Instructor Resources for Chromebook Data Science

Aboozar Hadavand

February 2019

Contents

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 Chromebook Data Science MOOCs as part of their curriculum for teaching data science. Chromebook Data Science is a result of a team of data scientists and biostatisticians at Johns Hopkins University School of Public Health.

Chromebook 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 Chromebook 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
Introduction to Chromebook Data Science	This is the first class in the Chromebook 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
How to Use A Chromebook	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
Google and the Cloud	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
Organizing Data Science Projects	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
Version Control	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
Introduction to R	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
Data Tidying	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
Data Visualization	This course will cover the different types of visualation 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
Getting Data	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
Data Analysis	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
Written and Oral Communication in Data Science	This course will discuss better practices for oral and written communication in data science.	Course 10
Getting a Job in Data Science	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 0: 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.

Course 1: How to Use a Chromebook

Instructors can skip this course if students are not using Google Chromebooks. If your studnets 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 usefule 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 screenshop on their computers and share that screetshot 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 aks you questions on Loom but can show you step by step how thye 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 mak
```

- 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 students 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.
- 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

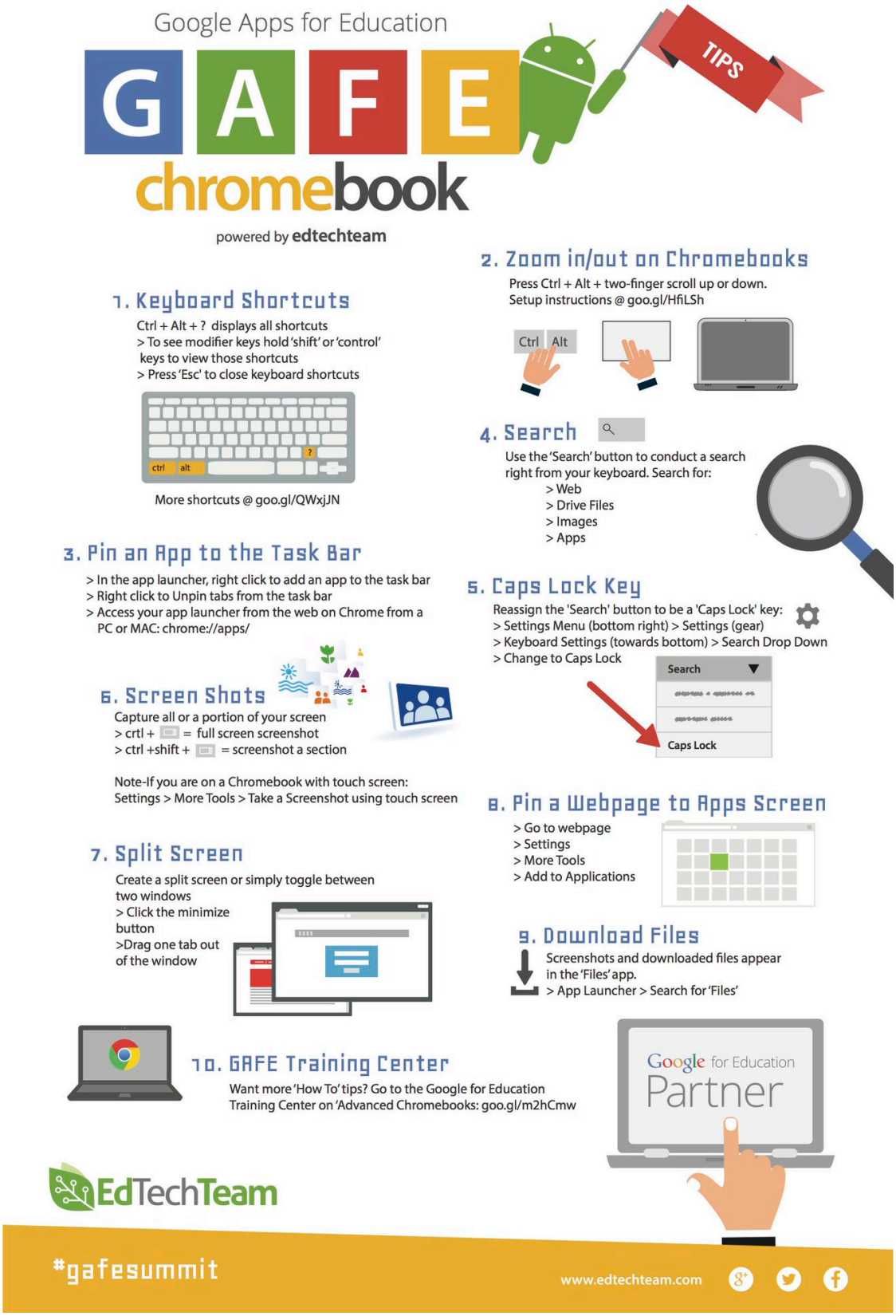


Figure 1: Chromebook cheatsheet

with that as well.

- While we mentoion 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 discuss 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 sunc on in Chrome. This is how.

Course 2: Google and the Cloud

Editing a text file offline

Each learner creates a google doc and invites another learner to the document to work collaboratively (practice different levels of access such as view/comment/edit)

Create a Google calendar event (called "TEST EVENT") and add others to it

Subscribing to a calendar (subscription to CBDS+ calendar)

How much space do you have left on your google drive?

Delete a file on Google Drive? Let's see how we can find it

Use a resume template on Google Docs

Create a google sheets document (we're gonna use it later), add 3 columns (artists, song, rating). List your top 10-30 favorite songs and decide how you rate the song on a scale of 1-5

Assignment: For the next session, make some slides about who you are/what you do/what are your hobbies/etc. You can present it to us next session.

Course 3: Organizing Data Science Projects

Go to this link, https://docs.google.com/document/d/1vFQzQ6sRsMJyrqLY49qhEkxvQ0G1Zp_lMZjltpeLvUw/edit?usp=sharing, add your name to the list, and try to make an Rmd file in Rstudio.cloud that when you knit looks similar to this google doc. Download the Google Sheet that you created last week as a csv file On R Studio Cloud, create a new project and call it my_music_project. Create all the folders necessary for a data science project. Go to the data folder and upload the csv file that you downloaded on your chromebook Create an RMarkdown document with the title Scope of the Project as the headline and a list that says objective 1, objective 2, objective 3. Make sure you save the document

Find the album cover for one of the songs on your list (find the link to the cover on the web) and add the image to your Rmd document. Preview your Rmarkdown document

Course 4: Version Control

Create an empty repo and name it anything you want. Go through the repo settings Delete the repository (warning about what deleting means) Create a new repo called "my_music_analysis" (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 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?

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

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. 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/RtmpBgxDis/devtools11d1c3049467f/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?
## Norma Lee.
##  Norma Lee who?
## Normally I ring the doorbell!
```

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

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.

str_view(names, "^M") str_view(names, "M$")

str_detect() str_count() str_subset()

Logicals cbds_names <- c("Andre", ... "Ashley", "Aboozar") startsWith(cbds_names, "An") cbds_age <- c(18, ..., 36) any(cbds_ages >= 30) all(cbds_ages >= 17)

Go to this IMDB link: https://www.imdb.com/search/title?genres=drama&groups=top_250&sort=user_rating,desc In this exercise we are going to create a data frame that has four columns: movie name, year, rating, genre. Create repo in github called imdb_movie_analysis and clone it to a new project on RSC. Create all the folder structure and within your code folder, create a script file called exploratory analysis. Create a data frame with these columns and enter the data from the top 10 movies. Call your data frame top_ten_movies. You can choose your own column names. Call the column that contains the information about the movie title Going back to your imdb data, What is the class of the column that contains the movie names? What's the class of the column with the year the movie was made? What's the difference between the maximum and minimum rating among the top 10 movies? What is the dimension of your data frame? Change the column names to this: movie_title, movie_year, movie_rating, movie_genre View the first 5 rows of the data frame Install the package skimr and using the function skim() look at the summary of the data Without checking manually, see if the movie "Groundhog Day" is among the top 10 movies. Count the number of movies that are made after your birth year. How many of the movies are in the genre Action. Create a list with three elements. First element contains your top 5 most favorite movies (as characters). The second element contains the number of states you have traveled to (as.integer). Note that each element has a different size and different class and this is the beauty of lists in R. Call the first element of the list (your top 5 movies) Replace the first row of the column movie_rating with NA. Now, what is the maximum rating in your data? Or mean?

Exercise 4: Reverse a String Function

Level: Difficult

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 functino.

```
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 5: Other Short Exercises

Level: Easy to moderate

Write a function that takes user's name and returns "Hello, Name!"

```
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 = ""))
    }
    }
}
```

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 a function that takes two numbers as input and checks whether the first one is bigger than the second one.

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

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"

Exercise 1: Tidying NBA Finals Data

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 '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 inpired by two kernells on Kaggle that can be found here and here. Ask your stduent 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 the 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_integer(),
##   Team = col_character(),
##   FGP = col_double(),
##   TPP = col_double(),
##   FTP = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
runners <- read_csv(file = "./data/nba/runnerupsdata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   Team = col_character(),
##   FGP = col_double(),
##   TPP = col_double(),
##   FTP = col_double()
## )
## 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 Warriorrs
```

Once your students run the code above, they will notice some peculiar values. For instance `'Heat'` and `Warriorrs`. 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 hte `iflese()` function. The line `mutate(Team = ifelse(Team == "Warriorrs","Warriors",Team))` creates a column called `Team` which is the same as the old column `Team` except whenever the value in the column is `Warriorrs` 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 == "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 == "Warriorrs","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 too bind the two dataframes together. Ask your students to find the appropritate **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 studnets how they would calculate their age based on the two objects. They probably say they'd subtract `mybdat` 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 7478 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] 20.48767
```

Now that your studetns 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] 36.86301
```

Tell your students to add a warning message that is shown to user warning them to enter date as "yyyy-md-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] 36.86301
```

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` argumenst 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] 322920
```

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

```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 36.9 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 inpired by two kernells on Kaggle that can be found here and here. Ask your stduent 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 the 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_integer(),
##   Team = col_character(),
##   FGP = col_double(),
##   TPP = col_double(),
##   FTP = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
runners <- read_csv(file = "./data/nba/runnerupsdata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   Team = col_character(),
##   FGP = col_double(),
##   TPP = col_double(),
##   FTP = col_double()
```

```
## )
## 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 == "Warriorrs","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

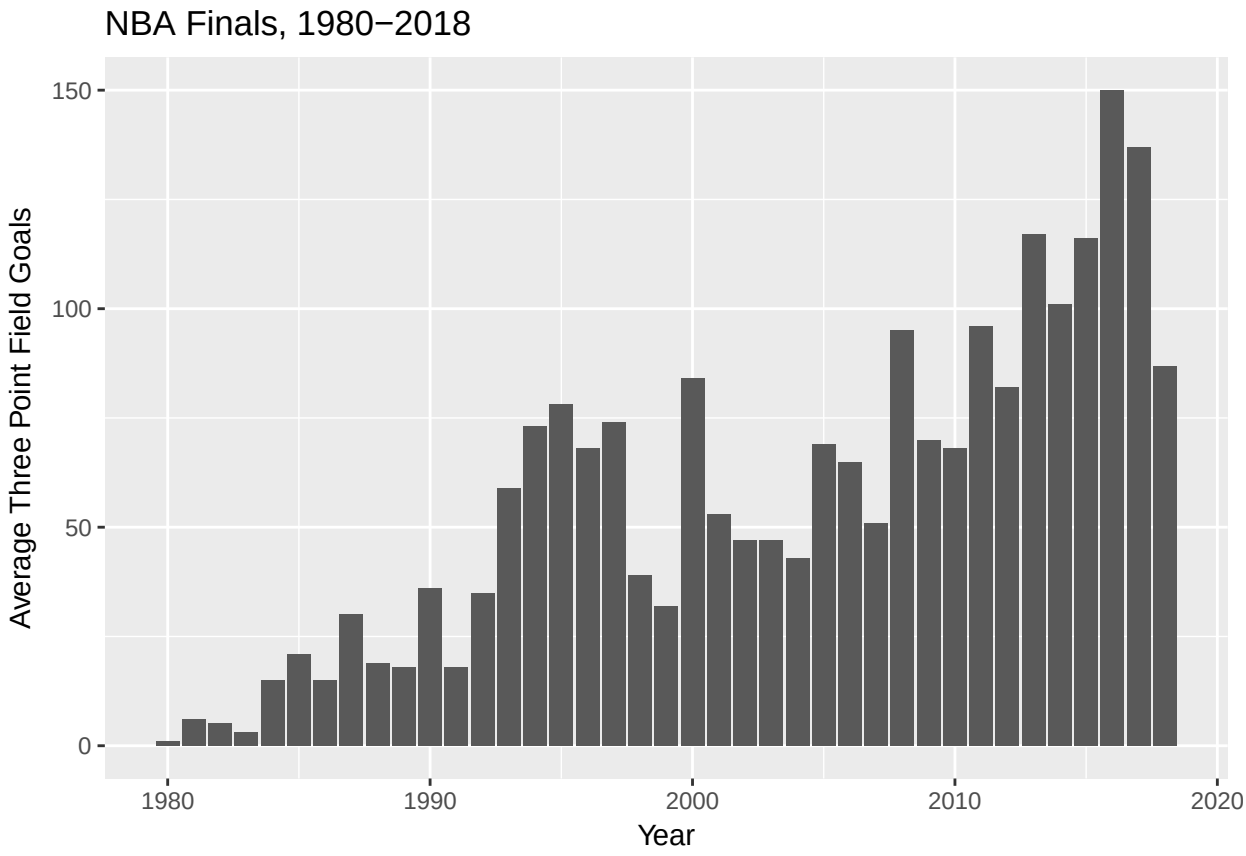
98.21101

Home

103.31532

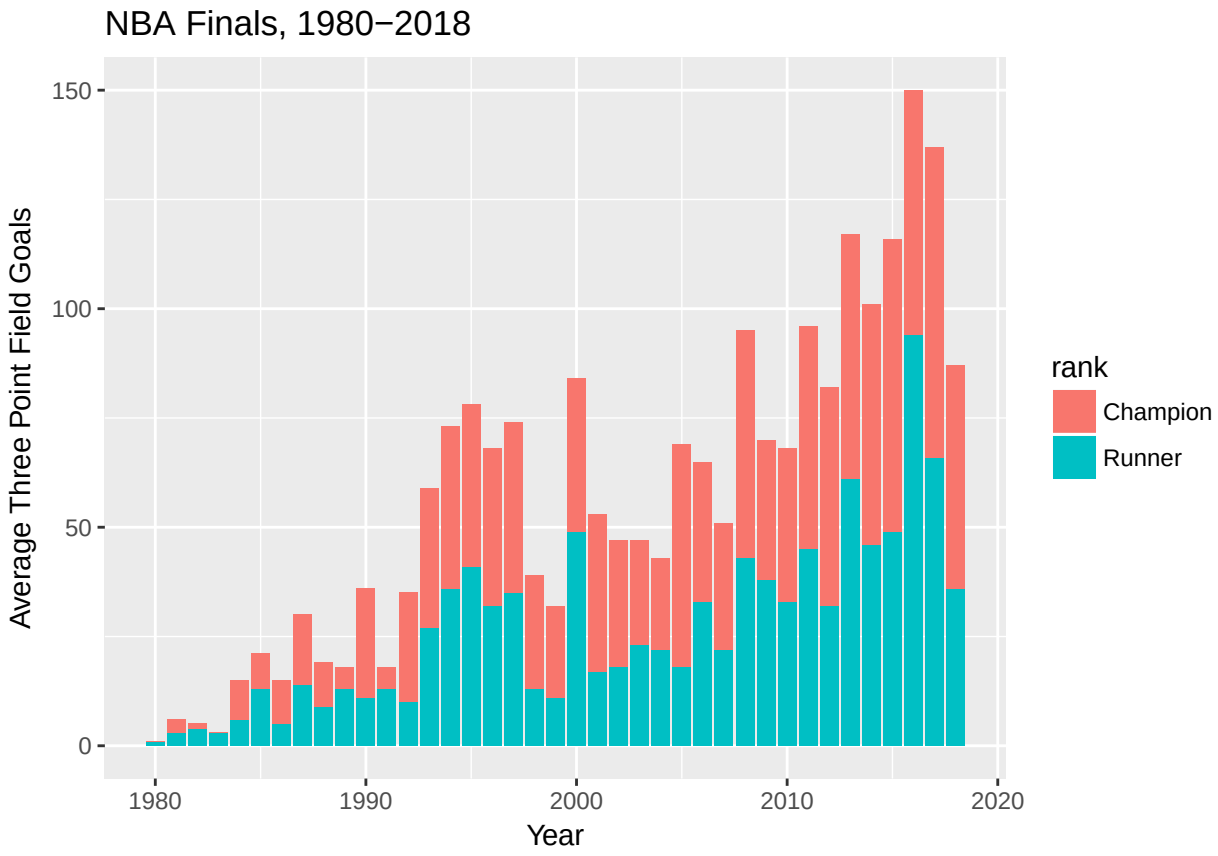
Now, we are going to do some data visualization using the package **ggplot2**. Ask them to first find the column that contains the three point field goals from the data guide. The column is `TP`. Then ask them to show the average number of three points per year using a bar plot. Note that they will have 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')
```



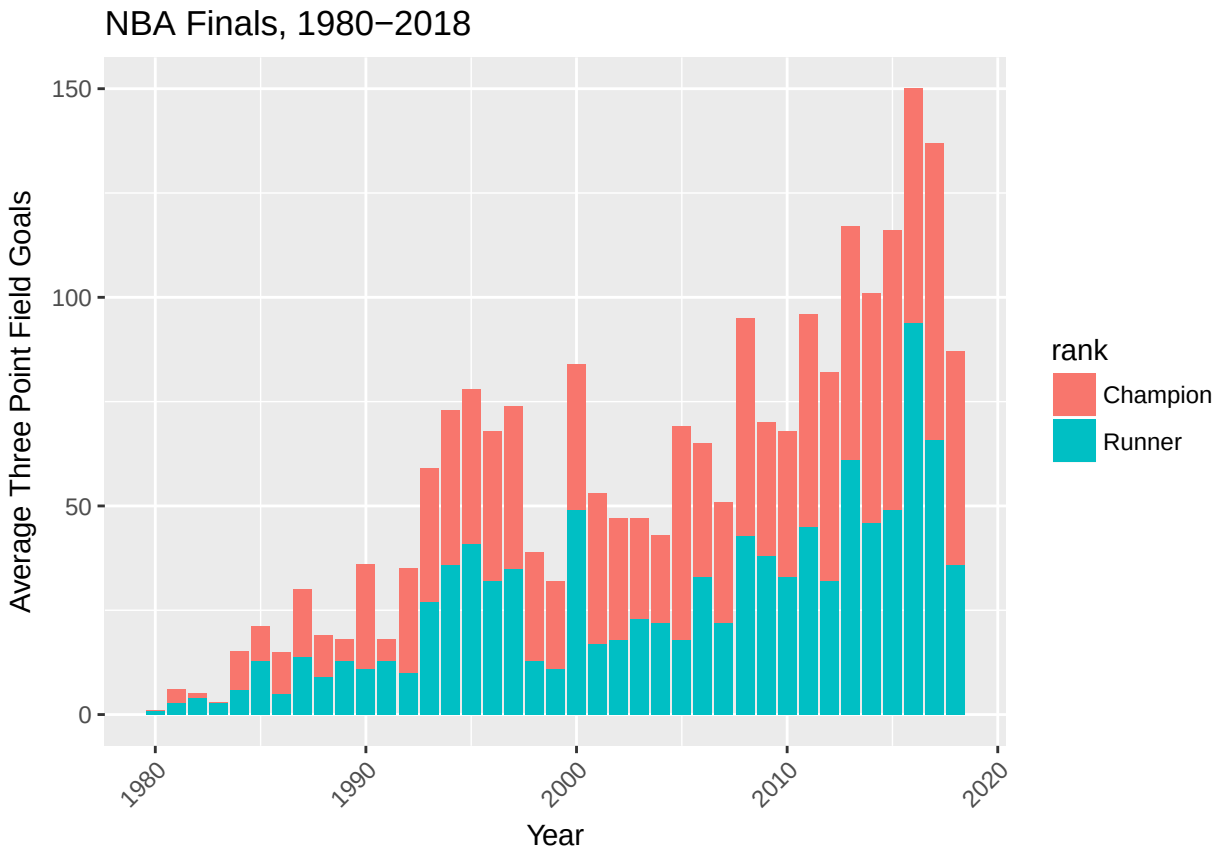
Now, ask them two repeat the pervious graph but this time, have the bar plot (with different colors) for champion and runner up teams. In other words, they will have to use the argument `fill` and make it equal to the newly created column `rank`. Then ask them to interpret the graph. What is obvious 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 axix 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')
```

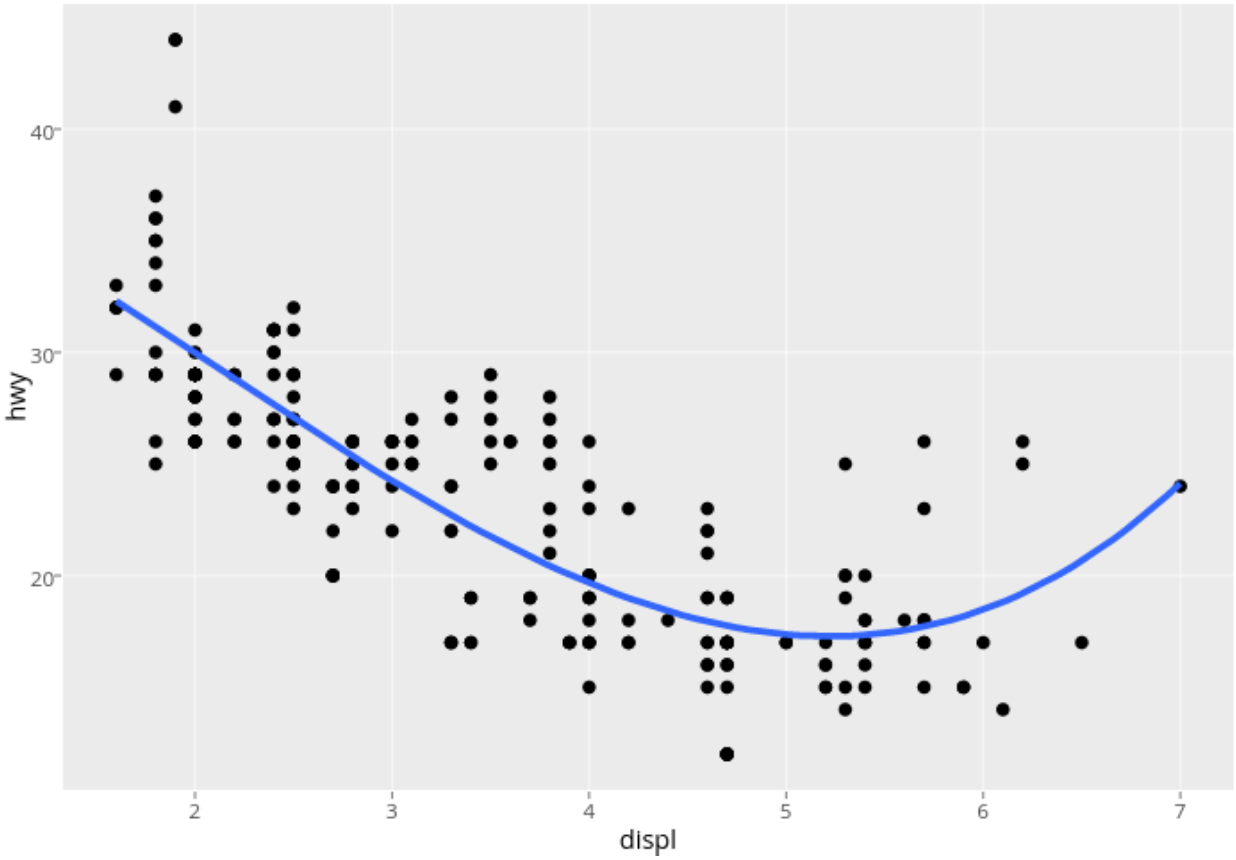
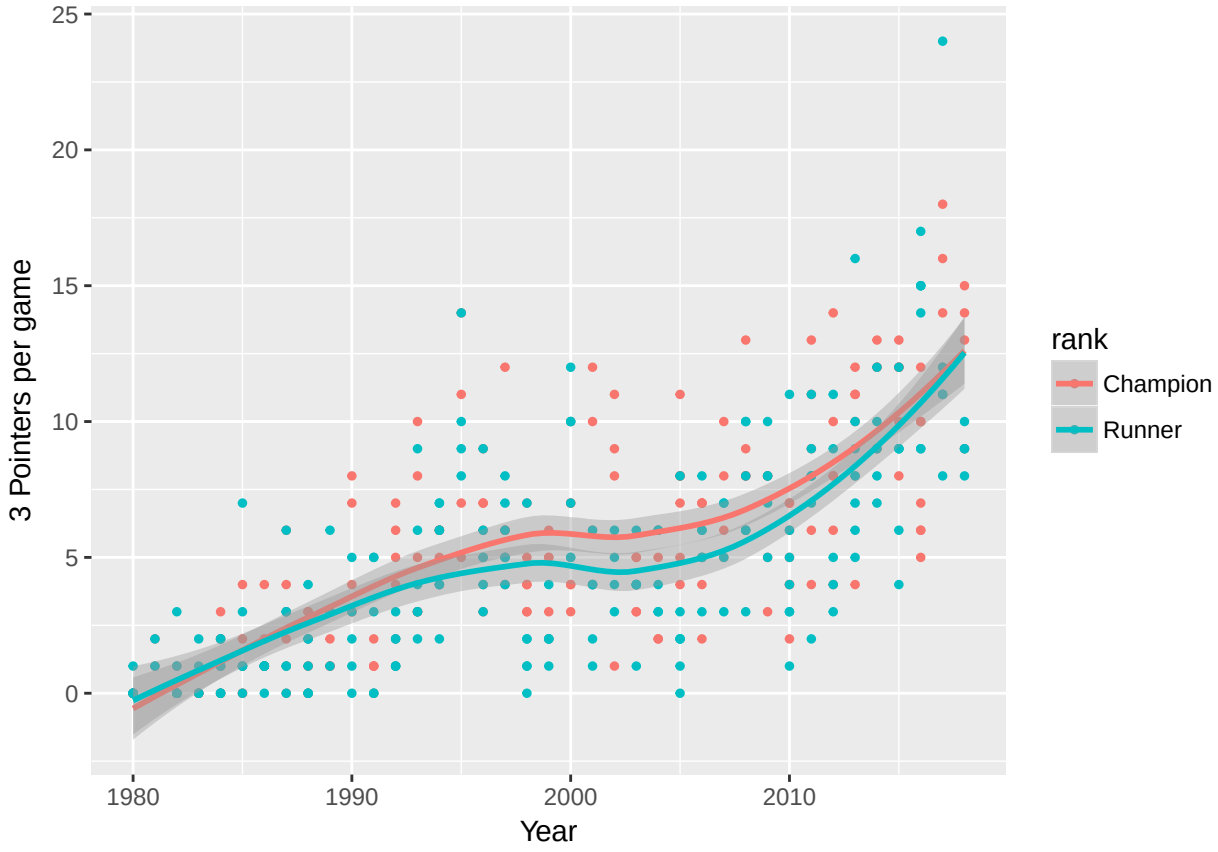


Figure 2: `geom_smooth()` function, linear approximation


25
20
15
10
5
0
3 Pointers per game
1980
1990
2000
2010
Year
rank
Champion
Runner

Course 8: Getting Data

Course 9: Data Analysis

Course 10: Written and Oral Communication in Data Science

Course 11: Getting a Job 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 3. 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).

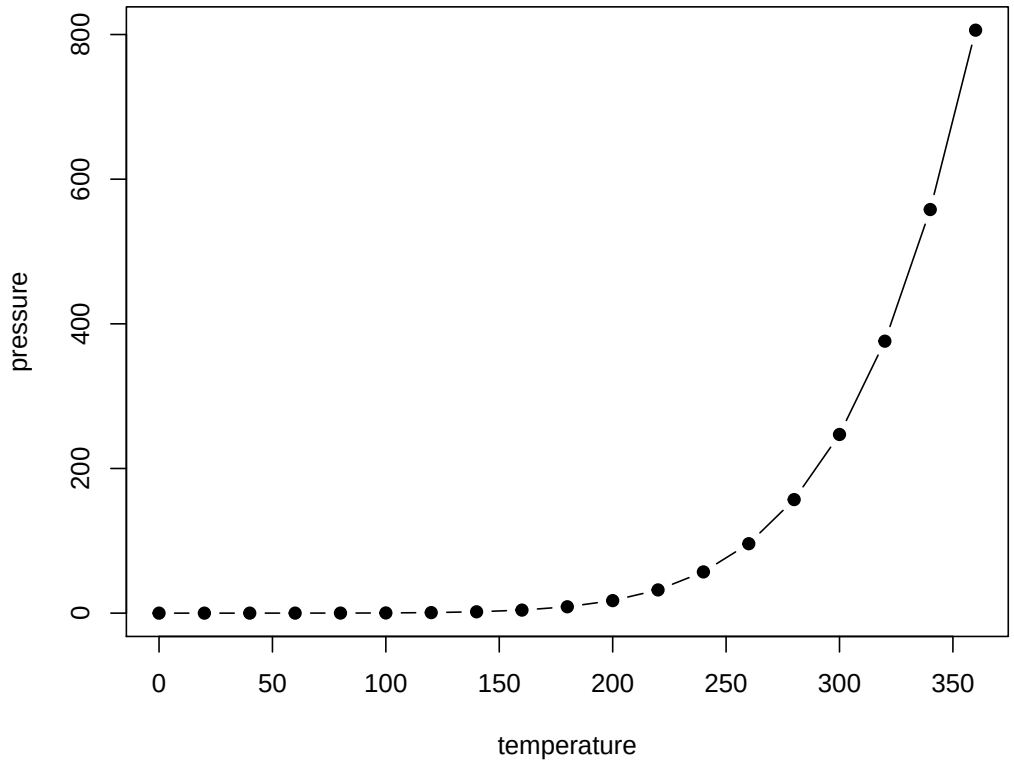


Figure 3: Here is a nice figure!

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.