Summative Assessment Section A

Keli Niu (ad21083)

2024-11-03

1. (Q1) Data Loading and Exploration

Step 1: Loading Required Libraries

Before loading the data, we need to load the necessary library.

```
# Load Tidyverse library
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflic
ts to become errors
```

Step 2: Load CSV Files into Data Frames

We need to load the three datasets (`debt_data.csv` , `country_data.csv` , and `indicator_data.csv`) into data frames called `debt_df` , `country_df` , and `indicator_df` , respectively.

```
# Load the CSV files into data frames
debt_df <- read_csv("debt_data.csv")
```

```
## Rows: 13824 Columns: 63
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (2): Country.Code, Year
## dbl (61): NY.GNP.MKTP.CD, FI.RES.TOTL.MO, FI.RES.TOTL.DT.ZS, FI.RES.TOTL.CD,...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
country_df <- read_csv("country_data.csv")
```

```
## Rows: 216 Columns: 5
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (5): Country.Code, Region, IncomeGroup, SpecialNotes, Country.Name
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
indicator_df <- read_csv("indicator_data.csv")
```

```
## Rows: 61 Columns: 2
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): INDICATOR_CODE, INDICATOR_NAME
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Step 3: Checking the Dimensions of `debt_df`

```
# Check the number of columns and rows in debt_df
num_rows <- nrow(debt_df)
num_cols <- ncol(debt_df)

# Display the results
cat("Number of Rows in debt_df: ", num_rows, "\n")
```

```
## Number of Rows in debt_df:  13824
```

```
cat("Number of Columns in debt_df: ", num_cols, "\n")
```

```
## Number of Columns in debt_df:  63
```

Explanation

- `read_csv()` is used to read CSV files.
- `cat()` and `print()` are both used to print results, but `print()` can only display one value at a time, whereas `cat()` can output multiple values, similar to Python’s `print()` function.

2 (Q2) Update by Reordering

Step 1: Reordering Rows by Indicator “DT.NFL.BLAT.CD”

```
#Reorder rows by the indicator "DT.NFL.BLAT.CD" in descending order
debt_df <- debt_df %>%
  arrange(desc(DT.NFL.BLAT.CD))
```

Step 2: Select specific columns

```
debt_subset <- debt_df%>%
  select(Country.Code, Year, NY.GNP.MKTP.CD, DT.NFL.BLAT.CD) %>%
  # selects the first 4 rows
  slice(1:4)

print(debt_subset)
```

```
## # A tibble: 4 × 4
##   Country.Code Year      NY.GNP.MKTP.CD DT.NFL.BLAT.CD
##   <chr>        <chr>              <dbl>          <dbl>
## 1 MEX          year_1995   366827000000     9398190731
## 2 EGY          year_2013   281028000000     7233642176
## 3 BRA          year_2017  2024940000000     6506490468
## 4 PAK          year_2018   350691000000     6201281870
```

Explanation

- `arrange()` : This function is used to sort rows of a data frame based on the values of one or more columns.
- `desc(DT.NFL.BLAT.CD)` : The `desc()` function is used to sort the values in the specified column in descending order.
- `select()` : Chooses specific columns (`Country.Code` , `Year` , `NY.GNP.MKTP.CD` , `DT.NFL.BLAT.CD`) that we want to focus on. Besides, if we want to remove certain columns from the data, we can use the `-` symbol before the column names.
- `slice(1:4)` : Selects the first 4 rows after sorting. ***Note:*** In this task, `head()` could also be used to get the first few rows. However, `slice()` is more flexible because it allows selecting rows from anywhere in the dataset, not just the top.

3 (Q3) Combining Data Frames to Add Indicator Names

Step 1: Observation data format

```
print(head(debt_df,5))
```

```
## # A tibble: 5 × 63
##   Country.Code Year      NY.GNP.MKTP.CD FI.RES.TOTL.MO FI.RES.TOTL.DT.ZS
##   <chr>        <chr>              <dbl>          <dbl>             <dbl>
## 1 MEX          year_1995   366827000000           2.83              10.2
## 2 EGY          year_2013   281028000000           2.73              35.5
## 3 BRA          year_2017  2024940000000          14.9               68.9
## 4 PAK          year_2018   350691000000           1.91              11.8
## 5 EGY          year_2016   327970000000           3.89              34.2
## # ℹ 58 more variables: FI.RES.TOTL.CD <dbl>, DT.TDS.MLAT.PG.ZS <dbl>,
## #   DT.TDS.MLAT.CD <dbl>, DT.TDS.DPPG.XP.ZS <dbl>, DT.TDS.DPPG.GN.ZS <dbl>,
## #   DT.TDS.DPPG.CD <dbl>, DT.TDS.DPPF.XP.ZS <dbl>, DT.TDS.DIMF.CD <dbl>,
## #   DT.TDS.DECT.GN.ZS <dbl>, DT.TDS.DECT.EX.ZS <dbl>, DT.TDS.DECT.CD <dbl>,
## #   DT.ODA.ODAT.PC.ZS <dbl>, DT.ODA.ODAT.GN.ZS <dbl>, DT.ODA.ODAT.CD <dbl>,
## #   DT.NFL.RDBN.CD <dbl>, DT.NFL.RDBC.CD <dbl>, DT.NFL.PRVT.CD <dbl>,
## #   DT.NFL.PROP.CD <dbl>, DT.NFL.PNGC.CD <dbl>, DT.NFL.PNGB.CD <dbl>, …
```

```
print(head(indicator_df,5))
```

```
## # A tibble: 5 × 2
##   INDICATOR_CODE    INDICATOR_NAME
##   <chr>             <chr>
## 1 NY.GNP.MKTP.CD    GNI (current US$)
## 2 FI.RES.TOTL.MO    Total reserves in months of imports
## 3 FI.RES.TOTL.DT.ZS Total reserves (% of total external debt)
## 4 FI.RES.TOTL.CD    Total reserves (includes gold, current US$)
## 5 DT.TDS.MLAT.PG.ZS Multilateral debt service (% of public and publicly guarant…
```

Step 2: Combine `debt_df` and `indicator_df` to create `debt_df2` with indicator names instead of codes

```
# By observing the data, we notice that in `debt_df`, the indicators are represented
by their codes. Because of that, we will merge `debt_df` with `indicator_df` to repla
ce the indicator codes with their respective names.

# The debt_df has been sorted in descending order
debt_df2 <- debt_df %>%
  pivot_longer(cols = -c(Country.Code, Year), names_to = "Indicator.Code", values_to
= "Value") %>%
  left_join(indicator_df, by = c("Indicator.Code" = "INDICATOR_CODE")) %>%
  select(-Indicator.Code) %>%
  pivot_wider(names_from = INDICATOR_NAME, values_from = Value)
```

Step 3: Error handling

```
# Check if row count matches after merge
if (nrow(debt_df) != nrow(debt_df2)) {
  warning("Row count mismatch after merging. Some data may have been lost during the
join.")
}else{
    print("Merger successful")
  }
```

```
## [1] "Merger successful"
```

Step 4: Display the result

```
# Display a subset of debt_df2 consisting of the first 5 rows and specific columns
debt_subset2 <- debt_df2 %>%
  select(Country.Code, Year, `Net financial flows, others (NFL, current US$)`) %>%
  slice(1:5)

print(debt_subset2)
```

```
## # A tibble: 5 × 3
##   Country.Code Year      `Net financial flows, others (NFL, current US$)`
##   <chr>        <chr>                                                <dbl>
## 1 MEX          year_1995                                              NA
## 2 EGY          year_2013                                       -14314777.
## 3 BRA          year_2017                                      -195705180.
## 4 PAK          year_2018                                       321846510.
## 5 EGY          year_2016                                      2141976215
```

Explanation

- `pivot_longer()` : Converts the wide format of `debt_df` into a long format so that each indicator code is represented as a separate row, making it easier to join with `indicator_df` .
- `left_join()` : Merges `debt_df` with `indicator_df` by matching `Indicator.Code` to add the indicator names. This keeps all rows from `debt_df` and adds relevant columns from `indicator_df` .
- `pivot_wider()` : Converts the long format back to a wide format, where each indicator name becomes a separate column.

4 (Q4) Combining Data Frames to Add Country Information

Step 1: Merge `debt_df2` with `country_df` to Create `debt_df3`

```
# Merge debt_df2 with country_df to add information about each country's region, inco
me group, and name.
debt_df3 <- debt_df2 %>%
  left_join(country_df %>%
              select(Country.Code, Region, IncomeGroup, Country.Name), by = "Country.
Code")
```

Step 2: Error handling

```
# Check if row count matches after merge
if (nrow(debt_df2) != nrow(debt_df3)) {
  warning("Row count mismatch after merging. Some data may have been lost during the
join.")
}else{
    print("Merger successful")
}
```

```
## [1] "Merger successful"
```

Step 3: Display the result

```
# Display a subset of debt_df3 consisting of the first 3 rows and specific columns
debt_subset3 <- debt_df3 %>%
  select(Country.Name, IncomeGroup, Year, `Total reserves in months of imports`)

print(head(debt_subset3,3))
```

```
## # A tibble: 3 × 4
##   Country.Name     IncomeGroup         Year      Total reserves in months of i…¹
##   <chr>            <chr>               <chr>                               <dbl>
## 1 Mexico           Upper middle income year_1995                            2.83
## 2 Egypt, Arab Rep. Lower middle income year_2013                            2.73
## 3 Brazil           Upper middle income year_2017                           14.9 
## # ℹ abbreviated name: ¹`Total reserves in months of imports`
```

Explanation

- `left_join()` : Merges `debt_df2` with `country_df` by their common `Country.Code` to add information such as `Region` , `IncomeGroup` , and `Country.Name` .

5 (Q5) Renaming Columns

Step 1: Rename Columns in debt_df3

```
# Rename columns in debt_df3
debt_df3 <- debt_df3 %>%
  rename(
    Total_reserves = `Total reserves in months of imports`,
    External_debt = `External debt stocks, total (DOD, current US$)`,
    Financial_flow = `Net financial flows, bilateral (NFL, current US$)`,
    Imports = `Imports of goods, services and primary income (BoP, current US$)`,
    IFC = `IFC, private nonguaranteed (NFL, US$)`
  )
```

Step 2: Display the result

```
print(head(debt_df3, 3))
```

```
## # A tibble: 3 × 66
##   Country.Code Year    `GNI (current US$)` Total_reserves Total reserves (% of…¹
##   <chr>        <chr>                 <dbl>          <dbl>                  <dbl>
## 1 MEX          year_1…        366827000000           2.83                   10.2
## 2 EGY          year_2…        281028000000           2.73                   35.5
## 3 BRA          year_2…       2024940000000          14.9                    68.9
## # ℹ abbreviated name: ¹`Total reserves (% of total external debt)`
## # ℹ 61 more variables: `Total reserves (includes gold, current US$)` <dbl>,
## #   `Multilateral debt service (% of public and publicly guaranteed debt service)`
<dbl>,
## #   `Multilateral debt service (TDS, current US$)` <dbl>,
## #   `Public and publicly guaranteed debt service (% of exports of goods, services
and primary income)` <dbl>,
## #   `Public and publicly guaranteed debt service (% of GNI)` <dbl>,
## #   `Debt service on external debt, public and publicly guaranteed (PPG) (TDS, cur
rent US$)` <dbl>, …
```

Explanation

- `rename()` : Can rename the name of a specified column in the dataframe.

6 (Q6) Generating Summary Data Frame

Step 1: Generate Summary Data Frame

```
#Create a summary data frame `debt_summary` from `debt_df3` with statistics grouped b
y region.
debt_summary <- debt_df3 %>%
  group_by(Region) %>%
  summarise(
    TR_mn = mean(Total_reserves, na.rm = TRUE),
    ED_md = median(External_debt, na.rm = TRUE),
    FF_quantile = quantile(Financial_flow, probs = 0.2, na.rm = TRUE),
    IFC_sd = sd(IFC, na.rm = TRUE)
  )
```

Step 2: Display the result

```
print(debt_summary)
```

```
## # A tibble: 7 × 5
##   Region                     TR_mn      ED_md FF_quantile    IFC_sd
##   <chr>                      <dbl>      <dbl>       <dbl>     <dbl>
## 1 East Asia & Pacific         5.19 2248479410   -2357020. 52498519.
## 2 Europe & Central Asia       3.58 8237728122  -53631246. 50820255.
## 3 Latin America & Caribbean   3.84 4159662669  -25144268. 62054545.
## 4 Middle East & North Africa  7.72 7481954468  -92269932. 21414719.
## 5 North America               1.99         NA          NA        NA
## 6 South Asia                  4.94 4940329805    -373253. 76630044.
## 7 Sub-Saharan Africa          3.32 1709094992   -1673594. 24748455.
```

```
# Note: The NA values in the summary result are due to North America having all NA va
lues for External_debt, Financial_flow, and IFC, so even with na.rm = TRUE, the resul
t remains NA.
```

Explanation

- `group_by(Region)` : Groups the data by `Region` , so that calculations are performed for each region separately.
- `summarise()` : Calculates summary statistics for each group.
  - `TR_mn` : The average of `Total_reserves` for each region, ignoring missing values (`na.rm = TRUE`).
  - `ED_md` : The median of `External_debt` for each region, ignoring missing values.
  - `FF_quantile` : The 0.2 quantile of `Financial_flow` for each region, ignoring missing values.
  - `IFC_sd` : The standard deviation of `IFC` for each region, ignoring missing values.

7 (Q7) Creating Violin Plot

Create a violin plot of `Financial_flow` for each of the regions, filtering out extreme values and missing values.

Step 1: Organise the data required for the plotting

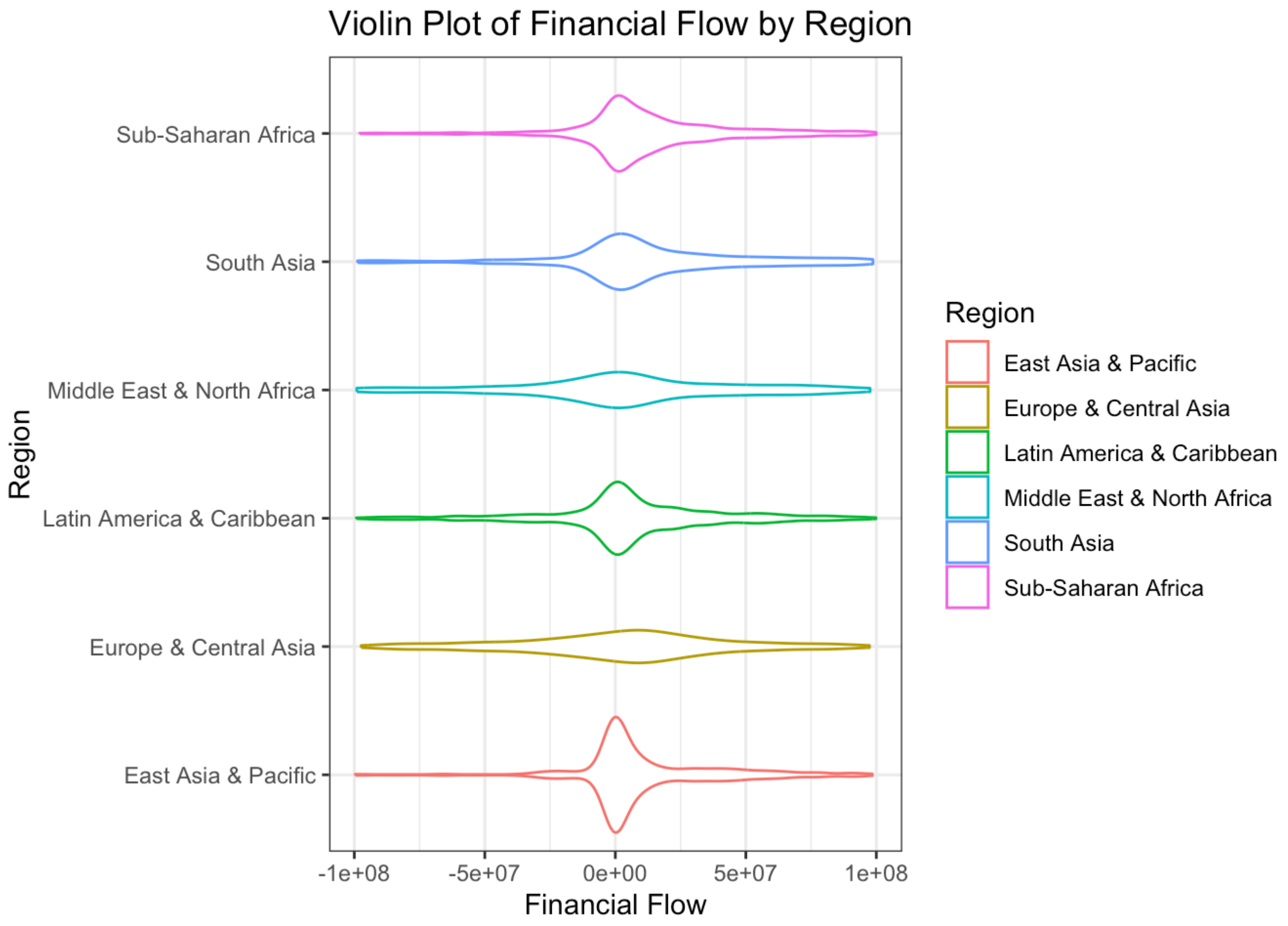
```
# Load ggplot2 library
library(ggplot2)

# Filter data for valid Financial_flow values and remove missing values
filtered_df <- debt_df3 %>%
  filter((Financial_flow > -1e8 & Financial_flow < 1e8), !is.na(Financial_flow))
```

Step 2: Create the violin plot

```
# Create violin plot
violin_plot <- ggplot(filtered_df, aes(x = Financial_flow, y = Region, color = Regio
n)) +
  geom_violin() +
  labs(title = "Violin Plot of Financial Flow by Region", x = "Financial Flow", y =
"Region") +
  theme_bw()

# Display the plot
print(violin_plot)
```



Explanation

- `filter()` : Filters the data between -10^8 and 10^8, and removes missing values.
- `ggplot()` : Initializes a ggplot object for visualization.
- `aes()` : Specifies the aesthetics, mapping `Financial_flow` to the x-axis and `Region` to the y-axis, with different colors for each region.
- `geom_violin()` : Creates the violin plot.
- `labs()` : Adds titles and labels to the plot.
- `theme_bw()` : Applies a black and white theme to make the plot cleaner.

8 (Q8) Creating Plot for Total Reserves

Create a plot of `Total_reserves` over the years (1960-2023) for specific countries, faceted by income group.

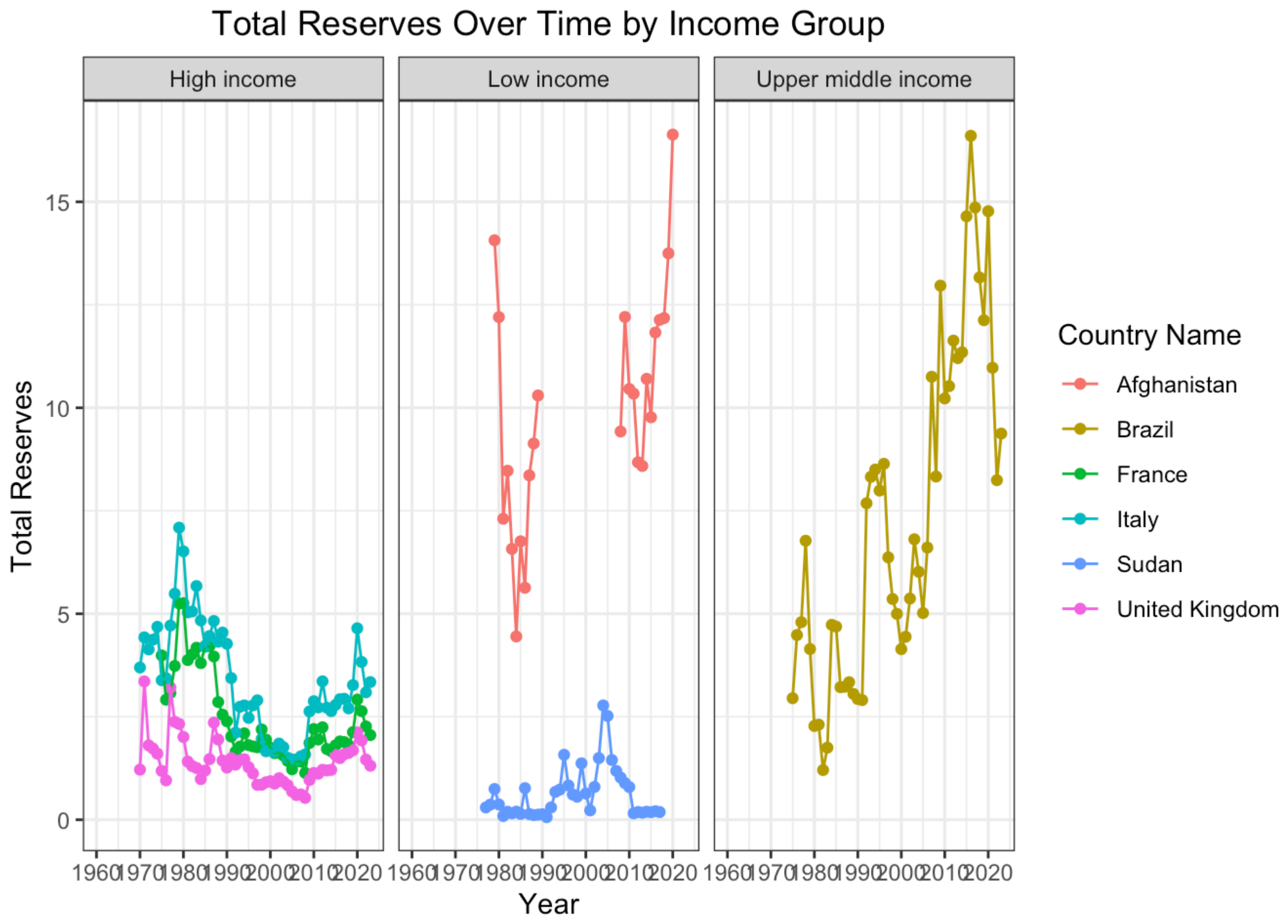
Step 1: Filter data

```
# Filter data for selected countries
filtered_reserves_df <- debt_df3 %>%
  filter(Country.Name=="Italy"|Country.Name=="France"|Country.Name=="United Kingdom"|
Country.Name=="Sudan"|Country.Name=="Afghanistan"|Country.Name=="Brazil")%>%
# Converts the year parameter to a numeric type for easy image plotting.
  mutate(Year = as.numeric(gsub("year_", "", Year)))
```

Step 2: Create plot

```
# Display the plot
reserves_plot <- ggplot(filtered_reserves_df, aes(x = Year, y = Total_reserves, color
= Country.Name, group = Country.Name)) +
  geom_point(na.rm = TRUE) +
  geom_line(na.rm = TRUE) +
  # Here you can add `scales = ‘free’` to make the image avoid some scale inconsisten
cies.
  facet_wrap(~IncomeGroup) +
  theme_bw() +
  #Setting the horizontal coordinates to display at 10-year intervals facilitates the
visualisation of the image.
  scale_x_continuous(breaks = seq(1960, 2023, by = 10)) +
  labs(title = "Total Reserves Over Time by Income Group", x = "Year", y = "Total Res
erves", color = "Country Name")+
  #Centering the title
  theme(plot.title = element_text(hjust = 0.5))

print(reserves_plot)
```



Explanation

- `mutate()` is used to modify or create new columns in a data frame. Here, it is converting the `Year` column from a character to a numeric format.
- `geom_point()` and `geom_line()` : Plots points and lines to show the trends in `Total_reserves` over the years.
- `facet_wrap(~IncomeGroup)` : Facets the plot by `IncomeGroup` , allowing each income group to have its own panel.