A Portable Workbook for Data Analysis: R for the Social Sciences

A PORTABLE WORKBOOK FOR DATA ANALYSIS: R FOR THE SOCIAL SCIENCES

RENEE ZAHNOW AND JINGYEONG SONG

The University of Queensland
Brisbane, Queensland

CONTENTS

First published in 2025 by The University of Queensland

DOI: https://10.14264/8e93b4f

eISBN: 978-1-74272-480-5

Please cite as: Zahnow, R. (2025). *A portable workbook for data analysis: R for the social sciences*. The University of Queensland. https://10.14264/8e93b4f

Cover image photos:

Adobe Stock image – "Dynamic workspace featuring financial graphs and a laptop, illustrating data analysis and business strategy concepts clearly." © venusvi – stock.adobe.com #1651355760

Adobe Stock image – "Laptop screen displays colorful website traffic data. Charts, graphs show blog ranking, visitor statistics. Website optimization analysis in progress. Colorful data visualizations represent online." © miss irine – stock.adobe.com #1204577158

Adobe Stock image – "organization chart team concept networking business." © vegefox.com – stock.adobe.com #415150038

Adobe Stock image – "global of ai robotic, signal of data, technology abstract futuristic cyber net web connect to network background illustration 3d rendering, atom molecule of science, social internet of things." © issaronow – stock.adobe.com #187635811

The University of Queensland, St Lucia QLD, Australia

ACKNOWLEDGMENT OF COUNTRY

We acknowledge the Traditional Owners and their custodianship of the lands on which this project originated. We pay our respects to their Ancestors and their descendants, who continue cultural and spiritual connections to Country. We recognise their valuable contributions to Australian and global society.

A Guidance Through Time by Casey Coolwell and Kyra Mancktelow © The University of Queensland

About the artwork

Quandamooka artists Casey Coolwell and Kyra Mancktelow have produced an artwork that recognises the three major campuses, while also championing the creation of a strong sense of belonging and truth-telling about Aboriginal and Torres Strait Islander histories, and ongoing connections with Country, knowledges, culture and kin. Although created as a single artwork, the piece can be read in three sections, starting with the blue/greys of the Herston campus, the purple of St Lucia and the orange/golds of Gatton.

The graphic elements overlaying the coloured background symbolise the five UQ values:

- The Brisbane River and its patterns represent our Pursuit of excellence. Within the River are tools used by Aboriginal people to teach, gather, hunt, and protect.
- Creativity and independent thinking is depicted through the spirit guardian, Jarjum (Child in Yugambeh language), and the kangaroo
- The jacaranda tree, bora ring, animal prints, footprints and stars collectively represent honesty and accountability, mutual respect and diversity and supporting our people.

Learn more about The University of Queensland's Reconciliation Action Plan.

PREFACE

This online book is designed to support the practice of statistics among social scientists. This is not a statistical textbook. It is a practical workbook that should be used to accompany studies in Social Science methodologies, research best practice and statistical training.

This workbook specifically focuses on programming in R. Chapter 1 provides foundational knowledge about R and its basic functionalities. Subsequent chapters build on these foundations, integrating programming skills.

Each chapter guides readers through a specific task commonly encountered in empirical research within the social sciences, providing analytical steps and associated R syntax. Exercises are included in each chapter to provide opportunities review statistical knowledge and enhance practical skills using real data.

As the field of social sciences evolves, learning at least one programming language like R can significantly enhance research capabilities. The demand for skilled data science practitioners in academia, industry, and government is growing rapidly, making R a valuable skill for any researcher to have. This book provides foundational knowledge that will introduce you to empirical analysis through the use of quantitative statistical software.

1.

INTRODUCTION TO R AND RSTUDIO

1. What is R?

R is a programming language and environment specifically designed for statistical computing and graphical data representation. R supports a extensive suite of statistical techniques including linear and nonlinear regression, time-series and spatial analyses and large language modelling. R can also be used to visualise data using advanced graphical techniques. This feature makes the R software useful for communicating complex statistical results in visual formats and enhances the utility of R across disciplinary and industry domains. A significant strength of R is its ability to produce publication-quality plots and even maps with ease. R is distributed as free software under the terms of the GNU General Public License provided by the Free Software Foundation. It is available in source code form and compatible with various operating systems, including Windows, macOS, and Linux.

R is not only a statistical data analysis tool but also a programming language in its own right. Like any language, R has its own vocabulary, syntax, and structure. Just as learning a foreign language requires mastering fundamental grammar and sentence patterns, effectively using R requires understanding its basic syntax and expressions.

The goal of the first chapter is to guide you through downloading and installing R and RStudio while familiarising yourself with R's structure and basic functions. In the first two chapters, we will focus on learning the fundamental elements of R. Building on this foundation, the subsequent chapters will provide an opportunity for you to develop your R skills by practicing more advanced statistical techniques.

2. Downloading R and RStudio

The first step is to set everything up to get started. There are two ways to run R:

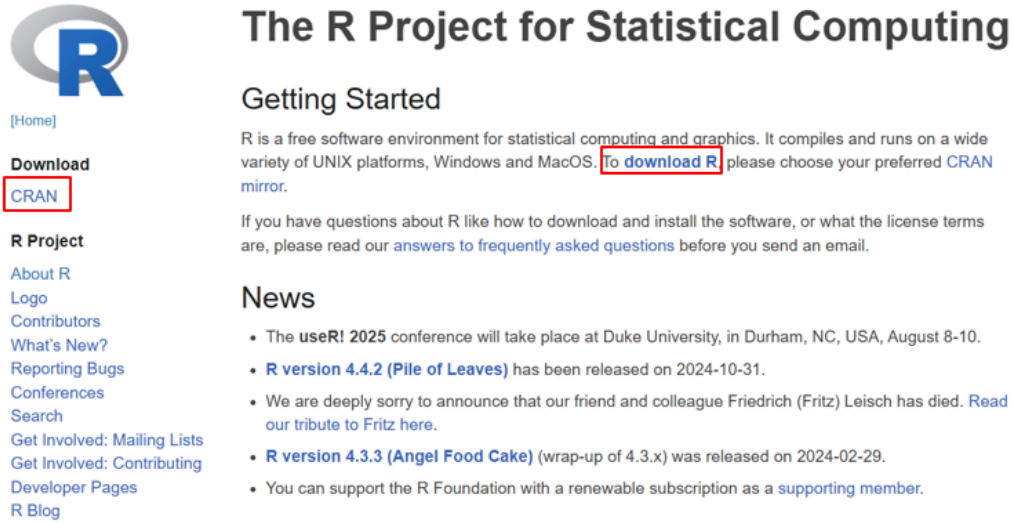
- Install R and RStudio, or
- Run RStudio in a browser via RStudio Cloud

Since we will be using R as we work through the exercises in this book, lets install R and RStudio. Once installed, R becomes much easier to execute both online and offline.

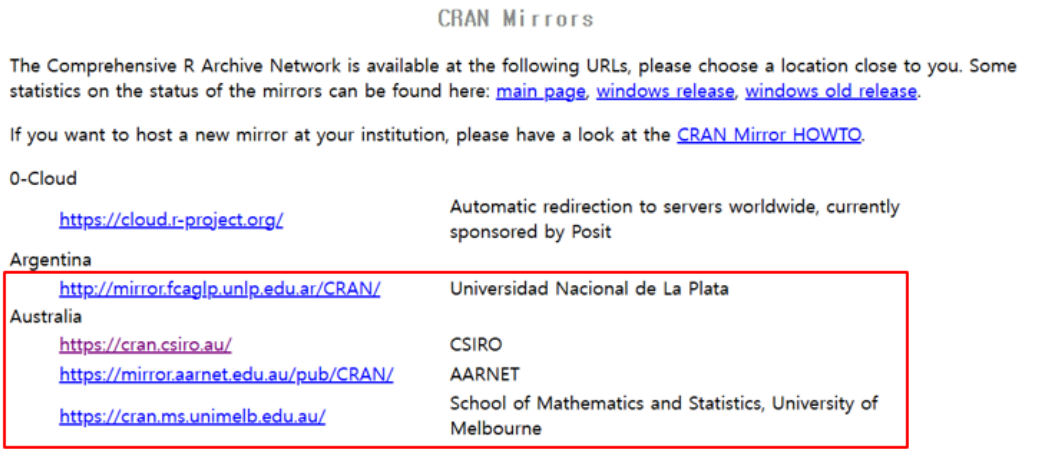
2.1 Installing R

The core module of our programming is R itself. As an open-source project, it is available for free on Windows, Mac, and Linux computers. Here's what you need to do to install it properly on your computer:

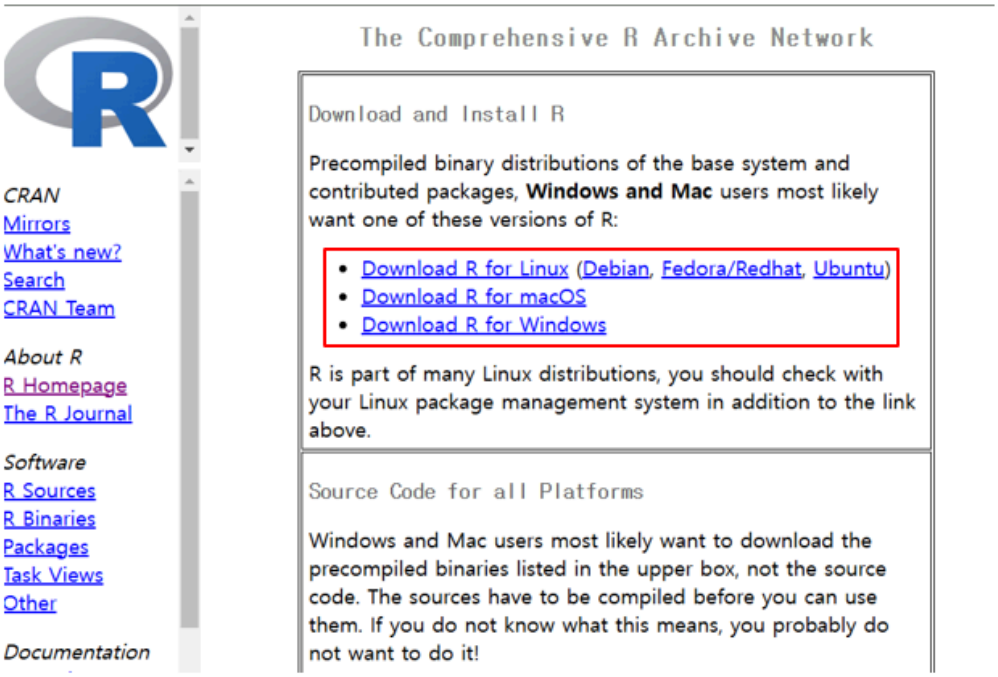
1. Go to https://www.r-project.org/
2. Click on 'CRAN' under the Download or select 'download R' directly.



3. Select a server in your country (any server will work but choosing one closer to your location improves performance).



4. Select your computer's operating system, such as "Download R for Windows."



Open the downloaded file and follow the installation instructions. It is recommended to keep the default settings. After completing the steps successfully, R is now installed on your system. While it is possible to operate R from inside the software itself, Rstudio provides a more user-friendly interface, especially for those new to the software. In this course we will use RStudio IDE (Integrated Development Environment). It is a comprehensive toolset designed to enhance productivity within R and includes a console, syntax-highlighting editor for direct code execution, and tools for plotting, viewing history, and managing your workspace.

2.2 Installing RStudio

Before installing RStudio, ensure that R is already installed on your computer.

1. Go to https://posit.co/download/rstudio-desktop/, scroll down, and select the right operating system (OS) for your computer (Windows users: choose windows, Mac users: choose macOS).

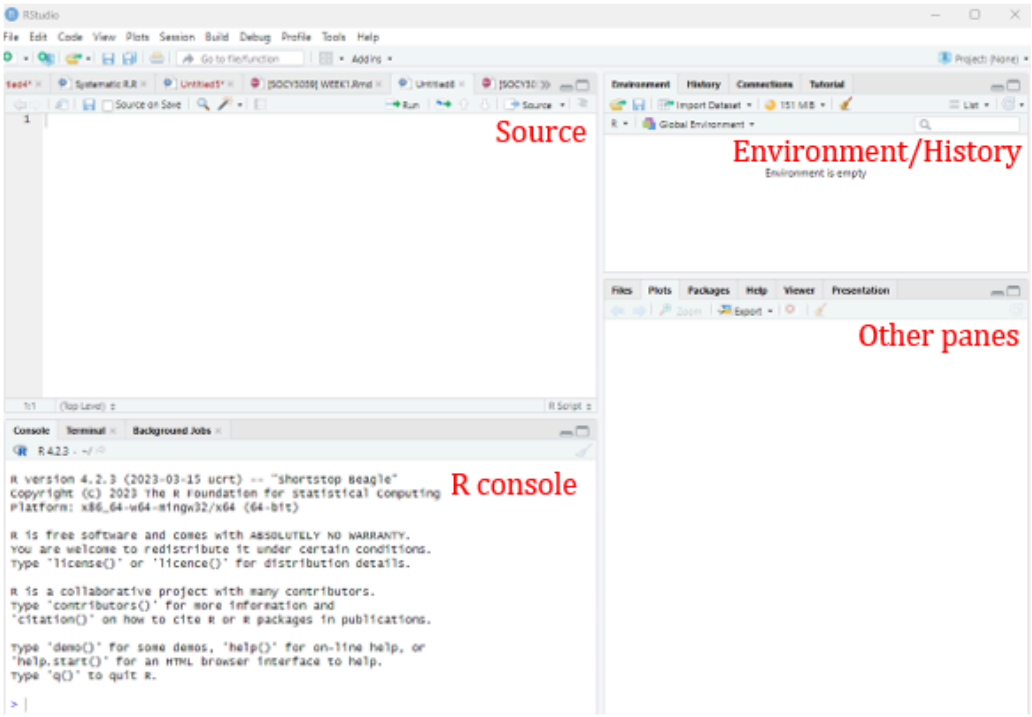
OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2024.12.0-467.EXE	265.27 MB	5EFCD188
macOS 13+	RSTUDIO-2024.12.0-467.DMG	617.71 MB	46958FB4
Ubuntu 20/Debian 11	RSTUDIO-2024.12.0-467-AMD64.DEB	203.15 MB	109B35F9
Ubuntu 22/Debian 12	RSTUDIO-2024.12.0-467-AMD64.DEB	203.15 MB	4A3872FB

2. Open the downloaded file and follow the installation instructions, keeping the default settings wherever possible.

Congratulations, you are all set to start learning R! From now on, you only need to open **RStudio** instead of R. It is highly recommended that you create a folder (e.g., MY_R) on your computer to save all files and outputs.

3. Downloading R and RStudio

Please open RStudio, and you will see a screen with four windows, as shown below:



3.1 The Source Window (top left)

In this space, you will see all your files listed. This will include all file types including data and code that you have prepared. The source panel can is where you will open, edit and run your programming code (R script). Once open in the source window, R scripts can be edited and/or executed. **You should write and run all your code in the Source window.**

3.2 The R Console Window (bottom left)

After executing your code, the output of your computations will appear in the R console window. While you can execute code directly in the console, this will not save your code history. Therefore, it is highly recommended that you execute all your code from the *source* window.

3.3 The Environment/History/Connections/Tutorial Window (top right)

This panel has several tabs:

- The *Environment* panel displays the workspace, allowing users to view and manage objects like variables, data frames, and functions currently loaded into memory.
- The *History* panel keeps a record of previously executed commands, making it easy to review, reuse, or modify code for reproducibility.
- The *Connections* panel enables seamless interaction with external databases, providing tools to establish and manage connections for efficient data import and analysis.
- The *Tutorial* panel offers built-in learning resources, giving users access to guided tutorials and interactive lessons within the RStudio interface.

For our purposes, we will mainly focus on the Environment panel.

3.4 The Files/Plots/Packages/Help/Viewer Window (bottom right)

- The *Files* tab provides access to the file system, allowing users to navigate directories, open scripts, and manage files directly within RStudio.
- The *Plots* tab displays visual outputs, such as graphs and charts, generated during data analysis, with options to zoom, export, or clear plots.
- The *Packages* tab manages installed R packages, enabling users to load, update, or install additional libraries needed for their work.
- The *Help* tab serves as a comprehensive resource for accessing documentation and function references, making it easy to look up details about R functions or packages.

- The *Viewer* tab is designed for rendering web content, such as interactive HTML reports or visualizations, directly within RStudio.

Together, these tools provide an integrated interface for efficient project management and data analysis.

4. R Basics

Before diving into R packages or running code with R functions, we will begin with a warm-up exercise by performing simple calculations like addition, subtraction, multiplication, and division. We will also cover the essential basics, including how to assign values to an object using <-.

4.1 Basic Computation

Basic computation in R is very intuitive and can be similar to using a calculator or mobile phone. Let's take a look at the following examples:

```
# Addition
10 + 10
```

```
## [1] 20
```

```
# Subtraction
10 - 10
```

```
## [1] 0
```

```
# Multiplication
8 * 15
# * operator is used for Multiplication in R
```

```
## [1] 120
```

```
# Division
40 / 4
```

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

```
# Exponentiation
14 ^ 2
```

```
## [1] 196
```

Q. Now, use R to practice computing the following exercises and write your answers:

```
# Exercise 1
40 + (4 + 7)^2
```

```
# Exercise 2
100 - 4*(3 + 3)^2/2
```

4.2 Assigning Values to Objects <-

An object is a named entity that stores data. In R, assigning values to an object means storing a value (such as a number, vector, or dataset) in a named variable using the assignment operator <-. Objects can be different data types (numeric values, text):

1. Numeric Values

```
x <- 10
 x
```

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

2. Character Strings

```
course <- "SOCY3039"
course
```

[1] “SOCY3039”

3. Vectors

```
numbers <- c(1, 2, 3, 4, 5)
 numbers
```

```
## [1] 1 2 3 4 5
```

4. Data Frames

```
df <- data.frame(Num.students = c(180, 55), Courses = c("SOCY2339", "SOCY3039"))
df
```

Remember

The <- symbol is the assignment operator in R, used to assign values to objects. For example, in the first case, the operator <- assigns value 5 to the object 'x'. The symbol `c()` stands for combine and is used to create vectors by combining multiple elements of the same type into a single sequence.

We will revisit the concepts of Numeric values, Character strings, Vectors, and Data frames in more detail in the next chapter.

5. R Packages

R provides a wide range of built-in functions, but additional functions can be added through R packages. These packages, developed by programmers and data scientists, are collections of functions, datasets, and documentation that extend R's capabilities. Think of

- R as a clothing factory with basic materials like fabric, thread, and tools.
- RStudio is your sewing station, where you gather all the tools (functions) you need to create your designs.
- R packages are like additional fabrics, patterns, and accessories that you can bring into your workspace, each offering new features to help you craft more specialized pieces of clothing for your project.

Num.students	Courses
180	SOCY2339
55	SOCY3039

5.1 Installing Packages

There are two ways to install R packages:

- Use `install.packages()` function, or
- Use the packages tab on the bottom right window

Installing packages using the function is recommended as it is the quickest and simplest way to install an R package.

One of the most used packages is ggplot2, which generates a variety of visualisations. Let's install it as an example by entering the
following command:

```
install.packages("ggplot2")
```

You can also install multiple packages at once using the `c()` function, which groups items together.

For example, to install ggplot2, dplyr, and tidyverse, execute the following code:

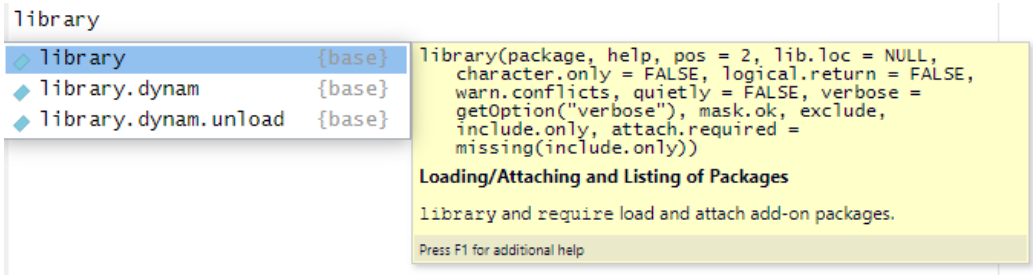
```
install.packages(c("ggplot2", "dplyr", "tidyverse"))
```

You can find additional information about a package or function by
placing a ? in front of the function name `(e.g., ?install.packages)`.

5.2 Loading Packages

Once your R packages are installed, the next step is to start using them. While you only need to install packages once, you will need to activate them each time you start a new session in RStudio. This process, known as **loading a package**, gives you access to all its functions. To load a package, simply use the `library()` function.

When you start typing a function like the 'library'in Rstudio, it will automatically suggest related functions and display a yellow box. This box shows the function's structure (also called its syntax or signature) along with a brief description of what the function does.



Screenshot of the R application. Available under a GNU General Public License.

For example, to activate ggplot2, we have already installed, run the following code:

```
library(ggplot2)
```

Unfortunately, you cannot use `c()` function to activate multiple packages at once like with `install.packages()`. You will need to activate each package individually. While there is a way to load multiple packages at once, it is beyond the scope of this book, but feel free to explore if you are curious.

2.

INTRODUCTION TO TIDYVERSE AND R MARKDOWN

This chapter focuses on learning and practicing two essential R packages, tidyverse and dplyr. These packages provide a range of functions for performing statistical analyses and serve as a fundamental building block for the code that will be used in later sessions. We then move on to learning how to use R Markdown.

1. Tidyverse Packages

The *tidyverse* is a collection of R packages designed for data science, including essential tools such as *dplyr* for data manipulation and *ggplot2* for data visualization. Loading the *tidyverse* with the `library()` function automatically makes these packages available.

Let's set up the working directory first.

```
#Set up the working directory
setwd("C:/Your folder path/SOCYR")
```

```
#Install & load the required packages for chapter 4
install.packages("tidyverse") #If you have not installed this package yet
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.0     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
```

```
## ✓ ggplot2   3.5.2     ✓ tibble    3.2.0
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.1
## ── Conflicts ─────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all con-
flicts to
become errors
```

As shown in the results, the *tidyverse* includes nine packages. Below is a brief overview of the key packages most relevant to this course.

Package	Description
ggplot2	**For data visualization:** ggplot allows users to create complex and customizable plots using a layered approach.
dplyr	**For data manipulation:** plyr provides functions for selecting, filtering, grouping, summarizing, and mutating data efficiently using a consistent and intuitive
readr	**For reading and writing data** (CSV, TSV, etc.)
tidyr	**For reshaping and tidying data:** The tidyr package provides functions to transform messy data into a structured format, making it easier to analyze by spreading, gathering, and separating columns.

You can find full descriptions of the remaining packages in the Tidyverse documentation.

2. dplyr Packages

The *dplyr* package is a widely used tool for data manipulation in R, offering efficient functions for tasks such as filtering, selecting, reshaping, and summarizing data. It simplifies data cleaning by requiring less code than *base R* while providing a more intuitive and readable syntax.

```
library(dplyr)
```

For this exercise, we will use data from the World Values Survey (WVS). Before performing any data manipulation, use the `glimpse()` function to explore the structure of the dataset.

```
# load the WVS dataset
```

```
load("WVS_Cross-National_Wave_7_Rdata_v6_0.rdata")
WVS <- `WVS_Cross-National_Wave_7_v6_0`
glimpse(WVS)  #The output  is similar to the 'str()' function
```

```
## Rows: 97,220
## Columns: 613
## $ version                   <chr> "6-0-0 (2024-04-30)", "6-0-0 (2024-04-30)",…
## $ doi                       <chr> "doi.org/10.14281/18241.24", "doi.org/10.14…
## $ A_WAVE                    <int> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7…
## $ A_YEAR                    <int> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2…
## $ A_STUDY                   <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ B_COUNTRY                 <int> 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,…
## $ B_COUNTRY_ALPHA           <chr> "AND", "AND", "AND", "AND", "AND", "AND", "…
## $ C_COW_NUM                 <int> 232, 232, 232, 232, 232, 232, 232, 232, 232…
## $ C_COW_ALPHA               <chr> "AND", "AND", "AND", "AND", "AND", "AND", "…
## $ D_INTERVIEW               <int> 20070001, 20070002, 20070003, 20070004, 200…
## $ S007                      <int> 20720001, 20720002, 20720003, 20720004, 207…
## $ J_INTDATE                 <int> 20180704, 20180714, 20180704, 20180702, 201…
## $ FW_START                  <int> 201807, 201807, 201807, 201807, 201807, 201…
## $ FW_END                    <int> 201809, 201809, 201809, 201809, 201809, 201…
## $ K_TIME_START              <dbl> 18.20, 9.35, 10.15, 17.05, 10.20, 12.15, 17…
## $ K_TIME_END                <dbl> 19.48, 11.00, 10.45, 18.20, 11.48, 13.00, 1…
## $ K_DURATION                <int> 88, 85, 30, 75, 89, 45, 44, 60, 74, 41, -5,…
## $ Q_MODE                    <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ N_REGION_ISO              <int> 20007, 20003, 20003, 20003, 20003, 20006, 2…
## $ N_REGION_WVS              <int> 20005, 20002, 20002, 20002, 20002, 20006, 2…
## $ N_REGION_NUTS2            <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ N_REG_NUTS1               <int> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3,…
## $ N_TOWN                    <int> 20005, 20002, 20002, 20002, 20002, 20006, 2…
## $ G_TOWNSIZE                <int> 5, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 2, 3, 3, 3…
## $ G_TOWNSIZE2               <int> 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2…
## $ H_SETTLEMENT              <int> 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ H_URBRURAL                <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ I_PSU                     <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ O1_LONGITUDE              <dbl> 1.52, 1.53, 1.58, 1.58, 1.58, 1.52, 1.49, 1…
## $ O2_LATITUDE               <dbl> 42.51, 42.51, 42.53, 42.53, 42.54, 42.50, 4…
## $ L_INTERVIEWER_NUMBER      <int> -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5,…
```

```
## $ S_INTLANGUAGE              <int> 810, 810, 810, 810, 810, 1270, 1270, 810, 8…
## $ LNGE_ISO                   <chr> "ca", "ca", "ca", "ca", "ca", "es", "es", "…
## $ E_RESPINT                  <int> 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ F_INTPRIVACY               <int> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1…
## $ E1_LITERACY                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ W_WEIGHT                   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ S018                       <dbl> 0.9960159, 0.9960159, 0.9960159, 0.9960159,…
## $ pwght                      <dbl> 0.006844622, 0.006844622, 0.006844622, 0.00…
## $ S025                       <int> 202018, 202018, 202018, 202018, 202018, 202…
## $ Q1                         <int> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1…
## $ Q2                         <int> 1, 1, 2, 1, 1, 3, 2, 1, 2, 1, 1, 1, 2, 1, 3…
## $ Q3                         <int> 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1…
## $ Q4                         <int> 3, 4, 2, 4, 3, 4, 4, 1, 1, 4, 4, 2, 3, 3, 4…
## $ Q5                         <int> 1, 1, 3, 2, 1, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1…
## $ Q6                         <int> 4, 4, 3, 4, 3, 3, 2, 3, 1, 2, 1, 2, 4, 3, 2…
## $ Q7                         <int> 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1…
## $ Q8                         <int> 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1…
## $ Q9                         <int> 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2…
## $ Q10                        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1…
## $ Q11                        <int> 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2…
## $ Q12                        <int> 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1…
## $ Q13                        <int> 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 2, 2…
## $ Q14                        <int> 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2…
## $ Q15                        <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2…
## $ Q16                        <int> 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1…
## $ Q17                        <int> 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2…
## $ Q18                        <int> 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1…
## $ Q19                        <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1…
## $ Q20                        <int> 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1…
## $ Q21                        <int> 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2…
## $ Q22                        <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q23                        <int> 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2…
## $ Q24                        <int> 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 2, 1…
## $ Q25                        <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q26                        <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q27                        <int> 3, 3, 3, 3, 3, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1…
## $ Q28                        <int> 3, 3, 3, 3, 3, 1, 1, 3, 3, 1, 1, 2, 1, 3, 1…
## $ Q29                        <int> 3, 3, 3, 4, 3, 2, 4, 4, 4, 3, 3, 3, 4, 4, 2…
```

```
## $ Q30                    <int> 3, 3, 3, 4, 3, 4, 4, 4, 4, 3, 3, 3, 4, 4, 4…
## $ Q31                    <int> 3, 3, 3, 4, 3, 4, 2, 4, 4, 3, 2, 3, 4, 4, 4…
## $ Q32                    <int> 3, 3, -2, 4, 2, 4, 1, 4, 3, 3, 1, 2, 2, 1, …
## $ Q33                    <int> 4, 4, 4, 4, 4, 5, 5, 5, 5, 4, 3, 4, 5, 5, 5…
## $ Q33_3                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2…
## $ Q34                    <int> 2, 2, 4, 2, 4, 5, 5, 1, 1, 4, 4, 4, 3, 5, 5…
## $ Q34_3                  <int> 1, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 3, 2, 2…
## $ Q35                    <int> 3, 4, 2, 4, 4, 5, 5, 1, 4, 5, 4, 4, 4, 3, 1…
## $ Q35_3                  <int> 3, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 3, 1…
## $ Q36                    <int> 3, 2, 4, 3, 3, 1, 1, 3, 3, 1, 3, -2, 1, 1, …
## $ Q37                    <int> 4, 4, 4, 4, 3, 1, 5, 5, 5, 3, 1, 2, 5, 5, 1…
## $ Q38                    <int> 3, 2, 3, 4, 3, 1, 3, 5, 3, 3, 1, 2, 2, 4, 1…
## $ Q39                    <int> 2, 2, 2, 2, 3, 1, 3, -2, 3, 1, 4, 2, 1, 3, …
## $ Q40                    <int> 3, 2, 4, 4, 3, 1, 1, 1, 2, 4, 1, 2, 1, 3, 1…
## $ Q41                    <int> 3, 2, 4, 4, 4, 1, 5, 3, 5, 5, 1, 4, 5, 4, 1…
## $ Q42                    <int> 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q43                    <int> 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 3, 2, 1, 3…
## $ Q44                    <int> 3, 1, 2, 1, 1, 3, 2, 1, 2, 1, 2, 1, 3, 2, 2…
## $ Q45                    <int> 1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1…
## $ Q46                    <int> 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 3…
## $ Q47                    <int> 3, 1, 1, 2, 2, 1, 1, 2, 2, 1, 3, 1, 1, 2, 2…
## $ Q48                    <int> 10, 9, 9, 9, 8, 10, 10, 8, 8, 10, 9, 8, 10,…
## $ Q49                    <int> 10, 9, 9, 8, 7, 10, 5, 8, 8, 10, 8, 8, 10, …
## $ Q50                    <int> 5, 9, 8, 6, 7, 6, 4, 9, 6, 10, 7, 8, 10, 5,…
## $ Q51                    <int> 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 3…
## $ Q52                    <int> 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 3…
## $ Q53                    <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2…
## $ Q54                    <int> 4, 4, 3, 4, 4, 4, 2, 4, 3, 4, 4, 4, 4, 4, 2…
## $ Q55                    <int> 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 4, 3…
## $ Q56                    <int> 1, 1, 2, 3, -2, 3, 1, -2, 3, 2, 2, 3, 1, 3,…
## $ Q57                    <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2…
## $ Q58                    <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1…
## $ Q59                    <int> 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 3…
## $ Q60                    <int> 2, 2, 2, 3, 2, 2, 2, 3, 2, 2, 2, 2, 1, 1, 1…
## $ Q61                    <int> 2, 3, 4, 3, 3, 2, 4, 3, 3, 3, 4, 2, 4, 2, 2…
## $ Q62                    <int> 2, 2, 3, 3, 3, 2, 3, 3, 2, 1, -2, 2, 2, 1, …
## $ Q63                    <int> 2, 2, 3, 3, 3, 2, 3, 3, 2, 1, 2, 2, 2, 1, 2…
## $ Q64                    <int> 1, 4, 2, 3, 3, 3, 3, 4, 4, 1, 4, 2, 3, 2, 4…
```

```
## $ Q65                    <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q66                    <int> 1, 3, 4, 3, 3, 3, 3, 4, 3, 2, 1, 2, 3, 3, 3…
## $ Q67                    <int> 1, 3, 4, 3, 3, 3, 3, 4, 3, 3, 1, 2, 3, 3, 1…
## $ Q68                    <int> 1, 4, 3, 3, 2, 4, 4, 4, 3, 4, -2, 2, 3, 3, …
## $ Q69                    <int> 1, 3, 2, 3, 2, 1, 1, 3, 2, 1, 1, 2, 1, 1, 1…
## $ Q70                    <int> 1, 3, 2, 3, 2, 1, 4, 3, 3, 3, 2, 2, 1, 1, 3…
## $ Q71                    <int> 1, 4, 3, 3, 2, 2, -2, 3, 3, 2, 2, 3, 2, 1, …
## $ Q72                    <int> 1, 4, 3, 3, 3, 2, 3, 4, 3, 4, 4, 2, 4, 3, 2…
## $ Q73                    <int> 1, 4, 3, 3, 2, 2, 4, 4, 3, 4, 2, 2, 4, 1, 2…
## $ Q74                    <int> 1, 3, 3, 3, 3, 2, 4, 4, 3, 3, 2, 2, 2, 1, 3…
## $ Q75                    <int> 1, 2, 2, -2, -2, 1, 2, 3, 1, 2, 1, 2, 2, 2,…
## $ Q76                    <int> 1, 3, 3, 3, 3, 2, -2, 4, 3, 3, 2, 2, -2, 2,…
## $ Q77                    <int> 2, 3, 3, 3, 2, 1, 4, 4, 3, 2, 1, 2, 3, 2, 1…
## $ Q78                    <int> 2, 3, 2, 3, 3, 1, 3, 4, 3, 2, 2, 2, 1, 3, 3…
## $ Q79                    <int> 2, 3, 2, 3, 3, 1, 1, 2, 1, 3, 1, 2, 2, 2, 2…
## $ Q80                    <int> 2, 3, 2, 3, 3, 1, 1, 2, 3, 3, 1, 2, 2, 2, 2…
## $ Q81                    <int> 2, 3, 2, 3, 3, 1, 1, 3, 1, -2, 3, 2, 2, 2, …
## $ Q82                    <int> 2, 3, 2, 3, 2, -2, 2, 2, 3, 3, 2, 2, 2, 2, …
## $ Q82_AFRICANUNION       <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_APEC               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_ARABLEAGUE         <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_ASEAN              <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_CIS                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_CUSMA              <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_ECO                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_EU                 <int> 2, 3, 2, 3, 2, -2, 2, 2, 3, 3, 2, 2, 2, 2, …
## $ Q82_GULFCOOP           <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_ISLCOOP            <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_MERCOSUR           <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_NAFTA              <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_OAS                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_SAARC              <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_SCO                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_TLC                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q82_UNDP               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q83                    <int> 2, 3, 2, 3, 2, -2, 3, 3, 3, 3, 2, 2, 2, 2, …
## $ Q84                    <int> 2, 3, 3, 3, 2, -2, 3, 4, 3, 2, 2, 2, 2, 2, …
## $ Q85                    <int> 2, 3, 2, 3, 2, -2, -2, 4, 3, 3, 2, 2, 2, 2,…
```

```
## $ Q86                    <int> 2, 3, 2, 3, 2, -2, -2, 4, 1, 2, 2, 2, 2, 2,…
## $ Q87                    <int> 2, 3, 3, 3, 3, -2, 2, 4, 3, 2, 2, 2, 3, 2, …
## $ Q88                    <int> 2, 3, 3, 3, 2, -2, 3, 4, 2, 2, 1, 2, 2, 2, …
## $ Q89                    <int> 2, 3, 3, 3, 3, -2, 3, 4, 3, 3, 1, 2, 2, 2, …
## $ Q90                    <int> 6, 6, 4, 1, 2, 1, 1, 4, 9, 10, 7, 2, 1, 5, …
## $ Q91                    <int> -2, 3, 3, 2, 3, -2, 3, 3, 2, 3, 3, 3, 3, 3,…
## $ Q92                    <int> 3, 3, 2, 2, 2, -2, 3, 3, 3, 3, 3, 1, 3, 3, …
## $ Q93                    <int> -2, 2, 2, 1, 2, -2, 2, 2, 2, 1, 2, 2, 2, 2,…
## $ Q94                    <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 1, 1, 2…
## $ Q94R                   <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1…
## $ Q95                    <int> 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q95R                   <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q96                    <int> 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q96R                   <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q97                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q97R                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q98                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q98R                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q99                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q99R                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q100                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0…
## $ Q100R                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…
## $ Q101                   <int> 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0…
## $ Q101R                  <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0…
## $ Q102                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q102R                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q103                   <int> 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0…
## $ Q103R                  <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…
## $ Q104                   <int> 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0…
## $ Q104R                  <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…
## $ Q105                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q105R                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Q106                   <int> 5, 5, 7, 5, 4, 1, 1, 5, 8, 10, 8, 9, 5, 6, …
## $ Q107                   <int> 8, 2, 5, 8, 7, 1, 9, 4, 6, 5, 7, 8, 3, 5, 9…
## $ Q108                   <int> 2, 2, 5, 5, 7, 1, 2, 3, 6, 1, 2, 5, 5, 6, 1…
## $ Q109                   <int> 10, 2, 2, 4, 2, 10, 1, 3, 5, 1, 1, 1, 1, 2,…
## $ Q110                   <int> 2, 2, 4, 8, 6, 10, 1, 2, 5, 1, 7, 2, 2, 5, …
## $ Q111                   <int> 2, 3, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1…
```

```
## $ Q112                  <int> 2, 10, 7, 5, 5, 6, 7, 8, 5, 10, 2, 8, 10, 1…
## $ Q113                  <int> -2, 3, 2, 3, 2, 2, 2, 3, 2, 2, 1, 1, 3, 1, …
## $ Q114                  <int> -2, 3, 2, 3, 2, 2, 2, 3, 2, 3, 1, 2, 3, 1, …
## $ Q115                  <int> -2, 3, 2, 3, 2, 2, 3, 3, 2, 3, 1, 1, 4, 1, …
## $ Q116                  <int> -2, 3, 2, 3, 2, 2, 2, 3, 1, 3, 1, 1, 2, 1, …
## $ Q117                  <int> -2, 3, 2, 2, 1, 2, 1, 3, 2, 1, 1, 1, 1, 1, …
## $ Q118                  <int> 1, 1, 1, 1, 1, 1, 3, 1, 2, 1, 2, 1, 1, 1, 3…
## $ Q119                  <int> 2, 3, 2, 4, 3, 4, 4, 4, 0, 3, 3, 3, 4, 4, 4…
## $ Q120                  <int> 6, 2, 7, 7, 7, 2, 5, 7, 5, 10, 6, 2, 2, 5, …
## $ Q121                  <int> 5, 4, 4, 4, 4, 4, 4, 4, 3, 4, 5, 5, 4, 5, 5…
## $ Q122                  <int> 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 0, 1, 2…
## $ Q123                  <int> -2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, …
## $ Q124                  <int> 0, 1, 0, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0, 1, 2…
## $ Q125                  <int> 2, 0, 1, 0, 1, 0, 2, 2, 0, 1, 0, 0, 1, 2, -…
## $ Q126                  <int> 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2…
## $ Q127                  <int> -2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, …
## $ Q128                  <int> 2, 0, 0, 1, 1, 0, 0, 2, 0, 0, 1, 0, 2, 1, 2…
## $ Q129                  <int> 0, 1, 0, 1, 1, 0, 2, 2, 1, 1, 0, 0, 2, 1, 2…
## $ Q130                  <int> 2, 2, 2, 3, 3, 2, 1, 2, 3, 3, 3, 2, 2, 3, 2…
## $ Q131                  <int> 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1…
## $ Q132                  <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 3, 3, 4…
## $ Q133                  <int> 4, 4, 2, 4, 4, 2, 2, 4, 2, 1, 4, 4, 2, 3, 4…
## $ Q134                  <int> 4, 4, 3, 4, 4, -2, 4, 4, 4, 4, 4, 4, 2, 4, …
## $ Q135                  <int> 4, 4, 3, 4, 4, 2, 4, 4, 4, 2, 4, 4, 3, 4, 4…
## $ Q136                  <int> 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 2, 4, 3…
## $ Q137                  <int> 4, 4, 3, 4, 4, 1, 2, 4, 3, 4, 4, 4, 2, 4, 2…
## $ Q138                  <int> 4, 4, 4, 4, 4, -2, 4, 4, 4, 4, 4, 4, 3, 4, …
## $ Q139                  <int> 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q140                  <int> 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q141                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2…
## $ Q142                  <int> 2, 2, 3, 2, 2, 1, 3, 2, 1, 4, 3, 2, 2, 3, 2…
## $ Q143                  <int> 2, 2, 4, 2, 2, 4, 1, 4, 4, 4, 1, 2, 2, 3, 1…
## $ Q144                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q145                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q146                  <int> 4, 4, 4, 3, 3, 1, 2, 2, 4, 3, 2, 2, 3, 4, 2…
## $ Q147                  <int> 2, 4, 4, 4, 2, 1, 2, 3, 4, 2, 1, 2, 3, 3, 2…
## $ Q148                  <int> 4, 4, 4, 4, 4, 1, 4, 4, 4, 4, 2, 2, 4, 4, 2…
## $ Q149                  <int> 1, 2, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1…
```

```
## $ Q150                  <int> 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2…
## $ Q151                  <int> 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1…
## $ Q152                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4, 3, 1, 1, 3…
## $ Q153                  <int> 4, 4, 3, 3, 3, 2, 3, 3, 3, 2, 1, 1, 3, 3, 1…
## $ Q154                  <int> 3, 4, 1, 3, 3, 2, 4, 4, 4, 1, 2, 2, 2, 1, 3…
## $ Q155                  <int> 1, 2, 2, 4, 1, 3, 2, 2, 2, 4, 4, 4, 4, 2, 1…
## $ Q156                  <int> 1, 1, 2, 1, 1, 3, 2, 1, 1, 2, 2, 2, 1, 1, 1…
## $ Q157                  <int> 3, 4, 3, 2, 3, 4, 4, 4, 4, 3, 3, 3, 2, 2, 4…
## $ Q158                  <int> 7, 9, 5, 7, 7, 10, 5, 5, 5, 8, 8, 8, 9, 7, …
## $ Q159                  <int> 6, 9, 6, 6, 7, 10, 5, 9, 8, 8, 7, 7, 7, 5, …
## $ Q160                  <int> 7, 9, 8, 5, 6, 10, 10, 6, 8, 10, 9, 8, 8, 5…
## $ Q161                  <int> 5, 2, 5, 6, 5, 10, 10, 4, 8, 8, 8, 6, 8, 3,…
## $ Q162                  <int> 2, 2, 4, 3, 2, 10, 10, 2, 8, 1, 2, 2, 10, 5…
## $ Q163                  <int> 6, 10, 5, 6, 6, 10, 7, 9, 8, 2, 9, 6, 7, 6,…
## $ Q164                  <int> 7, 1, 8, 1, 4, 3, 8, 4, 5, 10, 10, 8, 1, 6,…
## $ Q165                  <int> 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1…
## $ Q166                  <int> 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2…
## $ Q167                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2…
## $ Q168                  <int> 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2…
## $ Q169                  <int> 3, 3, 3, 3, 3, 1, 1, 3, 4, 4, 3, 3, 1, 4, 1…
## $ Q170                  <int> 1, 3, 4, 3, 3, 1, 4, 3, 3, 4, 3, 3, 1, 4, 1…
## $ Q171                  <int> 7, 7, 7, 7, 7, 3, 3, 7, 6, 2, 7, 3, 7, 4, 3…
## $ Q172                  <int> 6, 8, 8, 8, 8, 2, 5, 7, 7, 2, 4, 3, 8, 8, 1…
## $ Q172R                 <int> 3, 4, 4, 4, 4, 1, 3, 4, 4, 1, 3, 2, 4, 4, 1…
## $ Q173                  <int> 1, 2, 2, 3, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1…
## $ Q174                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q175                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2…
## $ Q176                  <int> 4, -2, 5, 4, 5, 4, 10, 6, 5, 10, 5, 7, 7, 1…
## $ Q177                  <int> 1, 1, 1, 1, 1, 10, 1, 10, 9, 1, 1, 1, 1, 1,…
## $ Q178                  <int> 1, 1, 1, 1, 1, 10, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ Q179                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ Q180                  <int> 1, 1, 1, 1, 1, 1, 1, 5, 5, 1, 1, 1, 1, 1, 1…
## $ Q181                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1…
## $ Q182                  <int> 4, 9, 10, 6, 8, 10, 10, 10, 8, 10, 7, 1, 10…
## $ Q183                  <int> 3, 5, 6, 6, 6, 1, 5, 6, 8, 1, 1, 3, 10, 4, …
## $ Q184                  <int> 2, 9, 5, 4, 5, 1, 10, 6, 5, 8, 4, 7, 10, 10…
## $ Q185                  <int> 5, 9, 6, 9, 7, 10, 10, 10, 5, -2, 8, 5, 10,…
## $ Q186                  <int> 4, 10, 6, 10, 8, 10, 10, 10, 7, 8, 4, 1, 10…
```

```
## $ Q187                  <int> 1, 1, 5, 3, 1, 1, 1, 10, 1, 1, 1, 1, 10, 3,…
## $ Q188                  <int> 1, 9, 8, 3, 5, 1, 10, 10, 8, 8, 4, 1, 10, 1…
## $ Q189                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ Q190                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1…
## $ Q191                  <int> 1, 1, 2, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1…
## $ Q192                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ Q193                  <int> 1, 5, 5, 7, 6, 10, 1, 10, 8, 3, 4, 1, 10, 1…
## $ Q194                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ Q195                  <int> 1, 1, 2, 1, 1, 1, 5, 6, 5, 8, 1, 1, 10, 1, …
## $ Q196                  <int> 2, 1, 4, 2, 1, 4, 1, 1, 4, 1, 1, 1, 4, 1, 1…
## $ Q197                  <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4…
## $ Q198                  <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4…
## $ Q199                  <int> 2, 4, 3, 2, 2, 4, 4, 3, 3, 4, 4, 1, 4, 3, 4…
## $ Q200                  <int> 2, 2, 2, -2, 2, 3, 3, 2, 2, 3, 3, 1, 2, 2, …
## $ Q201                  <int> 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 5…
## $ Q202                  <int> 2, 2, 1, 2, 2, 1, 5, 1, 1, 2, 1, 1, 1, 3, 1…
## $ Q203                  <int> 1, 5, 1, 5, 5, 5, 5, 1, 5, 5, 5, 1, 2, 1, 5…
## $ Q204                  <int> 5, 1, 5, 5, 2, 1, 1, 1, 1, 4, 1, 5, 1, 5, 1…
## $ Q205                  <int> 5, 1, 5, 5, 2, 5, 1, 1, 5, 4, 5, 5, 1, 5, 5…
## $ Q206                  <int> 5, 1, 2, 5, 2, 5, 1, 1, 5, 4, 1, 5, 1, 1, 1…
## $ Q207                  <int> 5, 1, 5, 5, 2, 5, 1, 1, 5, 1, 1, 5, 4, 1, 1…
## $ Q208                  <int> 2, 1, 4, 5, 2, 5, 1, 3, 2, 2, 5, 2, 4, 2, 1…
## $ Q209                  <int> 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2…
## $ Q210                  <int> 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 1, 3, 3, 2, 3…
## $ Q211                  <int> 3, 2, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2…
## $ Q212                  <int> 3, 2, 2, 2, 1, 2, 2, 1, 3, 3, 1, 2, 2, 2, 2…
## $ Q213                  <int> 3, 1, 1, 2, 1, 3, 2, 1, 1, 1, 3, 1, 1, 1, 1…
## $ Q214                  <int> 3, 2, 1, 3, 3, 2, 3, 1, 3, 3, 1, 1, 2, 2, 1…
## $ Q215                  <int> 3, 1, 1, 2, 2, 3, 3, 2, 2, 1, 1, 1, 1, 2, 1…
## $ Q216                  <int> 3, 1, 2, 2, 2, 2, 3, 3, 1, 3, 1, 3, 3, 2, 1…
## $ Q217                  <int> 3, 1, 2, 2, 1, 3, 2, 2, 3, 1, 1, 1, 2, 1, 1…
## $ Q218                  <int> 3, 2, 1, 3, 2, 3, 2, 1, 3, 3, 2, 1, 1, 2, 1…
## $ Q219                  <int> 3, 2, 2, 3, 2, 3, 3, 2, 3, 3, 1, 1, 1, 2, 1…
## $ Q220                  <int> 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 2, 3, 1, 2, 1…
## $ Q221                  <int> 1, 1, 2, 2, 2, 4, 4, 1, 4, 4, 1, 1, 4, 1, 4…
## $ Q222                  <int> 1, 1, 2, 2, 2, 4, 4, 1, 4, 4, 1, 1, 4, 1, 4…
## $ Q223                  <int> 20001, 20001, 20001, 2, 20005, -2, 2, -2, 2…
## $ Q223_ABREV            <int> 20001, 20001, 20001, 2, 20005, -2, 2, -2, 2…
```

```
## $ Q223_LOCAL                <int> 20001, 20001, 20001, 2, 20005, -2, 2, -2, 2…
## $ Q224                      <int> 3, 1, 1, 3, 2, -2, 1, 1, 1, 2, 1, 1, 2, 1, …
## $ Q225                      <int> 4, 4, 4, 4, 4, 1, 4, 4, 3, 4, 4, 4, 1, 4, 4…
## $ Q226                      <int> 2, 1, 2, 2, 3, 1, 1, 1, 1, 4, 2, 1, 4, 3, 3…
## $ Q227                      <int> 3, 4, 2, 2, 3, -2, 1, 1, 4, 4, 4, 3, 2, 2, …
## $ Q228                      <int> 1, 1, -2, 1, 2, 4, 1, 1, 2, 1, 2, 1, 2, 3, …
## $ Q229                      <int> 2, 1, 1, 3, 2, 1, 2, 1, 3, 2, 1, 1, 4, 1, 1…
## $ Q230                      <int> 3, 3, 2, 2, 3, -2, 1, 1, 2, 4, 4, 4, 1, 4, …
## $ Q231                      <int> 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4…
## $ Q232                      <int> 2, 1, 4, 2, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1…
## $ Q233                      <int> 3, 1, 2, 3, 2, 1, 1, 1, 2, 1, 2, 1, 4, 1, 1…
## $ Q234                      <int> 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1…
## $ Q234A                     <int> 2, 1, 4, 2, 3, 1, 4, 2, 3, 4, 1, 5, 4, 1, 5…
## $ Q235                      <int> 4, 4, 4, 3, 3, 1, 4, 4, 4, 4, 1, 4, 4, 4, -…
## $ Q236                      <int> 4, 4, 2, 2, 3, 1, 1, 1, 1, 1, 4, 4, 2, 2, 1…
## $ Q237                      <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4…
## $ Q238                      <int> 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1…
## $ Q239                      <int> 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4…
## $ Q240                      <int> 6, 5, 7, 5, 6, -2, 1, 8, 5, 5, 6, 5, 5, 5, …
## $ Q241                      <int> 7, 9, 7, 9, 8, 1, 4, 1, 1, 1, 9, 10, 8, 1, …
## $ Q242                      <int> 2, 7, 7, 8, 2, 1, 1, 1, 1, 1, 2, 6, 1, 1, 8…
## $ Q243                      <int> 8, 10, 10, 8, 8, 10, 10, 10, 8, 10, 9, 10, …
## $ Q244                      <int> 8, 6, 7, 10, 7, 10, -2, 10, 7, 6, 9, 8, 9, …
## $ Q245                      <int> 1, 1, 4, 2, 2, 1, 1, 1, 8, 1, 2, 1, 1, 1, 1…
## $ Q246                      <int> 7, 10, 8, 9, 8, 10, 10, 10, 1, 1, 1, 10, 10…
## $ Q247                      <int> 7, 4, 4, 7, 6, 1, 1, 1, 5, 1, 5, 10, 6, 1, …
## $ Q248                      <int> 7, 5, 7, 7, 8, 1, 1, -2, 5, 1, 9, 10, 8, 1,…
## $ Q249                      <int> 6, 10, 10, 9, 7, 10, 10, 10, 8, 10, 10, 10,…
## $ Q250                      <int> 8, 10, 10, 7, 8, 10, 7, 10, 10, 5, 10, 10, …
## $ Q251                      <int> 8, 6, 8, 6, 8, 10, 5, 7, 6, 5, 8, 8, 5, 9, …
## $ Q252                      <int> 8, 6, 3, 6, 7, 10, 4, 7, 5, 7, 9, 8, 1, 7, …
## $ Q253                      <int> 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 1, 1…
## $ Q254                      <int> 5, 1, 3, 5, 5, 5, 5, 1, 5, 5, 1, 2, 5, 1, 5…
## $ Q255                      <int> 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ Q256                      <int> 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1…
## $ Q257                      <int> 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1…
## $ Q258                      <int> 1, 1, 2, 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 2, 1…
## $ Q259                      <int> 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1…
```

```
## $ Q260                  <int> 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2…
## $ Q261                  <int> 1958, 1971, 1969, 1956, 1969, 1967, 1985, 1…
## $ Q262                  <int> 60, 47, 48, 62, 49, 51, 33, 55, 40, 38, 54,…
## $ X003R                 <int> 5, 4, 4, 5, 4, 4, 2, 5, 3, 3, 4, 3, 3, 4, 3…
## $ X003R2                <int> 3, 2, 2, 3, 2, 3, 2, 3, 2, 2, 3, 2, 2, 3, 2…
## $ Q263                  <int> 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2…
## $ Q264                  <int> 2, 2, 2, 2, 2, -3, -3, 2, 2, 2, 2, 2, 2, 1,…
## $ V002                  <int> 0, 0, 0, 0, 0, -3, -3, 0, 0, 0, 0, 0, 0, 1,…
## $ Q265                  <int> 2, 2, 2, 2, 2, -3, -3, 2, 2, 2, 2, 2, 2, 2,…
## $ V001                  <int> 0, 0, 0, 0, 0, -3, -3, 0, 0, 0, 0, 0, 0, 0,…
## $ Q266                  <int> 724, 20, 724, 724, 724, 724, 620, 724, 620,…
## $ X002_02B              <chr> "ES", "AD", "ES", "ES", "ES", "ES", "PT", "…
## $ Q267                  <int> 724, 724, 724, 724, 724, 724, 620, 724, 620…
## $ V002A_01              <chr> "ES", "ES", "ES", "ES", "ES", "ES", "PT", "…
## $ Q268                  <int> 724, 724, 724, 724, 724, 724, 620, 724, 620…
## $ V001A_01              <chr> "ES", "ES", "ES", "ES", "ES", "ES", "PT", "…
## $ Q269                  <int> 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2…
## $ Q270                  <int> 2, 4, 2, 2, 2, 4, 2, 1, 1, 3, 2, 4, 3, 4, 3…
## $ Q271                  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1…
## $ Q272                  <int> 1270, 1270, 810, 1270, 1270, 1270, 3530, 81…
## $ Q273                  <int> 1, 2, 4, 2, 2, 1, 1, 5, 6, 2, 1, 1, 2, 1, 4…
## $ Q274                  <int> 2, 0, 0, 1, 0, 3, 1, 2, 0, 1, 1, 2, 1, 2, 3…
## $ Q275                  <int> 3, 7, 7, 2, 2, 1, 1, 4, 1, 5, 3, 7, 1, 6, 1…
## $ Q275A                 <int> 20003, 20007, 20007, 20002, 20002, 20001, 2…
## $ Q275R                 <int> 2, 3, 3, 1, 1, 1, 1, 2, 1, 3, 2, 3, 1, 3, 1…
## $ Q276                  <int> 3, 6, -3, 2, 2, 1, 1, -3, -3, 3, 1, 7, 1, 6…
## $ Q276A                 <int> 20003, 20006, -3, 20002, 20002, 20001, 2000…
## $ Q276R                 <int> 2, 3, -3, 1, 1, 1, 1, -3, -3, 2, 1, 3, 1, 3…
## $ Q277                  <int> 3, 3, 5, 1, 1, 1, 0, 1, -2, 1, 1, 6, 1, 6, …
## $ Q277A                 <int> 20003, 20003, 20005, 20001, 20001, 20001, 2…
## $ Q277R                 <int> 2, 2, 3, 1, 1, 1, 1, 1, -2, 1, 1, 3, 1, 3, …
## $ Q278                  <int> 3, 3, 6, 1, 1, 1, 0, 1, -2, 1, 1, 6, 1, 1, …
## $ Q278A                 <int> 20003, 20003, 20006, 20001, 20001, 20001, 2…
## $ Q278R                 <int> 2, 2, 3, 1, 1, 1, 1, 1, -2, 1, 1, 3, 1, 1, …
## $ Q279                  <int> 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 7, 7, 1, 1, 1…
## $ Q280                  <int> 1, 1, -3, 1, 1, 1, 1, -3, -3, 1, 1, 1, 1, 1…
## $ Q281                  <int> 4, 1, 5, 4, 4, 8, 8, 4, 4, 4, 7, 5, 5, 1, 4…
## $ Q282                  <int> 7, 3, -3, 6, 5, 6, 7, -3, -3, 6, 5, 1, 5, 1…
```

```
## $ Q283                  <int> 9, 4, 5, 9, 8, 7, 7, 5, 7, 5, 5, 1, 5, 7, 7…
## $ Q284                  <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2…
## $ Q285                  <int> 2, 1, 1, 2, 1, 2, 2, 1, 1, -2, 2, 2, 2, 2, …
## $ Q286                  <int> 3, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1…
## $ Q287                  <int> 3, 2, 3, 4, 4, 4, 4, 2, 4, 3, 3, 2, 4, 2, 4…
## $ Q288                  <int> 5, 9, 5, 4, 4, 5, 2, -2, 4, 6, 4, 10, 5, 5,…
## $ Q288R                 <int> 2, 3, 2, 2, 2, 2, 1, -2, 2, 2, 2, 3, 2, 2, …
## $ Q289                  <int> 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1…
## $ Q289CS9               <int> 10100000, 10100000, 10100000, 100000020, 10…
## $ Q290                  <int> 20001, 20001, 20001, 20001, 20001, 20001, 2…
## $ Q291G1                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291G2                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291G3                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291G4                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291G5                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291G6                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P1                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P2                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P3                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P4                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P5                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291P6                <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN1               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN2               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN3               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN4               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN5               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q291UN6               <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292A                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292B                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292C                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292D                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292E                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292F                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292G                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292H                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292I                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292J                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
```

```
## $ Q292K                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292L                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292M                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292N                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q292O                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q293                  <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q294A                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Q294B                 <int> -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,…
## $ Y001                  <int> 1, 2, 4, 3, 2, 2, 4, 3, 3, 4, 4, 5, 4, 3, 1…
## $ Y002                  <int> 1, 3, 2, 2, 1, 2, 3, 3, 3, 2, 3, 3, 3, 2, 1…
## $ Y003                  <int> 0, -1, 2, 0, 1, 0, 0, -1, 1, 1, 2, 0, 2, 1,…
## $ sacsecval             <dbl> 0.287062, 0.467525, 0.425304, 0.556170, 0.4…
## $ resemaval             <dbl> 0.413241, 0.519722, 0.692917, 0.481065, 0.4…
## $ I_AUTHORITY           <dbl> 0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.5, 0.5…
## $ I_NATIONALISM         <dbl> NA, 0.00, 0.66, NA, NA, NA, NA, 0.00, NA, N…
## $ I_DEVOUT              <dbl> 0.66, 0.66, 0.66, 0.66, 0.66, 0.00, 0.00, 0…
## $ defiance              <dbl> 0.382580, 0.220000, 0.440000, 0.574580, 0.5…
## $ I_RELIGIMP            <dbl> 1.00, 1.00, 0.66, 1.00, 0.66, 0.66, 0.33, 0…
## $ I_RELIGBEL            <int> 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0…
## $ I_RELIGPRAC           <dbl> 1.000000, 1.000000, 1.000000, 1.000000, 1.0…
## $ disbelief             <dbl> 0.666667, 1.000000, 0.886667, 1.000000, 0.8…
## $ I_NORM1               <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ I_NORM2               <int> 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0…
## $ I_NORM3               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…
## $ relativism            <dbl> 0.000000, 0.000000, 0.000000, 0.000000, 0.0…
## $ I_TRUSTARMY           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,…
## $ I_TRUSTPOLICE         <dbl> 0.00, 0.66, 0.33, 0.66, 0.33, 0.00, 0.00, 0…
## $ I_TRUSTCOURTS         <dbl> 0.00, 0.66, 0.33, 0.66, 0.33, 0.00, 1.00, 0…
## $ scepticism            <dbl> 0.09900, 0.65010, 0.37455, 0.65010, 0.37455…
## $ I_INDEP               <int> 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1…
## $ I_IMAGIN              <int> 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0…
## $ I_NONOBED             <int> 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1…
## $ autonomy              <dbl> 0.666667, 0.000000, 1.000000, 0.000000, 0.3…
## $ I_WOMJOB              <dbl> 0.75, 0.75, 0.75, 0.75, 0.75, 1.00, 1.00, 1…
## $ I_WOMPOL              <dbl> 0.66, 0.66, 0.66, 1.00, 0.66, 0.33, 1.00, 1…
## $ I_WOMEDU              <dbl> 0.66, 0.66, 0.66, 1.00, 0.66, 1.00, 1.00, 1…
## $ equality              <dbl> 0.690000, 0.690000, 0.690000, 0.916667, 0.6…
## $ I_HOMOLIB             <dbl> 0.333333, 0.888889, 1.000000, 0.555556, 0.7…
```

```
## $ I_ABORTLIB                <dbl> 0.111111, 0.888889, 0.444444, 0.333333, 0.4…
## $ I_DIVORLIB                <dbl> 0.444444, 0.888889, 0.555556, 0.888889, 0.6…
## $ choice                    <dbl> 0.296296, 0.888889, 0.666667, 0.592593, 0.6…
## $ I_VOICE1                  <dbl> 0.00, 1.00, 0.33, 0.33, 0.00, 0.66, 1.00, 1…
## $ I_VOICE2                  <dbl> 0.0, 0.0, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5…
## $ I_VOI2_00                 <dbl> 0.000, 0.500, 0.415, 0.415, 0.250, 0.330, 0…
## $ voice                     <dbl> 0.000, 0.500, 0.415, 0.415, 0.250, 0.330, 0…
## $ secvalwgt                 <dbl> 0.830, 0.915, 0.915, 0.830, 0.830, 0.830, 0…
## $ resemavalwgt              <dbl> 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1…
## $ fhregion                  <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4…
## $ polregfh                  <int> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3…
## $ freestfh                  <int> 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94,…
## $ prfhrat                   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ prfhscore                 <int> 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,…
## $ clfhrat                   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ clfhscore                 <int> 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,…
## $ democ                     <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ autoc                     <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ polity                    <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ durable                   <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ regtype                   <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ ruleoflaw                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ corrupttransp             <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ electintegr               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btiregion                 <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btistatus                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btidemstatus              <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btistate                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btipolpart                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btiruleoflaw              <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btistability              <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btiintegration            <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btimarket                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btigovindex               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btigoveperform            <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ btiregime                 <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ regionWB                  <int> 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6…
## $ incomeWB                  <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4…
```

```
## $ landWB                  <dbl> 470, 470, 470, 470, 470, 470, 470, 470, 470…
## $ GDPpercap1              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ GDPpercap2              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ giniWB                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ incrichest10p           <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ popWB1990               <int> 54509, 54509, 54509, 54509, 54509, 54509, 5…
## $ popWB2000               <int> 65390, 65390, 65390, 65390, 65390, 65390, 6…
## $ popWB2019               <int> 77142, 77142, 77142, 77142, 77142, 77142, 7…
## $ lifeexpect              <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ popgrowth               <dbl> 0.18, 0.18, 0.18, 0.18, 0.18, 0.18, 0.18, 0…
## $ urbanpop                <dbl> 87.98, 87.98, 87.98, 87.98, 87.98, 87.98, 8…
## $ laborforce              <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ deathrate               <dbl> 4.4, 4.4, 4.4, 4.4, 4.4, 4.4, 4.4, 4.4, 4.4…
## $ unemployfem             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ unemploymale            <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ unemploytotal           <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ accessclfuel            <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100…
## $ accesselectr            <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100…
## $ renewelectr             <dbl> 86.12, 86.12, 86.12, 86.12, 86.12, 86.12, 8…
## $ co2emis                 <dbl> 469.38, 469.38, 469.38, 469.38, 469.38, 469…
## $ co2percap               <dbl> 6.07, 6.07, 6.07, 6.07, 6.07, 6.07, 6.07, 6…
## $ easeofbusiness          <int> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ militaryexp             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ Trade                   <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ healthexp               <dbl> 10.32, 10.32, 10.32, 10.32, 10.32, 10.32, 1…
## $ educationexp            <dbl> 3.2, 3.2, 3.2, 3.2, 3.2, 3.2, 3.2, 3.2, 3.2…
## $ medageun                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ meanschooling           <dbl> 10.2, 10.2, 10.2, 10.2, 10.2, 10.2, 10.2, 1…
## $ educationHDI            <dbl> 0.708, 0.708, 0.708, 0.708, 0.708, 0.708, 0…
## $ compulseduc             <int> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,…
## $ gii                     <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ dgi                     <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ womenparl               <dbl> 46.43, 46.43, 46.43, 46.43, 46.43, 46.43, 4…
## $ hdi                     <dbl> 0.857, 0.857, 0.857, 0.857, 0.857, 0.857, 0…
## $ incomeindexHDI          <dbl> 0.935, 0.935, 0.935, 0.935, 0.935, 0.935, 0…
## $ humanineqiality         <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ lifeexpectHDI           <dbl> 0.951, 0.951, 0.951, 0.951, 0.951, 0.951, 0…
## $ homiciderate            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
```

```
## $ Refugeesorigin            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ internetusers             <dbl> 91.6, 91.6, 91.6, 91.6, 91.6, 91.6, 91.6, 9…
## $ mobphone                  <dbl> 107.3, 107.3, 107.3, 107.3, 107.3, 107.3, 1…
## $ migrationrate             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ schoolgpi                 <dbl> 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0…
## $ femchoutsch               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ choutsch                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_polyarchy             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_libdem                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_partipdem             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_delibdem              <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_egaldem               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_freexp_altinf         <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_frassoc_thick         <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2xel_frefair             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2xcl_rol                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_cspart                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2xeg_eqdr                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2excrptps                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2exthftps                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2juaccnt                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2cltrnslw                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2clacjust                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2clsocgrp                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2clacfree                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2clrelig                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2csrlgrep                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2mecenefm                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2mecenefi                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2mebias                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2pepwrses                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2pepwrgen                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2peedueq                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2pehealth                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2peapsecon               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2peasjsoecon             <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2clgencl                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2peasjgen                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
```

```
## $ v2peasbgen                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2cafres                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2cafexch                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_corr                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_gender                <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_gencl                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_genpp                 <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2x_rule                  <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ v2xcl_acjst               <dbl> -9999, -9999, -9999, -9999, -9999, -9999, -…
## $ td_voiacc                 <dbl> 1.11, 1.11, 1.11, 1.11, 1.11, 1.11, 1.11, 1…
## $ td_polstab                <dbl> 1.6, 1.6, 1.6, 1.6, 1.6, 1.6, 1.6, 1.6, 1.6…
## $ td_goveff                 <dbl> 1.9, 1.9, 1.9, 1.9, 1.9, 1.9, 1.9, 1.9, 1.9…
## $ td_regqual                <dbl> 1.23, 1.23, 1.23, 1.23, 1.23, 1.23, 1.23, 1…
## $ td_rulelaw                <dbl> 1.57, 1.57, 1.57, 1.57, 1.57, 1.57, 1.57, 1…
## $ td_ctrlcorr               <dbl> 1.23, 1.23, 1.23, 1.23, 1.23, 1.23, 1.23, 1…
## $ ID_GPS                    <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ ID_PartyFacts             <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ Partyname                 <chr> "NA", "NA", "NA", "I would not vote", "NA",…
## $ Partyabb                  <chr> "NA", "NA", "NA", "NA", "NA", "NA", "NA", "…
## $ cparty                    <chr> "NA", "NA", "NA", "NA", "NA", "NA", "NA", "…
## $ cpartyabb                 <chr> "NA", "NA", "NA", "NA", "NA", "NA", "NA", "…
## $ Type_Values               <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ Type_Populism             <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ Type_Populist_Values      <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ Type_Partysize_vote       <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ Type_Partysize_seat       <int> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V4_Scale              <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V6_Scale              <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V8_Scale              <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V9                    <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V10                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V11                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V12                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V13                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V14                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V15                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V16                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ GPS_V17                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
```

```
## $ WVS_LR_PartyVoter          <dbl> 5.944634, 5.944634, 5.944634, NA, 3.899767,…
## $ WVS_LibCon_PartyVoter      <dbl> 3.908421, 3.908421, 3.908421, NA, 2.816576,…
## $ WVS_Polmistrust_PartyVoter <dbl> 62.43421, 62.43421, 62.43421, NA, 66.96429,…
## $ WVS_LR_MedianVoter         <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ WVS_LibCon_MedianVoter     <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2psbars                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2psorgs                   <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2psprbrch                 <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2psprlnks                 <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2psplats                  <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2xnp_client               <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
## $ v2xps_party                <dbl> -999, -999, -999, NA, -999, NA, NA, NA, -99…
```

2.1 Select

The `select()` function allows you to extract specific columns by name or by using functions that match column names. Since our dataset contains many variables, we will select only those relevant to the tutorial sessions.

```
# Create a new dataset containing only the selected columns:
Data <- WVS %>%
  select(D_INTERVIEW, B_COUNTRY, Q47, Q49, Q124, Q128, Q152, Q189, Q191, Q260, Q262,
Q273, Q275, Q288)

Data %>% head(10) #displaying the first 10 rows
```

```
D_INTERVIEWB_COUNTRYQ47Q49Q124Q128Q152Q189Q191
```

```
1200700012031002111
```

```
220070002201910111

320070003201900112

420070004202801111

520070005202711111

620070006201101011l

720070007201500111

820070008202802111

920070009202800111

1020070010201102031 3
```

These are the variables that we will use for the entire tutorial session. You can check the codebook to see what each variable refers to.

Using - with `select()` allows you to exclude specific columns while keeping all others:

```
Data %>% select(-D_INTERVIEW) %>% head(10)
```

	B_COUNTRY	Q47	Q49	Q124	Q128	Q152	Q189	Q191	Q260
1	20	3	10	0	2	1	1	1	2
2	20	1	9	1	0	1	1	1	1
3	20	1	9	0	0	1	1	2	1
4	20	2	8	0	1	1	1	1	2
5	20	2	7	1	1	1	1	1	1
6	20	1	10	1	0	1	1	1	2
7	20	1	5	0	0	1	1	1	2
8	20	2	8	0	2	1	1	1	1
9	20	2	8	0	0	1	1	1	2
10	20	1	10	2	0	3	1	3	2

There are additional `select()` arguments that allow you to filter columns based on specific patterns:

```
# Select columns that start with a given string: starts_with("prefix")
 Data %>% select(starts_with("D")) %>% head(10)
```

	D_INTERVIEW
1	20070001
2	20070002
3	20070003
4	20070004
5	20070005
6	20070006
7	20070007
8	20070008
9	20070009
10	20070010

```
# Select columns that end with a given string: ends_with("suffix")
 Data %>% select(ends_with("Y")) %>% head(10)
```

	B_COUNTRY
1	20
2	20
3	20
4	20
5	20
6	20
7	20
8	20
9	20
10	20

```
# Select columns that match a specific string or regular expression: matches("pattern")
Data %>% select(matches(".6.")) %>% head(10)
```

	Q260	Q262
1	2	60
2	1	47
3	1	48
4	2	62
5	1	49
6	2	51
7	2	33
8	1	55
9	2	40
10	2	38

2.2 Rename and Arrange

The `rename()` function in *dplyr* is used to change column names in a dataset.

```
# Rename columns: rename(new_name = old_name)

Cleaned_WVS <- Data %>%
    rename( Country = B_COUNTRY,
            Age = Q262,
            Sex = Q260,
            Marital_Status = Q273,
            Life_satisfaction = Q49,
            EDU = Q275,
            Income = Q288,
            Wife_Abuse = Q189,
            Violence_otherPPL = Q191)

Cleaned_WVS %>% head(10)
```

	D_INTERVIEW	Country	Q47	Life_satisfaction	Q124	Q128	Q152	Wife_Abuse
1	20070001	20	3	10	0	2	1	1
2	20070002	20	1	9	1	0	1	1
3	20070003	20	1	9	0	0	1	1
4	20070004	20	2	8	0	1	1	1
5	20070005	20	2	7	1	1	1	1
6	20070006	20	1	10	1	0	1	1
7	20070007	20	1	5	0	0	1	1
8	20070008	20	2	8	0	2	1	1
9	20070009	20	2	8	0	0	1	1
10	20070010	20	1	10	2	0	3	1

Additionally, the `arrange()` function is used to sort rows in a dataset based on a specified variable. By default, it orders values in ascending order, placing the highest category at the bottom. To display the highest category first, use the `desc()` function to arrange values in descending order.

```
#Order values in ascending order
Cleaned_WVS %>% select(Age) %>%
  arrange(Age) %>% head(10)
```

	Age
29089	-5
29096	-5
29097	-5
29214	-5
29231	-5
29257	-5
29258	-5
29274	-5
29281	-5
29331	-5

```
#Order values in descending order
Cleaned_WVS %>% select(Age) %>%
  arrange(desc(Age)) %>% head(10)
```

	Age
71442	103
61280	100
27054	99
39294	99
39363	99
39377	99
39553	99
39872	99
40300	99
40346	99

2.3 Filter

The `filter()` function extracts rows from a data frame based on specified conditions. For example, to select only **female** respondents from the WVS dataset, you can use:

```
Cleaned_WVS %>%
  select(Age, Sex, Marital_Status) %>%
  filter(Sex ==2) %>%     #The value for female in Sex is 2.
  head(10)
```

	Age	Sex	Marital_Status
1	60	2	1
4	62	2	2
6	51	2	1
7	33	2	1
9	40	2	6
10	38	2	2
13	44	2	2
14	53	2	1
15	43	2	4
18	61	2	1

You can apply multiple conditions by separating them with commas (,):

```
Cleaned_WVS %>%
  select(Age, Sex, Marital_Status) %>%
  filter(Sex ==2, Marital_Status ==1) %>%  #The value for female in Sex is 2, the value
for married in
Marital_Status is 1.
  head(10)
```

	Age	Sex	Marital_Status
1	60	2	1
6	51	2	1
7	33	2	1
14	53	2	1
18	61	2	1
20	51	2	1
21	43	2	1
27	63	2	1
28	66	2	1
29	48	2	1

***Note: The equivalent code in base R:**

```
head(Cleaned_WVS[Cleaned_WVS$Sex == 2 & Cleaned_WVS$Marital_Status == 1, c("Sex", "Mari-
tal_Status")], 10)
```

	Sex	Marital_Status
1	2	1
6	2	1
7	2	1
14	2	1
18	2	1
20	2	1
21	2	1
27	2	1
28	2	1
29	2	1

2.4 Mutate

The `mutate()` function is used to add new variables to an existing data frame. For example, you can recode a 10-point Likert scale into a 100-point percentage scale for practice:

```
Cleaned_WVS %>%
  mutate(Percen_Life_satisfaction = ((Life_satisfaction - 1) / (10 - 1)) * 100) %>%
                                    #Simple transformation: New Score=(Old Score-Min/
Max-Min)×100
  select(Life_satisfaction, Percen_Life_satisfaction) %>%
  head(10)
```

	Life_satisfaction	Percen_Life_satisfaction
1	10	100
2	9	88.88889
3	9	88.88889
4	8	77.77778
5	7	66.66667
6	10	100
7	5	44.44444
8	8	77.77778
9	8	77.77778
10	10	100

You can also recode the variable Q275 (EDU) into a new variable, EDU3, which classifies respondents into three education levels: High, Middle, and Low.

```
Cleaned_WVS %>%
  select(EDU) %>%
  mutate(EDU3 = if_else(EDU >= 5, "High_EDU",
                                if_else(EDU >= 3, "Middle_EDU",
                                        if_else(EDU >= 0, "Low_EDU", NA)))) %>%
  head(10)
```

	EDU	EDU3
1	3	Middle_EDU
2	7	High_EDU
3	7	High_EDU
4	2	Low_EDU
5	2	Low_EDU
6	1	Low_EDU
7	1	Low_EDU
8	4	Middle_EDU
9	1	Low_EDU
10	5	High_EDU

2.5 Summarise

The `summarise()` function allows you to generate summary statistics for variables in a dataset. In general, it enables you to apply any function to a column, as long as it returns a single value.

*Note: `summarise()` and `summarize()` are functionally identical. They are simply alternative spellings of the same function in the *dplyr* package in R.

We observed negative values in the results above (e.g., -5), which indicate missing data. Let's count the number of negative values in each column using the `summarise()` function.

```
#Count the number of negative values in each column
Cleaned_WVS %>%
  summarise(across(everything(), ~ sum(. < 0, na.rm = TRUE))) #across(everything()):
applies the
function to all variables, ~ sum(. < 0, na.rm = TRUE) counts how many negative values
exist in each
 column.
```

Let's treat the negative values in each column as missing values (`NA`).

```
# Treat negative values as NA
Cleaned_WVS <- Cleaned_WVS %>%
```

```
  mutate(across(everything(), ~ if_else(. < 0, NA_real_, .))) #  Checks all columns
(everything()).
#Replaces values < 0 with NA_real_ (setting missing values as NA in the numeric) and
leaves other values
unchanged.

#Check missing values
colSums(is.na(Cleaned_WVS))
```

```
##        D_INTERVIEW            Country                Q47 Life_satisfaction
##                  0                  0                267                520
##               Q124               Q128               Q152        Wife_Abuse
##               2727               2305               2557              4076
## Violence_otherPPL                Sex                Age    Marital_Status
##               1022                 95                511               589
##                EDU             Income
##               1071               2961
```

The negative values have been successfully removed from each variable. Let's now check the summary statistics for the means of *Life_satisfaction*.

```
#Create a multiple average
Cleaned_WVS %>%
  summarize( avg_Life_satisfaction = mean(Life_satisfaction, na.rm = T))
```

avg_Life_satisfaction
7.06213

2.6 Grouping

Grouping enables operations to be performed within specific subsets of data, making it particularly useful for summarising categorical variables. For example, to count each category within the *Marital_Status* variable, you can use the `group_by()` function with `summarise()`:

```
#To count each value in the Marital_Status variable using summarize(), you can use
```

```
group_by() along with
n():
Cleaned_WVS %>%
  group_by(Marital_Status) %>%
  summarise(count = n(), percentage = (n() / nrow(Cleaned_WVS)) * 100)
```

Marital_Status	count	percentage
1	53819	55.357951
2	7637	7.8553796
3	4298	4.420901
4	2098	2.1579922
5	5529	5.6871014
6	23250	23.9148323
NA	589	0.6058424

```
#Count without NAs
  Cleaned_WVS %>%
  filter(!is.na(Marital_Status)) %>%
  group_by(Marital_Status) %>%
  summarise(
    count = n(),
    percentage = (n() / nrow(Cleaned_WVS %>% filter(!is.na(Marital_Status)))) * 100
)
```

Marital_Status	count	percentage
1	53819	55.695377
2	7637	7.903261
3	4298	4.447848
4	2098	2.171146
5	5529	5.721766
6	23250	24.060602

We can also group respondents by country to calculate summary statistics for each country separately,

rather than for all respondents combined. For example, we can compare the average score on attitudes toward *wife abuse* between Australia and New Zealand.

```
Cleaned_WVS %>%
  filter(Country %in% c(36,554)) %>% #36: Australia, 554: New Zealand
  group_by(Country) %>%
  summarise(avg_Wife_Abuse = mean(Wife_Abuse, na.rm = T))
```

Country	avg_Wife_Abuse
36	1.241341
554	1.136319

The %in% operator checks whether the values on its left (e.g., country) are present in the values on its right (in this case, the codes 36 and 554, which represent Australia and New Zealand). It returns TRUE for each element in country that matches either of these codes.

You can also use grouping with mutate() to create new variables, as shown below:

```
WA <- Cleaned_WVS  %>%
  filter(Country %in% c(36,554)) %>%
  group_by(Country) %>%
  mutate(avg_Wife_Abuse= mean(Wife_Abuse, na.rm = T)) %>%
  select(D_INTERVIEW, Country, Wife_Abuse, avg_Wife_Abuse)


WA %>%  head(10)
```

D_INTERVIEW	Country	Wife_Abuse	avg_Wife_Abuse
36070000	36	1	1.241341
36070001	36	5	1.241341
36070002	36	1	1.241341
36070003	36	10	1.241341
36070004	36	1	1.241341
36070005	36	1	1.241341
36070006	36	1	1.241341
36070007	36	1	1.241341
36070008	36	1	1.241341
36070009	36	1	1.241341

After grouping, the `ungroup()` function restores the data frame to its original state, allowing operations on entire columns or enabling a switch to a different grouping. Additionally, you can group by multiple columns simultaneously. Let's explore a use case for `ungroup()` and grouping by multiple columns at once:

```
VO <- Cleaned_WVS %>%
              select(D_INTERVIEW, Country,Violence_otherPPL)  %>%
              filter(Country %in% c(36,554)) %>%
              group_by(Country) %>%
              mutate(avg_Violence_otherPPL = mean(Violence_otherPPL, na.rm = T)) %>%
              ungroup(Country) %>%
              mutate(total_Violence_otherPPL = mean(Violence_otherPPL, na.rm = T))


summary(VO)
```

```
##   D_INTERVIEW           Country      Violence_otherPPL avg_Violence_otherPPL
##  Min.   : 36070000   Min.   : 36.0   Min.   : 1.000    Min.   :1.596
##  1st Qu.: 36070717   1st Qu.: 36.0   1st Qu.: 1.000    1st Qu.:1.596
##  Median : 36071434   Median : 36.0   Median : 1.000    Median :1.596
##  Mean   :226846377   Mean   :226.8   Mean   : 1.615    Mean   :1.616
##  3rd Qu.:554070341   3rd Qu.:554.0   3rd Qu.: 1.000    3rd Qu.:1.650
##  Max.   :554071058   Max.   :554.0   Max.   :10.000    Max.   :1.650
##                                      NA's   :59
##  total_Violence_otherPPL
```

```
##  Min.   :1.615
##  1st Qu.:1.615
##  Median :1.615
##  Mean   :1.615
##  3rd Qu.:1.615
##  Max.   :1.615
##
```

Q1. How should we interpret *avg_Violence_otherPPL* and *total_Violence_otherPPL*? Why do thes

Q2. Which country has a higher average level of violence against other people?

Write your response here:

2.7 Joins

When working with multiple tables or data frames, you may need to combine them before applying other *dplyr* package functions. This process, known as joining or merging data, involves linking two datasets based on a shared key variable.

Merging datasets allows you to combine data that share some common observations (rows) but contain different variables (columns). A key variable ensures that R correctly matches rows from one dataset to the corresponding rows in the other. In some cases, multiple key variables may be used for merging.

For this exercise, we will join the WA and VO datasets.

```
head(WA, 10)
```

D_INTERVIEW	Country	Wife_Abuse	avg_Wife_Abuse
36070000	36	1	1.241341
36070001	36	5	1.241341
36070002	36	1	1.241341
36070003	36	10	1.241341
36070004	36	1	1.241341
36070005	36	1	1.241341
36070006	36	1	1.241341
36070007	36	1	1.241341
36070008	36	1	1.241341
36070009	36	1	1.241341

```
head(VO, 10)
```

D_INTERVIEW	Country	Violence_otherPPL	avg_Violence_otherPPL
36070000	36	10	1.595638
36070001	36	6	1.595638
36070002	36	1	1.595638
36070003	36	10	1.595638
36070004	36	3	1.595638
36070005	36	1	1.595638
36070006	36	1	1.595638
36070007	36	2	1.595638
36070008	36	1	1.595638
36070009	36	1	1.595638

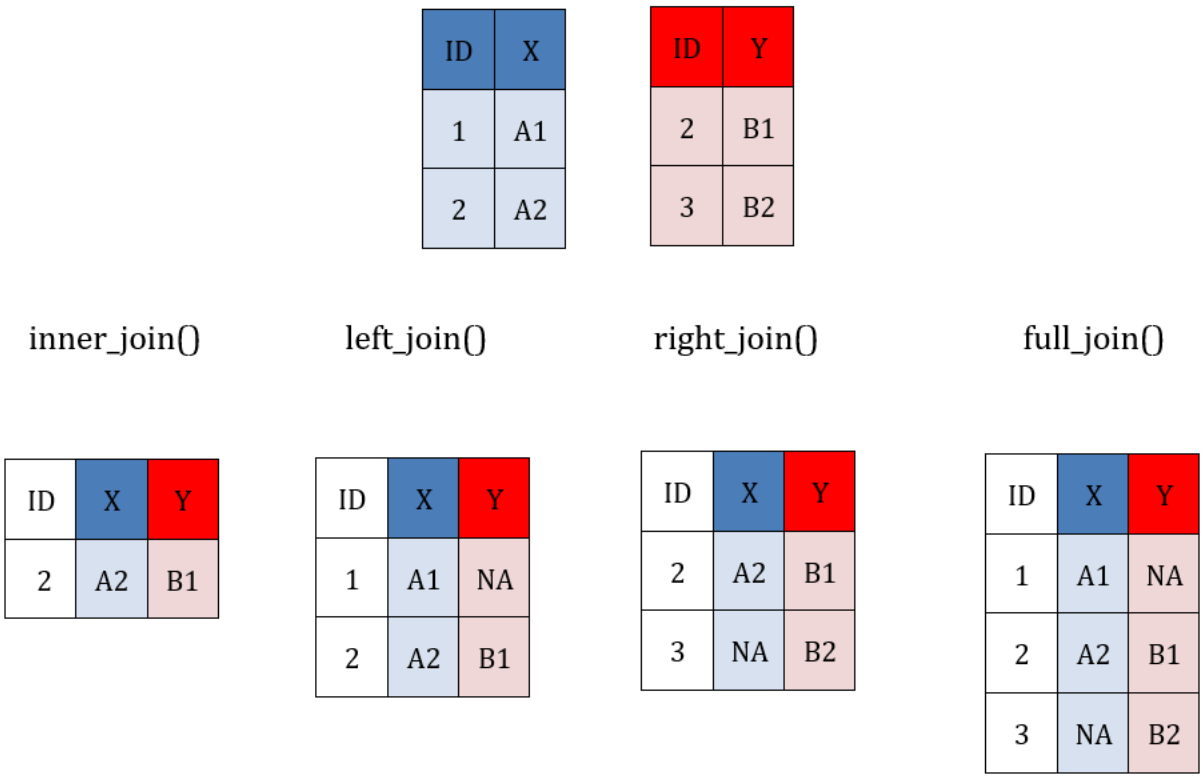
We aim to create a dataset on attitudes toward violence in Australia and New Zealand by merging two datasets that share two common columns: D_Interview and Country. The D_Interview variable serves as a unique identifier for individuals, making it the key variable for joining the datasets.

There are four main types of joins: `inner_join()`, `left_join()`, `right_join()`, and `full_join()`:

- inner_join(X, Y, by = "key"): keeps only rows that match in both X and Y, discarding any non-

matching rows.

- left_join(X, Y, by = "key"): keeps all rows from X (left dataset) and only matching rows from Y (right dataset). Missing values appear as NA for non-matching rows in Y.
- right_join(X, Y, by = "key"): keeps all rows from Y (right dataset) and only matching rows from X (left dataset). Non-matching rows in X appear as NA.
- full_join(X, Y, by = "key"): keeps all rows from both X and Y, filling in NA where there is no match.



Here's a simple example of how different join functions work in *dplyr*. We will merge the WA and VO datasets using D_INTERVIEW as the key variable, demonstrating inner, left, right, and full joins:

```
Inner <- WA %>% inner_join(VO, by = "D_INTERVIEW")
Inner
```

Data file (CSV, 3.9MB)

```
Left <- WA %>% left_join(VO, by = "D_INTERVIEW")
Left
```

Data file (CSV, 3.9MB)

```
Right<- WA %>% right_join(VO, by = "D_INTERVIEW")
Right
```

Data file (CSV, 3.9MB)

```
Full <- WA %>% full_join(VO, by = "D_INTERVIEW")
Full
```

Since both datasets share the same structure and observations, there is no difference across the four join types. To help you understand the differences in join outcomes, I have created a sample dataset for practice. Please refer to the appendix for details.

Additionally, if two datasets are already in the same order and have the same number of rows, you can use the `cbind()` funtion to bind columns together, similar to how rows were combined in the previous section. However, it is generally safer to merge datasets by matching on a key variable, as this ensures accuracy and allows for merging datasets of different sizes. In this case, since the datasets are already aligned, you can use the `cbind()` function as shown below:

```
cbind(WA, VO)
```

```
## New names:
## • `D_INTERVIEW` -> `D_INTERVIEW...1`
## • `Country` -> `Country...2`
## • `D_INTERVIEW` -> `D_INTERVIEW...5`
## • `Country` -> `Country...6`
```

Let's export the new WVS dataset (cleaned_WVS) in a csv file called **WVS** that you will be able to retrieve in the future.

```
#Export the dataset
write.csv(Cleaned_WVS, "WVS.csv", row.names = FALSE)
```

Regarding the remaining packages, we will cover *ggplot2* for data visualization in Chapter 9, and introduce *tidyr* when it becomes relevant to the content. If you are interested in exploring additional packages, you are encouraged to engage in self-study.

3. R Markdown

3.1 What is R Markdown?

R Markdown creates a file that contains text, code, and results from the .Rmd file. It allows you to present code alongside output (such as graphs and tables) with explanatory text. This is particularly useful for assignments and reports, ensuring that your workflow and results are well-documented. R Markdown uses Markdown, a simple markup language for creating documents with headers, images, links, and other formatting elements while keeping the plain text readable.

3.2 Download R Markdown

To use R Markdown in RStudio, you first need to install the R Markdown package. You can do this by running the following commands:

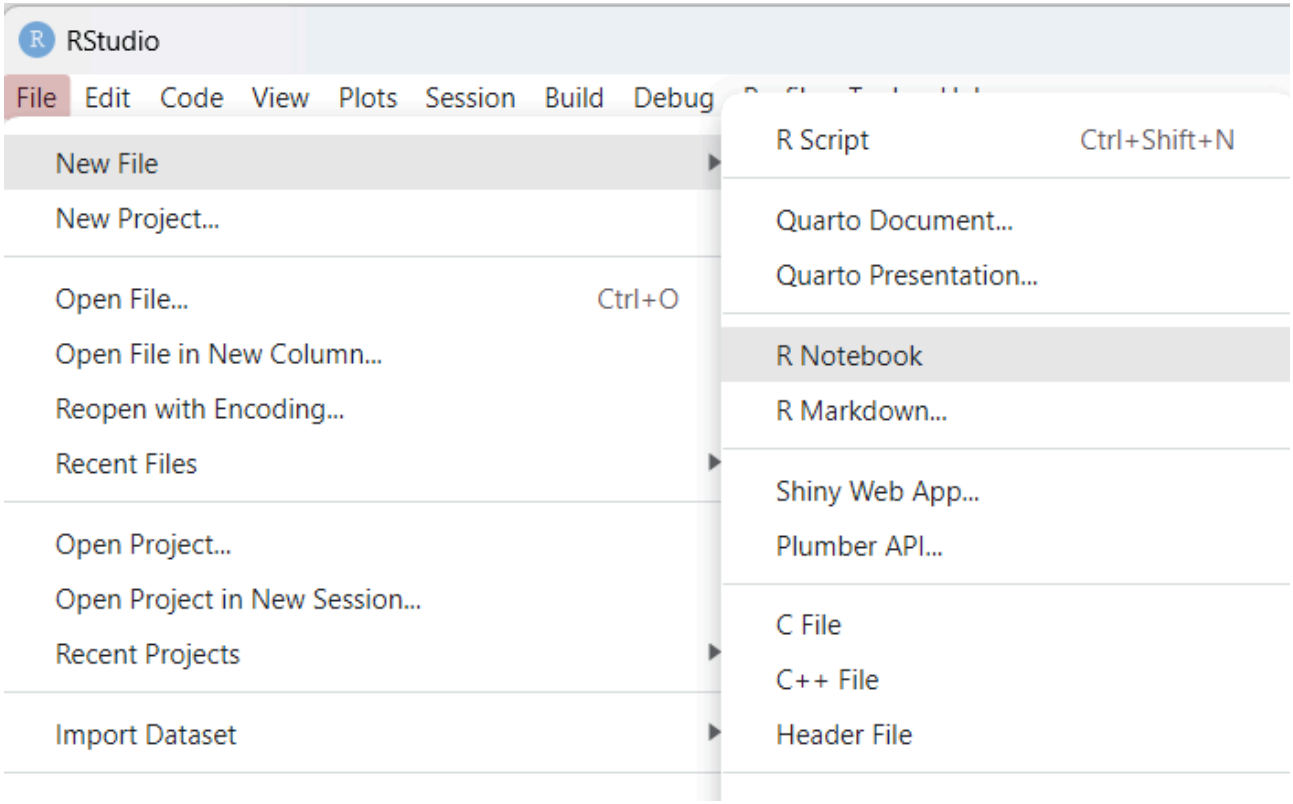
textbox shadenarrow

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

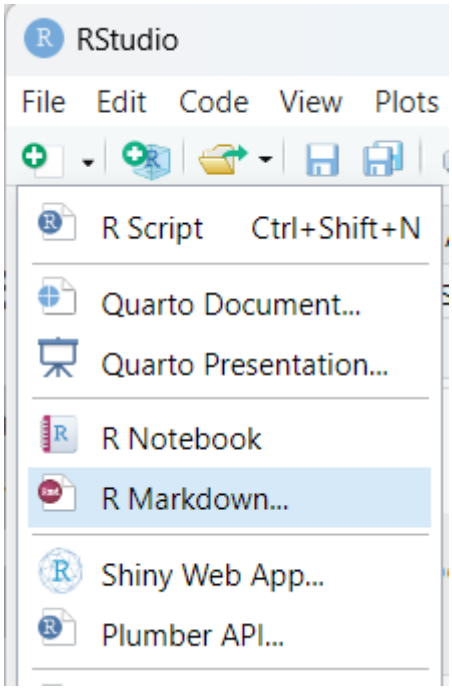
3.3 Create a R Markdown Document

To create a new R Markdown document in RStudio, follow one of these methods:

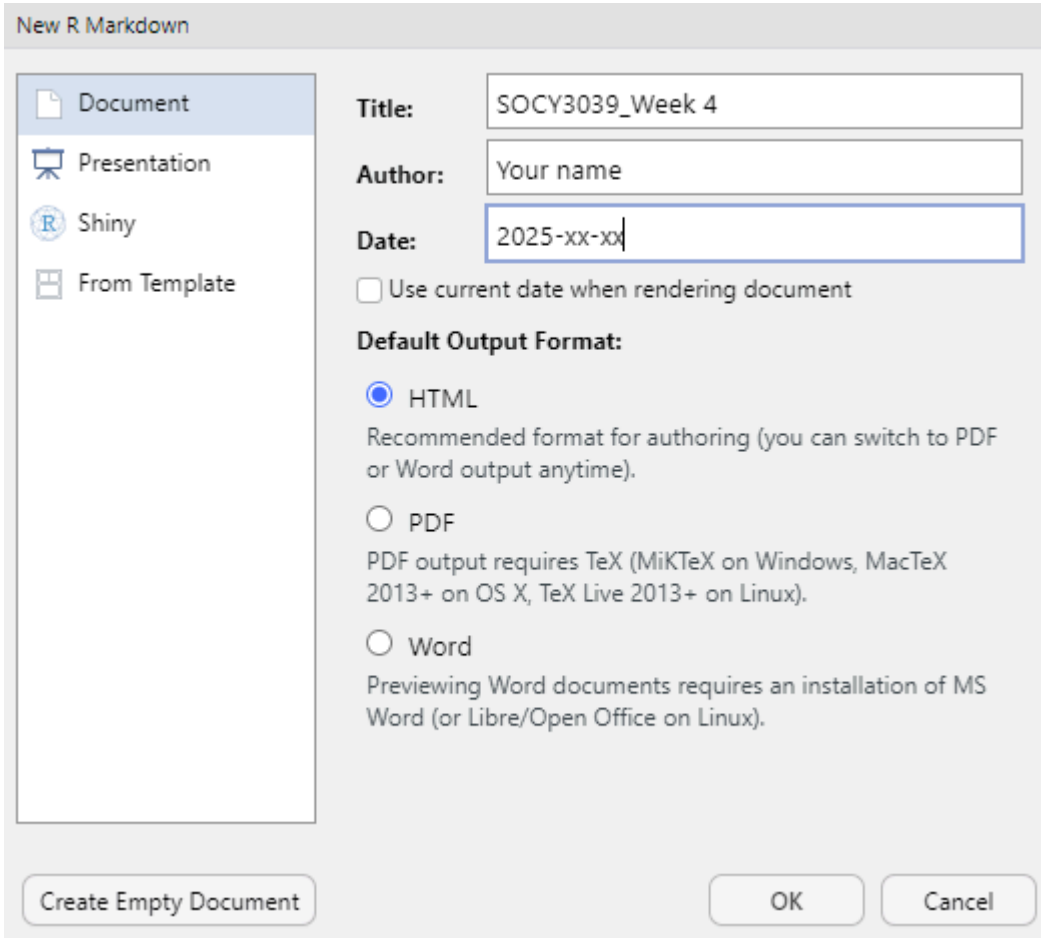
1. Select `File --> New File --> R Markdown...` in RStudio



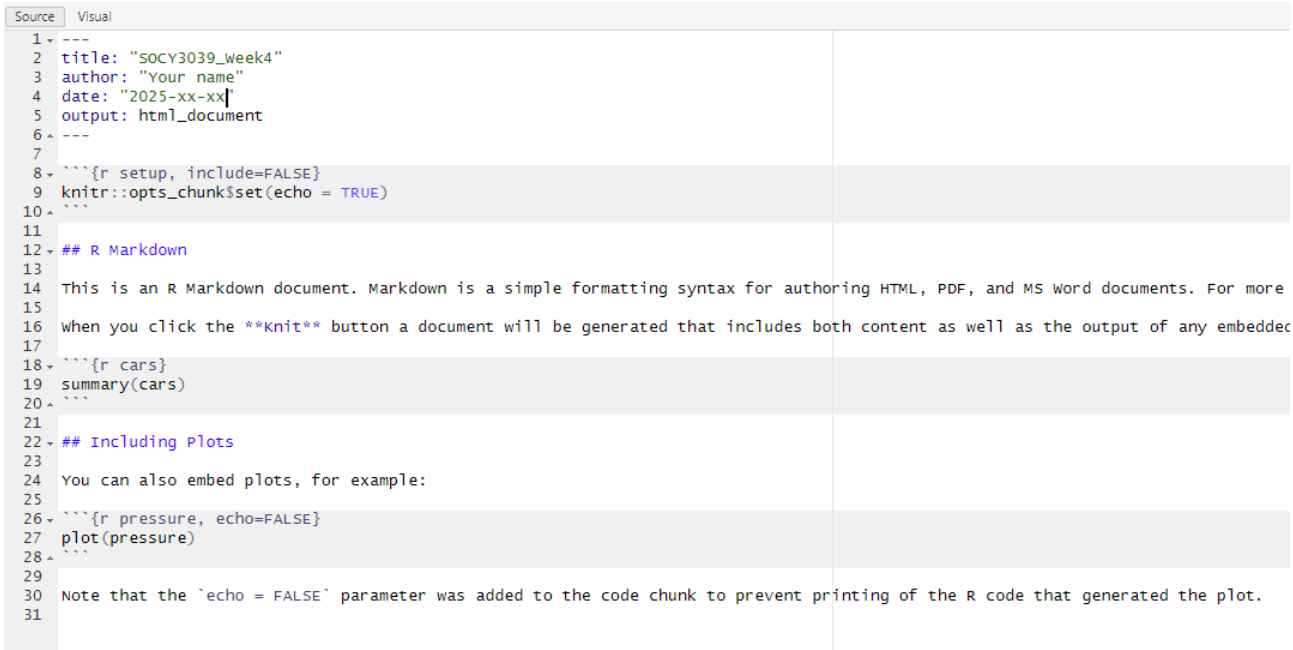
2. Click the green circle with a plus (+) under the File tab –> R Markdown...



Select *HTML* as the default output format. From now on, we'll be working with HTML files. Once you open a new R Markdown document, you can specify the title, author, and date in the document's YAML header at the top of the file as below.



When you create a new R Markdown document, it includes example content and instructions, including pre-written code chunks (grey boxes). Since we are creating our own script, we need to remove these default sections.



3.4 Markdown Syntax

Since R Markdown is a document, you can adjust text size, font styles, and other formatting using regular Markdown syntax. Once you knit the document, the output will be formatted according to the following rules:

```
# Header 1
```

Header 1

```
## Header 2
```

Header 2

*Note: When # is placed inside an R code chunk, it acts as a comment. In normal text, it defines header sizes.

```
**Bold**
```

Bold

```
*Italic*
```

Italic

```
This is `code` in text
```

This is `code` in text

```
- Unordered list item
```

- Unordered list item

```
1. Ordered list item
```

1. Ordered list item

For other formats, please visit the R Markdown website.

3.5 Code Chucks

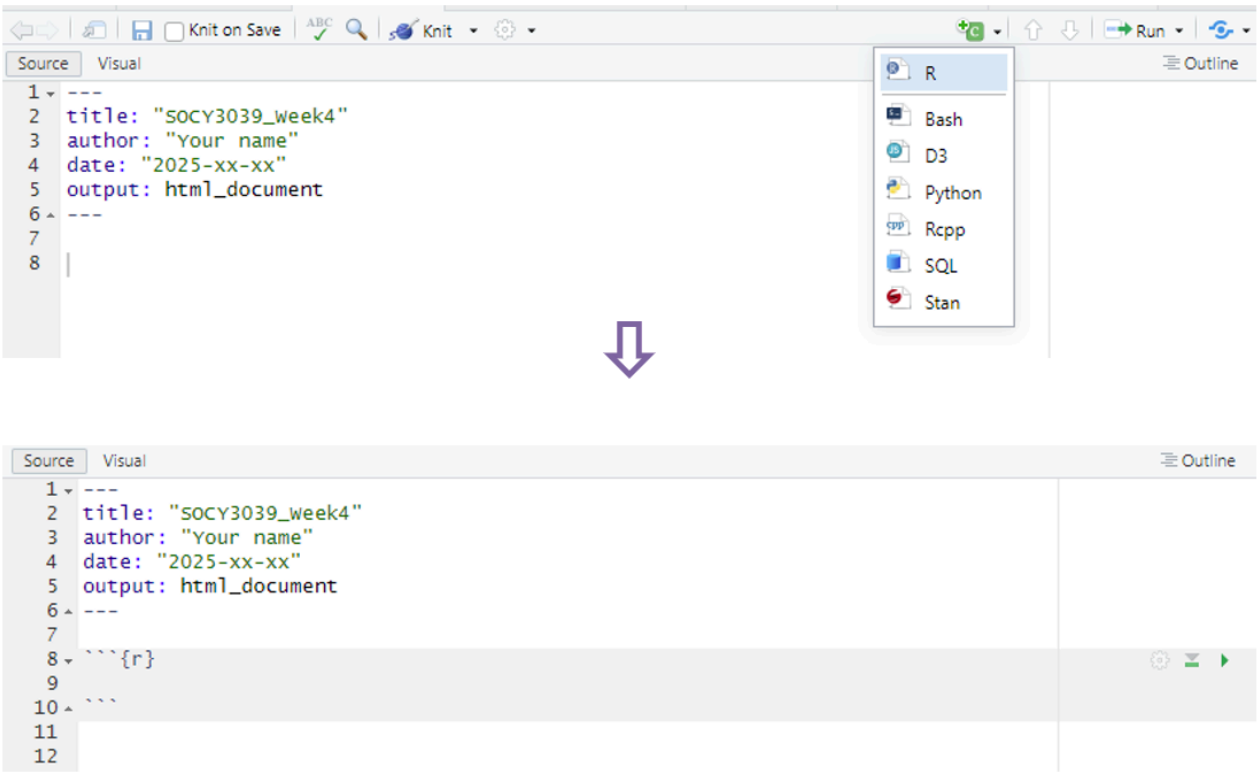
Below the YAML header in an R Markdown document, you have space to write your code, explanations, and outputs. Code in an .Rmd file should be enclosed within three backticks (“`), also known as code chunks.

Instead of manually typing code chunks, you can use shortcuts or the RStudio toolbar to insert them quickly.

1. Using keyboard shortcuts:

- Window/Linux: Ctrl + Alt + I
- Mac: Cmd + Option + I

2. Using the toolbar:



Once you open a new code chunk, you can write your R code and execute it using either the green play button or a keyboard shortcut (`Ctrl + Enter for Window/Linux`, `Cmd + Enter for Mac`).

```
Test <- c("R", "Markdown")

Test
```

```
## [1] "R"        "Markdown"
```

Preventing Errors in R Markdown

To avoid common errors when running code in R Markdown, keep these

points in mind:

1. Ensure all required packages are installed and loaded

```
install.packages("ggplot2") # Install if not already installed
library(ggplot2) # Load the package before using its functions
```

Note. After you have installed a package using the `install.packages()` function, you should comment out that line by placing a `#` in front of it (e.g., `#install.packages("packageName")`). This prevents the package from being reinstalled each time the document is knitted.

2. Check variable and dataset names.
3. Ensure variables are correctly spelled and exist in the dataset.
4. Use names(dataset) to list available variables.
5. Use the correct working directory.

3.6 Customizing Output in R Markdown Code Chunks

You can control how code and output appear in your R Markdown document by using chunk options inside the curly brackets `{}`.

If you want to display only the output (e.g., a plot) without showing the actual code, use `echo=FALSE`:

```
{r, echo = FALSE}
A <- c("a", "a", "b", "b")
B <- c(5, 10, 15, 20)
data <- data.frame(A, B)
print(data)
```

```
##   A  B
## 1 a  5
## 2 a 10
## 3 b 15
## 4 b 20
```

If you want to run a code chunk without displaying both the code and the output in the final .html file, use `include=FALSE`:

```
{r, include = FALSE}
Mean_A <- mean(data$A, na.rm  = T)
Mean_A
```

When loading packages or running code in RStudio, you might encounter warning messages such as: "Warning: package 'dplyr' was built under R version 3.4.4"

If you want to suppress warning messages in your R Markdown document, `use warning=FALSE` inside the code chunk:

```
{r, warning = FALSE}
library(dplyr)
```

```
library(dplyr)
```

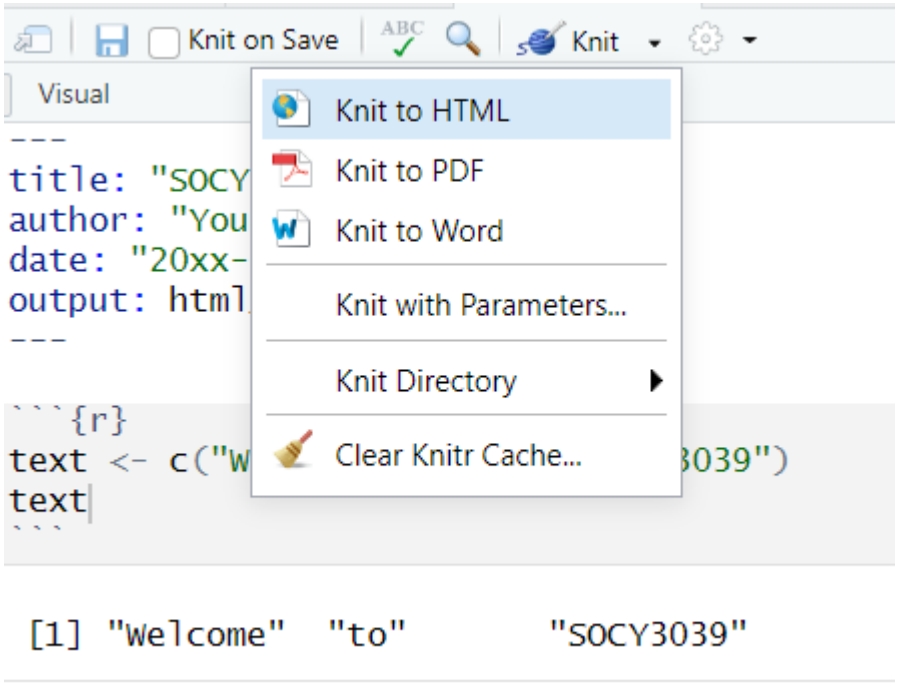
Lastly, R Markdown does not recognize objects or packages loaded in other R scripts. You must explicitly load all necessary packages and objects within the R Markdown document to ensure reproducibility. R Markdown provides various code chunk options to insert figures, tables, and format outputs effectively. For a detailed reference, visit the R Markdown website.

3.7 Knitting File

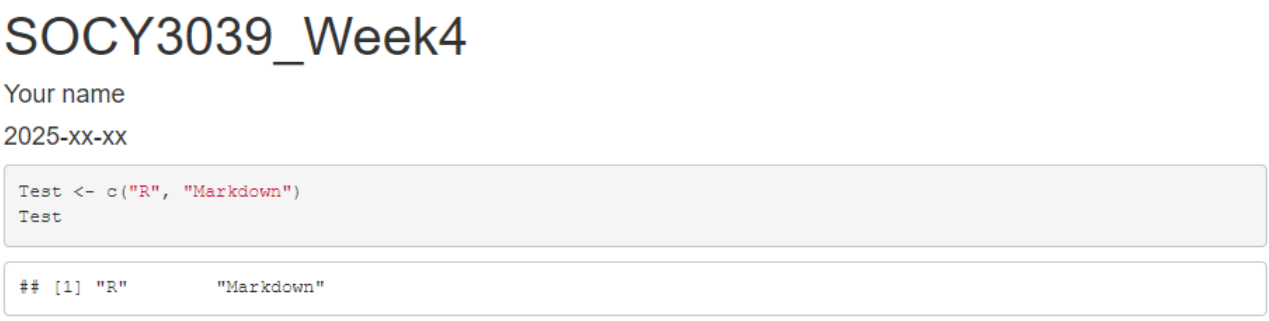
The most common formats for knitting an R Markdown document are: HTML, PDF, and Word.

For this course, we will focus on **HTML** and **Word** formats.

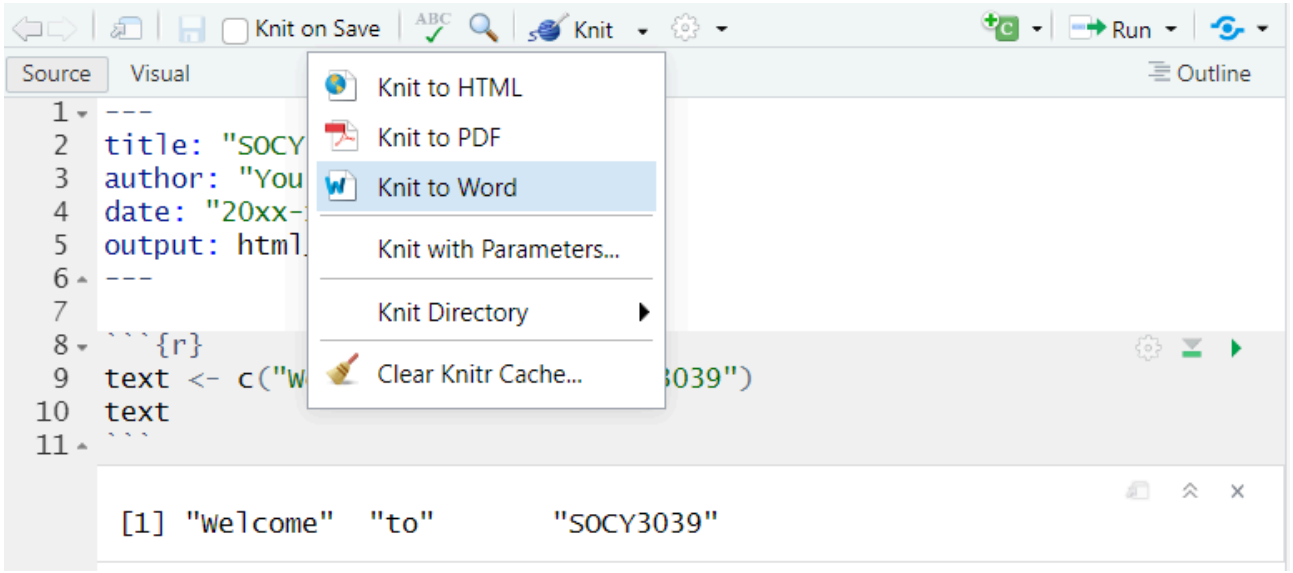
HTML is the default output format for R Markdown. You can easily knit your script by following these steps:



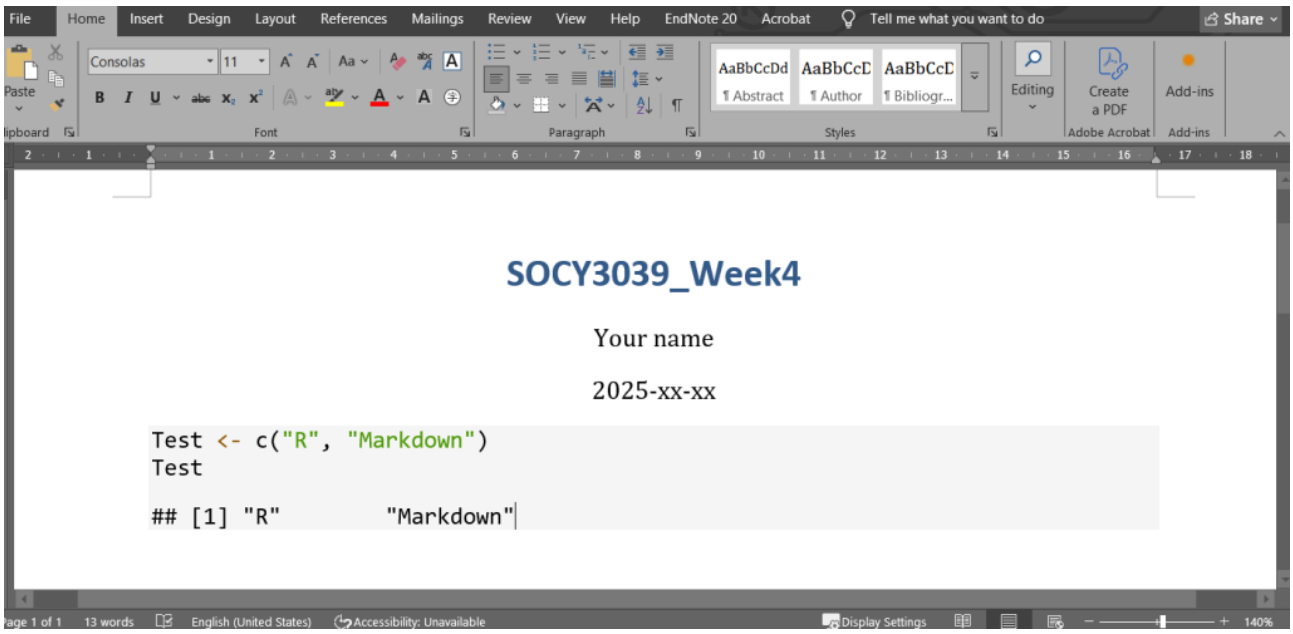
If there are no errors in the code, R Markdown will successfully knit the document, producing the output as shown below.



The second option is knitting to a Word document. Unlike HTML or PDF, a Word (.docx) file allows for easy editing. This makes it a more flexible choice, especially for assignments.



The output will be generated as a Word (.docx) document as below.



Once you complete all exercises, make sure to save your R script before closing R.

3.

DESCRIPTIVE STATISTICS AND BIVARIATE ANALYSES

This chapter focuses on performing basic descriptive statistics in R, including measures such as mean, median, mode, variance, frequency, and percentages. These fundamental statistics are essential for summarising and understanding the key characteristics of your data and form the basis for subsequent analysis. In this session we also introduce methods of bivariate analysis including the student t-test, chi-squared test, and correlation analysis. Bivariate analyses are used to explore the relationship between two variables.

1. Basic Descriptive Statistics

Descriptive statistics are used to summarise data and can be used to identify patterns of participant responses in a survey, such as the most frequently selected response or the average age of a sample. The primary purpose of descriptive statistics is to describe key patterns in the data and provide insights into the composition of the sample and the distribution of responses.

In this section, we will cover basic concepts such as measures of central tendency and dispersion. Let's set up the working directory and load the required packages.

```
#Set up the working directory
setwd("C:/Your Own Path/SOCYR")

#Load the required packages for chapter 3
library(dplyr) install.packages("psych") library(psych)
```

1.1 Measures of Central Tendency

1.1.1 Mean

The mean is the average value of a set of numbers. It is calculated by adding up all the values in a dataset and then dividing the sum by the total number of values. It is a widely used indicator, such as the average unemployment rate or crime rate in Australia. In R, it is simple to compute the mean using the `mean()` function.

```
mean(x)
```

*Before calculating the mean value, it is important to understand the **pipe operator %>%**. The pipe operator from the *dplyr* package allows the output of one function to be passed as the input to the next, enabling a streamlined sequence of operations. For example, there are two ways to compute the mean:

```
1. mean(data$x)
2. data %>% mean(x)
```

Let's examine the number of Bachelor of Arts (BA) programs offered at the University of Queensland (UQ) across different study areas. Specifically, we will compare the average number of programs available in STEM (Science, Technology, Engineering, and Mathematics) fields versus Non-STEM fields. The data used for this analysis is sourced from the UQ website.

```
library(dplyr)
```

```
#
 # Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':
##
## filter, lag
```

```
## The following objects are masked from 'package:base':
##
## intersect, setdiff, setequal, union
```

```
library(psych)
```

```
# Create a UQ dataset
UQ <- data.frame(Study.areas = c("Agriculture and animal sciences", "Architecture, design and urban planning",
                                 "Arts, humanities and social science", "Business and economics", "Communication, media and experience design", "Computer science and IT",  "Education", "Engineering", "Enviornment", "Health and medicine","Law", "Science and mathematics"),
                                   Num.of.Programs = c(32, 9, 55, 73, 47, 48, 11, 44, 50, 47,23,69),
                                   group = c("STEM", "Non-STEM", "Non-STEM", "Non-STEM", "Non-STEM",  "STEM", "Non-STEM", "STEM", "STEM", "STEM","Non-STEM", "STEM"))
#Architecture and urban planning can fall under either STEM or non-STEM categories, depending on the program's focus. In this analysis, they are classified as Non-STEM.

UQ %>%                    #The pipe (%>%) operator allows you to chain multiple operations together.
count(group)
```

group	n
Non-STEM	6
STEM	6

```
#The total number of programs of STEM fields and Non-STEM fields.
UQ %>%
   group_by(group) %>%
   summarise(sum = sum(Num.of.Programs))
```

group	sum
Non-STEM	218
STEM	290

```
#The average number of programs of STEM fields and Non-STEM fields.

UQ %>%
```

```
  group_by(group) %>%
  summarise(mean_Num.of.Programs = mean(Num.of.Programs)) %>%
  arrange(desc(mean_Num.of.Programs))
```

group	mean_Num.of.Programs
STEM	48.33333
Non-STEM	36.33333

The results indicate that, on average, STEM fields offer approximately 48 programs, while Non-STEM fields provide around 36 programs. This suggests that STEM fields have a larger number of programs compared to Non-STEM fields at the University of Queensland. Additionally, the `arrange()` function was used to sort rows in the dataset based on a specified variable (i.e., a column). You'll learn more about it in detail in Chapter 4.

1.1.2 Median

The median represents the middle value of a dataset when arranged in ascending order. If the dataset contains:

- an odd number of observations: the median is the exact middle value
- an even number of observations: the median is calculated as the average of the two middle values.

The median is often a more reliable measure of central tendency than the mean, as it is less affected by extreme values (outliers) or skewness. Returning to the UQ dataset, some study areas, such as 'Architecture, Design, and Urban Planning', fall significantly below their group average, while others, like 'Science and Mathematics', exceed it. These variations may distort the overall trend, making the median a more suitable measure than the mean in this case.

You can use the `median()` function to compute the median.

```
UQ %>%
  group_by(group) %>%
  summarise(median_Num.of.Programs = median(Num.of.Programs)) %>%
  arrange(desc(median_Num.of.Programs))
```

group	median_Num.of.Programs
STEM	47.5
Non-STEM	35.0

The median number of programs is 47 for STEM fields and 35 for Non-STEM fields.

The similarity between the mean and median is due to the small dataset size, with only 12 study areas, making extreme variations less likely. Additionally, there are no significant outliers, as no study area has an exceptionally high or low number of programs that would strongly skew the mean.

1.1.3 Mode

The mode represents the most frequently occurring value in a dataset. Unlike the mean and median, which measure central tendency numerically, the mode identifies the most common value within a distribution. It is particularly useful for analysing categorical data, where identifying the most frequent category or response is more relevant than calculating an average.

R does not have a built-in function for calculating the mode, but the `table()` function, which creates a frequency table of values can be used to check the most commonly occurring value.

Now, let's create a new variable, Postgrad_20Plus, which indicates whether each study area offers more than 20 postgraduate programs. The response is coded as "Yes" for study areas with more than 20 programs and "No" for those with fewer.

```
UQ$Postgrad_20Plus <- c("Yes", "No", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes",
"Yes", "No", "Yes")
table(UQ$Postgrad_20Plus)
```

```
##
##  No Yes
##   4   8
```

```
UQ
```

Study.areas	Num.of.Programs	group
Agriculture and animal sciences	32	STEM
Architecture, design and urban planning	9	Non-STEM
Arts, humanities and social science	55	Non-STEM
Business and economics	73	Non-STEM
Communication, media and experience design	47	Non-STEM
Computer science and IT	48	STEM
Education	11	Non-STEM
Engineering	44	STEM
Enviornment	50	STEM
Health and medicine	47	STEM
Law	23	Non-STEM
Science and mathematics	69	STEM

The results indicate that the mode for Postgrad_20Plus is 8, meaning that the most common response is "Yes." This suggests that the majority of study areas offer more than 20 postgraduate programs.

To calculate proportions for a categorical variable, the `prop.table()` function can be used. This function converts a frequency table into proportions.

```
prop.table(table(UQ$Postgrad_20Plus))
```

```
##
##        No       Yes
## 0.3333333 0.6666667
```

The output from the `prop.table()` function provides the proportion of each category within the variable. To execute these proportions as percentages, simply multiply the result by 100.

```
prop.table(table(UQ$Postgrad_20Plus))*100
```

```
##
##        No       Yes
## 33.33333 66.66667
```

1.2 Measures of Dispersion

Measures of dispersion (Range, Interquartile Range (IQR), Variance, and Standard Deviation) describe the spread of data points around a central value, such as the mean or median. They provide insights into how much the data varies (accessing the extent of variability within a dataset). To practice calculating measures of dispersion, we will use victimisation rates data in Australia (Victimisation rates (a), Selected personal crimes, 2008-09 to 2022-23). You can download it from the Australian Bureau of Statistics (ABS) website.

Once you have downloaded the dataset, run the following script to clean the data before analysis: getwd()

```
#Load the dataset
data <- read.csv("Victimisation rates(a), Selected personal crimes, 2008-09 to 2022-23.csv")
```

```
# Assign the column names
colnames(data) <- c("Year", "Physical_assault", "FacetoFace_threatened_assault",
"Non_facetoFace_threatened_assault", "Robbery", "Sexual_assault")# Remove the irrelevant row
data <- data[-c(1,17,18,19,20), ]

#Extract the first four characters from the Year column
data <- data %>% mutate(Year = substr(Year, 1, 4))

# Convert all crims columns to numeric (since all variables are character)
data[, 1:ncol(data)] <- lapply(data[, 1:ncol(data)], as.numeric)
head(data)
```

	Year	Physical_assault	FacetoFace_threatened_assault
2	2008	3.1	3.9
3	2009	2.9	3.1
4	2010	2.7	3.1
5	2011	3	3.3
6	2012	2.7	2.8
7	2013	2.3	2.7

1.2.1 Range

The range is calculated as the difference between the maximum and minimum values in a dataset: **Range = Maximum value − Minimum value**

If you want to find the range for a specific variable, such as physical assault, you can use the following R code:

```
# Calculate range
range_value <- max(data$Physical_assault) - min(data$Physical_assault)
range_value
```

```
## [1] 1.4
```

If you want to calculate the range for all variables in the dataset, you can use the following R code:

```
# Compute range (max - min) for each numeric variable
range_values <- apply(data[, 2:ncol(data)], 2, function(x) max(x) - min(x)) #This applies a function
across all numeric columns (excluding the Year column). It calculates max - min for each column to
get the range.
range_values
```

```
##                   Physical_assault     FacetoFace_threatened_assault
##                                1.4                               1.7
## Non_facetoFace_threatened_assault                            Robbery
##                                0.5                               0.4
##                     Sexual_assault
##                                0.3
```

1.2.2 Inter Quartile Range (IQR)

The interquartile range (IQR) represents the difference between the third quartile (Q3, or the 75th percentile) and the first quartile (Q1, or the 25th percentile): **IQR = Q3 − Q1**

This measure captures the spread of the middle 50% of the data, providing a robust indicator of variability that is less sensitive to outliers compared to the range.

To calculate the IQR for a specific variable, such as physical assault, in R:

```
#One variable
IQR_Physical_assault  <- IQR(data$Physical_assault)
IQR_Physical_assault
```

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

```
#All numerical variables in the dataset

IQR_values <- apply(data[, 2:ncol(data)], 2, IQR) #Applies the IQR() function column-wise
(2:ncol means applying it to each column starting at second column; 2, IQR means calcu-
late IQR using column (2)).
IQR_values
```

```
##                   Physical_assault     FacetoFace_threatened_assault
##                                0.50                              0.45
## Non_facetoFace_threatened_assault                            Robbery
##                                0.25                              0.10
##                     Sexual_assault
##                                0.10
```

1.2.3 Variance and Standard Deviation

Variance measures the average squared deviation of each data point from the mean, providing an indication of data spread. Standard deviation, the square root of variance, expresses this dispersion in the same unit as the original data, making it easier to interpret.

To compute variance and standard deviation for a specific variable, such as physical assault:

```
#One variable
var(data$Physical_assault)
```

```
## [1] 0.1631429
```

```
sd(data$Physical_assault)
```

```
## [1] 0.4039095
```

```
#All numerical variables in the dataset
dispersion <- data.frame(SD = apply(data[, 2:ncol(data)], 2, sd),
                         Variance = apply(data[, 2:ncol(data)], 2, var) )
```

Instead of calculating each measure of central tendency and dispersion separately, you can use the `summary()` or `describe()` function to obtain key statistics in one step.

The `summary()` function provides essential descriptive statistics, including the minimum, first quartile (Q1), median, mean, third quartile (Q3), and maximum for each numeric variable:

```
summary(data)
```

```
##       Year      Physical_assault FacetoFace_threatened_assault
##  Min.   :2008   Min.   :1.70     Min.   :2.20
##  1st Qu.:2012   1st Qu.:2.20     1st Qu.:2.50
##  Median :2015   Median :2.40     Median :2.60
##  Mean   :2015   Mean   :2.42     Mean   :2.74
##  3rd Qu.:2018   3rd Qu.:2.70     3rd Qu.:2.95
##  Max.   :2022   Max.   :3.10     Max.   :3.90
##  Non_facetoFace_threatened_assault    Robbery       Sexual_assault
##  Min.   :0.7000                     Min.   :0.2000   Min.   :0.2000
##  1st Qu.:0.8000                     1st Qu.:0.3000   1st Qu.:0.3000
##  Median :1.0000                     Median :0.4000   Median :0.3000
##  Mean   :0.9467                     Mean   :0.3667   Mean   :0.3533
##  3rd Qu.:1.0500                     3rd Qu.:0.4000   3rd Qu.:0.4000
##  Max.   :1.2000                     Max.   :0.6000   Max.   :0.5000
```

The `describe()` function from the *psych* package offers a more detailed summary, including additional statistics such as variance, standard deviation, skewness, and kurtosis:

```
describe(data)
```

	vars	n	mean	sd	median
Year	1	15	2015	4.47213595	2015
Physical_assault	2	15	2.42	0.40390947	2.4
FacetoFace_threatened_assault	3	15	2.74	0.46260134	2.6
Non_facetoFace_threatened_assault	4	15	0.9466667	0.1641718	1
Robbery	5	15	0.3666667	0.08997354	0.4
Sexual_assault	6	15	0.3533333	0.09154754	0.3

2. Bivariate Analyses

2.1 Student t-test

A t-test is a statistical test used to compare the means of two groups to determine if the difference between them is statistically significant. It helps assess whether the observed differences are due to actual effects or just random chance.

The basic structure of the t.test() function in R is as follow:

t.test

```
t.test(x, y,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

2.1.1 One-Sample T-Test

A one-sample t-test is used to compare the mean of a sample to a known theoretical or hypothetical mean (μ) to determine if there is a significant difference. For example, suppose the average percentage of face-to-face threatened assault in Australia from 2008 to 2023 is hypothetically 3, and we want to test whether the actual mean differs from this value.

Before performing the one-sample t-test, you should complete preliminary test to check one-sample t-test assumptions.

- Step 1: Check if the Sample Size is Large A sample is considered large if $n \geq 30$ (based on a common rule of thumb).

In this case, n < 30, so the sample is not large enough to rely on the central limit theorem.

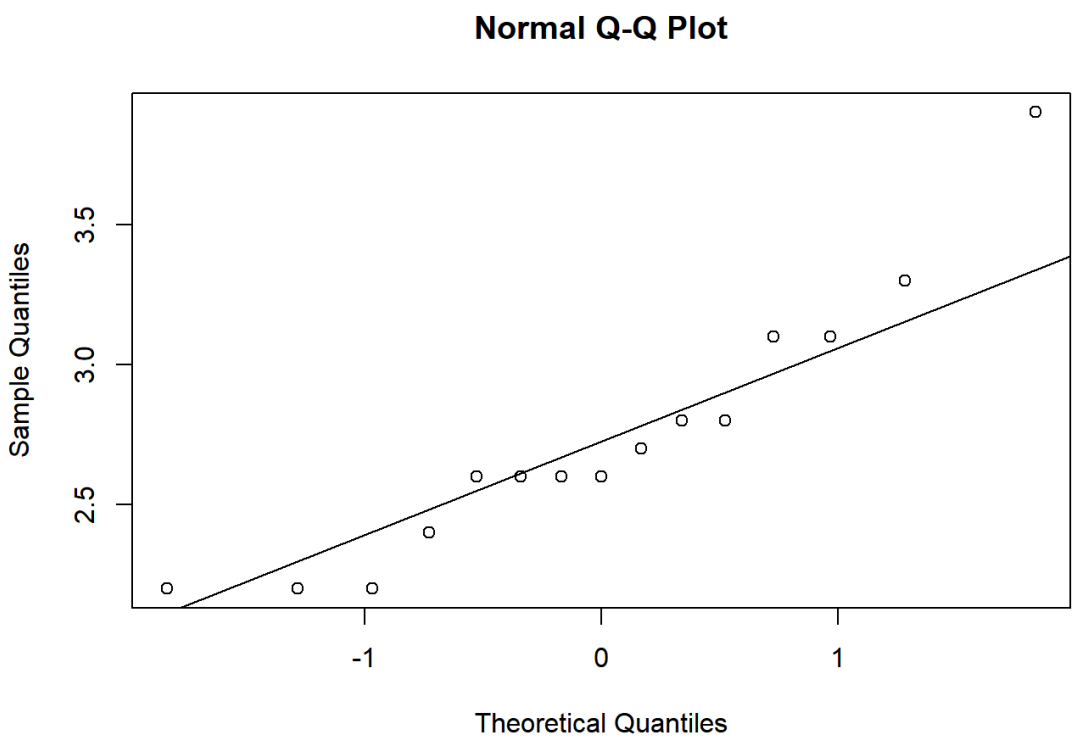
- Step 2: Check for Normality

Since the sample size is small, we need to verify whether the data follows a normal distribution, which is a key assumption for the one-sample t-test. This can be done using:

- QQ Plot (Quantile-Quantile Plot) – A graphical method to assess normality.
- Shapiro-Wilk Test – A formal statistical test for normality.

To visualise normality using a QQ plot:

```
qqnorm(data$FacetoFace_threatened_assault)
qqline(data$FacetoFace_threatened_assault)
```



Based on the QQ plot, the sample appears to follow a normal distribution, with some deviations at the tails, particularly at the upper end. However, these deviations are not extreme, so normality can be reasonably

assumed. To confirm this assumption, we will perform the **Shapiro-Wilk** test, which provides a more precise statistical check for normality:

```
#Shapiro-Wilk normality test
shapiro.test(data$FacetoFace_threatened_assault)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  data$FacetoFace_threatened_assault
## W = 0.90222, p-value = 0.1029
```

Since the p-value from the Shapiro-Wilk test is greater than 0.05 (p= 0.1), we cannot reject the null hypothesis, which means that the data does not significantly deviate from a normal distribution. In other words, the assumption of normality holds, allowing us to proceed with the one-sample t-test.

To perform a one-sample t-test, use the `t.test()` function as follows:

```
#One Sample t-test

t.test(data$FacetoFace_threatened_assault, mu=3) #x: a numeric vector containing your data
 values, mu: the theoretical mean. Default is 0.
```

```
##
##  One Sample t-test
##
## data:  data$FacetoFace_threatened_assault
## t = -2.1768, df = 14, p-value = 0.0471
## alternative hypothesis: true mean is not equal to 3
## 95 percent confidence interval:
##  2.48382 2.99618
## sample estimates:
## mean of x
##      2.74
```

Since the p-value of the test is 0.0471, which is less than the significance level ($\alpha = 0.05$), we reject the null

hypothesis. This means that the mean percentage of face-to-face threatened assault is significantly different from 3 at a 95% confidence level. Thus, we have statistical evidence to conclude that the actual mean percentage of face-to-face threatened assault in Australia from 2008 to 2023 differs from the 3.

2.1.2 Two Sample T-tests

NOTE: There are 2 types of two-sample t-tests:

1. **Independent t-tests**: is used to compare the mean score of two different groups of people or conditions. For example, the mean test scores of a class graduating in 2024 compared to a class graduating in 2025.
2. **Paired t-tests**: compare the mean score of the same people across different measures. For example, the mean scores of the same class of students for maths compared to English.

You can also compare the means of two independent groups using an independent (unpaired) t-test. For example, suppose face-to-face interactions decreased during the COVID-19 pandemic, potentially reducing the likelihood of face-to-face threatened assault encounters. To test this hypothesis, we assess whether the mean percentage of face-to-face threatened assaults during the pandemic differs from that observed in the pre-pandemic period.

Because we are comparing two independent groups, the pre-pandemic period and the pandemic period, we apply an independent-samples (unpaired) t-test to determine whether their means differ statistically.

Performing an Independent Samples t-Test in R

First, we split the data into two groups:

- Pre-pandemic (e.g., 2008–2019)
- Pandemic (e.g., 2020–2023)

```
#One Sample t-test
# Add a new column indicating the Covid-19 period
Pre_pandemic <- data %>%
  filter(Year < 2020) %>%
  mutate(Covid = "Pre_pandemic")
```

```
Pandemic <- data %>%
  filter(Year >= 2020) %>%
  mutate(Covid = "Pandemic")

# Combine both datasets
covid_data <- bind_rows(Pre_pandemic, Pandemic)
covid_data
```

Year	Physical_assault	FacetoFace_threatened_assault
2008	3.1	3.9
2009	2.9	3.1
2010	2.7	3.1
2011	3	3.3
2012	2.7	2.8
2013	2.3	2.7
2014	2.1	2.6
2015	2.4	2.6
2016	2.4	2.6
2017	2.4	2.6
2018	2.4	2.8
2019	2.3	2.4
2020	2	2.2
2021	1.9	2.2
2022	1.7	2.2

Before performing an independent (unpaired) t-test, it is important to check whether the test assumptions are met.

- Step 1: Are the Two Samples Independent?

Yes: The samples from the two groups are independent, as they come from different time periods and are not related.

- Step 2: Check for Normality

Since the sample size is less than 30, we need to check whether the data follows a normal distribution using the Shapiro-Wilk test. The results show $p < 0.05$, indicating that the normality assumption is not met. Despite this, we will proceed with the two-sample t-test for demonstration purposes.

- Step 3: Check for Equal Variances (Homogeneity of Variance)

We use the F-test to test whether the two groups have equal variances.

To test for homogeneity in variances, use the `var.test()` function as follows:

```
#F test to compare two variances
var.test(FacetoFace_threatened_assault ~ Covid, data = covid_data)
```

```
##
##  F test to compare two variances
##
## data:  FacetoFace_threatened_assault by Covid
## F = 0, num df = 2, denom df = 11, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0 0
## sample estimates:
## ratio of variances
##                  0
```

The F-test result (p-value < 0.05) suggests that the variances are significantly different, meaning that the assumption of equal variances does not hold. Since the standard Student's t-test assumes equal variances, we should instead use **Welch's** t-test, which adjusts for unequal variances.

To perform Welch's t-test, use the `t.test()` function as follow:

```
#Two Sample t-test
t.test(FacetoFace_threatened_assault ~ Covid, data = covid_data)
```

```
##
##  Welch Two Sample t-test
```

```
##
## data:  FacetoFace_threatened_assault by Covid
## t = -5.6225, df = 11, p-value = 0.000155
## alternative hypothesis: true difference in means between group Pandemic and group
Pre_pandemic is not equal to 0
## 95 percent confidence interval:
##  -0.9392363 -0.4107637
## sample estimates:
##     mean in group Pandemic mean in group Pre_pandemic
##                      2.200                      2.875
```

The outcome shows a significant difference in means between the two periods (p-value < 0.05). The confidence interval provides a range for the true mean difference.

*Note:

Although the independent two-sample t-test is statistically significant, it is important to acknowledge that the normality assumption was not met (Shapiro-Wilk test: $p < 0.05$). This means that the results should be interpreted with caution, and a non-parametric alternative such as the Mann-Whitney U test (Wilcoxon rank-sum test) may be more appropriate for non-normally distributed data.

To do this, you can use the `wilcox.test()` function. The structure is as follows:

```
wilcox.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
            conf.int = FALSE, conf.level = 0.95, …)
```

There are several other types of t-tests, including the paired two-sample t-test and the proportion t-test, each designed for different
analytical scenarios.

- Paired Two-Sample t-Test: Used when comparing two related (dependent) samples, such as pre-test and post-test measurements from the same individuals.

```
t.test(x, y, paired = TRUE, alternative = "two.sided"
```

- Proportion t-Test: Used to compare proportions between two groups, often applied in categorical data analysis. For a more in-depth understanding of different t-tests in R, refer to a statistical textbook or relevant online resources.

```
prop.test(x, n, p = NULL, alternative = "two.sided",
 correct = TRUE)
```

2.2 Correlations

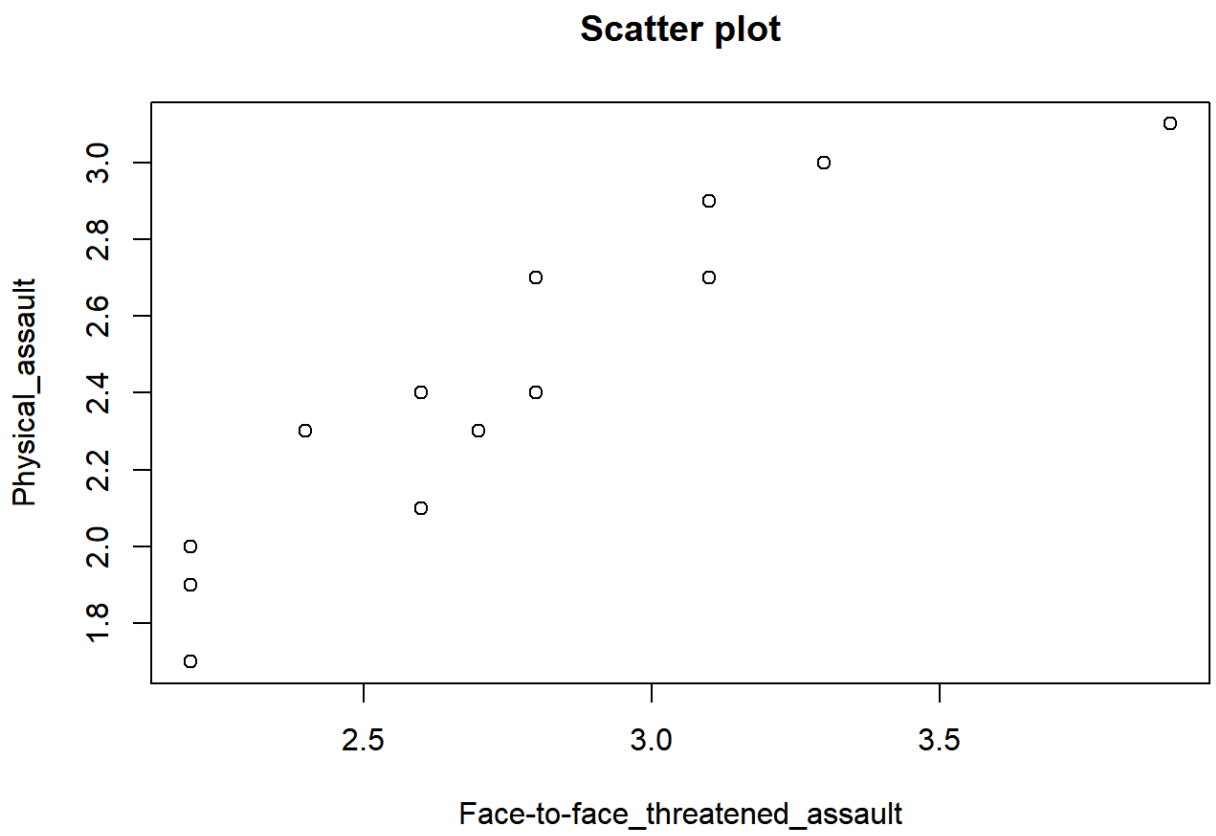
While means, medians, and standard deviations describe a single variable, and t-tests compare group means, they do not assess the strength or direction of relationships between continuous variables. Correlation addresses this gap by quantifying how strongly two continuous variables are associated and in what direction.

2.2.1 Scatter Plots

Correlations measure the relationship between two variables, they can be easily visualised using a scatterplot, where one variable is placed on the x-axis and the other on the y-axis. This graphical representation helps identify patterns, trends, and the strength of the association between the variables.

To perform a scatter plot, use the `plot(x, y)` function as follow:

```
#Create a scatter plot

plot(data$FacetoFace_threatened_assault, data$Physical_assault, main = "Scatter plot",
xlab =
"Face-to-face_threatened_assault", ylab = "Physical_assault")
```



Q. Based on the plot outcome, describe the pattern and what it suggests about the relationship between x and y in the data.

Write your response here:

Since we have observed the graphical relationship between the two variables, we can now compute the correlation coefficient to quantify the strength and direction of their association.

Correlations can be calculated using the `cor()` function:

```
#Calculating correlation coefficients between two variables

cor(data$FacetoFace_threatened_assault, data$Physical_assault)
```

```
## [1] 0.9167063
```

If you want to calculate the correlation between multiple variables in a dataset, you can generate a correlation matrix.

```
cor(data) # correlation coefficients between every pair of variables in the x dataset
```

```
##                                       Year Physical_assault
## Year                                1.0000000       -0.8897243
## Physical_assault                   -0.8897243        1.0000000
## FacetoFace_threatened_assault      -0.8838734        0.9167063
## Non_facetoFace_threatened_assault -0.7004722        0.6743164
## Robbery                            -0.7810788        0.8255101
## Sexual_assault                      0.7502029       -0.7070047
##                                   FacetoFace_threatened_assault
## Year                                                 -0.8838734
## Physical_assault                                      0.9167063
## FacetoFace_threatened_assault                         1.0000000
## Non_facetoFace_threatened_assault                     0.7260789
## Robbery                                               0.8752265
## Sexual_assault                                       -0.5936918
##                                   Non_facetoFace_threatened_assault    Robbery
## Year                                                      -0.7004722 -0.7810788
## Physical_assault                                           0.6743164  0.8255101
## FacetoFace_threatened_assault                              0.7260789  0.8752265
## Non_facetoFace_threatened_assault                          1.0000000  0.6931158
## Robbery                                                    0.6931158  1.0000000
## Sexual_assault                                            -0.6526836 -0.4624973
##                                   Sexual_assault
## Year                                   0.7502029
## Physical_assault                      -0.7070047
## FacetoFace_threatened_assault         -0.5936918
## Non_facetoFace_threatened_assault     -0.6526836
## Robbery                               -0.4624973
## Sexual_assault                         1.0000000
```

Q. Based on the results, describe what each correlation coefficient indicates.

Write your response here:

One caveat of using the `cor()` function is that if a variable contains missing values, the default behaviour is to return `NA`. To avoid this, you can use **pairwise** Pairwise deletion ignores missing values on a per-variable basis, meaning it does not remove entire rows but instead uses all available data for each pair of variables.

```
cor(data, use = "pairwise.complete.obs")
```

```
##                                       Year Physical_assault
## Year                                1.0000000       -0.8897243
## Physical_assault                   -0.8897243        1.0000000
## FacetoFace_threatened_assault      -0.8838734        0.9167063
## Non_facetoFace_threatened_assault -0.7004722        0.6743164
## Robbery                            -0.7810788        0.8255101
## Sexual_assault                      0.7502029       -0.7070047
##                                   FacetoFace_threatened_assault
## Year                                                 -0.8838734
## Physical_assault                                      0.9167063
## FacetoFace_threatened_assault                         1.0000000
## Non_facetoFace_threatened_assault                     0.7260789
## Robbery                                               0.8752265
## Sexual_assault                                       -0.5936918
##                                   Non_facetoFace_threatened_assault    Robbery
## Year                                                      -0.7004722 -0.7810788
## Physical_assault                                           0.6743164  0.8255101
## FacetoFace_threatened_assault                              0.7260789  0.8752265
## Non_facetoFace_threatened_assault                          1.0000000  0.6931158
## Robbery                                                    0.6931158  1.0000000
## Sexual_assault                                            -0.6526836 -0.4624973
##                                   Sexual_assault
## Year                                   0.7502029
```

```
## Physical_assault                      -0.7070047
## FacetoFace_threatened_assault         -0.5936918
## Non_facetoFace_threatened_assault     -0.6526836
## Robbery                               -0.4624
```

To test whether the correlation between two variables is statistically significant in R, use the `cor.test()` function. This function provides both the correlation coefficient and a p-value to determine statistical significance.

```
cor.test(data$Physical_assault, data$Robbery, use = "pairwise.complete.obs")
```

```
##
##  Pearson's product-moment correlation
##
## data:  data$Physical_assault and data$Robbery
## t = 5.2737, df = 13, p-value = 0.0001506
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.542775 0.940188
## sample estimates:
##       cor
## 0.8255101
```

Hypotheses for Correlation Test

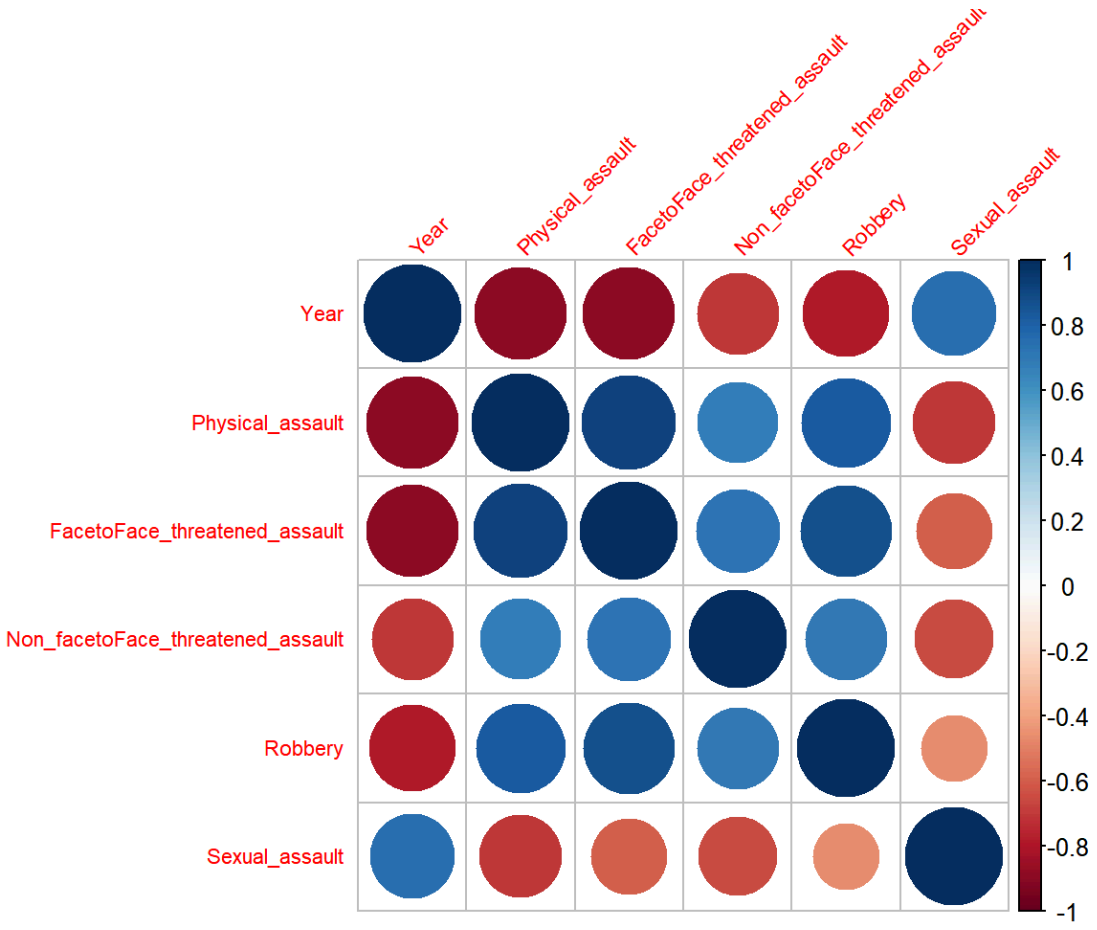
- Null Hypothesis (H_0): The correlation coefficient between the two variables is 0, meaning there is no relationship between them.
- Alternative Hypothesis (H_1): The correlation coefficient is not 0, indicating that a statistically significant relationship exists between the two variables.

Since the p-value is smaller than 0.05, we reject the null hypothesis, indicating that the true correlation is not zero, and the two variables have a statistically significant relationship.

Another way to analyse correlations is by using a **correlogram**, which provides a graphical representation of a correlation matrix. In R, this can be done using the *corrplot* package.

```
#install.packages("corrplot")  # Install if not already installed
library(corrplot)
```

```
# Compute correlation matrix
cor_matrix <- cor(data, use = "pairwise.complete.obs")
corrplot(cor_matrix, method = "circle", tl.cex = 0.7, tl.srt = 45)
# Adjust text size & Rotate labels
```



2.3 Cross-tabulation and Chi-squared Test

So far, we have explored bivariate analyses for continuous variables using correlation. Now, we turn to bivariate analyses for categorical variables. The Chi-Square Test of Independence is used to analyse frequency tables (contingency tables) that summarise the relationship between two categorical variables. This test determines whether there is a statistically significant association between the categories of the two variables.

In this example, we created a fictional dataset by categorising physical assault levels based on the mean value

(2.42). Cases above the mean were labelled as "High," while the rest were labelled as "Low." To address the low number of observations, I randomly increased the total to 50.

Before performing the Chi-Square Test, it is helpful to first create a contingency table, which summarises the frequency distribution of two categorical variables (Pandemic and Physical assault level).

The simplest way to create a contingency table in R is by using the `table()` function:

```
# Create contingency table


chi2_table <- matrix(c(20, 10, 10, 10), nrow = 2, byrow = TRUE,
                     dimnames = list(c("Pre-pandemic", "Pandemic"),
                                     c("High_Physical_Assualt", "Low_Physical_Assualt")))
chi2_table
```

```
##              High_Physical_Assualt Low_Physical_Assualt
## Pre-pandemic                    20                   10
## Pandemic                        10                   10
```

You can also use the `addmargins()` function to include row and column totals in a contingency table. This helps provide a clearer summary of the frequency distribution across categories.

```
addmargins(chi2_table)
```

```
##              High_Physical_Assualt Low_Physical_Assualt Sum
## Pre-pandemic                    20                   10  30
## Pandemic                        10                   10  20
## Sum                             30                   20  50
```

If you want proportions instead of row counts or column counts:

```
prop.table(chi2_table, margin = 1) #row-wise
```

```
##              High_Physical_Assualt Low_Physical_Assualt
## Pre-pandemic             0.6666667            0.3333333
## Pandemic                 0.5000000            0.5000000
```

```
prop.table(chi2_table, margin = 2) #column-wise
```

```
##              High_Physical_Assualt Low_Physical_Assualt
## Pre-pandemic             0.6666667                  0.5
## Pandemic                 0.3333333                  0.5
```

The Chi-square test examines whether physical assault level and pandemic period are statistically significantly associated in the contingency table and can be used to "test" hypotheses.

- Null hypothesis (H0): Physical assault level is independent of the pandemic period.
- Alternative hypothesis (H1): Physical assault level is associated with the pandemic period.

The chi-square statistic can be computed using the function the `chisq.test()` as follow:

```
chisq.test(chi2_table)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  chi2_table
## X-squared = 0.78125, df = 1, p-value = 0.3768
```

Important! Before finishing this session, let's save the new **Victim_rate** dataset so that it can be retrieved for future analyses.

```
#Export the dataset
write.csv(data, "Victim_rate.csv", row.names = FALSE)
```

The p-value (0.3768) is greater than the conventional alpha level of 0.05. Therefore, we cannot reject the

null hypothesis. This means there is no statistically significant association between the pandemic period and the level of physical assault in this sample.

Before closing R, be sure to save your script.

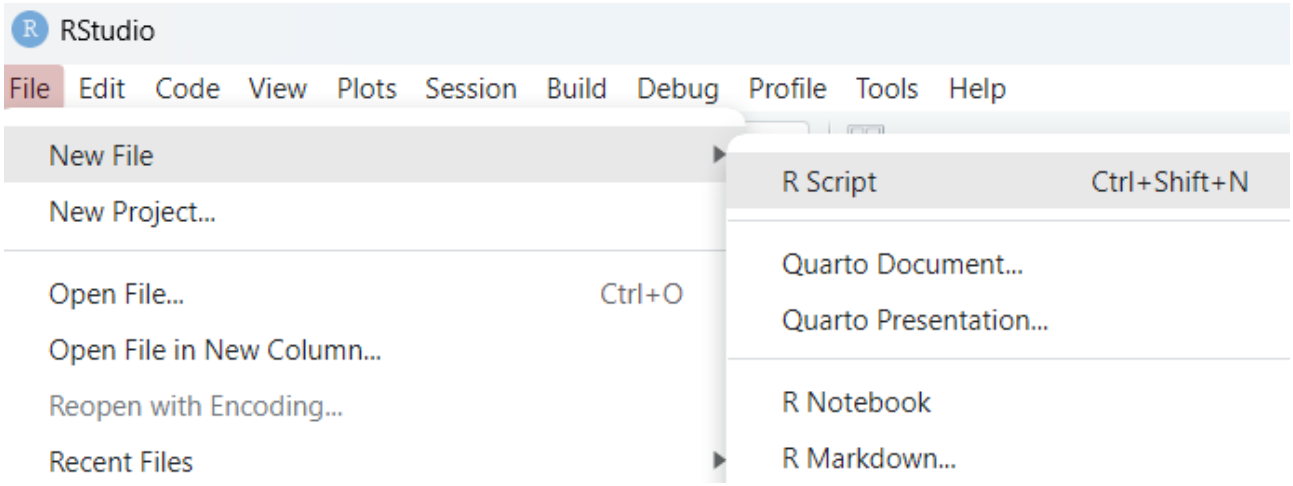
4.

DATA HANDLING AND BASIC DATA STRUCTURES

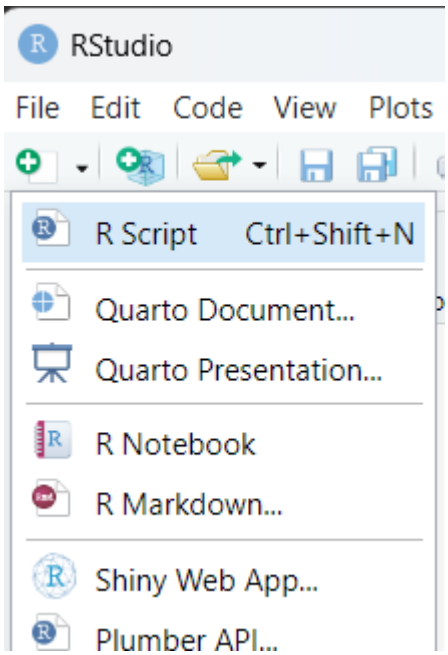
The aim of this chapter is to introduce essential data management skills in R, including setting up a working directory, importing and exporting data. You will learn about different data types in R, such as character, factor, double, integer, and logical, and how these types influence data analysis. Additionally, the chapter will cover fundamental data structures, including atomic vectors, matrices, and data frames. Finally, you will practice renaming column names, recoding values, and handling missing data to ensure data quality and consistency.

1. Importing and Saving Data

Let's open new script for chapter 2. Go to File -> New File -> R Script (Ctrl + Shift + N).



You can also simply click the icon (a green circle with a plus sign inside it) and select R Script.



1.1 Setting up the Working Directory

When you open R, it connects to a folder on your computer known as the 'working directory'. This is the default location where R looks for files (e.g., Dataset, Rscripts) when you attempt to import them. It is also where R saves or exports files. Since the default working directory varies depending on your computer, it is important to check its current location before working with files. You can determine the current working directory by executing the following command:

```
getwd() #You can check the location of your current working directory.
```

This code will return the path of the current working directory as above, if you need to change the working directory to a specific folder where your data and scripts are stored, you can use the setwd() function:

```
setwd ("C:/Your folder path") # For Windows

setwd ("/Your folder path") # For MacOS
```

Make sure to replace "C:/ Your folder path" or "/ Your folder path" with the actual path to your folder. It is highly recommended to create a SOCYR folder on your computer to store all files and scripts. For example, you can set your working directory as below:

```
setwd("C:/Your folder path/SOCYR") # For Windows

setwd("/Your folder path/SOCYR") # For MacOS
```

*The `setwd()` function has a limitation when used in R Markdown—it only sets the working directory temporarily within the specific chunk where it is called. R Markdown resets the working directory after each chunk finishes running, so the change does not persist across chunks. Therefore, if you're using R Markdown, you should use the code below to set the working directory for all chunks:

```
knitr::opts_knit$set(root.dir = "C:/Your folder path/SOCYR")
```

1.2 Importing Data

Once you have set up your working directory, the next step is to import data into R. The import method depends on the file format. In this section, we will practice importing two of the most commonly used file types: CSV (.csv) and Excel (.xlsx) using the Sample dataset.

1.2.1 Importing a CSV File

You can import a CSV file using the read.csv() function from base R or read_csv() function from the readr package.

```
#Install and load the readr package
install.packages("readr")
library(readr)
```

```
#Using base R
Data <- read.csv("C:/the file path/Sample.csv")

#Using readr package (faster and more efficient)
Data <- read_csv("C:/the file path/Sample.csv")
```

1.2.2 Importing an Excel File

Excel files (.xlsx) require external packages for importing. The readxl package is commonly used for this purpose.

```
#Install and load the readxl package
install.packages("readxl")
library(readxl)
```

```
#Import an Excel file
Data <- read_excel("C:/the file path/WVS.xlsx")
#If you are using Excel 2003 or earlier, use .xls instead of .xlsx
```

1.3 Exporting Data

Just as we can import data into R, we can also export datasets to external formats such as CSV or Excel files. Exporting data is useful when you modified the original dataset and want to save the updated version for future use or share it with others.

1.3.1 Exporting a CSV File

To save a dataset as a CSV file, use the write.csv() function in base R or write_csv() function from the readr package.

```
#Using base R
write.csv(Data, "NewSample.csv", row.names = FALSE) #Do not include row names when saving the data

# Using readr package (does not include row names by default)
write.csv(Data, "NewSample.csv")
```

1.3.2 Exporting an Excel File

To save data as an Excel file, you can use the writexl package:

```
#Install and load the writexl package
#install.packages("writexl")
library(writexl)

#Export data to an Excel file
write_xlsx(Data, "NewSample.xlsx") #If you are using Excel 2003 or earlier, use .xls instead of .xlsx
```

There are various options available when importing and exporting datasets in R. Visit R Data Import/Export for more details if you need to customise the process, such as specifying header, encoding, or column formats.

REMEMBER. R is case sensitive. So, when you label a data file, object or value you must be accurate with the use of capital and lowercase.

2. Data Types

You may be familiar with data types such as: Nominal (categorical data without order), Ordinal (categorical data with order), and Quantitative/Continuous data (numeric data).

In R, the data types are defined slightly differently:

- **Character / :** Represents text or string data, such as "Australia", "R programming", or "123" (when stored as text).
- **Factor / :** Represents categorical data with a fixed set of unordered categories, such as hair colour: "Black", "Blonde", "Brown", and "White".
- **Ordered / :** Represents categorical data with a fixed set of ordered categories, such as income range: "Low" < "Medium" < "High".
- **Double / :** Represents numeric values with decimal points (floating-point numbers), such as 3.14 or 172.5.
- **Integer / :** Represents whole numbers, such as 1, 42, or -7. In R, integers are explicitly indicated with an L suffix (e.g., 42L).
- **Logical / :** Represents Boolean values, such as TRUE or FALSE.

Visit Column types for a more detailed list of data types.

Now, we can explore our dataset to understand how each variable is classified. To view all variables along with their data types, use the glimpse() function from the dplyr package. This function provides a compact, easy-to-read summary of the dataset, making it useful for quickly browsing variable types and structures.

For this exercise, I've created a fictional dataset (chapter2_Data) for practice. Let's take a look at the variables it contains and examine their data types using the glimpse() function:

```
install.packages("dplyr") #If not previously downloaded
library(dplyr)
chapter2_Data <- read.csv("C:/Your folder path/SOCYR/chapter2_Data.csv")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
glimpse(chapter2_Data)
```

```
## Rows: 30
## Columns: 6
## $ ID            <chr> "ID001", "ID002", "ID003", "ID004", "ID005", "ID006", "I…
## $ Sex           <chr> "Male", "Male", "Male", "Male", "Female", "Male", "Femal…
## $ Income        <int> 60000, 105000, 45000, 85000, 80000, 105000, 55000, 65000…
## $ Health_Status <chr> "Poor", "Poor", "Very Good", "Poor", "Fair", "Fair", "Go…
## $ Region        <chr> "Fortitude Valley", "South Brisbane", "Paddington", "New…
## $ Age           <int> 22, 38, 18, 43, 21, 51, 37, 65, 30, 40, 54, 34, 34, 48, …
```

If any variables are misclassified, you can adjust their data type accordingly to ensure accurate analysis. For example, factors are specifically designed for categorical data, whereas character variables are simply treated as text. Using factors instead of characters improves data handling, statistical summaries, and visualisation.

For example, in our dataset, the Sex variable is currently defined as a character (chr), even though it represents categorical data. Since categorical variables are better handled as factors (fct), we can convert it using the as.factor() function:

```
# Convert the Sex variable to a factor
chapter2_Data$Sex <- as.factor(chapter2_Data$Sex)
# Check the updated structure
str(chapter2_Data$Sex)
```

```
## Factor w/ 2 levels "Female","Male": 2 2 2 2 1 2 1 2 1 2 ...
```

There may be cases where you need to convert data into formats other than character (chr) to factor (fct). Just like as.factor(), R provides several functions to transform variables into the appropriate format:

- as.integer() – converts values into integers (whole numbers). Note that this function does not round; instead, it truncates decimals (e.g., 5.9 and 5.2 would both become 5).

- as.numeric() – converts values into numeric format, retaining decimal places when present (e.g., 5.9 remains 5.9).
- as.character() – converts values into strings, even if they contain numbers. If you use the readr package to import data, you may not need this function often, as readr automatically imports unknown data types as chr by default.

Additionally, the output above displays the name of each column/variable following the $ symbol, such as Country. The $ operator is used to access specific variables within a dataset.

For example, to inspect only the Sex variable, you can use the following command:

```
chapter2_Data$Sex
```

```
##  [1] Male   Male   Male   Male   Female Male   Female Male   Female Male
## [11] Male   Male   Male   Male   Male   Male   Female Male   Male   Male
## [21] Female Male   Male   Female Male   Female Female Male   Female Female
## Levels: Female Male
```

Now that the dataset has been revised, we can export the updated version using both the write_xlsx and write.csv() functions. After exporting the files, be sure to check your folder to confirm that they have been saved successfully.

```
write.csv(chapter2_Data, "Chapter2_Data.csv", row.names = FALSE)
write_xlsx(chapter2_Data, "Chapter2_Data.xlsx")
```

3. Atomic Vectors

Vectors are the most basic data structure in R, with atomic vectors being the simplest form. An atomic vector consists of a sequence of elements of the same type. R provides six fundamental types of atomic vectors: doubles, integers, characters, logicals, complex, and raw. Each atomic vector can hold only a single data type, ensuring consistency within the vector.

You can create vectors by combining values using the c() function. To check whether an object is an atomic vector, you can use the is.vector() function, which returns TRUE if it is and FALSE otherwise.

For example:

```
Height <- c(180, 170, 165, 160, 173, 185)
Height
```

```
## [1] 180 170 165 160 173 185
```

```
#Checking the length of an atomic vector
length(Height)
```

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

```
#Testing an atomic vector
is.vector(Height)
```

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

3.1. Doubles

A double vector is the most common type of numeric vector in R, storing regular numbers that can be positive or negative, large or small, and may or may not include decimal places. By default, R treats any number you enter as a double.

For example, when you create a vector of ages:

```
Age <- c(18, 19, 20, 22, 24, 29)
Age
```

```
## [1] 18 19 20 22 24 29
```

To determine the underlying data type of an object in R, use the typeof() function:

```
typeof(Age)
```

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

In R, double vectors are often referred to as "numeric" vectors, and many R functions use the term "numeric" instead of "double." While "double" is a technical term in computer science—indicating the number of bytes allocated for storing a number—the term "numeric" is more intuitive for data analysis and everyday use in R.

3.2 Integers

Integer vectors store whole numbers, meaning values that do not have a decimal component. However, in most cases, you will not need to use the integer type explicitly, as R automatically treats numbers as double by default. This is because doubles can store both whole numbers and decimals, providing greater flexibility for calculations.

If you specifically need to store a number as an integer, you can use the L suffix:

```
int <- c(-3L, 6L, 9L)
int
```

```
## [1] -3 6 9
```

```
typeof(int)
```

```
## [1] "integer"
```

In R, a number is not stored as an integer unless you explicitly indicate it using the L suffix. Without L, R automatically saves numbers as doubles, even if they appear to be whole numbers. For example, the only difference between 4 and 4L is how R manages them in memory.

```
x1 <- 4   # Stored as a double
x2 <- 4L  # Stored as an integer

typeof(x1)
```

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

```
typeof(x2)
```

```
## [1] "integer"
```

Why would you save your data as an integer instead of a double?

Since doubles can store both whole numbers and decimals, R defaults to using them for flexibility. However, specifying integers with L can be useful in specific cases, such as optimising memory usage in large datasets or ensuring compatibility with functions that require integer inputs.

3.3 Characters

A character vector is used to store text data, including single characters, words, or entire sentences. To create a character vector, enclose each text element in double (" ") quotes:

```
text <- c("Welcome", "to", "SOCY")
text
```

```
## [1] "Welcome" "to" "SOCY"
```

```
typeof("Welcome")
```

[1] "character"

```
typeof("to")
```

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

```
typeof("SOCY")
```

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

Each element within a character vector is called a string. A string is simply a sequence of characters, which can include not only letters but also numbers, symbols, and spaces.

Quiz. Can you tell the difference between a character string and a number?

Take a look at the following values:

25, “25”, “twenty-five”

Which of these are character strings, and which are numbers?

Write your response here:

3.4 Logical

Logical vectors store TRUE and FALSE values, which represent Boolean data. These are especially useful for performing comparisons and making decisions in your code.

```
8 > 2
```

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

```
typeof(8>2)
```

```
## [1] "logical"
```

```
"Male" == "Female"
```

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

```
typeof("Male" == "Female")
```

```
## [1] "logical"
```

R also supports two additional types: complex and raw, which are rarely needed for data analysis. If you're interested in exploring these further, you can refer to R's documentation or experiment with them on your own.

4. Matrices

A matrix is a two-dimensional data structure where all elements must be of the same type—numeric, character, or logical. Unlike data frames, which can hold multiple data types in different columns, matrices maintain a consistent type across all elements.

You can create a matrix using the matrix() function:

```
matrix <- matrix(Age, nrow = 2)
matrix
```

```
##      [,1] [,2] [,3]
## [1,]   18   20   24
## [2,]   19   22   29
```

By default, the matrix() function fills the matrix column-wise. However, you can change this to row-wise by setting the argument byrow = TRUE:

```
matrix_byrow <- matrix(Age, nrow = 2, byrow = TRUE)
matrix_byrow
```

```
##      [,1] [,2] [,3]
## [1,]   18   19   20
## [2,]   22   24   29
```

The matrix() function includes several default arguments that allow you to customize the matrix. To explore these options, you can check the help page by running `? matrix` in R.

5. Lists

Lists are similar to atomic vectors in that they organise data into a one-dimensional structure. However, lists can contain a mix of R objects, including atomic vectors, matrices, data frames, and other lists. For example, you can create a list where the first element is a numeric vector of starting at 10 and ending at 30 (length 20), and the second element is a character vector of length 3.

To do this, use the list() function:

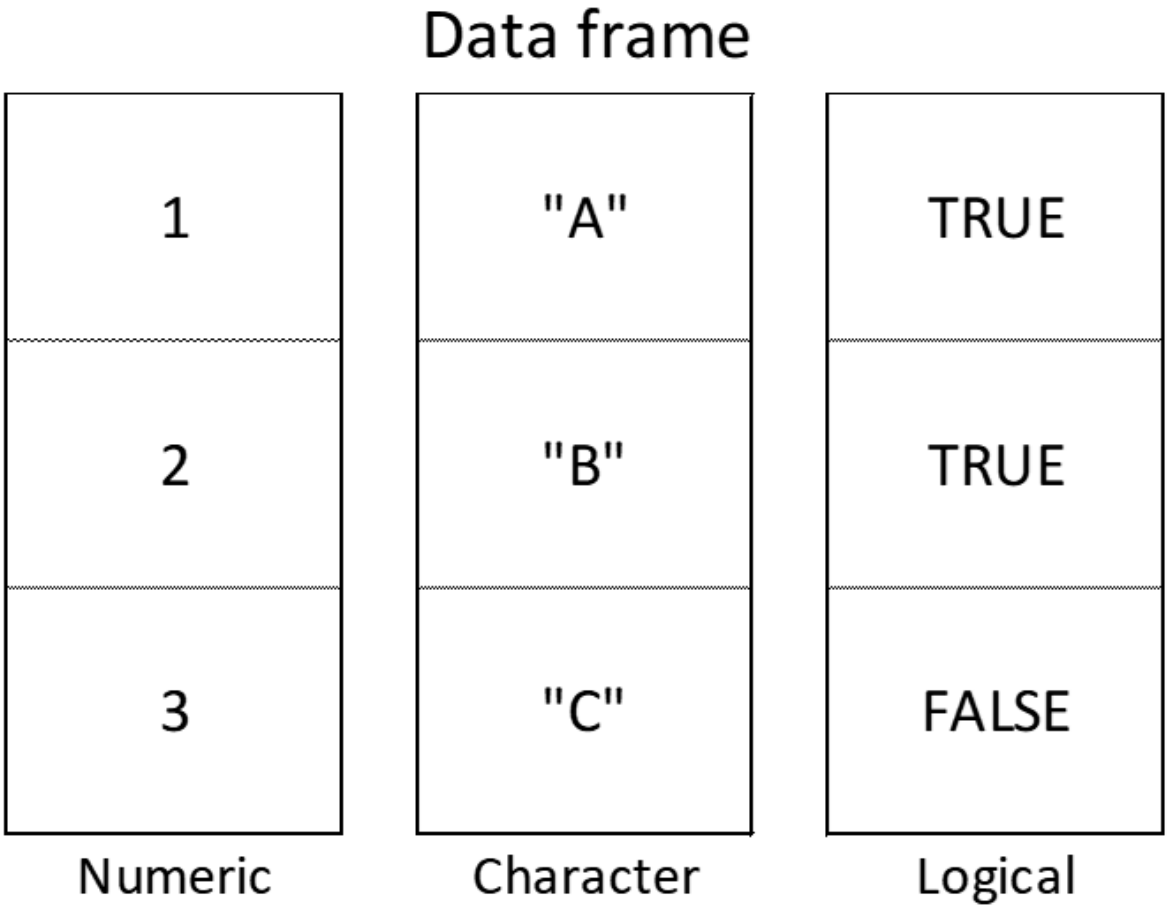
```
list <- list(10:30, c("Welcome", "to", "SOCY"))
# list creates a list the same way 'c()' creates a vector. Separate each element in the list with a
comma.
list
```

```
## [[1]]
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
##
## [[2]]
## [1] "Welcome"  "to"       "SOCY"
```

Double brackets [[ ]] indicate an entire element within a list, while single brackets [ ] specify a sub-element within that element. For example, in a list where the first element is the entire numeric vector, 10 would be its first sub-element. Similarly, if the second element is the full character vector, "Welcome" would be its first sub-element.

6. Data Frames

Data frames are a two-dimensional extension of lists, similar to an Excel spreadsheet, as they store and organize data in a structured format. They group vectors into a table, where each vector represents a column. This allows different columns to store various types of data, such as numeric, character, or logical values. However, within a single column, all elements must be of the same data type, and have a consistent length, as shown in the figure below.



Let's manually create a data frame since you're new to R! You can do this using the data.frame() function. Provide multiple vectors, each separated by a comma, and assign each one a descriptive name. The function will then convert these vectors into columns within the data frame. Ensure that all vectors are the same length, as data frames cannot accommodate columns of different lengths.

```
df <- data.frame(ID = c("ID001", "ID002", "ID003"),
                                 Occupation = c("Student", "Lawyer", "Software Devel-
oper"),
                                 Age = c(29, 24, 35))
df
```

ID	Occupation	Age
ID001	Student	29
ID002	Lawyer	24
ID003	Software Developer	35

There are many other important topics, such as Attributes, Arrays, Class, and Coercion. If you're interested in more advanced concepts related to vectors, you can refer to the R textbook .

7. Renaming Column Names and Recoding Values

It is useful to assign an intuitive name to the entries of a vector. If the original variable's name is not intuitive, you can change the column names to improve clarity and interpretation.

There are multiple ways to rename variables, with the most common methods being the names() and colnames() functions.

7.1 Renaming all Column Names

The names() function can be used to rename all column names in a data frame at once:

```
data <- data.frame(income = c(45000, 55000, NA, 72000, NA),
occupation = c("Retail Worker", "Administrative Assistant", "Teacher",
"Registered Nurse", "Marketing Specialist"))
data
```

income	occupation
45000	Retail Worker
55000	Administrative Assistant
NA	Teacher
72000	Registered Nurse
NA	Marketing Specialist

```
# Rename all column names
names(data) <- c("Income", "Occupations")

data #In R, variable names are case-sensitive, meaning Income and income are considered different.
```

Income	Occupations
45000	Retail Worker
55000	Administrative Assistant
NA	Teacher
72000	Registered Nurse
NA	Marketing Specialist

7.2 Renaming a Specific Column

To rename only a specific column while keeping others unchanged, indexing can be used within the colnames() function. For example, to rename only the second column:

```
colnames(data)[2] <- "Occupations_Aus"
data
```

Income	Occupations_Aus
45000	Retail Worker
55000	Administrative Assistant
NA	Teacher
72000	Registered Nurse
NA	Marketing Specialist

You can also use the rename() function:

```
library(dplyr)

data <- rename(data, Income_Aus = Income)
data
```

Income_Aus	Occupations_Aus
45000	Retail Worker
55000	Administrative Assistant
NA	Teacher
72000	Registered Nurse
NA	Marketing Specialist

In this example, the column Income is renamed Income_Aus, improving readability. The correct order of assignment is **new_name = old_name**.

7.3 Renaming Values in Columns

It is often necessary to recode or rename values within a variable to improve consistency and facilitate analysis. The approach to recoding values differs based on the variable type: character(string) variables require quotation marks, whereas numeric(continuous) variables allow direct value assignment.

When renaming values in a character variable, use quotation marks (“”) to specify the original and replacement values. For example:

```
data$Occupations[data$Occupations == "Administrative Assistant"] <- "Admin_Assistant"
data$Occupations
```

```
## [1] "Retail Worker" "Admin_Assistant" "Teacher"
## [4] "Registered Nurse" "Marketing Specialist"
```

For numeric variables, values can be directly assigned. Let’s replace missing values with the variable’s mean:

```
data$Income_Aus[is.na(data$Income_Aus)] <- 57333.33
data$Income_Aus
```

```
## [1] 45000.00 55000.00 57333.33 72000.00 57333.33
```

All missing values in the income variable will be replaced with its mean (57,333.33). A detailed discussion on handling missing values will follow in the next section.

8. Missing Data

In social science research, missing data is a common issue. Data may be absent due to corrupted or lost measurements, incomplete data collection, or participants opting not to respond. While this section does not provide an in-depth treatment of missing data analysis, it offers a general overview. For this exercise, we will use the air quality dataset, which comes pre-installed with R. This dataset contains daily air quality measurements recorded in New York during 1973, comprising 153 observations on 6 variables.

```
?airquality

data(airquality)Copy
```

```
summary(airquality)
```

```
##      Ozone           Solar.R          Wind             Temp
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##  NA's   :37       NA's   :7
##      Month            Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
```

8.1 Identifying Missing Data

One of the simplest ways to identify missing values in R is by using the is.na() function. This function identifies whether a value is NA, which is a special symbol in R representing not available data. The function returns a logical vector, where TRUE indicates the position of missing values, and FALSE indicates non-missing values. This makes it easy to locate and handle missing data in vectors, matrices, or data frames.

For example, to check for missing values in the airquality dataset:

```
is.na(airquality)
```

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

```
##   [2,] FALSE   FALSE FALSE FALSE FALSE FALSE
##   [3,] FALSE   FALSE FALSE FALSE FALSE FALSE
##   [4,] FALSE   FALSE FALSE FALSE FALSE FALSE
##   [5,]  TRUE    TRUE FALSE FALSE FALSE FALSE
##   [6,] FALSE    TRUE FALSE FALSE FALSE FALSE
##   [7,] FALSE   FALSE FALSE FALSE FALSE FALSE
##   [8,] FALSE   FALSE FALSE FALSE FALSE FALSE
##   [9,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [10,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [11,] FALSE    TRUE FALSE FALSE FALSE FALSE
##  [12,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [13,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [14,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [15,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [16,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [17,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [18,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [19,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [20,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [21,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [22,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [23,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [24,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [25,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [26,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [27,]  TRUE    TRUE FALSE FALSE FALSE FALSE
##  [28,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [29,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [30,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [31,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [32,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [33,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [34,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [35,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [36,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [37,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [38,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [39,]  TRUE   FALSE FALSE FALSE FALSE FALSE
```

```
##  [40,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [41,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [42,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [43,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [44,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [45,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [46,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [47,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [48,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [49,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [50,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [51,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [52,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [53,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [54,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [55,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [56,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [57,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [58,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [59,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [60,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [61,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [62,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [63,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [64,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [65,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [66,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [67,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [68,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [69,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [70,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [71,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [72,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [73,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [74,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [75,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [76,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [77,] FALSE   FALSE FALSE FALSE FALSE FALSE
```

```
##  [78,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [79,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [80,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [81,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [82,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [83,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [84,]  TRUE   FALSE FALSE FALSE FALSE FALSE
##  [85,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [86,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [87,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [88,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [89,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [90,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [91,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [92,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [93,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [94,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [95,] FALSE   FALSE FALSE FALSE FALSE FALSE
##  [96,] FALSE    TRUE FALSE FALSE FALSE FALSE
##  [97,] FALSE    TRUE FALSE FALSE FALSE FALSE
##  [98,] FALSE    TRUE FALSE FALSE FALSE FALSE
##  [99,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [100,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [101,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [102,]  TRUE   FALSE FALSE FALSE FALSE FALSE
## [103,]  TRUE   FALSE FALSE FALSE FALSE FALSE
## [104,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [105,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [106,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [107,]  TRUE   FALSE FALSE FALSE FALSE FALSE
## [108,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [109,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [110,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [111,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [112,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [113,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [114,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [115,]  TRUE   FALSE FALSE FALSE FALSE FALSE
```

```
## [116,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [117,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [118,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [119,]  TRUE   FALSE FALSE FALSE FALSE FALSE
## [120,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [121,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [122,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [123,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [124,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [125,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [126,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [127,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [128,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [129,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [130,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [131,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [132,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [133,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [134,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [135,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [136,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [137,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [138,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [139,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [140,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [141,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [142,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [143,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [144,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [145,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [146,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [147,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [148,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [149,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [150,]  TRUE   FALSE FALSE FALSE FALSE FALSE
## [151,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [152,] FALSE   FALSE FALSE FALSE FALSE FALSE
## [153,] FALSE   FALSE FALSE FALSE FALSE FALSE
```

Since this dataset contains more than 100 observations, manually inspecting each missing value would be inefficient. Instead, you can count the total number of missing values using the sum(is.na()) function:

```
sum(is.na(airquality))
```

```
## [1] 44
```

We can observe that there are a total of 44 missing values in the dataset. To examine missing values by column, you can use the colSums(is.na()) function:

```
colSums(is.na(airquality))
```

```
##   Ozone Solar.R    Wind    Temp   Month     Day
##      37       7       0       0       0       0
```

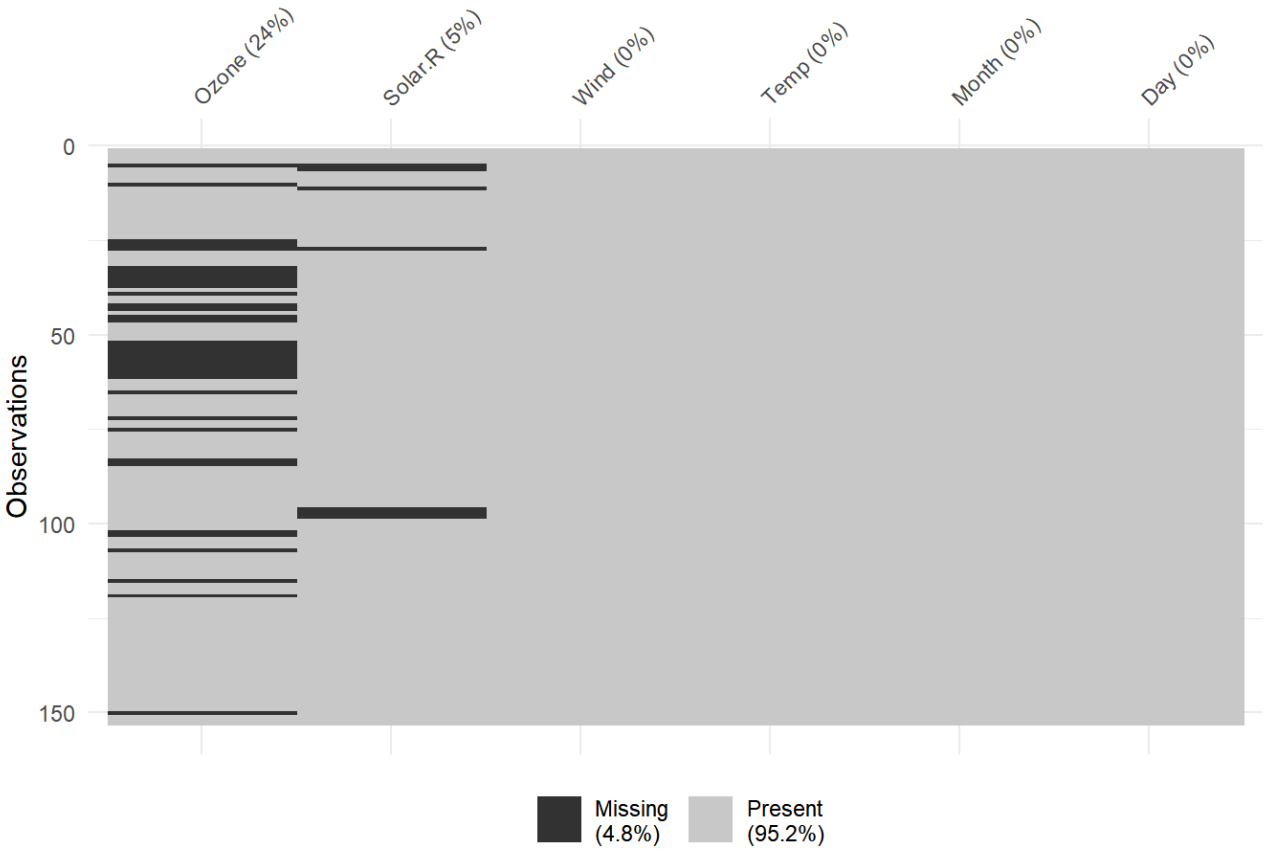
As shown, the Ozone variable has the most missing values (37), followed by Solar.R with 7 missing values. The remaining variables have no missing data.

Another effective way to identify missing values is by using a specialised package like naniar, which allows for quick and systematic detection. After loading the naniar package, you can use the vis_miss() function to visualise both the amount and location of missing data within your dataset.

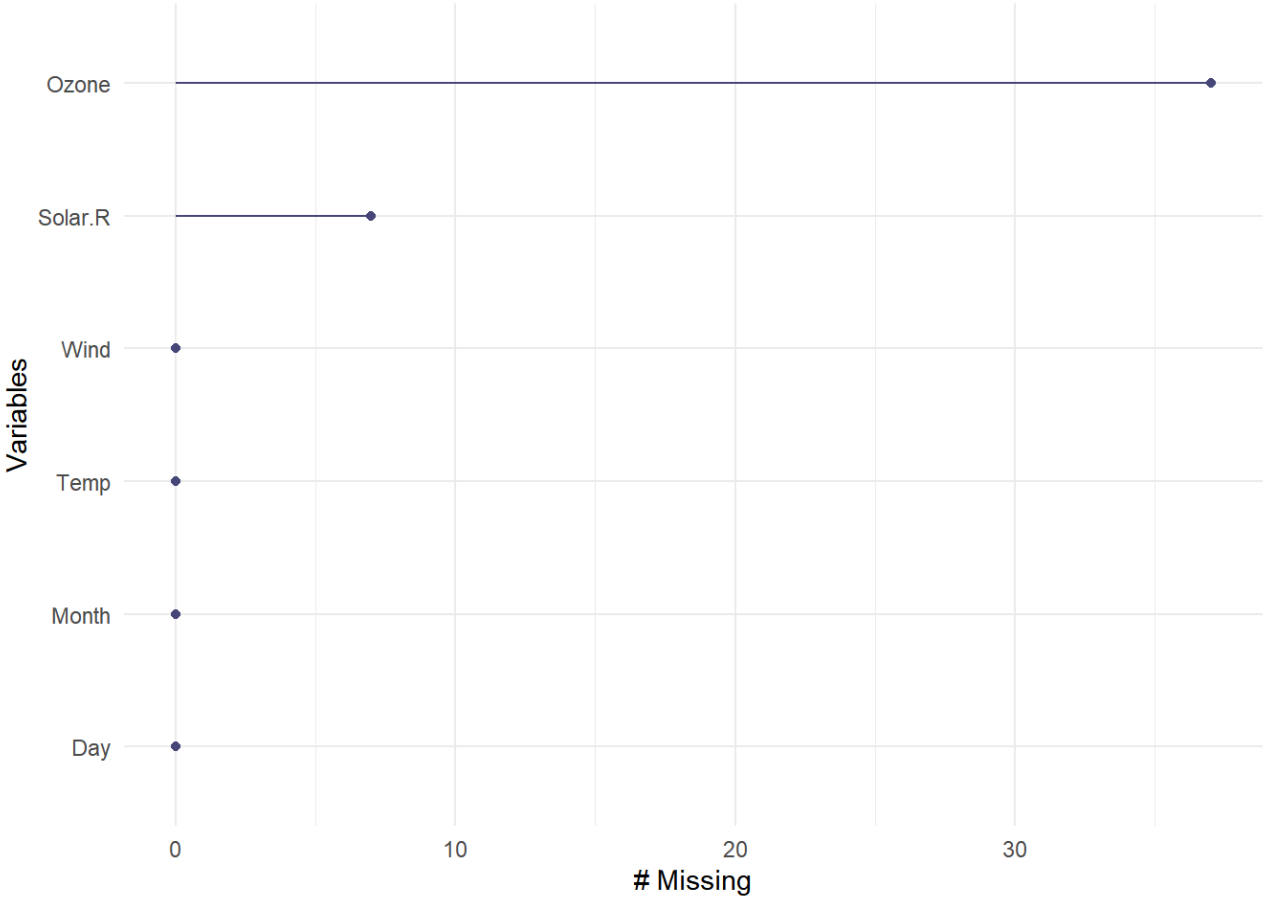
For the airquality dataset, this kind of visualisation can be especially useful to highlight which variables have the most missing values, helping you prioritise where to focus your data cleaning efforts.

```
library(naniar)
```

```
vis_miss(airquality) # A heat map-style visualisation showing where values are missing.
```




```
gg_miss_var(airquality) #A bar plot displaying the number of missing values per variable.
```




```
miss_var_summary(airquality) #A table listing the number and percentage of missing values for each variable.
```

variable	n_miss	pct_miss
Ozone	37	24.2
Solar.R	7	4.58
Wind	0	0
Temp	0	0
Month	0	0
Day	0	0

These functions offer a more intuitive understanding of missing data compared to basic numerical summaries. While functions like sum(is.na()) or colSums(is.na()) provide useful counts, visualisations make it easier to grasp the extent and patterns of missingness—especially in larger datasets. We will address how to handle these missing values in the following section.

8.2 Handling Missing Data

Handling missing data is crucial in data analysis, as missing values can impact statistical results and model accuracy. There are several ways to address missing values, depending on the nature of the data and the research context. In this section, I will provide a broad overview of two methods: Removing and replacing missing data.

8.2.1 Removing Missing Data

The simplest method to handle missing values is removing missing values. na.omit() functions removes entire row with missing values from a dataset.

```
airquality_clean <- na.omit(airquality)
airquality_clean
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
12	16	256	9.7	69	5	12
13	11	290	9.2	66	5	13
14	14	274	10.9	68	5	14
15	18	65	13.2	58	5	15
16	14	334	11.5	64	5	16
17	34	307	12	66	5	17
18	6	78	18.4	57	5	18
19	30	322	11.5	68	5	19
20	11	44	9.7	62	5	20
21	1	8	9.7	59	5	21
22	11	320	16.6	73	5	22
23	4	25	9.7	61	5	23
24	32	92	12	61	5	24
28	23	13	12	67	5	28
29	45	252	14.9	81	5	29
30	115	223	5.7	79	5	30
31	37	279	7.4	76	5	31
38	29	127	9.7	82	6	7
40	71	291	13.8	90	6	9
41	39	323	11.5	87	6	10
44	23	148	8	82	6	13
47	21	191	14.9	77	6	16
48	37	284	20.7	72	6	17
49	20	37	9.2	65	6	18
50	12	120	11.5	73	6	19

	Ozone	Solar.R	Wind	Temp	Month	Day
51	13	137	10.3	76	6	20
62	135	269	4.1	84	7	1
63	49	248	9.2	85	7	2
64	32	236	9.2	81	7	3
66	64	175	4.6	83	7	5
67	40	314	10.9	83	7	6
68	77	276	5.1	88	7	7
69	97	267	6.3	92	7	8
70	97	272	5.7	92	7	9
71	85	175	7.4	89	7	10
73	10	264	14.3	73	7	12
74	27	175	14.9	81	7	13
76	7	48	14.3	80	7	15
77	48	260	6.9	81	7	16
78	35	274	10.3	82	7	17
79	61	285	6.3	84	7	18
80	79	187	5.1	87	7	19
81	63	220	11.5	85	7	20
82	16	7	6.9	74	7	21
85	80	294	8.6	86	7	24
86	108	223	8	85	7	25
87	20	81	8.6	82	7	26
88	52	82	12	86	7	27
89	82	213	7.4	88	7	28
90	50	275	7.4	86	7	29
91	64	253	7.4	83	7	30
92	59	254	9.2	81	7	31
93	39	83	6.9	81	8	1
94	9	24	13.8	81	8	2
95	16	77	7.4	82	8	3
99	122	255	4	89	8	7
100	89	229	10.3	90	8	8

	Ozone	Solar.R	Wind	Temp	Month	Day
101	110	207	8	90	8	9
104	44	192	11.5	86	8	12
105	28	273	11.5	82	8	13
106	65	157	9.7	80	8	14
108	22	71	10.3	77	8	16
109	59	51	6.3	79	8	17
110	23	115	7.4	76	8	18
111	31	244	10.9	78	8	19
112	44	190	10.3	78	8	20
113	21	259	15.5	77	8	21
114	9	36	14.3	72	8	22
116	45	212	9.7	79	8	24
117	168	238	3.4	81	8	25
118	73	215	8	86	8	26
120	76	203	9.7	97	8	28
121	118	225	2.3	94	8	29
122	84	237	6.3	96	8	30
123	85	188	6.3	94	8	31
124	96	167	6.9	91	9	1
125	78	197	5.1	92	9	2
126	73	183	2.8	93	9	3
127	91	189	4.6	93	9	4
128	47	95	7.4	87	9	5
129	32	92	15.5	84	9	6
130	20	252	10.9	80	9	7
131	23	220	10.3	78	9	8
132	21	230	10.9	75	9	9
133	24	259	9.7	73	9	10
134	44	236	14.9	81	9	11
135	21	259	15.5	76	9	12
136	28	238	6.3	77	9	13
137	9	24	10.9	71	9	14

	Ozone	Solar.R	Wind	Temp	Month	Day
138	13	112	11.5	71	9	15
139	46	237	6.9	78	9	16
140	18	224	13.8	67	9	17
141	13	27	10.3	76	9	18
142	24	238	10.3	68	9	19
143	16	201	8	82	9	20
144	13	238	12.6	64	9	21
145	23	14	9.2	71	9	22
146	36	139	10.3	81	9	23
147	7	49	10.3	69	9	24
148	14	20	16.6	63	9	25
149	30	193	6.9	70	9	26
151	14	191	14.3	75	9	28
152	18	131	8	76	9	29
153	20	223	11.5	68	9	30

Rows containing NA values are completely removed using this method. While this approach is straightforward, it has a major drawback—it significantly reduces the sample size. The original dataset has 153 observations, but after removing missing values, only 111 remain. Such data loss can compromise the accuracy of statistical results, particularly when more than 10% of the data is missing.

An alternative is to use the na.rm = TRUE argument, which allows you to exclude missing values from calculations without removing entire observations. This option can be used in functions like mean(), sum(), and min(), as well as in analytical models such as regression. Unlike na.omit(), it retains the full dataset and simply tells the function to ignore NA values during computation.

```
#Mean of the Ozone variable in the dataset while ignoring NA values.
mean(airquality$Ozone, na.rm = TRUE)
```

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

```
#Mean of the Ozone variable in the modified dataset with all NA values removed.
mean(airquality_clean$Ozone)
```

```
## [1] 42.0991
```

The difference in outcomes occurs because mean(airquality$Ozone, na.rm = TRUE) ignores NA values in the Ozone column but retains all other rows. In contrast, mean(airquality_clean$Ozone) is calculated after removing entire rows that contain NA in any column, resulting in a smaller dataset.

8.2.2 Replacing Missing Data

In social science research, missing values can be handled through imputation rather than simply removing data. One of the approaches is replacing NA with the column's mean or median:

```
airquality$Ozone[is.na(airquality$Ozone)] <- mean(airquality$Ozone, na.rm = TRUE)
#Mean value: 42.12931
airquality$Ozone
```

Copy

```
##   [1]  41.00000  36.00000  12.00000  18.00000  42.12931  28.00000  23.00000
##   [8]  19.00000   8.00000  42.12931   7.00000  16.00000  11.00000  14.00000
##  [15]  18.00000  14.00000  34.00000   6.00000  30.00000  11.00000   1.00000
##  [22]  11.00000   4.00000  32.00000  42.12931  42.12931  42.12931  23.00000
##  [29]  45.00000 115.00000  37.00000  42.12931  42.12931  42.12931  42.12931
##  [36]  42.12931  42.12931  29.00000  42.12931  71.00000  39.00000  42.12931
##  [43]  42.12931  23.00000  42.12931  42.12931  21.00000  37.00000  20.00000
##  [50]  12.00000  13.00000  42.12931  42.12931  42.12931  42.12931  42.12931
##  [57]  42.12931  42.12931  42.12931  42.12931  42.12931 135.00000  49.00000
##  [64]  32.00000  42.12931  64.00000  40.00000  77.00000  97.00000  97.00000
```

```
##  [71]  85.00000  42.12931  10.00000  27.00000  42.12931   7.00000  48.00000
##  [78]  35.00000  61.00000  79.00000  63.00000  16.00000  42.12931  42.12931
##  [85]  80.00000 108.00000  20.00000  52.00000  82.00000  50.00000  64.00000
##  [92]  59.00000  39.00000   9.00000  16.00000  78.00000  35.00000  66.00000
##  [99] 122.00000  89.00000 110.00000  42.12931  42.12931  44.00000  28.00000
## [106]  65.00000  42.12931  22.00000  59.00000  23.00000  31.00000  44.00000
## [113]  21.00000   9.00000  42.12931  45.00000 168.00000  73.00000  42.12931
## [120]  76.00000 118.00000  84.00000  85.00000  96.00000  78.00000  73.00000
## [127]  91.00000  47.00000  32.00000  20.00000  23.00000  21.00000  24.00000
## [134]  44.00000  21.00000  28.00000   9.00000  13.00000  46.00000  18.00000
## [141]  13.00000  24.00000  16.00000  13.00000  23.00000  36.00000   7.00000
## [148]  14.00000  30.00000  42.12931  14.00000  18.00000  20.00000
```

The observation in the Ozone variable is retained. However, **it's important to choose your method for handling missing values carefully, as it can significantly affect the results of your analysis.**

Recently, advanced imputation techniques have become the preferred approach for handling missing data, as they provide more accurate estimates (compared to replacing missing values with the mean or median).

Some commonly used methods include:

- MICE (Multivariate Imputation by Chained Equations): Uses multiple imputations to generate plausible values for missing data.
- Amelia: Employs a bootstrapped-based algorithm to estimate missing values.
- missForest: Uses a random forest algorithm to predict and impute missing values.

These techniques are beyond the scope of this book. However, if you're interested, it is recommended to explore them further through research and practical application.

5.

MULTIPLE REGRESSION ANALYSIS

This chapter introduces multiple regression analysis, focusing on how one dependent variable is related to several independent variables. The session will begin with fitting regression models using continuous predictors. We then focus on how to interpret key statistics from the multiple regression output.

1. Multiple Regression

Multiple regression is an extension of bivariate linear regression that models the relationship between a single dependent variable (Y) and two or more independent variables (X_1, X_2, ..., X_k). By fitting a linear equation to observed data, multiple regression allows researchers to assess the impact of each predictor while controlling for the influence of the others.

Exercise 1 – Bivariate Linear Regression with ABS Data

In Chapter 5, Exercise 2 we fitted a bivariate linear regression model to examine the relationship between *Life Satisfaction* (dependent variable) and *Age* (independent variable). In today's session, we will expand this model by including two additional continuous predictors: Income and EDU (education level). We will explore how socio-economic factors (age, income, and education level) are associated with life satisfaction among Australian participants. The general equation of a multiple regression model is as follows:

$$Y = a + b_1X_1 + b_2X_2 + b_3X_3$$

Applying our variables to this equation:

$$\hat{Lifesatisfaction} = a + (b_1 * Age) + (b_2 * Income) + (b_3 * EDU)$$

- Y is the dependent variable (Life Satisfaction),
- X_1, X_2, X_3 are the independent variables (Age, Income, and EDU),
- a is the intercept (the expected value of Life satisfaction when all predictors are held constant (X_1, X_2, and X_3 = 0),
- b_1, b_2, b_3 are the slopes (regression coefficients representing the expected change in Life Satisfaction associated with a one-unit increase in the corresponding predictor, holding all other variables constant),

1.1 Loading and Inspecting the Dataset

Let's load the dataset and inspect the dataset to ensure it has been imported correctly.

```
#Set up the working directory first
setwd("C:/Your folder path/SOCYR")
WVS <- read.csv("WVS.csv")
```

```
#Load the required packages for Chapter 6
library(corrplot)
library(psych)
library(dplyr)
```

```
head(WVS, 10)
```

Row number	D_INTERVIEW	Country	Q47	Life_satisfaction	Q124	Q128	Q152	Wife_Abuse	Q128	Q152
1	20070001	20	3	10	0	2	1	1	2	1
2	20070002	20	1	9	1	0	1	1	0	1
3	20070003	20	1	9	0	0	1	1	0	1
4	20070004	20	2	8	0	1	1	1	1	1
5	20070005	20	2	7	1	1	1	1	1	1
6	20070006	20	1	10	1	0	1	1	0	1
7	20070007	20	1	5	0	0	1	1	0	1
8	20070008	20	2	8	0	2	1	1	2	1
9	20070009	20	2	8	0	0	1	1	0	1
10	20070010	20	1	10	2	0	3	1	0	3

1.2 Cleaning the Data

Before proceeding with the multiple regression, let's clean the variables first. We then restrict the dataset to Australian respondents (**Country code = 36**) and rename relevant variables.

```
Aus_data <- WVS %>%
  filter(Country == 36)  #Filter only Australian participants
```

1.3 Summary Statistics

We begin by examining the summary statistics of the key variables that will be included in the regression model.

```
Aus_data %>%
  select(Life_satisfaction, Age, Income, EDU) %>%
summary()
```

```
##  Life_satisfaction      Age            Income            EDU
##  Min.   : 1.000    Min.   :17.0   Min.   : 1.000   Min.   :0.000
##  1st Qu.: 7.000    1st Qu.:40.0   1st Qu.: 4.000   1st Qu.:3.000
##  Median : 8.000    Median :56.0   Median : 5.000   Median :4.000
##  Mean   : 7.528    Mean   :54.3   Mean   : 5.128   Mean   :4.655
##  3rd Qu.: 9.000    3rd Qu.:68.0   3rd Qu.: 7.000   3rd Qu.:6.000
##  Max.   :10.000    Max.   :98.0   Max.   :10.000   Max.   :8.000
##  NA's   :16        NA's   :18     NA's   :65       NA's   :70
```

```
# The summary() function in R automatically excludes missing values (NA) when computing
summary statistics for numeric variables.
```

1.4 Correlation Analysis of Predictors

We next inspect the bivariate relationships among the independent variables (Age, Income, and EDU) by computing a correlation matrix.

```
Aus_data %>%
  select(Age, Income, EDU) %>%
  cor(, use ="pairwise.complete.obs")
```

```
##                Age     Income        EDU
## Age     1.0000000 -0.1951478 -0.2725097
## Income -0.1951478  1.0000000  0.4137615
## EDU    -0.2725097  0.4137615  1.0000000
```

To examine the strength and statistical significance of the correlations, we use the `corr.test()` function from the *psych* package:

```
# Compute correlations and p-values
corr_results <- psych::corr.test(Aus_data %>% select(Age, Income, EDU), use = "pair-
wise.complete.obs")

# View correlation matrix (coefficients)
corr_results$r
```

```
##                Age     Income        EDU
## Age     1.0000000 -0.1951478 -0.2725097
## Income -0.1951478  1.0000000  0.4137615
## EDU    -0.2725097  0.4137615  1.0000000
```

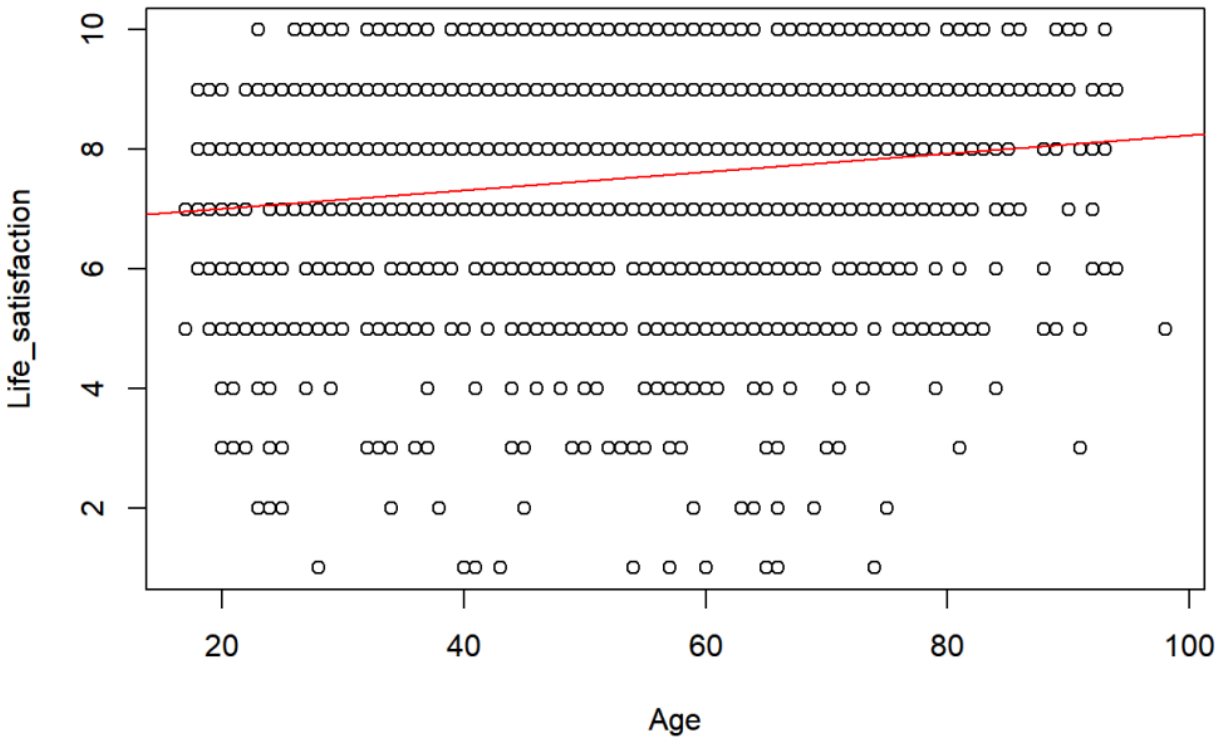
```
# View p-value matrix
corr_results$p
```

```
##                  Age       Income          EDU
## Age    0.000000e+00 2.106425e-16 1.330663e-30
## Income 2.106425e-16 0.000000e+00 1.728738e-70
## EDU    6.653314e-31 5.762461e-71 0.000000e+00
```

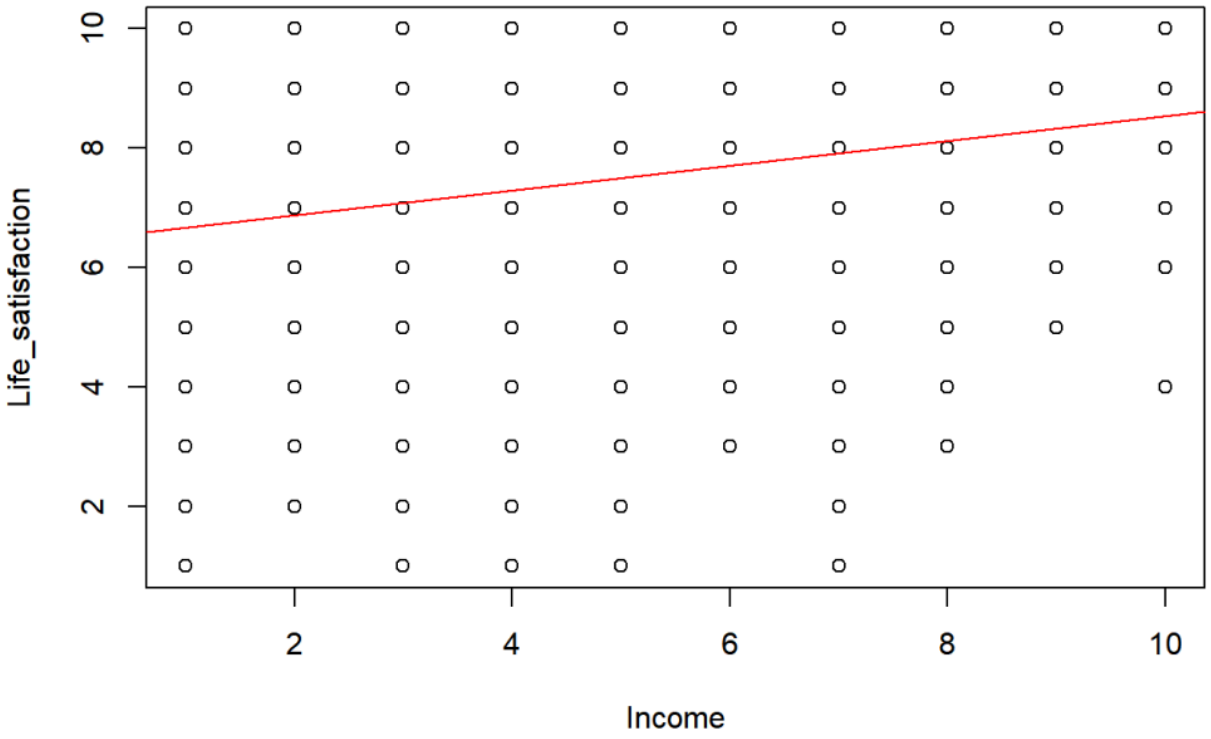
1.5 Visualising Bivariate Relationships

Scatter plots provide a visual inspection of the bivariate relationships between Life Satisfaction and each continuous predictor. A regression line is added to assess the linearity of each relationship.

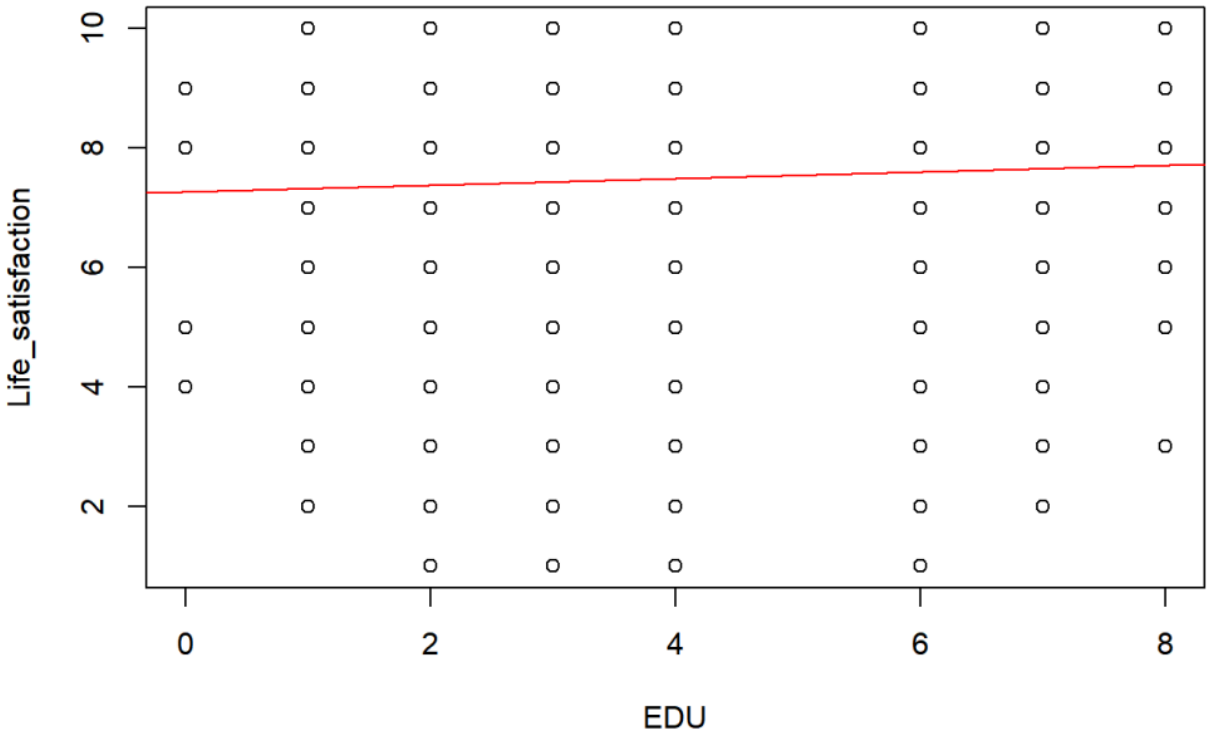
```
#Scatter plot of Life satisfaction - Age
plot(Aus_data$Age, Aus_data$Life_satisfaction, xlab = "Age", ylab = "Life_satisfaction")
abline(lm(Life_satisfaction~ Age, data = Aus_data), col = "red")
```




```
#Scatter plot of Life satisfaction - Income
plot(Aus_data$Income,Aus_data$Life_satisfaction, xlab = "Income", ylab = "Life_satisfac-
tion")
abline(lm(Life_satisfaction~ Income, data = Aus_data), col = "red")
```




```
#Scatter plot of Life satisfaction - EDU
plot(Aus_data$EDU, Aus_data$Life_satisfaction, xlab = "EDU", ylab = "Life_satisfaction")
abline(lm(Life_satisfaction ~ EDU, data = Aus_data), col = "red")
```



Quiz. Interpret the results shown in each scatter plot

Write your response here:

1.6 Fitting Multiple Regression Models

To demonstrate how the results of bivariate and multiple regression models can differ from those obtained in a multiple regression model, we will begin by estimating separate bivariate regressions for each of the three explanatory variables with Life Satisfaction as the outcome variable. We will then fit a multiple regression model that includes all three predictors simultaneously.

```
Model1 <- lm(Life_satisfaction ~ Age, data=Aus_data)
Model2 <- lm(Life_satisfaction ~ Income, data=Aus_data)
```

```
Model3 <- lm(Life_satisfaction ~ EDU, data=Aus_data)
Model4 <- lm(Life_satisfaction ~ Age + Income + EDU , data=Aus_data)
```

Each model estimates the effect of a single or combined set of predictors on life satisfaction. By comparing the coefficients from the bivariate models (Models 1–3) with those from the multiple regression model (Model 4), we can observe how controlling for additional variables influences the estimated associations.

R stores the results of these models in the objects Model1, Model2, Model3, and Model4. These stored model objects will be revisited later in this session for further analysis and comparison.

The output for the multiple regression model (Model4) should appear as below:

```
summary(Model4)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Age + Income + EDU, data = Aus_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7379 -0.8079  0.2286  1.0211  3.7725
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.191910   0.211175  24.586   <2e-16 ***
## Age          0.020778   0.002435   8.532   <2e-16 ***
## Income       0.244169   0.021101  11.572   <2e-16 ***
## EDU         -0.009442   0.026167  -0.361    0.718
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.641 on 1671 degrees of freedom
##   (138 observations deleted due to missingness)
## Multiple R-squared:  0.1052, Adjusted R-squared:  0.1036
## F-statistic: 65.48 on 3 and 1671 DF,  p-value: < 2.2e-16
```

Quiz. Using the output above, answer the following questions:

Q1. How many observations contribute to the regression model?

Q2. Fully interpret the regression coefficient for Age.

Q3. Fully interpret the regression coefficient for Income.

Q4. Fully interpret the regression coefficient for EDU.

Q5. How would you interpret the results of the F-test?

Q6. What are the values for R2 and adjusted R2 for this model? What do these tell you?

Write your responses here:

Important: Remember that fully interpreting a regression coefficient for b / β requires you to comment on: (i) the sign/direction of the effect, (ii) the size/magnitude of the effect, and (iii) the statistical significance of the effect. If the coefficient comes from a multiple regression model, then you also ought to acknowledge that this is a partial regression coefficient. You can do that by adding something like "all else being equal", "holding all other variables constant", "controlling/adjusting for the other covariates in the model" or the Latin expression "ceteris paribus" at the beginning or the end of the sentence containing your interpretation.

1.7. Creating a Regression Table

To present the results from all four regression models in a structured format, we will use the `stargazer()` function from the *stargazer* package. This function generates professional formatted regression tables.

First, install and load the package:

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

Once the package is loaded, the `stargazer()` function can be used to produce a basic table of regression results. The output can be formatted in either plain text or HTML by specifying the type argument:

```
stargazer(Model1, Model2, Model3, Model4, type = "text")
```

```
##
##
================================================================================================
============================
##                                                          Dependent vari-
able:
##
------------------------------------------------------------------------------------------------
--------
##                                                           Life_satisfac-
tion
##                                (1)                       (2)
(3)                      (4)
##
------------------------------------------------------------------------------------------------
----------------------------
## Age
0.015***                                                                  0.021***
##
(0.002)                                                                   (0.002)
##

## Income
0.207***                                        0.244***
##
(0.019)                                        (0.021)
##

## EDU
0.055**                   -0.009
##
(0.024)                   (0.026)
##
```

```
## Constant                       6.695***                  6.462***
7.271***                 5.192***
##                                (0.134)                    (0.107)
(0.120)                  (0.211)
##
##
-------------------------------------------------------------------------------------------
----------------------------
## Observations                   1,781                      1,735
1,730                    1,675
## R2                             0.023                      0.062
0.003                    0.105
## Adjusted R2                    0.023                      0.062
0.002                    0.104
## Residual Std. Error    1.729 (df = 1779)          1.698 (df = 1733)        1.733 (df =
1728)       1.641 (df = 1671)
## F Statistic          42.756*** (df = 1; 1779) 114.823*** (df = 1; 1733) 5.193** (df =
1; 1728) 65.484*** (df = 3; 1671)
##
===========================================================================================
============================
##
Note:
 *p<0.1; **p<0.05; ***p<0.01
```

Note: when using type = "text", the alignment of the regression output may not render properly in knitted documents (e.g., HTML or Word). Therefore, I recommended to use type = "html" for better formatting in output documents.

To render the HTML output correctly within an R Markdown document, include the argument `results = 'asis'` in the code chunk options:

```
{r, results='asis'}
stargazer(Model1, Model2, Model3, Model4, type = "html")
```

Table 6.2

Dependent variable:				
	Life_satisfaction			
Dependent variable:	1)	(2)	(3)	(4)
Age	0.015*** (0.002)			0.021*** (0.002)
Income		0.207*** (0.019)		0.244*** (0.021)
EDU			0.055** (0.024)	-0.009 (0.026)
Constant	6.695*** (0.134)	6.462*** (0.107)	7.271*** (0.120)	5.192*** (0.211)
Observations	1,781	1,735	1,730	1,675
R2	0.023	0.062	0.003	0.105
Adjusted R2	0.023	0.062	0.002	0.104
Residual Std. Error	1.729 (df = 1779)	1.698 (df = 1733)	1.733 (df = 1728)	1.641 (df = 1671)
F Statistic	42.756*** (df = 1; 1779)	114.823*** (df = 1; 1733)	5.193** (df = 1; 1728)	65.484*** (df = 3; 1671)

Note: p<0.1; p<0.05; p<0.01

1.7.1 Customising the Regression Table

Below is a more advanced usage of `stargazer()` that demonstrates how to customise the output table:

- Assign a descriptive title to the table.
- Define explicit column labels for each model.
- Specify a caption for the dependent variable (e.g., Life Satisfaction (1–10 scale)).
- Add customised covariate labels (e.g., Age, Income Scale, Education Level, Intercept) for clarity.
- Disable automatic column and model numbering using `colnames = FALSE` and `model.numbers = FALSE`.

The command below produces a clean and interpretable regression table, suitable for academic presentation:

```
stargazer(Model1, Model2, Model3, Model4, type = "html",
```

```
title = "Four Regression Models Predicting Variation in Life satisfaction",
column.labels = c("Model1", "Model2", "Model3", "Model4"),
colnames = FALSE,
model.numbers = FALSE,
dep.var.caption = "Life satisfaction (1-10 scale)",
dep.var.labels = "",
covariate.labels=c("Age","Income Scale","Education Level","Intercept"))
```

Table 6.3 Four Regression Models Predicting Variation in Life satisfaction

Life satisfaction (1-10 scale)				
	Model1	Model2	Model3	Model4
Age	0.015*** (0.002)			0.021*** (0.002)
Income Scale		0.207*** (0.019)		0.244*** (0.021)
Education Level			0.055** (0.024)	-0.009 (0.026)
Intercept	6.695*** (0.134)	6.462*** (0.107)	7.271*** (0.120)	5.192*** (0.211)
Observations	1,781	1,735	1,730	1,675
R2	0.023	0.062	0.003	0.105
Adjusted R2	0.023	0.062	0.002	0.104
Residual Std. Error	1.729 (df = 1779)	1.698 (df = 1733)	1.733 (df = 1728)	1.641 (df = 1671)
F Statistic	42.756*** (df = 1; 1779)	114.823*** (df = 1; 1733)	5.193** (df = 1; 1728)	65.484*** (df = 3; 1671)

Note: p<0.1; p<0.05; p<0.01

Quiz. Based on the regression output presented above, what are the key differences among the four models we previously estimated? In particular, consider the following elements:

- The coefficients of each independent variable

- The number of observations used in each model
- The R-squared and adjusted R-squared values
- The F-statistic

Write your responses here:

6.

INTRODUCTION TO BIVARIATE LINEAR REGRESSION

This chapter introduces the logic of bivariate regression analysis. We will analyse real-world data on physical assaults in Australia from 2008 to 2022 to examine whether there is a trend or pattern in the number of victims over time. Before conducting the regression analysis, we will begin by exploring the data using scatter plots and calculating the correlation coefficient to assess linearity. We will then proceed with a bivariate regression analysis, focusing on how to interpret the results. Afterwards, you will apply the same analytical approach to the World Values Survey (WVS) dataset by conducting your own regression analysis.

1. What is a Linear Regression?

Linear regression is a statistical method used to model the relationship between a dependent variable (Y) and one or more independent variables (X) by fitting a linear equation to the observed data.

A simple linear regression model follows the equation:

$$Y = \alpha + \beta * X$$

- Y is the dependent variable,
- X is the independent variable,
- $alpha$ is the intercept (the expected value of Y when X is held constant (X=0),
- $beta$ is the slope (the change in Y associated with a 1-unit increase in X)

Quick note: In regression analysis, certain terms are used interchangeably, though their usage may vary depending on the context:

Dependent = outcome = response variables

Independent = explanatory = predictor variables

2. A Bivariate Linear Regression Model

Bivariate linear regression is a type of linear regression where a single independent variable (X) is regressed on a single dependent variable (Y). The term "bivariate" refers to the presence of two variables—one predictor and one outcome. When multiple independent variables are included, the model becomes a multiple linear regression. Additionally, regression models can be nonlinear. We will explore these different types of regression in the coming Chapters.

Exercise 1 – Bivariate Linear Regression with ABS Data

In this first exercise, we will practice bivariate regression by using the Australian Bureau of Statistics (ABS). Specifically, we will analyse the **victim_rate** dataset, which was previously introduced in Chapter 3 (Section 1.2: Measure of Dispersion). If you have not saved this dataset, please refer back to the materials.

We will focus on *physical assault* and examine its relationship with the *year* variable. Let's load the dataset and display the first ten observations to ensure the data has been imported correctly.

```
#Set up the working directory
setwd("C:/Your folder path/SOCYR")
Victim <- read.csv("Victim_rate.csv")
```

```
#Load the required packages for Chapter 5library(ggplot2)
library(dplyr)
```

```
head(Victim, 10)
```

	Year	Physical_assault	FacetoFace_threatened_assault
1	2008	3.1	3.9
2	2009	2.9	3.1
3	2010	2.7	3.1
4	2011	3	3.3
5	2012	2.7	2.8
6	2013	2.3	2.7
7	2014	2.1	2.6
8	2015	2.4	2.6
9	2016	2.4	2.6
10	2017	2.4	2.6

2.1 Graphical Analysis

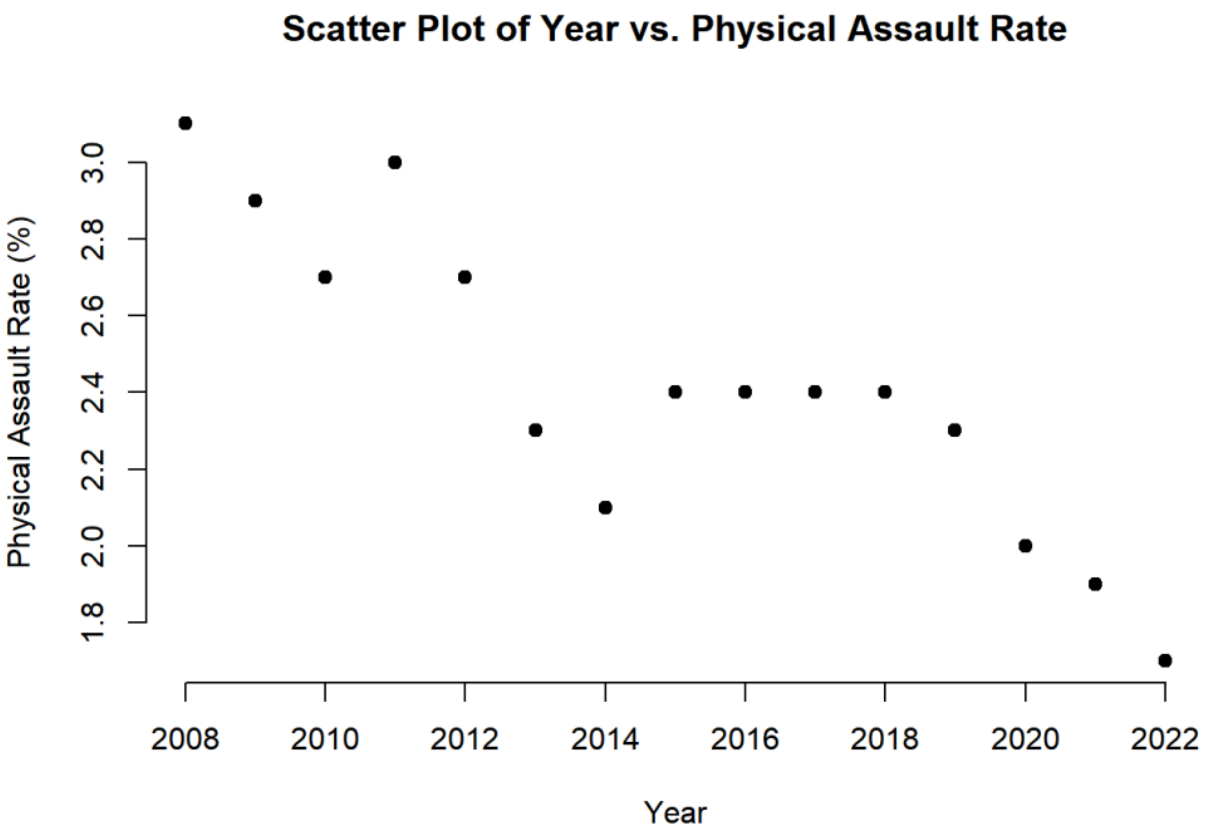
Before proceeding with the regression analysis, let's explore the patterns and relationships between the variables visually. Graphical analysis helps to identify potential linear trends, or outliers. We will use scatter plots to visualize the relationship between the independent (*Year*) variable and the outcome (*Physical Assault*) variable.

The basic structure of `plot()` function is as follows:

```
plot(x, y,
    main = "Title of the plot",
    xlab = "Label for the x-axis",
    ylab = "Label for the y-axis",
    pch = "point shape" (e.g., 19 is a circle),
    frame = FALSE "remove a box around the plot")
```

To visualise the relationship between *Year* and *Physical Assault*:

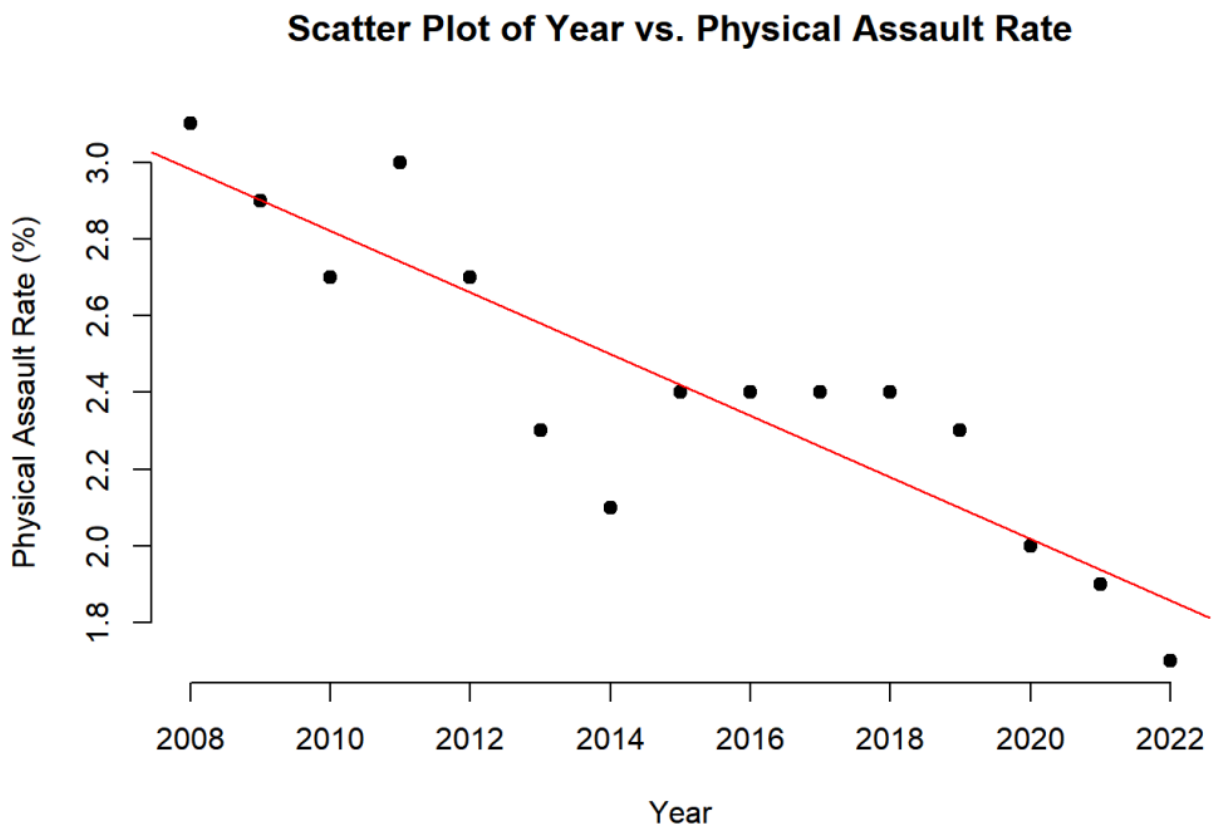
```
plot(x = Victim$Year, y = Victim$Physical_assault,
     main = "Scatter Plot of Year vs. Physical Assault Rate",
     xlab = "Year", ylab = "Physical Assault Rate (%)",
     pch = 19, frame = FALSE)
```



We can also overlay a regression line onto the scatter plot to help to visually confirm whether there is an increasing or decreasing trend in physical assault rates over time.

To add a regression line, we use the `abline()` function:

```
plot(x = Victim $Year, y = Victim $Physical_assault,
     main = "Scatter Plot of Year vs. Physical Assault Rate",
     xlab = "Year", ylab = "Physical Assault Rate (%)",
     pch = 19, frame = FALSE)

# Add the regression line in red
abline(lm(Physical_assault ~ Year, data = Victim), col = "red")
```



Q. Interpret the plot above by describing any patterns or trends.

Write your response here:

2.2 Correlation Analysis

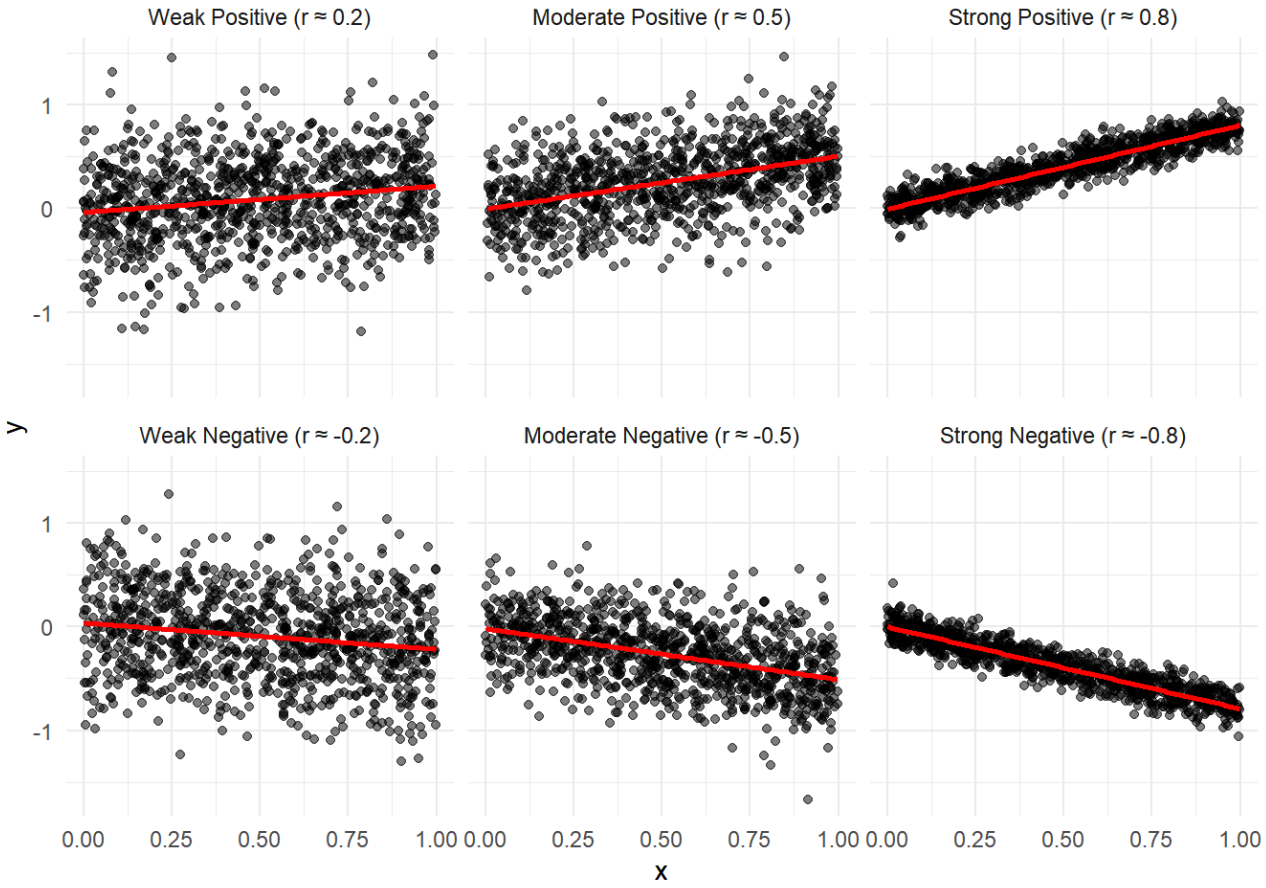
Beyond visualising relationships with scatter plots, we can also quantify the strength and direction using correlation. The correlation coefficient provides a numerical measure of the linear dependence between two variables.

2.2.1 Calculating the Correlation Coefficient

The Pearson correlation coefficient is commonly used to assess the strength of a linear relationship between two continuous variables. The correlation coefficient ranges from -1 to 1:

- r > 0 indicates a positive correlation (as one variable increases, the other tends to increase).
- r < 0 indicates a negative correlation (as one variable increases, the other tends to decrease).
- r ≈ 0 suggests little to no linear relationship between the variables.

The strength of a correlation is typically classified as weak (≈ 0.2), moderate (≈ 0.5), or strong (≈ 0.8). These thresholds are commonly used in social sciences and may vary slightly based on the research context.



Let's compute the correlation between *Year* and *Physical Assault* using the `cor()` function:

```
cor(Victim$Year, Victim$Physical_assault, use = "pairwise.complete.obs")
```

```
## [1] -0.8897243
```

You can test the statistical significance of this correlation coefficient using the `cor.test()` function.

```
#To test statistical significance.
cor.test(Victim$Year, Victim$Physical_assault, use = "pairwise.complete.obs")
```

```
##
##  Pearson's product-moment correlation
##
## data:  Victim$Year and Victim$Physical_assault
## t = -7.0273, df = 13, p-value = 8.969e-06
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9630536 -0.6935732
## sample estimates:
##        cor
## -0.8897243
```

Q. What is the correlation coefficient between the two variables, and is the relationship statistically significant based on the hypothesis test?

Write your response here:

2.3 Fitting a Bivariate Linear Regression Model

We can now build a bivariate regression model using the `lm()` function. This function requires two key arguments: the regression formula and the dataset. The basic syntax is:

```
lm (y~x, data = "yourdata")


Formula `(y ~ x)`: Specifies the dependent and independent variables. The outcome vari-
able is placed
before the tilde (~), and the explanatory, covariate, or control variables are placed
```

```
after it.


Data (`data = "yourdata"`): Typically a data.frame containing the dataset.
```

The formula is usually written directly within the function call, as shown below:

```
# Build a bivariate regression model: lm(y~x, data=)


Reg <- lm(Physical_assault ~ Year, data= Victim)
Reg
```

```
##
## Call:
## lm(formula = Physical_assault ~ Year, data = Victim)
##
## Coefficients:
## (Intercept)         Year
##   164.33964     -0.08036
```

2.3.1 Build Linear Model

By fitting the model, we establish a mathematical relationship between the independent and outcome variable. The output provides key coefficients: **Intercept**: 164.34, **Year coefficient**: -0.08

This allows us to write down the prediction equation as:

$$\hat{y} = \text{Intercept} + (\text{Year coefficient} * \text{Year})$$

Substituting the given values:

$$\hat{y} = 164.34 - 0.08 * \text{Year}$$

Q. Now, use this equation to calculate the predicted physical assault rates for the following years: (i) 2008, (ii) 2014 and (iii) 2022.

Write your response here:

Precited physical assault in 2008:

Precited physical assault in 2014:

Precited physical assault in 2022:

Now that the regression model is built, we can use it to predict physical assault rate for a given year. However, before using the model, we must check its statistical significance. You can check the summary statistics for Reg using the `summary()` function.

```
summary(Reg)
```

```
##
## Call:
## lm(formula = Physical_assault ~ Year, data = Victim)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40036 -0.07982 -0.00214  0.12911  0.25857
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 164.33964   23.04160   7.132 7.68e-06 ***
## Year         -0.08036    0.01144  -7.027 8.97e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1913 on 13 degrees of freedom
## Multiple R-squared:  0.7916, Adjusted R-squared:  0.7756
## F-statistic: 49.38 on 1 and 13 DF,  p-value: 8.969e-06
```

The summary statistics above provide key insights, including **estimates**, **standard errors**, **t-statistics**, and **p-values** for the intercept and the slope. Additionally, they include the **residual standard error**, **R-squared**, and **Adjusted R-squared**, which assess the overall model fit.

Here, we will go through these components step by step to understand how to interpret the results.

2.3.2 Coefficients Estimate

In the coefficients section, you will find the estimates for both the intercept and the slope (Year).

- The intercept represents the predicted physical assault rate when Year = 0.
- The slope indicates the change in physical assault rates for a one-unit increase in Year. A positive slope suggests an increasing trend, while a negative slope indicates a decreasing trend.

Q. Based on the estimated coefficients, interpret the effect of Year on the outcome. What do the intercept and slope represent?

Write your response here:

2.3.3 Confidence Interval

You can also examine the confidence intervals (CIs) for the intercept and slope coefficients. While the summary output does not include this information, you can use the `confint()` function.

```
confint(Reg) #Defualt is 95%
```

```
##                  2.5 %       97.5 %
## (Intercept) 114.561303 214.11798265
## Year         -0.105061  -0.05565331
```

```
round(confint(Reg),2) #Display only the second decimal place
```

```
##              2.5 % 97.5 %
## (Intercept) 114.56 214.12
```

```
## Year         -0.11  -0.06
```

```
#confint(Reg, level = 0.90) #Confidnce intervals at 90%
#confint(Reg, level = 0.99) #Confidnce intervals at 99%
```

Intercept [114.56, 214.12]: This means that we are 95% confident that the true intercept value lies between 114.56 and 214.12.

Year [-0.11, -0.06]: This means that we are 95% confident that the true slope lies between -0.11 and -0.06.

***Note. If the confidence interval includes the value 'zero' (e.g., -0.02 to 0.06), it suggests that the effect of Year may not be statistically significant.**

2.3.4 The T-statistic and its P-value

Assessing the statistical significance of the estimated coefficients is crucial. We can determine this by examining the t-statistic and its corresponding p-value, which indicate whether the relationship between the independent and dependent variable is meaningful or merely a result of random variation.

Hypothesis Testing: Every p-value in regression analysis corresponds to a hypothesis test involving a null and an alternative hypothesis. The hypotheses for each coefficient are as follows:

- Null Hypothesis (H_0): The coefficient of the explanatory variable is equal to zero ($\beta = 0$), meaning there is no relationship between the explanatory and the outcome variable.
- Alternative Hypothesis (H_1): The coefficient of the explanatory variable is not equal to zero ($\beta \neq 0$), indicating that there exists a statistically significant relationship between the explanatory variable and the outcome variable.

Understanding the t-value

The t-statistic (t-value) measures how many standard deviations the estimated coefficient is away from zero:

$$t = \frac{\hat{\beta}}{SE(\hat{\beta})}$$

* $\hat{\beta}$: The estimated coefficient, $SE(\hat{\beta})$: The standard error of the coefficient

A larger t-value indicates stronger evidence against the null hypothesis (β = 0), meaning the coefficient is likely to be statistically significant.

Understanding the p-value

The p-value (Pr(>|t|)) represents the probability of obtaining a t-value as extreme as the one observed, assuming the null hypothesis is
true. The standard significance level is typically set at 0.05, corresponding to a 95% confidence level.

- If p < 0.05, we reject the null hypothesis, indicating that the coefficient of the predictor (β) is significantly different from zero. This suggests that the predictor has a statistically significant effect on the dependent variable.
- If p > 0.05, we fail to reject the null hypothesis, meaning there is insufficient evidence to conclude that the predictor has a significant effect. In this case, we assume that β may be equal to zero, implying no statistically significant relationship with the outcome variable.

Returning to our results, we now answer the following questions:

Q1. What are the t-values for both the intercept and slope?

Q2. What are the p-values for both the intercept and slope?

Q3. What is the null hypothesis, and can it be rejected at the 95% confidence level?

Write your responses here:

2.3.5 R-Squared (R^2)

Regression models aim to explain the variation in the dependent variable based on one or more independent (predictor) variables. R-squared (R^2) is commonly used method to evaluate the explanatory power of a regression model. R-squared measures the proportion of the variance in the dependent variable that is explained by the independent variable(s) in the model:

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

*SSE (Sum of Squared Errors): Measures the unexplained variation (the residual sum of squares),

SST (Total Sum of Squares): Represents the total variation in the dependent variable.

A high R^2 value suggests that the predictor variable(s) effectively explain the variation in the outcome, while a low R^2 indicates that much of the variation remains unexplained.

- R^2 = 1 (or 100%) → The model explains all the variability in the dependent variable.
- R^2 = 0 (or 0%) → The model explains none of the variability in the dependent variable.

Again, let's return to your results and answer the questions below:

Q1. What is the R-squared value?

Q2. How do you interpret it?

Write your responses here:

2.3.6 Adjust R-Squared ($\bar{R}^2$)

One limitation of R^2 is that it **always** increases when more predictors are added to a model, even if the new predictors contribute little to explaining the variance. This makes it unreliable for comparing models with different numbers of predictors. To address this, Adjusted R-squared ($\bar{R}^2$) modifies R^2 by penalizing the inclusion of additional predictors that do not improve model fit. The formula is:

$$\bar{R}^2 = 1 - \frac{\text{MSE}}{\text{MST}}$$

*MSE (Mean Squared Errors): $SSE/(n-k-1)$, accounting for model complexity,

MST (Mean Squared Total): $SST/(n-1)$, accounting for total variation,

n: sample size, k: the number of explanatory variables in the model.

Key difference between them is $\bar{R}^2$ penalizes unnecessary predictors, providing a more accurate measure of the model's explanatory power. If a new predictor contributes significantly, $\bar{R}^2$ value will increase; otherwise, it may decrease.

Note:

In practice, a high R^2 **alone** does not guarantee a good model. We must also consider factors such as p-values, residual diagnostics, and model assumptions before drawing conclusions. Adding too many predictors may increase the R^2 or Adjusted R^2 value, but it can also lead to overfitting. Therefore, you should carefully select which predictors to include or exclude in the model.

You should also consider standardising the variables, which means rescaling them to improve interpretability. To do this, use the `scale()` function as shown below:

```
Standreg <- lm(scale(Physical_assault) ~ scale(Year), data= Victim)
Standreg
```

Exercise 2 – Bivariate Linear Regression Using WVS Dataset: Running Your Own Models

So far, we have practiced bivariate linear regression analysis using real-world data. Now, it is your turn to apply these concepts independently. Using the World Values Survey (WVS) dataset, you will run a regression model and interpret the results.

For this exercise, you will develop a regression model where: dependent variable is *Life Satisfaction* (Q49), and independent variable: *Age* (Q262) Using the R code from **Exercise 1** as a reference, your task is to create an R Markdown file, conduct the analysis, and export the results into an HTML or Word document.

Please follow the below steps:

1. Open a new R Markdown file.
2. Import and view the summary statistics:
    1. Selecting only Life_satisfaction and Age variables.
    2. Checking and ensuring the data type.
    3. Checking the summary statistics

3. Create scatter plots:
    1. Generate a scatter plot without a fitted line to visualize the relationship.
    2. Generate a scatter plot with a fitted regression line.
    3. Interpret the outcomes.

4. Calculate the correlation coefficient:
    0. Compute the correlation coefficient2.
    1. Test whether the correlation is statistically significant.
    2. Interpret the results.

5. Fit a bivariate regression model where *Life_satisfaction* is the dependent variable and *Age* is the independent variable.

6. Estimated the predicted Life Satisfaction for individuals aged:
    1. 29
    2. 41
    3. 55

7. Interpret the regression results, including:
    1. Estimated coefficient for Age
    2. t-value for Age
    3. p-value for Age
    4. Hypothesis testing for Age
    5. R-squared and Adjusted R-squared values

8. Knit the R Markdown file to export all results into an HTML or Word document.

You are strongly encouraged to complete the exercise on your own.

Once you complete all exercises, make sure to save your R script before closing R.

7.

CATEGORICAL PREDICTORS IN REGRESSION MODEL

In this chapter, we will continue our work on multiple linear regression by extending the analysis to include categorical predictors (i.e., discrete or polytomous variables). We will demonstrate how these variables can be incorporated using appropriate coding techniques (e.g., dummy coding). We will then calculate a marginal effect and draw its plot.

1. Multiple Regression with Categorical Predictors

In this exercise, you will learn how to build and interpret categorical variables within a multiple regression model. This is an important skill, as many of the factors that social scientists are interested in tend to be categorical (e.g., gender, marital status, and region or country of residence).

Categorical variables, also referred to as factors in R, classify observations into distinct groups. These variables consist of a limited number of values known as levels. For example, *sex* is a categorical variable with two levels: Male and Female, while *marital status* can be categorised into three levels: Married, Divorced, and Single.

Your dataset may contain: 1) binary (dichotomous) variables, which have exactly two values (e.g., sex: male or female), 2) polytomous variables, which have more than two values (e.g., marital status: married, divorced, single), or a combination of both.

Exercise 1 – Categorical Variable with two Levels (Dichotomous/Binary Variables)

1.1 Dummy Variables

In R, categorical variables are typically encoded as **dummy** variables. Dummy variable is an independent

variable which take the value of either 0 or 1. By default, the baseline (or reference) category is assigned a value of 0, while each comparison category is assigned a value of 1.

For a binary variable such as Sex in WVS dataset:

- The reference category is coded as 0 (e.g., Male),
- The comparison category is coded as 1 (e.g., Female).

Note: even if your raw data uses 1 for male and 2 for female, R will automatically handle the factor levels internally when you include the variable in a regression model. You do not need to manually recode it to 0 and 1.

Recall that, the regression equation, it can be simply written as
$y = a + b * x$. Suppose we include Sex
as a binary predictor in the regression model. The regression model equation is:

$$\hat{Lifesatisfaction} = a + b * Sex$$

- When Sex = 0 (male): $\hat{Lifesatisfaction} = a + b * 0(male) = a$

When Sex = 1 (female):

$$\hat{Lifesatisfaction} = a + b_1 * 1(female) = a + b_1$$

- a: The average life satisfaction score among males (the reference group),
- b: The average difference in life satisfaction between females and males,
- a + b: The average life satisfaction score among females.

1.2 Encoding the Dummy Variables

In order to create a dummy variable for the categorical variable, the simplest way is to change the data type to factor using the `as.factor()` function. R creates dummy variables automatically if the data type is factor. You can also use the `factor()` function which you can assign the labels for each value.

Let's load the dataset and inspect the dataset to ensure it has been imported correctly.

```
#Set up the working directory first
```

```
setwd("C:/Your folder path/SOCYR")
```

```
WVS <- read.csv("WVS.csv")
```

```
#Load the required packages for Chapter 7
library(corrplot)
library(psych)
library(dplyr)
library(ggplot2)
```

```
AUS_data <- WVS %>%
  filter(Country == 36)  %>% #Filter only Australian participants
  select(Life_satisfaction, Age, Sex, Marital_Status, EDU)

#Check the data type
str(AUS_data)
```

```
## 'data.frame':    1813 obs. of  5 variables:
##  $ Life_satisfaction: int  7 10 8 10 8 6 8 9 5 8 ...
##  $ Age              : int  60 41 43 48 57 25 62 78 44 83 ...
##  $ Sex              : int  1 1 2 2 1 2 1 1 1 1 ...
##  $ Marital_Status   : int  1 1 1 1 1 6 3 1 1 3 ...
##  $ EDU              : int  3 7 7 4 7 6 8 6 6 2 ...
```

Sex variable's data type is integer, so let's change to factor type.

```
#Simply way to change the data type as factor
AUS_data$Sex <- as.factor(AUS_data$Sex)

# Convert Sex into a factor with labels
AUS_data$Sex <- factor(AUS_data$Sex, levels = c(1, 2), labels = c("Male", "Female"))
```

You can check the reference category using `levels()` function. The first level (here male) shown is the reference category by default.

```
# check the current reference category
levels(AUS_data$Sex)
```

```
## [1] "Male"   "Female"
```

1.3 Fitting Multiple Regression Models with Categorical Predictor

Let's fit a multiple regression model with categorical *Sex* variable. The following model is bivariate regression outcome of the life satisfaction difference between males and females.

```
#Run a bivariate regression
Female <- lm(Life_satisfaction ~ Sex, data = AUS_data)
summary(Female)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Sex, data = AUS_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.5947 -0.5947  0.4053  1.4053  2.5731
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.42693    0.06621 112.170   <2e-16 ***
## SexFemale    0.16774    0.08483   1.977   0.0482 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.749 on 1784 degrees of freedom
##   (27 observations deleted due to missingness)
## Multiple R-squared:  0.002187,   Adjusted R-squared:  0.001627
```

```
## F-statistic:  3.91 on 1 and 1784 DF,  p-value: 0.04817
```

Based on the output above, the coefficient (*SexFemale*) is 0.17 which is the average difference in life satisfaction between females and males. The p-value of *SexFemale* is statistically significant ($p < 0.05$), indicating evidence of difference in average life satisfaction between sex. Therefore, we can say that the female group tends to report life satisfaction that is 0.17 points higher than the male group.

You can calculate predicted life satisfaction value by sex by applying the equation as below:

- For male group: the average life satisfaction score for males is estimated at 7.43.

$$\hat{Lifesatisfaction} = 7.43 + 0.17 * 0(male) = 7.43$$

- For female group, the estimated average is 7.6.

$$\hat{Lifesatisfaction} = 7.43 + 0.17 * 1(female) = 7.6$$

1.4 Changing the Baseline (Reference) Category

In R, the reference category for categorical variables is, by default, the first level alphabetically or numerically when the variable is coded as a factor. You can change the baseline category for a categorical variable. For example, setting *Female* instead of *Male* as the reference group. While changing the reference category does not affect the overall model fit or computation, it does change the interpretation of the regression coefficients.

To change the reference group, you can use the `relevel()` function:

```
AUS_data$Sex <- relevel(AUS_data$Sex, ref = "Female")
levels(AUS_data$Sex)
```

```
## [1] "Female" "Male"
```

The reference category has been changed to Female. Let's re-run the regression and examine how the interpretation of the coefficients differs.

```
Male <- lm(Life_satisfaction ~ Sex, data = AUS_data)
summary(Male)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Sex, data = AUS_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.5947 -0.5947  0.4053  1.4053  2.5731
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.59467    0.05303 143.207   <2e-16 ***
## SexMale     -0.16774    0.08483  -1.977   0.0482 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.749 on 1784 degrees of freedom
##   (27 observations deleted due to missingness)
## Multiple R-squared:  0.002187,   Adjusted R-squared:  0.001627
## F-statistic:  3.91 on 1 and 1784 DF,  p-value: 0.04817
```

The output shows that not only estimate of intercept is different, but also the coefficient for *SexMale* is negative, indicating that being male is associated with a decrease in life satisfaction relative to females (the reference group). The p-value associated of *SexFemale* is statistically significant ($p < 0.05$), indicating evidence of difference in average life satisfaction between sex. Therefore, we can say that the male group tends to report life satisfaction that is 0.17 points lower than the female group.

You can also calculate predicted life satisfaction value with the model where female is reference category by applying the equation as below:

- For male group: the average life satisfaction score for males is estimated at 7.42.

$$\hat{Lifesatisfaction} = 7.59 + -0.17 * 1(male) = 7.42$$

- For female group, the estimated average is 7.59.

$$\hat{Lifesatisfaction} = 7.59 + -0.17 * 0(female) = 7.59$$

1.4.1 Strategies for Choosing the Reference Category in Dummy Coding

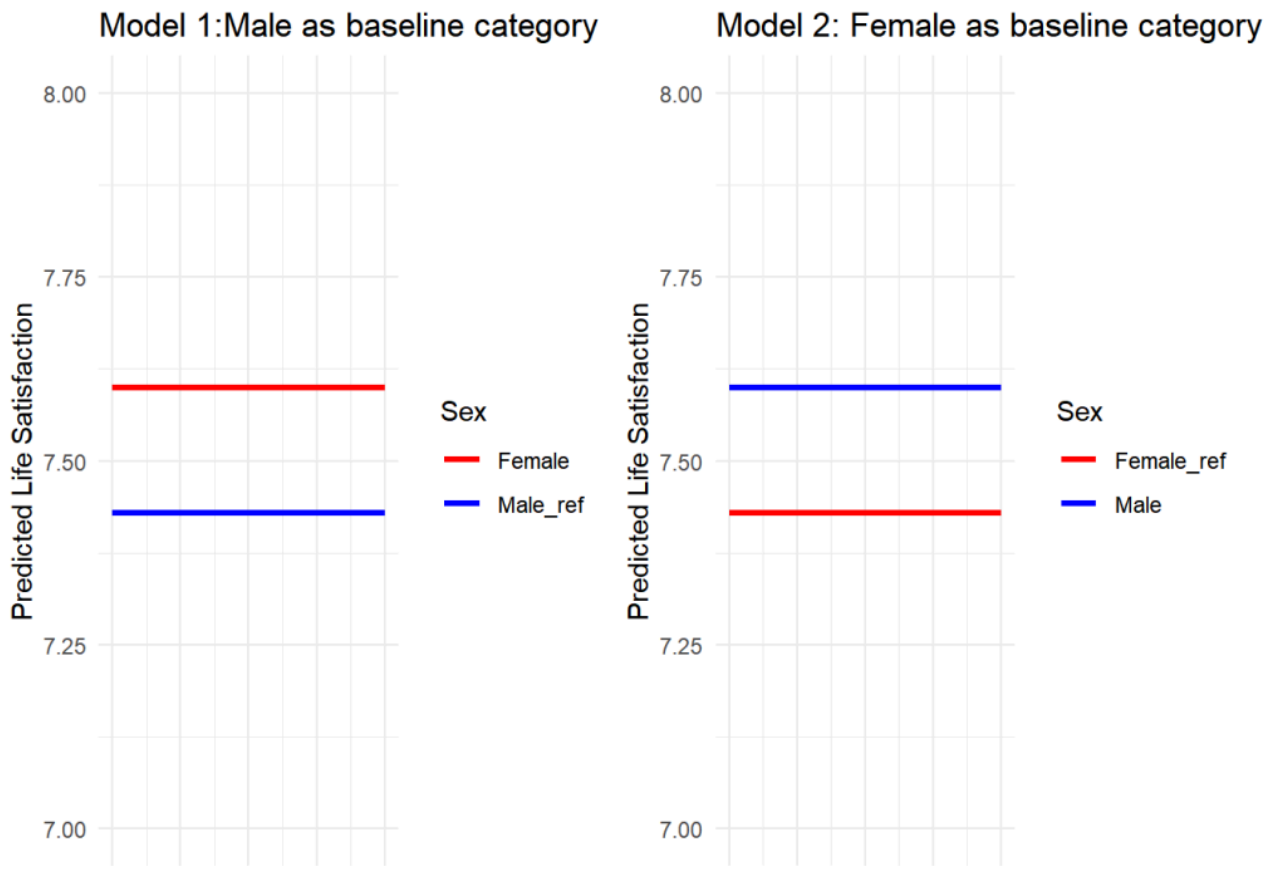
The best choice is to select a category that makes interpretation of the results easier and aligns with your research aims. While the choice of reference group does not critically affect the statistical estimation itself, using a category with a very small sample size may lead to statistical issues such as unstable estimates. In such cases, it is better to choose a reference category with a reasonably large or the largest sample size.

1.5 Visualising the Differences

You can also visualise the results by creating a plot, which offer a more intuitive understanding of the differences between groups.

```
#Male as baseline category
data1 <- data.frame(
      Sex = rep(c("Male_ref", "Female"), each = 2),
      Predicted_LS = c(rep(7.43, 2), rep(7.6, 2)),
      X = rep(1:2, 2)
    )

#Female as baseline category
data2 <- data.frame(
      Sex = rep(c("Male", "Female_ref"), each = 2),
      Predicted_LS = c(rep(7.6, 2), rep(7.43, 2)),
      X = rep(1:2, 2)
    )

plot1 <- ggplot(data1, aes(x = X, y = Predicted_LS, color = Sex, group = Sex)) +
      geom_line(linewidth  = 1.2) +
      labs(y = "Predicted Life Satisfaction",
           title = "Model 1:Male as baseline category") +
      scale_color_manual(values = c("Male_ref" = "blue", "Female" = "red")) +
      scale_y_continuous(limits = c(7, 8)) +
      theme_minimal() +
```

```
      theme(axis.title.x = element_blank(),
            axis.text.x = element_blank())

plot2 <- ggplot(data2, aes(x = X, y = Predicted_LS, color = Sex, group = Sex)) +
      geom_line(linewidth  = 1.2) +
      labs(y = "Predicted Life Satisfaction",
           title = "Model 2: Female as baseline category") +
      scale_color_manual(values = c("Male" = "blue", "Female_ref" = "red")) +
      scale_y_continuous(limits = c(7, 8)) +
      theme_minimal() +
      theme(axis.title.x = element_blank(),
            axis.text.x = element_blank())

# Combine the two plots
#install.packages("patchwork")
library(patchwork)
plot1 + plot2  # side-by-side layout
```



The positions of the lines for males and females appear to have switched between the two plots. This is due

to the change in reference category. However, the predicted life satisfaction scores for each group remain the same across both models.

Exercise 2 – Categorical Variable With More Than two Levels

When a categorical predictor has more than two levels (e.g., *Marital Status*: Married, Divorced, Single, etc.), it must be transformed into dummy variables in order to be included in a regression model. There are two ways to handle this transformation in R.

In our dataset, the variable *Marital_Status* has six categories (1.- Married 2.- Living together as married 3.- Divorced 4.- Separated 5.- Widowed 6.- Single).

For simplicity, we will recode this variable into three categories (Married, Divorced, and Single).

```
AUS_data <- AUS_data %>%
            mutate(NMarital_Status =
             ifelse(Marital_Status == 1, 1,  # If Marital_Status is 1, assign 1 to NMari-
tal_Status
             ifelse(Marital_Status ==3, 2,  # If not 1, but Marital_Status is 3, assign 2
             ifelse(Marital_Status == 6, 3, NA_real_)))) # If not 1 or 3, but Marital_Sta-
tus is 6, assign 3, If none of the above conditions are met (2,4,5), assign NA
```

2.1 Manual Recoding

The first option involves manually creating dummy variables. In general, a categorical variable with n levels is represented by `n - 1` dummy variables in a regression model. One level is selected as the reference category, and the other levels are compared against it.

We can inspect the contrast matrix to see how R encodes the levels by default:

```
contrasts(as.factor(AUS_data$NMarital_Status))
```

```
##   2 3
## 1 0 0
## 2 1 0
## 3 0 1
```

Here, *Married* is treated as the baseline (reference) category. Based on this structure, we can manually create the dummy variables using the `ifelse()` function:

```
# Dummy variable for Divorced (1 if Divorced, 0 otherwise)
AUS_data$Divorced <- ifelse(AUS_data$NMarital_Status == 2, 1, 0)


# Dummy variable for Single (1 if Single, 0 otherwise)
AUS_data$Single <- ifelse(AUS_data$NMarital_Status == 3, 1, 0)
```

These dummy variables can now be included as predictors in a regression model, with Married serving as the reference group.

2.2 R Factor Function

The second, more efficient approach is using the `factor()` function, as you have learned in section 1.2. R will automatically create the appropriate dummy variables during model estimation.

```
AUS_data$NMarital_Status <- factor(AUS_data$NMarital_Status, levels = c(1, 2,3),
                                   labels = c("Married", "Divorced", "Single"))
```

By default, R will treat the first level (*Married*) as the reference category and compare the remaining groups against it.

2.3 Comparison two Options

Now that we have prepared the variable in two different ways (manual dummy creation and factor conversion), we can fit regression models using both methods and compare the outcomes.

```
#Manual calculation
Option1 <- lm(Life_satisfaction ~  Divorced + Single, data = AUS_data)
summary(Option1)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Divorced + Single, data = AUS_data)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8473 -0.8473  0.1527  1.1527  3.1010
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.84728    0.05342 146.903  < 2e-16 ***
## Divorced    -0.53522    0.14900  -3.592 0.000339 ***
## Single      -0.94826    0.10835  -8.752  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.652 on 1401 degrees of freedom
##   (409 observations deleted due to missingness)
## Multiple R-squared:  0.0546, Adjusted R-squared:  0.05325
## F-statistic: 40.46 on 2 and 1401 DF,  p-value: < 2.2e-16
```

```
#R function
Option2 <- lm(Life_satisfaction ~   NMarital_Status, data = AUS_data)
summary(Option2)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ NMarital_Status, data = AUS_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8473 -0.8473  0.1527  1.1527  3.1010
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                7.84728    0.05342 146.903  < 2e-16 ***
## NMarital_StatusDivorced   -0.53522    0.14900  -3.592 0.000339 ***
## NMarital_StatusSingle     -0.94826    0.10835  -8.752  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1.652 on 1401 degrees of freedom
##   (409 observations deleted due to missingness)
## Multiple R-squared:  0.0546, Adjusted R-squared:  0.05325
## F-statistic: 40.46 on 2 and 1401 DF,  p-value: < 2.2e-16
```

```
#Create a regression table with the `tab_model()` function
#install.packages("sjplot")
library(sjPlot)
```

```
## Learn more about sjPlot with 'browseVignettes("sjPlot")'.
```

```
tab_model(Option1, Option2,
          dv.labels = c("Model 1: Option2", "Model 2: Option2"))
```

	Model 1: Option2			**Model 2: Option2**		
Predictors	Estimates	CI	p	Estimates	CI	p
(Intercept)	7.85	7.74 – 7.95	0.001	7.85	7.74 – 7.95	0.001
Divorced	-0.54	-0.83 – -0.24	0.001			
Single	-0.95	-1.16 – -0.74	0.001			
NMarital Status [Divorced]				-0.54	-0.83 – -0.24	0.001
NMarital Status [Single]				-0.95	-1.16 – -0.74	0.001
Observations	1404			1404		
R2 / R2 adjusted	0.055 / 0.053			0.055 / 0.053		

Quiz. Is there any difference between the two regression outcomes?

Write your response here:

2.4 Multiple Regression Model with all Categorical Variables

Now, let's include all the independent variables we have used so far to estimate a final multiple regression model.

```
Full_Reg <- lm(Life_satisfaction ~ Age + EDU + Sex + NMarital_Status, data = AUS_data)
summary(Full_Reg)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Age + EDU + Sex + NMarital_Status,
##     data = AUS_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8406 -0.7725  0.1914  1.0855  3.4003
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                6.84194    0.23499  29.116  < 2e-16 ***
## Age                        0.01485    0.00297   5.000 6.48e-07 ***
## EDU                        0.05881    0.02674   2.199 0.028015 *
## SexMale                   -0.24535    0.09138  -2.685 0.007341 **
## NMarital_StatusDivorced -0.57394    0.15246  -3.765 0.000174 ***
## NMarital_StatusSingle   -0.69314    0.11476  -6.040 2.00e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.613 on 1344 degrees of freedom
##   (463 observations deleted due to missingness)
## Multiple R-squared:  0.07061,    Adjusted R-squared:  0.06715
## F-statistic: 20.42 on 5 and 1344 DF,  p-value: < 2.2e-16
```

Using the output above, answer the following questions:

Q1. How many observations contribute to the regression model?

Q2. What's the predicted life satisfaction for each married, divorced, and single?

Q3. What variables are statistically significant? Fully interpret the significant variables

Q4. How would you interpret the results of the F-test?

Q5. What are the values for R2 and adjusted R2 for this model? What do these tell you?

Write your responses here:

Exercise 3 – Marginal Predictions

You can also calculate marginal effect to calculate model predictions for Y at different values of the X variables. Marginal effects measure how a change in a predictor variable affects the outcome variable, while holding other variables constant or averaging over their observed values.

3.1 What's the Difference Between Regression Outcome and Marginal Effect?

Regression coefficients and marginal effects both quantify the relationship between predictors and outcomes, but they answer different questions and are interpreted differently, especially in nonlinear models (e.g., logistic regression).

- Linear Regression: The expected change in the outcome for a 1-unit change in the predictor, holding other variables constant.
- Marginal effects: The actual change in the outcome (on its original scale) for a 1-unit change in a predictor, averaged across the data (Average Marginal Effect) or at specific values (Marginal Effect at the Mean).

In linear regression model, there is no big difference between regression outcome and marginal effect. However, if your model is a non-linear regression model, it is less intuitive to interpret the model outcome. Therefore, it would be more beneficial to use marginal effects for interpretations. We will discuss this further in Chapter 9.

3.2 Marginal Effects

Let's practice the marginal effects with the full multiple regression model. We will request the predicted value of the outcome variable (*Life satisfaction*) by *Sex* using the `prediction()` function.

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

```
#Calculate the predicted value of life satisfaction by sex
predictions(Full_Reg, by = "Sex")
```

```
##
##     Sex Estimate Std. Error   z Pr(>|z|)   S 2.5 % 97.5 %
##  Male       7.51     0.0689 109   <0.001 Inf  7.38   7.65
##  Female     7.66     0.0570 135   <0.001 Inf  7.55   7.77
##
## Columns: Sex, estimate, std.error, statistic, p.value, s.value, conf.low, conf.high
## Type:  response
```

The outcome above shows the average predicted life satisfaction for males (7.51) and females (7.66), based on the Sex variable. However, here we have not accounted for the other variables in the models, this is the predicted values not the marginal effects.

By including other variables in the marginal effect function, we can bypass the manual calculation that we have previously undertaken using the prediction equation. If you want to compare the predicted life satisfaction outcome by sex between the manual equation and using marginal effect.

Equation:

$$\hat{Lifesatisfaction} = a + b_1 * female + b_2 * Age + b_3 * EDU + b_4 * Divorced + b5 * Single :$$

Predicted life satisfaction for female: 6.84, for male: 6.6

```
predictions(
  Full_Reg,
  newdata = datagrid(Sex = c("Male", "Female"),
```

```
                    Age = 0,  # Same as intercept
                    EDU = 0,  # Same as intercept
                    NMarital_Status = "Married")  # Reference level
)
```

```
##
##     Sex Age EDU NMarital_Status Estimate Std. Error    z Pr(>|z|)     S 2.5 %
##  Male     0   0         Married     6.60      0.254 26.0   <0.001 491.6  6.10
##  Female   0   0         Married     6.84      0.235 29.1   <0.001 616.7  6.38
##  97.5 %
##    7.09
##    7.30
##
## Columns: rowid, estimate, std.error, statistic, p.value, s.value, conf.low, conf.high,
Sex, Age, EDU, NMarital_Status, Life_satisfaction
## Type:  response
```

You can see that the manual calculation and using marginal effects' the results are same.

Further, we can request that the values of all of the other explanatory variables in the prediction equation (i.e., Age, EDU, and NMarital_Status) are held constant at their respective sample means.

```
predictions(
  Full_Reg,
  newdata = datagrid(
    Sex = c("Male", "Female"),
    Age = mean(AUS_data$Age, na.rm = TRUE),
    EDU = mean(AUS_data$EDU, na.rm = TRUE),
    NMarital_Status = "Married"  # Reference category
  )
)
```

```
##
##     Sex  Age  EDU NMarital_Status Estimate Std. Error   z Pr(>|z|)   S 2.5 %
##  Male   54.3 4.66         Married     7.68     0.0765 100   <0.001 Inf  7.53
```

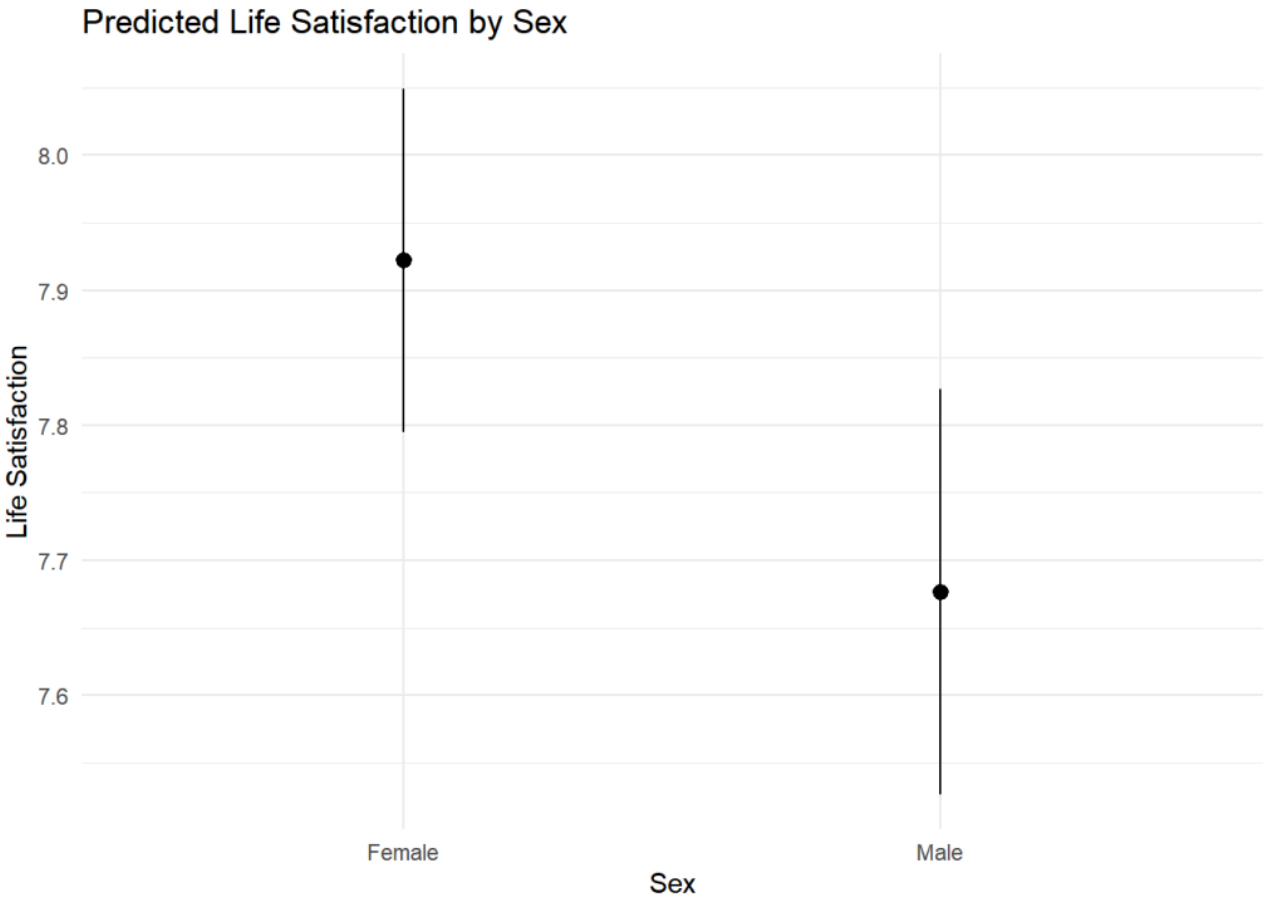
```
##  Female 54.3 4.66         Married     7.92     0.0649 122   <0.001 Inf  7.80
##  97.5 %
##    7.83
##    8.05
##
## Columns: rowid, estimate, std.error, statistic, p.value, s.value, conf.low, conf.high,
Sex, Age, EDU, NMarital_Status, Life_satisfaction
## Type:  response
```

You can see that the predicted life satisfaction for males and females has changed depending on how the other variables are controlled (at their respective sample means). The option that you use depends on your research question and dataset.

3.3 Marginal Effect Plot

The model predictions that we have generated using the margins command can be easily plotted by invoking the `plot_predictions()` functions.

```
plot_predictions(
  Full_Reg,
  by = "Sex",  # Compare Sex = "1" vs. "2"
  newdata = datagrid(
    Sex = c("Male", "Female"),  # Explicitly include both sexes
    Age = mean(AUS_data$Age, na.rm = TRUE),
    EDU = mean(AUS_data$EDU, na.rm = TRUE),
    NMarital_Status = "Married"  # Reference level
  )
) +
  labs(
    title = "Predicted Life Satisfaction by Sex",
    x = "Sex",
    y = "Life Satisfaction"
  ) +
  scale_x_discrete(labels = c("1" = "Male", "2" = "Female")) +
  theme_minimal()
```



Q. How do you interpret the results depicted in the graph?

Write your response here:

Once you complete all exercises, make sure to save your R script before closing R.

8.

STATISTICAL MODERATION AND MEDIATION

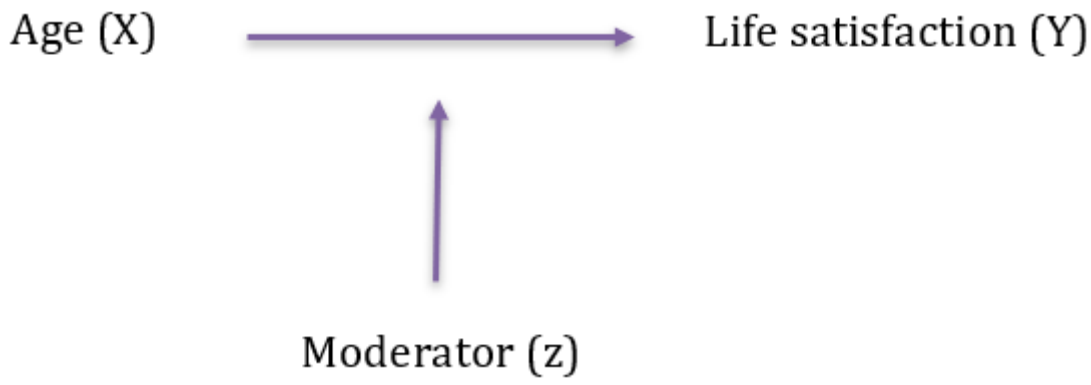
So far, we have focused on relationships between two variables. In this chapter, we extend our analysis to relationships among three variables, with particular attention to statistical moderation and mediation. The goal is to help you develop a clear understanding of these concepts and learn how to interpret and estimate regression models that incorporate them. We will begin by conceptualising moderation and mediation, and then apply these ideas through practical exercises.

1. Moderation Analysis

1.1 What is Moderation?

To introduce moderation, consider a bivariate relationship between an independent variable (e.g., age) and a dependent variable (e.g., life satisfaction). We hypothesise that life satisfaction increases with age, illustrated by a basic diagram:

A moderator variable (Z) influences the strength and/or direction of the relationship between X and Y. For example, the link between age and life satisfaction may vary by sex. If the association is stronger for one sex, sex acts as a moderator. This means the effect of age on life satisfaction depends on whether the individual is male or female. In a diagram, moderation is shown when the slope of the relationship differs across groups.



1.2 Conducting a Moderation Analysis

Moderation analysis can be conducted by including one or multiple interaction terms in a regression analysis. For example, if Z moderates the relationship between X and Y, we can fit a regression model as

$$\hat{Lifesatisfaction} = a + b_1 * X + b_2 * Z + b_3 * (X * Z)$$

If Z represents a binary variable
(e.g., sex), coded as 0 for males and 1 for females:

- For male respondents ($Z = 0$):

$$\hat{Lifesatisfaction} = a + b_1 * X$$

- For female respondents ($Z = 1$):

$$\hat{Lifesatisfaction} = a + b_2 + (b_1 + b_3) * X$$

In this case, b_3 represents the difference in the effect of X on Y between the two groups. If $b_3 \neq 0$, this indicates that the strength or direction of the relationship between X and Y varies by Z , which is a clear sign of a moderation effect.

Exercise 1 – Moderation Effect in R

We have conceptually and mathematically explored the moderation effect. We now turn to a practical application using WVS dataset. The explanatory variable is *Age*, the moderator variable is *Sex*, and the

outcome variable is *Life_satisfaction*. We also include a series of control variables: health status (Q47), education level (*EDU*), marital status (*NMarital_Status*), and income (*Income*).

1.1 Data Preparation

Let's import the dataset and begin by cleaning the control variables:

```
#Set up the working directory first
setwd("C:/Your folder path/SOCYR")


#Load the dataset
WVS <- read.csv("WVS.csv")
```

```
#Load the required packages for Chapter 8
library(dplyr)
library(knitr)
library(sjPlot)
library(marginaleffects)
library(ggeffects)
library(ggplot2)
library(psych)
```

```
New_data <- WVS %>%
             filter(Country == 36)  %>% #Filter only Australian participants %>%
              mutate(NMarital_Status = ifelse(Marital_Status == 1, 1,
                                    ifelse(Marital_Status ==3, 2,
                                           ifelse(Marital_Status == 6, 3, NA_real_))))
New_data <- New_data %>%
  mutate(Health = 6 - Q47)      #Now higher value indicates better health status
```

1.2 Creating Interaction Terms

To examine a moderation effect, the first step is to create an interaction term. There are two common approaches to generating interaction terms: **Manual computation** and using **R syntax**.

1.2.1 Manual Computation

We begin with the manual approach. An interaction term can be created by simply multiplying the two variables.

When creating an interaction with a categorical variable (here *Sex*), ensure it is numeric. If coded as a factor, direct multiplication may cause errors or unpredictable results.

```
New_data$Inter_term <- New_data$Sex * New_data$Age


Model1 <- lm(Life_satisfaction ~ Age + Sex +Inter_term + Income +Health +EDU + NMari-
tal_Status , data = New_data)
summary(Model1)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Age + Sex + Inter_term + Income +
##     Health + EDU + NMarital_Status, data = New_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9636 -0.7168  0.1517  0.9878  3.9311
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.864544   0.555437   3.357 0.000811 ***
## Age              0.037288   0.008235   4.528 6.51e-06 ***
## Sex              0.867092   0.279099   3.107 0.001932 **
## Inter_term      -0.012601   0.004954  -2.543 0.011093 *
## Income           0.118423   0.021962   5.392 8.26e-08 ***
## Health           0.822961   0.051027  16.128  < 2e-16 ***
## EDU             -0.030899   0.025996  -1.189 0.234820
## NMarital_Status -0.220405   0.051395  -4.288 1.93e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.435 on 1301 degrees of freedom
##   (504 observations deleted due to missingness)
```

```
## Multiple R-squared:  0.2656, Adjusted R-squared:  0.2616
## F-statistic: 67.21 on 7 and 1301 DF,  p-value: < 2.2e-16
```

Quiz. Is the interaction term statistically significant? How can you determine this, and interpret the results.

Write your response here:

1.2.2 Using R Syntax

An alternative approach is to include the interaction term directly in the regression model.

By including the interaction term using the * operator (Age * Sex), R will automatically estimate both the main effects of Age and Sex as well as their interaction effect. Sex should be coded as a factor rather than as a numeric.

```
New_data$Sex <- factor(New_data$Sex, levels = c(1, 2), labels = c("Male", "Female"))


Model2 <- lm(Life_satisfaction ~ Age*Sex + Income+ Health + EDU + NMarital_Status , data
= New_data)
summary(Model2)
```

```
##
## Call:
## lm(formula = Life_satisfaction ~ Age * Sex + Income + Health +
##     EDU + NMarital_Status, data = New_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9636 -0.7168  0.1517  0.9878  3.9311
```

```
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.731636   0.363318   7.519 1.03e-13 ***
## Age              0.024687   0.003892   6.343 3.10e-10 ***
## SexFemale        0.867092   0.279099   3.107  0.00193 **
## Income           0.118423   0.021962   5.392 8.26e-08 ***
## Health           0.822961   0.051027  16.128  < 2e-16 ***
## EDU             -0.030899   0.025996  -1.189  0.23482
## NMarital_Status -0.220405   0.051395  -4.288 1.93e-05 ***
## Age:SexFemale   -0.012601   0.004954  -2.543  0.01109 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.435 on 1301 degrees of freedom
##   (504 observations deleted due to missingness)
## Multiple R-squared:  0.2656, Adjusted R-squared:  0.2616
## F-statistic: 67.21 on 7 and 1301 DF,  p-value: < 2.2e-16
```

1.3 Creating Regression Tables

To compare the results from two models, let's use the `tab_model()` function to generate a regression table. This table presents a side-by-side comparison of the estimates from both models.

```
tab_model(Model1,Model2,
          dv.labels = c("Manual computation", "R function"))
```

Table 8.1

	Manual computation			**R function**		
Predictors	Estimates	CI	p	Estimates	CI	p
(Intercept)	1.86	0.77 – 2.95	0.001	2.73	2.02 – 3.44	0.001
Age	0.04	0.02 – 0.05	0.001	0.02	0.02 – 0.03	0.001
Sex	0.87	0.32 – 1.41	0.002			
Inter term	-0.01	-0.02 – -0.00	0.011			
Income	0.12	0.08 – 0.16	0.001	0.12	0.08 – 0.16	0.001
Health	0.82	0.72 – 0.92	0.001	0.82	0.72 – 0.92	0.001
EDU	-0.03	-0.08 – 0.02	0.235	-0.03	-0.08 – 0.02	0.235
NMarital Status	-0.22	-0.32 – -0.12	0.001	-0.22	-0.32 – -0.12	0.001
Sex [Female]				0.87	0.32 – 1.41	0.002
Age × Sex [Female]				-0.01	-0.02 – -0.00	0.011
Observations	1309			1309		
R2 / R2 adjusted	0.266 / 0.262			0.266 / 0.262		

Q. Do you observe any differences between the two sets of results?

Write your response here:

1.4 Visualising the Moderation Effect in R

We will explore how to calculate model-based predictions and visualise moderation effects using R. We will base our predictions on the results of Model 2. To begin, we use the `ggpredict()` function, which allows us to compute predicted values of life satisfaction at specific values of the interacted variables.

Recall from Chapter 7 that we used the `prediction()` function. While both functions are useful, `ggpredict()` is often more effective for visualising interaction effects.

```
pred <- ggpredict(
```

```
  Model2,
  terms = c("Age [20:80 by=10]", "Sex")  # Assign the Age range and grouping by Sex
)
print(pred)
```

```
## # Predicted values of Life_satisfaction
##
## Sex: Male
##
## Age | Predicted |     95% CI
## ----------------------------
##  20 |      6.75 | 6.45, 7.06
##  30 |      7.00 | 6.76, 7.24
##  40 |      7.25 | 7.07, 7.43
##  50 |      7.49 | 7.35, 7.64
##  60 |      7.74 | 7.60, 7.88
##  80 |      8.24 | 8.01, 8.46
##
## Sex: Female
##
## Age | Predicted |     95% CI
## ----------------------------
##  20 |      7.37 | 7.13, 7.61
##  30 |      7.49 | 7.31, 7.67
##  40 |      7.61 | 7.47, 7.75
##  50 |      7.73 | 7.62, 7.85
##  60 |      7.85 | 7.72, 7.98
##  80 |      8.09 | 7.87, 8.32
##
## Adjusted for:
## *          Income = 6.00
## *          Health = 3.99
## *             EDU = 4.00
## * NMarital_Status = 1.54
```

The terms argument in `ggpredict()` specifies which variables should vary and how they should vary when generating predicted values. In the example above, `Age [20:80 by=10]` argument indicates that age should

range from 20 to 80 in increments of 10. For each age value, predictions are generated separately for each category of *Sex* (Male and Female).

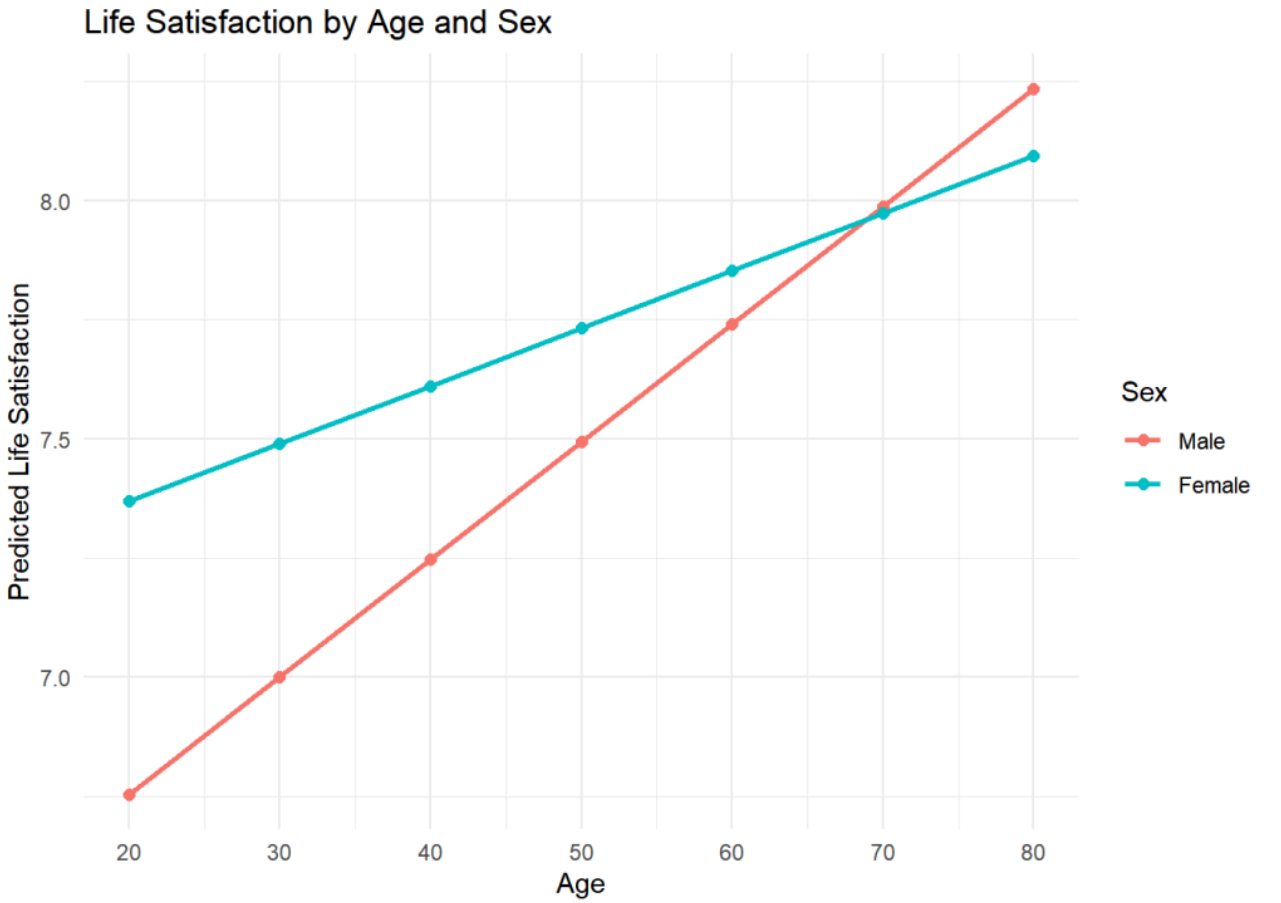
You can also modify the terms argument to suit different scenarios. For instance, to generate predictions across all observed values of *Age* by each level of *Sex*, you can use: `terms = c("Age", "Sex")`. To learn more about the function and available options, you can run: `?ggpredict`.

Quiz. What does the output above represent?

Write your response here:

Now, let's create a plot using the predicted values. To do this, we will use the `ggplot()` function.

```
ggplot(pred, aes(x = x, y = predicted, color = group, group = group)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  labs(
    title = "Life Satisfaction by Age and Sex",
    x = "Age",
    y = "Predicted Life Satisfaction",
    color = "Sex"
  ) +
  scale_x_continuous(breaks = seq(20, 80, by = 10)) +  # Custom age range
  theme_minimal()
```



Quiz. Interpret the plot outcome above.

Write your response here:

1.5 Continuous Moderator

Suppose we are interested in examining whether education level (*EDU*) moderates the relationship between age and income. In this case, note that the potential moderator, *EDU*, is a continuous variable.

```
Model3 <- lm(Income ~ Age*EDU +Sex + Health   + NMarital_Status , data = New_data)
summary(Model3)
```

```
##
## Call:
## lm(formula = Income ~ Age * EDU + Sex + Health + NMarital_Status,
##     data = New_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3630 -1.1631  0.1455  1.2342  5.6347
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.709912   0.632094   7.451 1.67e-13 ***
## Age              -0.042833   0.009071  -4.722 2.59e-06 ***
## EDU               0.108470   0.104703   1.036  0.30041
## SexFemale        -0.218932   0.103583  -2.114  0.03474 *
## Health            0.480500   0.062680   7.666 3.44e-14 ***
## NMarital_Status -0.414464   0.063771  -6.499 1.14e-10 ***
## Age:EDU           0.005034   0.001796   2.802  0.00515 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.809 on 1309 degrees of freedom
##   (497 observations deleted due to missingness)
## Multiple R-squared:  0.2301, Adjusted R-squared:  0.2266
## F-statistic:  65.2 on 6 and 1309 DF,  p-value: < 2.2e-16
```

The results of the regression analysis are shown above. The coefficient for the interaction term (Age:EDU) is 0.007, with a p-value < .001. This indicates that education level significantly moderates the relationship between age and income.

When you check the main effect of Age and EDU, each of them is negatively and positively associated with Income respectively and interaction term is positive. This result suggests that the negative effect of age on income weakens as education level increases. In other words, higher education may buffer against age-related declines in income.

Since EDU is a continuous variable, there are no discrete groups for direct comparison. To interpret the interaction effect more intuitively, we examine the relationship between age and income at three representative values of EDU: the **mean**, **one standard deviation below the mean**, and **one standard deviation above the mean**. In this example, the three values are 3.02, 5.13, and 7.24.

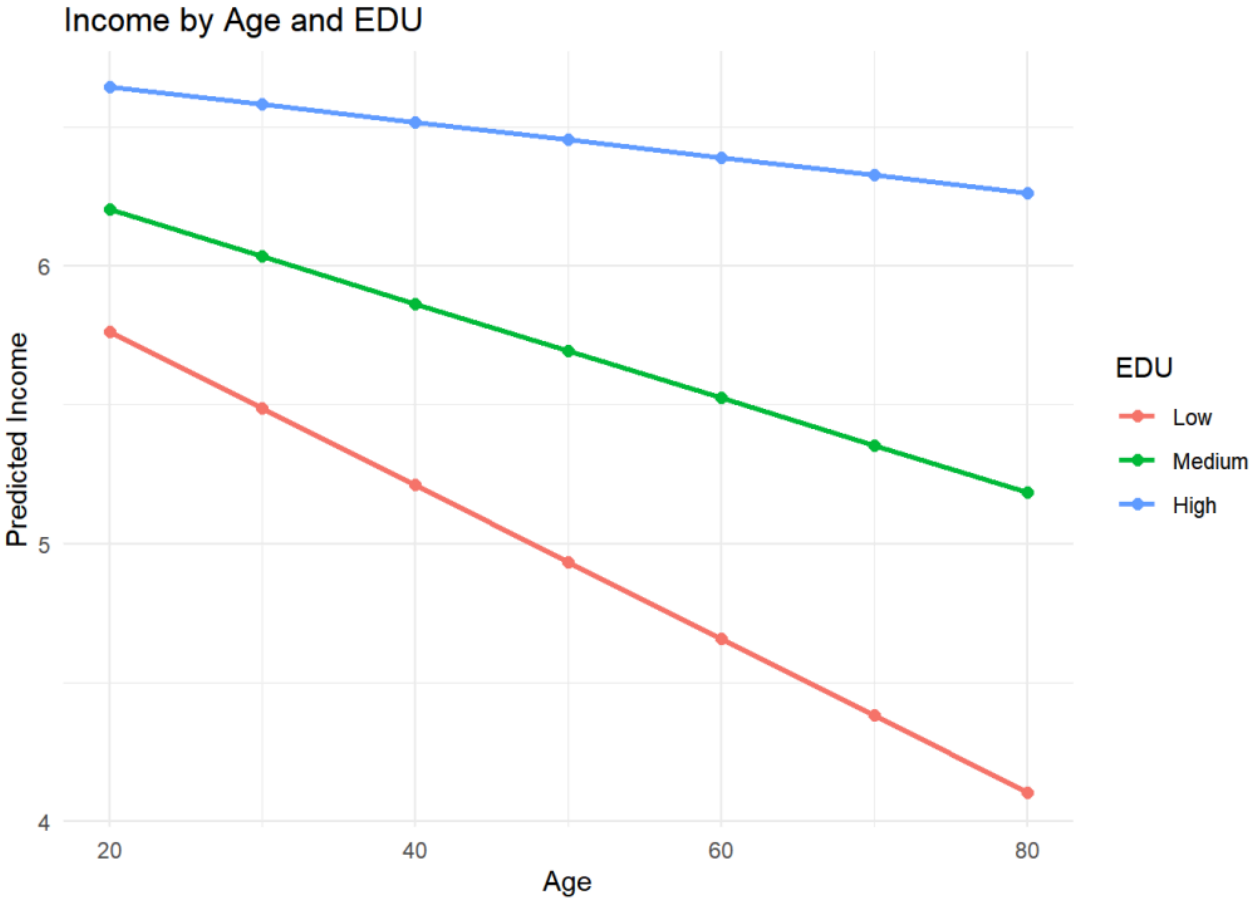
You can generate predicted values for these levels using the `ggpredict()` function:

```
describe(New_data$Income) #Check the mean (4.83) and standard deviation (1.67) values
```

Table 8.2

	vars	n	mean	sd	median	trimmed	mad	min	max	sd	median	trimmed	mad
X1	1	1748	5.127574	2.109014	5	5.208571	2.9652	1	10	2.109014	5	5.208571	2.9652

```
pred2 <- ggpredict(Model3, terms = c("Age[20:80 by=10]", "EDU [3.02, 5.13, 7.24]"))


ggplot(pred2, aes(x = x, y = predicted, color = group, group = group)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  scale_color_discrete(labels = c("Low", "Medium", "High")) +  # Low:3.02, Medium: 5.13,
High: 7.24
  labs(
    title = "Income by Age and EDU",
    x = "Age",
    y = "Predicted Income",
    color = "EDU"  # This sets the legend title
  ) +
  theme_minimal()
```



Quiz. Interpret the plot outcome above.

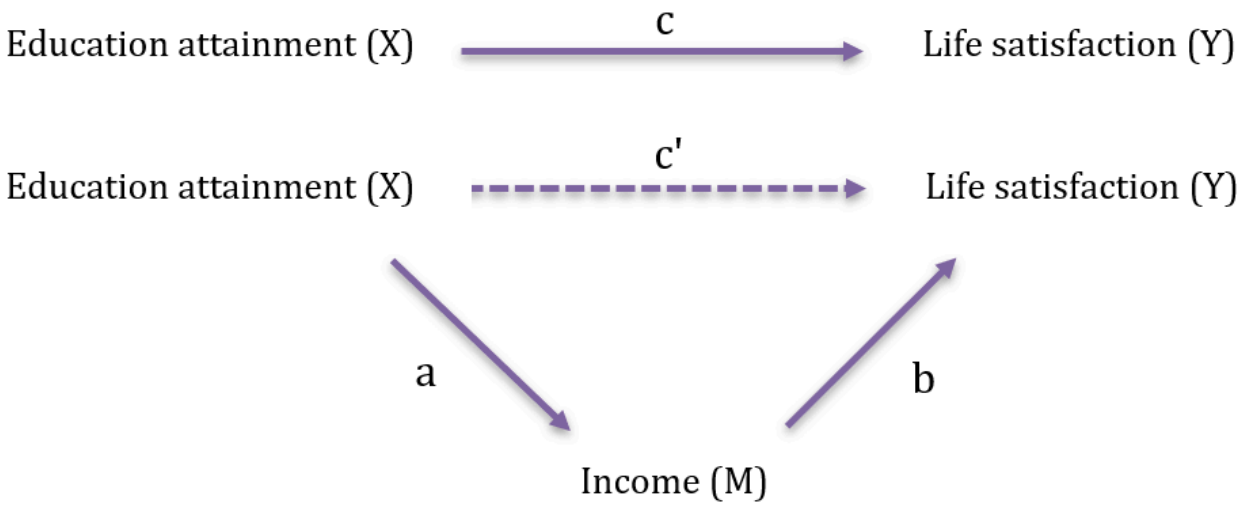
Write your response here:

2. Statistical Mediation Analysis

2.1 What is Mediation?

Mediation is when the effect of an independent variable (X) on a dependent variable (Y) is transmitted through a third variable – called a mediator (M). For example, suppose higher educational attainment is associated with greater life satisfaction (**path c**). However, this relationship may be explained by an underlying mechanism: individuals with higher education tend to earn more income (**path a**), and higher

income, in turn, is associated with greater life satisfaction (**path b**). In this scenario, income plays as a mediator, contributing to explain how or why education influences life satisfaction.



2.2 Conducting a Mediation Analysis

To systematically assess mediation, we follow the four-step approach proposed by Baron and Kenny (1986):

1. Test the total effect (path c): Estimate the association between the independent variable (X) and the outcome (Y) while including control variables but excluding the mediator. If X is not significantly related to Y, there is no effect to mediate.
2. Test path a (X → M): Regress the mediator (M) on the independent variable (X) to determine whether X significantly predicts M. This step treats the mediator as the outcome.
3. Test path b and the direct effect (path c′): Regress the outcome (Y) on both the independent variable (X) and the mediator (M). This model tests whether the mediator has a significant effect on the outcome while controlling for X, and whether the direct effect of X on Y (path c′) is reduced compared to the total effect (c).
4. Assess full or partial mediation: If the effect of X on Y becomes non-significant when controlling for M, this suggests full mediation. If the effect is reduced but still significant, it indicates partial mediation.

Exercise 2 – Mediation Effect in R

We now turn to a practical example of mediation analysis using the WVS dataset. In this exercise, our key independent variable is education level (*EDU*), the mediator (M) is *Income*, and the dependent variable (Y) is *Life_satisfaction*. We also control for two basic demographic variables: Age and Sex.

Note however that for mediation analysis it is convenient that we focus only on observations in which individuals have **no missing data** in any of the variables involved in the analysis. Else, each of the models in

Steps 1 to 4 of the Baron & Kenny mediation framework could be based on different individuals and have a different number of observations. That would invalidate our mediation analysis.

We will follow the Baron and Kenny's (1986) approach step-by-step in R:

```
#Remove all NAs
model_data <- na.omit(New_data[, c("Life_satisfaction","Income", "Sex",  "Age", "EDU")])


#Step 1 (X -> Y)
Model1 <- lm(Life_satisfaction ~  EDU + Sex + Age     , data = model_data)


#Step 2 (X -> M)
Model2 <- lm(Income ~ EDU + Sex +  Age, data = model_data)


#Step 3 & 4 (X +M -> Y)
Model3 <- lm(Life_satisfaction ~  EDU + Income   + Sex+  Age   , data = model_data)


tab_model(Model1,Model2,Model3,
          dv.labels = c("Step 1: X -> Y", "Step 2: X -> M",  "Step 3 & 4: X + M -> Y"),
          title = "Mediation analysis")
```

Table 8.3 Mediation analysis

	Step 1: X -> Y			**Step 2: X -> M**			**Step 3 & 4: X + M -> Y**		
Predictors	Estimates	CI	p	Estimates	CI	p	Estimates	CI	p
(Intercept)	5.83	5.40 – 6.27	0.001	3.84	3.36 – 4.33	0.001	4.88	4.43 – 5.33	0.001
EDU	0.11	0.06 – 0.16	0.001	0.47	0.41 – 0.52	0.001	0	-0.06 – 0.05	0.862
Sex [Female]	0.27	0.10 – 0.44	0.002	-0.28	-0.47 – -0.09	0.004	0.34	0.17 – 0.50	0.001
Age	0.02	0.01 – 0.02	0.001	-0.01	-0.02 – -0.01	0.001	0.02	0.02 – 0.03	0.001
Income							0.25	0.21 – 0.29	0.001
Observations	1672			1672			1672		
R2 / R2 adjusted	0.037 / 0.036			0.184 / 0.182			0.111 / 0.109		

The total effect of EDU on life satisfaction can be decomposed into the direct effect and the indirect effect, calculated as follows:

$$Total\ effect = (b_2 * b_4) + b_3 = (0.47 \times 0.25) + (-0.00) = 0.12$$

You can compare this value with the coefficient for EDU (b_1) in Model 1 to evaluate consistency across models.

Based on the output above and address the following questions for each step:

Q1. Step 1 – Is the independent variable (EDU) significantly associated with the outcome (Life satisfaction)? What evidence supports this?

Q2. Step 2 – Is the independent variable significantly associated with the mediator (Income)? What is the evidence?

Q3. Step 3 – Is the mediator (Income) significantly associated with the outcome (Life satisfaction), controlling for EDU? What evidence supports this?

Q4. Step 4 – Is the coefficient for EDU reduced or no longer significant when Income is added to the model? What does this suggest about mediation?

Write your responses here:

2.3 Sobel Test

Although significant relationships between the independent variable and the mediator, and between the mediator and the outcome, may suggest mediation, they do not provide formal statistical confirmation. To assess whether mediation is occurring, a **Sobel** test can be used as a post-estimation test.

The Sobel test evaluates whether the indirect effect of an independent variable on a dependent variable—via a mediator—is statistically different from zero. Specifically, it tests the significance of the product of the coefficients for **path a (X → M)** and **path b (M → Y)**, that is:

$$Indirect\ effect = b_2 * b_4$$

This can also be expressed as a proportion of the total effect:

$$\frac{Indirect\ effect}{Total\ effect} = \frac{b_2 \cdot b_4}{b_1}$$

To conduct the Sobel test, we use the `mediation.test()` function from the *bda* package:

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

```
# Testing mediation effect


mediation.test(model_data$Income, model_data$EDU, model_data$Life_satisfaction) #media-
tion.test(mv, iv, dv): This is the basic structure to order the variables: mediator, inde-
pendent, and dependent variable
```

The output includes several test statistics, but we are particularly interested in the Sobel test result, which directly indicates whether the indirect effect is statistically significant.

Based on the output above and address the following questions:

Q1. What does the Sobel test output tell about the statistical significance of the mediating role of income in the relationship between education and life satisfaction?

Q2. Based on the results from all models and the Sobel test, what can be concluded about the presence and strength of mediation in this analysis?

Write your responses here:

Once you complete all exercises, make sure to save your R script before closing R.

9.

INTRODUCTION TO DATA VISUALISATION

In this chapter, we will practise data visualisation techniques. Data visualisation is widely used in the social sciences to present results in a more accessible way. By using graphics and visual displays of the data we can help readers interpret quantitative findings more intuitively which increases the impact of our research. We will begin by exploring the basic yet essential grammar of the *ggplot2* package and then apply it to create a variety of different plot types.

1. Learning ggplot Package

Data visualisation is one of R's greatest strengths. From basic graphs to advanced packages, R offers a wide range of tools for creating insightful and appealing visualisations. In this session, we focus on the *ggplot2* package, which enables users to build complex and professionally presented plots using intuitive and modular syntax.

What makes *ggplot2* particularly accessible for beginners is its foundation in the Grammar of Graphics (the *gg* in *ggplot2*). This approach is similar to how learning basic grammar enables the construction of countless sentences. Likewise, learning a few core components of *ggplot2*, such as data, aesthetics, and geometry, allows you to create a wide variety of plots.

We begin by exploring four key components using scatter plots, then apply them to other types of visualisations. For this session, we use the built-in **USArrests** dataset. We'll first examine and prepare the data before moving into plotting.

We begin by loading the dataset and the required packages:

```
#Set the working directory
setwd("C:/your own path/SOCYR")
```

```
# View dataset documentation
?USArrests

#Load the dataset
USA <- USArrests

#Load the required packages for chapter 9
library(dplyr)
library(ggplot2)
library(tidyr)
options(scipen = 999) #turning off scientific notation
```

We then classify the states into four U.S. Census regions—Northeast, Midwest, South, and West—and create new variables for both **region** and **state**.

```
# Assign region and state variables
northeast <- c("Connecticut", "Maine", "Massachusetts", "New Hampshire", "Rhode Island",
"Vermont",
               "New Jersey", "New York", "Pennsylvania")
midwest <- c("Illinois", "Indiana", "Michigan", "Ohio", "Wisconsin",
             "Iowa", "Kansas", "Minnesota", "Missouri", "Nebraska", "North Dakota",
"South Dakota")
south <- c("Delaware", "Florida", "Georgia", "Maryland", "North Carolina", "South Car-
olina",
           "Virginia", "West Virginia", "Alabama", "Kentucky", "Mississippi", "Tennessee",
           "Arkansas", "Louisiana", "Oklahoma", "Texas")
west <- c("Arizona", "Colorado", "Idaho", "Montana", "Nevada", "New Mexico", "Utah",
"Wyoming",
          "Alaska", "California", "Hawaii", "Oregon", "Washington")
USA$Region <- ifelse(rownames(USA) %in% northeast, "Northeast",
                             ifelse(rownames(USA) %in% midwest, "Midwest",
                                    ifelse(rownames(USA) %in% south, "South",
                                           ifelse(rownames(USA) %in% west, "West", NA))))

USA$State <- rownames(USA)
rownames(USA) <- NULL
#Check the updated dataset
USA
```

Table 9.1

Murder	Assault	UrbanPop	Rape	Region	State
13.2	236	58	21.2	South	Alabama
10	263	48	44.5	West	Alaska
8.1	294	80	31	West	Arizona
8.8	190	50	19.5	South	Arkansas
9	276	91	40.6	West	California
7.9	204	78	38.7	West	Colorado
3.3	110	77	11.1	Northeast	Connecticut
5.9	238	72	15.8	South	Delaware
15.4	335	80	31.9	South	Florida
17.4	211	60	25.8	South	Georgia
5.3	46	83	20.2	West	Hawaii
2.6	120	54	14.2	West	Idaho
10.4	249	83	24	Midwest	Illinois
7.2	113	65	21	Midwest	Indiana
2.2	56	57	11.3	Midwest	Iowa
6	115	66	18	Midwest	Kansas
9.7	109	52	16.3	South	Kentucky
15.4	249	66	22.2	South	Louisiana
2.1	83	51	7.8	Northeast	Maine
11.3	300	67	27.8	South	Maryland
4.4	149	85	16.3	Northeast	Massachusetts
12.1	255	74	35.1	Midwest	Michigan
2.7	72	66	14.9	Midwest	Minnesota
16.1	259	44	17.1	South	Mississippi
9	178	70	28.2	Midwest	Missouri
6	109	53	16.4	West	Montana
4.3	102	62	16.5	Midwest	Nebraska
12.2	252	81	46	West	Nevada
2.1	57	56	9.5	Northeast	New Hampshire
7.4	159	89	18.8	Northeast	New Jersey
11.4	285	70	32.1	West	New Mexico
11.1	254	86	26.1	Northeast	New York

Murder	Assault	UrbanPop	Rape	Region	State
13	337	45	16.1	South	North Carolina
0.8	45	44	7.3	Midwest	North Dakota
7.3	120	75	21.4	Midwest	Ohio
6.6	151	68	20	South	Oklahoma
4.9	159	67	29.3	West	Oregon
6.3	106	72	14.9	Northeast	Pennsylvania
3.4	174	87	8.3	Northeast	Rhode Island
14.4	279	48	22.5	South	South Carolina
3.8	86	45	12.8	Midwest	South Dakota
13.2	188	59	26.9	South	Tennessee
12.7	201	80	25.5	South	Texas
3.2	120	80	22.9	West	Utah
2.2	48	32	11.2	Northeast	Vermont
8.5	156	63	20.7	South	Virginia
4	145	73	26.2	West	Washington
5.7	81	39	9.3	South	West Virginia
2.6	53	66	10.8	Midwest	Wisconsin
6.8	161	60	15.6	West	Wyoming

1.1 The Setup

To begin, you need to specify the dataset that *ggplot2* will use. This is done using the `ggplot()` function. The *ggplot2* package is designed to work with data in tidy format, where each row represents an observation and each column represents a variable. This initial step does not produce a plot on its own—it simply establishes the foundation for further layers to be added.

```
#Assign the dataset
ggplot(USA)
```

```
#Alternatively, you can use the pipe operator (%>%)
#USA %>% ggplot()
```

1.2 Geometries

Plots in *ggplot2* are constructed by adding layers. Layers typically define geometric objects, or geoms, which specify the type of visual representation—such as points, bars, or lines.

To add a layer, use the + symbol. A typical line of code in *ggplot2* looks like this:

$$ggplot() + layer1 + layer2 + \ldots + layerN$$

The first layer usually defines the geometry using a `geom_()` function:

```
ggplot(USA) + geom_point()  # Create a scatter plot
ggplot(USA) + geom_bar()    # Create a bar chart from counts
ggplot(USA) + geom_line()   # Draw a line graph for trends
```

If you run the above examples without specifying variables, you will receive an **error message** stating that required aesthetics (such as x and y) are missing. Once you have chosen a dataset and geometry, the next essential step is to define aesthetic mappings, which tell *ggplot2* how variables should be mapped to visual properties like position, colour, or size.

1.3 Aesthetic mapping

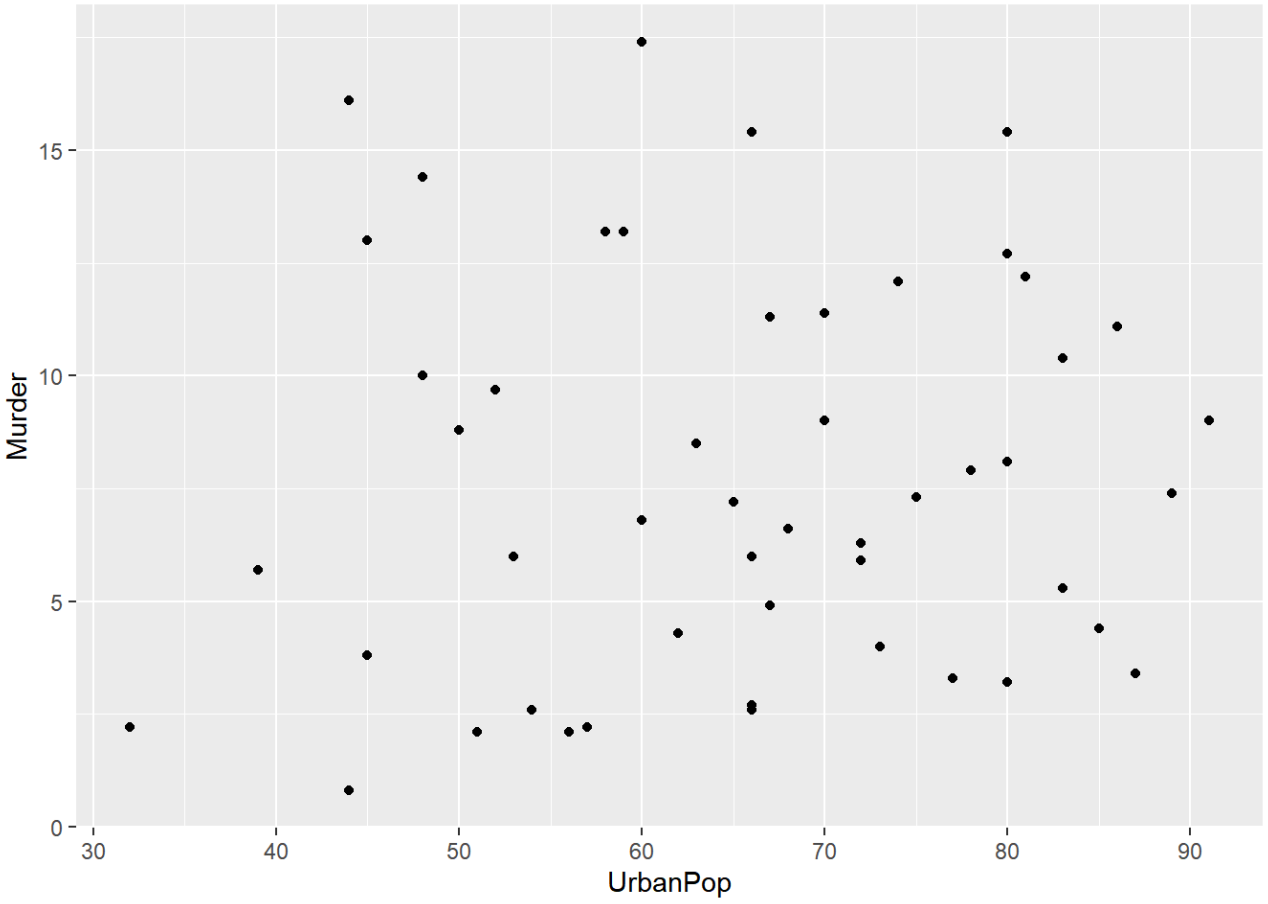
Aesthetic mapping defines how variables in the dataset are visually represented in the plot, such as through position (x and y axes), colour, or size. The `aes()` function is used to map data variables to these visual features.

You can place `aes()` inside the main `ggplot()` function or within individual `geom_()` layers. Both approaches are valid, and the choice often depends on whether the aesthetic applies to the entire plot or just a specific layer.

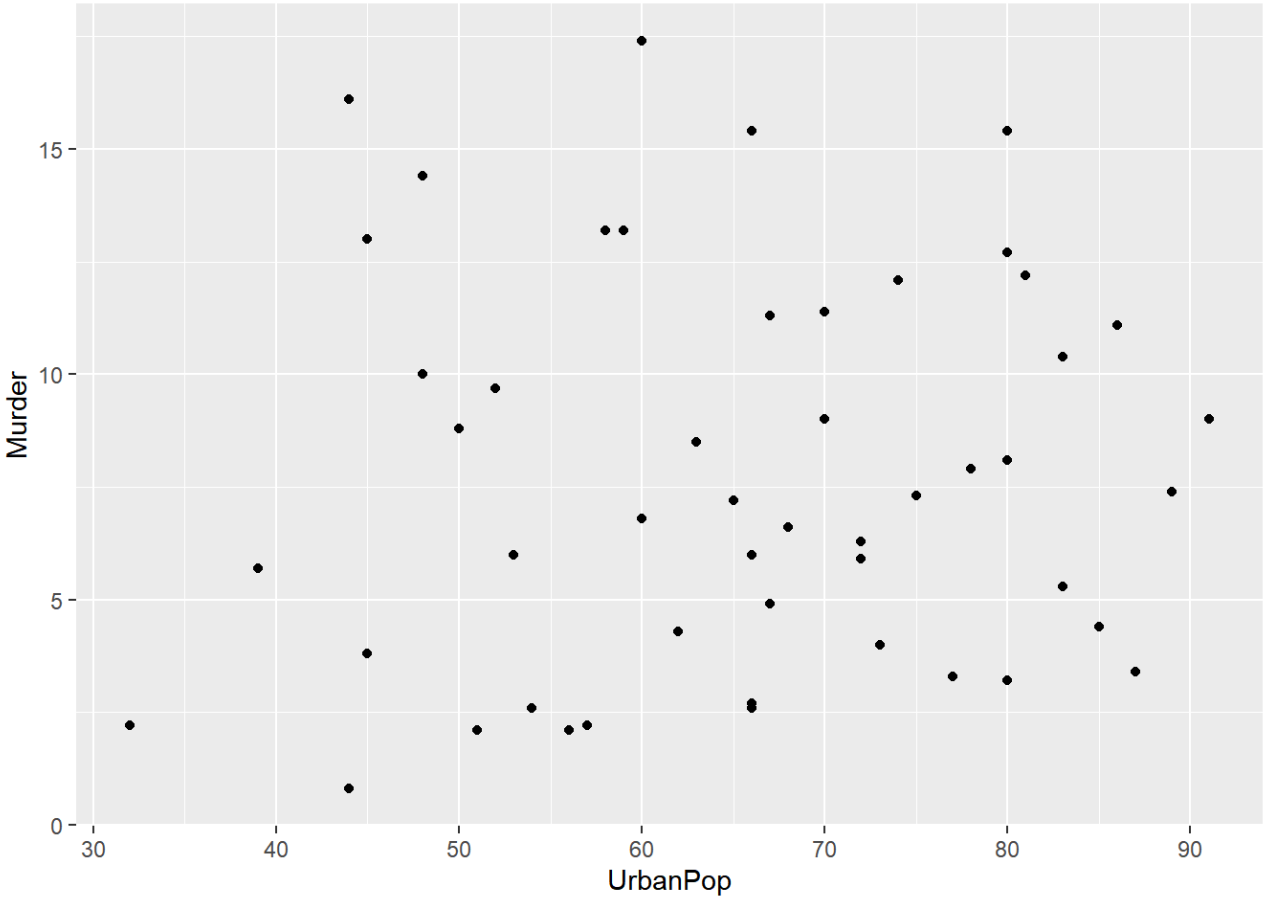
For example, you can create a scatter plot by mapping *urban population* to the x-axis and *murder rate* to the y-axis, and assign colour based on a third variable (*region*):

```
#Basic syntax
ggplot(USA, aes()) +geom_point()
ggplot(USA) +geom_point(aes())

# Aesthetic defined in ggplot()
```

```
ggplot(USA, aes(x = UrbanPop, y = Murder)) + geom_point()
```




```
# Aesthetic defined in geom_point()
#ggplot(USA) + geom_point(aes(x = UrbanPop, y = Murder))

# To add a third variable (Region), as a colour aesthetic
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) + geom_point()
```




```
#ggplot(USA) + geom_point(aes(x = UrbanPop, y = Murder, color = Region))
```

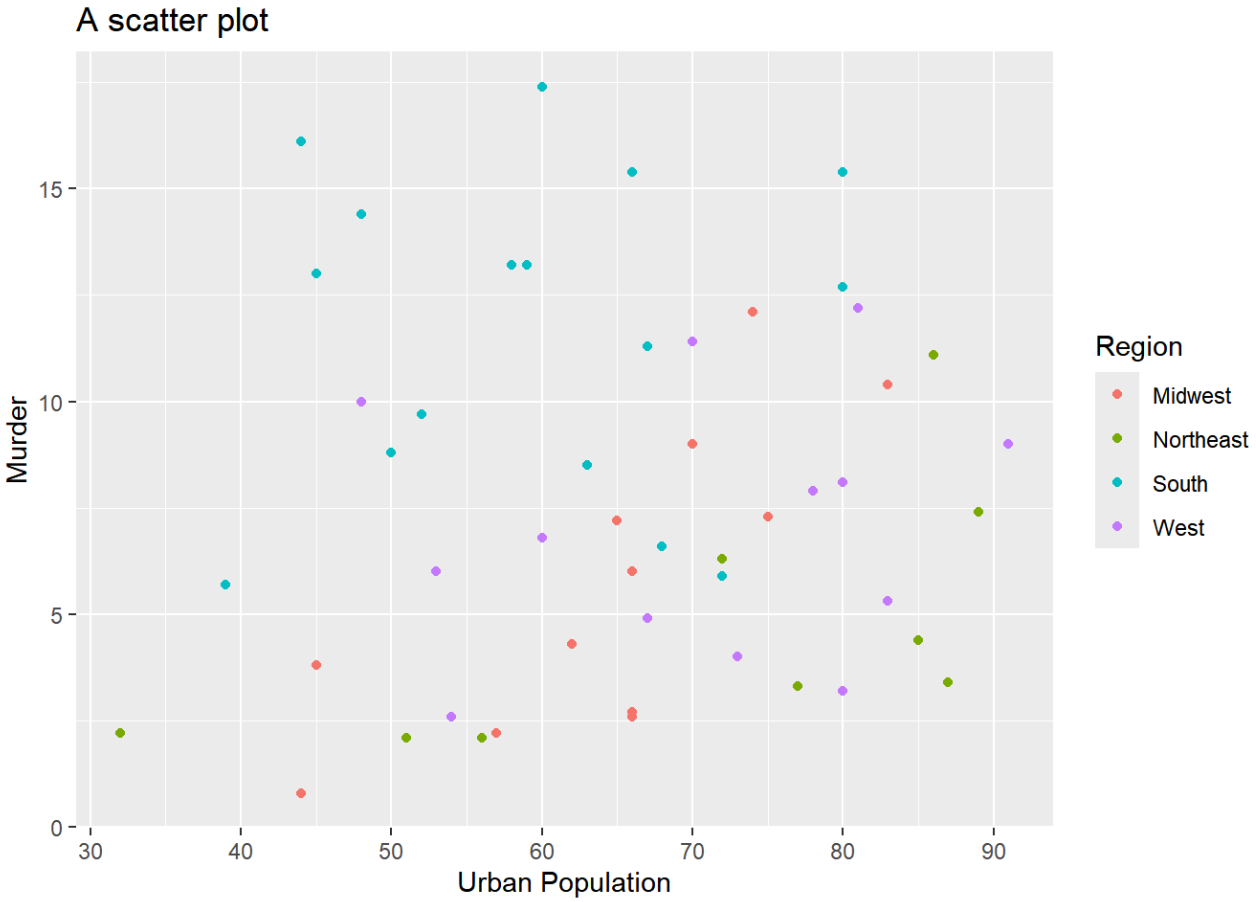
1.4 Additional Features

Beyond the core plot structure, *ggplot2* allows for easy enhancements such as labels, themes, and size adjustments. These features improve readability and customise your visual output.

1.4.1 Adding Labels

By default, axis labels and scales are derived from the data. However, you can customise them using the `labs()` function to add a plot title and modify axis labels.

```
# Adding x- and y-xis labels and plot title.
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) +
                        geom_point() +
                           labs(title = "A scatter plot", x = "Urban Population", y = "Mur-
der")
```



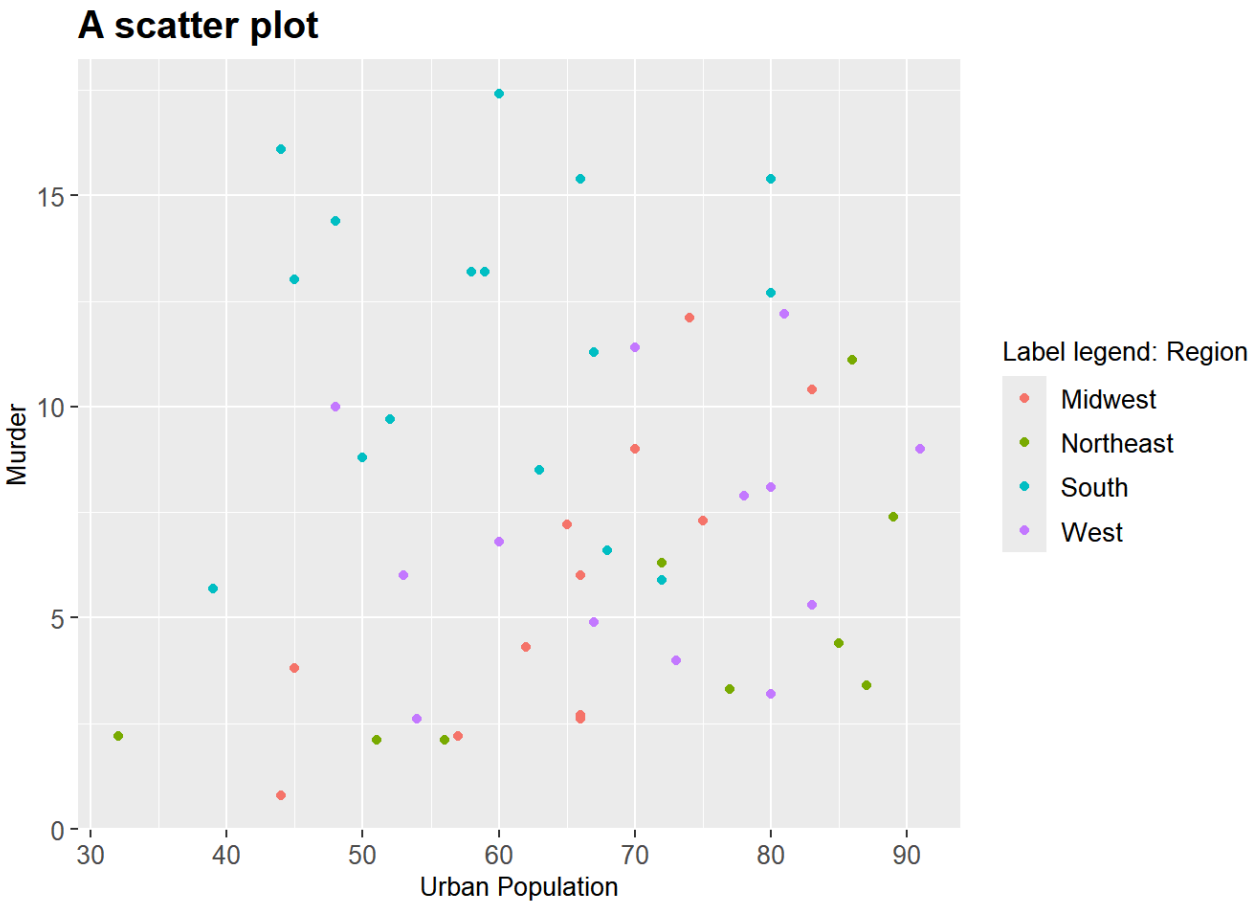
1.4.2 Themes

Once your basic plot is constructed, you may want to refine its appearance. For example, you can adjust the size of the title and axis labels, or change the legend title. These enhancements are made using the `theme()` function, which allows you to customise individual plot elements by specifying them inside `element_text()`. If you wish to hide any element, use `element_blank()` to remove it entirely.

To modify the legend title associated with a colour aesthetic that maps a factor variable, use the `scale_color_discrete()` function. This is because the default legend in such cases is based on discrete colour mapping.

```
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) +
                geom_point() +
                labs(title = "A scatter plot", x = "Urban Population", y = "Murder") +
                theme(
                 plot.title = element_text(size = 15, face = "bold"),    # Sets the plot
title size and makes it bold
                 axis.text.x = element_text(size = 10),                 # Sets the font
```

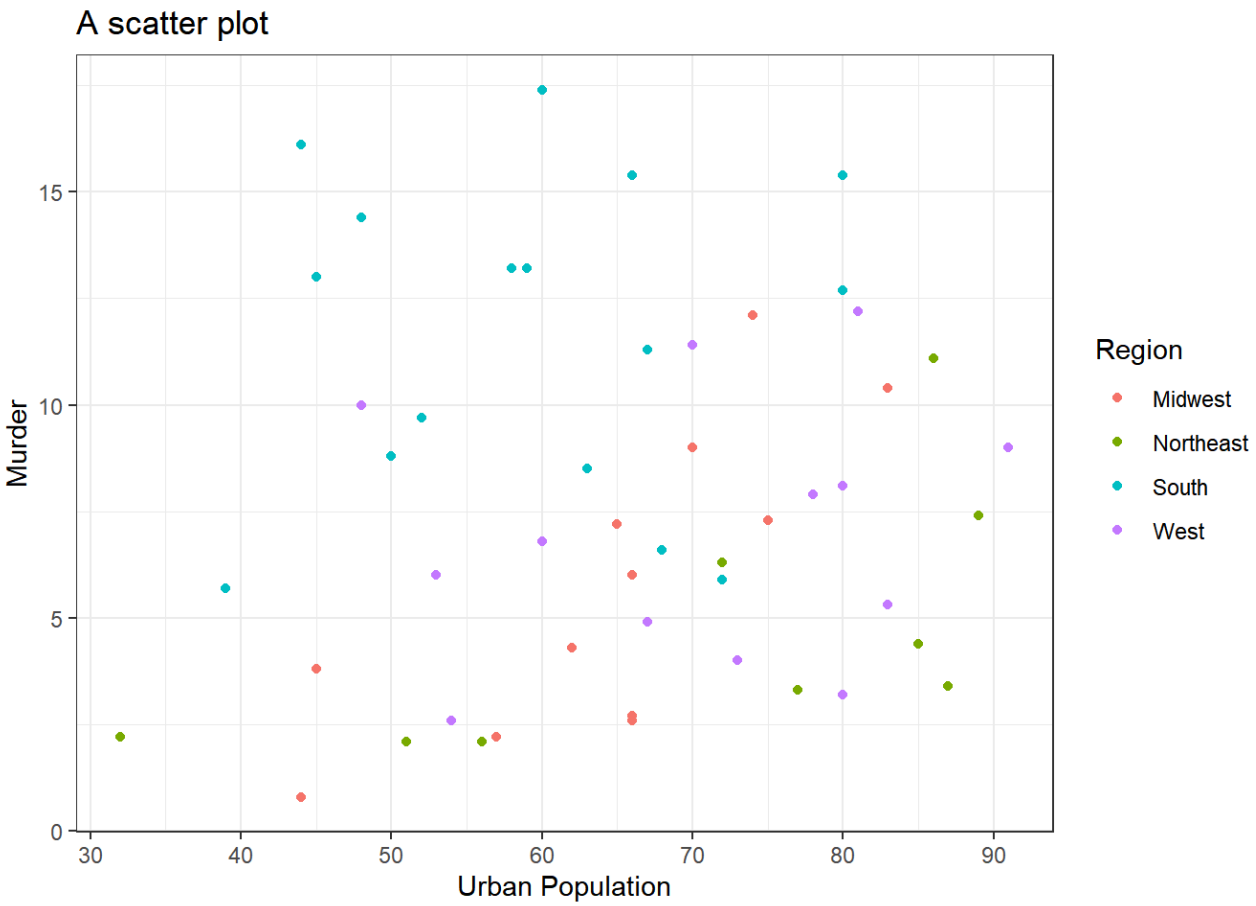
```
size of x-axis tick labels
                  axis.text.y = element_text(size = 10),                # Sets the font
size of y-axis tick labels
                  axis.title.x = element_text(size = 10),               # Sets the font
size of the x-axis title
                  axis.title.y = element_text(size = 10),               # Sets the font
size of the y-axis title
                  legend.title = element_text(size = 10),               # Sets the leg-
end title font size
                  legend.text = element_text(size = 10))  +             # Sets the leg-
end item labels font size
                  scale_color_discrete(name = "Label legend: Region")   # Sets the leg-
end title
```



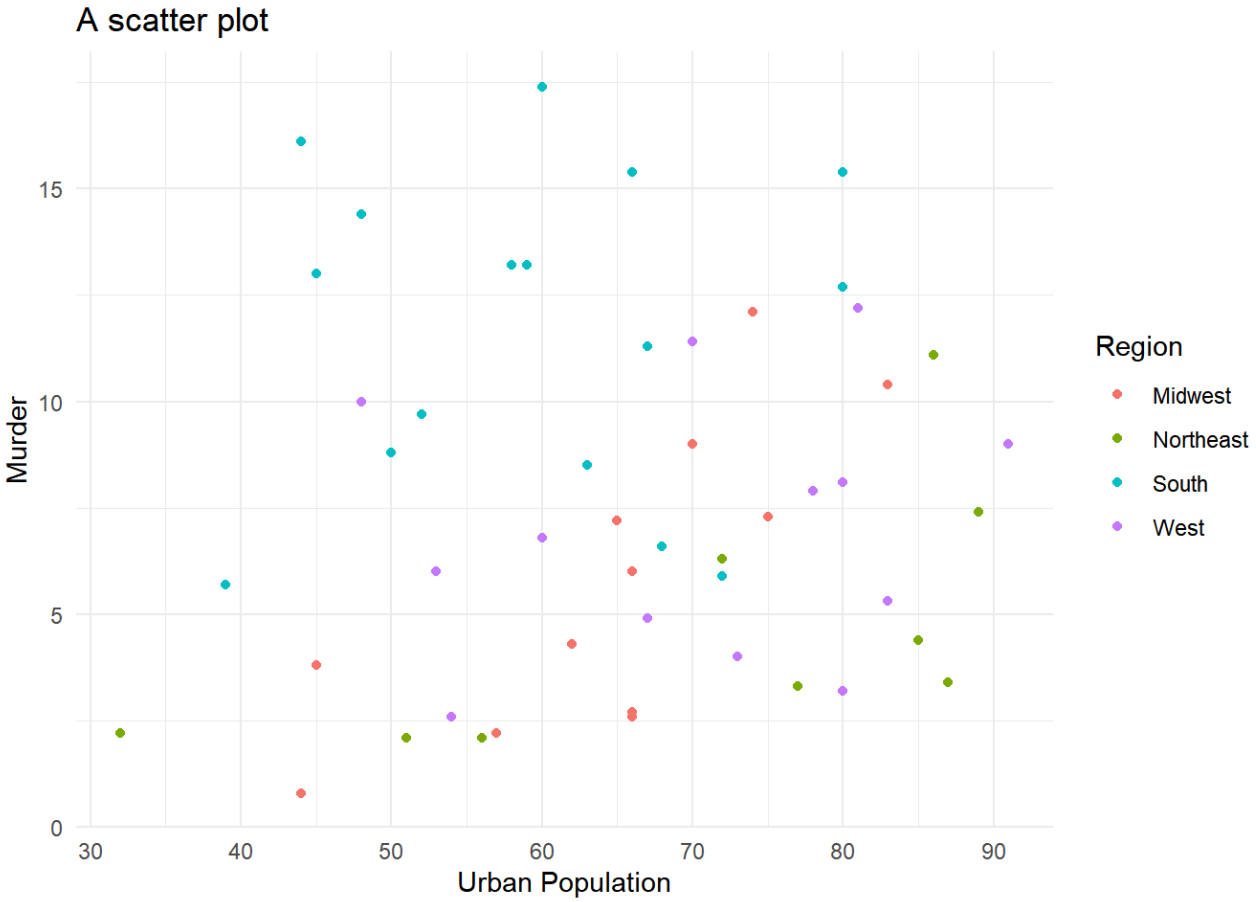
You can also modify the overall style of the plot background using pre-built themes:

```
#White background with grid liens
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) +
```

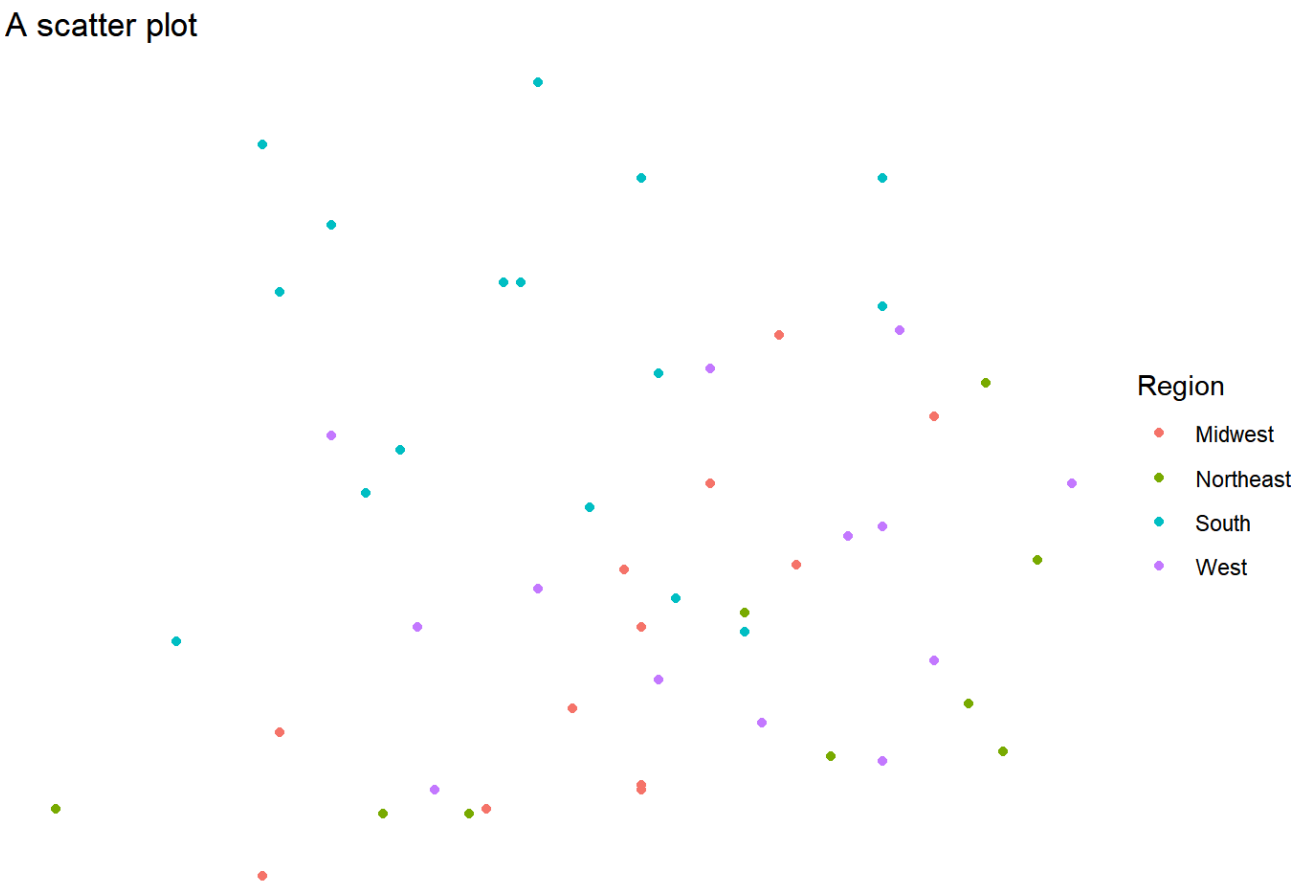
```
  geom_point() +
  labs(title = "A scatter plot", x = "Urban Population", y = "Murder")  +
  theme_bw()
```




```
# Minimalist background with no annotations
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) +
  geom_point() +
  labs(title = "A scatter plot", x = "Urban Population", y = "Murder")  +
  theme_minimal()
```



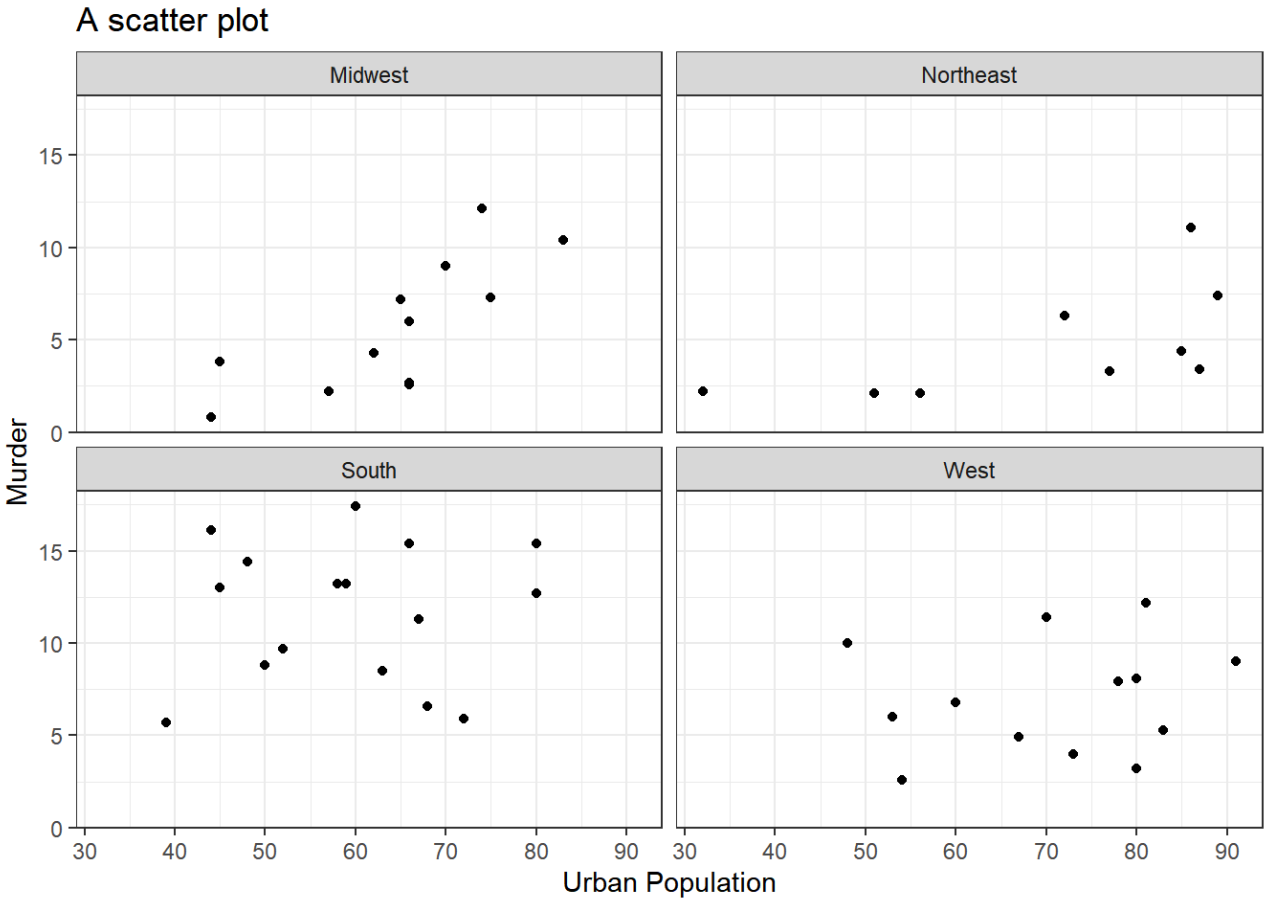

```
#Empty background
ggplot(USA, aes(x = UrbanPop, y = Murder, color = Region)) +
  geom_point() +
  labs(title = "A scatter plot", x = "Urban Population", y = "Murder")  +
  theme_void()
```



1.4.3. Facets

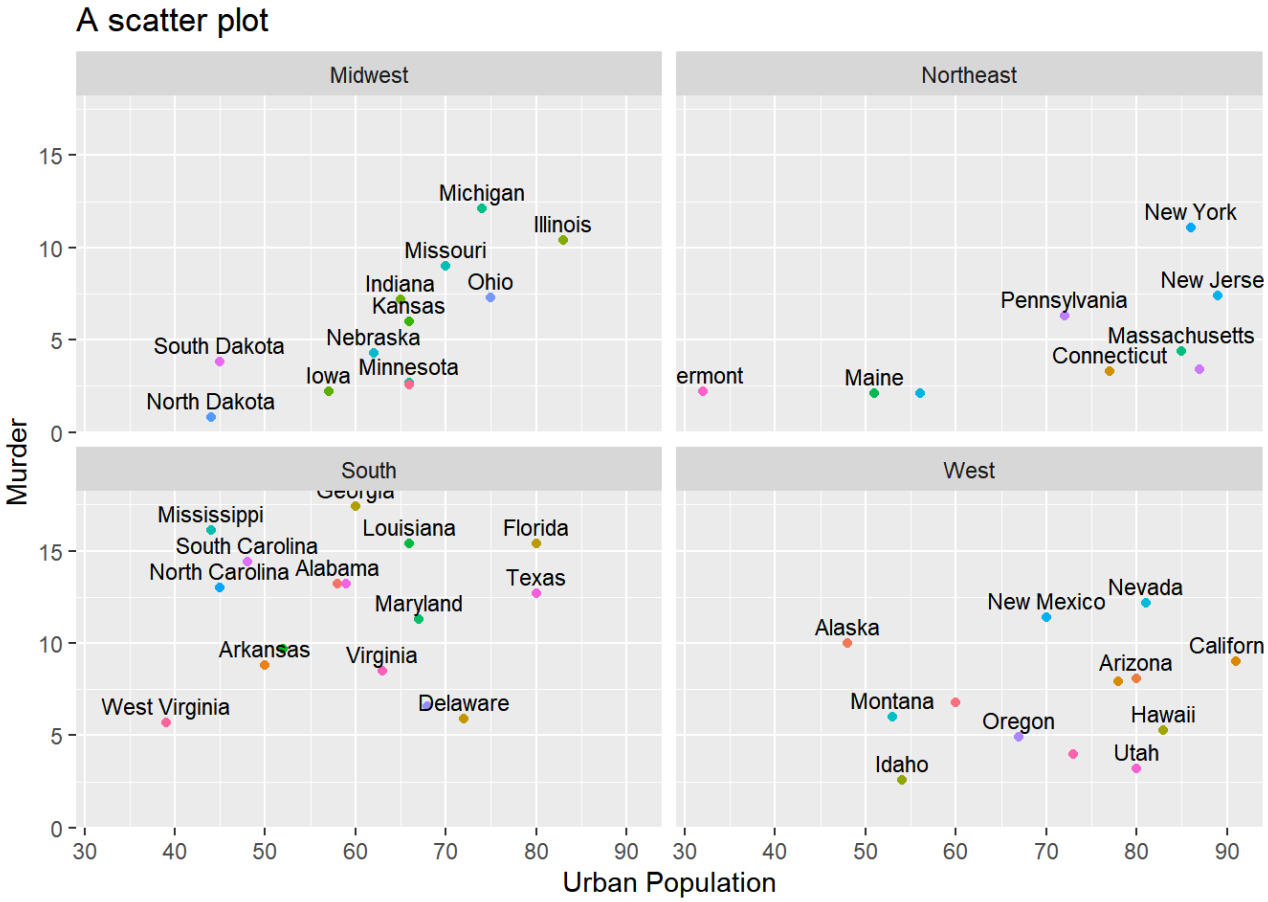
So far, we have visualised either single variables or relationships between two variables in a single plot. However, in many cases, it is useful to examine how these patterns differ across categories. The `facet_wrap()` function allows you to split a plot into multiple panels, each representing a subset of the data based on a categorical variable.

```
# Facet by Region (2 columns)
ggplot(USA) +geom_point(aes(x=UrbanPop, y = Murder)) +
  labs(title = "A scatter plot", x = "Urban Population", y = "Murder")  +
  theme_bw()+
  facet_wrap(~ Region, ncol = 2)
```



You can also colour the points by state and add labels using `geom_text()`:

```
ggplot(USA) + geom_point(aes(x=UrbanPop, y = Murder, color = State)) +
  labs(title = "A scatter plot", x = "Urban Population", y = "Murder")  +
    geom_text(aes(x = UrbanPop, y = Murder, label = State),
    size = 3, vjust = -0.5, check_overlap = TRUE) +
    theme(legend.position = "none") +
    facet_wrap(~ Region, ncol = 2)
```



1.5 Summary: Building a Plot Template

Once you understand the building blocks of *ggplot2*, you can construct a wide range of plots by combining them. Below is a general template for assembling a plot:

```
ggplot(data) +
  <Geom_Function>(<Aesthetic_Mappings>) +
  labs(...) +
  facet_wrap(...) +
  theme(...)
```

For further customisation and advanced plotting techniques, refer to the official gglot2 cheatsheet (PDF, 2.1MB).

2. Practicing Visualisations in R

2.1 Histogram Plot

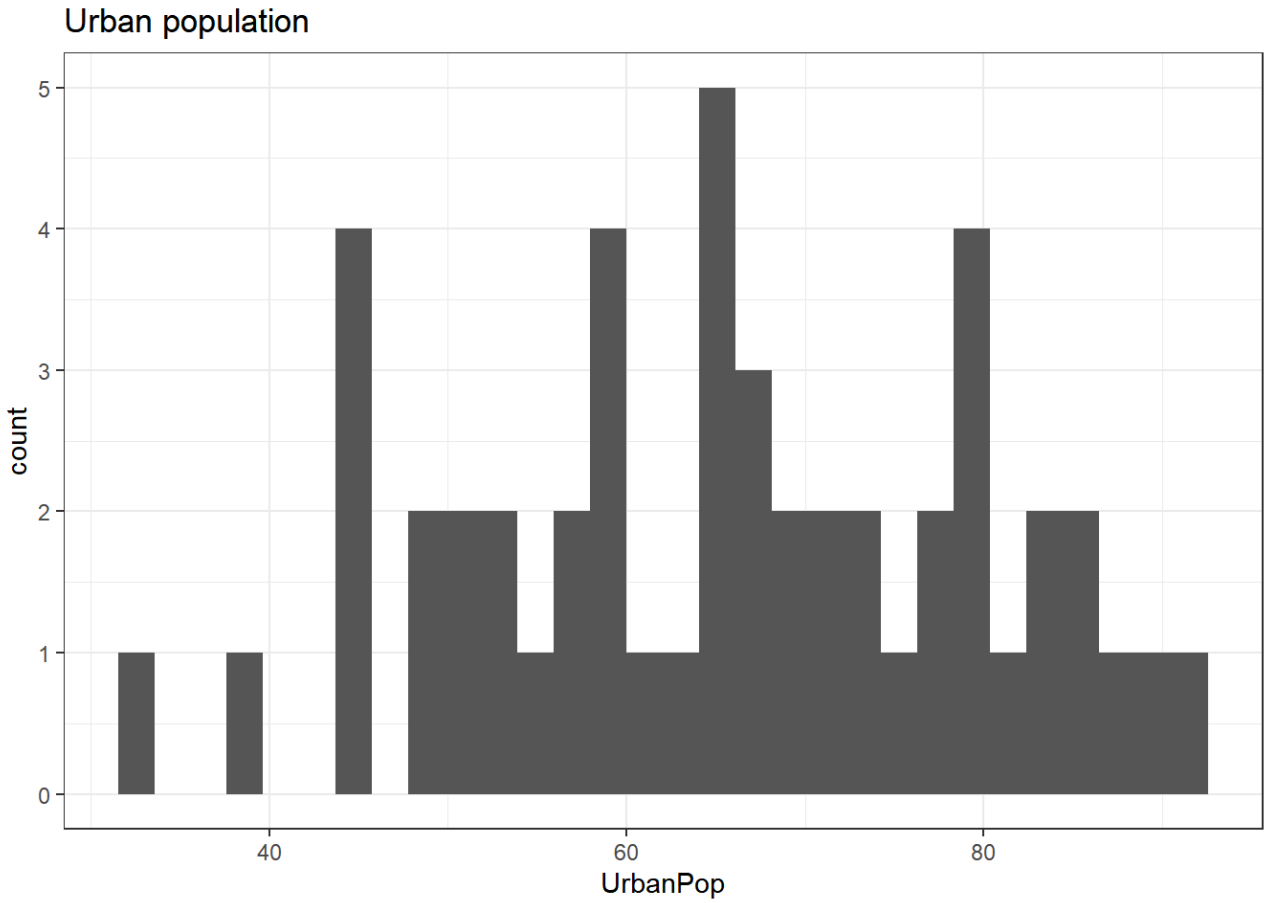
Histogram plots are a useful way to visualise the distribution of a continuous variable. This is especially

important in regression analysis, where one of the assumptions is that the outcome variable follows a normal distribution.

Let's begin by creating a histogram of the urban population variable:

```
#Basic histogram
ggplot(data = USA, aes(x=UrbanPop)) +
  geom_histogram() +
  labs(title = "Urban population")+
   theme(plot.title = element_text(hjust = 0.5))+ theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

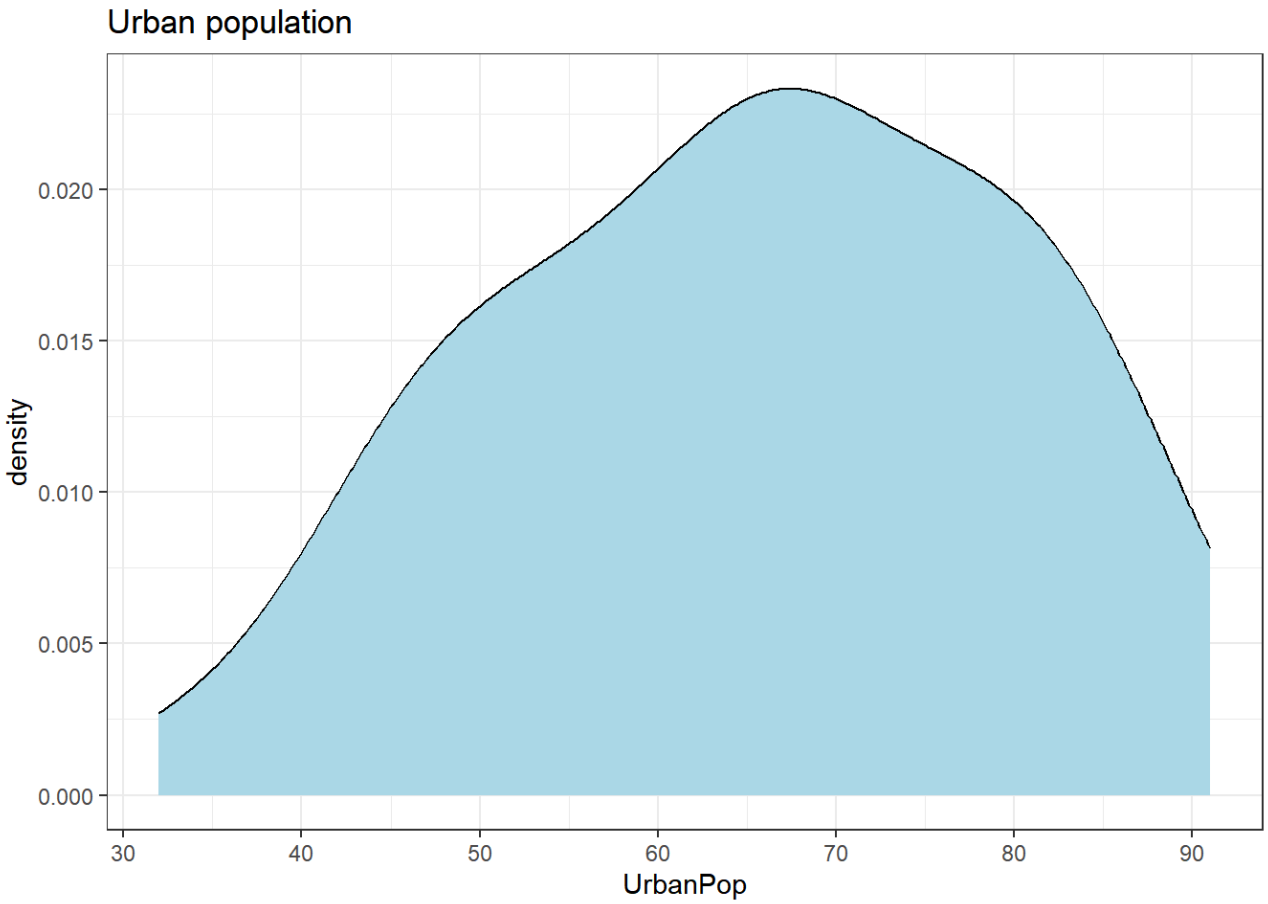


Q. Based on the plot above, do you think the variable follows a normal distribution?

Write your response here:

While histograms are helpful, they may not always clearly reveal whether a distribution is normal. In such cases, a **density** plot offers a smoother visualisation.

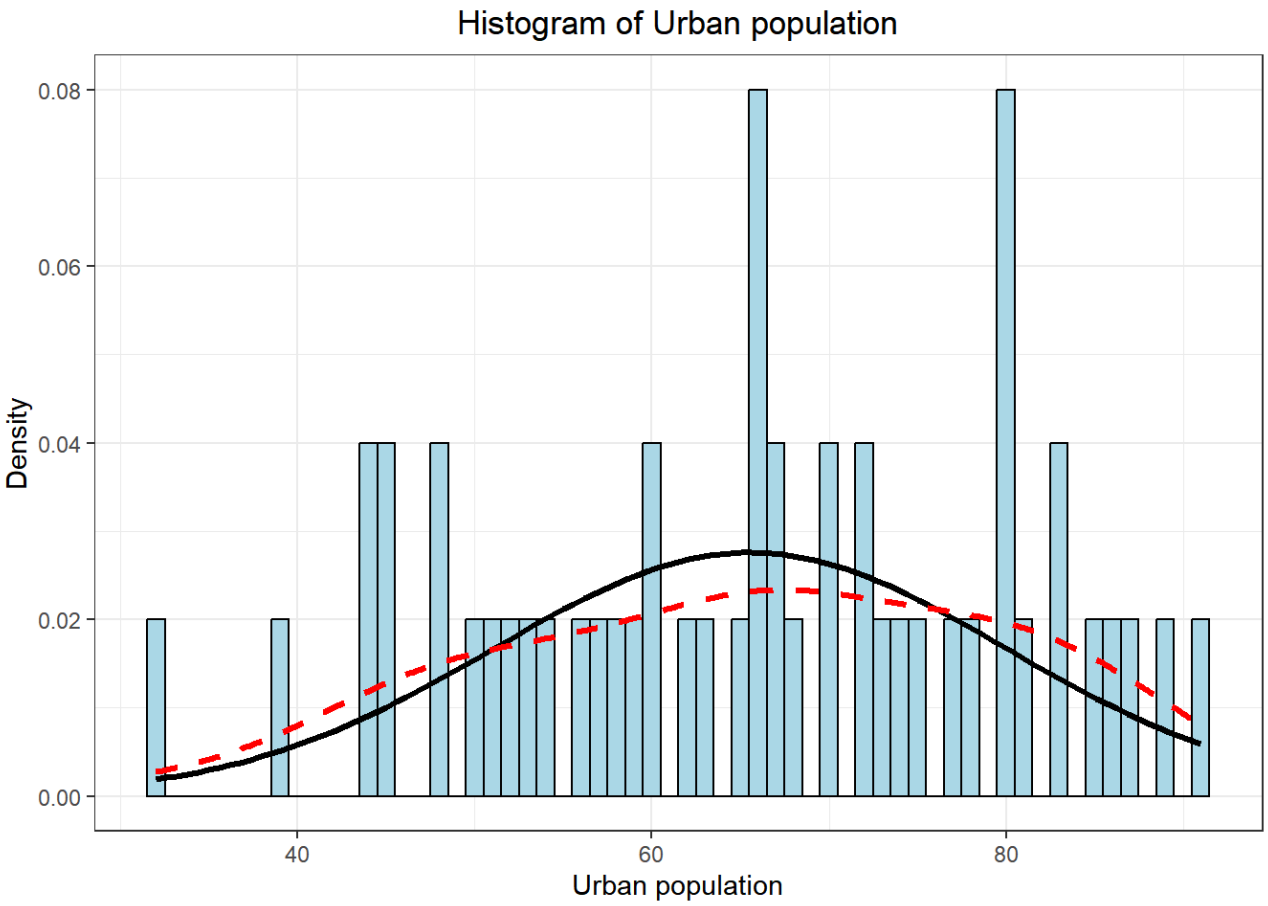
```
#Basic density plot
ggplot(data = USA, aes(x=UrbanPop)) +
  geom_density(fill = "lightblue") +
  labs(title = "Urban population")+ theme_bw()
```



We can see that the density plot does not follow a clear or symmetrical pattern. To explore this further, you can overlay a normal distribution curve based on the variable's mean and standard deviation.

```
#adding density line in the histogram plot
ggplot(USA, aes(x = UrbanPop)) +
```

```
geom_histogram(aes(y = ..density..),
               binwidth = 1,
               fill = "lightblue",
               color = "black") +
stat_function(fun = dnorm, #draw a normal distribution curve
              args = list(mean = mean(USA$UrbanPop, na.rm = TRUE),
                          sd = sd(USA$UrbanPop, na.rm = TRUE)),
              color = "black",
              size = 1.2) +
geom_density(color = "red", linetype = "dashed", size = 1.2) + #Draw actual data density
labs(title = "Histogram of Urban population",
     x = "Urban population",
     y = "Density") +
theme_bw() +
theme(plot.title = element_text(hjust = 0.5))
```



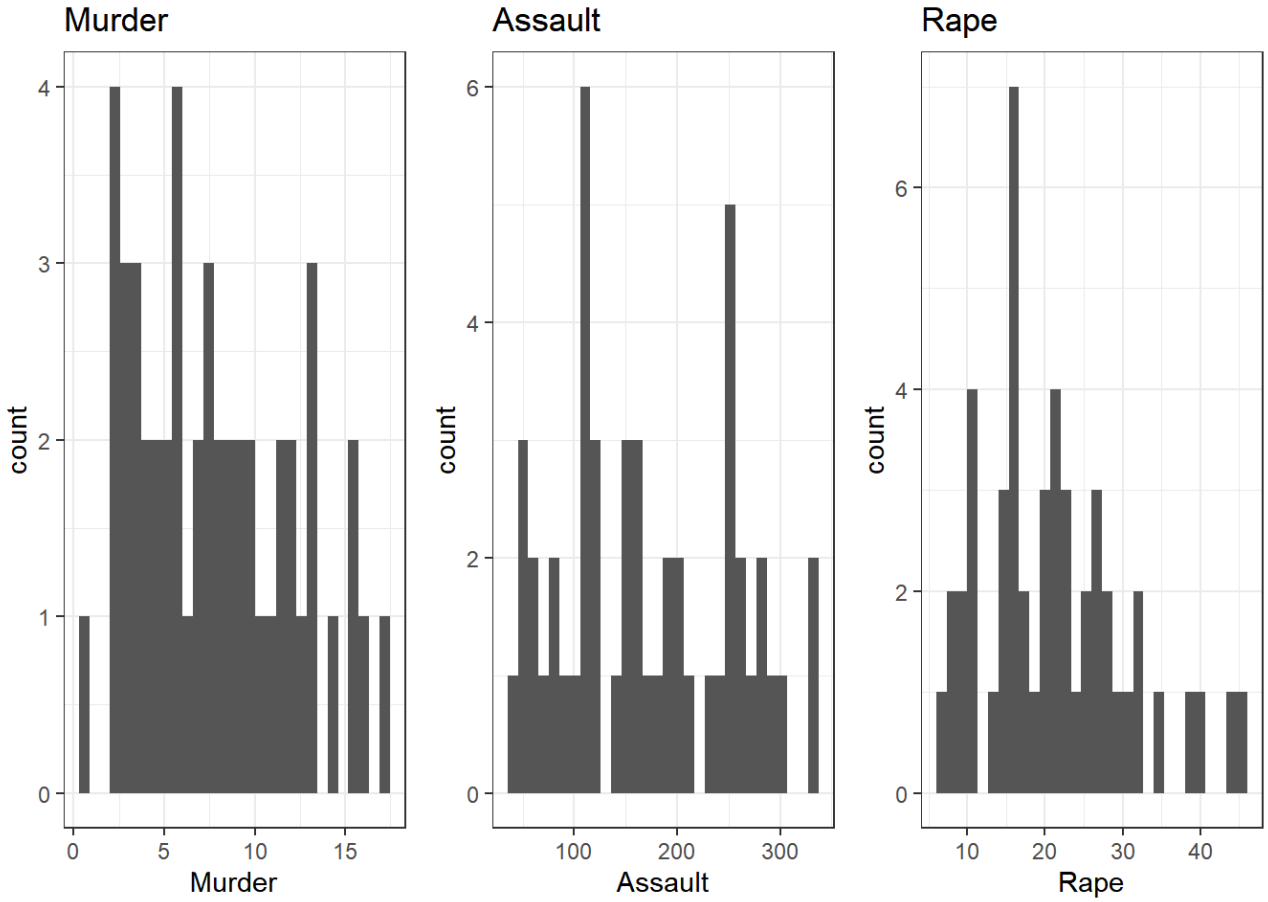
- Solid black line: Theoretical normal distribution
- Red dashed line: Actual data density

Q. Based on the comparison between the theoretical and actual density curves, how would you assess whether the data is normally distributed?

Write your response here:

You can repeat the above procedure to visualise the distributions of other crime-related variables such as *Murder*, *Assault*, and *Rape*. Instead of presenting each plot individually, we can combine them into a single display using the `grid.arrange()` function.

```
# Individual histogram plots
p1 <- ggplot(data = USA, aes(x=Murder)) +geom_histogram() + labs(title = "Murder") +
theme_bw()
p2 <- ggplot(data = USA, aes(x=Assault)) +geom_histogram() + labs(title ="Assault") +
theme_bw()
p3 <- ggplot(data = USA, aes(x=Rape)) +geom_histogram() + labs(title ="Rape") +
theme_bw()

install.packages("gridExtra")
library(gridExtra)

# Arrange plots in one row
grid.arrange(p1, p2, p3, ncol = 3)
```



Q. Interpret the distribution patterns observed in each histogram. Are any of the variables skewed?

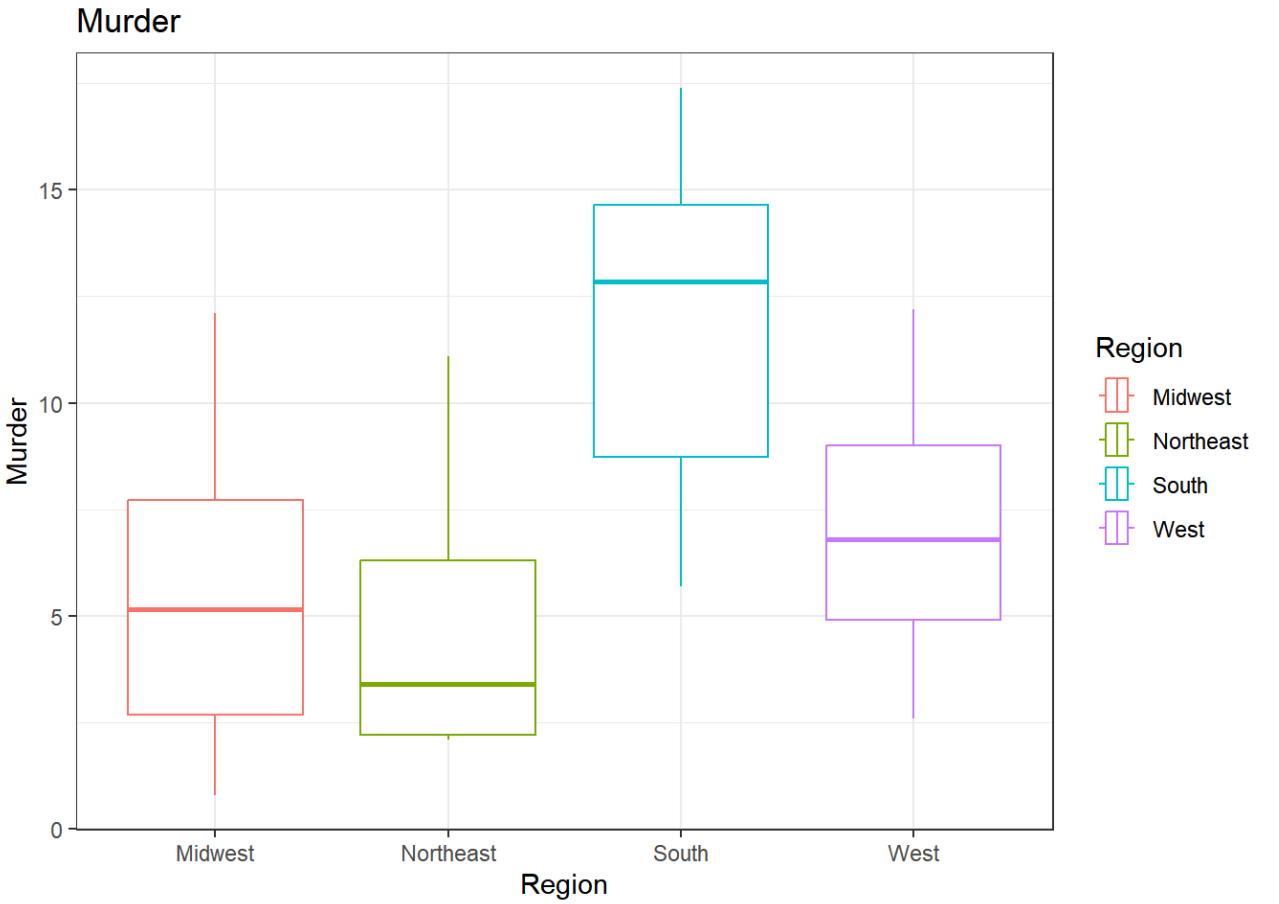
Write your response here:

2.2 Box Plot

Box plots are a useful tool for visualising the distribution of a continuous variable and identifying the central tendency, spread, and outliers. In this example, we will examine murder rates across different U.S. regions.

```
ggplot(data = USA, aes(x= Murder, y = Region, color= Region)) +
```

```
geom_boxplot() +
labs(title = "Murder") + #Add the title "Murder" to the plot
theme_bw() + #Apply a clean black-and-white theme.
coord_flip() # Flips the x and y axes
```



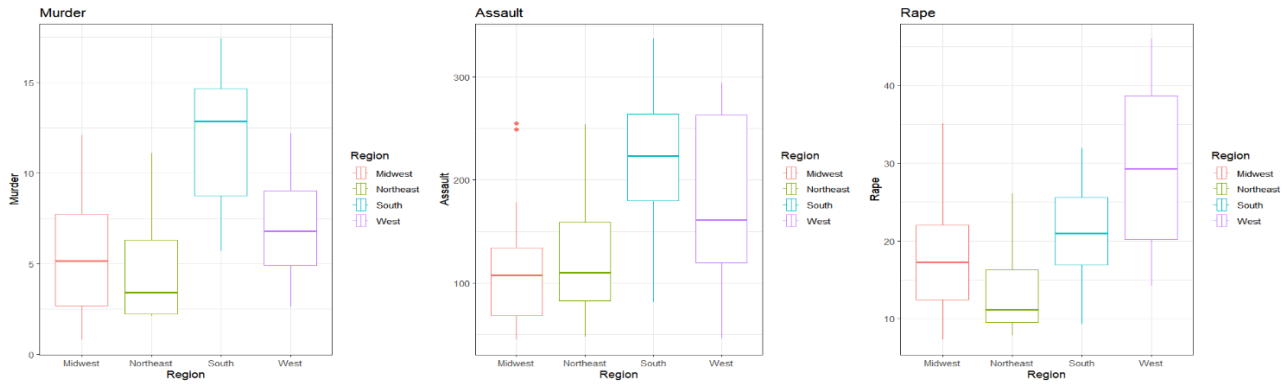
Q. Which region shows the highest median murder rate? Can you identify any outliers?

Write your response here:

Sometimes, it is helpful to display individual data points alongside the boxplot to observe the actual distribution of values within each group. One simple way to do this is by using the `geom_jitter()` function, which adds a small amount of random noise to prevent overlapping points. You can explore its usage further by typing `?geom_jitter` in the console.

Below, we compare all three crime variables: *Murder*, *Assault*, and *Rape* using box plots.

```
ggplot(data = USA, aes(x= Murder, y = Region, color= Region)) +
  geom_boxplot() +
  geom_jitter() + #Add individual data points with slight random noise (jitter) to avoid
overlap
  labs(title = "Murder") +
  theme_bw() +
  coord_flip()

#Save each plot without individual data points
b1 <- ggplot(data = USA, aes(x= Murder, y = Region, color= Region)) +geom_boxplot() +
labs(title = "Murder") + theme_bw() +coord_flip()
b2 <- ggplot(data = USA, aes(x= Assault, y = Region, color= Region)) +geom_boxplot() +
labs(title = "Assault") + theme_bw() +coord_flip()
b3 <- ggplot(data = USA, aes(x= Rape, y = Region, color= Region)) +geom_boxplot() +
labs(title = "Rape") + theme_bw() +coord_flip()

# Display all three plots side by side
grid.arrange(b1, b2, b3, ncol = 3)
```



Q. Interpret the results shown in the box plots. How do the distributions of the three crime variables compare across regions? Are there differences in variability or presence of outliers?

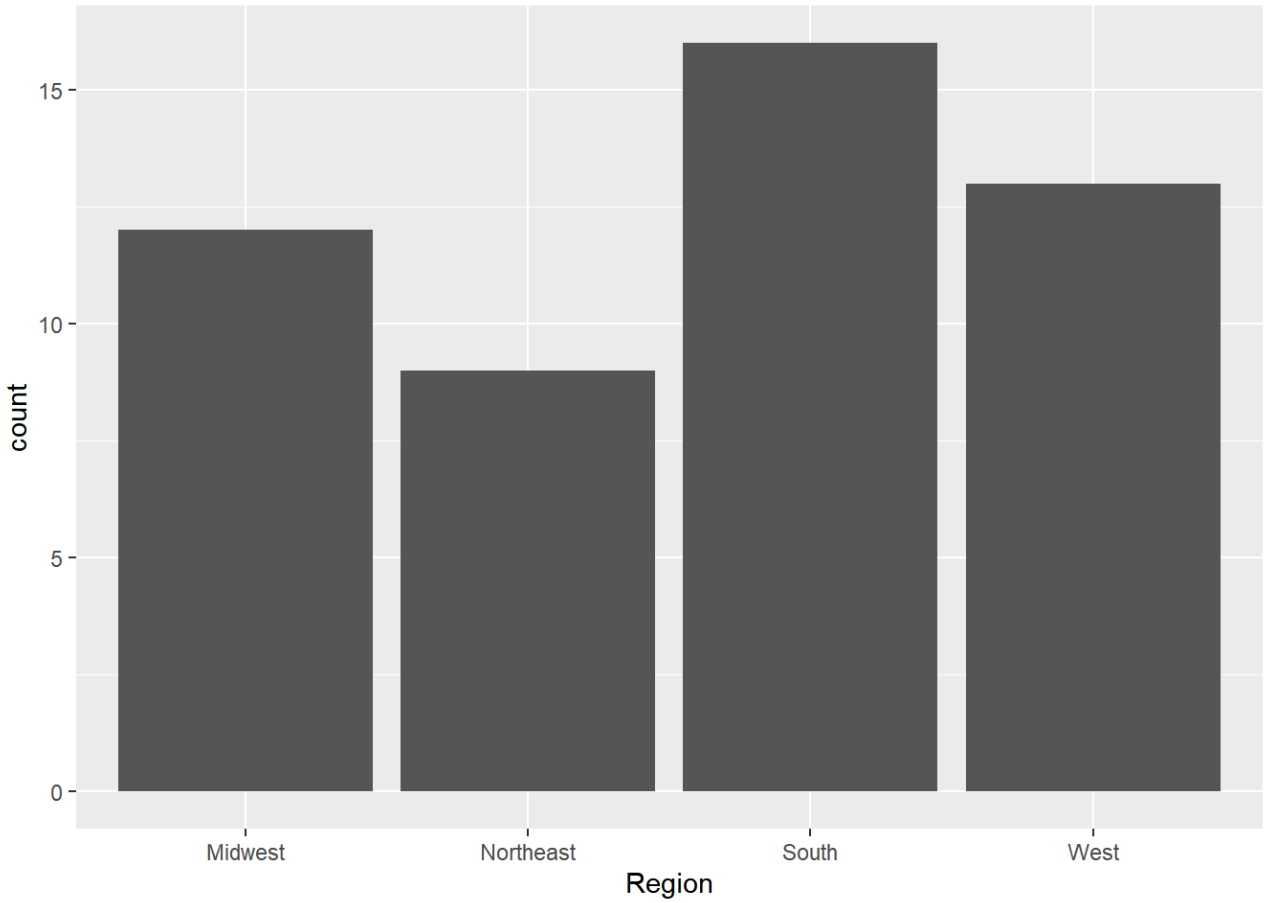
Write your response here:

2.3 Bar Plot

While histograms, density plots, and box plots are ideal for visualising numerical variables, bar plots are used to visualise the distribution of categorical variables. Bar plots are especially helpful for comparing the frequency or proportion of categories.

Let's begin by examining the number of states in each U.S. region:

```
# Basic bar plot showing counts by region
ggplot(USA) +geom_bar(aes(x=Region))
```



In the output, you will notice that the South region has the highest count, with over 15 states falling under this category. This plot uses the default setting, which counts the number of observations for each category on the x-axis.

In many cases, you may already have a table summarising frequencies or proportions. For example:

```
USA_region <- USA %>%
  count(Region) %>% #  #Count number of rows per Region (creates 'n')
```

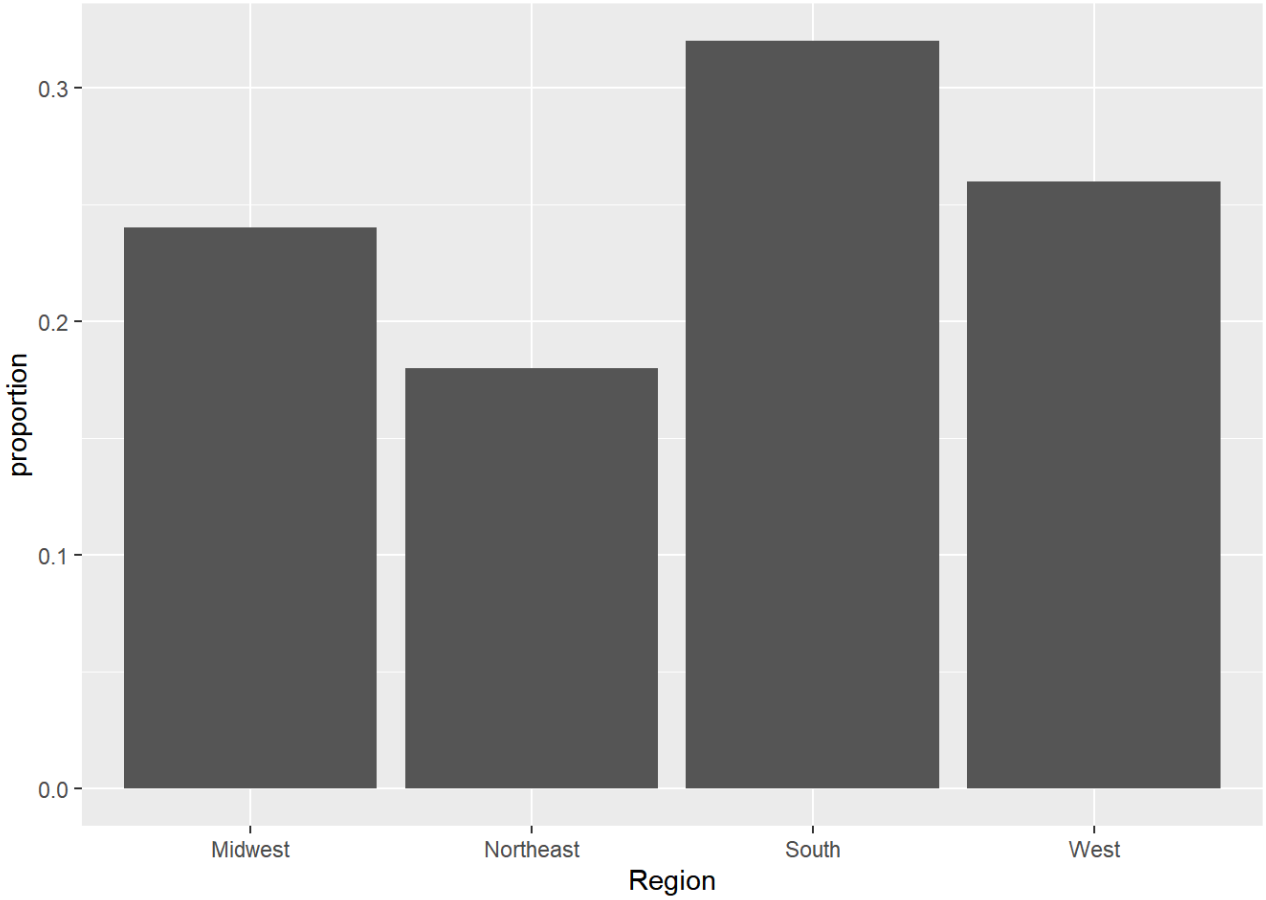
```
  mutate(proportion = n/sum(n)) #Calculate proportion by dividing by total count (creates
'proportion')
USA_region
```

Table 9.2

Region	n	proportion
Midwest	12	0.24
Northeast	9	0.18
South	16	0.32
West	13	0.26

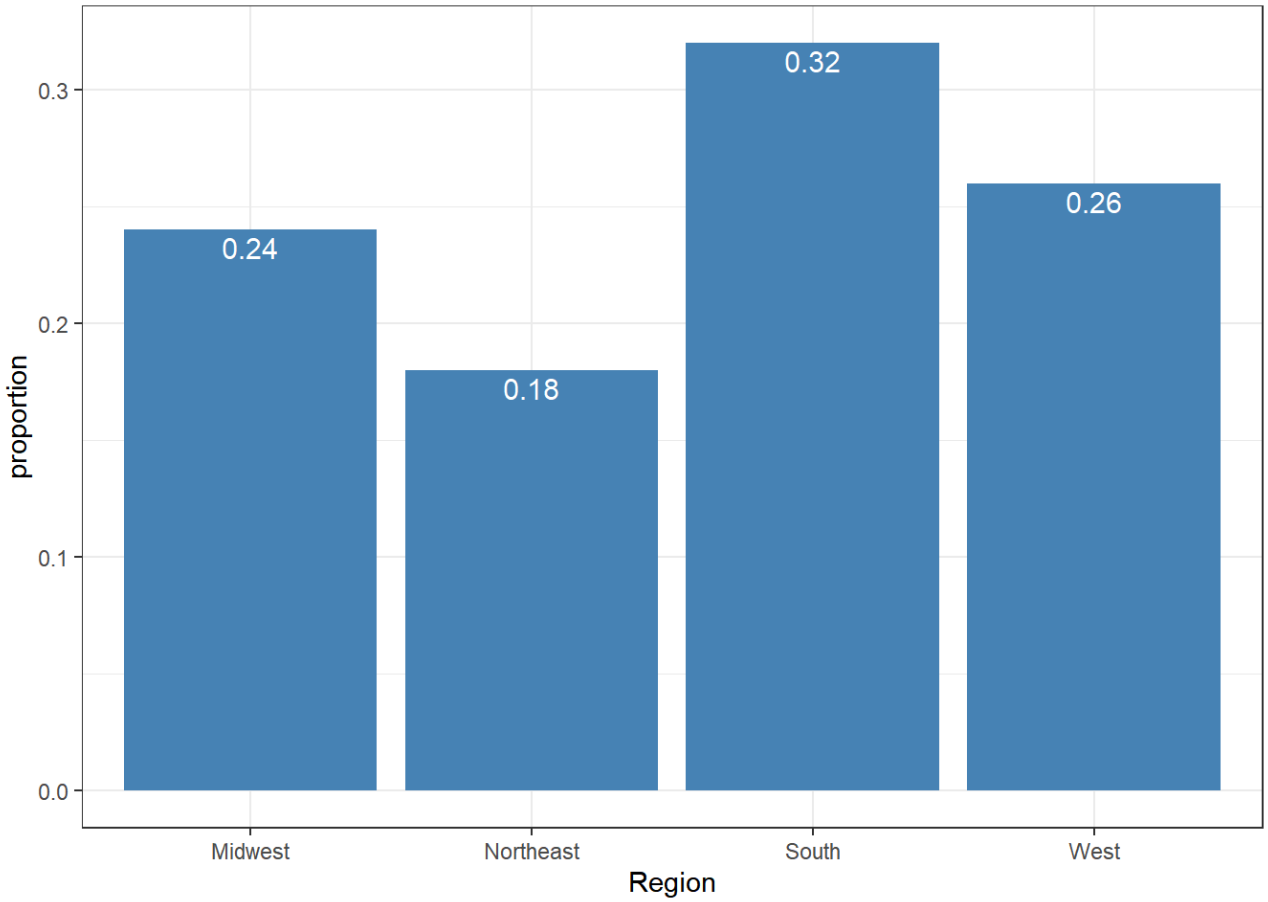
To visualise these pre-calculated proportions, you need to use `stat = "identity"` argument in your `geom_bar()` function. This tells *ggplot2* to use the provided y-values instead of computing counts internally.

```
# Bar plot using pre-calculated proportions
ggplot(USA_region) +geom_bar(aes(x=Region, y=proportion), stat = "identity")
```



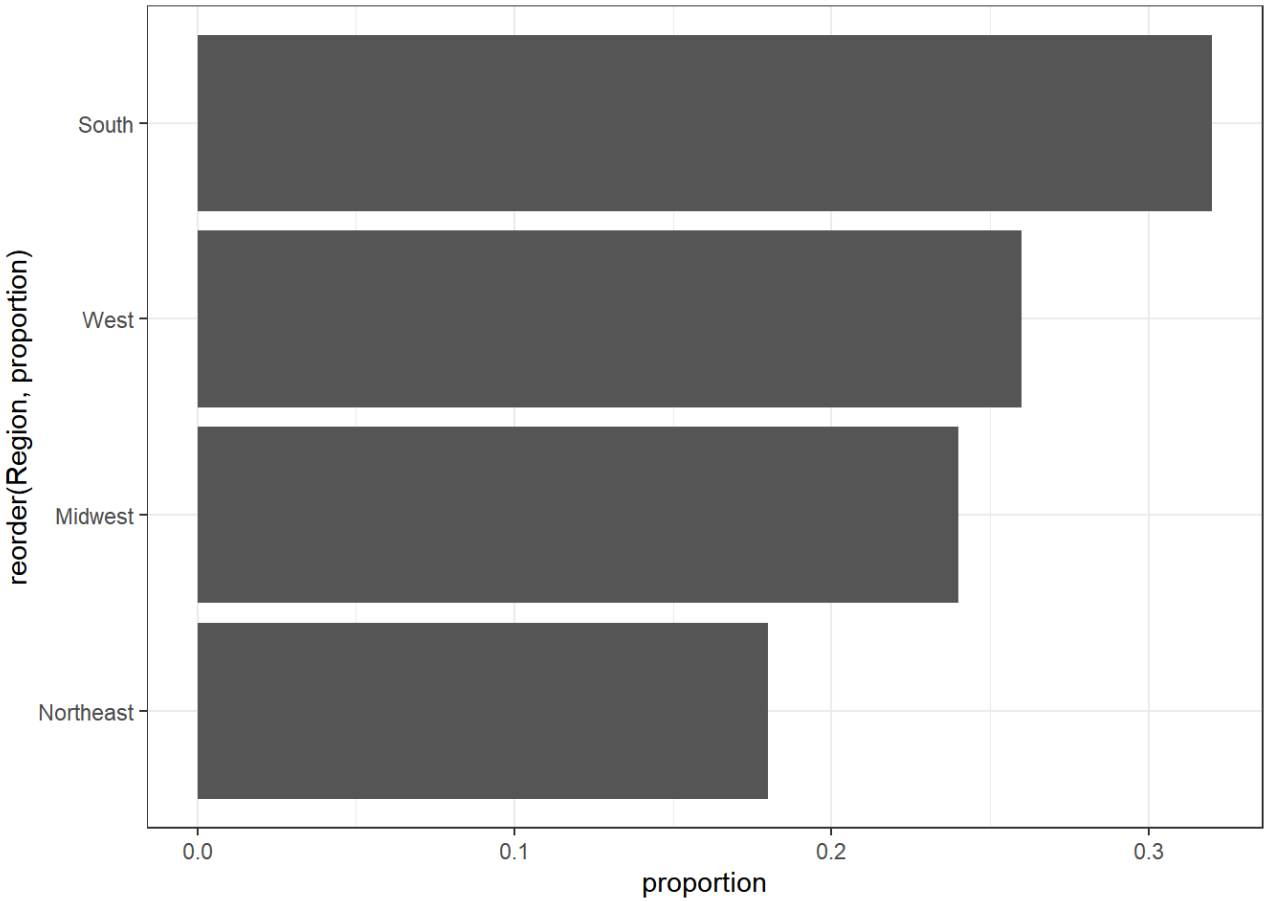
You can also customise the plot's appearance and add labels to display the proportion values on each bar.

```
ggplot(data = USA_region, aes(x=Region, y=proportion)) +
  geom_bar(stat = "identity", fill = "steelblue") +
     geom_text(aes(label=proportion), #Display the formatted percentage as the label.
               vjust = 1.3,     #Vertically adjust the label position (1.3 pushes it down
inside the bar).
               color = "white", #Set the text colour to white
               size =4)    +    #the size of the text label
  theme_bw()
```

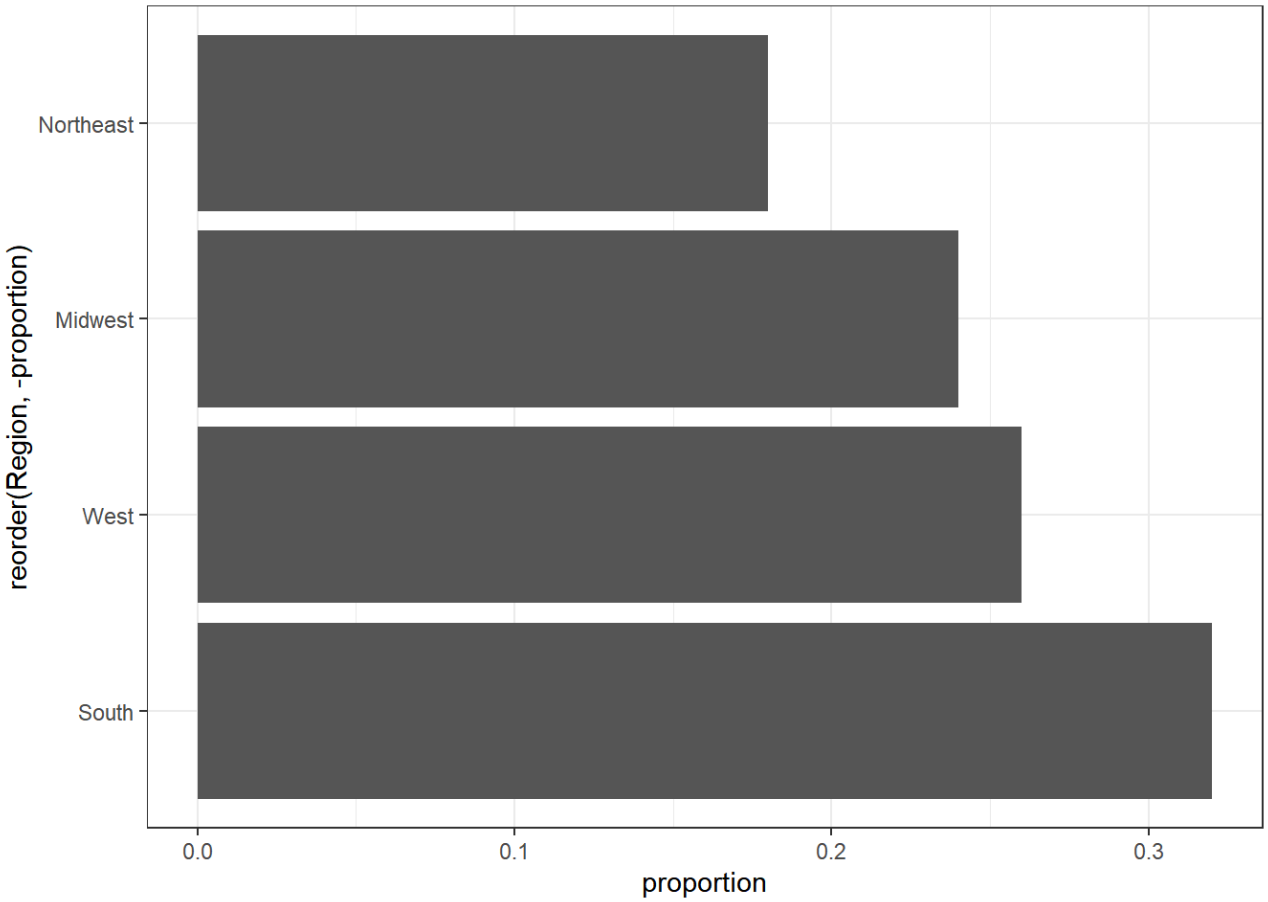


To improve readability, you can reorder the bars based on the proportions. Use the `reorder()` function in the `aes()` to sort categories in ascending or descending order. Combining it with `coord_flip()` can make horizontal bar plots that are often easier to read:

```
# Ascending order
ggplot(USA_region) +
  geom_bar(aes(x= reorder(Region, proportion),    #Reorder the bars based on the propor-
tion values from lowest to highest.
                y = proportion),
                stat = "identity") +
                 theme_bw()    +
                   coord_flip()
```




```
# Descending order
ggplot(USA_region) +
  geom_bar(aes(x= reorder(Region, -proportion), #Reorder the bars based on the proportion
values from highest to lowest.
                y = proportion),
                stat = "identity") +
                   theme_bw()    +
                   coord_flip()
```



2.4 Correlation Heat map

In Chapters 5 and 6, we explored correlation plots and added fitted regression lines to examine relationships between variables. In this section, we introduce another powerful visualisation technique: the correlation **heatmap**.

A heatmap represents data values using colour intensity, allowing you to easily visualise the strength and direction of correlations between variables. In this case, colours will indicate the magnitude of correlation coefficients, ranging from –1 (strong negative) to +1 (strong positive).

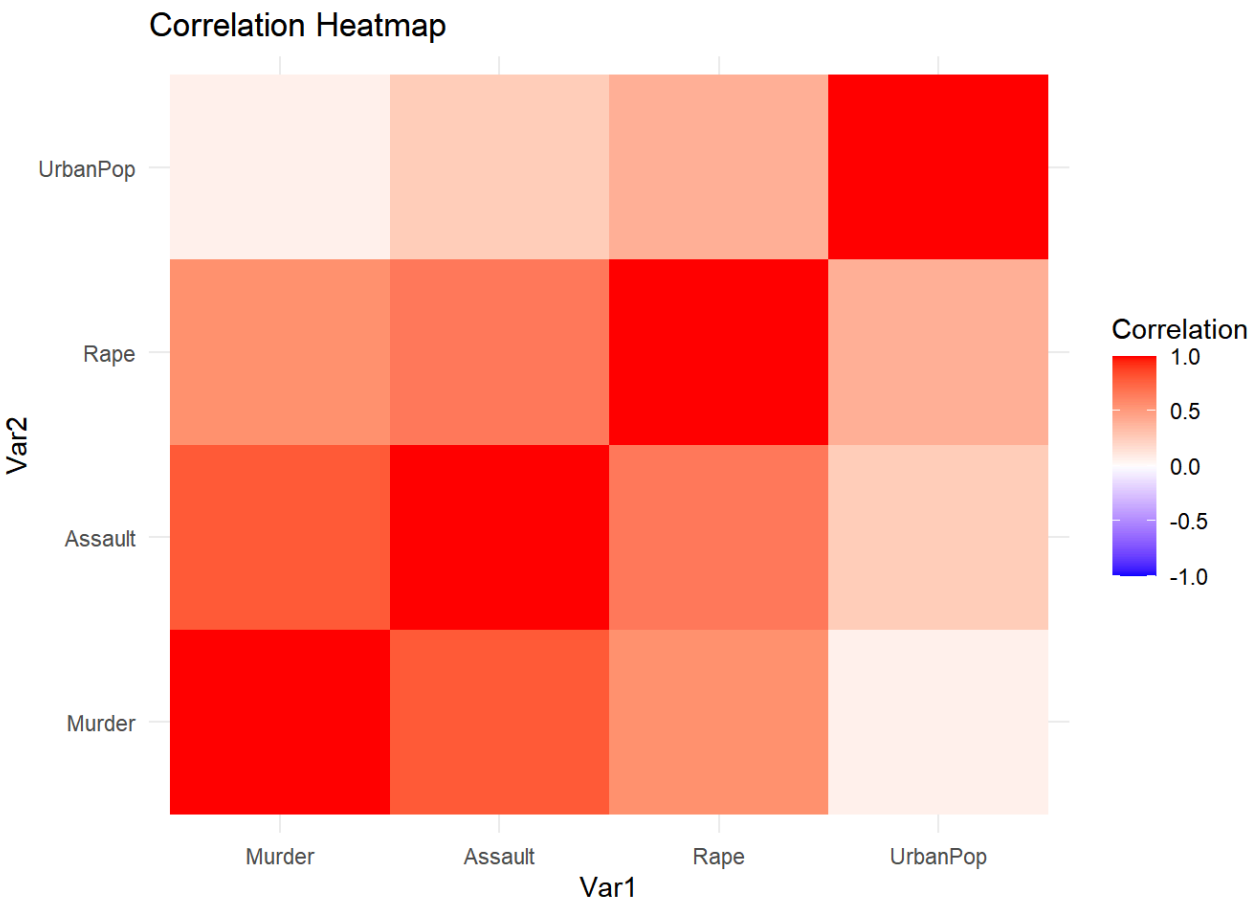
Let's create a heatmap using the `geom_tile()` function, with *UrbanPop* on one axis and the three crime related variables: *Murder*, *Assault*, and *Rape* on the other.

```
# Load required package
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

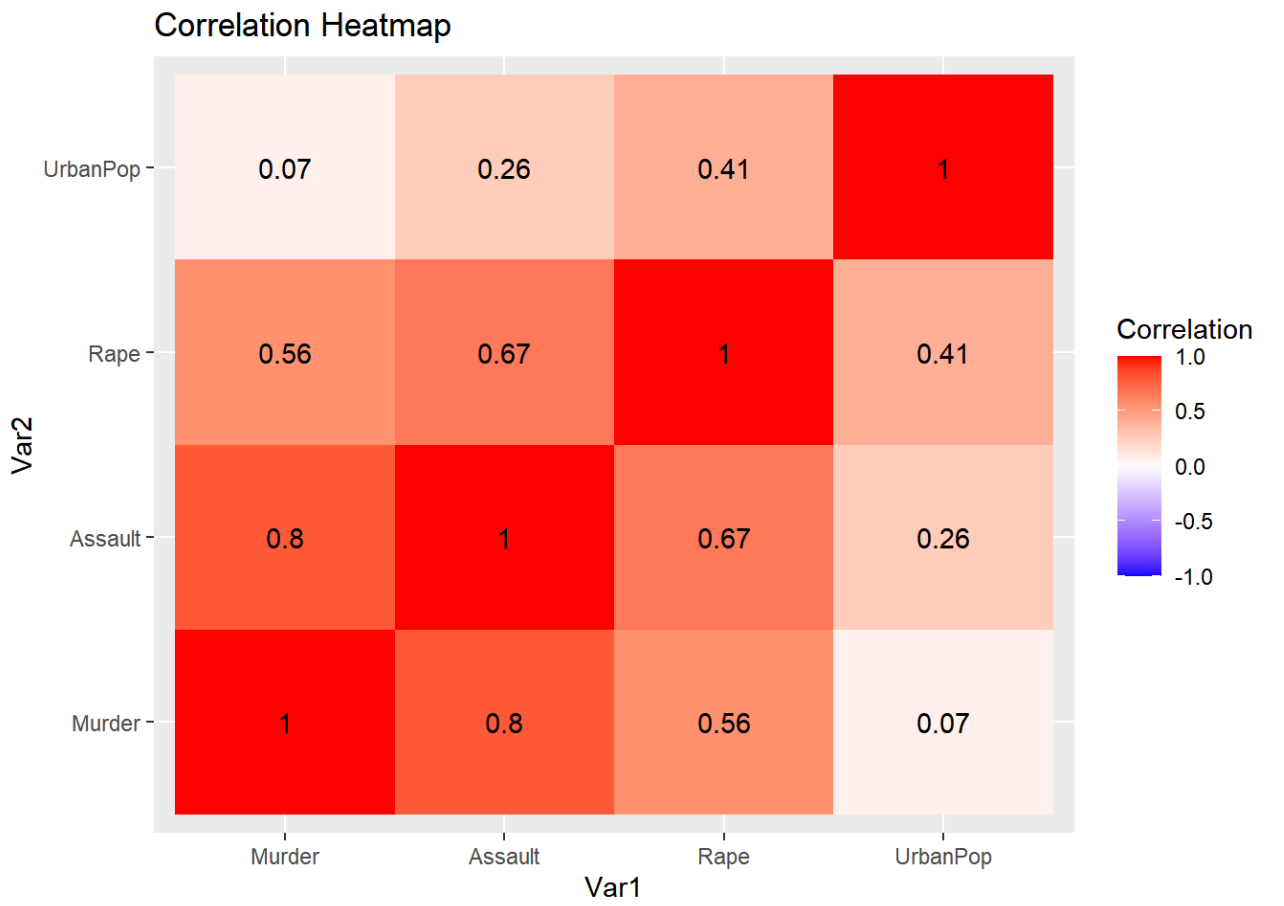
```
# Select relevant variables
cor_data <- USA %>%
  select(Murder, Assault, Rape, UrbanPop)  # Include the variable that you are interested in to explore the correlation

# Compute the correlation matrix
cor_matrix <- cor(cor_data, use = "complete.obs")

# Convert the matrix to long format
cor_melted <- melt(cor_matrix) #The correlation matrix is originally in wide (square) format, but ggplot2 requires data in long format for plotting. The 'melt()' function from the reshape2 package transforms it accordingly.

#Correlation heatmap
ggplot(cor_melted, aes(x = Var1, y = Var2, #Each tile of heat meap represents a pair of variables (Var1 and Var2)
                       fill = value)) +  #Each tile is filled with the corresponding correlation value.
  geom_tile() +
  scale_fill_gradient2(low = "blue", #Colour for negative correlations (closer to -1)
                       high = "red", #Colour for positive correlations (closer to +1)
                       mid = "white",#Colour at the midpoint, which is 0 (no correlation).
                       midpoint = 0, #Set value 0 as the midpoint
                       limit = c(-1, 1), #The range of correaltion values from -1 to 1
                       name = "Correlation") + #Title for the legend
  theme_minimal() +
  labs(title = "Correlation Heatmap")
```



Each tile in the heatmap represents a pairwise correlation between two variables, with colour indicating the strength and direction of the relationship.

You can also display the correlation coefficient values directly on the heatmap tiles:

```
# Plot heatmap of correlations with text labels
ggplot(cor_melted, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  geom_text(aes(Var2, Var1,                    #Places text at each tile
                label = round(value ,2))) + #Present the correlation value, rounded to 2
decimal place
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit =
c(-1, 1), name = "Correlation") +
  labs(title = "Correlation Heatmap")
```



Q. Which pair of variables shows the highest positive correlation? Which pair has the weakest (or most negative) correlation?

Write your response here:

2.5 Time Series

Time series visualisations are essential for exploring data collected over multiple time periods. These plots help reveal how a variable changes over time, making them particularly useful for longitudinal datasets. In this exercise, we use the **Victim_rate** dataset.

```
# Import the dataset
```

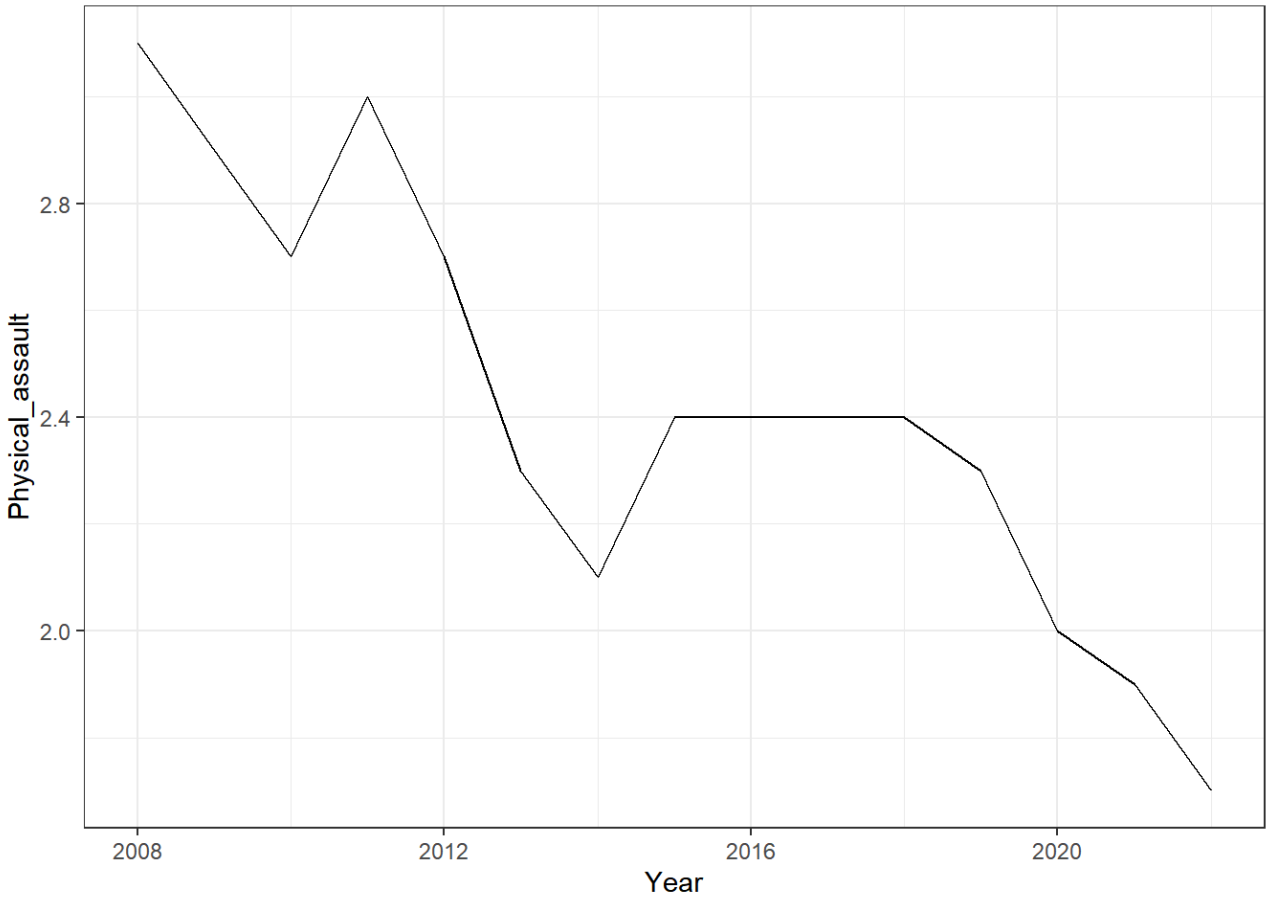
```
Victim <- read.csv("Victimisation_rate.csv")
head(Victim)
```

Table 9.3

	Year	Physical_assault	FacetoFace_threatened_assault	Non_facetoFace_threatened_assault	Robbery	Sexual_
1	2008	3.1	3.9	1.2	0.6	0.3
2	2009	2.9	3.1	0.8	0.4	0.3
3	2010	2.7	3.1	1	0.4	0.3
4	2011	3	3.3	1.2	0.4	0.3
5	2012	2.7	2.8	1.1	0.4	0.2
6	2013	2.3	2.7	1	0.4	0.3

Let's begin by examining how physical assault has changed over time.

```
# Simple line plot of physical assault over time
ggplot(Victim, aes(x= Year, y = Physical_assault)) + geom_line() + theme_bw()
```



Overall, this plot suggests a downward trend in the rate of physical assault in Australia over the years.

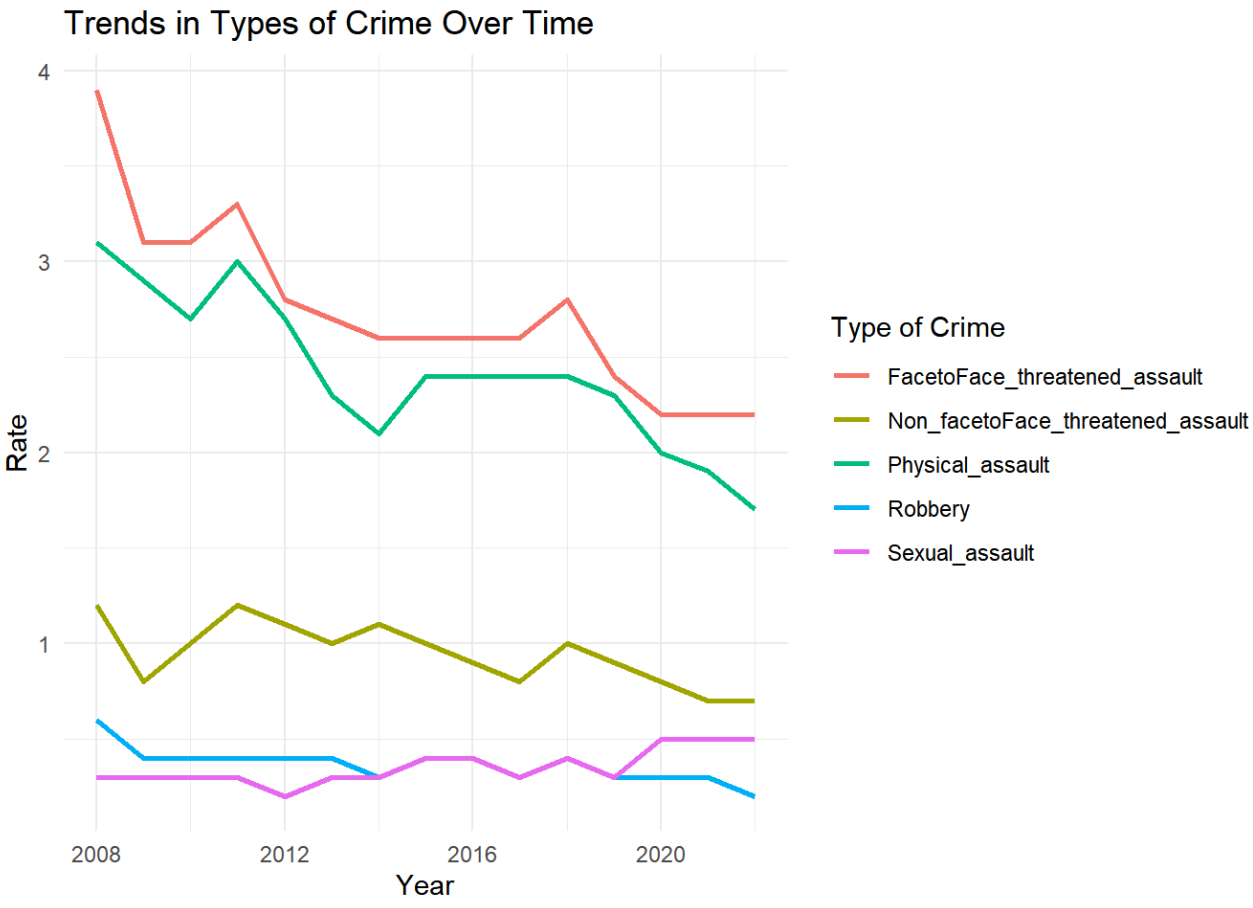
To display trends for multiple crime types on a single plot, we first need to **reshape** the data. The current structure is in wide format (each crime type in a separate column), but for plotting multiple lines, we need to convert it to long format using `pivot_longer()`.

```
# Reshape to long format
Victim_long <- Victim %>%
  pivot_longer(
    cols = c("Physical_assault", "FacetoFace_threatened_assault",
             "Non_facetoFace_threatened_assault", "Robbery", "Sexual_assault"),
    names_to = "Crime_Type",  #Create a new column called 'Crime_Type' to store the original column names
    values_to = "Rate"        #Create a new column called 'Rate' to store the corresponding values
    )

#pivot_longer() turns your dataset from wide to long, stacking all five crime types into
```

```
a single column with their values in Rate.
```

You can compare head(Victim) (wide format) vs head(Victim_long) (long format) to observe the transformation.

```
# Multi-Line Time Series Plot
ggplot(Victim_long, aes(x = Year, y = Rate, color = Crime_Type)) +
  geom_line(size = 1) +
  labs(title = "Trends in Types of Crime Over Time",
       x = "Year",
       y = "Rate",
       color = "Type of Crime") +
  theme_minimal()
```



This plot displays trends for all five crime types in one figure, allowing for easier comparison across years.

Q. Interpret the result. Which crime type shows the most fluctuation over time? Which appears to have the most stable trend?

Write your response here:

2.6 Mapping

The final exercise involves geospatial visualisation using maps. While a wide range of global and regional maps can be generated in R (e.g., using *worldmap*), this exercise focuses on mapping Australian states using real-world data. We will create choropleth and bubble maps to visualise state-level.

To do this, we will use the *ozmaps* package, which provides shapefiles for Australian geographic boundaries.

```
# Load Required Packages and Plot a Base Map
install.packages("ozmaps")
library(ozmaps)
library(sf)
```

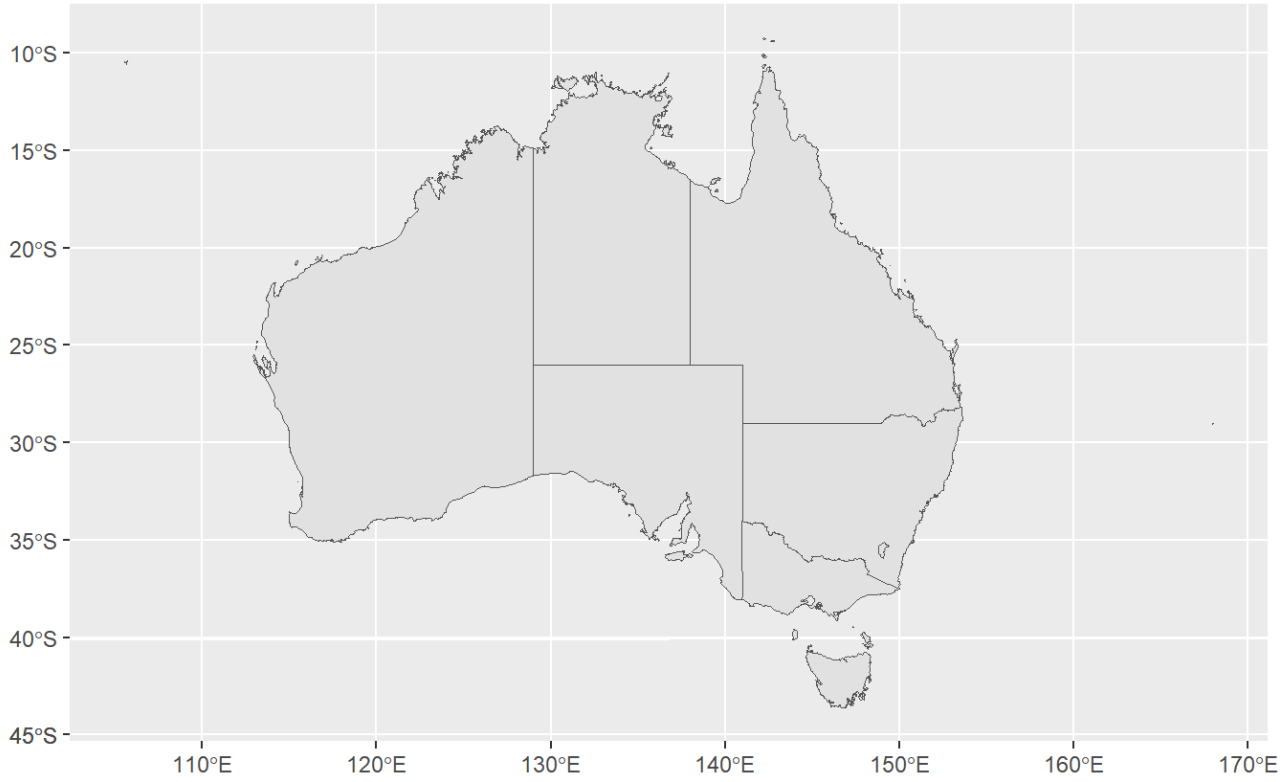
We can draw a simple map of Australia using the `ozmap()` function:

```
# Draw base map of Australia
ozmap()
```

Now, we extract the shapefile of Australian states and store it in an object for plotting in *ggplot2*.

```
# Load state-level map data
OZ_2023 <- ozmap("states")
```

```
#Create a map using ggplot
ggplot(data = OZ_2023) + geom_sf()
```



With the map data prepared, we can now add real-world state-level data. In this case, we map two variables:

- Net Overseas Migration (NOM) in 2023 (from ABS)
- International Student Commencements in 2023 (from the Department of Education)

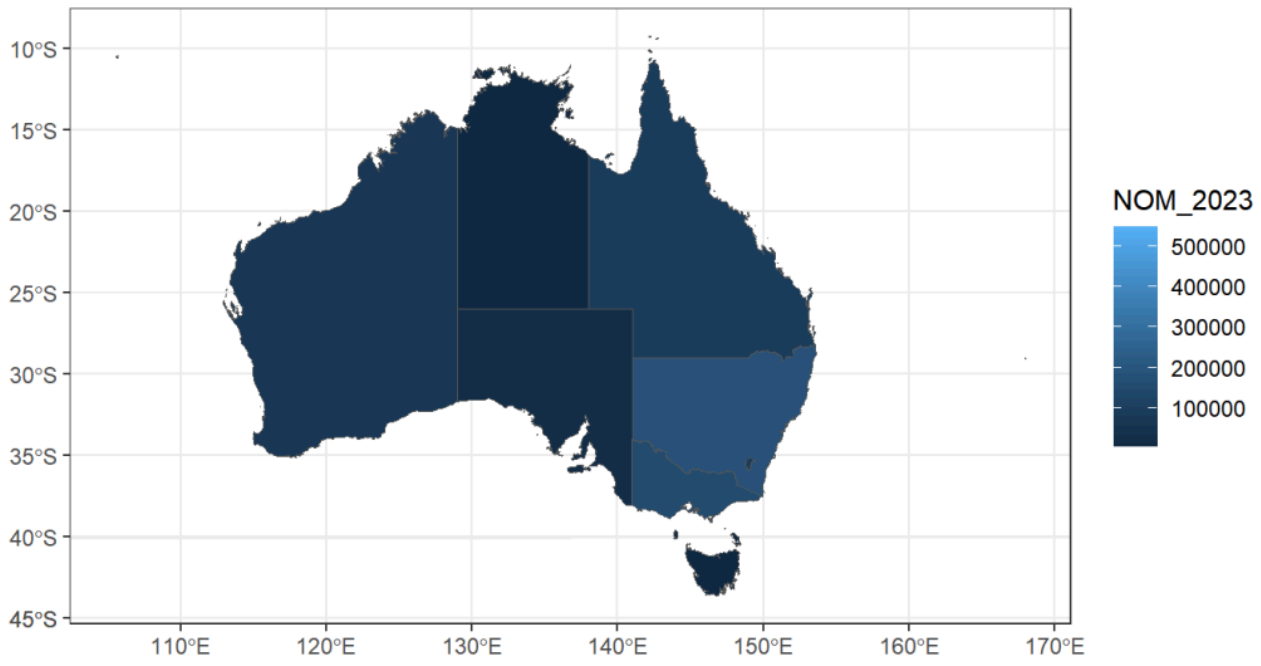
```
# Net Overseas Migration values in 2023
NOM_2023 <- c(185000 , 160000 , 88000 , 29000 , 69000 , 5100 , 4300 , 81000 , 547000)
 OZ_2023$NOM_2023 <- NOM_2023
OZ_2023
```

Table 9.4

	NAME	geometry	NOM_2023
1	New South Wales		185000
2	Victoria		160000
3	Queensland		88000
4	South Australia		29000
5	Western Australia		69000
6	Tasmania		5100
7	Northern Territory		4300
8	Australian Capital Territory		81000
9	Other Territories		547000

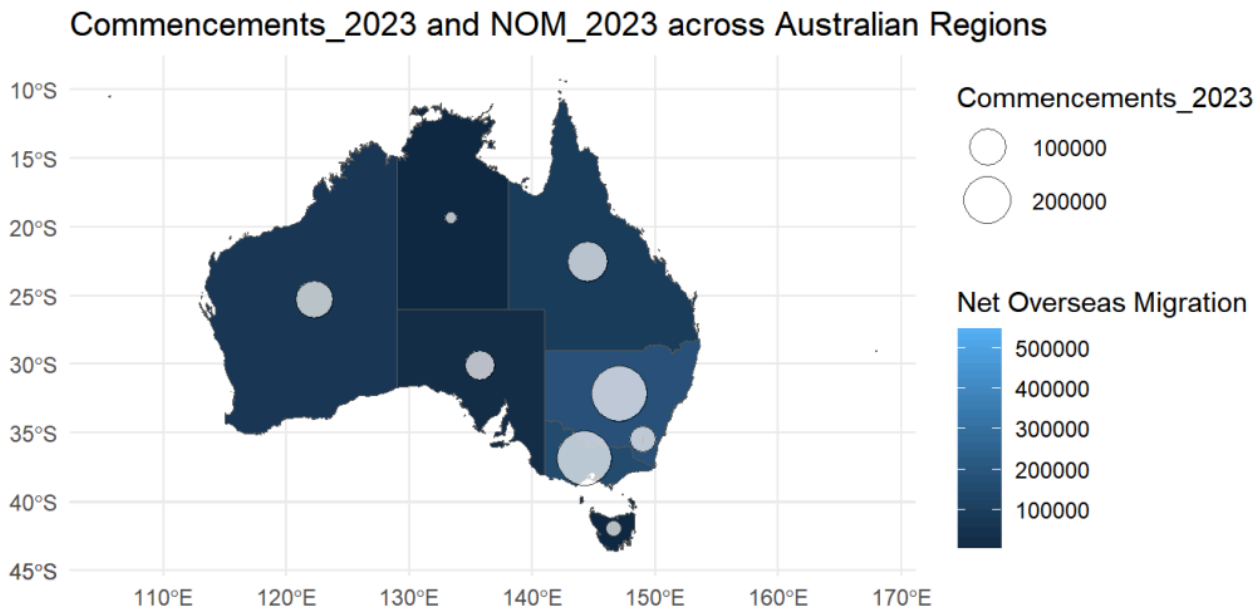
we have successfully included *NOM_2023* variable into the OZ_2023 dataset. let's map the net oversea migration in 2023 into the map:

```
ggplot(data = OZ_2023, aes(fill = NOM_2023)) + geom_sf() +theme_bw()
```



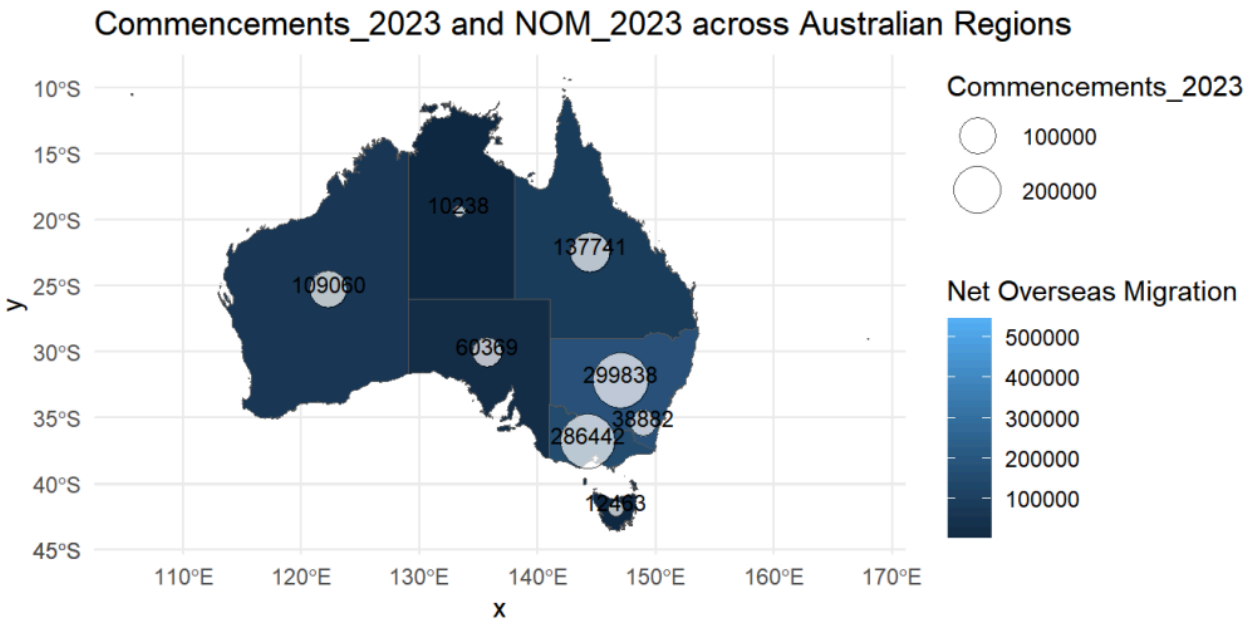
Now, let's include *Commencements_2023* variable into the dataset as well and draw a bubble map

```
#Commencements_2023: The number of new students who have started a course or program for
the first time.
Commencements_2023 <- c(299838, 286442, 137741, 60369, 109060, 12463, 10238, 38882, NA)
OZ_2023$Commencements_2023 <- Commencements_2023

st_centroid() # Computes the geometric center (centroid) of each shape (here center of
each state)
# Add Centroids for Bubble Placement
OZ_centroids_2023 <- st_centroid(OZ_2023)

# Plot Map with Choropleth and Bubbles
Result_2023 <- ggplot(data = OZ_2023) +
  geom_sf(aes(fill = NOM_2023)) +  # Base map with NOM_2023 as fill
  geom_sf(data = OZ_centroids_2023, aes(size = Commencements_2023), shape = 21, fill =
"white", color = "black", alpha = 0.7) +
  scale_size(range = c(2, 10)) +  # Adjust circle size
  labs(title = "Commencements_2023 and NOM_2023 across Australian Regions",
       fill = "Net Overseas Migration",
       size = "Commencements_2023") +
  theme_minimal()
Result_2023
```



You can also add labels to display the commencement values directly on the map using `geom_sf_text()`.

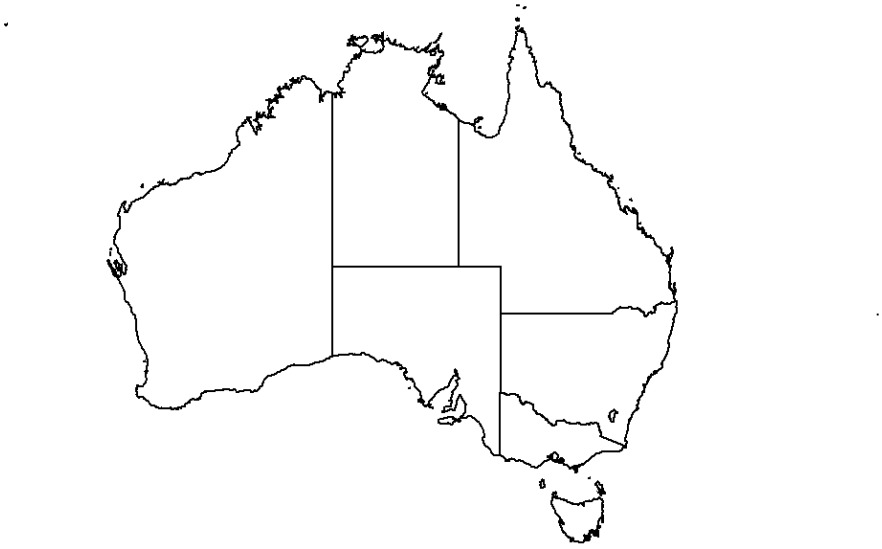
```
ggplot(data = OZ_2023) +
  geom_sf(aes(fill = NOM_2023)) +
  geom_sf(data = OZ_centroids_2023, aes(size = Commencements_2023),
          shape = 21, fill = "white", color = "black", alpha = 0.7) +
  geom_sf_text(data = OZ_centroids_2023, aes(label = Commencements_2023), size = 3,
nudge_y = 0.5) +  # Add text labels
  scale_size(range = c(2, 10)) +
  labs(title = "Commencements_2023 and NOM_2023 across Australian Regions",
       fill = "Net Overseas Migration",
       size = "Commencements_2023") +
  theme_minimal()
```



*Additional Exercise

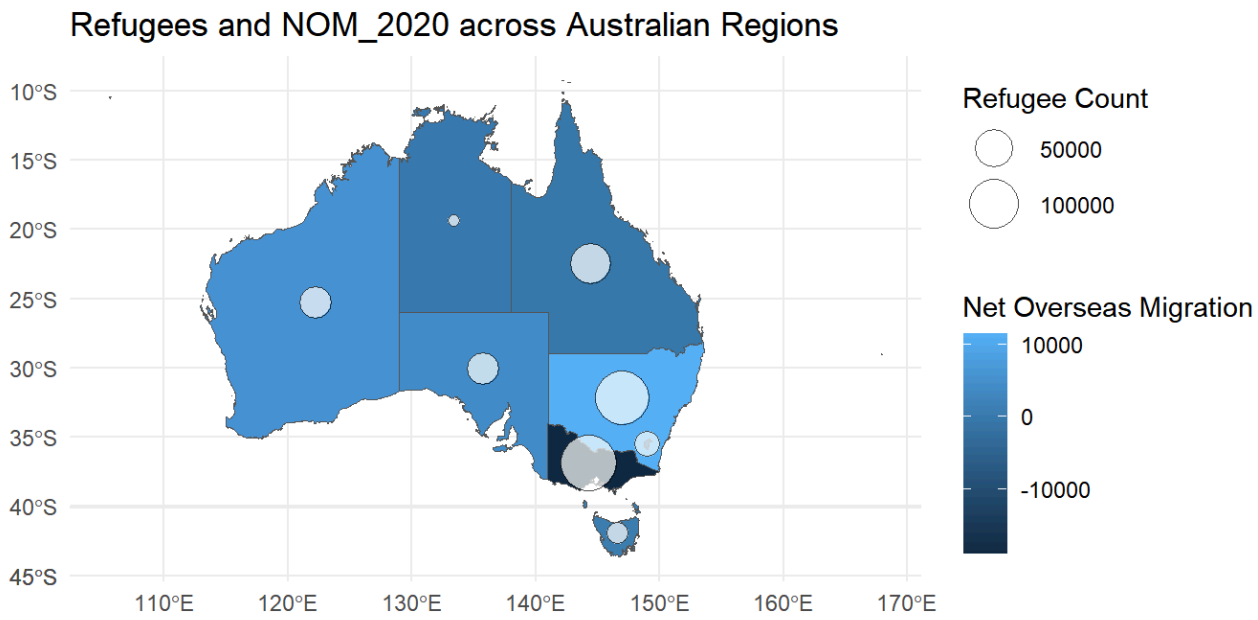
Let's draw a map with 2020 year and compare how the 2023 and 2020 were different.

```
#Load map
OZ_2020 <- ozmap("states")
```



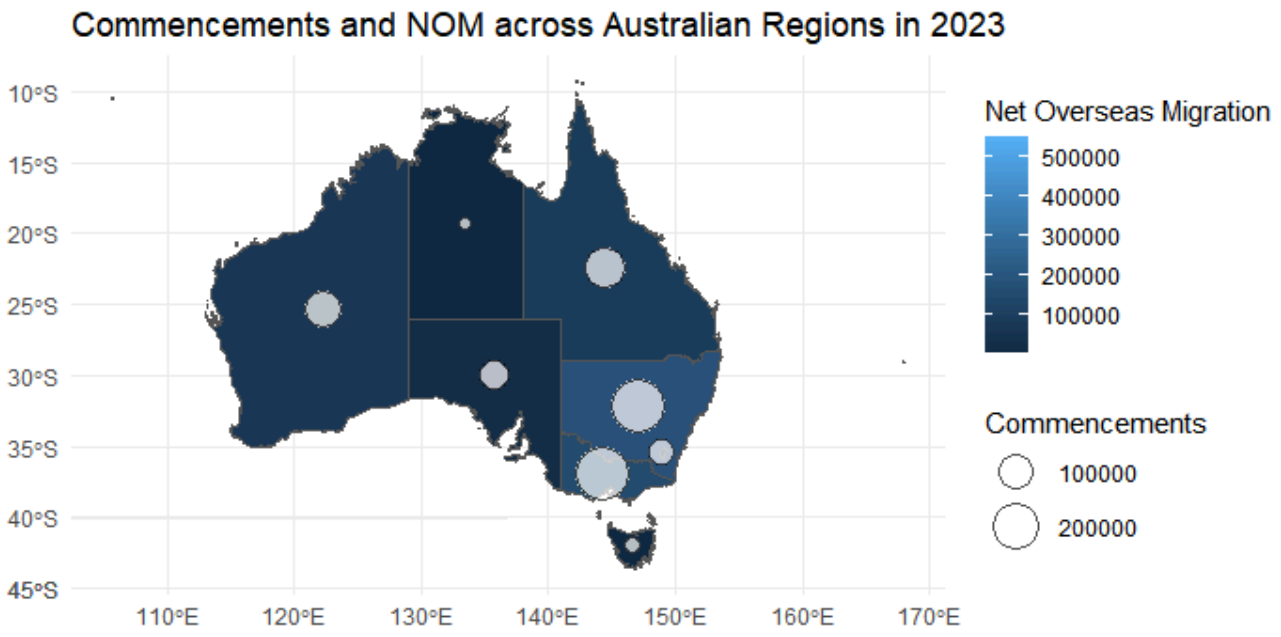
```
# Net Overseas Migration and International Student Commencements in 2020
NOM_2020 <- c(11513 , -18950 , -24 , 4414 , 5720 , 729 , 76 , -252 , 3253)
Commencements_2020 <- c(129993, 139747, 59176, 30330, 29922, 8586, 2391, 15806, NA)

OZ_2020$NOM_2020 <- NOM_2020
OZ_2020$Commencements_2020 <- Commencements_2020



# Add Centroids for Bubble Placement
OZ_centroids_2020 <- st_centroid(OZ_2020)

# Plot Map with Choropleth and Bubbles
Result_2020 <- ggplot(data = OZ_2020) +
  geom_sf(aes(fill = NOM_2020)) +  # Base map with NOM_2020 as fill
  geom_sf(data = OZ_centroids_2020, aes(size = Commencements_2020), shape = 21, fill =
"white", color = "black", alpha = 0.7) +
  scale_size(range = c(2, 10)) +  # Adjust circle size
  labs(title = "Refugees and NOM_2020 across Australian Regions",
       fill = "Net Overseas Migration",
```

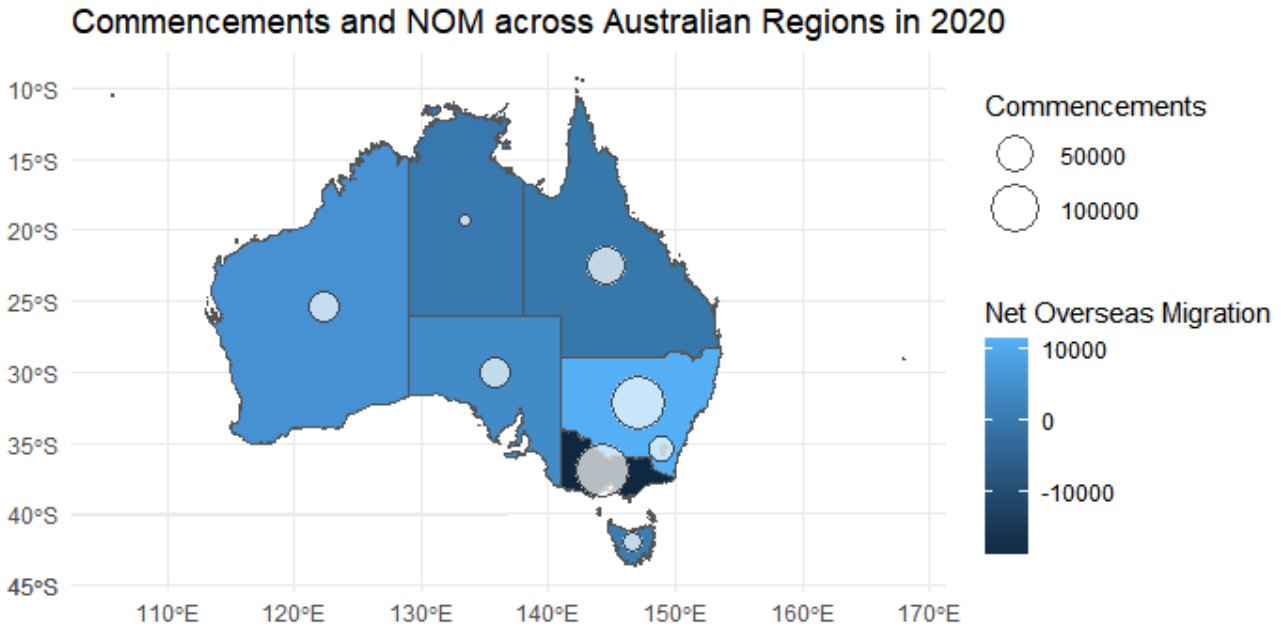
```
        size = "Refugee Count") +
  theme_minimal()
Result_2020
```



Let's compare the two years side by side:

```
grid.arrange(Result_2023, Result_2020, nrow = 2)
```





Q. Based on the maps, how did Net Overseas Migration and international student commencements differ between 2020 and 2023? What might explain these changes?

Write your responses here:

Once you complete all exercises, make sure to save your R script before closing R.

10.

NON-LINEAR ASSOCIATIONS

In this chapter, we will explore regression models that capture non-linear relationships between variables. Our focus will be on polynomial models, especially those with quadratic terms, which help us model curved relationships between a continuous predictor and an outcome variable. We will then practise visualising quadratic relationships, calculate turning points, and interpret the results.

1. A Non-linear Association

A quadratic function captures curvilinear patterns, such as U-shaped or inverted U-shaped trends where the direction of the association changes at a turning point. For example, student enrolments may initially increase and later decline due to policy shifts or demographic transitions. Similarly, income may rise during early career stages and plateau or fall in later years. By including a squared term ($\mathbf{X^2}$) of the independent variable, regression models can capture these non-linear dynamics.

In this exercise, we revisit the dataset provided by the Department of Education (available through the International student monthly summary: Interactive Report) and investigate the relationship between school enrolment numbers (dependent variable) and year (independent variable) from 2015 to 2023. We explore whether a quadratic functional form provides a better fit to the data than a linear regression model.

1.1 Load the Dataset and View Summary Statistics

We begin by creating a dataset that includes the annual number of school enrolments from 2015 to 2023:

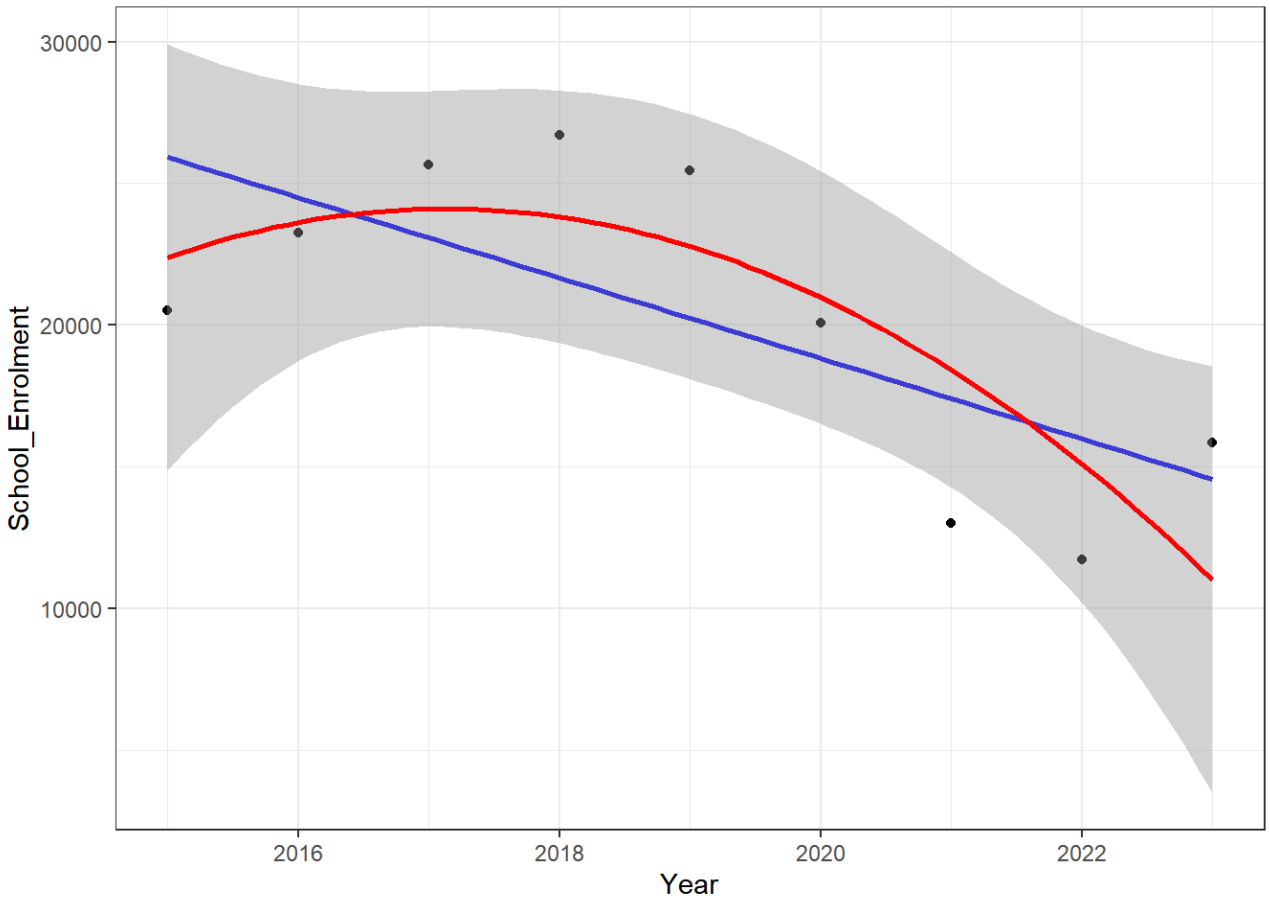
```
#Load the required packages for Chapter 10
library(dplyr)
library(psych)
library(ggplot2)
library(marginaleffects)

# Create a dataset
Data <- data.frame(Year = c(2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023),
```

```
                School_Enrolment = c(20526, 23245, 25644, 26708, 25459, 20070, 13019,
11714, 15830))
head(Data)
```

1.2 Visualising the Association Using a Scatter Plot

Next, we draw a scatterplot to explore the relationship between *School_Enrolment* and *Year*. To compare model fits, we overlay two regression lines: a **linear fit** (blue) and a **quadratic fit** (red).

```
ggplot(Data, aes(x = Year , y = School_Enrolment)) +
  geom_point() +  # Scatter plot
  geom_smooth(method = "lm", se = FALSE, formula = y ~ x, colour = "blue") +  # Linear fit
  geom_smooth(method = "lm", formula = y ~ x + I(x^2), color = "red") +# Quadratic fit
  theme_bw()
```



Q. Which of the two lines seems to fit the data better?

Write your responses here:

1.3 Fitting a Quadratic Model Using Manual Computation

To estimate a non-linear association between year and school enrolment, we include both the original predictor (*Year*) and its squared term (*sq_ Year*) as independent variables.

We begin by manually computing the squared term and then fitting the regression model:

```
# Generate the quadratic term manually
Data$sq_Year <- Data$Year^2 #^2 computes the square of the Year variable


# Fit a quadratic regression model
Manual_outcome <- lm(School_Enrolment ~ Year + sq_Year, data = Data)


# View model summary
summary(Manual_outcome)
```

```
##
## Call:
## lm(formula = School_Enrolment ~ Year + sq_Year, data = Data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5398.0 -1858.7  -377.9  2680.2  4814.0
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.546e+09  8.798e+08  -1.757    0.129
## Year         1.533e+06  8.715e+05   1.759    0.129
## sq_Year     -3.799e+02  2.158e+02  -1.760    0.129
##
## Residual standard error: 3788 on 6 degrees of freedom
## Multiple R-squared:  0.658,  Adjusted R-squared:  0.544
```

```
## F-statistic: 5.772 on 2 and 6 DF,  p-value: 0.04
```

Q. Based on this output, what do you conclude about the linearity of the relationship between Year and School_Enrolment? What piece of information did you use to reach your conclusion?

Write your responses here:

1.4 The Turning Point

In a quadratic model, the turning point represents the value of the predictor at which the direction of the relationship shifts. This point represents either the peak or the trough of the curve. For a model in the form:

$$X = \frac{-b_1}{2 \times b_2}$$

Here, b_1 refers to the coefficient on the linear term (Year), and b_2 refers to the coefficient on the quadratic term (sq_Year).

In R, we can calculate the turning point as follow:

```
# Extract coefficients
b1 <- coef(Manual_outcome)["Year"]
b2 <- coef(Manual_outcome)["sq_Year"]


# Calculate the turning point
turning_point <- -b1 / (2 * b2)
turning_point
```

```
##    Year
## 2017.13
```

Q. What is the value of the turning point?

Write your response here:

1.5 Visualising Model Predictions

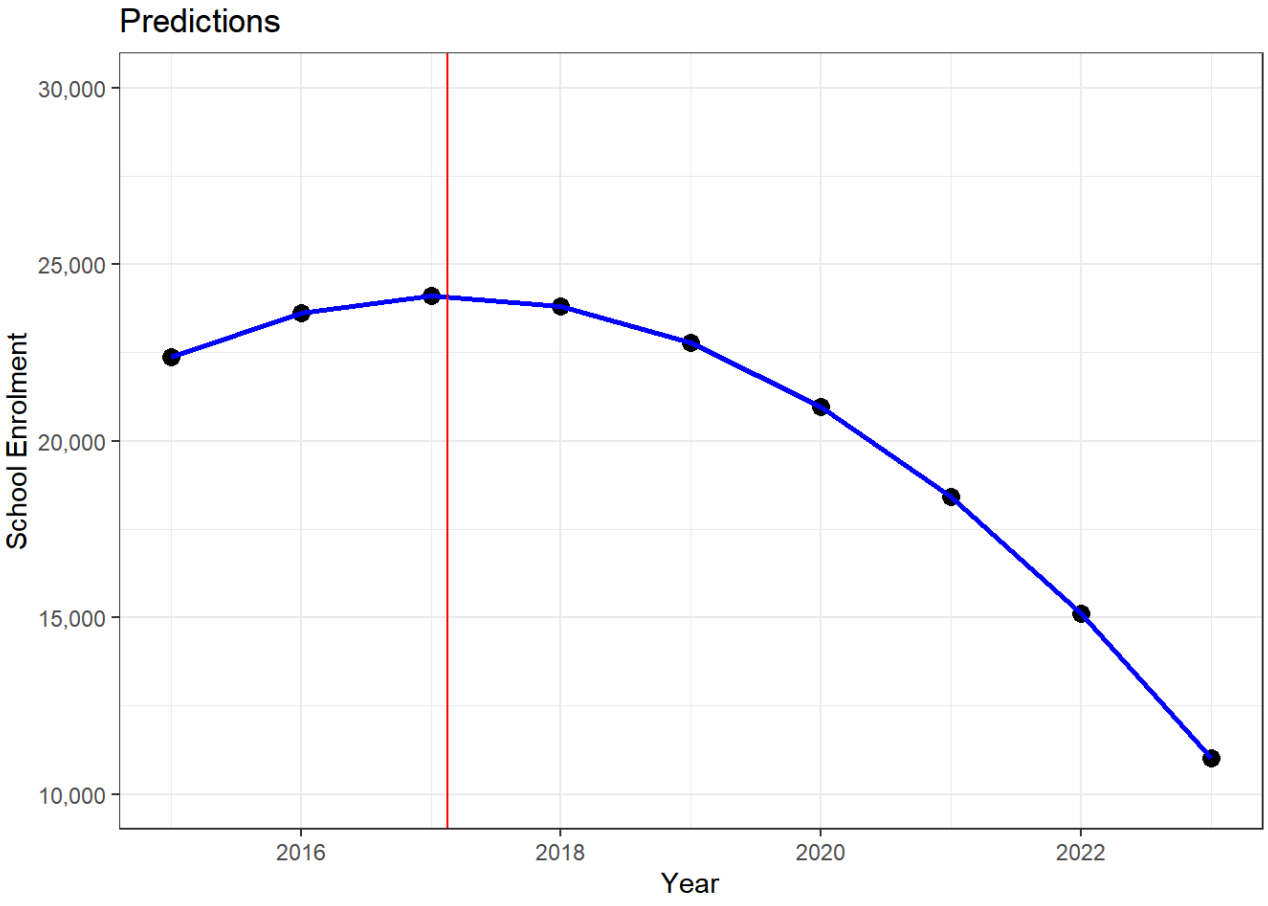
To help interpret the quadratic relationship more intuitively, we can generate **predicted values** from the fitted model and visualise them using a plot. We begin by creating a new dataset that includes the range of years used in the original analysis and then generate predicted school enrolment values using the `predictions()` function.

```
Pred <-predictions(Manual_outcome)
Pred
```

```
##
##  Estimate Std. Error     z Pr(>|z|)     S 2.5 % 97.5 %
##     22385       3073  7.28   <0.001  41.5 16362  28407
##     23623       2001 11.81   <0.001 104.4 19701  27545
##     24101       1697 14.21   <0.001 149.7 20776  27427
##     23820       1826 13.04   <0.001 126.8 20241  27399
##     22779       1917 11.89   <0.001 105.8 19022  26535
##     20978       1818 11.54   <0.001  99.9 17415  24541
##     18417       1697 10.85   <0.001  88.7 15090  21744
##     15096       2001  7.54   <0.001  44.3 11174  19019
##     11016       3094  3.56   <0.001  11.4  4953  17079
##
## Columns: rowid, estimate, std.error, statistic, p.value, s.value, conf.low, conf.high,
School_Enrolment, Year, sq_Year
## Type:  response
```

We then visualise the predicted values with the following plot:

```
ggplot(Pred, aes(x = Year, y = estimate)) +
  geom_point(data = Pred, aes( y = estimate), size = 3, color = "black") +  #predicted
values
  geom_line(color = "blue", linewidth = 1) +  # Predicted trend line
  geom_vline(xintercept = 2017.13, color = "red") +  #Turning point
  scale_y_continuous(
    limits = c(10000, 30000),
    breaks = seq(10000, 30000, by = 5000),
    labels = scales::comma  # Format numbers with commas
  ) +
  labs(
    title = "Predictions",
    y = "School Enrolment",
    x = "Year"
  ) +
  theme_bw()
```



Q. What insights can you draw from the plot of predicted values? How does the graph help you interpret the pattern of the association between Year and School Enrolment, particularly around the turning point?

Write your responses here:

2. Non-Linear Association Using Survey Data

We now turn to a non-linear modelling application using survey data. Income generally increases with accumulated work experience, often approximated by age. However, this relationship is rarely linear across the life course. Income may rise during early adulthood but tends to plateau or decline as individuals approach retirement. In this exercise, we use data from the World Values Survey (**WVS**) to examine how *income* varies with *age*, while controlling for education level and sex.

2.1 Load and Prepare the Data

We begin by importing the dataset and selecting relevant variables. For this analysis, we only focus on Australian respondents (Country code: 36).

```
#Set up the working directory first
setwd("C:/Your folder path/SOCYR")


#Load the dataset
WVS <- read.csv("WVS.csv")

#Data cleaning
WVS <- WVS %>%
   filter(Country ==36) %>% #36 - Australia
   select(Age, Income, Sex, EDU) %>%
   mutate(Sex =  factor(Sex,levels = c(1, 2), labels = c("Male", "Female")))


#View summary statistics
describe(WVS)
```

2.2 Fitting a Quadratic Model Using R syntax

In this exercise, we use the built-in `I(Age^2)` syntax to generate the quadratic term directly within the regression formula. We also control for sex and education.

```
# Fit quadratic regression
Outcome <- lm(Income ~ Age + I(Age^2) +Sex + EDU, data  = WVS)


# View model summary
summary(Outcome)
```

```
##
## Call:
## lm(formula = Income ~ Age + I(Age^2) + Sex + EDU, data = WVS)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8673 -1.1873  0.1683  1.2193  6.1037
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.6473110  0.4222981   6.269 4.61e-10 ***
## Age          0.0404180  0.0157252   2.570 0.010247 *
## I(Age^2)    -0.0005028  0.0001475  -3.409 0.000667 ***
## SexFemale   -0.2771026  0.0956675  -2.897 0.003822 **
## EDU          0.4558558  0.0281223  16.210  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.89 on 1679 degrees of freedom
##   (129 observations deleted due to missingness)
## Multiple R-squared:  0.1899, Adjusted R-squared:  0.188
## F-statistic: 98.39 on 4 and 1679 DF,  p-value: < 2.2e-16
```

Q. Based on this output, what do you conclude about the linearity of the relationship between *Income* and *Age*? What piece of information did you use to reach your conclusion?

Write your responses here:

2.3 Turning Point

To determine the age at which income peaks (or begins to decline), we calculate the turning point of the quadratic function using the coefficients from the regression model.

```
# Extract coefficients
b1 <- coef(Outcome)["Age"]
b2 <- coef(Outcome)["I(Age^2)"]

# Calculate the turning point
turning_point <- -b1 / (2 * b2)
turning_point
```

```
##      Age
## 40.19171
```

Q. What is the value of the turning point?

Write your responses here:

2.4 Plotting Model Predictions

We calculate the predicted income values based on the regression model and visualise them across a range of ages, holding the control variables (sex and education) constant.

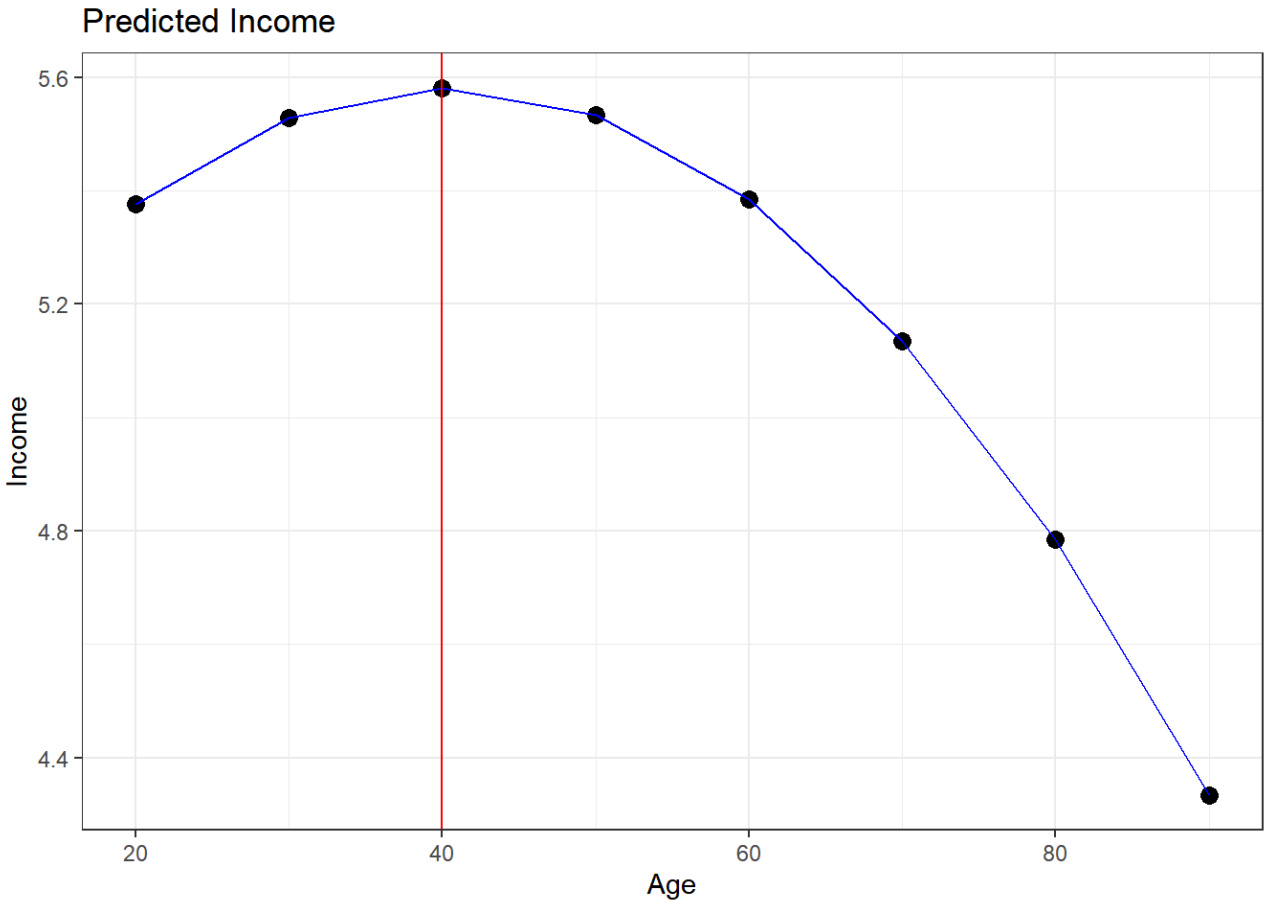
```
# Calculate the predicted income values (holding Sex and EDU constant)

Pred2 <- predictions(
  Outcome,
  newdata = datagrid(Age = seq(20, 90, by = 10),
                     Sex = "Male",
                     EDU = mean(WVS$EDU, na.rm = TRUE))
)

Pred2
```

```
##
##  Age  Sex  EDU Estimate Std. Error    z Pr(>|z|)     S 2.5 % 97.5 %
##   20 Male 4.66     5.38     0.1689 31.8   <0.001 736.0  5.05   5.71
##   30 Male 4.66     5.53     0.1101 50.2   <0.001   Inf  5.31   5.75
##   40 Male 4.66     5.58     0.0907 61.5   <0.001   Inf  5.40   5.76
##   50 Male 4.66     5.53     0.0881 62.8   <0.001   Inf  5.36   5.71
##   60 Male 4.66     5.38     0.0828 65.0   <0.001   Inf  5.22   5.55
##   70 Male 4.66     5.13     0.0826 62.2   <0.001   Inf  4.97   5.30
##   80 Male 4.66     4.78     0.1193 40.1   <0.001   Inf  4.55   5.02
##   90 Male 4.66     4.33     0.2021 21.4   <0.001 336.4  3.94   4.73
##
## Columns: rowid, estimate, std.error, statistic, p.value, s.value, conf.low, conf.high,
Age, Sex, EDU, Income
## Type:  response
```

Now, we visualise the predicted income at different ages, including a vertical line at the calculated turning point (age 40) to indicate where the income trajectory changes direction.

```
ggplot(Pred2, aes(x = Age, y = estimate)) +
  geom_point(size = 3, color = "black") +  # predicted observations
  geom_line(color = "blue") +
  geom_vline(xintercept = 40, color = "red") +  # Turning point
  labs(
    title = "Predicted Income",
```

```
    y = "Income",
    x = "Age"
  ) +
  theme_bw()
```



Q. What does the graph indicate about the relationship between age and income? How does the predicted curve help you understand the timing and direction of income changes across the life course?

Write your responses here:

Once you complete all exercises, make sure to save your R script before closing R.

11.

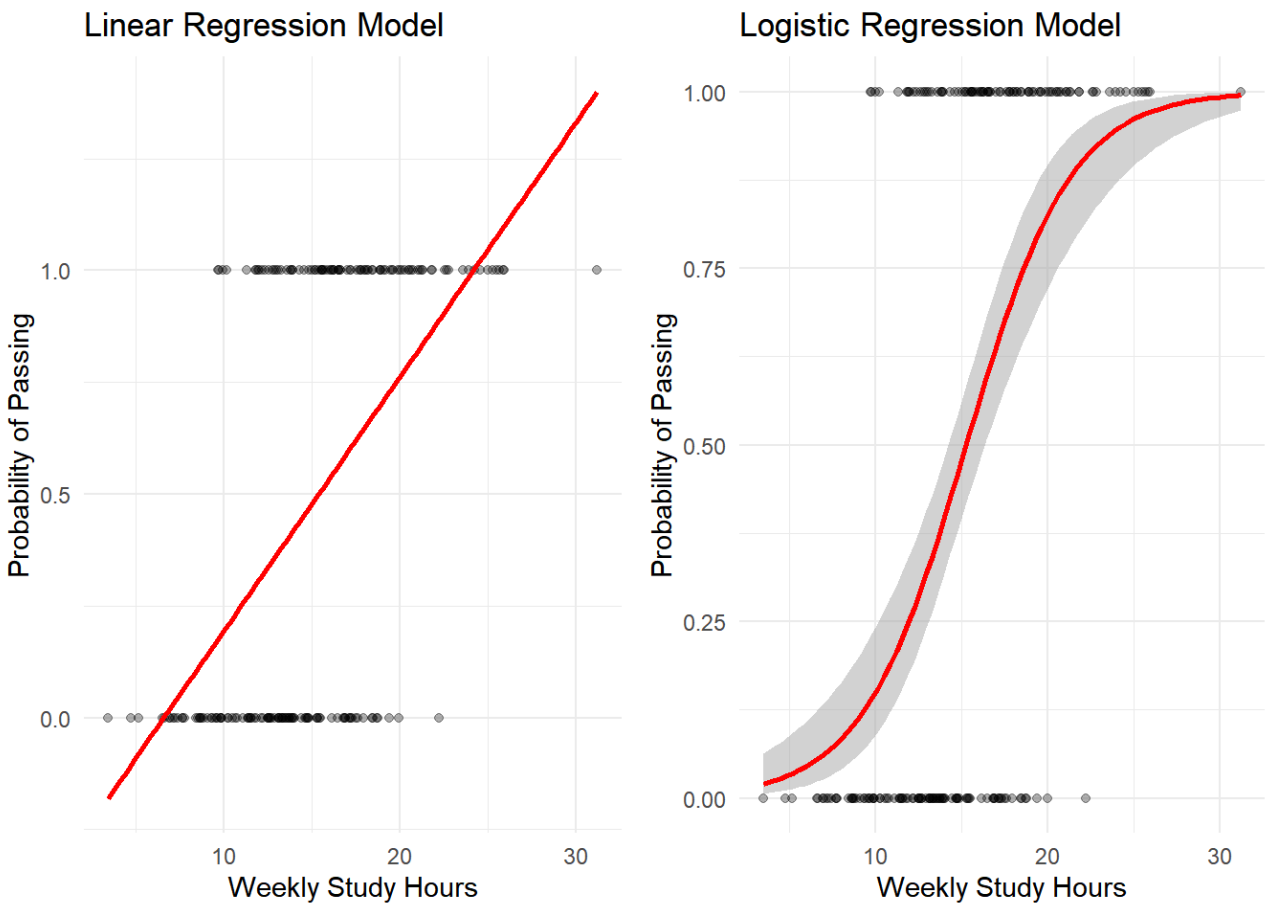
BEYOND LINEAR MODEL: LOGISTIC REGRESSION

This chapter focuses on practicing logistic regression. We will start with a brief overview of key concepts, followed by a hands-on application using the WVS data. First, we will fit a logistic regression model with a binary outcome and interpret the results using odds ratios and predicted probabilities. Then, we will practice recoding a continuous variable into binary form and apply logistic regression using the newly created binary outcome.

1. Logistic Regression

Linear regression is commonly used to model the relationship between a continuous outcome and one or more predictor variables. However, it is not suitable when the outcome is binary (e.g., Yes/No, Pass/Fail, Agree/Disagree). In such cases, logistic regression provides a more appropriate approach. While it shares many similarities with linear regression, logistic regression is specifically designed for categorical outcome and is widely used in the social sciences. In this session, we will explore key concepts and practical applications of logistic regression for modelling binary response variables.

For example, suppose we have data on students' weekly study hours and whether they pass the statistics course. We might initially consider applying linear regression to predict the probability of passing, as shown in the left plot in the figure below. However, linear regression can produce predicted probabilities below 0 or above 1, which are not meaningful. Logistic regression resolves this issue by modelling the probability of the outcome using a **logistic function**, which generates an S-shaped curve that appropriately constrains predicted probabilities between 0 and 1 (right plot).



<Predicted probabilities from linear regression (left) can fall outside the 0–1 range, while logistic regression (right) correctly constrains predictions between 0 and
1.>

Mathematically, the logistic function is defined as:

$$p(X) = \frac{e^{b_0+b_1X}}{1+e^{b_0+b_1X}}$$

Here, $p(X)$ represents the probability of the outcome (e.g., passing the course), and the coefficients b_0 and b_1 function similarly to those in linear regression.

Rearranging the logistic function gives the logit transformation:

$$g(X) = ln(\frac{p(X)}{1-P(X)}) = b_0 + b_1X$$

This transformation establishes a linear relationship between the predictor X and the log-odds of the outcome. The coefficient b_1 reflects the change in the log-odds associated with a one-unit increase in X,

and when exponentiated (e^{b_1}), it can be interpreted as the multiplicative change in the odds of the outcome occurring.

We will now practise fitting a logistic regression model using data from the World Values Survey (WVS). In this exercise, we aim to explore a binary outcome variable whether Australians believe **immigration increases crime**. We will examine how this perception is associated with age, sex, and education level

1.1 Load the Dataset and View Summary Statistics

Begin by loading the dataset and preparing the necessary the variables for analysis.

```
setwd("C:/Your Own Path/SOCYR")
#Load the dataset
WVS <- read.csv("WVS.csv")
```

```
#Load the required packages for Chapter 11
library(ggplot2)
library(dplyr)
library(tidyr)
library(gridExtra)
library(sjPlot)
library(ggeffects)
options(scipen = 999) #turning off scientific notation
```

```
#Data cleaning
AUS_WVS <- WVS %>%
  filter(Country == 36)%>%
  mutate(Sex = factor(Sex,levels = c(1, 2), labels = c("Male", "Female")))
```

The outcome variable: perceptions of whether immigration increases crime rates is originally coded with three categories: **0 = Disagree, 1 = Hard to say, and 2 = Agree**. Since logistic regression requires a binary outcome, we will recode these variables by treating responses coded as 1 (“Hard to say”) as missing values, retaining only the two clear categories of disagreement (0) and agreement (1).

```
#Recode the outcomes variables as a binary outcome
```

```
AUS_WVS <- AUS_WVS %>%
  mutate(Immi_Crime = ifelse(Q124 == 0, 0,
                                  ifelse(Q124 ==1, NA,
                                         ifelse(Q124 == 2, 1, NA)))) %>%
  mutate(Immi_Crime = factor(Immi_Crime,levels = c(0, 1), labels = c("Disagree",
"Agree")))
```

1.2 Exploring the Outcome Variable

Before proceeding with the regression analysis, we first explore the distribution of the binary outcome variable. You can check the frequency of each response using the `table()` function:

```
# Examine the original and recoded outcome variables
table(AUS_WVS$Q124)
```

```
##
##   0   1   2
## 415 810 571

table(AUS_WVS$Immi_Crime)

##
## Disagree    Agree
##      415      571
```

1.3 Visualising the Outcome Variables

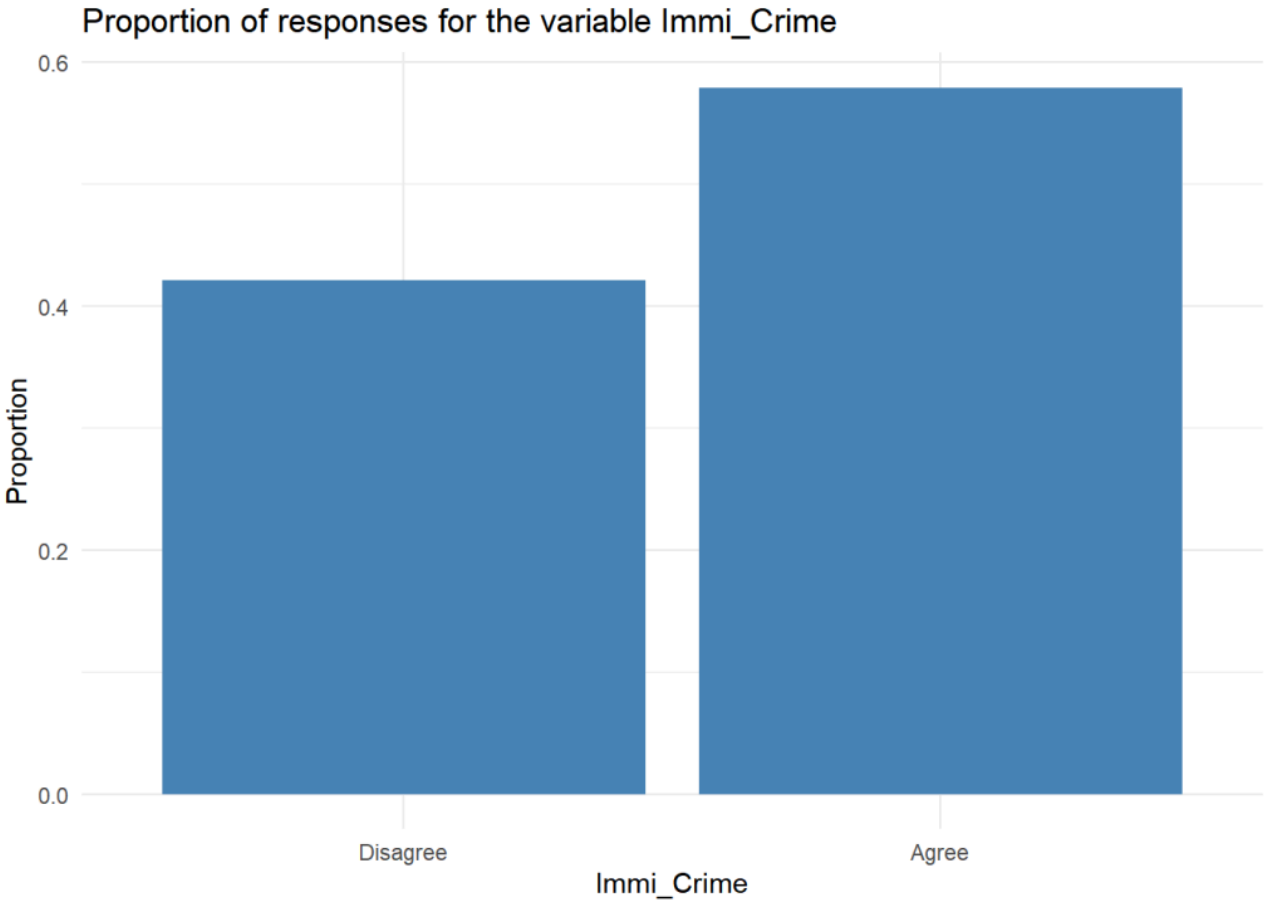
Next, we create bar plots to visualise the distribution of the newly recoded binary outcomes.

First, calculate the proportions:

```
#Calculate the proportion
Immi_Crime <- AUS_WVS %>%
  filter(!is.na(Immi_Crime)) %>%        # Exclude NA values
  count(Immi_Crime) %>%                 # Count non-missing values
  mutate(proportion = n / sum(n))       # Calculate proportion
```

Then, draw bar plots for the outcome variable:

```
#Draw a plot
ggplot(Immi_Crime) +
  geom_bar(aes(x=Immi_Crime, y=proportion), stat = "identity", fill = "steelblue") +
  labs(title = "Proportion of responses for the variable Immi_Crime",
       y = "Proportion") +
  theme_minimal()
```



Q. What was the most commonly selected response for this variable?

Write your response here:

1.4 Fitting a Logistic Regression Model

We now proceed to fit a logistic regression model. In R, logistic regression can be fitted easily using the `glm()` function. The process is similar to fitting a linear regression model with `lm()` function, except that we must specify the argument `family = "binomial"` to indicate that we are fitting a logistic model.

We first fit a model predicting perceptions that immigration increases crime. Running `summary(Crime)` will produce output similar to the table below:

```
#Fit a logistic model
Crime <- glm(Immi_Crime ~ EDU + Sex + Age , data = AUS_WVS, family = binomial)
summary(Crime)
```

```
##
## Call:
## glm(formula = Immi_Crime ~ EDU + Sex + Age, family = binomial,
##     data = AUS_WVS)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0800  -1.0529   0.6900   0.9194   1.7846
##
## Coefficients:
##              Estimate Std. Error z value             Pr(>|z|)
## (Intercept)  3.298098   0.391129   8.432 < 0.0000000000000002 ***
## EDU         -0.470049   0.045137 -10.414 < 0.0000000000000002 ***
## SexFemale   -0.291994   0.145128  -2.012              0.04422 *
## Age         -0.011572   0.004333  -2.671              0.00756 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1284.5  on 940  degrees of freedom
## Residual deviance: 1158.2  on 937  degrees of freedom
##   (872 observations deleted due to missingness)
## AIC: 1166.2
##
```

```
## Number of Fisher Scoring iterations: 4
```

The table presents the estimated coefficients and associated statistics for the logistic regression model predicting the probability that a respondent agrees immigration increases crime.

It is important to remember that logistic regression coefficients are expressed on the **log-odds scale**. For example, the estimated coefficient for Age is b_1=−0.012, indicating that a one-year increase in age is associated with a 0.012 decrease in the **log-odds of agreeing that immigration increases crime**, holding sex and education constant.

1.5 Odds Ratios

What does **the estimated log-odds of agreeing that immigration increases crime actually mean**? Because interpreting effects on the log-odds scale can be unintuitive, it is common to transform logistic regression coefficients into odds ratios for easier interpretation.

An **odds ratio** (OR) is a measure of association between an exposure and an outcome. It represents the odds that an outcome will occur given a particular exposure, compared to the odds of the outcome occurring without that exposure. In simpler terms, odds reflect the likelihood that an event will occur and can be understood as the ratio of the probability of the event happening to the probability that it does not (e.g., the odds of agreeing versus disagreeing).

In the context of logistic regression, the estimated regression coefficient (b_1) represents the change in the log-odds of the outcome associated with a one-unit increase in the predictor. Taking the exponential of the coefficient, e^{b_1}, gives the odds ratio associated with a one-unit increase in the explanatory variable.

Thus, odds ratios indicate how the **likelihood** of agreeing that immigration increases crime, compared to disagreeing, varies across different values of predictor variables such as age, sex, and education.

Interpretation of the odds ratio follows:

- OR = 1: Exposure has no effect on the odds of the outcome.
- OR > 1: Exposure is associated with higher odds of the outcome.
- OR < 1: Exposure is associated with lower odds of the outcome.

You can also calculate the percent change in odds using the formula:

$$(OR - 1) * 100.$$

To obtain the odds ratios from a logistic regression model, you can simply exponentiate (`exp()`) the estimated log-odds coefficients:

```
# Get odds ratios for model coefficients
exp(coef(Crime))
```

```
## (Intercept)         EDU   SexFemale         Age
##  27.0611225   0.6249714   0.7467727   0.9884943
```

This output presents the odds ratios for each predictor in the model. Let's practise interpreting each variable. There are two common ways to do this:

Option 1:

- Age (0.99): A one-year increase in age is associated with **0.99 times** lower odds of agreeing that immigration increases crime, holding sex and education constant.
- Female (0.75): Being female compared to men is associated with **0.75 times** lower odds of agreeing that immigration increases crime, holding sex and education constant.
- Education (0.62): A one-unit increase in education level is associated with **0.62 times** lower odds of agreeing that immigration increases crime, holding sex and education constant.

Option 2:

- Age (0.99): For each additional year of age, the odds of agreeing that immigration increases crime decrease by about **1%**, holding sex and education constant.
- Female (0.75): Female respondents, the odds of agreeing that immigration increases crime decreases by **25%** compared to male respondents, holding age and education constant.
- Education (0.62): For each additional level of education attainment, the odds of agreeing that immigration increases crime decrease by about **38%**, holding age and education constant.

1.6 Goodness of fit

The regression summary presents an Akaike Information Criterion (AIC) value instead of an R-squared value. AIC is commonly used to compare statistical models, particularly to determine which model best balances goodness of fit and model complexity. Specifically, AIC penalises models with more parameters, helping to avoid over-fitting. A lower AIC indicates a better fitting model relative to others being compared.

In addition, since traditional R-squared is not applicable to logistic regression, alternative measures known as **pseudo R-squared** values are used to provide a rough indication of model fit. There are several types of pseudo R-squared, including McFadden's and Tjur's. In a later section, we will examine Tjur's pseudo R-squared, which is suited for models with a binary outcome variable.

1.7 Predicted Probability

While odds ratios are more intuitive than log-odds, they can still be challenging to interpret. For clearer interpretation, we can use **predicted probabilities** instead. The process is similar to what we used when calculating predicted means in linear models.

We will first focus on predicted probabilities of the outcome variable by *Sex*, while holding the other explanatory variables (Age and Education) constant at their mean values.

To do this, we use the `ggpredict()` function:

```
# Predict probabilities for "Sex", holding other variables at their means
pred_sex <- ggpredict(Crime, terms = "Sex")
pred_sex
```

```
pred_sex$predicted    # predicted probabilities
```

```
## [1] 0.6859858 0.6199708
```

```
pred_sex$conf.low     # lower CI
```

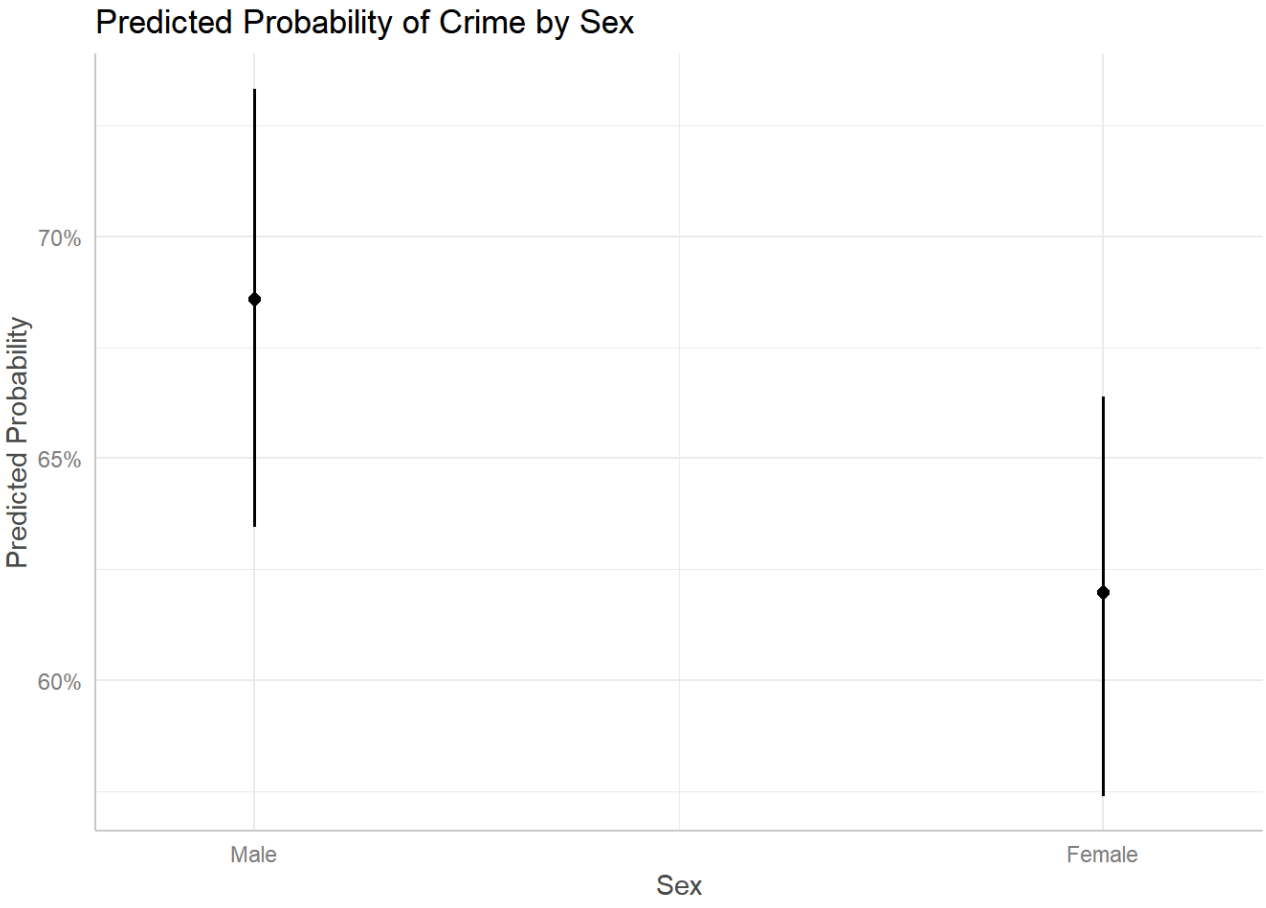
```
## [1] 0.6345951 0.5740910
```

```
pred_sex$conf.high    # upper CI
```

```
## [1] 0.7331877 0.6638026
```

We can visualise the predicted probabilities using the `plot()` function:

```
# Plot
plot(pred_sex) +
  labs(title = "Predicted Probability of Crime by Sex",
       y = "Predicted Probability",
       x = "Sex")
```

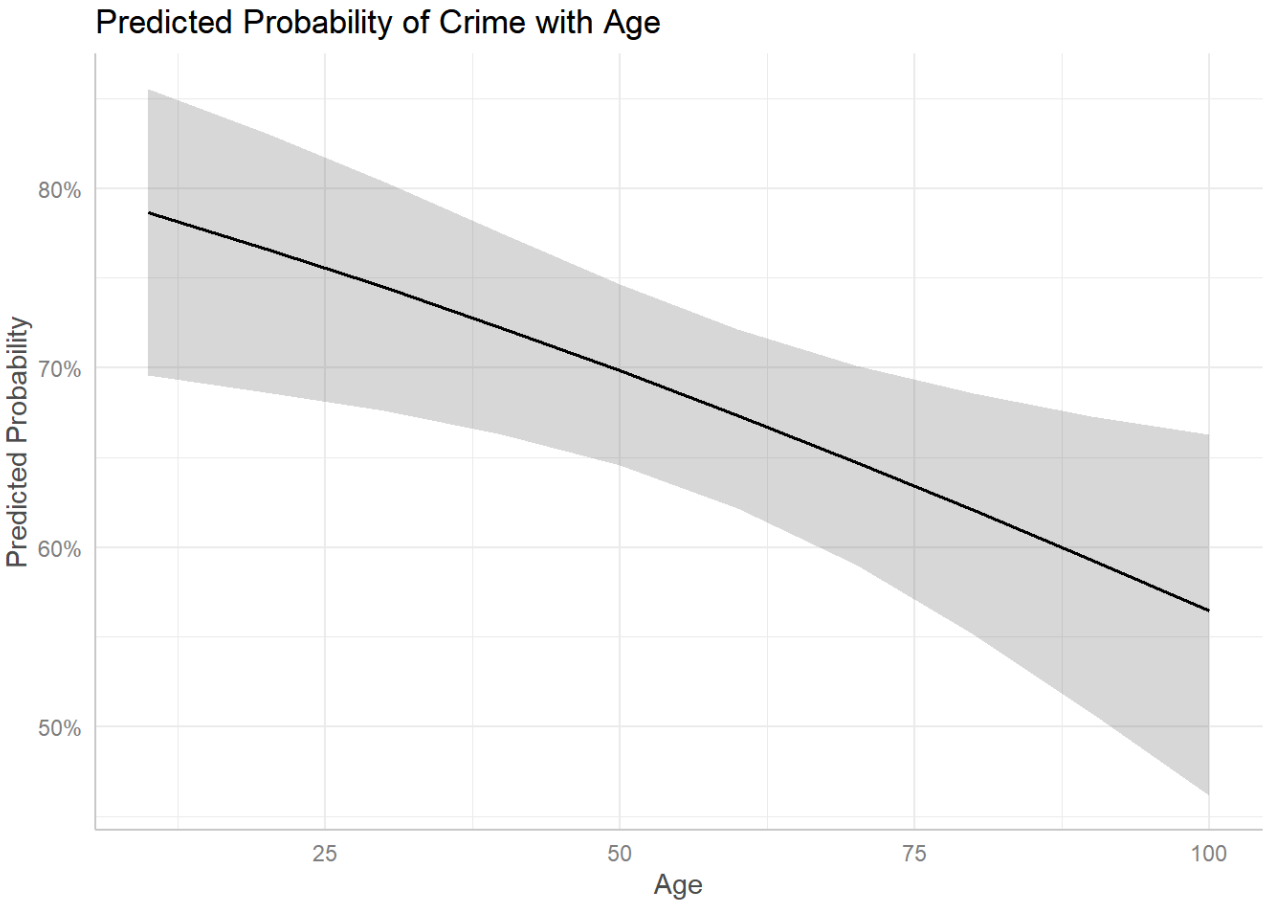


Q. Based on the output above, what are your conclusions? How is sex related to the probability of agreeing that immigration increases crime?

Write your response here:

You can also explore how the predicted probability of agreeing that immigration increases crime varies with *age*:

```
pred_age <- ggpredict(Crime, terms = "Age")
plot(pred_age) +
  labs(title = "Predicted Probability of Crime with Age",
       x = "Age", y = "Predicted Probability")
```

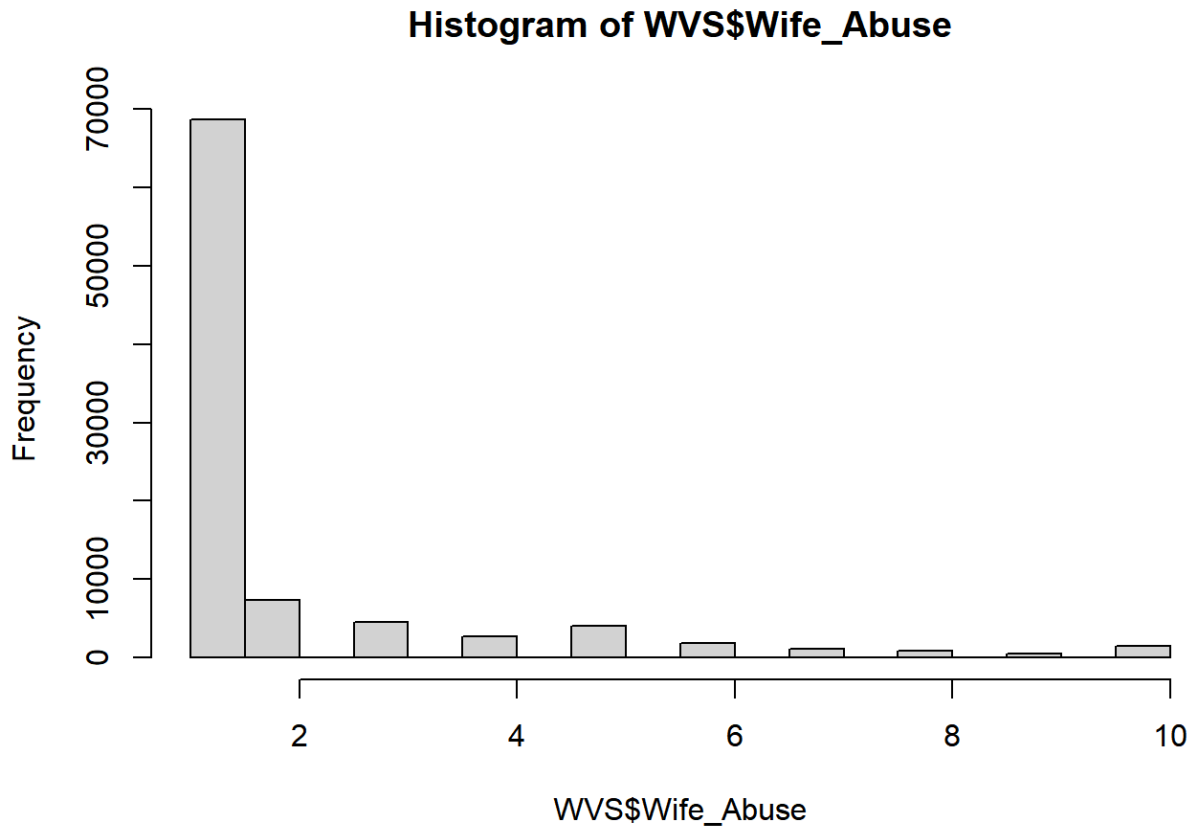


Q. Based on the output above, what are your conclusions? How is age related to the probability of agreeing that immigration increases crime?

Write your response here:

2. Logistic Regression – Recording a Numeric variable as a Binary Variable

Sometimes, a non-binary variable can be recoded into a binary format to fit a logistic regression model. For example, consider the variable *Justifiable: For a man to beat his wife* (hereafter referred to as *Wife_Abuse*). As shown in the figure below, the distribution of responses is right-skewed, with the tail extending to the right. Due to the lack of variation, a linear regression model would not be appropriate. Instead, we can convert the variable into a binary outcome and fit a logistic regression model.



Let's now recode the *Wife_Abuse* variable into a binary format (**0: Never justify vs 1: Somewhat justify**) for use as the outcome variable in our logistic regression model.

2.1 Load the Dataset and View Summary Statistics

We begin by recoding the original Wife_Abuse variable into a new binary variable, *DWife_Abuse*:

```
#Load the dataset
WVS <- WVS %>%
  mutate(DWife_Abuse = case_when(
    Wife_Abuse == 1 ~ 0,
    Wife_Abuse >= 2 & Wife_Abuse <= 10 ~ 1,
    TRUE ~ NA_real_
  )) %>%
  mutate(Sex = factor(Sex,levels = c(1, 2), labels = c("Male", "Female")))
```

After recoding, we can confirm that the new variable has been created correctly using the `table()` function:

```
table(WVS$Wife_Abuse)
```

```
##
##     1     2     3     4     5     6     7     8     9    10
## 68628  7383  4583  2748  4070  1815  1110   826   499  1482
```

```
table(WVS$DWife_Abuse)
```

```
##
##     0     1
## 68628 24516
```

Q. What was the most commonly selected response for each variable?

Write your response here:

2.2 Fitting a Logistic Regression Model

We now fit a logistic regression model predicting justifiability of wife abuse based on education, sex, and age:

```
# Fit a logistic regression model
Wife_Abuse <- glm(DWife_Abuse ~ EDU + Sex + Age , data = WVS, family = binomial)
# View the model summary
summary(Wife_Abuse)
```

```
##
## Call:
## glm(formula = DWife_Abuse ~ EDU + Sex + Age, family = binomial,
##     data = WVS)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0489  -0.8094  -0.7194   1.4169   2.0708
##
## Coefficients:
##               Estimate Std. Error z value             Pr(>|z|)
## (Intercept) -0.1154880  0.0275056  -4.199            0.0000268 ***
## EDU         -0.0715070  0.0038294 -18.673 < 0.0000000000000002 ***
## SexFemale   -0.3706489  0.0151239 -24.508 < 0.0000000000000002 ***
## Age         -0.0114444  0.0004728 -24.206 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 106010  on 92142  degrees of freedom
## Residual deviance: 104607  on 92139  degrees of freedom
```

```
##   (5077 observations deleted due to missingness)
## AIC: 104615
##
## Number of Fisher Scoring iterations: 4
```

You can also present the results in a formatted regression table displaying odds ratios using the `table_model()` function. You can either manually compute the odds ratios and 95% confidence intervals by exponentiating the coefficients (**Exercise 1**), or use the `table_model()` function.

```
tab_model(Wife_Abuse,
          p.style = "stars",
          dv.labels = "Justifiability of Wife Abuse",
          title = "Logistic Regression Outcome")
```

```
## Profiled confidence intervals may take longer time to compute.
##   Use `ci_method="wald"` for faster computation of CIs.
```

Logistic Regression Outcome

Justifiability of Wife Abuse

Predictors	Odds Ratios	CI
(Intercept)	0.89 ***	0.84 – 0.94
EDU	0.93 ***	0.92 – 0.94
Sex [Female]	0.69 ***	0.67 – 0.71
Age	0.99 ***	0.99 – 0.99
Observations	92143	
R2 Tjur	0.015	

- p<0.05 ** p<0.01 *** p<0.001

Q. Provide a full interpretation of the Justifiability of Wife Abuse model results using odds ratios

(refer to Option 2)

Write your response here:

2.3 Predicted Probability

Next, we explore the predicted probabilities for the discrete variable Sex, holding the other explanatory variables (Age and Education) constant at their mean values by using the `ggpredict()` function.

```
# Predict probabilities for "Sex"
pred_WA_sex <- ggpredict(Wife_Abuse, terms = "Sex")


# View predicted probabilities and confidence intervals
pred_WA_sex
```

```
pred_WA_sex$predicted    # predicted probabilities
```

```
## [1] 0.3101904 0.2368770
```

```
pred_WA_sex$conf.low     # Lower 95% CI
```

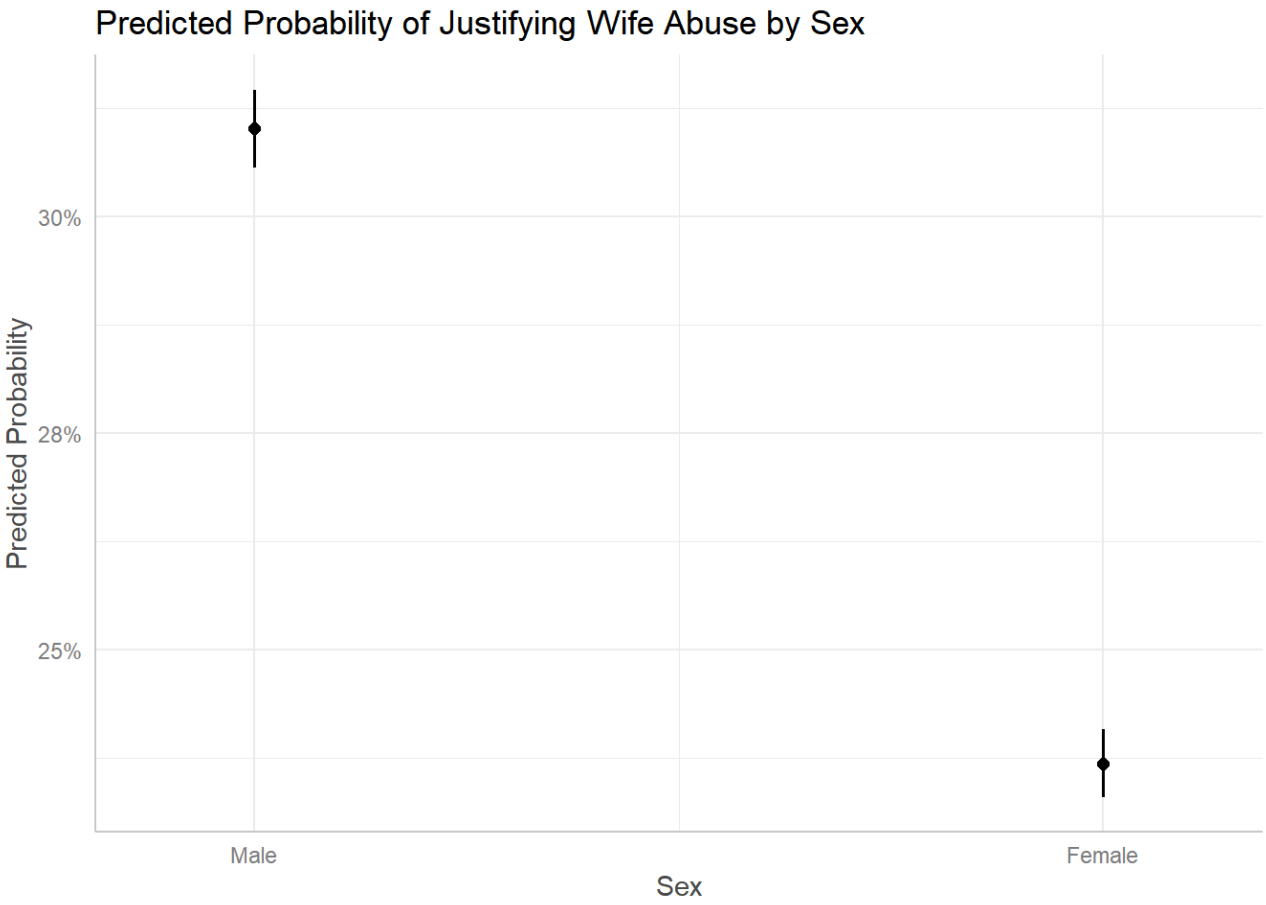
```
## [1] 0.3057183 0.2330209
```

```
pred_WA_sex$conf.high    # Upper 95% CI
```

```
## [1] 0.3146982 0.2407769
```

We can then visualise the predicted probabilities:

```
plot(pred_WA_sex) +
  labs(title = "Predicted Probability of Justifying Wife Abuse by Sex",
       y = "Predicted Probability",
       x = "Sex")
```

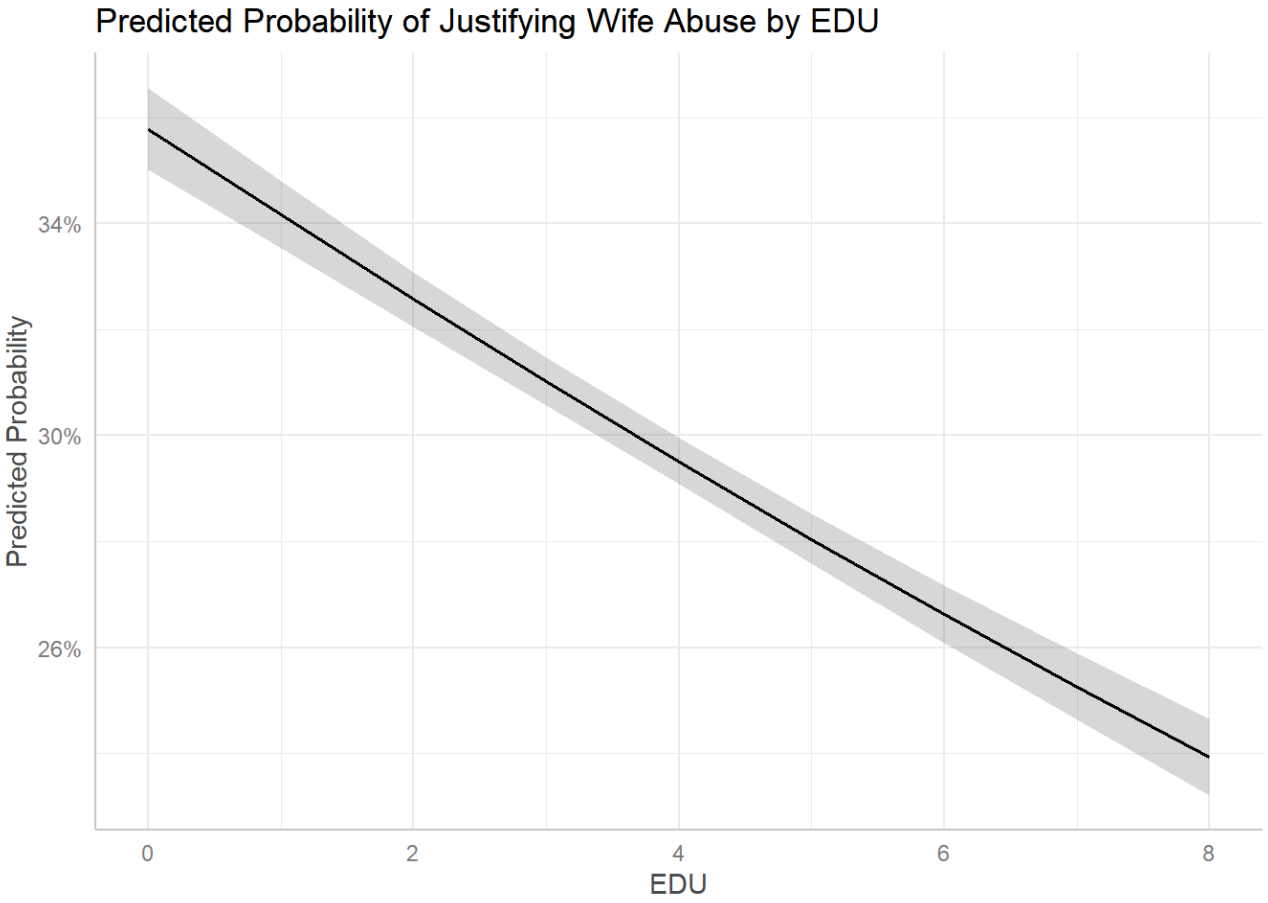


Q. Based on the output above, what conclusions can you draw? How is sex related to the probability of justifying wife abuse?

Write your response here:

You can also explore how the predicted probability of justifying wife abuse by *EDU*:

```
pred_WA_edu <- ggpredict(Wife_Abuse, terms = "EDU")
plot(pred_WA_edu) +
  labs(title = "Predicted Probability of Justifying Wife Abuse by EDU",
       x = "EDU", y = "Predicted Probability")
```



Q. Based on the output above, what are your conclusions? How is education level related to the probability of justifying wife abuse?

Write your response here:

After completing all exercises, make sure to knit your R Markdown file.

12.

BEYOND LINEAR MODEL: FITTING ORDINAL AND MULTINOMIAL LOGISTIC REGRESSION MODEL

In this chapter, we will explore regression models that capture non-linear relationships between variables. Our focus will be on polynomial models, especially those with quadratic terms, which help us model curved relationships between a continuous predictor and an outcome variable. We will then practise visualising quadratic effects, calculate turning points, and interpret the results.

1. Ordinal Logistic Regression

An ordinal variable is a type of categorical variable in which the response categories have a natural and meaningful order (e.g., Low, Medium, and High, or Disagree, Hard to say, and Agree). To model the relationship between predictor variables and an ordinal outcome, the `polr()` function is commonly used to estimate ordinal logistic regression models in R.

> We will call the function using `MASS::polr()` rather than loading the *MASS* package, as attaching the package can lead to conflicts with other libraries, particularly by masking the `select()` function from the *dplyr* package.

In this session, we will examine cross-national variation in agreement with the statement: **'Immigration increases unemployment'**. Specifically, we will compare responses from four countries: Australia, Canada, New Zealand, and the United States. The model will also include a set of demographic control variables (e.g., age, sex, and education level).

1.1 Data Loading and Cleaning

We begin by importing the dataset and preparing the necessary the variables for analysis.

```
setwd("C:/Your Own Path/SOCYR")


#Load the dataset
WVS <- read.csv("WVS.csv")
```

```
#Load the required packages for Chapter 12
library(ggplot2)
library(dplyr)
library(tidyverse)
library(gridExtra)
library(sjPlot)
library(ggeffects)
options(scipen = 999) #turning off scientific notation
```

```
#Data cleaning
EN_WVS <- WVS %>%
  filter(Country %in% c(36, 124, 554, 840))%>%  #36: Australia, 124: Canada, 554: New
Zealand, 840: USA
    mutate(Immi_Unemploy = factor(Q128,
                           levels = c(0,1,2), labels = c("Disagree", "Hard to say",
"Agree"))) %>%
    mutate(Sex = factor(Sex,levels = c(1, 2), labels = c("Male", "Female"))) %>%
    mutate(COUNTRY = factor(Country,
                           levels = c(36, 124, 554, 840),
                           labels = c("Australia", "Canada", "New Zealand", "United
States")))
```

To confirm that the outcome variable has been correctly re-labelled, we inspect the distribution of both the original and the recoded variables using the `table()` function:

```
table(EN_WVS$Q128)
```

```
##
##    0    1    2
## 3088 3843 2457
```

```
table(EN_WVS$Immi_Unemploy)
```

```
##
##    Disagree Hard to say       Agree
##        3088        3843        2457
```

1.2 Summary Statistics

Before conducting regression analysis, we should first review an informative overview of the data. Below, we present the relative frequencies of categorical variables and the mean values of continuous variables.

```
#Percentage of each categorical variable
prop.table(table(EN_WVS$COUNTRY))
```

```
##
##     Australia        Canada   New Zealand United States
##     0.1911641     0.4236609     0.1114509     0.2737242
```

```
prop.table(table(EN_WVS$Immi_Unemploy))
```

```
##
##    Disagree Hard to say       Agree
##   0.3289305   0.4093524   0.2617171
```

```
prop.table(table(EN_WVS$Sex))
```

```
##
##      Male    Female
## 0.4861861 0.5138139
```

```
# Mean values for each continuous predictor
EN_WVS %>%
  summarise(
    Mean_Age = mean(Age, na.rm = TRUE),
    Mean_EDU = mean(EDU, na.rm = TRUE)
  )
```

Q. Interpret the summary statistics output.

Write your response here:

1.3 Fitting an Ordinal Logistic Regression

We now proceed to fit the ordinal logistic regression model using the `MASS::polr()` function. You should include the argument `Hess = TRUE` to ensure that the variance-covariance matrix is returned, which is necessary for calculating standard errors and confidence intervals.

```
Ordinal_model <- MASS::polr(Immi_Unemploy ~ COUNTRY + Age + Sex + EDU,
                        data = EN_WVS,
                        Hess= TRUE)           # Hess = TRUE saves the variance-covariance
matrix
summary(Ordinal_model)
```

```
## Call:
## MASS::polr(formula = Immi_Unemploy ~ COUNTRY + Age + Sex + EDU,
```

```
##     data = EN_WVS, Hess = TRUE)
##
## Coefficients:
##                           Value Std. Error  t value
## COUNTRYCanada         -0.526494   0.055291  -9.5223
## COUNTRYNew Zealand    -0.545578   0.074689  -7.3047
## COUNTRYUnited States   0.223130   0.059628   3.7421
## Age                    0.001165   0.001184   0.9838
## SexFemale             -0.097822   0.039802  -2.4577
## EDU                   -0.211179   0.012286 -17.1890
##
## Intercepts:
##                       Value    Std. Error t value
## Disagree|Hard to say  -1.9719   0.1100   -17.9323
## Hard to say|Agree     -0.1327   0.1078    -1.2311
##
## Residual Deviance: 19376.90
## AIC: 19392.90
## (255 observations deleted due to missingness)
```

The outcome differs from that of linear or binary logistic regression models. The `polr()` function provides: **Coefficient estimates**, **Standard errors**, and **t-values**. However, it does not report p-values or 95% confidence intervals by default. We will demonstrate how to compute these values manually in a later section.

1.4 Interpretation of the Regression Table

In an ordinal logistic regression model, each coefficient represents the change in the **log-odds of being in a higher category of the outcome variable** (in this case, stronger agreement that immigration increases unemployment) for a one-unit increase in the predictor, holding all other variables constant.

For example, a one-year increase in age is associated with a 0.001 increase in the log-odds of being in a higher category of agreement that immigration increases unemployment. In contrast, Canada (relative to the reference category, Australia) is associated with a 0.53 decrease in the log-odds of being in a higher agreement category. This indicates that Canadian respondents are less likely than Australians to believe that immigration increases unemployment

Q. Interpret the coefficients for New Zealand, sex (Female), and education (EDU).

Write your response here:

1.5 Intercepts (Thresholds) in Ordinal Logistic Regression

Since the outcome variable consists of three ordered categories (Disagree, Hard to say, and Agree), the model includes two intercepts, also known as thresholds or cut points. These thresholds represent the boundaries on the latent scale that separate adjacent outcome categories.

For example:

- The first threshold (−1.97) defines the point at which the model distinguishes between Disagree versus the combined categories Hard to say and Agree.
- The second threshold (−0.13) defines the boundary between the combined categories Disagree and Hard to say versus Agree.

Ordinal logistic regression relies on the **proportional odds assumption**, which assumes each predictor has a consistent effect across all outcome thresholds. In other words, the relationship between the independent variables and the outcome is assumed to be consistent, regardless of which boundary is being evaluated. We will examine this assumption and how to test it further in Section 1.8.

1.6 Odds Ratios

As with binary logistic regression, odds ratios in ordered logistic regression can be obtained by exponentiating the log-odds coefficients. We begin by extracting the coefficient estimates and computing the odds ratios, 95% confidence intervals, and p-values.

*Note: To calculate 95% confidence intervals (CIs):

$$\mathrm{CI}^{b}_{0.95} = \left[\hat{b} - 1.96 \cdot \mathrm{SE}(\hat{b}),\ \hat{b} + 1.96 \cdot \mathrm{SE}(\hat{b})\right]$$

```
# Extract coefficients and standard errors
coefs <- summary(Ordinal_model)$coefficients

# Calculate odds ratios
OR <- exp(coefs[, "Value"])
Std_Error = exp(coefs[, "Std. Error"])



lower <- exp(coefs[, "Value"] - 1.96 * coefs[, "Std. Error"])
upper <- exp(coefs[, "Value"] + 1.96 * coefs[, "Std. Error"])

# Calculate the two-tailed p-values = 2 * (1 - CDF of normal distribution at |t-value|)
p_values <- 2 * (1 - pnorm(abs(coefs[, "t value"])))      #A two-tailed p-value from a z-
score.  #The pnorm() is   cumulative probability and abs()is an absolute value (x)
```

Once all values have been calculated, you can compile the results into a clean table using the `data.frame()` function:

```
# Compile results into a tidy data frame
Result <- data.frame(
  OR = round(OR, 3),                  #round() means show the decimal place to third
  Std_Error = round(Std_Error, 3),
  lower = round(lower, 3),
  upper = round(upper, 3),
  p_value = round(p_values, 3)
)
Result
```

The output includes odds ratios, 95% confidence intervals, and p-values for each predictor. The interpretation of odds ratios is consistent with logistic regression: values greater than 1 indicate increased odds of being in a higher outcome category, while values less than 1 indicate decreased odds. You can also express the percent change in odds using the formula: (OR – 1) * 100.

Country:

Canada (compared to Australia) is associated with a 41% decrease in the odds of expressing stronger agreement that immigration increases unemployment, holding other variables constant (OR = 0.59, $p < 0.05$).

New Zealand (compared to Australia) is associated with a 42% decrease in the odds of expressing stronger agreement that immigration increases unemployment, holding other variables constant (OR = 0.58, $p < 0.05$).

The United States (compared to Australia) is associated with a 25% increase in the odds of expressing stronger agreement that immigration increases unemployment, holding other variables constant (OR = 1.25, $p < 0.05$).

Education Level:

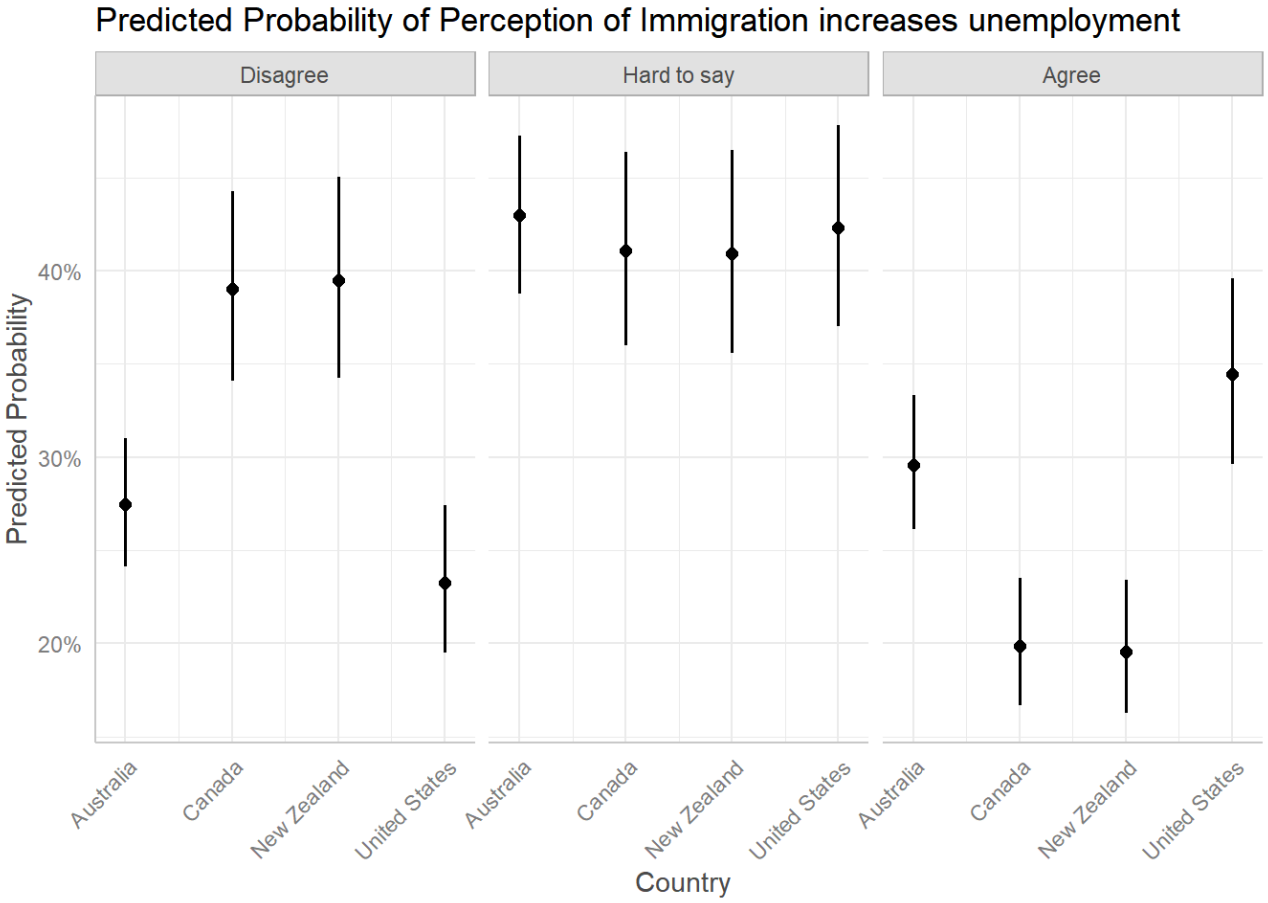
A higher level of education is associated with a 19% decrease in the odds of expressing stronger agreement that immigration increases unemployment, holding other variables constant (OR = 0.81, $p < 0.05$).

1.7 Predicted Probability

We now compute and visualise predicted probabilities for each response category across countries. This allows us to observe how the likelihood of agreement with the statement: **'Immigration increases unemployment'** varies by country.

We use the `ggpredict()` function to generate predicted probabilities, followed by the `plot()` function to visualise the results.

```
# Compute predicted probabilities by country
pred <- ggpredict(Ordinal_model, terms = "COUNTRY")


# Plot predicted probabilities
plot(pred) +
  labs(title = "Predicted Probability of Perception of Immigration increases unemployment",
       y = "Predicted Probability",
       x = "Country") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  #Rotate the x-axis labels 45 degrees
```



Q. Based on the output above, what conclusions can you draw? Do the patterns of perception differ across countries?

Write Your response here:

1.8 The Proportional Odds Assumption

The proportional odds assumption is a key assumption in ordinal logistic regression. It means that the effect of each predictor is assumed to be constant across all thresholds of the ordinal outcome.

In other words, the relationship between the predictors and the outcome is the same, whether you're comparing:

- Disagree vs. Hard to say and Agree, or
- Disagree and Hard to say vs. Agree.

If this assumption holds, you can use a single set of coefficients for all comparisons across the ordered outcome levels. If it is violated, you may need a more flexible model (e.g., partial proportional odds or multinomial regression).

To test the proportional odds assumption, we can use the **Brant** test, which is specifically designed to assess this assumption in models fitted with the `polr()` function.

```
install.packages("brant")  # Install the package if not already installed
library(brant)
```

```
brant(Ordinal_model)
```

```
## ------------------------------------------------------
## Test for     X2  df  probability
## ------------------------------------------------------
## Omnibus           59.98   6   0
## COUNTRYCanada         2.27    1   0.13
## COUNTRYNew Zealand   2.73    1   0.1
## COUNTRYUnited States 10.9    1   0
## Age           1.85    1   0.17
## SexFemale         18.66   1   0
## EDU           0.24    1   0.62
## ------------------------------------------------------
##
## H0: Parallel Regression Assumption holds
```

The Brant test evaluates whether the coefficients significantly differ across thresholds. A non-significant result ($p > .05$) suggests that the assumption holds for that variable. If the test yields significant results, the proportional odds assumption is violated for one or more predictors. If this assumption is violated, you can use a generalised ordinal logistic regression (also called a partial proportional odds model), which allows the effects of some predictors to vary across thresholds while keeping others constant.

2. Multinomial Logistic Regression Model

Multinomial logistic regression is an extension of binary logistic regression used when the outcome variable is categorical with more than two unordered categories. Unlike ordinal logistic regression, multinomial logistic regression assumes no inherent ordering or rank among outcome categories. Intuitively, this model

can be thought of as stacking multiple binary logistic regressions, each comparing one outcome category to a chosen reference category.

For this exercise, we use the variable *Q152: Aims of Country as the outcome*. This question asks respondents to identify the most important national priority for the next ten years. The response options are: **1 (A high level of economic growth)**, **2 (Strong defense forces)**, **3 (People have more say about how things are done)**, and **4 (Trying to make our cities and countryside more beautiful)**. Our analysis focuses only on **Australian** respondents and examines how education level (EDU) is associated with their top national priorities, while controlling for age and sex.

2.1 Data Cleaning

Before fitting the model, we first prepare the data. The outcome variable (*Aim*) is initially coded as numeric, but because it represents categorical, non-ordinal responses, we need to recode it as a factor and assign labels to each category.

```
#Data cleaning
AUS_WVS <- WVS %>%
  filter(Country == 36) %>%
  mutate(Sex = factor(Sex,levels = c(1, 2), labels = c("Male", "Female"))) %>%
  mutate(Aim = factor(Q152,
         levels = c(1,2,3,4), labels = c("A high level of economic growth", "Strong
defense forces",
                                "People have more say about how things are done", "Trying
to make our cities and countryside more beautiful")))
```

2.2 Summary Statistics

We begin by exploring the distribution of the outcome variable using the 'table()' and 'prop.table()' functions.

```
# Frequency and proportion of responses
table(AUS_WVS$Aim)
```

```
##
##                          A high level of economic growth
##                                                      912
```

```
##                                     Strong defense forces
##                                                       322
##            People have more say about how things are done
##                                                       477
## Trying to make our cities and  countryside more beautiful
##                                                        75

prop.table(table(AUS_WVS$Aim))

##
##                            A high level of economic growth
##                                                 0.51063830
##                                     Strong defense forces
##                                                 0.18029115
##            People have more say about how things are done
##                                                 0.26707727
## Trying to make our cities and  countryside more beautiful
##                                                 0.04199328
```

Q. What national priority was selected most by Australian respondents?

Write your response here:

2.3 Fitting a Multinominal Logistic Regression

Before fitting the multinomial logistic regression model, we should understand how the model differs from binary and ordinal logistic regression. In multinomial logistic regression, the outcome variable consists of more than two unordered categories, and the model estimates the log-odds of each non-reference category relative to a base (reference) category.

Rather than modeling a single log-odds outcome, the model estimates the relative log-odds—that is, the

natural logarithm of the probability of choosing a **specific category** over the probability of choosing the **base category**.

These log-odds are interpreted similarly to those in binary logistic regression, but each category is compared to the **same reference point**. We now proceed to fit the multinominal logistic regression model using the `multinom()` function from the *nnet* package.

```
install.packages("nnet")  # Install the package if not already installed
library(nnet)
```

```
# Fit the multinomial logistic regression model
Multinom_model <- multinom(Aim ~ EDU + Age + Sex  , data = AUS_WVS)
```

```
## # weights:  20 (12 variable)
## initial  value 2374.722241
## iter  10 value 1888.841361
## final  value 1861.959814
## converged
```

```
# View model summary
summary<- summary(Multinom_model)


# Compute p-values from z-statistics
z <- summary$coefficients / summary$standard.errors
p <- 2 * (1 - pnorm(abs(z)))
```

The summary(Multinom_model) output provides coefficient estimates and standard errors for each non-reference category of the outcome variable. The reference category in this case is *A high level of economic growth*, which is omitted from the output. All reported coefficients represent the **relative log-odds** of selecting a given category compared to this baseline reference.

For example, a one-level increase in education is associated with a 0.29 decrease in the **relative log-odds** of selecting Strong defense forces over A high level of economic growth. This means that individuals with higher education levels are less likely to prioritise strong defense forces relative to economic growth.

2.4 Relative Risk Ratios (RRR)

```
install.packages("broom")  # Install the package if not already installed
library(broom) # for using the tidy() function
```

In multinomial logistic regression, the **relative risk ratio** (RRR) represents the ratio of the probability of selecting a given outcome category relative to the baseline reference, for a one-unit change in the predictor variable.

Although the term relative risk is sometimes used interchangeably with odds, in this context it reflects the change in relative risk of choosing one category over the reference.

You can compute the RRR using the `tidy()` function from the *broom* package. You should also include the argument `exponentiate = TRUE`.

```
# Calculate Relative Risk Ratios
tidy(Multinom_model, conf.int = TRUE, exponentiate = TRUE) %>%
  mutate(p.value = as.vector(p)) %>%
  mutate(across(where(is.numeric), ~ round(.x, 3))) # Round all numeric columns to 3 deci-
mal places.
```

The output includes RRRs, confidence intervals, and p-values for each non-reference category relative to the baseline (A high level of economic growth).

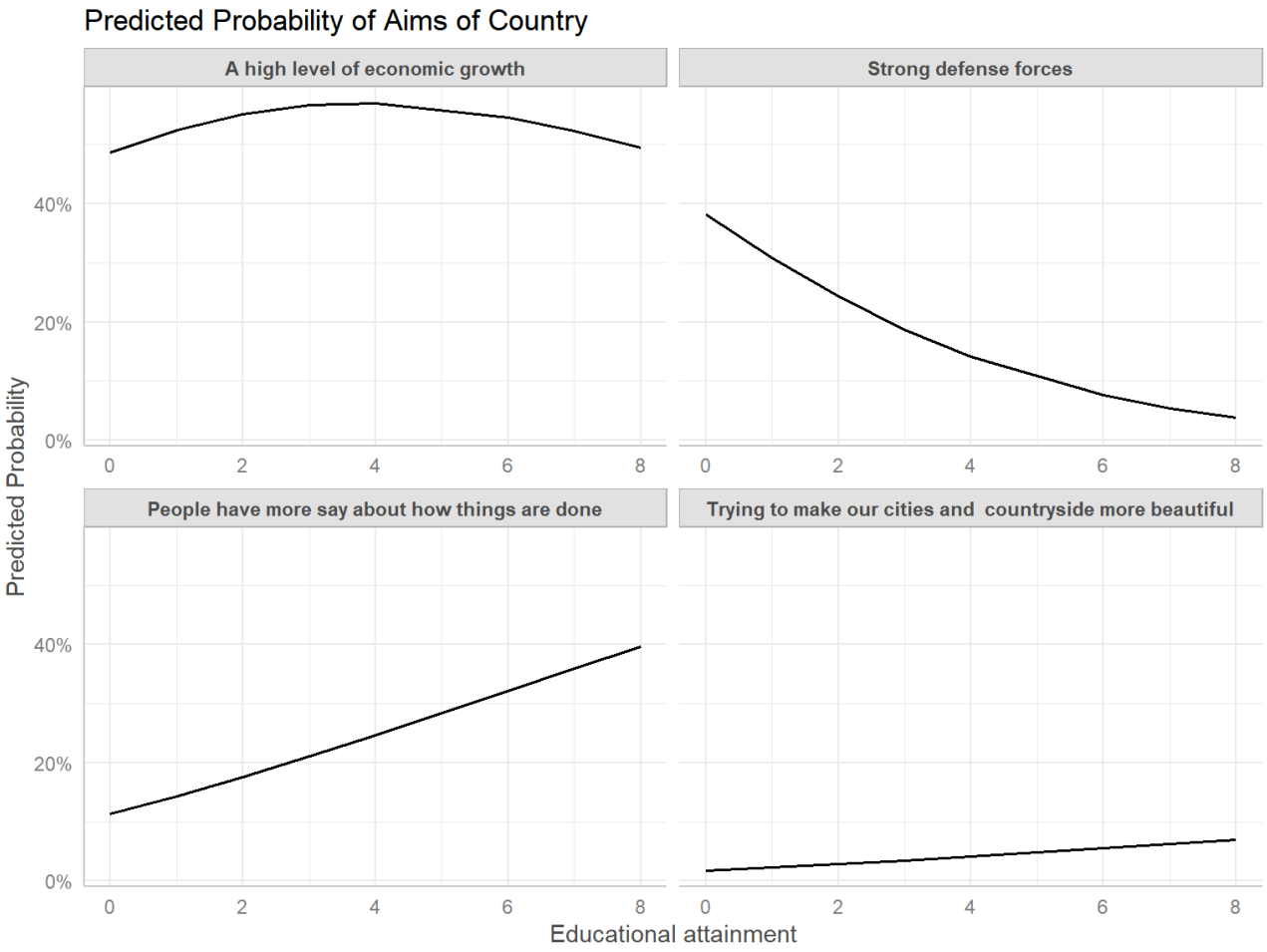
For interpretation, you can express the percent change in relative risk using the formula: (RRR−1)*100

- Strong defense forces: Higher educational level is associated with a 25% decrease in the relative risk of selecting Strong defense forces as the top national priority, compared to A high level of economic growth, holding other variables constant (RRR = 0.75, $p < 0.05$).
- People have more say about how things are done: Higher education level is associated with a 17% increase in the relative risk of selecting people have more say about how things are done over A high level of economic growth, holding other variables constant (RRR = 1.17, $p < 0.05$).
- Trying to make our cities and countryside more beautiful: higher education level is associated with an 18% increase in the relative risk of selecting this option compared to A high level of economic growth, holding other variables constant (RRR = 1.18, $p < 0.05$).

2.5 Predicted Probability

We now calculate and visualise predicted probabilities across levels of educational level. This allows us to examine how the likelihood of selecting each national priority varies by education level, while holding other variables constant.

```
# Compute predicted probabilities by education level
pred <- ggpredict(Multinom_model, terms = "EDU")

# Plot predicted probabilities
plot(pred) +
  labs(title = "Predicted Probability of Aims of Country",
       y = "Predicted Probability",
       x = "Educational attainment") +
  theme(strip.text = element_text(face = "bold"))             #Make each outcome category
label bold
```



Q. Based on the output above, what conclusions can you draw? Do patterns in national priorities differ by educational attainment?

Write your response here:

2.6 Independence of Irrelevant Alternatives (IIA) Assumption

One of key assumptions in multinomial logistic regression is the **Independence of Irrelevant Alternatives** (IIA) assumption. This assumption holds that the relative odds of choosing one category over another are independent of the presence or absence of other categories. In other words, the addition or removal of an alternative category should not affect the odds between any two existing categories.

The IIA assumption can be tested using the **Hausman-McFadden** test. However, this test cannot be directly applied to models estimated using the nnet package. To test the IIA assumption, you can re-estimate your model using the mlogit package. For further details and examples, refer to the mlolgit package documentation (PDF, 175KB).

Final Note

This concludes the final chapter of the project. We hope it has strengthened your ability to think critically about social science issues through a quantitative lens and inspired you to apply these tools in your future studies and research.

Appendix

A1. A list of required R packages

```
install.packages("tidyverse")
install.packages("dplyr")
install.packages("readxl")
install.packages("writexl")
install.packages("corrplot")
install.packages("tidyverse")
install.packages("rmarkdown")
install.packages("ggplot2")
install.packages("patchwork")
install.packages("stargazer")
```

```
install.packages("sjplot")
install.packages("marginaleffects")
install.packages("bda")
install.packages("gridExtra")
install.packages("ozmaps")
install.packages("brant")
install.packages("nnet")
install.packages("ggeffects")
install.packages("broom")
```

A2. Code Demonstration of Inner, Left, Right, and Full Joins (Chapter 4)

```
# Creating sample WA dataset
WA_df <- data.frame(
  D_INTERVIEW = c(36070000, 36070001, 36070002, 36070004, 36070005,
                  36070006, 36070010, 36070011, 36070012, 36070013),
  Wife_Abuse = c(1, 5, 1, 1, 1, 1, 1, 1, NA, 1),
  avg_Wife_Abuse = rep(1.24, 10)
)

# Creating sample VO dataset
VO_df <- data.frame(
  D_INTERVIEW = c(36070000, 36070001, 36070002, 36070003, 36070004,
                  36070005, 36070006, 36070007, 36070008, 36070009),
  Violence_otherPPL = c(10, 6, 1, 10, 3, 1, 1, 2, 1, 1),
  avg_Violence_otherPPL = rep(1.60, 10),
  total_Violence_otherPPL = rep(1.62, 10)
)

# Inner Join (only matching D_INTERVIEW values)
inner_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Left Join (all VO dataset values, matching WA dataset where possible)
left_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")
```

```
# Right Join (all WA dataset values, matching VO dataset where possible)
right_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Full Join (all values from both datasets)
full_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Display datasets
print(inner_join)
print(left_join)
print(right_join)
print(full_join)
```

REFERENCES

Agresti, A. (2003). *Categorical data analysis* (2nd ed.). Wiley.

Arel-Bundock, V., Greifer, N., & Heiss, A. (2024). How to interpret statistical models using marginaleffects for R and Python. *Journal of Statistical Software*, *111*(9),1-32. https://doi.org/10.18637/jss.v111.i09

Australian Government Centre for Population. (n.d.). *Data and forecasts*. https://population.gov.au/data-and-forecasts/

Australian Government Department of Education. (n.d.). *International education data and research*. https://www.education.gov.au/international-education-data-and-research

Baron, R. M., & Kenny, D. A. (1986). The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, *51*(6), 1173–1182. https://doi.org/10.1037/0022-3514.51.6.1173

Brant, R. (1990). Assessing proportionality in the Proportional Odds Model for Ordinal Logistic Regression. *Biometrics*, *46*(4), 1171–1178. https://doi.org/10.2307/2532457

Centre for Population. (n.d.). *Data and forecasts. Australian Government*. Retrieved March 11, 2025, from https://population.gov.au/data-and-forecasts/

Croissant, Y. (2022). *mlogit: Multinomial logit models*. https://cran.r-project.org/web/packages/mlogit/mlogit.pdf

Dauber, D. (2025). *R for non-programmers: A guide for social scientists*. Chapman & Hall/CRC.

Grolemund, G. (2014). *Hands-on programming with R*. O'Reilly Media.

Hair, J.F., Hult, G.T.M., Ringle, C.M., Sarstedt, M., Danks, N.P. & Ray, S. (2021). Moderation analysis. In *Partial least squares structural equation modeling (PLS-SEM) using R*. Classroom Companion: Business. Springer. https://doi.org/10.1007/978-3-030-80519-7_8

Hausman, J., & McFadden, D. (1984). Specification tests for the multinomial logit model. *Econometrica: Journal of the Econometric Society*, *52*(5), 1219-1240. https://doi.org/10.2307/1910997

Irizarry, R. (2019). *Introduction to data science: Data analysis and prediction algorithms with R*. Chapman & Hall/CRC.

MacKinnon, D. P., Fairchild, A. J., & Fritz, M. S. (2007). Mediation analysis. *Annual Review of Psychology*, *58*(1), 593–614. https://doi.org/10.1146/annurev.psych.58.110405.085542

Müller, K., & Wickham, H. (2021). *Column data types*. https://tibble.tidyverse.org/articles/types.html

Nahhas, R. W. (2024). *Introduction to regression methods forpublic health using R*. Chapman & Hall/CRC.

R Graph Gallery. (n.d.). *ggplot2*. https://r-graph-gallery.com/ggplot2-package.html

RStudio. (n.d.). *Data visualization with ggplot2: Cheat sheet*. https://rstudio.github.io/cheatsheets/data-visualization.pdf

Small, K. A., & Hsiao, C. (1985). Multinomial Logit Specification Tests. *International Economic Review*, *26*(3), 619.

Szumilas, M. (2010). Explaining odds ratios. *Journal of the Canadian Academy of Child and Adolescent Psychiatry*, *19*(3), 227-229.

Tidyverse. (n.d.). *Tidyverse packages*. https://www.tidyverse.org/packages/

Zhang, Z., & Wang, L. (2017). *Advanced statistics using R*. ISDSA Press.

APPENDIX

A1. A list of required R packages

```
install.packages("tidyverse")
install.packages("dplyr")
install.packages("readxl")
install.packages("writexl")
install.packages("corrplot")
install.packages("tidyverse")
install.packages("rmarkdown")
install.packages("ggplot2")
install.packages("patchwork")
install.packages("stargazer")
install.packages("sjplot")
install.packages("marginaleffects")
install.packages("bda")
install.packages("gridExtra")
install.packages("ozmaps")
install.packages("brant")
install.packages("nnet")
install.packages("ggeffects")
install.packages("broom")
```

A2. Code Demonstration of Inner, Left, Right, and Full Joins (Chapter 4)

```
# Creating sample WA dataset
WA_df <- data.frame(
  D_INTERVIEW = c(36070000, 36070001, 36070002, 36070004, 36070005,
                  36070006, 36070010, 36070011, 36070012, 36070013),
  Wife_Abuse = c(1, 5, 1, 1, 1, 1, 1, 1, NA, 1),
  avg_Wife_Abuse = rep(1.24, 10)
```

```
)

# Creating sample VO dataset
VO_df <- data.frame(
  D_INTERVIEW = c(36070000, 36070001, 36070002, 36070003, 36070004,
                  36070005, 36070006, 36070007, 36070008, 36070009),
  Violence_otherPPL = c(10, 6, 1, 10, 3, 1, 1, 2, 1, 1),
  avg_Violence_otherPPL = rep(1.60, 10),
  total_Violence_otherPPL = rep(1.62, 10)
)

# Inner Join (only matching D_INTERVIEW values)
inner_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Left Join (all VO dataset values, matching WA dataset where possible)
left_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Right Join (all WA dataset values, matching VO dataset where possible)
right_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Full Join (all values from both datasets)
full_join <- merge(VO_df, WA_df, by = "D_INTERVIEW")

# Display datasets
print(inner_join)
print(left_join)
print(right_join)
print(full_join)
```

GLOSSARY

Data frame

A two-dimensional table-like structure where each column can contain different data types, commonly used for datasets.

Element

A single item or value within a vector, list, matrix, or other data structure.

Function

A reusable block of code that performs a specific task when called, often taking inputs (arguments) and returning outputs.

Library

A collection of R functions, data, and documentation packaged together; you load it using 'library(package name)'.

List

A flexible data structure in R that can contain elements of different types and lengths, including other lists.

Matrix

A two-dimensional structure of data elements of the same type, arranged in rows and columns.

Tibble

A modern version of a data frame from the tibble package that prints more cleanly and handles data types consistently.

Vector

A one-dimensional sequence of data elements of the same type (e.g., numeric, character, logical).

OPEN TEXTBOOKS @ UQ

A portable workbook for data analysis: R for the social sciences is made possible by the Library's *Open Textbooks @ UQ* program. The Library supports the adoption or creation of open educational resources, such as this open textbook, that are free for everyone. Open textbooks provide an alternative to commercial textbooks and benefit you, your students and the community. Find out how you can author, adapt or adopt open textbooks.

Your feedback is always welcome

Contact Pressbooks@library.uq.edu.au if you have questions or feedback about *Open Textbooks @ UQ*.