LBOMETR Course Book

Jem Marie M. Nario

2025-01-27

Contents

Chapter 1

Introduction

Welcome to the **LBOMETR Course Book**! This book is designed to guide students through the course by providing all necessary resources, materials, and instructions.

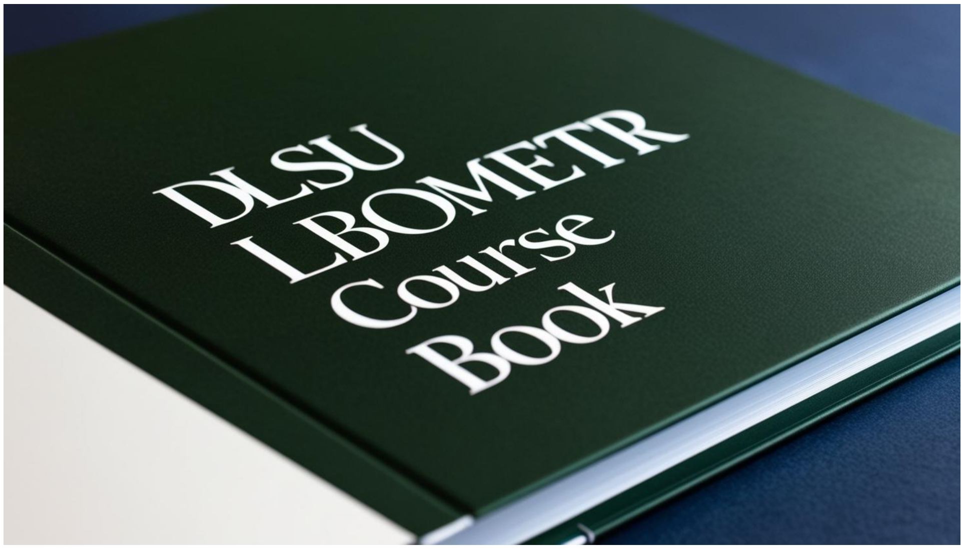


Figure 1.1: LBOMETR

This course book is intended to ensure that DLSU Carlos L. Tiu-School of Economics students will be able to learn more about Econometrics using R. You will find sections on the syllabus, course assessments, and group projects, as well as guidance for navigating the course effectively.

1.1 About Me

My name is **Jem Marie M. Nario**, and I am your lecturer for this course. I am excited to guide you through this journey of learning and discovery since I am also on a journey of learning and discovery while teaching part-time.

This book is a trial version which will be updated along the course as it also serves as a practice for me.

- **Email**: jem.nario@dlsu.edu.ph

- **LinkedIn**: linkedin.com/in/jmnario/

Feel free to reach out with any questions or concerns throughout the course.

Chapter 2

Syllabus

You can download the course syllabus using the link below:

Download Syllabus (Word Document)

2.1 Course Description

This course introduces Economics majors to more advanced commands and techniques used in the econometric software package **R**, which is commonly used in empirical research.

2.2 Learning Outcomes

2.2.1 Knowledge

- To be able to distinguish a theoretical economic model from a statistical econometric model.

- To be able to use the R software package in estimating advanced econometric models.
- To learn advanced econometric models so that students can learn new methods of research.

2.2.2 Skills

- Apply numerical and statistical techniques in economic analysis.
- Use statistical concepts as a language in economic discourse.
- Confidently write script files for economic analysis.

2.2.3 Behavior/Attitude

- To imbibe in the student the need for transparency and academic integrity when handling data analysis.
- To allow the student to learn to construct more complex programs from basic commands learned in class.

2.3 Grading

2.3.1 Grade Components

Component	Weight (%)
Attendance	5%
Group Participation	10%
Data Story Presentation	35%
Data Story Archive	50%

Component	Weight (%)
Total	100%

2.3.2 Grade Scale

Percentage Range	Grade
96 - 100	4.0
90 - 95.99	3.5
84 - 89.99	3.0
78 - 83.99	2.5
72 - 77.99	2.0
66 - 71.99	1.5
60 - 65.99	1.0

Chapter 3

Course Assessments

3.1 Data Story Archive

The **Data Story Archive** is the culmination of your group's work throughout the course. It includes your group's data story report, R script, practical assignments, and a group reflection, all compiled into a single professionally formatted PDF file.

3.1.1 Requirements

Your submission should follow this structure:

1. **Cover Page**:
   - Include the title of the Data Story, group members, and submission date.
2. **Table of Contents**:

- Provide a clear list of sections with page numbers.

3. **Data Story Report**:

   - The complete report should include:
     - **Introduction**: Problem statement and research question.
     - **Methods**: Data sources, methodology, and analysis techniques.
     - **Results**: Key findings supported by R-generated visuals.
     - **Discussion**: Implications of the findings and any limitations.
     - **Conclusion**: Summary and recommendations.
     - **Appendix**: Supporting tables, additional plots, or materials.

4. **R Script**:

   - Render your R script as an **HTML** using Quarto Markdown.
   - Ensure the script is well-structured, commented, and includes outputs like plots and tables.

5. **Computer Practicals**:

   - Include PDFs of all Quarto Markdown files from your computer practicals by printing the html as pdf.

6. **Group Reflection**:

   - Write a 1-2 page reflection on:
     - Your teamwork experience (challenges and successes).
     - What you learned from working on the data story.

- How the course contributed to your growth in data analysis and collaboration.

3.1.2 Submission

- Combine all the components into a **single PDF** file.
- Name your file as: `LBOMETR[Section_GroupNo.]_DataStoryArchive.pdf`
- **Deadline**: [11 April 2025, 21:00].
- In the event that the file is too big for Animospace, kindly submit as pdf to my email.

3.1.3 Grading Rubric for Data Story Archive

The grading rubric for the Data Story Archive is divided into three categories: **Content**, **Analysis and Technical Work**, and **Overall Presentation Quality**.

Category	**Criteria**	**Points**	**Description**
1. Content			

Category	Criteria	Points	Description
	Clarity of Objective	10	Clearly defined problem/question and its relevance to the course.
	Data Story Report	20	Completeness and quality of the report, including introduction, methods, results, and discussion.
	Appendix	10	Completeness of additional materials (e.g., tables, plots) in the appendix.
2. Analysis and Technical Work			

Category	Criteria	Points	Description
	R Script Quality	15	Well-structured, commented, and reproducible R script with outputs rendered as a PDF.
	Practical Assignments	15	Quality and completeness of PDFs rendered from Quarto Markdown files.
	Visualizations	15	Clear, meaningful, and well-designed plots and tables generated in R.
3. Overall Presentation Quality			

Category	Criteria	Points	Description
	Group Reflection	15	Thoughtful insights on teamwork, learning, and course experience.
	Formatting and Organization	10	Overall organization, formatting, and adherence to submission guidelines.
	Total	**100**	

3.2 Data Story Presentation

The **Data Story Presentation** is your group's opportunity to communicate your findings and insights through a live presentation. This format allows you to showcase animated visualizations and engage directly with the audience in real time. A room will be requested for you to be able to present in front of your classmates and I will be present online *hopefully this will be applicable;*

3.2.1 Requirements

1. **Objective**:

    - Your live presentation should effectively communicate your data story with clarity, engagement, and professionalism, making full use of visuals and animations to enhance understanding.

2. **Presentation Structure**: The presentation must include the following sections:

    - **Introduction**: Briefly introduce your topic, research question, and the significance of your data story (1 slide).
    - **Methods**: Provide a concise explanation of your data and analysis methodology (1-2 slides).
    - **Results**: Highlight the most important findings using R-generated visualizations, including animations if applicable (3-4 slides).
    - **Discussion and Conclusion**: Discuss the implications of your findings and conclude with actionable insights or recommendations (1 slide).

3. **Delivery**:

    - Each group member must actively participate in the presentation.
    - Presentation duration: **10 minutes**, followed by a **5-minute Q&A session**.

4. **Visualizations**:

    - Use animated or interactive visualizations (e.g., created with `gganimate` or other R packages) to effectively demonstrate key

trends and insights.
- Ensure visuals are clear, professional, and aligned with your narrative.

5. **Tools**:
   - Create your presentation using tools like Google Slides, Microsoft PowerPoint, or Canva.
   - Incorporate animated visualizations as needed.

6. **Submission**:
   - Submit your presentation slides as a **PDF file** named:

     `LBOMETR[Section_GroupNo.]_DataStoryPresentation.pdf`

   - Submit the file before your scheduled presentation time.

3.2.2 Grading Rubric

The grading rubric for the Data Story Presentation is divided into three categories: **Content**, **Visualizations**, and **Delivery and Engagement**.

Category	Criteria	Points	Description
1. Content			

Category	Criteria	Points	Description
	Introduction and Methods	10	Clear and concise introduction and explanation of methods.
	Results	20	Logical flow and depth of results, focusing on key findings.
	Discussion and Conclusion	10	Insightful discussion and actionable conclusion.
2. Visualizations			
	Quality of Visuals	20	Professional and well-designed visualizations, including appropriate use of animations.

Category	Criteria	Points	Description
	Relevance of Visuals	10	Visuals strongly support the analysis and enhance understanding.
3. Delivery and Engagement			
	Delivery	20	Confident, clear, and professional delivery by all group members.
	Audience Engagement	10	Creativity and ability to maintain audience attention.
	Q&A Session	10	Ability to effectively respond to audience questions.

Category	**Criteria**	**Points**	**Description**
	Time Management	10	Adherence to the 10-minute time limit and logical pacing.
	Total	**100**	

Chapter 4

Grouping Process

Students will be randomly assigned to groups of **4-5 members** based on their responses to a pre-course survey. The survey collects information that will be used to ensure fair and balanced groupings. The group assignments will be announced on the first day of the course.

4.1 Survey

Please complete the survey **before 14:30 PM on January 6, 2025** using the link:

- Google Form Survey Link

4.2 How Groups Are Formed

The groupings are created using RStudio. The coding ensures randomness while incorporating some aspects of the survey responses to balance groups.

If you wish to see the code used for grouping, you may contact me directly. However, please note: - The **CSV file with survey responses will not be shared** to protect your anonymity and privacy.

4.3 Announcement of Groups

The group assignments will be distributed on the **first day of the course**. Please check your assigned group and connect with your group members as soon as possible.

Chapter 5

Data Story Research Question

5.1 Guidelines in Conducting Data Story

5.1.1 Research Question:

In writing your Research Question, you should keep in mind the following acronym:

S-M-A-R-T

S-Specific

M-Measurable

A-Achievable

R-Relevant

T-Time-bound

What do you think of this research question:

“*What is the effect of X(Twitter) on mental health?*“

Do you think this question is SMART?

How about this question:

“*What is the relationship between hours spent per day on X/Twitter and reported levels of anxiety among undergraduate students in DLSU during 2020-2021?*“

Which of the two achieves the SMART elements?

5.1.2 Relevance

What will be asked by the audience?

You need to ask yourselves the following questions; if you are able to, you are close to achieving a good presentation!

1. What is the problem to be solved?
2. Who cares about this problem and why?

5.1.3 For the data story, should I use Kaggle?

The answer is **NO** because of the following reasons:

1. It has already been cleaned and curated based on the owner’s own purposes. It may not accurately reflect the real-world data.
2. It has limited context, potentially limiting your ability to interpret findings accurately.

3. While some Kaggle datasets have clear sources, others may have unclear origins or have undergone modifications by multiple users raising concerns about data integrity.

Thus, you should use credible sources like statistics from World Bank, government sites, etc. I am not saying that you can already drop Kaggle datasets. You can still use them for the following:

1. Benchmarking: Compare your model on a well-known Kaggle dataset
2. For learning: You can try to match how the datasets in Kaggle have been cleaned. You can also use the datasets to practice your codes before applying them to the data you found from other credible sources.

Important to note: Check data documentation from Kaggle; this will help you find the sources and will help supplement your data story with reputable sources.

Furthermore, this is LBOMETR class wherein you were taught how to clean a dataset. You have to gather and clean the data yourselves to align with your research question.

5.2 Comments on Research Questions:

Please check your emails for comments regarding your research questions. The approved and final research questions will be posted in Animospace as an announcement.

Chapter 6

Basic Introduction to R

This portion of the book offers an introduction to the basics of R. R offers a wide variety of functionality. Note that this book only offers basic Econometric analysis. It will be useful to have some basic familiarity with R and its syntax but this is not strictly necessary.

Each chapter includes both R code and results to make it easier for students to follow along, even without detailed knowledge of R.

6.1 Session Information

This version of the book was built using R version 4.4.2. See below for the session information:

```
## R version 4.4.2 (2024-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 22631)
```

```
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_Netherlands.utf8  LC_CTYPE=English_Netherlands.utf8
## [4] LC_NUMERIC=C                         LC_TIME=English_Netherlands.utf8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] zoo_1.8-12      bookdown_0.41   lubridate_1.9.4 forcats_1.0.0   stringr_1
##  [8] readr_2.1.5     tidyr_1.3.1     tibble_3.2.1    ggplot2_3.5.1   tidyverse
##
## loaded via a namespace (and not attached):
##  [1] sass_0.4.9        utf8_1.2.4        generics_0.1.3    lattice_0.22-6    s
##  [7] digest_0.6.37     magrittr_2.0.3    evaluate_1.0.1    grid_4.4.2        t
## [13] cellranger_1.1.0  jsonlite_1.8.9    tinytex_0.54      scales_1.3.0      j
## [19] rlang_1.1.4       munsell_0.5.1     withr_3.0.2       cachem_1.1.0      y
## [25] tzdb_0.4.0        colorspace_2.1-1  vctrs_0.6.5       R6_2.5.1          l
## [31] pillar_1.10.0     bslib_0.8.0       gtable_0.3.6      glue_1.8.0        x
## [37] rstudioapi_0.17.1 knitr_1.49        farver_2.1.2      htmltools_0.5.8.1 r
## [43] compiler_4.4.2
```

6.2 Preliminaries

The first step is to gain access to R, which is free and available on the R website: http://cran.r-project.org/. Simply go to the R website, select the appropriate location and operating system, and follow the instructions to download the base distribution of R. RStudio offers a user friendly environment to run R and is recommended.

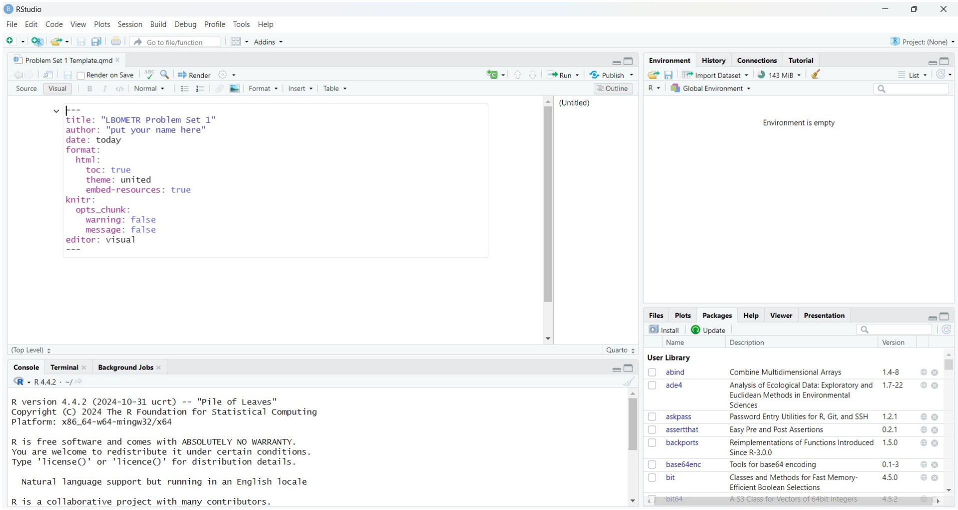


Figure 6.1: RStudio Screen

Once R is opened, we can begin to run commands. R commands can be run directly from the console, from the R script editor or from a text editor separate from R.

R offers detailed help files for each function. To access help, run:

```
?sum
```

All lines proceeded by a # are comments and will not run. For example:

```
# This is a comment. R will not recognize this as a command.
```

6.3 Quarto Markdown

In LBOMETR, Quarto Markdown will be used by the students when submitting the Scripts for the Data Story Archive. Quarto Markdown is a tool for creating documents, reports and presentations using Markdown and executable code. Below is a concise guide to help you get started, along with key shortcuts for both Mac and Windows.

6.3.1 1. Starting a Quarto File

To begin creating a Quarto document, follow these steps:

1. Open RStudio.
2. Go to **File > New File > Quarto Document.**
3. Choose the document type (e.g., HTML, PDF, Word, etc.) and specify whether the document will include code. For ease, we will use the html document type. I have also added a sample Quarto Markdown file you can copy.

   Quarto Markdown Template

   ### 6.3.2 2. Quarto Key Features

   #### Code Chunks

   Code chunks allow you to include and run code inside your document.

Inline Code

Embed R code in text using backticks and `r` .

6.3.3 Quarto Markdown Shortcuts

Action	Windows Shortcut	Mac Shortcut
Insert a new code chunk	Ctrl+Alt+I	Cmd+Alt+I
Run current code chunk	Ctrl+Shift+Enter	Cmd+Shift+Enter
Run all code chunks	Ctrl+Alt+R	Cmd+Alt+R
Run current line/selection	Ctrl+Enter	Cmd+Enter
Knit/Render document	Ctrl+Shift+K	Cmd+Shift+K
Comment/uncomment lines	Ctrl+Shift+C	Cmd+Shift+C
Insert pipe (%>%)	Ctrl+Shift+M	Cmd+Shift+M
Headings	/Number of Heading (if in Visual mode) Prefix line with #, ##, etc. manually (in Source mode)	/Number of Heading (if in Visual mode) Prefix line with #, ##, etc. manually (in Source mode)

Action	Windows Shortcut	Mac Shortcut
Bold	Ctrl+B	Cmd+B
Italic	Ctrl+I	Cmd+I
Inline code	Surround with backticks (') manually	Surround with backticks (') manually

*Note: you can choose between Source or Visual (upper left); personally, it is easier for me to use the Visual Mode compared to the Source Mode.

6.4 Packages

Each package of interest must be installed and loaded before it can be used. The packages will not be immediately available when R is opened. A package only has to be installed once on a computer, but the package will have to be loaded every time R is restarted.

We can install a package individually as we need them. For example, to install **tidyverse** and **psych**, we would do:

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

In the tidyverse package, the **ggplot2** is usually included; if you do not see the package in the Packages list at the lower right, you can do this:

```
if(!("ggplot2" %in% installed.packages()[,"Package"])) install.packages("ggplot2")
```

Now that we have our packages successfully installed, we can go ahead and load them into R. Here we will load the tidyverse package as an example. We can use of all the functions available in that package once it is loaded into R. We load packages by using a `library()` function. The input is the name of the package, not in quotes.

```
library(tidyverse)
```

We can look up all of the functions within a package by using a `help()` function. For example, let's look at the functions available in the **tidyverse** package.

```
help(package = tidyverse)
```

Note that the package argument is necessary to look up all of the functions. We can also detach a package if we no longer want it loaded. This is sometimes useful if two packages do not play well together. Here we will use a `detach()` function.

```
detach(package:tidyverse)
```

For simplicity, we will assume that the reader has restarted R at the beginning of each tutorial.

6.5 Instructions for Managing Working Directories

This guide outlines how team members should set up their local working directories for collaboration, handle `.qmd` files, and organize them in a shared Google Drive.

6.5.1 1. Local Working Directory Setup

Each team member should create a local folder on their own laptops to work on `qmd` files. This folder is where you will store and edit your files before uploading them to the shared Google Drive.

6.5.1.1 Steps:

1. Create a folder on your laptop named: **DLSU_LBOMETR_Section**
2. Use this folder to save and organize your `.qmd` files while working locally.

6.5.2 2. File Naming Convention

To avoid confusion, ensure all `.qmd` files are named as follows:

- Include your name or initials and a brief description of the content
- Example:
  - `jem_nario_descriptivestatistics.qmd`
  - `jmn_piechart.qmd`

6.5.3 3. Shared Google Drive Setup

A **shared Google Drive** will serve as the central repository for all project files, including:

- `.qmd` files from all team members
- Data files
- Rendered HTML and PDF files for final submission
- Supporting documents or references.

6.5.4 4. Workflow for `.qmd` files

For each team member:

1. Work locally
    - Create your `.qmd` file in your local `DLSU_LBOMETR_Section` folder
    - Ensure it is well-documented and organized.
2. Upload to Google Drive

For the Team Leader:

1. Collect and Combine Files
    - Gather all `.qmd` files from the team folder on the shared drive.
    - Combine them
2. Render the final report

6.5.5 5. Rendering the Final Report

The final report should be rendered in HTML and printed by the team leader.

6.5.6 Summary Workflow

- **Each Team Member**:
  - Work on your `.qmd` file locally.
  - Upload your file to the shared Google Drive under `team-members-qmd`.
- **Team Leader**:
  - Collect `.qmd` files from the shared drive.
  - Combine them into a single `final_report.qmd`.
  - Render the final report into HTML and PDF.
  - Upload the rendered files to the `final-report` folder on the shared Google Drive.

This ensures an organized and efficient workflow while centralizing all files in the shared Google Drive for easy access and submission.

Chapter 7

Data Management - Cross-Sectional Data

7.1 Where to Get Data?

Before we proceed to Data Management, let us first find where we can get data for the Data Story Archive. Note that the data you collect should still ensure that you are following the Code of Ethics and analyze Ethical Considerations.

Please view the necessary documents from the Office of the Vice Chancellor for Research and Innovation (https://www.dlsu.edu.ph/research/research-manual/)

A list of links you can search and get data from:

Note: I will not include the best links as they are pretty straightforward and these are governmental databases like the ones from World Bank, IMF, UN, Philippine Statistics Authority, and Bangko Sentral ng Pilipinas. The list here is a general list but use with proper discretion.

Name	Link	Notes
Kaggle	https://www.kaggle.com/datasets	Kaggle is where users can provide datasets; it is important to cite the sources. Mostly, datasets in Kaggle can be used for your practice.
Awesome Public Datasets	https://github.com/awesomedata/awesome-public-datasets	This repository is filled with public datasets, mostly from International contexts.
Google Dataset Search	https://datasetsearch.research.google.com/	You can download publicly available datasets from searching through Google. Though, sometimes the datasets come from 'Statista.com'. You can check the sources from the search.

7.2 Preliminaries

7.2.1 Dataset

The dataset to be used can be downloaded here: Chapter2 Practice and will be included in the Files in Animospace. The dataset was modified from the Wooldridge package in R as practice material. The particular dataset is the 'htv' dataset.

```
#install the wooldridge package. Check previous chapter on how to install packages.
load(wooldridge)
?htv #to find out about the particular dataset.
```

NOTE: The htv dataset help will tell you what the variables mean, however, for our practice, we will use the modified version of this dataset.

7.2.2 Packages

We will mostly use the `tidyverse` package, in particular, the `dplyr` package and the `tidyr` package; double-check in your Packages list whether you have these two packages; if not, you can simply install them.

7.2.3 Setting up the Directory

This is the most important step! Make sure to place the downloaded file in this folder: **DLSU_LBOMETR_Section** in your laptops. Remember, this is your local working directory. This is the working directory you choose. You can set up your working directory in the following ways:

1. Using the R Studio Menu (works for both Mac and Windows)

   a. Go to **Session > Set Working Directory > Choose Working Directory**

2. Windows:

```
# Use double backslashes `\\` or forward slashes `/`
setwd("C:\\Users\\YourUsername\\Documents")  # Example with backslashes
setwd("C:/Users/YourUsername/Documents")     # Example with forward slashes
```

3. Mac

```
# Use forward slashes `/`
setwd("/Users/YourUsername/Documents")
```

To check:

```
getwd()
```

7.2.4 Clean Everything

Do this step every time you use other data or when we do the other chapters.

```
# Remove all objects in the global environment
rm(list = ls())

# Perform garbage collection to free up memory
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2288255 122.3    3701068 197.7  3701068 197.7
## Vcells 6391397  48.8   12255594  93.6 12255594  93.6
```

7.2.5 Importing the Dataset

We can use the `read.csv()` to load the `csv` file into R. Always call the file as something short and easily understandable. Ensure the downloaded file is in the working directory before you load the file. If the downloaded file is not located in the working directory, you will encounter issues.

I will name the file as `ch2_p1`

```
ch2_p1<-read.csv("Ch2Practice.csv")
```

We can use the `head()` function to inspect the first six rows of the dataset:

```
head(ch2_p1)
```

```
##        WAGE  ABILITY EDUCATION NORTHEAST NORTHCENTRAL WEST SOUTH EXPERIENCE MOTHEREDUC
## 1 12.019231 5.027738        15        no           no  yes    no          9         12
## 2  8.912656 2.037170        13       yes           no   no    no          8         12
## 3 15.514334 2.475895        15       yes           no   no    no         11         12
## 4 13.333333 3.609240        15       yes           no   no    no          6         12
## 5 11.070110 2.636546        13       yes           no   no    no         15         12
## 6 17.482517 3.474334        18       yes           no   no    no          8         12
##   SIBLINGS URBAN X18INNORTHEAST X18INNORTHCENTRAL X18INSOUTH X18INWEST X18INURBAN X17T
## 1        1   yes              1                 0          0         0          1   7.
## 2        4   yes              1                 0          0         0          1   8.
```

```
## 3         2   yes              1                 0          0         0
## 4         1   yes              1                 0          0         0
## 5         2   yes              1                 0          0         0
## 6         2   yes              1                 0          0         0
##   EXPER.2
## 1      81
## 2      64
## 3     121
## 4      36
## 5     225
## 6      64
```

7.3 Data Cleaning

As you can see, there are 22 columns. Let's simplify by only choosing the following: WAGE, URBAN, X17TUITION, X18TUITION and EXPER.2. We can do this using the `select()` function in `dplyr`. We will save them into a new data frame, `ch2_p1.1`.

```
library(dplyr)
```

```
ch2_p1.1<-select(ch2_p1, #the original dataset
       WAGE, URBAN, X17TUITION, X18TUITION, EXPER.2)
```

7.3.1 Renaming the Variables

We will edit the names to much easier conventions. First, let us say that we just want to change them to lowercase names.

```
names(ch2_p1.1)<-tolower(names(ch2_p1.1))
```

Inspect:

```
head(ch2_p1.1)
```

```
##        wage urban x17tuition x18tuition exper.2
## 1 12.019231   yes   7.582914   7.260242      81
## 2  8.912656   yes   8.595144   9.499537      64
## 3 15.514334   yes   7.311346   7.311346     121
## 4 13.333333   yes   9.499537  10.162070      36
## 5 11.070110   yes   7.311346   7.311346     225
## 6 17.482517   yes   7.311346   7.311346      64
```

Let us change the names of `x17tuition, x18tuition,`and`exper.2` to the names similar to what is found in the 'htv' dataset: `x17tuition` to `tuit17`, `x18tuition` to `tuit18` and `exper.2` to `expersq` . To do this, we will use the `rename()` in `dplyr`. We will also use the (`%>%`) for this.

```
ch2_p1.1<-ch2_p1.1 %>%
  rename(
    tuit17 = x17tuition,
    tuit18 = x18tuition,
```

```
    expersq = exper.2
  )
```

Inspect again:

```
head(ch2_p1.1)
```

```
##        wage urban   tuit17    tuit18 expersq
## 1 12.019231   yes 7.582914  7.260242      81
## 2  8.912656   yes 8.595144  9.499537      64
## 3 15.514334   yes 7.311346  7.311346     121
## 4 13.333333   yes 9.499537 10.162070      36
## 5 11.070110   yes 7.311346  7.311346     225
## 6 17.482517   yes 7.311346  7.311346      64
```

7.3.2 Sorting certain values

Let's say, we want to arrange `wage`. We will create a different data for this. We use `arrange` in `dplyr` package.

```
ch2_p1sort<-arrange(ch2_p1.1,
                    wage)
```

Inspect:

```
head(ch2_p1sort)
```

```
##       wage urban    tuit17    tuit18 expersq
```

```
## 1 1.023529   yes  2.088957  2.251239     169
## 2 1.073345    no  7.520245  7.520245     196
## 3 1.102362   yes  7.355460  6.922371     196
## 4 1.250000   yes  9.417682  8.826549     100
## 5 1.373626   yes  9.692757  9.692757     225
## 6 1.442308    no 11.280367 11.280367      NA
```

What if you have this kind of data?

```r
head(ch2_p2)
```

```
##    ch2_p2
## 1 $10,000
## 2 $20,500
## 3 $15,250
## 4 $30,000
## 5 $50,750
```

Let's sort this:

```r
ch2_p2sort<-arrange(ch2_p2)
head(ch2_p2)
```

```
##    ch2_p2
## 1 $10,000
## 2 $20,500
## 3 $15,250
## 4 $30,000
## 5 $50,750
```

It did not work. The problem is, `ch2_p2` is not numeric. We can check:

```
class(ch2_p2$ch2_p2)
```

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

We need to make it into a numeric value but we have a `,` and $. We need to remove them. We use the `str_replace` function in the `stringr` package.

```
library(stringr)
```

```
ch2_p2$ch2_p2<-str_replace(
  ch2_p2$ch2_p2, #column we want to edit
  pattern = ',', #what to find
  replacement = '' #what to replace it with
)
```

```
head(ch2_p2)
```

```
##   ch2_p2
## 1 $10000
## 2 $20500
## 3 $15250
## 4 $30000
## 5 $50750
```

Now, let us remove the dollar sign; usually, simply doing the same thing we did with the comma works, but, there are some symbols that are used as "special character". To "force" R to replace the presence of '$', we add two backslashes before the dollar sign.

```
ch2_p2$ch2_p2<-str_replace(
  ch2_p2$ch2_p2,
  pattern = '\\$',
  replacement = ''
)
```

Can you inspect it on your own?

Simply type the code in the empty code chunk then run it by pressing **Ctrl+Enter** or **Cmd+Enter**

Now, sort `ch2_p2`

```
ch2_p2sort<-arrange(
  ch2_p2,
  ch2_p2
)
```

```
head(ch2_p2sort)
```

```
##   ch2_p2
## 1  10000
## 2  15250
## 3  20500
## 4  30000
## 5  50750
```

We can see that it was arranged, however, take a look at the way ch2__p2 was encoded; it is not numeric. So, we need to change this.

```
class(ch2_p2$ch2_p2)
```

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

Change to numeric through `as.numeric()`

```
ch2_p2$ch2_p2<-as.numeric(ch2_p2$ch2_p2)
```

Inspect on your own:

7.3.3 Pipe Operator

`%>%` allows functions to be chained; it can be read as "then" - it tells R to do whatever comes after it to the stuff that comes before it.

7.3.4 Adding columns

We will be using the pipe operator and the `mutate` to add a new column to `ch2_p1.1` based from details found in `ch2_p1`, particularly, NORTHEAST, NORTHCENTRAL, WEST, and SOUTH. We will call this new column as `location`

```
ch2_p1.1<-ch2_p1.1 %>%
  mutate(
    location = case_when( #creates conditional statements
```

```
      ch2_p1$NORTHEAST == "yes" ~ "northeast", #If NE is "yes", location is "northeast"
    ch2_p1$WEST == "yes"~"west", #If WEST is "yes", location is "west"
    ch2_p1$NORTHCENTRAL == "yes"~ "northcen",
      ch2_p1$SOUTH == "yes"~ "south",
    TRUE~"other")
  )
```

Inspect the data:

Now I want you to create a new column called, `tuit_diff` wherein it is the difference between `tuit18` and `tuit17`. In this case, there is no need to use `case_when` since there is no conditional statements to be used. It is straightforward that you simply need to subtract `tuit17` from `tuit18`. You will need to use `mutate` still. How will you create that?

7.3.5 Transforming values

Now, you can see that urban is a `character` that is "yes/no". We need to change that to numeric value. This is particularly useful when we use dummy variables later on. We will not use `case_when` as it is not necessary; rather, we will use `ifelse`:

```
ch2_p1.1<-ch2_p1.1 %>%
  mutate(
    urban = ifelse(urban=="yes", 1,0) #replace "yes" with 1 and "no" with 0
  )
head(ch2_p1.1) #default is first 6 rows
```

```
##        wage urban   tuit17    tuit18 expersq  location
## 1 12.019231     1 7.582914  7.260242      81      west
## 2  8.912656     1 8.595144  9.499537      64 northeast
## 3 15.514334     1 7.311346  7.311346     121 northeast
## 4 13.333333     1 9.499537 10.162070      36 northeast
## 5 11.070110     1 7.311346  7.311346     225 northeast
## 6 17.482517     1 7.311346  7.311346      64 northeast
```

Now, I want you to create groups for `expersq`. `NA` should now be 0, assign 1 if less than 50, assign 2 if between 50 and 100, assign 3 if between 100 and 200, and for the rest, assign 4.

Clue: conditional statements like between 50 and 100 should be like this:

`values>=50 & values < 100`

Your answer should look like this:

```
##        wage urban   tuit17    tuit18 expersq  location
## 1 12.019231     1 7.582914  7.260242       2      west
## 2  8.912656     1 8.595144  9.499537       2 northeast
## 3 15.514334     1 7.311346  7.311346       3 northeast
## 4 13.333333     1 9.499537 10.162070       1 northeast
## 5 11.070110     1 7.311346  7.311346       4 northeast
## 6 17.482517     1 7.311346  7.311346       2 northeast
```

7.3.6 Summarizing

Let us get the average of wages by location, which we'll call `ave.wage`, by using the `group_by()` and `summarise()` functions in dplyr

```
ch2_p1.1ave<-ch2_p1.1 %>%
  group_by(location) %>% #group by location, THEN
  summarise(ave.wage=mean(wage)) #calculate the mean of wages for each location
head(ch2_p1.1ave)
```

```
## # A tibble: 4 x 2
##   location  ave.wage
##   <chr>        <dbl>
## 1 northcen      12.5
## 2 northeast     15.0
## 3 south         12.4
## 4 west          14.1
```

Say that you want to see the average wage in the south area. We can do this by using `filter()`

```
ch2_p1.1ave %>% filter(location=="south")
```

```
## # A tibble: 1 x 2
##   location ave.wage
##   <chr>       <dbl>
## 1 south        12.4
```

How would you sort the dataset by average wage, from highest to lowest? Now you see that it is arranged alphabetically, so how will you arrange it?

7.3.7 Merging datasets

We have two main datasets, `ch2_p1.1` and `ch2_p1.1ave`. By doing this, we could compare side-by-side each observation compared to the average per location.

We will join the datasets by `location` variable, since that is consistent across both datasets. We name the new file as `ch2_p1merged`:

```
ch2_p1merged<-merge(x=ch2_p1.1, y=ch2_p1.1ave, by="location")
head(ch2_p1merged)
```

```
##   location      wage urban   tuit17   tuit18 expersq ave.wage
## 1 northcen 15.294118     1 8.334936 8.334936       3 12.54078
## 2 northcen 17.006804     1 8.334936 8.334936       0 12.54078
## 3 northcen  3.755868     1 8.334936 8.334936       3 12.54078
## 4 northcen  5.288462     1 6.742574 7.198132       2 12.54078
## 5 northcen  9.072165     1 7.305873 7.356897       0 12.54078
## 6 northcen  9.384164     1 8.334936 8.334936       3 12.54078
```

7.3.8 Splitting datasets

Say I want to save different datasets based on the `location` column.

```
northcen_data<-ch2_p1.1 %>% filter(location=="northcen")
```

You can save it as a `.csv` file:

```r
write.csv(northcen_data, "northcentral.csv", row.names = FALSE)
```

Can you do the others?

7.3.9 Dates

We are going to work on dates when we move to the next chapter but, here is something initial and necessary.

```
##   ID date_of_birth
## 1  1    15-05-1990
## 2  2    20-08-1985
## 3  3    01-12-2000
## 4  4    10-03-1995
## 5  5    25-07-2010
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ ID           : int  1 2 3 4 5
##  $ date_of_birth: chr  "15-05-1990" "20-08-1985" "01-12-2000" "10-03-1995" ...
```

To convert the `character` format to date format, we do this:

```r
date_data$date_of_birth <- as.Date(date_data$date_of_birth, format = "%d-%m-%Y")

head(date_data)
```

```
##   ID date_of_birth
## 1  1    1990-05-15
```

```
## 2  2    1985-08-20
## 3  3    2000-12-01
## 4  4    1995-03-10
## 5  5    2010-07-25
```

Say you want to calculate the age:

```
date_data$age <-as.numeric(floor((Sys.Date()-date_data$date_of_birth)/365.25))
head(date_data)
```

```
##   ID date_of_birth age
## 1  1    1990-05-15  34
## 2  2    1985-08-20  39
## 3  3    2000-12-01  24
## 4  4    1995-03-10  29
## 5  5    2010-07-25  14
```

7.3.9.1 Custom Reference Date:

```
ref_date<-as.Date("2020-01-01")
date_data$age2<-as.numeric(floor((ref_date-date_data$date_of_birth)/365.25))
head(date_data)
```

```
##   ID date_of_birth age age2
## 1  1    1990-05-15  34   29
## 2  2    1985-08-20  39   34
## 3  3    2000-12-01  24   19
```

```
## 4  4     1995-03-10  29    24
## 5  5     2010-07-25  14     9
```

Chapter 8

Data Management Practical

In your Quarto Markdown files, you need to answer the following questions. Answers will be given the week after the Practical as a form of Feedback. You can answer by group. It would be great to include which group member did what.

In the first part of the practical, answer the following reflection:

1. Do you think you were able to input correctly all the codes in the empty code chunks? Why do you think so? What did you find difficult?

Now comes the practical proper:

Using the `kpop idols` dataset which you download: Practical 1 - also made available in Animospace, answer the questions.

Ensure that you can render individually before uploading in your shared Google Drive. Failure to render the file means you were unable to do the

Practical. The leader must take note of this since accomplishing the practicals are part of your grade in Group Participation and the Data Story Archive.

1. Separate the dataset into two datasets: one for `males` and one for females. Save these datasets as `males.csv` and `females.csv`
2. Edit the column names for both `males` and `females` datasets to make them shorter, easier to understand, and consistent
3. Remove the following columns from both datasets: Instagram, Korean Name, K.Stage Name, Stage Name
4. How many males and females are in the dataset? *Hint:* `nrow()`
5. Create a binary variable `not_seoul` for both datasets.
    1. Assign `1` if `birthplace` is **not Seoul,** and `0` otherwise
    2. Count how many individuals are from Seoul and how many are not for both datasets.
6. How many males are eligible for military service (ages 18-28 as of February 27, 2024)?
    1. Filter the `males` dataset for this age range, how many from the Country of South Korea (only filter according to Country for straightforwardness) and count how many qualify.
7. Assign generations based on age (as of June 29, 2023 when the Korean Age system was abolished) for both datasets.
    1. Generation criteria:
        1. 1st Gen: Age>=40
        2. 2nd Gen: 31<=Age<=39

    3. 3rd Gen: 25<=Age<=30
    4. 4th Gen: Age<=24

   Count the number of individuals in each generation for males and females

8. Create a new column `income` for both datasets using hypothetical values based from your hypothesis on age influencing income levels of idols. Explain first what your hypothesis is - are idols who are older earning more or less? Why or why not? Also think if you believe females earn more than males or vice versa?

   1. Use `set.seed()` for reproducibility and generate random income values.

      1. Please have different income values depending on their age. You can do similar groupings as Step 7 for this.

   2. Compare the mean income

9. Combine the `males` and `females` datasets
10. Calculate the income difference between males and females

    1. Create a column `income_diff` to calculate how much more or less each individual's income is compared to the average income of the other gender.

11. Save the final dataset named **Ch2_Practical_Section_GrpNo.csv**

Chapter 9

Data Management (Cross-Sectional) Feedback

This feedback will contain only what the answers should look like and some clues and hints. It does not contain the entire codes.

1. Loading the dataset and loading the needed libraries: `dplyr` and `lubridate`

```
##   Stage.Name.Stage.Name Full.Name.Full.Name Korean.Name.Korean.Name K..Stage.Name.K..St
## 1              Taeyeon         Kim Taeyeon
## 2                Sunny          Lee Sunkyu
## 3              Tiffany       Hwang Miyoung
## 4              Hyoyeon         Kim Hyoyeon
## 5                 Yuri           Kwon Yuri
## 6             Sooyoung       Choi Sooyoung
##   Date.of.Birth.Date.of.Birth Group.Group Country.Country Height.Height Weight.Weight B
```

```
## 1                    3/9/1989        SNSD     South Korea           160
## 2                   5/15/1989        SNSD     South Korea           158
## 3                    8/1/1989        SNSD     South Korea           163
## 4                   9/22/1989        SNSD     South Korea           158
## 5                   12/5/1989        SNSD     South Korea           167
## 6                   2/10/1990        SNSD     South Korea           170
##   Gender.Gender Instagram.Instagram
## 1             F          taeyeon_ss
## 2             F          svnnynight
## 3             F        xolovestephi
## 4             F         watasiwahyo
## 5             F             yulyulk
## 6             F          hotsootuff
```

2. Separate the dataset into Males and Females and save as CSVs. You might notice that the column name is `Gender.Gender`. You have to type it out. Later, we will change the names.

```
##   Stage.Name.Stage.Name Full.Name.Full.Name Korean.Name.Korean.Name K..St
## 1                 T.O.P      Choi Seunghyun
## 2               Taeyang       Dong Youngbae
## 3              G-Dragon         Kwon Jiyong
## 4               Daesung             Daesung
## 5               Seungri       Lee Seunghyun
## 6               Leeteuk        Park Jeongsu
##   Date.of.Birth.Date.of.Birth  Group.Group Country.Country Height.Height
## 1                   11/4/1987      BIGBANG     South Korea           180
```

```
## 2                 5/18/1988      BIGBANG     South Korea           174
## 3                 8/18/1988      BIGBANG     South Korea           177
## 4                 4/26/1989      BIGBANG     South Korea           178
## 5                12/12/1990                  South Korea           176
## 6                  7/1/1983 Super Junior     South Korea           179
##   Gender.Gender Instagram.Instagram
## 1             M
## 2             M
## 3             M
## 4             M
## 5             M
## 6             M

##   Stage.Name.Stage.Name Full.Name.Full.Name Korean.Name.Korean.Name K..Stage.Name.
## 1               Taeyeon         Kim Taeyeon
## 2                 Sunny          Lee Sunkyu
## 3               Tiffany       Hwang Miyoung
## 4               Hyoyeon         Kim Hyoyeon
## 5                  Yuri           Kwon Yuri
## 6              Sooyoung       Choi Sooyoung
##   Date.of.Birth.Date.of.Birth Group.Group Country.Country Height.Height Weight.Wei
## 1                    3/9/1989        SNSD     South Korea           160
## 2                   5/15/1989        SNSD     South Korea           158
## 3                    8/1/1989        SNSD     South Korea           163
## 4                   9/22/1989        SNSD     South Korea           158
## 5                   12/5/1989        SNSD     South Korea           167
## 6                   2/10/1990        SNSD     South Korea           170
```

```
##   Gender.Gender Instagram.Instagram
## 1             F          taeyeon_ss
## 2             F          svnnynight
## 3             F        xolovestephi
## 4             F         watasiwahyo
## 5             F             yulyulk
## 6             F          hotsootuff
```

3. Edit Column Names and Remove Unnecessary Columns

```
We are going to use the pipe operator for this and if you are lazy to type all th



``` r
#only those that remain
#col<-c("Full.Name.Full.Name", "Date.of.Birth.Date.of.Birth"...)
```


THEN, after you select the columns to keep, you can rename. Note that you need to


```
##       Full_Name       DOB  Grp     Country  Ht Wt            BP Gender
## 1   Kim Taeyeon  3/9/1989 SNSD South Korea 160 44        Jeonju      F
## 2    Lee Sunkyu 5/15/1989 SNSD South Korea 158 43    California      F
## 3 Hwang Miyoung  8/1/1989 SNSD South Korea 163 50 San Francisco      F
## 4   Kim Hyoyeon 9/22/1989 SNSD South Korea 158 48       Incheon      F
```
```

```
## 5     Kwon Yuri 12/5/1989 SNSD South Korea 167 45        Goyang      F
## 6 Choi Sooyoung 2/10/1990 SNSD South Korea 170 48       Gwangju      F
```

```
##         Full_Name        DOB          Grp      Country  Ht Wt        BP Gender
## 1 Choi Seunghyun  11/4/1987      BIGBANG South Korea 180 65     Seoul      M
## 2  Dong Youngbae  5/18/1988      BIGBANG South Korea 174 56 Uljeongbu      M
## 3    Kwon Jiyong  8/18/1988      BIGBANG South Korea 177 58     Seoul      M
## 4        Daesung  4/26/1989      BIGBANG South Korea 178 63   Incheon      M
## 5  Lee Seunghyun 12/12/1990              South Korea 176 60   Gwangju      M
## 6   Park Jeongsu   7/1/1983 Super Junior South Korea 179 59     Seoul      M
```

4. Count the Number of Males and Females

   As mentioned, use `nrow`. Now, I will introduce you to the `cat` function, The `cat` function will print in the result some text and what will be seen when you include the object you created that reveals the number of males (or females). We should all have the same number. If not, something is wrong.

   ```
   #cat("Number of males:", nummale,"\n")
   ```

   ```
   ## Number of males: 843
   ```

   ```
   ## Number of females: 823
   ```

5. Create a binary variable `not_seoul`

Here, create a new column for not_seoul in both males and females. Use `ifelse` and the statement should include `!=`

6. Count Individuals from and not from Seoul

   I introduce the `count` function. Simply add `count` after choosing the dataset. So, the code is choose males THEN count those that are not_seoul. We should have the same number.

   ```
   ## Males from Seoul and not Seoul:
   ```

   ```
   ## From Seoul: 98
   ```

   ```
   ## Not from Seoul: 745
   ```

   ```
   ## Females from Seoul and not Seoul:
   ```

   ```
   ## From Seoul: 108
   ```

   ```
   ## Not from Seoul: 715
   ```

7. Filter Males Eligible for Military Service

   Use reference date and when you filter, you can actually combine filtering the Age to be: Age`>= 18, Age<=28, Country == "South Korea"`. We should have the same number.

```
## Number of males eligible for military service: 496
```

8. Assign Generations Based on Age

We should have the same numbers. If you have created an `Age` column for males before, you cannot use that for this number since the reference dates are different. You need to create a new column for `Age` for both males and females.

```
## Generation distribution for males:

## 1st Gen: 8
## 2nd Gen: 137
## 3rd Gen: 358
## 4th Gen: 340

## Generation distribution for females:

## 1st Gen: 4
## 2nd Gen: 131
## 3rd Gen: 302
## 4th Gen: 386
```

9. Create Income Column Based on Hypothesis

   My hypothesis is that older idols earn more since they have been in the industry much longer. I also hypothesize that females earn less than males.

   I introduce a new function here, it is the `runif(n(), value, value)`. This function is to just create income values which will be in-line with the number of rows in the dataset. However, I am amenable in any strategy you employ to generate income here. I just need to know your hypothesis and how you plan to do the income column. In fact, if you want, you can even create the income column in Excel already. Just let me know.

Another strategy is to still do `case_when` and have smaller increments in `Age` then have income values. Another, have the same income for each generation, that is also fine. Again, this is for practice purposes.

```
# Generate income values based on age group directly
set.seed(123)  # For reproducibility

males <- males %>%
  mutate(Income = case_when(
    Age2 >= 40 ~ runif(n(), 50000, 70000),  # 1st Gen
    Age2 >= 31 & Age2 <= 39 ~ runif(n(), 40000, 60000),  # 2nd Gen
    Age2 >= 25 & Age2 <= 30 ~ runif(n(), 30000, 50000),  # 3rd Gen
    Age2 <= 24 ~ runif(n(), 20000, 40000)  # 4th Gen
  ))

females <- females %>%
  mutate(Income = case_when(
    Age2 >= 40 ~ runif(n(), 50000, 70000),  # 1st Gen
    Age2 >= 31 & Age2 <= 39 ~ runif(n(), 40000, 60000),  # 2nd Gen
    Age2 >= 25 & Age2 <= 30 ~ runif(n(), 30000, 50000),  # 3rd Gen
    Age2 <= 24 ~ runif(n(), 20000, 40000)  # 4th Gen
  ))
```

10. Combine datasets

So, I taught you how to merge the datasets. You can do that as well, however, it is important to determine which came from the males, which came from the females so, you need to create a new column in the males and the females that would include the Gender.

I also show a different way of merging datasets. I use `bind_rows` since males and females have the same columns. However, if you view the `combined` dataset, you will notice that the females will appear after the males. This is fine.

```r
# Ensure column names match and add a Gender column
males <- males %>% mutate(Gender = "Male")
females <- females %>% mutate(Gender = "Female")

# Combine the datasets
combined <- bind_rows(males, females)

# View the combined dataset
View(combined)
```

Calculate the Income Difference

I use the `group_by` function and the `summarise` function. I got the Mean and the Median. Later on, I will discuss the difference between getting the mean and the median or when best to use the mean or the median.

```
## # A tibble: 2 x 3
##   Gender Mean_Income Median_Income
##   <chr>        <dbl>         <dbl>
## 1 Female      36858.        36414.
```

```
## 2 Male        37907.        37246.
```

```
## The gender with the higher mean income is: Male
```

```
## The gender with the higher median income is: Male
```

11. Now that we have the combined dataset, simply save it as a CSV file and you are done with the practical.

Chapter 10

Data Management - Time Series and Panel Data

For this portion of the discussion, we will use one dataset then generate randomly here in R some practice data frames.

10.1 Topic Guide:

1. Changing from daily to monthly to quarterly to yearly
2. Change from long to wide and vice-versa
3. Missing values

10.2 Time Series Data

For this, we will use Chapter3 Practice found in the Modules.

10.2.1 Preliminaries

Always remember the first steps: Set Working Directory and Clean the Global Environment.

```
rm(list=ls())
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2288103 122.2    3701068 197.7  3701068 197.7
## Vcells 6413290  49.0   12255594  93.6 12255594  93.6
```

To make sure that you are using the correct directory and that you have all the files you need, use `list.files()` function.

```
list.files()
```

Now, we load the following packages: `dplyr`, `lubridate`, and `zoo`. Make sure you have all 3 installed; if not, install them.

```
library(dplyr)
library(lubridate)
library(zoo)
```

Now we load the csv file:

```
ch3_p1<-read.csv("Ch3Practice.csv")
head(ch3_p1)
```

```
##            ds       y
## 1 2015-06-13 232.402
## 2 2015-06-14 233.543
## 3 2015-06-15 236.823
## 4 2015-06-16 250.895
## 5 2015-06-17 249.284
## 6 2015-06-18 249.007
```

We need to understand our data;

```
str(ch3_p1)
```

```
## 'data.frame':    1825 obs. of  2 variables:
##  $ ds: chr  "2015-06-13" "2015-06-14" "2015-06-15" "2015-06-16" ...
##  $ y : num  232 234 237 251 249 ...
```

10.2.2 Convert to Date Format

As you can see, our ds is what our date column is, however, it is in the `character` format. We need to convert it to the `Date` class. We also need to do this to a copy of the raw data for further modifications.

```
ch3_p1.1<-ch3_p1
head(ch3_p1.1$ds)
```

```
## [1] "2015-06-13" "2015-06-14" "2015-06-15" "2015-06-16" "2015-06-17" "2015-06-18"
```

```
class(ch3_p1.1$ds)
```

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

```
ch3_p1.1$ds <- as.Date(ch3_p1.1$ds, format = "%Y-%m-%d")

class(ch3_p1.1$ds)
```

```
## [1] "Date"
```

10.2.3 Aggregate Data

We now have `daily` data. Say we want to create weekly data, we use the `cut` function to group dates by week, month, quarter and year.

Since we know for sure that the `date` column is in `date` format, no need to check, however, it is always useful to check the class of `date`.

10.2.3.1 Aggregate by Week

Add new columns for each aggregation.

```
ch3_p1.1$week<-cut(ch3_p1.1$ds, breaks = "week")
```

The `cut` is used to divide the date into intervals while the `breaks` specifies that the `date` be divided into weekly intervals.

Check creation of the week column

```
head(ch3_p1.1)
```

```
##           ds       y       week
## 1 2015-06-13 232.402 2015-06-08
## 2 2015-06-14 233.543 2015-06-08
## 3 2015-06-15 236.823 2015-06-15
## 4 2015-06-16 250.895 2015-06-15
## 5 2015-06-17 249.284 2015-06-15
## 6 2015-06-18 249.007 2015-06-15
```

Since we have the `y` column which is actually Bitcoin Price, we need to aggregate that weekly using the `aggregate` function

```
week_y<-aggregate(. ~ week, #refers to all other columns in the dataset and groups them b
                  data=ch3_p1.1,
                  FUN = mean #specifies that the mean function should be applied to the n
                  )
```

You will notice that this creates a separate data frame. We will merge `week_y` with `ch3_p1.1`

```
ch3_p1.1<-merge(ch3_p1.1, week_y, by = "week", #ensures the merge aligns based on the wee
                suffixes = c("","_weekly")) #adds _weekly to the column name to distingui
```

We will slightly do the same thing when aggregating by month, quarter and year. I will do the initial steps, but please do the succeeding steps on your own.

10.2.3.2 Aggregate by Month

```
# Add a month column
ch3_p1.1$month <- format(ch3_p1.1$ds, "%Y-%m")
```

```
# Calculate monthly means
month_y <- aggregate(. ~ month, data = ch3_p1.1, FUN = mean)
```

10.2.3.3 Aggregate by Quarter

This is different since we will use the `paste0` and the `format` functions. The format function extracts the year from the date and extracts the quarter from the date. The paste0 combines the year and quarter without a space between them so that it results in which quarter of which year.

```
ch3_p1.1$quarter <- paste0(format(ch3_p1.1$ds, "%Y"), " ", quarters(ch3_p1.1$ds)
```

10.2.3.4 Aggregate by Year

```
ch3_p1.1$year<-format(ch3_p1.1$ds, "%Y")
```

Can you aggregate the Bitcoin values on your own?

10.3 Modifying Long and Wide Datasets

10.3.1 How to Determine

Aspect	Long Format	Wide Format
Rows	Each row represents a single observation (or measurement)	Each row represents an entity (like a group)
Columns	Variables are split into multiple rows	Variables are spread across multiple columns
Compactness	More rows, fewer columns	Fewer rows, more columns

10.3.1.1 Examples of Long and Wide Formats

10.3.1.1.1 1. Student Scores

10.3.1.1.1.1 Long Format

Student	Subject	Score
Alice	Math	90
Alice	Science	85
Bob	Math	88
Bob	Science	85

10.3.1.1.1.2 Wide Format

Student	Math	Science
Alice	90	85
Bob	88	85

10.3.1.1.2 2. Temperature Data

10.3.1.1.2.1 Long Format

Date	Location	Temperature
2023-01-01	City A	15
2023-01-01	City B	20
2023-01-02	City A	16
2023-01-02	City B	21

10.3.1.1.2.2 Wide Format

Date	City A	City B
2023-01-01	15	20
2023-01-02	16	21

10.3.2 Setting up the Dataset

10.3.2.1 Generate a Long Dataset

For practice, we will generate a long dataset. We will need `set.seed` for reproducibility when you want to run the entire code again and to generate random values, we will use the `rnorm` function.

```
set.seed(123)

#Generate a long dataset
long<-data.frame(
  id = rep(1:5, each = 3), #creates 5 groups and repeated 3 times each
  variable = rep(c("A", "B", "C"), times=5), #Variables A,B,C are created and repeated 5
  value = rnorm(15, mean = 50, sd = 10) #creates random values for 15 observations with m
)
print(long)
```

```
##    id variable    value
## 1   1        A 44.39524
## 2   1        B 47.69823
## 3   1        C 65.58708
## 4   2        A 50.70508
## 5   2        B 51.29288
## 6   2        C 67.15065
## 7   3        A 54.60916
## 8   3        B 37.34939
## 9   3        C 43.13147
## 10  4        A 45.54338
## 11  4        B 62.24082
## 12  4        C 53.59814
## 13  5        A 54.00771
## 14  5        B 51.10683
```

```
## 15  5        C 44.44159
```

10.3.2.2 Converting Long to Wide Format

We will use the `pivot_wider()` function from the `tidyr` package.

```r
library(tidyr)

wide<-pivot_wider(
  data = long,
  names_from = variable, #Columns to become new column names
  values_from = value #Values to put under new columns
)

print(wide)
```

```
## # A tibble: 5 x 4
##      id     A     B     C
##   <int> <dbl> <dbl> <dbl>
## 1     1  44.4  47.7  65.6
## 2     2  50.7  51.3  67.2
## 3     3  54.6  37.3  43.1
## 4     4  45.5  62.2  53.6
## 5     5  54.0  51.1  44.4
```

10.3.2.3 Convert Wide to Long

We will use the converted wide dataset for this. We will use the `pivot_longer` function.

```
long2<-pivot_longer(
  data = wide,
  cols = A:C, #Which columns to collapse
  names_to = "variable", #New column for variable names
  values_to = "value" #new column for values
)

print(long2)
```

```
## # A tibble: 15 x 3
##        id variable value
##     <int> <chr>    <dbl>
##  1      1 A         44.4
##  2      1 B         47.7
##  3      1 C         65.6
##  4      2 A         50.7
##  5      2 B         51.3
##  6      2 C         67.2
##  7      3 A         54.6
##  8      3 B         37.3
##  9      3 C         43.1
## 10      4 A         45.5
## 11      4 B         62.2
```

```
## 12     4 C          53.6
## 13     5 A          54.0
## 14     5 B          51.1
## 15     5 C          44.4
```

10.4 Missing Values

In handling missing values in R, we focus on 3 common methods:

1. Replacing missing values with `0`
2. Removing rows or columns with missing values
3. Replacing missing values with the mean

10.4.1 Setting up a Sample Dataset

```r
set.seed(123)
ch3_p2<-data.frame(
  id = 1:5,
  var1 = c(10, NA, 30, 40, NA),
  var2 = c(NA, 15, 25, 35, 45),
  var3 = rnorm(5, mean = 50, sd=10)
)

print(ch3_p2)
```

```
##   id var1 var2     var3
```

```
## 1  1   10   NA 44.39524
## 2  2   NA   15 47.69823
## 3  3   30   25 65.58708
## 4  4   40   35 50.70508
## 5  5   NA   45 51.29288
```

10.4.2 Replace with 0

10.4.2.1 For a Specific Column:

```
ch3_p2.1<-ch3_p2
ch3_p2.1$var1[is.na(ch3_p2.1$var1)]<-0
print(ch3_p2.1)
```

```
##   id var1 var2     var3
## 1  1   10   NA 44.39524
## 2  2    0   15 47.69823
## 3  3   30   25 65.58708
## 4  4   40   35 50.70508
## 5  5    0   45 51.29288
```

10.4.2.2 For the Entire Dataset

```
ch3_p2.2<-ch3_p2
ch3_p2.2[is.na(ch3_p2.2)]<-0


print(ch3_p2.2)
```

```
##   id var1 var2     var3
## 1  1   10    0 44.39524
## 2  2    0   15 47.69823
## 3  3   30   25 65.58708
## 4  4   40   35 50.70508
## 5  5    0   45 51.29288
```

10.4.3 Remove Rows or Columns with Missing Values

10.4.3.1 Remove Rows with Missing Values

```r
ch3_p2.3<-ch3_p2
ch3_p2.3<-na.omit(ch3_p2.3)
print(ch3_p2.3)
```

```
##   id var1 var2     var3
## 3  3   30   25 65.58708
## 4  4   40   35 50.70508
```

10.4.3.2 Remove Columns with Missing Data

```r
ch3_p2.4<-ch3_p2
ch3_p2.4<-ch3_p2.4[, colSums(is.na(ch3_p2.4)) ==0]
print(ch3_p2.4)
```

```
##   id     var3
```

```
## 1   1 44.39524
## 2   2 47.69823
## 3   3 65.58708
## 4   4 50.70508
## 5   5 51.29288
```

10.4.4 Imputing Missing Values with the Mean

Here, we will impute missing values with the mean and unlike previous methods, this method uses a `for` function to loop through columns and check for columns with missing values. However, this method needs to have numeric format.

```
ch3_p2.5<-ch3_p2
for(col in names(ch3_p2.5)){
  if(is.numeric(ch3_p2.5[[col]])) {#Check if column is numeric
    ch3_p2.5[[col]][is.na(ch3_p2.5[[col]])]<-mean(ch3_p2.5[[col]], na.rm=TRUE)
    }
}
print(ch3_p2.5)
```

```
##   id     var1 var2     var3
## 1  1 10.00000   30 44.39524
## 2  2 26.66667   15 47.69823
## 3  3 30.00000   25 65.58708
## 4  4 40.00000   35 50.70508
## 5  5 26.66667   45 51.29288
```

10.4.5 Note:

When doing the three methods, there are pros and cons.

1. Replacing with Zero can introduce bias if `0` is not realistic in the context of the data
2. Remove Missing Values can result in significant data loss
3. Imputing the Mean assumes the data is evenly distributed and can distort the variability in the data

Always make sure you should understand your data.

10.4.6 Best Practice:

1. Always **explore and visualize** patterns first.

Leading to…

10.4.7 Next Meeting:

1. Visualizing the data with base R package

10.4.8 Final Note:

No Practical for this session; please work on the previous practical and the research problem to be submitted on January 24.

Next week, we will have a computer practical.

Chapter 11

Visualizations using ggplot2

For this lecture, we will be using `ggplot2` package. Please make sure that you have it installed. The package works best with data in the 'long' format so it helps to modify the dataset to this format rather than a wide format.

11.1 Preliminaries

11.1.1 Load the dataset

The dataset can be downloaded from the Modules. We will use `Ch4PracticeA` for this portion of the discussion.

Unlike previous datasets that we loaded, we are now loading an Excel file. We will need the `readxl` package for this. Also, it is important to check if there are additional sheets and which sheet you will need. There are actually two sheets in the Excel file, but we will only use the first sheet named `base`.

```
rm(list=ls())
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2288231 122.3    3701068 197.7  3701068 197.7
## Vcells 6394201  48.8   12255594  93.6 12255594  93.6
```

```
library(readxl)
ch4_p1 <- read_excel("Ch4PracticeA.xlsx",
    sheet = "base")
```

I will not delve deeply in the description. Please read up on it. The description can be found in the last sheet of the Excel file. Next, we load the following package: `tidyverse`

```
library(tidyverse)
```

11.1.2 Template

There is a basic template that can be used for different types of plots:

```
<DATA> %>%
  ggplot(aes(<MAPPINGS>))+
  <GEOM_FUNCTION>()
```

`ggplot` is a function that expects a data frame to be the first argument. This allows for us to change from specifying the `data =` argument within the ggplot function and instead pipe the data into the function.

Use the `ggplot()` function…

```
ch4_p1 %>%
  ggplot()
```

Now, we define the mapping (using the aesthetic (`aes`) function), by selecting the variables to be plotted and specifying how to present them in the graph, e.g. as x/y positions or characteristics such as size, shape, color, etc.

```
ch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures))
```

The next step is to add `geom` which will make the graphical representations of the data. These include:

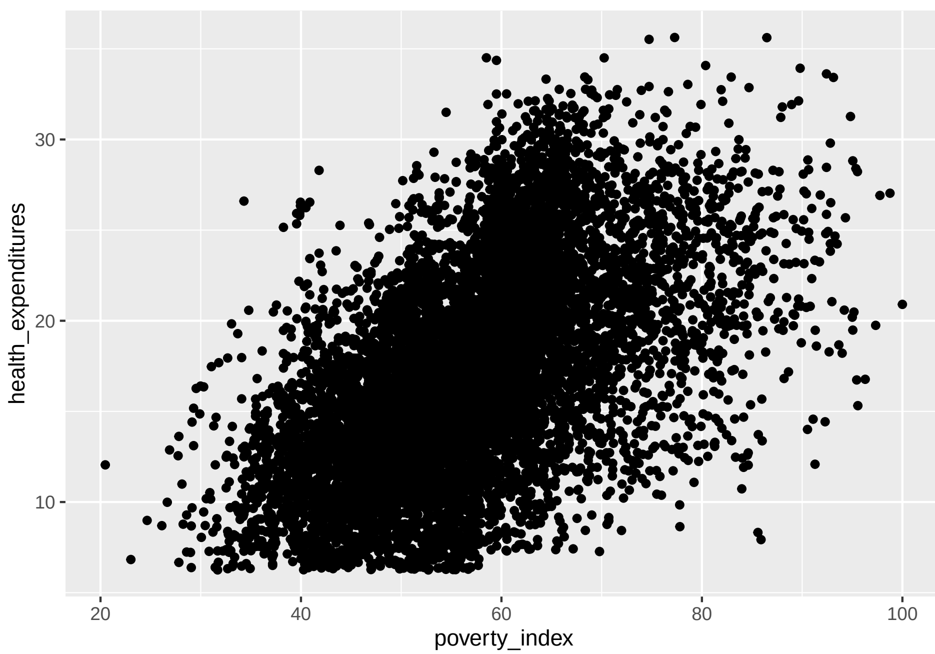
- `geom_point()` for scatter plots, dot plots, etc.
- `geom_boxplot()` for boxplots
- `geom_line()` for trend lines, time series, etc.
- `geom_bar()` for bar plots and pie charts

11.2 Visualizing Data using `geom`

11.2.1 Scatterplots

Let us use the `geom_point()` first then we will do the others after. Also, scatterplots are useful when you want to display the relationship between two continuous variables. Can you give me an example of when to use scatterplots?

```
ch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures)) +
  geom_point()
```

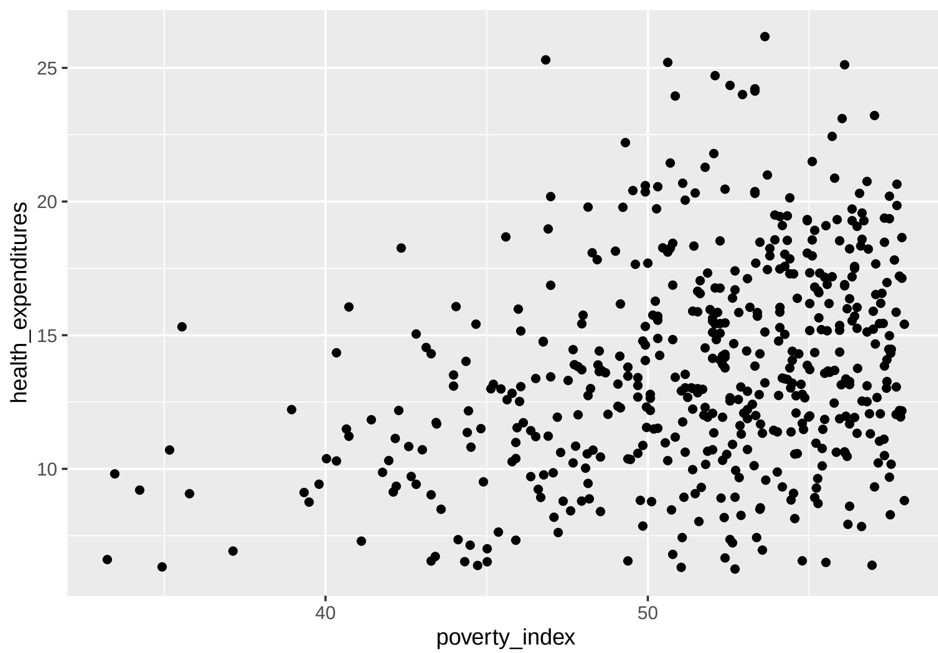


This visualization is so unclear; this is due to the number of observations being more than 9,000. Let's just use the first 500 rows as this is for our practice and for visualization purposes.

```
fch4_p1 <- head(ch4_p1, 500)
View(fch4_p1)
```

This is more manageable; let's try the scatterplot again, this time using the `fch4_p1` dataset.

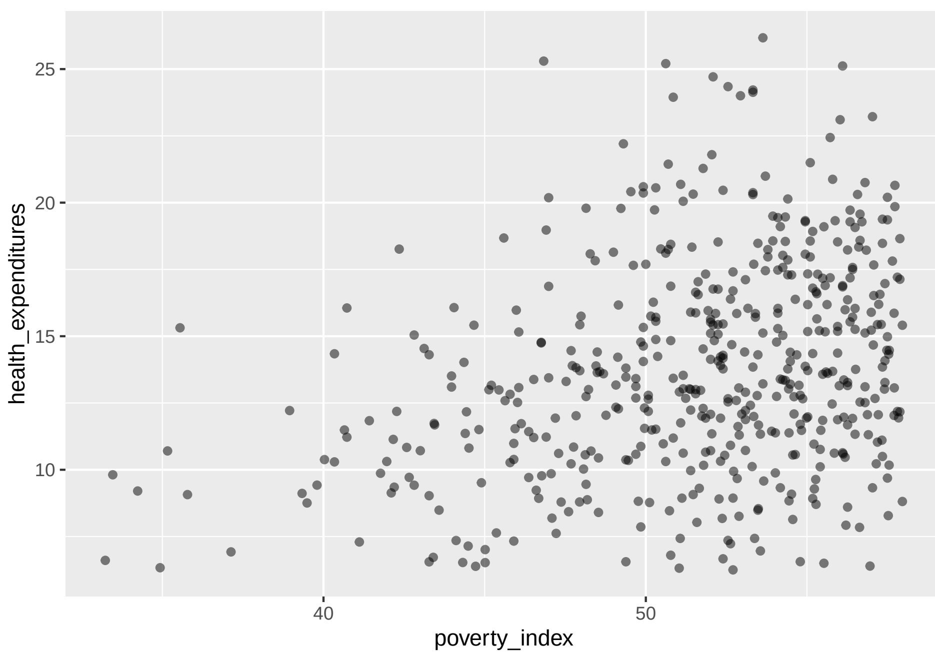
```
fch4_p1%>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures)) +
  geom_point()
```



This is more visible; There are some points that overlap with each other. Let us incorporate some strategies to try and ensure that there will be no overplotting issues.

The first strategy is changing the transparency of the points. To control the transparency of points, we add the `alpha` argument. The range of transparency is from 0 to 1, with lower values corresponding to more transparency. The default value is 1. Let's try to change the `alpha` to 0.5.

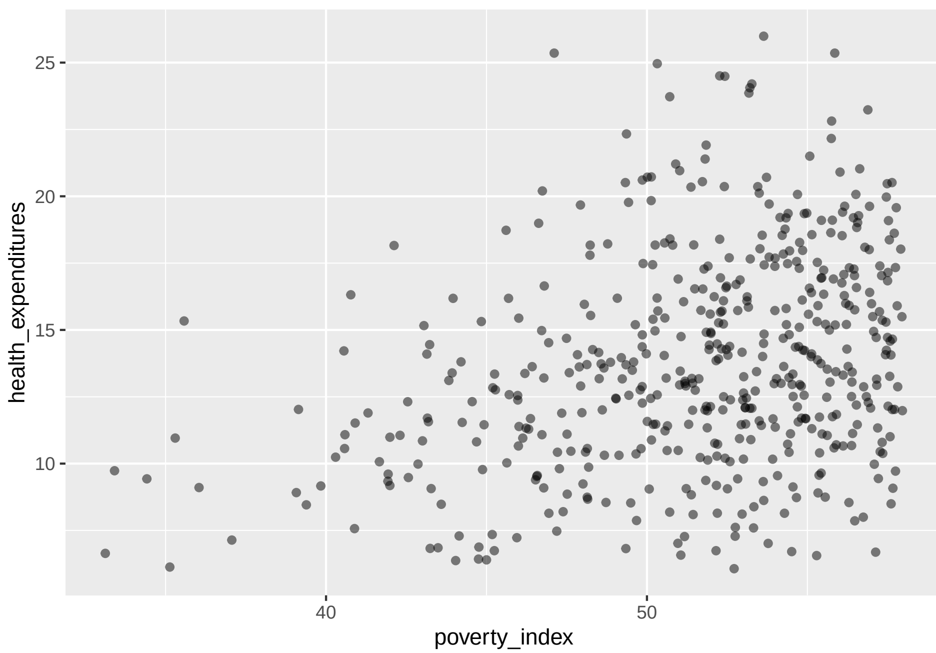
```
fch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures)) +
  geom_point(alpha=0.5)
```



Some of the points are gray while the others are much darker, then we can see (*slightly*) the difference.

Another method that we can do is jittering the points on the plot to see the locations where there are overplotting points. Jittering adds randomness into the position of the points. To do this, we add `geom_jitter()` rather than `geom_point()`. Also, we need to edit the `width` and `height`. You can experiment but if you want less spread, pick values between 0.1 and 0.4.

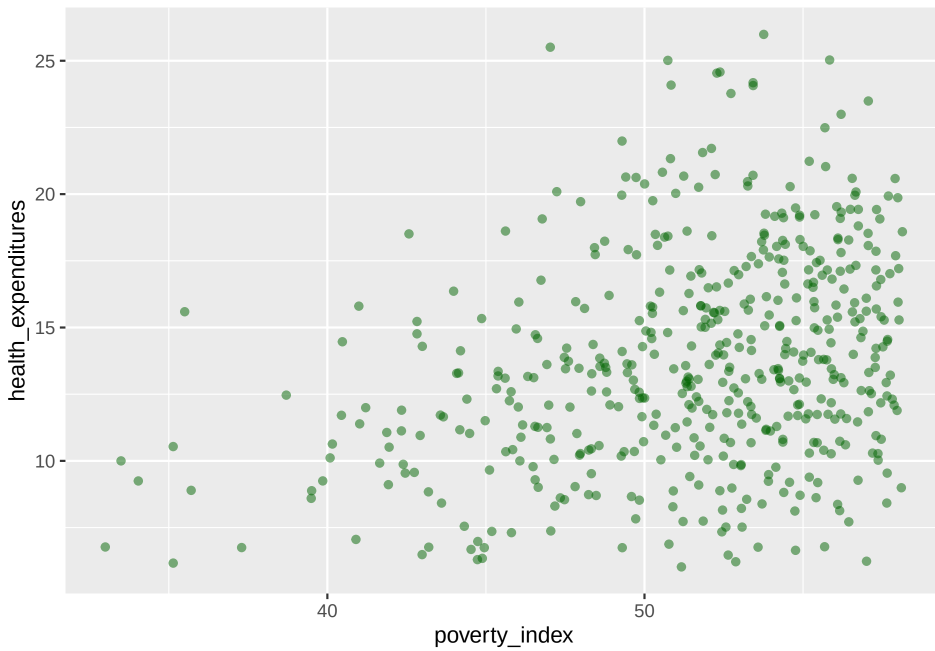
```
fch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures)) +
  geom_jitter(alpha = 0.5,
              width= 0.3,
              height= 0.3)
```



Let us add color to `geom_jitter()`

```
fch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures)) +
  geom_jitter(alpha = 0.5,
              color = "darkgreen",
```

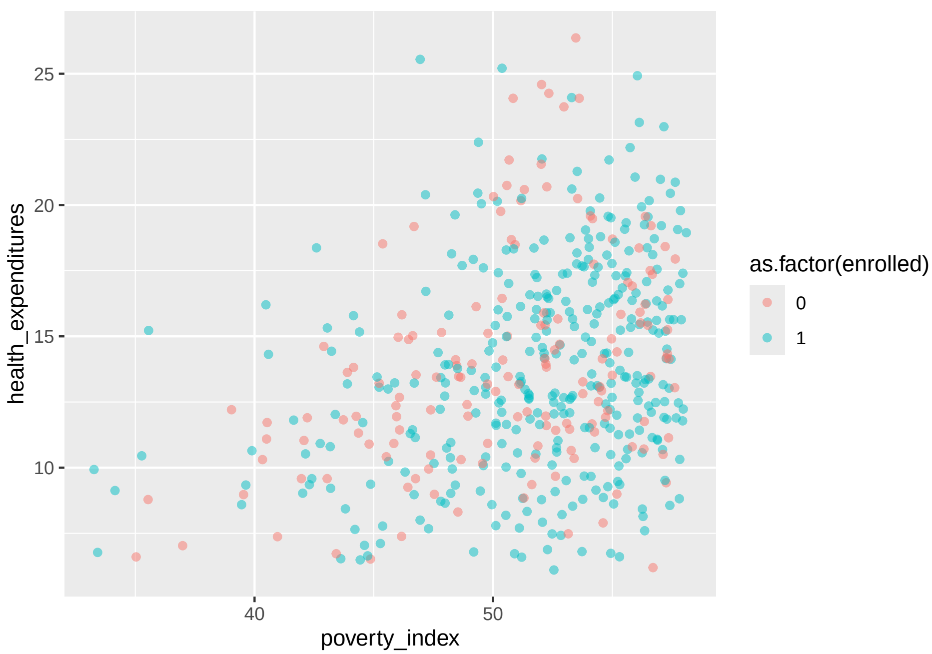
```
              width= 0.3,
              height= 0.3)
```



If you want to have different colors depending on a certain variable, we need to use a vector as an input in the argument `color`. Here though, we map features of the data to a certain color. When we map a variable in our data to the color of the points, ggplot2 will provide a different color corresponding to the different values of the variable. We will continue to specify the value of alpha, width, and height outside of the aes function because we are using the same value for every point.

```
fch4_p1 %>%
  ggplot(aes(x=poverty_index,
             y=health_expenditures))+
```

```
  geom_jitter(aes(color=as.factor(enrolled)), #this is because the enrolled is of num typ
              alpha=0.5,
              width=0.3,
              height=0.3)
```



11.2.2 Boxplots

Here is how to make a box plot that is useful when summarizing the distribution of a continuous variable, highlighting the median, quartiles, and potential outliers.

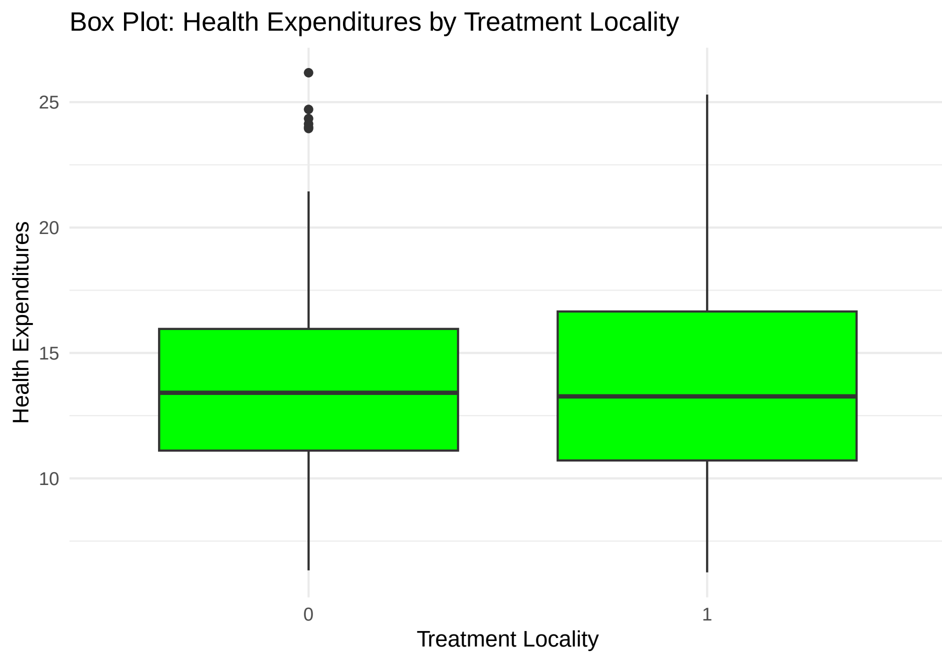
To interpret a boxplot, take note of the following things:

1. Horizontal Line in the Box: this is the median wherein it represents the 50% of the data.

2. The Box Itself: The box represents the middle 50% of the data, the bottom of the box represent first quartile so 25% of the data falls below this value, and the top of the box represent the third quartile so 75% of the data falls below this value.
3. Whiskers extend from the box to represent the range of the data excluding outliers so it usually end at the smallest and largest data points.
4. Outliers are points outside the whiskers.

*The `labs` argument is to add labels in the plot.

```
fch4_p1 %>%
ggplot(aes(x = as.factor(treatment_locality), y = health_expenditures)) +
  geom_boxplot(fill = "green") +
  labs(title = "Box Plot: Health Expenditures by Treatment Locality",
       x = "Treatment Locality",
       y = "Health Expenditures") +
  theme_minimal()
```



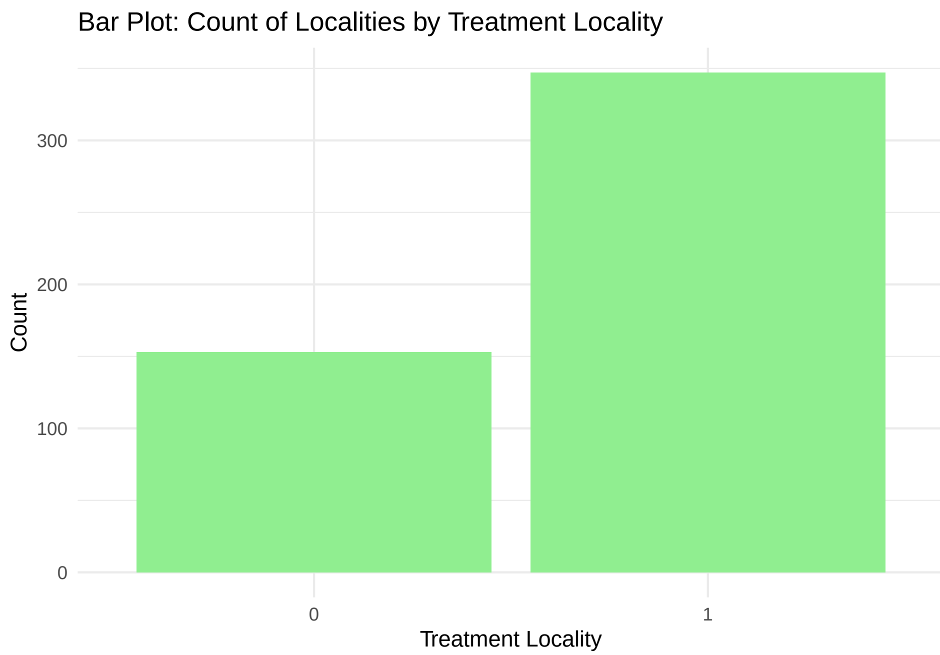
You can also add some points in the box plot by adding `geom_jitter`

```
fch4_p1 %>%
ggplot(aes(x = as.factor(treatment_locality), y = health_expenditures)) +
  geom_boxplot(fill="green") +
geom_jitter(alpha = 0.5,
color = "tomato",
width = 0.2,
height = 0.2) +
  labs(title = "Box Plot: Health Expenditures by Treatment Locality",
       x = "Treatment Locality",
       y = "Health Expenditures") +
  theme_minimal()
```

11.2.3 Bar plots

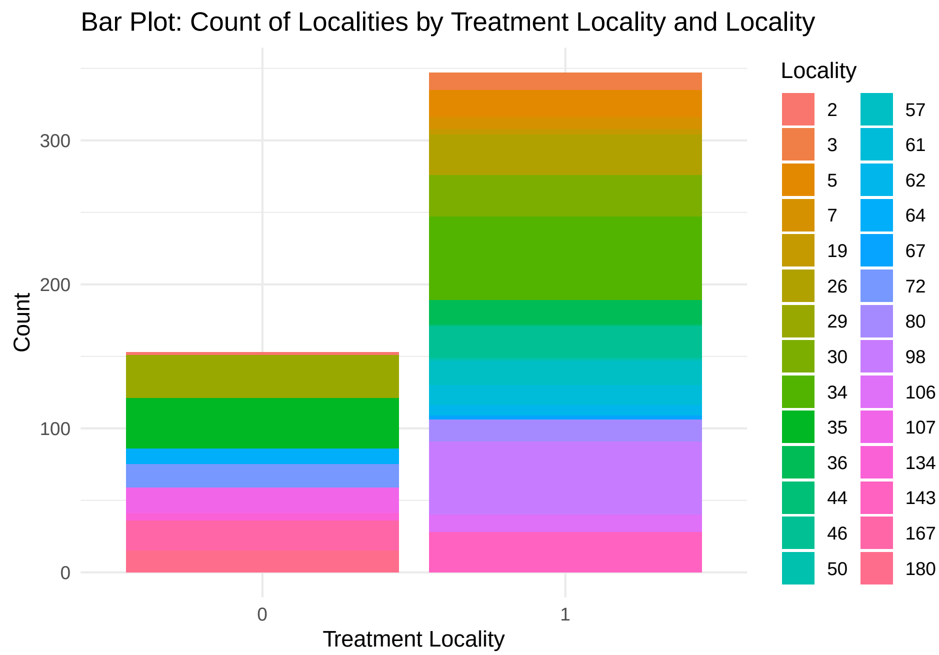
Barplots are also useful for visualizing categorical data. By default, `geom_bar` accepts a variable for x, and plots the number of instances each value of x (in this case, treatment_locality) appears in the dataset.

```
fch4_p1 %>%
ggplot(aes(x = as.factor(treatment_locality))) +
  geom_bar(fill = "lightgreen") +
  labs(title = "Bar Plot: Count of Localities by Treatment Locality",
       x = "Treatment Locality",
       y = "Count") +
  theme_minimal()
```



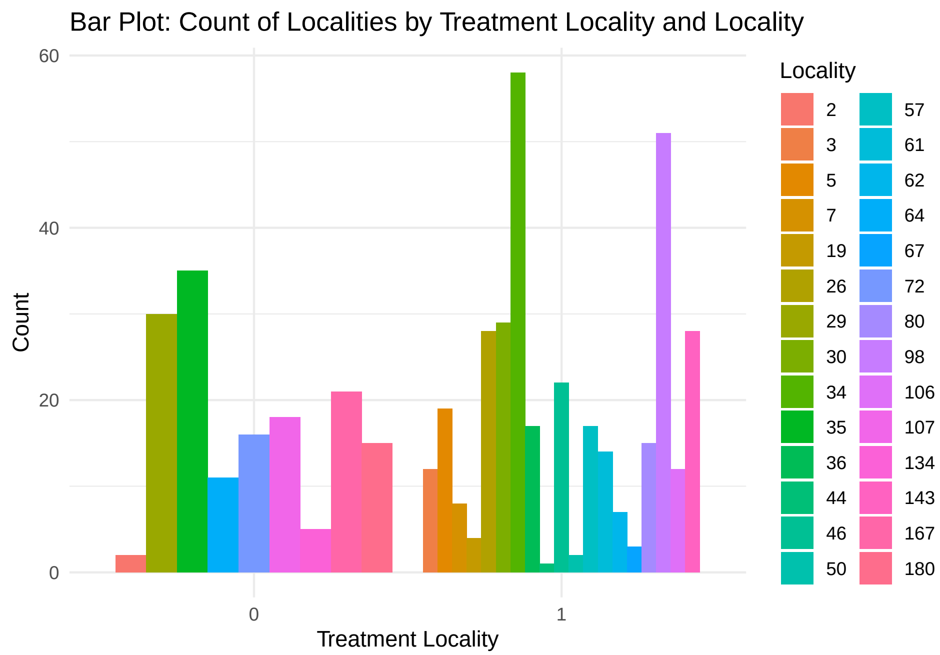
Let us change the fill to be `locality_identifier`. Note, we will have a lot of colors here but this is done for visualization purposes and practice.

```
fch4_p1 %>%
  ggplot(aes(x = as.factor(treatment_locality), fill = as.factor(locality_identifier))) +
  geom_bar() +
  labs(title = "Bar Plot: Count of Localities by Treatment Locality and Locality",
       x = "Treatment Locality",
       y = "Count",
       fill = "Locality") +
  theme_minimal()
```



This creates a stacked bar chart. These are generally more difficult to read than side-by-side bars. We can separate the portions of the stacked bar that correspond to each village and put them side-by-side by using the position argument for `geom_bar()` and setting it to "dodge".

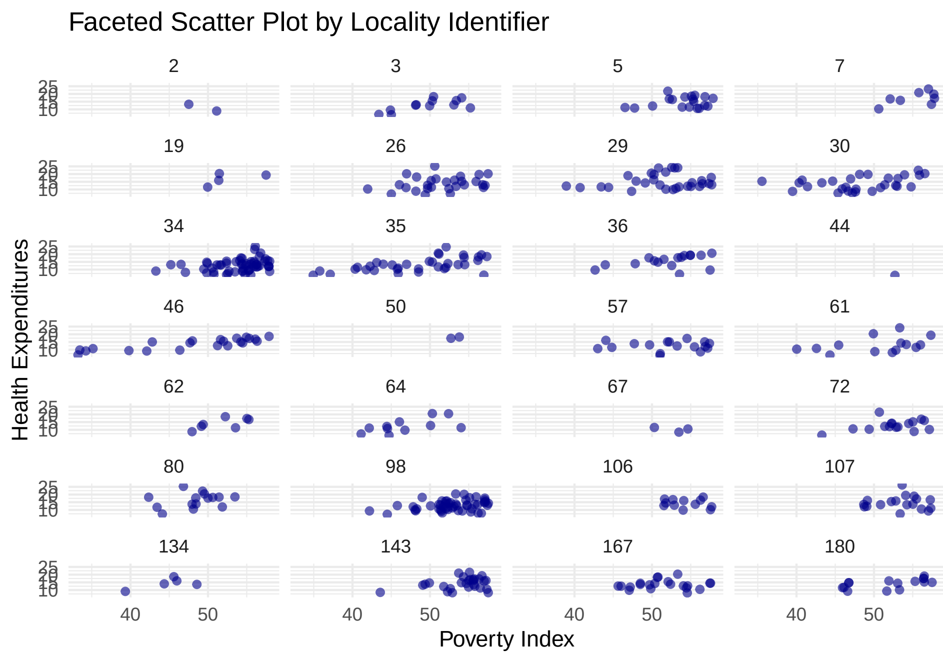
```
fch4_p1 %>%
  ggplot(aes(x = as.factor(treatment_locality), fill = as.factor(locality_identi
  geom_bar(position = "dodge") +
  labs(title = "Bar Plot: Count of Localities by Treatment Locality and Locality"
       x = "Treatment Locality",
       y = "Count",
       fill = "Locality") +
  theme_minimal()
```



11.2.4 Faceting

ggplot2 has a special technique called faceting that allows the user to split one plot into multiple plots based on a factor included in the dataset.

```
fch4_p1 %>%
ggplot(aes(x = poverty_index, y = health_expenditures)) +
  geom_point(alpha = 0.6, color = "darkblue") +
  facet_wrap(~ locality_identifier, ncol = 4) +
  labs(title = "Faceted Scatter Plot by Locality Identifier",
       x = "Poverty Index",
       y = "Health Expenditures") +
  theme_minimal()
```

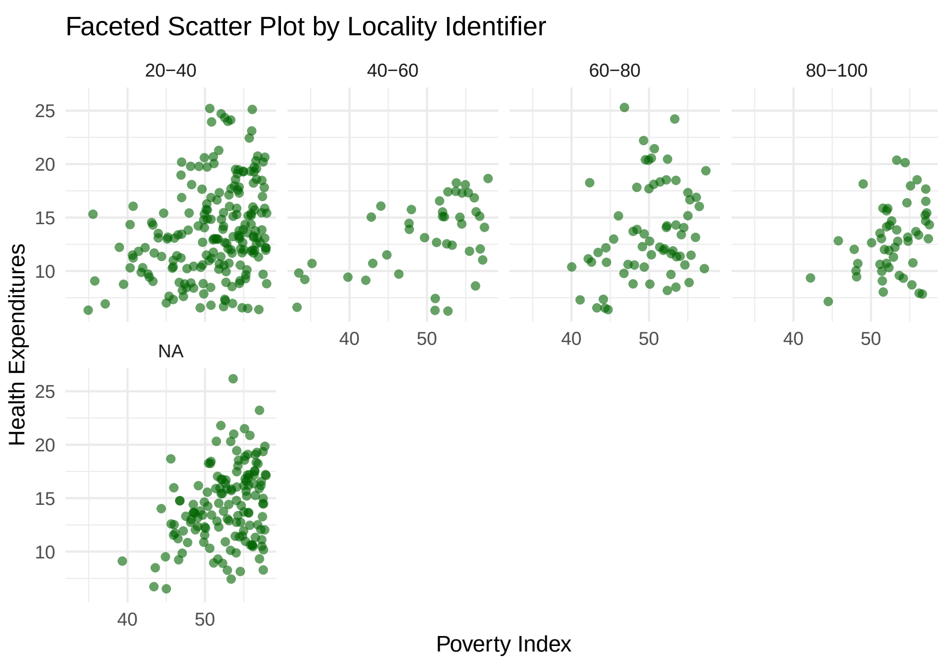


It doesn't look that nice; let's group the locality_identifier to make it more visually appealing:

```
fch4_p1 <- fch4_p1 %>%
  mutate(locality_group = cut(locality_identifier,
                              breaks = c(20, 40, 60, 80, 100),
                              labels = c("20-40", "40-60", "60-80", "80-100"),
                              include.lowest = TRUE))

table(fch4_p1$locality_group)
```

```
##
##  20-40  40-60  60-80 80-100
##    197     42     66     51
```

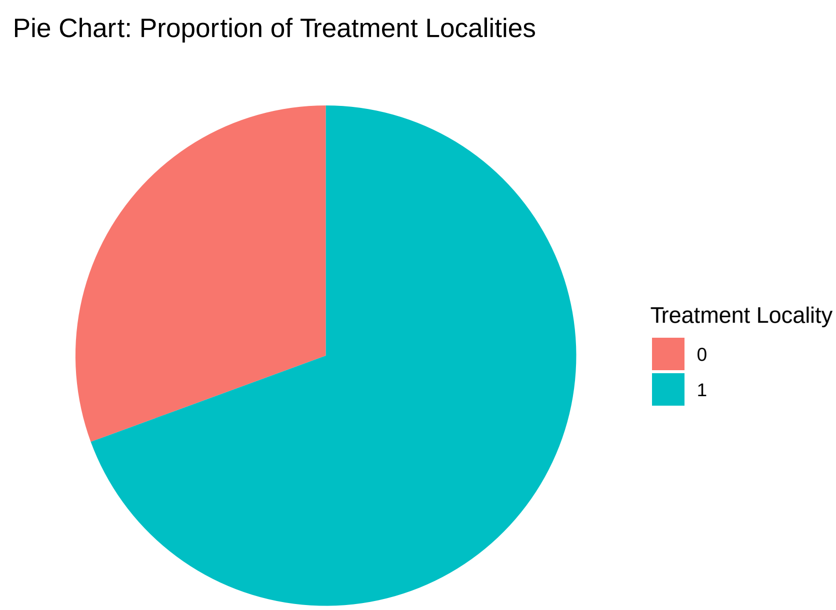
```
fch4_p1 %>%
ggplot(aes(x = poverty_index, y = health_expenditures)) +
  geom_point(alpha = 0.6, color = "darkgreen") +
  facet_wrap(~ locality_group, ncol = 4) +
  labs(title = "Faceted Scatter Plot by Locality Identifier",
       x = "Poverty Index",
       y = "Health Expenditures") +
  theme_minimal()
```



11.2.5 Pie Chart

Pie charts are used to illustrate proportions or parts of a whole (limited to a small number of categories). Before we do the pie chart, we need to count how many observations there are in the variable we want to analyze.

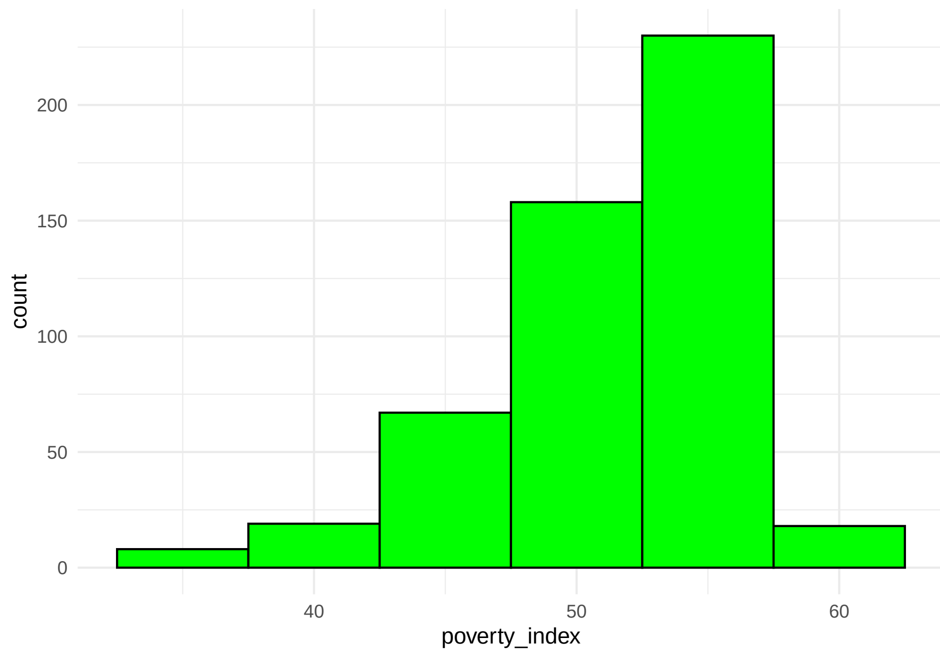
```
treatment_counts <- fch4_p1 %>%
  count(treatment_locality)

ggplot(treatment_counts, aes(x = "", y = n, fill = as.factor(treatment_locality))
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") + #this makes it a pie chart
  labs(title = "Pie Chart: Proportion of Treatment Localities",
       fill = "Treatment Locality") +
  theme_void() #another way to have a nice background
```



11.2.6 Histogram

Though it looks similar to a bar plot, a histogram is different since it displays the distribution of a continuous variable. It groups the data into intervals called bins and shows the frequency of data points within each bin.

```
fch4_p1 %>%
  ggplot(aes(x=poverty_index))+
  geom_histogram(binwidth = 5, fill="green", color="black")+
  theme_minimal()
```



Here is a cheat sheet for ggplot2 from the ones who developed the package: [ggplot2 Cheat Sheet]

11.3 Visualizing Time Series Data

When visualizing time series data, it is important to ensure that the time variable is formatted as Date. For this portion of the lecture, we use

`Ch4PracticeB.xlsx` which is found in the Modules. Make sure to clean the environment, load the file and rename the columns since they are quite long. I will not show the codes for this portion anymore as I am sure you already know how.

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2288498 122.3    3701069 197.7  3701069 197.7
## Vcells 6395985  48.8   12255594  93.6 12255594  93.6
```

```
## # A tibble: 6 x 12
##   year  nominal_gdp_current nominal_gdp_constant gdp_growth_current gdp_growtl
##   <chr>               <dbl>                <dbl>              <dbl>
## 1 1986               692852              4298952              0.065
## 2 1987               777283              4486464              0.122
## 3 1988               910280              4786920              0.171
## 4 1989              1054529              5082939              0.158
## 5 1990              1227882              5239629              0.164
## 6 1991              1422958              5216764              0.159
## # i 5 more variables: interest_payments <dbl>, amortization_payments <dbl>, pl
## #   inflation <dbl>, fdi_net <dbl>
```

1. Ensure the time variable is formatted as `Date`

   I will not show the code for this since this has been done in previous lectures

11.3.1 Template

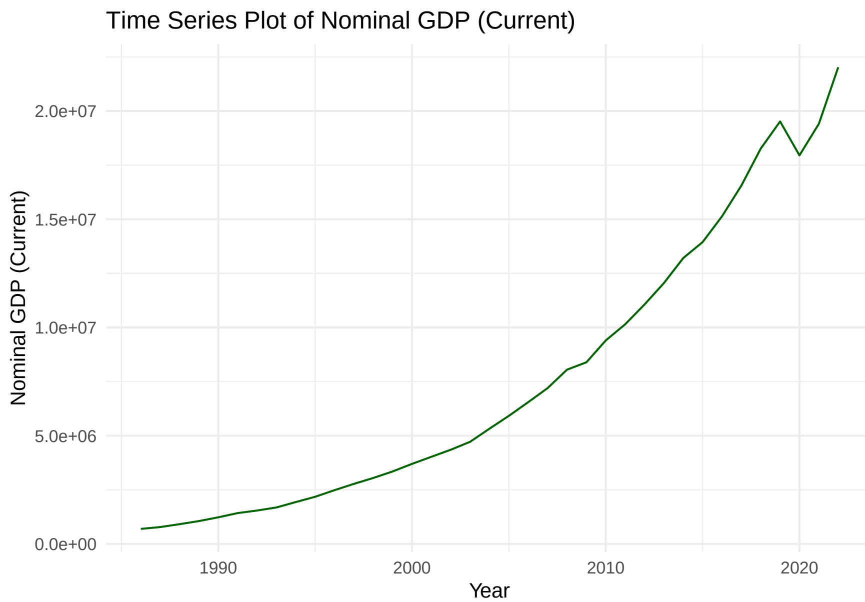
To make a time series visualization, this is the template:

```
ggplot(data, aes(x = time_variable, y = value_variable)) +
  geom_line(color = "blue") +   #adds a trend line
  labs(title = "Time Series Plot", x = "Time", y = "Value") +
  theme_minimal()
```

11.3.2 Time Series Plot

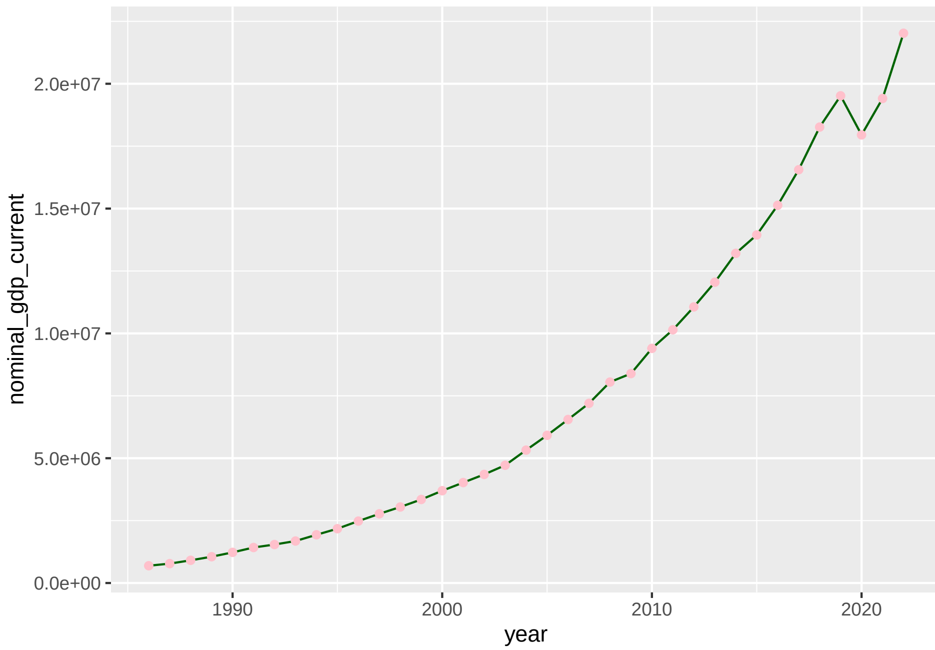
You might wonder why mine does not have a warning. Usually, there will be warnings that will come out. To remove it, simply add inside the `{r}`, `warning=FALSE` like: `{r,warning=FALSE}`

```
ch4_p2 %>%
  ggplot(aes(x=year, y=nominal_gdp_current))+
  geom_line(color="darkgreen")+
  labs(title = "Time Series Plot of Nominal GDP (Current)", x = "Year", y = "Nominal
  theme_minimal()
```



We can add points for clarity.

```
ch4_p2 %>%
  ggplot(aes(x=year, y=nominal_gdp_current))+
  geom_line(color="darkgreen")+
  geom_point(color= "pink")
```




```
labs(title = "Time Series Plot of Nominal GDP (Current)", x = "Year", y = "Nominal GDP
```

```
## $x
## [1] "Year"
##
## $y
## [1] "Nominal GDP (Current)"
##
## $title
## [1] "Time Series Plot of Nominal GDP (Current)"
##
## attr(,"class")
## [1] "labels"
```

Chapter 12

Practical: Visualizations

For this practical, we will use the `quotas` dataset. Please read the quotas_codebook to understand what each column means. Both files are found in the modules.

1. Inspect the dataset by using `str()` and `summary()` and describe the types of variables available.
2. Plot a histogram of the total population (`tot_pop71_true`). Adjust the number of bins to 15. What does this tell you about the distribution of population sizes in 1971?
3. Plot a bar plot of the count of Assembly Constituencies (`AC_type_noST`) based on reservation status. Which category has the highest count?
4. Plot the literacy rates (`Plit71`) against employment rates (`P_W71`). Add a regression line using `geom_smooth()`. What relationship do you observe?
5. Add color to the scatter plot by using color to distinguish between different reservation statuses (`AC_type_noST`). How do the patterns

differ across groups?

6. Create a box plot of literacy rates (`Plit71`) for different reservation statuses (`AC_type_noST`). What do you notice about the spread of literacy rates across categories?
7. Create a faceted scatter plot of literacy rates (`Plit71`) vs. employment rates (`P_W71`), grouped by reservation status (`AC_type_noST`). What differences can you identify between the facets?
8. Create a pie chart showing the proportion of constituencies by reservation status (`AC_type_noST`). What does this chart reveal about the dataset?
9. What did you find most challenging or rewarding about working with this dataset? Which visualization techniques did you find most useful for communicating your insights?
10. For the time series plot, I want you to search the **Literacy rate, adult total (% of people ages 15 and above)**from the World Bank. I want you to download as CSV the data. Then, I want you to choose India only. Create a time series plot for India.