DUTTA Proposal

June 27, 2025

[191]:
```
#Understanding Dataset
import pandas as pd

df = pd.read_csv("world_bank_data_2025.csv")
df.shape
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3472 entries, 0 to 3471
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   country_name                    3472 non-null   object
 1   country_id                      3472 non-null   object
 2   year                            3472 non-null   int64
 3   Inflation (CPI %)               2694 non-null   float64
 4   GDP (Current USD)               2933 non-null   float64
 5   GDP per Capita (Current USD)    2938 non-null   float64
 6   Unemployment Rate (%)           2795 non-null   float64
 7   Interest Rate (Real, %)         1735 non-null   float64
 8   Inflation (GDP Deflator, %)     2904 non-null   float64
 9   GDP Growth (% Annual)           2912 non-null   float64
 10  Current Account Balance (% GDP) 2563 non-null   float64
 11  Government Expense (% of GDP)   1820 non-null   float64
 12  Government Revenue (% of GDP)   1829 non-null   float64
 13  Tax Revenue (% of GDP)          1833 non-null   float64
 14  Gross National Income (USD)     2796 non-null   float64
 15  Public Debt (% of GDP)          852 non-null    float64
dtypes: float64(13), int64(1), object(2)
memory usage: 434.1+ KB
```

[191]:

	year	Inflation (CPI %)	GDP (Current USD)	\
count	3472.000000	2694.000000	2.933000e+03	
mean	2017.500000	6.233154	3.964323e+11	
std	4.610436	19.726903	1.749315e+12	
min	2010.000000	-6.687321	3.210541e+07	
25%	2013.750000	1.402112	6.264757e+09	

50%	2017.500000	3.213523	2.587360e+10
75%	2021.250000	6.186626	1.874939e+11
max	2025.000000	557.201817	2.772071e+13

	GDP per Capita (Current USD)	Unemployment Rate (%)	\
count	2938.000000	2795.000000	
mean	18483.495612	7.841141	
std	27301.814024	5.964358	
min	193.007146	0.100000	
25%	2280.748732	3.611000	
50%	6827.668145	5.771000	
75%	23727.024581	10.731500	
max	256580.515123	35.359000	

	Interest Rate (Real, %)	Inflation (GDP Deflator, %)	\
count	1735.000000	2904.000000	
mean	5.405051	6.634865	
std	9.740924	25.820196	
min	-81.132121	-28.760135	
25%	1.734057	1.218347	
50%	5.079009	3.223184	
75%	8.869434	6.905463	
max	61.882604	921.535652	

	GDP Growth (% Annual)	Current Account Balance (% GDP)	\
count	2912.000000	2563.000000	
mean	2.853544	-2.363241	
std	6.053786	13.740986	
min	-54.336155	-60.877754	
25%	0.997032	-7.496525	
50%	3.100442	-2.656009	
75%	5.355110	1.854710	
max	86.826748	235.750605	

	Government Expense (% of GDP)	Government Revenue (% of GDP)	\
count	1820.000000	1829.000000	
mean	27.325359	26.677467	
std	12.642464	18.116253	
min	0.000136	0.000081	
25%	17.511484	17.639153	
50%	26.000850	24.821425	
75%	34.884582	32.700782	
max	103.725787	344.999451	

	Tax Revenue (% of GDP)	Gross National Income (USD)	\
count	1833.000000	2.796000e+03	
mean	16.969924	4.142237e+11	

```
std                   8.218539                1.799783e+12
min                   0.000063                5.107533e+07
25%                  12.285344                7.475538e+09
50%                  16.321438                2.986520e+10
75%                  21.448658                1.972529e+11
max                 147.640196                2.757614e+13

       Public Debt (% of GDP)
count              852.000000
mean                61.863736
std                 40.409792
min                  1.845685
25%                 33.894232
50%                 51.651469
75%                 81.930649
max                249.366027
```

[193]:
```python
#Check for Missing or NULL values with detail analysis
import seaborn as sns
import matplotlib.pyplot as plt

#sns.heatmap(df.null(), cbar = False)
#print(df.isnull().sum())

def data_quality_assessment(df):
    """
    DATA QUALITY ASSESSMENT
    """
    print("DATA QUALITY ASSESSMENT")
    print("-" * 50)
    missing_data = pd.DataFrame({
        'Columns': df.columns,
        'Missing Values': df.isnull().sum(),
        'Missing Percentage': (df.isnull().sum() / len(df)) * 100
    }).sort_values('Missing Percentage', ascending=False)

    print(missing_data)

    # Visualisation des valeurs manquantes
    plt.figure(figsize=(12, 4))
    sns.heatmap(df.isnull(), cbar=True, yticklabels=False, cmap='viridis')
    plt.title('Map of Missing Values by Variable', fontsize=16,␣
 ↪fontweight='bold')
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
```

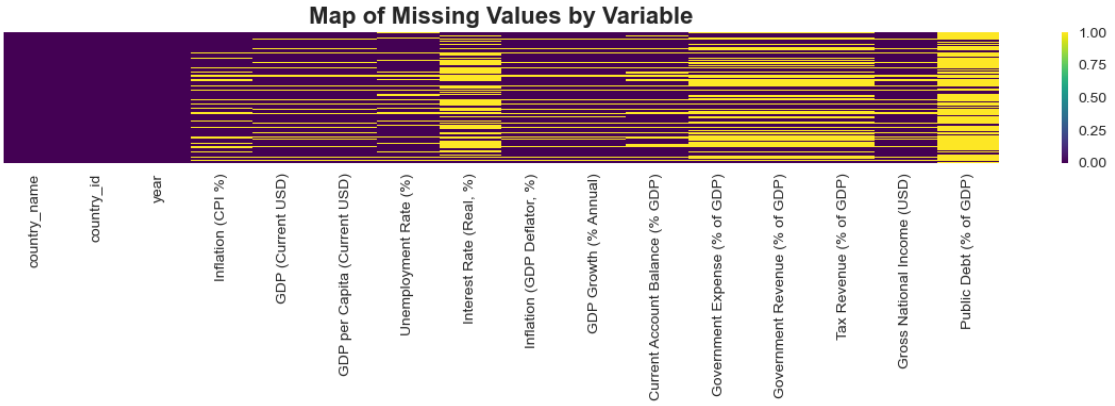
```
    return missing_data
```

[195]:
```
data_quality_assessment(df)
```

```
DATA QUALITY ASSESSMENT
--------------------------------------------------
```

	Columns
Public Debt (% of GDP)	Public Debt (% of GDP)
Interest Rate (Real, %)	Interest Rate (Real, %)
Government Expense (% of GDP)	Government Expense (% of GDP)
Government Revenue (% of GDP)	Government Revenue (% of GDP)
Tax Revenue (% of GDP)	Tax Revenue (% of GDP)
Current Account Balance (% GDP)	Current Account Balance (% GDP)
Inflation (CPI %)	Inflation (CPI %)
Unemployment Rate (%)	Unemployment Rate (%)
Gross National Income (USD)	Gross National Income (USD)
Inflation (GDP Deflator, %)	Inflation (GDP Deflator, %)
GDP Growth (% Annual)	GDP Growth (% Annual)
GDP (Current USD)	GDP (Current USD)
GDP per Capita (Current USD)	GDP per Capita (Current USD)
country_name	country_name
country_id	country_id
year	year

	Missing Values	Missing Percentage
Public Debt (% of GDP)	2620	75.460829
Interest Rate (Real, %)	1737	50.028802
Government Expense (% of GDP)	1652	47.580645
Government Revenue (% of GDP)	1643	47.321429
Tax Revenue (% of GDP)	1639	47.206221
Current Account Balance (% GDP)	909	26.180876
Inflation (CPI %)	778	22.407834
Unemployment Rate (%)	677	19.498848
Gross National Income (USD)	676	19.470046
Inflation (GDP Deflator, %)	568	16.359447
GDP Growth (% Annual)	560	16.129032
GDP (Current USD)	539	15.524194
GDP per Capita (Current USD)	534	15.380184
country_name	0	0.000000
country_id	0	0.000000
year	0	0.000000



[195]:

	Columns
Public Debt (% of GDP)	Public Debt (% of GDP)
Interest Rate (Real, %)	Interest Rate (Real, %)
Government Expense (% of GDP)	Government Expense (% of GDP)
Government Revenue (% of GDP)	Government Revenue (% of GDP)
Tax Revenue (% of GDP)	Tax Revenue (% of GDP)
Current Account Balance (% GDP)	Current Account Balance (% GDP)
Inflation (CPI %)	Inflation (CPI %)
Unemployment Rate (%)	Unemployment Rate (%)
Gross National Income (USD)	Gross National Income (USD)
Inflation (GDP Deflator, %)	Inflation (GDP Deflator, %)
GDP Growth (% Annual)	GDP Growth (% Annual)
GDP (Current USD)	GDP (Current USD)
GDP per Capita (Current USD)	GDP per Capita (Current USD)
country_name	country_name
country_id	country_id
year	year

	Missing Values	Missing Percentage
Public Debt (% of GDP)	2620	75.460829
Interest Rate (Real, %)	1737	50.028802
Government Expense (% of GDP)	1652	47.580645
Government Revenue (% of GDP)	1643	47.321429
Tax Revenue (% of GDP)	1639	47.206221
Current Account Balance (% GDP)	909	26.180876
Inflation (CPI %)	778	22.407834
Unemployment Rate (%)	677	19.498848
Gross National Income (USD)	676	19.470046
Inflation (GDP Deflator, %)	568	16.359447
GDP Growth (% Annual)	560	16.129032
GDP (Current USD)	539	15.524194
GDP per Capita (Current USD)	534	15.380184

country_name	0	0.000000
country_id	0	0.000000
year	0	0.000000

[197]:
```
#In-Depth Descriptive Analysis
import numpy as np

def descriptive_analysis(df):
    """
    Descriptive Analysis for Numerical Variables
    """
    print("DESCRIPTIVE STATISTICS")
    print("-" * 50)

    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    numeric_cols = [col for col in numeric_cols if col != 'year']

    # Statistiques de base
    desc_stats = df[numeric_cols].describe()
    print(desc_stats.round(2))

    # Visualisation des distributions
    fig, axes = plt.subplots(4, 4, figsize=(20, 16))
    axes = axes.ravel()

    for i, col in enumerate(numeric_cols):
        if i < len(axes):
            # Histogramme avec courbe de densité
            df[col].hist(bins=30, alpha=0.7, ax=axes[i], color='skyblue',␣
↪edgecolor='black')
            axes[i].axvline(df[col].mean(), color='red', linestyle='--',␣
↪label=f'Moyenne: {df[col].mean():.2f}')
            axes[i].set_title(f'Distribution: {col}', fontweight='bold')
            axes[i].legend()
            axes[i].grid(True, alpha=0.3)

    # Masquer les sous-graphiques inutilisés
    for i in range(len(numeric_cols), len(axes)):
        axes[i].set_visible(False)

    plt.tight_layout()
    plt.suptitle('Distributions of Economic Variables', fontsize=16,␣
↪fontweight='bold', y=1.02)
    plt.show()
```

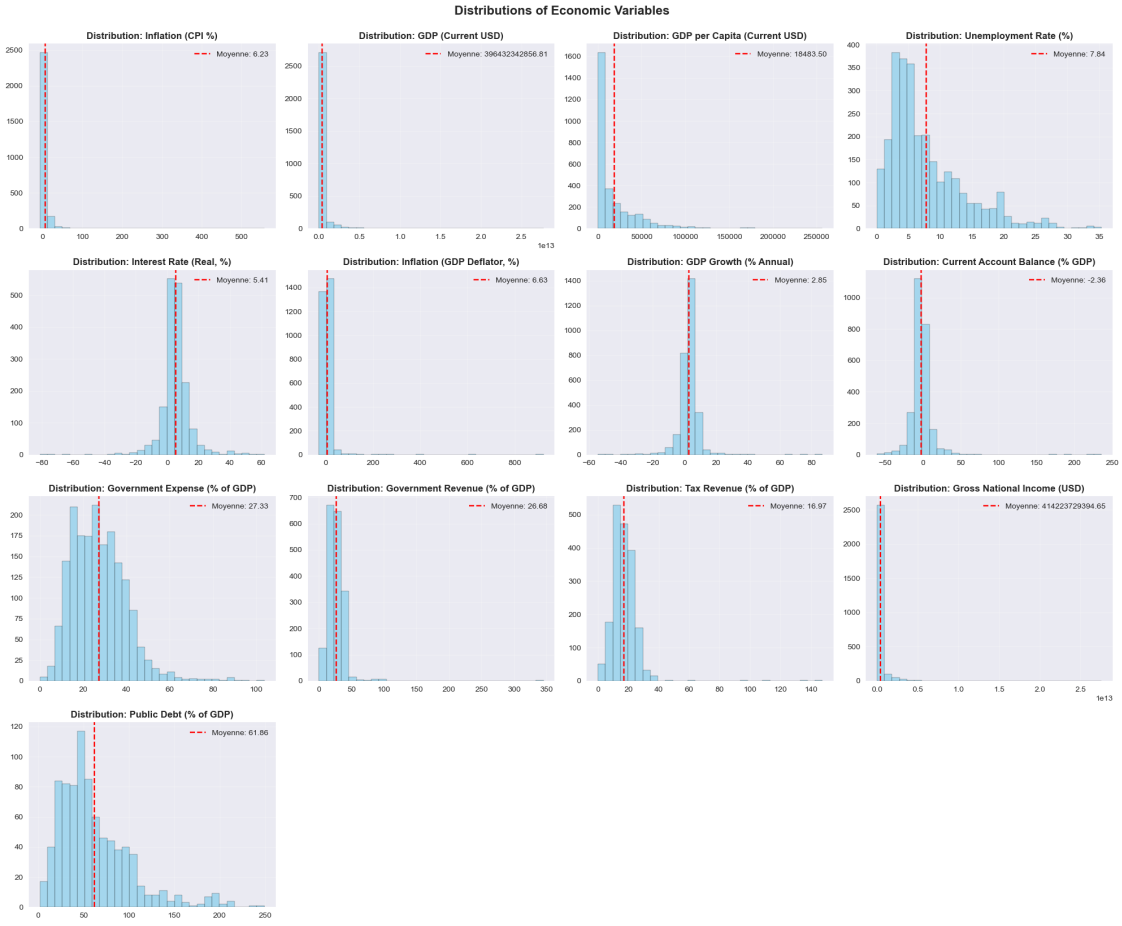
[199]:
```
descriptive_analysis(df)
```

```
DESCRIPTIVE STATISTICS
```

```
------------------------------------------------
       Inflation (CPI %)  GDP (Current USD)  GDP per Capita (Current USD)  \
count            2694.00       2.933000e+03                       2938.00
mean                6.23       3.964323e+11                      18483.50
std                19.73       1.749315e+12                      27301.81
min                -6.69       3.210541e+07                        193.01
25%                 1.40       6.264757e+09                       2280.75
50%                 3.21       2.587360e+10                       6827.67
75%                 6.19       1.874939e+11                      23727.02
max               557.20       2.772071e+13                     256580.52

       Unemployment Rate (%)  Interest Rate (Real, %)  \
count                2795.00                  1735.00
mean                    7.84                     5.41
std                     5.96                     9.74
min                     0.10                   -81.13
25%                     3.61                     1.73
50%                     5.77                     5.08
75%                    10.73                     8.87
max                    35.36                    61.88

       Inflation (GDP Deflator, %)  GDP Growth (% Annual)  \
count                      2904.00                2912.00
mean                          6.63                   2.85
std                          25.82                   6.05
min                         -28.76                 -54.34
25%                           1.22                   1.00
50%                           3.22                   3.10
75%                           6.91                   5.36
max                         921.54                  86.83

       Current Account Balance (% GDP)  Government Expense (% of GDP)  \
count                          2563.00                        1820.00
mean                             -2.36                          27.33
std                              13.74                          12.64
min                             -60.88                           0.00
25%                              -7.50                          17.51
50%                              -2.66                          26.00
75%                               1.85                          34.88
max                             235.75                         103.73

       Government Revenue (% of GDP)  Tax Revenue (% of GDP)  \
count                        1829.00                 1833.00
mean                           26.68                   16.97
std                            18.12                    8.22
min                             0.00                    0.00
25%                            17.64                   12.29
50%                            24.82                   16.32
```

```
75%                          32.70                        21.45
max                         345.00                       147.64
```

```
       Gross National Income (USD)  Public Debt (% of GDP)
count                 2.796000e+03                  852.00
mean                  4.142237e+11                   61.86
std                   1.799783e+12                   40.41
min                   5.107533e+07                    1.85
25%                   7.475538e+09                   33.89
50%                   2.986520e+10                   51.65
75%                   1.972529e+11                   81.93
max                   2.757614e+13                  249.37
```

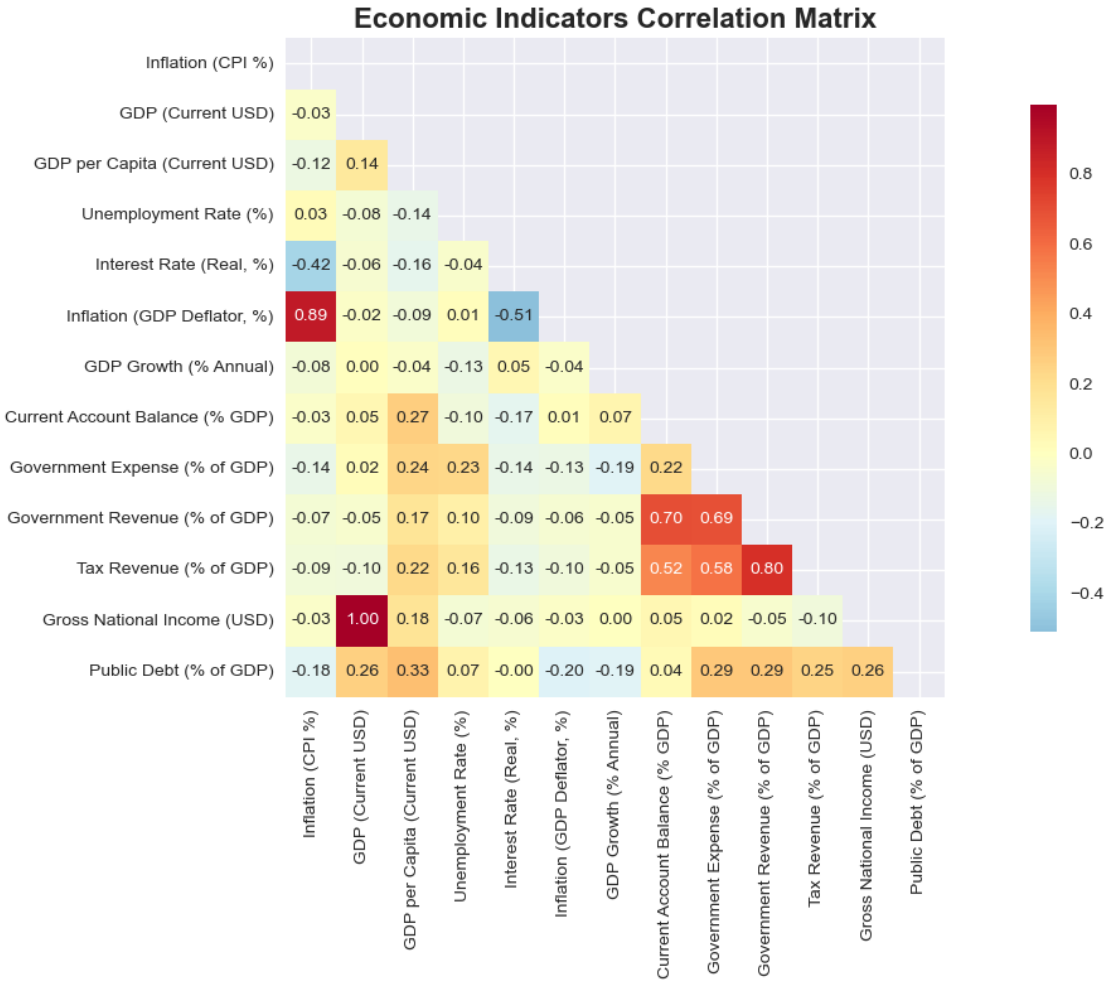


[201]:
```
#Correlation Analysis
def correlation_analysis(df):
    """
    Correlation analysis between variables
    """
```

```
    print("Correlation Analysis")
    print("-" * 50)

    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    numeric_cols = [col for col in numeric_cols if col != 'year']

    # Matrice de corrélation
    corr_matrix = df[numeric_cols].corr()

    # Heatmap des corrélations
    plt.figure(figsize=(14, 8))
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    sns.heatmap(corr_matrix, annot=True, cmap='RdYlBu_r', center=0,
                square=True, mask=mask, fmt='.2f', cbar_kws={"shrink": .8})
    plt.title('Economic Indicators Correlation Matrix', fontsize=16,␣
↪fontweight='bold')
    plt.tight_layout()
    plt.show()

    # Top corrélations
    corr_pairs = []
    for i in range(len(corr_matrix.columns)):
        for j in range(i+1, len(corr_matrix.columns)):
            corr_pairs.append({
                'Variable_1': corr_matrix.columns[i],
                'Variable_2': corr_matrix.columns[j],
                'Correlation': corr_matrix.iloc[i, j]
            })

    corr_df = pd.DataFrame(corr_pairs)
    corr_df = corr_df.sort_values('Correlation', key=abs, ascending=False)

    print("Top 10 Strongest Correlations:")
    print(corr_df.head(10))
```

[203]:
```
correlation_analysis(df)
```

```
Correlation Analysis
--------------------------------------------------
```



Top 10 Strongest Correlations:

	Variable_1	Variable_2	Correlation
21	GDP (Current USD)	Gross National Income (USD)	0.999904
4	Inflation (CPI %)	Inflation (GDP Deflator, %)	0.887656
72	Government Revenue (% of GDP)	Tax Revenue (% of GDP)	0.798062
64	Current Account Balance (% GDP)	Government Revenue (% of GDP)	
68	Government Expense (% of GDP)	Government Revenue (% of GDP)	
69	Government Expense (% of GDP)	Tax Revenue (% of GDP)	
65	Current Account Balance (% GDP)	Tax Revenue (% of GDP)	
42	Interest Rate (Real, %)	Inflation (GDP Deflator, %)	
3	Inflation (CPI %)	Interest Rate (Real, %)	
32	GDP per Capita (Current USD)	Public Debt (% of GDP)	

```
64     0.696132
68     0.693013
69     0.578602
65     0.521580
42    -0.508959
3     -0.417850
32     0.331592
```

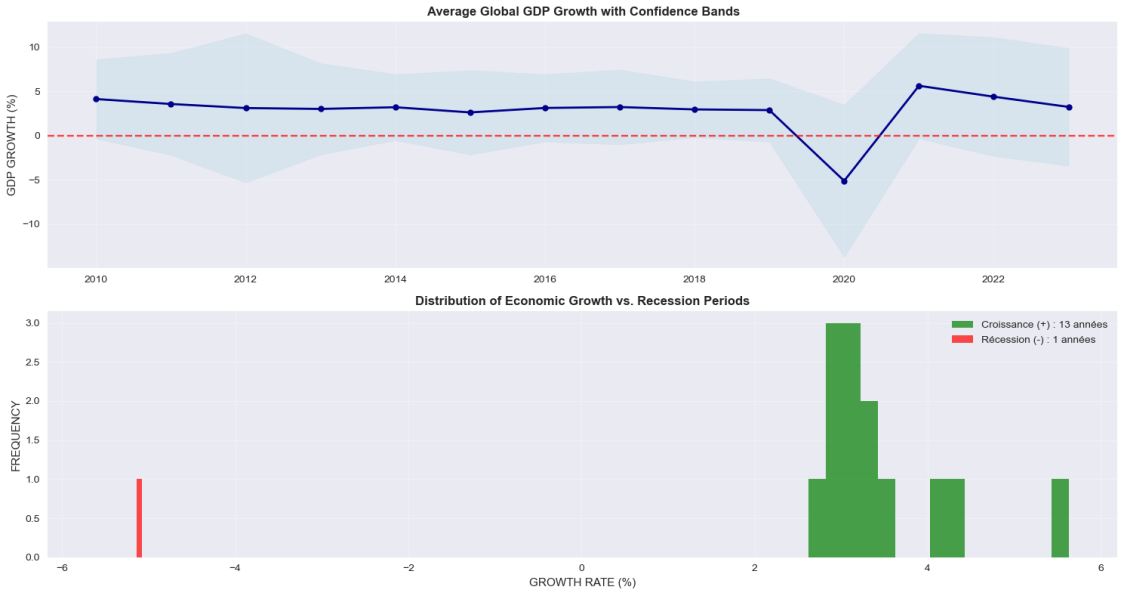
[205]:
```
#Importing all necessary modules
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
#import country_converter as coco

# Configuration
warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

# Configuration Plotly pour Kaggle
import plotly.io as pio
pio.renderers.default = "notebook"
```

[215]:
```
#Economic Cycle Analysis
def economic_cycles_analysis(df):
    """
    Analysis of Economic Cycles
    """
    print("ANALYSIS OF ECONOMIC CYCLES")
    print("-" * 50)

    # Focus sur la croissance du PIB
    gdp_growth_data = df.groupby('year')['GDP Growth (% Annual)'].agg(['mean',␣
 ↪'std', 'count'])
    gdp_growth_data = gdp_growth_data.dropna()

    plt.figure(figsize=(15, 8))

    # Graphique principal
    plt.subplot(2, 1, 1)
    plt.plot(gdp_growth_data.index, gdp_growth_data['mean'],
             marker='o', linewidth=2, markersize=6, color='darkblue')
```

```
    plt.fill_between(gdp_growth_data.index,
                     gdp_growth_data['mean'] - gdp_growth_data['std'],
                     gdp_growth_data['mean'] + gdp_growth_data['std'],
                     alpha=0.3, color='lightblue')
    plt.axhline(y=0, color='red', linestyle='--', alpha=0.7)
    plt.title(' Average Global GDP Growth with Confidence Bands',␣
↪fontweight='bold')
    plt.ylabel('GDP GROWTH (%)')
    plt.grid(True, alpha=0.3)

    # Histogramme des périodes de récession/croissance
    plt.subplot(2, 1, 2)
    positive_growth = gdp_growth_data[gdp_growth_data['mean'] > 0]['mean']
    negative_growth = gdp_growth_data[gdp_growth_data['mean'] <= 0]['mean']

    plt.hist(positive_growth, bins=15, alpha=0.7, color='green',␣
↪label=f'Croissance (+) : {len(positive_growth)} années')
    plt.hist(negative_growth, bins=15, alpha=0.7, color='red',␣
↪label=f'Récession (-) : {len(negative_growth)} années')
    plt.title('Distribution of Economic Growth vs. Recession Periods',␣
↪fontweight='bold')
    plt.xlabel('GROWTH RATE (%)')
    plt.ylabel('FREQUENCY')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()
```

[217]:
```
economic_cycles_analysis(df)
```

```
ANALYSIS OF ECONOMIC CYCLES
--------------------------------------------------
```

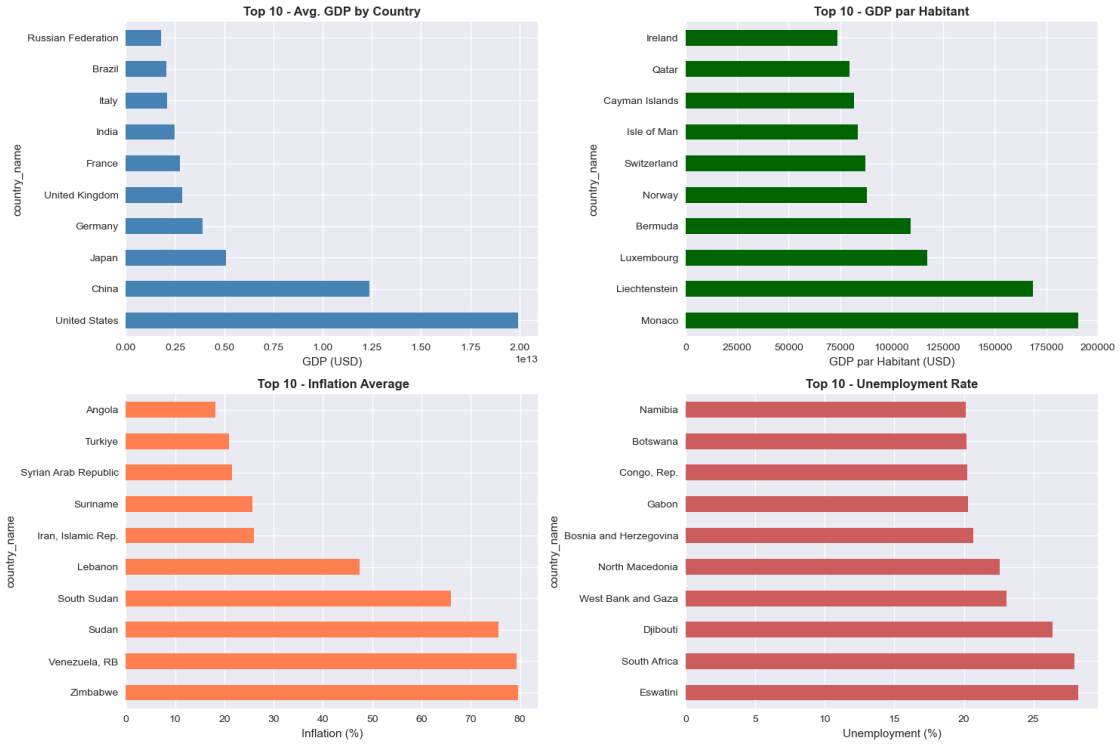


[219]:
```python
#Geographical Analysis
def geographical_analysis(df):
    """
    Geographic Analysis perRegion
    """
    print("GEOGRAPHIC ANALYSIS")
    print("-" * 50)

    # Ajouter les régions
    # df['region'] = df['country_name'].map(country_to_region_mapping)

    # Top 10 des pays par PIB moyen
    top_gdp_countries = df.groupby('country_name')['GDP (Current USD)'].mean().
↪sort_values(ascending=False).head(10)

    plt.figure(figsize=(15, 10))

    # Graphique en barres horizontales
    plt.subplot(2, 2, 1)
    top_gdp_countries.plot(kind='barh', color='steelblue')
    plt.title('Top 10 - Avg. GDP by Country', fontweight='bold')
    plt.xlabel('GDP (USD)')

    # PIB par habitant
    top_gdp_per_capita = df.groupby('country_name')['GDP per Capita (Current␣
↪USD)'].mean().sort_values(ascending=False).head(10)
```

```
    plt.subplot(2, 2, 2)
    top_gdp_per_capita.plot(kind='barh', color='darkgreen')
    plt.title('Top 10 - GDP par Habitant', fontweight='bold')
    plt.xlabel('GDP par Habitant (USD)')

    # Inflation moyenne
    top_inflation = df.groupby('country_name')['Inflation (CPI %)'].mean().
↪sort_values(ascending=False).head(10)

    plt.subplot(2, 2, 3)
    top_inflation.plot(kind='barh', color='coral')
    plt.title(' Top 10 - Inflation Average', fontweight='bold')
    plt.xlabel('Inflation (%)')

    # Chômage moyen
    top_unemployment = df.groupby('country_name')['Unemployment Rate (%)'].
↪mean().sort_values(ascending=False).head(10)

    plt.subplot(2, 2, 4)
    top_unemployment.plot(kind='barh', color='indianred')
    plt.title('Top 10 - Unemployment Rate', fontweight='bold')
    plt.xlabel('Unemployment (%)')

    plt.tight_layout()
    plt.show()
```

[221]:
```
geographical_analysis(df)
```

```
GEOGRAPHIC ANALYSIS
--------------------------------------------------
```



[23]:
```
#DATA PREPARATION AND CLEANING
df = df.drop(columns=['Interest Rate (Real, %)'])
```

[25]:
```
#DATA PREPARATION AND CLEANING
df = df.dropna()
```

[27]:
```
#DATA PREPARATION AND CLEANING
df = df.drop(columns=['country_id', 'country_name', 'year'])
```

[29]:
```
#DATA PREPARATION AND CLEANING
X=df.drop('Government Revenue (% of GDP)',axis=1)
y=df['Government Revenue (% of GDP)']
```

[31]:
```
# DATA PREPARATION AND CLEANING
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

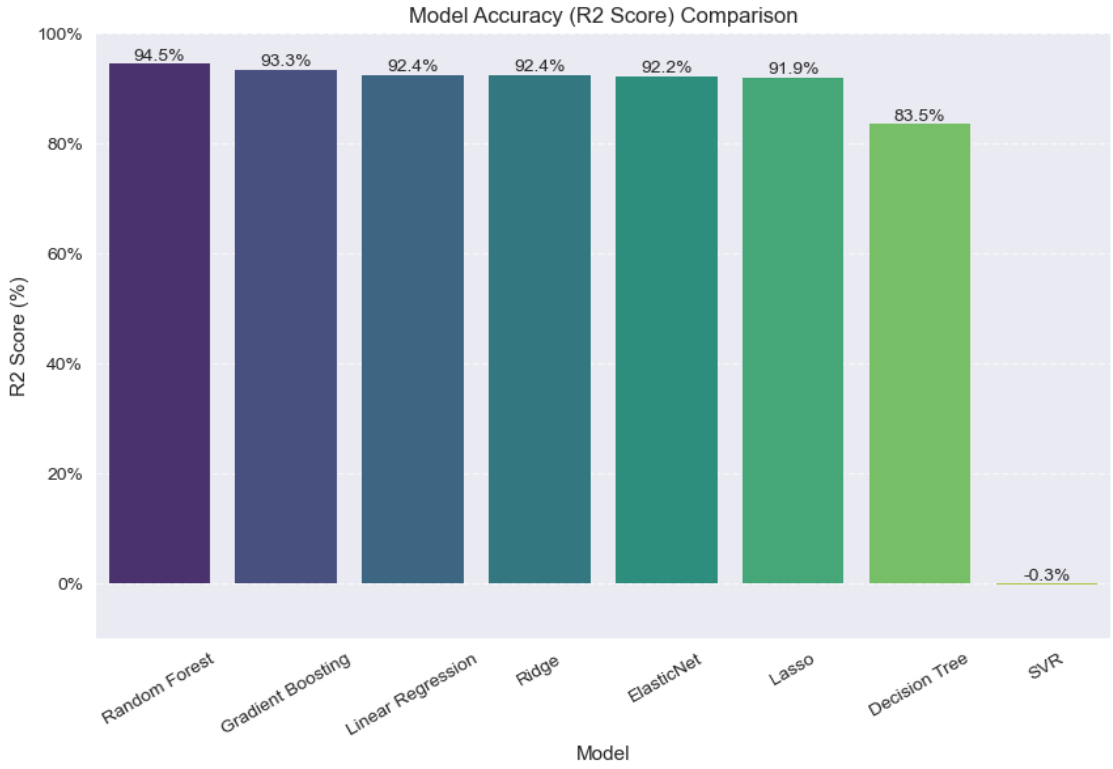
[33]:
```
#MODEL TRAINING & EVALUATION
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
```

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "ElasticNet": ElasticNet(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "SVR": SVR()
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results.append({
        "Model": name,
        "MSE": mse,
        "MAE": mae,
        "R2": r2
    })

results_df = pd.DataFrame(results).sort_values(by="R2", ascending=False)
print(results_df)
```

```
               Model         MSE       MAE        R2
5      Random Forest    4.445608  1.347680  0.944644
6  Gradient Boosting    5.359111  1.600407  0.933269
0  Linear Regression    6.074939  1.974870  0.924356
1              Ridge    6.075025  1.974901  0.924355
3         ElasticNet    6.253538  2.021869  0.922132
2              Lasso    6.477820  2.059282  0.919339
4      Decision Tree   13.233082  1.790987  0.835224
7                SVR   80.525774  7.348224 -0.002691
```

[35]:
```
#MODEL TRAINING & EVALUATION
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='Model', y='R2', data=results_df, palette='viridis')
plt.title('Model Accuracy (R2 Score) Comparison')
plt.ylabel('R2 Score (%)')
```

```
plt.xlabel('Model')
plt.ylim(-0.1, 1.0)
plt.xticks(rotation=30)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show percentage on top of bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height*100:.1f}%',
                (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontsize=10)

# Format y-axis as percentage
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.0f}%'.
 ↪format(y*100)))
plt.show()
```



[37]:
```
#Hyperparameter using GridSearchCV

from sklearn.model_selection import GridSearchCV

param_grid = [
```

```
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(forest_reg, param_grid,
                           scoring='neg_mean_squared_error',
                           return_train_score=True,
                           cv=10,
                          )

grid_search.fit(X_train, y_train)
```

[37]:
```
GridSearchCV(cv=10, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

[43]:
```
grid_search.best_params_
```

[43]:
```
{'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

[45]:
```
cv_scores = grid_search.cv_results_

##printing all the parameters along with their scores
for mean_score, params in zip(cv_scores['mean_test_score'],␣
 ↪cv_scores["params"]):
    print(np.sqrt(-mean_score), params)
```

```
3.8135496055128986 {'max_features': 2, 'n_estimators': 3}
3.0679054446530585 {'max_features': 2, 'n_estimators': 10}
2.891936477361396 {'max_features': 2, 'n_estimators': 30}
3.375796808448963 {'max_features': 4, 'n_estimators': 3}
2.8773083191688933 {'max_features': 4, 'n_estimators': 10}
2.783731958622698 {'max_features': 4, 'n_estimators': 30}
3.3110570229568204 {'max_features': 6, 'n_estimators': 3}
2.9494289127881688 {'max_features': 6, 'n_estimators': 10}
2.767398137867996 {'max_features': 6, 'n_estimators': 30}
3.127671095452778 {'max_features': 8, 'n_estimators': 3}
2.8308368237809347 {'max_features': 8, 'n_estimators': 10}
2.7765940524838606 {'max_features': 8, 'n_estimators': 30}
3.4833994347486925 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
2.747790506699974 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
3.3975226343338876 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
2.861291017421245 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
```

```
3.1181246592878895 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
2.737590806034221 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

[47]:
```
#Evaluating the entire system on Test Data
final_model = grid_search.best_estimator_

final_predictions = final_model.predict(X_test)
final_mse = mean_squared_error(y_test, y_pred)
final_rmse = np.sqrt(final_mse)
```

[49]:
```
final_rmse
```

[49]:
```
8.973615447768513
```

[69]:
```
final_model
```

[69]:
```
RandomForestRegressor(bootstrap=False, max_features=4, n_estimators=10,
                      random_state=42)
```

[185]:
```
#Custom function
def predict_GovRevnGDP(GeoEco_stats, model):
    if type(GeoEco_stats) == dict:
        df = pd.DataFrame(GeoEco_stats)
    else:
        df = GeoEco_stats
    #df = df.drop(columns=['Interest Rate (Real, %)'])
    #df = df.dropna()
    #df = df.drop(columns=['country_id', 'country_name', 'year'])
    X=df.drop('Government Revenue (% of GDP)',axis=1)
    y=df['Government Revenue (% of GDP)']
    results = []
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
    #y_pred = model.predict(X_test)
    #return y_pred
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred
    #return y_pred
    #mse = mean_squared_error(y_test, y_pred)
    #mae = mean_absolute_error(y_test, y_pred)
    #rmse = np.sqrt(mse)
    #r2 = r2_score(y_test, y_pred)
    #results.append({
    #    "MSE": mse,
    #    "MAE": mae,
    #    "R2": r2,
```

```
    #    "RMSE": rmse
    #})

    #results_df = pd.DataFrame(results).sort_values(by="R2", ascending=False)
    #print(results_df)
```

[113]:
```
import pickle
##saving the model
with open("./model_files/model.bin", 'wb') as f_out:
    pickle.dump(final_model, f_out)
    f_out.close()
##loading the model from the saved file
#with open('model.bin', 'rb') as f_in:
    #model = pickle.load(f_in)
```

[187]:
```
GeoEco_stats = {
    "Inflation (CPI %)" : [1.1,2.1,3.1,4.2,2.0,1.5],
    "GDP (Current USD)" : [208768,239768,134789,136789,285678,305678],
    "GDP per Capita (Current USD)" : [190.45,230.45,122.54,128.54,282.45,289.
 ↪45],
    "Unemployment Rate (%)" : [8.1,7.1,7.3,6.3,4.6,5.6],
    "Inflation (GDP Deflator, %)" : [-2.5,3.0,1.3,2.0,1.5,2.5],
    "GDP Growth (% Annual)" : [-2.6,3.2,2.5,1.3,-2.1,2.3],
    "Current Account Balance (% GDP)" : [-20.79,11.20,12.09,13.09,13.23,12.23],
    "Government Expense (% of GDP)" : [60,21.89,13.78,23.56,32.8,12.34],
    "Government Revenue (% of GDP)" : [9.01,10.03,8.0,9.2,9.0,10],
    "Tax Revenue (% of GDP)" : [10.01,11.60,13.45,15,13,16],
    "Gross National Income (USD)" : [190768,228765,133678,125467,274523,297845],
    "Public Debt (% of GDP)" : [70.01,65.23,40.23,55.67,23.08,45.09]
}

predict_GovRevnGDP(GeoEco_stats, final_model)
```

[187]:
```
array([8.48, 8.7 ])
```

[115]:
```
##loading the model from the saved file
import pickle
with open('./model_files/model.bin', 'rb') as f_in:
    model = pickle.load(f_in)
```

[171]:
```
import requests

url = "http://localhost:9696/predict"
r = requests.post(url, json = GeoEco_stats)
r.text.strip()
```

[171]:
```
'{\n  "GovRevnGDP_prediction": [\n    8.48,\n    8.7\n  ]\n}'
```

[189]:
```
import requests

url = "https://globaleco-flask-app-15275c2300f2.herokuapp.com/predict"
r = requests.post(url, json = GeoEco_stats)
r.text.strip()
```

[189]:
```
'{"GovRevnGDP_prediction":[8.48,8.7]}'
```

[ ]:
```

```