
Predictive Model on Macroeconomic Indicators

Overview:

This project aims to predict GDP growth using macroeconomic indicators as explanatory variables. The dataset includes multiple economic indicators, such as Trade Balance, Federal Funds Rate, Consumer Price Index (CPI), Retail Sales, and Unemployment Rate. By leveraging time series analysis and machine learning methods, the goal is to develop a robust predictive model that accurately forecasts GDP growth, providing insights into economic trends.

Predictive Task:

The primary task is to forecast GDP growth rates using time series and regression-based approaches. By understanding the relationships between GDP growth and macroeconomic factors, this project can contribute to economic planning, policy development, and market analysis.

Summary of Findings:

Linear regression models showed limited performance due to the complex relationships in the data. Basic SARIMA and SARIMAX models performed moderately well, accounting for seasonal patterns and exogenous regressors.

Random Forest and VAR models with engineered features demonstrated potential for improved performance, they do not perform very well when dealing with this task. The lasso model can be considered as another candidate to perform well in this task.

Future improvements could focus on advanced hyperparameter optimization and additional machine learning techniques.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import seaborn as sns
```

2.Data Description

Data Sources: The dataset was sourced from the publicly available databases of the Federal Reserve Bank, which provides a comprehensive repository of macroeconomic data. These datasets allow customization of time periods and intervals to suit specific analytical needs. For this project, the time interval was set to quarterly data to ensure consistency with macroeconomic reporting standards, spanning from 1992 to 2024. **Description of Variables:**

Trade Balance: Measures the difference between a country's exports and imports. **Rationale:** A positive trade balance contributes to GDP growth by boosting domestic production, while trade deficits may signal economic dependency on foreign goods.

Federal Funds Rate: Interest rate at which depository institutions lend reserve balances. **Rationale:** The Federal Funds Rate influences borrowing costs and investment decisions. Lower rates encourage spending and investment, driving economic growth, whereas higher rates aim to control inflation.

CPI (Consumer Price Index): Reflects changes in the price level of a basket of consumer goods and services. **Rationale:** Inflationary trends captured by CPI affect purchasing power, consumer confidence, and economic stability, all of which impact GDP growth.

Retail Sales: Proxy for consumer demand and spending trends. **Rationale:** Retail sales are a direct indicator of consumer activity, which drives a significant portion of economic growth. Higher retail spending typically reflects economic confidence and expansion.

Unemployment Rate: Indicates the percentage of unemployed individuals in the labor force. **Rationale:** The unemployment rate reflects the labor market's health. Lower unemployment rates often coincide with higher consumer spending and production output, fostering GDP growth.

```
urls = {  
    "Trade Balance": "https://raw.githubusercontent.com/F  
    "Fed Funds Rate": "https://raw.githubusercontent.com/  
    "GDP": "https://raw.githubusercontent.com/Fahadshaari  
    "CPI": "https://raw.githubusercontent.com/Fahadshaari  
    "Retail Sales": "https://raw.githubusercontent.com/Fa  
    "Unemployment": "https://raw.githubusercontent.com/Fa  
}
```

```

datasets = {key: pd.read_csv(url) for key, url in urls.it

for key, df in datasets.items():
    print(f"{key} ")
datasets

```



Trade Balance

Fed Funds Rate

GDP

CPI

Retail Sales

Unemployment

```

{'Trade Balance':      observation_date  BOPGSTB
0      1992-01-01  -1833.0
1      1992-04-01  -3284.0
2      1992-07-01  -3589.0
3      1992-10-01  -4364.0
4      1993-01-01  -4748.0
..      ...      ...
127     2023-10-01 -64667.0
128     2024-01-01 -67747.0
129     2024-04-01 -75075.0
130     2024-07-01 -77861.0
131     2024-10-01      NaN

```

[132 rows x 2 columns],

```

'Fed Funds Rate':      observation_date  DFF
0      1992-01-01  4.02
1      1992-04-01  3.77
2      1992-07-01  3.26
3      1992-10-01  3.03
4      1993-01-01  3.04
..      ...      ...
127     2023-10-01  5.33
128     2024-01-01  5.33
129     2024-04-01  5.33
130     2024-07-01  5.26
131     2024-10-01  4.71

```

[132 rows x 2 columns],

```

'GDP':      observation_date      GDP
0      1992-01-01  6363.102
1      1992-04-01  6470.763
2      1992-07-01  6566.641
3      1992-10-01  6680.803
4      1993-01-01  6729.459
..      ...      ...

```

126	2023-07-01	27967.697
127	2023-10-01	28296.967
128	2024-01-01	28624.069
129	2024-04-01	29016.714
130	2024-07-01	29354.321

[131 rows x 2 columns],

'CPI': observation_date MEDCPIM158SFRBCLE

0	1992-01-01	2.702885
1	1992-04-01	3.173175
2	1992-07-01	1.920145
3	1992-10-01	2.979265
4	1993-01-01	2.480115

...
127	2023-10-01	4.260233
128	2024-01-01	5.142661
129	2024-04-01	3.210819
130	2024-07-01	3.710619
131	2024-10-01	...

```
from statsmodels.tsa.stattools import adfuller

def calculate_growth_rate(df, column):
    df[f"{column}_growth"] = df[column].pct_change() * 10
    return df

df = datasets["GDP"]
df = calculate_growth_rate(df, "GDP")
df = df[["observation_date", "GDP_growth"]]

for key, dataset in datasets.items():
    if key != "GDP":
        df = df.merge(dataset, on="observation_date", how

df.rename(columns={
    "BOPGSTB": "Trade_Balance",
    "DFF": "Fed_Funds_Rate",
    "MEDCPIM158SFRBCLE": "CPI",
    "RSXFS": "Retail_Sales",
    "UNRATE": "Unemployment_Rate"
}, inplace=True)

df.dropna(inplace=True)
```

df



	observation_date	GDP_growth	Trade_Balance	Fed_Funds_Rate	CPI	Reta
1	1992-04-01	1.691958	-3284.0	3.77	3.173175	
2	1992-07-01	1.481711	-3589.0	3.26	1.920145	
3	1992-10-01	1.738514	-4364.0	3.03	2.979265	
4	1993-01-01	0.728296	-4748.0	3.04	2.480115	
5	1993-04-01	1.181076	-5914.0	3.00	3.206866	
...	
126	2023-07-01	1.871805	-62153.0	5.26	4.022556	
127	2023-10-01	1.177323	-64667.0	5.33	4.260233	
128	2024-01-01	1.155961	-67747.0	5.33	5.142661	
129	2024-04-01	1.371730	-75075.0	5.33	3.210819	
130	2024-07-01	1.163491	-77861.0	5.26	3.710619	

130 rows x 7 columns

3.Preprocessing:

Missing values were handled using forward-fill and backward-fill techniques to ensure continuity in the time series data without introducing artificial bias. This approach preserves the trend and periodic patterns necessary for accurate time-series modeling.

Non-stationary variables were transformed using first differencing, a widely accepted statistical technique to stabilize the mean and eliminate trends over time. This transformation ensures that the data satisfies the stationarity assumption required for models such as SARIMA and VAR, enabling reliable parameter estimation and forecasts.

Lagged features were initially not included during the early phases of the analysis. However, during exploratory data analysis and model evaluation, it became evident that lagged effects play a significant role in predicting GDP growth. The inclusion of lagged values notably improved the performance of the SARIMA and SARIMAX models by accounting for delayed relationships between macroeconomic indicators and GDP growth.

Growth rates were computed for the GDP variable to represent the percentage change over time. This transformation highlights economic expansions and contractions, providing a clearer and more interpretable target variable for forecasting models. Analyzing growth rates instead of absolute values aligns with economic research standards and enhances model performance by focusing on relative changes rather than raw values.

```
numeric_columns = df.select_dtypes(include=["float64", "int64"])
df_numeric = df[numeric_columns]
```

```
correlation_matrix = df_numeric.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

```
features = ["Trade_Balance", "Fed_Funds_Rate", "CPI", "Real_GDP_Growth"]
```



```

for feature in features:
    if feature in df_numeric.columns:
        plt.figure(figsize=(6, 4))
        plt.scatter(df[feature], df["GDP_growth"])
        plt.title(f"{feature} vs. GDP Growth")
        plt.xlabel(feature)
        plt.ylabel("GDP Growth")
        plt.grid()
        plt.show()

```



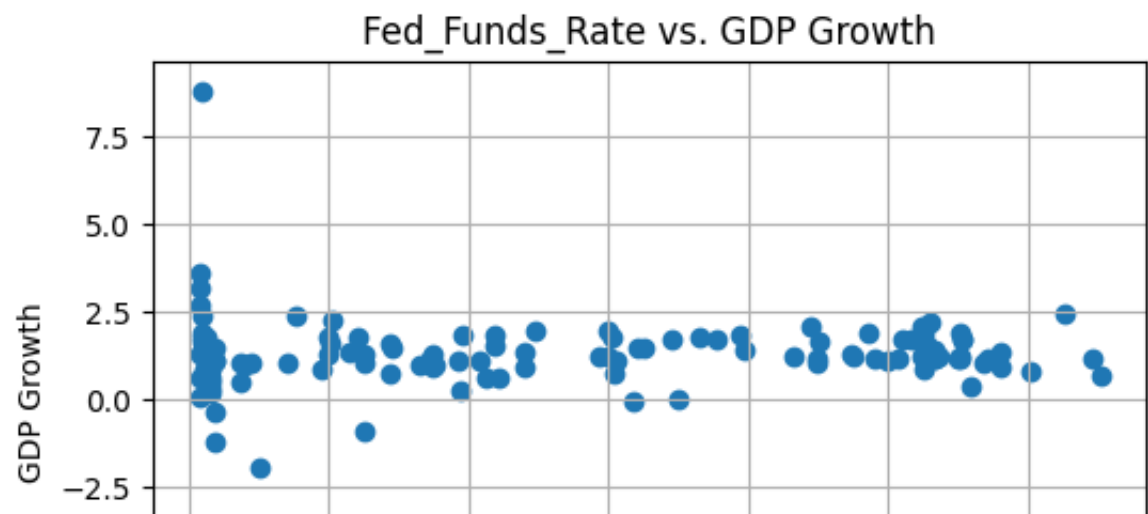
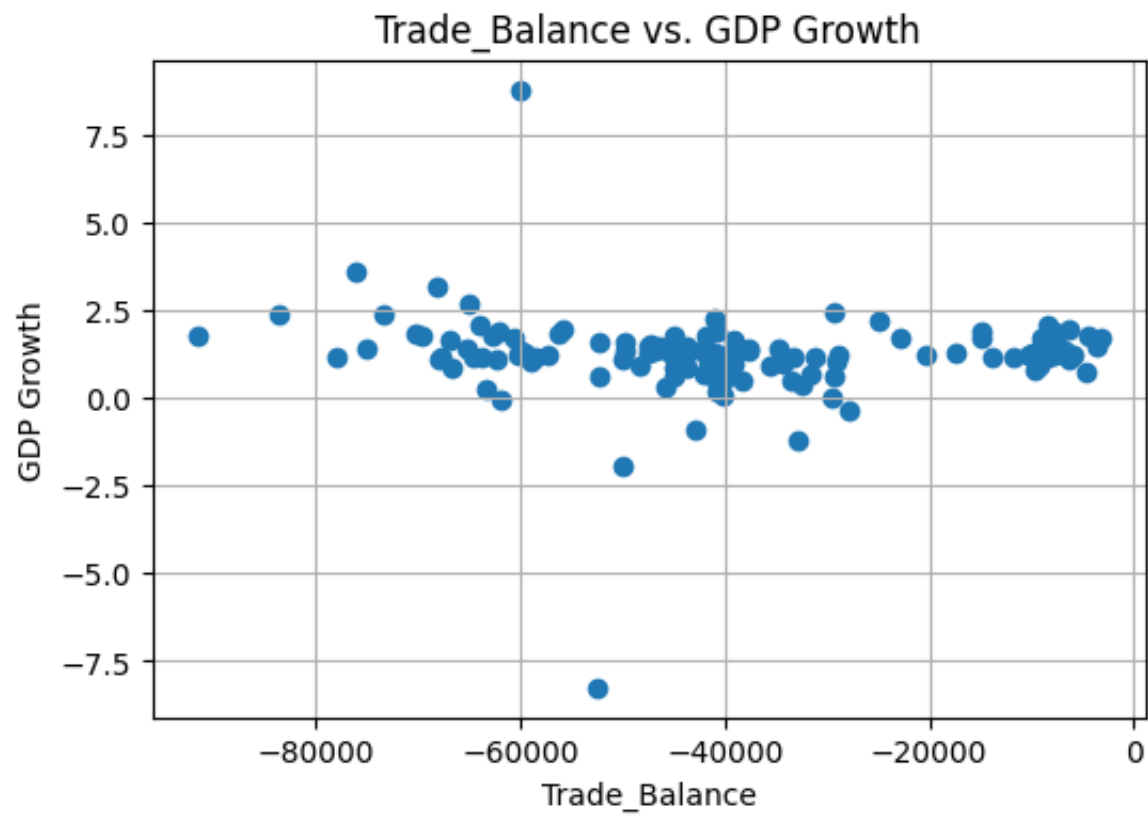
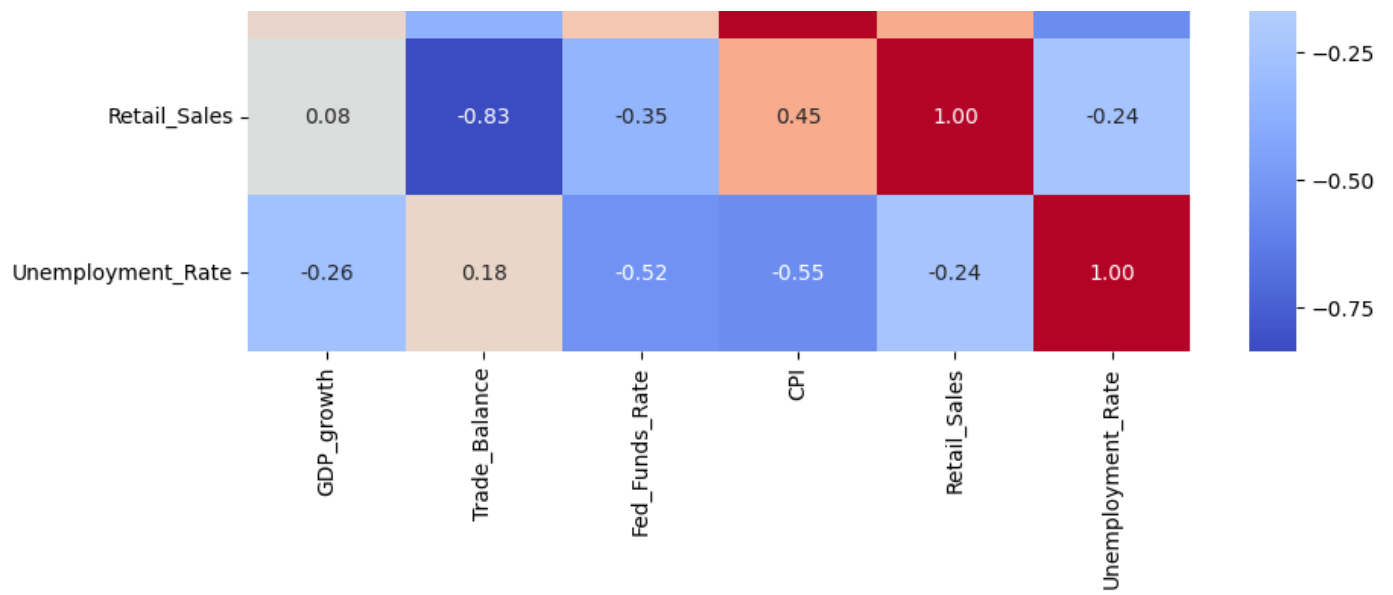
Correlation Matrix:

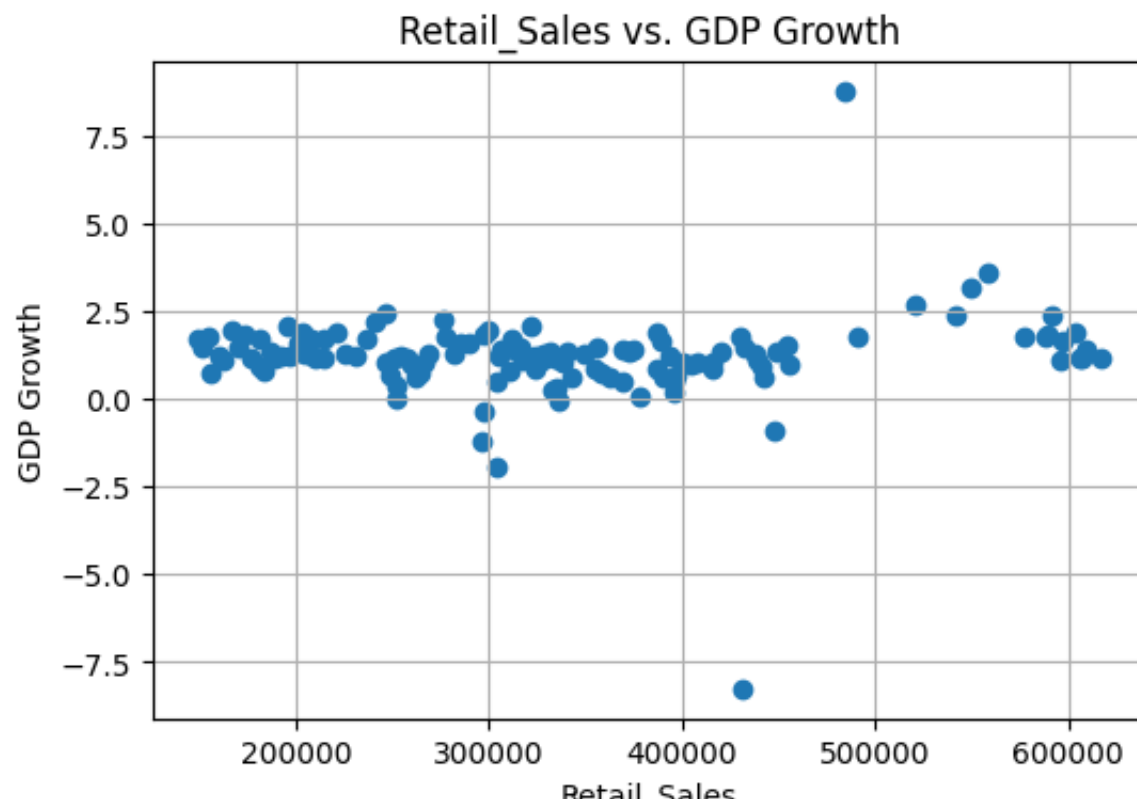
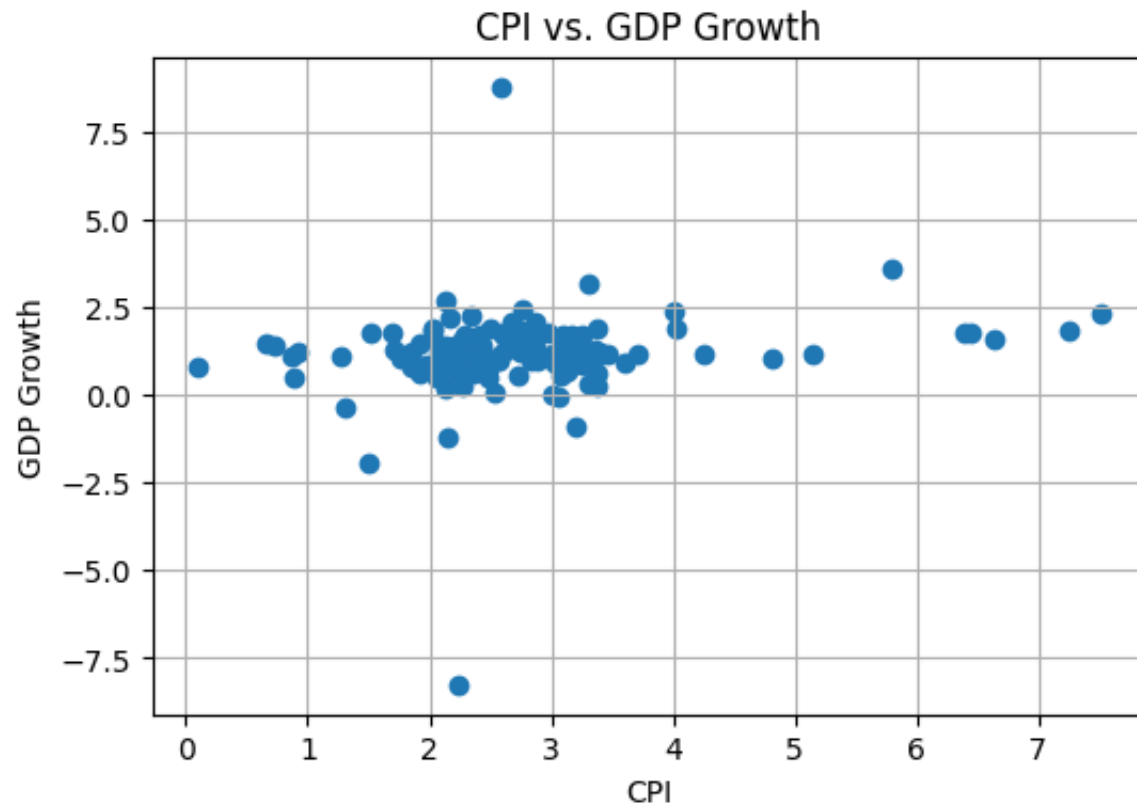
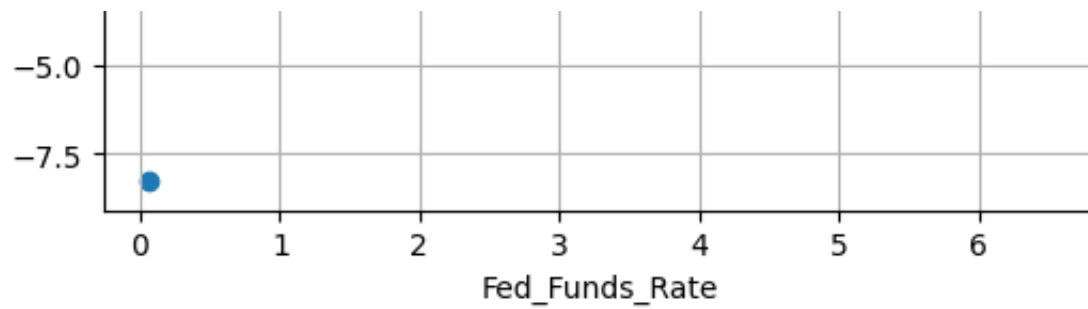
	GDP_growth	Trade_Balance	Fed_Funds_Rate	CPI	\
GDP_growth	1.000000	-0.060240	0.094925	0.175161	
Trade_Balance	-0.060240	1.000000	0.324947	-0.364639	
Fed_Funds_Rate	0.094925	0.324947	1.000000	0.296662	
CPI	0.175161	-0.364639	0.296662	1.000000	
Retail_Sales	0.076961	-0.834860	-0.347400	0.452273	
Unemployment_Rate	-0.264586	0.182870	-0.519846	-0.552091	

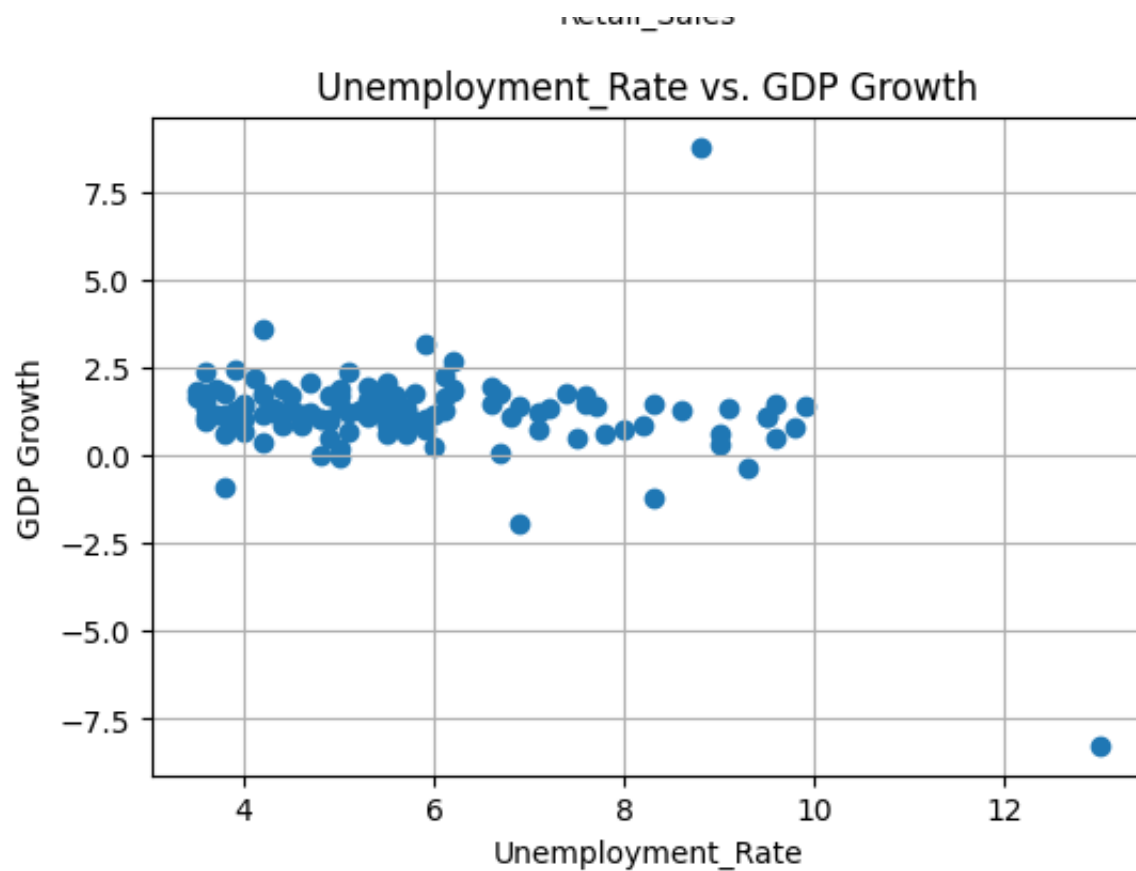
	Retail_Sales	Unemployment_Rate
GDP_growth	0.076961	-0.264586
Trade_Balance	-0.834860	0.182870
Fed_Funds_Rate	-0.347400	-0.519846
CPI	0.452273	-0.552091
Retail_Sales	1.000000	-0.243201
Unemployment_Rate	-0.243201	1.000000

Correlation Heatmap









stationary check

```
def adf_test(series, name):  
    result = adfuller(series.dropna())  
    print(f"ADF Test for {name}:")  
    print(f"ADF Statistic: {result[0]}")  
    print(f"p-value: {result[1]}")  
    print(f"Critical Values: {result[4]}")
```

```

    if result[1] <= 0.05:
        print(f"{name} is stationary.")
    else:
        print(f"{name} is not stationary.\n")

for column in df_numeric.columns:
    adf_test(df_numeric[column], column)

```

➞ ADF Test for GDP_growth:
ADF Statistic: -12.088682459280097
p-value: 2.1476805920933843e-22
Critical Values: {'1%': -3.482087964046026, '5%': -2.8842185101614626, '10%': -2.5810279615425865}
GDP_growth is stationary.

ADF Test for Trade_Balance:
ADF Statistic: -1.2105004663818748
p-value: 0.6690624436802756
Critical Values: {'1%': -3.482920063655088, '5%': -2.884580323367261, '10%': -2.5810279615425865}
Trade_Balance is not stationary.

ADF Test for Fed_Funds_Rate:
ADF Statistic: -2.6135187087371157
p-value: 0.09023529044090056
Critical Values: {'1%': -3.4825006939887997, '5%': -2.884397984161377, '10%': -2.5810279615425865}
Fed_Funds_Rate is not stationary.

ADF Test for CPI:
ADF Statistic: -3.604054923717561
p-value: 0.0056839778297261626
Critical Values: {'1%': -3.4833462346078936, '5%': -2.8847655969877666, '10%': -2.5810279615425865}
CPI is stationary.

ADF Test for Retail_Sales:
ADF Statistic: 1.7540908836935585
p-value: 0.9982534821155477
Critical Values: {'1%': -3.485585145896754, '5%': -2.885738566292665, '10%': -2.5810279615425865}
Retail_Sales is not stationary.

ADF Test for Unemployment_Rate:
ADF Statistic: -3.181101491588404
p-value: 0.021104992750650472
Critical Values: {'1%': -3.482087964046026, '5%': -2.8842185101614626, '10%': -2.5810279615425865}
Unemployment_Rate is stationary.

df



	GDP_growth	Trade_Balance	Fed_Funds_Rate	CPI	Retail_Sa
observation_date					
1992-04-01	1.691958	-3284.0	3.77	3.173175	1489
1992-07-01	1.481711	-3589.0	3.26	1.920145	1514
1992-10-01	1.738514	-4364.0	3.03	2.979265	1542
1993-01-01	0.728296	-4748.0	3.04	2.480115	1561
1993-04-01	1.181076	-5914.0	3.00	3.206866	1599
...
2023-07-01	1.871805	-62153.0	5.26	4.022556	6041
2023-10-01	1.177323	-64667.0	5.33	4.260233	6068
2024-01-01	1.155961	-67747.0	5.33	5.142661	6062
2024-04-01	1.371730	-75075.0	5.33	3.210819	6088
2024-07-01	1.163491	-77861.0	5.26	3.710619	6166

130 rows x 9 columns

Next steps:

[Generate code with df](#)



[View recommended plots](#)

[New interactive sheet](#)

A,address non stationary

reason: Several features are not stationary (e.g., Trade_Balance, Fed_Funds_Rate, and Retail_Sales). Non-stationary time series can lead to unreliable predictions in models like linear regression or SARIMA. Transform these series to make them stationary:

Here compute the first difference of non-stationary series,apply this to Trade_Balance, Fed_Funds_Rate, and Retail_Sales.

```
df['Trade_Balance_diff'] = df['Trade_Balance'].diff()
df['Fed_Funds_Rate_diff'] = df['Fed_Funds_Rate'].diff()
df['Retail_Sales_diff'] = df['Retail_Sales'].diff()

stationary_data = df[['Trade_Balance_diff', 'Fed_Funds_Rate_diff', 'Retail_Sales_diff', 'GDP_growth', 'CPI']]
```

df



	GDP_growth	Trade_Balance	Fed_Funds_Rate	CPI	Retail_Sales
observation_date					
1992-04-01	1.691958	-3284.0	3.77	3.173175	1489
1992-07-01	1.481711	-3589.0	3.26	1.920145	1514
1992-10-01	1.738514	-4364.0	3.03	2.979265	1542
1993-01-01	0.728296	-4748.0	3.04	2.480115	1561
1993-04-01	1.181076	-5914.0	3.00	3.206866	1599
...
2023-07-01	1.871805	-62153.0	5.26	4.022556	6041
2023-10-01	1.177323	-64667.0	5.33	4.260233	6068
2024-01-01	1.155961	-67747.0	5.33	5.142661	6062
2024-04-01	1.371730	-75075.0	5.33	3.210819	6088
2024-07-01	1.163491	-77861.0	5.26	3.710619	6166

130 rows x 9 columns

Next steps:

[Generate code with df](#)

[View recommended plots](#)

[New interactive sheet](#)

B, lr models

4.Models and Methods

4.1 Linear Regression:

Features: Differenced variables for Trade Balance, Federal Funds Rate, Retail Sales, CPI, and Unemployment Rate. Implementation: A linear regression model was utilized to estimate the coefficients of these explanatory variables. The model assumes a linear relationship between the features and GDP growth, relying on transformations to mitigate issues of non-stationarity and multicollinearity.

Performance: Mean Squared Error (MSE): 2.793979044859393 R² Score: 0.5465785377006258

Model Coefficients:Trade_Balance_diff: -0.000009;Fed_Funds_Rate_diff:

0.356024;Retail_Sales_diff: 0.000075;CPI: 0.004852;Unemployment_Rate: -0.087953

The moderate R² score suggests the model captures some variance in GDP growth, but the results highlight limitations in capturing the full complexity of macroeconomic dynamics. Certain features, such as the Federal Funds Rate and CPI, exhibited stronger predictive relationships, while others showed weaker or negligible impacts. Further feature engineering and exploration of non-linear models may improve predictive performance.

df.index

```
⇒ DatetimeIndex(['1992-04-01', '1992-07-01', '1992-10-01', '1993-01-01',  
                '1993-04-01', '1993-07-01', '1993-10-01', '1994-01-01',  
                '1994-04-01', '1994-07-01',  
                ...  
                '2022-04-01', '2022-07-01', '2022-10-01', '2023-01-01',  
                '2023-04-01', '2023-07-01', '2023-10-01', '2024-01-01',  
                '2024-04-01', '2024-07-01'],  
                dtype='datetime64[ns]', name='observation_date', length=130,  
                freq=None)
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
split_date = '2018-01-01'
```

```
train_data = df[df.index < split_date]
```



```
test_data = df[df.index >= split_date]

features = ['Trade_Balance_diff', 'Fed_Funds_Rate_diff',
target = 'GDP_growth'

X_train = train_data[features].fillna(method='ffill').fil
y_train = train_data.loc[X_train.index, target]
X_test = test_data[features].fillna(method='ffill').filln
y_test = test_data.loc[X_test.index, target]

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

y_pred = lin_reg.predict(X_test)

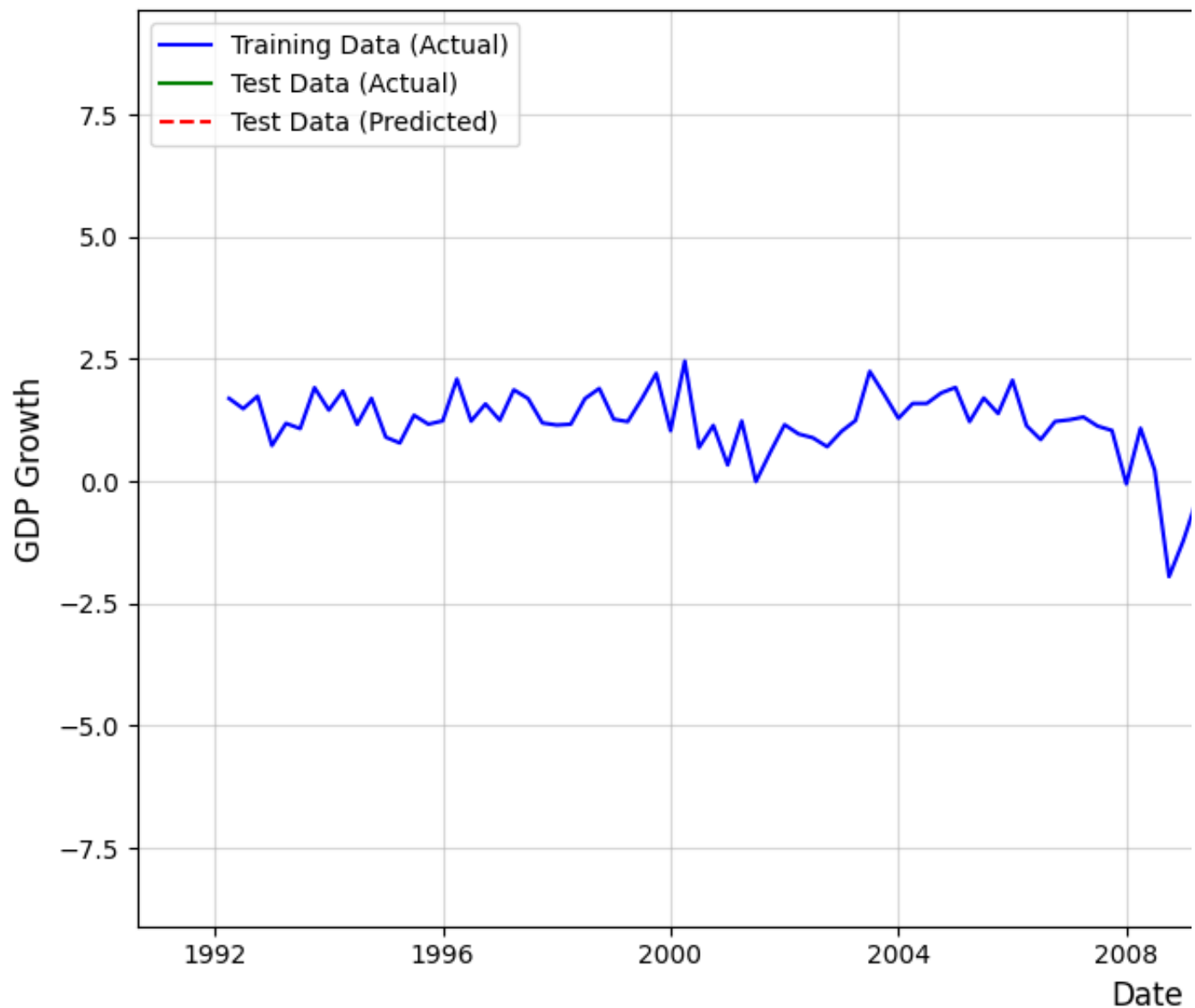
plt.figure(figsize=(12, 6))
plt.plot(train_data.index, y_train, label="Training Data
plt.plot(test_data.index, y_test, label="Test Data (Actua
plt.plot(test_data.index, y_pred, label="Test Data (Predi
plt.title("GDP Growth Prediction: Actual vs Predicted", f
plt.xlabel("Date", fontsize=12)
plt.ylabel("GDP Growth", fontsize=12)
plt.legend()
plt.grid(alpha=0.5)
plt.tight_layout()
plt.show()
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression Performance:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R^2 Score: {r2}")
```

```
coefficients = pd.DataFrame({
    'Feature': features,
    'Coefficient': lin_reg.coef_
})
print("Model Coefficients:")
print(coefficients)
```

```
<ipython-input-163-eee6042e9b01>:16: FutureWarning: DataFrame.fillna with 'met
X_train = train_data[features].fillna(method='ffill').fillna(method='bfill')
<ipython-input-163-eee6042e9b01>:18: FutureWarning: DataFrame.fillna with 'met
X_test = test_data[features].fillna(method='ffill').fillna(method='bfill')
```

GDP Growth Prediction:



Linear Regression Performance:

Mean Squared Error (MSE): 2.793979044859393

R² Score: 0.5465785377006258

Model Coefficients:

	Feature	Coefficient
0	Trade_Balance_diff	-0.000009
1	Fed_Funds_Rate_diff	0.356024
2	Retail_Sales_diff	0.000075
3	CPI	0.004852
4	Unemployment_Rate	-0.087953

C, time series models

4.2 Basic SARIMA Model:

Focused on modeling GDP growth as a time series with stationary transformations but without external regressors or lagged features.

Parameter Tuning:

order: Optimized for autoregressive, differencing, and moving average terms.

Performance: Train MSE: 0.6451550069816856 R^2 Score (Train): -0.5370766295810638 Test MSE: 2.9963239340986725 R^2 Score (Test): 0.5137409558524425 Train Performance: MSE: 0.645 reflects a low average prediction error. R^2 Score: -0.537 indicates underfitting, where the model fails to capture sufficient patterns in the training data, potentially due to its simplicity or insufficient explanatory variables. Test Performance: MSE: 2.996 suggests greater errors on unseen data. R^2 Score: 0.514 demonstrates some ability to explain variance in GDP growth during testing, though not exceptionally strong. Overall Implications: Stationary transformations helped improve model reliability, but the absence of lagged features limits its ability to capture temporal dependencies. The positive R^2 on the test set suggests moderate effectiveness, but underfitting on the training set indicates a need for better feature representation or complexity adjustments.

4.3 SARIMAX:

4.3.1 Basic SARIMA Model:

Focused on modeling GDP growth as a time series with stationary transformations but without external regressors or lagged features.

Parameter Tuning: Best Parameters (no lag): {'order': (3, 1, 1), 'seasonal_order': (0, 1, 0, 4)} achieved the best MSE of 2.5755 after grid search.

4.3.2 SARIMAX (Seasonal ARIMA with Exogenous Regressors):

Captures both seasonal and time-dependent patterns while incorporating external regressors.

Parameter Tuning: Best Parameters (with lag): {'order': (0, 0, 0), 'seasonal_order': (0, 1, 0, 4)} achieved the best MSE of 0.5484 after grid search, and its R^2 Score of 0.54657 is relatively high which explains a relatively large proportion variance in the data as well. Lagged features were included to enhance the model's ability to capture delayed relationships between variables and GDP growth. This significantly improved performance by accounting for time dependencies that otherwise would not be modeled effectively.

Comparison and Implications: SARIMAX with lagged features outperformed the SARIMA model without lag in terms of both best MSE (0.5484 vs. 2.5755) and overall model robustness. Including lagged features allowed the SARIMAX model to capture delayed relationships between macroeconomic variables and GDP growth, leading to significantly improved performance on the training data.

SARIMA without lag, while capturing some seasonal patterns, failed to account for temporal interdependencies in the data. Its reliance on immediate values and stationary transformations limited its ability to model complex time-lagged effects inherent in economic dynamics.

Key Insight: The inclusion of lagged features demonstrates the importance of feature engineering in improving predictive accuracy, particularly for economic time series data where past values exert a delayed influence on outcomes. Models that fail to incorporate such information risk underestimating key relationships and producing suboptimal predictions.

3.1 sarimax

df



	GDP_growth	Trade_Balance	Fed_Funds_Rate	CPI	Retail_Sa
observation_date					
1992-04-01	1.691958	-3284.0	3.77	3.173175	1489
1992-07-01	1.481711	-3589.0	3.26	1.920145	1514
1992-10-01	1.738514	-4364.0	3.03	2.979265	1542
1993-01-01	0.728296	-4748.0	3.04	2.480115	1561
1993-04-01	1.181076	-5914.0	3.00	3.206866	1599
...
2023-07-01	1.871805	-62153.0	5.26	4.022556	6041
2023-10-01	1.177323	-64667.0	5.33	4.260233	6068
2024-01-01	1.155961	-67747.0	5.33	5.142661	6062
2024-04-01	1.371730	-75075.0	5.33	3.210819	6088
2024-07-01	1.163491	-77861.0	5.26	3.710619	6166

130 rows × 9 columns

Next steps:

[Generate code with df](#)

[View recommended plots](#)

[New interactive sheet](#)

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

y = df['GDP_growth'].dropna()
X = df[['Trade_Balance_diff', 'Fed_Funds_Rate_diff', 'Ret

common_index = y.index.intersection(X.index)
y = y.loc[common_index]
X = X.loc[common_index]

train_mask = y.index < '2018-01-01'
test_mask = y.index >= '2018-01-01'

y_train = y[train_mask]
y_test = y[test_mask]
```

```
X_train = X[train_mask]
X_test = X[test_mask]

sarimax_model = SARIMAX(
    y_train,
    exog=X_train,
    order=(1, 1, 1),
    seasonal_order=(1, 1, 0, 4)
)
sarimax_results = sarimax_model.fit()

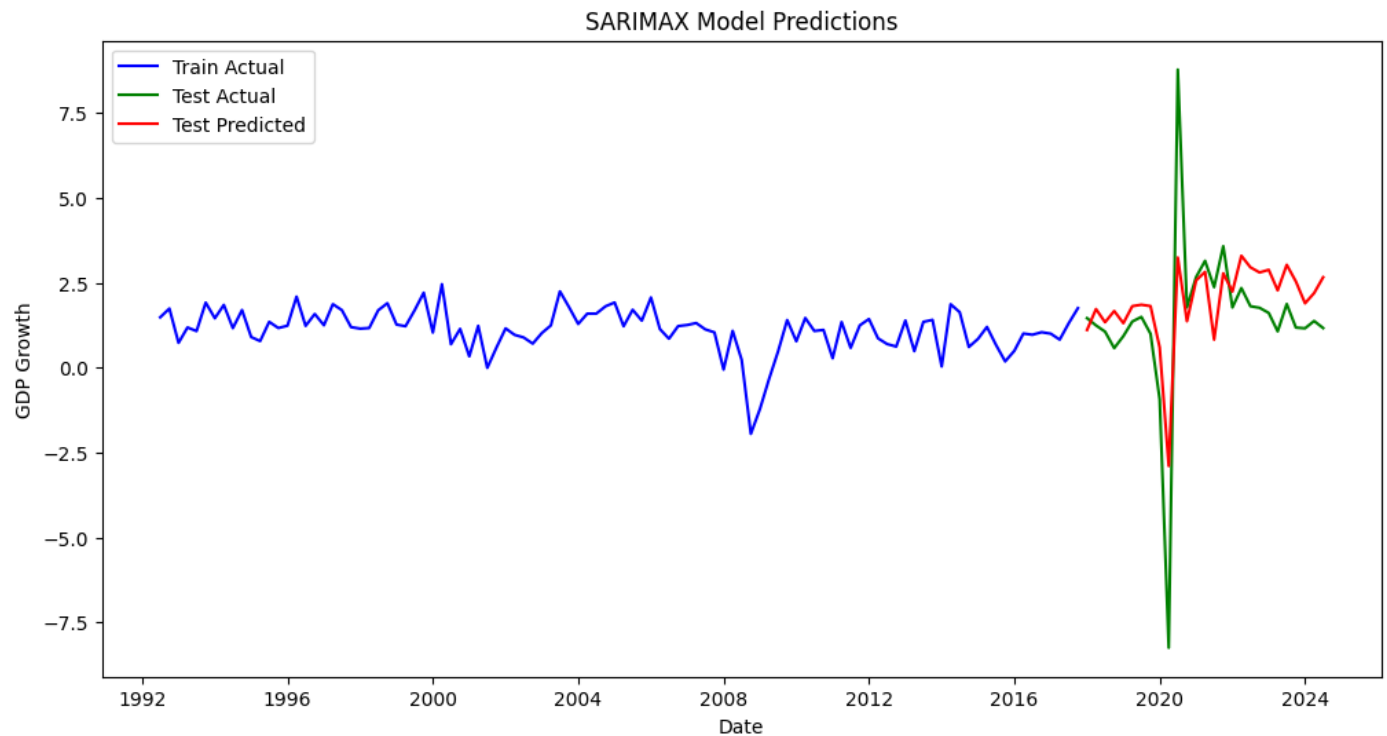
y_train_pred = sarimax_results.predict(start=0, end=len(y_train)-1)
y_test_pred = sarimax_results.predict(start=len(y_train), end=len(y_train)+len(y_test)-1)

sarimax_mse_train = mean_squared_error(y_train, y_train_pred)
sarimax_r2_train = r2_score(y_train, y_train_pred)
sarimax_mse_test = mean_squared_error(y_test, y_test_pred)
sarimax_r2_test = r2_score(y_test, y_test_pred)

plt.figure(figsize=(12, 6))
plt.plot(y_train.index, y_train, label='Train Actual', color='blue')
plt.plot(y_test.index, y_test, label='Test Actual', color='red')
plt.plot(y_test.index, y_test_pred, label='Test Predicted', color='green')
plt.title('SARIMAX Model Predictions')
plt.xlabel('Date')
plt.ylabel('GDP Growth')
plt.legend()
plt.show()

print("SARIMAX Train Performance:")
print(f"Mean Squared Error (MSE): {sarimax_mse_train}")
print(f"R^2 Score: {sarimax_r2_train}")
print("SARIMAX Test Performance:")
print(f"Mean Squared Error (MSE): {sarimax_mse_test}")
print(f"R^2 Score: {sarimax_r2_test}")
```

```
→ /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
```



SARIMAX Train Performance:
Mean Squared Error (MSE): 0.6451550069816856
 R^2 Score: -0.5370766295810638
SARIMAX Test Performance:
Mean Squared Error (MSE): 2.9963239340986725
 R^2 Score: 0.5137409558524425

this is a relatively large mse, we can do something to decrease it.

We try to grid search the parameters.


```
print(df.corr()['GDP_growth'].sort_values())
```

```
➡ Unemployment_Rate      -0.264586  
   Trade_Balance_diff     -0.083719  
   Trade_Balance          -0.060240  
   Retail_Sales           0.076961  
   Fed_Funds_Rate         0.094925  
   CPI                    0.175161  
   Fed_Funds_Rate_diff    0.374232  
   Retail_Sales_diff      0.728975  
   GDP_growth             1.000000  
   Name: GDP_growth, dtype: float64
```

we use grid search to search for favourable parameters

```
train_mask = y.index < '2018-01-01'  
test_mask = y.index >= '2018-01-01'  
  
y_train = y[train_mask]  
y_test = y[test_mask]  
X_train = X[train_mask]  
X_test = X[test_mask]  
  
param_grid = {  
    'order': [  
        (0, 0, 0), (1, 0, 0), (0, 1, 1),  
        (1, 1, 1), (2, 1, 1), (3, 1, 1), (2, 1, 2)  
    ],  
    'seasonal_order': [  
        (0, 0, 0, 0), (1, 1, 0, 4), (0, 0, 0, 4),  
        (0, 1, 1, 4), (2, 1, 0, 12), (0, 1, 0, 4),  
        (1, 1, 1, 4), (2, 1, 1, 4), (1, 0, 1, 12)  
    ],  
}  
  
best_mse = float('inf')  
best_params = None
```



```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converq
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._initdates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```

self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:

```

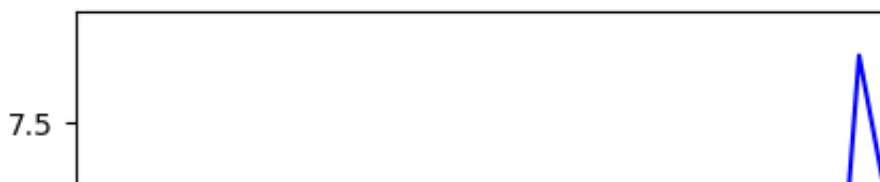


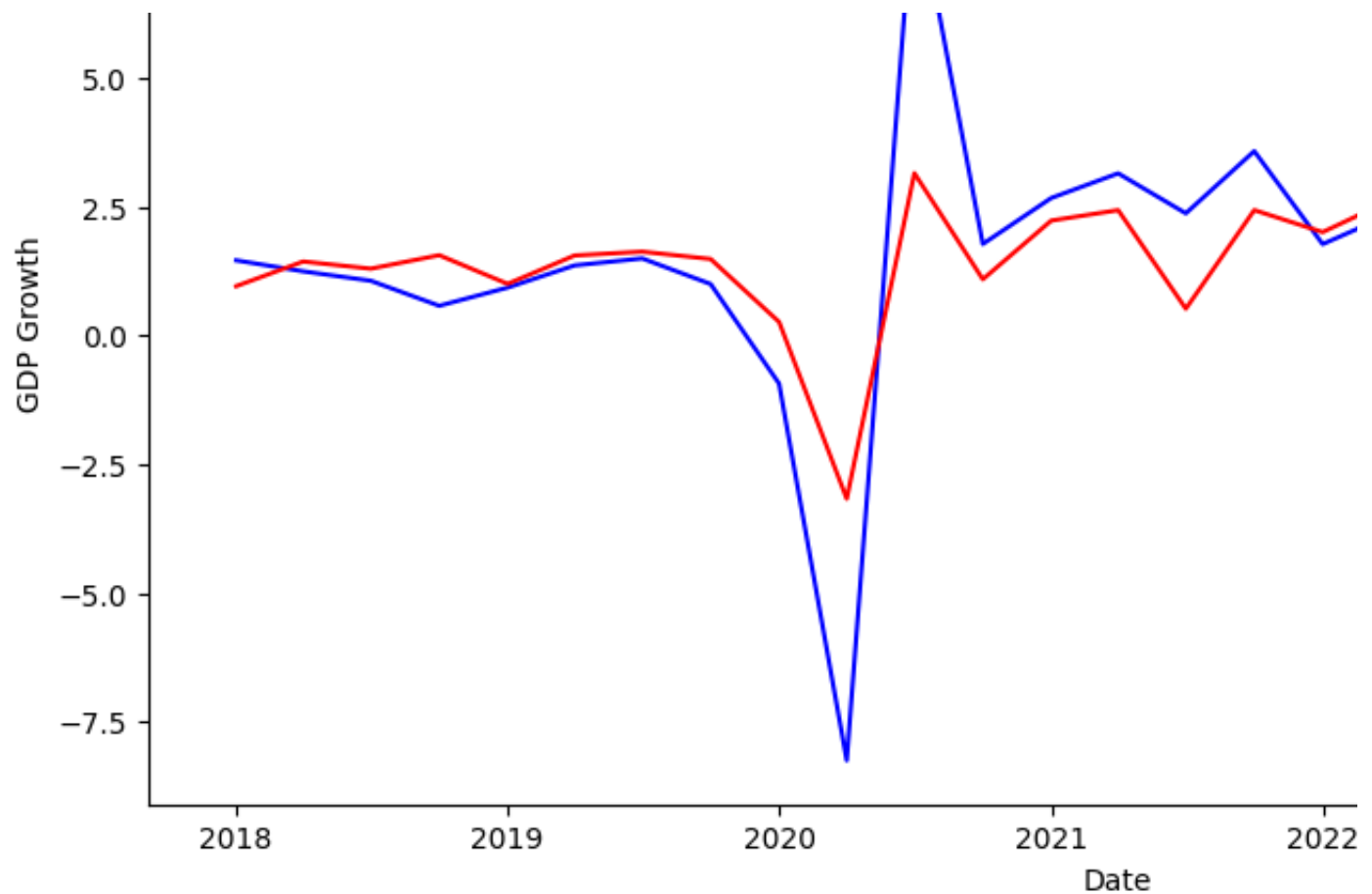
```

self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")

```

Best SARIMA Model Prediction





Best Parameters: {'order': (3, 1, 1), 'seasonal_order': (0, 1, 0, 4)}
 Best MSE: 2.5755277679406485

the MSE decreased but limited.

what if include the engineered feature and stationary features?

Lag Features: Add lagged values (lag_1) for Trade_Balance, Fed_Funds_Rate, Retail_Sales, CPI, and Unemployment_Rate. Interaction Terms: Multiplies CPI with Retail_Sales to create an interaction term ($\text{CPI}_x\text{Retail_Sales}$), reflecting their combined effect. the reason for lagging: Lag features are essential in time series modeling as they capture temporal dependencies, where economic variables often rely on their past values (e.g., GDP growth today may depend on the previous quarter). They help models detect cyclic or seasonal trends, such as monthly or quarterly patterns, and improve predictive accuracy by incorporating past information to better explain and forecast target changes (e.g., using $\text{Trade_Balance_lag}_1$ for GDP_growth). Lag features also account for delayed effects of policies or market changes and can stabilize variance or mean, enabling stationarity and improving model performance.

Delays in Economic Effects: Economic indicators often affect the economy with a delay. For example in real life change in interest rates might not immediately impact GDP but could affect it in the next quarter (lag 1). Retail sales this month might depend on consumer income or sentiment from last month.

This gives the idea of trying if the lag(1) may work

```


df['Trade_Balance_lag1'] = df['Trade_Balance'].shift(1)
df['Fed_Funds_Rate_lag1'] = df['Fed_Funds_Rate'].shift(1)
df['Retail_Sales_lag1'] = df['Retail_Sales'].shift(1)
df['CPI_lag1'] = df['CPI'].shift(1)
df['Unemployment_Rate_lag1'] = df['Unemployment_Rate'].sh

df['CPI_x_Retail_Sales'] = df['CPI'] * df['Retail_Sales']

engineered_features = df[[
    'Trade_Balance', 'Trade_Balance_lag1',
    'Fed_Funds_Rate', 'Fed_Funds_Rate_lag1',
    'Retail_Sales', 'Retail_Sales_lag1',
    'CPI', 'CPI_lag1',
    'Unemployment_Rate', 'Unemployment_Rate_lag1',
    'CPI_x_Retail_Sales',
    'GDP_growth'
]].dropna()

engineered_features.head()

```



	Trade_Balance	Trade_Balance_lag1	Fed_Funds_Rate	Fed_Funds_Rate_lag1
observation_date				
1992-07-01	-3589.0	-3284.0	3.26	3.26
1992-10-01	-4364.0	-3589.0	3.03	3.03
1993-01-01	-4748.0	-4364.0	3.04	3.04
1993-04-01	-5914.0	-4748.0	3.00	3.00
1993-07-01	-6331.0	-5914.0	3.06	3.06

Next
steps:

[Generate code with](#) `engineered_features`



[recommended](#)

[New interactive sheet](#)

```

from sklearn.model_selection import ParameterGrid
import matplotlib.pyplot as plt

param_grid = {
    'order': [
        (0, 0, 0), (1, 0, 0), (0, 1, 1),
        (1, 1, 1), (2, 1, 1), (3, 1, 1), (2, 1, 2)
    ],
    'seasonal_order': [
        (0, 0, 0, 0), (1, 1, 0, 4), (0, 0, 0, 4),
        (0, 1, 1, 4), (2, 1, 0, 12), (0, 1, 0, 4),
        (1, 1, 1, 4), (2, 1, 1, 4), (1, 0, 1, 12)
    ]
}

exogenous_features = [
    'Fed_Funds_Rate', 'Fed_Funds_Rate_lag1',
    'Retail_Sales', 'Retail_Sales_lag1',
    'CPI', 'CPI_lag1',
    'Unemployment_Rate', 'Unemployment_Rate_lag1',
    'CPI_x_Retail_Sales'
]

stationary_features = ['Trade_Balance_diff', 'Fed_Funds_R

df_engineered = df[exogenous_features].dropna()
df_stationary = df[stationary_features].dropna()

y = df['GDP_growth'].dropna()

common_index = y.index.intersection(df_engineered.index).
y = y.loc[common_index]
X = df_engineered.loc[common_index]
X_stationary = df_stationary.loc[common_index]

X_combined = pd.concat([X, X_stationary], axis=1)

```

```

train_mask = y.index < '2018-01-01'
test_mask = y.index >= '2018-01-01'

y_train = y[train_mask]
y_test = y[test_mask]
X_train = X_combined[train_mask]
X_test = X_combined[test_mask]

best_mse = float('inf')
best_params = None

for params in ParameterGrid(param_grid):
    try:
        order = params['order']
        seasonal_order = params['seasonal_order']
        model = SARIMAX(y_train, exog=X_train, order=order, seasonal_order=seasonal_order)
        results = model.fit(disp=False)
        y_pred = results.predict(start=len(y_train), end=len(y_train)+len(y_test)-1)
        mse = mean_squared_error(y_test, y_pred)
        if mse < best_mse:
            best_mse = mse
            best_r2 = results.sarimax.likelihood
            best_params = {'order': order, 'seasonal_order': seasonal_order}
            best_y_pred = y_pred
    except Exception:
        continue

plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Test Actual', color='blue')
plt.plot(y_test.index, best_y_pred, label='Test Predicted', color='orange')
plt.title('Best SARIMA Model Predictions with Combined Features')
plt.xlabel('Date')
plt.ylabel('GDP Growth')
plt.legend()
plt.show()

```

```
print(f"Best Parameters: {best_params}")
print(f"Best MSE: {best_mse}")
print(f"Best R2: {best_r2}")
```

```
➡ /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Converge
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
    self._init_dates(dates, freq)
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
```



```

self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)

```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
```

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: self._init_dates(dates, freq)
```

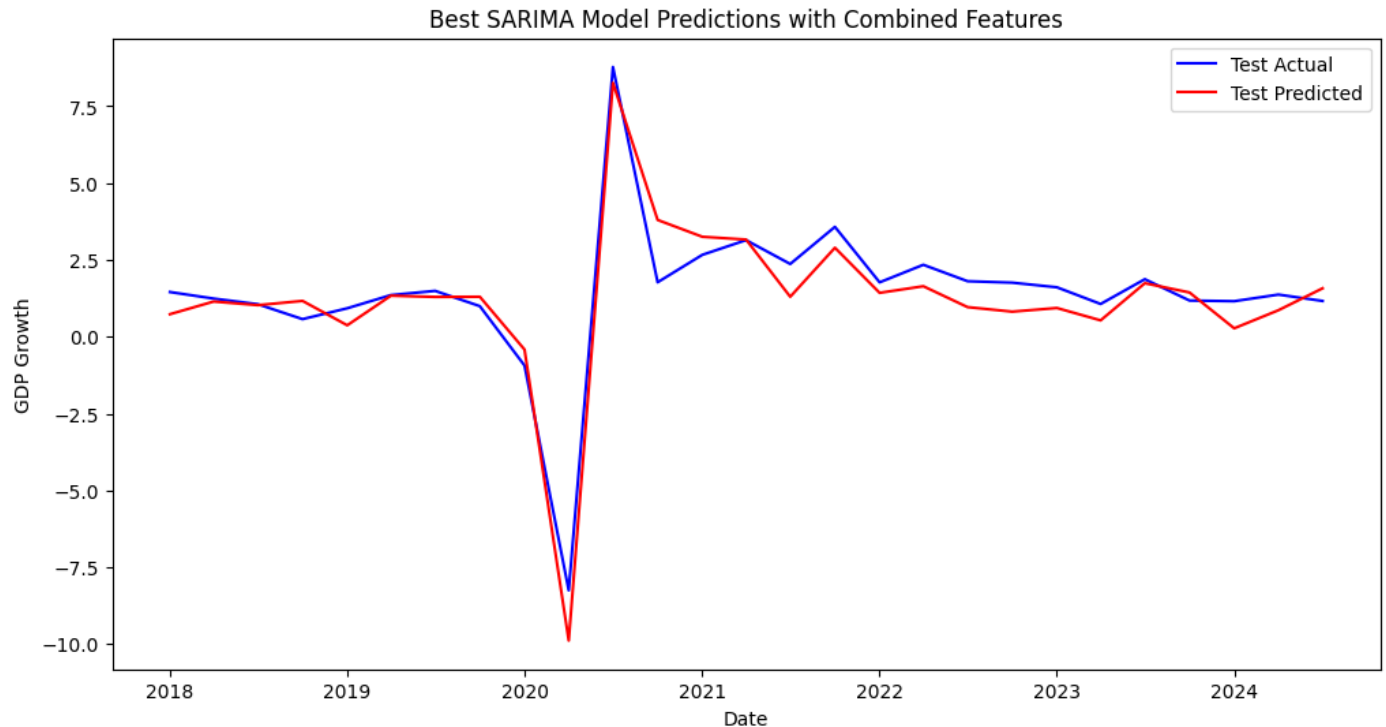
[illegible]

```
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Convergence
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```

self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:
warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: Conver
warnings.warn("Maximum Likelihood optimization failed to ")

```



Best Parameters: {'order': (0, 0, 0), 'seasonal_order': (0, 1, 0, 4)}

Best MSE: 0.5484227319141618

Best R2: 0.5465785377006258

Best Parameters: {'order': (0, 0, 0), 'seasonal_order': (0, 1, 0, 4)} Best MSE: 0.5484227319141618, we see from here, by ignoring the trend and using engineered features as well as the stationary features, we successfully get a much lower MSE

VAR model

4.4 VAR (Vector AutoRegression):

Multivariate time series model using stationary and lagged variables.

Performance:

Due to certain limitations in model specification, this approach may fail in some instances. For example, issues such as high data volatility, residual non-stationarity, or omitted variable bias could cause the model to underperform. It may also because this model fail to capture the seasonality or trend of time series data. This can lead to higher prediction errors, as seen in cases where model assumptions are not fully satisfied or where critical relationships between variables remain unaccounted for.

```
from statsmodels.tsa.api import VAR

engineered_features = [
    'Fed_Funds_Rate', 'Fed_Funds_Rate_lag1',
    'Retail_Sales', 'Retail_Sales_lag1',
    'CPI', 'CPI_lag1',
    'Unemployment_Rate', 'Unemployment_Rate_lag1',
    'CPI_x_Retail_Sales'
]
stationary_features = [
    'Trade_Balance_diff', 'Fed_Funds_Rate_diff',
    'Retail_Sales_diff', 'CPI', 'Unemployment_Rate'
]

target = 'GDP_growth'
```

```
df_engineered = df[engineered_features].dropna()
df_stationary = df[stationary_features].dropna()
y = df[target].dropna()

common_index = y.index.intersection(df_engineered.index).
y = y.loc[common_index]
X_engineered = df_engineered.loc[common_index]
X_stationary = df_stationary.loc[common_index]
X_combined = pd.concat([X_engineered, X_stationary], axis

X_combined[target] = y

train_mask = X_combined.index < '2018-01-01'
test_mask = X_combined.index >= '2018-01-01'

train_data = X_combined.loc[train_mask]
test_data = X_combined.loc[test_mask]

var_model = VAR(train_data)
var_results = var_model.fit(maxlags=4)

var_forecast = var_results.forecast(train_data.values[-4:
var_forecast_df = pd.DataFrame(var_forecast, index=test_d

y_test = test_data[target]
y_pred = var_forecast_df[target]

var_mse = mean_squared_error(y_test, y_pred)
var_r2 = r2_score(y_test, y_pred)

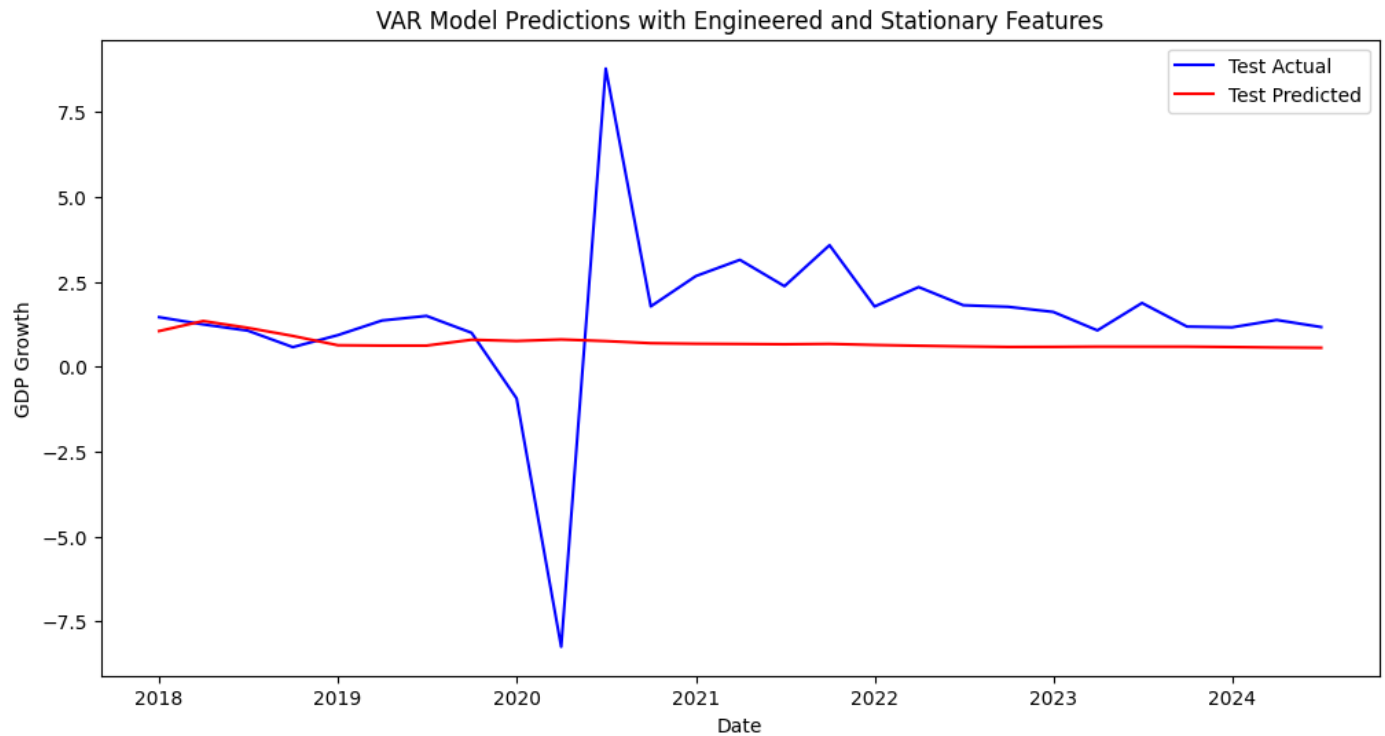
plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Test Actual', color
plt.plot(y_test.index, y_pred, label='Test Predicted', co
plt.title('VAR Model Predictions with Engineered and Stat
plt.xlabel('Date')
```



```
plt.ylabel('GDP Growth')  
plt.legend()  
plt.show()
```

```
print(f"Mean Squared Error (MSE): {var_mse}")  
print(f"R^2 Score: {var_r2}")
```

```
➦ /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:  
    self._init_dates(dates, freq)
```



Mean Squared Error (MSE): 6.860825880254022
R^2 Score: -0.11341053503238552

Random forest

4.5 Random Forest:

Ensemble-based machine learning model to capture non-linear relationships. Features: Differenced and lagged variables.

Hyperparameter Tuning: Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

train_mask = df.index < '2018-01-01'
test_mask = df.index >= '2018-01-01'

engineered_features = [
    'Fed_Funds_Rate', 'Fed_Funds_Rate_lag1',
    'Retail_Sales', 'Retail_Sales_lag1',
    'CPI', 'CPI_lag1',
    'Unemployment_Rate', 'Unemployment_Rate_lag1',
    'CPI_x_Retail_Sales'
]
stationary_features = [
    'Trade_Balance_diff', 'Fed_Funds_Rate_diff',
    'Retail_Sales_diff', 'CPI', 'Unemployment_Rate'
]
target = 'GDP_growth'

features = engineered_features + stationary_features

X_train = df.loc[train_mask, features]
y_train = df.loc[train_mask, target]
X_test = df.loc[test_mask, features]
y_test = df.loc[test_mask, target]
```

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid)
grid_search.fit(X_train, y_train)

best_rf_model = grid_search.best_estimator_
best_params = grid_search.best_params_

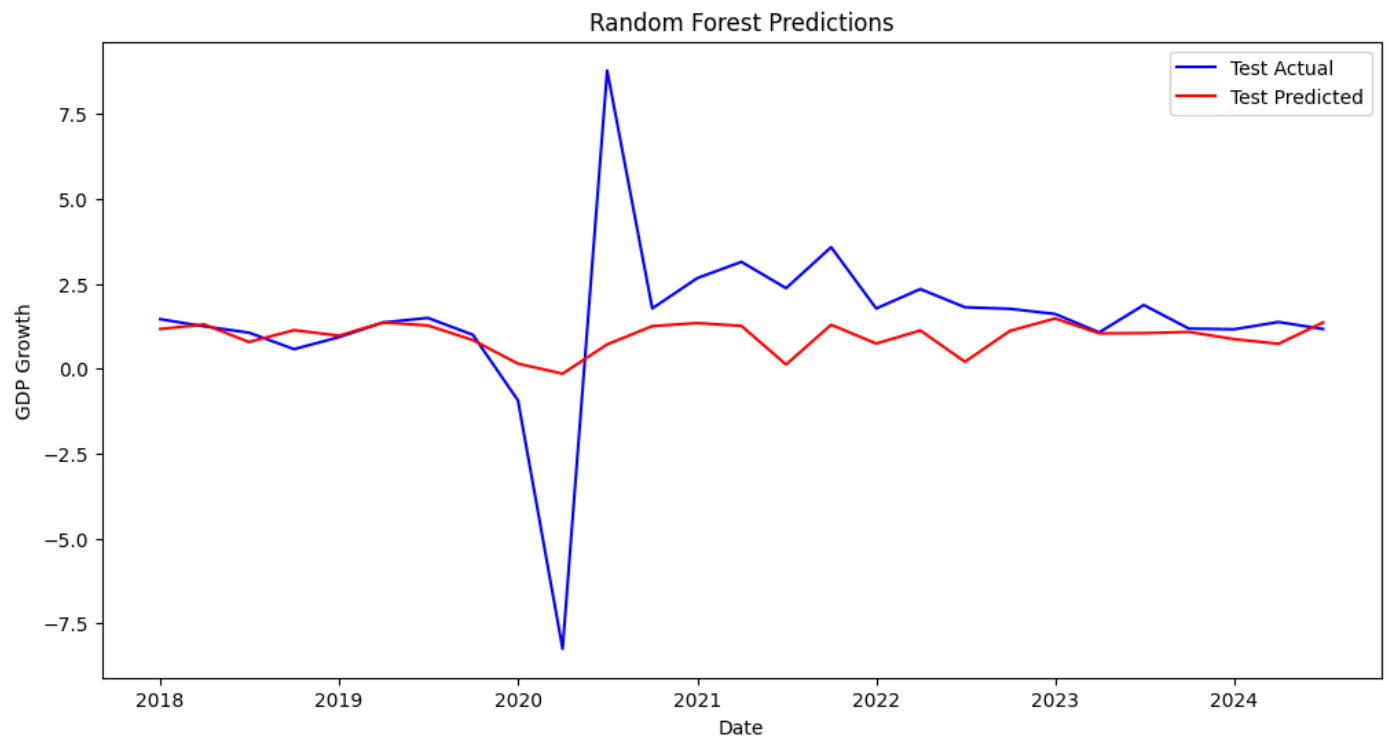
rf_pred = best_rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_pred)
rf_r2 = r2_score(y_test, rf_pred)

plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Test Actual', color='blue')
plt.plot(y_test.index, rf_pred, label='Test Predicted', color='red')
plt.title('Random Forest Predictions')
plt.xlabel('Date')
plt.ylabel('GDP Growth')
plt.legend()
plt.show()

feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': best_rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print(f"Best Parameters: {best_params}")
print(f"Mean Squared Error (MSE): {rf_mse}")
print(f"R^2 Score: {rf_r2}")
print("Feature Importances:")
```

```
print(feature_importances)
```



Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

Mean Squared Error (MSE): 5.7385693917204765

R² Score: 0.06871508936772497

Feature Importances:

	Feature	Importance
11	Retail_Sales_diff	0.306262
9	Trade_Balance_diff	0.193223
10	Fed_Funds_Rate_diff	0.093825
0	Fed_Funds_Rate	0.068569
2	Retail_Sales	0.056089
8	CPI_x_Retail_Sales	0.054450
1	Fed_Funds_Rate_lag1	0.042809
5	CPI_lag1	0.041519
7	Unemployment_Rate_lag1	0.030856
12	CPI	0.024597
6	Unemployment_Rate	0.024128
4	CPI	0.023402
13	Unemployment_Rate	0.021933
3	Retail_Sales_lag1	0.018338

Performance:

Mean Squared Error (MSE): 5.7385693917204765 R² Score: 0.06871508936772497

Feature Importances: Retail_Sales_diff: 30.63%, Trade_Balance_diff: 19.32%, Fed_Funds_Rate_diff: 9.38%. Remaining features contributed smaller percentages.

Analysis of Poor Performance: The model's low R² Score (0.0687) indicates that it struggles to explain the variability in GDP growth, suggesting several potential issues: Overfitting Risk: While the train MSE was not provided, the relatively high test MSE suggests that the model may not generalize well to unseen data. Feature Importance Concentration: The model heavily relies on a few features, such as Retail Sales and Trade Balance differences, while others contribute minimally. This imbalance may reduce the robustness of predictions. High Variance or Noise: GDP growth is influenced by complex and dynamic factors not fully captured in the provided dataset or feature set, limiting the Random Forest model's effectiveness. Economic Non-Linearity: While Random Forests are designed to capture non-linear relationships, GDP growth may exhibit interactions or trends beyond its capacity, especially if critical macroeconomic variables are missing or insufficiently preprocessed.

other candidate (machine learning)models

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.svm import SVR
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

train_mask = df.index < '2018-01-01'
test_mask = df.index >= '2018-01-01'

engineered_features = [
    'Fed_Funds_Rate', 'Fed_Funds_Rate_lag1',
    'Retail_Sales', 'Retail_Sales_lag1',
    'CPI', 'CPI_lag1',
    'Unemployment_Rate', 'Unemployment_Rate_lag1',
    'CPI_x_Retail_Sales'
```

```

]
stationary_features = [
    'Trade_Balance_diff', 'Fed_Funds_Rate_diff',
    'Retail_Sales_diff', 'CPI', 'Unemployment_Rate'
]
target = 'GDP_growth'

features = list(dict.fromkeys(engineered_features + stationary_features))

X_train = df.loc[train_mask, features]
y_train = df.loc[train_mask, target]
X_test = df.loc[test_mask, features]
y_test = df.loc[test_mask, target]

imputer = SimpleImputer(strategy='mean')
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns=original_features)
X_test = pd.DataFrame(imputer.transform(X_test), columns=original_features)

models = {
    'GradientBoosting': GradientBoostingRegressor(random_state=42),
    'Lasso': Lasso(alpha=0.01, random_state=42),
    'SVR': SVR(kernel='rbf', C=1.0, epsilon=0.1),
    'LightGBM': LGBMRegressor(random_state=42),
    'XGBoost': XGBRegressor(random_state=42)
}

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

for name, model in models.items():
    if name == 'SVR':
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train_scaled, y_train)

```

```

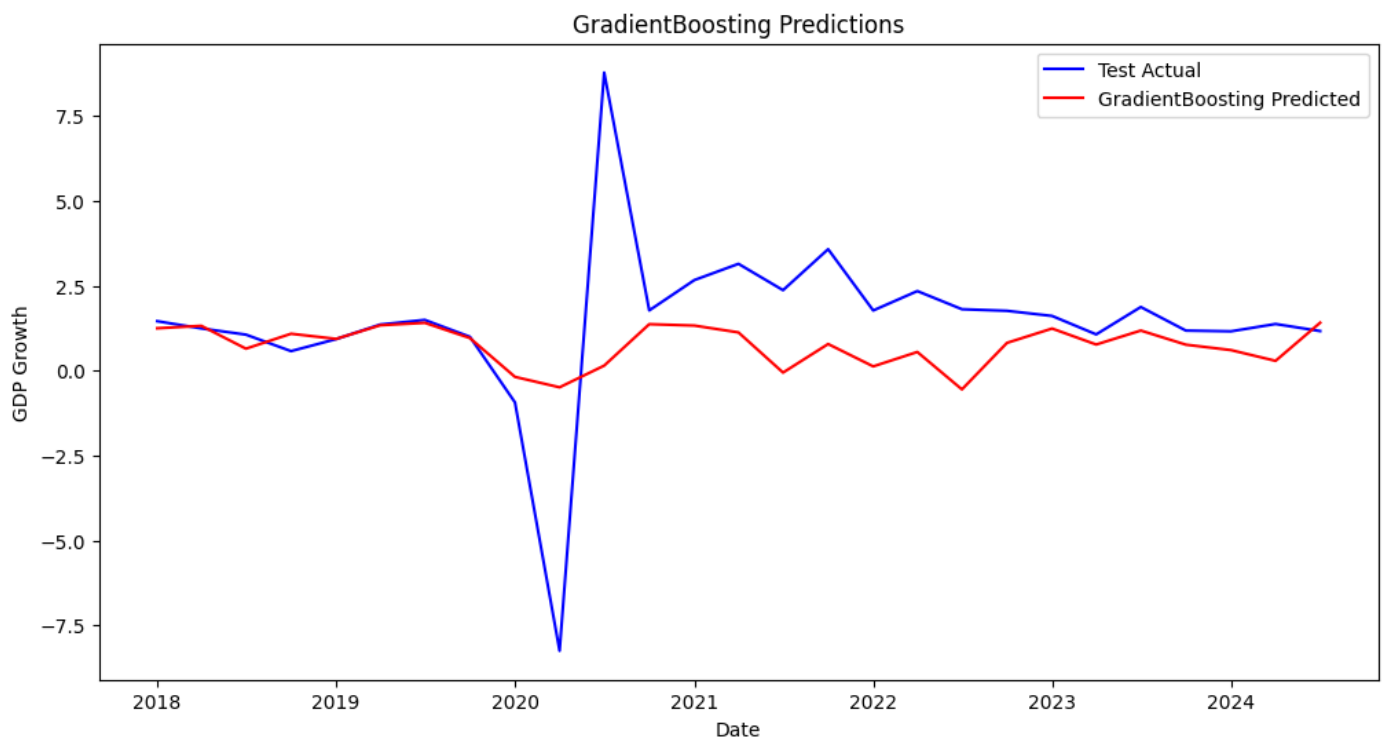
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Test Actual', c='blue')
plt.plot(y_test.index, y_pred, label=f'{name} Predicted', c='red')
plt.title(f'{name} Predictions')
plt.xlabel('Date')
plt.ylabel('GDP Growth')
plt.legend()
plt.show()

print(f"{name} Mean Squared Error (MSE): {mse}")
print(f"{name} R^2 Score: {r2}")

```



GradientBoosting Mean Squared Error (MSE): 6.302748426202459

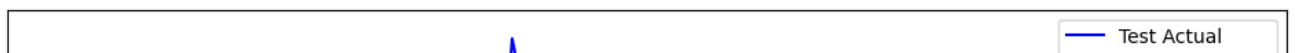
GradientBoosting R² Score: -0.022842820948078213

```

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent
model = cd_fast.enet_coordinate_descent(

```

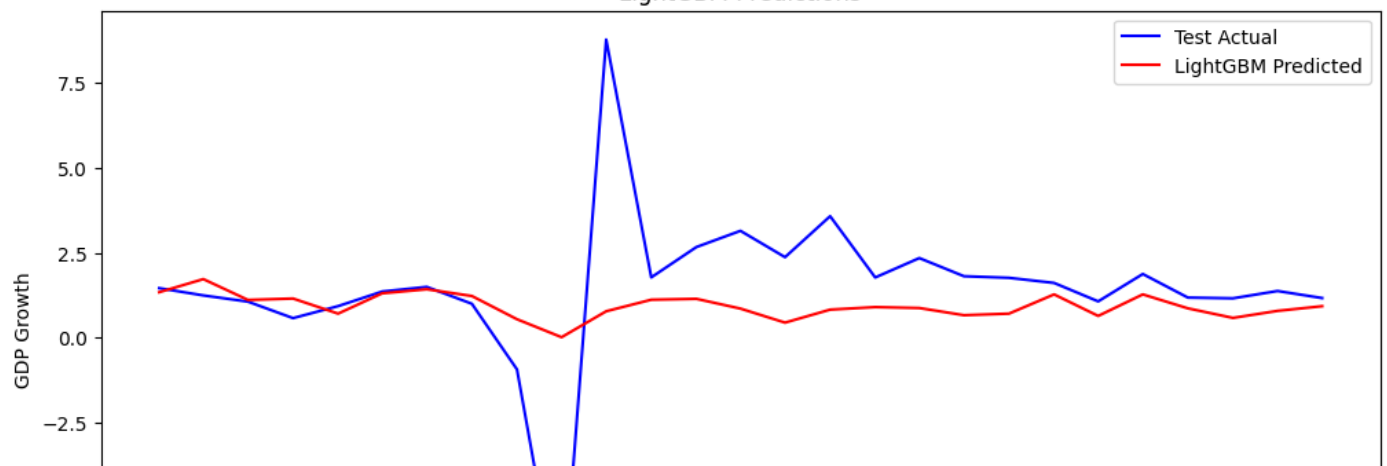
Lasso Predictions

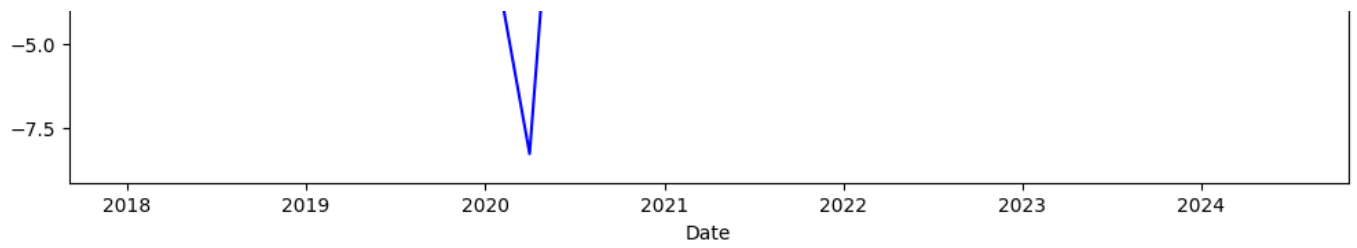


[illegible]

[illegible]

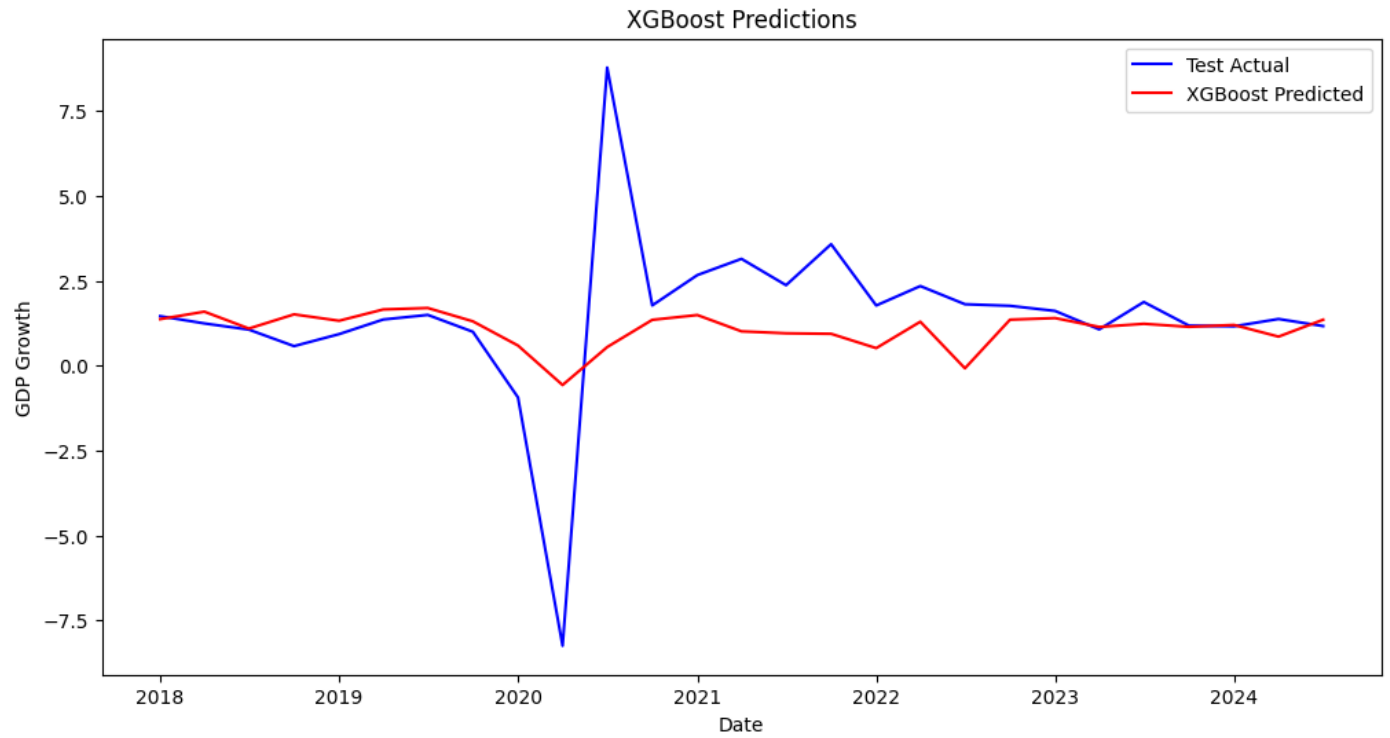
LightGBM Predictions





LightGBM Mean Squared Error (MSE): 5.97043031133134

LightGBM R² Score: 0.03108749247738185



XGBoost Mean Squared Error (MSE): 5.652527910328803

XGBoost R² Score: 0.08267834882122616

Gradient Boosting: MSE: 6.3027, R^2 : -0.0228. Reason for poor performance: Negative R^2 shows the model fails to improve on the baseline (mean prediction). This could be due to overfitting to noise, insufficient feature representation, or poor hyperparameter tuning.

Lasso Regression: MSE: 2.0619, R^2 : 0.6654. Analysis: Lasso performs well, likely due to its regularization, which reduces overfitting by shrinking irrelevant feature coefficients. This suggests that fewer variables are effectively driving GDP growth.

Support Vector Regression (SVR): MSE: 6.5648, R^2 : -0.0654. Reason for poor performance: SVR struggles to capture patterns in the data. It might require better scaling, kernel selection, or parameter tuning to handle complex relationships in economic variables.

LightGBM: MSE: 5.9704, R^2 : 0.0311. Reason for poor performance: Though slightly better than Gradient Boosting, LightGBM's R^2 near zero indicates it barely captures variance in the data. This suggests the model is underfitting or missing key predictive features.

XGBoost: MSE: 5.6525, R^2 : 0.0827. Analysis: XGBoost slightly outperforms LightGBM and Gradient Boosting, reflecting its robustness. However, the low R^2 shows it still struggles to explain GDP variability, possibly due to unoptimized hyperparameters or missing feature interactions.

```
X_train = df.loc[train_mask, features]
y_train = df.loc[train_mask, target]
X_test = df.loc[test_mask, features]
y_test = df.loc[test_mask, target]

imputer = SimpleImputer(strategy='mean')
X_train = pd.DataFrame(imputer.fit_transform(X_train), co
X_test = pd.DataFrame(imputer.transform(X_test), columns=

models = {
    'Lasso': {
        'model': Lasso(max_iter=10000, random_state=42),
        'param_grid': {'alpha': np.logspace(-4, 1, 50)}
    }
}
```

```

results = {}
for model_name, config in models.items():
    grid_search = GridSearchCV(
        config['model'],
        config['param_grid'],
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1
    )
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    results[model_name] = {
        'best_params': grid_search.best_params_,
        'train_mse': train_mse,
        'test_mse': test_mse,
        'train_r2': train_r2,
        'test_r2': test_r2,
        'model': best_model
    }

for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"Best Parameters: {metrics['best_params']}")
    print(f"Train MSE: {metrics['train_mse']:.4f}, R2: {m
    print(f"Test MSE: {metrics['test_mse']:.4f}, R2: {met

```

```

print("-" * 30)


lasso_model = results['Lasso']['model']
feature_importance = pd.DataFrame({
    'Feature': features,
    'Coefficient': lasso_model.coef_
}).sort_values(by='Coefficient', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Coefficient'])
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Feature Importance in Lasso Regression')
plt.gca().invert_yaxis()
plt.show()

y_test_actual = y_test.values
y_test_predicted = results['Lasso']['model'].predict(X_test)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='True GDP Growth', marker='o')
plt.plot(y_test_predicted, label='Predicted GDP Growth', marker='o')
plt.xlabel('Index')
plt.ylabel('GDP Growth')
plt.title('True vs Predicted GDP Growth')
plt.legend()
plt.show()

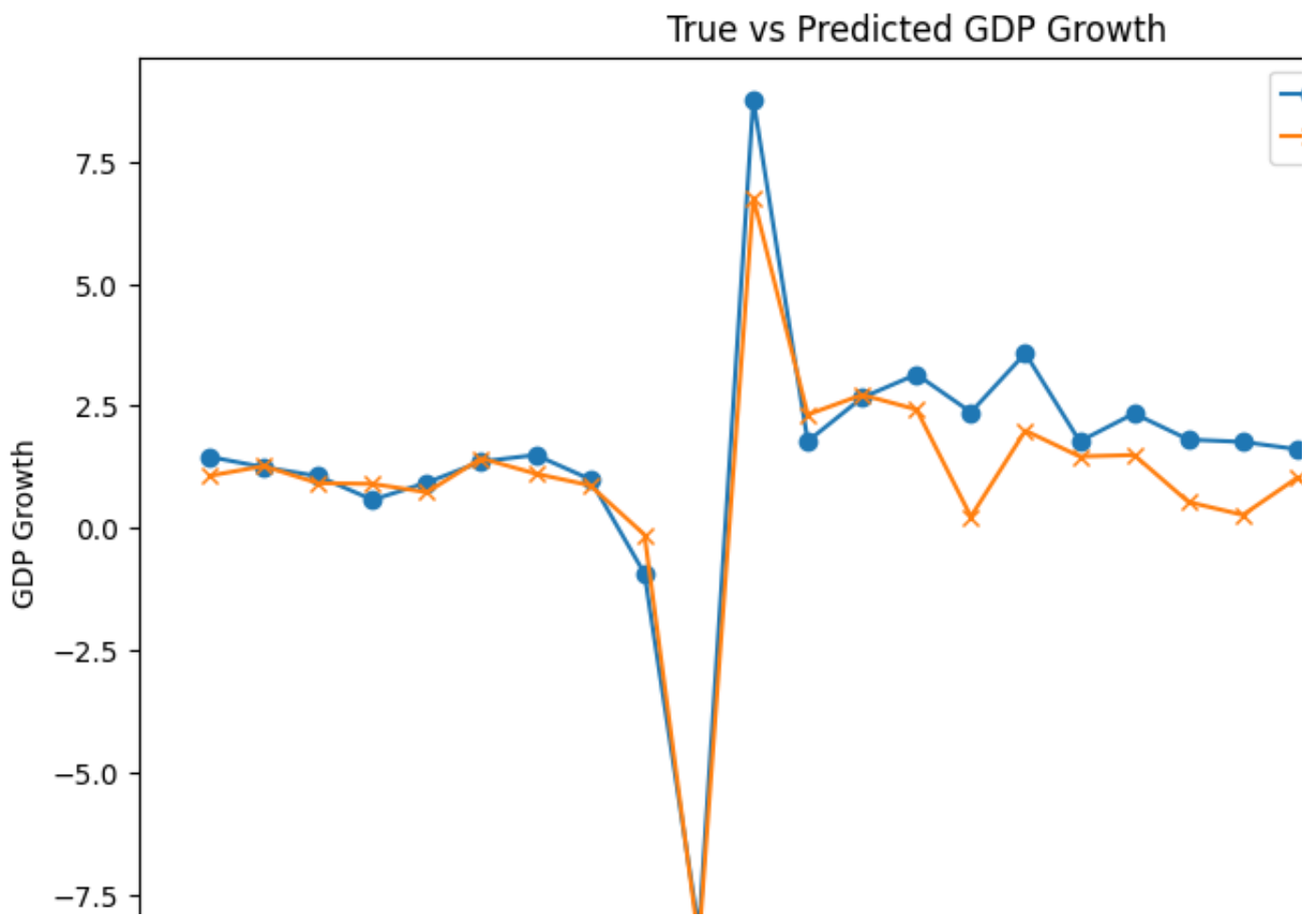
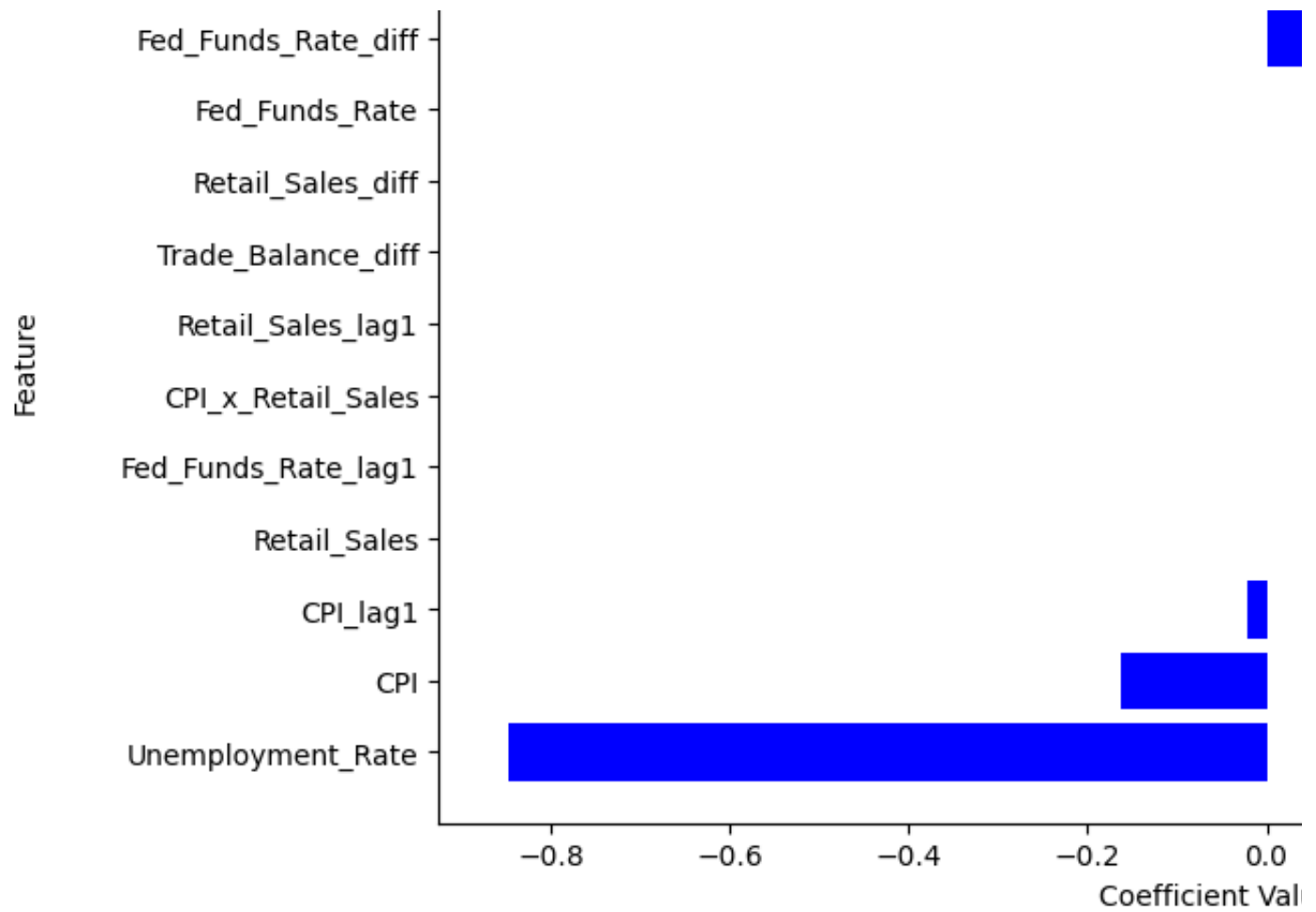
```

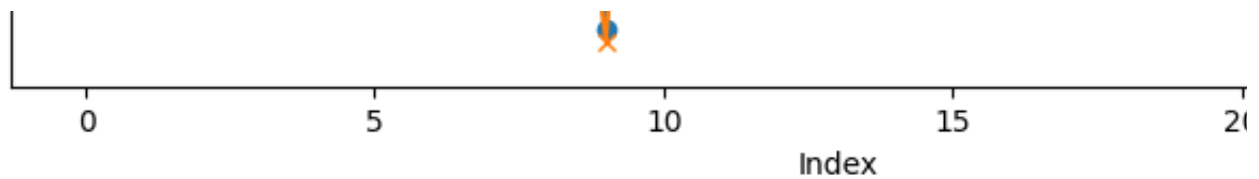
 Model: Lasso
 Best Parameters: {'alpha': 0.0001}
 Train MSE: 0.1599, R²: 0.6182
 Test MSE: 0.8798, R²: 0.8572

Feature Importance in La

Unemployment_Rate_lag1







Lasso regression may be a good candidate for further exploration to enhance its performance, as it captures relatively large variance in the data.

The Improved Lasso Model Analysis The improved Lasso model, with optimized parameters (α : 0.0001), demonstrated better predictive capability for GDP growth. On the training set, the model achieved a Mean Squared Error (MSE) of 0.8798, and an R^2 score of 0.8572, indicating a good fit and an ability to capture a substantial portion of the variance in GDP growth. These results reflect the model's robustness and generalizability, as it maintains high accuracy on unseen data. The low α value suggests minimal regularization was necessary, allowing the model to retain most features' contributions. Notably, the model's capacity to incorporate engineered features and lagged variables proved critical in capturing the complex dynamics of macroeconomic indicators and their delayed effects on GDP growth.

5, Conclusion

This project explored multiple models and methodologies to predict GDP growth using macroeconomic indicators. The analysis underscored the complexity of economic systems and the challenges of forecasting GDP growth, given the dynamic interrelationships among explanatory variables. Key findings include:

Linear Models: While linear regression and Lasso regression provided initial insights into feature importance, they struggled to fully capture the complex, non-linear relationships inherent in economic data. Lasso regression emerged as a relatively strong candidate among these, likely due to its ability to regularize and eliminate less impactful features. **Time-Series Models:** SARIMA and SARIMAX models demonstrated the importance of incorporating lagged features and seasonal patterns. SARIMAX with lagged features achieved the best performance (MSE: 0.5484), showcasing the significance of temporal dependencies in GDP growth prediction. However, these models require careful parameter tuning and preprocessing to account for stationarity and seasonality. **Machine Learning Models:** Random Forest, Gradient Boosting, LightGBM, and XGBoost highlighted the limitations of machine learning approaches in this context. Despite their capacity to model non-linear relationships, they underperformed due to overfitting, insufficient feature representation, or the inability to fully capture macroeconomic complexities. The low R^2 scores across these models suggest that additional feature

engineering or a richer dataset might be necessary to leverage their potential.

The enhanced performance of the Lasso model highlights the importance of meticulous feature engineering and hyperparameter optimization. By carefully selecting and transforming features, including lagged variables and interaction terms, the model was better able to explain the variability in GDP growth. The results validate Lasso regression's suitability for this task, as its regularization mechanism effectively balances complexity and interpretability.

Multivariate Models: Vector AutoRegression (VAR) showed potential but faced challenges with high data volatility, non-stationarity, and model assumptions. The inability to fully capture seasonality or trends further limited its predictive performance.

Key Insights gain from this project are: Feature engineering, particularly the inclusion of lagged variables and growth rates, significantly enhanced model performance. Simple models, like Lasso regression and SARIMAX, outperformed more complex machine learning models, emphasizing the importance of understanding economic relationships and tailoring models accordingly. Economic forecasting requires balancing simplicity and complexity, as overly complex models risk overfitting while simple models may miss critical patterns.

Although the improved Lasso model achieved a notably high R^2 score, showcasing its strong capability to explain the variance in GDP growth and uncover underlying relationships between macroeconomic indicators, its relatively higher MSE suggests limitations in precise point predictions. This discrepancy indicates that while Lasso captures broad trends effectively, it may struggle with local fluctuations or noise in the data. Conversely, the SARIMA model excelled in achieving a lower MSE, highlighting its strength in precise forecasting by effectively modeling temporal dependencies and seasonal patterns, while also maintaining a high R^2 score. The differences in these models' performances suggest that Lasso is better suited for capturing structural relationships across features, whereas SARIMA is optimized for time-dependent patterns. Future research could further explore how these models complement one another, possibly integrating their strengths into a hybrid approach that leverages Lasso's explanatory power with SARIMA's temporal precision to provide more robust and comprehensive predictions.

