 8.8: First sentence of Jane Eyre

CHAPTER I

There was no possibility of taking a walk that day. We had been wandering, indeed, in the leafless shrubbery an hour in the morning; but since dinner (Mrs. Reed, when there was no company, dined early) the cold winter wind had brought with it clouds so sombre, and a rain so penetrating, that further out-door exercise was now out of the question.

I was glad of it: I never liked long walks, especially on chilly afternoons: dreadful to me was the coming home in the raw twilight, with nipped fingers and toes, and a heart saddened by the chidings of Bessie, the nurse, and humbled by the consciousness of my physical inferiority to Eliza, John, and Georgiana Reed.

The said Eliza, John, and Georgiana were now clustered round their mama in the drawing-room: she lay reclined on a sofa by the fireside, and with her darlings about her (for the time neither quarrelling nor crying) looked perfectly happy. Me, she had dispensed from joining the group; saying, "She regretted to be under the necessity of keeping me at a distance; but that until she heard from Bessie, and could discover by her own observation, that I was endeavouring in good earnest to acquire a more sociable and childlike disposition, a more attractive and sprightly manner—something lighter, franker, more natural, as it were—she really must exclude me from privileges intended only for contented, happy, little children."

"What does Bessie say I have done?" I asked.

"Jane, I don't like cavillers or questioners; besides, there is something truly forbidding in a child taking up her elders in that manner. Be seated somewhere; and until you can speak pleasantly, remain silent."

A breakfast-room adjoined the drawing-room, I slipped in there. It contained a bookcase: I soon possessed myself of a volume, taking care that it should be one stored with pictures. I mounted into the window-seat: gathering up my feet, I sat cross-legged, like a Turk; and, having drawn the red moreen curtain nearly close, I was shrined in double retirement.

Folds of scarlet drapery shut in my view to the right hand; to the left were the clear panes of glass, protecting, but not separating me from the drear November day. At intervals, while turning over the leaves of my book, I studied the aspect of that winter afternoon. Afar, it offered a pale blank of mist and cloud; near a scene of wet lawn and storm-beat shrub, with ceaseless rain sweeping away wildly before a long and lamentable blast.

Figure 8.9: First few paragraphs of Jane Eyre

```
class(second_example)
```

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

Again, we have a character vector. The end of each line is signalled by '\n', but other than that it looks pretty good.

Finally, we consider the first two pages.

We use the same function as before.

```
third_example <- pdftools::pdf_text("inputs/pdfs/third_example.pdf")

third_example
```

```
## [1] "CHAPTER I\nThere was no possibility of taking a walk that day. We had been wandering, ind
something lighter, franker, more natural, as it were-she really must exclude me from\nprivileges
Bewick's History of British Birds: the letterpress thereof I cared little\nfor, generally speakin
\n"Where the Northern Ocean, in vast whirls,\nBoils round the naked, melancholy isles\n"
## [2] "Of farthest Thule; and the Atlantic surge\nPours in among the stormy Hebrides."\nNor coul
that reservoir of frost and snow, where firm fields of ice, the\naccumulation of centuries of win
bad animal!"\n"It is well I drew the curtain," thought I; and I wished fervently he might not dis
\n"
```

```
class(third_example)
```

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

Now, notice that the first page is the first element of the character vector and the second page is the second element.

As we're most familiar with rectangular data we'll try to get it into that format as quickly as possible. And then we can use our regular tools to deal with it.

First we want to convert the character vector into a tibble. At this point we may like to add page numbers as well.

```
jane_eyre <- tibble(raw_text = third_example,
                    page_number = c(1:2))
```

We probably now want to separate the lines so that each line is an observation. We can do that by looking for the '\n' remembering that we need to escape the backslash as it's a special character.

```
jane_eyre <- separate_rows(jane_eyre, raw_text, sep = "\\n", convert = FALSE)
head(jane_eyre)
```

```
## # A tibble: 6 x 2
##   raw_text                                                        page_number
##   <chr>                                                                 <int>
## 1 CHAPTER I                                                                 1
```

```
## 2 There was no possibility of taking a walk that day. We had been w~     1
## 3 leafless shrubbery an hour in the morning; but since dinner (Mrs.~     1
## 4 company, dined early) the cold winter wind had brought with it cl~     1
## 5 penetrating, that further out-door exercise was now out of the qu~     1
## 6 I was glad of it: I never liked long walks, especially on chilly ~     1
```

8.8 Case-study: US Total Fertility Rate, by state and year (2000-2018)

8.8.1 Introduction

If you're married to a demographer it is not too long until you are asked to look at a US Department of Health and Human Services Vital Statistics Report. In this case we are interested in trying to get the total fertility rate (the average number of births per woman assuming that woman experience the current age-specific fertility rates throughout their reproductive years)[2] for each state for nineteen years. Annoyingly, the US persists in only making this data available in PDFs, but it makes a nice case study.

In the case of the year 2000 the table that we are interested in is on page 40 of a PDF that is available https://www.cdc.gov/nchs/data/nvsr/nvsr50/nvsr50_05.pdf and it is the column labelled: "Total fertility rate" (Figure **??**).

8.8.2 Begin with an end in mind

The first step when getting data out of a PDF is to sketch out what you eventually want. A PDF typically contains a lot of information, and so it is handy to be very clear about what you need. This helps keep you focused, and prevents scope creep, but it is also helpful when thinking about data checks. Literally write down on paper what you have in mind.

In this case, what is needed is a table with a column for state, year and TFR (Figure **??**).

8.8.3 Start simple, then iterate.

There are 19 different PDFs and we are interested in a particular column in a particular table in each of them. Unfortunately there is nothing magical about what is coming. This first step requires working out the link for each, and the page and column name that is of interest. In the end, this looks like this.

```
monicas_data <- read_csv("inputs/tfr_tables_info.csv")

monicas_data %>%
```

[2] And if you'd like to know more about this then I'd recommend starting a PhD with Monica Alexander.

40 National Vital Statistics Report, Vol. 50, No. 5, Revised May 15, 2002

Table 10. Number of births, birth rates, fertility rates, total fertility rates, and birth rates for teenagers 15-19 years by age of mother: United States, each State and territory, 2000

[By place of residence. Birth rates are live births per 1,000 estimated population in each area; fertility rates are live births per 1,000 women aged 15-44 years estimated in each area; total fertility rates are sums of birth rates for 5-year age groups multiplied by 5; birth rates by age are live births per 1,000 women in specified age group estimated in each area]

State	Number of births	Birth rate	Fertility rate	Total fertility rate	Teenage birth rate		
					15-19 years		
					Total	15-17 years	18-19 years
United States [1]	4,058,814	14.7	67.5	2,130.0	48.5	27.4	79.2
Alabama	63,299	14.4	65.0	2,021.0	62.9	37.9	97.3
Alaska	9,974	16.0	74.6	2,437.0	42.4	23.6	69.4
Arizona	85,273	17.5	84.4	2,652.5	69.1	41.1	111.3
Arkansas	37,783	14.7	69.1	2,140.0	68.5	36.7	114.1
California	531,959	15.8	70.7	2,186.0	48.5	28.6	75.6
Colorado	65,438	15.8	73.1	2,356.5	49.2	28.6	79.8
Connecticut	43,026	13.0	61.2	1,931.5	31.9	16.9	56.3
Delaware	11,051	14.5	63.5	2,014.0	51.6	30.5	80.2
District of Columbia	7,666	14.8	63.0	1,975.5	80.7	60.7	101.8
Florida	204,125	13.3	66.9	2,157.5	52.6	29.7	88.0
Georgia	132,644	16.7	71.4	2,239.5	64.2	36.8	104.3
Hawaii	17,551	14.9	72.3	2,337.0	45.1	24.7	70.5
Idaho	20,366	16.0	74.8	2,314.0	43.1	21.3	72.8
Illinois	185,036	15.2	69.5	2,190.5	49.5	28.5	81.1
Indiana	87,699	14.7	66.8	2,109.0	50.3	26.2	85.9
Iowa	38,266	13.3	64.0	2,052.5	34.7	17.4	60.3
Kansas	39,666	14.9	69.2	2,205.0	45.3	22.4	78.5
Kentucky	56,029	14.1	63.6	1,992.5	55.3	29.2	92.2
Louisiana	67,898	15.5	69.1	2,128.5	62.1	36.3	97.1
Maine	13,603	10.8	49.5	1,611.5	28.7	13.4	52.8
Maryland	74,316	14.2	61.9	1,974.5	41.6	23.8	68.8
Massachusetts	81,614	13.2	59.2	1,799.0	27.1	15.0	44.9
Michigan	136,171	13.7	62.0	1,969.5	39.2	21.3	66.3
Minnesota	67,604	14.0	63.8	2,062.0	29.6	15.6	51.0
Mississippi	44,075	15.8	70.3	2,124.0	72.0	45.0	109.9
Missouri	76,463	13.9	64.0	2,047.5	48.8	26.5	82.2
Montana	10,957	12.3	61.3	2,003.0	35.8	19.1	60.8
Nebraska	24,646	14.8	68.9	2,209.0	37.2	19.3	62.7
Nevada	30,829	16.4	79.8	2,560.0	62.2	34.2	106.7
New Hampshire	14,609	12.0	52.2	1,664.0	23.4	9.8	45.4
New Jersey	115,632	14.1	65.8	2,086.0	31.7	17.0	54.9
New Mexico	27,223	15.6	72.7	2,313.0	66.2	40.2	105.1
New York	258,737	14.2	65.0	2,022.0	35.6	20.1	58.1
North Carolina	120,311	15.5	71.6	2,269.5	59.9	32.8	101.4
North Dakota	7,676	12.2	58.7	1,875.5	28.2	12.5	51.4
Ohio	155,472	13.8	63.0	1,995.5	45.6	24.1	77.2
Oklahoma	49,782	14.7	69.9	2,184.0	60.1	32.9	99.8
Oregon	45,804	13.7	65.8	2,086.0	43.2	23.5	72.8
Pennsylvania	146,281	12.2	58.2	1,868.0	35.2	19.6	58.8
Rhode Island	12,505	12.6	58.1	1,822.0	38.4	21.3	64.0
South Carolina	56,114	14.3	63.3	1,971.5	60.6	36.7	92.9
South Dakota	10,345	14.0	66.7	2,148.0	37.2	19.4	62.2
Tennessee	79,611	14.4	65.2	2,063.5	61.5	34.2	101.6
Texas	363,414	17.8	80.0	2,500.5	69.2	42.7	107.1
Utah	47,353	21.9	94.5	2,761.5	40.0	22.0	62.7
Vermont	6,500	10.9	48.8	1,565.5	24.1	10.6	44.5
Virginia	98,938	14.2	61.2	1,904.0	40.8	21.7	66.9
Washington	81,036	13.9	63.2	2,011.5	38.2	20.3	64.5
West Virginia	20,865	11.6	55.9	1,723.5	46.4	22.8	79.8
Wisconsin	69,326	13.1	60.4	1,940.0	34.5	18.3	58.8
Wyoming	6,253	13.0	62.7	1,976.5	40.8	19.0	73.4

Figure 8.10: Example Vital Statistics Report, from 2000

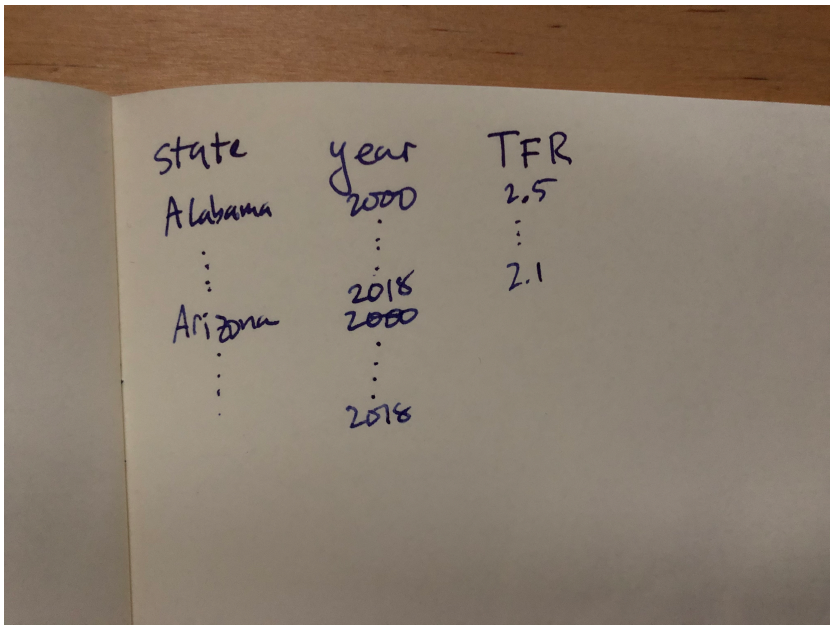


Figure 8.11: Desired output from the PDF

```
select(year, page, table, column_name, url) %>%
gt()
```

year	page	table	column_name	url
2000	40	10	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr50/nvsr50_05.p
2001	41	10	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr51/nvsr51_02.p
2002	46	10	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr52/nvsr52_10.p
2003	45	10	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr54/nvsr54_02.p
2004	52	11	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr55/nvsr55_01.p
2005	52	11	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr56/nvsr56_06.p
2006	49	11	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr57/nvsr57_07.p
2007	41	11	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr58/nvsr58_24.p
2008	43	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr59/nvsr59_01.p
2009	43	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr60/nvsr60_01.p
2010	42	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr61/nvsr61_01.p
2011	40	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr62/nvsr62_01.p
2012	38	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr62/nvsr62_09.p
2013	37	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr64/nvsr64_01.p
2014	38	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr64/nvsr64_12.p
2015	42	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr66/nvsr66_01.p
2016	29	8	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr67/nvsr67_01.p
2016	30	8	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr67/nvsr67_01.p

2017	23	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr67/nvsr67_08-508.pdf
2017	24	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr67/nvsr67_08-508.pdf
2018	23	12	Total fertility rate	https://www.cdc.gov/nchs/data/nvsr/nvsr68/nvsr68_13-508.pdf

The first step is to get some code that works for one of them. I'll step through the code in a lot more detail than normal because we're going to use these pieces a lot.

We will choose the year 2000. We first download the data and save it.

```
download.file(url = monicas_data$url[1],
              destfile = "inputs/pdfs/dhs/year_2000.pdf")
```

We now want to read the PDF in as a character vector.

```
dhs_2000 <- pdftools::pdf_text("inputs/pdfs/dhs/year_2000.pdf")
```

Convert it to a tibble, so that we can use familiar verbs on it.

```
dhs_2000 <- tibble(raw_data = dhs_2000)

head(dhs_2000)
```

```
## # A tibble: 6 x 1
##   raw_data
##   <chr>
## 1 "Volume 50, Number 5                                                          ~
## 2 "2     National Vital Statistics Report, Vol. 50, No. 5, February 12, 2002\nH~
## 3 "                                                                             ~
## 4 "4     National Vital Statistics Report, Vol. 50, No. 5, February 12, 2002\nD~
## 5 "                                                                             ~
## 6 "6     National Vital Statistics Report, Vol. 50, No. 5, February 12, 2002\n ~
```

Grab the page that is of interest (remembering that each page is a element of the character vector, hence a row in the tibble).

```
dhs_2000 <-
  dhs_2000 %>%
  slice(monicas_data$page[1])

head(dhs_2000)
```

```
## # A tibble: 1 x 1
##   raw_data
##   <chr>
## 1 "40 National Vital Statistics Report, Vol. 50, No. 5, Revised May 15, 20022\n~
```

Now we want to separate the rows.

```
dhs_2000 <-
  dhs_2000 %>%
  separate_rows(raw_data, sep = "\\n", convert = FALSE)

head(dhs_2000)
```

```
## # A tibble: 6 x 1
##   raw_data
##   <chr>
## 1 40 National Vital Statistics Report, Vol. 50, No. 5, Revised May 15, 20022
## 2 Table 10. Number of births, birth rates, fertility rates, total fertility rat~
## 3 United States, each State and territory, 2000
## 4 [By place of residence. Birth rates are live births per 1,000 estimated popul~
## 5 estimated in each area; total fertility rates are sums of birth rates for 5-y~
## 6 age group estimated in each area]
```

Now we are searching for patterns that we can use. (If you have a lot of tables that you are interested in grabbing from PDFs then it may also be worthwhile considering the tabulizer package which is specifically designed for that. The issue is that it depends on Java and I always seem to run into trouble when I need to use Java so I avoid it when I can.)

Let's look at the first ten lines of content.

```
dhs_2000[13:22,]
```

```
## # A tibble: 10 x 1
##    raw_data
##    <chr>
##  1 United States 1 ..................................................      ~
##  2 Alabama ..........................................................      ~
##  3 Alaska ...........................................................      ~
##  4 Arizona ..........................................................      ~
##  5 Arkansas .........................................................      ~
##  6 California .......................................................      ~
##  7 Colorado .........................................................      ~
##  8 Connecticut ......................................................      ~
##  9 Delaware .........................................................      ~
## 10 District of Columbia .............................................      ~
```

It doesn't get much better than this:

1. We have dots separating the states from the data.
2. We have a space between each of the columns.

So we can now separate this in to separate columns. First we want to match on when there is at least two dots (remembering that the dot is a special character and so needs to be escaped).

```
dhs_2000 <-
  dhs_2000 %>%
  separate(col = raw_data,
           into = c("state", "data"),
           sep = "\\.{2,}",
           remove = FALSE,
           fill = "right"
           )

head(dhs_2000)
```

```
## # A tibble: 6 x 3
##   raw_data                                state                                data
##   <chr>                                   <chr>                                <chr>
## 1 40 National Vital Statistics Report,~ 40 National Vital Statistics Repo~ <NA>
## 2 Table 10. Number of births, birth ra~ Table 10. Number of births, birth~ <NA>
## 3 United States, each State and territ~ United States, each State and ter~ <NA>
## 4 [By place of residence. Birth rates ~ [By place of residence. Birth rat~ <NA>
## 5 estimated in each area; total fertil~ estimated in each area; total fer~ <NA>
## 6 age group estimated in each area]     age group estimated in each area]  <NA>
```

We get the expected warnings about the top and the bottom as they don't have multiple dots.

(Another option here is to use the `pdf_data()` function which would allow us to use location rather than delimiters.)

We can now separate the data based on spaces. There is an inconsistent number of spaces, so we first squish any example of more than one space into just one.

```
dhs_2000 <-
  dhs_2000 %>%
  mutate(data = str_squish(data)) %>%
  tidyr::separate(col = data,
           into = c("number_of_births",
                    "birth_rate",
                    "fertility_rate",
                    "TFR",
                    "teen_births_all",
                    "teen_births_15_17",
                    "teen_births_18_19"),
           sep = "\\s",
           remove = FALSE
           )

head(dhs_2000)
```

```
## # A tibble: 6 x 10
```

```
##   raw_data state data  number_of_births birth_rate fertility_rate TFR
##   <chr>    <chr> <chr> <chr>            <chr>      <chr>          <chr>
## 1 40 Nati~ 40 N~ <NA>  <NA>             <NA>       <NA>           <NA>
## 2 Table 1~ Tabl~ <NA>  <NA>             <NA>       <NA>           <NA>
## 3 United ~ Unit~ <NA>  <NA>             <NA>       <NA>           <NA>
## 4 [By pla~ [By ~ <NA>  <NA>             <NA>       <NA>           <NA>
## 5 estimat~ esti~ <NA>  <NA>             <NA>       <NA>           <NA>
## 6 age gro~ age ~ <NA>  <NA>             <NA>       <NA>           <NA>
## # ... with 3 more variables: teen_births_all <chr>, teen_births_15_17 <chr>,
## #   teen_births_18_19 <chr>
```

This is all looking fairly great. The only thing left is to clean up.

```
dhs_2000 <-
  dhs_2000 %>%
  select(state, TFR) %>%
  slice(13:69) %>%
  mutate(year = 2000)

dhs_2000
```

```
## # A tibble: 57 x 3
##    state                  TFR      year
##    <chr>                  <chr>   <dbl>
##  1 "United States 1 "     2,130.0  2000
##  2 "Alabama "             2,021.0  2000
##  3 "Alaska "              2,437.0  2000
##  4 "Arizona "             2,652.5  2000
##  5 "Arkansas "            2,140.0  2000
##  6 "California "          2,186.0  2000
##  7 "Colorado "            2,356.5  2000
##  8 "Connecticut "         1,931.5  2000
##  9 "Delaware "            2,014.0  2000
## 10 "District of Columbia " 1,975.5  2000
## # ... with 47 more rows
```

And we're done for that year. Now we want to take these pieces, put them into a function and then run that function over all 19 years.

8.8.4 Iterating

8.8.4.1 Get the PDFs

The first part is downloading each of the 19 PDFs that we need. We're going to build on the code that we used before. That code was:

```
download.file(url = monicas_data$url[1], destfile = "inputs/pdfs/dhs/year_2000.pdf")
```

To modify this we need:

1. To have it iterate through each of the lines in the dataset that contains our CSVs (i.e. where it says 1, we want 1, then 2, then 3, etc.).
2. Where it has a filename, we need it to iterate through our desired filenames (i.e. year_2000, then year_2001, then year_2002, etc).
3. We'd like for it to do all of this in a way that is a little robust to errors. For instance, if one of the URLs is wrong or the internet drops out then we'd like it to just move onto the next PDF, and then warn us at the end that it missed one, not to stop. (This doesn't really matter because it's only 19 files, but it's pretty easy to find yourself doing this for thousands of files).

We will draw on the `purrr` package for this **?**.

```
library(purrr)
monicas_data <-
  monicas_data %>%
  mutate(pdf_name = paste0("inputs/pdfs/dhs/year_", year, ".pdf"))
```

```
purrr::walk2(monicas_data$url, monicas_data$pdf_name, purrr::safely(~download.file(.x , .y)))
```

What this code does it take the function `download.file()` and give it two arguments: `.x` and `.y`. The function `walk2()` then applies that function to the inputs that we give it, in this case the URLs columns is the `.x` and the pdf_names column is the `.y`. Finally, the `safely()` function means that if there are any failures then it just moves onto the next file instead of throwing an error.

We now have each of the PDFs saved and we can move onto getting the data from them.

8.8.4.2 Get data from the PDFs

Now we need to get the data from the PDFs. As before, we're going to build on the code that we used before. That code (overly condensed) was:

```
dhs_2000 <- pdftools::pdf_text("inputs/pdfs/dhs/year_2000.pdf")

dhs_2000 <-
  tibble(raw_data = dhs_2000) %>%
  slice(monicas_data$page[1]) %>%
  separate_rows(raw_data, sep = "\\n", convert = FALSE) %>%
  separate(col = raw_data, into = c("state", "data"), sep = "\\.{2,}", remove = FALSE) %>%
  mutate(data = str_squish(data)) %>%
  separate(col = data,
           into = c("number_of_births", "birth_rate", "fertility_rate", "TFR", "teen_births_all",
           sep = "\\s",
           remove = FALSE) %>%
  select(state, TFR) %>%
```

```
  slice(13:69) %>%
  mutate(year = 2000)

dhs_2000
```

There are a bunch of aspects here that have been hardcoded, but the first thing that we want to iterate is the argument to `pdf_text()`, then the number in in `slice()` will also need to change (that is doing the work to get only the page that we are interested in).

Two aspects are hardcoded and these may need to be updated. In particular: 1) The separate only works if each of the tables has the same columns in the same order; and 2) the slice (which restricts the data to just the states) only works. Finally, we add the year only at the end, whereas we'd need to bring that up earlier in the process.

We'll start by writing a function that will go through all the files, grab the data, get the page of interest, and then expand the rows. We'll then use a function from `purrr` to apply that function to all of the PDFs and to output a tibble.

```
get_pdf_convert_to_tibble <- function(pdf_name, page, year){

  dhs_table_of_interest <-
    tibble(raw_data = pdftools::pdf_text(pdf_name)) %>%
    slice(page) %>%
    separate_rows(raw_data, sep = "\\n", convert = FALSE) %>%
    separate(col = raw_data,
             into = c("state", "data"),
             sep = "[ |\\.]\\s+(?=[[:digit:]])",
             remove = FALSE) %>%
    mutate(
      data = str_squish(data),
      year_of_data = year)

  print(paste("Done with", year))

  return(dhs_table_of_interest)
}

raw_dhs_data <- purrr::pmap_dfr(monicas_data %>% select(pdf_name, page, year),
                                get_pdf_convert_to_tibble)
```

```
## [1] "Done with 2000"
## [1] "Done with 2001"
## [1] "Done with 2002"
## [1] "Done with 2003"
## [1] "Done with 2004"
```