Lab 2 | Data Wrangling in R

ST 437 Data Visualization

Student Name

August 26, 2024

Learning Objectives

1. **Understand the Importance of Data Wrangling**: Recognize the critical role of data wrangling in preparing datasets for accurate and meaningful analysis.
2. **Master Data Transformation Techniques**: Develop the ability to convert raw, untidy data into a structured format suitable for analysis and visualization.
3. **Emphasize Data Cleaning and Preparation**: Learn to identify and correct common data issues such as missing values, inconsistent formats, and incorrect entries.
4. **Ensure Data Accuracy and Consistency**: Gain skills in validating and maintaining the integrity of data throughout the wrangling process.
5. **Draw Reliable Conclusions**: Build confidence in making well-supported decisions and insights based on meticulously prepared and clean data.

What is Data Wrangling?

Data wrangling is the process of converting messy, untidy data into a tidy format, making it suitable for data visualization and analysis.

- **Data is often messy:** Real-world data is rarely provided in a tidy format.
- **Industry challenges:** Many industries have poorly designed data structures, requiring data preparation before visualization.
- **Rarely tidy datasets:** It is uncommon to receive a dataset that is already tidy.

What is Tidy Data?

Tidy data is a structured format that aligns the organization of a dataset with its underlying meaning. In tidy data:

- **Each variable has its own column:** Every column in the dataset corresponds to a specific variable or attribute.
- **Each observation has its own row:** Every row captures a single observation or data entry.
- **Each cell contains a single value:** Each cell holds one distinct piece of information for a particular variable and observation.

In most cases, data is often imported using SQL to create narrower datasets. While we won't cover SQL in this course, it's a valuable skill to learn in the future. For now, we'll focus on using R to manipulate and create subset datasets from larger datasets for focused analysis.

What Causes Untidy Data?

- **Incorrect/Inconsistent Dates:** Dates can be tricky because they might be formatted differently across datasets or have errors like typos or missing parts. For example, some data might use "MM/DD/YYYY" while others use "YYYY-MM-DD," leading to confusion and potential errors when analyzing time-based data.
- **Wide Format Times:** Time data is sometimes presented in a wide format, where each column represents a different time period. This structure can make it difficult to perform certain types of analysis, as many statistical and visualization tools prefer data in a long format, where each row represents a single observation at a specific time.
- **Void or Misspelled Descriptions:** Descriptions and labels are often incomplete, missing, or contain typos. These errors can make it challenging to interpret the data correctly, especially when variables are not clearly defined or are inconsistent across different parts of the dataset.
- **Missing Values:** Missing data is common and can occur in any part of a dataset, leading to gaps that can skew analysis or result in errors. Handling these missing values is crucial for ensuring that any conclusions drawn from the data are accurate.
- **Condensed or Incorrect Headers:** Column names might be too short, unclear, or incorrectly labeled, leading to confusion about what the data actually represents. For example, a column labeled "Pop" might be ambiguous—does it refer to population, popularity, or something else?
- **Row Content Split:** Sometimes, a single column contains data that should be divided into multiple columns, such as when a "Location" column includes both city and state. This issue can make it difficult to analyze the data separately or perform operations that rely on more granular details. These common issues contribute to untidy data, which can complicate analysis and lead to inaccurate results.

Mastering data wrangling is crucial because you might have to handle datasets with millions of rows and hundreds of columns.

Getting Started

First, ensure you have the necessary packages installed and loaded. We will use the `dplyr`, `lubridate`, `readr`, `ggplot2`, and `tidyr` packages for our examples.

> **!** Downloading R-packages
>
> Use `install.packages('Name of Package')` to install an R package. Careful! Package names are case sensitive, so install.packages('GGplot2') will not work, but `install.packages('ggplot2')` will.

```
library(ggplot2)
library(dplyr)
library(lubridate)
library(tidyr)
library(readr)
library(lubridate)
```

Download the Data

> Running Code Block Shortcuts
>
> `Mac users`: Use + return to run single or highlighted line(s). Use shift + return to run entire code block
> `Windows users`: Use ctrl + enter to run single or highlighted line(s). Use shift + enter to run entire code block

```
# Specify the data URL using HTTPS
url1 <- "https://howarder.github.io/ST_437_Data_Viz/Datasets/countries.csv"
url2 <- "https://howarder.github.io/ST_437_Data_Viz/Datasets/countriesExtra.csv"

# Download the data files from the HTTPS URL and save it as
# countries.csv & countriesExtraInfo.csv
cat("Downloading the data ...\n")
download.file(url1, "countries.csv")
download.file(url2, "countriesExtraInfo.csv")

# Check for the data.
cat("After downloading the data, we now have:\n")
list.files()
```

```r
# Read the countries data into R
countriesDs <- read_csv("countries.csv", show_col_types = FALSE)
countriesExtraDs <- read_csv("countriesExtraInfo.csv", show_col_types = FALSE)
```

Verify Datasets with `head()`

```r
head(countriesDs, 5)
head(countriesExtraDs, 5)
```

Description of the Dataset

- **`country`**: Country name from a predefined list of 10 countries.
- **`year`**: Years between 2010 and 2023.
- **`population`**: Real-world population size for each country and year.
- **`gdp`**: Gross Domestic Product (GDP) in USD millions for each country and year.
- **`gdp_per_capita`**: GDP per capita, calculated as GDP divided by population.
- **`life_expectancy`**: Life expectancy of citzens
- **`birth_rate`**: Birth rate for each country and year
- **`temperature`**: Average temperature in celsius per country
- **`region`**: Geographical region corresponding to each country.
- **`category`**: Classification of the country as "First World," "Second World," or "Third World."

Additional Dataset (`countriesExtra.csv`)

- **`country`**: Matches the country names from the main dataset.
- **`continent`**: Continent corresponding to each country.

Tidy Data Wrangling

Important Concepts

- `Filtering`: Filter data based on conditions such as year, country, or region.
- `Selecting`: Select specific columns for focused analysis.
- `Mutating`: Create new columns, such as cases per 100,000 population.
- `Summarizing`: Aggregate data by country, year, or region to find totals and averages.
- `Joining`: Combine the main dataset with the additional data based on country.

Filtering

Filtering is essential for narrowing down datasets to the most relevant information, making patterns easier to identify.

Example 1

You are tasked with visualizing trends in life expectancy in Asian countries between 2010 and 2020.

Filtering Process

```
filteredData <- countriesDs %>%
  filter(year >= 2010 & year <= 2020 & region == "Asia")

head(filteredData, 5)
```

Visualization

```
ggplot(filteredData, aes(x = factor(year), y = life_expectancy, color = country)) +
  geom_point(size = 2) +
  labs(title = "Life Expectancy Trends in Asia (2010-2020)",
       x = "Year", y = "Life Expectancy") +
  theme_minimal()
```

What did we do here?

By focusing on specific countries and years, filtering allows for more targeted and relevant visualizations, making it easier to analyze trends and patterns specific to the context.

Example 2

Analyze the relationship between GDP per capita and life expectancy in European countries with a GDP per capita above $30,000 for the years 2015-2020.

Filtering Process

```r
filteredData <- countriesDs %>%
  filter(year >= 2015 & year <= 2020 & region == "Europe" & gdp_per_capita > 30000) %>%
  left_join(countriesExtraDs, by = "country")

head(filteredData, 5)
```

Visualization

```r
ggplot(filteredData, aes(x = gdp_per_capita, y = life_expectancy, color = country)) +
  geom_point(size = 2) +
  geom_smooth(method = "lm") +
  labs(title = "GDP per Capita vs. Life Expectancy in Europe (2015-2020)",
       x = "GDP per Capita", y = "Life Expectancy") +
  theme_minimal()
```

What did we do here?

Filtering based on economic indicators allows for focused analysis on the relationship between wealth and life expectancy, removing noise from countries with different economic conditions.

Filtering Challenges

1. **Population Size in Africa**:
    - **Filter**: Only countries from Africa where the population exceeds 50 million.
    - **Purpose**: Isolate data for large African nations to analyze trends specific to highly populated areas.

```r
# Your code goes here...
```

2. **Economic Data in Europe**:
    - **Filter**: Show data only for the years 2015-2020 for European countries with GDP per capita above $30,000.
    - **Purpose**: Focus on wealthy European countries during a specific period to study economic outcomes.

```r
# Your code goes here...
```

3. **High Birth Rates in Asia**:
    - **Filter**: Data for Asian countries where the birth rate is above 2.5.

- **Purpose**: Analyze regions with high birth rates, possibly indicating population growth trends.

```
# Your code goes here...
```

4. **Cold Regions in Asia**:
   - **Filter**: Asian countries where the average temperature is below 10°C between 2010 and 2020.
   - **Purpose**: Focus on colder regions in Asia to study how temperature may correlate with other demographic factors.

```
# Your code goes here...
```

5. **GDP Data with Missing Values**:
   - **Filter**: Remove any entries with missing `gdp_per_capita` values for the years 2010-2020.
   - **Purpose**: Ensure clean data for economic analysis, removing incomplete records that could skew results.

```
# Your code goes here...
```

Selecting

Selecting allows you to focus on specific columns relevant to your analysis.

Example

```
selectedData <- filteredData %>%
  select(country, year, gdp_per_capita, life_expectancy)

head(selectedData, 5)
```

Visualization

```
ggplot(selectedData, aes(x = factor(year), y = gdp_per_capita, fill = country)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "GDP per Capita Over Time in Selected European Countries",
       x = "Year", y = "GDP per Capita") +
  theme_minimal()
```

Selecting Challenges

1. **Select columns** related to economic indicators (e.g., `country`, `gdp_per_capita`, `population`) for further analysis.

```
# Your code goes here...
```

2. **Create a dataset** with only the `year`, `life_expectancy`, and `temperature` columns for all countries and show the first 5 rows.

```
# Your code goes here...
```

3. **Choose columns** that exclude any geographical information and check the first 10 rows.

```
# Your code goes here...
```

4. **Select and rename** the `country` and `population` columns to `nation` and `pop_size`, respectively.

```
# Your code goes here...
```

5. **Create a new dataset** with only the `year`, `population`, and a newly created column, `population_in_millions` (which should be calculated as `population / 1e6`).

```
# Your code goes here...
```

Mutating

Mutating helps create new columns based on existing data.

Example

```
mutatedData <- countriesDs %>%
  mutate(gdp_total = gdp_per_capita * population)

head(mutatedData, 5)
```

Visualization

```r
ggplot(mutatedData, aes(x = population, y = gdp_total, color = region)) +
  geom_point(size = 2) +
  labs(title = "Population vs. Total GDP by Region",
       x = "Population", y = "Total GDP") +
  theme_minimal()
```

Mutating Challenges

1. **Create a new column** called `gdp_total` that multiplies `gdp_per_capita` by `population`.

```r
# Your code goes here...
```

2. **Add a new column** that indicates whether a country's GDP per capita is above or below a certain threshold (e.g., $20,000).

```r
# Your code goes here...
```

3. **Mutate the `temperature` column** to create a new column, `temperature_f`, that converts Celsius to Fahrenheit.

```r
# Your code goes here...
```

4. **Create an `avg_region_birthrate` column** by summing the birth rates of each country within the same region and dividing by the number of countries in that region.

```r
# Your code goes here...
```

5. **Generate a column** that calculates the ratio of birth rate to life expectancy for each country.

```r
# Your code goes here...
```

Summarizing

Summarizing aggregates data by country, year, or region to find totals and averages.

Example

```r
summaryData <- countriesDs %>%
  group_by(country) %>%
  summarize(
    avg_life_expectancy = mean(life_expectancy, na.rm = TRUE),
    total_population = sum(population, na.rm = TRUE)
  )

head(summaryData, 5)
```

Visualization

```r
ggplot(summaryData, aes(x = total_population, y = avg_life_expectancy, label = country)) +
  geom_point(size = 2) +
  geom_text(vjust = -0.5) +
  labs(title = "Total Population vs. Average Life Expectancy by Country",
       x = "Total Population", y = "Average Life Expectancy") +
  theme_minimal()
```

Summarizing Challenges

1. **Summarize the dataset** by finding the average temperature for each region.

```r
# Your code goes here...
```

2. **Aggregate the data** to find the total population and average life expectancy for each continent.

```r
# Your code goes here...
```

3. **Group the data** by country and summarize to find the maximum and minimum GDP per capita for each country.

```r
# Your code goes here...
```

4. **Summarize by year** to find the total population and average birth rate each year.

```r
# Your code goes here...
```

5. **Create a summary** that calculates the total population and average GDP per capita for countries classified as "First World."

```
# Your code goes here...
```

Joining

Joining combines data from the main dataset with additional data.

Example

```
joinedData <- left_join(countriesDs, countriesExtraDs, by = "country")

head(joinedData, 5)
```

Visualization

```
ggplot(joinedData, aes(x = continent, y = gdp_per_capita, fill = region)) +
  geom_boxplot() +
  labs(title = "GDP per Capita Distribution by Continent and Region",
       x = "Continent", y = "GDP per Capita") +
  theme_minimal()
```

Joining Challenges

1. **Perform a left join** on the main dataset and an additional dataset that contains information on urbanization rates by country.

```
# Your code goes here...
```

2. **Join the datasets** to add `continent` and then filter for countries in a specific continent.

```
# Your code goes here...
```

3. **Create a new dataset** by performing an inner join on `countriesDs` and `countriesExtraDs`, then analyze the countries present in both datasets.

```
# Your code goes here...
```

4. **Perform a full join** to combine the datasets and identify countries that are missing from one dataset but present in the other.

```
# Your code goes here...
```

5. **Use an anti-join** to find the countries in the main dataset that do not have corresponding information in the additional dataset.

```
# Your code goes here...
```

Untidy Data Wrangling

Handling Wide vs. Long Formats

Untidy data often comes in a "wide" format, where multiple variables are stored across columns rather than in a long format where each observation is a row.

Creating an Untidy Version

To demonstrate this, let's take our dataset and convert it into a wide format, then back to a long format.

```
# Create a wide format from the existing long format
wideData <- countriesDs %>%
  pivot_wider(names_from = year, values_from = c(population, gdp))

# Preview the wide data
head(wideData, 5)
```

Now, let's convert this wide data back to a long format.

```
# Convert the wide format back to long format
longData <- wideData %>%
  pivot_longer(cols = starts_with(c("population", "gdp")),
               names_to = c(".value", "year"),
               names_sep = "_")

# Preview the long data
head(longData, 5)
```

This process shows how data can be reshaped for different analytical needs. Wide format is useful for certain analyses but often needs to be converted to long format for modeling and visualization.

Handling Misspelled Header Column Names

Sometimes datasets come with misspelled or inconsistent column names, which can lead to errors in data manipulation.

Creating Misspelled Header Names

Let's create a dataset with intentionally misspelled column names and then fix them.

```
# Create a dataset with misspelled column names
misspelledHeaders <- countriesDs %>%
  rename(
    poplation = population,
    gdp_pper_capita = gdp_per_capita,
    lif_expectancy = life_expectancy
  )

# Preview the dataset with misspelled headers
head(misspelledHeaders, 5)
```

Fixing the Misspelled Column Names

```
# Correct the misspelled column names
correctedHeaders <- misspelledHeaders %>%
  rename(
    population = poplation,
    gdp_per_capita = gdp_pper_capita,
    life_expectancy = lif_expectancy
  )

# Preview the corrected dataset
head(correctedHeaders, 5)
```

This illustrates how to identify and correct misspelled column names, which is a crucial step in data cleaning.

Handling Row Content Split and Reverse

Sometimes, data stored in a single column needs to be split into multiple columns or vice versa.

Merging Two Columns into One

Let's take the `country` and `year` columns and merge them into a single column.

```
# Combine 'country' and 'year' into a single column
mergedData <- countriesDs %>%
  unite("country_year", country, year, sep = "_")

# Preview the merged dataset
head(mergedData, 5)
```

Splitting the Merged Column Back into Two

```
# Split the 'country_year' column back into 'country' and 'year'
splitData <- mergedData %>%
  separate(country_year, into = c("country", "year"), sep = "_")

# Preview the split dataset
head(splitData, 5)
```

This demonstrates how to handle situations where data needs to be recombined or separated for different purposes.

Handling Dates

In some cases, it might be necessary to convert a `year` column from a numeric format (double) into a proper date format for time series analysis or plotting purposes. Here's how you can do that in R using the `lubridate` package.

Example

Converting `year` to a Date

Let's convert the `year` column into a date format, setting it as January 1st of that year.

```
# Ensure the lubridate package is loaded
# Convert the year column to a date format (January 1st of each year)
countriesDs <- countriesDs %>%
  mutate(year_date = ymd(paste0(year, "-01-01")))
```

```r
# Check the first few rows to see the new column
head(countriesDs, 5)
```

What did we do here?

- `paste0(year, "-01-01")`: Combines the year with the string "-01-01" to create a date string like "2010-01-01".
- `ymd()`: Converts the resulting string into a date object in the "Year-Month-Day" format.

This creates a new column, `year_date`, which is now in the proper date format.

Visualization

```r
ggplot(countriesDs, aes(x = year_date, y = population, color = country)) +
  geom_line(size = 1) +
  labs(title = "Population Trends Over Time by Country",
       x = "Year", y = "Population") +
  theme_minimal()
```

Challenges

1. **Convert year to end of year date**:
    - **Task**: Convert the `year` column to a date format, but set it as December 31st of that year.
    - **Purpose**: Useful for representing data that summarizes annual results.

```r
# Your code goes here...
```

2. **Create a quarterly date**:
    - **Task**: Convert the `year` column into a date representing the first quarter (e.g., "2010-03-31").
    - **Purpose**: Useful for quarterly analysis.

```r
# Your code goes here...
```

3. **Mid-year date conversion**:
    - **Task**: Convert the `year` column to a date format, setting it as June 30th of each year.
    - **Purpose**: Represents mid-year data points.

```
# Your code goes here...
```

4. **Use year as a dynamic time period**:
    - **Task**: Convert the `year` column to represent the last day of a chosen month (e.g., November).
    - **Purpose**: Allows for flexibility depending on the analysis context.

```
# Your code goes here...
```

5. **Convert year to fiscal year start date**:
    - **Task**: Convert the `year` column into a date representing the start of the fiscal year (e.g., April 1st).
    - **Purpose**: Useful for financial and budgetary analyses.

```
# Your code goes here...
```

Handling Missing Data

Dealing with missing data is a crucial aspect of data wrangling. Missing data can occur for various reasons, and understanding the nature of these missing values is essential for appropriate handling.

Types of Missing Data

- **MCAR (Missing Completely at Random)**: Data is missing entirely at random, with no relationship between the missing data and any other observed or unobserved data. The analysis remains unbiased if this missing data is ignored.
- **MAR (Missing at Random)**: The likelihood of missing data on a variable is related to other observed variables but not to the value of the variable itself.
- **MNAR (Missing Not at Random)**: The missingness is related to the unobserved data itself, meaning the missing values are related to the actual value that is missing.

Simulating MAR Data

We'll create a scenario where `gdp_per_capita` is more likely to be missing if the population is below a certain threshold, making it "missing at random" based on population size.

```r
set.seed(10)  # For reproducibility
# Directly introduce missing values in a random sample of rows
mar_data <- countriesDs %>%
  mutate(
    gdp_per_capita = ifelse(runif(n()) < 0.4, NA, gdp_per_capita)  # 20% chance of being NA
  )
# Check for missing values
summary(mar_data$gdp_per_capita)  # Provides a summary of the column, including missing value
```

Simulating MNAR Data

We'll simulate a scenario where the likelihood of `life_expectancy` being missing is higher if life expectancy is lower than 60 years, making it "missing not at random."

```r
set.seed(456)  # For reproducibility

mnar_data <- countriesDs %>%
  mutate(
    life_expectancy = ifelse(life_expectancy < 60 & runif(n()) < 0.4, NA, life_expectancy)
  )
# Check for missing values
summary(mnar_data$life_expectancy)
```

Missing Data | Before Wrangling

Before Handling Missing Data

Let's visualize the data before handling missing values, focusing on the relationship between GDP per capita and life expectancy.

Visualization with MAR Data

```r
ggplot(mar_data, aes(x = gdp_per_capita, y = life_expectancy, color = country)) +
  geom_point(size = 2, alpha = 0.6) +
  labs(title = "GDP per Capita vs. Life Expectancy (with MAR)",
       x = "GDP per Capita", y = "Life Expectancy") +
  theme_minimal()
```

Visualization with MNAR Data

```
ggplot(mnar_data, aes(x = gdp_per_capita, y = life_expectancy, color = country)) +
  geom_point(size = 2, alpha = 0.6) +
  labs(title = "GDP per Capita vs. Life Expectancy (with MNAR)",
       x = "GDP per Capita", y = "Life Expectancy") +
  theme_minimal()
```

Missing Data | How to Wrangle

To handle the missing data, we'll apply a simple imputation strategy, filling in missing values with the median of the respective variable.

Imputing Missing Values for MAR Data

```
mar_data_imputed <- mar_data %>%
  mutate(
    gdp_per_capita = ifelse(is.na(gdp_per_capita), median(gdp_per_capita, na.rm = TRUE), gdp
  )
# Check imputation
summary(mar_data_imputed$gdp_per_capita)
```

Imputing Missing Values for MNAR Data

```
mnar_data_imputed <- mnar_data %>%
  mutate(
    life_expectancy = ifelse(is.na(life_expectancy), median(life_expectancy, na.rm = TRUE), l
  )
# Check imputation
summary(mnar_data_imputed$life_expectancy)
```

Missing Data | After Wrangling

Let's visualize the data again after imputing the missing values.

Visualization with MAR Data (Imputed)

```r
ggplot(mar_data_imputed, aes(x = gdp_per_capita, y = life_expectancy, color = country)) +
  geom_point(size = 2, alpha = 0.6) +
  labs(title = "GDP per Capita vs. Life Expectancy (after MAR Imputation)",
       x = "GDP per Capita", y = "Life Expectancy") +
  theme_minimal()
```

Visualization with MNAR Data (Imputed)

```r
ggplot(mnar_data_imputed, aes(x = gdp_per_capita, y = life_expectancy, color = country)) +
  geom_point(size = 2, alpha = 0.6) +
  labs(title = "GDP per Capita vs. Life Expectancy (after MNAR Imputation)",
       x = "GDP per Capita", y = "Life Expectancy") +
  theme_minimal()
```

Summary

Understanding the different types of missing data—MCAR, MAR, and MNAR—is crucial for choosing the right approach to handle them. We explored how to simulate and visualize MAR and MNAR scenarios in our dataset, highlighting the importance of addressing missing data for accurate analysis. By comparing visualizations before and after imputation, students can grasp the significant impact missing data can have on their results and learn effective strategies to mitigate these issues.

Creating a Pseudo-Publication Ready Visualization

We'll combine all the data wrangling techniques you've learned—`filtering`, `selecting`, `mutating`, `summarizing`, and `joining`—to perform a detailed analysis and produce a polished, publication-ready visualization.

Creating a Professional-Quality Visualization

Here's a step-by-step guide to transform and visualize data from 10 countries in the dataset:

```r
# Step 1: Filter the dataset for Asian and European countries from 2020 to 2023
filteredData <- countriesDs %>%
  filter(year >= 2020 & year <= 2023)

# Step 2: Join with the additional dataset to get continent information
joinedData <- filteredData %>%
```

```r
  left_join(countriesExtraDs, by = "country")

# Step 3: Select relevant columns for analysis
selectedData <- joinedData %>%
  select(country, year, population, gdp_per_capita, life_expectancy, birth_rate, continent)

# Step 4: Mutate the data to create new metrics
mutatedData <- selectedData %>%
  mutate(
    gdp_total = gdp_per_capita * population,
    year_date = ymd(paste0(year, "-", ifelse(runif(n()) > 0.5, "06-30", "12-31")))
  )

# Step 5: Summarize the data to get average life expectancy, GDP per capita, and total popula
summaryData <- mutatedData %>%
  group_by(continent, country) %>%
  summarize(
    avg_life_expectancy = mean(life_expectancy, na.rm = TRUE),
    avg_gdp_per_capita = mean(gdp_per_capita, na.rm = TRUE),
    total_population_millions = sum(population, na.rm = TRUE) / 1e6  # Convert population to
  ) %>%
  ungroup()

# Step 6: Create a publication-ready visualization
ggplot(summaryData, aes(
  x = avg_gdp_per_capita,
  y = avg_life_expectancy,
  size = total_population_millions,  # Use population in millions for size mapping
  color = country
)) +
  geom_point(alpha = 0.4) +
  scale_y_continuous(limits = c(min(summaryData$avg_life_expectancy) - 1, 66)) +  # Adjust y-
  scale_size_continuous(
    range = c(5, 15),  # Adjust the bubble size range
    name = "Total Population (Millions)",
    breaks = c(50, 200, 500, 1000, 1500),  # Adjust based on realistic population ranges in r
    labels = c("50M", "200M", "500M", "1B", "1.5B")  # Correct labels in millions
  ) +
  scale_color_viridis_d(guide = "none") +  # Hide the color legend
  labs(
    title = "Relationship Between GDP per Capita and Life Expectancy Across Countries",
    subtitle = "Bubble size represents total population across 2020-2023",
```

```
    x = "Average GDP per Capita (USD)",
    y = "Average Life Expectancy (Years)",
    caption = "Data Source: Synthetic Dataset"
  ) +
  geom_text(aes(label = country), hjust = 0.5, size = 3, color = "black") + # Labels centered
  theme_minimal() +
  theme(
    text = element_text(family = "Georgia"),
    plot.title = element_text(size = 11, face = "bold"),
    plot.subtitle = element_text(size = 9, face = "italic"),
    axis.title = element_text(size = 8),
    legend.title = element_text(size = 8),
    legend.text = element_text(size = 8),
    legend.position = "bottom"
  )
```

Review & Scrutinize

What is this visualization?: Write a brief paragraph describing the design, purpose, and key message of the plot. Explain what the visualization is intended to show and how it effectively communicates the data.

Why is this visualization nearly publication-ready?: In 2-3 sentences, discuss what makes the plot polished and professional, highlighting any elements that could make it suitable for publication.