

# Instructor Resources for Cloud-Based Data Science

*Cloud-Based Data Science Team*

*February 2019*



# Contents

<b>What is this book</b>	<b>5</b>
<b>Course 1: Introduction to Chromebook Data Science</b>	<b>7</b>
<b>Optional Course: How to Use a Chromebook</b>	<b>9</b>
<b>Course 2: Google and the Cloud</b>	<b>13</b>
Exercise 1: Sharing Google Docs with Others . . . . .	13
Exercise 2: Working on Google Docs offline . . . . .	13
Exercise 3: Creating a Google calendar event and inviting others . . . . .	13
Exercise 4: Subscribing to a Google Calendar . . . . .	13
Exercise 5: Deleting and recovering a file on Google Drive . . . . .	15
Exercise 6: Using a resume template on Google Doc . . . . .	19
Exercise 7: Creating a Google Sheets document . . . . .	19
Exercise 8: Creating a Google Slides document . . . . .	20
<b>Course 3: Organizing Data Science Projects</b>	<b>21</b>
Exercise 1: Replicating a Google doc as Rmd . . . . .	21
Exercise 2: Creating a New Project and the Workflow . . . . .	21
<b>Course 4: Version Control</b>	<b>23</b>
Exercise 1: Creating a new repository . . . . .	23
Exercise 2: Adding an issue to a classmate's repository . . . . .	23
Exercise 3: Deleting a repository . . . . .	23
Exercise 4: Creating an RStudio project from a Github repository . . . . .	23
<b>Course 5: Introduction to R</b>	<b>31</b>
Exercise 1: Installing and Uninstalling Packages . . . . .	31
Exercise 2: Using the script editor . . . . .	32
Exercise 3: Basic Commands in R . . . . .	33
Exercise 4: Creating a Data Frame . . . . .	33
Exercise 5: Creating Lists in R . . . . .	35
Exercise 6: Missing Values in R . . . . .	35
Exercise 7: Reverse a String Function . . . . .	36
Exercise 8: A simple function . . . . .	37
Exercise 9: The scan function . . . . .	38
Exercise 10: Function for the class of a data frame . . . . .	38
Exercise 11: Function for the # of times a value is repeated . . . . .	39
Exercise 12: Cumulative sum function . . . . .	39
Exercise 13: Cumulative sum function . . . . .	39
<b>Course 6: Data Tidying</b>	<b>41</b>
Exercise 1: Tidying NBA Finals Data . . . . .	41

Exercise 2: Calculating Age Based on Birthdate . . . . .	43
<b>Course 7: Data Visualization</b>	<b>47</b>
Exercise 1: Visualizing NBA Finals Data . . . . .	47
<b>Course 8: Getting Data</b>	<b>55</b>
Exercise 1: Saving a data frame as an rds file . . . . .	55
Exercise 2: Scraping IMDB data . . . . .	55
Exercise 3: Scraping NBA data from ESPN.com . . . . .	59
<b>Course 9: Data Analysis</b>	<b>63</b>
<b>Course 10: Written and Oral Communication in Data Science</b>	<b>65</b>
<b>Course 11: Getting a Job in Data Science</b>	<b>67</b>
<b>Course 11: Random Stuff</b>	<b>69</b>
Classroom activity 1 . . . . .	69

# What is this book

**Note:** If you would like to contribute to the instructor resources, please contribute here. If you have any questions or comments about this guide, you can contact us here.

The following book provides additional resources to instructors who use Cloud-Based Data Science MOOCs as part of their curriculum for teaching data science. Cloud-Based Data Science is a result of a team of data scientists and biostatisticians at Johns Hopkins University School of Public Health.

Cloud-Based Data Science (CBDS) is a free, massive open online educational offered through Leanpub to help anyone who can read, write, and use a computer to move into data science. CBDS lets students do data science using only a web browser and cheap computers like Chromebooks.

CBDS program is entirely online and students can take it for free. Students get a certificate for each CBDS online class from Leanpub. There are currently 12 courses that are offered in the Cloud-Based Data Science Curriculum. Courses can be assigned as homeworks and therefore the teacher can flip the classroom and use the class time to work on projects and answering questions. However, the instructor can also use the class time to have students take the courses depending on tastes and school resources.

Course	Course Description	Leanpub Link
<b>Introduction to Cloud-Based Data Science</b>	This is the first class in the Cloud-Based Data Science series. Data science is one of the most exciting and fastest growing careers in the world. The goal of this series is to help people with no background and limited resources transition into data science. The only pre-requisites are a computer with a web browser and the ability to type and follow instructions. We guide you through the rest.	Course 0
<b>How to Use A Chromebook</b>	This course will introduce you to using a Chromebook. The Introduction and Setup course might sound simple, but it will set up the infrastructure for success with the later, more challenging courses.	Course 1
<b>Google and the Cloud</b>	The Google and the Cloud course introduces using Google's in-built apps, which form the fundamental backbone of a Chromebook. We'll go step by step through the process to integrating these apps together to form your productivity workflow.	Course 2
<b>Organizing Data Science Projects</b>	Projects are central to the role of any data scientist. These lessons will discuss how to organize projects and the files that are part of each project and will introduce you to Markdown, a simple way to compile text documents to a standard format.	Course 3

Course	Course Description	Leanpub Link
<b>Version Control</b>	Github is the world's most popular version control website. With GitHub and Markdown, they provide a powerful way for you to get your code out to the world. In this course, we will tour GitHub, discussing the basic features of the website, what a repository is, and how to work with repositories on GitHub.	Course 4
<b>Introduction to R</b>	R is a simple to learn programming language that is powerful for data analysis. The R Basics course will teach you how to get started from ground zero. We will discuss what objects and packages are, introduce some basic R commands, and discuss RMarkdown, which you will use to write all your reports and to develop a personal website.	Course 5
<b>Data Tidying</b>	This course will focus on how to organize and tidy data sets in R, this is the first step most data scientist's do before analyzing data!	Course 6
<b>Data Visualization</b>	This course will cover the different types of visualization most commonly used by data scientists as well as how to make these different plots in R. We will cover how to make basic tables and figures as well as how to make interactive graphics.	Course 7
<b>Getting Data</b>	Data is often misunderstood in both subject and application. The Data course will focus on understanding what data is, what the data you'll encounter will look like, and how to analyze and use data. Additionally, we'll start to discuss important ethical and legal considerations when working in data science, where to find data, and how to work with these data in RStudio.	Course 8
<b>Data Analysis</b>	This course will discuss the various types of data analysis, what to consider when carrying out an analysis, and how to approach a data analysis project.	Course 9
<b>Written and Oral Communication in Data Science</b>	This course will discuss better practices for oral and written communication in data science.	Course 10
<b>Getting a Job in Data Science</b>	After you learn all of these skills, it is still crucial that you learn the best ways to network and get a job in data science. This course will focus on so-called soft skills on how to give presentations, how to present yourself in the online community, how to network, and how to do data science interviews.	Course 11

**Note:** If the courses are used as part of a K-12 curriculum, the instructors can skip the last two courses that discuss getting jobs in data science and soft skills needed for jobs in data science.

# Course 1: Introduction to Chromebook Data Science

The instructor can use the following activities for this course:

- First and foremost, this session is a great time to talk about data science and what data scientists do. You can assign articles or Youtube videos to help students have some understanding of data science. To start this video does a good job explaining some of areas of data science. Note that in explaining data science, you can have a career-oriented approach or have a data-literacy approach. In the career-oriented approach you can focus on data science as an occupation and in the data-literacy approach you can focus on how data science can be used to help us understand the world. Tailor this part depending on the educational level of the students.
- Talk about the program and what the expectations are. You can discuss the length of the program, the requirements, the assignments, etc. We run the program in 4 months including the time spent on courses (3 months) and the career mentoring phase. If you are a K-12 teacher, you can skip the career mentoring phase.
- Talk about how students can customized their Chromebooks. Different programs may use different computers so this section can vary depending on the
- Explain what the videos at the end of lessons on Leanpub are. These videos use automated voice using Amazon technology and we use these automated voices to make editing the videos faster and more efficient. Explain to your students that the videos are not complimentary or necessary to completing the lessons and are designed for students with disabilities or students who are more comfortable with voice lectures rather than texts. If your students are comfortable with reading the text, they can skip the videos.
- If your students are using Chromebook throughout the program, here's a good time to tell them little fun things about their Chromebooks and why they are chosen for the program (cost and speed since they work on the cloud). These article explains some of the Chromebook tricks: Top 10 Chromebook tips and tricks and 8 useful Chromebook tricks you aren't using but should.
- We use appear.in to communicate with our students outside of classroom. appear.in is a user-friendly paltform for chatting with your students and is free for up to 4 participants. If you need to communicate with more than 3 students at the same time, you can pay for a premium service. This article explains how appear.in can be used in classroom.





# Optional Course: How to Use a Chromebook

Instructors can skip this course if students are not using Google Chromebooks. If your students use Chromebooks, use the following activities for making your student more comfortable with their computers.

- A lot of times, we notice that students do not know about keyboard shortcuts that can be extremely helpful. This session is a good opportunity to teach them how to use. A lot of these keyboard shortcuts are platform independent, i.e., one can use them across Chromebooks, PCs, or Macs. Follow this link for a list of useful keyboard shortcuts. It's a long list so if you just want to tell them about the most important ones, focus on `Ctrl + a`, `Ctrl + c`, `Ctrl + p`, `Ctrl + z`, and `Ctrl + f`.
- There are also some tricks that students can learn about their Chromebooks. This is our favorite from EdTechTeam.
- Depending on how you would like to communicate with your students outside of the classroom, here is a good place to introduce them the tool and make them comfortable with it. We use Slack for creating a community outside of classroom so students can ask their questions and get answers from you or their classmates. There are two main advantages for using Slack: 1) all conversations are kept in one place rather than all over your email so it's easy to review conversation history. 2) Slack provides *public* and *private* channels so you can post things that everyone can see or post things that only specific students can see. It's also easy to share links, code, and use emojis on Slack so it's our absolute favorite. It's mainly free (unless you need to use specific features) and there is also discounts for education purposes. Follow this link to learn about Slack for Education. If you want to use Slack, create an account and a channel for your classroom/school and during the classroom ask students to ask a question on the general channel.
- One of the things that will be handy, is for students to know how to capture a screenshot on their Chromebook. In this exercise, ask students to capture a screenshot on their computers and share that screenshot with you on Slack. This feature will be super helpful for when they need to ask your questions remotely and you need to see their code.
- Another tool that we use is an Chrome extension called Loom. In Loom, you or your students can record/capture their screen which can be extremely helpful in explaining things step by step. So it's a tool that you may use often (and not students) but even students can ask you questions on Loom but can show you step by step how they do it. We use Loom to guide students through specific instruction (how to debug their code or how to perform other tasks). Loom is also free for the most part. Watch the following video to learn about how to use Loom in your classroom.

**## PhantomJS not found.** You can install it with `webshot::install_phantomjs()`. If it is installed, please

- Teach your students about some of the most important apps they should use on their Chromebooks. We have mentioned some of these apps in this course but you can add your favorites to the list. You can also give an assignment that by next class, each student should find one app they think it's useful/cool that others should use as well. There are a lot of resources and lists on the most useful apps on Chromebook.

Google Apps for Education

# GA F E


## chromebook

powered by edtechteam

### 1. Keyboard Shortcuts

Ctrl + Alt + ? displays all shortcuts


- > To see modifier keys hold 'shift' or 'control' keys to view those shortcuts
- > Press 'Esc' to close keyboard shortcuts



More shortcuts @ [goo.gl/QWxJN](http://goo.gl/QWxJN)

### 2. Zoom in/out on Chromebooks

Press Ctrl + Alt + two-finger scroll up or down.  
Setup instructions @ [goo.gl/HfiLSh](http://goo.gl/HfiLSh)




### 3. Pin an App to the Task Bar

- > In the app launcher, right click to add an app to the task bar
- > Right click to Unpin tabs from the task bar
- > Access your app launcher from the web on Chrome from a PC or MAC: [chrome://apps/](http://chrome://apps/)

### 4. Search

Use the 'Search' button to conduct a search right from your keyboard. Search for:

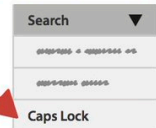
- > Web
- > Drive Files
- > Images
- > Apps



### 5. Caps Lock Key

Reassign the 'Search' button to be a 'Caps Lock' key:

- > Settings Menu (bottom right) > Settings (gear)
- > Keyboard Settings (towards bottom) > Search Drop Down
- > Change to Caps Lock




### 6. Screen Shots

Capture all or a portion of your screen

- > ctrl + [img icon] = full screen screenshot
- > ctrl + shift + [img icon] = screenshot a section


Note-If you are on a Chromebook with touch screen:  
Settings > More Tools > Take a Screenshot using touch screen



### 7. Split Screen

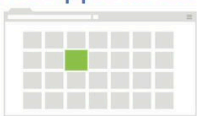
Create a split screen or simply toggle between two windows

- > Click the minimize button
- > Drag one tab out of the window



### 8. Pin a Webpage to Apps Screen


- > Go to webpage
- > Settings
- > More Tools
- > Add to Applications



### 9. Download Files


Screenshots and downloaded files appear in the 'Files' app.

- > App Launcher > Search for 'Files'



### 10. GA FE Training Center

Want more 'How To' tips? Go to the Google for Education Training Center on 'Advanced Chromebooks: [goo.gl/m2hCmw](http://goo.gl/m2hCmw)



#gafesummit

[www.edtechteam.com](http://www.edtechteam.com)








Figure 1: Chromebook cheatsheet

- Ask your students in the class to practice installing and uninstalling an app from the Google Play store. To use Google Play store, your students may need to create an account so you can help them with that as well.
- While we mention this in this course, ask your student to practice finding how much storage they have on their Chromebooks.
- Students should be comfortable using the Chrome browser by now. If you think they're not, this is a good opportunity to give them more practice. One of the exercises you can assign, is to practice bookmark favorite websites. Ask your students to go to a website they regularly check and bookmark that webpage. Then ask them to close the tab and go to the same webpage using the bookmark menu on the Chrome browser.
- One of the features of the Chrome browser is that it can save passwords so the user does not have to enter them every time they visit a website. Make sure you remind students of this feature and help them make an informed decision about whether they should or should not use this feature. This New York Times article discusses that. This article by Google also explains how to manage saved passwords on Google Chrome. Tell them they can sync their devices so the passwords are saved across all devices. They will have to turn sync on in Chrome. This is how.



# Course 2: Google and the Cloud

Devote this session to helping your students familiarize with Google products. This course is important because a lot of what we do in this program is on the cloud and Google provides many tools for working on the cloud.

## Exercise 1: Sharing Google Docs with Others

Level: Easy Length: Short

Ask your student to create a Google Doc from one of the templates available. After creating the document, ask them to share them with their classmates using their email addresses. Remind them about the different levels of access they can set when sharing a document with others (edit, comment, view). Then ask them to remove their classmates' access to the document.

## Exercise 2: Working on Google Docs offline

Level: Easy Length: Short

In this exercise, ask your student open the Google Doc they created in the previous exercise. Then ask them to disconnect from internet and see what happens. They see a message that they are offline and that to work on the document offline, they can turn on offline sync.

Now, ask them to turn on the offline mode and keep working on the document while they are not connected to the internet.

Now, ask them to get back online (connect to the internet) and see what happens. All the changes they made offline should be seen in the online version.

## Exercise 3: Creating a Google calendar event and inviting others

Level: Easy Length: Short

In this short exercise, ask your students to login to their Google Calendar (by going to [calendar.google.com](https://calendar.google.com) and logging in). Then ask them to create an event called "test event" that happens the next day at 10:00AM and ask them to invite at least one of their classmates to it.

The classmates should receive a calendar invite. Ask the recipients of the event to choose whether they are attending the event or not (or maybe).

## Exercise 4: Subscribing to a Google Calendar

Level: Easy Length: Medium

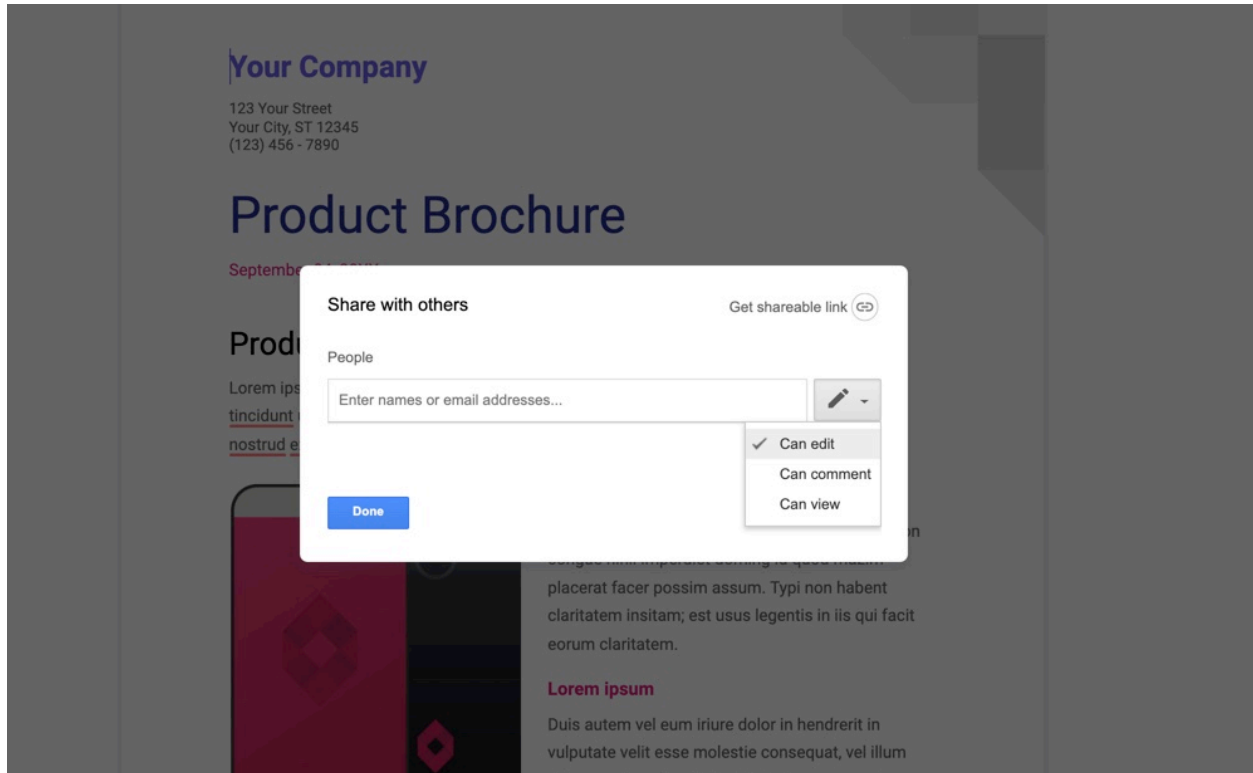


Figure 2: Sharing Google Docs

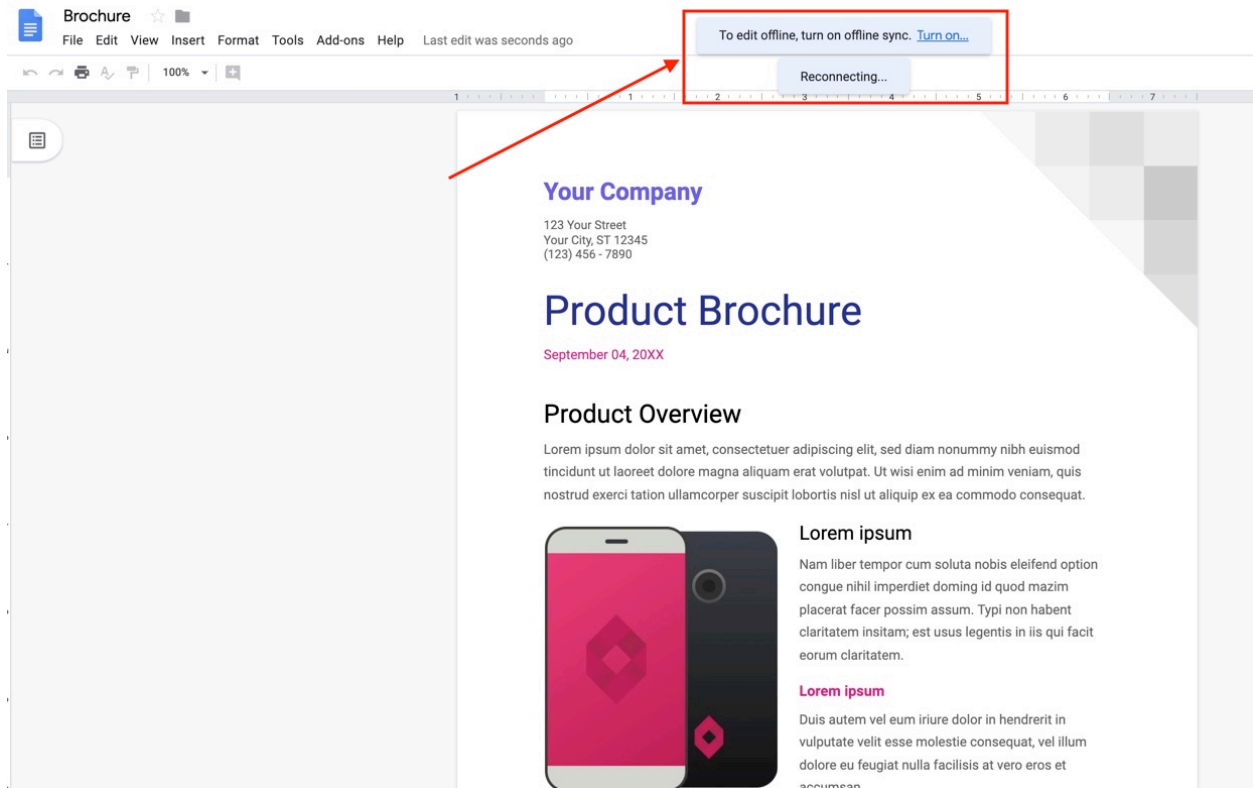


Figure 3: No internet connection while working with Google Docs

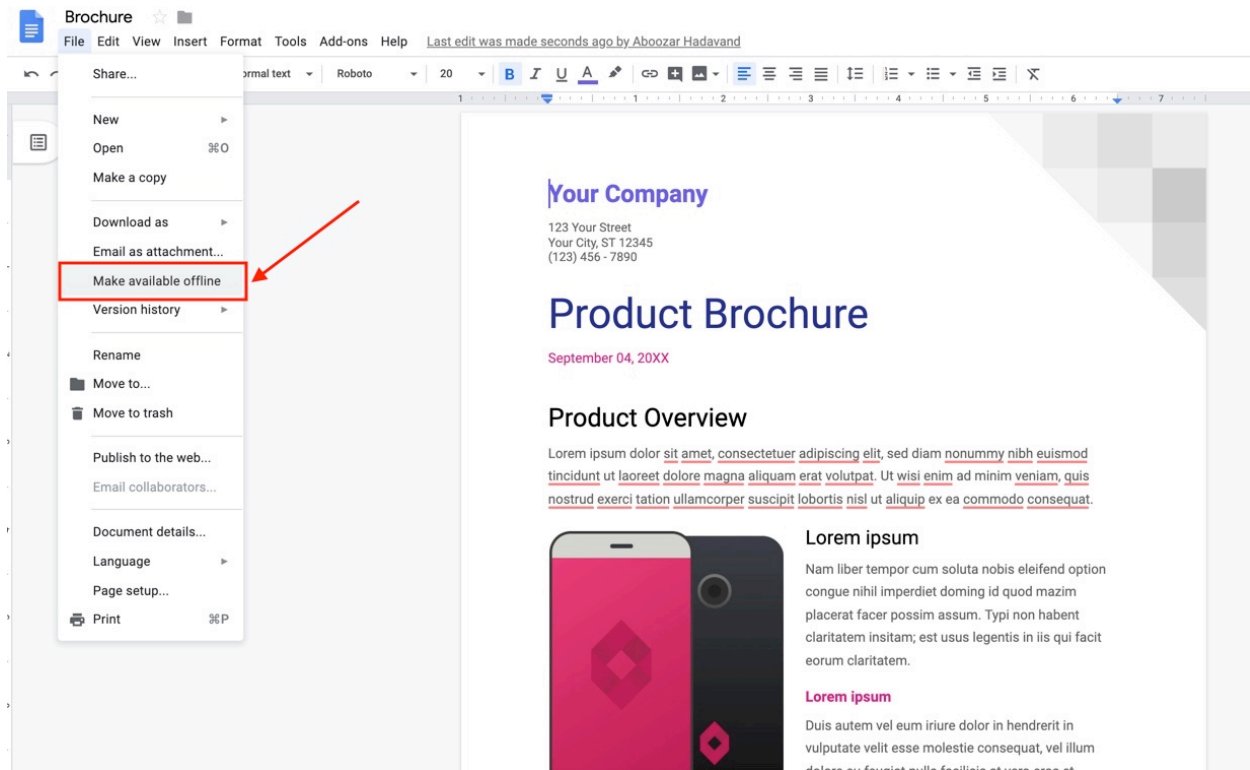


Figure 4: Working offline in Google Docs

For this exercise, you as an instructor, can first create a Google Calendar for your class with all the important events related to your class. To set up a new calendar, on your browser, open Google Calendar. Then on the left side, above “My calendars,” click Add other calendars Add and then New calendar.

Then, add a name and description for your calendar, and click Create calendar.

If you want to share your calendar, click on it in the left bar, then select Share with specific people.

You can also share the link to the calendar with others. For this, under Access Permissions, click on Get shareable link.

At the end, ask your students to subscribe to the calendar by accepting your invitations or by going to the link you shared with them. By subscribing to your calendar, they will have access to all the event you share in that calendar.

For more information on creating a calendar, you can go [here](#)

## Exercise 5: Deleting and recovering a file on Google Drive

Level: Easy Length: Short

In the first exercise, your students created a test Google Doc. In this exercise, they will learn how to recover a deleted file from the trash folder. Ask them to delete the file by clicking on File and then Move to trash.

To recover the file, they should go to Google Drive and click on the Trash folder. To restore the file, they have to find the file in the trash folder, then right click on it, and select Restore.

Once they click, they should be able to see a pop-up message in the bottom left corner of the screen. If they click on SHOW FILE LOCATION, they will be taken to the folder where the original file was located.

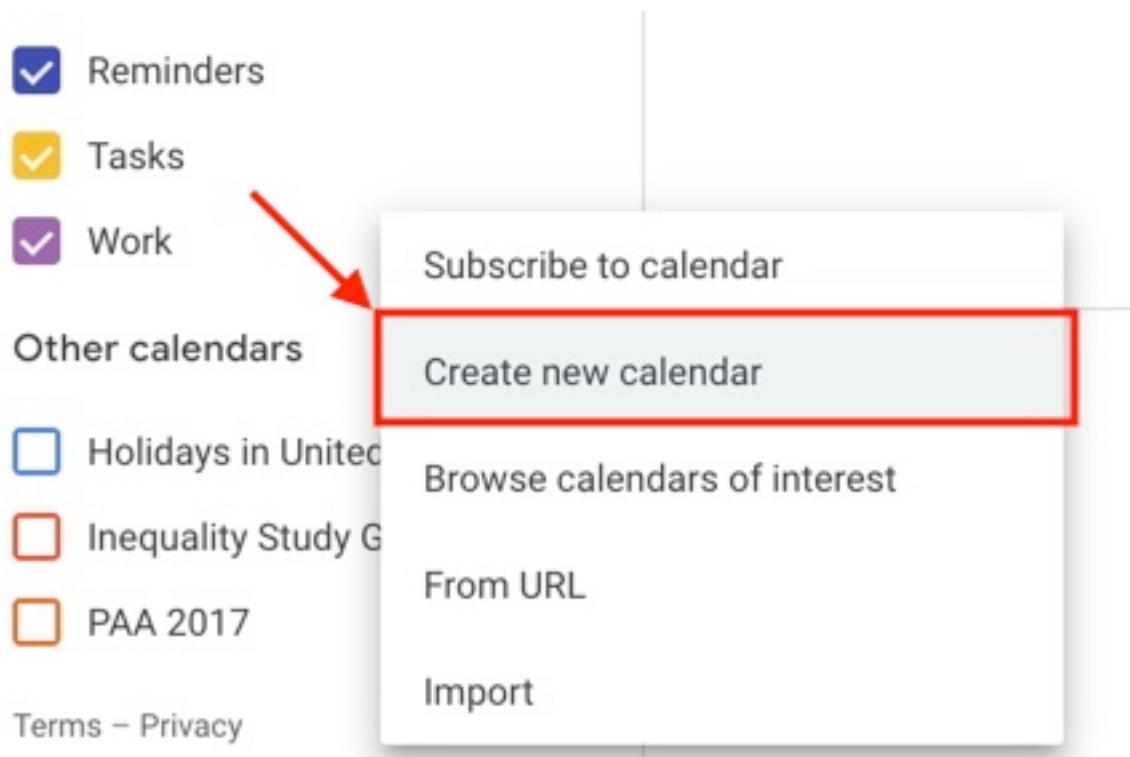


Figure 5: New Google Calendar

### Create new calendar

Name  
Classroom Calendar

Description  
This calendar is for all the events related to our classroom.

Time zone  
(GMT-04:00) Eastern Time - New York

Owner  
[Redacted]

Create calendar

Figure 6: Creating new Google Calendar



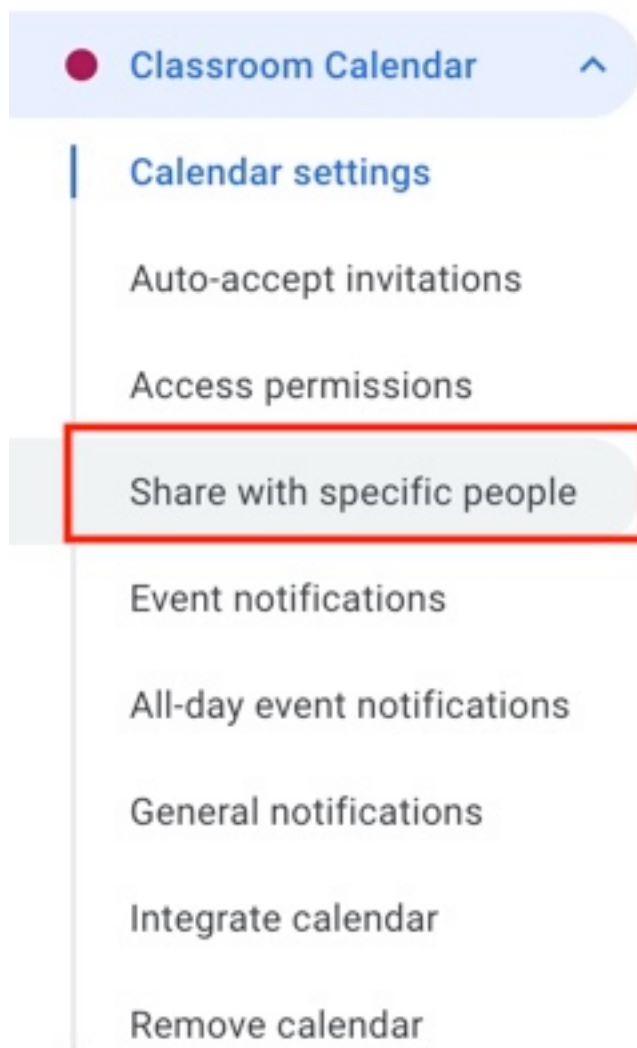


Figure 7: Sharing the new Google Calendar

### Access permissions



Make available to public

See all event details ▼

[Get shareable link](#)

Learn more about [sharing your calendar](#)

Figure 8: Get sharable link

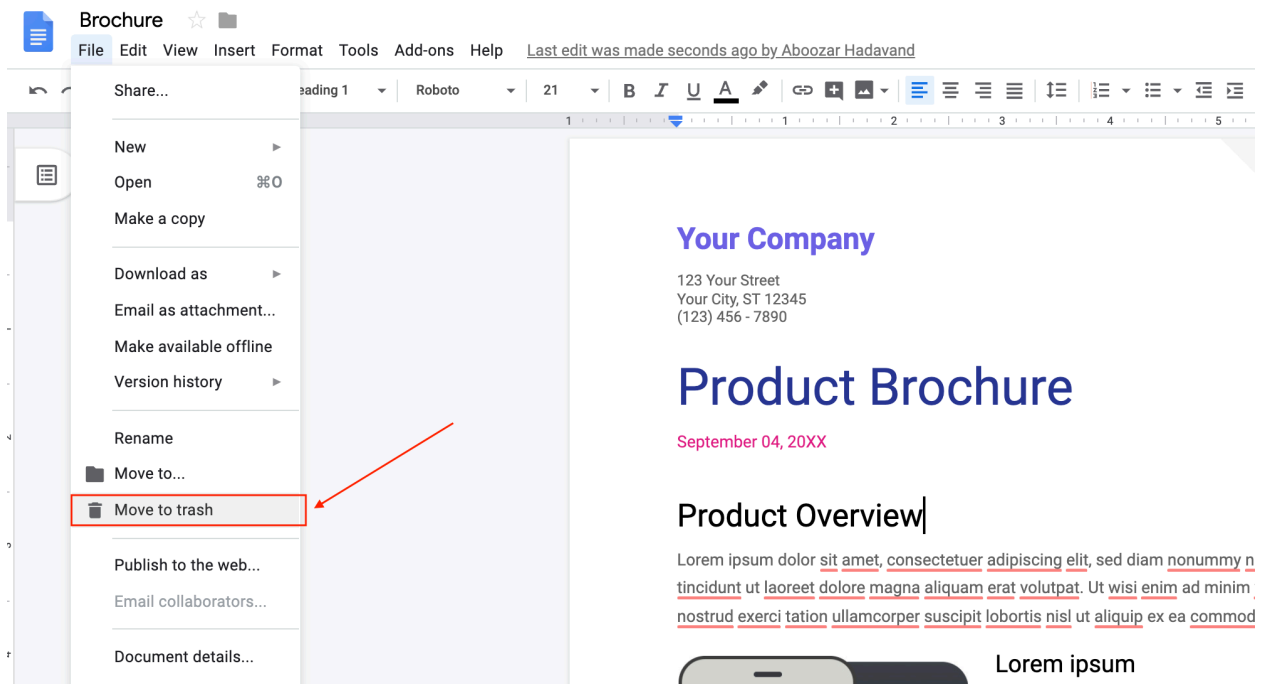


Figure 9: Deleting a Google Doc

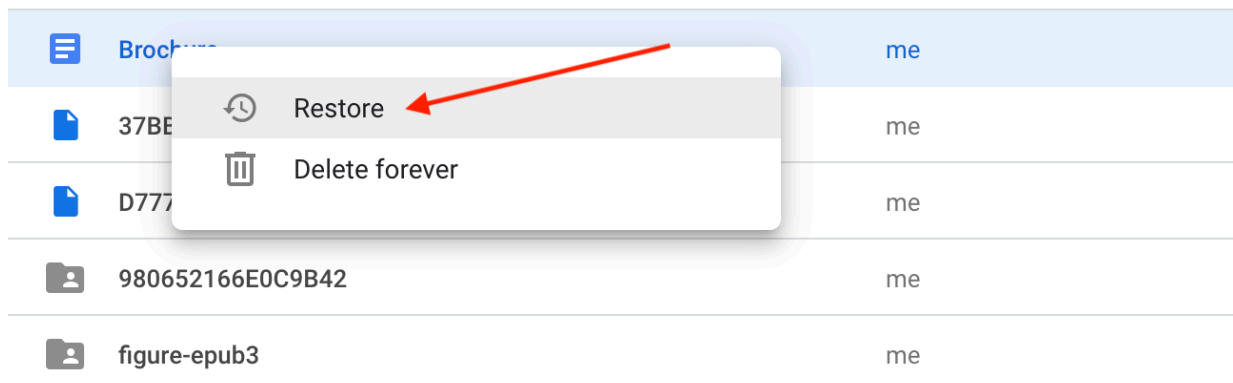


Figure 10: Restoring files in Google Drive

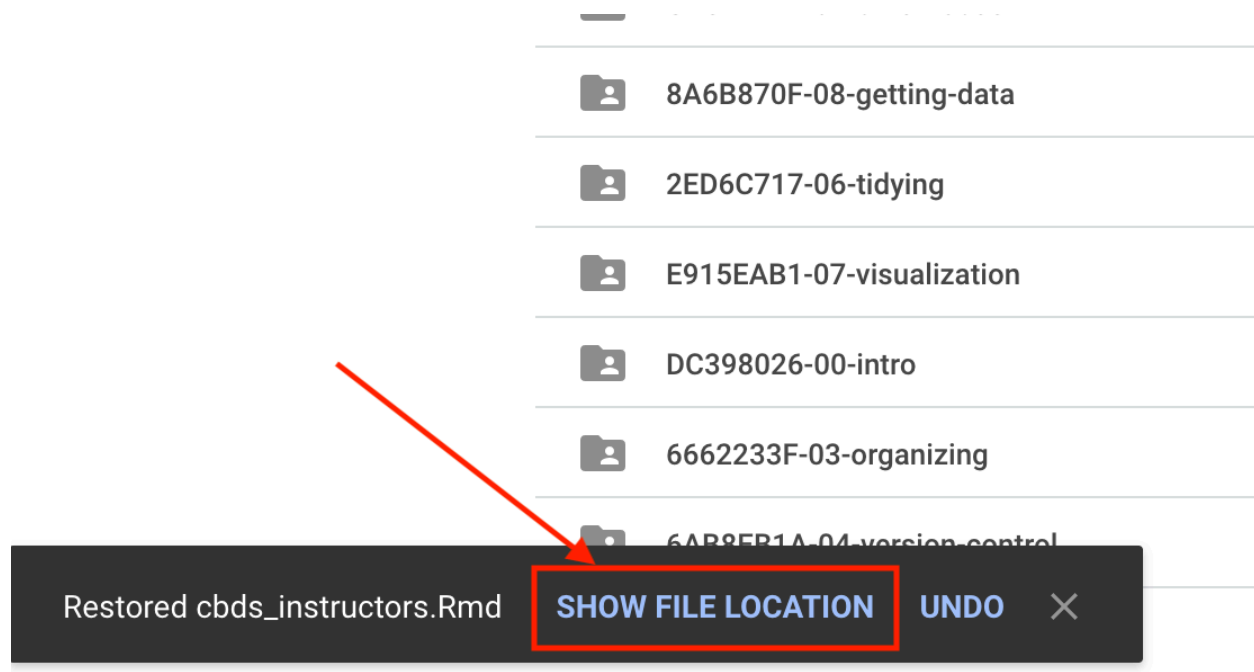


Figure 11: Restored file location

## Exercise 6: Using a resume template on Google Doc

Level: Easy Length: Medium

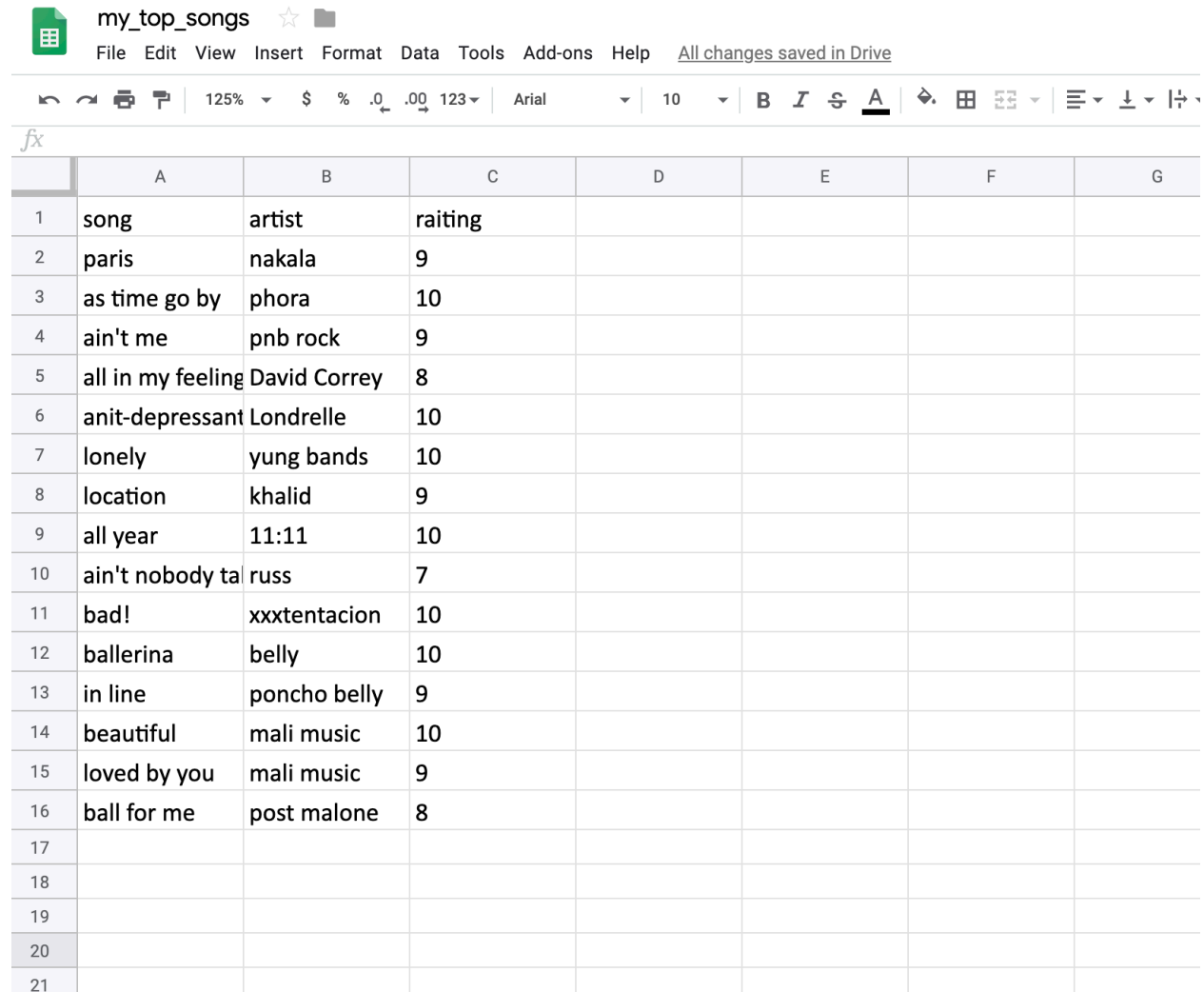
This exercise will be useful at the end of the program. Later in the program, students are encouraged to create their resumes (if they don't already have one) and updating it. In this session, you can ask students to create a new resume from templates on Google Doc and modify it by adding their personal info, their education, and other stuff. They will later have the chance to go back to this resume and tailoring it for data science jobs.

## Exercise 7: Creating a Google Sheets document

Level: Easy Length: Medium

Ask your students to create a blank Google Sheets document and create 3 columns: song, artist, year. Also ask them to name the file as `my_top_songs`.

Finally ask them to enter top 10 songs and their artists and years in the table like below.



The screenshot shows a Google Sheets interface with a document titled "my\_top\_songs". The menu bar includes File, Edit, View, Insert, Format, Data, Tools, Add-ons, and Help. The status bar indicates "All changes saved in Drive". The toolbar shows various editing and formatting options. The spreadsheet has columns A through G and rows 1 through 21. The data is as follows:

	A	B	C	D	E	F	G
1	song	artist	rating				
2	paris	nakala	9				
3	as time go by	phora	10				
4	ain't me	pnb rock	9				
5	all in my feeling	David Correy	8				
6	anit-depressant	Londrelle	10				
7	lonely	yung bands	10				
8	location	khalid	9				
9	all year	11:11	10				
10	ain't nobody ta	russ	7				
11	bad!	xxxtentacion	10				
12	ballerina	belly	10				
13	in line	poncho belly	9				
14	beautiful	mali music	10				
15	loved by you	mali music	9				
16	ball for me	post malone	8				
17							
18							
19							
20							
21							

We will later use this document in future exercises.

## Exercise 8: Creating a Google Slides document

Level: Easy Length: Medium

This exercise can be assigned as homework. Ask your students to create a Google Slides document with 4-5 slides that gives an introduction to who they are, what they do, and what their hobbies are. They can share as much as they are comfortable with. You can ask them to share the slides with you or present the slides in the next session when the class meets.

# Course 3: Organizing Data Science Projects

## Exercise 1: Replicating a Google doc as Rmd

Level: Easy Length: Medium

As your students to go to the link [https://docs.google.com/document/d/1vFQzQ6sRsMJyrqLY49qhEkxvQ0G1Zp\\_IMZjltpLvUw/edit?usp=sharing](https://docs.google.com/document/d/1vFQzQ6sRsMJyrqLY49qhEkxvQ0G1Zp_IMZjltpLvUw/edit?usp=sharing). Then ask them to go to Rstudio.cloud and create a new project. In the new project they should create a new .Rmd file. Instead of the text in the file, they should write text/code that will replicate the Google doc file in the link above. Once the knit the file as html/Word/pdf it should look similar to the Google doc.

## Exercise 2: Creating a New Project and the Workflow

Level: Easy Length: Short

This exercise is based on an exercise from the last course.

Ask your students to download the Gogole Sheets that they created on musicians. Then tell them to create a new project on RStudio.cloud and call it **my\_\_music\_\_project**. Then they should create all the folders necessary for a data science project as they've learned in the lesson. Ask them to upload the csv file they downloaded from Google Sheets in the folder **tidy\_\_data**.

The next step is to create an Rmd file called **Readme** in the main folder with the headline **Scope of the Project** as the headline and a list that says objective 1, objective 2, objective 3. Make sure they save the document.

Additionally you can ask them to find the album cover of one of the songs on their list (tell them to find the link to the cover on the web) and add the image in their Rmd file.

They should use the style:

```
![Name of the image](link to the image)
```

Inside the parenthesis they should insert the link to the image on the web. Ask them to *knit* the document and see if the image appears in the html output.



# Course 4: Version Control

## Exercise 1: Creating a new repository

Level: Easy Length: Short

In this easy exercise, ask your students to log in to Github and create a new repository named **test\_repo** and share the link to the repository with the class. The links should look something like:

[https://github.com/JaneEverydayDoe/test\\_repo](https://github.com/JaneEverydayDoe/test_repo)

and comprise of three sections:

1. The Github link (<https://github.com>)
2. The student's Github username (JaneEverydayDoe)
3. The repository name (test\_repo)

## Exercise 2: Adding an issue to a classmate's repository

Level: Easy Length: Short

In this exercise, ask your students to share the link to their Github repository with the person next to them (you can encourage them to share the link on Slack as a private message). Each student should then go to the link that is shared with them, go to the Issues section and create a new issue.

In the title and the body of the issue, ask them to type *This is just a test issue*.

## Exercise 3: Deleting a repository

Level: Easy Length: Short

Now that the students have created a test repository named **test\_repo**, ask them to delete the repository by going to the Settings tab and scrolling to the bottom of the page called the **Danger Zone**.

Students will have the option to archive or delete the repository. You can explain the difference between the two options. According to Github, archiving a repository will only mark the repository as archived and read-only and will not delete it (students will be able to restore the repo in future). However, deleting a repository will erase the repository forever. Since this was only a test repository, ask your students to delete the repository. Note that Github will ask them to confirm the decision by asking them to type in the name of the repository. Once they click on **I understand**, Github may ask them to log in again and then will delete the repo forever.

## Exercise 4: Creating an RStudio project from a Github repository

Level: Easy Length: Medium

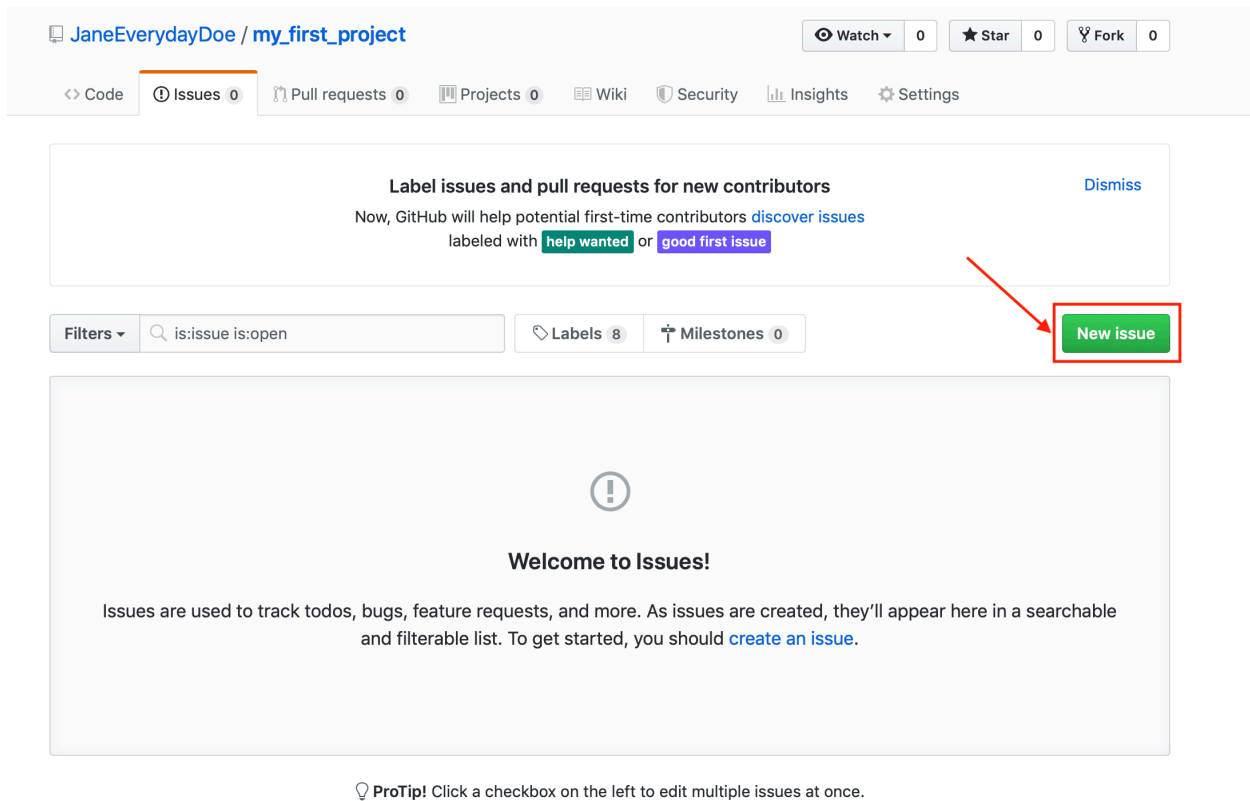


Figure 12: Github Issues

## Danger Zone

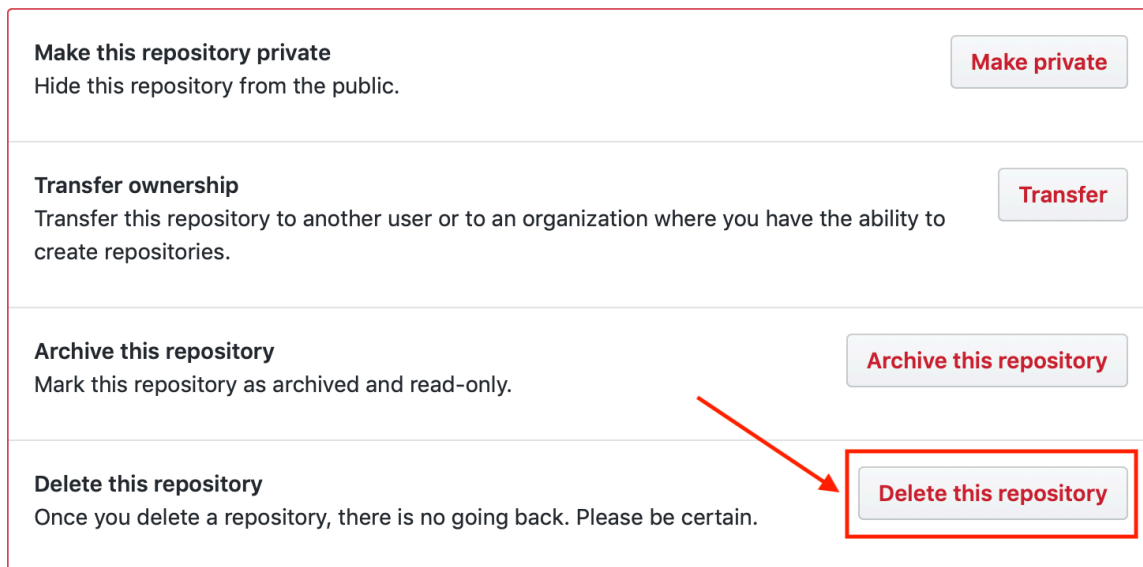


Figure 13: Deleting a repository



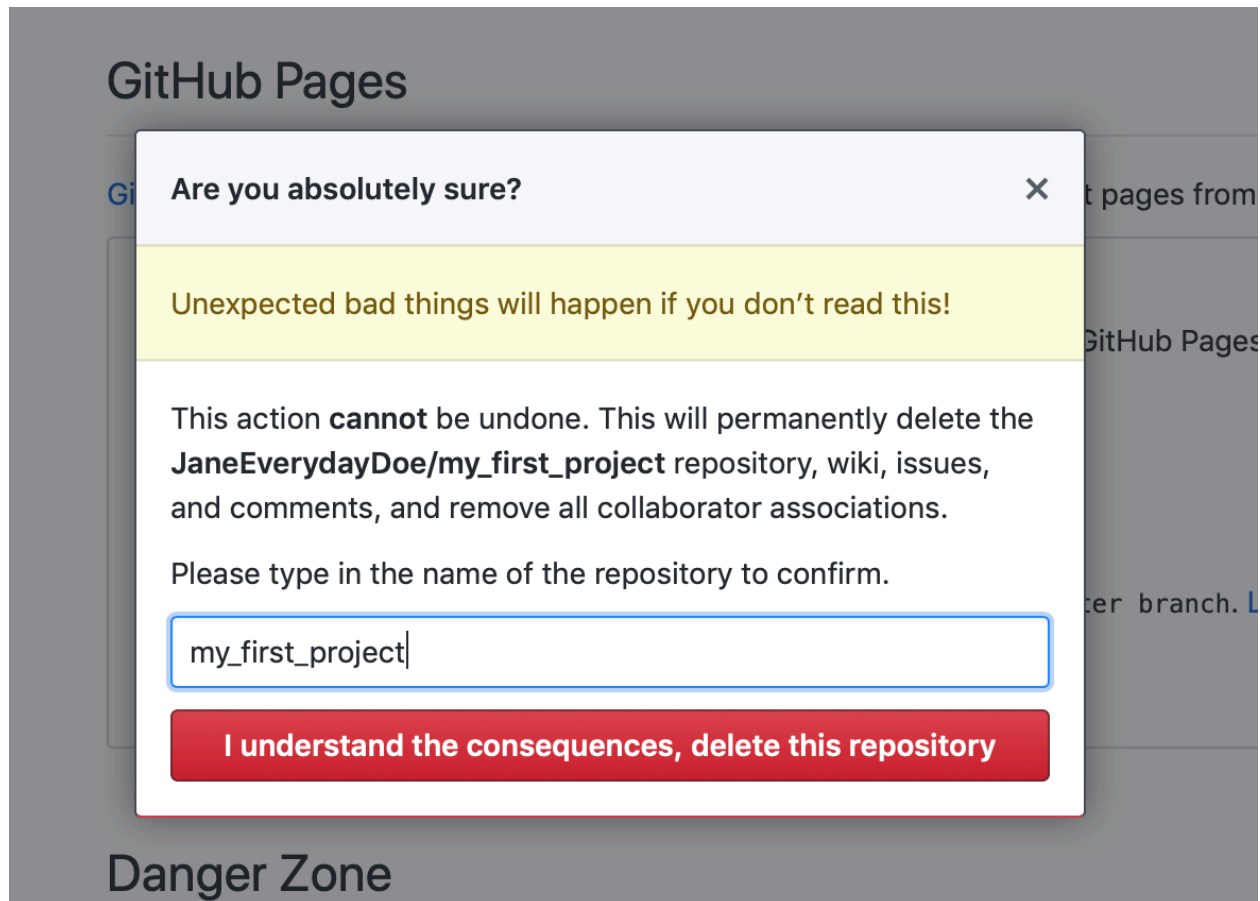


Figure 14: Popup window to confirm deleting a repository

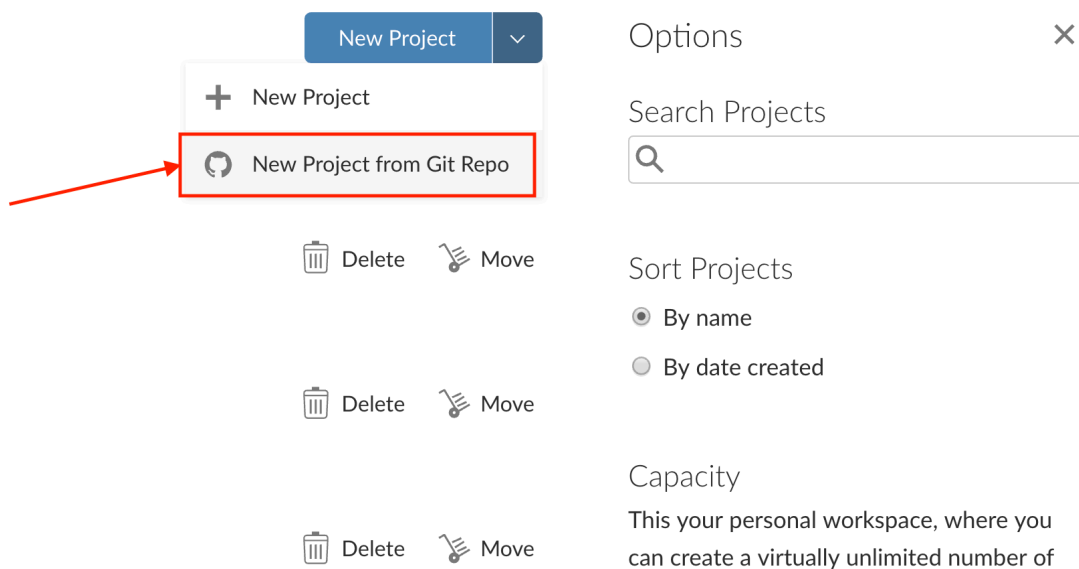


Figure 15: Create a new project from Git repo

Ask your students to create a new repository called **my\_music\_analysis**. Make sure the repository is public and they don't don't initiate it with a README file. Once they create it, ask them to copy the HTTPS link to the repo, something like [https://github.com/JaneEverydayDoe/my\\_first\\_project.git](https://github.com/JaneEverydayDoe/my_first_project.git).

Now, ask them to login to RStudio.cloud and create a new project from Git repo.

In the next window that pops up, ask them to paste the link they copies from Github and press OK.

Once they click on OK, they will see an empty project (with only the .Rproj file). Now, ask your students to create the folder structure necessary for a data science project as they learned from the course Organizing Data Science Projects. If they don't remember, ask them to refer to the course.

In one of the class activities in the course Google and the Cloud, your stduents had to create a Google Sheets document in which they listed their top favorite songs as well as the artists and the production year. Ask your students to go to that document on Google Sheets and download it as a .csv files. To do this, they have to click on the File menu and choose Download as and then select Comma Separated File (.csv).

After they downloaded the file as .csv, ask them to upload it on the newly created project on RStudio.cloud in the **tidy\_data** folder.

The last step is to ask them to push everything on Github. Make sure they follow the main three steps: add, commit, and push. Once they complete pushing, they should be able to see the uploaded file on the remote repository on Github.

Create a new repo called "" (make sure you DON'T initiate with readme) Go to your RStudio.cloud project called **my\_music\_project** and push your project to the repository you made above. Follow these instructions <https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/> Check your repository on github to see if your project is there. Make some changes to your repository on github and commit your changes Go to your project on rstudio.cloud and pull your changes Go to somebody else's repository and add an issue there and ask a question about their repo

Look at the website <http://jhudatascience.org/chromebookdatascience/> The website is hosted here <https://github.com/jhudsl/chromebookdatascience> Fork the repo and then clone it to a new project on rstudio.cloud Edit the file **faq.Rmd**. Add just a random line, knit the document and then push it to github. Go to the

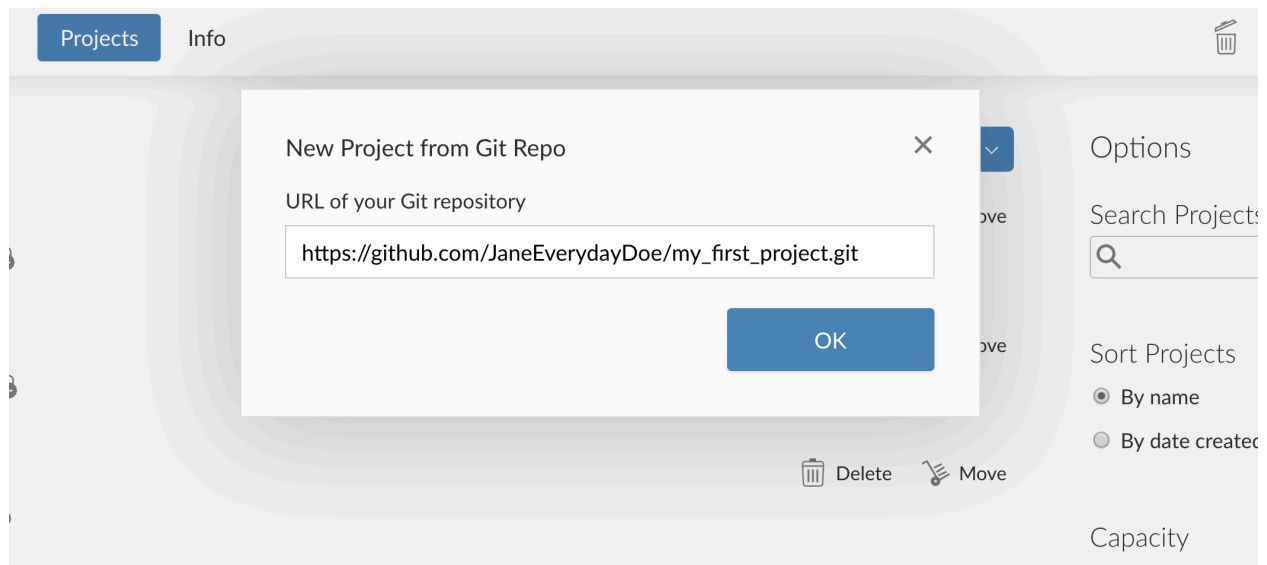


Figure 16: Pasting the Github HTTPS link

- data/
  - raw\_data/
  - tidy\_data/
- code/
  - raw\_code/
  - final\_code/
- figures/
  - exploratory\_figures/
  - explanatory\_figures/
- products/
  - writing/

Figure 17: Folder structure

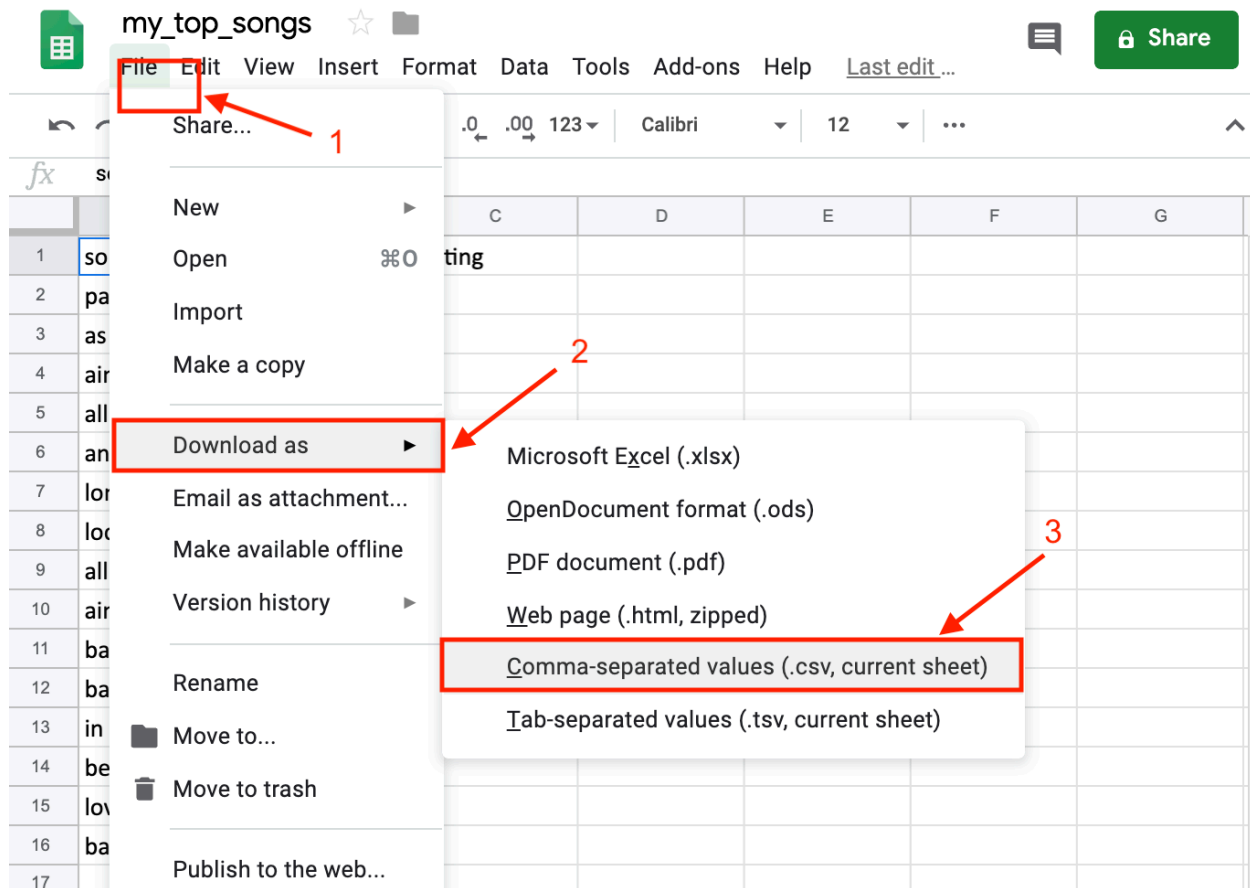


Figure 18: Download Google Sheets document as .csv

repo on jhudsl again and make a pull request. If you had to summarize what github does in one sentence, how would you describe it

clone [https://github.com/JaneEverydayDoe/my\\_first\\_project](https://github.com/JaneEverydayDoe/my_first_project) modify it

1. Going back to the rstudio cloud project they called my\_\_music (or something similar).
2. finding the google sheet they created that included the table of their top 10 songs and downloading it as a csv
3. In the rstudio project creating all the necessary folders for an organized data science project (like tidy\_\_data/raw\_\_data, tidy\_\_code/etc). they should know the folder structure from the previous course. You can ask them to go back to the course to recall how to do this
4. Uploading the csv file in step 2 to the tidy\_\_data folder
5. pushing everything on github
6. going to somebody else's repository, forking it, then cloning it, then edit the csv file by adding another row (of their favorite song), then push it to their own github, then going to that person's repository and making a pull request.



# Course 5: Introduction to R

One of the most important exercises for this course is to make students comfortable with RStudio and RStudio cloud. These are some of the exercises you can ask students to do during the class.

- Uploading files on RStudio.cloud
- Exporting (downloading) files on RStudio.cloud
- Renaming files in the Files section
- Deleting files in the Files section
- Creating new folders in the Files section
- Looking at the packages installed
- Looking at the environment through the Environment tab
- Looking at history through the History tab
- Browsing over the Console and the Terminal
- Creating a new R script file
- Creating a new R Markdown file
- Opening a new R script file

## Exercise 1: Installing and Uninstalling Packages

Level: Easy Length: Short

Start with telling students what CRAN is. CRAN hosts additional packages that sit on top of the core (base) R software. While there are thousands of packages on CRAN, a lot of the packages are hosted on other hosting sites such as Github. In this exercise, we will ask students to install packages from both CRAN and Github.

First, ask students to install the **devtools** package. Ask them to use the package name without the quotations as below:

```
install.packages(devtools)
# Error in install.packages : object 'devtools' not found
```

They will get an error saying the “object ‘devtools’ not found”. Remind them when we install packages, we have to put the package name in quotations. So the correct way is:

```
install.packages("devtools")
```

The package **Devtools** is hosted on CRAN, therefore, we don’t have to specify where the package is hosted. We just have to pass the name of the package to the function `install.packages()` to install it. Now ask them to load the package. Remind them that to load packages we use the following function:

```
suppressWarnings(library(devtools))
```

Note that when we load packages we don’t necessarily have to use quotation marks. It is recommended, however, that to use quotations for both `install.packages()` and `library()` functions. The package **devtools** makes package development easier. One of the applications of the package is that it allows us to

install packages that are not on CRAN. Now ask students to install the package **knockknockjokes** from github using. The link to the Github repo of the package is [www.github.com/psolymos/KnockKnockJokes](https://github.com/psolymos/KnockKnockJokes). Ask them how they are supposed to install a package from Github.

```
suppressWarnings(devtools::install_github("psolymos/KnockKnockJokes"))
```

```
## Downloading GitHub repo psolymos/KnockKnockJokes@master
```

```
## from URL https://api.github.com/repos/psolymos/KnockKnockJokes/zipball/master
```

```
## Installing KnockKnockJokes
```

```
## '/Library/Frameworks/R.framework/Resources/bin/R' --no-site-file \
```

```
## --no-environ --no-save --no-restore --quiet CMD INSTALL \
```

```
## '/private/var/folders/6c/94269x114tlc1x1c901c6_8m0000gn/T/RtmpbX5MFA/devtoolse28b25c4c2a6/psolymos
```

```
## --library='/Library/Frameworks/R.framework/Versions/3.4/Resources/library' \
```

```
## --install-tests
```

```
##
```

```
## Reloading installed KnockKnockJokes
```

Note that we have to specify where the package is hosted when we use the command `install_github()` from the package **devtools**. `psolymos` is the username of the user who developed the package and `KnockKnockJokes` is the name of the repository. Now ask students to load the package.

```
suppressWarnings(library(KnockKnockJokes))
```

This package is very simple. It basically tells you knock knock jokes. You can ask students to browse through the package documentation on its Github repo and see what function they should use for a joke. Here's one for a random joke:

```
KnockKnock()
```

```
## Knock-knock!
```

```
## Who's there?
```

```
## Sally.
```

```
## Sally who?
```

```
## Sally-brate the moments of your life!
```

As an assignment (in class or at home), you can ask students to find an R package they find interesting and tell the class about it.

## Exercise 2: Using the script editor

Level: Easy Length: Short

Ask students to go to the `my_music_project` on RStudio.cloud that they previously created. Ask them to create a new R script file in the folder `code/tidy_code` and save it as `exploratory_analysis`. Ask them to install and load the package **dplyr** in the beginning of the file. One of the nice things about RStudio is its keyboard shortcuts. One that is used a lot is `Ctrl+Enter`.

Tell your students to type in `2 + 2` in the script file and while the cursor is on that line, hold the `Ctrl` (Command) key and then press `Enter` (or `Return`). What happens? `Ctrl+Enter` is used for running code without clicking on the Run tab. It makes running codes line by line a lot easier. As long as the cursor is somewhere on the line, `Ctrl+Enter` will run that line.

You can also remind them that they can select and highlight multiple line and use `Ctrl+Enter`. For instance, ask them to type these two lines in an R script file and highlight both line together and then use `Ctrl+Enter`.

```
x <- 2
```

```
x + 3
```



```
## [1] 5
```

## Exercise 3: Basic Commands in R

Level: Easy Length: Short

In this exercise, we will perform some basic commands in R on simple vectors. First, ask your students to create an vector object called `ages` that contains the following numbers:

```
ages <- c(22, 18, 20, 25, 22, 19, 32, 17)
```

After giving them the above numbers and saving the object, ask your students to use an R function to find whether there is anybody who is older than 30 years old. Tell them you simply just need a TRUE/FALSE answer to this question. They can use the following command:

```
any(ages > 30)
```

```
## [1] TRUE
```

Now, ask them to use an R function to find whether all ages are above 15. Again, remind them that the answer should be a simple TRUE/FALSE. They should use something like:

```
all(ages > 15)
```

```
## [1] TRUE
```

You can further ask them to find which ages are larger than 20. For this, they can use the function `which()` to obtain the index of the element that is larger than 20 like below.

```
a <- which(ages > 20)
a
```

```
## [1] 1 4 5 7
```

```
ages[a]
```

```
## [1] 22 25 22 32
```

## Exercise 4: Creating a Data Frame

Level: Medium Length: Long

In this exercise, students will learn how to create a data frame. But since they've already learned how to create a repository on Github and push things to Github, ask them to create a repository there called `movie_project`. Further, ask them to clone the repository to an RStudio.cloud project with the same title. In the RStudio project, they should create all the necessary folders as they've learned before.

We are going to create a data frame that has four columns: `title`, `year`, `rating`, and `genre` for the top 10 movies on imdb. Ask your students to go to the link [here](#). Ask them to create an R script file in the folder called `code` and type their code there. Based on the information on the link, ask them to input the title, year, rating, and genre information for the top 10 movies in a data frame. Ask them to call the data frame `movies_df`. They should do something like:

```
title <- c("The Shawshank Redemption", "The Godfather", "The Dark Knight", "The Godfather: Part II",
          "The Lord of the Rings: The Return of the King", "Pulp Fiction", "Schindler's List",
          "12 Angry Men", "Fight Club", "The Lord of the Rings: The Fellowship of the Ring")
year <- c(1994, 1972, 2008, 1974, 2003, 1994, 1993, 1957, 1999, 2001)
rating <- c(9.3, 9.2, 9.0, 9.0, 8.9, 8.9, 8.9, 8.9, 8.8, 8.8)
genre <- c("Drama", "Crime", "Action", "Crime", "Adventure", "Crime", "Biography", "Drama", "Drama", "Action")
movies_df <- data.frame(title, year, rating, genre)
```

Now that they've created the data frame, ask them to show the first 4 rows of the data:

```
head(movies_df, 4)
```

```
##           title year rating genre
## 1 The Shawshank Redemption 1994    9.3  Drama
## 2           The Godfather 1972    9.2  Crime
## 3           The Dark Knight 2008    9.0 Action
## 4 The Godfather: Part II 1974    9.0  Crime
```

Ask them to call the column that contains information about movie titles.

```
movies_df$title
```

```
## [1] The Shawshank Redemption
## [2] The Godfather
## [3] The Dark Knight
## [4] The Godfather: Part II
## [5] The Lord of the Rings: The Return of the King
## [6] Pulp Fiction
## [7] Schindler's List
## [8] 12 Angry Men
## [9] Fight Club
## [10] The Lord of the Rings: The Fellowship of the Ring
## 10 Levels: 12 Angry Men Fight Club Pulp Fiction ... The Shawshank Redemption
```

Ask them to find the “class” of the columns year and genre:

```
class(movies_df$year)
```

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

```
class(movies_df$genre)
```

```
## [1] "factor"
```

Ask your student to find the difference between the maximum and the minimum rating in the data.

```
dif <- max(movies_df$rating) - min(movies_df$rating)
dif
```

```
## [1] 0.5
```

Then ask them to find the dimension of the data frame:

```
dim(movies_df)
```

```
## [1] 10  4
```

Now ask them to change the column called `title` to `name`:

```
colnames(movies_df)[1] <- "name"
```

Without checking manually, ask them to see if the movie “Groundhog Day” is among the top 10 movies:

```
any(movies_df$name == "Groundhog Day")
```

```
## [1] FALSE
```

Ask them to find the number of the movies are in the genre Drama:

```
drama <- which(movies_df$genre == "Drama")
length(drama)
```

```
## [1] 3
```

```
d <- list(movies = c("m1", "m2"),
         states = 4)
```

## Exercise 5: Creating Lists in R

Level: Easy Length: Short

The beauty of lists are that you can have elements with different sizes and structures in them. For this exercise, ask your students to create a list with three elements. The first element called `places` should be a vector of the states (provinces) or countries your students have been to. The next element called `ice_cream` should contain the name of their top two ice cream flavors. The last element called `sports` should contain their favorite sports. The list should be called `about_me`. The code should look like:

```
about_me <- list(places = c("New Mexico", "Vermont", "Washington", "California"),
               ice_creams = c("Vanilla", "Chocolate"),
               sports = c("Soccer"))
```

Ask them to call the first element of the list:

```
about_me[[1]]
```

```
## [1] "New Mexico" "Vermont"      "Washington" "California"
```

```
## note that they can also call it by
about_me[["places"]]
```

```
## [1] "New Mexico" "Vermont"      "Washington" "California"
```

Now, ask them to add another sport that they like to the element `sports`.

```
about_me[["sports"]][[2]] <- "Volleyball"
```

## Exercise 6: Missing Values in R

Level: Moderate Length: Medium

Ask your students to load the `airquality` dataset in R. Ask them to show the columns that exist in the data and find the dimension of the data.

```
colnames(airquality)
```

```
## [1] "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
```

```
dim(airquality)
```

```
## [1] 153 6
```

Ask them to look at the first 10 rows of the data:

```
head(airquality, 10)
```

```
##      Ozone Solar.R Wind Temp Month Day
## 1      41      190  7.4   67     5   1
## 2      36      118  8.0   72     5   2
## 3      12      149 12.6   74     5   3
```

```
## 4      18      313 11.5   62      5   4
## 5      NA      NA 14.3   56      5   5
## 6      28      NA 14.9   66      5   6
## 7      23     299  8.6   65      5   7
## 8      19      99 13.8   59      5   8
## 9       8      19 20.1   61      5   9
## 10     NA     194  8.6   69      5  10
```

Do they see any missing values in the first 10 rows? They should answer yes as the first 10 rows clearly has some missing values. Now, ask them to use the appropriate function that checks whether the column Temp has missing values. How about the column Ozone?

```
any(is.na(airquality$Temp))
```

```
## [1] FALSE
```

```
any(is.na(airquality$Ozone))
```

```
## [1] TRUE
```

The column Ozone, contains mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island. What rows in the column Ozone have missing values

```
which(is.na(airquality$Ozone))
```

```
## [1] 5 10 25 26 27 32 33 34 35 36 37 39 42 43 45 46 52
## [18] 53 54 55 56 57 58 59 60 61 65 72 75 83 84 102 103 107
## [35] 115 119 150
```

Now, ask your students to use the appropriate function to calculate the average of the column Ozone.

```
mean(airquality$Ozone)
```

```
## [1] NA
```

If they use the code above, the answer will be NA. Tell them that this is due to the missing values in the column and because of that, the function `mean()` does not calculate the average. For this, to see the average, they will have to use the argument `na.rm` and set it to `TRUE`. By doing this, they will tell R to ignore the missing values and only perform the function on the non-missing values.

```
mean(airquality$Ozone, na.rm = TRUE)
```

```
## [1] 42.12931
```

Now, ask them to remove all the rows containing missing values from the entire data and save the new object as `airquality_nomissing`. Note that they should use the function `na.omit()`. What is the dimension of the new data?

```
airquality_nomissing <- na.omit(airquality)
dim(airquality_nomissing)
```

```
## [1] 111 6
```

Finally, ask them to write code that will replace all values of the column Wind that are equal to 8.0 with NA.

```
airquality$Wind[airquality$Wind == 8.0] <- NA
```

## Exercise 7: Reverse a String Function

Level: Difficult Length: Long

Ask your students to write a function that takes a string and returns the reverse version of the string. So the function returns the string “book” as “koob”. This function is a simple version of a decoding function. Let’s decompose this problem. There are probably various ways to do this problem. One way to go about solving this problem is to decompose the words into characters. Then reverse the order of the characters and then paste them back together. The first step can be done using the `str_split()` function from the **stringr** package.

```
library(stringr)
word <- "book"
word_decom <- str_split("book", "")
word_decom
```

```
## [[1]]
## [1] "b" "o" "o" "k"
```

The `word_decom` object is a list and its first element is a vector of all the character that make up the word *book*. If we have a vector, we can use the function `rev()` to reverse the order of its elements. So `rev(c(1,2,3))` will return 3 2 1. So to reverse the order of characters in the object `word_decom` we write

```
word_decom_rev <- rev(word_decom[[1]])
word_decom_rev
```

```
## [1] "k" "o" "o" "b"
```

`[[1]]` is for calling the vector in the list. Now, we have the decomposed word reversed. But the result is a vector of characters and not a string. To create a string from all the characters, we can use the function `paste()` as we have used before.

```
word_rev <- paste(word_decom_rev, collapse = "")
word_rev
```

```
## [1] "koob"
```

Your students now can combine all the steps and create the function. This would be the simplest form of the function.

```
word_reverse <- function(x){
  word_decom <- str_split(x, "")
  word_decom_rev <- rev(word_decom[[1]])
  word_rev <- paste(word_decom_rev, collapse = "")
  return(word_rev)
}
word_reverse("chromebook")
```

```
## [1] "koobemorhc"
```

## Exercise 8: A simple function

Level: Easy to moderate Length: Medium

Write a function that takes user’s name and returns “Hello, Name!” Note that they can use the function `paste()` for pasting two or more character strings. The argument `sep` determines how the strings are going to be attached, e.g. if `sep = "_"` then the strings will be separated by an underline.

```
hello <- function(name){
  print(paste("Hello ", name, "!", sep = ""))
}
```

Write a function that takes number 1 and number 2 and checks whether number 1 is divisible by number 2.

If divisible, returns a message saying the first number is divisible by the second number and if not returns a message saying the first number is not divisible by the second number.

```
check_division <- function(num1, num2){
  if (num1 %% num2 == 0) {
    print(paste(num1, " is divisible by ", num2, ".", sep=""))
  } else {
    print(paste(num1, " is NOT divisible by ", num2, ".", sep = ""))
  }
}
```

In the function above, check to make sure number 1 is bigger than number 2. If not, halt and show a message that the first number should be larger than the second number.

```
check_division <- function(num1, num2){
  if(num1 < num2) stop("The first number should be larger than the second number.")
  else {
    if (num1 %% num2 == 0) {
      print(paste(num1, " is divisible by ", num2, ".", sep=""))
    } else {
      print(paste(num1, " is NOT divisible by ", num2, ".", sep = ""))
    }
  }
}
```

## Exercise 9: The scan function

Level: Easy Length: Short

For this exercise, create a user defined function that takes values from the user using the `scan` function and returns the values.

```
readfun <- function(){
  myvalues = scan()
  return(myvalues)
}

print(readfun())
```

## Exercise 10: Function for the class of a data frame

Level: Easy Length: Short

Ask your students to write a function that takes a data frame as input and will print how many columns the data frame has (e.g. The data frame has x columns.) and the name of each column and the class of data each column contains using the `cat` function (e.g. Variable1 is Numeric).

They can use different functions for printing the messages but one of the options is the `cat` function. Note that the `"\n"` is for line breaks. If they don't use it, everything will be printed in one line.

```
classfun <- function (df) {
  cat("The data frame has", ncol(df), "columns", "\n")
  for (i in 1:ncol(df)) {
    cat(names(df)[i], "is", class(df[, i]), "\n")
  }
}
```

```
}  
classfun(cars)  
  
## The data frame has 2 columns  
## speed is numeric  
## dist is numeric
```

## Exercise 11: Function for the # of times a value is repeated

Level: Easy Length: Short

Tell your students to write a function that takes a vector as input and a value and return how many times the value is repeated inside the vector. They have seen the `which()` and `length()` functions in the previous exercises so they should be comfortable doing this exercise.

```
numfun <- function(v, value){  
  c <- which(v == value)  
  return(length(c))  
}  
numfun(c(1, 1, 1, 4, 0, 4, 3, 3), 4)  
  
## [1] 2
```

## Exercise 12: Cumulative sum function

Level: Easy Length: Short

Ask your students to Google a function in `r` that when takes a numerical vector, calculates a vector in which each element is the cumulative sum of the previous elements. For instance, vector `c(1, 3, 6, 4)` is converted to `c(1, 4, 10, 14)`

Hint: They'll likely find the function `cumsum()`. You can also encourage them to write their own function without using the `cumsum()` function.

## Exercise 13: Cumulative sum function

Level: Medium Length: Short

Ask your students to write a function that takes a vector as input and returns the second highest or second lowest value in the vector depending on what the user chooses.

Note that to programmatically pass a function name (`max` or `min`) they should use the function `get()`. The `get()` function returns the value of a named object.

```
second_maxmin <- function(x, maxmin){  
  fname = get(maxmin)  
  fname(x[x < max(x)])  
}  
second_maxmin(c(2,3,4,5,6,7), maxmin = "max")  
  
## [1] 6
```





# Course 6: Data Tidying

Creating an object such as a data frame and saving it as a rda or rds file `save(mpg, cyl, file="mtcars_objects.rda")`  
`load("mtcars_objects.rda") saveRDS(mpg, file="mpg.rds") mpg <- readRDS("mpg.rds")`

Write an encoder function that takes a word from the user and encodes each character in the alphabet to the next character and returns it. e.g. "peach" will be encoded to "qfbdi"

Let's create a data frame in R And then save the data as rda and rds and then load the data and share the data on slack Let's go to your imdb project on RStudio.Cloud. Create an Rmd file in your code folder. Edit the YAML in the file. Load the imdb dataframe. The rest of the session we'll work with different columns in the data. Push all the changes to Github

```
str_view(names, "^M") str_view(names, "M$")  
str_detect() str_count() str_subset()
```

## Exercise 1: Tidying NBA Finals Data

Level: Difficult

In this exercise we will work with NBA final teams stats on Kaggle. There is two .csv files. The 'champs.csv' file contains game-by-game team totals for the championship team from every finals game between 1980 and 2017. The 'runners.csv' contains game-by-game team totals for the runner-up team from every finals game between 1980 and 2017. The 1980 NBA Finals was the first Finals series since the NBA added the three point line.

This exercise is inspired by two kernels on Kaggle that can be found [here](#) and [here](#). Ask your student to download the data on Kaggle or on our Github repository for this guide.

In order to download data on Kaggle, students are required to create a Kaggle account. They can do so by linking their Google account.

The first step is to import the data into R. Since importing .csv files is in a future lesson, tell your students the commands they need to import the .csv files. We are going to use `read_csv` command from the **readr** package. Ask them to save the two files in two separate objects. Here we called them **champs** and **runners**.

```
knitr::opts_chunk$set(echo = TRUE)  
library(tidyverse)  
# importing the data  
champs <- read_csv(file = "../data/nba/championsdata.csv")  
  
## Parsed with column specification:  
## cols(  
##   .default = col_double(),  
##   Team = col_character()  
## )
```

```
## See spec(...) for full column specifications.
runners <- read_csv(file = "./data/nba/runnerupsdata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character()
## )
## See spec(...) for full column specifications.
```

You can ask your students to find out what column contains team names. They can go to the Kaggle link to find out. The column is called `Team`.

Now ask them to find the *distinct* team names. The function used to find the distinct values of a variable is `distinct()`. You can either tell them the function or ask them to google and find the function themselves. Here's how they can do it in a pipe.

```
## check the names, see any issue
champs %>%
  select(Team) %>%
  distinct()
```

```
## # A tibble: 13 x 1
##   Team
##   <chr>
## 1 Lakers
## 2 Celtics
## 3 Sixers
## 4 Pistons
## 5 Bulls
## 6 Rockets
## 7 Spurs
## 8 Heat
## 9 Mavericks
## 10 'Heat'
## 11 Warriors
## 12 Cavaliers
## 13 Warriorr
```

Once your students run the code above, they will notice some peculiar values. For instance 'Heat' and Warriorr. These kind of mistakes can happen in any data analysis. Ask your students to fix the issue using a `dplyr` function. Here's how we did it using the `ifelse()` function. The line `mutate(Team = ifelse(Team == "Warriorr", "Warriors", Team))` creates a column called `Team` which is the same as the old column `Team` except whenever the value in the column is `Warriorr` it is replaced by `Warriors`. Note that there are other ways to do this so you can ask your students to figure out a way on their own.

The other fix we need to do is to convert the two columns `Win` and `Home`. Ask your students to look at the class of the two columns. They will see that the values stored in the two columns are integer. Ask them to convert those two columns to factors and save the new columns back to the columns `Win` and `Home`. Here's how to do so for both datasets.

```
# fix the heat and warriors
champs <- champs %>%
  mutate(Team = ifelse(Team == "Warriorr", "Warriors", Team)) %>%
  mutate(Team = ifelse(Team == "'Heat'", "Heat", Team)) %>%
  mutate(Win = as.factor(Win)) %>%
  mutate(Home = as.factor(Home))
```

```
runners <- runners %>%
  mutate(Team = ifelse(Team == "Warriorrrs", "Warriors", Team)) %>%
  mutate(Team = ifelse(Team == "'Heat'", "Heat", Team)) %>%
  mutate(Win = as.factor(Win)) %>%
  mutate(Home = as.factor(Home))
```

Now, a good exercise would be to bind the two dataframes together. Ask your students to find the appropriate **dplyr** function for binding the rows together. Note that since the two dataframes have the same set of columns, this is easily done. They can use the function `bind_rows()` function as is in the lessons or use the base R function `row.bind()`.

```
## bind the two data sets together
allteams <- bind_rows(champs, runners)
```

It is in general better when we bind two datasets to have an indicator variable that shows from which dataset each row comes from. This can be done using the `.id` argument in the `bind_rows()` function. Note that by saying `"Champion" = champs` and `"Runner" = runners` we are making the identifier equal to `Champion` for the `champs` dataframe and equal to `Runner` for the `runners` dataframe.

```
## add a column that shows champions vs. runnerups call it rank
allteams <- bind_rows("Champion" = champs, "Runner" = runners, .id = "rank")
```

## Exercise 2: Calculating Age Based on Birthdate

In this exercise, your students will learn how to work with data objects. They have to combine their skills in writing functions and working with the package **lubridate**. Ask them to install and load the package **lubridate**.

```
install.packages("lubridate")
library("lubridate")
```

First, ask them to create an object called and call it `mybday` that stores their birthday.

```
library(lubridate)
mybday <- ymd("1998-09-20")
```

Note that they can also save their birthday like:

```
mybday <- dmy("20-Sep-1998")
```

Now ask your students to find the function in R that returns today's date. You can ask them to google this as it's likely that the first link has the answer to their question. The function they need to use is `Sys.Date()`. Ask them to save today's date to an object called `today`.

```
today <- Sys.Date()
```

The interesting thing about date objects is that we can do algebraic operations on them just like we do to numeric objects. We can add them or subtract one from another. Now, ask your students how they would calculate their age based on the two objects. They probably say they'd subtract `mybday` from `today` and that's true. When they do so, they will realize the answer is in days and not in years.

```
today - mybday
```

```
## Time difference of 7610 days
```

Well, then can then easily convert the answer to years by dividing the number by 365 or they can use the function `time_length()` again from the **lubridate** package.

```
time_length(today - mybday, unit = "year")
```

```
## [1] 20.84932
```

Now that your students are comfortable calculating age based on a birthdate and current date, ask them to write a function that will take a person's birthdate and will return the age in years. This should be easy given the solution above.

```
age_calc <- function(bday){
  bday = ymd(bday)
  today = Sys.Date()
  dif = time_length(today - bday, unit = "year")
  return(dif)
}
age_calc("1982-05-10")
```

```
## [1] 37.22466
```

Tell your students to add a warning message that is shown to user warning them to enter date as “yyyy-mm-dd”. So the function should show a warning message after the calculation is done.

```
age_calc <- function(bday){
  warning("Please enter the date as 'yyyy-mm-dd'")
  bday = ymd(bday)
  today = Sys.Date()
  dif = time_length(today - bday, unit = "year")
  return(dif)
}
age_calc("1982-05-10")
```

```
## Warning in age_calc("1982-05-10"): Please enter the date as 'yyyy-mm-dd'
```

```
## [1] 37.22466
```

Next, you can tell your students to modify the function so that the user determines whether age should be reported in years, months, days, hours, etc. In other words, the `unit` argument should be entered by user but its default value is in days.

```
age_calc <- function(bday, unit = "day"){
  warning("Please enter the date as 'yyyy-mm-dd'")
  bday = ymd(bday)
  today = Sys.Date()
  dif = time_length(today - bday, unit = unit)
  return(dif)
}
age_calc("1982-05-10", unit = "hour")
```

```
## Warning in age_calc("1982-05-10", unit = "hour"): Please enter the date as
## 'yyyy-mm-dd'
```

```
## [1] 326088
```

As the last step, show this function to your students and ask them to guess what the output will look like without running the function in R.

```
age_calc <- function(bday, unit = "day"){
  warning("Please enter the date as 'yyyy-mm-dd'")
  bday = ymd(bday)
  today = Sys.Date()
```

```
    dif = round(time_length(today - bday, unit = unit), 1)
    print(paste("You are ", dif, " ", unit, "s ", "old", sep=""))
}
age_calc("1982-05-10", unit = "year")
```

```
## Warning in age_calc("1982-05-10", unit = "year"): Please enter the date as
## 'yyyy-mm-dd'
## [1] "You are 37.2 years old"
```



# Course 7: Data Visualization

## Exercise 1: Visualizing NBA Finals Data

In this exercise we will work with NBA final teams stats on Kaggle. We have used this data before in the course. There is two .csv files. The 'champs.csv' file contains game-by-game team totals for the championship team from every finals game between 1980 and 2017. The 'runnerups.csv' contains game-by-game team totals for the runner-up team from every finals game between 1980 and 2017. The 1980 NBA Finals was the first Finals series since the NBA added the three point line.

This exercise is inspired by two kernels on Kaggle that can be found [here](#) and [here](#). Ask your student to download the data on Kaggle or on our Github repository for this guide.

In order to download data on Kaggle, students are required to create a Kaggle account. They can do so by linking their Google account.

The first step is to import the data into R. Since importing .csv files is in a future lesson, tell your students the commands they need to import the .csv files. We are going to use `read_csv` command from the **readr** package. Ask them to save the two files in two separate objects. Here we called them **champs** and **runners**. Like before, we will tidy the data a bit to fix the team name issues and converting the columns **Win** and **Home** to factors.

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
# importing the data
champs <- read_csv(file = "../data/nba/championsdata.csv")

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character()
## )

## See spec(...) for full column specifications.
runners <- read_csv(file = "../data/nba/runnerupsdata.csv")

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character()
## )

## See spec(...) for full column specifications.
# fix the heat and warriors
champs <- champs %>%
  mutate(Team = ifelse(Team == "Warriorrs", "Warriors", Team)) %>%
  mutate(Team = ifelse(Team == "'Heat'", "Heat", Team)) %>%
```

```

mutate(Win = as.factor(Win)) %>%
mutate(Home = as.factor(Home))

runners <- runners %>%
  mutate(Team = ifelse(Team == "Warriorrrs", "Warriors", Team)) %>%
  mutate(Team = ifelse(Team == "'Heat'", "Heat", Team)) %>%
  mutate(Win = as.factor(Win)) %>%
  mutate(Home = as.factor(Home))
## bind the two data sets together
## and add a column that shows champions vs. runnerups call it rank
allteams <- bind_rows("Champion" = champs, "Runner" = runners, .id = "rank")

```

After importing and wrangling the data, ask your students to figure out whether *champion* teams are likely to score more at home or away. Ask them to write the code in a pipe and present the table using the **knitr** package.

```

## loading the knitr package
library(knitr)
## does home has an advantage
allteams %>%
  ## first filtering so we only look at champion teams
  filter(rank == "Champion") %>%
  ## grouping by the variable home
  group_by(Home) %>%
  ## calculating the average of points scored
  summarise(avg = mean(PTS)) %>%
  ## presenting the table using the knitr package
  kable("html")

```

Home

avg

0

98.21101

1

103.31532

Ask your students what this means. To make the table above make more sense, ask your students to use the appropriate function for changing factor levels so that the column *Home* is equal to *Home* if its value is 1 and is equal to *Away* if its value is 0. Your students should use the function `fct_recode()` in order to do that. Now they know that champion teams on average score 98.2 points at away games and 103.3 points at home games.

```

## loading the knitr package
library(knitr)
## does home has an advantage
allteams %>%
  ## first filtering so we only look at champion teams
  filter(rank == "Champion") %>%
  ## changing factor levels so they mean better
  mutate(Home = fct_recode(Home, Home = "1", Away = "0")) %>%
  ## grouping by the variable home
  group_by(Home) %>%
  ## calculating the average of points scored

```



```
summarise(avg = mean(PTS)) %>%
## presenting the table using the knitr package
kable("html")
```

Home

avg

Away

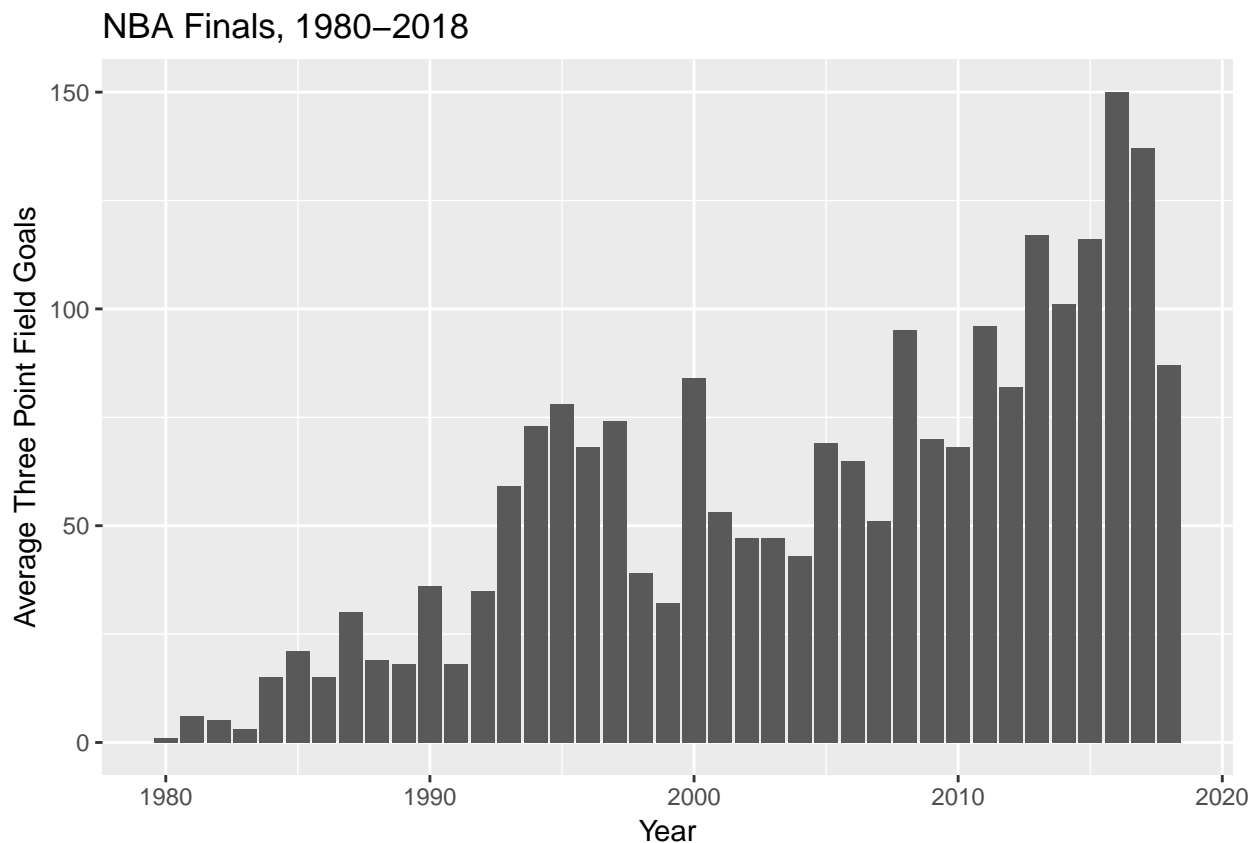
98.21101

Home

103.31532

Now, we are going to do some data visualization using the package **ggplot2**. Ask them to first find the column that contains the three point field goals from the data guide. The column is TP. Then ask them to show the average number of three points per year using a bar plot. Note that they will have to use the `geom_bar()` function from the **ggplot2** package. Ask them to use the title “NBA Finals, 1980-2018” as the title of the graph and “Average Three Point Field Goals” as the label for the Y axis. Note that they will have to use the argument `stat = 'identity'` inside the `geom_bar()` function.

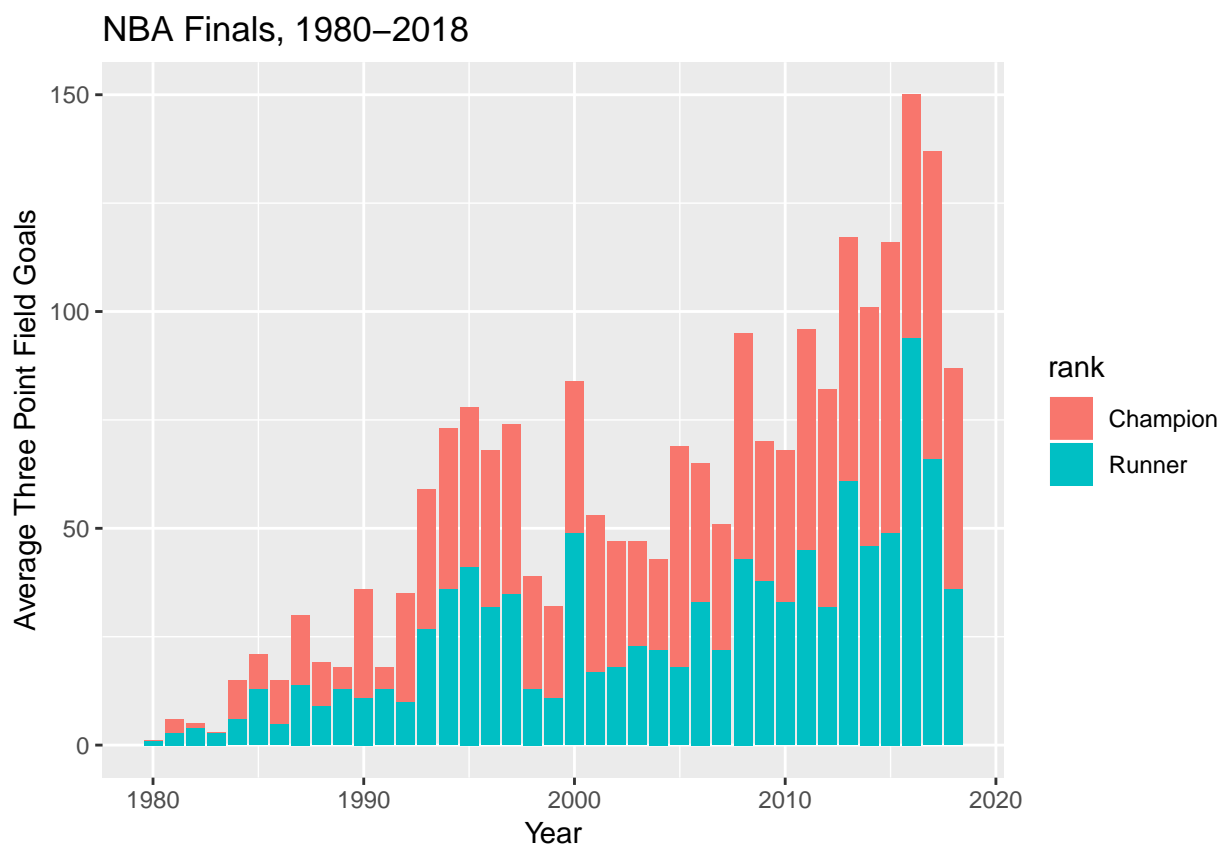
```
ggplot(data = allteams, aes(x = Year, y = TP)) +
  geom_bar(stat = 'identity') +
  ggtitle("NBA Finals, 1980-2018") +
  ylab('Average Three Point Field Goals')
```



Now, ask them to repeat the previous graph but this time, have the bar plot (with different colors) for champion and runner up teams. In other words, they will have to use the argument `fill` and make it equal

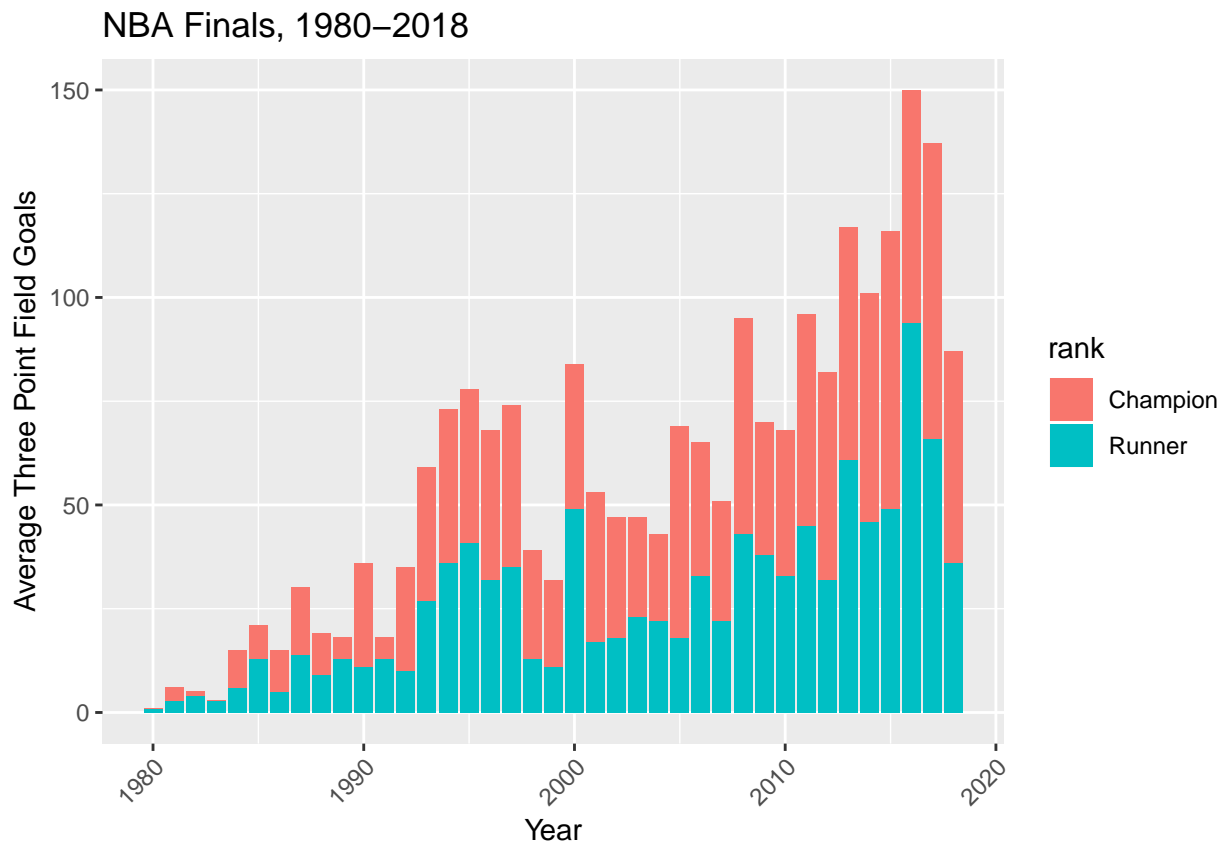
to the newly created column `rank`. Then ask them to interpret the graph. What is obvious is that champion teams do a much better job at three-pointers.

```
p <- ggplot(data = allteams, aes(x = Year, y = TP, fill = rank)) +  
  geom_bar(stat = 'identity') +  
  ggtitle("NBA Finals, 1980-2018") +  
  ylab('Average Three Point Field Goals')  
p
```



Now, ask them to find the right function in **ggplot2** that makes the Y axis ticks (the years) with an angle (45 degrees). They will have to use the `theme()` argument and inside it they will have to use the `element_text()` function to specify the angle.

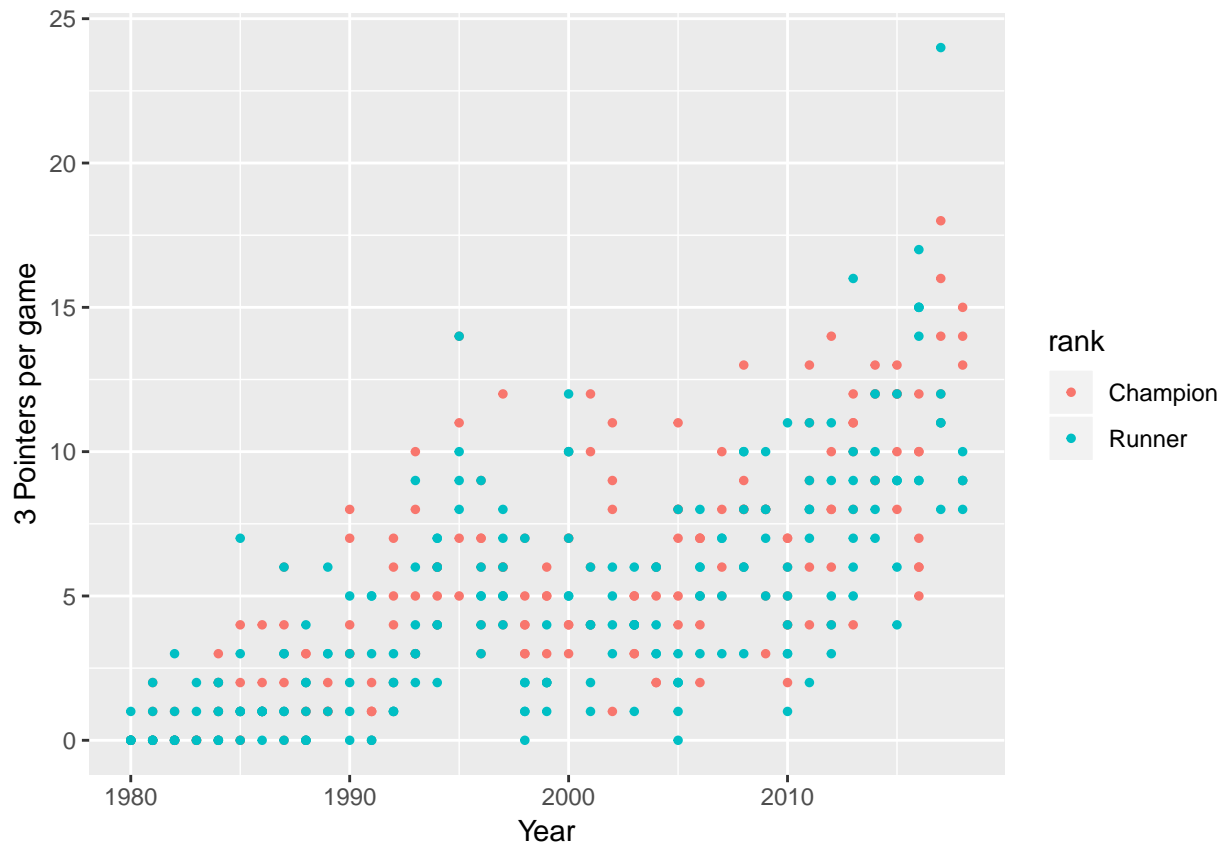
```
p + theme(axis.text.x = element_text(angle=45, hjust = 1, vjust = 1))
```



In the last exercises, we are going to work with scatter plots. Ask your students have a scatter plot with number of 3 pointers per game on the vertical axis and year on the horizontal axis. Note that the two columns they need to use are TP and Year. Ask them to have two different colors for champions and runner ups. Ask them to make the point sizes equal to 1.

```
p2 = ggplot(data = allteams, aes(x = Year, y = TP, color = rank)) +  
  geom_point(size = 1) +  
  ylab('3 Pointers per game')
```

p2



Your students are now required to use the `geom_smooth()` function from the **ggplot2** package. This function help your students in seeing patterns. This is an example of the `geom_smooth()` function.

When using `geom_smooth()`, the argument `method` has to be specified otherwise its default value will be used. Ask your students to set `method` as `'loess'`). The shaded areas around the lines are the confidence intervals which can be disabled by using the argument `se = FALSE`.

`geom_smooth()` and `stat_smooth()` are effectively aliases: they both use the same arguments. Use `stat_smooth()` if you want to display the results with a non-standard geom. Ask students what the two lines suggest? What is the trend over time?

```
p2 + geom_smooth(method = 'loess')
```

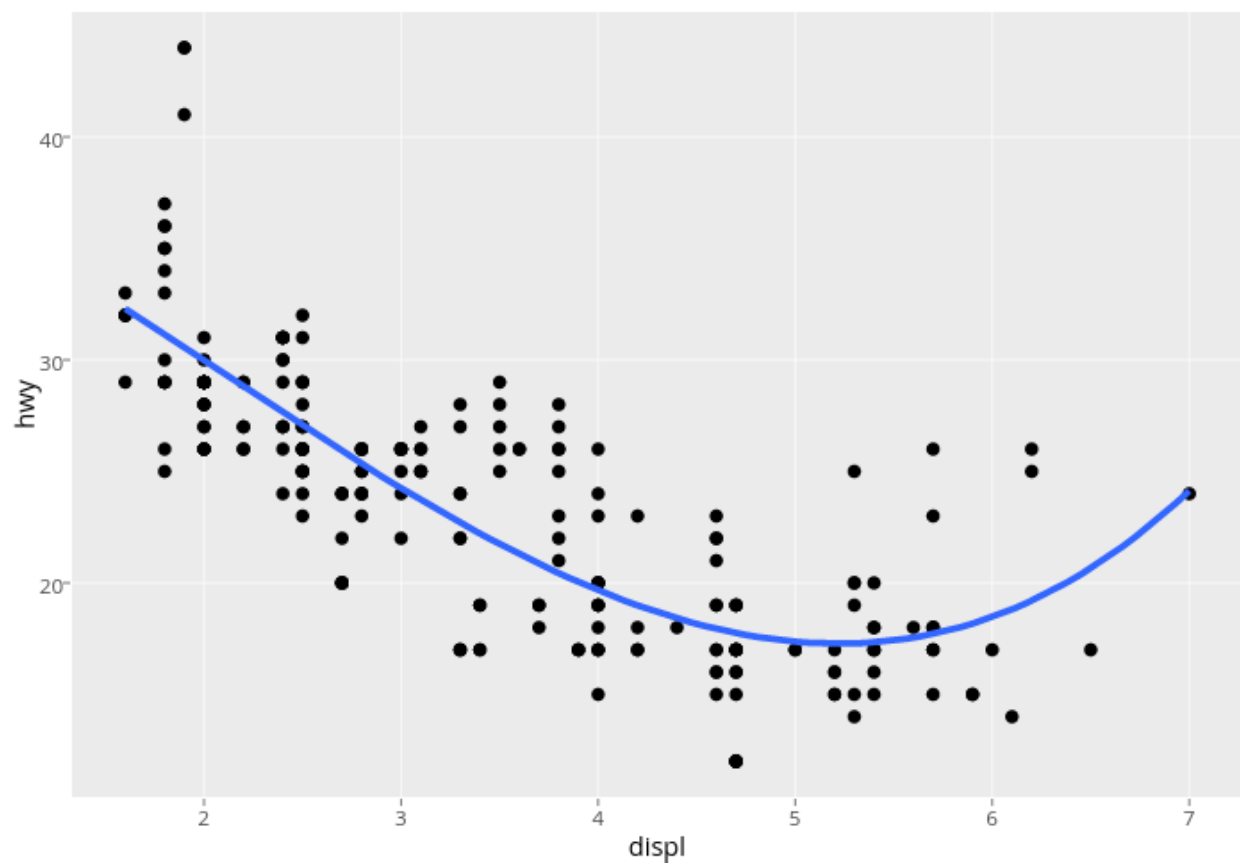
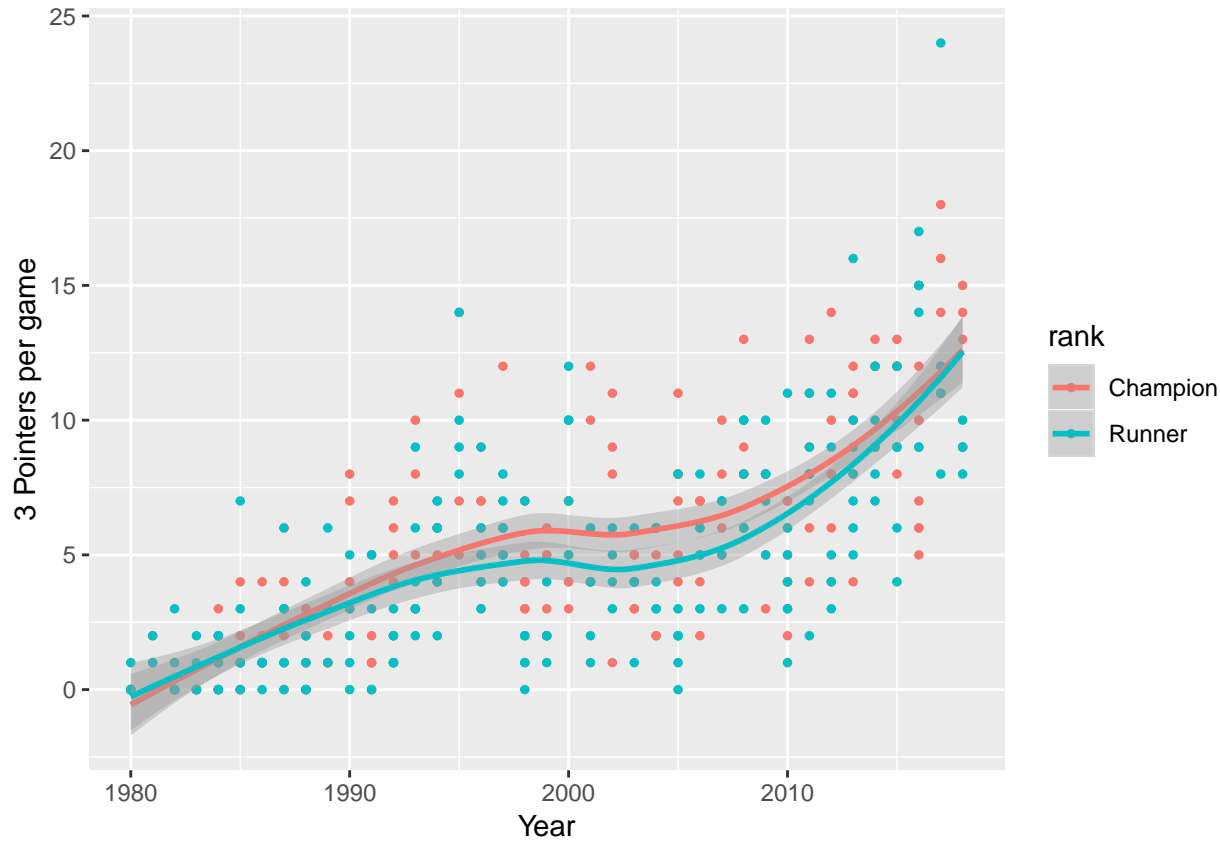


Figure 19: `geom_smooth()` function, linear approximation



# Course 8: Getting Data

## Exercise 1: Saving a data frame as an rds file

Level: Easy

In this exercise, your student will learn how to save a data frame as a rds or rda file. Before, this exercise remind your students about the best practices of web scraping. Too much scraping at higher rates will put pressure on the website servers. There is a great article here about the best practices of web scraping.

First ask your students to load the dataset **mtcars** and save it to an object called **data**.

```
data <- mtcars
head(data)
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710      22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant         18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Now ask them to simply save the data frame **data** as an rds file with the same name. They should use a code similar to

```
saveRDS(data, file = "data.rds")
```

You can also ask them to save the data frame as an rda file:

```
save(data, file="data.rda")
```

## Exercise 2: Scraping IMDB data

Level: Difficult

In this exercise, your students will practice working with the package **rvest** and scraping data from the web. We'll more specifically scrape top 250 movies from the website IMDB. This information can be found on this link. Ask your students to load the packages **rvest**, **tidyverse** and **stringr**.

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

The first step in using the **rvest** package and scraping data from the web is to read the content of the webpage we're interested in. For this, we use the function **read\_html()** from the package **rvest**. The address above only contains the first 50 movies but we can use similar process for the pages that come after.

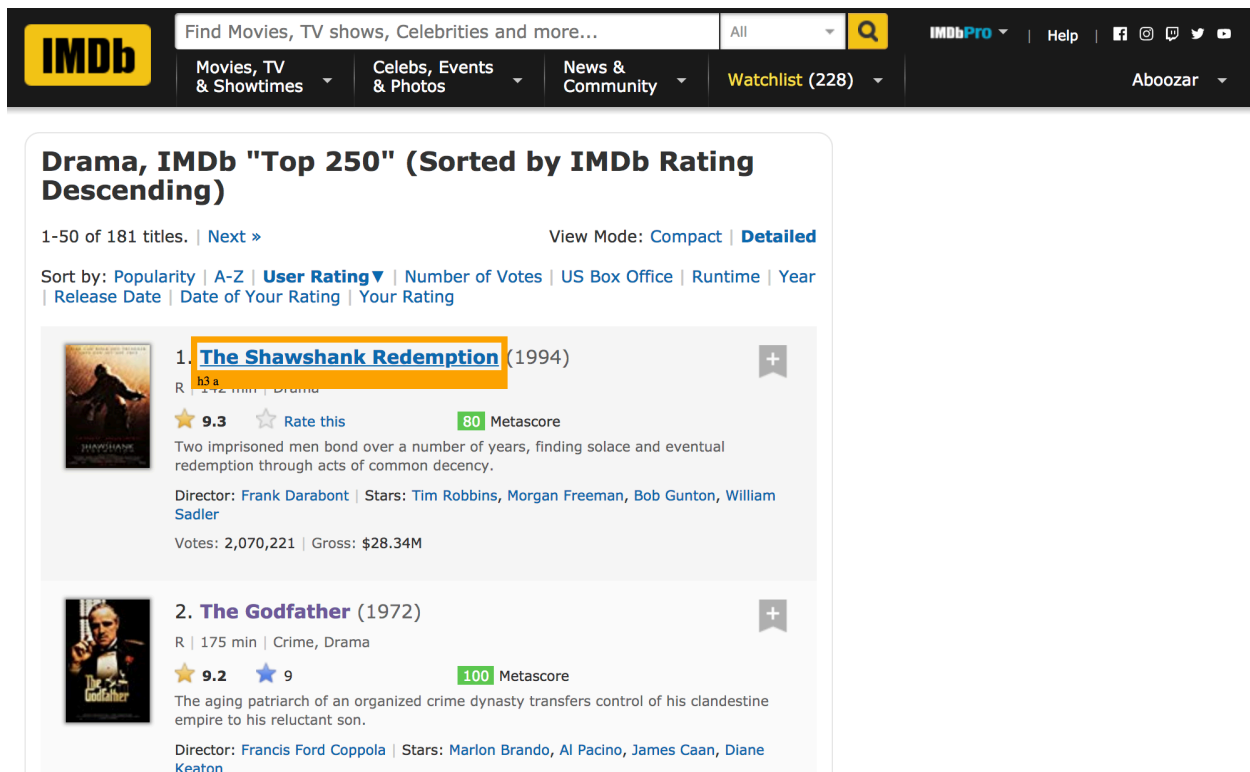


Figure 20: SelectorGadget

```
imdb <- read_html("https://www.imdb.com/search/title?genres=drama&groups=top_250&sort=user_rating")
```

The object `imdb` contains all the html content of the page in the form of an **xml** document that contains the *head* and the *body* of the page. Now, here is the important part. Let's say we would like to make a data frame that has 4 columns. The first column is the name of the movies, the second column is their ratings, the third column is their genres, and the last column is the year the movies are made. We want to make this data frame for the top 50 movies. For scraping different components of a webpage we need to know the CSS *selector* of the specific part of the html file they're scraping.

We are going to use a Chrome Extension called SelectorGadget. SelectorGadget is an open source tool that makes CSS selector generation and discovery on complicated sites a breeze. They should install the Chrome Extension. They can download SelectorGadget Chrome Extension here. They can learn more about the SelectorGadget in this video.

So how do we use the SelectorGadget to find the movie title? After installing the extension, ask your students to drag the bookmarklet to their bookmark bar, then go to any page and launch it. A box will open in the bottom right of the website. Ask them to click the title of the first movie on the list which is "The Shawshank Redemption". Without clicking, SelectorGadget will show an orange box around the element under which it says `h3 a`. This is the CSS selector we need. Now how do we use `rvest` to choose the selector and read the information with that CSS selector?

Your students will have to use the function `html_nodes()` in order to read information with that style. We will use the selector that we found using SelectorGadget and use the function `html_text("h3 a")` to convert everything into text. Here's what we need to do in order to read the titles of the movies.

```
titles <- imdb %>%
  html_nodes("h3 a") %>%
  html_text()
```



## titles

```
## [1] "The Shawshank Redemption"
## [2] "The Godfather"
## [3] "The Dark Knight"
## [4] "The Godfather: Part II"
## [5] "The Lord of the Rings: The Return of the King"
## [6] "Pulp Fiction"
## [7] "Schindler's List"
## [8] "12 Angry Men"
## [9] "Fight Club"
## [10] "The Lord of the Rings: The Fellowship of the Ring"
## [11] "Forrest Gump"
## [12] "The Lord of the Rings: The Two Towers"
## [13] "Goodfellas"
## [14] "One Flew Over the Cuckoo's Nest"
## [15] "Seven Samurai"
## [16] "Interstellar"
## [17] "City of God"
## [18] "Saving Private Ryan"
## [19] "The Green Mile"
## [20] "Life Is Beautiful"
## [21] "Se7en"
## [22] "Léon: The Professional"
## [23] "The Silence of the Lambs"
## [24] "It's a Wonderful Life"
## [25] "Whiplash"
## [26] "The Intouchables"
## [27] "The Prestige"
## [28] "The Departed"
## [29] "The Pianist"
## [30] "Gladiator"
## [31] "American History X"
## [32] "The Lion King"
## [33] "Cinema Paradiso"
## [34] "Grave of the Fireflies"
## [35] "Apocalypse Now"
## [36] "Casablanca"
## [37] "The Great Dictator"
## [38] "Modern Times"
## [39] "City Lights"
## [40] "Your Name."
## [41] "Dangal"
## [42] "Django Unchained"
## [43] "3 Idiots"
## [44] "Taare Zameen Par"
## [45] "Babam ve Oglum"
## [46] "The Lives of Others"
## [47] "Oldeuboi"
## [48] "American Beauty"
## [49] "Braveheart"
## [50] "Once Upon a Time in America"
## [51] " "
## [52] " "
```

```
## [53] " "
## [54] " "
## [55] " "
```

The object `titles` now contains the name of the movies. It should have 50 elements but it has 55. If they check the object they will realize that the last 5 elements are empty. So we should get rid of them. This is what we should do.

```
titles <- titles[titles != " "]
```

Note that the object is a vector. We will create separate vectors for ratings, genres, and years and then will at the end create a data frame that contains all the 4 vectors. Now, we will use the SelectorGadget to find the CSS selector for years. The selector for years is `h3 span`. So similar to

```
years <- imdb %>%
  html_nodes("h3 span") %>%
  html_text()
years
```

```
## [1] "1." "(1994)" "2." "(1972)" "3." "(2008)" "4."
## [8] "(1974)" "5." "(2003)" "6." "(1994)" "7." "(1993)"
## [15] "8." "(1957)" "9." "(1999)" "10." "(2001)" "11."
## [22] "(1994)" "12." "(2002)" "13." "(1990)" "14." "(1975)"
## [29] "15." "(1954)" "16." "(2014)" "17." "(2002)" "18."
## [36] "(1998)" "19." "(1999)" "20." "(1997)" "21." "(1995)"
## [43] "22." "(1994)" "23." "(1991)" "24." "(1946)" "25."
## [50] "(2014)" "26." "(2011)" "27." "(2006)" "28." "(2006)"
## [57] "29." "(2002)" "30." "(2000)" "31." "(1998)" "32."
## [64] "(1994)" "33." "(1988)" "34." "(1988)" "35." "(1979)"
## [71] "36." "(1942)" "37." "(1940)" "38." "(1936)" "39."
## [78] "(1931)" "40." "(2016)" "41." "(2016)" "42." "(2012)"
## [85] "43." "(2009)" "44." "(2007)" "45." "(2005)" "46."
## [92] "(2006)" "47." "(2003)" "48." "(1999)" "49." "(1995)"
## [99] "50." "(1984)"
```

If they look at the object `years`, they will notice the peculiar content. The years are in parentheses and there are some other numbers as well. Then their next task is to write code that will only keep inside the parentheses. This is a bit tricky so feel free to give the code below. The function `str_match_all()` and the regex (regular expression) inside it will return everything inside the parentheses but since the output is in the form of a list, they can use the function `unlist()` to convert the output to a vector. Finally to convert the vector to numbers (since the output is the years movies are made and we want them to be numbers) we use the function `as.numeric()` as they have used many times.

```
years <- years %>%
  str_match_all("(?<=\\(\\.+(?=\\))") %>%
  unlist() %>%
  as.numeric()
```

Object `years` should now look like a clean vector of the years movies are made.

```
## [1] 1994 1972 2008 1974 2003 1994 1993 1957 1999 2001 1994 2002 1990 1975
## [15] 1954 2014 2002 1998 1999 1997 1995 1994 1991 1946 2014 2011 2006 2006
## [29] 2002 2000 1998 1994 1988 1988 1979 1942 1940 1936 1931 2016 2016 2012
## [43] 2009 2007 2005 2006 2003 1999 1995 1984
```

The following code will scrape genres and ratings from the website.

```
genre = imdb %>%
  html_nodes(".genre") %>%
```

```

html_text() %>%
# getting rid of the "\n" in the strings
str_replace_all("[\r\n]" , "") %>%
# getting rid of the extra spaces in both ends of the strings
str_trim(side = c("both"))

ratings = imdb %>%
  html_nodes(".mode-advanced strong , .runtime , .genre") %>%
  html_text() %>%
  as.numeric()

```

```
## Warning in function_list[[k]](value): NAs introduced by coercion
```

```

# There are some empty elements so we get rid of them here
ratings <- ratings[!is.na(ratings)]

```

Now that all the data is downloaded, ask your students to combine all the vectors in a data frame called `imdb_df`. They should know to use the `data.frame()` function in order to do that.

```

imdb_df = data.frame(titles, years, genre, ratings, stringsAsFactors = FALSE)
head(imdb_df)

```

```
##               titles years
## 1      The Shawshank Redemption 1994
## 2              The Godfather 1972
## 3              The Dark Knight 2008
## 4      The Godfather: Part II 1974
## 5 The Lord of the Rings: The Return of the King 2003
## 6              Pulp Fiction 1994
##           genre ratings
## 1           Drama    9.3
## 2      Crime, Drama    9.2
## 3 Action, Crime, Drama    9.0
## 4      Crime, Drama    9.0
## 5 Adventure, Drama, Fantasy    8.9
## 6      Crime, Drama    8.9
```

They can continue doing this for the rest of the top 250 movies.

## Exercise 3: Scraping NBA data from ESPN.com

Level: Difficult

In this exercise, your students will work on another scraping exercise. This time NBA standings from the [espn.com](http://www.espn.com) website. The link to the webpage we want to scrape is [here](http://www.espn.com/nba/standings). Note that by the time you will see this exercise, the content of the website might have changed so be aware of that. The changes on the website will likely change the html nodes we need to use. This is the table we would like to scrape the table below:

Again, we're gonna use the package **rvest** for this exercise. If you want to familiarize your students with the basic commands, look at the previous exercise. First, let's use the `read_html()` function to scrape the entire webpage.

```

library(rvest)
library(tidyverse)
library(stringr)
nba = read_html("http://www.espn.com/nba/standings")

```

## NBA Standings - 2018-19

[Resources](#)

Standings

Expanded

Vs. Division

League
















Conference

Division

2018-19

Regular Season

### Eastern Conference

		<u>W</u>	<u>L</u>	<u>PCT</u>	<u>GB</u> ^	HOME	AWAY	DIV	CONF	<u>PPG</u>	<u>OPP PPG</u>	<u>DIFF</u>	<u>STRK</u>	L10
1	z -- 	59	20	.747	-	32-6	27-14	14-2	39-11	118.1	108.8	+9.3	W2	7-3
2	y -- 	56	23	.709	3	31-9	25-14	12-4	35-15	114.4	108.4	+6.0	W5	7-3
3	x -- 	49	30	.620	10	30-10	19-20	8-8	29-20	115.3	112.6	+2.6	L3	5-5
4	x -- 	47	32	.595	12	28-12	19-20	10-6	33-16	112.4	108.0	+4.4	W2	5-5
5	x -- 	47	32	.595	12	29-10	18-22	11-5	32-17	108.0	104.1	+3.8	W2	3-7
6		39	39	.500	19.5	25-14	14-25	8-8	26-24	107.2	107.6	-0.3	L2	4-6
7		39	40	.494	20	22-18	17-22	8-8	26-23	112.0	112.6	-0.5	L2	3-7
8		39	40	.494	20	24-16	15-24	8-6	27-22	106.5	106.4	+0.1	W1	8-2
		38	40	.487	20.5	18-22	20-18	7-9	22-27	105.6	105.7	-0.2	L2	6-4
		36	42	.462	22.5	24-15	12-27	10-5	26-22	110.6	112.2	-1.6	W1	5-5
e --		32	47	.405	27	22-17	10-30	7-9	19-31	114.2	116.8	-2.7	L1	3-7
e --		29	50	.367	30	17-23	12-27	6-9	16-33	113.2	118.8	-5.7	W1	5-5
e --		22	57	.278	37	9-30	13-27	3-13	16-33	105.2	113.4	-8.2	W1	3-7
e --		19	60	.241	40	13-26	6-34	4-12	15-36	104.6	113.9	-9.3	L7	2-8
e --		15	63	.192	43.5	8-31	7-32	2-14	9-40	104.9	114.1	-9.2	L1	2-8

### Western Conference

Figure 21: NBA standings on ESPN.com

Now, we need to know the css selector for each column. Again, let's use the **SelectorGadget** extension on the Chrome browser to find the CSS selectors. More information on the **SelectorGadget** extension can be found in the previous exercise.

```
## first we will create an empty list
column <- list()
for(i in 1:13){
  node = paste(".Table2__td:nth-child(",i,") .stat-cell", sep="")
  a <- nba %>%
    html_nodes(node) %>%
    html_text()
  column[[i]] <- a
}
names(column) <- c("W", "L", "PCT", "GB", "HOME", "AWAY", "DIV", "CONF", "PPG", "OPP PPG", "DIFF", "STR")

df <- as.data.frame(column, stringsAsFactors = FALSE)

df <- data.frame(column,stringsAsFactors=FALSE)
```



# Course 9: Data Analysis





# Course 10: Written and Oral Communication in Data Science



# Course 11: Getting a Job in Data Science

a project on reading indeed jobs in data science



# Course 11: Random Stuff

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation  $a^2 + b^2 = c^2$ .

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")  
# or the development version  
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading #.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): <https://yihui.name/tinytex/>.

You can label chapter and section titles using {#label} after them, e.g., we can reference Chapter . If you do not manually label them, there will be automatic labels anyway, e.g., Chapter ??.

## Classroom activity 1

Figures and tables with captions will be placed in **figure** and **table** environments, respectively.

```
par(mar = c(4, 4, .1, .1))  
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the **fig:** prefix, e.g., see Figure 22. Similarly, you can reference tables generated from **knitr::kable()**, e.g., see Table 2.

```
knitr::kable(  
  head(iris, 20), caption = 'Here is a nice table!',  
  booktabs = TRUE  
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

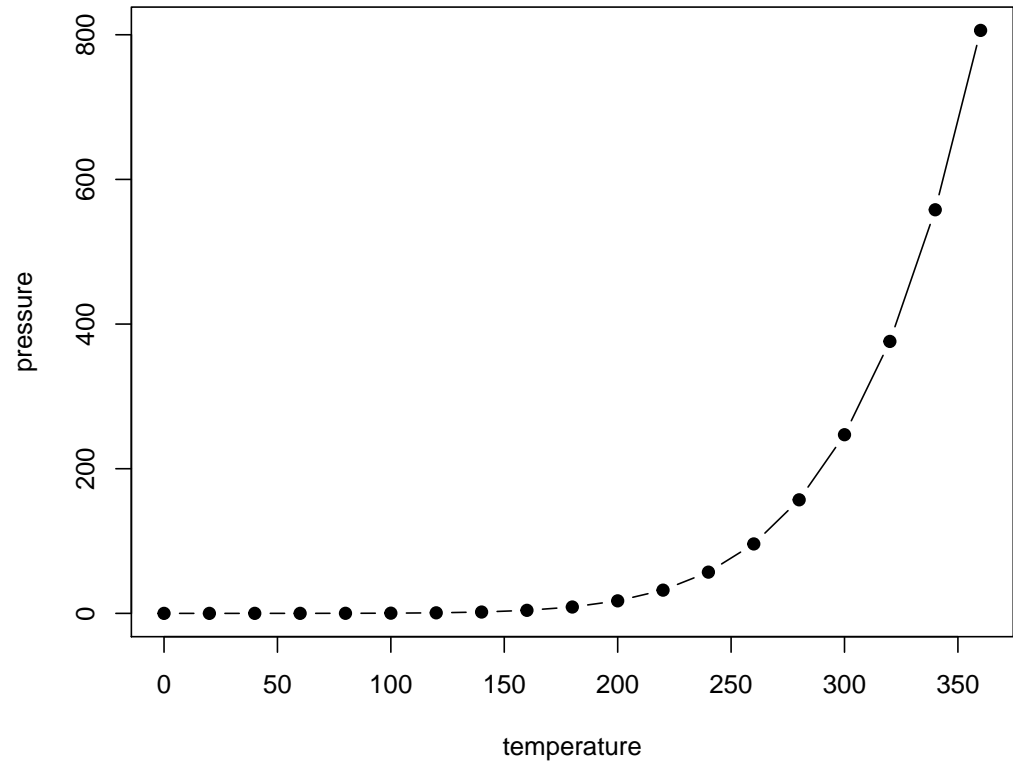


Figure 22: Here is a nice figure!

Table 2: Here is a nice table!				
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

# Bibliography

- Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.
- Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.