Stat 220 Introduction to Data Science

Deepak Bastola

2023-05-21

Contents

Course overview

Greetings and welcome to Introduction to Data Science! In this course, we will delve into the computational aspects of data analysis, covering topics such as data acquisition, management, and visualization tools. Throughout this course, we will emphasize the principles of data-scientific, reproducible research and dynamic programming, utilizing the R/RStudio ecosystem.

If you have taken Stat 120, 230, or 250 at Carleton, you will find yourself well-equipped to handle the material. However, it is important to refresh your R and R-markdown skills before the start of the class. Specifically, I expect all students to be able to load a data set into R, calculate basic summary statistics, and perform basic exploratory data analysis. In the first week of class, we will delve into Git and GitHub version control, though prior exposure to these topics is not necessary.

0.1 Learning Objectives

- Develop research questions that can be answered by data. Import/scrape data into R and reshape it to the form necessary for analysis.
- Manipulate common types of data, including numeric, categorical (factors), text, date-times, geo-location variables in order to provide insight into your data and facilitate analysis.
- Explore data using both graphical and numeric methods to provide insight and uncover relationships/patterns.
- Utilize fundamental programming concepts such as iteration, conditional execution, and functions to streamline your code.
- Build, tune, use, and evaluate basic statistical learning models to uncover clusters and classify observations.
- Draw informed conclusions from your data and communicate your findings using both written and interactive platforms.

0.2 Course Requirements

Set-up Instructions

Chapter 1

What is R, RStudio, and RMarkdown?

R is a free and open source statistical programming language that facilitates statistical computation. There are a myriad of application that can be done in R, thanks to a huge online support community and dedicated packages. However, R has no graphical user interface and it has to be run by typing commands into a text interface.

1.1 What is RStudio?

RStudio provides graphical interface to R! You can think of RStudio as a graphical front-end to R that that provides extra functionality. The use of the R programming language with the RStudio interface is an essential component of this course.

1.2 R Studio Server

The quickest way to get started is to go to https://maize2.mathcs.carleton.edu, which opens an R Studio window in your web browser. Once logged in, I recommend that you do the following:

- Step 1: Create a folder for this course where you can save all of your work. In the Files window, click on New Folder.
- Step 2: Click on Tools -> Global Options -> R Markdown. Then uncheck the box that says "Show output inline…"

(It is also possible to download RStudio on your own laptop. Instructions may be found at the end of this document.)

1.3 R/RStudio

The use of the R programming language with the RStudio interface is an essential component of this course. You have two options for using RStudio:

- The **server version** of RStudio on the web at (https://maize2.mathcs.carleton.edu). The advantage of using the server version is that all of your work will be stored in the cloud, where it is automatically saved and backed up. This means that you can access your work from any computer on campus using a web browser. This server may run slow during peak days/hours. I also recommend you to download a local version of R server in your computer in case of rare outages.
- A **local version** of RStudio installed on your machine. This option is highly recommended due to the computational resources this course demands. Using this version you can only store your files in your local machine. Additionally, we can save our work on GitHub. We will learn how to use GitHub in the beginning of the course. Both R and RStudio are free and open-source. Please make sure that you have recently updated both R and RStudio.

1.4 Installing R/RStudio (not needed if you are using the maize server)

> Download the latest version of R: https://cran.r-project.org/ Download the free Rstudio desktop version: https://www.rstudio.com/products/rstudio/download/

Use the default download and install options for each. For R, download the "precompiled binary" distribution rather than the source code

Updating R/RStudio (not needed if you are using the maize server)

If you have used a local version of R/RStudio before and it is still installed on your machine, then you should make sure that you have the most recent versions of each program.

- To check your version of R, run the command `getRversion()` and compare your version to the newest version posted on https://cran.r-project.org/. If you need an update, then install the newer version using the installation directions above.

- In RStudio, check for updates with the menu option `Help > Check for updates`. Follow directions if an update is needed.

** Did it work? (A sanity check after your install/update) **

Do whatever is appropriate for your operating system to launch RStudio. You should get a window similar to the screenshot you see here, but yours will be more boring because you haven't written any code or made any figures yet!

Put your cursor in the pane labeled *Console*, which is where you interact with the live R process. Create a simple object with code like `x <- 2 * 4` (followed by enter or return). Then inspect the `x` object by typing `x` followed by enter or return. You should see the value `8` printed. If this happened, you've succeeded in installing R and RStudio!

1.5 What is RMarkdown?

An R Markdown file (.Rmd file) combines R commands and written analyses, which are 'knit' together into an HTML, PDF, or Microsoft Word document.

An R Markdown file contains three essential elements:

- Header: The header (top) of the file contains information like the document title, author, date and your preferred output format (pdf_document, word_document, or html_document).
- Written analysis: You write up your analysis after the header and embed R code where needed. The online help below shows ways to add formatting details like bold words, lists, section labels, etc to your final pdf/word/html document. For example, adding ** before and after a word will bold that word in your compiled document.
- R chunks: R chunks contain the R commands that you want evaluated. You embed these chunks within your written analysis and they are evaluated when you compile the document.

1.6 Install LaTeX (for knitting R Markdown documents to PDF):

You need a Latex compiler to create a pdf document from a R Markdown file. If you use the maize server, you don't need to install anything. If you are using a local RStudio, you should install a Latex compiler. Below are the recommended installers for Windows and Mac:

- MacTeX for Mac (3.2GB)
- MiKTeX for Windows (190MB)
- Alternatively, you can install the `tinytex` R package by running `install.packages("tinytex")` in the console.

1.7 Updating R/RStudio (not needed if you are using the maize2 server)

If you have used a local version of R/RStudio before and it is still installed on your machine, then you should make sure that you have the most recent versions of each program.

- To check your version of R, run the command `getRversion()` and compare your version to the newest version posted on https://cran.r-project.org/. If you need an update, then install the newer version using the installation directions above.
- In RStudio, check for updates with the menu option `Help > Check for updates`. Follow directions if an update is needed.

1.8 Opening a new file

If using Rstudio on your computer, using the **File>Open File** menu to find and open this .Rmd file.

If using Maize Rstudio from your browser:

- In the Files tab, select **Upload** and **Choose File** to find the .Rmd that you downloaded. Click *OK* to upload to your course folder/location in the maize server account.
- Click on the .Rmd file in the appropriate folder to open the file.

1.9 Running codes and knitting .Rmd files:

- You can run a line of code by placing your cursor in the line of code and clicking **Run Selected Line(s)**
- You can run an entire chunk by clicking the green triangle on the right side of the code chunk.

- After each small edit or code addition, **Knit** your Markdown. If you wait until the end to Knit, it will be harder to find errors in your work.
- Format output type: You can use any of pdf_document, html_document type, or word_document type.
- **Maize users**: You may also need to allow for "pop-up" in your web browser when knitting documents.

1.10 Few More Instructions

The default setting in Rstudio when you are running chunks is that the "output" (numbers, graphs) are shown **inline** within the Markdown Rmd. If you prefer to have your plots appear on the right of the console and not below the chunk, then change the settings as follows:

1. Select Tools > Global Options.
2. Click the R Markdown section and uncheck (if needed) the option Show output inline for all R Markdown documents.
3. Click OK.

Now try running R chunks in the .Rmd file to see the difference. You can recheck this box if you prefer the default setting.

1.11 VPN

If you plan to do any work off campus this term, you need to install Carleton's VPN. This will allow you to access the **maize** server (if needed).

Installing the GlobalProtect VPN

Follow the directions here to install VPN.

Chapter 2

Assignments in Stat 220

2.1 Do's and Don't of collaboration for individual assignments

- You *can* discuss homework problems with classmates but you must write up **your own** homework solutions and **do your own work in R (no sharing commands or output)**.
  - **Do not share R commands/code in any way**, including, but not limited to, sending commands via email, slack, text, or showing commands in a shared screen with the intention of showing a classmate your solution to a problem.
  - You **can** share a screen to help troubleshoot a coding problem in R.
- You *can* use the following resources to complete your homework:
  - Carleton faculty (myself, other math or statistics faculty, etc)
  - discussions with classmates (see above) or knowledgeable friends
  - Carleton resources like stats lab assistants
  - student solutions provided in the back of your student textbook or in the student solution manual.
- You *cannot* use any resources other than the ones listed above to complete assignments (homework, reports, etc) for this class. (e.g. you cannot use a friend's old assignments or reports, answers found on the internet, textbook (instructor) solutions manual, etc.)

2.1.1 Examples that violate the academic integrity policy

- sending your .Rmd homework file to another person in the class

- receiving an .Rmd homework file from another person
- sharing a screen and copying code, verbatim, from another person
- sending/receiving R commands
- neglecting to acknowledge classmates with whom you worked with on an assignment

2.2 Format and Content

Submit via GitHub (for most assignments) an organized and correctly ordered assignment.

- Content: Good data scientists need to do more than just write code; they should be able to interpret and explain their analyzes.
  - Provide a **written answer** first, followed by any required R code and output.
  - Use **complete sentences** when answering any problem that requires an explanation or overall problem summary.
- When including code:
  - Be sure to show the natural sequence of work needed to answer the problem.
  - Include brief comments explain your code steps.
  - Do not include typos or unnecessary commands/output.
  - Always include code output.
- At the top of each individual assignment **include the names of classmates that you worked with** on all or part of the assignment (but each person must write up their assignment on their own)

Disability Accommodations: Carleton College is committed to providing equitable access to learning opportunities for all students. The Disability Services office (Henry House, 107 Union Street) is the campus office that collaborates with students who have disabilities to provide and/or arrange reasonable accommodations. If you have, or think you may have, a disability (e.g., mental health, attentional, learning, autism spectrum disorders, chronic health, traumatic brain injury and concussions, vision, hearing, mobility, or speech impairments), please contact disability@carleton.edu or call Sam Thayer ('10), Accessibility Specialist (x4464) or Chris Dallager, Director of Disability Services (x5250) to arrange a confidential discussion regarding equitable access and reasonable accommodations.

Academic Honesty: All work that you turn in under your name must follow Carleton's academic integrity policy. The use of textbook solution manuals (physical or online solutions), homework, reports or exams done by past students are not allowed. Look at the College's Writing Across the Curriculum website for additional guidance on plagiarism and how to avoid plagiarism in their writing.

Chapter 3

Software in Stat 220

You will work with many .Rmd Markdown files in this course. These include class activities, homework template, project helper files etc. To stay organized, I *strongly* suggest you create a **stat220** folder that contains the following subfolders:

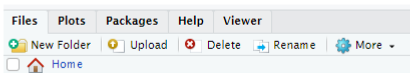
- **stat220** folder
  - **Assignments:** This folder will contain subfolders for each assignment. Each assignment subfolder (e.g. homework1, homework2, …) will be a Github connected RStudio project that you will create **once an assignment is posted.**
  - **Content:** This folder should be used to save any non-assignment files (e.g. slides, examples) for this class. You will create this subfolder by creating an RStudio project (see step 5 below).

To get started with this organization, follow the steps below.

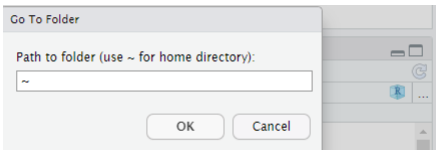
3.1 File organization: Using maize

The server (online) version of Rstudio is run from a unix server. You can navigate this file system using unix commands, but I assume that most or all of you will just use Rstudio to access your files on this server.

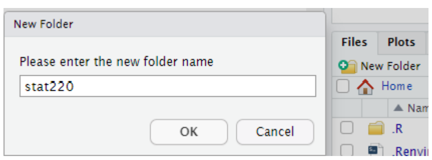
> **1.** In Rstudio, click the **Files** *tab* in the lower right-hand window.
> Note: this is **not** the same as the **File** *menu* option.



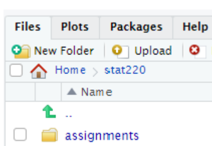
2. Verify that you are in your **HOME** folder (should simply say Home right under the New Folder button). To navigate to your Home folder (if somehow you are not in it), click the **...** button (far right side of the **Files** tab) and enter a ~ (tilde) symbol



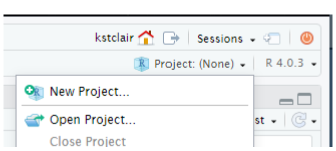
- **3.** Click the **New Folder** button and name the folder **stat220**.



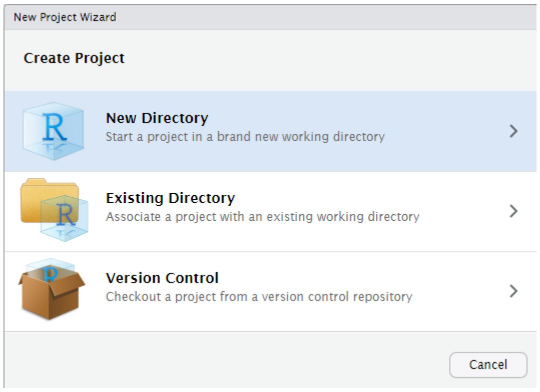
- **4.** Click on this newly created (empty) **stat220** folder. Within the folder create another **New Folder** and name it **assignments**.

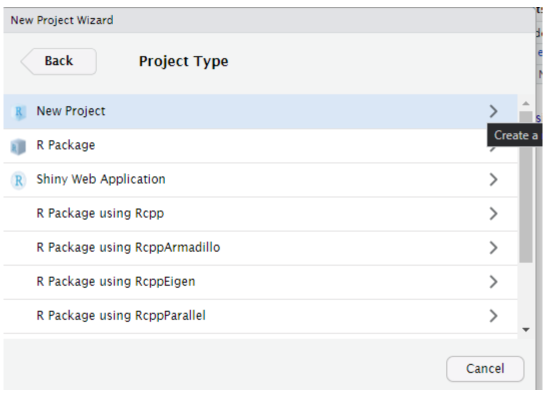


- **5.** Within the **stat220** folder, create an **RStudio project** called **content** with the following steps:
  - **a.** Click the **Project** button in the upper righthand corner of your RStudio window and select **New Project...**.

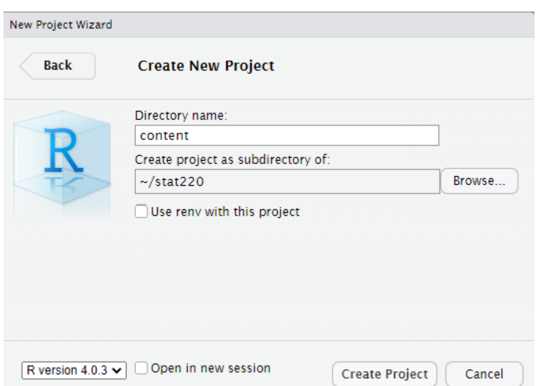


- **b.** Select **New Directory** and then **New Project**

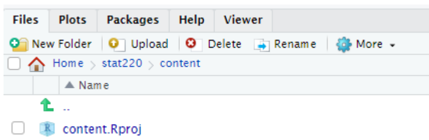




- **c.** Enter **content** as the **Directory name** and use the **Browse** button to find your **stat220** folder. Then click **Create Project**.



- **d.** You should now have a new folder called **content** in your **stat220** folder and this folder will contain an RStudio project `.Rproj`. Feel free to add subfolders to this **content** folder (e.g. slides, examples, etc).



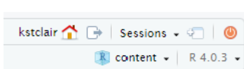
Warning: Do not create an RStudio project in the main stat220 folder because it is not good practice to have RStudio projects in subfolders of another project (e.g. a project within a project is not recommended).

3.2 File organization: Using your own Rstudio

Create a folder called **stat220** somewhere on your computer. Within this folder create an **assignments** subfolder. Then complete **step 5** from above to create a **content** RStudio project folder.

3.3 RStudio projects

Once you've created a project, your R session should be running within that project folder. You can check which project you are in by checking the project name in the upper righthand part of your RStudio window. Here we see the **content** project is open:



Running R from an RStudio project sets your **working directory** to the project folder:

```
> getwd()
[1] "/Accounts/kstclair/stat220/content"
```

This allows for easy file path access to all files related to this project.

To **start** a project, click on the `.Rproj` file or use the **Open Project...** option shown in step 5 above.

3.4 Best practices (or what not to do)

- Never save files to a lab computer hard drive (e.g. desktop, downloads, etc). They will be erased when you log off.
- Do not use gmail as a file storage system! Avoid emailing yourself files that you created (and saved) on a lab computer. Eventually you will lose work this way.
- Avoid using online versions of google drive and dropbox. Similar to gmail, downloading, editing a doc, then uploading it back to drive/dropbox is another great way to lose work.
- Avoid this and this.

3.5 Git and GitHub

Git is version control software that you install locally on your computer. Git is already installed on the maize RStudio server.

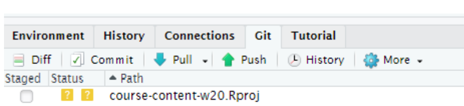
Github is a cloud-based service for hosting git projects. It allows multiple users to share and contribute to projects and it is how you will be submitting homework assignments and projects for this class. More information about Git and Github can also be found in Getting setup with Git and GitHub and Git and Github.

If you are using a local install of R/RStudio, then you will need to install Git.

Installing Git

Directions for both Windows & Mac here: http://happygitwithr.com/install-git.html.

1. If you are using **maize**, then there is nothing you need to install.
2. Windows users should follow Option 1 in 6.2.
3. Mac users can follow Option 1 in 6.3 if comfortable, otherwise follow Option 2
4. Linux users can follow 6.4.



3.6 Slack

We will use Slack for all course communication. Sign up for our course Slack team here! You will need to create an account with a username, and log in to read and post. You can download a standalone Slack application to your Mac, Windows, Linux and/or Android/iOS device. You can control whether you receive notifications on new posts by going to Preferences, as well as decide which 'channels' to subscribe to. A 'channel' is a discussion thread, which is used to organize communications into topics. You can learn more about Slack features here.

Several channels have been set up for specific parts of the course. Feel free to ask questions anytime. You can browse the available channels in our team by clicking on "Channels" on the left-hand panel.

3.7 Acknowledgements

This installation guide is based on the guide from Adam Loy and Katie St. Clair.

Chapter 4

GitHub Guide for Students in Stat 220

4.1 Overview

If you are using the maize RStudio server, then you can connect to GitHub without any extra software downloads. If you are using RStudio on your computer, then you will need to download Git software (as directed in Software in Stat 220) to use GitHub connected projects. You will use GitHub to submit homework and collaborate on projects.

4.2 Getting setup with Git and GitHub

If you are **not** working on the maize RStudio server, then make sure that you have installed all of the software mentioned in Software in Stat 220. In addition, you should install the `usethis` and `gitcreds` R packages.

Everyone needs to connect Git and GitHub by doing the following:

1. Register for account on GitHub (https://github.com/). I recommend using a username that incorporates your name (e.g., dbastola) and Carleton email address for your Github account.

2. If you haven't done so already, accept the invite to the class organization DataScienceSpring23. This organization is where all course homework files and project repositories will live.

3. Setup options in Git by running the following code chunk in your console:

```r
#install.packages("usethis")  # uncomment to install
usethis::use_git_config(user.name = "Jane Doe", user.email = "jane@example.org")
```

changing the first two arguments to your own name and email (this should be the email associated with your GitHub account).

4. In order to push changes to github (i.e. to track changes and submit homework), you will need to prove that you have permission to change a Github repo. This is done with a personal access token (PAT). Note that you will need to install the packages usethis and gitcreds to do this.

```r
usethis::create_github_token()
```

```
 Call `gitcreds::gitcreds_set()` to register this token in the local Git credentia
 It is also a great idea to store this token in any password-management software t
   Opening URL 'https://github.com/settings/tokens/new?scopes=repo,user,gist,workf
```

"Generate token" and store your tokens somewhere safe in your local computer as you will need this again in the future. You can additionally add PAT to your `.Renviron` file as well. Copy it and paste it into your .Renviron file as system variable GITHUB_PAT using

```r
usethis::edit_r_environ()
```

Add to the file and save. You can also set the PAT token in R using the following.

```r
#install.packages("gitcreds") # uncomment to install
gitcreds::gitcreds_set()
```

You can check that you've stored a credential with `gitcreds_get()`:

```r
gitcreds::gitcreds_get()
```

You should get something like this:

~~~
```
#> <gitcreds>
#>   protocol: https
#>   host    : github.com
#>   username: PersonalAccessToken
#>   password: <-- hidden -->
```
~~~

Treat your PAT token like a password! For details, follow the step in Section 9.1 on this page to do this: https://happygitwithr.com/https-pat.html.

4.3 Individual assignments

If you followed the suggestions in the File organization in RStudio page, then you should already have an assignments folder on your computer or maize account.

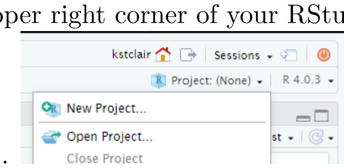
Each new assignment/project will be posted as a repository on GitHub and added directly to your account (within the Stat220 organization). This repository will contain assignment details (README, .Rmd).

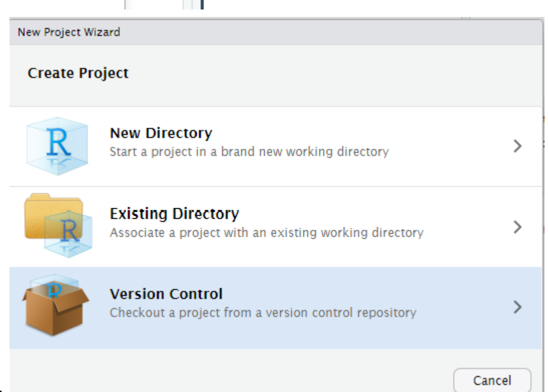
4.3.1 Creating an individual assignment repo and project

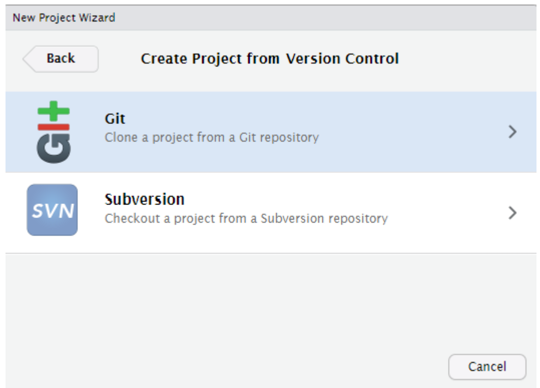
1. Go to our course GitHub organization page (DataScienceSpring23) and find your homework repo, such as `hw1-username` (where your username is attached).

2. Enter the online assignment repository on GitHub. Click the green **"Code"** button. Most of you should just use the default setting which is to "clone" (copy) using HTTPS. Click the clipboard to the right of the URL to copy the repo location.

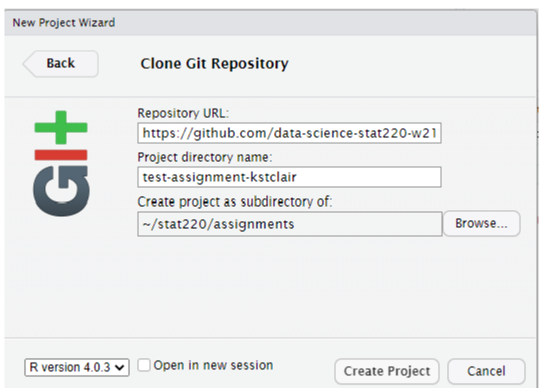
3. Now open up RStudio and create a project as follows:

    - Click the **Project** button in the upper right corner of your RStudio window and select **New Project...**. 

    - Select **Version Control** and then **New Project** 



- Paste the link you just copied into the Repository URL box. Leave the Project directory name blank (or keep the auto-filled name). Use the **Browse** button to find your **assignments** folder, then click **Create Project**



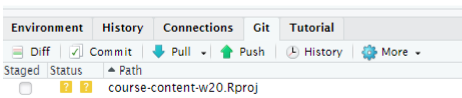
4.3.2 Working on your assignment

An RStudio project should now open, which will allow you to start working on your homework assignment. You should see the project assignment name in the top right side of Rstudio. You will probably see a blank console screen when you open a new project. Look in the **Files** tab for your homework .Rmd file. Click on whatever file you want to edit (probably the .Rmd file) and edit away. Make sure that your current assignment's project is the one open and showing in the upper rightproject name. To **open** a project, click on the `.Rproj` file or use the **Open Project...** option available in the upper right project link.

4.3.2.1 Commits

After you make changes to the homework assignment, commit them. What are commits you ask? Commits are essentially taking a snapshot of your projects. Commits save this snapshot to your local version of Git (located on your hard drive or the maize server). For example, if I make changes to a code so that it prints "Hello world", and then commit them with an informative message, I can look at the history of my commits and view the code that I wrote at that time. If I made some more changes to the function that resulted in an error, I could go back to the commit where the code was originally working. This prevents you from creating several versions of your homework (homework-v1, homework-v2, ...) or from trying to remember what your code originally looked like.

You can make commits in the Git tab in RStudio.



Click the **Commit** button in the Git tab. Check the boxes of the files that you want to commit, enter your commit message (briefly state what changes have been made), then hit **Commit**. You can read how to do this in RStudio in more detail here: http://r-pkgs.had.co.nz/git.html#git-commit.

Two things about committing.

- You should **commit somewhat frequently**. At minimum, if you're doing a homework assignment, you should make a commit each time that you've finished a question.
- Leave **informative commit messages**. "Added stuff" will not help you if you're looking at your commit history in a year. A message like "Added initial version of hello-world function" will be more useful.

4.3.2.2 Pushing changes to Github

At some point you'll want to get the updated version of the assignment back onto GitHub, either so that we can help you with your code or so that it can be graded. You will also want to push work frequently when you have a shared GitHub repo for project collaborations (i.e. more than one person is working on a project and code). If you are ready to push, you can again click on the "Up" **Push** arrow in the Git tab or in the Commit pop-up window or in the Git tab (shown above).

To "turn in" an assignment, all you need to do is push all your relevant files to Github by the deadline.

4.4 Group work

Collaborative Github assignments are pretty similar to individual assignments.

4.4.1 Creating a group/partner assignment repo and project

Go to our course GitHub organization page(DataScienceSpring23) and find the repo for your group, for example if your group name is "team01" the you might find the `mp1-team01` repo. Clone this repo to your computer/maize account using the same steps done for an individual assignment (see steps 2-3).

4.4.1.1 Working with collaborative repos

For group homework, I suggest that only the *recorder* edit the group-homework-x.Rmd file to avoid merge conflicts! Other group members can create a new Markdown doc to run and save commands. Only the recorder needs to **push** changes (answers) to the Github repo and all others can then **pull** these changes (i.e. the final answers) after the HW is submitted.

When you are working together on a Github project, you should commit and push your modifications frequently. You will also need to frequently **pull** updates from Github down to your local version of RStudio. These updates are changes that your teammates have made since your last pull. To pull in changes, click the "Down" **Pull** arrow in the Git tab (shown above).

If you get an error about conflict after pulling or pushing, don't freak out! This can happen if you edit a file (usually an .Rmd or .R file) in a location that was also changed by a teammate. When this happens you should attempt to fix the **merge conflict**. Take a look at this resource site and try to fix the merge conflict in Rstudio.

4.5 Additional resources

- Happy Git and GitHub for the useR
- Rstudio, Git and GitHub
- Interactive learning guide for Git
- GitHub Guides
- Git setup for Windows (video)
- Git setup for Mac (video)
- How to clone, edit, and push homework assignments with GitHub Classroom (video)

4.6 Acknowledgements

Most of this content in this guide was taken from https://github.com/jfiksel/github-classroom-for-students, edited for our classroom use by Katie St. Clair.

4.7 Reuse

Chapter 5

R Markdown Syntax

Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

5.0.1 Lists in R Markdown:

You can use asterisk mark to provide emphasis, such as `*italics* or **bold**`. You can create lists with a dash:

```
- Item 1
- Item 2
- Item 3
  + Subitem 1
* Item 4
```

to produce

- Item 1
- Item 2
- Item 3
  - Subitem 1
- Item 4

You can embed Latex equations in-line, `$\frac{1}{n} \sum_{i=1}^{n} x_{i}$` to produce $\frac{1}{n}\sum_{i=1}^{n} x_i$ or in a new line as `$$\text{Var}(X) = \frac{1}{n-1}\sum_{i-1}^{n} (x_{i} - \bar{x})^2$$` to produce

$$\text{Var}(X) = \frac{1}{n-1}\sum_{i-1}^{n} (x_i - \bar{x})^2$$

5.0.2 Embed an R code chunk:

Use the following

````
```r
Use back ticks to
create a block of code
```
````

to produce:

```
Use back ticks to
create a block of code
```

You can also evaluate and display the results of R code. Each tasks can be accomplished in a suitably labeled chunk like the following:

```r
summary(cars)
```

```
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
```

```r
fit <- lm(dist ~ speed, data = cars)
fit
```

```

Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed  
    -17.579        3.932  
```

5.0.3 Including Plots:

You can also embed plots. See Figure 5.1 for example:

```
par(mar = c(0, 1, 0, 1))
pie(
  c(280, 60, 20),
  c('Sky', 'Sunny side of pyramid', 'Shady side of pyramid'),
  col = c('#0292D8', '#F7EA39', '#C4B632'),
  init.angle = -50, border = NA
)
```

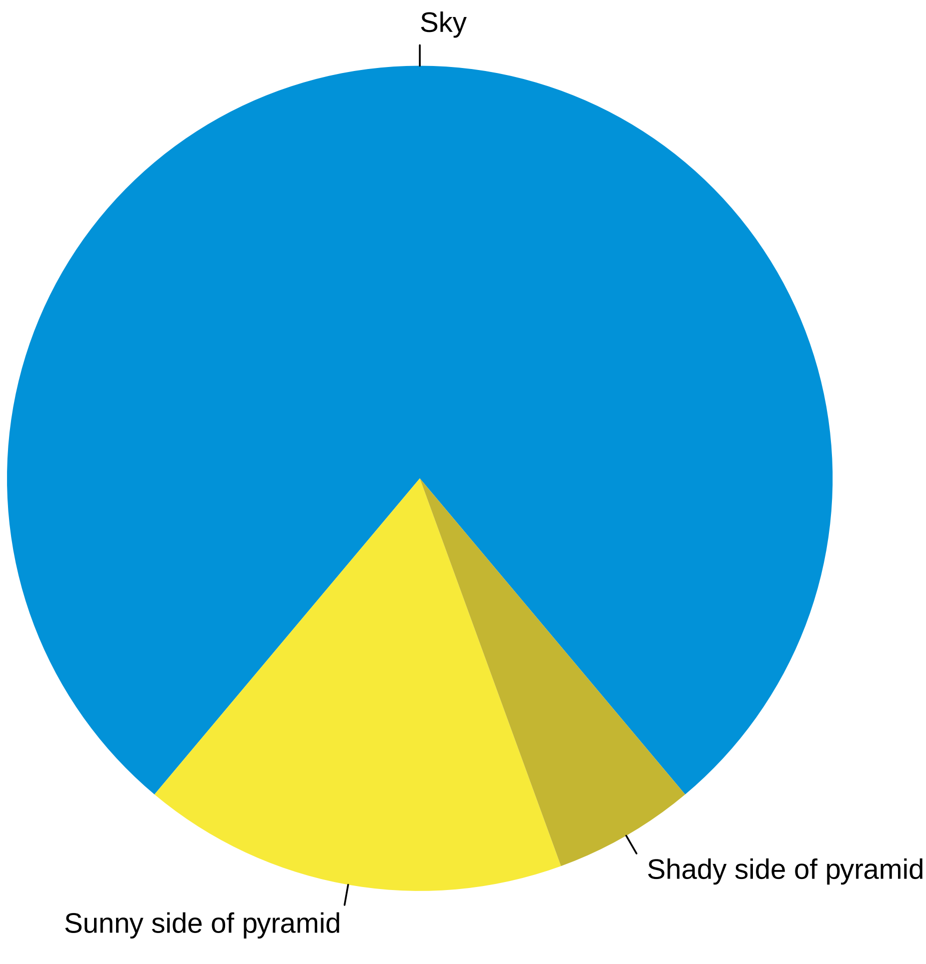


Figure 5.1: A fancy pie chart.

(Credit: Yihui Xie)

5.0.4 Read in data files:

```
simple_data <- read.csv("https://deepbas.io/data/simple-1.dat", )
summary(simple_data)
```

```
   initials             state                 age      
 Length:3           Length:3           Min.   :45.0  
 Class :character   Class :character   1st Qu.:47.5  
 Mode  :character   Mode  :character   Median :50.0  
                                       Mean   :52.0  
                                       3rd Qu.:55.5  
                                       Max.   :61.0  
     time          
 Length:3          
 Class :character  
 Mode  :character  
```

```
knitr::kable(simple_data)
```

initials	state	age	time
vib	MA	61	6:01
adc	TX	45	5:45
kme	CT	50	4:19

5.0.5 Hide the code:

If we enter the `echo = FALSE` option in the R chunk (see the .Rmd file). This prevents the R code from being printed to your document; you just see the results.

initials	state	age	time
vib	MA	61	6:01
adc	TX	45	5:45
kme	CT	50	4:19

Class Activities

Chapter 6

Class Activity 0

```r
# load the required libraries
library(credentials) # to help with PAT access
library(gitcreds)
library(usethis)
```

```r
# STEPS INVOLVED TO ESTABLISH GIT CREDENTIALS / PAT

# Step 1

# usethis::use_git_config(user.name = "deepbas", user.email = "deepbas99@gmail.com")

# Step 2

# usethis::create_github_token()

# Step 3

# if this is the second/subsequent iteration start from here

# gitcreds::gitcreds_set()

# Verify

# gitcreds::gitcreds_get()
```

In this worksheet, you will practice creating a GitHub repository using the usethis::use_github() function and cloning it back to your local machine using RStudio's menu options.

6.1 Tutorial 1: Creating and cloning a Repository starting from Github to RStudio

1. Visit the GitHub website at https://github.com and sign in using your GitHub account. If you don't have an account yet, you can create one for free.

2. Once logged in, click on the "+" icon in the top right corner of the webpage, then click on "New repository".

3. Enter a name for your new repository in the "Repository name" field. You may also provide an optional description.

4. Choose the visibility of your repository by selecting either "Public" or "Private". Public repositories are visible to anyone, while private repositories are only visible to you and any collaborators you invite.

5. (Optional) Check the box to initialize the repository with a README file.

6. Click on the "Create repository" button to create your new repository.

This will create a new GitHub repository on your Github account. Follow further to clone the repository to your local folder using RStudio.

1. Go to your GitHub repository webpage and click on the green "Code" button. This will display a dropdown menu with a URL for your repository. Click on the clipboard icon to copy the URL to your clipboard.

2. Open RStudio, and from the "File" menu, select "New Project".

3. In the "New Project" dialog, choose "Version Control".

4. Select "Git" as the version control system.

5. In the "Repository URL" field, paste the URL that you copied from your GitHub repository webpage.

6. Choose a local directory where you want to clone the repository by clicking on the "Browse" button and navigating to the desired folder on your computer.

7. Click on "Create Project" to clone the GitHub repository to your local computer.

6.2 Tutorial 2: Creating a new GitHub repository using `usethis` R package (RStudio to Github) (Works ONLY on local RStudio)

6.2.1 Prerequisites

1. Install the usethis package if you haven't already: install.packages("usethis")

2. Make sure you have a GitHub account, and you are logged in.

3. Configure Git with your name and email address if you haven't already. Run the following commands in the R console, replacing "Your Name" and "youremail@example.com" with your information:

```
usethis::use_git_config(user.name = "Your Name", user.email = "youremail@example.com")
```

4. Create a new R project in RStudio by clicking on "File" > "New Project" > "New Directory" > "New Project." Give your project a name and choose a location on your computer to save it. Click "Create Project."

5. Make a new file or copy and paste a .Rmd file that you want to have in your repo and save it to your requirement.

6. In the R console, load the usethis package:

```
library(usethis)
```

7. Initialize a Git repository for your project by running:

```
usethis::use_git()
```

8. Now, let's create a new GitHub repository using the usethis::use_github() function. Run the following command:

```
usethis::use_github()
```

9. Follow the instructions in the R console, and your GitHub repository will be created. Note the repository URL, as you will need it in the next activity.

6.3 (Optional) Tutorial 3: Creating a new GitHub repository using RStudio's Git terminal

1. Create a new R project in RStudio by clicking on "File" > "New Project" > "New Directory" > "New Project." Give your project a name and choose a location on your computer to save it. Click "Create Project."

2. Initialize a Git repository for your project. Click on the "Terminal" tab in the bottom right pane of RStudio and run the following command:

```
git init
```

3. Configure Git with your name and email address if you haven't already. Run the following commands in the terminal, replacing "Your Name" and "youremail@example.com" with your information:

```
git config user.name "Your Name"
git config user.email "youremail@example.com"
```

4. Commit your project files to the Git repository. Run the following commands in the terminal:

```
git add .
git commit -m "Initial commit"
```

5. Go to your GitHub account, click on the "+" icon in the upper right corner, and select "New repository." Give your repository a name (it's recommended to use the same name as your R project), and click "Create repository."

6. In the "…or push an existing repository from the command line" section of your new GitHub repository, copy the commands under this section. It should look similar to the following (replace and with your GitHub username and repository name):

```
git remote add origin https://github.com/<your-username>/<your-repo-name>.git
git branch -M main
git push -u origin main
```

7. Paste the copied commands into the RStudio terminal and execute them. This will link your local Git repository to the remote GitHub repository and push your initial commit to the GitHub repository.

Chapter 7

Class Activity 1

The R package `babynames` provides data about the popularity of individual baby names from the US Social Security Administration. Data includes all names used at least 5 times in a year beginning in 1880.

```
#install.packages("babynames")  # uncomment to install
library(babynames)
```

Below is the list for first few cases of baby names.

```
head(babynames)
```

```
# A tibble: 6 x 5
   year sex   name          n   prop
  <dbl> <chr> <chr>     <int>  <dbl>
1  1880 F     Mary       7065 0.0724
2  1880 F     Anna       2604 0.0267
3  1880 F     Emma       2003 0.0205
4  1880 F     Elizabeth  1939 0.0199
5  1880 F     Minnie     1746 0.0179
6  1880 F     Margaret   1578 0.0162
```

1. How many cases and variables are in the dataset `babynames`?

Answer:

```
dim(babynames)
```

```
[1] 1924665       5
```

There are 1924665 cases and 5 variables in the dataset `babynames`.

Let's use the package tidyverse to do some exploratory data analysis.

```
#install.packages("tidyverse")   # uncomment to install
library(tidyverse)
babynames %>% filter(name=='Aimee')
```

```
# A tibble: 150 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     Aimee    13 0.000133
 2  1881 F     Aimee    11 0.000111
 3  1882 F     Aimee    13 0.000112
 4  1883 F     Aimee    11 0.0000916
 5  1884 F     Aimee    15 0.000109
 6  1885 F     Aimee    17 0.000120
 7  1886 F     Aimee    17 0.000111
 8  1887 F     Aimee    18 0.000116
 9  1888 F     Aimee    12 0.0000633
10  1889 F     Aimee    16 0.0000846
# ... with 140 more rows
```

```
filtered_names <- babynames %>% filter(name=='Aimee')
```

```
#install.packages("ggplot2")   # uncomment to install
library(ggplot2)
```

```
ggplot(data=filtered_names, aes(x=year, y=prop)) +
  geom_line(aes(colour=sex)) +
  xlab('Year') +
  ylab('Prop. of Babies Named Aimee')
```

2. What do you see in the Figure 1? Explain in a few sentences.

Click for answer

Answer:

In Figure 1, we can see the proportion of babies named Aimee by year for both males and females. We notice that the name Aimee has been more popular among females than males throughout the years. There is a peak in popularity around the 1970s for female babies, and then the popularity declines.

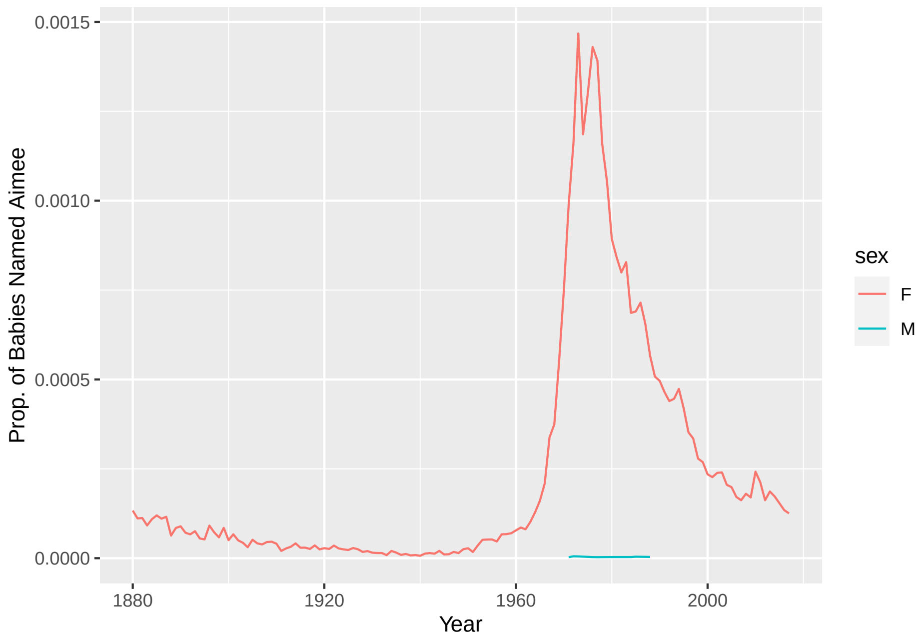


Figure 7.1: A trend chart

3. Repeat question 2 to infer how does the proportion of babies with your first name trend over time. Examine the generated plot and describe the trend of your name's popularity over time. Consider the following points:

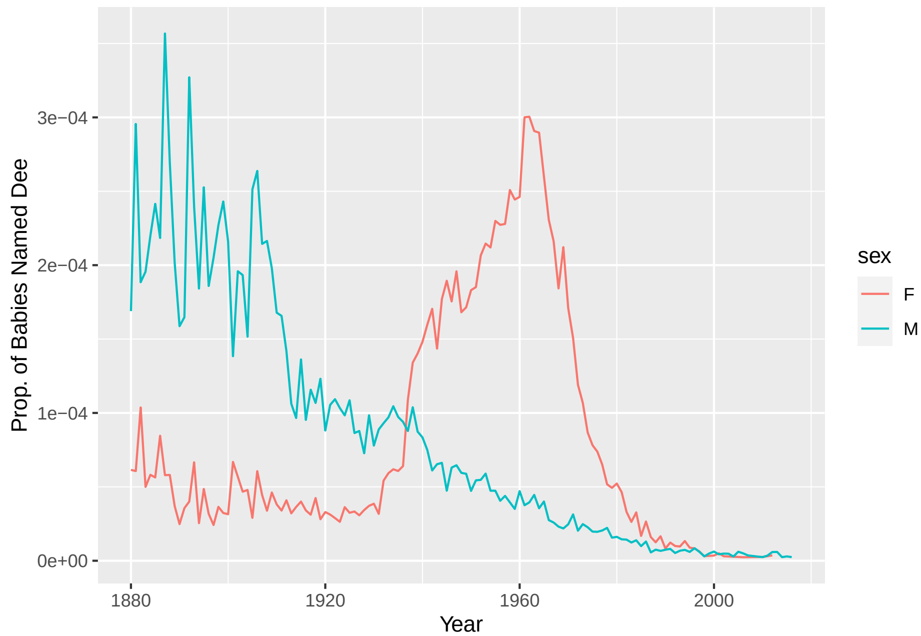
Has the popularity of your name increased, decreased, or remained stable over the years? Is there a noticeable difference in popularity between sexes? Are there any interesting patterns or trends, such as sudden increases or decreases in popularity?

Answer: Answers will vary.

```
# Replace 'YourName' with your first name
your_name <- "Dee"

your_name_data <- babynames %>% filter(name == your_name)

ggplot(data=your_name_data, aes(x=year, y=prop)) +
  geom_line(aes(colour=sex)) +
  xlab('Year') +
  ylab(paste('Prop. of Babies Named', your_name))
```



4. Compare the popularity of your first name with a randomly chosen name from the dataset. Examine the generated plot and compare the popularity of your first name with the randomly chosen name. Consider the following points:

Are there differences in popularity trends between the two names? Is one name consistently more popular than the other, or do their popularity levels change over time? Are there any interesting patterns or trends in the data, such as periods of rapid increase or decrease in popularity?

Answer Answers will vary

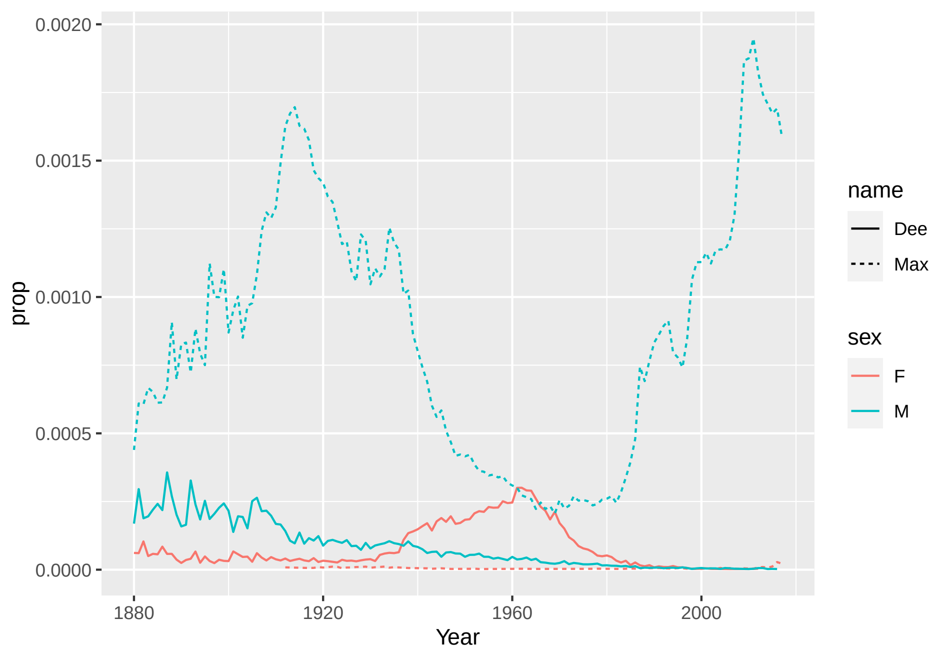
```
# Replace 'YourName' with your first name
your_name_data <- babynames %>% filter(name == 'Dee')

# Replace 'RandomName' with a randomly chosen name from the dataset
random_name_data <- babynames %>% filter(name == 'Max')

# Combine the two datasets
combined_data <- bind_rows(your_name_data, random_name_data)

# Plot the data
ggplot(data=combined_data, aes(x=year, y=prop)) +
```

```
  geom_line(aes(colour=sex, linetype=name)) +
  xlab('Year')
```



7.1 Extras (optional)

7.1.1 Part 1: Setting Working Directory and Loading Data

1. Set your working directory to a folder on your computer where you would like to save your R scripts and data files.

```
# Replace 'your_directory_path' with the path to your desired folder
# setwd("your_directory_path")
```

2. Load the mtcars dataset which comes preloaded with R. This dataset consists of various car features and their corresponding miles per gallon (mpg) values.

```
data(mtcars)
head(mtcars)
```

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

7.1.2 Part 2: Downloading Packages

1. Install the "tidyverse" package, which is a collection of useful R packages for data manipulation, exploration, and visualization.

```r
# Uncomment the line below to install the package
# install.packages("tidyverse")
```

2. Load the "tidyverse" package into your R session.

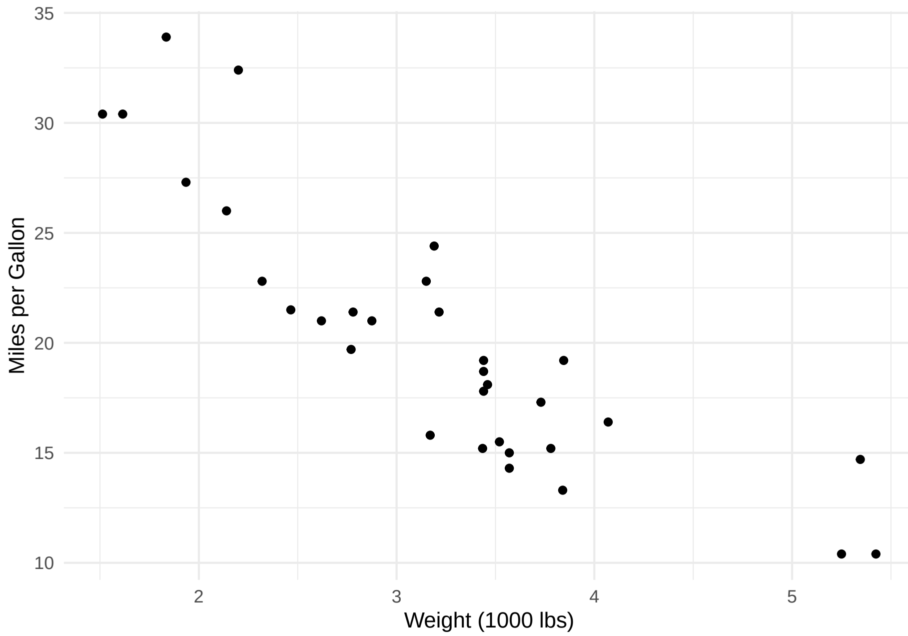
```r
library(tidyverse)
```

7.1.3 Part 3: Creating and Compiling an R Markdown File

1. Create a new R Markdown file in RStudio by clicking on "File" > "New File" > "R Markdown…". Save the file in your working directory.

2. Add the following code to your R Markdown file to create a scatter plot of the mtcars dataset, showing the relationship between miles per gallon (mpg) and the weight of the car (wt).

```r
# Create a scatter plot
ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  xlab("Weight (1000 lbs)") +
  ylab("Miles per Gallon") +
  theme_minimal()
```



3. Knit your R Markdown file to create an output document. Click the "Knit" button at the top of the RStudio script editor, and choose the output format you prefer (e.g., HTML, PDF, or Word).

7.2 Questions

7.2.1 1. How does the weight of a car (wt) affect its miles per gallon (mpg) based on the scatter plot you created?

Click for answer

Answer:

Based on the scatter plot, there appears to be a negative relationship between the weight of a car (wt) and its miles per gallon (mpg). As the weight of a car increases, its fuel efficiency (mpg) tends to decrease.

7.2.2 2. What is the importance of setting a working directory in R?

Click for answer

Answer:

Setting a working directory in R is important because it determines the default location where R will read from or write to when loading or saving files. This makes it easier to keep your files organized and ensures that your R scripts can access the necessary files without needing to specify the full file paths. It also simplifies sharing your R projects with others since the file paths within your scripts will be relative to the working directory.

7.2.3 3. Explain the role of R Markdown in creating reproducible research documents.

Click for answer

Answer:

R Markdown plays a crucial role in creating reproducible research documents by allowing you to combine text, code, and output (e.g., tables, figures) within a single document. This integration of narrative, data, and results makes it easier to document your data analysis process, ensuring that others can easily understand, reproduce, and build upon your work. R Markdown also supports various output formats (e.g., HTML, PDF, Word) to make it easy to share your research findings with others.

Chapter 8

Class Activity 2

Let's practice some common data assignments and manipulations in R.

a. Create a vector of all integers from 4 to 10, and save it as `a1`.

Click for answer

```
a1 <- 4:10
a1
```

```
[1]  4  5  6  7  8  9 10
```

b. Create a vector of *even* integers from 4 to 10, and save it as `a2`.

Click for answer

```
a2 <- seq(4, 10, by=2)
a2
```

```
[1]  4  6  8 10
```

c. What do you get when you add `a1` to `a2`?

Click for answer

```
a1_plus_a2 <- a1 + a2
a1_plus_a2
```

```
[1]  8 11 14 17 12 15 18
```

Answer: When you add `a1` to `a2`, you get a vector containing the element-wise sum: 8, 11, 14, 17, 12, 15, 18.

d. What does the command `sum(a1)` do?

Click for answer

```r
sum_a1 <- sum(a1)
sum_a1
```

```
[1] 49
```

Answer: The command `sum(a1)` calculates the sum of all elements in the vector `a1`. In this case, it returns 49.

e. What does the command `length(a1)` do?

Click for answer

```r
length_a1 <- length(a1)
length_a1
```

```
[1] 7
```

Answer: The command `length(a1)` returns the number of elements in the vector `a1`. In this case, there are 7 elements.

f. Use the `sum` and `length` commands to calculate the average of the values in `a1`.

Click for answer

```r
average_a1 <- sum(a1) / length(a1)
average_a1
```

```
[1] 7
```

Answer: The average of the values in `a1` is 7.

8.1 Extras (Optional)

In this worksheet, you will learn how to use the `write_csv()` function from the `readr` package to save a data object from your R session to a file in your working directory.

First, let's load the necessary package.

```
library(readr)
```

Suppose we have a simple data frame called `my_data`, with three columns: `Name`, `Age`, and `City`.

```
my_data <- data.frame(Name = c("Alice", "Bob", "Charlie", "David"),
                      Age = c(25, 30, 22, 35),
                      City = c("New York", "Los Angeles", "Chicago", "San Francisco"))
my_data
```

```
     Name Age          City
1   Alice  25      New York
2     Bob  30   Los Angeles
3 Charlie  22       Chicago
4   David  35 San Francisco
```

Now we want to save this data frame as a CSV file in our working directory. To do this, we will use the `write_csv()` function. The first argument is the data object you want to save, and the second argument is the file name (including the .csv extension).

```
write_csv(my_data, "my_data.csv")
```

This command will save `my_data` as a file called `my_data.csv` in your working directory.

Question 1: What is the purpose of the `write_csv()` function?

Click for answer

Answer:

The `write_csv()` function is used to save a data object from an R session to a CSV file in your working directory.

Question 2: How do you save a data frame called `my_data` as a file named "example_data.csv"?

```
# Your code here
```

Click for answer

Answer:

```
write_csv(my_data, "example_data.csv")
```

Question 3: If you want to save a data frame called `students` as a file named "student_data.csv", what would be the appropriate command?

```
# Your code here
```

Click for answer

Answer:

```
write_csv(students, "student_data.csv")
```

Chapter 9

Class Activity 3

```
# some interesting data objects
x <- c(3,6,9,5,10)
x.mat <- cbind(x, 2*x)
x.df <- data.frame(x=x,double.x=x*2)
my.list <- list(myVec=x, myDf=x.df, myString=c("hi","bye"))
```

9.1 Question 1: data types

- What data type is `x`?

Click for answer

Answer:

```
# code
typeof(x)
```

```
[1] "double"
```

- What data type is `c(x, x/2)`?

Click for answer

Answer:

```
# code
typeof(c(x, x/2))
```

```
[1] "double"
```

- What data type is `c(x,NA)`? What data type is `c(x,"NA")`?

Click for answer

Answer:

```
# code
typeof(c(x, NA))
```

```
[1] "double"
```

```
typeof(c(x, "NA"))
```

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

9.2 Question 2: Subsetting and coercion

- How can we reverse the order of entries in `x`?

Click for answer

Answer:

```
# code
rev(x)
```

```
[1] 10  5  9  6  3
```

```
x[length(x):1]
```

```
[1] 10  5  9  6  3
```

- What does `which(x < 5)` equal?

Click for answer

Answer:

```
# code
which(x<5)
```

```
[1] 1
```

- Extract the element of x that corresponds to the location in the preceding question.

Click for answer

Answer:

```
# code
x[which(x<5)]
```

```
[1] 3
```

- What does `sum(c(TRUE,FALSE,TRUE,FALSE))` equal?

Click for answer

Answer:

```
# code
sum(c(TRUE,FALSE,TRUE,FALSE))
```

```
[1] 2
```

- What does `sum(x[c(TRUE,FALSE,TRUE,FALSE)])` equal?

Click for answer

Answer:

```
# code
sum(x[c(TRUE,FALSE,TRUE,FALSE, TRUE)])
```

```
[1] 22
```

- What does `sum(x < 5)` equal?

Click for answer

Answer:

```
# code
sum(x < 5)
```

```
[1] 1
```

- What does `sum(x[x < 5])` equal?

Click for answer

Answer:

```
# code
sum(x[x < 5])
```

```
[1] 3
```

- Why `dim(x.mat[1:2,1])` return `NULL` while `dim(x.mat[1:2,1:2])` returns a dimension?

Click for answer

Answer:

```
# code
dim(x.mat[1:2,1])
```

```
NULL
```

```
dim(x.mat[1:2,1:2])
```

```
[1] 2 2
```

9.3 Question 3: Lists

- Using `my.list`, show three ways to write one command that gives the 3rd entry of variable `x` in data frame `myDf`

Click for answer

Answer:

```
# code
my.list[[1]][3]
```

```
[1] 9
```

```
my.list[["myVec"]][3]
```

```
[1] 9
```

```
my.list[1]$myVec[3]
```

```
[1] 9
```

```
my.list$myVec[3]
```

```
[1] 9
```

- What class of object does the command `my.list[3]` return?

Click for answer

Answer:

```
# code
class(my.list[3])
```

```
[1] "list"
```

- What class of object does the command `my.list[[3]]` return?

Click for answer

Answer:

```
# code
class(my.list[[3]])
```

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

- What class of object does the command `unlist(my.list)` return? Why are all the entries `characters`?

Click for answer

Answer:

```
# code
class(unlist(my.list))
```

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

Chapter 10

Class Activity 4

```
# Load the required libraries
library(tidyverse)
library(ggplot2)
library(datasauRus)
```

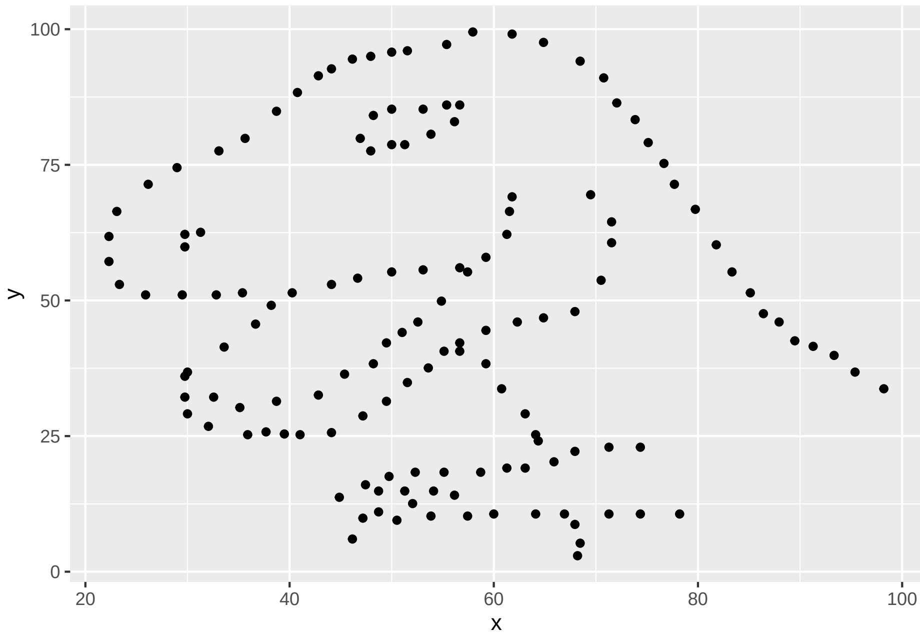
10.1 Your turn 1

This worksheet will guide you through creating various plots using the `ggplot2` package in R. We will be using the `datasaurus`_dozen dataset from the `datasauRus` package for demonstration purposes. The dataset contains 13 different datasets, and we'll use them to create a variety of plots.

10.1.1 Scatterplot

a. Run the following code.

```
ggplot(data = dino_data, mapping = aes(x = x, y = y)) +
  geom_point()
```



b. You *must* remember to put the aesthetic mappings in the `aes()` function! What happens if you forget?

Click for answer

Answer:

If you forget to put the aesthetic mappings inside the aes() function, ggplot2 will not be able to map the variables to the aesthetics correctly, and you might encounter an error or unexpected behavior in your plot.

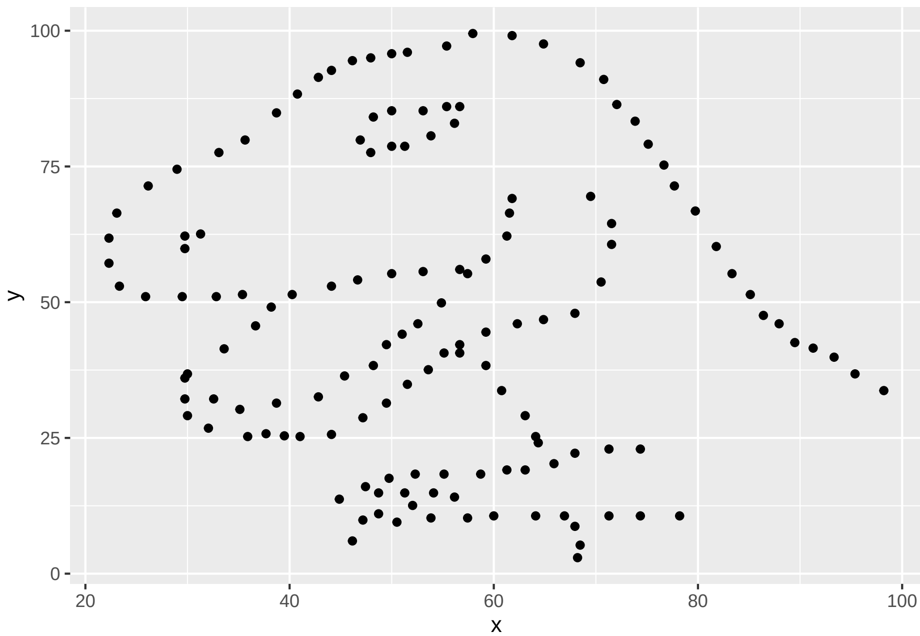
```
# Add a layer and see what happens
ggplot(data = dino_data , x = x , y = y)
```

c. The aesthetic mappings can be specified in the geom layer if you prefer, instead of the main `ggplot()` call. Give it a try:

Click for answer

Answer:

```
# Rebuild the scatterplot with your aesthetic mapping in the geom layer
ggplot(data = dino_data) +
  geom_point(aes(x = x, y = y))
```



10.1.2 Bar Plot

In this problem, we'll explore creating a bar plot using the `datasaurus_dozen` dataset.

a. Create a new data frame containing the count of observations in each dataset.

Click for answer

Answer:

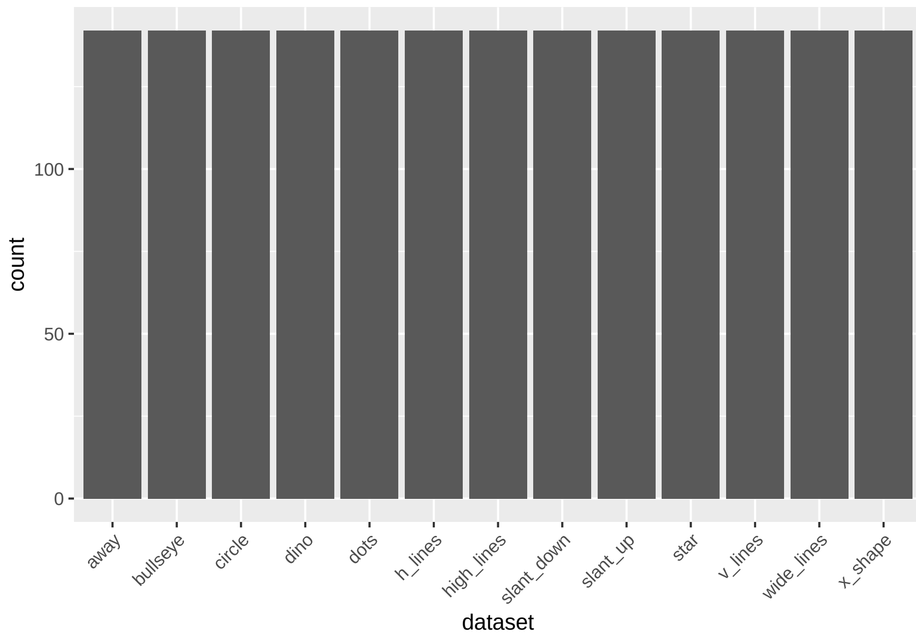
```
dataset_counts <- datasaurus_dozen %>%
  group_by(dataset) %>%
  summarise(count = n())
```

b. Create a bar plot showing the number of observations in each dataset.

Click for answer

Answer:

```
ggplot(data = dataset_counts, aes(x = dataset, y = count)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



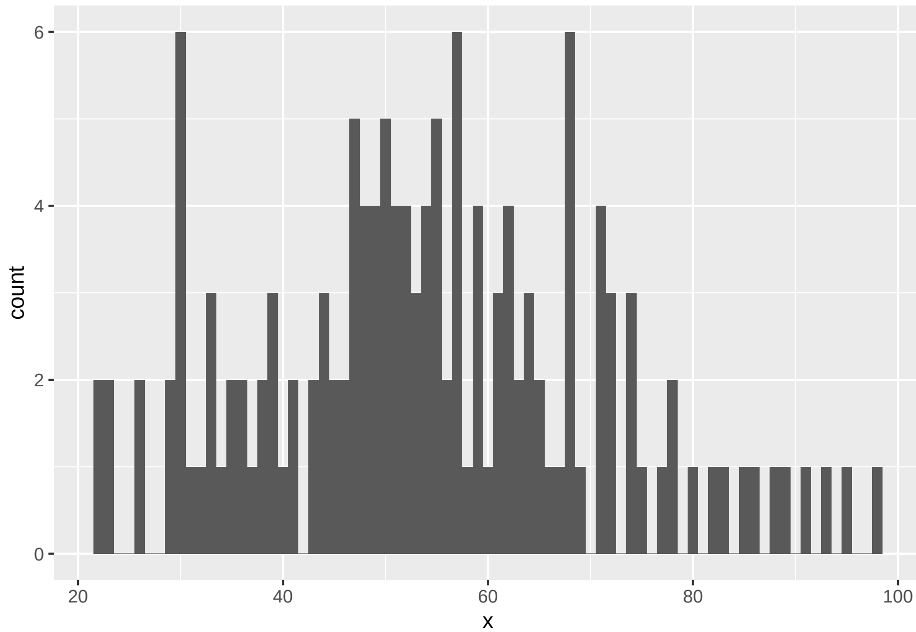
10.1.3 Histogram

a. Create a histogram of the `x` variable for the `dino` dataset.

Click for answer

Answer:

```
ggplot(data = dino_data, aes(x = x)) +
  geom_histogram(binwidth = 1)
```

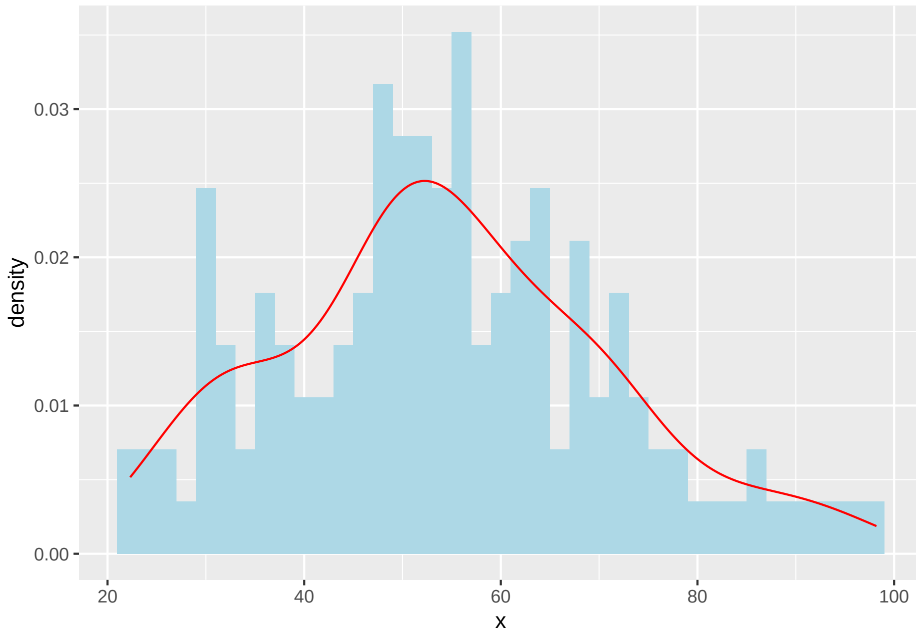


b. Overlay a density curve on the histogram.

Click for answer

Answer:

```
ggplot(data = dino_data, aes(x = x)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 2, fill = "lightblue") +
  geom_density(color = "red")
```



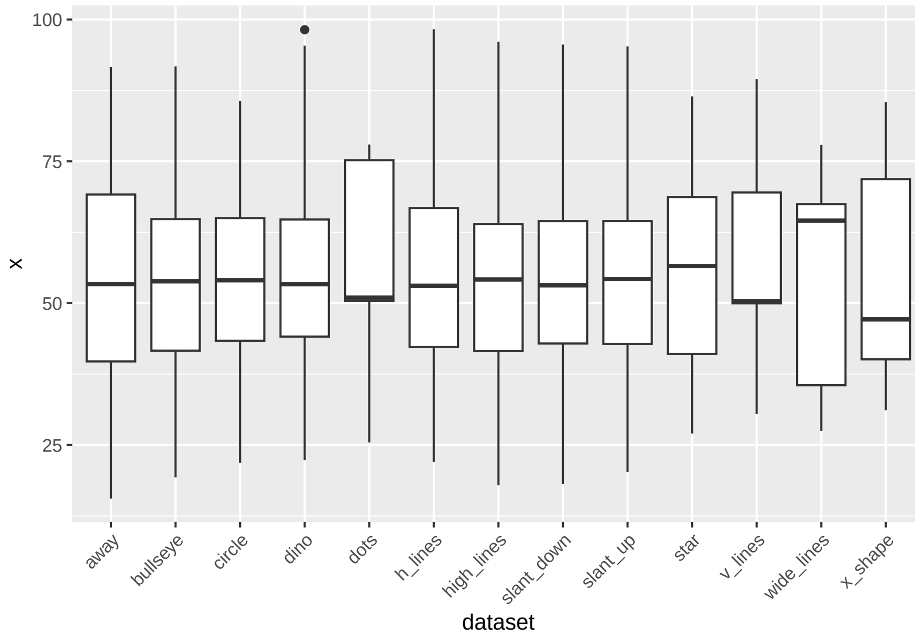
10.1.4 Boxplot

a. Create a boxplot of the x variable for each dataset in datasaurus_dozen.

Click for answer

Answer:

```
ggplot(data = datasaurus_dozen, aes(x = dataset, y = x)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



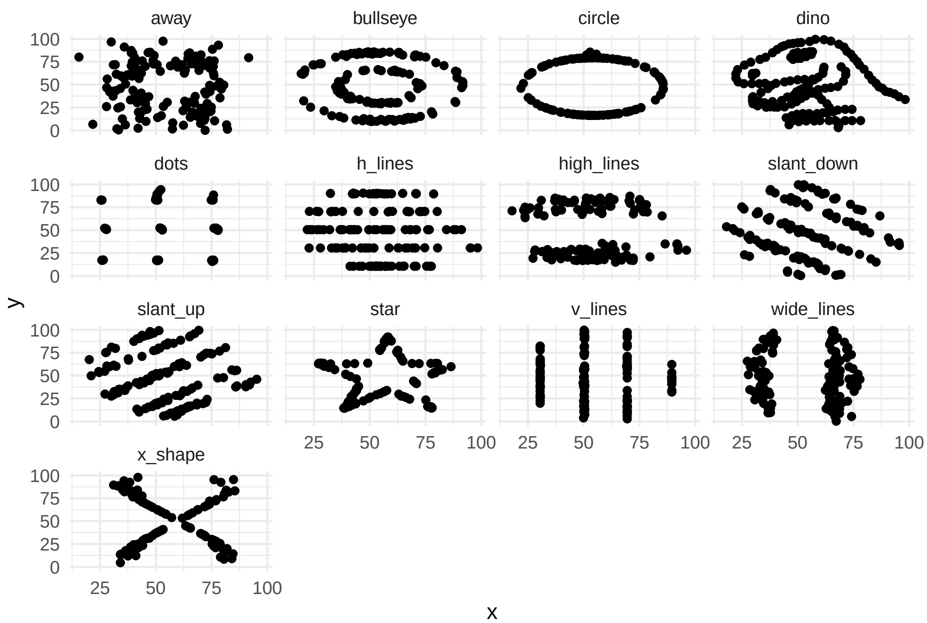
10.1.5 Faceting

Click for answer

Answer:

a. Create a scatterplot of `x` vs. `y` for each dataset in `datasaurus_dozen` using `facet_wrap()`.

```
ggplot(data = datasaurus_dozen, aes(x = x, y = y)) +
  geom_point() +
  facet_wrap(~ dataset) +
  theme_minimal()
```



10.1.6 Variable Transformation

a. The scatterplot of the `dino` dataset without any transformations is given below.

Click for answer

Answer:

```
ggplot(data = dino_data, aes(x = x, y = y)) +
  geom_point() +
  theme_minimal() -> p1
```

b. Now, apply the square root transformation to both the `x` and `y` axes using the `scale_x_sqrt()` and `scale_y_sqrt()` functions in the `dino` dataset.

Click for answer

Answer:

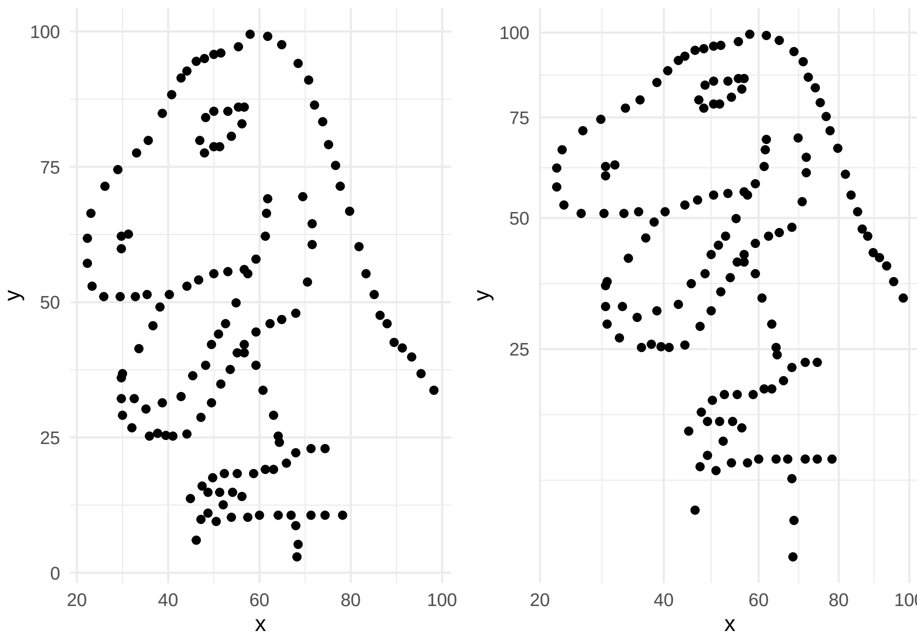
```
ggplot(data = dino_data, aes(x = x, y = y)) +
  geom_point() +
  scale_x_sqrt() +
  scale_y_sqrt() +
  theme_minimal() -> p2
```

c. Finally, use `grid.arrange()` function from `gridExtra` package to plot the above two plots side-by-side. Which plot do you prefer and why?

Click for answer

Answer: The second plot is more revealing of a dinosaur than the first plot.

```
library(gridExtra)
grid.arrange(p1, p2, nrow = 1)
```



10.1.7 (Optional) Lne plot

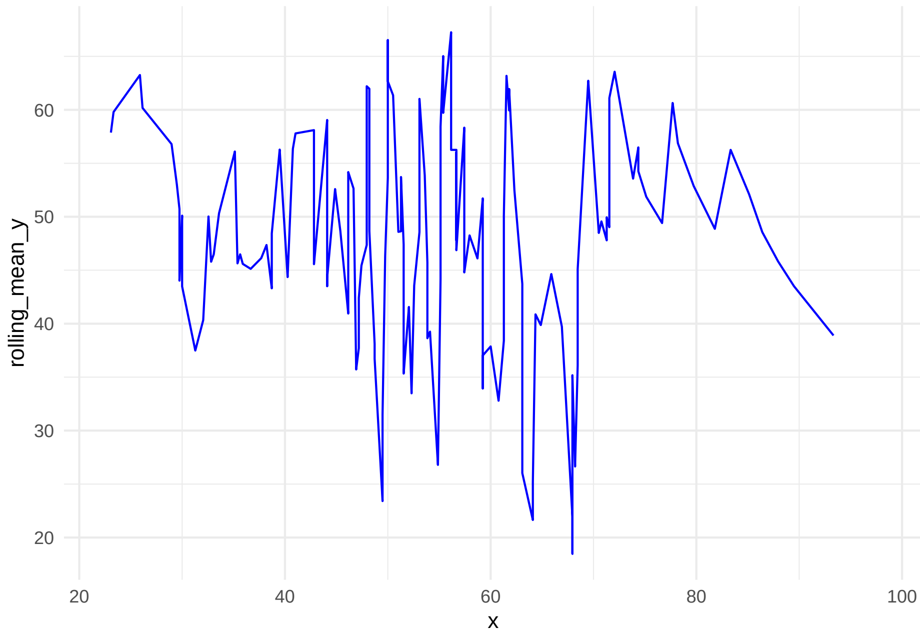
a. Create a line plot of the `x` variable over the `y` variable for the `dino` dataset. To make it more interesting, let's first calculate the rolling mean of the `y` variable.

Click for answer

Answer:

```
dino_data <- dino_data %>%
  arrange(x) %>%
  mutate(rolling_mean_y = zoo::rollmean(y, k = 5, fill = NA))
```

```r
# Line plot
ggplot(data = dino_data, aes(x = x, y = rolling_mean_y)) +
  geom_line(color = "blue") +
  theme_minimal()
```



Chapter 11

Class Activity 5

```r
# Load the required libraries
library(tidyverse)
library(ggplot2)
library(ggthemes)
```

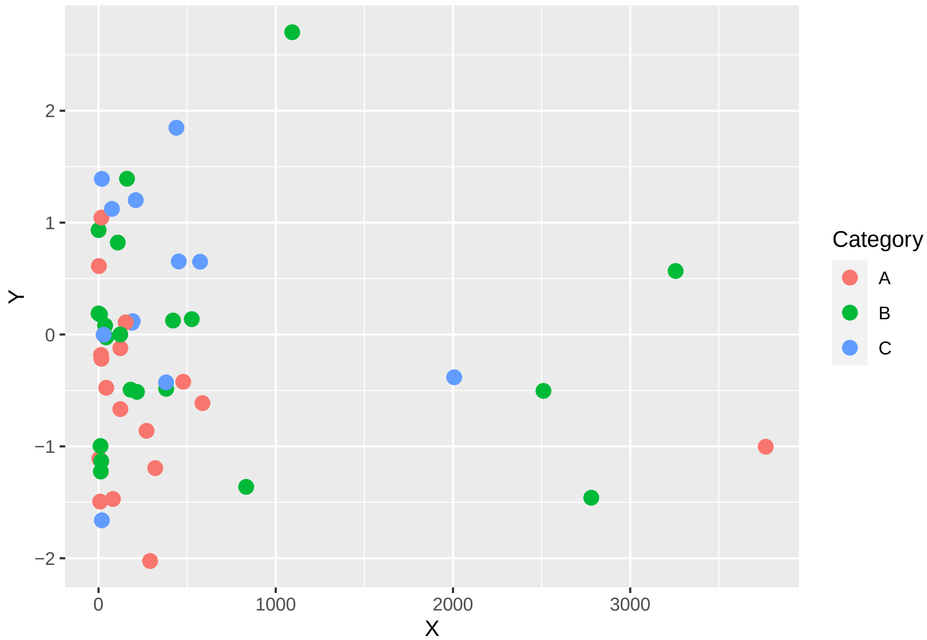
11.1 Problem 1: Changing color and shape scales

In this problem, you will learn about the effects of changing colors, scales, and shapes in ggplot2 for both gradient and discrete color choices. You will be given a series of questions and examples to enhance your understanding. Consider the following scatter plot

```r
# Generate sample data
set.seed(42)
data <- data.frame(
  Category = factor(sample(1:3, 50, replace = TRUE), labels = c("A", "B", "C")),
  X = 10 ^ rnorm(50, mean = 2, sd = 1),
  Y = rnorm(50, mean = 0, sd = 1)
)

p <- ggplot(data, aes(x = X, y = Y, color = Category)) +
  geom_point(size = 3)

p
```

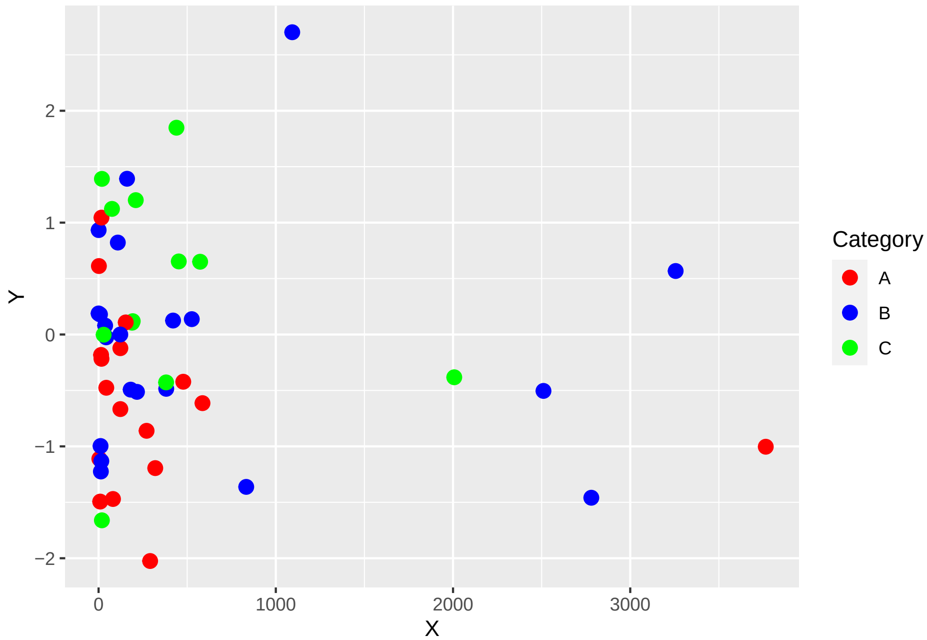


a. Modify the scatter plot to use custom colors for each category using `scale_color_manual()`. What is the effect of changing the colors on the plot's readability?

Click for answer

Answer: Changing colors using `scale_color_manual()` allows for better distinction between categories and enhances the plot's readability.

```
p <- ggplot(data, aes(x = X, y = Y, color = Category)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("A" = "red", "B" = "blue", "C" = "green"))
p
```



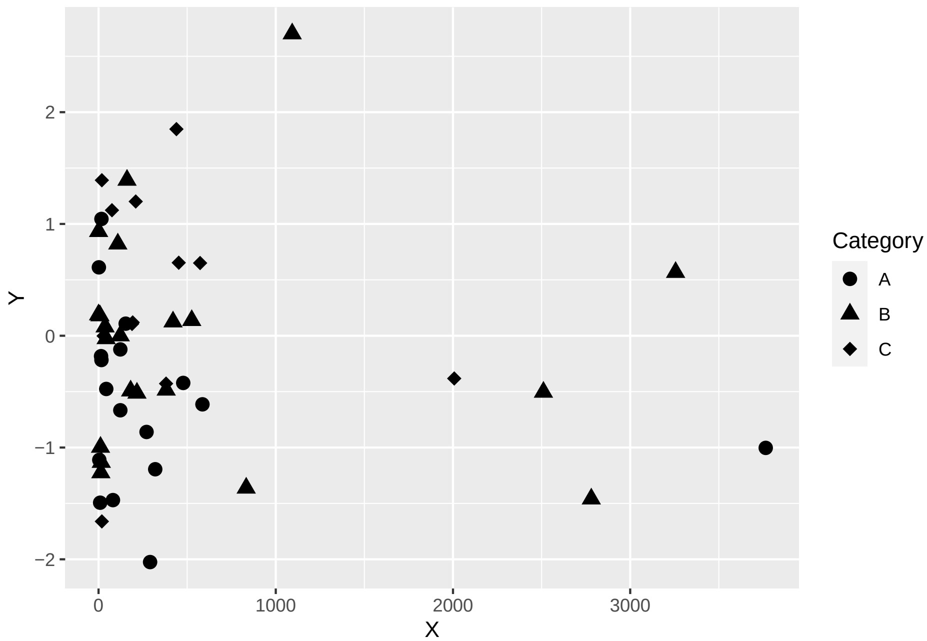
b. Modify the scatter plot to use custom shapes for each category using `scale_shape_manual()`. What is the effect of changing the shapes on the plot's readability?

Click for answer

Answer: Changing the shapes using scale_shape_manual() helps to distinguish between categories and improves the plot's readability

```
p <- ggplot(data, aes(x = X, y = Y, shape = Category)) +
  geom_point(size = 3) +
  scale_shape_manual(values = c("A" = 16, "B" = 17, "C" = 18))

p
```



11.2 Problem 2: US maps

Now, let's learn about the effect of changing various coordinate systems in `ggplot2` using a map example from the `usmap` package. We will explore the different types of coordinate systems available in ggplot2 and how they can be applied to the map visualization.

```
#install.packages("usmap")   #uncomment to install
library(usmap)
```

11.2.1 a. Plot a simple map of the United States using `ggplot2` and the `usmap` package.

Click for answer

Answer:

```
us <- plot_usmap()
us
```

11.2.2 b. Apply the `coord_flip()` function to the map to flip the x and y axes.

Click for answer

Answer:

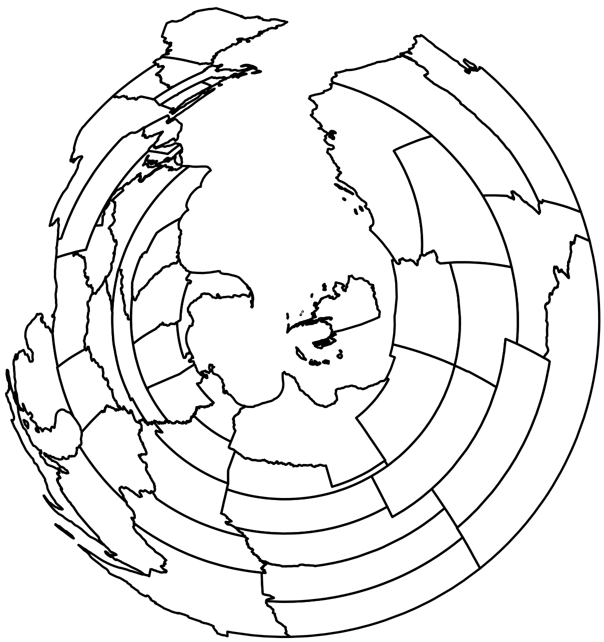
```
us_flipped <- us + coord_flip()
us_flipped
```

11.2.3 c. Apply the `coord_polar()` function to the map to transform the plot to a polar coordinate system

Click for answer

Answer:

```
us_polar <- us + coord_polar()
us_polar
```



11.2.4 d. Apply the `coord_quickmap()` function to the map to provide an approximation for a map projection.

Click for answer

Answer:

```
us_quickmap <- us + coord_quickmap()
us_quickmap
```

11.3 Problem 3: Chloropeth map

In today's class we created `cloropleth` maps of states in the US based on ACS data.

```
states <- map_data("state")
ACS <- ACS <- read.csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/ACS
ACS <- dplyr::filter(ACS, !(region  %in% c("Alaska", "Hawaii"))) # only 48+D.C.
ACS$region <- tolower(ACS$region)  # lower case (match states regions)
```

11.3.1 (a) Mapping median income

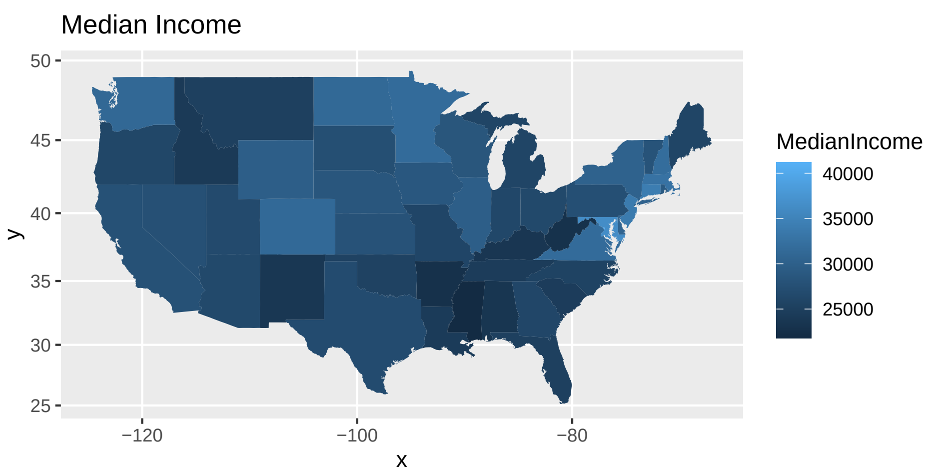
Create a `cloropleth` plot that uses color to create a `MedianIncome` map of the US.

Click for answer

Answer:

```
# map median income
ggplot(data=ACS) + coord_map() +
```

```
  geom_map(aes(map_id = region, fill = MedianIncome), map = states) +
  expand_limits(x=states$long, y=states$lat) + ggtitle("Median Income")
```



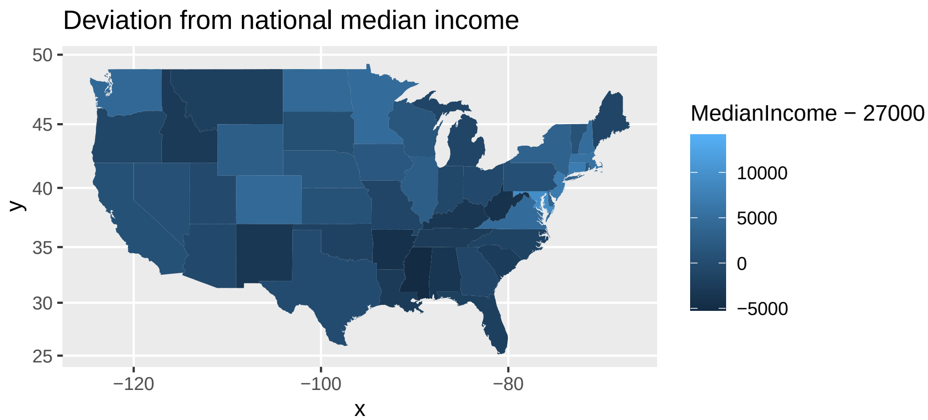
11.3.2 (b) Mapping deviations from national median income

The median income in the US in 2016 was estimated to be $27,000. Redraw your map in (a) to visualize each state's deviation from national median income.

Click for answer

Answer:

```
# compare state income to national income
ggplot(data=ACS) + coord_map() +
  geom_map(aes(map_id = region, fill = MedianIncome - 27000), map = states) +
  expand_limits(x=states$long, y=states$lat) + ggtitle("Deviation from national median income")
```



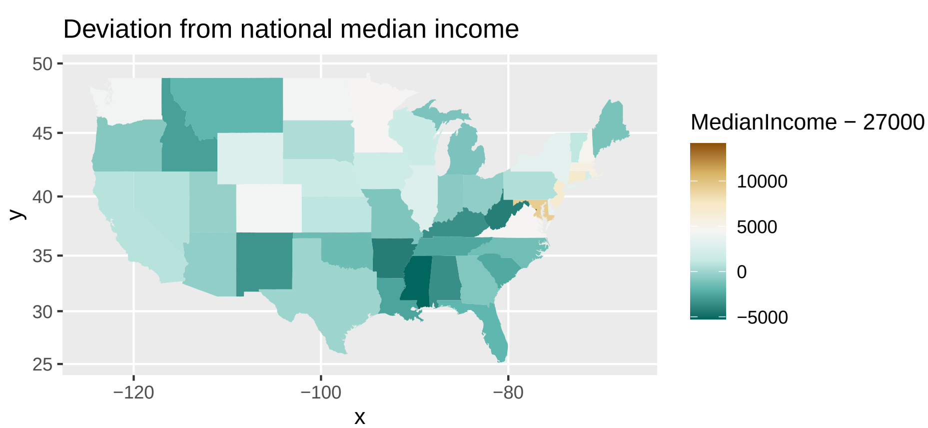
11.3.3 (c) Changing numerically scaled color

You should use a *diverging* color for (b) to highlight larger deviations from the national median. Add `scale_fill_distiller` to the map from (b) and select a diverging palette.

Click for answer

Answer:

```
# change to a diverging color
ggplot(data=ACS) + coord_map() +
  geom_map(aes(map_id = region, fill = MedianIncome - 27000), map = states) +
  expand_limits(x=states$long, y=states$lat) + ggtitle("Deviation from national median
  scale_fill_distiller(type = "div")
```



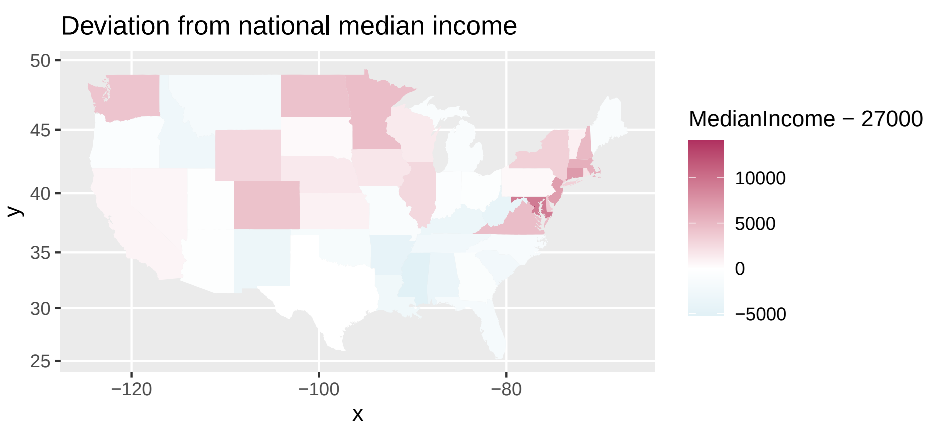
11.3.4 (d) Fixing a midpoint on a diverging scale

Use `scale_fill_gradient2` to fix a midpoint scale value at white color, with diverging colors for larger positive and negative values. Apply these colors to your map in (b) and fix the `midpoint` at an appropriate value.

Click for answer

Answer:

```
# change to a gradient fill color
ggplot(data=ACS) + coord_map() +
  geom_map(aes(map_id = region, fill = MedianIncome - 27000), map = states) +
  expand_limits(x=states$long, y=states$lat) + ggtitle("Deviation from national median income") +
  scale_fill_gradient2(
    low = "lightblue",     # Set the low color to red
    mid = "white", # Set the mid color to yellow
    high = "maroon", # Set the high color to green
    midpoint = 0
  )
```



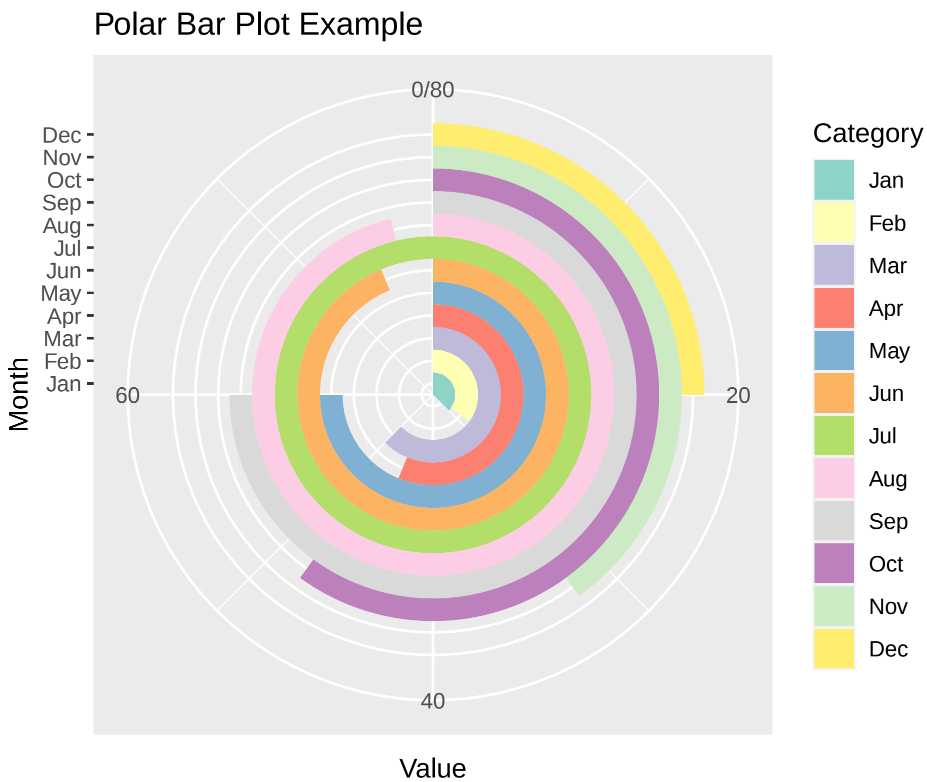
11.4 (Optional)

Let's learn how to create a polar bar plot with custom colors.

```
data <- data.frame(
  Category = factor(1:12, labels = month.abb),
  Value = c(30, 28, 50, 45, 60, 75, 80, 77, 60, 48, 32, 20)
)
```

Click for answer

```
p <- ggplot(data, aes(x = Category, y = Value, fill = Category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  scale_fill_brewer(palette = "Set3") +
  labs(title = "Polar Bar Plot Example", x = "Month", y = "Value")
p
```



Chapter 12

Class Activity 6

```r
# load the necessary libraries
library(dplyr)
library(ggplot2)
library(babynames)
```

We will work with the `babynames` dataset again in this class activity. The header of the dataset looks like this:

```r
knitr::kable(head(babynames))
```

year	sex	name	n	prop
1880	F	Mary	7065	0.0723836
1880	F	Anna	2604	0.0266790
1880	F	Emma	2003	0.0205215
1880	F	Elizabeth	1939	0.0198658
1880	F	Minnie	1746	0.0178884
1880	F	Margaret	1578	0.0161672

In this tutorial, we will learn about the five main verbs of dplyr and how to use them to manipulate data:

- `select()`: Choose columns from a data frame
- `filter()`: Choose rows based on a condition
- `arrange()`: Sort the rows of a data frame
- `mutate()`: Add new columns based on existing columns
- `summarise()`: Aggregate data and compute summary statistics

12.1 Problem 1: select()

Which of these is NOT a way to select the `name` and `n` columns together?

```
select(babynames, -c(year, sex, prop))  #1
select(babynames, name:n)   #2
select(babynames, starts_with("n")) #3
select(babynames, ends_with("n"))  #4
```

Click for answer

Answer: 4 is not the way to select the `name` and `n` columns together

12.2 Problem 2: filter()

Use `filter()` with the logical operators to extract:

12.2.1 a. All of the names where prop is greater than or equal to 0.08

Click for answer

```
filter(babynames, prop >= 0.08)
```

```
# A tibble: 3 x 5
   year sex   name        n   prop
  <dbl> <chr> <chr>   <int>  <dbl>
1  1880 M     John     9655 0.0815
2  1880 M     William  9532 0.0805
3  1881 M     John     8769 0.0810
```

```
# alternate
babynames %>% filter(prop >= 0.08)
```

```
# A tibble: 3 x 5
   year sex   name        n   prop
  <dbl> <chr> <chr>   <int>  <dbl>
1  1880 M     John     9655 0.0815
2  1880 M     William  9532 0.0805
3  1881 M     John     8769 0.0810
```

12.2.2 b. All of the babies named "Rose"

Click for answer

```
babynames %>% filter(name == "Rose")
```

```
# A tibble: 247 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     Rose    700 0.00717
 2  1880 M     Rose      7 0.0000591
 3  1881 F     Rose    734 0.00743
 4  1882 F     Rose    886 0.00766
 5  1883 F     Rose    877 0.00730
 6  1883 M     Rose      5 0.0000445
 7  1884 F     Rose   1060 0.00770
 8  1884 M     Rose      5 0.0000407
 9  1885 F     Rose   1164 0.00820
10  1885 M     Rose      9 0.0000776
# ... with 237 more rows
```

12.2.3 c. Use filter() to choose all rows where name is "John" and sex is "M".

Click for answer

```
babynames %>% filter(name == "John", sex == "M")
```

```
# A tibble: 138 x 5
    year sex   name      n   prop
   <dbl> <chr> <chr> <int>  <dbl>
 1  1880 M     John   9655 0.0815
 2  1881 M     John   8769 0.0810
 3  1882 M     John   9557 0.0783
 4  1883 M     John   8894 0.0791
 5  1884 M     John   9388 0.0765
 6  1885 M     John   8756 0.0755
 7  1886 M     John   9026 0.0758
 8  1887 M     John   8110 0.0742
 9  1888 M     John   9247 0.0712
10  1889 M     John   8548 0.0718
# ... with 128 more rows
```

12.3 Problem 3: `arrange()`

12.3.1 a. Use `arrange()` to sort the babynames dataset by the prop column in descending order.

Click for answer

```
babynames %>% arrange(desc(prop))
```

```
# A tibble: 1,924,665 x 5
    year sex   name        n   prop
   <dbl> <chr> <chr>   <int>  <dbl>
 1  1880 M     John     9655 0.0815
 2  1881 M     John     8769 0.0810
 3  1880 M     William  9532 0.0805
 4  1883 M     John     8894 0.0791
 5  1881 M     William  8524 0.0787
 6  1882 M     John     9557 0.0783
 7  1884 M     John     9388 0.0765
 8  1882 M     William  9298 0.0762
 9  1886 M     John     9026 0.0758
10  1885 M     John     8756 0.0755
# ... with 1,924,655 more rows
```

12.3.2 b. Use `arrange()` to sort the babynames dataset by year (ascending) and then by prop (descending).

Click for answer

```
babynames %>% arrange(year, desc(prop))
```

```
# A tibble: 1,924,665 x 5
    year sex   name        n   prop
   <dbl> <chr> <chr>   <int>  <dbl>
 1  1880 M     John     9655 0.0815
 2  1880 M     William  9532 0.0805
 3  1880 F     Mary     7065 0.0724
 4  1880 M     James    5927 0.0501
 5  1880 M     Charles  5348 0.0452
 6  1880 M     George   5126 0.0433
 7  1880 M     Frank    3242 0.0274
 8  1880 F     Anna     2604 0.0267
 9  1880 M     Joseph   2632 0.0222
```

```
10  1880 M     Thomas   2534 0.0214
# ... with 1,924,655 more rows
```

12.4 Problem 4: mutate()

12.4.1 a. Use mutate() to create a new column called decade which contains the decade the record is in (e.g., 1990 for the years 1990-1999).

Click for answer

```
babynames %>% mutate(decade = (year %/% 10) * 10)
```

```
# A tibble: 1,924,665 x 6
    year sex   name          n   prop decade
   <dbl> <chr> <chr>     <int>  <dbl>  <dbl>
 1  1880 F     Mary       7065 0.0724   1880
 2  1880 F     Anna       2604 0.0267   1880
 3  1880 F     Emma       2003 0.0205   1880
 4  1880 F     Elizabeth  1939 0.0199   1880
 5  1880 F     Minnie     1746 0.0179   1880
 6  1880 F     Margaret   1578 0.0162   1880
 7  1880 F     Ida        1472 0.0151   1880
 8  1880 F     Alice      1414 0.0145   1880
 9  1880 F     Bertha     1320 0.0135   1880
10  1880 F     Sarah      1288 0.0132   1880
# ... with 1,924,655 more rows
```

12.5 Problem 5: summarize() or summarise()

Use the codes mentioned so far to compute three statistics:

- the total number of children who ever had your name
- the maximum number of children given your name in a single year
- the mean number of children given your name per year/decade (optional)

Click for answer

```
babynames %>%
  filter(name == "Dee", sex == "M")
```

```
# A tibble: 136 x 5
    year sex   name      n     prop
   <dbl> <chr> <chr> <int>    <dbl>
 1  1880 M     Dee      20 0.000169
 2  1881 M     Dee      32 0.000296
 3  1882 M     Dee      23 0.000188
 4  1883 M     Dee      22 0.000196
 5  1884 M     Dee      27 0.000220
 6  1885 M     Dee      28 0.000241
 7  1886 M     Dee      26 0.000218
 8  1887 M     Dee      39 0.000357
 9  1888 M     Dee      35 0.000269
10  1889 M     Dee      24 0.000202
# ... with 126 more rows
```

```
babynames %>%
  filter(name == "Dee", sex == "M") %>%
  summarise(max_number = max(n))
```

```
# A tibble: 1 x 1
  max_number
       <int>
1        125
```

```
babynames %>%
  filter(name == "Dee", sex == "M") %>%
  mutate(decade = (year %/% 10) * 10) %>%
  group_by(decade) %>%
  summarise(total = sum(n),
            max = max(n),
            mean = mean(n))
```

```
# A tibble: 14 x 4
   decade total   max  mean
    <dbl> <int> <int> <dbl>
 1   1880   276    39  27.6
 2   1890   271    43  27.1
 3   1900   302    38  30.2
 4   1910   818   125  81.8
 5   1920  1090   125 109
 6   1930  1010   118 101
 7   1940   967   120  96.7
 8   1950   957   118  95.7
 9   1960   683   102  68.3
```

```
10    1970    380     57   38
11    1980    217     30   21.7
12    1990    130     17   13
13    2000     87     13    9.67
14    2010     52     12    7.43
```

12.6 Problem 6

12.6.1 a. Use `min_rank()` and `mutate()` to rank each row in babynames from largest prop to smallest prop.

Click for answer

```
babynames %>% mutate(rank = min_rank(desc(prop))) %>% arrange(rank)
```

```
# A tibble: 1,924,665 x 6
    year sex   name        n   prop  rank
   <dbl> <chr> <chr>   <int>  <dbl> <int>
 1  1880 M     John     9655 0.0815     1
 2  1881 M     John     8769 0.0810     2
 3  1880 M     William  9532 0.0805     3
 4  1883 M     John     8894 0.0791     4
 5  1881 M     William  8524 0.0787     5
 6  1882 M     John     9557 0.0783     6
 7  1884 M     John     9388 0.0765     7
 8  1882 M     William  9298 0.0762     8
 9  1886 M     John     9026 0.0758     9
10  1885 M     John     8756 0.0755    10
# ... with 1,924,655 more rows
```

12.6.2 b. Compute each name's rank within its year and sex.

Click for answer

```
babynames %>% group_by(year, sex) %>% mutate(rank = min_rank(desc(prop)))
```

```
# A tibble: 1,924,665 x 6
# Groups:   year, sex [276]
    year sex   name          n   prop  rank
   <dbl> <chr> <chr>     <int>  <dbl> <int>
```

```
 1  1880 F     Mary       7065 0.0724     1
 2  1880 F     Anna       2604 0.0267     2
 3  1880 F     Emma       2003 0.0205     3
 4  1880 F     Elizabeth  1939 0.0199     4
 5  1880 F     Minnie     1746 0.0179     5
 6  1880 F     Margaret   1578 0.0162     6
 7  1880 F     Ida        1472 0.0151     7
 8  1880 F     Alice      1414 0.0145     8
 9  1880 F     Bertha     1320 0.0135     9
10  1880 F     Sarah      1288 0.0132    10
# ... with 1,924,655 more rows
```

12.6.3 c. Then compute the median rank for each combination of name and sex, and arrange the results from highest median rank to lowest.

Click for answer

```
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  group_by(name, sex) %>%
  summarize(score = median(rank)) %>%
  arrange(score)
```

```
# A tibble: 107,973 x 3
# Groups:   name [97,310]
   name      sex   score
   <chr>     <chr> <dbl>
 1 Mary      F       1
 2 James     M       3
 3 John      M       3
 4 William   M       4
 5 Robert    M       6
 6 Michael   M       7.5
 7 Charles   M       9
 8 Elizabeth F      10
 9 Joseph    M      10
10 Thomas    M      11
# ... with 107,963 more rows
```

Chapter 13

Class Activity 7

```
# load the necessary libraries
library(tidyverse)
library(babynames)
```

13.1 Problem 1: Boolean Operators

Use Boolean operators to alter the code below to return only the rows that contain:

13.1.1 a. Girls named Rhea

Click for answer

```
filter(babynames, name == "Rhea", sex == "F")
```

```
# A tibble: 136 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1882 F     Rhea      7 0.0000605
 2  1883 F     Rhea      8 0.0000666
 3  1884 F     Rhea     13 0.0000945
 4  1885 F     Rhea     11 0.0000775
 5  1886 F     Rhea     13 0.0000846
 6  1887 F     Rhea     14 0.0000901
 7  1888 F     Rhea     20 0.000106
```

```
 8   1889 F     Rhea     31 0.000164
 9   1890 F     Rhea     39 0.000193
10   1891 F     Rhea     24 0.000122
# ... with 126 more rows
```

```
babynames %>%  filter(name == "Rhea", sex == "F")
```

```
# A tibble: 136 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1882 F     Rhea      7 0.0000605
 2  1883 F     Rhea      8 0.0000666
 3  1884 F     Rhea     13 0.0000945
 4  1885 F     Rhea     11 0.0000775
 5  1886 F     Rhea     13 0.0000846
 6  1887 F     Rhea     14 0.0000901
 7  1888 F     Rhea     20 0.000106
 8  1889 F     Rhea     31 0.000164
 9  1890 F     Rhea     39 0.000193
10  1891 F     Rhea     24 0.000122
# ... with 126 more rows
```

13.1.2 b. Names that were used by exactly 5 or 6 children in 1990

Click for answer

```
filter(babynames, year == 1990, n == 5 | n == 6)
```

```
# A tibble: 6,144 x 5
    year sex   name         n       prop
   <dbl> <chr> <chr>    <int>      <dbl>
 1  1990 F     Aariel       6 0.00000292
 2  1990 F     Aarion       6 0.00000292
 3  1990 F     Abagael      6 0.00000292
 4  1990 F     Abbye        6 0.00000292
 5  1990 F     Abiola       6 0.00000292
 6  1990 F     Abreanna     6 0.00000292
 7  1990 F     Abygail      6 0.00000292
 8  1990 F     Acadia       6 0.00000292
 9  1990 F     Adilenne     6 0.00000292
10  1990 F     Adriena      6 0.00000292
# ... with 6,134 more rows
```

```
babynames %>% filter(year == "1990", n == 5 | n == 6)
```

```
# A tibble: 6,144 x 5
    year sex   name          n       prop
   <dbl> <chr> <chr>     <int>      <dbl>
 1  1990 F     Aariel        6 0.00000292
 2  1990 F     Aarion        6 0.00000292
 3  1990 F     Abagael       6 0.00000292
 4  1990 F     Abbye         6 0.00000292
 5  1990 F     Abiola        6 0.00000292
 6  1990 F     Abreanna      6 0.00000292
 7  1990 F     Abygail       6 0.00000292
 8  1990 F     Acadia        6 0.00000292
 9  1990 F     Adilenne      6 0.00000292
10  1990 F     Adriena       6 0.00000292
# ... with 6,134 more rows
```

13.1.3 c. Names that are one of Apple, Yoroi, Ada

Click for answer

```
filter(babynames, name == "Apple" | name == "Yoroi" | name == "Ada")
```

```
# A tibble: 200 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     Ada     652 0.00668
 2  1881 F     Ada     628 0.00635
 3  1882 F     Ada     689 0.00596
 4  1883 F     Ada     778 0.00648
 5  1884 F     Ada     854 0.00621
 6  1885 F     Ada     876 0.00617
 7  1885 M     Ada       5 0.0000431
 8  1886 F     Ada     915 0.00595
 9  1886 M     Ada       6 0.0000504
10  1887 F     Ada     910 0.00586
# ... with 190 more rows
```

13.1.4 d. Store the data tibble in part c into a new tibble and change all the character columns to upper case. Also, rename the `n` variable to `count`.

Click for answer

```
aya <- babynames %>% filter(name == "Apple" | name == "Yoroi" | name == "Ada")
aya %>% mutate_if(is.character, toupper)
```

```
# A tibble: 200 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     ADA     652 0.00668
 2  1881 F     ADA     628 0.00635
 3  1882 F     ADA     689 0.00596
 4  1883 F     ADA     778 0.00648
 5  1884 F     ADA     854 0.00621
 6  1885 F     ADA     876 0.00617
 7  1885 M     ADA       5 0.0000431
 8  1886 F     ADA     915 0.00595
 9  1886 M     ADA       6 0.0000504
10  1887 F     ADA     910 0.00586
# ... with 190 more rows
```

```
aya %>% mutate_at(vars(name), toupper)
```

```
# A tibble: 200 x 5
    year sex   name      n      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     ADA     652 0.00668
 2  1881 F     ADA     628 0.00635
 3  1882 F     ADA     689 0.00596
 4  1883 F     ADA     778 0.00648
 5  1884 F     ADA     854 0.00621
 6  1885 F     ADA     876 0.00617
 7  1885 M     ADA       5 0.0000431
 8  1886 F     ADA     915 0.00595
 9  1886 M     ADA       6 0.0000504
10  1887 F     ADA     910 0.00586
# ... with 190 more rows
```

```
aya %>% rename(count = n)
```

```
# A tibble: 200 x 5
    year sex   name  count      prop
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     Ada     652 0.00668
 2  1881 F     Ada     628 0.00635
 3  1882 F     Ada     689 0.00596
```

```
 4  1883 F     Ada     778 0.00648
 5  1884 F     Ada     854 0.00621
 6  1885 F     Ada     876 0.00617
 7  1885 M     Ada       5 0.0000431
 8  1886 F     Ada     915 0.00595
 9  1886 M     Ada       6 0.0000504
10  1887 F     Ada     910 0.00586
# ... with 190 more rows
```

13.1.5 e. Change all the column names to upper case in the previous problem.

Click for answer

```
aya %>% rename_at(vars(year:prop), toupper)
```

```
# A tibble: 200 x 5
    YEAR SEX   NAME      N      PROP
   <dbl> <chr> <chr> <int>     <dbl>
 1  1880 F     Ada     652 0.00668
 2  1881 F     Ada     628 0.00635
 3  1882 F     Ada     689 0.00596
 4  1883 F     Ada     778 0.00648
 5  1884 F     Ada     854 0.00621
 6  1885 F     Ada     876 0.00617
 7  1885 M     Ada       5 0.0000431
 8  1886 F     Ada     915 0.00595
 9  1886 M     Ada       6 0.0000504
10  1887 F     Ada     910 0.00586
# ... with 190 more rows
```

13.1.6 f. What do these commands do?

```
polluted_cities <- tribble(
       ~city,   ~size, ~amount,
  "New York", "large",      23,
  "New York", "small",      14,
    "London", "large",      22,
    "London", "small",      16,
   "Beijing", "large",     121,
   "Beijing", "small",      56
)
```

```
polluted_cities
```

```
# A tibble: 6 x 3
  city     size  amount
  <chr>    <chr>  <dbl>
1 New York large     23
2 New York small     14
3 London   large     22
4 London   small     16
5 Beijing  large    121
6 Beijing  small     56
```

```
polluted_cities %>% select_if(is.numeric) #1
polluted_cities %>% rename_all(toupper) #2
polluted_cities %>% rename_if(is.character, toupper) #3
polluted_cities %>% rename_at(vars(contains("it")), toupper) #4
```

Click for answer

answer:

1. Selects all numeric columns from the polluted_cities dataset.
2. Renames all column names in the polluted_cities dataset to uppercase.
3. Renames column names with character data type in the polluted_cities dataset to uppercase.
4. Renames column names containing "it" in the polluted_cities dataset to uppercase.

```
polluted_cities %>% select_if(is.numeric) #1
```

```
# A tibble: 6 x 1
  amount
   <dbl>
1     23
2     14
3     22
4     16
5    121
6     56
```

```
polluted_cities %>% rename_all(toupper) #2
```

```
# A tibble: 6 x 3
  CITY     SIZE  AMOUNT
  <chr>    <chr>  <dbl>
1 New York large     23
2 New York small     14
3 London   large     22
4 London   small     16
5 Beijing  large    121
6 Beijing  small     56
```

```
polluted_cities %>% rename_if(is.character, toupper) #3
```

```
# A tibble: 6 x 3
  CITY     SIZE  amount
  <chr>    <chr>  <dbl>
1 New York large     23
2 New York small     14
3 London   large     22
4 London   small     16
5 Beijing  large    121
6 Beijing  small     56
```

```
polluted_cities %>% rename_at(vars(contains("it")), toupper) #4
```

```
# A tibble: 6 x 3
  CITY     size  amount
  <chr>    <chr>  <dbl>
1 New York large     23
2 New York small     14
3 London   large     22
4 London   small     16
5 Beijing  large    121
6 Beijing  small     56
```

Let's look at an interesting example on how to join related information on various artists, bands, songs, and their labels.

```
artists <- tibble(first = c("Jimmy", "George", "Mick", "Tom", "Davy", "John",
                            "Paul", "Jimmy", "Joe", "Elvis", "Keith", "Paul",
                            "Ringo", "Joe", "Brian", "Nancy"),
                  last = c("Buffett", "Harrison", "Jagger", "Jones", "Jones",
```

```
                            "Lennon", "McCartney", "Page", "Perry", "Presley",
                            "Richards", "Simon", "Starr", "Walsh", "Wilson", "Wilson"),
                   instrument = c("Guitar", "Guitar", "Vocals", "Vocals", "Vocals",
                                  "Guitar", "Bass", "Guitar", "Guitar", "Vocals", "Guita
                                  "Guitar", "Drums", "Guitar", "Vocals", "Vocals"))


bands <- tibble(first = c("John", "John Paul", "Jimmy", "Robert", "George", "John",
                           "Paul", "Ringo", "Jimmy", "Mick", "Keith", "Charlie", "Ronni
                last = c("Bonham", "Jones", "Page", "Plant", "Harrison", "Lennon",
                          "McCartney", "Starr", "Buffett", "Jagger", "Richards", "Watts'
                band = c("Led Zeppelin", "Led Zeppelin", "Led Zeppelin", "Led Zeppelin'
                          "The Beatles", "The Beatles", "The Beatles", "The Beatles",
                          "The Coral Reefers", "The Rolling Stones", "The Rolling Stone:
                          "The Rolling Stones", "The Rolling Stones"))

albums <- tibble(album = c("A Hard Day's Night", "Magical Mystery Tour", "Beggar's Banc
                            "Abbey Road", "Led Zeppelin IV", "The Dark Side of the Moon'
                            "Rumours", "Hotel California"),
                 band = c("The Beatles", "The Beatles", "The Rolling Stones", "The Bea
                           "Led Zeppelin", "Pink Floyd", "Aerosmith", "Fleetwood Mac", '
                 year = c(1964,1967,1968,1969,1971,1973,1973,1977,1982))


songs <- tibble(song = c("Come Together", "Dream On", "Hello, Goodbye", "It's Not Unusu
                album  = c("Abbey Road", "Aerosmith", "Magical Mystery Tour", "Along Ca
                first = c("John", "Steven", "Paul", "Tom"),
                last = c("Lennon", "Tyler", "McCartney", "Jones"))


labels <- tibble(album = c("Abbey Road", "A Hard Days Night", "Magical Mystery Tour",
                            "Led Zeppelin IV", "The Dark Side of the Moon", "Hotel Cali
                            "Rumours", "Aerosmith", "Beggar's Banquet"),
                 label = c("Apple", "Parlophone", "Parlophone", "Atlantic", "Harvest",
                            "Asylum", "Warner Brothers", "Columbia", "Decca"))
```

Let's take a glimpse of the tibbles `artists` and `bands`. Notice that there are different number of rows in the dataset.

```
glimpse(artists)
```

```
Rows: 16
Columns: 3
$ first      <chr> "Jimmy", "George", "Mick", "Tom", "Davy~
```

```
$ last       <chr> "Buffett", "Harrison", "Jagger", "Jones~
$ instrument <chr> "Guitar", "Guitar", "Vocals", "Vocals",~
```

```
glimpse(bands)
```

```
Rows: 13
Columns: 3
$ first <chr> "John", "John Paul", "Jimmy", "Robert", "Geo~
$ last  <chr> "Bonham", "Jones", "Page", "Plant", "Harriso~
$ band  <chr> "Led Zeppelin", "Led Zeppelin", "Led Zeppeli~
```

```
glimpse(albums)
```

```
Rows: 9
Columns: 3
$ album <chr> "A Hard Day's Night", "Magical Mystery Tour"~
$ band  <chr> "The Beatles", "The Beatles", "The Rolling S~
$ year  <dbl> 1964, 1967, 1968, 1969, 1971, 1973, 1973, 19~
```

```
glimpse(songs)
```

```
Rows: 4
Columns: 4
$ song  <chr> "Come Together", "Dream On", "Hello, Goodbye~
$ album <chr> "Abbey Road", "Aerosmith", "Magical Mystery ~
$ first <chr> "John", "Steven", "Paul", "Tom"
$ last  <chr> "Lennon", "Tyler", "McCartney", "Jones"
```

```
glimpse(labels)
```

```
Rows: 9
Columns: 2
$ album <chr> "Abbey Road", "A Hard Days Night", "Magical ~
$ label <chr> "Apple", "Parlophone", "Parlophone", "Atlant~
```

13.2 Problem 2: Joining artists and bands data

13.2.1 a. Join the artists and bands tibbles using `left_join()`, `right_join()`, and `full_join()`. Verify that the datasets obtained from `left_join()` and `right_join()` are the same using `setequal()`.

Click for answer

```
bands2 <- left_join(bands, artists, by = c("first", "last"))
bands3 <- right_join(artists, bands, by = c("first", "last"))
full_bands <- full_join(artists, bands, by = c("first", "last"))

# Check if the datasets are the same
setequal(bands2, bands3)
```

```
[1] TRUE
```

13.2.2 b. Use the pipe operator, %>%, to create one table that combines all information from `artists`, `bands`, `albums`, `songs`, and `labels`.

Click for answer

```
all_info <- artists %>%
  full_join(bands, by = c("first", "last")) %>%
  full_join(songs, by = c("first", "last")) %>%
  full_join(albums, by = c("album", "band")) %>%
  full_join(labels, by = c("album"))
all_info
```

```
# A tibble: 30 x 8
   first  last      instrument band  song  album  year label
   <chr>  <chr>     <chr>      <chr> <chr> <chr> <dbl> <chr>
 1 Jimmy  Buffett   Guitar     The ~ <NA>  <NA>     NA <NA>
 2 George Harrison  Guitar     The ~ <NA>  <NA>     NA <NA>
 3 Mick   Jagger    Vocals     The ~ <NA>  <NA>     NA <NA>
 4 Tom    Jones     Vocals     <NA>  It's~ Alon~    NA <NA>
 5 Davy   Jones     Vocals     <NA>  <NA>  <NA>     NA <NA>
 6 John   Lennon    Guitar     The ~ Come~ Abbe~  1969 Apple
 7 Paul   McCartney Bass       The ~ Hell~ Magi~  1967 Parl~
 8 Jimmy  Page      Guitar     Led ~ <NA>  <NA>     NA <NA>
 9 Joe    Perry     Guitar     <NA>  <NA>  <NA>     NA <NA>
10 Elvis  Presley   Vocals     <NA>  <NA>  <NA>     NA <NA>
# ... with 20 more rows
```

13.3 Problem 3: Filtering and counting rows in the data

13.3.1 a. Collect artists that have songs provided, and return rows of artists that don't have bands info.

Click for answer

```
# Artists with songs
artists_with_songs <- artists %>%
  semi_join(songs, by = c("first", "last"))

# Artists without bands info
artists_without_bands <- artists %>%
  anti_join(bands, by = c("first","last"))

artists_with_songs
```

```
# A tibble: 3 x 3
  first last      instrument
  <chr> <chr>     <chr>
1 Tom   Jones     Vocals
2 John  Lennon    Guitar
3 Paul  McCartney Bass
```

```
artists_without_bands
```

```
# A tibble: 8 x 3
  first last    instrument
  <chr> <chr>   <chr>
1 Tom   Jones   Vocals
2 Davy  Jones   Vocals
3 Joe   Perry   Guitar
4 Elvis Presley Vocals
5 Paul  Simon   Guitar
6 Joe   Walsh   Guitar
7 Brian Wilson  Vocals
8 Nancy Wilson  Vocals
```

13.3.2 b. Collect the albums made by a band, count the number of rows, find the rows of songs that match a row in labels, and count the number of rows.

Click for answer

```r
# Albums made by a band
albums_by_band <- albums %>% semi_join(bands, by = "band")
n_albums_by_band <- nrow(albums_by_band)

# Rows of songs that match a row in labels
songs_with_labels <- songs %>% semi_join(labels, by = "album")
n_songs_with_labels <- nrow(songs_with_labels)

n_albums_by_band
```

```
[1] 5
```

```r
n_songs_with_labels
```

```
[1] 3
```

Chapter 14

Class Activity 8

```r
# load the necessary libraries
library(tidyverse)
library(lubridate)
```

14.1 Your turn 1

The following code chinks create the data objects used in this exercise. Please run the chunks and answer the questions.

```r
DBP_wide <- tibble(id = letters[1:4],
 sex = c("F", "M", "M", "F"),
 v1.DBP = c(88, 84, 102, 70),
 v2.DBP = c(78, 78, 96, 76),
 v3.DBP = c(94, 82, 94, 74),
 age=c(23, 56, 41, 38)
 )
DBP_wide
```

```
# A tibble: 4 x 6
  id    sex   v1.DBP v2.DBP v3.DBP   age
  <chr> <chr>  <dbl>  <dbl>  <dbl> <dbl>
1 a     F         88     78     94    23
2 b     M         84     78     82    56
3 c     M        102     96     94    41
4 d     F         70     76     74    38
```

```
BP_wide <- tibble(id = letters[1:4],
                  sex = c("F", "M", "M", "F"),
                  SBP_v1 = c(130, 120, 130, 119),
                  SBP_v2 = c(110, 116, 136, 106),
                  SBP_v3 = c(112, 122, 138, 118))
BP_wide
```

```
# A tibble: 4 x 5
  id    sex   SBP_v1 SBP_v2 SBP_v3
  <chr> <chr>  <dbl>  <dbl>  <dbl>
1 a     F        130    110    112
2 b     M        120    116    122
3 c     M        130    136    138
4 d     F        119    106    118
```

```
BP_long <- BP_wide %>%
  pivot_longer(names_to = "visit", values_to = "SBP", SBP_v1:SBP_v3) %>%
  mutate(visit = parse_number(visit))
BP_long
```

```
# A tibble: 12 x 4
   id    sex   visit   SBP
   <chr> <chr> <dbl> <dbl>
 1 a     F         1   130
 2 a     F         2   110
 3 a     F         3   112
 4 b     M         1   120
 5 b     M         2   116
 6 b     M         3   122
 7 c     M         1   130
 8 c     M         2   136
 9 c     M         3   138
10 d     F         1   119
11 d     F         2   106
12 d     F         3   118
```

14.1.1 a. Create a long dataframe from `DBP_wide` based on the repeated DBP columns and save it as `DBP_long`.

Click for answer

Answer:

```
DBP_long <- DBP_wide %>%
  pivot_longer(names_to = "visit",
               values_to = "DBP",
               cols = v1.DBP:v3.DBP)
DBP_long
```

```
# A tibble: 12 x 5
   id    sex     age visit    DBP
   <chr> <chr> <dbl> <chr>  <dbl>
 1 a     F        23 v1.DBP    88
 2 a     F        23 v2.DBP    78
 3 a     F        23 v3.DBP    94
 4 b     M        56 v1.DBP    84
 5 b     M        56 v2.DBP    78
 6 b     M        56 v3.DBP    82
 7 c     M        41 v1.DBP   102
 8 c     M        41 v2.DBP    96
 9 c     M        41 v3.DBP    94
10 d     F        38 v1.DBP    70
11 d     F        38 v2.DBP    76
12 d     F        38 v3.DBP    74
```

14.1.2 b. Clean up the visit column of `DBP_long` so that the values are 1, 2, 3, and save it as `DBP_long`.

Click for answer

Answer:

```
DBP_long <- DBP_long %>%
  mutate(visit = parse_number(visit))
DBP_long
```

```
# A tibble: 12 x 5
   id    sex     age visit   DBP
   <chr> <chr> <dbl> <dbl> <dbl>
 1 a     F        23     1    88
 2 a     F        23     2    78
 3 a     F        23     3    94
 4 b     M        56     1    84
 5 b     M        56     2    78
 6 b     M        56     3    82
 7 c     M        41     1   102
 8 c     M        41     2    96
```

```
 9 c     M         41      3     94
10 d     F         38      1     70
11 d     F         38      2     76
12 d     F         38      3     74
```

14.1.3 c. Make `DBP_long` wide with column names visit.1, visit.2, visit.3 for the DBP values, and save it as `DBP_wide2`

Click for answer

Answer:

```
DBP_wide2 <- DBP_long %>%
  pivot_wider(names_from = "visit",
              values_from = "DBP",
              names_prefix = "visit.")
DBP_wide2
```

```
# A tibble: 4 x 6
  id    sex     age visit.1 visit.2 visit.3
  <chr> <chr> <dbl>   <dbl>   <dbl>   <dbl>
1 a     F        23      88      78      94
2 b     M        56      84      78      82
3 c     M        41     102      96      94
4 d     F        38      70      76      74
```

14.1.4 d. Join `DBP_long` with `BP_long2` to create a single data frame with columns id, sex, visit, SBP, DBP, and age. Save this as `BP_both_long`.

Click for answer

Answer:

```
BP_both_long <- left_join(BP_long, DBP_long, by = c("id", "sex", "visit"))
BP_both_long
```

```
# A tibble: 12 x 6
   id    sex   visit   SBP   age   DBP
   <chr> <chr> <dbl> <dbl> <dbl> <dbl>
 1 a     F         1   130    23    88
 2 a     F         2   110    23    78
```

```
 3 a     F          3    112     23     94
 4 b     M          1    120     56     84
 5 b     M          2    116     56     78
 6 b     M          3    122     56     82
 7 c     M          1    130     41    102
 8 c     M          2    136     41     96
 9 c     M          3    138     41     94
10 d     F          1    119     38     70
11 d     F          2    106     38     76
12 d     F          3    118     38     74
```

14.1.5 e. Calculate the mean SBP and DBP for each visit and save the result as `mean_BP_by_visit`.

Click for answer

Answer:

```
mean_BP_by_visit <- BP_both_long %>%
  group_by(visit) %>%
  summarize(mean_SBP = mean(SBP),
            mean_DBP = mean(DBP))
mean_BP_by_visit
```

```
# A tibble: 3 x 3
  visit mean_SBP mean_DBP
  <dbl>    <dbl>    <dbl>
1     1     125.       86
2     2     117        82
3     3     122.       86
```

14.2 Your turn 2

14.2.1 a. Parsing Complex Dates: Use `dmy_hms()` to parse the following date-time string: "25-Dec-2020 17:30:00"

Click for answer

Answer:

```r
parsed_date <- dmy_hms("25-Dec-2020 17:30:00")
parsed_date
```

```
[1] "2020-12-25 17:30:00 UTC"
```

14.2.2 b. Advanced Date Arithmetic: Calculate the exact age in years for someone born on "1995-05-15 09:30:00".

Click for answer

Answer:

```r
dob <- ymd_hms("1995-05-15 09:30:00")
exact_age <- as.duration(interval(dob, now())) / dyears(1)
exact_age
```

```
[1] 28.01825
```

14.2.3 c. Creating Date-Time Objects: Create a date-time object for March 15, 2020, 13:30:00 using `make_datetime()`.

Click for answer

Answer:

```r
new_date_time <- make_datetime(2020, 3, 15, 13, 30, 0)
new_date_time
```

```
[1] "2020-03-15 13:30:00 UTC"
```

14.2.4 d. Extracting Components from Date-Time Objects: Extract the year, month (as a number), day, hour, and minute from "2022-07-01 14:45:00".

Click for answer

Answer:

```
example_date_time <- ymd_hms("2022-07-01 14:45:00")
extracted_components <- tibble(
  year = year(example_date_time),
  month = month(example_date_time),
  day = day(example_date_time),
  hour = hour(example_date_time),
  minute = minute(example_date_time)
)
extracted_components
```

```
# A tibble: 1 x 5
   year month   day  hour minute
  <dbl> <dbl> <int> <int>  <int>
1  2022     7     1    14     45
```

14.2.5 e. Advanced Date-Time Arithmetic with Periods: Add 2 months and 15 days to "2021-08-01".

Click for answer

Answer:

```
initial_date <- ymd("2021-08-01")
new_date <- initial_date + months(2) + days(15)
new_date
```

```
[1] "2021-10-16"
```

14.2.6 f. Duration and Time Differences: Calculate the duration in days, weeks, months, and years between "2019-04-01" and "2022-04-01".

Click for answer

Answer:

```
start_date <- ymd("2019-04-01")
end_date <- ymd("2022-04-01")
time_diff <- end_date - start_date
duration_days <- as.duration(time_diff)
duration_weeks <- duration_days / dweeks(1)
duration_months <- duration_days / dmonths(1)
duration_years <- duration_days / dyears(1)
```

```
duration_results <- tibble(
  days = duration_days,
  weeks = duration_weeks,
  months = duration_months,
  years = duration_years
)
duration_results
```

```
# A tibble: 1 x 4
  days                 weeks months years
  <Duration>           <dbl>  <dbl> <dbl>
1 94694400s (~3 years)  157.   36.0  3.00
```

Chapter 15

Class Activity 9

```r
# load the necessary libraries
library(tidyverse)
```

15.1 Your Turn 1

15.1.1 a) `read_csv()`

Use `read_csv()` to import the `desserts` data set from https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv. Use `glimpse` to see if the data import is alright.

```r
url <- "https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv"
desserts <- read_csv(url)
glimpse(desserts)
```

```
Rows: 549
Columns: 16
$ series                <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ episode               <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ baker                 <chr> "Annetha", "David", "Edd", "~
$ technical             <chr> "2nd", "3rd", "1st", "N/A", ~
$ result                <chr> "IN", "IN", "IN", "IN", "IN"~
$ uk_airdate            <chr> "17 August 2010", "17 August~
$ us_season             <dbl> NA, NA, NA, NA, NA, NA, NA, ~
$ us_airdate            <date> NA, NA, NA, NA, NA, NA, NA,~
$ showstopper_chocolate <chr> "chocolate", "chocolate", "n~
```

```
$ showstopper_dessert   <chr> "other", "other", "other", "~
$ showstopper_fruit     <chr> "no fruit", "no fruit", "no ~
$ showstopper_nut       <chr> "no nut", "no nut", "no nut"~
$ signature_chocolate   <chr> "no chocolate", "chocolate",~
$ signature_dessert     <chr> "cake", "cake", "cake", "cak~
$ signature_fruit       <chr> "no fruit", "fruit", "fruit"~
$ signature_nut         <chr> "no nut", "no nut", "no nut"~
```

15.1.2 b) Are there any issues with the data import? If so, what are they?

Click for answer

Answer: Based on the output of glimpse, we can see that the 'technical' column should be a numeric column and the 'uk_airdate' column should be a date column. We can also identify any issues with missing values.

```
# your r-code

desserts <- read_csv(url,
  col_types = list(
    technical = col_number(),
    uk_airdate = col_date()
  )
)

problems(desserts)
```

```
# A tibble: 556 x 5
     row   col expected        actual         file
   <int> <int> <chr>           <chr>          <chr>
 1     2     6 date in ISO8601 17 August 2010 ""
 2     3     6 date in ISO8601 17 August 2010 ""
 3     4     6 date in ISO8601 17 August 2010 ""
 4     5     4 a number        N/A            ""
 5     5     6 date in ISO8601 17 August 2010 ""
 6     6     6 date in ISO8601 17 August 2010 ""
 7     7     4 a number        N/A            ""
 8     7     6 date in ISO8601 17 August 2010 ""
 9     8     6 date in ISO8601 17 August 2010 ""
10     9     4 a number        N/A            ""
# ... with 546 more rows
```

15.1.3 c) Import the dataset with correct data types, if needed. Fix the problems, if any.

Click for answer

```
desserts <- read_csv(url,
  col_types = list(
    technical = col_number(),
    uk_airdate = col_date()
  )
)

problems(desserts)
```

```
# A tibble: 556 x 5
     row   col expected        actual         file
   <int> <int> <chr>           <chr>          <chr>
 1     2     6 date in ISO8601 17 August 2010 ""
 2     3     6 date in ISO8601 17 August 2010 ""
 3     4     6 date in ISO8601 17 August 2010 ""
 4     5     4 a number        N/A            ""
 5     5     6 date in ISO8601 17 August 2010 ""
 6     6     6 date in ISO8601 17 August 2010 ""
 7     7     4 a number        N/A            ""
 8     7     6 date in ISO8601 17 August 2010 ""
 9     8     6 date in ISO8601 17 August 2010 ""
10     9     4 a number        N/A            ""
# ... with 546 more rows
```

```
desserts <- read_csv(url,
    col_types = list(
    technical = col_number(),
    uk_airdate = col_date(format = "%d %B %Y")
  )
)

problems(desserts)
```

```
# A tibble: 7 x 5
    row   col expected actual file
  <int> <int> <chr>    <chr>  <chr>
1     5     4 a number N/A    ""
2     7     4 a number N/A    ""
3     9     4 a number N/A    ""
```

```
4     11      4 a number N/A    ""
5     35      4 a number N/A    ""
6     36      4 a number N/A    ""
7     37      4 a number N/A    ""
```

```
desserts <- read_csv(url,
  col_types = list(
    technical = col_number(),
    uk_airdate = col_date(format = "%d %B %Y")
  ),
  na = c("", "NA", "N/A")
)

problems(desserts)
```

```
# A tibble: 0 x 5
# ... with 5 variables: row <int>, col <int>,
#   expected <chr>, actual <chr>, file <chr>
```

15.2 Your Turn 2

Use the appropriate `read_<type>()` function to import the following data sets:

- `https://deepbas.io/data/simple-1.dat`
- `https://deepbas.io/data/mild-1.csv`
- `https://deepbas.io/data/tricky-1.csv`
- `https://deepbas.io/data/tricky-2.csv`

Identify and fix any issues you encounter.

15.2.1 a) Importing simple data:

Click for answer

```
simple1 <- readr::read_csv("https://deepbas.io/data/simple-1.dat")
problems(simple1)
```

```
# A tibble: 0 x 5
# ... with 5 variables: row <int>, col <int>,
#   expected <chr>, actual <chr>, file <chr>
```

15.2.2 b) Importing mildly tricky data:

Click for answer

```
mild1 <- readr::read_delim("https://deepbas.io/data/mild-1.csv", delim = "|")
problems(mild1)
```

```
# A tibble: 0 x 5
# ... with 5 variables: row <int>, col <int>,
#   expected <chr>, actual <chr>, file <chr>
```

15.2.3 c) Importing tricky data 1:

Click for answer

```
tricky1 <-  read_csv("https://deepbas.io/data/tricky-1.csv")
problems(tricky1)
```

```
# A tibble: 2 x 5
    row   col expected  actual    file
  <int> <int> <chr>     <chr>     <chr>
1     4     4 5 columns 4 columns ""
2     7     4 5 columns 4 columns ""
```

```
# Fix missing values
tricky1[3, ] <- c(tricky1[3, 1:2], NA, tricky1[3, 3:4])
tricky1[6, ] <- c(tricky1[4, 1], NA, tricky1[4, 3:5])
```

15.2.4 d) Importing tricky data 2:

Click for answer

```
tricky2 <- read_csv("https://deepbas.io/data/tricky-2.csv")
problems(tricky2)
```

```
# A tibble: 0 x 5
# ... with 5 variables: row <int>, col <int>,
#   expected <chr>, actual <chr>, file <chr>
```

```r
# Fix missing values
tricky2_part1 <- read_csv("https://deepbas.io/data/tricky-2.csv", n_max = 7) %>%
  separate(city, c("city", "state"), sep = ",") %>%
  select(-c(7))

tricky2_part2 <- read_csv(
  "https://deepbas.io/data/tricky-2.csv",
  skip = 8,
  col_names = c("iata", "airport", "city", "state",  "latitude", "longitude")
)

# Combine parts
data_combined <- full_join(tricky2_part1, tricky2_part2)
```

Chapter 16

Class Activity 10

```
# load the necessary libraries
library(tidyverse)
library(tidyr)
```

16.1 Your Turn 1

```
students <- tibble(
  id = 1:24,
  grade = sample(c("9th", "10th", "11th"), 24, replace = TRUE),
  region = sample(c("North America", "Europe", "Asia", "South America", "Middle East", "Africa"),
  score = round(runif(24,50, 100))
)
```

16.1.1 a. Create a new column `grade_fac` by converting the grade column into a factor. Reorder the levels of `grade_fac` to be "9th", "10th", and "11th". Sort the dataset based on the `grade_fac` column.

Click for answer

Answer:

```
students_a <- students %>%
  mutate(grade_fac = factor(grade)) %>%
```

```
  mutate(grade_fac = fct_relevel(grade_fac, c("9th", "10th", "11th"))) %>%
  arrange(grade_fac)
print(students_a, n = 24)
```

```
# A tibble: 24 x 5
      id grade region        score grade_fac
   <int> <chr> <chr>         <dbl> <fct>
 1     3 9th   Africa           65 9th
 2     9 9th   North America    85 9th
 3    11 9th   Africa           83 9th
 4    15 9th   North America    88 9th
 5     1 10th  South America    57 10th
 6     2 10th  Africa           99 10th
 7     4 10th  South America    61 10th
 8     5 10th  Europe           54 10th
 9    10 10th  South America    92 10th
10    12 10th  Middle East      70 10th
11    18 10th  Africa           50 10th
12    23 10th  Middle East      78 10th
13    24 10th  Middle East      62 10th
14     6 11th  Europe           82 11th
15     7 11th  Africa           85 11th
16     8 11th  Europe           52 11th
17    13 11th  Middle East     100 11th
18    14 11th  Africa           59 11th
19    16 11th  Middle East      65 11th
20    17 11th  Asia             72 11th
21    19 11th  South America    67 11th
22    20 11th  Middle East      97 11th
23    21 11th  Europe           62 11th
24    22 11th  Africa           64 11th
```

16.1.2 b. Create a new column `region_fac` by converting the `region` column into a factor. Collapse the `region_fac` levels into three categories: "Americas", "EMEA" and "Asia". Count the number of students in each collapsed region category.

Click for answer

Answer:

```
students_b <- students_a %>%
  mutate(region_fac = factor(region)) %>%
  mutate(region_collapsed = fct_collapse(region_fac,
                                         Americas = c("North America", "South America"),
                                         EMEA = c("Europe", "Middle East", "Africa"),
                                         Asia = "Asia")) %>%
  count(region_collapsed)
print(students_b)
```

```
# A tibble: 3 x 2
  region_collapsed     n
  <fct>            <int>
1 EMEA                17
2 Asia                 1
3 Americas             6
```

16.1.3 c. Create a new column **grade_infreq** that is a copy of the **grade_fac** column. Reorder the levels of **grade_infreq** based on their frequency in the dataset. Print the levels of **grade_infreq** to check the ordering.

Click for answer

Answer:

```
students_c <- students_a %>%
  mutate(grade_infreq = grade_fac) %>%
  mutate(grade_infreq = fct_infreq(grade_infreq))

levels(students_c$grade_infreq)
```

```
[1] "11th" "10th" "9th"
```

16.1.4 d. Create a new column **grade_lumped** by lumping the least frequent level of the **grade_fac** column into an 'Others' category. Count the number of students in each of the categories of the **grade_lumped** column.

Click for answer

Answer:

```
students_d <- students_a %>%
  mutate(grade_lumped = fct_lump(grade_fac, n = 1, other_level = "Others")) %>%
  count(grade_lumped)
students_d
```

```
# A tibble: 2 x 2
  grade_lumped     n
  <fct>        <int>
1 11th            11
2 Others          13
```

16.2 Your Turn 2

Lets import the `gss_cat` dataset from the `forcats` library. This datast contains a sample of categorical variables from the General Social survey.

```
# import gss_cat dataset from forcats library
forcats::gss_cat
```

```
# A tibble: 21,483 x 9
    year marital       age race  rincome partyid relig denom
   <int> <fct>       <int> <fct> <fct>   <fct>   <fct> <fct>
 1  2000 Never marr~    26 White $8000 ~ Ind,ne~ Prot~ Sout~
 2  2000 Divorced       48 White $8000 ~ Not st~ Prot~ Bapt~
 3  2000 Widowed        67 White Not ap~ Indepe~ Prot~ No d~
 4  2000 Never marr~    39 White Not ap~ Ind,ne~ Orth~ Not ~
 5  2000 Divorced       25 White Not ap~ Not st~ None  Not ~
 6  2000 Married        25 White $20000~ Strong~ Prot~ Sout~
 7  2000 Never marr~    36 White $25000~ Not st~ Chri~ Not ~
 8  2000 Divorced       44 White $7000 ~ Ind,ne~ Prot~ Luth~
 9  2000 Married        44 White $25000~ Not st~ Prot~ Other
10  2000 Married        47 White $25000~ Strong~ Prot~ Sout~
# ... with 21,473 more rows, and 1 more variable:
#   tvhours <int>
```

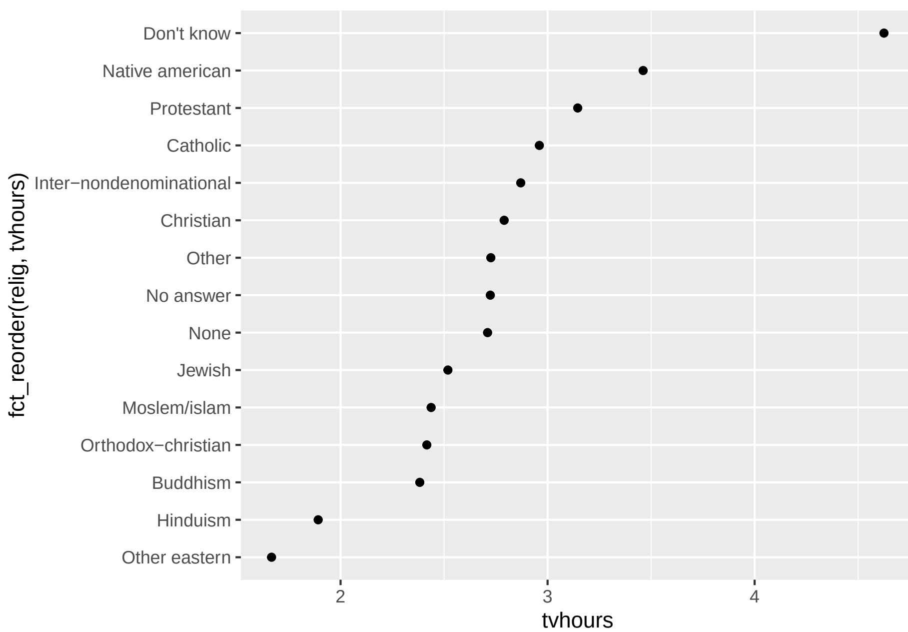
Use `gss_cat` to answer the following questions.

16.2.1 a. Which religions watch the least TV?

Click for answer

Answer:

```
# your r-code
gss_cat %>%
  drop_na(tvhours) %>%
  group_by(relig) %>%
  summarize(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, fct_reorder(relig, tvhours))) +
    geom_point()
```

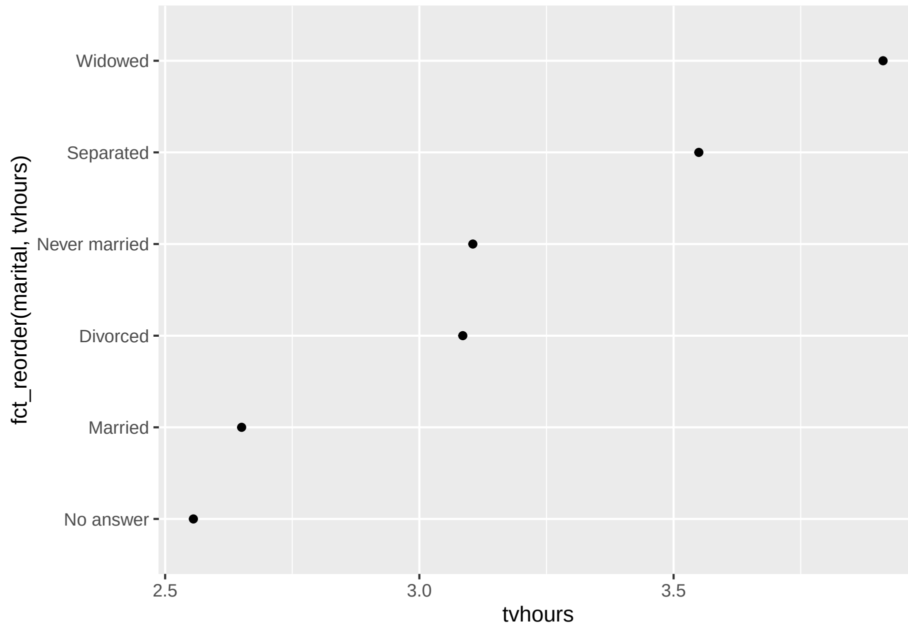


16.2.2 b. Do married people watch more or less TV than single people?

Click for answer

Answer:

```
# your r-code
gss_cat %>%
  drop_na(tvhours) %>%
  group_by(marital) %>%
  summarize(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, fct_reorder(marital, tvhours))) +
    geom_point()
```



16.2.3 c. Collapse the `marital` variable to have levels `married`, `not_married`, and `no_answer`. Include `"Never married"`, `"Divorced"`, and “`Widowed"` in `not_married`

Click for answer

Answer:

```
# your r-code
gss_cat %>%
  drop_na(tvhours) %>%
  select(marital, tvhours) %>%
  mutate(
    maritalStatus =
      fct_collapse(
        marital,
        married = c("Married",
                    "Separated"),
        not_married = c("Never married",
                        "Divorced",
                        "Widowed"),
        no_answer = c("No answer"))
  ) -> marital_c
```

```
levels(marital_c$maritalStatus)
```

```
[1] "no_answer"   "not_married" "married"
```

Chapter 17

Class Activity 11

```
# load the necessary libraries
library(tidyverse)
library(stringr)
```

17.1 Problem 1

Let's learn about combining strings with different separators first.

```
place <- "Central Park"
activity <- "jogging"
activities <- c("jogging", "picnicking", "boating")
my_sentence <- str_c(place, " is great for ", activity, ".", sep = "")
my_sentence
```

```
[1] "Central Park is great for jogging."
```

a. What happens when a `str_c` entry is a vector?

Click for answer

Answer: When an entry in `str_c` is a vector, it will combine the strings with each element of the vector, creating multiple combined strings.

```
my_sentences <- str_c(place, " is great for ", activities, ".", sep = "")
my_sentences
```

```
[1] "Central Park is great for jogging."
[2] "Central Park is great for picnicking."
[3] "Central Park is great for boating."
```

b. How do you combine strings with `str_glue`?

Click for answer

Answer: You can combine strings with `str_glue` using curly braces `{}` to insert variables directly into the string.

```
my_sentence <- str_glue("{place} is great for {activity}.")
my_sentence
```

```
Central Park is great for jogging.
```

```
my_sentences1 <- str_glue("{place} is great for {activities}.")
my_sentences1
```

```
Central Park is great for jogging.
Central Park is great for picnicking.
Central Park is great for boating.
```

c. What does `str_flatten` do?

Click for answer

Answer: `str_flatten` collapses a character vector into a single string by concatenating the elements with a specified separator.

```
str_flatten(my_sentences, collapse = " and ")
```

```
[1] "Central Park is great for jogging. and Central Park is great for picnicking. and C
```

d. What will using a `\n` separator do in the command below?

Click for answer

Answer: Using a `\n` separator in the command will insert a newline character between the strings being combined, making them display on separate lines when printed.

```
p <- str_c(place, " is great for ", activity, sep = "\n")
writeLines(p)
```

```
Central Park
 is great for
jogging
```

e. Does `str_length` count spaces and special characters??

Click for answer

Answer: Yes, `str_length` counts spaces and special characters as part of the string's length.

```
p
```

```
[1] "Central Park\n is great for \njogging"
```

```
str_length(p)
```

```
[1] 35
```

f. How do you count the number of `e`'s in a string?

Click for answer

Answer: You can count the number of `e`'s in a string using `str_count` with a pattern that matches the character 'e'.

```
text <- "The quick brown fox jumps over the lazy dog."
pattern <- "e"
vowel_count <- str_count(text, pattern)
vowel_count
```

```
[1] 3
```

g. What happens with negative positions?

Click for answer

Answer: Negative positions in `str_sub` count the positions from the end of the string rather than from the beginning.

```
str_sub(my_sentence, start = -3, end = -1)
```

```
[1] "ng."
```

h. How do you extract a `substring` with positive and negative positions?

Click for answer

Answer: You can extract a `substring` with positive and negative positions using `str_sub` and specifying the start and end positions with either positive or negative numbers.

```
my_sentence <- "Central Park is great for jogging."
positive_substr <- str_sub(my_sentence, start = 1, end = 12)
negative_substr <- str_sub(my_sentence, start = -8, end = -1)
positive_substr
```

```
[1] "Central Park"
```

```
negative_substr
```

```
[1] "jogging."
```

i. With a vector of positions?

Click for answer

Answer: Using a vector of positions with `str_sub` will extract `substrings` starting and ending at the specified positions in the vector.

```
str_sub(my_sentence, start = c(1, 9), end = c(4, 15))
```

```
[1] "Cent"    "Park is"
```

j. How do you extract multiple `substrings` using a vector of positions?

Click for answer

Answer: You can extract multiple `substrings` using a vector of positions with `str_sub` by specifying the start and end positions in separate vectors.

```
my_sentence <- "Central Park is great for jogging."
substrs <- str_sub(my_sentence, start = c(1, 14, 24), end = c(12, 19, 30))
substrs
```

```
[1] "Central Park" "is gre"       "or jogg"
```

17.2 Problem 2

a. Use the string parsing functions that you learned today to do tasks described in the comments below:

```
s1 <- "12%"  # remove %
s2 <- "New Jersey_*"  # remove _*
s3 <- "2,150"    # remove comma(,)
s4 <- "Learning #datascience is fun!"   # extract #datascience
s5 <- "123 Main St, Springfield, MA, 01101"   # separate info
```

Click for answer

```
# Cleaning steps
s1_clean <- str_replace(s1, "%", "")
s2_clean <- str_replace(s2, "_\\*", "")
s3_clean <- str_replace(s3, ",", "")
s4_clean <- str_extract(s4, "#\\w+")
s5_clean <- str_split(s5, ",\\s?")

# Print cleaned strings
s1_clean
```

```
[1] "12"
```

```
s2_clean
```

```
[1] "New Jersey"
```

```
s3_clean
```

```
[1] "2150"
```

```
s4_clean
```

```
[1] "#datascience"
```

```
s5_clean
```

```
[[1]]
[1] "123 Main St" "Springfield" "MA"          "01101"
```

17.3 Problem 3

a. Use the string parsing functions that you learned today to do tasks described in the comments below:

```
s1 <- "25%"  # remove %
s2 <- "Los Angeles_#"  # remove _#
s3 <- "1,250"    # remove comma(,)
s4 <- "Discover #machinelearning today!"   # extract #machinelearning
s5 <- "456 Main St, San Francisco, CA, 94107"   # separate info
```

Click for answer

```
# Cleaning steps
s1_clean <- str_replace(s1, "%", "")
s2_clean <- str_replace(s2, "_#", "")
s3_clean <- str_replace(s3, ",", "")
s4_clean <- str_extract(s4, "#\\w+")
s5_clean <- str_split(s5, ",\\s?")

# Print cleaned strings
s1_clean
```

```
[1] "25"
```

```
s2_clean
```

```
[1] "Los Angeles"
```

```
s3_clean
```

```
[1] "1250"
```

```
s4_clean
```

```
[1] "#machinelearning"
```

```
s5_clean
```

```
[[1]]
[1] "456 Main St"   "San Francisco" "CA"
[4] "94107"
```

17.4 Problem 4

a. Let's look at the following dataset containing information about movies and their release years. We'll extract the release year from the movie title, create a new column with decades, and count the number of movies in each decade.

```
# Sample dataset
movies <- tibble(
  title = c(
    "The Godfather (1972)", "Pulp Fiction (1994)", "The Dark Knight (2008)",
    "Forrest Gump (1994)", "The Shawshank Redemption (1994)", "The Matrix (1999)",
    "Inception (2010)", "Interstellar (2014)", "Parasite (2019)", "Fight Club (1999)"
  )
)
movies
```

```
# A tibble: 10 x 1
   title
   <chr>
 1 The Godfather (1972)
 2 Pulp Fiction (1994)
 3 The Dark Knight (2008)
 4 Forrest Gump (1994)
 5 The Shawshank Redemption (1994)
 6 The Matrix (1999)
 7 Inception (2010)
 8 Interstellar (2014)
 9 Parasite (2019)
10 Fight Club (1999)
```

Click for answer

```
# Processing the dataset
movies_processed <- movies %>%
  mutate(
    release_year = as.integer(str_extract(title, "\\d{4}")),
    decade = floor(release_year / 10) * 10
  ) %>%
  count(decade) %>%
  rename(num_movies = n)

# Print the processed dataset
movies_processed
```

```
# A tibble: 4 x 2
  decade num_movies
   <dbl>      <int>
1   1970          1
2   1990          5
3   2000          1
4   2010          3
```

Chapter 18

Class Activity 12

In-class midterm!

Chapter 19

Class Activity 13

```r
# load the necessary libraries
library(stringr)
library(dplyr)
library(readr)
```

In this tutorial, we will learn about string manipulations using regular expressions and the `stringr` library in R. We will cover different examples and use cases to help you understand the concepts and functions related to string manipulation.

19.1 Group Activity 1

```r
x <- "My SSN is 593-29-9502 and my age is 55"
y <- "My phone number is 612-643-1539"
z <- "My old SSN number is 39532 9423."
out <- str_flatten(c(x,y,z), collapse = ". ")
```

19.1.1 a. What characters in x will `str_view_all(x, "-..-")` find?

Click for answer

answer:

The pattern searches for a dash, followed by any two characters, followed by another dash. In x, it finds "-29-" which is a part of the SSN.

```
str_view_all(x, "-..-")
```

```
[1] | My SSN is 593<-29->9502 and my age is 55
```

19.1.2 b. What pattern will `str_view_all(x, "-\\d{2}-")` find?

Click for answer

answer:

The pattern searches for a dash, followed by two digits, followed by another dash. In x, it finds the same "-29-" as in the previous example, which is a part of the SSN.

```
str_view_all(x, "-\\d{2}-")  # "-" then 2 digits then "-"
```

```
[1] | My SSN is 593<-29->9502 and my age is 55
```

19.1.3 c. What pattern will `str_view_all(out, "\\d{2}\\.*")` find?

Click for answer

answer:

The pattern searches for two digits followed by an optional period. In out, it finds "55" and "55.", which represent the age in the first sentence.

```
str_view_all(out, "\\s\\d{2}\\.")  # 2 digits then "."
```

```
[1] | My SSN is 593-29-9502 and my age is< 55.> My phone number is 612-643-1539. My ol
```

19.1.4 d. Use `str_view_all` to determine the correct regex pattern to identify all SSN in `out`

We can get the SSN with the usual format (###-##-####) with a regex that has 3, 2, and 4 digits separated by a dash.

```
str_view_all(out,"([0-8]\\d{2})-(\\d{2})-(\\d{4})")
```

```
[1] | My SSN is <593-29-9502> and my age is 55. My phone number is 612-643-1539. My ol
```

This misses the oddly formatted SSN in the third entry. Rather than use a dash, we can specify the divider as `[-\\s]?` which allows either 0 or 1 occurrences of either a dash or space divider:

```
str_view_all(out,"([0-8]\\d{2})[-\\s]?(\\d{2})[-\\s]?(\\d{4})")
```

```
[1] | My SSN is <593-29-9502> and my age is 55. My phone number is 612-643-1539. My old SSN numbe
```

Click for answer

answer:

The first pattern finds the SSNs in the standard format (###-##-####) by searching for 3 digits, a dash, 2 digits, another dash, and 4 digits. The second pattern does the same but allows for a space instead of a dash as a divider. It finds all SSNs in out, including the oddly formatted one in the third sentence.

19.1.5 e. Write a regular expression to extract dates in the format YYYY-MM-DD from a given text.

```
date_pattern <- "\\d{4}-\\d{2}-\\d{2}"
text <- "The event will take place on 2023-07-20 and end on 2023-07-22."
str_extract_all(text, date_pattern)
```

```
[[1]]
[1] "2023-07-20" "2023-07-22"
```

Click for answer

Answer: The pattern searches for 4 digits, a dash, 2 digits, another dash, and 2 digits. In the given text, it finds the dates “2023-07-20” and “2023-07-22”.

19.1.6 f. Write a regular expression to extract all words that start with a capital letter in a given text.

```
capital_pattern <- "\\b[A-Z][a-zA-Z]*\\b"
text <- "Alice and Bob went to the Market to buy some Groceries."
str_extract_all(text, capital_pattern)
```

```
[[1]]
[1] "Alice"     "Bob"       "Market"    "Groceries"
```

Click for answer

Answer: The pattern searches for a word boundary, followed by an uppercase letter, and then any sequence of letters. In the given text, it finds the words "Alice", "Bob", "Market", and "Groceries".

19.1.7 g (Optional) Create a regular expression to match URLs, considering both http and https protocols.

```
url_pattern <- "https?://(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,}(?::\\d+)?(?:/\\S*)?"
urls <- c("https://www.example.com", "http://example.org/resource?query=123", "invalid_
str_view(urls, url_pattern)
```

```
[1] | <https://www.example.com>
[2] | <http://example.org/resource?query=123>
```

Click for answer

Answer: The pattern searches for either "http://" or "https://", followed by one or more domain segments separated by periods, an optional port number, and an optional path. In the given urls, it matches the first two valid URLs but not the invalid one.

`https?://`: This part matches the URL protocol (http or https). The `?` means that the preceding character "s" is optional. `(?:[a-zA-Z0-9-]+\\.)+`: This part matches the domain name. The `(?: )` is a non-capturing group, and `[a-zA-Z0-9-]+` matches one or more alphanumeric characters or hyphens. The `\\.` matches a period (dot), and the + outside the group means that the pattern inside the group should occur one or more times. `[a-zA-Z]{2,}`: This part matches the top-level domain (TLD), such as .com or .org, which has at least two alphabetic characters. `(?::\\d+)?`: This part is an optional group that matches a colon followed by one or more digits. It is used to specify a port number in the URL. The ? outside the group makes it optional. `(?:/\\S*)?`: This part is another optional group that matches a forward slash followed by zero or more non-whitespace characters (representing the URL path). Again, the `?` outside the group makes it optional.

19.2 Group Activity 2

19.2.1 a. Let's deal with a number string that is longer than 9 digits.

```
ssn <- "([0-8]\\d{2})[-\\s]?(\\d{2})[-\\s]?(\\d{4})"
test <- c("123-45-67890","1123 45 6789")
str_view_all(test, ssn)
```

```
[1] | <123-45-6789>0
[2] | 1<123 45 6789>
```

This example captures a 9-digit string as an SSN, but these strings are longer than 9 digits and may not represent an SSN. One way to deal with this is to use the negative lookbehind `?<!` and negative lookahead `?!` operators to ensure that the identified 9-digit string does not have a leading 0 or does not contain more digits.

If we "look behind" from the start of the SSN, we should not see another digit:

```
str_view_all(test, "(?<!\\d)([0-8]\\d{2})[-\\.\\s]?(\\d{2})[-\\.\\s]?(\\d{4})")
```

```
[1] | <123-45-6789>0
[2] | 1123 45 6789
```

And if we "look ahead" from the end of the SSN, we should not see another digit:

```
str_view_all(test, "(?<!\\d)([0-8]\\d{2})[-\\.\\s]?(\\d{2})[-\\.\\s]?(\\d{4})(?!\\d)")
```

```
[1] | 123-45-67890
[2] | 1123 45 6789
```

For parts b and c, consider the following string.

```
string1 <- "100 dollars 100 pesos"
```

19.2.2 b. Explain why the following matches the first 100 and not the second.

Click for answer

answer: It looks for one or more digits followed by a space and `dollars`

```
str_view(string1, "\\d+(?= dollars)")
```

```
[1] | <100> dollars 100 pesos
```

19.2.3 c. Explain why the following matches the second 100 and not the first.

Click for answer

answer: It looks for one or more digits not followed by either a digit or space followed by `dollars`

```
str_view(string1, "\\d+(?!\\d| dollars)")
```

```
[1] | 100 dollars <100> pesos
```

For parts d and e, please take a look at `string2`.

```
string2 <- "USD100 PESO100"
```

19.2.4 d. Explain why the following matches the first 100 and not the second.

Click for answer

answer: It looks for exactly 3 digits preceded by USD

```
str_view(string2, "(?<=USD)\\d{3}")
```

```
[1] | USD<100> PESO100
```

19.2.5 e. Explain why the following matches the second 100 and not the first.

Click for answer

answer: It looks for exactly 3 digits that is not preceded by USD

```
str_view(string2, "(?<!USD)\\d{3}")
```

```
[1] | USD100 PESO<100>
```

19.3 Group Activity 3

```
tweets<- read_csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/TrumpTweetData.csv
```

19.3.1 a. What proportion of tweets (text) mention "America"?

```
tweets %>%
  summarize(prop = mean(str_detect(str_to_title(text), "America")))
```

```
# A tibble: 1 x 1
    prop
   <dbl>
1 0.0926
```

Click for answer

Answer: About 10% of tweets mention "America".

19.3.2 b. What proportion of these tweets include "great"?

```
tweets %>% filter(str_detect(str_to_title(text), "America")) %>%
  summarize(prop = mean(str_detect(str_to_lower(text), "great")))
```

```
# A tibble: 1 x 1
   prop
  <dbl>
1   0.4
```

Click for answer

Answer: About 40% of tweets mention "great".

19.3.3 c. What proportion of the tweets mention `@`?

```
tweets %>% mutate(ct = str_count(text, "@")) %>%
  select(text, ct) %>%
  summarize(prop = mean(ct>0))
```

```
# A tibble: 1 x 1
   prop
  <dbl>
1 0.317
```

Click for answer

Answer: About 32% of tweets mention `@`.

19.3.4 d. Remove the tweets having mentions `@`.

```
Mentions <- c("@[^\\s]+")

tw_noMentions <- tweets %>% mutate(textNoMentions = str_replace_all(text, Mentions, "")
tw_noMentions$text[38]
```

```
[1] "My daughter @IvankaTrump will be on @Greta tonight at 7pm. Enjoy! https://t.co/QyS
```

```
tw_noMentions$textNoMentions[38]
```

```
[1] "My daughter  will be on  tonight at 7pm. Enjoy! https://t.co/QySC5PLFMy"
```

Click for answer

Answer: `@`: This part of the pattern matches the "@" symbol, which usually indicates the beginning of a mention in a tweet. [^\s]+: This part of the pattern matches one or more characters that are NOT whitespaces. The `^` inside the square brackets `[ ]` negates the character class (meaning it matches any character that is NOT in the specified class). The double backslash `\\` is used to escape the backslash in the R string, so the pattern `\\s` represents the whitespace character class `\s`. Finally, the + indicates that the pattern should match one or more occurrences of the non-whitespace characters. Together, this regular expression pattern `@[^\\s]+` matches any mention in a tweet, which usually starts with "@" followed by one or more non-whitespace characters.

19.3.5 e. What poportion of tweets originated from an iPhone?

```
tweets %>% group_by(source) %>% summarize(count = n()) %>%
  mutate(prop = count / sum(count)) %>%  filter(source == "iPhone")
```

```
# A tibble: 1 x 3
  source count  prop
  <chr>  <int> <dbl>
1 iPhone   628 0.415
```

Click for answer

Answer: About 42% of the tweets originated from an iPhone.

Chapter 20

Class Activity 14

```r
# load the necessary libraries
library(wordcloud)
library(reshape2)
library(tidyverse)
library(tidyr)
library(tidytext)
library(dplyr)
```

20.1 Group Activity 1

20.1.1 a. Variance and Skewness

The variance of a random variable x is defined as:

$$\mathrm{Var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where $x_i = (\sum_i^n x_i)/n$ is the sample mean. Also, the skewness of the random variable x is defined as:

$$\mathrm{Skew}(x) = \frac{\frac{1}{n-2}\left(\sum_{i=1}^{n}(x_i - \bar{x})^3\right)}{\mathrm{Var}(x)^{3/2}}$$

Please write functions to calculate the variance and skewness of $\{12, 45, 54, 34, 56, 30, 67, \mathrm{NA}\}$.

```
x <- c(12, 45, 54, 34, 56, 30, 67, NA)
```

Click for answer

```
# function to calculate the variance of a vector
var <- function(x){
  x <- na.omit(x) # omit NA values
  sum((x - mean(x)) ^ 2) / (length(x) - 1)
}
```

```
var(x)
```

```
[1] 346.619
```

```
# function to calculate the skewness of a vector
skewness <- function(x) {
  x <- na.omit(x) # omit NA values
  sum((x - mean(x)) ^ 3) /((length(x) - 2) * var(x) ^ (3 / 2))
}
```

```
skewness(x)
```

```
[1] -0.3930586
```

20.1.2 b. Conditions and breaks

Create a function that iterates through a numeric vector and stops when it encounters the first negative number, returning the position of that negative number. If there are no negative numbers in the vector, the function should return a message stating that there are no negative numbers.

Click for answer

```
find_first_negative <- function(x) {
  negative_positions <- which(x < 0)

  if (length(negative_positions) > 0) {
    return(paste("The first negative number is at position", negative_positions[1]))
  } else {
    return("There are no negative numbers in the vector")
  }
}
```

```
test_vector <- c(5, 12, -7, 20, 15)
find_first_negative(test_vector)
```

```
[1] "The first negative number is at position 3"
```

20.2 Group Activity 2

```
musical_instr_reviews <-  read_csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/m
  rename(ratingOverall = overall)
glimpse(musical_instr_reviews)
```

```
Rows: 10,261
Columns: 3
$ reviewerName  <chr> "cassandra tu \"Yeah, well, that's j~
$ reviewText    <chr> "not much to write about here but it~
$ ratingOverall <dbl> 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 5, 4, ~
```

20.2.1 a. Write a function to filter the dataset based on the provided rating:

Click for answer

```
filter_reviews_by_rating <- function(data, rating) {
  data %>% filter(ratingOverall == rating)
}
```

20.2.2 b. Write a function to process the text and remove stop words:

Click for answer

```
process_text <- function(data) {
  data %>%
    select(reviewText) %>%
    unnest_tokens(output = word, input = reviewText) %>%
    anti_join(stop_words)
}
```

20.2.3 c. Write a function to join the processed text with sentiment data and create a word count table.

Click for answer

```
create_word_count_table <- function(data) {
  data %>%
    inner_join(get_sentiments("bing")) %>%
    count(word, sentiment, sort = TRUE) %>%
    reshape2::acast(word ~ sentiment, value.var = "n", fill = 0)
}
```

20.2.4 d. Create the final function that takes the rating and number of words as input arguments and returns a word cloud plot.

Click for answer

```
word_cloud <- function(rating, num.words) {
  rating <- as.numeric(rating)
  num.words <- as.numeric(num.words)

  if (rating >= 1 & rating <= 5) {
    filtered_reviews <- filter_reviews_by_rating(musical_instr_reviews, rating)
    processed_reviews <- process_text(filtered_reviews)
    word_count_table <- create_word_count_table(processed_reviews)

    comparison.cloud(
      word_count_table,
      colors = c("blue", "purple"),
      scale = c(2, 0.5),
      max.words = num.words,
      title.size = 2
    )
  } else {
    warning(" Please enter a rating from 1 to 5")
  }
}

word_cloud(rating = "4", num.words = 300)
```


negative
intermittent sticky dull
slowly complained uncomfortable smudges pain error fat dust fragile worried falls blowannoyance
failed odd twist tired slow attack fell losing weird
bent heck rust useless stuck complain disappointed mistake monster kill suspect unnecessary
funky nasty terrible bulky negative warning lacking scratches
complaining distorted breaking punch dark concerned noisy drawback strange picky
junk snare loss worse dead damage complaint rough bother shock crap complex worthless
smell darker hate complaints fail difficult hiss bump costly
damaged risk weak gripe stiff loud distortion broken knock flimsy cheaply aggressive
lacks terror hell falling dirty thicker cons ax trouble trick muddy slap
buzzing stress hum frets issue broke expensive scratch awkward suck friction difficulty
breakup breaks abuse miss jam louder limited break bad noise downside vibration audacity
latency horrible dirt wrong fret lost fall harsh loose tank concern steal
crack lemon cold hang fault hard cheap delay wound doubt poor pricey tricky
criticism flaw lack issues worry lose tension annoying mess harm
popular support super clean nice pretty sharp gentle
effective convenient classic soft top easy excellent fast quiet luck beautiful
rich cheaper gain fine decent ideal led proper protection helped
leads reliable handy recommend prefer strong crisp
smoother enjoy pleased sweet flawlessly adequate
favor loves nicely perfect ease solid bright boost stable dynamic
versatility impressed accurate warm love happy boom versatile clarity correctly
improvement adjustable sturdy fair worth wow durable smooth wonderful
champ tough cool perfectly fairly fun cleaner fantastic
advantage portable inexpensive easier free favorite amazing softer correct
dedicated hot lead awesome comfortable glad helpful nicer silent precise
improved gold recommended slick lighter sensitive bargain
afford satisfied secure properly brighter compact pure
impressive balanced pros reasonable flexible ready securely
modern trust affordable variety wise likes attractive
transparent protect improve master bonus thinner
fortunately faster robust flexibility superior intuitive
honest
positive

Chapter 21

Class Activity 15

```
# load the necessary libraries
library(tidyverse)
library(dplyr)
library(stringr)


energy <- read_csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/energy.csv",
                   col_type = cols(
                    .default = col_double(),
                     Timestamp = col_datetime(format = ""),
                     dayWeek = col_factor(levels=c("Mon","Tues","Wed","Thurs","Fri","Sat","Sun")
                    ))
```

21.1 Group Activity 1

21.1.1 a. `if` and `for` loop

Write a for loop to iterate over the columns of the 'energy' dataset and print the names of all columns containing the string "House". Please use the function `colnames()` to extract the column names and store the results in a list.

Click for answer

Answer:

```
# Create an empty list to store the column names
house_columns <- list()
```

```r
# Iterate over the columns of the 'energy' dataset
for (i in seq_along(colnames(energy))) {
  col_name <- colnames(energy)[i]

  # Check if the column name contains the string "House"
  if (str_detect(col_name, "House")) {
    # Add the column name to the list
    house_columns[[length(house_columns) + 1]] <- col_name
  }
}

# Print the list of house columns
house_columns <- unlist(house_columns)
house_columns
```

```
 [1] "Allen_House"
 [2] "Alumni_Guest_House/Johnson_House"
 [3] "Benton_House"
 [4] "Berg_House"
 [5] "Bird_House"
 [6] "Chaney_House"
 [7] "Clader_House"
 [8] "Dacie_Moses_House"
 [9] "Douglas_House"
[10] "Farm_House"
[11] "Geffert_House"
[12] "Headley_House"
[13] "Henrickson_House"
[14] "Henry_House"
[15] "Hill_House"
[16] "Hilton_House"
[17] "Hoppin_House_(Alumni)"
[18] "Huntington_House"
[19] "Jewett_House"
[20] "Jones_House"
[21] "Nutting_House"
[22] "Page_House_West"
[23] "Parish_House_"
[24] "Parr_House"
[25] "Pollock_House"
[26] "Prentice_House"
[27] "Rayment_House"
[28] "Rice_House"
[29] "Rogers_House"
[30] "Ryberg_House"
```

```
[31] "Seccombe_House"
[32] "Sperry_House"
[33] "Stimson_House"
[34] "Strong_House"
[35] "Whittier_House"
[36] "Wilson_House"
```

21.1.2 b. `for` loop and mean

Using a for loop, calculate and print the mean energy consumption of houses you identified in part a.

Click for answer

Answer:

```
# Assuming the house_columns vector from the previous step

# Create an empty numeric vector to store the mean energy consumption
mean_energy_consumption <- numeric()

# Iterate over the house_columns vector
for (house_col in house_columns) {
  # Calculate the mean energy consumption for the current house column
  mean_val <- mean(energy[[house_col]], na.rm = TRUE)

  # Add the mean energy consumption to the vector
  mean_energy_consumption <- c(mean_energy_consumption, mean_val)
}

# Combine the house names and mean energy consumption into a dataframe
house_mean_energy <- bind_cols(House = house_columns, MeanEnergyConsumption = mean_energy_consump

# Print the dataframe
house_mean_energy %>%  knitr::kable()
```

House	MeanEnergyConsumption
Allen_House	0.9821865
Alumni_Guest_House/Johnson_House	20.2631152
Benton_House	1.8849290
Berg_House	1.3174340
Bird_House	2.3222680
Chaney_House	1.0715123
Clader_House	0.4646776
Dacie_Moses_House	1.2776465
Douglas_House	0.7219500
Farm_House	5.0599020
Geffert_House	0.9360400
Headley_House	1.4555605
Henrickson_House	3.4407858
Henry_House	1.3639619
Hill_House	1.4735884
Hilton_House	0.4248030
Hoppin_House_(Alumni)	1.8760474
Huntington_House	1.2395238
Jewett_House	0.8987697
Jones_House	0.8680271
Nutting_House	4.3967234
Page_House_West	1.8923490
Parish_House_	12.6793378
Parr_House	9.7210618
Pollock_House	1.1831426
Prentice_House	0.9089497
Rayment_House	0.8005664
Rice_House	1.1568457
Rogers_House	0.5634289
Ryberg_House	1.0729988
Seccombe_House	2.6874199
Sperry_House	0.7052983
Stimson_House	2.0659904
Strong_House	2.5410595
Whittier_House	1.0424369
Wilson_House	1.0435830

21.2 Group Activity 2

1. Make a data frame of quantiles for `energy` buildings in columns 9-90 (you will need `na.rm = TRUE`)

Click for answer

Answer:

```
qdf <- energy %>% select(9:90) %>%
  map_dfc(quantile, probs = seq(.1,.9,.1), na.rm = TRUE)
qdf
```

```
# A tibble: 9 x 82
  100_Neva~1 104_M~2 106_W~3 Allen~4 Alumn~5 Arbor~6 Art_S~7
       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1     0.0972    1.04   0.601   0.756    17.0    0.13    0.23
2     0.120     1.11   0.632   0.781    18.1    0.23    0.28
3     0.183     1.18   0.673   0.941    18.4    0.25    0.33
4     0.461     1.18   0.681   0.983    20.3    0.28    0.4 
5     0.710     1.42   0.692   1.00     21.0    0.32    0.47
6     0.795     1.42   0.865   1.01     21.8    0.38    0.57
7     0.915     1.54   1.10    1.07     21.9    0.44    0.73
8     1.11      1.56   1.20    1.07     22      0.52    0.88
9     1.24      1.67   1.27    1.25     22.5    0.71    1.09
# ... with 75 more variables: Benton_House <dbl>,
#   Berg_House <dbl>, Bird_House <dbl>,
#   Boliou_Memorial_Art_Bldg. <dbl>, Burton_Hall <dbl>,
#   `Cassat_Hall_/_James_Hall` <dbl>,
#   `Center_for_Mathematics_&_Computing` <dbl>,
#   Chaney_House <dbl>, Clader_House <dbl>,
#   College_Warehouse <dbl>, Cowling_Gym <dbl>, ...
```

2. Add a variable to identify the quantile

Click for answer

Answer:

```
qdf <- energy %>% select(9:90) %>%
  map_dfc(quantile, probs = seq(.1,.9,.1), na.rm = TRUE) %>%
  mutate(stat = str_c("quantile_", seq(10,90,10)))
qdf
```

```
# A tibble: 9 x 83
  100_Neva~1 104_M~2 106_W~3 Allen~4 Alumn~5 Arbor~6 Art_S~7
       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1     0.0972    1.04   0.601   0.756    17.0    0.13    0.23
2     0.120     1.11   0.632   0.781    18.1    0.23    0.28
3     0.183     1.18   0.673   0.941    18.4    0.25    0.33
4     0.461     1.18   0.681   0.983    20.3    0.28    0.4 
```

```
5     0.710     1.42   0.692   1.00     21.0    0.32    0.47
6     0.795     1.42   0.865   1.01     21.8    0.38    0.57
7     0.915     1.54   1.10    1.07     21.9    0.44    0.73
8     1.11      1.56   1.20    1.07     22      0.52    0.88
9     1.24      1.67   1.27    1.25     22.5    0.71    1.09
# ... with 76 more variables: Benton_House <dbl>,
#   Berg_House <dbl>, Bird_House <dbl>,
#   Boliou_Memorial_Art_Bldg. <dbl>, Burton_Hall <dbl>,
#   `Cassat_Hall_/_James_Hall` <dbl>,
#   `Center_for_Mathematics_&_Computing` <dbl>,
#   Chaney_House <dbl>, Clader_House <dbl>,
#   College_Warehouse <dbl>, Cowling_Gym <dbl>, ...
```

3. Reshape the data frame to make variables `stat` (describing the quantile), `building` and `quant` (quantile value)

Click for answer

Answer:

```
qdf <- energy %>% select(9:90) %>%
  map_dfc(quantile, probs = seq(.1,.9,.1), na.rm = TRUE) %>%
  mutate(stat = str_c("quantile_", seq(10,90,10))) %>%
  pivot_longer(names_to = "building", values_to = "quantiles", 1:82)
qdf
```

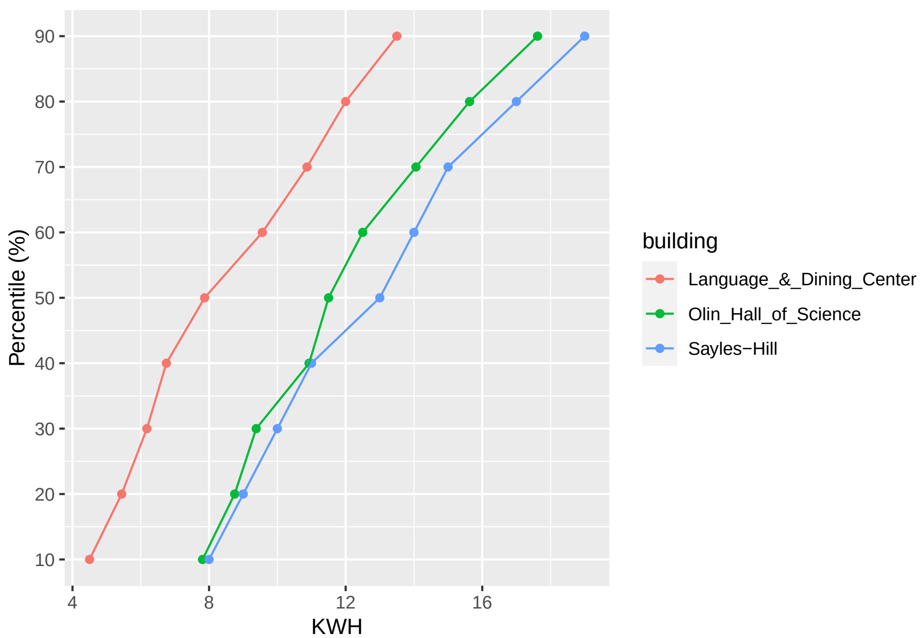
```
# A tibble: 738 x 3
   stat        building                          quantiles
   <chr>       <chr>                                 <dbl>
 1 quantile_10 100_Nevada_Street                    0.0972
 2 quantile_10 104_Maple_St.                        1.04
 3 quantile_10 106_Winona_St.                       0.601
 4 quantile_10 Allen_House                          0.756
 5 quantile_10 Alumni_Guest_House/Johnson_House    17.0
 6 quantile_10 Arboretum_Office                     0.13
 7 quantile_10 Art_Studios                          0.23
 8 quantile_10 Benton_House                         1.59
 9 quantile_10 Berg_House                           1.06
10 quantile_10 Bird_House                           1.42
# ... with 728 more rows
```

4. Plot the KWH value for each quantile on the x-axis for the buildings `Sayles-Hill`, `Language_&_Dining_Center`, `Olin_Hall_of_Science`

Click for answer

Answer:

```
qdf %>%
  filter(building %in% c("Sayles-Hill" ,"Language_&_Dining_Center", "Olin_Hall_of_Science"))  %>%
  ggplot(aes(x=quantiles, y=parse_number(stat), color=building)) +
  geom_point() +
  geom_line(aes(group=building)) +
  labs(y="Percentile (%)",x="KWH") +
  scale_y_continuous(breaks=seq(10,90,by=10))
```



Chapter 22

Class Activity 16

```
# load the necessary libraries
library(tidyverse)
library(stringr)
library(polite)
library(rvest)
```

22.1 Group Activity 1

```
url <- 'https://www.imdb.com/search/title/?groups=best_picture_winner&sort=year,desc&count=100&vi
webpage <- bow(url) %>% scrape()
```

As seen in the slides, we can extract the movie titles using the `.lister-item-header a` css selector.

```
title_data <- webpage %>%
  html_nodes(css = ".lister-item-header a") %>%
  html_text()
```

Now, lets scrape other elements of the webpage one by one using the selector gadget tool.

22.1.1 a. Use the selector gadget tool to find the CSS for extracting year the movie came out.

- Tidy the data

  - Using parse_number()
  - Using any regex

Click for answer

```
year_data <- webpage %>%
  html_nodes(css = '.text-muted.unbold') %>%
  html_text() %>%
  parse_number()

year_data1 <- webpage %>%
  html_nodes(css = '.text-muted.unbold') %>%
  html_text() %>%
  str_extract_all("[0-9]+") %>%
  unlist() %>%
  as.numeric()
```

22.1.2 b. Next, parse the webpage to produce a vector of the descriptions. Tidy the description by removing unwanted regexes.

Click for answer

```
description_data <- webpage %>%
  html_nodes('.ratings-bar+ .text-muted') %>%
  html_text() %>%
  str_trim()
head(description_data,3)
```

```
[1] "A middle-aged Chinese immigrant is swept up into an insane adventure in which she
[2] "As a CODA (Child of Deaf Adults) Ruby is the only hearing person in her deaf famil
[3] "A woman in her sixties, after losing everything in the Great Recession, embarks or
```

22.1.3 c. Now, parse the runtime data by following similar tools used for extracting year.

Click for answer

```
runtime_data <- webpage %>%
  html_nodes('.runtime') %>%
  html_text() %>%
  parse_number()
head(runtime_data)
```

```
[1] 139 111 107 132 130 123
```

22.1.4 d. Do the same for getting the ratings data using an appropriate selector.

Click for answer

```
rating_data <- webpage %>%
  html_nodes('.ratings-imdb-rating') %>%
  html_text() %>%
  as.numeric()
```

22.1.5 e. Finally, get the number of votes following similar tools

Click for answer

```
votes_data <- webpage %>%
  html_nodes('.sort-num_votes-visible span:nth-child(2)') %>%
  html_text() %>%
  parse_number()
```

We can combine all the information we scraped to produce a nice table that we ca use for further analysis. Please run the following code, once you are done with the prior codes.

Click for answer

```
movies_df <- data.frame(Year = year_data,
                      Title = title_data,
                      Description = description_data,
                      Runtime = runtime_data,
                      Rating = rating_data,
                      Votes = votes_data) %>% as_tibble()
movies_df %>% knitr::kable()
```

Year	Title	Description
2022	Everything Everywhere All at Once	A middle-aged Chinese immigrant is swep
2021	CODA	As a CODA (Child of Deaf Adults) Ruby
2020	Nomadland	A woman in her sixties, after losing every
2019	Parasite	Greed and class discrimination threaten t
2018	Green Book	A working-class Italian-American bounce
2017	The Shape of Water	At a top secret research facility in the 19
2016	Moonlight	A young African-American man grapples
2015	Spotlight	The true story of how the Boston Globe
2014	Birdman or (The Unexpected Virtue of Ignorance)	A washed-up superhero actor attempts to
2013	12 Years a Slave	In the antebellum United States, Solomon
2012	Argo	Acting under the cover of a Hollywood pr
2011	The Artist	An egomaniacal film star develops a relat
2010	The King's Speech	The story of King George VI, his unexpe
2008	The Hurt Locker	During the Iraq War, a Sergeant recently
2008	Slumdog Millionaire	A Mumbai teenager reflects on his life aft
2007	No Country for Old Men	Violence and mayhem ensue after a hunte
2006	The Departed	An undercover cop and a mole in the pol
2004	Million Dollar Baby	Frankie, an ill-tempered old coach, reluct
2004	Crash	Los Angeles citizens with vastly separate
2003	The Lord of the Rings: The Return of the King	Gandalf and Aragorn lead the World of M
2002	Chicago	Two death-row murderesses develop a fier
2001	A Beautiful Mind	After John Nash, a brilliant but asocial m
2000	Gladiator	A former Roman General sets out to exac
1999	American Beauty	A sexually frustrated suburban father ha
1998	Shakespeare in Love	The world's greatest ever playwright, Wi
1997	Titanic	A seventeen-year-old aristocrat falls in lo
1996	The English Patient	At the close of World War II, a young nu
1995	Braveheart	Scottish warrior William Wallace leads hi
1994	Forrest Gump	The presidencies of Kennedy and Johnson
1993	Schindler's List	In German-occupied Poland during World
1992	Unforgiven	Retired Old West gunslinger William Mu
1991	The Silence of the Lambs	A young F.B.I. cadet must receive the he
1990	Dances with Wolves	Lieutenant John Dunbar, assigned to a re
1989	Driving Miss Daisy	An old Jewish woman and her African-A
1988	Rain Man	After a selfish L.A. yuppie learns his estr
1987	The Last Emperor	Dramatization of China's last emperor, P
1986	Platoon	Chris Taylor, a neophyte recruit in Vietn
1985	Out of Africa	In 20th-century colonial Kenya, a Danish
1984	Amadeus	The life, success and troubles of Wolfgang
1983	Terms of Endearment	Follows hard-to-please Aurora looking for
1982	Gandhi	The life of the lawyer who became the fa
1981	Chariots of Fire	Two British track athletes, one a determi
1980	Ordinary People	The accidental death of the older son of a
1979	Kramer vs. Kramer	After his wife leaves him, a work-obsessed
1978	The Deer Hunter	An in-depth examination of the ways in w
1977	Annie Hall	Alvy Singer, a divorced Jewish comedian
1976	Rocky	A small-time Philadelphia boxer gets a su
1975	One Flew Over the Cuckoo's Nest	In the Fall of 1963, a Korean War veteran
1974	The Godfather Part II	The early life and career of Vito Corleone
1973	The Sting	Two grifters team up to pull off the ultim
1972	The Godfather	Don Vito Corleone, head of a mafia famil
1971	The French Connection	A pair of NYPD detectives in the Narcot
1970	Patton	The World War II phase of the career of

22.2 Group Activity 2

22.2.1 a. Scrape the names, scores, and years of most popular TV shows on IMDB:

www.imdb.com/chart/tvmeter. Create a data frame called `tvshows` with four variables (`rank`, `name`, `score`, `year`)

Note:

It's easier to use the `SelectorGadget` and choose the CSS selectors wisely. Otherwise, you'll have more cleaning to do after scraping.

Click for answer

```
page <- read_html("http://www.imdb.com/chart/tvmeter")
name <- page %>%
  html_nodes(".titleColumn a") %>%
  html_text()

ranks <- page %>%
  html_nodes(".velocity") %>%
  html_text() %>%
  str_extract("\\d+") %>%
  as.numeric()

scores <-  page %>%
  html_nodes(".imdbRating") %>%
  html_text() %>%
  str_extract("\\d+.\\d+") %>%
  as.numeric()

# If you don't use the gadget selector carefully,
# more string manipulation is needed here

years <- page %>%
  html_nodes("a+ .secondaryInfo") %>%
  html_text() %>%
  str_extract("\\d+") %>%
  as.numeric()
```

```
tvshows <- tibble(
  rank = ranks,
```

```
  name = name,
  score = scores,
  year = years
)

tvshows
```

```
# A tibble: 100 x 4
    rank name                               score  year
   <dbl> <chr>                              <dbl> <dbl>
 1     1 Queen Charlotte: A Bridgerton Story  7    2023
 2     2 Silo                                 8.3  2023
 3     3 Succession                           8.8  2018
 4     4 Ted Lasso                            8.8  2020
 5     5 The Diplomat                         8.1  2023
 6     6 Love & Death                         7.6  2023
 7     7 Citadel                              6.2  2023
 8     8 Queen Cleopatra                      1    2023
 9     9 Barry                                8.4  2018
10    10 The Marvelous Mrs. Maisel            8.7  2017
# ... with 90 more rows
```

22.3 Group Activity 3

22.3.1 a. Scrape the first table in `List_of_NASA_missions` wiki page. Additionally, use `janitor::clean_names()` to clean the column names and store the resulting table as `NASA_missions.csv` in your working folder.

Click for answer

```
# extract data from
# the first table on the page
wiki_NASA <- "https://en.wikipedia.org/wiki/List_of_NASA_missions"
NASA_missions <- bow(wiki_NASA) %>%scrape() %>%
  html_nodes("table") %>%
  .[[1]] %>%
  html_table() %>%
  janitor::clean_names()
```

```r
# Exporting data to CSV
readr::write_csv(NASA_missions, "NASA_missions.csv")
```

22.3.2 b. Now, write a code snippet to scrape all the URLs from the anchor tags (`<a>`) on a given Wikipedia page, convert the relative URLs to absolute URLs, and store the results in a tibble and save it as `websites.csv` in your working folder.

Click for answer

```r
# extract URLs
websites <-  bow(wiki_NASA) %>% scrape() %>%
  html_nodes("a") %>%
  html_attr("href") %>%
  url_absolute("https://en.wikipedia.org/")
```

```r
# Exporting data to CSV
readr::write_csv(websites, "websites.csv")
```

Chapter 23

Class Activity 17

```
# load the necessary libraries
library(tidyverse)
library(stringr)
library(purrr)
library(ggthemes)
library(rvest)
library(polite)
```

23.1 Group Activity 1

1. Go to the the numbers webpage and extract the table on the front page.

Click for answer

```
session1 <- bow(url = "https://www.the-numbers.com/movie/budgets/all") %>% scrape() %>%
  html_nodes(css = "table") %>%
  html_table()

table_base <- session1 %>% .[[1]]
```

2. Find out the number of pages that contain the movie table, while looking for the changes in the url in the address bar. How does the url changes when you go to the next page?

Click for answer

Answer: The starting count of the movie gets concatenated to the url in increments of 100.

3. Write a for loop to store all the data in multiple pages to a single data frame.

Click for answer

```
library(tidyverse)
library(rvest)

new_urls <- "https://www.the-numbers.com/movie/budgets/all/"

# Create an empty data frame
df1 <- list()

# Generate a vector of indices
index <- seq(1, 6301, 100)
```

```
# Loop through indices, scrape data, and bind the resulting data frames
for (i in 1:length(index)) {
  url <- str_glue("{new_urls}{index[i]}")
  webpage <- read_html(url)
  table_new <- html_table(webpage)[[1]] %>%
    tibble::as_tibble(.name_repair = "unique") %>%
    janitor::clean_names() %>%
    mutate(x1 = as.character(x1))
  df1[[i]] <- table_new
}

df1_final <- do.call(rbind, df1)
df1_final1 <- reduce(df1, dplyr::bind_rows)
```

```
# alternate using map/lapply
urls <- map(index, function(i) str_glue({new_urls}, {index[i]}))
urls <- map(index, ~str_glue({new_urls}, {.x}))



sessions <- map(urls, ~read_html(.x) %>%
                  html_nodes("table") %>%
                  html_table() %>%
                  tibble::as_tibble(.name_repair = "unique") %>%
                  janitor::clean_names())

movies_data <- do.call(rbind, lapply(1:length(urls), function(i) sessions[[i]][[1]]))
glimpse(movies_data)
```

```
Rows: 6,393
```

```
Columns: 6
$ ``               <chr> "1", "2", "3", "4", "5", "6", "7"~
$ ReleaseDate      <chr> "Dec 9, 2022", "Apr 23, 2019", "M~
$ Movie            <chr> "Avatar: The Way of Water", "Aven~
$ ProductionBudget <chr> "$460,000,000", "$400,000,000", "~
$ DomesticGross    <chr> "$684,060,555", "$858,373,000", "~
$ WorldwideGross   <chr> "$2,319,748,227", "$2,794,731,755~
```

23.2 Group Activity 2

1. Go to the the numbers webpage and extract the table on the front page.

Click for answer

```
session1 <- bow(url = "https://www.scrapethissite.com/pages/forms/") %>% scrape() %>%
  html_nodes(css = "table") %>%
  html_table()

table_base <- session1 %>% .[[1]]
```

2. Find out the number of pages that contain the movie table, while looking for the changes in the url in the address bar. How does the url changes when you go to the next page?

Click for answer

Answer: The url field has `?page_num=` added with the number of pages running from 1 to 24.

3. Write a for loop to store all the data in multiple pages to a single data frame.

Click for answer

```
library(tidyverse)
library(rvest)

new_urls <- "http://scrapethissite.com/pages/forms/?page_num="

# Create an empty data frame
df2 <- list()

# Generate a vector of indices
index <- seq(1, 24)
```

```r
# Loop through indices, scrape data, and bind the resulting data frames
for (i in index) {
  url <- str_glue("{new_urls}{i}")
  webpage <- read_html(url)
  table_new <- html_table(webpage)[[1]] %>%
    tibble::as_tibble(.name_repair = "unique")
  df2[[i]] <- table_new
}

df2_final <- do.call(rbind, df2)
df2_final1 <- reduce(df2, dplyr::bind_rows)
```

```r
# alternate using map
urls <- map(index, function(i) str_glue({new_urls}, {i}))
urls <- map(index, ~str_glue({new_urls}, {.x}))


sessions <- map(urls, ~read_html(.x) %>%
                  html_nodes("table") %>%
                  html_table() %>%
                  tibble::as_tibble(.name_repair = "unique") %>%
                  janitor::clean_names())

sports_data <- do.call(rbind, lapply(1:length(urls), function(i) sessions[[i]][[1]]))
sports_data1 <- map_df(1:length(urls), ~sessions[[.x]][[1]])

glimpse(sports_data)
```

```
Rows: 582
Columns: 9
$ `Team Name`           <chr> "Boston Bruins", "Buffalo Sab~
$ Year                  <int> 1990, 1990, 1990, 1990, 1990,~
$ Wins                  <int> 44, 31, 46, 49, 34, 37, 31, 4~
$ Losses                <int> 24, 30, 26, 23, 38, 37, 38, 2~
$ `OT Losses`           <int> NA, NA, NA, NA, NA, NA, NA, N~
$ `Win %`               <dbl> 0.550, 0.388, 0.575, 0.613, 0~
$ `Goals For (GF)`      <int> 299, 292, 344, 284, 273, 272,~
$ `Goals Against (GA)`  <int> 264, 278, 263, 211, 298, 272,~
$ `+ / -`               <int> 35, 14, 81, 73, -25, 0, -38, ~
```

4. Create an interactive bar plot to display the number of wins per team and year.

Click for answer

```
library(plotly)

bar_plot <- ggplot(sports_data, aes(x = Year, y = Wins, fill = `Team Name`)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Number of Wins per Team and Year") +
  theme(legend.position = "bottom")

plotly_bar <- ggplotly(bar_plot)
plotly_bar
```

5. Explore the relationship between the number of goals scored and the number of goals against for each team in each year with an interactive scatter plot.

Click for answer

```
scatter_plot <- ggplot(sports_data, aes(x = `Goals For (GF)`, y = `Goals Against (GA)`, color = `
  geom_point() +
  labs(title = "Goals Scored vs. Goals Against per Team and Year") +
  theme(legend.position = "bottom") +
  xlab("Goals Scored (GF)") +
  ylab("Goals Against (GA)")
```

```
plotly_scatter <- ggplotly(scatter_plot, tooltip = "text")
plotly_scatter
```

6. Visualize team performance per year (wins, losses, and OT losses) using a stacked bar plot.

Click for answer

```
stacked_bar_plot <- ggplot(sports_data, aes(x = Year, fill = `Team Name`)) +
  geom_bar(aes(y = Wins), position = "stack", stat = "identity", width = 0.4, alpha = 0.8) +
  geom_bar(aes(y = Losses), position = "stack", stat = "identity", width = 0.4, alpha = 0.8) +
  geom_bar(aes(y = `OT Losses`), position = "stack", stat = "identity", width = 0.4, alpha = 0.8)
  labs(title = "Team Performance per Year (Wins, Losses, and OT Losses)") +
  theme(legend.position = "bottom") +
  xlab("Year") +
  ylab("Number of Games")

plotly_stacked_bar <- ggplotly(stacked_bar_plot)
plotly_stacked_bar
```

Chapter 24

Class Activity 18

```
# load the necessary libraries
library(tidyverse)
library(shiny)
library(readr)
library(janitor)
library(purrr)
library(lubridate)
library(plotly)
library(DT)
library(ggthemes)
library(rvest)
library(polite)
```

24.1 Shiny App Structure

24.2 User Interface (UI)

UI is just a web document that the user gets to see, it's HTML that you write using Shiny's functions. The UI is responsible for creating the layout of the app and telling Shiny exactly where things go. The server is responsible for the logic of the app; it's the set of instructions that tell the web page what to show when the user interacts with the page.

24.2.1 Hello World!

```
ui <- fluidPage("Hello World!")
server <- function(input, output) {}
shinyApp(ui = ui, server = server, options = list(height = 200))
```

24.2.2 Add more information

```
fluidPage(
  titlePanel("Tracking Covid in Minnesota"),
  h1("Some nice header"),
  "elements1",
  "elements2",
  br(),
  "things1",
  strong("things2")
)
```

24.2.3 Add a layout

```
sidebarLayout(
  sidebarPanel("our inputs will go here"),
  mainPanel("the results will go here")
)
```

24.3 Read Data

```
table_usafacts <- bow(url = "https://usafacts.org/visualizations/coronavirus-covid-19-s
  scrape() %>%  html_nodes("a") %>%       # find all links
  html_attr("href") %>%     # get the url
  str_subset(".csv")        # find those that end in csv
```

```
library(lubridate)
covid_data <- read_csv(table_usafacts[2]) %>% filter(State == "MN") %>%
  select(-countyFIPS, -StateFIPS, -State) %>%
  filter(!row_number() %in% c(1)) %>%
  pivot_longer(-1, names_to = "Dates", values_to = "Cases") %>%
```

```
  janitor::clean_names() %>%
  mutate(county_name = str_remove(county_name, " County"),
         dates = ymd(dates),
         counties = as.factor(str_remove(county_name, " County")),
         month = month(dates),
         year = year(dates)) %>%
  select(-county_name)
head(covid_data)
```

```
# County level data
county_names <- covid_data %>% pull(counties) %>% unique()
county_data <- lapply(1:length(county_names), function(i) filter(covid_data, counties == county_r
county_data %>% pluck(which(county_names == "Dakota"))
```

24.3.1 A complete skeleton app

```
ui <- fluidPage(
  titlePanel("Tracking Covid in Minnesota"),
  sidebarLayout(
    sidebarPanel("our inputs will go here"),
    mainPanel("the results will go here")
  )
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

24.3.2 Add inputs to the UI

```
ui <- fluidPage(
  titlePanel("Tracking Covid in Minnesota"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("monthInput", "Month", 0, 12, c(3, 6)),
      radioButtons("yearInput", "Year",
                   choices = c("2020", "2021", "2022", "2023"),
                   selected = "2022"),
       selectInput(inputId = "dv", label = "County",
                         choices = levels(covid_data$counties),
```

```
                        selected = c("Aitkin"))
    ),
    mainPanel("the results will go here")
  )
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server, options = list(height = 800))
```

24.3.3 Add placeholders for outputs

```
ui <- fluidPage(
  titlePanel("Tracking Covid in Minnesota"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("monthInput", "Month", 0, 12, c(3, 6)),
      radioButtons("yearInput", "Year",
                  choices = c("2020", "2021", "2022", "2023"),
                  selected = "2022"),
       selectInput(inputId = "dv", label = "County",
                        choices = levels(covid_data$counties),
                        selected = c("Aitkin"))
    ),
    mainPanel(
      plotOutput("coolplot"),
      br(), br(),
      tableOutput("results")
    )
  )
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server, options = list(height = 800))
```

24.4 Server Function

`Server` function will be responsible for listening to changes to the inputs and creating outputs to show in the app.

24.4.1 Implementation 1

```
ui1 <- fluidPage(
    titlePanel("Tracking Covid in Minnesota"),
    sidebarLayout(
        sidebarPanel(
          sliderInput("monthInput", "Month", 0, 12, c(3, 6)),
          radioButtons("yearInput", "Year",
                  choices = c("2020", "2021", "2022", "2023"),
                  selected = "2020"),
          selectInput(inputId = "dv", label = "County",
                        choices = levels(covid_data$counties),
                        selected = c("Aitkin"))
    ),
        mainPanel(
            plotOutput(outputId = "plot"), br(),
            DT::dataTableOutput(outputId = "table")
        )
    )
)
```

```
server1 <- function(input, output) {
    filtered_data <- reactive({
        subset(covid_data,
               counties %in% input$dv &
               month >= input$monthInput[1] & month <= input$monthInput[2] &
          year == input$yearInput) })

    output$plot <- renderPlot({
        ggplot(filtered_data(), aes(x=dates, y=cases, color=counties)) + theme_economist_white()+
                geom_point(alpha=0.5, color = "blue") + theme(legend.position = "none") +
                    ylab("Number of Cases") + xlab("Date")})

    output$table <- DT::renderDataTable({
        filtered_data()})

}
```

```
app1 <- shinyApp(ui = ui1, server = server1, options = list(height = 1200))
app1
```

Chapter 25

Class Activity 19

```r
# List of required packages
pkgs <- c("tidyverse", "ggthemes", "rvest", "polite", "ggiraph", "shiny", "bslib", "stringr")

# Check and install missing packages
for (pkg in pkgs) {
  if (!requireNamespace(pkg, quietly = TRUE))
    install.packages(pkg)
}
```

25.1 Reactive

25.1.1 a. Create a Shiny app that allows users to input a number and displays its square and cube in real-time using reactive expressions and text outputs.

```r
library(shiny)

ui <- fluidPage(
  numericInput("number", "Enter a number", value = 1, min = 1),
  textOutput("square"),
  textOutput("cube")
)

server <- function(input, output) {
  # Create a reactive expression to store the number
```

```
  number <- reactive({
    input$number
  })

  output$square <- renderText({
    paste("Square of the number:", number()^2)
  })

  output$cube <- renderText({
    paste("Cube of the number:", number()^3)
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.2 Action button

25.2.1 b. Create a Shiny app that takes user-selected gear numbers and generates a polar coordinate plot of miles per gallon (mpg) using the mtcars dataset and ggplot2.

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  selectInput("gear", "Number of Gears",
              choices = unique(mtcars$gear),
              selected = 3),
  actionButton("goButton", "Go!"),
  plotOutput("distPlot")
)

server <- function(input, output) {
  output$distPlot <- renderPlot({

    input$goButton

    # Use isolate() to avoid dependency on input$gear
    selected_gear <- isolate(input$gear)
    gear_data <- subset(mtcars, gear == selected_gear)
```

```
    # Plot the polar coordinate plot of miles per gallon (mpg) for the selected number of gears
    p <- ggplot(gear_data, aes(x = factor(1), y = mpg, fill = factor(cyl))) +
      geom_bar(stat = "identity", width = 1) +
      coord_polar(theta = "y") +
      theme_void() +
      labs(title = stringr::str_c("Polar Coordinate Plot for ", selected_gear, " Gears"),
           fill = "Cylinders")

    print(p)
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.3 Reactive Values

25.3.1 c. Create a Shiny app with increment and decrement buttons to manipulate a counter and display its value in real-time using reactive values and text output.

```
library(shiny)

ui <- fluidPage(
  actionButton("increment", "Increment"),
  actionButton("decrement", "Decrement"),
  textOutput("counter")
)

server <- function(input, output) {
  # Create a reactiveValues object to store the counter
  counter <- reactiveValues(value = 0)

  observeEvent(input$increment, {
    counter$value <- counter$value + 1
  })

  observeEvent(input$decrement, {
    counter$value <- counter$value - 1
  })

  output$counter <- renderText({
```

```
    paste("Counter value:", counter$value)
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.4 Isolate

25.4.1 d. Create a Shiny app that allows users to select two variables from the mtcars dataset and generates a color-coded scatter plot, which refreshes only when an action button is clicked, using ggplot2 and isolate() function.

```
ui <- fluidPage(
  theme = bslib::bs_theme(version = 4, bootswatch = "flatly"),
  titlePanel("Panel Plot Demo with Shiny and bslib"),

  sidebarLayout(
    sidebarPanel(
      selectInput("var1", "Variable 1:", choices = colnames(mtcars), selected = "mpg")
      selectInput("var2", "Variable 2:", choices = colnames(mtcars), selected = "wt"),
      actionButton("refresh", "Refresh Plot")
    ),

    mainPanel(
      plotOutput("panelPlot")
    )
  )
)

server <- function(input, output, session) {

  output$panelPlot <- renderPlot({
    input$refresh

    isolate({
       panel_plot <- ggplot(mtcars, aes(x = !!sym(input$var1), y = !!sym(input$var2)))
        geom_point(aes(color = factor(cyl))) +
        theme_minimal() +
        labs(x = input$var1, y = input$var2, color = "Cylinders") +
```

```
      theme(legend.position = "bottom")

    print(panel_plot)
   })
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.5 EventReactive

25.5.1 e. Create a Shiny app that takes user-selected MPG threshold and calculates the number of observations above the threshold, updating the results only when an action button is clicked, using eventReactive() and renderText().

```
library(shiny)

ui <- fluidPage(
  sliderInput("mpg_threshold", "MPG Threshold", min(mtcars$mpg), max(mtcars$mpg), value = 20, ste
  actionButton("goButton", "Go!"),
  textOutput("num_obs")
)

server <- function(input, output) {
  # Use eventReactive() to update the number of observations above the threshold
  num_obs_above_threshold <- eventReactive(input$goButton, {
    sum(mtcars$mpg > input$mpg_threshold)
  })

  output$num_obs <- renderText({
    paste("Number of observations above the threshold:", num_obs_above_threshold())
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.5.2 More with isolate

```
library(shiny)
library(bslib)
library(ggplot2)

ui <- fluidPage(
  theme = bs_theme(version = 4, bootswatch = "flatly"),
  titlePanel("Interactive Plot Demo with Shiny, bslib, and ggplot2"),

  sidebarLayout(
    sidebarPanel(
      selectInput("var1", "Variable 1:", choices = colnames(mtcars), selected = "mpg")
      selectInput("var2", "Variable 2:", choices = colnames(mtcars), selected = "wt"),
      actionButton("refresh", "Refresh Plot")
    ),

    mainPanel(
      plotOutput("Plot")
    )
  )
)

server <- function(input, output, session) {

  output$Plot <- renderPlot({
    input$refresh

    isolate({
      p <- ggplot(data = mtcars,
                  mapping = aes(x = !!sym(input$var1), y = !!sym(input$var2), color = :
        geom_point(size = 3) +
        scale_color_viridis_d(option = "viridis", name = "Cylinders") +
        theme_minimal() +
        theme(legend.position = "bottom") +
        labs(x = input$var1, y = input$var2)

      print(p)
    })
  })
}

shinyApp(ui, server, options = list(height = 800))
```

25.6 Exercises

25.6.1 Question 1

Explain how to create a Shiny app that takes a user-inputted number and displays its square and cube in real-time using reactive expressions and text outputs.

Answer: Create a numericInput for the user to enter a number, and two textOutput elements for square and cube. In the server function, define a reactive expression to store the user input. Use renderText to display the square, cube, and factorial of the number.

25.6.2 Question 2

How can you create a Shiny app that allows users to select two variables from the mtcars dataset and generates a color-coded scatter plot, which refreshes only when an action button is clicked, using ggplot2 and isolate() function?

Answer: Use selectInput to let users choose variables, and an actionButton to trigger plot refresh. In the server function, create a renderPlot that depends on the action button input. Within renderPlot, use isolate() to access the selected variables without triggering reactivity. Create the scatter plot with ggplot2, and display it using print().

25.6.3 Question 3

How do you create a Shiny app that takes a user-selected MPG threshold and calculates the number of observations above the threshold, updating the results only when an action button is clicked, using eventReactive() and renderText()?

Answer: Create a sliderInput for the MPG threshold and an actionButton to trigger the calculation. In the server function, use eventReactive() to perform the calculation based on the action button input. Then, use renderText() to display the result, which updates only when the action button is clicked.

Chapter 26

Class Activity 20

```
# load the necessary libraries
library(tidyverse)
library(shiny)
library(rvest)
library(polite)
library(leaflet)
library(sp)
library(maptools)
library(rgeos)
library(stringr)
library(RColorBrewer)
library(terra)
library(raster)
```

26.1 Group Activity 1

26.1.1 a. Explore COVID-19 vaccination rates across the Midwest with this R script, which scrapes data, processes it, and creates an interactive, state-specific leaflet map for clear visualization.

Click for answer

Answer:

```r
# Scrape the data
covid_final <- read_html("https://usafacts.org/visualizations/covid-vaccine-tracker-sta
  html_elements(css = "table") %>% html_table() %>% .[[1]] %>%
  janitor::clean_names() %>%
  mutate_at(2:4, parse_number) %>% mutate(state = str_to_lower(state)) %>%
  filter(state %in% c("ohio", "indiana", "michigan", "illinois", "missouri", "wisconsi
                                      "iowa", "kansas", "nebraska", "south dakota", "no

# Midwest
USA_Midwest <- maps::map("state",
                         regions = c("ohio", "indiana", "michigan", "illinois", "misso
                                     "iowa", "kansas", "nebraska", "south dakota", "no
                         plot = FALSE, fill = TRUE) %>%
  map2SpatialPolygons(IDs = str_remove(.$names, "(?=:).+"))

# Merge the data and the map
map <- SpatialPolygonsDataFrame(USA_Midwest, covid_final, match.ID = FALSE)

# Create bins and color palette
bins <- seq(min(map$percent_fully_vaccinated), max(map$percent_fully_vaccinated), leng
pal <- colorBin("viridis", domain = map$percent_fully_vaccinated, bins = bins)

# Create labels
labels <- sprintf("<strong> %s </strong> <br/> Observed: %s", str_to_upper(map$state),
  lapply(htmltools::HTML)

# Plot the map
leaflet(map) %>%
  addTiles() %>%
  setView(lng = -93.1616, lat = 44.4583, zoom = 4) %>%
  addPolygons(
    color = "grey",
    weight = 1,
    fillColor = ~pal(percent_fully_vaccinated),
    fillOpacity = 0.7,
    highlightOptions = highlightOptions(weight = 5),
    label = labels
  ) %>%
  addLegend(
    pal = pal,
    values = ~percent_fully_vaccinated,
    opacity = 0.5,
    title = "Percent Vaccn.",
    position = "bottomright"
  )
```

26.2 Group Activity 2

26.2.1 a. Explore the COVID-19 vaccination status across various US regions using this interactive Shiny app. The script extracts data from USAFacts, presents user options to customize the map projection, color palette, and region, and dynamically generates a Leaflet map to visualize the selected data.

Click for answer

Answer:

```
# Scrape the data
covid_final <- read_html("https://usafacts.org/visualizations/covid-vaccine-tracker-states/state/
  html_elements(css = "table") %>% html_table() %>% .[[1]] %>%
  janitor::clean_names() %>%
  mutate_at(2:4, parse_number) %>% mutate(state = str_to_lower(state))



# UI
ui <- fluidPage(
  titlePanel("Map of States and their vaccination status"),
  sidebarLayout(
    sidebarPanel(
      selectInput("myvar", "What to project?", choices = names(covid_final)[2:4], selected = name
      selectInput("palette", "Choose color palette:", choices = c("viridis", "magma", "inferno",
      selectInput("region", "Select region:", choices = c("Northeast", "Southeast", "Midwest", "S
      actionButton("goButton", "Update map")
    ),
    mainPanel(
      leafletOutput("myplot")
    )
  )
)


# UI
ui <- fluidPage(
  titlePanel("Map of States and their vaccination status"),
  sidebarLayout(
```

```
    sidebarPanel(
      selectInput("myvar", "What to project?", choices = names(covid_final)[2:4], selec
      selectInput("palette", "Choose color palette:", choices = c("viridis", "magma", "
      selectInput("region", "Select region:", choices = c("Northeast", "Southeast", "Mi
      actionButton("goButton", "Update map")
    ),
    mainPanel(
      leafletOutput("myplot")
    )
  )
)

# Server
server <- function(input, output, session) {
  map_data <- eventReactive(input$goButton, {
    req(input$region)

    region_states <- switch(input$region,
                        "Northeast" = c("maine", "new hampshire", "vermont", "massachus
                                        "connecticut", "new york", "pennsylvania", "new
                        "Southeast" = c("delaware", "maryland", "virginia", "west virgi
                                        "south carolina", "georgia", "florida", "kentuc
                                        "mississippi", "alabama", "arkansas", "louisian
                        "Midwest" = c("ohio", "indiana", "michigan", "illinois", "wisco
                                      "iowa", "missouri", "kansas", "nebraska", "south
                        "South" = c("oklahoma", "texas", "new mexico", "arizona"),
                        "West" = c("colorado", "wyoming", "montana", "idaho", "utah", "
                                   "oregon", "washington"))



    covid_region <- isolate(covid_final %>%
      filter(state %in% region_states))

    USA_region <- maps::map("state", regions = region_states, plot = FALSE, fill = TRUE
      map2SpatialPolygons(IDs = str_remove(.$names, "(?=:).+"))

    SpatialPolygonsDataFrame(USA_region, covid_region, match.ID = FALSE)
  }, ignoreNULL = FALSE)

  output$myplot <- renderLeaflet({
    req(map_data()) # ensure that map_data() is available

    bins <- seq(min(map_data()[[input$myvar]]), max(map_data()[[input$myvar]]), length
    pal <- colorBin(input$palette, domain = map_data()[[input$myvar]], bins = bins)
```

```
    labels <- sprintf("<strong> %s </strong> <br/> Observed <strong> %s </strong>  : %s", str_to_
      lapply(htmltools::HTML)

    m <- leaflet(map_data()) %>%
      addTiles() %>%  setView(lng = -93.1616, lat = 40.4583, zoom = 3.5) %>%
      addPolygons(
        color = "grey",
        weight = 1,
        fillColor = ~pal(get(input$myvar)),
        fillOpacity = 0.7,
        highlightOptions = highlightOptions(weight = 5),
        label = labels
      ) %>%
      addLegend(
        pal = pal,
        values = ~get(input$myvar),
        opacity = 0.5,
        title = str_to_title(input$myvar),
        position = "bottomright"
      )
    m
  })
}

# Run the app
shinyApp(ui = ui, server = server)
```

26.2.2 b. Modify the app in 2a to add an "All" option in the "Select region" input to allow users to view all states at once.

```
# your r-code
```

26.2.3 c. Open an account on shinyapps.io and follow the steps to deploy one of the apps to shinyapps.io.

Chapter 27

Class Activity 21

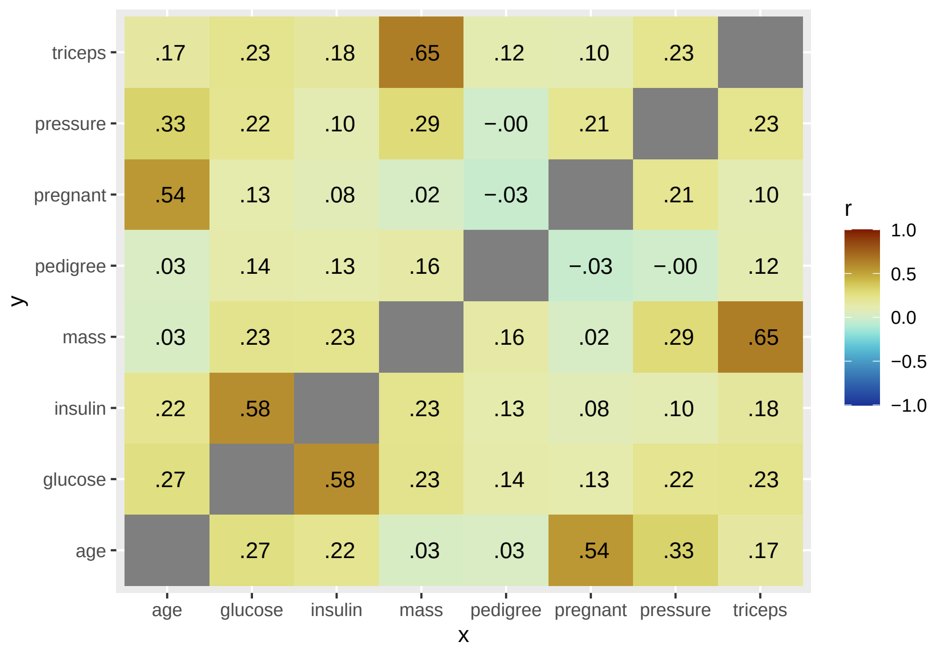
```
# load the necessary libraries
library(tidyverse)
library(tidymodels)
library(mlbench)
library(janitor)
library(parsnip)
library(kknn)
library(paletteer)
library(corrr)
library(forcats)
library(ggthemes)
```

27.1 Group Activity 1

```
# Load the data
data(PimaIndiansDiabetes2)
db <- PimaIndiansDiabetes2

# correlation plot of the variables
db %>%
  select(-diabetes) %>%  # only numerical variables
  correlate() %>%
  stretch() %>%
  ggplot(aes(x, y, fill = r)) +
  geom_tile() +
```

```
  geom_text(aes(label = as.character(fashion(r)))) +
  scale_fill_paletteer_c("scico::roma", limits = c(-1, 1), direction = -1)
```



a. Let's perform all the steps involved in classifying whether a patient with certain glucose and insulin would have diabetes or not using `parsnip` package.

Click for answer

Answer:

```
# 1 Prepare raw data
db_raw <- db %>% drop_na() %>%  select(glucose, insulin, diabetes)
```

```
# 2 Create a recipe for data pre-processing
db_recipe <- recipe(diabetes ~ ., data = db_raw) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors()) %>%
  prep()
```

```
# 3 Apply the recipe to the data set
db_scaled <-  bake(db_recipe, db_raw)
```

```
# 4 Create a model specification
knn_spec <- nearest_neighbor(mode = "classification",
                             engine = "kknn",
                             neighbors = 5)
```

```
# 5 Fit the model on the pre-processed data
knn_fit <- knn_spec %>%
 fit(diabetes ~ ., data = db_scaled)
```

```
# 6 Classify
# These are standardized value!!
new_observations <- tibble(glucose = c(1, 2), insulin = c(-1, 1))
predict(knn_fit, new_data = new_observations)
```

```
# A tibble: 2 x 1
  .pred_class
  <fct>
1 neg
2 pos
```

b. We already know the labels of some of the patients in the dataset. How well does the model predict their diabetes status? We will see more of this in the coming lectures, but for now try to compare the results for the first 10 cases in the dataset.

Click for answer

Answer:

```
scaled_observations <- db_scaled[1:50,]
predictions <- predict(knn_fit, new_data = scaled_observations)
bind_cols(scaled_observations, predictions, db_raw %>%
            select(diabetes) %>%
            slice(1:50)) %>% knitr::kable()
```

glucose	insulin	diabetes...3	.pred_class	diabetes...5
-1.0896533	-0.5221747	neg	neg	neg
0.4657189	0.1005024	pos	pos	pos
-1.4460927	-0.5726620	pos	neg	pos
2.4099341	3.2559608	pos	pos	pos
2.1507054	5.8055711	pos	pos	pos
1.4054229	0.1594043	pos	pos	pos
-0.1499493	0.6222049	pos	pos	pos
-0.6360031	-0.6147348	neg	neg	neg
-0.2471600	-0.5053456	pos	neg	pos
0.1092794	0.6642776	neg	neg	neg
0.6601404	-0.0846178	pos	pos	pos
0.0768759	-0.3454690	pos	neg	pos
-0.8304246	-0.1351051	neg	neg	neg
0.7249476	-0.3875418	neg	neg	neg
1.1461942	0.7484232	pos	pos	pos
-1.1220569	-0.8587569	neg	neg	neg
-0.6360031	0.3024518	neg	neg	neg
-0.3767744	0.4286701	pos	neg	pos
1.8590732	-0.7241240	neg	pos	neg
1.5674409	0.7063504	pos	pos	pos
-0.6360031	-0.6231494	neg	neg	neg
-0.7008102	-1.0102189	neg	neg	neg
-1.1220569	-1.1196081	neg	neg	neg
1.7294588	1.2112238	pos	pos	pos
0.8869655	1.5646351	neg	pos	neg
2.0858983	1.2448820	pos	pos	pos
-0.7332138	-0.3875418	neg	neg	neg
-0.5711959	-0.1182760	neg	neg	neg
0.5953333	-0.2360798	neg	pos	neg
-0.8952318	-0.9933898	neg	neg	neg
0.7573512	-0.4716874	neg	neg	neg
-0.7332138	-0.5558329	pos	neg	pos
0.5305261	-0.1351051	neg	neg	neg
0.2064902	0.9587871	neg	neg	neg
-1.2840748	-0.7157095	neg	neg	neg
-0.4091780	-0.2613235	neg	neg	neg
-0.7332138	-0.7157095	neg	neg	neg
0.4333153	-0.3875418	pos	pos	pos
0.0120687	0.1678189	neg	neg	neg
-1.3488820	-0.9092442	neg	neg	neg
0.6277368	-0.7746114	neg	neg	neg
0.6925440	0.6053758	neg	neg	neg
-1.6729179	-0.6736367	neg	neg	neg
-0.9600389	-0.7746114	neg	neg	neg
-0.0203349	0.5380593	pos	neg	pos
-1.3488820	-0.9765607	neg	neg	neg
0.1092794	-0.0341305	neg	neg	neg
0.6925440	-0.1351051	neg	pos	neg
-1.2840748	-1.1616809	neg	neg	neg
-0.8952318	-1.0102189	pos	neg	pos

What is the accuracy percentage?

Answer:

```
sum(predictions == db_raw %>% select(diabetes) %>% slice(1:50))/50
```

```
[1] 0.78
```

c. Repeat part b. with a different model fitted with different number of neighbors. See if the accuracy percentage change in this new setting.

Click for answer

Answer:

```
knn_spec <- nearest_neighbor(mode = "classification",
                             engine = "kknn",
                             weight_func = "rectangular",
                             neighbors = 3)
knn_fit <- knn_spec %>%
 fit(diabetes ~ ., data = db_scaled)

scaled_observations <- db_scaled
predictions <- predict(knn_fit, new_data = scaled_observations)
# bind_cols(scaled_observations, predictions, db_raw %>%
#             select(diabetes)) %>% knitr::kable()

sum(predictions == db_raw %>% select(diabetes))/392
```

```
[1] 0.8239796
```

Chapter 28

Class Activity 22

```
# load the necessary libraries
library(tidyverse)
library(tidymodels)
library(mlbench)     # for PimaIndiansDiabetes2 dataset
library(janitor)
library(parsnip)
library(kknn)
library(ggthemes)
library(purrr)
library(forcats)
```

28.1 Group Activity 1

Load the `mlbench` package to get `PimaIndiansDiabetes2` dataset.

```
# Load the data - diabetes
data(PimaIndiansDiabetes2)
db <- PimaIndiansDiabetes2
db <- db %>% drop_na() %>% mutate(diabetes = fct_rev(factor(diabetes)))
db_raw <- db %>% select(glucose, insulin, diabetes)
```

a. Split the data `75-25` into training and test set using the following code.

Click for answer

Answer:

```r
set.seed(123)

db_split <- initial_split(db, prop = 0.75)

# Create training data
db_train <- db_split %>% training()

# Create testing data
db_test <- db_split %>%  testing()
```

b. Follow the steps to train a 7-NN classifier using the `tidymodels` toolkit

Click for answer

Answer:

```r
# define recipe and preprocess the data
db_recipe <- recipe(diabetes ~ ., data = db_raw) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors()) %>%
  prep()
```

```r
# specify the model
db_knn_spec7 <- nearest_neighbor(mode = "classification",
                             engine = "kknn",
                             weight_func = "rectangular",
                             neighbors = 7)
```

```r
# define the workflow
db_workflow <- workflow() %>%
  add_recipe(db_recipe) %>%
  add_model(db_knn_spec7)
```

```r
# fit the model
db_fit <- fit(db_workflow, data = db_train)
```

c. Classify the penguins in the `test` data frame.

Click for answer

Answer:

```
test_features <- db_test %>% select(glucose, insulin)
db_pred <- predict(db_fit, test_features)

db_results <- db_test %>%
  select(glucose, insulin, diabetes) %>%
  bind_cols(predicted = db_pred)

head(db_results, 6)
```

```
   glucose insulin diabetes .pred_class
4       89      94      neg         neg
7       78      88      pos         neg
15     166     175      pos         pos
19     103      83      neg         neg
32     158     245      pos         pos
36     103     192      neg         neg
```

28.2 Group Activity 2

Calculate the accuracy, sensitivity, specificity, and positive predictive value by hand using the following confusion matrix.

```
conf_mat(db_results, truth = diabetes, estimate = .pred_class)
```

```
          Truth
Prediction pos neg
       pos  17   8
       neg  12  61
```

Click for answer

Answer:

```
accuracy(db_results, truth = diabetes,
         estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.796
```

```
sens(db_results, truth = diabetes,
         estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.586
```

```
spec(db_results, truth = diabetes,
         estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.884
```

```
ppv(db_results, truth = diabetes,
         estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 ppv     binary          0.68
```

28.3 Extra: Code to recreate the plot in the slides for the `diabetes` dataset.

Click for answer

Answer:

```
metrics_for_k <- function(k, db_train, db_test){
db_knn_spec <- nearest_neighbor(mode = "classification",
                             engine = "kknn",
                             weight_func = "rectangular",
                             neighbors = k)

db_knn_wkflow <- workflow() %>%
  add_recipe(db_recipe) %>%
  add_model(db_knn_spec)
```

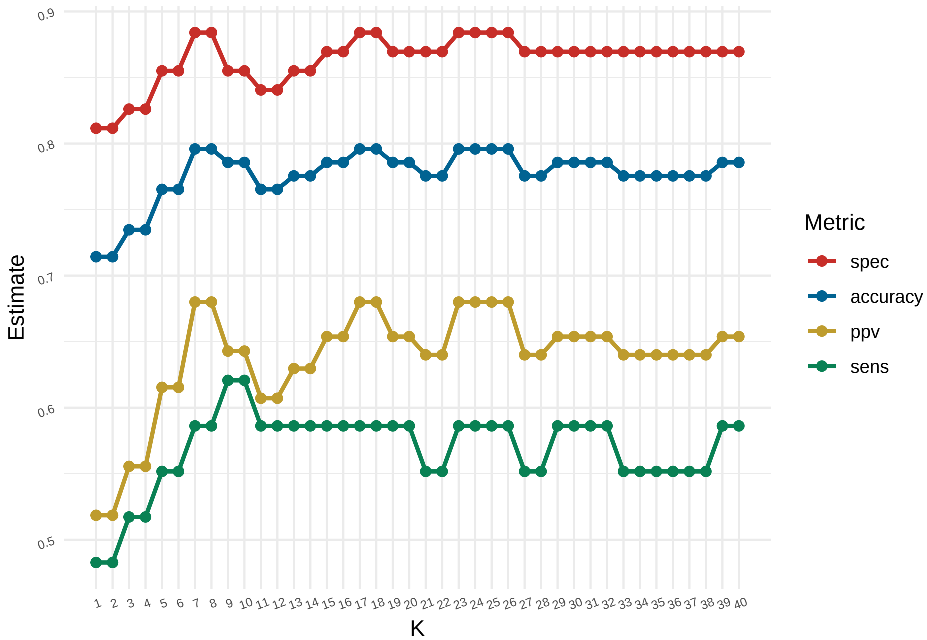
```
db_knn_fit <- fit(db_knn_wkflow, data = db_train)
test_features <- db_test %>% select(glucose, insulin)
nn1_pred <- predict(db_knn_fit, test_features, type = "raw")

db_results <- db_test %>%
  select(diabetes) %>%
  bind_cols(predicted = nn1_pred)
custom_metrics <- metric_set(accuracy, sens, spec, ppv)

metrics <- custom_metrics(db_results,
               truth = diabetes,
               estimate = predicted)
metrics <- metrics %>% select(-.estimator) %>% mutate(k = rep(k,4))

return(list = metrics)
}
```

```
optim.results %>%
  ggplot(aes(x = k, y = .estimate, color = forcats::fct_reorder2(.metric, k, .estimate ))) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  theme_minimal() +
  ggthemes::scale_color_wsj() +
  scale_x_continuous(breaks = k) +
  theme(panel.grid.minor.x = element_blank(),
        axis.text=element_text(size=6, angle = 20))+
  labs(color='Metric', y = "Estimate", x = "K")
```



Estimate
0.9
0.8
0.7
0.6
0.5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
K
Metric
spec
accuracy
ppv
sens

Chapter 29

Class Activity 23

```r
# load the necessary libraries
library(tidyverse)
library(tidymodels)
library(mlbench)     # for PimaIndiansDiabetes2 dataset
library(janitor)
library(yardstick) # extra package for getting metrics
library(parsnip) # tidy interface to models
library(ggthemes)
library(forcats)
library(probably)
library(yardstick)
```

29.1 Group Activity 1

Load the `mlbench` package to get `PimaIndiansDiabetes2` dataset.

```r
# Load the data - diabetes
data(PimaIndiansDiabetes2)
db <- PimaIndiansDiabetes2
db <- db %>% drop_na()
db_raw <- db %>% select(glucose, insulin, diabetes)

db_split <- initial_split(db_raw, prop = 0.80)
# Create training data
db_train <- db_split %>% training()
# Create testing data
db_test <- db_split %>% testing()
```

a. Creating the recipe

Click for answer

Answer:

```
db_recipe <- recipe(diabetes ~  glucose + insulin, data = db_train) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors()) %>%
  prep()
```

b. Create your model specification and use `tune()` as a placeholder for the number of neighbors

Click for answer

Answer:

```
knn_spec <- nearest_neighbor(weight_func = "rectangular",
                             engine = "kknn",
                             mode = "classification",
                             neighbors = tune())
```

c. Split the `db_train` data set into `v = 10` folds, stratified by `diabetes`

Click for answer

Answer:

```
db_vfold <- vfold_cv(db_train, v = 10, strata = diabetes)
```

d. Create a grid of `K` values, the number of neighbors and run 10-fold CV on the `k_vals` grid, storing four performance metrics. The vizualization code is provided for your reference.

Click for answer

Answer:

```
k_vals <- tibble(neighbors = seq(from = 1, to = 40, by = 1))
```

```
knn_fit <- workflow() %>%
  add_recipe(db_recipe) %>%
  add_model(knn_spec) %>%
  tune_grid(
    resamples = db_vfold,
    grid = k_vals,
    metrics = metric_set(yardstick::ppv, yardstick::accuracy, sens, spec),
    control = control_resamples(save_pred = TRUE))
```

```
cv_metrics <- collect_metrics(knn_fit)
cv_metrics %>% group_by(.metric) %>% slice_max(mean)
```

```
# A tibble: 8 x 7
# Groups:   .metric [4]
  neighbors .metric  .estimator  mean     n std_err .config
      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1        33 accuracy binary     0.777    10  0.0214 Preproc~
2        34 accuracy binary     0.777    10  0.0214 Preproc~
3        19 ppv      binary     0.810    10  0.0224 Preproc~
4        20 ppv      binary     0.810    10  0.0224 Preproc~
5        33 sens     binary     0.893    10  0.0135 Preproc~
6        34 sens     binary     0.893    10  0.0135 Preproc~
7        19 spec     binary     0.543    10  0.0640 Preproc~
8        20 spec     binary     0.543    10  0.0640 Preproc~
```

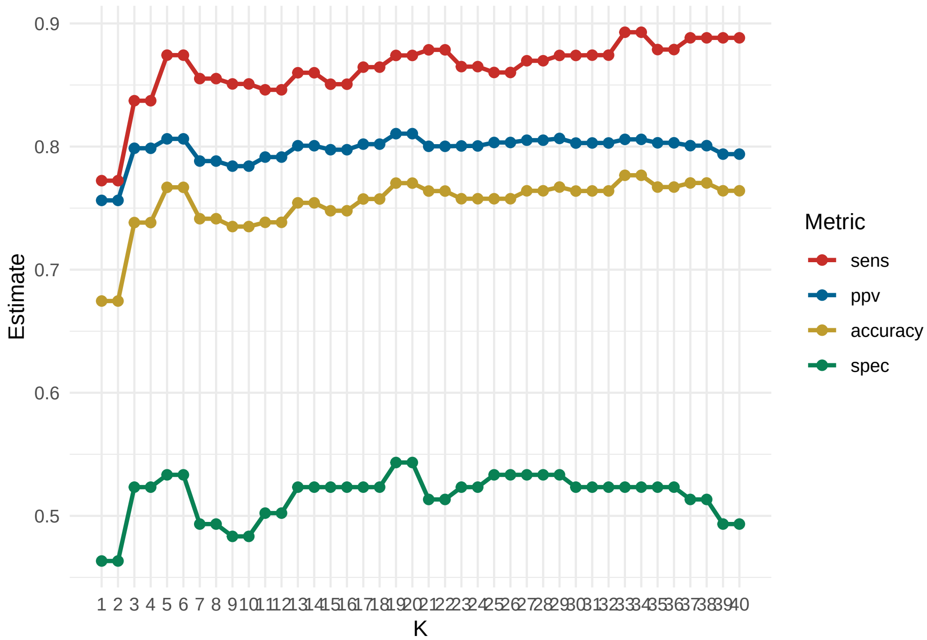
29.1.1 Extra: Plot the metrics

Click for answer

Answer:

```
final.results <- cv_metrics %>%  mutate(.metric = as.factor(.metric)) %>%
  select(neighbors, .metric, mean)

final.results %>%
  ggplot(aes(x = neighbors, y = mean, color = forcats::fct_reorder2(.metric, neighbors, mean))) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  theme_minimal() +
  scale_color_wsj() +
  scale_x_continuous(breaks = k_vals[[1]]) +
  theme(panel.grid.minor.x = element_blank())+
  labs(color='Metric', y = "Estimate", x = "K")
```



29.2 Group Activity 2

a. Let's fit the logistic regression model.

```
set.seed(12345)
db_single <- db %>% select(diabetes, glucose)
db_split <- initial_split(db_single, prop = 0.80)

# Create training data
db_train <- db_split %>% training()

# Create testing data
db_test <- db_split %>% testing()

fitted_logistic_model <- logistic_reg() %>% # Call the model function
        # Set the engine/family of the model
        set_engine("glm") %>%
        # Set the mode
        set_mode("classification") %>%
        # Fit the model
        fit(diabetes~., data = db_train)
```

```
tidy(fitted_logistic_model)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  -5.61     0.678       -8.28 1.20e-16
2 glucose       0.0392   0.00514      7.62 2.55e-14
```

b. We are interested in predicting the diabetes status of patients depending on the amount of glucose. Verify that the glucose value of 143.11 gives the probability of having diabetes as 1/2.

Click for answer

Answer:

$$log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x$$

```
(p <- round(exp(-5.61 + 0.0392* 143.11) / (1 + exp(-5.61 + 0.0392* 143.11)),2))
```

```
[1] 0.5
```

c. What value of glucose is needed to have a probability of diabetes of 0.75?

Click for answer

Answer:

```
p <- 0.75
(x <- (log(p/(1-p)) - (-5.61))/0.0392)
```

```
[1] 171.1381
```

d. Make a classifier that classifies the diabetes status of new patients with a threshold of 0.75, i.e, a new patient is classified as negative if the estimated class probability is less than 0.75. Also, create a confusion matrix of the resulting predictions.

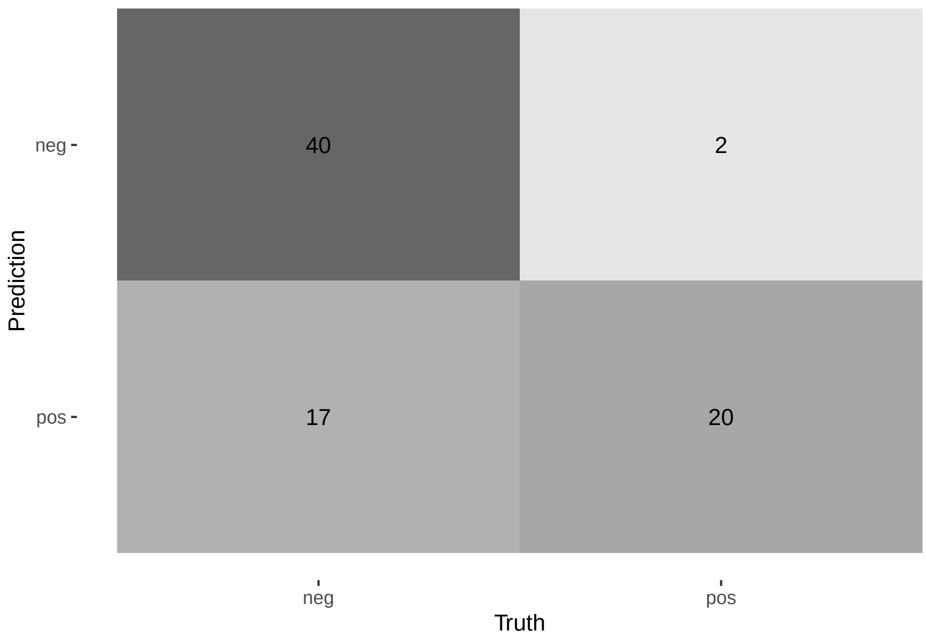
Click for answer

Answer:

```
# Prediction Probabilities
library(probably)
pred_prob <- predict(fitted_logistic_model,  new_data = db_test,   type = "prob")

db_results <- db_test %>% bind_cols(pred_prob) %>%
  mutate(.pred_class = make_two_class_pred(.pred_neg, levels(diabetes), threshold = .75
  select(diabetes, glucose, contains(".pred"))


db_results %>%
  conf_mat(diabetes,.pred_class) %>%
  autoplot(type = "heatmap")
```



Chapter 30

Class Activity 24

```
# load the necessary libraries
library(tidyverse)
library(ggthemes)
library(factoextra)
library(janitor)
library(broom)

select <- dplyr::select
theme_set(theme_stata(base_size = 10))


standardize <- function(x, na.rm = FALSE) {
  (x - mean(x, na.rm = na.rm)) / sd(x, na.rm = na.rm)
}
```

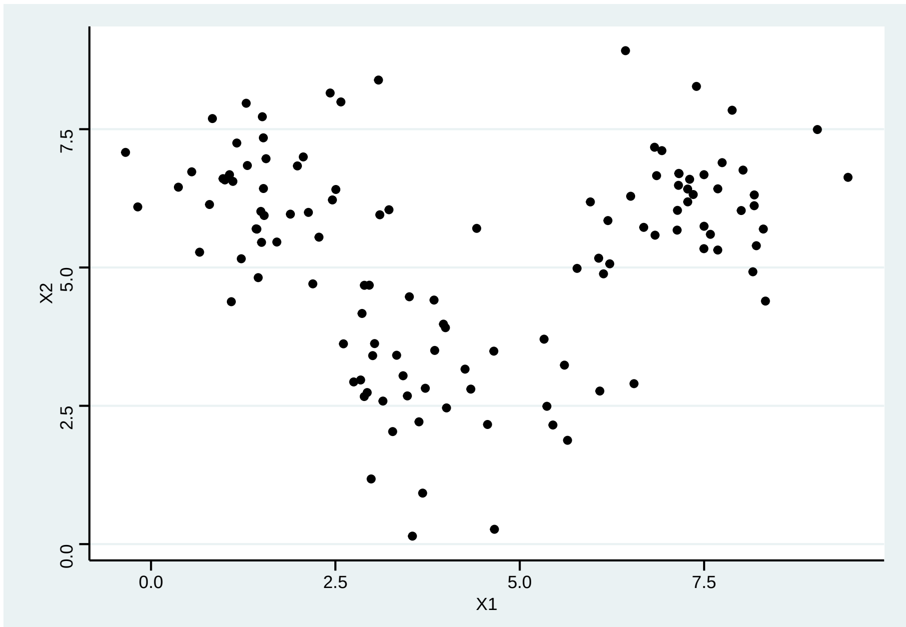
30.1 Group Activity 1

Let's look at the following data tibble that randomly creates some `x-` and `y-` coordinates around the cluster centroids that we just saw in class. Please answer the questions based on this data.

```
set.seed(1234)

my_df <- tibble(
  X1 = rnorm(n = 120, mean = rep(c(2, 4, 7.33), each = 40)),
  X2 = rnorm(n = 120, mean = rep(c(6.33, 3, 6), each = 40))
)
```

```
my_df %>%
  ggplot(aes(X1, X2)) +
  geom_point()
```



a. How many clusters can you identify in the data?

Click for answer

Answer: Answers may vary

b. Fit `kmeans` algorithm to the data picking the number of clusters you previously identified in part `a`.

Click for answer

Answer:

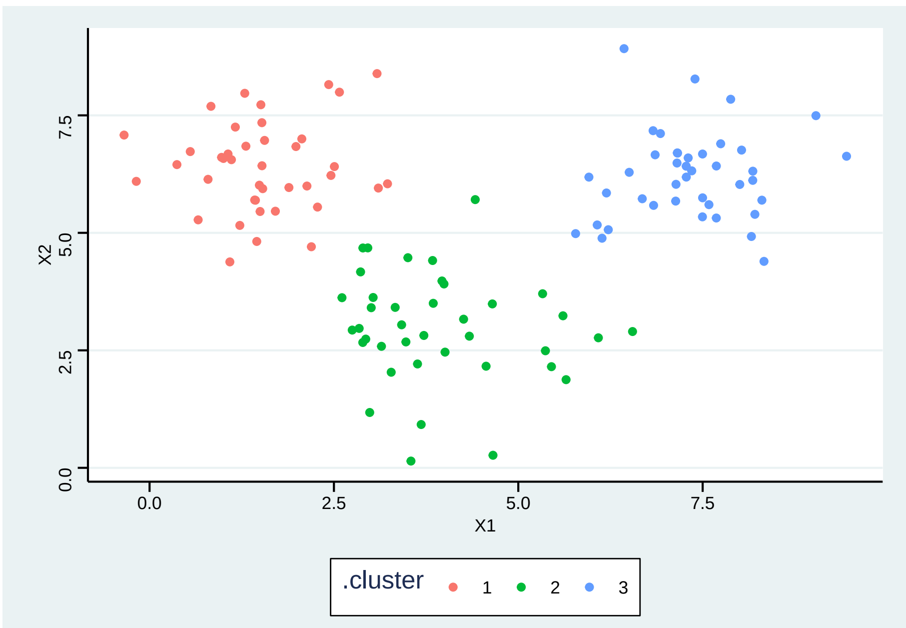
```
set.seed(1234)
res_kmeans <- kmeans(my_df, centers = 3, nstart = 25)
```

c. Add the cluster association to the dataset and make a scatter plot color-coded by the cluster association.

Click for answer

Answer:

```
augment(res_kmeans, data = my_df) %>%
  ggplot(aes(X1, X2, color = .cluster)) +
  geom_point()
```



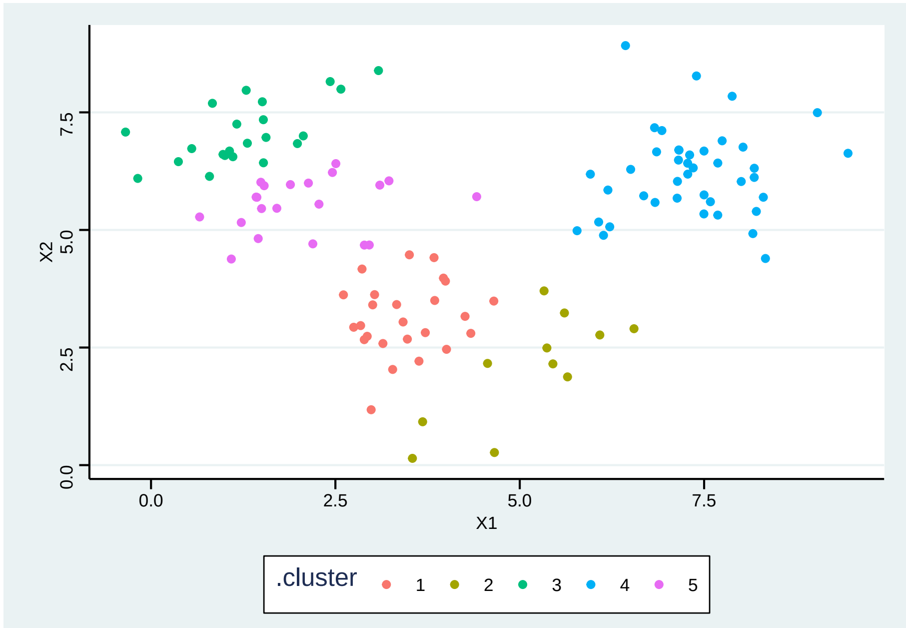
d. Repeat parts `b-c` for identifying more number of clusters than what you picked in part a.

Click for answer

Answer:

```
set.seed(1234)
res_kmeans <- kmeans(my_df, centers = 5, nstart = 25)
```

```
augment(res_kmeans, data = my_df) %>%
  ggplot(aes(X1, X2, color = .cluster)) +
  geom_point()
```



30.2 Group Activity 2

a. Aggregate the total within sum of squares for each k to the data table `multi_kmeans`.

Click for answer

Answer:

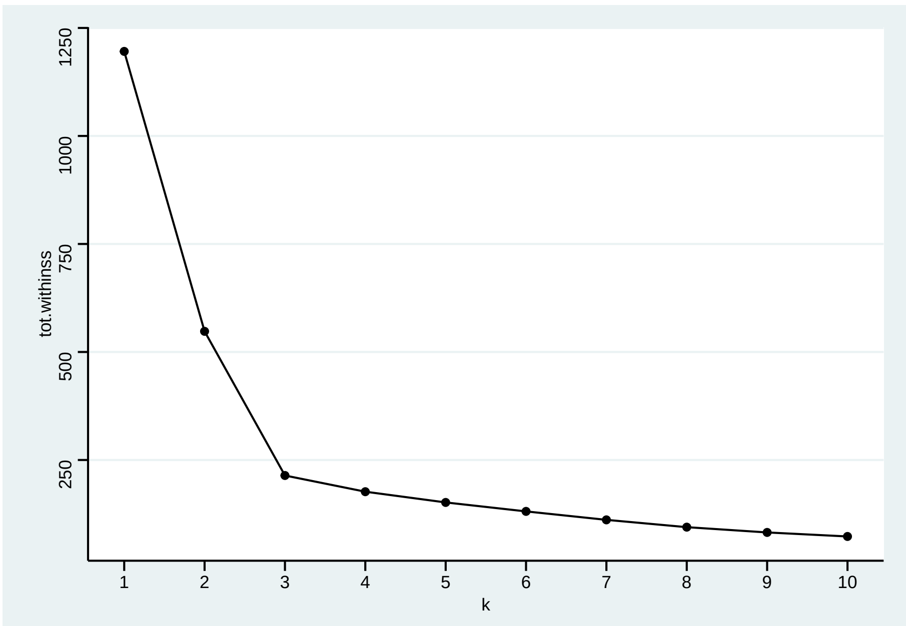
```
multi_kmeans <- tibble(k = 1:10) %>%
  mutate(
    model = purrr::map(k, ~ kmeans(my_df, centers = .x, nstart = 25)),
    tot.withinss = purrr::map_dbl(model, ~ glance(.x)$tot.withinss)
  )
```

b. Make an elbow plot modifying the code below:

Click for answer

Answer:

```
multi_kmeans %>%
  ggplot(aes(k, tot.withinss)) +
  geom_point() +
  geom_line()+
  scale_x_continuous(breaks = 1:15)
```

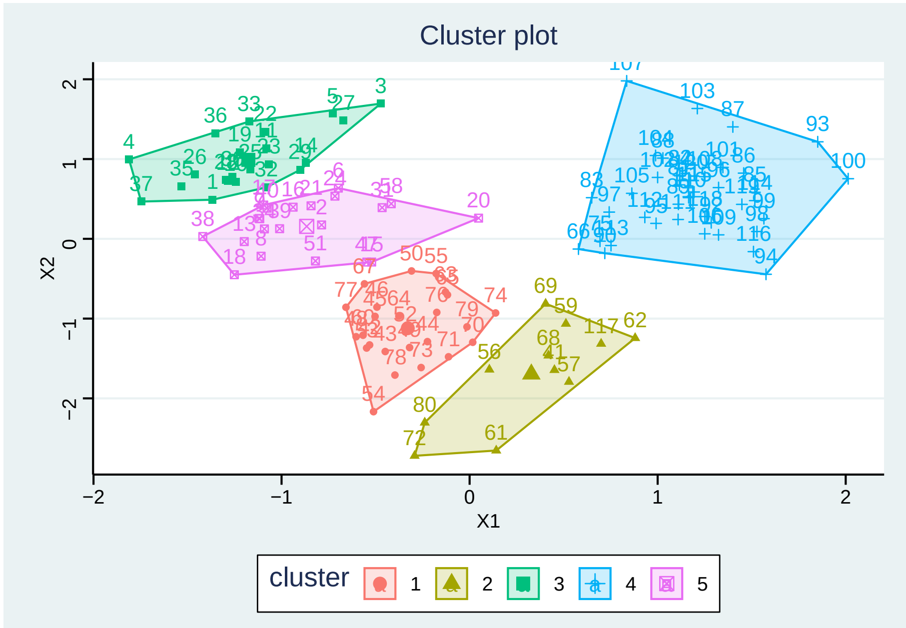


c. After picking an optimal number of cluster, use the in-built function in the `factoextra` package to construct the final cluster plot.

Click for answer

Answer:

```
set.seed(1234)
kmeans.final <- kmeans(my_df, 5, nstart = 25)
fviz_cluster(kmeans.final, data = my_df, ggtheme = theme_stata())
```



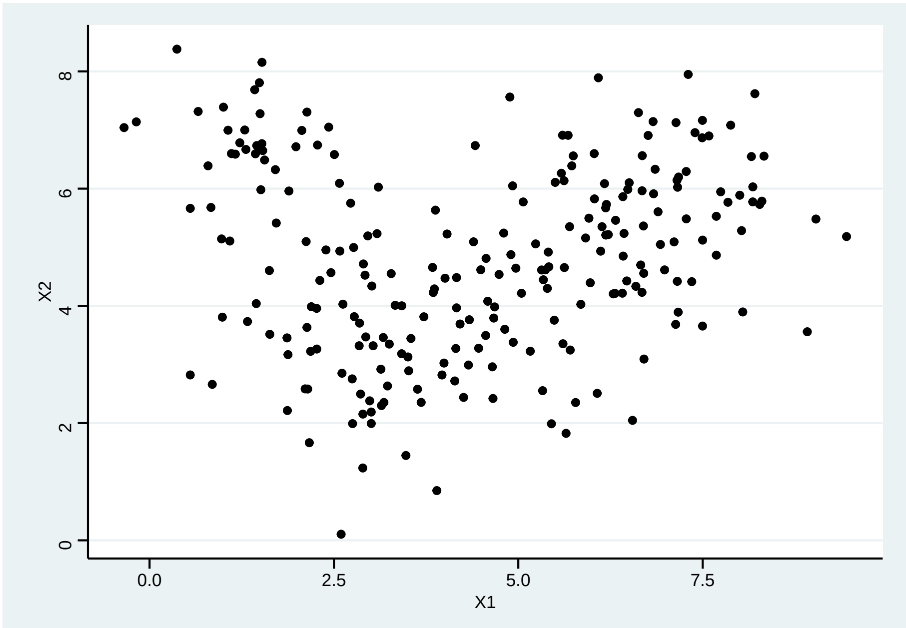
30.3 (Extra) Group Activity 3

Let's look at the following data tibble that randomly creates some x- and y-coordinates around the cluster centroids. Now, there are more clusters and the data points are closer to each other. Please repeat the analysis as seen above to find the optimal number of clusters.

```
set.seed(1234)

my_df <- tibble(
  X1 = rnorm(n = 240, mean = rep(c(2, 4, 7.33, 2.5, 5, 6), each = 40)),
  X2 = rnorm(n = 240, mean = rep(c(6.33, 3, 6, 3.5, 4.5, 5.5), each = 40))
)

my_df %>%
  ggplot(aes(X1, X2)) +
  geom_point()
```



X2
0
2
4
6
8
X1
0.0
2.5
5.0
7.5