Stat 220 Introduction to Data Science

Deepak Bastola

2024-01-28

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.

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:

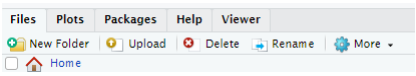
- **stat220** folder
  - **Assignments:** This folder will contain subfolders for each assignment. Each assignment subfolder (e.g. homework1, homework2, …) will be a Github connected RStudio project that you will create **once an assignment is posted.**
  - **Content:** This folder should be used to 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

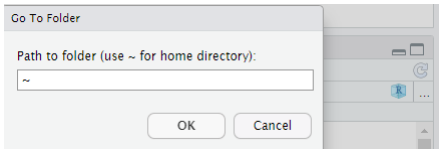
The server (online) version of Rstudio is run from a unix server. You can navigate this file system using unix commands, but I assume that most or all of you will just use Rstudio to access your files on this server.

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

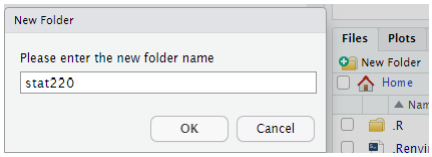


2. Verify that you are in your **HOME** folder (should simply say

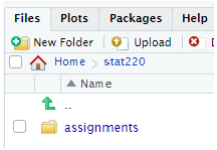
Home right under the New Folder button). To navigate to your Home folder (if somehow you are not in it), click the **...** button (far right side of the **Files** tab) and enter a ~ (tilde) symbol



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

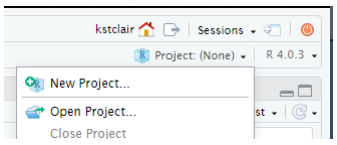


- **4.** Click on this newly created (empty) **stat220** folder. Within the folder create another **New Folder** and name it **assignments**.

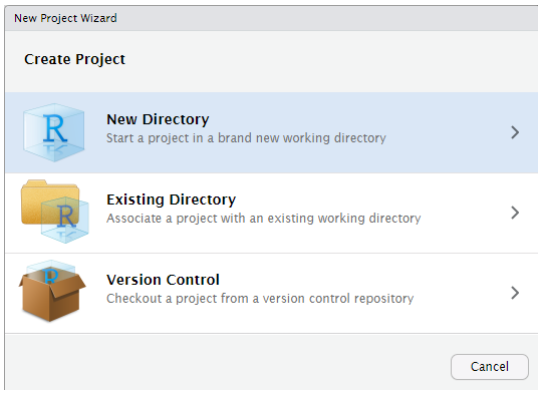


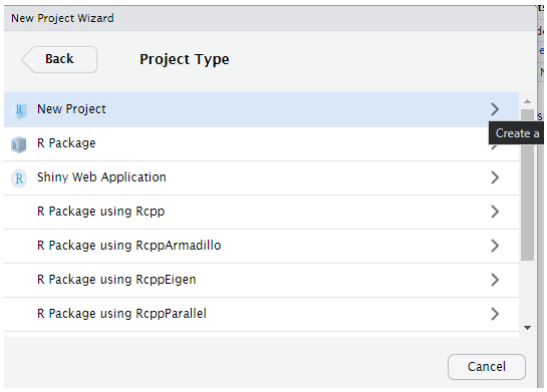
- **5.** Within the **stat220** folder, create an **RStudio project** called **content** with the following steps:

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

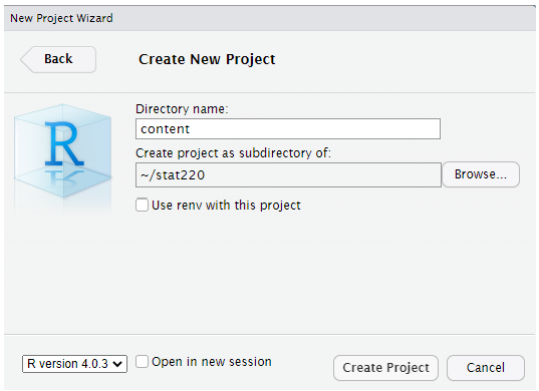


  - **b.** Select **New Directory** and then **New Project**

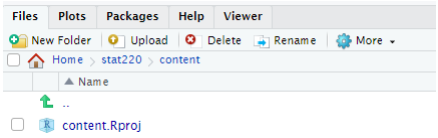




- **c.** Enter **content** as the **Directory name** and use the **Browse** button to find your **stat220** folder. Then click **Create Project**.



- **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).



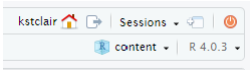
Warning: Do not create an RStudio project in the main stat220 folder because it is not good practice to have RStudio projects in subfolders of another project (e.g. a project within a project is not recommended).

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.

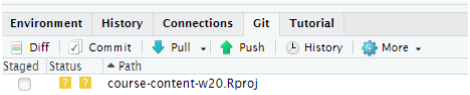
Github is a cloud-based service for hosting git projects. It allows multiple users to share and contribute to projects and it is how you will be submitting homework assignments and projects for this class. More information about Git and Github can also be found in Getting setup with Git and GitHub and 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 DataScienceWinter24. 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:

```
#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.

```
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

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

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

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

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

```
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

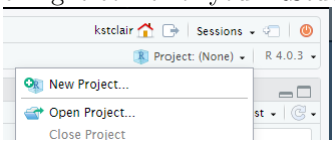
If you followed the suggestions in the File organization in RStudio page, then you should already have an assignments folder on your computer or maize account.

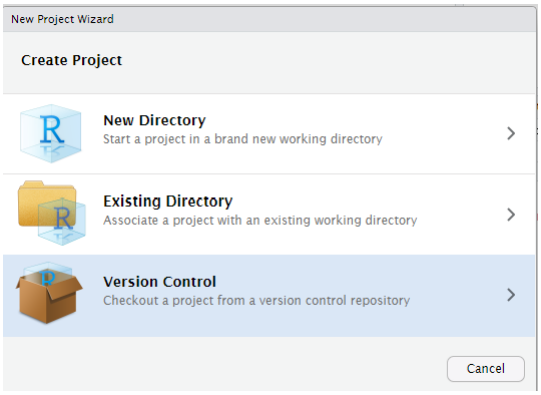
Each new assignment/project will be posted as a repository on GitHub and

added directly to your account (within the Stat220 organization). This repository will contain assignment details (README, .Rmd).

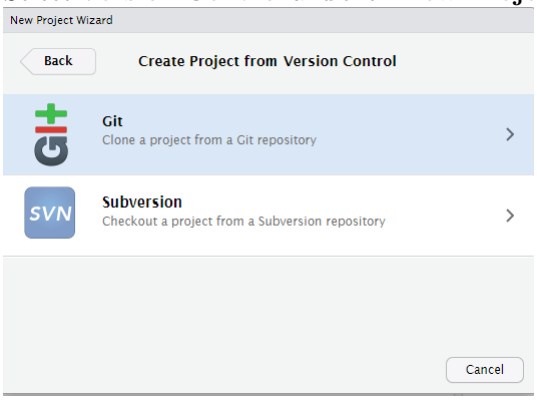
4.3.1 Creating an individual assignment repo and project

1. Go to our course GitHub organization page (DataScienceWinter24) 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...**.



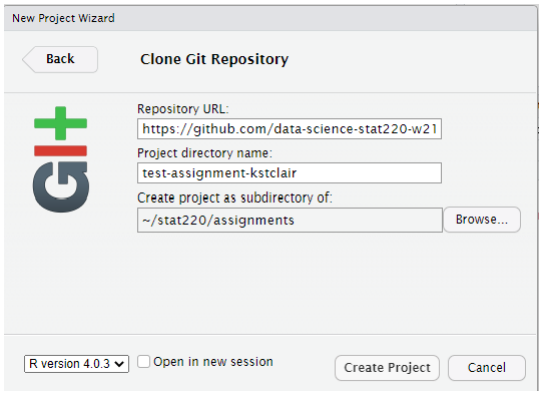


   - Select **Version Control** and then **New Project**



   - 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**



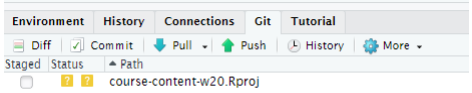
4.3.2 Working on your assignment

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(DataScienceWinter24) 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:

```r
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
)
```

(Credit: Yihui Xie)

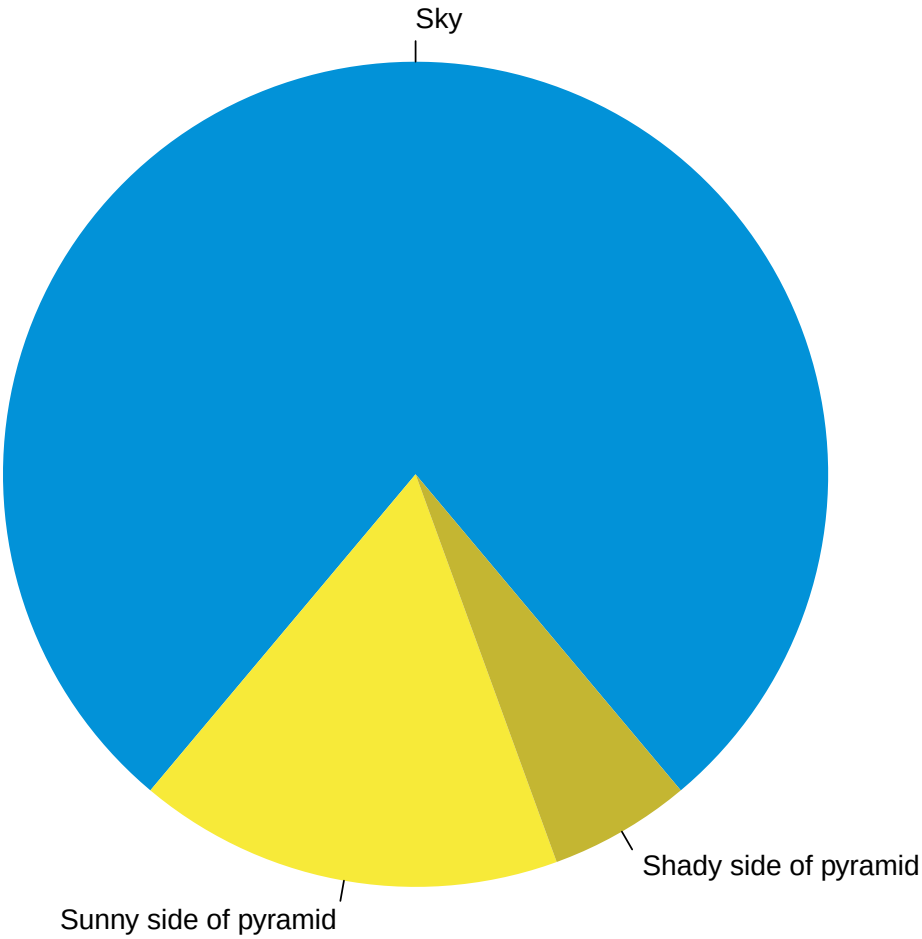


Figure 5.1: A fancy pie chart.

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

Chapter 6

Github Tutorial

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

```
# 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.

Class Activities

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
# i 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.

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?

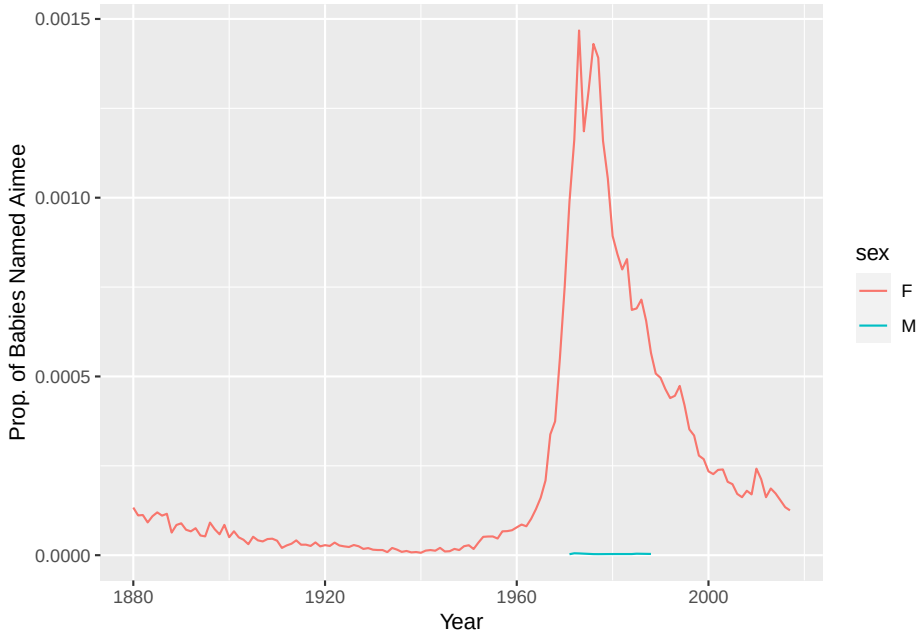


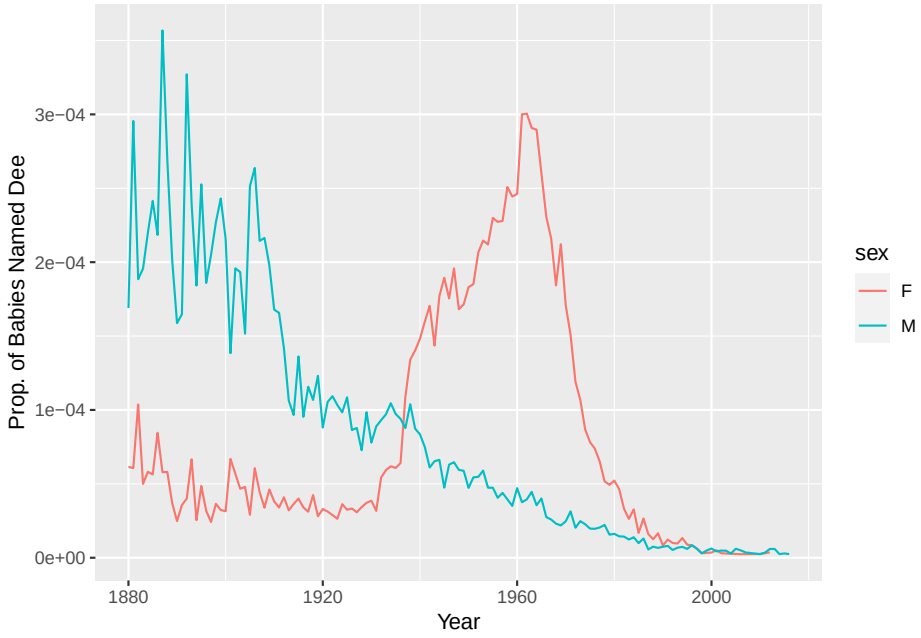
Figure 7.1: A trend chart

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 (Optional). 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?

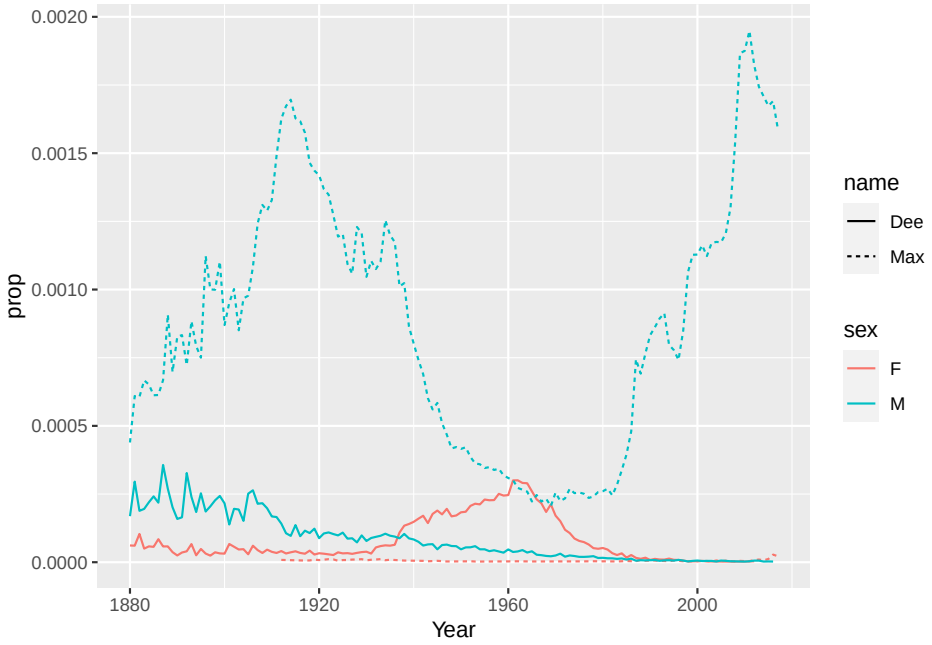
Answer Answers will vary

```
# 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.

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

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

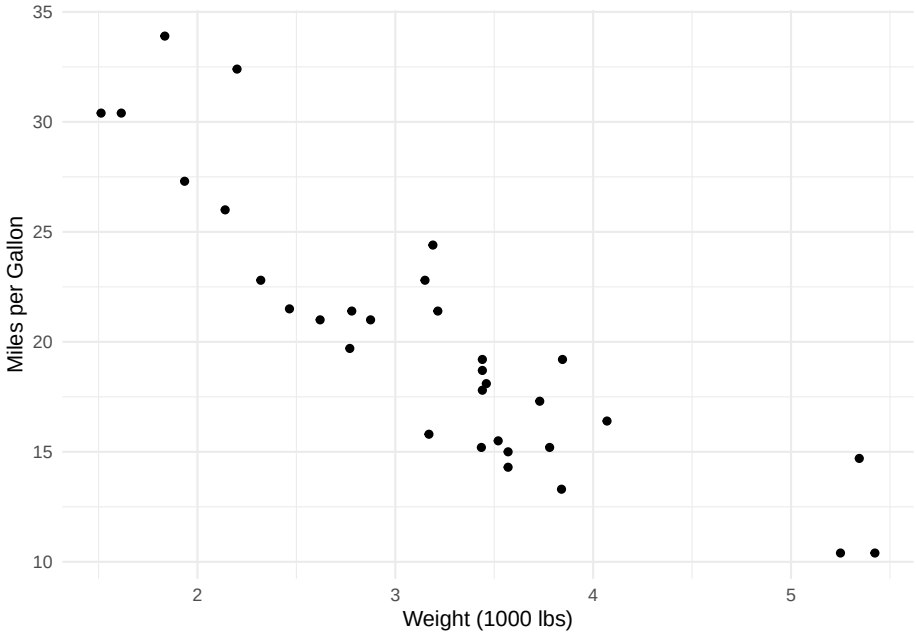
```
library(tidyverse)
```

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).

```
# 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

```
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

```
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

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

```
[1] 7
```

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

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:

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

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

Chapter 10

Class Activity 4

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

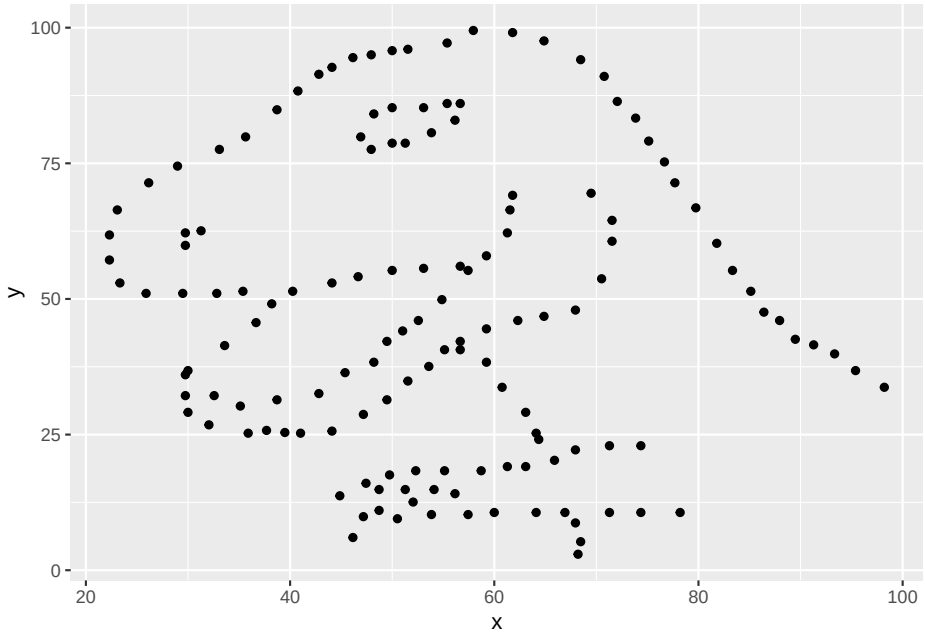
10.1 Your turn 1

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.

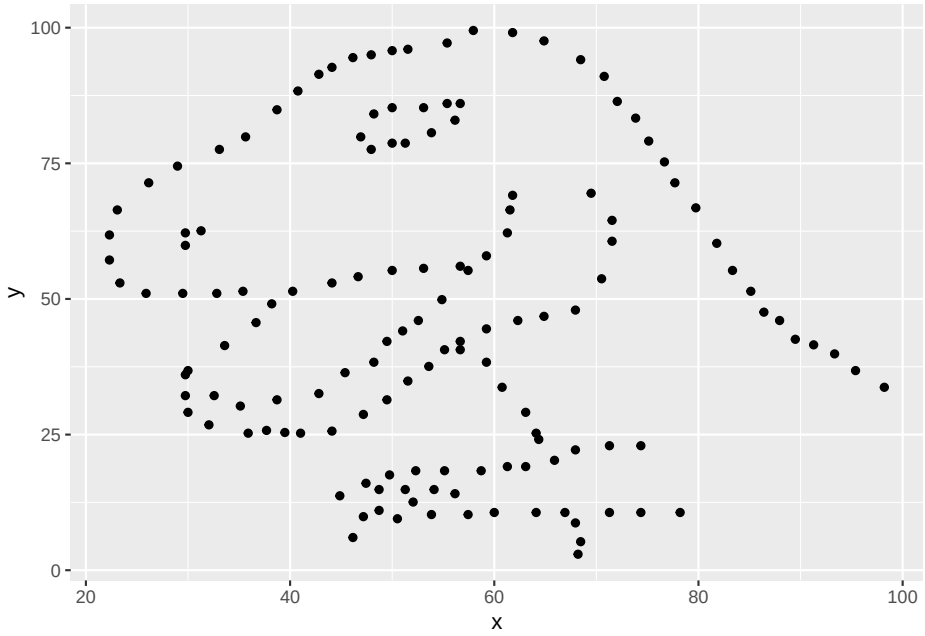
```
# Add a layer and see what happens
ggplot(data = dino_data , x = x , y = y)
```

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:

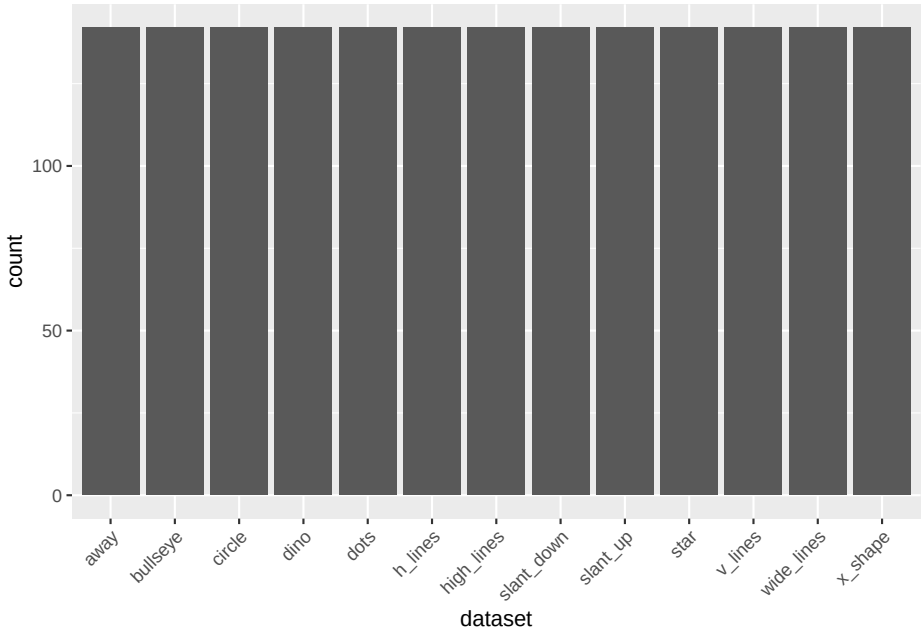
```
dataset_counts <- datasaurus_dozen %>%
  group_by(dataset) %>%
  summarise(count = n())
```

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))
```



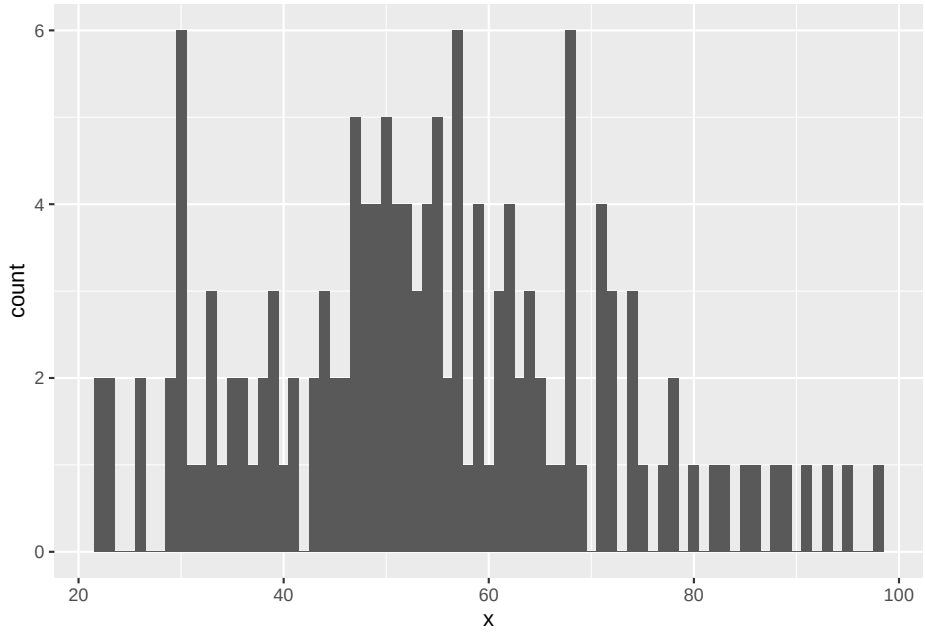
10.1.3 Histogram

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)
```

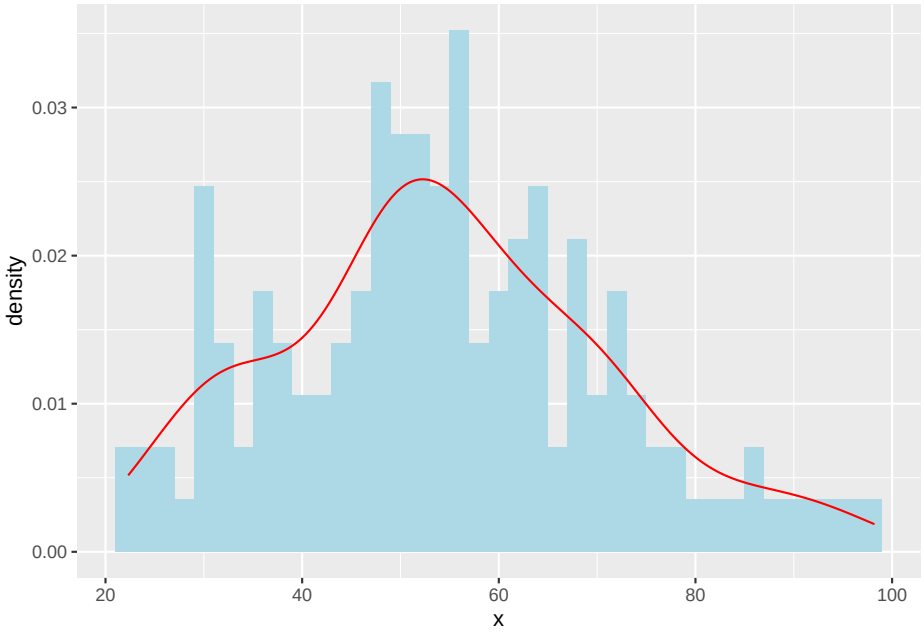


b. Overlay a density curve on the histogram.

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")
```



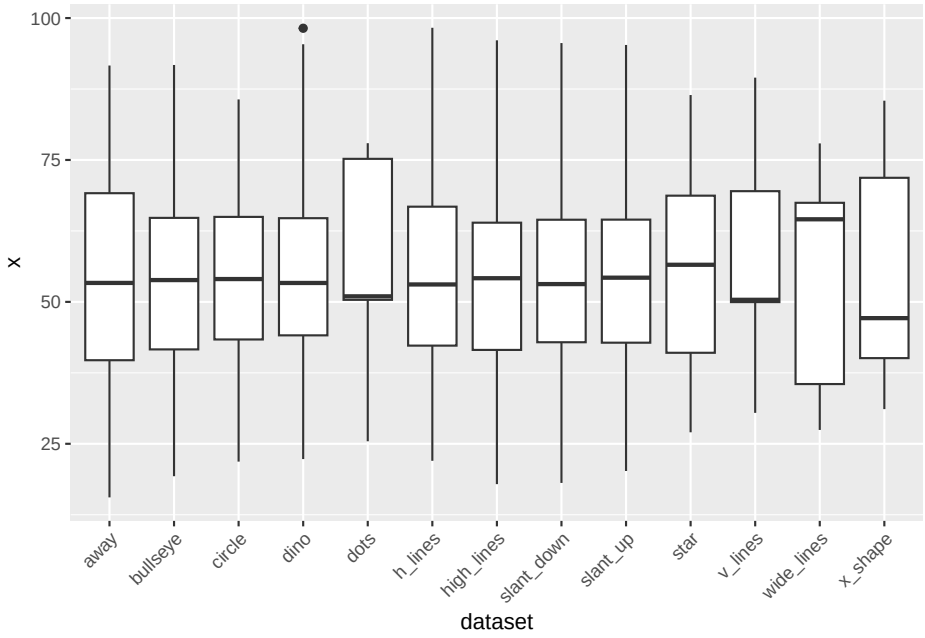
10.1.4 Boxplot

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))
```



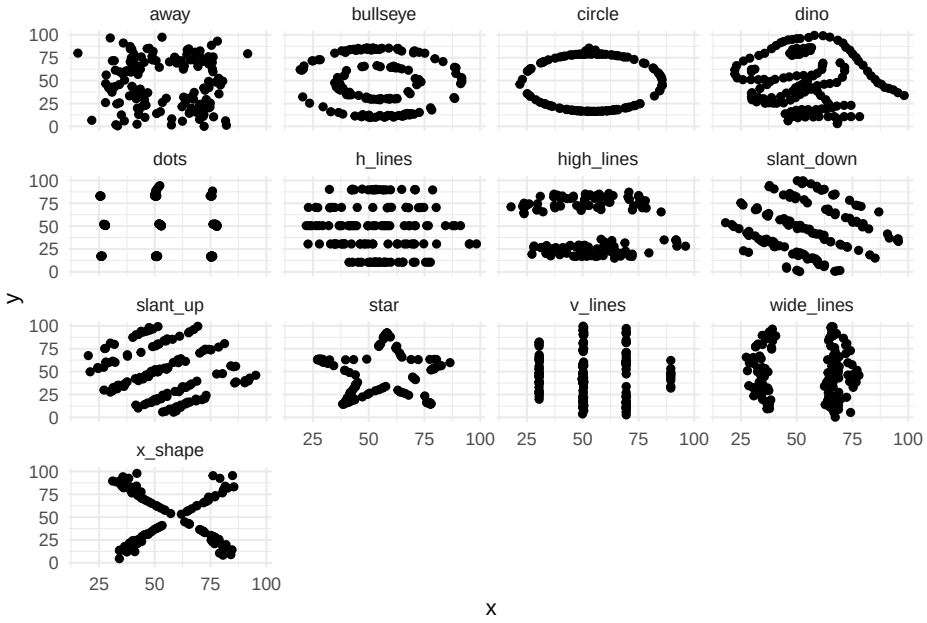
10.1.5 Faceting

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:

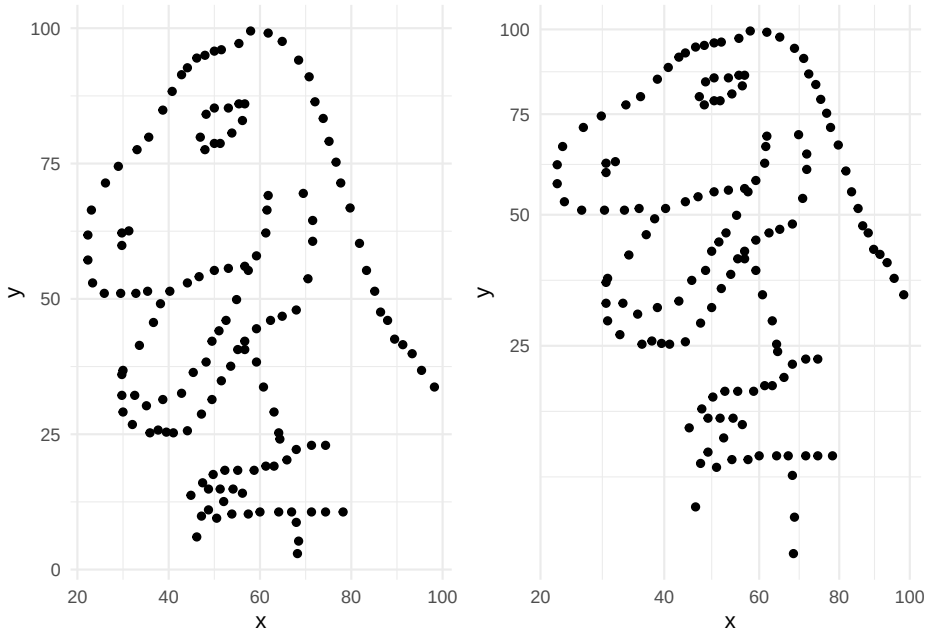
```
ggplot(data = dino_data, aes(x = x, y = y)) +
  geom_point() +
  scale_x_sqrt() +
  scale_y_sqrt() +
  theme_minimal() -> p2
```

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

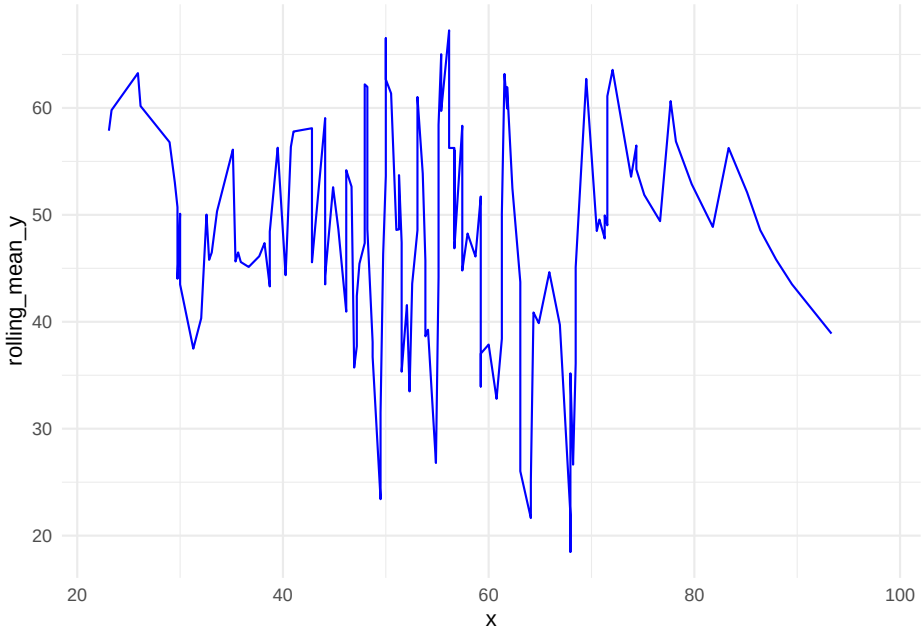
a. Create a line plot of the `x` variable over the `y` variable for the `dino` dataset. To make it more interesting, let's first calculate the rolling mean of the `y` variable.

Click for answer

Answer:

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

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


rolling_mean_y
60
50
40
30
20
20
40
60
80
100
x

Chapter 11

Class Activity 5

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

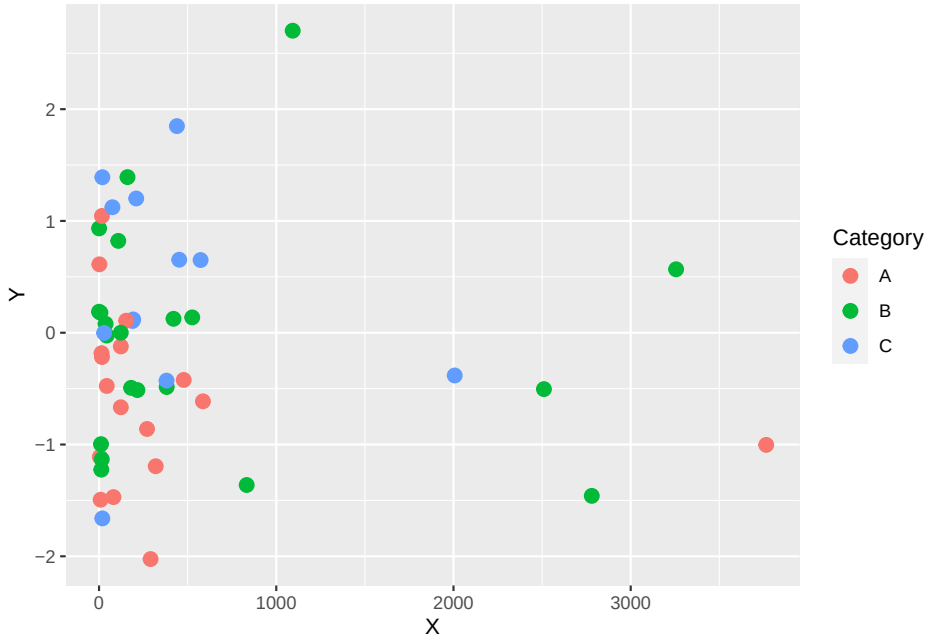
11.1 Problem 1: Changing color and shape scales

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
```

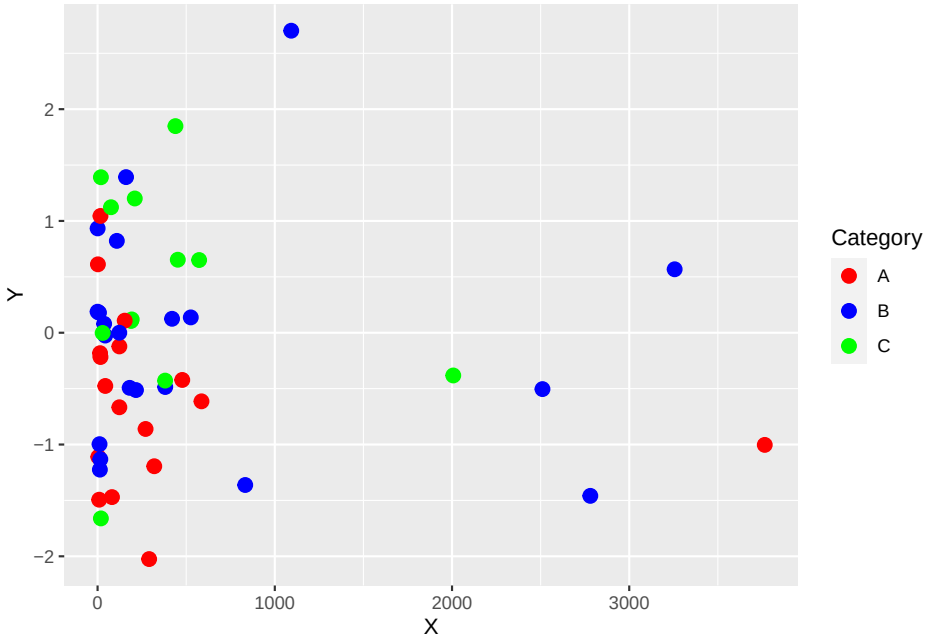


a. Modify the scatter plot to use custom colors for each category using `scale_color_manual()`. What is the effect of changing the colors on the plot's readability?

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, group = Category)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("A" = "red", "B" = "blue", "C" = "green"))
p
```



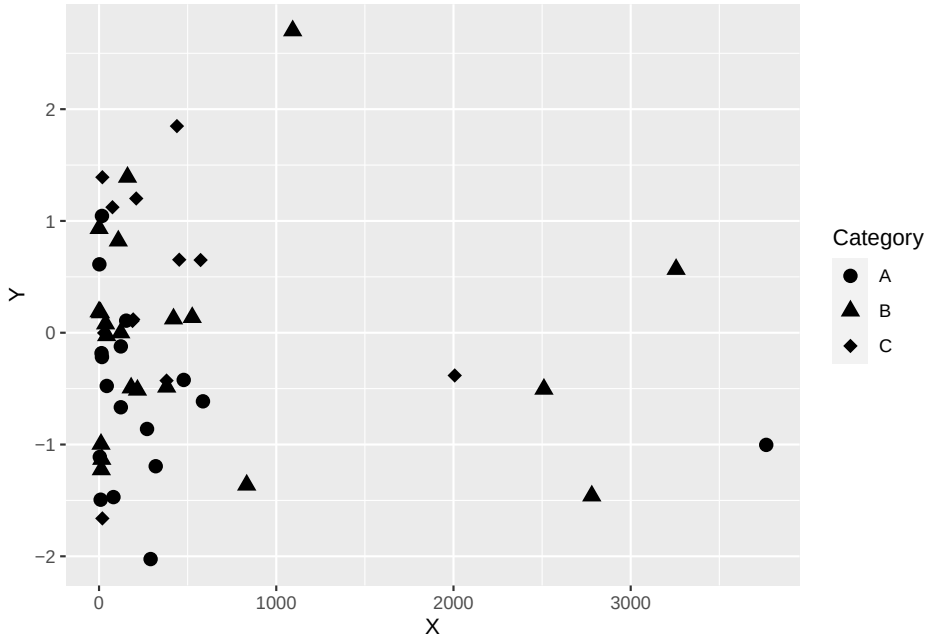
b. Modify the scatter plot to use custom shapes for each category using `scale_shape_manual()`. What is the effect of changing the shapes on the plot's readability?

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,  group = Category)) +
  geom_point(size = 3) +
  scale_shape_manual(values = c("A" = 16, "B" = 17, "C" = 18))

p
```



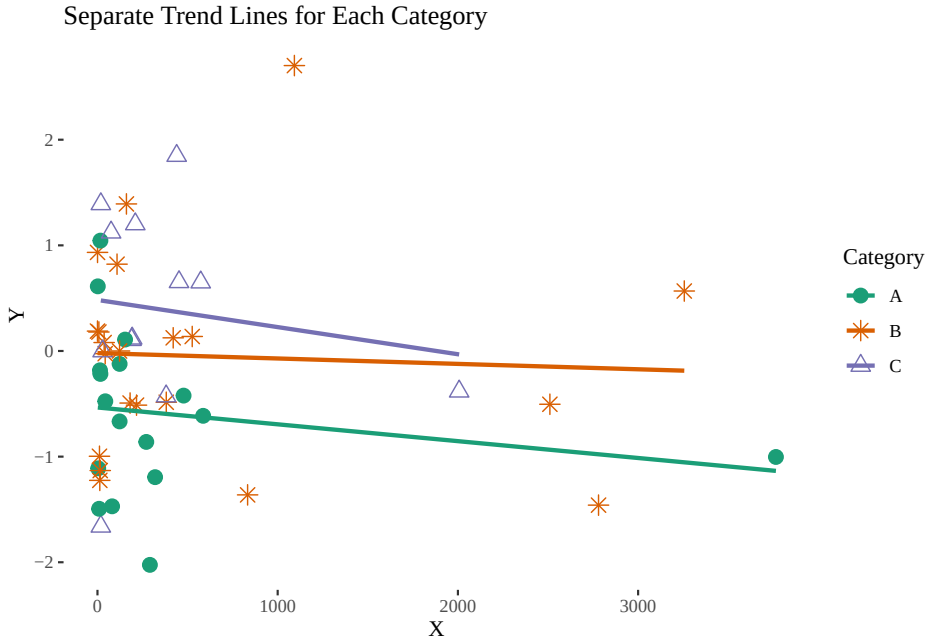
c. Try modifying the plot by combining color, shape, and theme customizations. Additionally, try using `geom_smooth()` to add trend lines for each category. Pay attention to how each element affects the overall readability and interpretability of the plot.

Click for answer

Answer:

```
# Base plot
p <- ggplot(data, aes(x = X, y = Y)) +
  geom_point(aes(color = Category, shape = Category), size = 3) + # Assign color and s
  geom_smooth(aes(group = Category, color = Category), method = "lm", se = FALSE) + #
  scale_shape_manual(values = c("A" = 19, "B" = 8, "C" = 24)) + # Customize shapes for
  scale_color_brewer(palette = "Dark2") + # Customize color palette
  ggthemes::theme_tufte() +
  labs(title = "Separate Trend Lines for Each Category")

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:

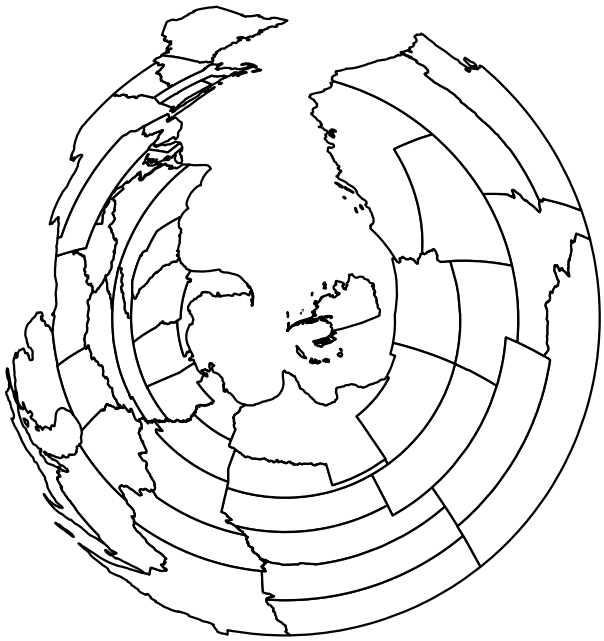
```
us_flipped <- us + coord_flip()
us_flipped
```

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.csv")
ACS <- dplyr::filter(ACS, !(region  %in% c("Alaska", "Hawaii"))) # only 48+D.C.
ACS$region <- tolower(ACS$region)  # lower case (match states regions)
```

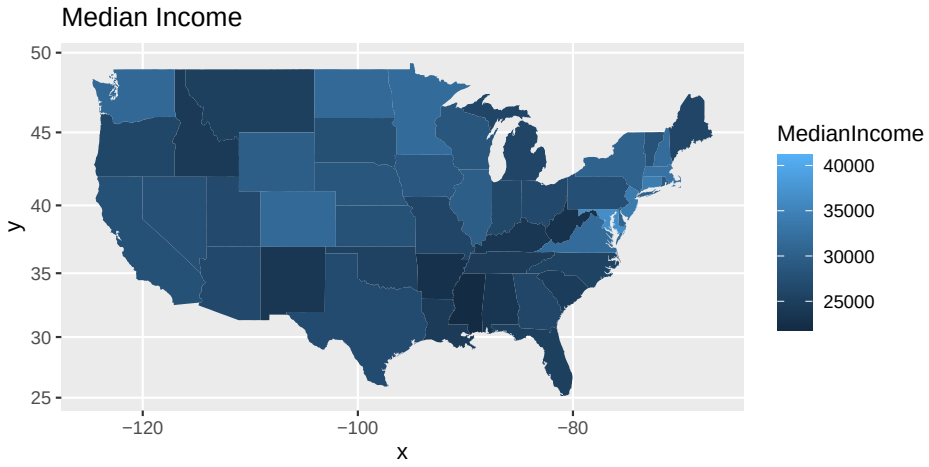
11.3.1 (a) Mapping median income

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")
```



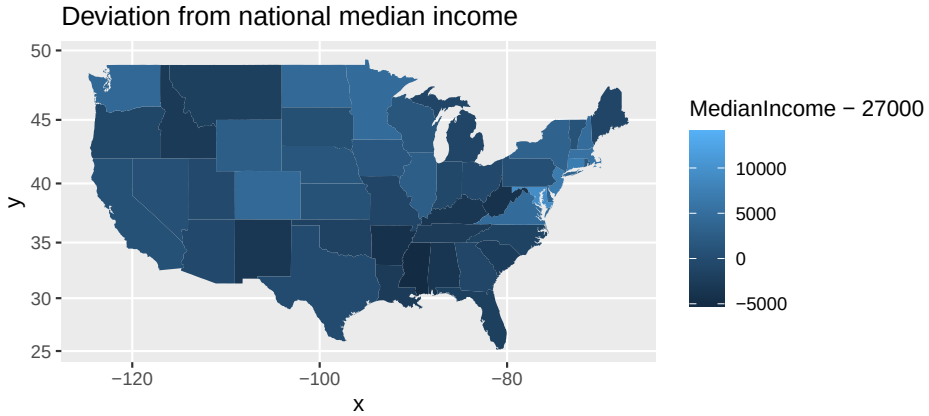
11.3.2 (b) Mapping deviations from national median income

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
```



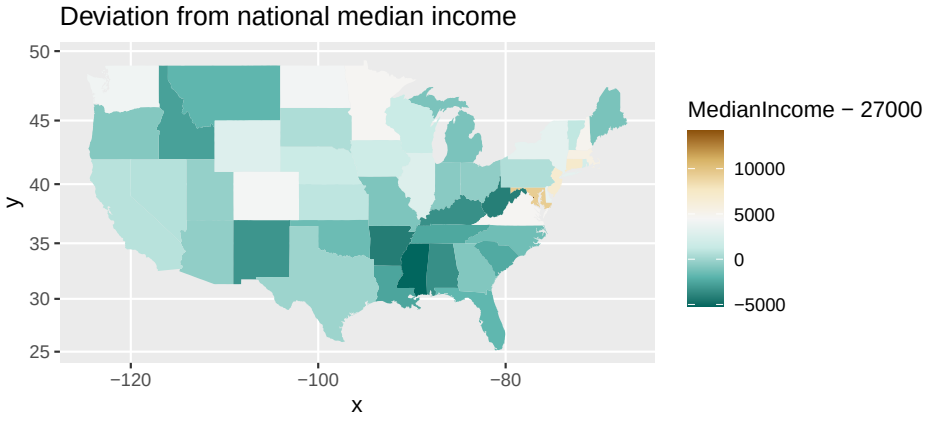
11.3.3 (c) Changing numerically scaled color

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 income") +
  scale_fill_distiller(type = "div")
```



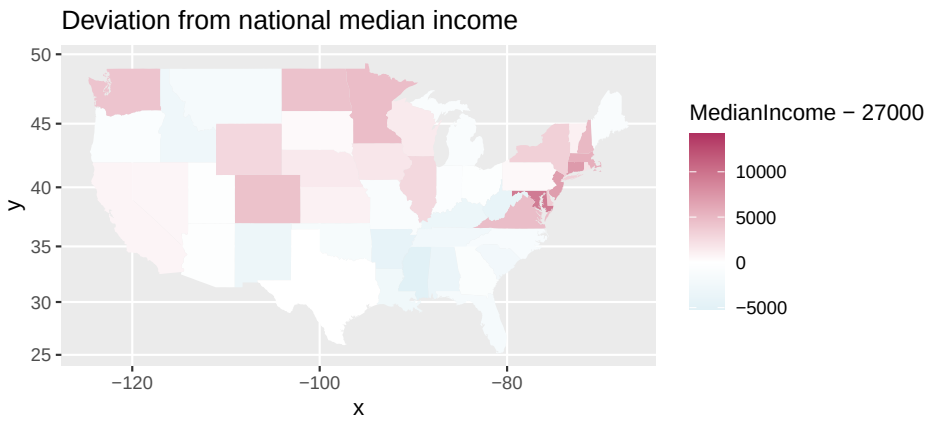
11.3.4 (d) Fixing a midpoint on a diverging scale

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
  )
```



11.3.5 (e) Polygon map

```
# Merge income data with geographic information
income_data <- left_join(states, ACS, by = c("region" = "region"))
```

Next, we will use this merged data to create a polygon map that focuses on the boundaries and shapes of each state, colored by median income.

11.3.5.1 Understanding Mercator Projection

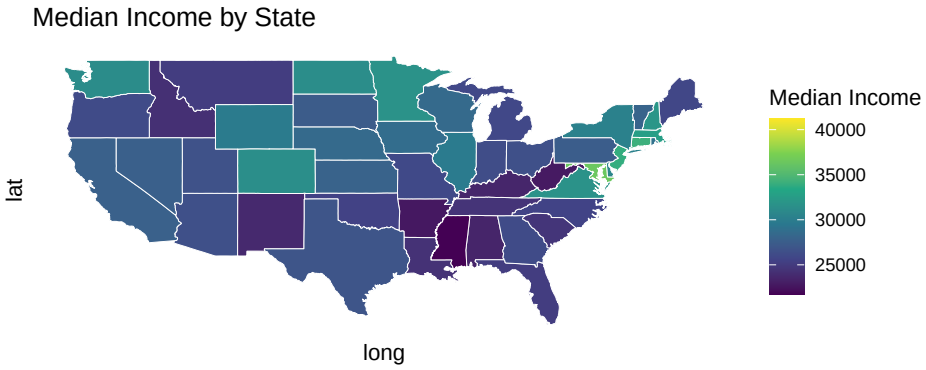
The Mercator projection is a cylindrical map projection that was widely used for navigation charts because it represents lines of constant course, known as rhumb lines, as straight segments. However, this projection distorts the size of objects as the latitude increases from the Equator to the poles. For example, Greenland appears larger than Africa on a Mercator projection map, while in reality, Africa is about 14 times larger.

For this task, you will create a polygon map to visualize the `MedianIncome` across different states using the Mercator projection. Pay attention to the shapes and sizes of states as depicted on the map.

Click for answer

```
library(sf)

ggplot(data = income_data) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = MedianIncome), color = "wh
  coord_sf(crs = st_crs("+proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m
  labs(fill = "Median Income", title = "Median Income by State") +
  theme_minimal() +
  scale_fill_viridis_c()
```



11.3.6 (f) Visualizing Relative Income Deviation with Robinson Projection

The Robinson projection is a map projection of a world map which shows the entire globe as if it were flat. It was specifically created in an attempt to find a good compromise to the problem of readily showing the whole globe as a flat image. The projection is neither equal-area nor conformal, abandoning both for a compromise. The Robinson projection is widely used for thematic and educational maps due to its visually pleasing representation of the Earth.

For this task, you will visualize the relative income deviation across states using the Robinson projection. Consider how the projection's compromise between size and shape affects the presentation of income data.

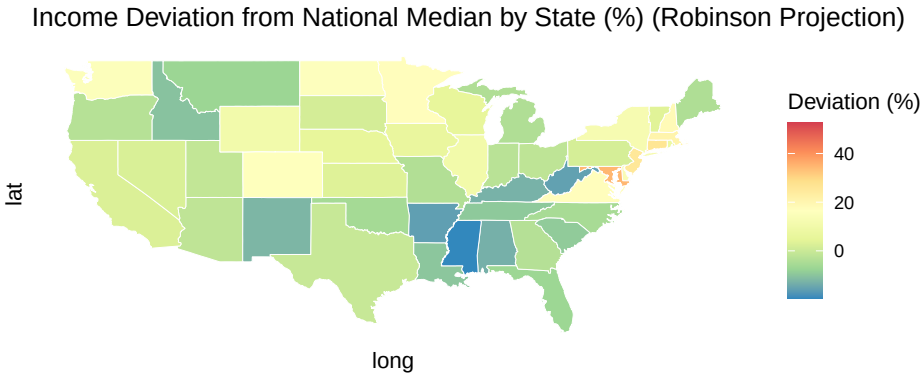
Click for answer

```
# Calculate income deviation as a percentage
national_median <- 27000

# Merge the updated income data with geographic information
ACS$IncomeDeviationPercent <- ((ACS$MedianIncome - national_median) / national_median) * 100
income_data <- left_join(states, ACS, by = c("region" = "region"))

# Define the CRS for Robinson projection
robinson_crs <- st_crs("+proj=robin +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs")

# Plot the income deviation using Robinson projection with geom_polygon
ggplot(data = income_data) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = IncomeDeviationPercent), color = "whi
  coord_sf(crs = robinson_crs, datum = NA) +
  labs(fill = "Income Deviation (%)", title = "Income Deviation from National Median by State (%)
  theme_minimal() +
  scale_fill_distiller(palette = "Spectral", name = "Deviation (%)")
```


Income Deviation from National Median by State (%) (Robinson Projection)
Deviation (%)
40
20
0
lat
long

Chapter 12

Class Activity 6

```
# 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:

```
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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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
# i 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", "Guitar",
                                 "Guitar", "Drums", "Guitar", "Vocals", "Vocals"))


bands <- tibble(first = c("John", "John Paul", "Jimmy", "Robert", "George", "John",
                          "Paul", "Ringo", "Jimmy", "Mick", "Keith", "Charlie", "Ronnie"),
                last = c("Bonham", "Jones", "Page", "Plant", "Harrison", "Lennon",
                         "McCartney", "Starr", "Buffett", "Jagger", "Richards", "Watts", "Wood"),
                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 Stones",
                         "The Rolling Stones", "The Rolling Stones"))

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


songs <- tibble(song = c("Come Together", "Dream On", "Hello, Goodbye", "It's Not Unusual"),
                album  = c("Abbey Road", "Aerosmith", "Magical Mystery Tour", "Along Came Jones")
                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>
```

```
# i 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

```
# 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
```

```
n_songs_with_labels
```

```
[1] 3
```

Chapter 14

Class Activity 8

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

14.1 Your turn 1

In the provided R code, we start with two datasets, `DBP_wide` and `BP_wide`, representing blood pressure measurements in a wide format. We then demonstrate how to transform `BP_wide` into a long format using `pivot_longer()`.

```
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:

```
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:

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

```
[1] 28.7081
```

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:

```
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

```
# 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.

```
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            ""
# i 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            ""
# i 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
# i 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
# i 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
# i 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
# i 5 variables: row <int>, col <int>, expected <chr>,
#   actual <chr>, file <chr>
```

```
# 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     2 9th   South America    51 9th
 2     3 9th   Middle East      78 9th
 3     4 9th   Middle East      76 9th
 4     5 9th   Africa           68 9th
 5     6 9th   Africa           92 9th
 6     8 9th   Europe           72 9th
 7    11 9th   Africa           93 9th
 8    13 9th   Middle East      64 9th
 9    16 9th   Europe           85 9th
10    19 9th   Africa           91 9th
11     1 10th  South America    97 10th
12     7 10th  North America    53 10th
13     9 10th  Africa           73 10th
14    17 10th  Asia             81 10th
15    18 10th  Africa           51 10th
16    21 10th  Europe           59 10th
17    23 10th  Africa           72 10th
18    24 10th  South America    90 10th
19    10 11th  Middle East      65 11th
20    12 11th  Middle East      99 11th
21    14 11th  Middle East      53 11th
22    15 11th  Africa           70 11th
23    20 11th  South America    87 11th
24    22 11th  North America    84 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] "9th"  "10th" "11th"
```

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 9th             10
2 Others          14
```

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~
# i 21,473 more rows
# i 1 more variable: tvhours <int>
```

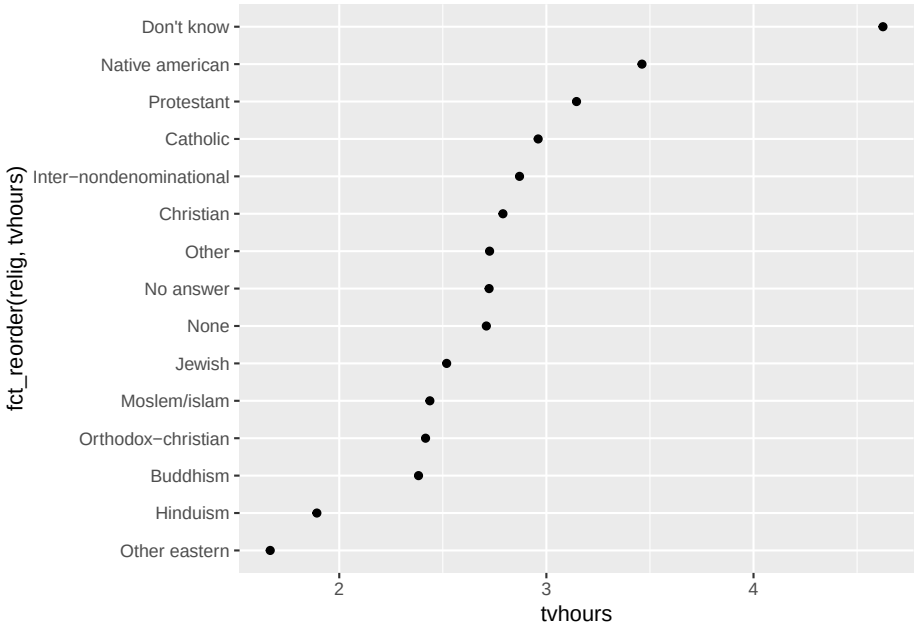
Use `gss_cat` to answer the following questions.

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()
```

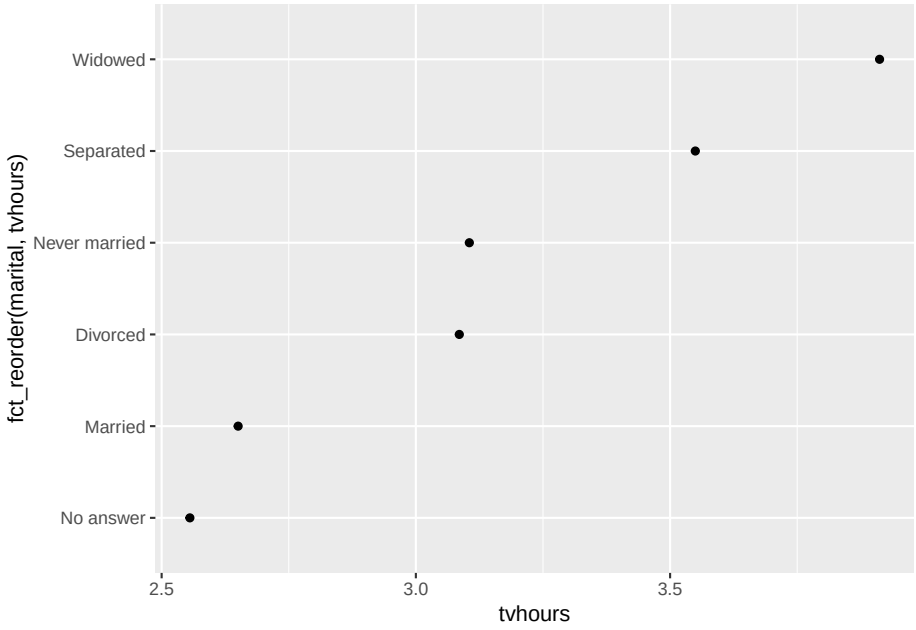


16.2.2 b. Do married people watch more or less TV than single people?

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?

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

c. What does `str_flatten` do?

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

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

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

g. What happens with negative positions?

i. With a vector of positions?

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

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

```
# 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.

18.1 Group Activity 1

```
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 = ". ")
```

18.1.1 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
```

18.1.2 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
```

18.1.3 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 old
```

18.1.4 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 old
```

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
```

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.

18.1.5 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".

18.1.6 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".

18.2 Group Activity 2

18.2.1 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"
```

18.2.2 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
```

18.2.3 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"
```

18.2.4 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
```

18.2.5 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>
```

18.3 Group Activity 3

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

18.3.1 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".

18.3.2 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".

18.3.3 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 @.

18.3.4 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.

18.3.5 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.