National University of Sciences and Technology (NUST)

College of Electrical and Mechanical Engineering (CEME)

Department: Mechatronics Engineering

Cs#117 Application of ICT

Project title:

"Global Economic Dynamics"

Group members:

Muhammad Salman (541313)

Shabbir Haider (570137)

Aalyan Maqsood (547280)

Submitted to:

LE KASHAN ANSARI

Overview of the Project:

1. Executive Summary

This project analyzes a comprehensive multi-country economic dataset spanning 1970–2022. The workflow includes advanced data cleaning, exploratory data analysis (EDA) of global trends, predictive modeling using Linear Regression to forecast GDP through 2030, and final deployment to a cloud-based Supabase environment.

2. Dataset Scope &amp; Preparation

The project utilizes 'global_economy_indicators.csv', containing records for over 200 countries. Key preprocessing steps included:

• Standardizing column headers and the 'Year'; column.

• Handling missing values via interpolation for time-series consistency. • Mapping countries to continents for regional comparative analysis.

3. Exploratory Data Analysis (EDA)

Analysis focused on identifying the world's leading economic contributors (USA, Japan, China) and calculating the Annual GDP Growth Rate (%) across continents from 2001–2017.

4. Predictive Modeling &amp; Forecasting

The study implemented Scikit-learn Linear Regression models to project metrics up to 2030:

• Log-Linear Regression for GDP to capture exponential growth trends.

• Linear Regression for Population trends.

• Derivation of Predicted GDP Per Capita to estimate future individual economic welfare.

5. Cloud Integration

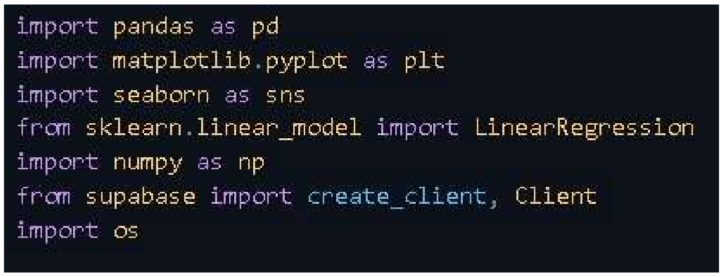
For final deployment, a Supabase cloud storage system was integrated. Automated Python scripts were used to authenticate and stream finalized datasets and visualizations to a secure cloud bucket.

Here are different contents of the following topic which we have covered in the project:

Data modules:

We use different modules to ensure our data is fully calculated and result is fully accurate and with the help of the following modules we can generate these graphs and calculations,

Python:



```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
import numpy as np
from supabase import create_client, Client
import os
```

Data analysis and cleaning:

1. Defining scope and performing initial refinement

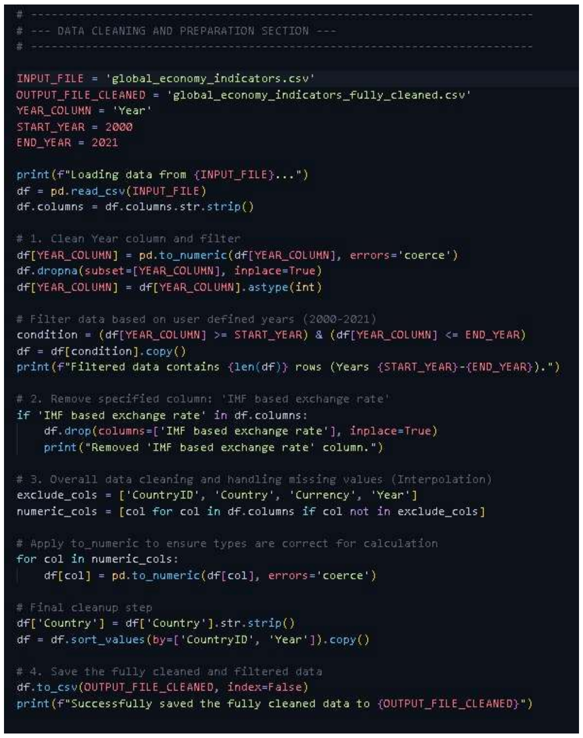
- The process begins by establishing the analytical timeframe and conducting preliminary data cleaning.
- The dataset **global_economy_indicators.csv** is loaded, the **Year** column is standardized, and the data is restricted to **2000–2021**.
- Non-essential variables, such as the **“IMF based exchange rate”**, are removed to maintain analytical relevance.

2. Enhancing data quality through missing-value treatment

- Missing values in numeric fields are addressed using **interpolation**, preserving the continuity of time-series patterns.
- Categorical identifiers (**CountryID, Country, Currency, Year**) are excluded from imputation due to their non-numeric nature.
- The data set is then reduced to the final set of clean and relevant indicators.

3. Structuring and exporting the final dataset

- Records are sorted by **CountryID** and **Year** to ensure analytical coherence.
- The final cleaned dataset is exported as **global_economy_fully_cleaned.csv**, providing a reliable foundation for subsequent analysis.



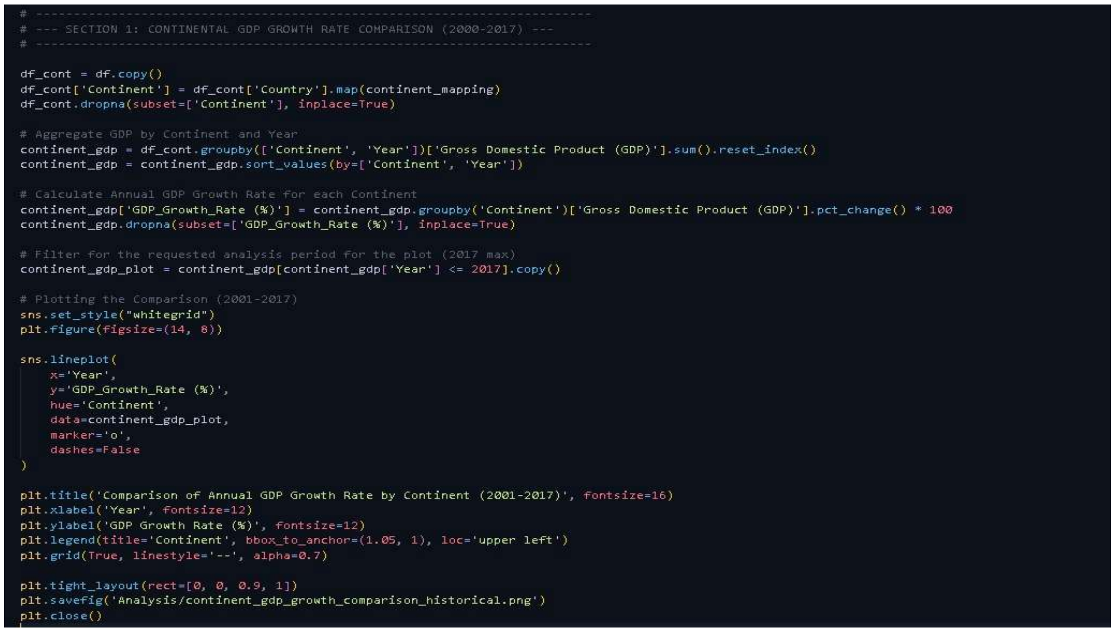
```python
# ----------------------------------------------------------------------
# --- DATA CLEANING AND PREPARATION SECTION ---
# ----------------------------------------------------------------------

INPUT_FILE = 'global_economy_indicators.csv'
OUTPUT_FILE_CLEANED = 'global_economy_indicators_fully_cleaned.csv'
YEAR_COLUMN = 'Year'
START_YEAR = 2000
END_YEAR = 2021

print(f"Loading data from {INPUT_FILE}...")
df = pd.read_csv(INPUT_FILE)
df.columns = df.columns.str.strip()

# 1. Clean Year column and filter
df[YEAR_COLUMN] = pd.to_numeric(df[YEAR_COLUMN], errors='coerce')
df.dropna(subset=[YEAR_COLUMN], inplace=True)
df[YEAR_COLUMN] = df[YEAR_COLUMN].astype(int)

# Filter data based on user defined years (2000-2021)
condition = (df[YEAR_COLUMN] >= START_YEAR) & (df[YEAR_COLUMN] <= END_YEAR)
df = df[condition].copy()
print(f"Filtered data contains {len(df)} rows (Years {START_YEAR}-{END_YEAR}).")

# 2. Remove specified column: 'IMF based exchange rate'
if 'IMF based exchange rate' in df.columns:
    df.drop(columns=['IMF based exchange rate'], inplace=True)
    print("Removed 'IMF based exchange rate' column.")

# 3. Overall data cleaning and handling missing values (Interpolation)
exclude_cols = ['CountryID', 'Country', 'Currency', 'Year']
numeric_cols = [col for col in df.columns if col not in exclude_cols]

# Apply to_numeric to ensure types are correct for calculation
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Final cleanup step
df['Country'] = df['Country'].str.strip()
df = df.sort_values(by=['CountryID', 'Year']).copy()

# 4. Save the fully cleaned and filtered data
df.to_csv(OUTPUT_FILE_CLEANED, index=False)
print(f"Successfully saved the fully cleaned data to {OUTPUT_FILE_CLEANED}")
```

Data comparison:

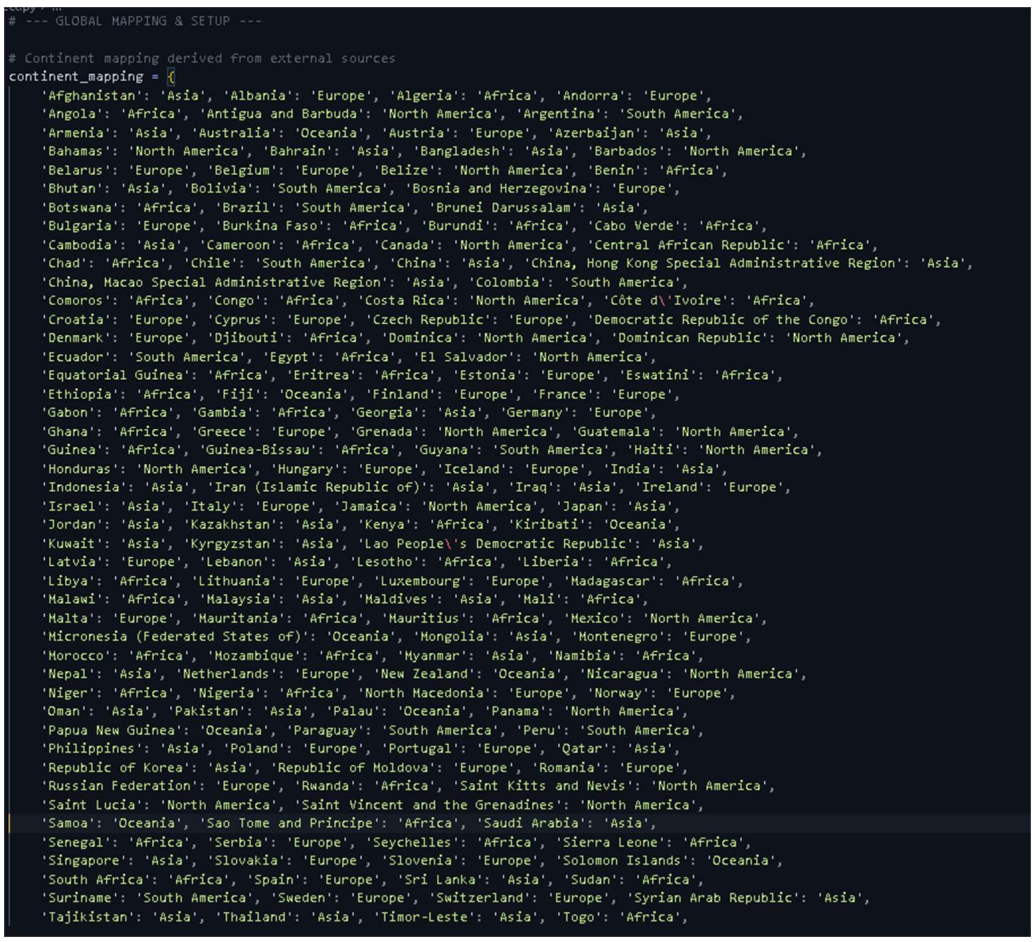
Our work involved a structured two-stage process: first, preparing the economic data, and second, performing a comparative analysis of continental GDP growth.

1. Continental GDP Growth Analysis

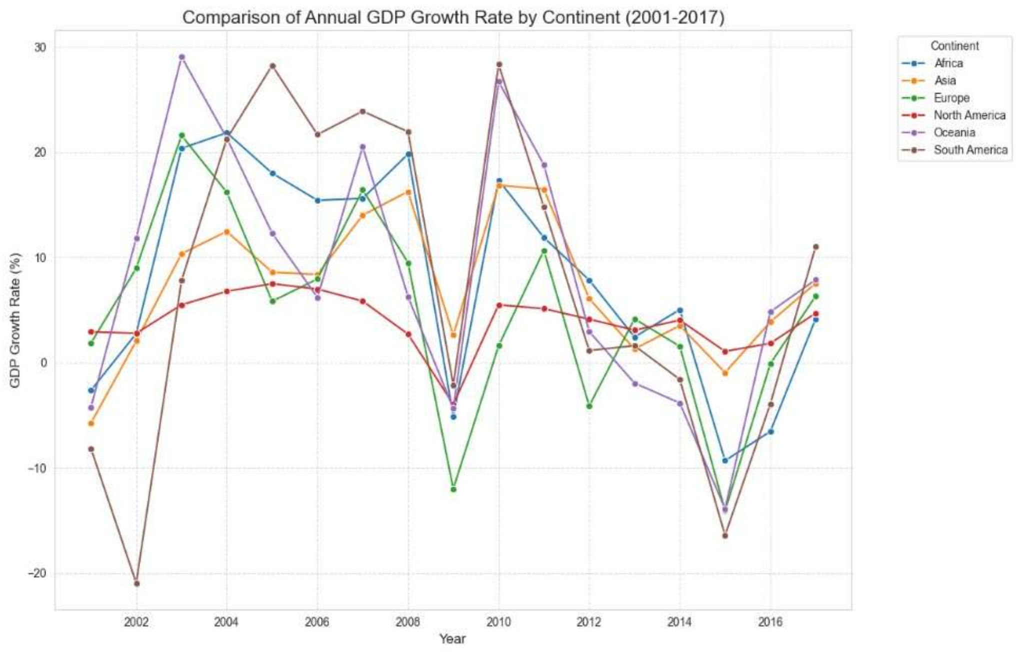
- **Data Mapping:** The clean data was first mapped to include a Continent column, derived from the Country information.
- **Aggregation:** The GDP data was aggregated by **Continent** and **Year** to establish total economic output for each region.
- **Growth Calculation:** The **Annual GDP Growth Rate (%)** was calculated for each continent using the .pct_change () function on the aggregated GDP.
- **Analysis Period:** The final analysis and plot were filtered to focus on the historical comparison period of **2001–2017**.
- **Visualization:** A line plot (sns. line plot) was generated to visually compare the Annual GDP Growth Rate across all continents, which was then saved as continent_gdp_growth_comparison_historical.png.



```python
# ------------------------------------------------------------------------
# --- SECTION 1: CONTINENTAL GDP GROWTH RATE COMPARISON (2000-2017) ---
# ------------------------------------------------------------------------

df_cont = df.copy()
df_cont['Continent'] = df_cont['Country'].map(continent_mapping)
df_cont.dropna(subset=['Continent'], inplace=True)

# Aggregate GDP by Continent and Year
continent_gdp = df_cont.groupby(['Continent', 'Year'])['Gross Domestic Product (GDP)'].sum().reset_index()
continent_gdp = continent_gdp.sort_values(by=['Continent', 'Year'])

# Calculate Annual GDP Growth Rate for each Continent
continent_gdp['GDP_Growth_Rate (%)'] = continent_gdp.groupby('Continent')['Gross Domestic Product (GDP)'].pct_change() * 100
continent_gdp.dropna(subset=['GDP_Growth_Rate (%)'], inplace=True)

# Filter for the requested analysis period for the plot (2017 max)
continent_gdp_plot = continent_gdp[continent_gdp['Year'] <= 2017].copy()

# Plotting the Comparison (2001-2017)
sns.set_style("whitegrid")
plt.figure(figsize=(14, 8))

sns.lineplot(
    x='Year',
    y='GDP_Growth_Rate (%)',
    hue='Continent',
    data=continent_gdp_plot,
    marker='o',
    dashes=False
)

plt.title('Comparison of Annual GDP Growth Rate by Continent (2001-2017)', fontsize=16)
plt.xlabel('Year', fontsize=12)
plt.ylabel('GDP Growth Rate (%)', fontsize=12)
plt.legend(title='Continent', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout(rect=[0, 0, 0.9, 1])
plt.savefig('Analysis/continent_gdp_growth_comparison_historical.png')
plt.close()
```



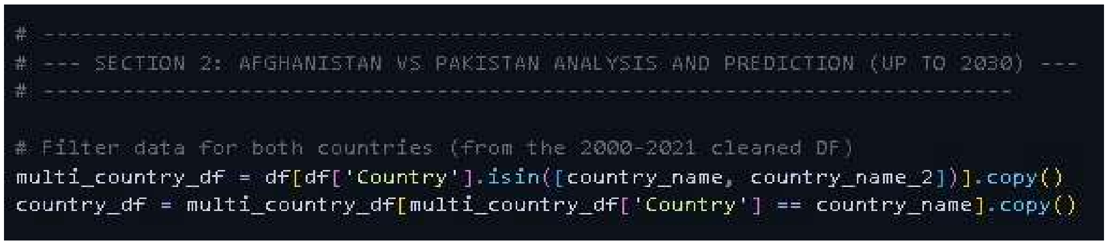
```python
# --- GLOBAL MAPPING & SETUP ---

# Continent mapping derived from external sources
continent_mapping = {
    'Afghanistan': 'Asia', 'Albania': 'Europe', 'Algeria': 'Africa', 'Andorra': 'Europe',
    'Angola': 'Africa', 'Antigua and Barbuda': 'North America', 'Argentina': 'South America',
    'Armenia': 'Asia', 'Australia': 'Oceania', 'Austria': 'Europe', 'Azerbaijan': 'Asia',
    'Bahamas': 'North America', 'Bahrain': 'Asia', 'Bangladesh': 'Asia', 'Barbados': 'North America',
    'Belarus': 'Europe', 'Belgium': 'Europe', 'Belize': 'North America', 'Benin': 'Africa',
    'Bhutan': 'Asia', 'Bolivia': 'South America', 'Bosnia and Herzegovina': 'Europe',
    'Botswana': 'Africa', 'Brazil': 'South America', 'Brunei Darussalam': 'Asia',
    'Bulgaria': 'Europe', 'Burkina Faso': 'Africa', 'Burundi': 'Africa', 'Cabo Verde': 'Africa',
    'Cambodia': 'Asia', 'Cameroon': 'Africa', 'Canada': 'North America', 'Central African Republic': 'Africa',
    'Chad': 'Africa', 'Chile': 'South America', 'China': 'Asia', 'China, Hong Kong Special Administrative Region': 'Asia',
    'China, Macao Special Administrative Region': 'Asia', 'Colombia': 'South America',
    'Comoros': 'Africa', 'Congo': 'Africa', 'Costa Rica': 'North America', 'Côte d\'Ivoire': 'Africa',
    'Croatia': 'Europe', 'Cyprus': 'Europe', 'Czech Republic': 'Europe', 'Democratic Republic of the Congo': 'Africa',
    'Denmark': 'Europe', 'Djibouti': 'Africa', 'Dominica': 'North America', 'Dominican Republic': 'North America',
    'Ecuador': 'South America', 'Egypt': 'Africa', 'El Salvador': 'North America',
    'Equatorial Guinea': 'Africa', 'Eritrea': 'Africa', 'Estonia': 'Europe', 'Eswatini': 'Africa',
    'Ethiopia': 'Africa', 'Fiji': 'Oceania', 'Finland': 'Europe', 'France': 'Europe',
    'Gabon': 'Africa', 'Gambia': 'Africa', 'Georgia': 'Asia', 'Germany': 'Europe',
    'Ghana': 'Africa', 'Greece': 'Europe', 'Grenada': 'North America', 'Guatemala': 'North America',
    'Guinea': 'Africa', 'Guinea-Bissau': 'Africa', 'Guyana': 'South America', 'Haiti': 'North America',
    'Honduras': 'North America', 'Hungary': 'Europe', 'Iceland': 'Europe', 'India': 'Asia',
    'Indonesia': 'Asia', 'Iran (Islamic Republic of)': 'Asia', 'Iraq': 'Asia', 'Ireland': 'Europe',
    'Israel': 'Asia', 'Italy': 'Europe', 'Jamaica': 'North America', 'Japan': 'Asia',
    'Jordan': 'Asia', 'Kazakhstan': 'Asia', 'Kenya': 'Africa', 'Kiribati': 'Oceania',
    'Kuwait': 'Asia', 'Kyrgyzstan': 'Asia', 'Lao People\'s Democratic Republic': 'Asia',
    'Latvia': 'Europe', 'Lebanon': 'Asia', 'Lesotho': 'Africa', 'Liberia': 'Africa',
    'Libya': 'Africa', 'Lithuania': 'Europe', 'Luxembourg': 'Europe', 'Madagascar': 'Africa',
    'Malawi': 'Africa', 'Malaysia': 'Asia', 'Maldives': 'Asia', 'Mali': 'Africa',
    'Malta': 'Europe', 'Mauritania': 'Africa', 'Mauritius': 'Africa', 'Mexico': 'North America',
    'Micronesia (Federated States of)': 'Oceania', 'Mongolia': 'Asia', 'Montenegro': 'Europe',
    'Morocco': 'Africa', 'Mozambique': 'Africa', 'Myanmar': 'Asia', 'Namibia': 'Africa',
    'Nepal': 'Asia', 'Netherlands': 'Europe', 'New Zealand': 'Oceania', 'Nicaragua': 'North America',
    'Niger': 'Africa', 'Nigeria': 'Africa', 'North Macedonia': 'Europe', 'Norway': 'Europe',
    'Oman': 'Asia', 'Pakistan': 'Asia', 'Palau': 'Oceania', 'Panama': 'North America',
    'Papua New Guinea': 'Oceania', 'Paraguay': 'South America', 'Peru': 'South America',
    'Philippines': 'Asia', 'Poland': 'Europe', 'Portugal': 'Europe', 'Qatar': 'Asia',
    'Republic of Korea': 'Asia', 'Republic of Moldova': 'Europe', 'Romania': 'Europe',
    'Russian Federation': 'Europe', 'Rwanda': 'Africa', 'Saint Kitts and Nevis': 'North America',
    'Saint Lucia': 'North America', 'Saint Vincent and the Grenadines': 'North America',
    'Samoa': 'Oceania', 'Sao Tome and Principe': 'Africa', 'Saudi Arabia': 'Asia',
    'Senegal': 'Africa', 'Serbia': 'Europe', 'Seychelles': 'Africa', 'Sierra Leone': 'Africa',
    'Singapore': 'Asia', 'Slovakia': 'Europe', 'Slovenia': 'Europe', 'Solomon Islands': 'Oceania',
    'South Africa': 'Africa', 'Spain': 'Europe', 'Sri Lanka': 'Asia', 'Sudan': 'Africa',
    'Suriname': 'South America', 'Sweden': 'Europe', 'Switzerland': 'Europe', 'Syrian Arab Republic': 'Asia',
    'Tajikistan': 'Asia', 'Thailand': 'Asia', 'Timor-Leste': 'Asia', 'Togo': 'Africa',
```



Objective: Prediction and Insight via Visualization

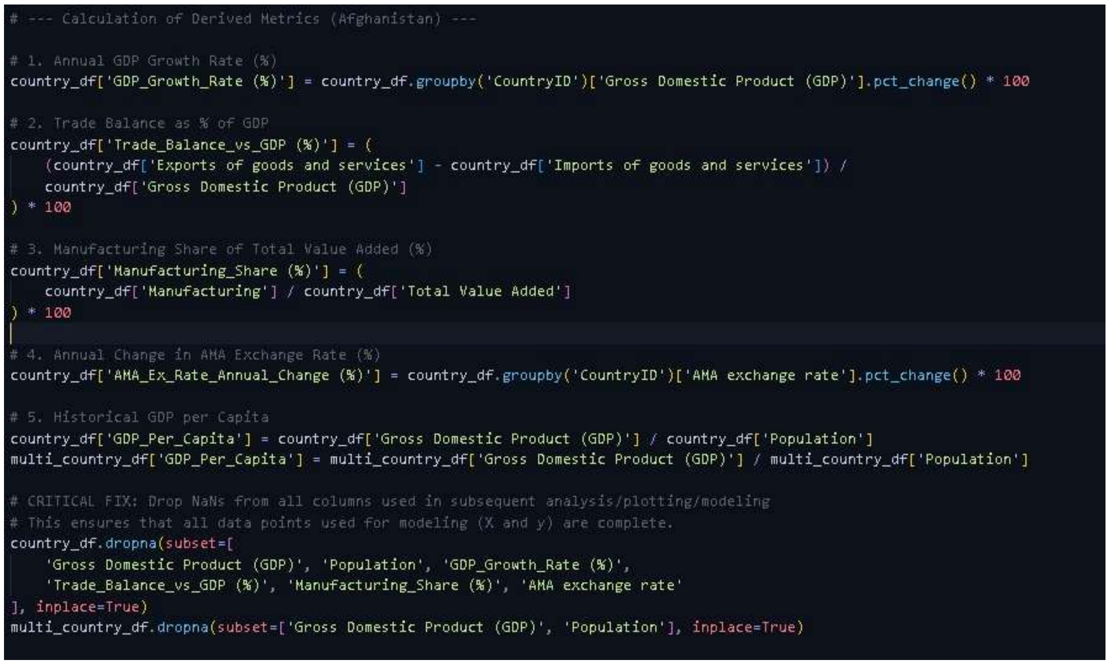
The central aim of Section 2 is not only to compare past developments but also to project future trends through 2030. This part of the report relies heavily on graphical analysis, using visual comparisons to illustrate how key indicators have evolved over time for both countries. Afghanistan is treated as the core case study, with its economic performance examined in detail and benchmarked against Pakistan using the full range of available time-series data. By structuring the analysis this way, the report delivers meaningful, forward-looking insights into whether the two economies are moving toward convergence or divergence, ultimately supporting better strategic planning and regional.

Data calculation and derivation:



```
# ---------------------------------------------------------------------------
# --- SECTION 2: AFGHANISTAN VS PAKISTAN ANALYSIS AND PREDICTION (UP TO 2030) ---
# ---------------------------------------------------------------------------

# Filter data for both countries (from the 2000-2021 cleaned DF)
multi_country_df = df[df['Country'].isin([country_name, country_name_2])].copy()
country_df = multi_country_df[multi_country_df['Country'] == country_name].copy()
```

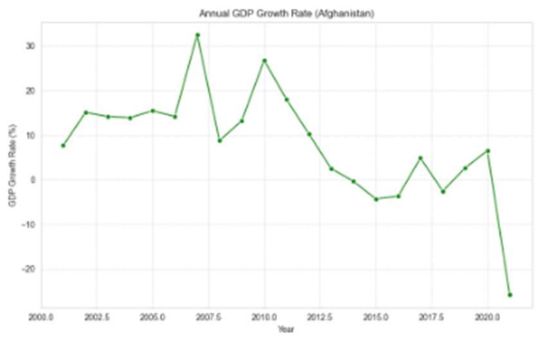
In this section, we use the available data to compute GDP and derive key economic metrics. We determine the manufacturing sector's share of total value to support our forecast, and we also analyze the currency trend through the exchange rate (AMA), expressed as units of local currency per USD.

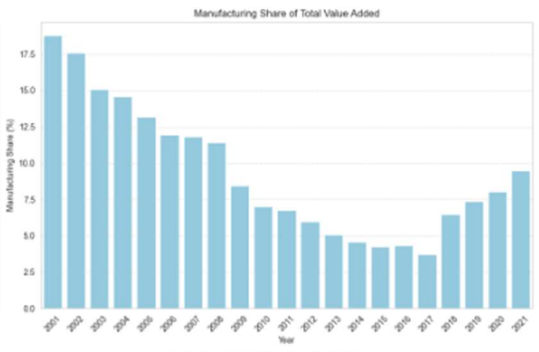


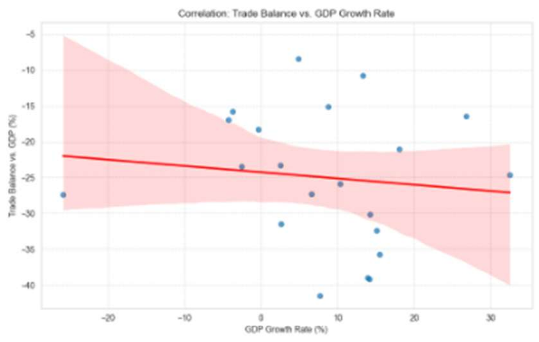
```
# --- Calculation of Derived Metrics (Afghanistan) ---

# 1. Annual GDP Growth Rate (%)
country_df['GDP_Growth_Rate (%)'] = country_df.groupby('CountryID')['Gross Domestic Product (GDP)'].pct_change() * 100

# 2. Trade Balance as % of GDP
country_df['Trade_Balance_vs_GDP (%)'] = (
    (country_df['Exports of goods and services'] - country_df['Imports of goods and services']) /
    country_df['Gross Domestic Product (GDP)']
) * 100

# 3. Manufacturing Share of Total Value Added (%)
country_df['Manufacturing_Share (%)'] = (
    country_df['Manufacturing'] / country_df['Total Value Added']
) * 100

# 4. Annual Change in AMA Exchange Rate (%)
country_df['AMA_Ex_Rate_Annual_Change (%)'] = country_df.groupby('CountryID')['AMA exchange rate'].pct_change() * 100

# 5. Historical GDP per Capita
country_df['GDP_Per_Capita'] = country_df['Gross Domestic Product (GDP)'] / country_df['Population']
multi_country_df['GDP_Per_Capita'] = multi_country_df['Gross Domestic Product (GDP)'] / multi_country_df['Population']

# CRITICAL FIX: Drop NaNs from all columns used in subsequent analysis/plotting/modeling
# This ensures that all data points used for modeling (X and y) are complete.
country_df.dropna(subset=[
    'Gross Domestic Product (GDP)', 'Population', 'GDP_Growth_Rate (%)',
    'Trade_Balance_vs_GDP (%)', 'Manufacturing_Share (%)', 'AMA exchange rate'
], inplace=True)
multi_country_df.dropna(subset=['Gross Domestic Product (GDP)', 'Population'], inplace=True)
```

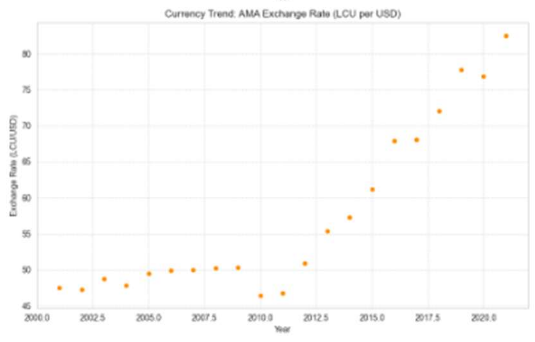
Data visualization using modules

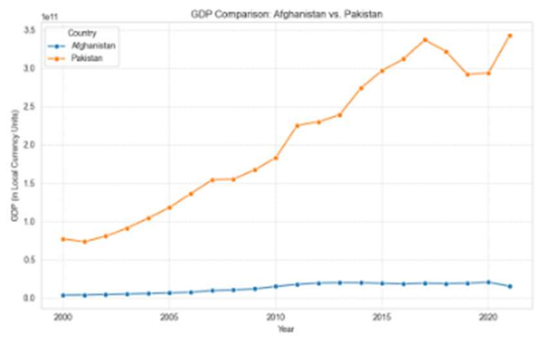
The provided Python snippet utilizes the **pandas** library for crucial steps in data preparation for a pie chart visualization of GDP sector contributions. First, pandas is used to efficiently locate the latest year's data for the country (Afghanistan) by finding the maximum year in the 'Year' column and filtering the Data Frame down to a single row (sector data). Then, a dictionary mapping is applied using the. rename(index=sectors) method on the resulting pandas Series. This step transforms the detailed, technical sector names (e.g., 'Agriculture, hunting, forestry, fishing') into simpler, display-ready labels (e.g., 'Agriculture'). Finally, the Series is sorted in descending order using. sort values(ascending=False) to prioritize the largest sectors, producing the final sector labels (the index) and sector sizes (the values) that are perfectly structured and ordered for plotting.

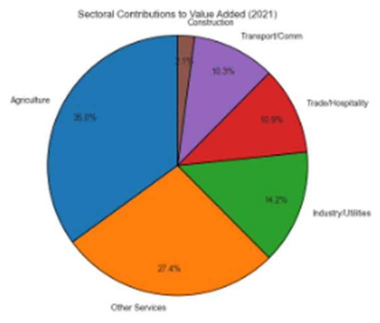


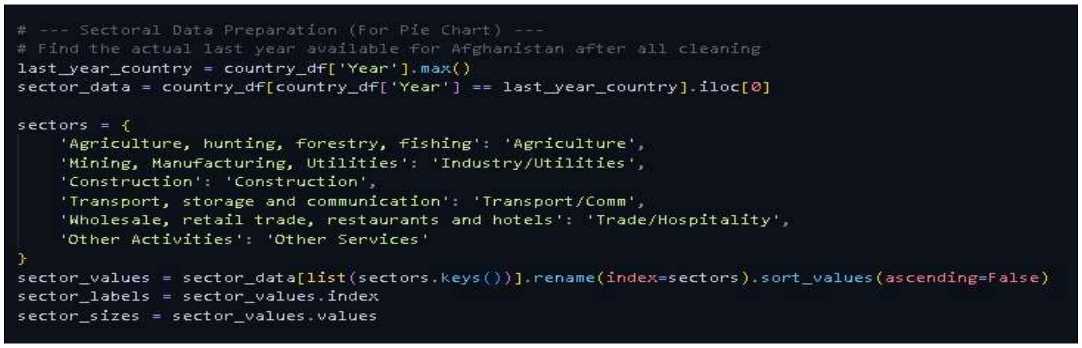












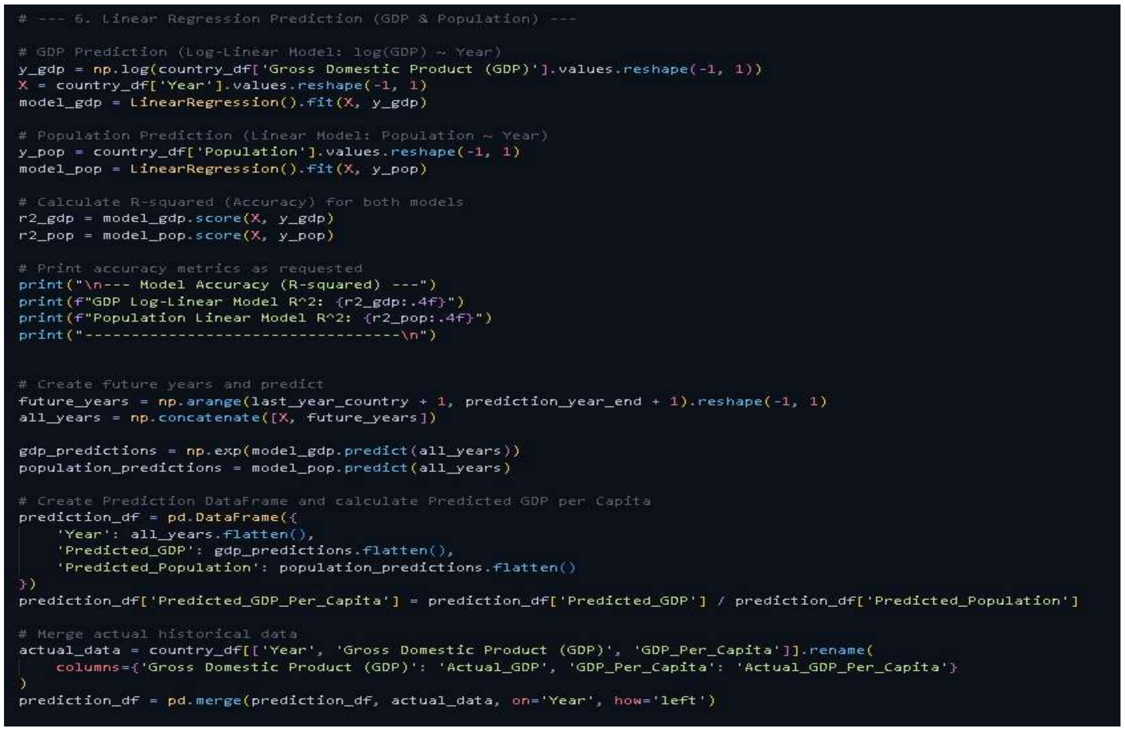
```python
# --- Sectoral Data Preparation (For Pie Chart) ---
# Find the actual last year available for Afghanistan after all cleaning
last_year_country = country_df['Year'].max()
sector_data = country_df[country_df['Year'] == last_year_country].iloc[0]

sectors = {
    'Agriculture, hunting, forestry, fishing': 'Agriculture',
    'Mining, Manufacturing, Utilities': 'Industry/Utilities',
    'Construction': 'Construction',
    'Transport, storage and communication': 'Transport/Comm',
    'Wholesale, retail trade, restaurants and hotels': 'Trade/Hospitality',
    'Other Activities': 'Other Services'
}
sector_values = sector_data[list(sectors.keys())].rename(index=sectors).sort_values(ascending=False)
sector_labels = sector_values.index
sector_sizes = sector_values.values
```

Data prediction for future:

This Python code uses two **Linear Regression** models to forecast economic metrics until 2030.

The first model, a log-linear model, predicts the **logarithm of GDP** against the Year, which is then exponentiated back to get the predicted **GDP**. The second model predicts **Population** linearly based on the Year. Both models' accuracies are quantified using the **R-squared** metric.

Finally, the code uses these predictions for future years (up to 2030) to calculate the forecasted **GDP Per Capita** (). The resulting predictions are compiled into a Data Frame and merged with the historical data for a complete, merged forecast record. It tells us what will happen in the future with the GDP per capita of a certain country with the help of linear regression.



```python
# --- 6. Linear Regression Prediction (GDP & Population) ---

# GDP Prediction (Log-Linear Model: log(GDP) ~ Year)
y_gdp = np.log(country_df['Gross Domestic Product (GDP)'].values.reshape(-1, 1))
X = country_df['Year'].values.reshape(-1, 1)
model_gdp = LinearRegression().fit(X, y_gdp)

# Population Prediction (Linear Model: Population ~ Year)
y_pop = country_df['Population'].values.reshape(-1, 1)
model_pop = LinearRegression().fit(X, y_pop)

# Calculate R-squared (Accuracy) for both models
r2_gdp = model_gdp.score(X, y_gdp)
r2_pop = model_pop.score(X, y_pop)

# Print accuracy metrics as requested
print("\n--- Model Accuracy (R-squared) ---")
print(f"GDP Log-Linear Model R^2: {r2_gdp:.4f}")
print(f"Population Linear Model R^2: {r2_pop:.4f}")
print("----------------------------------\n")


# Create future years and predict
future_years = np.arange(last_year_country + 1, prediction_year_end + 1).reshape(-1, 1)
all_years = np.concatenate([X, future_years])

gdp_predictions = np.exp(model_gdp.predict(all_years))
population_predictions = model_pop.predict(all_years)

# Create Prediction DataFrame and calculate Predicted GDP per Capita
prediction_df = pd.DataFrame({
    'Year': all_years.flatten(),
    'Predicted_GDP': gdp_predictions.flatten(),
    'Predicted_Population': population_predictions.flatten()
})
prediction_df['Predicted_GDP_Per_Capita'] = prediction_df['Predicted_GDP'] / prediction_df['Predicted_Population']

# Merge actual historical data
actual_data = country_df[['Year', 'Gross Domestic Product (GDP)', 'GDP_Per_Capita']].rename(
    columns={'Gross Domestic Product (GDP)': 'Actual_GDP', 'GDP_Per_Capita': 'Actual_GDP_Per_Capita'}
)
prediction_df = pd.merge(prediction_df, actual_data, on='Year', how='left')
```

Data plotting using different modules:

• Data visualization with Seaborn begins by loading and structuring the dataset in a Pandas Data Frame, where each column represents a specific variable (e.g., Year, GDP_Growth_Rate).

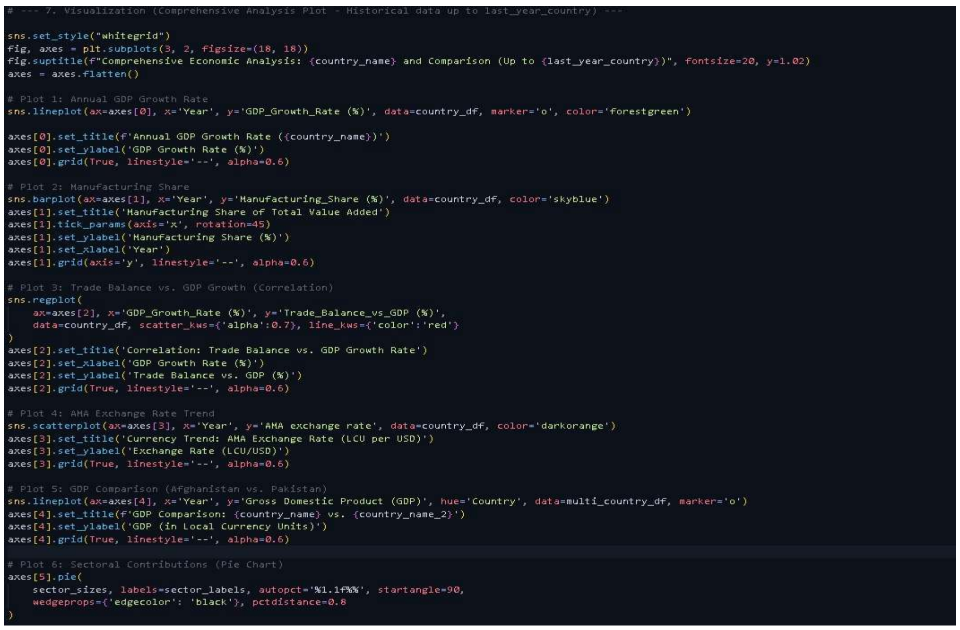
• The plotting function is selected according to the analytical objective:

- **sns. line plot ()** for time-series trends,
- **sns. scatterplot ()** or **sns. regplot ()** for correlations,
- **sns. barplot ()** for categorical comparisons or distributions.

• Core parameters include the axis object (**ax=axes[i]**), the dataset (**data=country_df**), and variable mappings for x and y.

• Professional clarity is achieved through axis customizations such as **set_title ()**, **set_xlabel ()**, and **grid ()**, which improve context, labeling, and readability.

• Overall, modules like Seaborn help convert structured data into clear, interpretable visual insights.



```
# --- 7. Visualization (Comprehensive Analysis Plot - Historical data up to last_year_country) ---

sns.set_style("whitegrid")
fig, axes = plt.subplots(3, 2, figsize=(18, 18))
fig.suptitle(f"Comprehensive Economic Analysis: {country_name} and Comparison (Up to {last_year_country})", fontsize=20, y=1.02)
axes = axes.flatten()

# Plot 1: Annual GDP Growth Rate
sns.lineplot(ax=axes[0], x='Year', y='GDP_Growth_Rate (%)', data=country_df, marker='o', color='forestgreen')

axes[0].set_title(f'Annual GDP Growth Rate ({country_name})')
axes[0].set_ylabel('GDP Growth Rate (%)')
axes[0].grid(True, linestyle='--', alpha=0.6)

# Plot 2: Manufacturing Share
sns.barplot(ax=axes[1], x='Year', y='Manufacturing_Share (%)', data=country_df, color='skyblue')
axes[1].set_title('Manufacturing Share of Total Value Added')
axes[1].tick_params(axis='x', rotation=45)
axes[1].set_ylabel('Manufacturing Share (%)')
axes[1].set_xlabel('Year')
axes[1].grid(axis='y', linestyle='--', alpha=0.6)

# Plot 3: Trade Balance vs. GDP Growth (Correlation)
sns.regplot(
    ax=axes[2], x='GDP_Growth_Rate (%)', y='Trade_Balance_vs_GDP (%)',
    data=country_df, scatter_kws={'alpha':0.7}, line_kws={'color':'red'}
)
axes[2].set_title('Correlation: Trade Balance vs. GDP Growth Rate')
axes[2].set_xlabel('GDP Growth Rate (%)')
axes[2].set_ylabel('Trade Balance vs. GDP (%)')
axes[2].grid(True, linestyle='--', alpha=0.6)

# Plot 4: AMA Exchange Rate Trend
sns.scatterplot(ax=axes[3], x='Year', y='AMA exchange rate', data=country_df, color='darkorange')
axes[3].set_title('Currency Trend: AMA Exchange Rate (LCU per USD)')
axes[3].set_ylabel('Exchange Rate (LCU/USD)')
axes[3].grid(True, linestyle='--', alpha=0.6)

# Plot 5: GDP Comparison (Afghanistan vs. Pakistan)
sns.lineplot(ax=axes[4], x='Year', y='Gross Domestic Product (GDP)', hue='Country', data=multi_country_df, marker='o')
axes[4].set_title(f'GDP Comparison: {country_name} vs. {country_name_2}')
axes[4].set_ylabel('GDP (in Local Currency Units)')
axes[4].grid(True, linestyle='--', alpha=0.6)

# Plot 6: Sectoral Contributions (Pie Chart)
axes[5].pie(
    sector_sizes, labels=sector_labels, autopct='%1.1f%%', startangle=90,
    wedgeprops={'edgecolor': 'black'}, pctdistance=0.8
)
```

Final Execution:

Here is the final execution where we have calculated all the required calculations of different sectors of different countries.

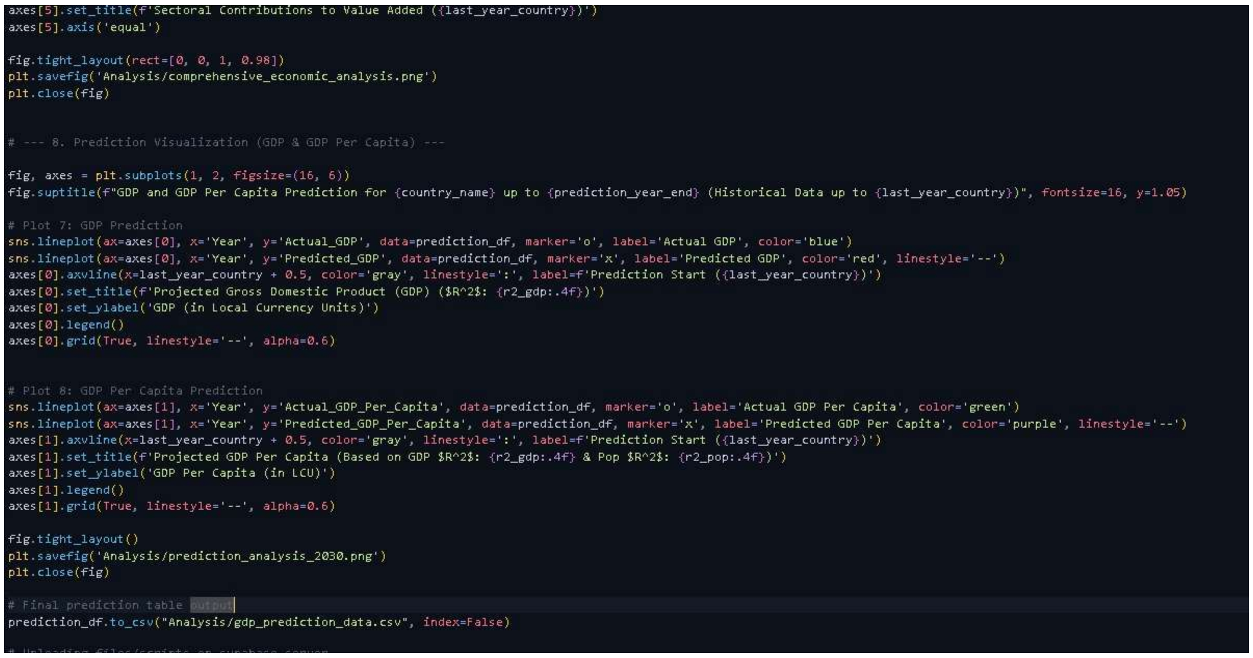
This project delivered finalized predictions for **GDP and GDP Per Capita** up to the year 2030.

Visualization & Results

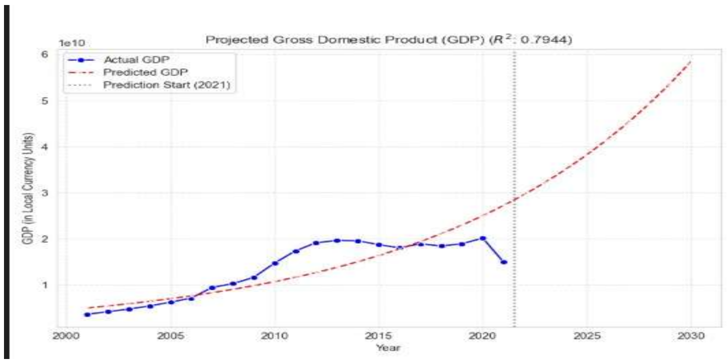
- **Plot 7 (GDP Prediction):**
  - **Goal:** Visualize Actual vs. Predicted GDP (in Local Currency Units).
  - **Historical Data:** Actual GDP (blue circles) and Model-Fit Predicted GDP (red crosses) show strong agreement.
  - **Forecast:** The red dashed line projects future GDP beyond.
- **Plot 8 (GDP Per Capita Prediction):**
  - **Goal:** Visualize Actual vs. Predicted GDP Per Capita.
  - **Historical Data:** Actual Per Capita (green squares) vs. Model-Fit Predicted (purple x's).
  - **Forecast Basis:** Projections are based on **Predicted GDP** () and **Predicted Population** ().
  - **Projection:** The purple dashed line shows the anticipated trend in average economic output per person.
- **Prediction Start:** Both plots use a vertical grey dashed line to clearly mark the beginning of the forward forecast (after).

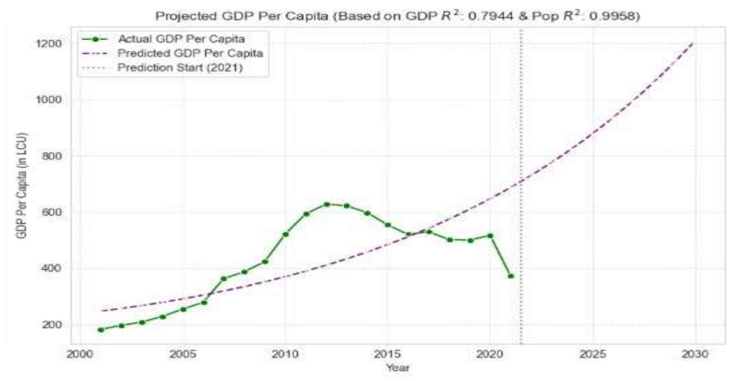
Final Output:

- All final predicted values, including sectoral contributions and final GDP/Per Capita, were exported to:
  - **File:** `analysis/prediction\_GDP\_data.`
  - **Format:** Clean data output with the index suppressed for easy consumption.



```python
axes[5].set_title(f'Sectoral Contributions to Value Added ({last_year_country})')
axes[5].axis('equal')

fig.tight_layout(rect=[0, 0, 1, 0.98])
plt.savefig('Analysis/comprehensive_economic_analysis.png')
plt.close(fig)


# --- 8. Prediction Visualization (GDP & GDP Per Capita) ---

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle(f"GDP and GDP Per Capita Prediction for {country_name} up to {prediction_year_end} (Historical Data up to {last_year_country})", fontsize=16, y=1.05)

# Plot 7: GDP Prediction
sns.lineplot(ax=axes[0], x='Year', y='Actual_GDP', data=prediction_df, marker='o', label='Actual GDP', color='blue')
sns.lineplot(ax=axes[0], x='Year', y='Predicted_GDP', data=prediction_df, marker='x', label='Predicted GDP', color='red', linestyle='--')
axes[0].axvline(x=last_year_country + 0.5, color='gray', linestyle=':', label=f'Prediction Start ({last_year_country})')
axes[0].set_title(f'Projected Gross Domestic Product (GDP) ($R^2$: {r2_gdp:.4f})')
axes[0].set_ylabel('GDP (in Local Currency Units)')
axes[0].legend()
axes[0].grid(True, linestyle='--', alpha=0.6)


# Plot 8: GDP Per Capita Prediction
sns.lineplot(ax=axes[1], x='Year', y='Actual_GDP_Per_Capita', data=prediction_df, marker='o', label='Actual GDP Per Capita', color='green')
sns.lineplot(ax=axes[1], x='Year', y='Predicted_GDP_Per_Capita', data=prediction_df, marker='x', label='Predicted GDP Per Capita', color='purple', linestyle='--')
axes[1].axvline(x=last_year_country + 0.5, color='gray', linestyle=':', label=f'Prediction Start ({last_year_country})')
axes[1].set_title(f'Projected GDP Per Capita (Based on GDP $R^2$: {r2_gdp:.4f} & Pop $R^2$: {r2_pop:.4f})')
axes[1].set_ylabel('GDP Per Capita (in LCU)')
axes[1].legend()
axes[1].grid(True, linestyle='--', alpha=0.6)

fig.tight_layout()
plt.savefig('Analysis/prediction_analysis_2030.png')
plt.close(fig)

# Final prediction table output
prediction_df.to_csv("Analysis/gdp_prediction_data.csv", index=False)
```





Final Data Commanding to Cloud:

Here is the final and the most important task of our project where we submitted the code, the data file and each and everything to the cloud-based server for future use and experimentation.

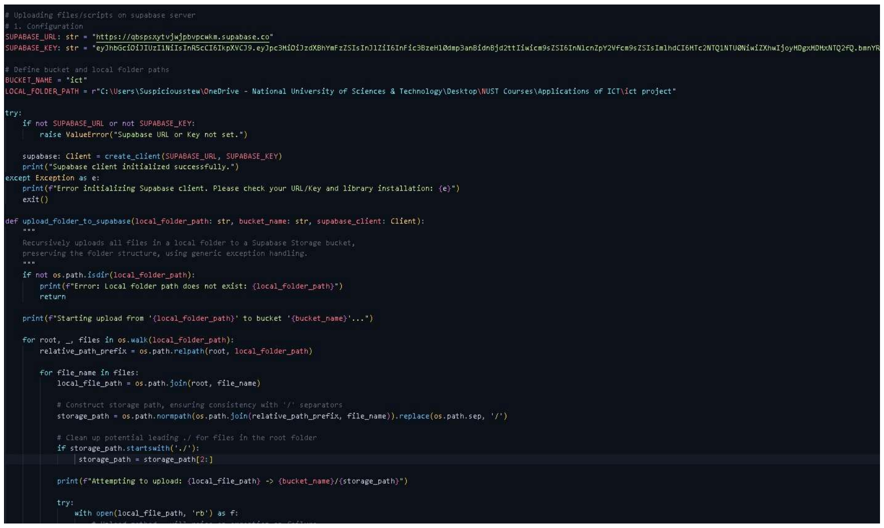
 Authentication**:** We use our unique URL and Key to securely connect to the Supa base client.

 Validation**:** The script verifies the local folder we want to upload exists.

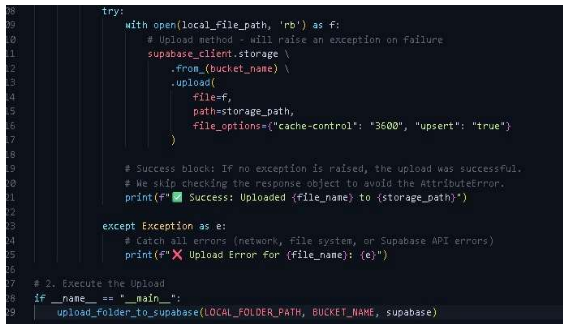
 Traversal**:** It systematically walks through *all* files and subfolders in the local path.

 Path **Mapping:** For each file, it calculates the correct, standardized storage address (path) for the cloud.

 Secure **Upload:** We open the file locally and stream its data to the designated Supa base bucket.

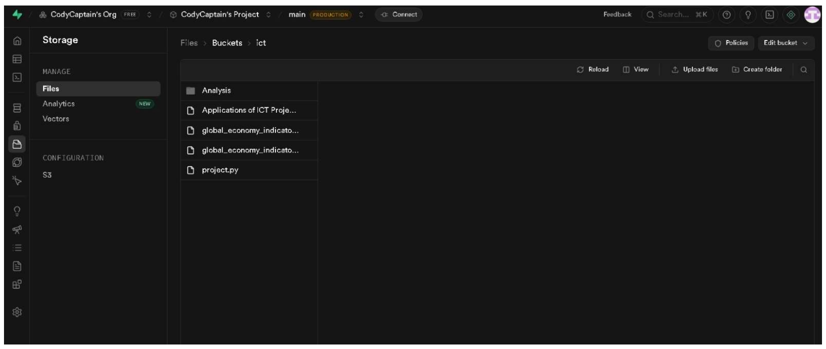


```python
# Uploading files/scripts on supabase server
# 1. Configuration
SUPABASE_URL: str = "https://qbspsxytvjwjpbvpcwkm.supabase.co"
SUPABASE_KEY: str = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InFic3BzeHl0dmp3anBidnBjd2ttIiwicm9sZSI6ImNlcnZpY2Vfcm9sZSIsImlhdCI6MTc2NTQ1NTU0NiwiZXhwIjoyMDgxMDMxNTQ2fQ.bmnYR

# Define bucket and local folder paths
BUCKET_NAME = "ict"
LOCAL_FOLDER_PATH = r"C:\Users\Suspicousstew\OneDrive - National University of Sciences & Technology\Desktop\NUST Courses\Applications of ICT\ict project"

try:
    if not SUPABASE_URL or not SUPABASE_KEY:
        raise ValueError("Supabase URL or Key not set.")

    supabase: Client = create_client(SUPABASE_URL, SUPABASE_KEY)
    print("Supabase client initialized successfully.")
except Exception as e:
    print(f"Error initializing Supabase client. Please check your URL/Key and library installation: {e}")
    exit()

def upload_folder_to_supabase(local_folder_path: str, bucket_name: str, supabase_client: Client):
    """
    Recursively uploads all files in a local folder to a Supabase Storage bucket,
    preserving the folder structure, using generic exception handling.
    """
    if not os.path.isdir(local_folder_path):
        print(f"Error: Local folder path does not exist: {local_folder_path}")
        return

    print(f"Starting upload from '{local_folder_path}' to bucket '{bucket_name}'...")

    for root, _, files in os.walk(local_folder_path):
        relative_path_prefix = os.path.relpath(root, local_folder_path)

        for file_name in files:
            local_file_path = os.path.join(root, file_name)

            # Construct storage path, ensuring consistency with '/' separators
            storage_path = os.path.normpath(os.path.join(relative_path_prefix, file_name)).replace(os.path.sep, '/')

            # Clean up potential leading ./ for files in the root folder
            if storage_path.startswith('./'):
                storage_path = storage_path[2:]

            print(f"Attempting to upload: {local_file_path} -> {bucket_name}/{storage_path}")

            try:
                with open(local_file_path, 'rb') as f:
```

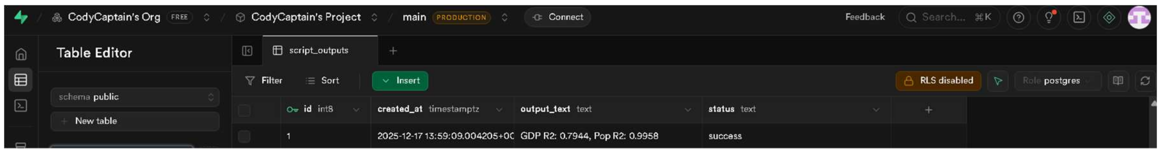


```python
            try:
                with open(local_file_path, 'rb') as f:
                    # Upload method - will raise an exception on failure
                    supabase_client.storage \
                        .from_(bucket_name) \
                        .upload(
                            file=f,
                            path=storage_path,
                            file_options={"cache-control": "3600", "upsert": "true"}
                        )

                # Success block: If no exception is raised, the upload was successful.
                # We skip checking the response object to avoid the AttributeError.
                print(f"✅ Success: Uploaded {file_name} to {storage_path}")

            except Exception as e:
                # Catch all errors (network, file system, or Supabase API errors)
                print(f"❌ Upload Error for {file_name}: {e}")

# 2. Execute the Upload
if __name__ == "__main__":
    upload_folder_to_supabase(LOCAL_FOLDER_PATH, BUCKET_NAME, supabase)
```

☐ Result**:** All our local code and data are now securely saved and accessible on the cloud and the output of python script is showing on supabase.



Analysis
Applications of ICT Proje...
global_economy_indicato...
global_economy_indicato...
project.py



id int8	created_at timestamptz	output_text text	status text
1	2025-12-17 13:59:09.004205+0C	GDP R2: 0.7944, Pop R2: 0.9958	success