GWP_3(Python_Code)

January 7, 2024

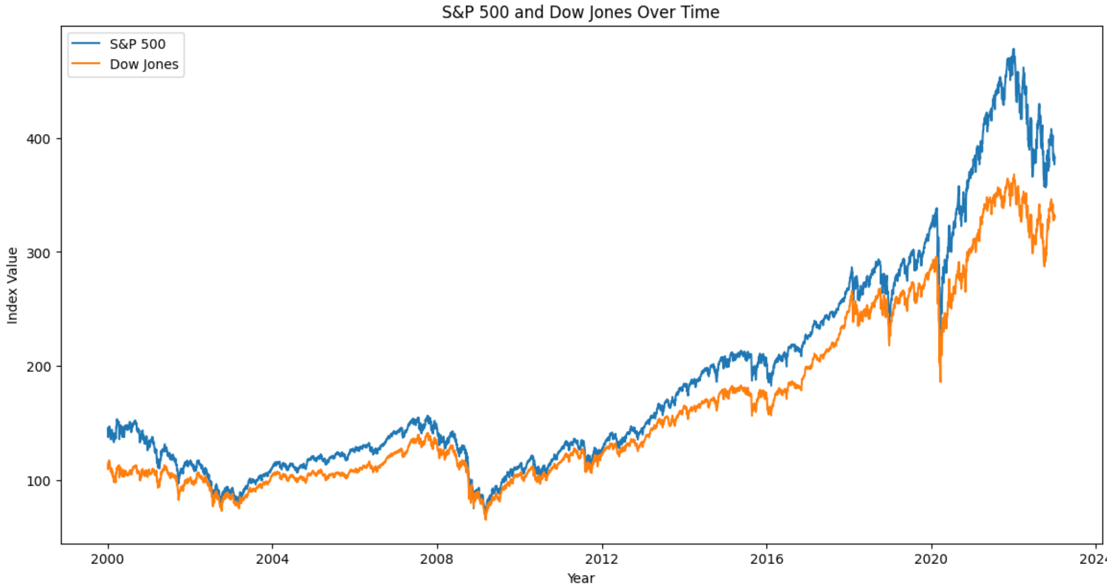
1 MScFE 610 FINANCIAL ECONOMETRICS Group Work Project # 3

1.1 First Proposed Choice for Dataset - Stock Market Indices (S&P 500, Dow Jones)

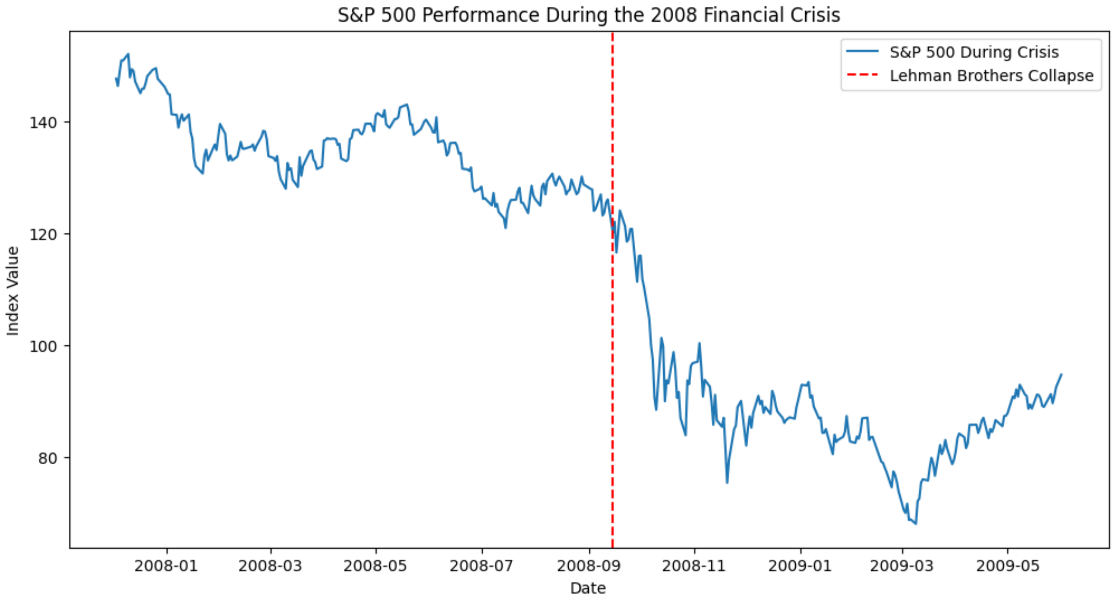
[1]:

```
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns

# Downloading data for S&P 500 (represented by the ticker SPY) and Dow Jones␣
 ↪(represented by DIA)
sp500 = yf.download('SPY', start='2000-01-01', end='2023-01-01')
dow_jones = yf.download('DIA', start='2000-01-01', end='2023-01-01')

# Plotting the closing prices to visualize trends and cycles
plt.figure(figsize=(14, 7))
plt.plot(sp500['Close'], label='S&P 500')
plt.plot(dow_jones['Close'], label='Dow Jones')
plt.title('S&P 500 and Dow Jones Over Time')
plt.xlabel('Year')
plt.ylabel('Index Value')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

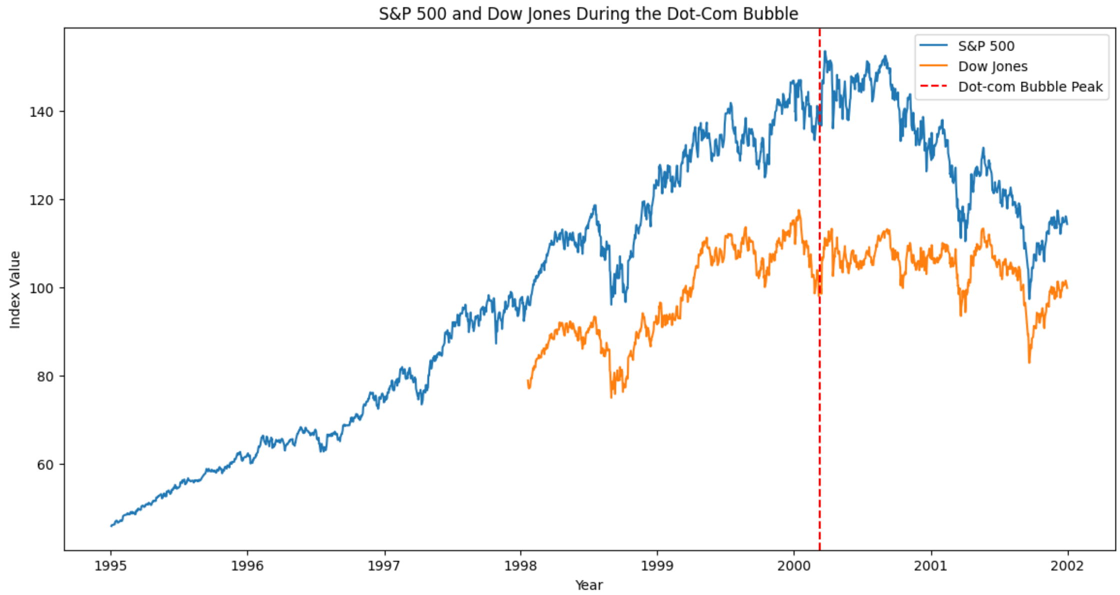


[2]:
```
# Example: Analyzing the impact of an event (e.g., the 2008 financial crisis)
crisis_start = '2007-12-01'
crisis_end = '2009-06-01'

# Plotting S&P 500 around the financial crisis
plt.figure(figsize=(12, 6))
plt.plot(sp500[crisis_start:crisis_end]['Close'], label='S&P 500 During Crisis')
plt.axvline(x=pd.to_datetime('2008-09-15'), color='r', linestyle='--',
↪label='Lehman Brothers Collapse')
plt.title('S&P 500 Performance During the 2008 Financial Crisis')
plt.xlabel('Date')
plt.ylabel('Index Value')
plt.legend()
plt.show()
```



[3]:
```
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt

# Define the time period for the dot-com bubble
start_date = '1995-01-01'
peak_date = '2000-03-10'  # Approximate peak of the dot-com bubble
end_date = '2002-01-01'

# Downloading data for S&P 500 (SPY) and Dow Jones (DIA)
sp500 = yf.download('SPY', start=start_date, end=end_date)
dow_jones = yf.download('DIA', start=start_date, end=end_date)

# Plotting the closing prices during the dot-com bubble period
plt.figure(figsize=(14, 7))
plt.plot(sp500['Close'], label='S&P 500')
plt.plot(dow_jones['Close'], label='Dow Jones')

# Highlighting the peak of the bubble
plt.axvline(x=pd.to_datetime(peak_date), color='r', linestyle='--',
 ↪label='Dot-com Bubble Peak')

plt.title('S&P 500 and Dow Jones During the Dot-Com Bubble')
plt.xlabel('Year')
plt.ylabel('Index Value')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

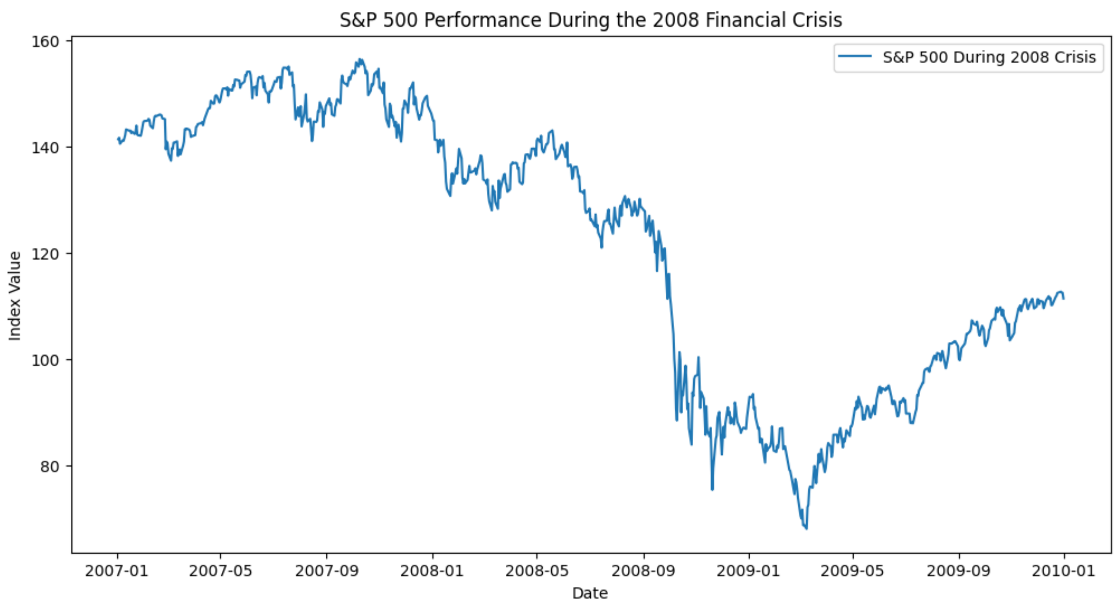


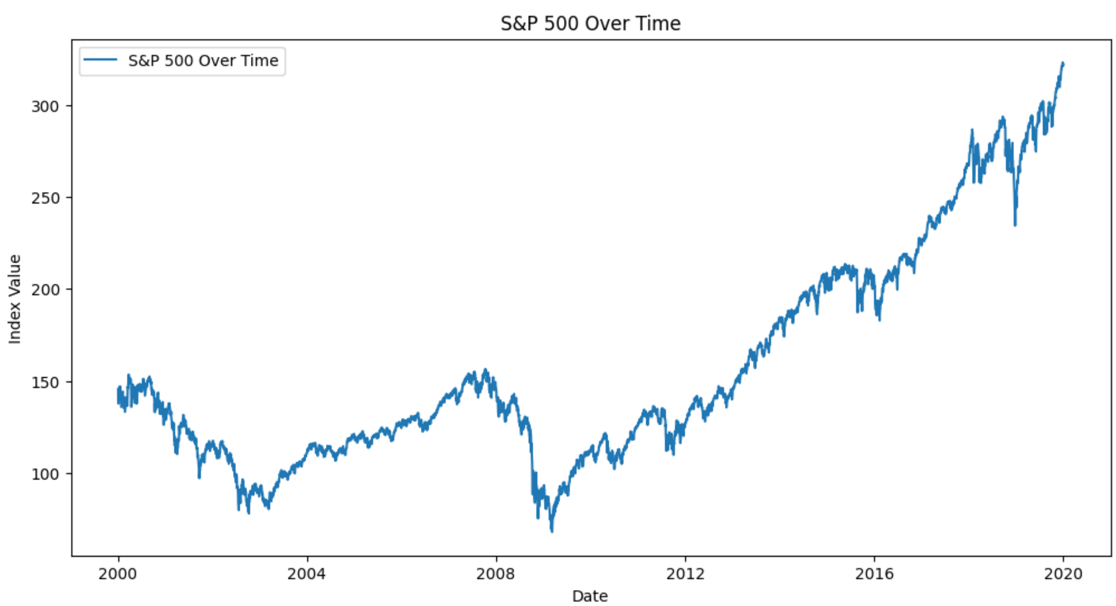
[4]:

```
import pandas as pd
import yfinance as yf
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt

# Downloading S&P 500 data
sp500 = yf.download('SPY', start='2000-01-01', end='2020-01-01')

# Check if the DataFrame is not empty
if not sp500.empty:
    # ADF test on S&P 500 data during the 2008 Financial Crisis
    crisis_data = sp500['2007-01-01':'2009-12-31']
    if not crisis_data.empty:
        # Proceed with ADF test
        adf_result_crisis = adfuller(crisis_data['Close'])
        print('ADF Statistic for 2008 Crisis: %f' % adf_result_crisis[0])
        print('p-value: %f' % adf_result_crisis[1])
    else:
        print("No data available for the 2008 Financial Crisis period.")

    # ADF test on a longer time period
    long_term_data = sp500['2000-01-01':'2020-01-01']
    if not long_term_data.empty:
        adf_result_long_term = adfuller(long_term_data['Close'])
```

```python
        print('ADF Statistic for Long Term: %f' % adf_result_long_term[0])
        print('p-value: %f' % adf_result_long_term[1])
    else:
        print("No data available for the long term period.")

    # Plotting S&P 500 performance during the 2008 Financial Crisis
    plt.figure(figsize=(12, 6))
    plt.plot(crisis_data['Close'], label='S&P 500 During 2008 Crisis')
    plt.title('S&P 500 Performance During the 2008 Financial Crisis')
    plt.xlabel('Date')
    plt.ylabel('Index Value')
    plt.legend()
    plt.show()

    # Plotting S&P 500 over a longer time period
    plt.figure(figsize=(12, 6))
    plt.plot(long_term_data['Close'], label='S&P 500 Over Time')
    plt.title('S&P 500 Over Time')
    plt.xlabel('Date')
    plt.ylabel('Index Value')
    plt.legend()
    plt.show()

else:
    print("S&P 500 data is empty. Check data downloading step.")
```

```
[*********************100%%***********************]  1 of 1 completed
ADF Statistic for 2008 Crisis: -0.954805
p-value: 0.769389
ADF Statistic for Long Term: 1.502699
p-value: 0.997534
```





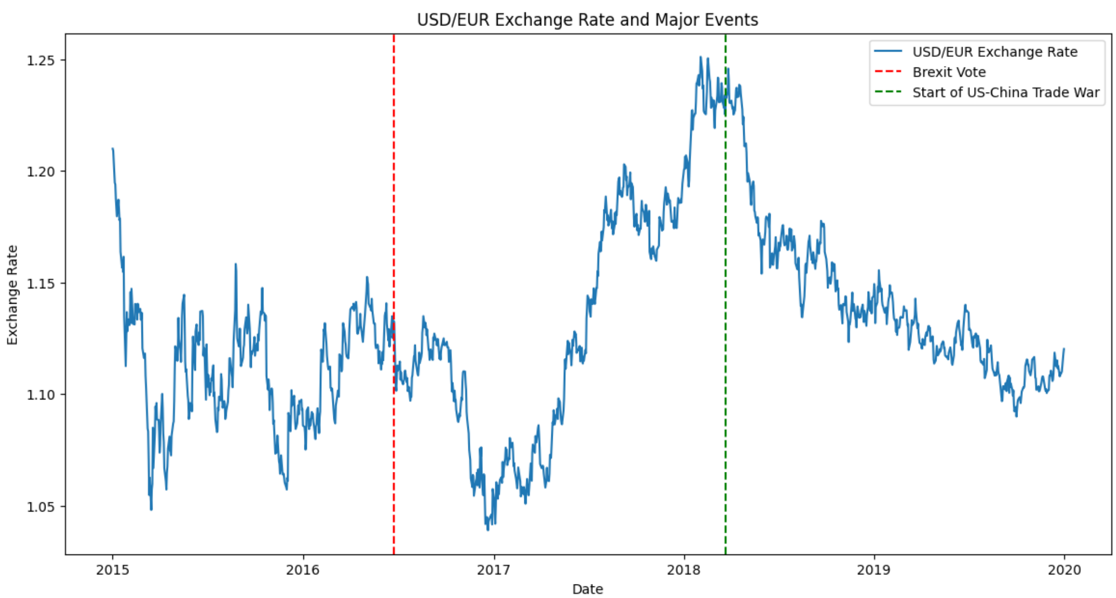
1.2 Second Proposed Choice for Dataset - Foreign Exchange Rates (USD/EUR)

[5]:

```
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd

# Download USD/EUR exchange rate data
```

```
usd_eur = yf.download('EURUSD=X', start='2015-01-01', end='2020-01-01')

# Ensure that 'Date' is in datetime format
usd_eur.index = pd.to_datetime(usd_eur.index)

# Plotting around major events like Brexit and US-China trade war
plt.figure(figsize=(14, 7))
plt.plot(usd_eur['Close'], label='USD/EUR Exchange Rate')

# Convert the event dates to pandas Timestamp
brexit_vote = pd.Timestamp('2016-06-23')
start_us_china_trade_war = pd.Timestamp('2018-03-22')

plt.axvline(x=brexit_vote, color='r', linestyle='--', label='Brexit Vote')
plt.axvline(x=start_us_china_trade_war, color='g', linestyle='--', label='Start␣
↪of US-China Trade War')

plt.title('USD/EUR Exchange Rate and Major Events')
plt.xlabel('Date')
plt.ylabel('Exchange Rate')
plt.legend()
plt.show()
```

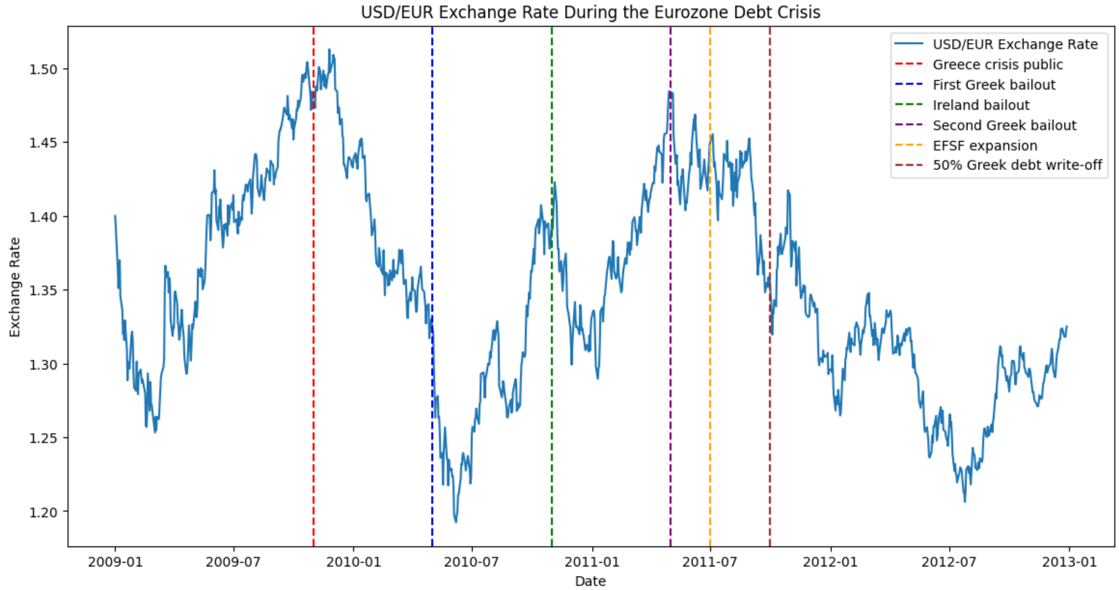
```
[*********************100%%**********************]  1 of 1 completed
```



[6]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import yfinance as yf

# Download USD/EUR exchange rate data
usd_eur = yf.download('EURUSD=X', start='2009-01-01', end='2012-12-31')

# Ensure the index is in datetime format
usd_eur.index = pd.to_datetime(usd_eur.index)

# Major events of the Eurozone crisis with distinctive colors
major_events = {
    '2009-11-01': {'event': 'Greece crisis public', 'color': 'red'},
    '2010-05-01': {'event': 'First Greek bailout', 'color': 'blue'},
    '2010-11-01': {'event': 'Ireland bailout', 'color': 'green'},
    '2011-05-01': {'event': 'Second Greek bailout', 'color': 'purple'},
    '2011-07-01': {'event': 'EFSF expansion', 'color': 'orange'},
    '2011-10-01': {'event': '50% Greek debt write-off', 'color': 'brown'}
}

# Plotting USD/EUR exchange rate with annotations for major events
plt.figure(figsize=(14, 7))
plt.plot(usd_eur['Close'], label='USD/EUR Exchange Rate')

# Add lines and legends for major events
for date, info in major_events.items():
    plt.axvline(x=pd.Timestamp(date), color=info['color'], linestyle='--',␣
 ↪label=info['event'])

plt.title('USD/EUR Exchange Rate During the Eurozone Debt Crisis')
plt.xlabel('Date')
plt.ylabel('Exchange Rate')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
```



[7]:
```python
import yfinance as yf
from statsmodels.tsa.stattools import adfuller

# Downloading USD/EUR exchange rate data
usd_eur = yf.download('EURUSD=X', start='2005-01-01', end='2020-01-01')

# Ensure the index is in datetime format
usd_eur.index = pd.to_datetime(usd_eur.index)

# Filtering data for the Eurozone debt crisis period (2009-01-01 to 2012-12-31)
crisis_data = usd_eur['2009-01-01':'2012-12-31']['Close']

# Filtering data for the Brexit and US-China Trade War period (2016-01-01 to␣
↪2020-01-01)
brexit_trade_war_data = usd_eur['2016-01-01':'2020-01-01']['Close']

# Perform ADF test on Eurozone debt crisis data
adf_result_crisis = adfuller(crisis_data)
print('ADF Statistic for Eurozone Crisis: %f' % adf_result_crisis[0])
print('p-value: %f' % adf_result_crisis[1])

# Perform ADF test on Brexit and US-China Trade War data
adf_result_brexit_trade_war = adfuller(brexit_trade_war_data)
print('ADF Statistic for Brexit and US-China Trade War: %f' %␣
↪adf_result_brexit_trade_war[0])
print('p-value: %f' % adf_result_brexit_trade_war[1])
```

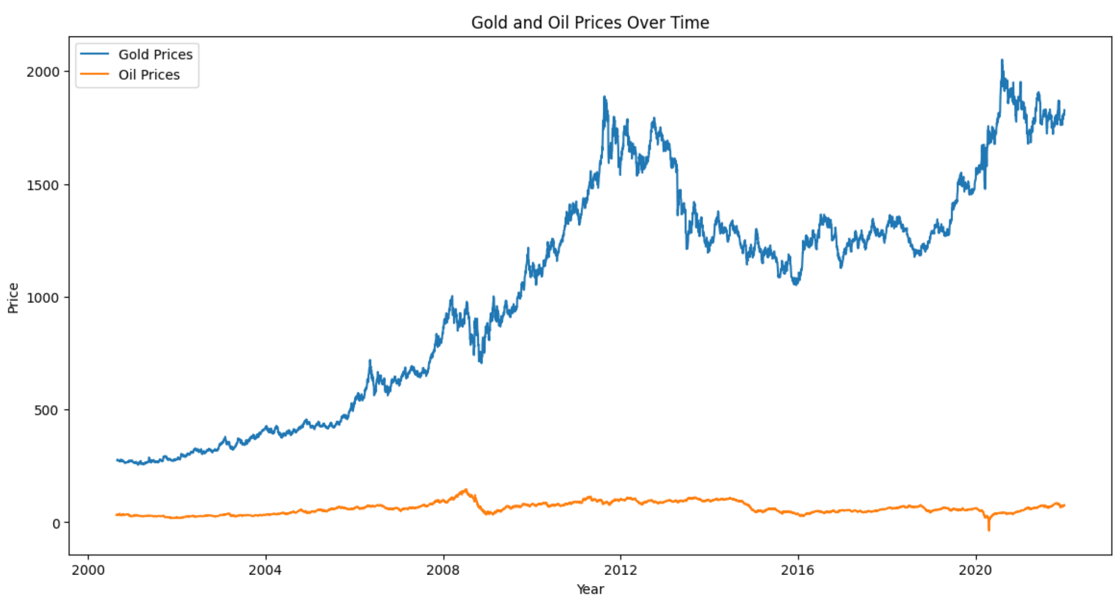
```
[*********************100%%**********************]  1 of 1 completed
```

```
ADF Statistic for Eurozone Crisis: -1.937125
p-value: 0.314803
ADF Statistic for Brexit and US-China Trade War: -1.970017
p-value: 0.299872
```

1.3 Third Proposed Choice for Dataset - Commodity Prices (Gold, Oil)

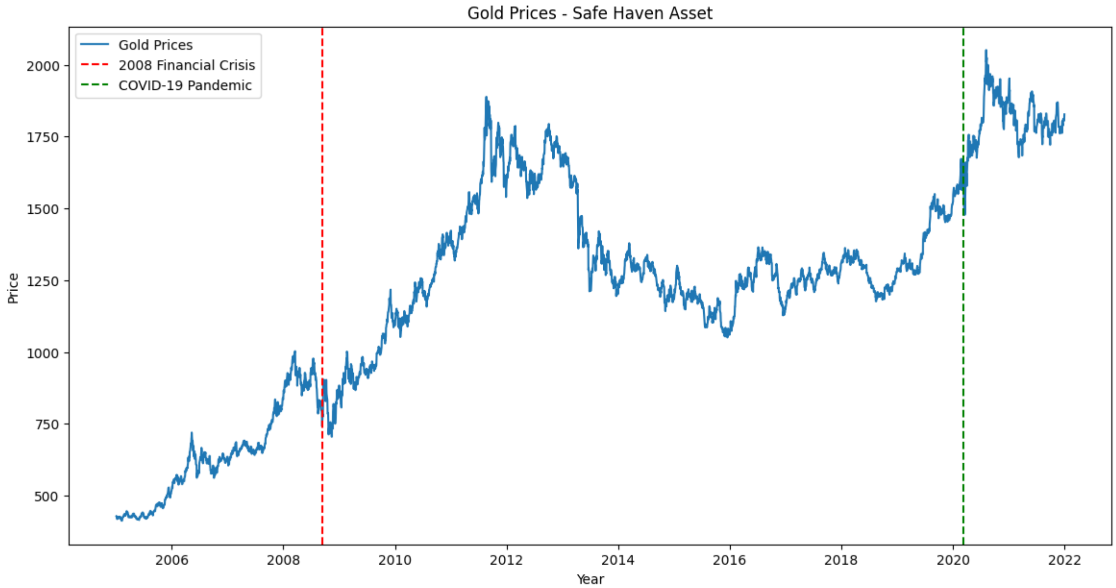
[8]:
```python
import yfinance as yf
import matplotlib.pyplot as plt

# Downloading historical data for Gold and Oil
gold = yf.download('GC=F', start='2000-01-01', end='2022-01-01')
oil = yf.download('CL=F', start='2000-01-01', end='2022-01-01')

# Plotting Gold and Oil prices
plt.figure(figsize=(14, 7))
plt.plot(gold['Close'], label='Gold Prices')
plt.plot(oil['Close'], label='Oil Prices')
plt.title('Gold and Oil Prices Over Time')
plt.xlabel('Year')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```



[9]:
```
import yfinance as yf
import matplotlib.pyplot as plt

# Downloading historical data for Gold
gold = yf.download('GC=F', start='2005-01-01', end='2022-01-01')

# Plotting Gold prices during the 2008 financial crisis and COVID-19 pandemic
plt.figure(figsize=(14, 7))
plt.plot(gold['Close'], label='Gold Prices')
plt.axvline(x=pd.Timestamp('2008-09-15'), color='r', linestyle='--',␣
 ↪label='2008 Financial Crisis')
plt.axvline(x=pd.Timestamp('2020-03-11'), color='g', linestyle='--',␣
 ↪label='COVID-19 Pandemic')
plt.title('Gold Prices - Safe Haven Asset')
plt.xlabel('Year')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
```



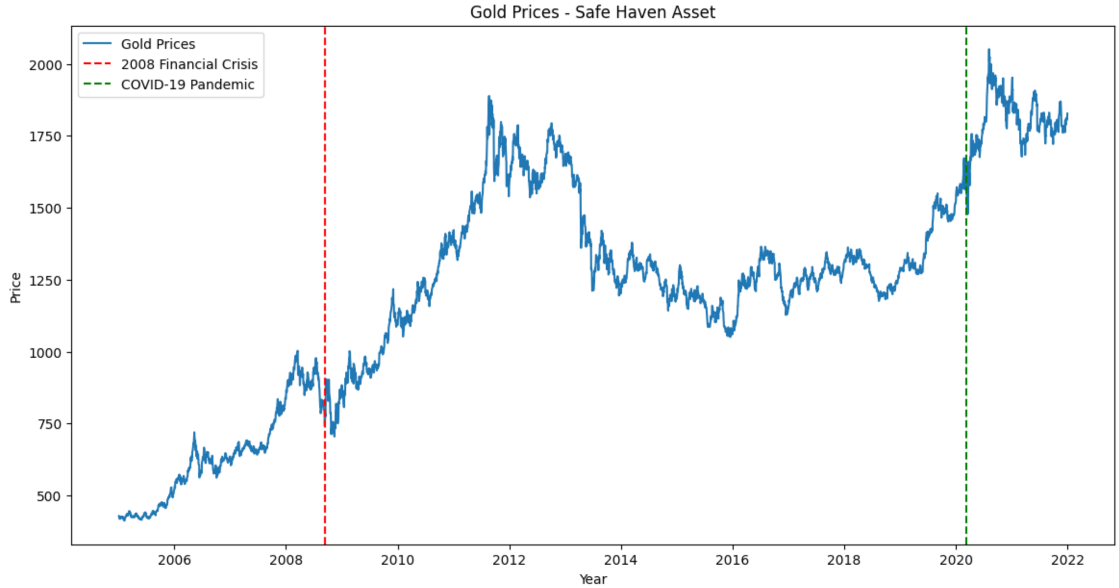
[10]:
```
import yfinance as yf
from statsmodels.tsa.stattools import adfuller

# Downloading historical data for Gold and Oil
gold = yf.download('GC=F', start='2000-01-01', end='2022-01-01')['Close']
oil = yf.download('CL=F', start='2000-01-01', end='2022-01-01')['Close']
```

```
# Performing the ADF test on Gold prices
adf_result_gold = adfuller(gold.dropna())
print('ADF Statistic for Gold: %f' % adf_result_gold[0])
print('p-value for Gold: %f' % adf_result_gold[1])

# Performing the ADF test on Oil prices
adf_result_oil = adfuller(oil.dropna())
print('ADF Statistic for Oil: %f' % adf_result_oil[0])
print('p-value for Oil: %f' % adf_result_oil[1])
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
ADF Statistic for Gold: -0.811471
p-value for Gold: 0.815729
ADF Statistic for Oil: -2.600286
p-value for Oil: 0.092938
```

[11]:
```
import yfinance as yf
import matplotlib.pyplot as plt

# Downloading historical data for Gold
gold = yf.download('GC=F', start='2005-01-01', end='2022-01-01')

# Plotting Gold prices during the 2008 financial crisis and COVID-19 pandemic
plt.figure(figsize=(14, 7))
plt.plot(gold['Close'], label='Gold Prices')
plt.axvline(x=pd.Timestamp('2008-09-15'), color='r', linestyle='--',␣
 ↪label='2008 Financial Crisis')
plt.axvline(x=pd.Timestamp('2020-03-11'), color='g', linestyle='--',␣
 ↪label='COVID-19 Pandemic')
plt.title('Gold Prices - Safe Haven Asset')
plt.xlabel('Year')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
```



1.4 Non-stationarity model

[12]:
```
import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
from math import sqrt
import itertools
import warnings

# Download S&P 500 data
ticker_symbol = "^GSPC"
start_date = "2010-01-01"
end_date = "2024-01-01"
data = yf.download(ticker_symbol, start=start_date, end=end_date)
data.head()

# Preprocess the data
data = data[['Adj Close']].dropna()

# Split the data into training and test sets
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False)
```

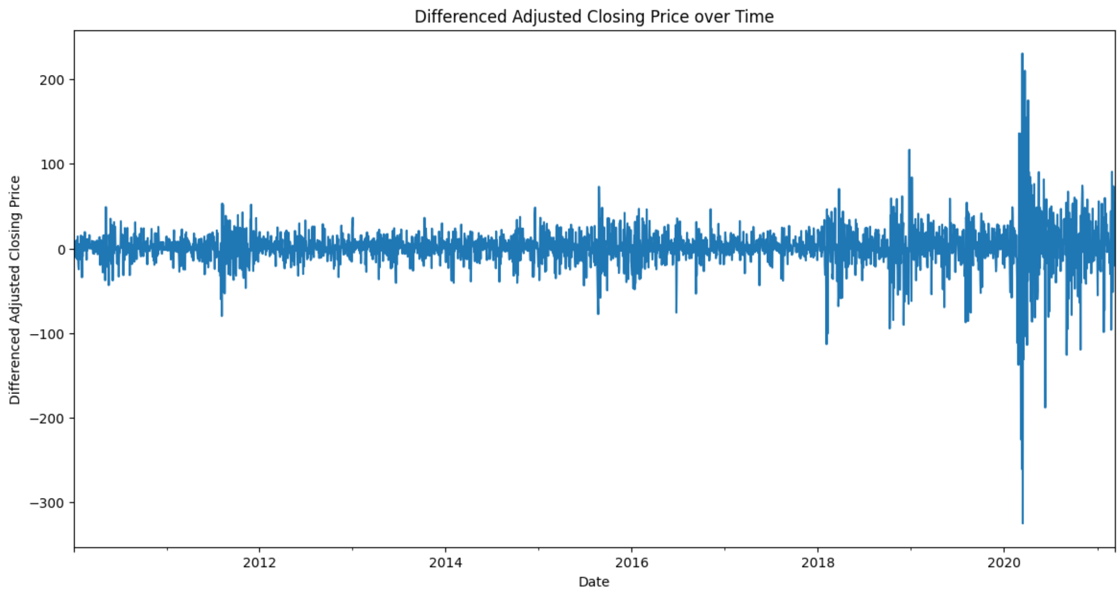
```
test_data.head()

# Ensure DateTimeIndex in the datasets and set the frequency
train_data.index = pd.to_datetime(train_data.index).to_period('B')
test_data.index = pd.to_datetime(test_data.index).to_period('B')

# Dickey-Fuller Test
print('Results of Dickey-Fuller Test:')
dftest = adfuller(train_data['Adj Close'], autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags␣
↪Used', 'Number of Observations Used'])
for key, value in dftest[4].items():
    dfoutput['Critical Value (%s)' % key] = value
print(dfoutput)

# Differencing the data
train_data_diff = train_data['Adj Close'].diff().dropna()

# Plot the differenced data
train_data_diff.plot(figsize=(14, 7))
plt.title('Differenced Adjusted Closing Price over Time')
plt.xlabel('Date')
plt.ylabel('Differenced Adjusted Closing Price')
plt.show()

# Perform Dickey-Fuller test on the differenced data
print('Results of Dickey-Fuller Test on Differenced Data:')
dftest_diff = adfuller(train_data_diff, autolag='AIC')
dfoutput_diff = pd.Series(dftest_diff[0:4], index=['Test Statistic', 'p-value',␣
↪'#Lags Used', 'Number of Observations Used'])
for key, value in dftest_diff[4].items():
    dfoutput_diff['Critical Value (%s)' % key] = value
print(dfoutput_diff)

# ACF and PACF plots
plt.figure(figsize=(14,7))
plt.subplot(211)
plot_acf(train_data_diff, ax=plt.gca(), lags=40)
plt.title('Autocorrelation Function')

plt.subplot(212)
plot_pacf(train_data_diff, ax=plt.gca(), lags=40)
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
plt.show()

# Fit the SARIMA model
```

```python
model = SARIMAX(train_data['Adj Close'], order=(1, 1, 1), seasonal_order=(0, 0,␣
↪0, 0))
results = model.fit()
print(results.summary())

# Get the predictions using integer-based locations
start_loc = train_data.shape[0]
end_loc = start_loc + test_data.shape[0] - 1
predictions = results.get_prediction(start=start_loc, end=end_loc,␣
↪dynamic=False)
predictions_aligned = pd.Series(predictions.predicted_mean.values,␣
↪index=test_data.index)

# Convert PeriodIndex back to DatetimeIndex for plotting
train_data_plot = train_data.to_timestamp()
test_data_plot = test_data.to_timestamp()

# Plot the predictions
plt.figure(figsize=(14, 7))
plt.plot(train_data_plot['Adj Close'], label='Training Data')
plt.plot(test_data_plot['Adj Close'], label='Actual Prices')
plt.plot(predictions_aligned.to_timestamp(), label='Predicted Prices')  #␣
↪Convert predictions index for plotting
plt.title('S&P 500 Prices Prediction')
plt.xlabel('Date')
plt.ylabel('Prices')
plt.legend()
plt.show()

# Calculate RMSE for the initial model
rmse = sqrt(mean_squared_error(test_data['Adj Close'], predictions_aligned))
print(f'The Root Mean Squared Error of our forecasts is {rmse}')

# Grid search for SARIMA parameters
p = d = q = range(0, 3)
pdq = list(itertools.product(p, d, q))
warnings.filterwarnings("ignore")  # Ignore warning messages

best_aic = float("inf")
best_pdq = None
best_model = None

for param in pdq:
    try:
        mod = SARIMAX(train_data['Adj Close'], order=param, seasonal_order=(0,␣
↪0, 0, 0),
                            enforce_stationarity=False, enforce_invertibility=False)
```

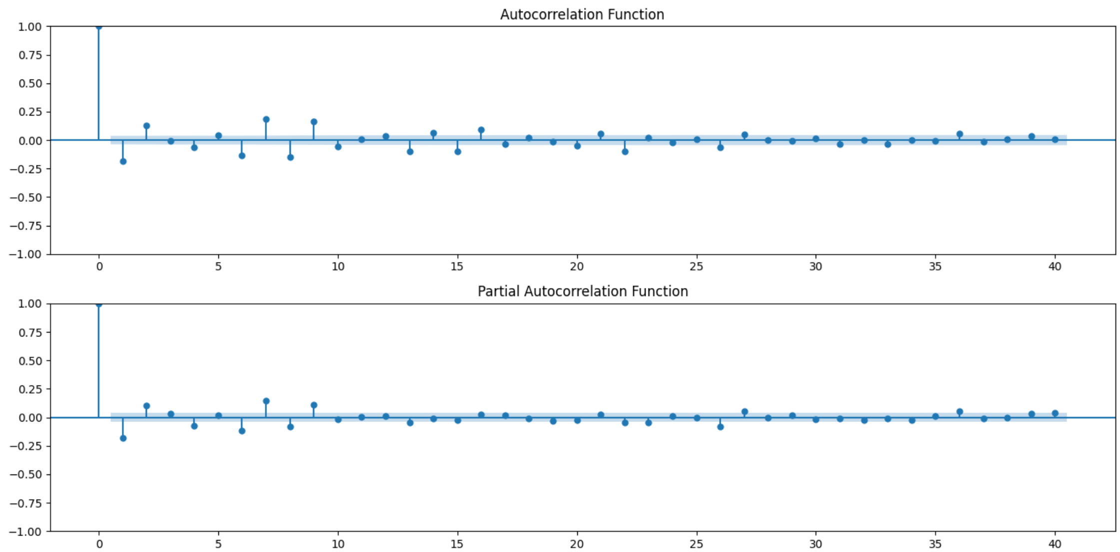
```python
        results = mod.fit()

        if results.aic < best_aic:
            best_aic = results.aic
            best_pdq = param
            best_model = results
    except:
        continue

print(f"Best SARIMA{best_pdq} model - AIC:{best_aic}")

# Fit the best SARIMA model
best_model = SARIMAX(train_data['Adj Close'], order=best_pdq,␣
↪seasonal_order=(0, 0, 0, 0),
                     enforce_stationarity=False, enforce_invertibility=False)
best_results = best_model.fit()
print(best_results.summary())

# Get predictions aligned with test data for best model
best_predictions = best_results.get_prediction(start=start_loc, end=end_loc,␣
↪dynamic=False)
best_predictions_aligned = pd.Series(best_predictions.predicted_mean.values,␣
↪index=test_data.index)

# Plot the predictions of the best model
plt.figure(figsize=(14, 7))
plt.plot(train_data_plot['Adj Close'], label='Training Data')
plt.plot(test_data_plot['Adj Close'], label='Actual Prices')
plt.plot(best_predictions_aligned.to_timestamp(), label='Predicted Prices')
plt.title('Best SARIMA Model - S&P 500 Prices Prediction')
plt.xlabel('Date')
plt.ylabel('Prices')
plt.legend()
plt.show()

# RMSE calculation for best model
best_rmse = sqrt(mean_squared_error(test_data['Adj Close'],␣
↪best_predictions_aligned))
print(f'The Root Mean Squared Error of the best model forecasts is {best_rmse}')
```

```
[*********************100%%**********************]  1 of 1 completed
Results of Dickey-Fuller Test:
Test Statistic                    0.802741
p-value                           0.991680
#Lags Used                       27.000000
Number of Observations Used    2789.000000
Critical Value (1%)              -3.432697
```

```
Critical Value (5%)            -2.862577
Critical Value (10%)           -2.567322
dtype: float64
```

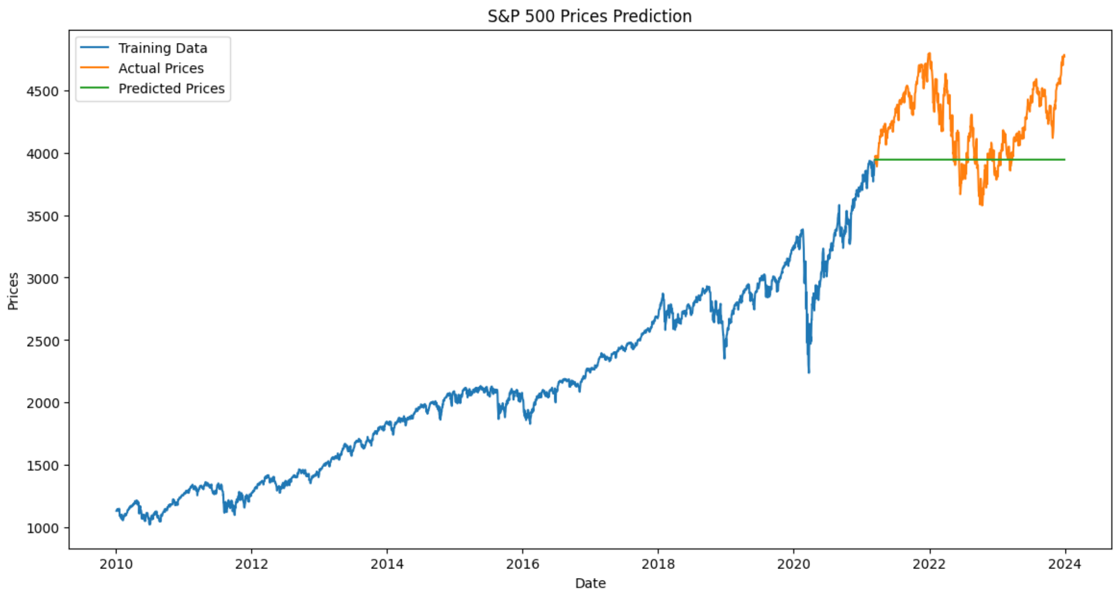



```
Results of Dickey-Fuller Test on Differenced Data:
Test Statistic                -1.128234e+01
p-value                        1.445866e-20
#Lags Used                     2.600000e+01
Number of Observations Used    2.789000e+03
Critical Value (1%)           -3.432697e+00
Critical Value (5%)           -2.862577e+00
Critical Value (10%)          -2.567322e+00
dtype: float64
```



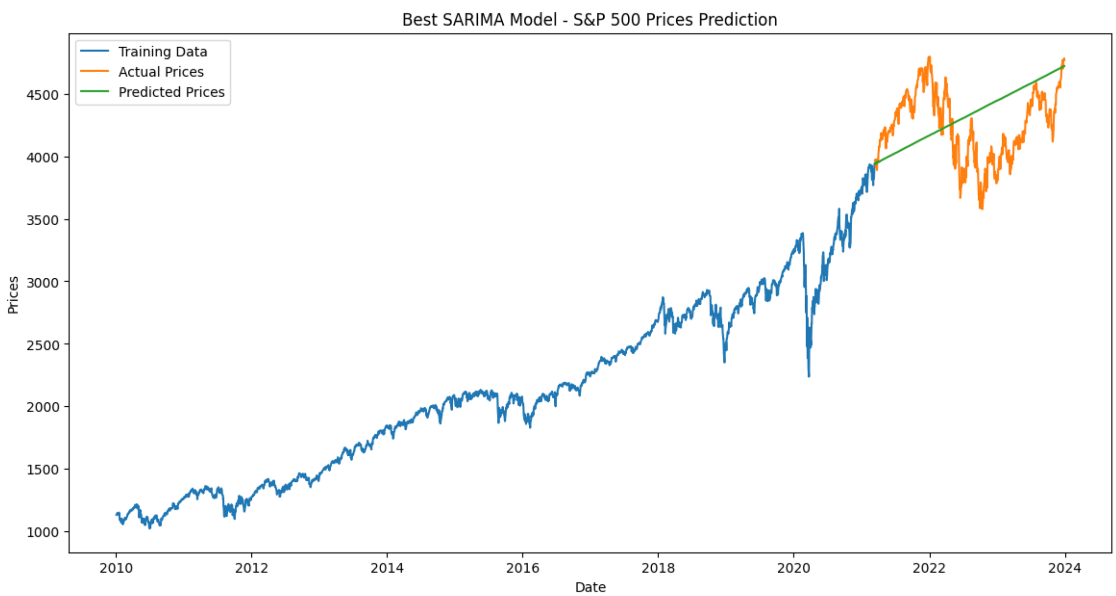

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Adj Close   No. Observations:                 2817
Model:               SARIMAX(1, 1, 1)   Log Likelihood              -13019.622
Date:                Sun, 07 Jan 2024   AIC                          26045.243
Time:                        01:13:36   BIC                          26063.073
Sample:                    01-04-2010   HQIC                         26051.677
                         - 03-12-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4534      0.024    -18.850      0.000      -0.501      -0.406
ma.L1          0.2763      0.026     10.469      0.000       0.225       0.328
sigma2       607.2641      4.722    128.611      0.000     598.010     616.519
================================================================================
===
Ljung-Box (L1) (Q):                   0.40   Jarque-Bera (JB):
67035.03
Prob(Q):                              0.53   Prob(JB):
0.00
Heteroskedasticity (H):               7.34   Skew:
-1.49
Prob(H) (two-sided):                  0.00   Kurtosis:
26.72
================================================================================
===

Warnings:
```

[1] Covariance matrix calculated using the outer product of gradients (complex-step).




```
The Root Mean Squared Error of our forecasts is 406.8216677069516
Best SARIMA(2, 2, 2) model - AIC:25992.49377739704
                               SARIMAX Results
==============================================================================
Dep. Variable:              Adj Close   No. Observations:                 2817
Model:               SARIMAX(2, 2, 2)   Log Likelihood              -12991.247
Date:                Sun, 07 Jan 2024   AIC                          25992.494
Time:                        01:13:54   BIC                          26022.202
Sample:                    01-04-2010   HQIC                         26003.215
                         - 03-12-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.1348      0.011    -99.420      0.000      -1.157      -1.112
ar.L2         -0.2090      0.006    -36.858      0.000      -0.220      -0.198
ma.L1         -0.0393      0.012     -3.303      0.001      -0.063      -0.016
ma.L2         -0.9615      0.011    -86.302      0.000      -0.983      -0.940
sigma2       600.9165      5.412    111.029      0.000     590.309     611.524
================================================================================
===
Ljung-Box (L1) (Q):                   2.58   Jarque-Bera (JB):
52156.84
Prob(Q):                              0.11   Prob(JB):
0.00
```

```
Heteroskedasticity (H):                6.86   Skew:
-1.57
Prob(H) (two-sided):                   0.00   Kurtosis:
23.86
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```




```
The Root Mean Squared Error of the best model forecasts is 373.64985991306514
```