Web Scaping and Data Management - Exam 2

Introduction:

Objective:

The aim of this detailed guide is to understand whether higher GDP per capita is indicative of better quality democracy. The relevant data from all countries for different democracy indices and the GDP (nominal) per capita will be scraped from the following Wikipedia websites:

List of countries by GDP (nominal) per capita

Democracy indices (V-Dem)

The Economist Democracy Index

Bertelsmann Transformation Index

Then, a database will be created containing the relevant information from the democracy indices. Moreover, a new standardised democracy index will be created by averaging the normalised values from the different categories of each index, bringing them onto a common scale (e.g., 0 to 1). The GDP (nominal) per capita data will be added to the table without further adjustment.

Finally, a plot will be produced depicting the GDP (nominal) per capita on the x-axis and the average democracy score on the y-axis. The three indicies will be differentiated on the plot using different coloured dots.

Please Note:

The analysis requires GDP (nominal) per capita data, not GDP (nominal) per country. Initially, it was assumed that the data would be sourced from the Wikipedia page titled List of countries by GDP (nominal). However, this page was updated on June 20, 2024, and now only includes GDP (nominal) per country.

To obtain the correct data, the information will be scraped from the Wikipedia page List of countries by GDP (nominal) per capita, which provides the relevant GDP (nominal) per capita figures.

Prerequisites:

Software Requirements

- R version 4.4.1 or higher
- Required R packages:
    - To install the required packages, use the following command: `install.packages(c("rvest", "tidyverse", "stringr", "pacman", "purrr", "DBI", "RSQLite", "ggplot2"))`.

Data Requirements

- No specific data requirements, since this script directly scrapes the data from the URLs below.

Procedure Steps:

Step 1: Load Required Libraries

- Use the `p_load` function from the `pacman` package to load multiple libraries simultaneously. This includes:
  - `rvest` for web scraping.
  - `tidyverse` for comprehensive data manipulation.
  - `stringr` for string operations.
  - `purrr` for functional programming tools.
  - `xml2` for parsing XML and HTML data.
  - `DBI` for interacting with relational databases in R.
  - `RSQLite` for implementing the `DBI` interface specifically with SQLite databases.
  - `ggplot2` for data visualisation using plots.

```
pacman::p_load(rvest, tidyverse, stringr, purrr, xml2, DBI, RSQLite, ggplot2)
```

Step 2: Web Scraping the Wikipedia Data:

- Request and collect the raw HTML content from the Wikipedia URLs:

```
# Request & collect raw html content from the Wikipedia URLs
GDP_Capita_html <- read_html("https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)
VDem_html <- read_html("https://en.wikipedia.org/w/index.php?title=V-Dem_Democracy_Indices&old
Econ_html <- read_html("https://en.wikipedia.org/w/index.php?title=The_Economist_Democracy_Ind
Bert_html <- read_html("https://en.wikipedia.org/w/index.php?title=Bertelsmann_Transformation_
```

Using the `read_html` function from the `rvest` package, request and collect the raw HTML data from the following Wikipedia URLs, storing them in their associated objects: `GDP_Capital_html`, `VDem_html`, `Econ_html` and `Bert_html`.

Step 3: Parse the Data

- Parse the HTML content from the unstructured data sources, `GDP_Capita_html`, `VDem_html`, `Econ_html`, and `Bert_html`, to extract relevant information and transform it into a structured format. This involves identifying and selecting specific HTML tables from each source and converting them into data frames that can be easily analysed.

```
# Parse the GDP per Capita table
GDP_Capita_table <- GDP_Capita_html |>
  html_elements(".wikitable") |>  # Select the right element
  html_table(fill = TRUE) |>      # Parse the table, handling merged cells
  pluck(1)                        # Extract the first table from the list
```

```
# Parse the VDem_table
VDem_table <- VDem_html |>
  html_elements(".wikitable") |>  # Select the right element
  pluck(1) |>                     # Extract the first table from the list
  html_table(fill = TRUE)         # Parse the table, handling merged cells

# Parse the Econ_table
Econ_table <- Econ_html |>
  html_elements(".wikitable") |> # select the right element
  html_table() |>                # special function for tables
  pluck(4)                       # extract the fourth table

# Parse the Bert_table
Bert_table <- Bert_html |>
  html_elements(".wikitable") |>  # Select the right element
  pluck(1) |>                     # Extract the first table from the list
  html_table(fill = TRUE)         # Parse the table, handling merged cells
```

- **Identify the HTML Tables:** Use the `html_elements` function from the `rvest` package to target tables identified by the `.wikitable` class in the HTML documents.
- **Parse and Handle Merged Cells:** Apply the `html_table(fill = TRUE)` function to correctly parse the tables, ensuring that any merged cells are handled appropriately.
- **Extract Specific Tables:** Use the `pluck` function from the `purrr` package to select the relevant table from the list of parsed tables within each HTML document.
- **Store in Data Frames:** The extracted tables are stored in the following data frames:
  - `GDP_Capita_table`: Contains data parsed from `GDP_Capita_html`.
  - `VDem_table`: Contains data parsed from `VDem_html`.
  - `Econ_table`: Contains data parsed from `Econ_html`.
  - `Bert_table`: Contains data parsed from `Bert_html`.

Step 4: Wrangling the Data

- In this step, the raw data is transformed and structured into a clean, consistent format to enhance data quality, making it more usable for analysis. The data wrangling process is applied to each table, following a systematic approach that involves renaming columns, removing unwanted rows, filtering out irrelevant data, cleaning the content, and converting data types.

1. Wrangling the `GDP_Capita_table`

- **Rename Columns:** Use the `colnames` function from `Base R` to provide clear, descriptive names for the columns to ensure that the data is easy to interpret.
- **Remove Unwanted Rows:** The `slice(-1)` function from the `tidyverse` package removes the first row, which often contains redundant header information.

- **Filter the Data:** The `filter` function from the `tidyverse` package, combined with `str_detect` from the `stringr` package, is used to exclude rows with non-numeric values in the `World_Bank_Estimate` column, specifically those marked with an em dash (`"—"`).
- **Clean the Data:** The `mutate` and `across` functions from the `tidyverse` package are employed to clean the data. `str_remove_all` from the `stringr` package is used to remove em dashes, commas, and any content within brackets, ensuring that the numeric data is accurate.
- **Convert Data Types:** Columns are converted to numeric and integer types using `mutate` with `as.numeric` and `as.integer` to ensure that the data is ready for analysis.

```
# Data Wrangling: GDP per Capita table
# Rename columns using colnames from Base R
colnames(GDP_Capita_table) <- c("Country_Territory", "IMF_Estimate", "IMF_Year", "World_Bank_E

# Remove the first row with the unwanted headers
GDP_Capita_table <- GDP_Capita_table|>
  slice(-1)

# Filter out rows with non-numeric World Bank estimates (i.e., entries with em dashes)
GDP_Capita_table <- GDP_Capita_table |>
  filter(
    !str_detect(World_Bank_Estimate, "—")
  )

# Clean data by removing em dashes, commas, and content within brackets
GDP_Capita_table <- GDP_Capita_table |>
  mutate(across(c(IMF_Estimate, IMF_Year, UN_Estimate, UN_Year), ~ str_remove_all(., "[—]")))
  mutate(across(c(IMF_Estimate, World_Bank_Estimate, UN_Estimate), ~ str_remove_all(., ",")))
  mutate(across(c(IMF_Year, World_Bank_Year, UN_Year), ~ str_remove_all(., "\\[.*?\\]"))) # Re

# Convert columns to numeric and integer types for appropriate data analysis
GDP_Capita_table <- GDP_Capita_table %>%
  mutate(
    IMF_Estimate = as.numeric(IMF_Estimate),
    World_Bank_Estimate = as.numeric(World_Bank_Estimate),
    UN_Estimate = as.numeric(UN_Estimate),
    IMF_Year = as.integer(IMF_Year),
    World_Bank_Year = as.integer(World_Bank_Year),
    UN_Year = as.integer(UN_Year)
  )
```

2. Wrangling the `VDem_table`

- **Rename Columns:** The `colnames` function is used to rename the columns for clarity, aligning with the standard naming conventions.
- **Remove Unwanted Rows:** The `slice(-1)` function is used again to eliminate the first row, which is unnecessary for analysis.

```
# Data Wrangling: VDem table
# Rename columns using colnames from Base R
```

```
 colnames(VDem_table) <- c("VDem_Country", "Democracy_Indices_Electorial", "Democracy_Indicies_
```

```
# Remove the first row which is an unwanted header
VDem_table <- VDem_table |>
  slice(-1)
```

3. Wrangling the `Econ_table`

- **Rename Columns:** The `rename` function from the `tidyverse` package is used to standardise column names to ensure that they are clear and consistent across the dataset.

```
# Data Wrangling: Econ table
# Rename columns using rename from tidyverse
Econ_table <- Econ_table |>
  rename(
   Econ_Country = "Country",
   Rank_2023 = "2023 rank",
   Regime_type = "Regime type",
   Year_2023 = "2023",
   Year_2022 = "2022",
   Year_2021 = "2021",
   Year_2020 = "2020",
   Year_2019 = "2019",
   Year_2018 = "2018",
   Year_2017 = "2017",
   Year_2016 = "2016",
   Year_2015 = "2015",
   Year_2014 = "2014",
   Year_2013 = "2013",
   Year_2012 = "2012",
   Year_2011 = "2011",
   Year_2010 = "2010",
   Year_2008 = "2008",
   Year_2006 = "2006"
  )
```

4. Wrangling the `Bert_table`

- **Rename Columns:** Similar to the other tables, the `rename` function from the `tidyverse` package is used to provide descriptive names for each column.
- **Clean the Data:** The `mutate` and `across` functions remove em dashes from relevant columns using `str_remove_all`.
- **Convert Data Types:** Columns are converted to numeric and integer types using `mutate` with `as.numeric` and `as.integer` to ensure that the data is ready for analysis.

```
# Data wrangling: Bert table
#Rename columns using rename from tidyverse and clean data
 Bert_table <- Bert_table  |>
  rename(
    Bert_Country = "Country",
```

```
    Status_Index_2022 = "Status-Index 2022",
    Governance_Index_2022 = "Governance-Index 2022",
    Status_Index_2020 = "Status-Index 2020",
    Governance_Index_2020 = "Governance-Index 2020",
    Status_Index_2018 = "Status-Index 2018",
    Governance_Index_2018 = "Governance-Index 2018",
    Status_Index_2016 = "Status-Index 2016",
    Governance_Index_2016 = "Governance-Index 2016"
  ) |>
  mutate(across(c(Status_Index_2018, Governance_Index_2018, Status_Index_2016, Governance_Inde

# Convert columns to numeric and integer types
Bert_table <- Bert_table %>%
  mutate(
    Status_Index_2022 = as.numeric(Status_Index_2022),
    Governance_Index_2022 = as.numeric(Governance_Index_2022),
    Status_Index_2020 = as.numeric(Status_Index_2020),
    Governance_Index_2020 = as.integer(Governance_Index_2020),
    Status_Index_2018 = as.integer(Status_Index_2018),
    Governance_Index_2018 = as.integer(Governance_Index_2018),
    Status_Index_2016 = as.integer(Status_Index_2016),
    Governance_Index_2016 = as.integer(Governance_Index_2016)
  )
```

This methodical approach ensures that each table is properly formatted, cleaned, and prepared for further analysis, leading to more accurate and meaningful results.

Step 5: Create a Local SQLite Database

- Create a local SQLite database, connect to it, and define tables to store the data.
- This process involves setting up the database connection, creating tables with specified data types, and importing data from the data frames into the database for storage and further analysis.

1. Set Up and Connect to the SQLite Database

- **Establish Connection:** Use the `dbConnect` function from the `DBI` package to create and connect to a SQLite database. The `SQLite` function provides the driver necessary for SQLite database management.
- **Database Creation:** The following code snippet creates and connects to a new SQLite database named `db.sql`:

```
#create and connect to a SQLite database called db
db <- dbConnect(SQLite(), "data/db.sql")
```

2. Create the `GDP_Capita` Table in the Database

- **Define the Table Structure:** Use the `dbExecute` function from the `DBI` package to create the `GDP_Capita` table within the database. The table includes columns such as `Country_Territory`, `IMF_Estimate`, `IMF_Year`, etc. Each column is assigned a specific data type:
  - `varchar`: Stores text data, such as country names.

  - `real`: Stores floating-point numbers, ideal for estimates and indices.
  - `integer`: Stores whole numbers, such as years.
- **List Existing Tables:** After creating the table, use the `dbListTables` function from the `DBI` package to verify that the table was successfully added to the database.
- **Insert Data into the Table:** Use the `dbWriteTable` function from the `DBI` package to insert data from the `GDP_Capita_table` data frame into the `GDP_Capita` table within the database.
- **Verify Data Import:** Optionally, you can use the `dbReadTable` function to read the table and verify that the data was correctly imported. If unsatisfied with the table import, `dbRemoveTable` function can be used to remove it. Both functions are from the `DBI` package.

```
#create GDP_Capita table in the db
dbExecute(db,
          "CREATE TABLE GDP_Capita (
            Country_Territory varchar PRIMARY KEY,
            Democracy_ real,
            IMF_Estimate real,
            IMF_Year integer,
            World_Bank_Estimate real,
            World_Bank_Year integer,
            UN_Estimate real,
            UN_Year integer
          )")
```

```
[1] 0
```

```
#List the tables within the db database
dbListTables(db)
```

```
[1] "GDP_Capita"
```

```
# Write the data from GDP_Capita_table to the GDP_Capita table in the database
dbWriteTable(db, "GDP_Capita", GDP_Capita_table, append = TRUE, row.names = FALSE)

# Verify that the data was imported from GDP_Capita_Table
#dbReadTable(conn = db, "GDP_Capita")

#dbRemoveTable(conn = db, "GDP_Capita")
```

3. Create the `VDem` Table in the Database

- **Table Creation:** Similar to the `GDP_Capita` table, define the structure of the `VDem` table using `dbExecute`. This table stores various democracy indices.
- **Data Insertion:** Insert data from the `VDem_table` data frame into the `VDem` table using the `dbWriteTable` function.
- **Verification:** List the tables and verify data import if needed using the `dbListTables` and `dbReadTable` functions.

```
#create VDem table in the db
dbExecute(db,
          "CREATE TABLE VDem (
            VDem_Country varchar,
            Democracy_Indices_Electorial real,
            Democracy_Indicies_Liberal real,
            DCI_Liberal real,
            DCI_Egalitarian real,
            DCI_Participatory real,
            DCI_Deliberative real
          )")
```

```
[1] 0
```

```
#List the tables within the db database
dbListTables(db)
```

```
[1] "GDP_Capita" "VDem"
```

```
# Write the data from VDem_table to the VDem table in the database
dbWriteTable(db, "VDem", VDem_table, append = TRUE, row.names = FALSE)

# Verify that the data was imported from VDem_Table
#dbReadTable(conn = db, "VDem")

#dbRemoveTable(conn = db, "VDem")
```

4. Create the `Econ` Table in the Database

- **Table Creation:** Use `dbExecute` to create the `Econ` table, designed to store economic data across different years for various countries and regions.
- **Data Insertion:** Insert the data from the `Econ_table` data frame into the `Econ` table in the database using the `dbWriteTable` function.
- **Verification:** List the tables and verify data import if needed using the `dbListTables` and `dbReadTable` functions.

```
#create Econ table in the db
dbExecute(db,
          "CREATE TABLE Econ (
            Region varchar,
            Rank_2023 integer,
            Econ_Country varchar,
            Regime_type varchar,
            Year_2023 real,
            Year_2022 real,
            Year_2021 real,
            Year_2020 real,
            Year_2019 real,
            Year_2018 real,
            Year_2017 real,
```

```
            Year_2016 real,
            Year_2015 real,
            Year_2014 real,
            Year_2013 real,
            Year_2012 real,
            Year_2011 real,
            Year_2010 real,
            Year_2008 real,
            Year_2006 real
          )")
```

```
[1] 0
```

```
#List the tables within the db database
dbListTables(db)
```

```
[1] "Econ"       "GDP_Capita" "VDem"
```

```
# Write the data from VDem_table to the VDem table in the database
dbWriteTable(db, "Econ", Econ_table, append = TRUE, row.names = FALSE)

# Verify that the data was imported from VDem_Table
#dbReadTable(conn = db, "Econ")

#dbRemoveTable(conn = db, "Econ")
```

5. Create the `Bert` Table in the Database

- **Table Creation:** Define and create the `Bert` table using `dbExecute`. This table holds various governance and status indices.
- **Data Insertion:** Insert the data from the `Bert_table` data frame into the `Bert` table in the database using the `dbWriteTable` function..
- **Verification:** List the tables and verify data import if needed using the `dbListTables` and `dbReadTable` functions.

```
#create Econ table in the db
dbExecute(db,
          "CREATE TABLE Bert (
            Bert_Country varchar,
            Status_Index_2022 real,
            Governance_Index_2022 real,
            Status_Index_2020 real,
            Governance_Index_2020 real,
            Status_Index_2018 real,
            Governance_Index_2018 real,
            Status_Index_2016 real,
            Governance_Index_2016 real
          )")
```

```
[1] 0
```

```
#List the tables within the db database
dbListTables(db)
```

```
[1] "Bert"       "Econ"       "GDP_Capita" "VDem"
```

```
# Write the data from VDem_table to the VDem table in the database
dbWriteTable(db, "Bert", Bert_table, append = TRUE, row.names = FALSE)

# Verify that the data was imported from VDem_Table
#dbReadTable(conn = db, "Bert")

#dbRemoveTable(conn = db, "Bert")
```

Step 6: Write a query that combines the data in the analysis dataset

- Write an SQL query to combine the data from multiple tables in the SQLite database (e.g., `GDP_Capita`, `VDem`, `Econ`, and `Bert`) into a single dataset for analysis called `Combined_table`. This combined dataset will allow for a comprehensive analysis of various economic, democratic, and governance indicators.

1. Review the SQL Query Structure

The SQL query is structured to select specific columns from each table and join these tables based on a **primary key** and **foreign keys**. In this case, the primary key is the `Country_Territory` column from the `GDP_Capita` table, while the foreign keys are the `VDem_Country`, `Econ_Country`, and `Bert_Country` columns in the `VDem`, `Econ`, and `Bert` tables, respectively.

- **SELECT Clause:** This clause specifies the columns that will be included in the `Combined_table`. Columns from each of the four tables are selected based on their relevance to the analysis.
- **FROM Clause:** The `GDP_Capita` table is used as the primary table in this query. It serves as the base table for combining the data, given that it contains the primary key `Country_Territory`.
- **LEFT JOIN Operations:** The query uses `LEFT JOIN` operations to merge the tables. The `LEFT JOIN` ensures that all records from the `GDP_Capita` table are included in the combined dataset, even if there are no matching records in the other tables. The joins are structured as follows:
  - **VDem Table:** Joins the `GDP_Capita` table using `Country_Territory = VDem_Country` to include democracy indices.
  - **Econ Table:** Joins the `GDP_Capita` table using `Country_Territory = Econ_Country` to add economic data across multiple years.
  - **Bert Table:** Joins the `GDP_Capita` table using `Country_Territory = Bert_Country` to incorporate governance and status indices.

```
query <- "
SELECT
  GDP_Capita.Country_Territory,
  GDP_Capita.IMF_Estimate,
  GDP_Capita.IMF_Year,
```

```
  GDP_Capita.World_Bank_Estimate,
  GDP_Capita.World_Bank_Year,
  GDP_Capita.UN_Estimate,
  GDP_Capita.UN_Year,
  VDem.VDem_Country,
  VDem.Democracy_Indices_Electorial,
  VDem.Democracy_Indicies_Liberal,
  VDem.DCI_Liberal,
  VDem.DCI_Egalitarian,
  VDem.DCI_Participatory,
  VDem.DCI_Deliberative,
  Econ.Econ_Country,
  Econ.Year_2023,
  Econ.Year_2022,
  Econ.Year_2021,
  Econ.Year_2020,
  Econ.Year_2019,
  Econ.Year_2018,
  Econ.Year_2017,
  Econ.Year_2016,
  Econ.Year_2015,
  Econ.Year_2014,
  Econ.Year_2013,
  Econ.Year_2012,
  Econ.Year_2011,
  Econ.Year_2010,
  Econ.Year_2008,
  Econ.Year_2006,
  Bert.Bert_Country,
  Bert.Status_Index_2022,
  Bert.Governance_Index_2022,
  Bert.Status_Index_2020,
  Bert.Governance_Index_2020,
  Bert.Status_Index_2018,
  Bert.Governance_Index_2018,
  Bert.Status_Index_2016,
  Bert.Governance_Index_2016
FROM
  GDP_Capita
LEFT JOIN
  VDem
ON
  GDP_Capita.Country_Territory = VDem.VDem_Country
LEFT JOIN
  Econ
ON
  GDP_Capita.Country_Territory = Econ.Econ_Country
LEFT JOIN
  Bert
ON
  GDP_Capita.Country_Territory = Bert.Bert_Country
"
```

2. Execution and Verification

After writing the query, it is executed using the `dbGetQuery` function from the `DBI` package, which retrieves the combined data into the `Combined_table` dataframe. This dataframe can then be used for subsequent analysis.

- **Execution:** Run the SQL query using the `dbGetQuery` function to generate the `Combined_table`.
- **Verification:** Optionally, verify the structure and contents of `Combined_table` to ensure that the data has been correctly combined using the `View` function

```
# Execute the query and store the result in Combined_table
Combined_table <- dbGetQuery(db, query)

# Optional: View the first few rows of the Combined_table to verify the data
#View(Combined_table)
```

Step 7: Create Normalised Indices for VDem, Econ and Bert Indices

- Create normalised indices for the VDem, Econ, and Bert datasets.
- This involves calculating an average score from the various categories within each index and then normalising these scores to a standard scale (from 0, representing the lowest value, to 1, representing the highest value).

Normalising the indices makes it easier to compare the different measures on a common scale.

1. Calculate the Average VDem Score

- To create a composite measure for VDem, the average of six different democracy-related indices is calculated.
- These indices include electoral democracy, liberal democracy, and four categories of the Democracy-Centred Index (DCI): Liberal, Egalitarian, Participatory, and Deliberative.
    - **Calculation:** The VDem average score is computed by summing the values of these six indices for each record and then dividing by 6 (the total number of indices).

```
# Calculate the average VDem score
Combined_table$Avg_VDem <- (
  Combined_table$Democracy_Indices_Electorial +
  Combined_table$Democracy_Indicies_Liberal +
  Combined_table$DCI_Liberal +
  Combined_table$DCI_Egalitarian +
  Combined_table$DCI_Participatory +
  Combined_table$DCI_Deliberative
) / 6
```

2. Normalise the VDem Score (0-1 Scale)

- Normalisation involves rescaling the average VDem scores to fit within a 0 to 1 range. This is achieved by subtracting the minimum value of the average VDem scores from each score and then dividing by the range (i.e., the difference between the maximum and minimum values).

- The result is a normalized VDem score where 0 represents the lowest score and 1 represents the highest.

```
# Calculate the normalised VDem score (0-1 scale)
Combined_table$Norm_VDem <- (Combined_table$Avg_VDem - min(Combined_table$Avg_VDem, na.rm = TR
  (max(Combined_table$Avg_VDem, na.rm = TRUE) - min(Combined_table$Avg_VDem, na.rm = TRUE))
```

3. Calculate the Average Econ Score

- For the Econ index, the average score is calculated based on economic data collected over multiple years. Specifically, the score is the average of values from 16 different years (from 2006 to 2023). This gives a composite measure of economic performance over time.
    - **Calculation:** The average Econ score is computed by summing the economic indicators for the specified years and then dividing by 16.

```
# Calculate the average Econ score
Combined_table$Avg_Econ  <- (
  Combined_table$Year_2023 +
  Combined_table$Year_2022 +
  Combined_table$Year_2021 +
  Combined_table$Year_2020 +
  Combined_table$Year_2019 +
  Combined_table$Year_2018 +
  Combined_table$Year_2017 +
  Combined_table$Year_2016 +
  Combined_table$Year_2015 +
  Combined_table$Year_2014 +
  Combined_table$Year_2013 +
  Combined_table$Year_2012 +
  Combined_table$Year_2011 +
  Combined_table$Year_2010 +
  Combined_table$Year_2008 +
  Combined_table$Year_2006
) / 16
```

4. Normalize the Econ Score (0-1 Scale)

- As with the VDem index, the Econ scores are normalised to fit within a 0 to 1 scale. This allows for direct comparison of the Econ index with other indices.

```
# Calculate the normalised Econ score (0-1 scale)
Combined_table$Norm_Econ  <- (Combined_table$Avg_Econ - min(Combined_table$Avg_Econ, na.rm = T
  (max(Combined_table$Avg_Econ, na.rm = TRUE) - min(Combined_table$Avg_Econ, na.rm = TRUE))
```

5. Calculate the Average Bert Score

The Bertelsmann Transformation Index (Bert) measures governance and status indices over different years. The average Bert score is calculated by taking the mean of eight different measures, which include Status and Governance indices from four different years (2022, 2020, 2018, and 2016).

- **Calculation:** The average Bert score is computed by summing the eight indices and dividing by 8.

```
# Calculate the average Bert score
Combined_table$Avg_Bert  <- (
  Combined_table$Status_Index_2022 +
  Combined_table$Governance_Index_2022 +
  Combined_table$Status_Index_2020 +
  Combined_table$Governance_Index_2020 +
  Combined_table$Status_Index_2018 +
  Combined_table$Governance_Index_2018 +
  Combined_table$Status_Index_2016 +
  Combined_table$Governance_Index_2016
) / 8
```

6. Normalize the Bert Score (0-1 Scale)

- The Bert score is also normalised to a 0 to 1 scale to align it with the other indices.

```
# Calculate the normalised Bert score (0-1 scale)
Combined_table$Norm_Bert  <- (Combined_table$Avg_Bert - min(Combined_table$Avg_Bert, na.rm = T
  (max(Combined_table$Avg_Bert, na.rm = TRUE) - min(Combined_table$Avg_Bert, na.rm = TRUE))
```

7. Remove Rows with Missing Data in Normalised Indices

Finally, any rows with missing values (`NA`) in the normalised indices (`Norm_VDem`, `Norm_Econ`, or `Norm_Bert`) are removed using the `drop_na` function. This ensures that the analysis dataset is complete and free of missing data, which could otherwise skew the results.

```
# Remove rows with NA on Normalised indices
Combined_table <- Combined_table |>
  drop_na(Norm_VDem)|>
  drop_na(Norm_Econ)|>
  drop_na(Norm_Bert)
```

Step 8: Plot the Data

Objective:

- Create a scatter plot to visualise the relationship between GDP per capita and the average democracy score. The plot should:
- Display GDP per capita on the x-axis.
- Show the average democracy score on the y-axis.
- Differentiate between the three indices (`Norm_VDem`, `Norm_Econ`, and `Norm_Bert`) using distinct colours for each index.

1. Transform Data with `pivot_longer`:

- To effectively compare the indices, reshape the data into a long format using the `pivot_longer` function from the `tidyverse` package. This transformation allows for easy differentiation between indices in the plot (`Norm_VDem` , `Norm_Econ` and `Norm_Bert`).

```
# Convert the Combined_table from wide to long format
plot_data <- pivot_longer(
  Combined_table,
  cols = c(Norm_VDem, Norm_Econ, Norm_Bert),
  names_to = "Index",
  values_to = "Score"
)
```

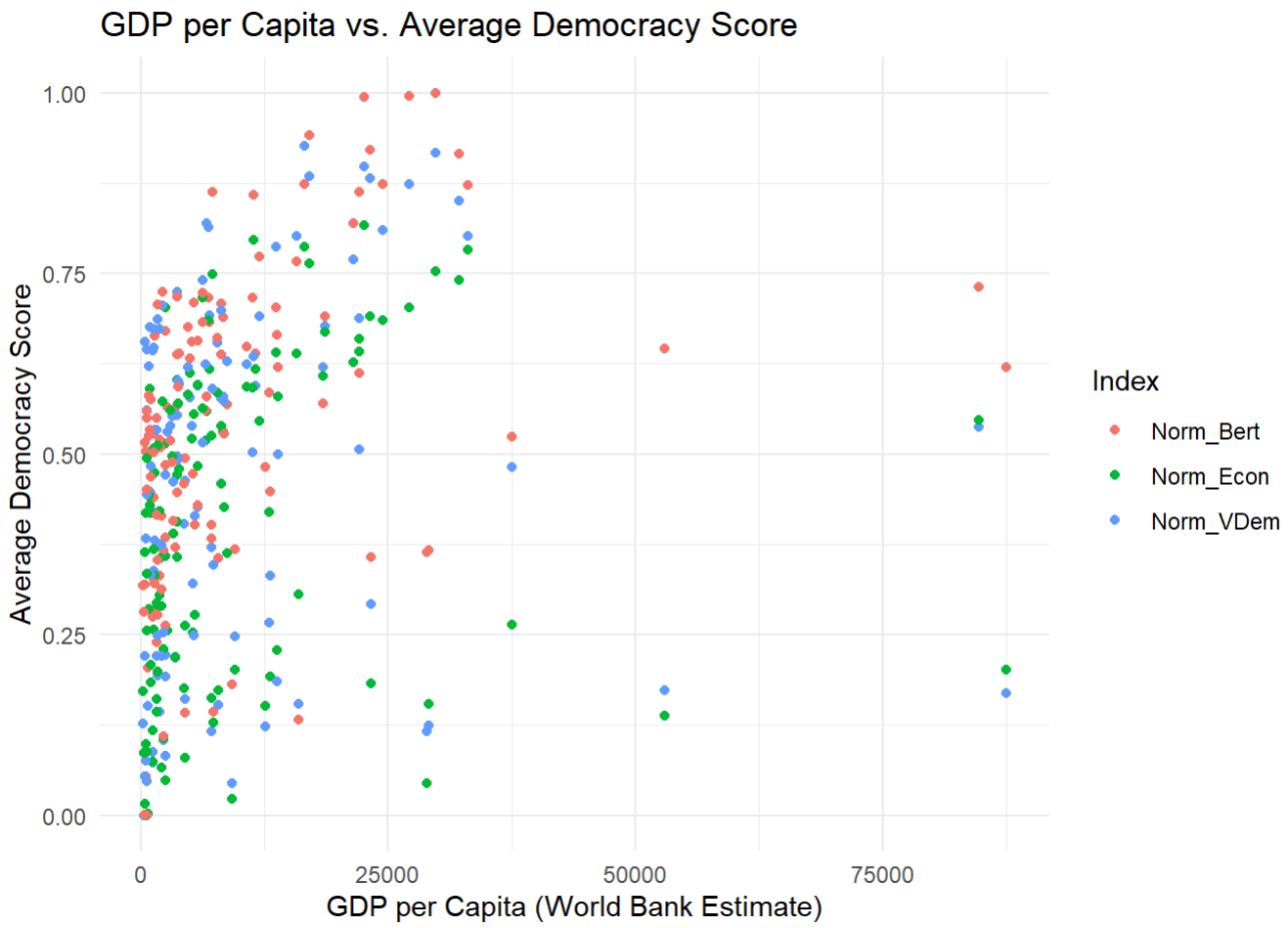
- `cols = c(Norm_VDem, Norm_Econ, Norm_Bert)` : Specifies the columns to be transformed from wide format to long format. These columns contain the normalised scores for each index.
- `names_to = "Index"` : Creates a new column named `Index` that will hold the names of the original columns (i.e., `Norm_VDem`, `Norm_Econ` and `Norm_Bert`).
- `values_to = "Score"` : Creates a new column named `Score` that contains the values from the specified columns. This column will represent the average democracy scores for each index.

2. Plotting the data:

Use `ggplot` from the `ggplot2` package to create the scatter plot to visualise the relationship between GDP per capita and the average democracy score across the three indices: `Norm_VDem`, `Norm_Econ`, and `Norm_Bert`. The data is sourced from the `plot_data` object, which is nowin a long format to facilitate colour coding by index type:

Please Note: GDP per capita is represented by using estimates from the World Bank. This choice is based on the assumption that no average of GDP per capita estimates from multiple sources (IMF, World Bank, UN) are required.

```
ggplot(plot_data, aes(x = World_Bank_Estimate, y = Score, color = Index)) +
  geom_point() +
  labs(
    title = "GDP per Capita vs. Average Democracy Score",
    x = "GDP per Capita (World Bank Estimate)",
    y = "Average Democracy Score",
    color = "Index"
  ) +
  theme_minimal()
```



- `aes(x = World_Bank_Estimate, y = Score, color = Index)`: Maps GDP per capita to the x-axis, average democracy score to the y-axis, and differentiates data points by the `Index` column using colour.
- `geom_point()`: Adds scatter plot points to visualise the data.
- `labs()`: Sets the title and axis labels, and provides a legend for the color coding.
- `theme_minimal()`: Applies a clean and minimal theme to the plot, ensuring that the focus remains on the data.

Outcome:

The resulting scatter plot illustrates how different democracy indices relate to GDP per capita. By using distinct colours for each index, the plot provides a clear and organised view of how the average democracy scores (across the V-Democracy Index, Economist Democracy Index, and Bertelsmann Transformation Index) vary with GDP per capita, revealing any patterns or relationships between these variables.

Result:

Based on the plot, the results show a general trend where countries with a higher GDP per capita also tend to have higher democracy scores. However, this relationship is not perfectly linear or universal. Here are a few observations:

1. **Clustered Data:** Many data points are clustered in the lower GDP per capita range (0 - 25,000 USD). Within this range, there is a wide variation in democracy scores, indicating that countries with

similar GDP per capita can have very different levels of democracy.

2. **High Democracy Scores at Higher GDP**: As GDP per capita increases, especially beyond 25,000 USD, the democracy scores tend to be higher (closer to 1.0). This suggests that wealthier countries are more likely to have higher-quality democracies.

3. **Outliers**: There are a few outliers where countries with higher GDP per capita have lower democracy scores. Similarly, some countries with lower GDP per capita have relatively high democracy scores.

4. **Different Indices**: The plot depicts data points from different indices (Norm_Bert, Norm_Econ, Norm_VDem), all indicating a similar pattern of increasing democracy scores with higher GDP per capita, though with slight variations between them.

Thus, while higher GDP per capita seems to be associated with better democracy quality on average, the plot also indicates variation and exceptions to this trend. Therefore, while GDP per capita is an indicator, it may not be the sole determinant of democracy quality. Further analysis, including univariate outlier detection and linear regression analysis may be required to more accurately assess the significance of this relationship.

Conclusion:

This guide outlines a thorough approach to collecting, processing, and analysing data from multiple sources. The process begins with extracting data from Wikipedia using web scraping, followed by parsing and structuring the data into a manageable format. The use of an SQLite database facilitates organised storage and efficient querying of the combined datasets. Normalising the indices ensures consistency across different measures, allowfiling for accurate comparison. Finally, visualising the relationship between GDP per capita and various democracy indices through a scatter plot reveals insightful patterns and correlations. This approach demonstrates a structured approach for data integration and analysis, ultimately providing a clearer understanding of the interplay between economic and democratic indicators.