test

Loading libraries

For this project, we ended up using a number of packages from RStudio that significantly enhanced our data analysis and visualization capabilities. Each package serves a specific purpose and collectively forms a powerful toolkit for R programming. Let's briefly explore the functionalities of some of the key packages we utilized:

- **tidyverse**: Collection of packages for data manipulation and visualization.
- **eurostat**: Facilitates retrieval of statistical data from Eurostat database.
- **magrittr**: Provides the pipe operator %>% for more readable code.
- **httr**: Handles HTTP requests and responses, useful for web APIs.
- **rjstat**: Works with data in the JSON-stat format.
- **rlang**: Provides functions for working with language objects.
- **colorspace**: Manipulates and converts color spaces in R.
- **imputeTS**: Offers imputation methods for time series data.
- **zoo**: Handles irregular time series data with the zoo class.
- **shiny**: Allows building interactive web applications in R.
- **shinydashboard**: Specializes in creating attractive dashboards.
- **plotly**: Creates interactive visualizations in R.
- **RColorBrewer**: Provides color palettes for appealing plots.

```
pacman::p_load(tidyverse, eurostat, magrittr, httr, rjstat, rlang, colorspace, imputeTS, z
```

Downloading data

Downloading and importing data is a critical step in any data analysis project. In this section, we'll explain how we retrieved the necessary data from Eurostat's database and transformed it into a dataframe for further analysis.

Before diving into the code, we identified the specific data requirements for our project. We needed economic indicators, such as "Value added at factor cost - million euro" (V12150) and

"Persons employed - number" (V16110), for a set of countries and NACE industry classifications. To access this data, we utilized Eurostat's query builder that to give us the desired indicators, countries, and time period. We additionally tested out the Eurostat API, but in the end we ended up using the Eurostat query builder.

```
## Importing json file and saving it as df
df <- fromJSONstat("Eurostat SBS Data.json")

# Removing scientific number format
options(scipen=999)
```

Data Cleaning and Transformation

In the "Wrangling Data" section, we focused on cleaning and reorganizing the data in the dataframe df1 to create a new and more structured dataframe named df2. This process involved several steps to ensure that the data was in a more suitable format for analysis and visualization.

In this step, we performed data cleaning and transformation tasks to prepare the data for analysis. Here's a summary of the actions taken:

- **Removing Unnecessary Columns:** We removed the columns "Time frequency" and "Size classes in number of persons employed" as they contained redundant information using the distinct() function.
- **Renaming Columns:** For clarity and consistency, we renamed certain columns using the rename() function.
- **Handling Fake Duplicates:** To address fake duplicates in the data, we kept only one entry for each unique combination of "category," "country," and "year" using the distinct() function with the .keep_all = TRUE argument.
- **Transforming Data with pivot_wider():** We used the pivot_wider() function to create new columns for different indicators, such as "value_added" and "employed," reducing the number of rows in the dataframe.
- **Fixing Year Format:** We converted the "year" column from characters to a numeric format using the as.numeric() function.
- **Setting Negative Value Added to Zero:** Negative values in the "value_added" column were replaced with zeros using the mutate() function and the ifelse() statement.

These steps resulted in a cleaned and structured dataframe, **df2** ready for further analysis and visualization.

```
### Removing and renaming colomns
# Removing duplicates
df1 <- df %>%
  distinct()
df1$`Time frequency` <- NULL
df1$`Size classes in number of persons employed` <- NULL
df1 <- df1 %>%
  rename(category = "Statistical classification of economic activities in the European Com
  indicator = "Economical indicator for structural business statistics",
  country = "Geopolitical entity (reporting)",
  year = "Time",
  value_added1 = "value")

### Find and remove fake duplicates with one NA and one non NA value
df1 <- df1 %>% distinct(category, country, year, indicator, .keep_all = TRUE)


### Make the indicators into new Columns to cut down number of observations
df2 <- df1 %>%
  pivot_wider(names_from = indicator, values_from = value_added1)

df2 <- df2 %>%
  rename(value_added = "Value added at factor cost - million euro",
  employed = "Persons employed - number")

# Fix year to be numeric
df2 <- df2 %>%
  mutate(year = as.numeric(year))

# Set negative value added to equal 0
df2 <- df2 %>%
  mutate(value_added = ifelse(value_added < 0, 0, value_added))
```

Handling Missing Values - Interpolation

In our dataset, we encountered missing values in the "value_added" and "employed" columns for certain years and categories. The years 2005 to 2008 had a significantly higher number of missing observations compared to other years. We needed to address these missing values before proceeding with our analysis to ensure the accuracy and completeness of the data.

To handle missing values, we explored various methods but encountered limitations with each:

- **Mice and missForest:** Require a substantial amount of complete data, not suitable for our dataset with numerous missing observations.
- **Linear Regression::** Assumes linearity and may not be robust to outliers or non-linear trends.
- **Mean and Median::** Simple imputation may lead to biased estimates and overlook variations between categories and countries.

As a better approach, we chose two interpolation methods:

- **na.locf (Last Observation Carried Forward):** : Imputes missing values with the last known non-missing value within the same category and country, suitable for sequences of missing values.
- **na.approx:** : Performs linear interpolation between non-missing values, estimating missing observations based on a roughly linear trend.

We used these interpolation techniques to create the cleaned dataframe **df3**, which now contains more complete data while preserving overall trends. This dataset is now ready for further analysis and visualization in our project. It also keeps track if a value has been edited through the Yes/No variable estimated.

```
### Interpolation see the trend
#Lising what nations to be kept
comparable_countries_full <- c("Norway", "Finland", "Sweden", "Denmark", "Luxembourg", "Ne

#comparable_countries_full <- c("Norway")

# Create new dataset keeping only comparable countries
df3 <- df2 %>%
  filter(country %in% comparable_countries_full)

# Count number of NAs in 'value_added' for each year
df3_na_counts <- df3 %>%
  group_by(year) %>%
  summarize(na_count = sum(is.na(value_added)))
# Here we see that there is a significant higher amount of NA observations in 2005-2007. A

# Filtering data for years after 2007, removing 'value_per_employee' column,
# converting 'category' and 'country' into factors, and creating NA markers
df3 <- df3 %>%
  filter(year > 2007) %>%
  mutate(
```

```r
    category = as.factor(category),
    country = as.factor(country),
    value_added_na = ifelse(is.na(value_added), 1, 0),
    employed_na = ifelse(is.na(employed), 1, 0),
    estimated = ifelse(value_added_na == 1 | employed_na == 1, "Yes", "No")
  )

# Grouping by 'country' and 'category', sorting by 'year',
# filling NA with linear approximations or the last non-NA (from the right)
df3 <- df3 %>%
  group_by(country, category) %>%
  arrange(year) %>%
  mutate(
    value_added = ifelse(is.na(value_added), na.approx(value_added, na.rm = FALSE), value_
    employed = ifelse(is.na(employed), na.approx(employed, na.rm = FALSE), employed),
    value_added = ifelse(is.na(value_added), na.locf(value_added, fromLast = TRUE), value_
    employed = ifelse(is.na(employed), na.locf(employed, fromLast = TRUE), employed)
  ) %>%
  ungroup()
```

Adding Letter Codes to Dataset