2441843-week-1-time-series

October 21, 2024

1 Time Series, ARIMA, ARIMAX, SARIMA

Objective : Working with the Microsoft Stocks dataset to perform time series analysis.

1.0.1 Import the required libraries:

Import Pandas and alias it as pd.

Import NumPy and alias it as np.

Import Matplotlib and alias it as plt.

Import Seaborn and alias it as sns.

Import statsmodels and alias it as sm.

Import Scikit-learn and alias it as sklearn.

1.0.2 Load the Microsoft Stocks dataset:

Load the 'microsoft_stocks.csv' file using the Pandas library and assign it to a variable named 'data'

Ensure the 'Date' column is of type datetime.

1.0.3 Explore the dataset:

Display the first 5 rows of the dataset.

Display the number of rows and columns in the dataset.

Display the summary statistics of the dataset.

[1]:
```
!pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata
(7.8 kB)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.10/site-
packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
/opt/conda/lib/python3.10/site-packages (from pmdarima) (3.0.10)
Requirement already satisfied: numpy>=1.21.2 in /opt/conda/lib/python3.10/site-
```

```
packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in /opt/conda/lib/python3.10/site-
packages (from pmdarima) (2.2.3)
Requirement already satisfied: scikit-learn>=0.22 in
/opt/conda/lib/python3.10/site-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /opt/conda/lib/python3.10/site-
packages (from pmdarima) (1.14.1)
Requirement already satisfied: statsmodels>=0.13.2 in
/opt/conda/lib/python3.10/site-packages (from pmdarima) (0.14.2)
Requirement already satisfied: urllib3 in /opt/conda/lib/python3.10/site-
packages (from pmdarima) (1.26.18)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/opt/conda/lib/python3.10/site-packages (from pmdarima) (70.0.0)
Requirement already satisfied: packaging>=17.1 in
/opt/conda/lib/python3.10/site-packages (from pmdarima) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from packaging>=17.1->pmdarima) (3.1.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.10/site-packages (from pandas>=0.19->pmdarima)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas>=0.19->pmdarima) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-
packages (from pandas>=0.19->pmdarima) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=0.22->pmdarima)
(3.5.0)
Requirement already satisfied: patsy>=0.5.6 in /opt/conda/lib/python3.10/site-
packages (from statsmodels>=0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages
(from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1.16.0)
Downloading pmdarima-2.0.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (2.1 MB)
                         2.1/2.1 MB
18.9 MB/s eta 0:00:0000:0100:01
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

[2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import sklearn
import plotly.express as px
from statsmodels.tsa.stattools import adfuller
import numpy as np
from scipy.fft import fft
```

```python
from scipy import fftpack
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.stats import boxcox
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from pmdarima import auto_arima
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)
from sklearn.metrics import mean_absolute_error, mean_squared_error
from prophet import Prophet
```

[3]:
```python
data = pd.read_csv('/kaggle/input/microsoft-stock-csv/Microsoft_Stock.csv')
data['Date']  = pd.to_datetime(data['Date'])
data['Date']
```

[3]:
```
0      2015-04-01 16:00:00
1      2015-04-02 16:00:00
2      2015-04-06 16:00:00
3      2015-04-07 16:00:00
4      2015-04-08 16:00:00
               …
1506   2021-03-25 16:00:00
1507   2021-03-26 16:00:00
1508   2021-03-29 16:00:00
1509   2021-03-30 16:00:00
1510   2021-03-31 16:00:00
Name: Date, Length: 1511, dtype: datetime64[ns]
```

[4]:
```python
data.head()
```

[4]:

	Date	Open	High	Low	Close	Volume
0	2015-04-01 16:00:00	40.60	40.76	40.31	40.72	36865322
1	2015-04-02 16:00:00	40.66	40.74	40.12	40.29	37487476
2	2015-04-06 16:00:00	40.34	41.78	40.18	41.55	39223692
3	2015-04-07 16:00:00	41.61	41.91	41.31	41.53	28809375
4	2015-04-08 16:00:00	41.48	41.69	41.04	41.42	24753438

[5]:
```python
#number of rows and columns in the dataset.
data.shape
```

[5]: (1511, 6)

[6]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1511 entries, 0 to 1510
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1511 non-null   datetime64[ns]
 1   Open    1511 non-null   float64
 2   High    1511 non-null   float64
 3   Low     1511 non-null   float64
 4   Close   1511 non-null   float64
 5   Volume  1511 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 71.0 KB
```

[7]:
```python
#summary statistics of the dataset.
data.describe(include='all')
```

[7]:
```
                                Date         Open         High          Low  \
count                           1511  1511.000000  1511.000000  1511.000000
mean   2018-03-31 17:23:44.751820032   107.385976   108.437472   106.294533
min              2015-04-01 16:00:00    40.340000    40.740000    39.720000
25%              2016-09-29 04:00:00    57.860000    58.060000    57.420000
50%              2018-04-02 16:00:00    93.990000    95.100000    92.920000
75%              2019-10-01 04:00:00   139.440000   140.325000   137.825000
max              2021-03-31 16:00:00   245.030000   246.130000   242.920000
std                              NaN    56.691333    57.382276    55.977155

             Close        Volume
count  1511.000000  1.511000e+03
mean    107.422091  3.019863e+07
min      40.290000  1.016120e+05
25%      57.855000  2.136213e+07
50%      93.860000  2.662962e+07
75%     138.965000  3.431962e+07
max     244.990000  1.352271e+08
std      56.702299  1.425266e+07
```

[ ]:
```python

```

1.0.4 Data preprocessing:

Check for missing values and handle them appropriately.

Set the 'Date' column as the index of the dataset.

Visualize the stock prices over time using an appropriate plot.

1.0.5 Time series decomposition:

Perform time series decomposition to extract the trend, seasonality, and residual components of the stock prices.

Visualize the trend, seasonality, and residual components using appropriate plots.

Analyze the components and interpret the results.

[8]:
```
# Check for missing values
data.isnull().sum()
```

[8]:
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

[9]:
```
# Date' column as the index of the dataset.
data.reset_index(drop = True)
```

[9]:
```
                    Date    Open    High     Low   Close    Volume
0    2015-04-01 16:00:00   40.60   40.76   40.31   40.72  36865322
1    2015-04-02 16:00:00   40.66   40.74   40.12   40.29  37487476
2    2015-04-06 16:00:00   40.34   41.78   40.18   41.55  39223692
3    2015-04-07 16:00:00   41.61   41.91   41.31   41.53  28809375
4    2015-04-08 16:00:00   41.48   41.69   41.04   41.42  24753438
…                     …      …       …       …       …         …
1506 2021-03-25 16:00:00  235.30  236.94  231.57  232.34  34061853
1507 2021-03-26 16:00:00  231.55  236.71  231.55  236.48  25479853
1508 2021-03-29 16:00:00  236.59  236.80  231.88  235.24  25227455
1509 2021-03-30 16:00:00  233.53  233.85  231.10  231.85  24792012
1510 2021-03-31 16:00:00  232.91  239.10  232.39  235.77  43623471

[1511 rows x 6 columns]
```

[10]:
```
fig = px.line(data, x='Date', y=['Close', 'Open', 'High'],
              labels={'value': 'Price', 'variable': 'Stock Metric'},
              title='Stock Prices over Time')
fig
```

[11]:
```
# Checking the stationarity of data : Augmented Dickey Fuller Test(ADF)

# Null Hypothesis (H0) : The time series  is non-stationary.
# Alternative Hypothesis (H1): The time series  is stationary.

for col in data.columns[1:]:
    print(f'Checking stationarity of {col} column')
    adf = adfuller(data[col])
    print('ADF Statistic:', adf[0])
    print('p-value:', adf[1])
```

```
    if adf[1] < 0.05:
        print(f"The '{col}' column series is stationary")
    else:
        print(f"The '{col}' column series is non-stationary")
    print('-'*100)
```

```
Checking stationarity of Open column
ADF Statistic: 0.8239150328103294
p-value: 0.9920125565435898
The 'Open' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of High column
ADF Statistic: 1.570419462304025
p-value: 0.997766061521655
The 'High' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Low column
ADF Statistic: 1.2248279399412962
p-value: 0.9961530109651404
The 'Low' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Close column
ADF Statistic: 1.7371362899270981
p-value: 0.9982158366942122
The 'Close' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Volume column
ADF Statistic: -6.899655612142817
p-value: 1.2918117349076232e-09
The 'Volume' column series is stationary
--------------------------------------------------------------------------------
--------------------
```

[12]:
```
# Checking the stationarity of data : Kwiatkowski-Phillips-Schmidt-Shin (KPSS)

# Null Hypothesis (H0): The series is stationary.
# Alternative Hypothesis (H1): The series is non-stationary.

for col in data.columns[1:]:
    print(f'Checking stationarity of {col} column')
    kpss_test = kpss(data[col])
    print('kpss Statistic:', kpss_test[0])
```

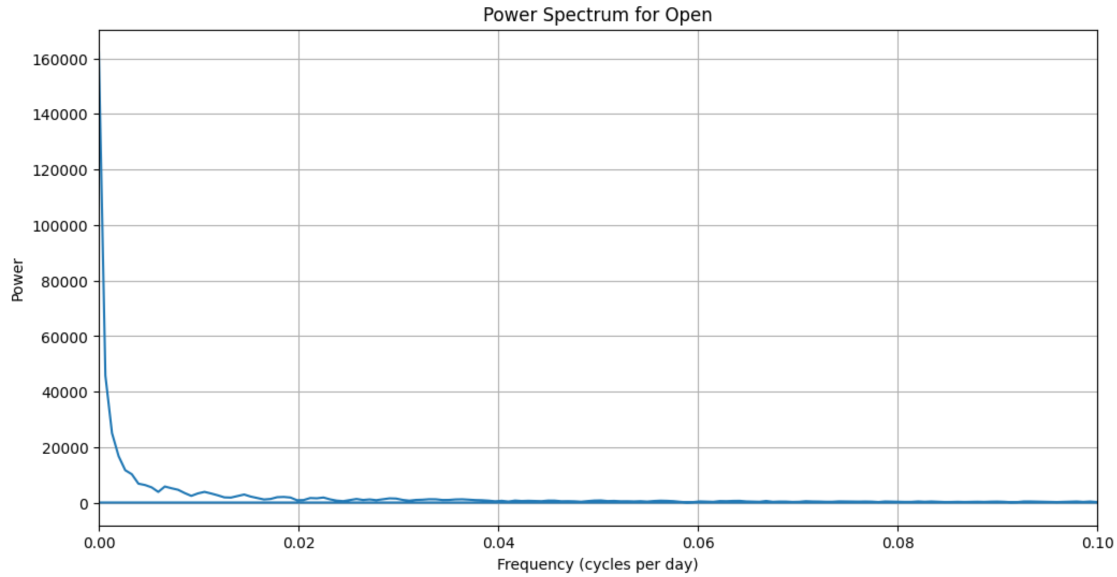
```python
    print('p-value:', kpss_test[1])
    print(f"Lags used: {kpss_test[2]}")

    if kpss_test[1] < 0.05:
        print(f"The '{col}' column series is non-stationary")
    else:
        print(f"The '{col}' column series is stationary")
    print('-'*100)
```

```
Checking stationarity of Open column
kpss Statistic: 5.4112290715555735
p-value: 0.01
Lags used: 25
The 'Open' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of High column
kpss Statistic: 5.409085565773386
p-value: 0.01
Lags used: 25
The 'High' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Low column
kpss Statistic: 5.409819063229962
p-value: 0.01
Lags used: 25
The 'Low' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Close column
kpss Statistic: 5.4106492349635955
p-value: 0.01
Lags used: 25
The 'Close' column series is non-stationary
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Volume column
kpss Statistic: 0.3169866007846065
p-value: 0.1
Lags used: 22
The 'Volume' column series is stationary
--------------------------------------------------------------------------------
--------------------
```

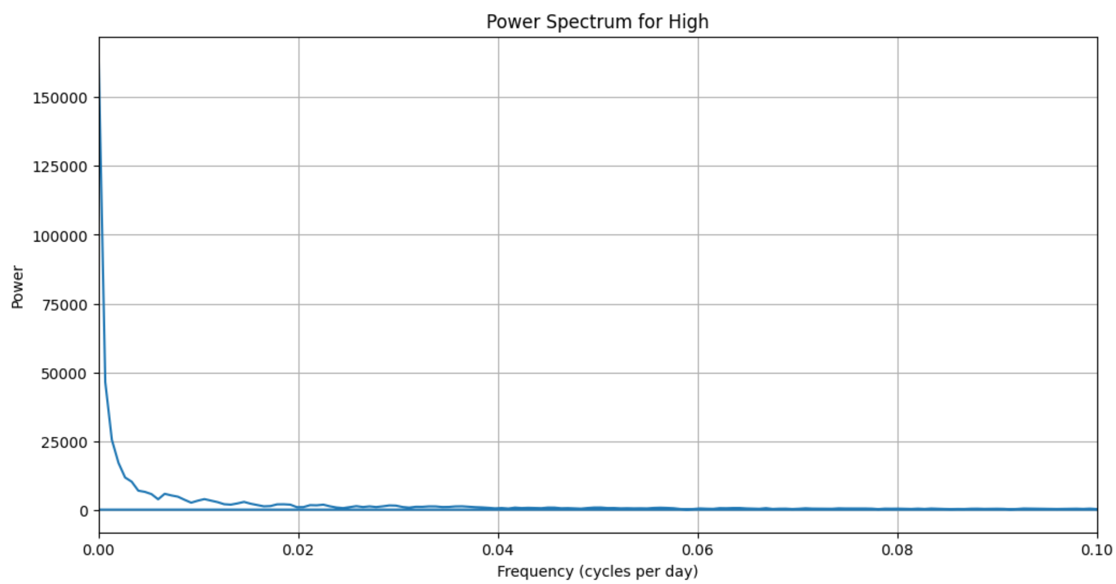
1.0.6 Perform time series decomposition to extract the trend, seasonality, and residual components of the stock prices.

[13]:
```
# Determining time period for seasonality

for col in data.columns[1:]:
    # Fourier Transformation
    open_fft = fft(data[col].dropna())  # Drop NaN values if any

    # Power of each frequency
    power = np.abs(open_fft)
    frequencies = fftpack.fftfreq(len(data[col].dropna()))  # Frequency array

    # Peak frequency (ignoring the zero frequency component)
    peak_frequency = frequencies[np.argmax(power[1:]) + 1]

    # Convert peak frequency to time period
    time_period = 1 / peak_frequency

    # Print results
    print(f"Time period for '{col}' column series")
    print(f"Detected peak frequency: {peak_frequency:.4f} cycles per day")
    print(f"Estimated period: {time_period:.2f} days")
    print(f'Time period in data set: {len(data)} days')

    # visualize the power spectrum
    plt.figure(figsize=(12, 6))
    plt.plot(frequencies, power)
    plt.title(f'Power Spectrum for {col}')
    plt.xlabel('Frequency (cycles per day)')
    plt.ylabel('Power')
    plt.xlim(0, 0.1)  # Adjust this to zoom in on significant frequencies
    plt.grid()
    plt.show()
    print('-'*100)
```

```
Time period for 'Open' column series
Detected peak frequency: 0.0007 cycles per day
Estimated period: 1511.00 days
Time period in data set: 1511 days
```

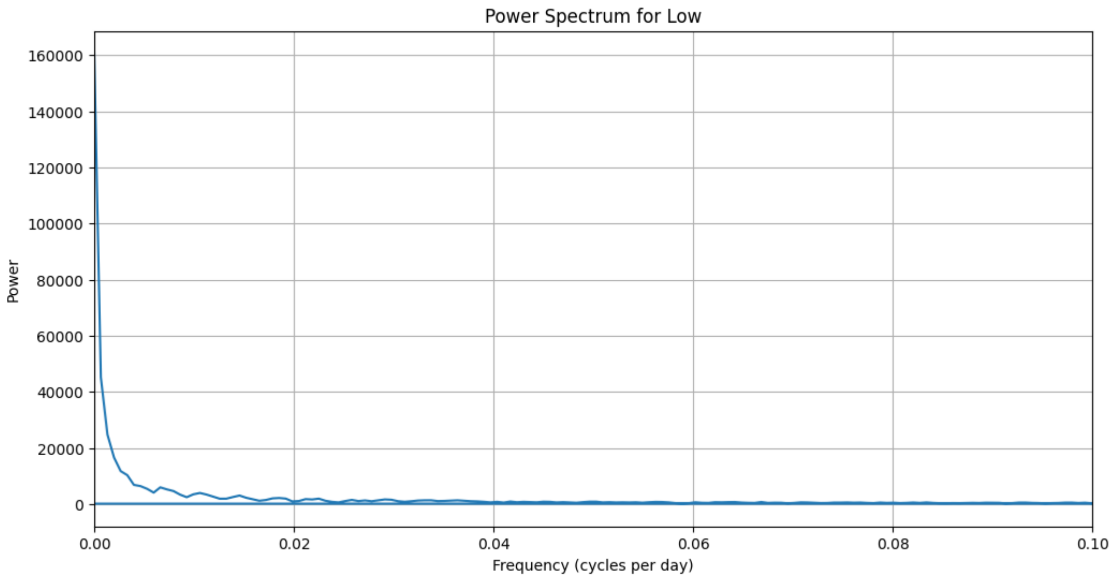



```
--------------------------------------------------------------------------------
--------------------
Time period for 'High' column series
Detected peak frequency: 0.0007 cycles per day
Estimated period: 1511.00 days
Time period in data set: 1511 days
```

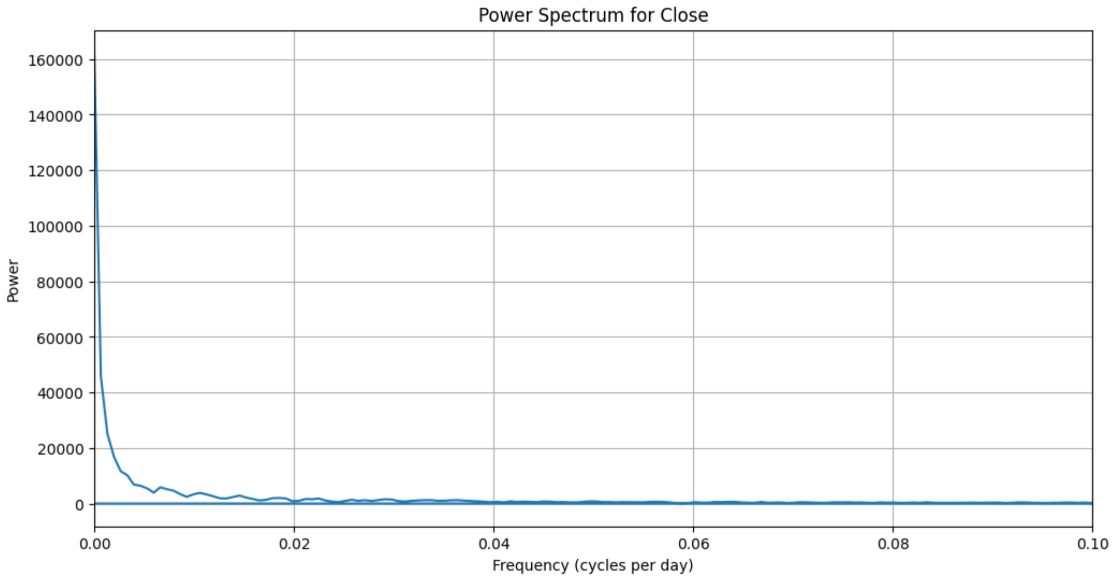



```
--------------------------------------------------------------------------------
--------------------
```

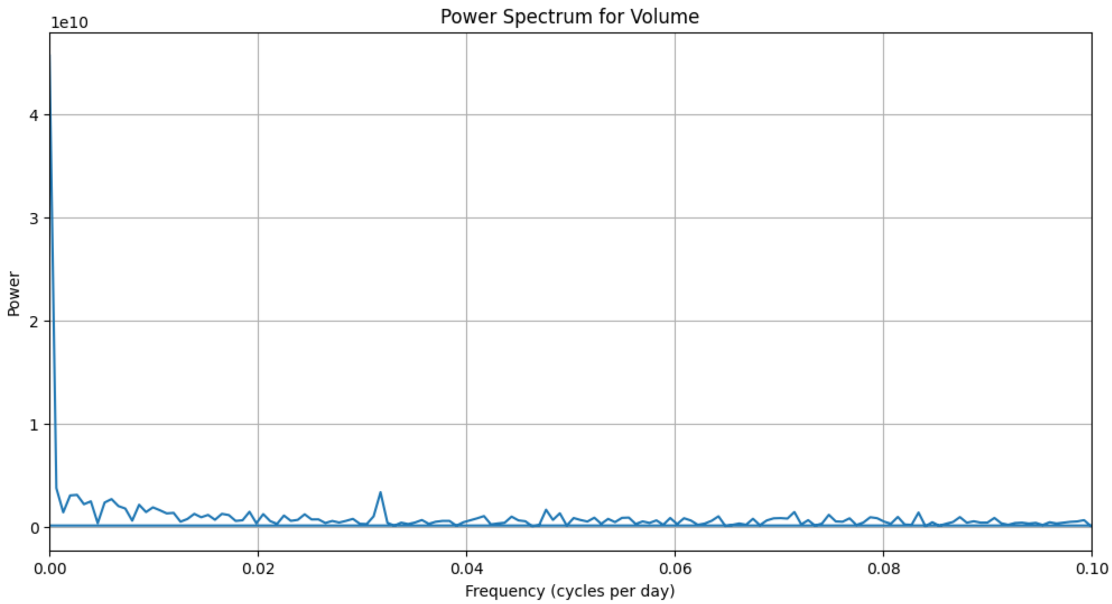
```
Time period for 'Low' column series
Detected peak frequency: 0.0007 cycles per day
Estimated period: 1511.00 days
Time period in data set: 1511 days
```




```
--------------------------------------------------------------------------------
--------------------
Time period for 'Close' column series
Detected peak frequency: 0.0007 cycles per day
Estimated period: 1511.00 days
Time period in data set: 1511 days
```



```
--------------------------------------------------------------------------------
--------------------
Time period for 'Volume' column series
Detected peak frequency: 0.0007 cycles per day
Estimated period: 1511.00 days
Time period in data set: 1511 days
```




```
--------------------------------------------------------------------------------
--------------------
```

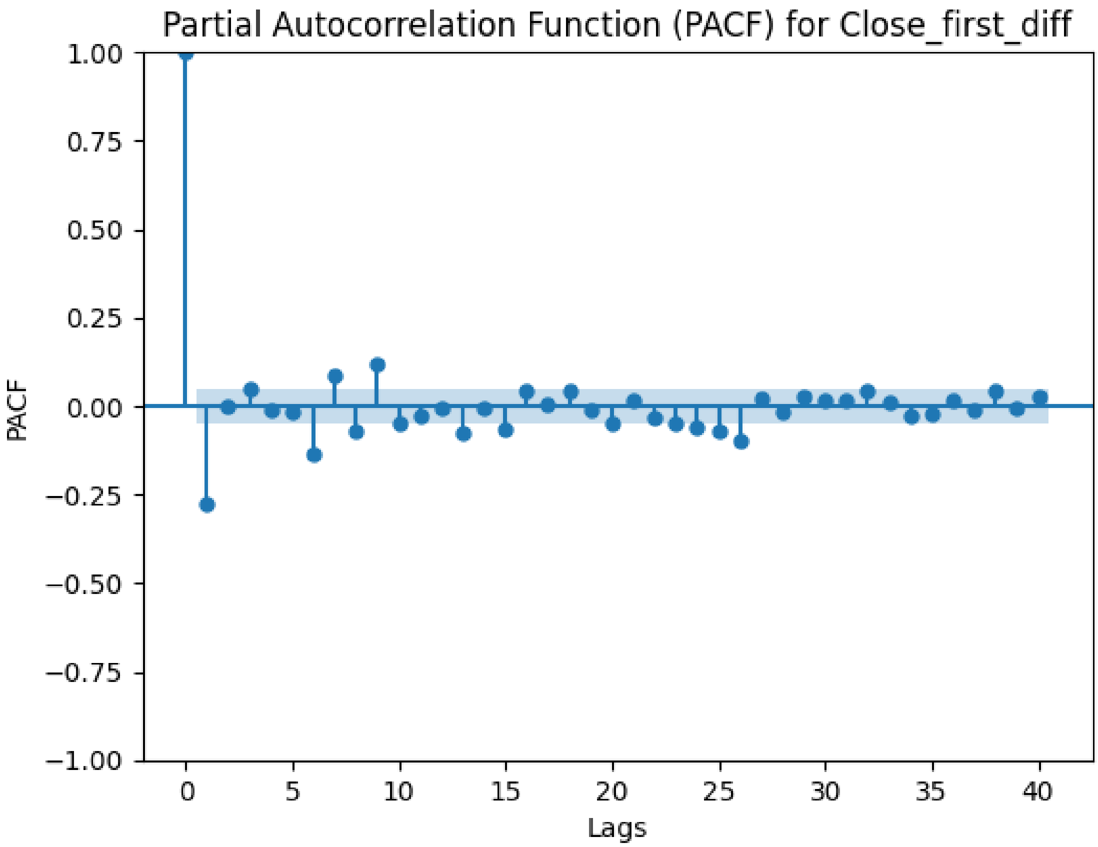
[14]:

```
# To determine seasonality in the Time Series

diff_dataframe = pd.DataFrame()

for col in ['Open','High','Low','Close']:
    print(f'Checking stationarity of {col} column')
    diff_dataframe[col + '_first_diff'] = data[col].diff()
    adf = adfuller(diff_dataframe[col + '_first_diff'].dropna())
    print('ADF Statistic:', adf[0])
    print('p-value:', adf[1])

    if adf[1] < 0.05:
        print(f"The '{col + '_first_diff'}' column series is stationary")
        d = 1
```

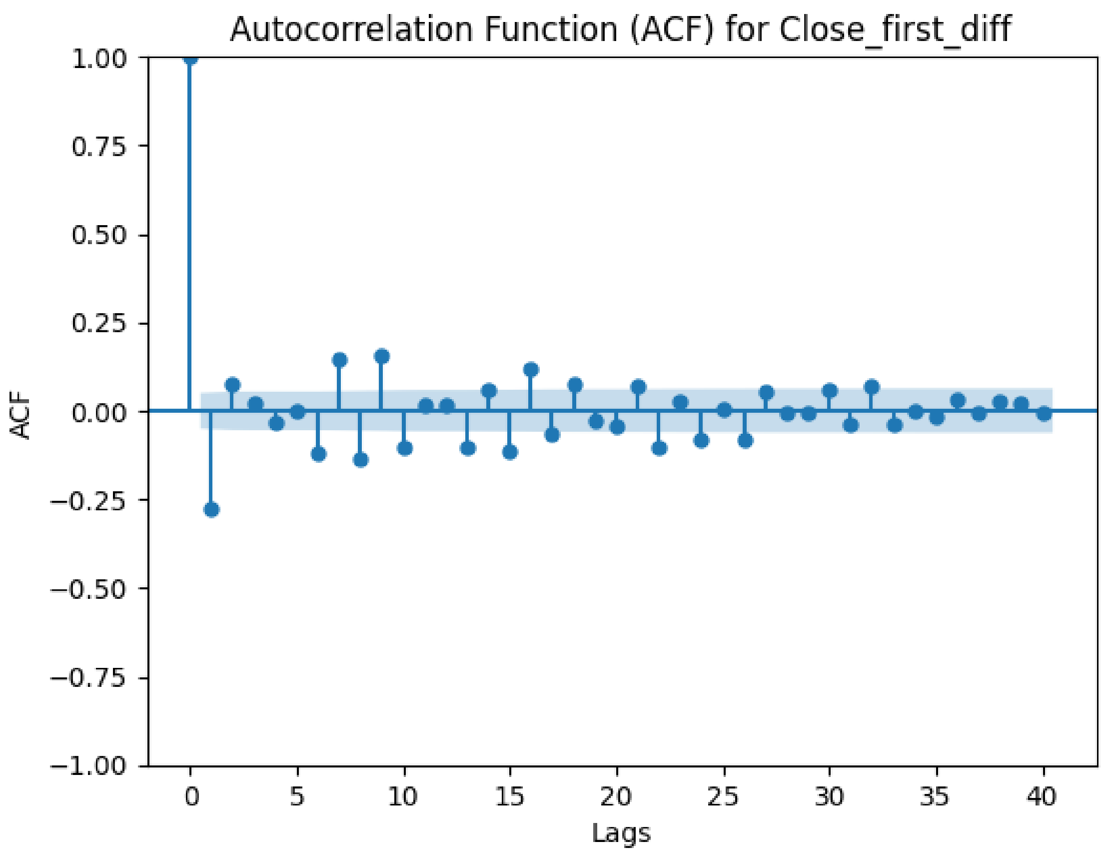
```
        print(f"since series became stationary in first differencing, the value␣
↪of, d={d}")
    else:
        print(f"The '{col + '_first_diff'}' column series is non-stationary")
    print('-'*100)

for col in ['Close_first_diff']:
    plt.figure(figsize=(10, 5))
    plot_pacf(diff_dataframe[col].dropna(), lags=40)
    plt.title(f"Partial Autocorrelation Function (PACF) for {col}")
    plt.xlabel('Lags')
    plt.ylabel('PACF')
    plt.show()

for col in ['Close_first_diff']:
    plt.figure(figsize=(10, 5))
    plot_acf(diff_dataframe[col].dropna(), lags=40)  # Plot ACF for the column
    plt.title(f'Autocorrelation Function (ACF) for {col}')
    plt.xlabel('Lags')
    plt.ylabel('ACF')
```

```
Checking stationarity of Open column
ADF Statistic: -9.913565139370291
p-value: 3.122658246367882e-17
The 'Open_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of High column
ADF Statistic: -9.976423755522784
p-value: 2.1715284202008017e-17
The 'High_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Low column
ADF Statistic: -11.808716652970046
p-value: 8.966571585274388e-22
The 'Low_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of Close column
ADF Statistic: -10.038331065146442
p-value: 1.5195939917528e-17
The 'Close_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
```

<Figure size 1000x500 with 0 Axes>



<Figure size 1000x500 with 0 Axes>



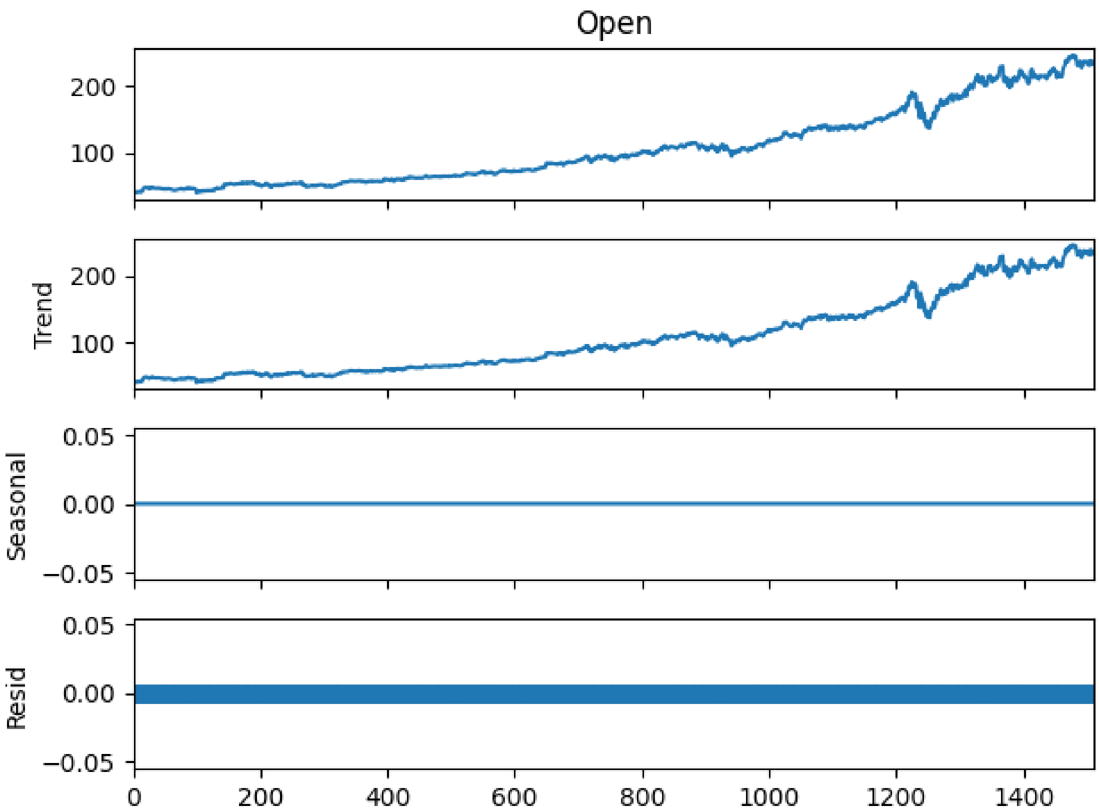
1.1 Additive Decomposisiton

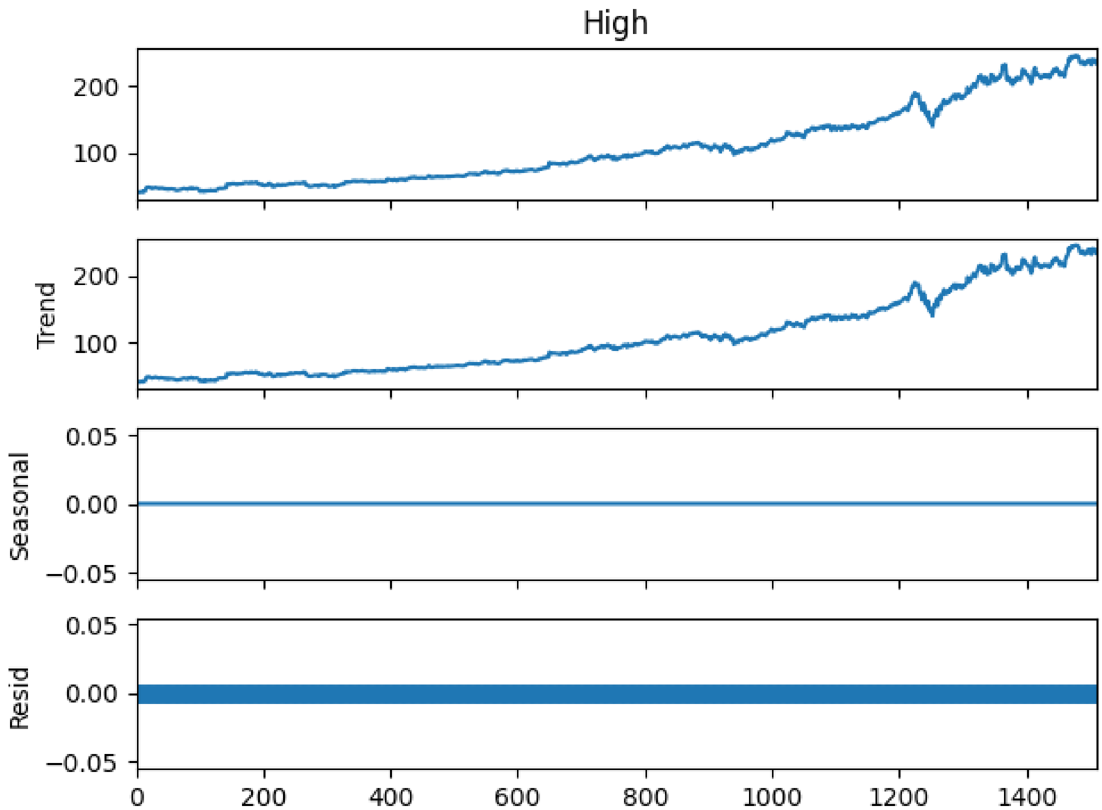
[15]:
```
period = 1 # No-seasonality
figsize = (10,5)
```

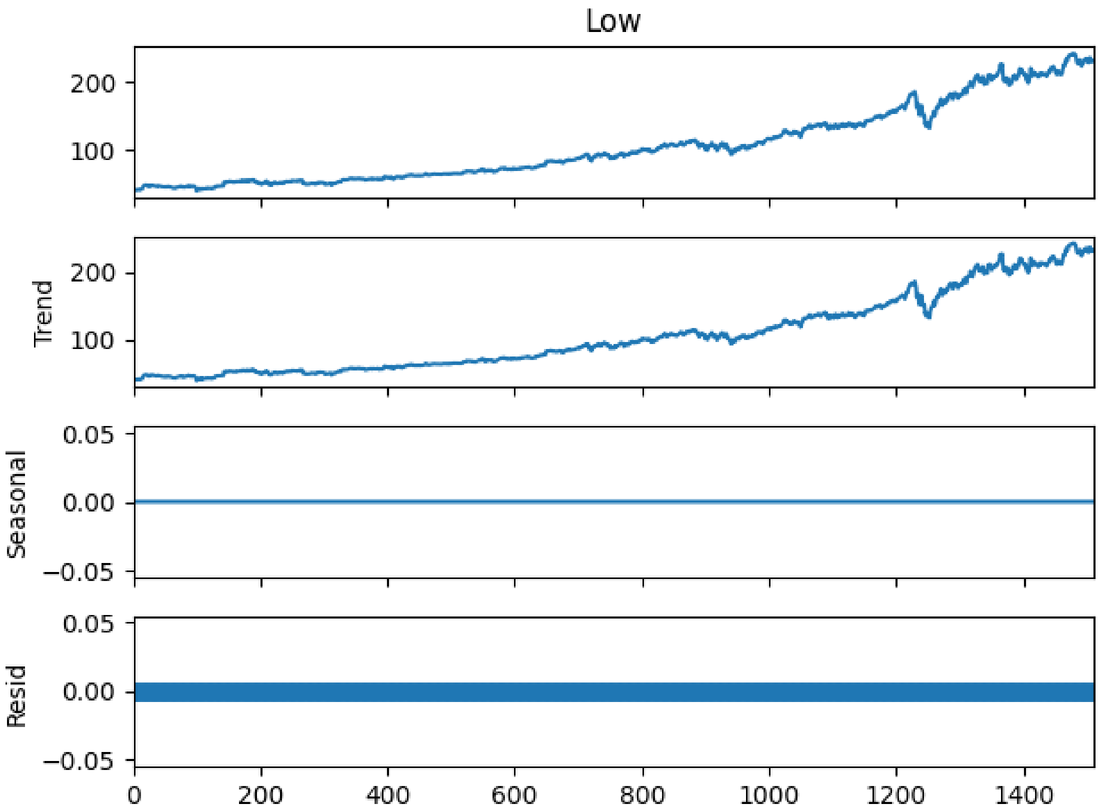
[16]:
```
### Additive Model

plt.figure(figsize=figsize)
for col in data.columns[1:]:
    print(f'Additive decomposisiton plot of {col} column')
    decomposition_additive = 
 ↪seasonal_decompose(data[col],model='additive',period=period)
    decomposition_additive.plot()
```

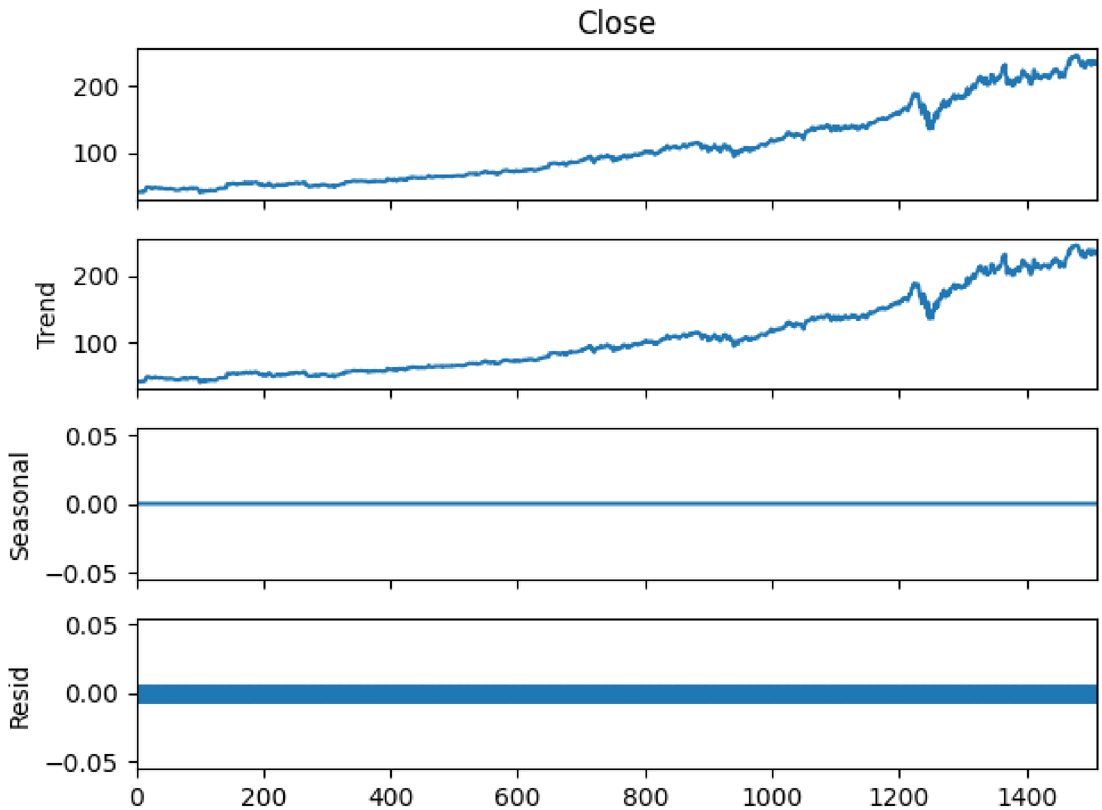
```
Additive decomposisiton plot of Open column
Additive decomposisiton plot of High column
Additive decomposisiton plot of Low column
Additive decomposisiton plot of Close column
Additive decomposisiton plot of Volume column
```

```
<Figure size 1000x500 with 0 Axes>
```

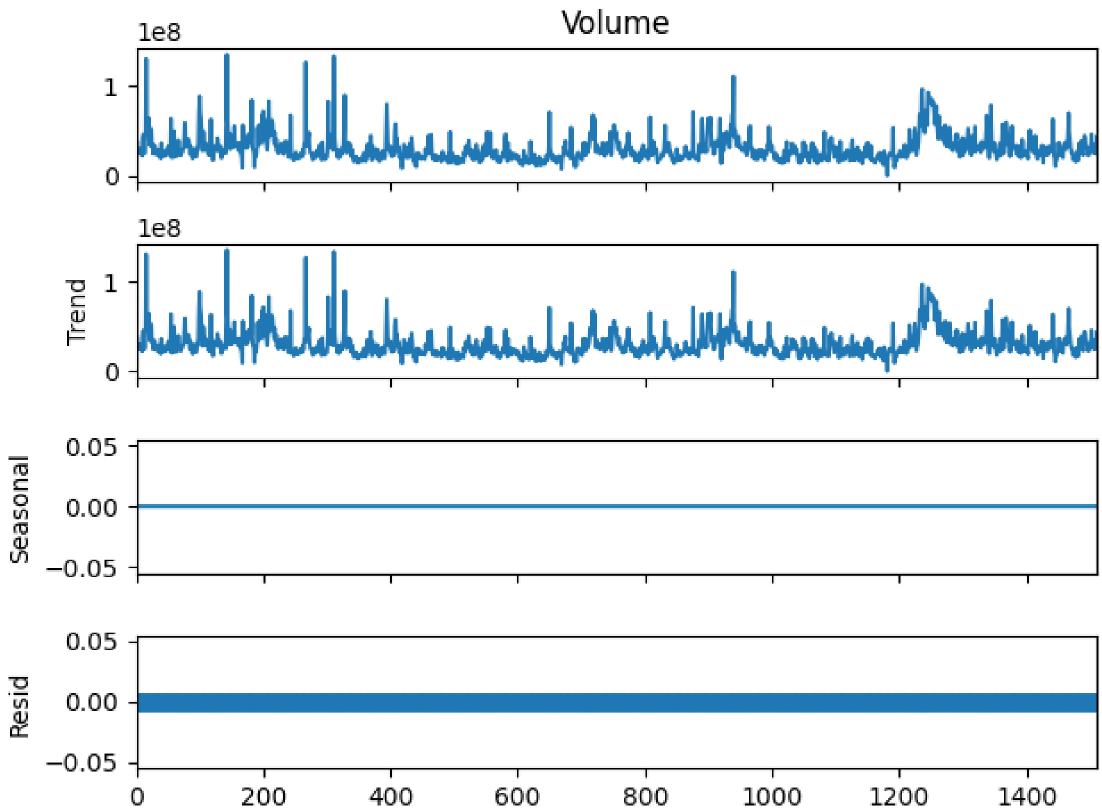

Open
200
100
Trend
200
100
Seasonal
0.05
0.00
−0.05
Resid
0.05
0.00
−0.05
0
200
400
600
800
1000
1200
1400


High
Trend
Seasonal
Resid
200
100
0.05
0.00
−0.05
0
200
400
600
800
1000
1200
1400


Low
200
100
Trend
200
100
Seasonal
0.05
0.00
−0.05
Resid
0.05
0.00
−0.05
0
200
400
600
800
1000
1200
1400

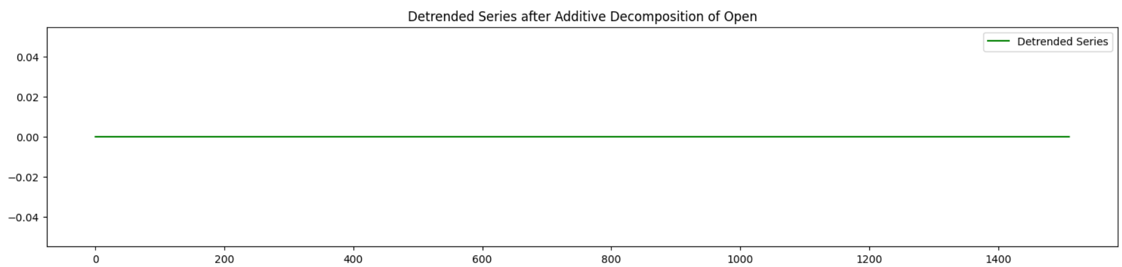


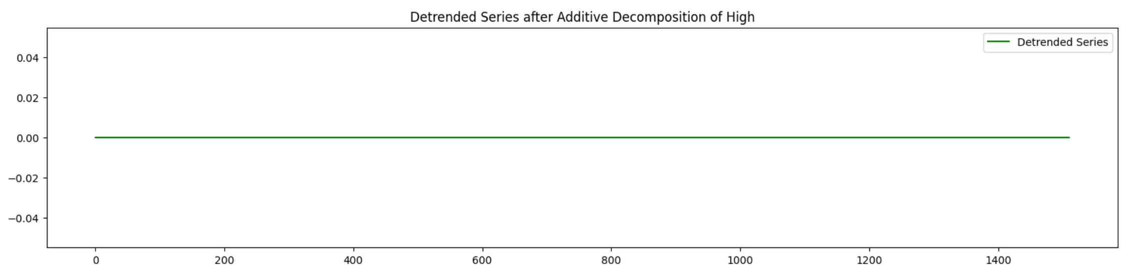
Close
200
100
Trend
200
100
Seasonal
0.05
0.00
−0.05
Resid
0.05
0.00
−0.05
0
200
400
600
800
1000
1200
1400

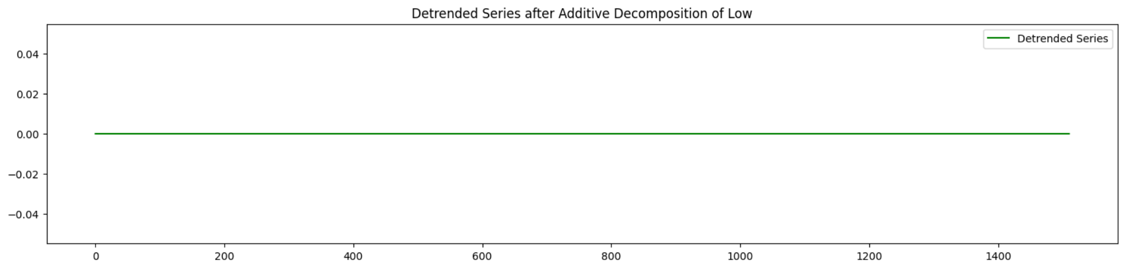


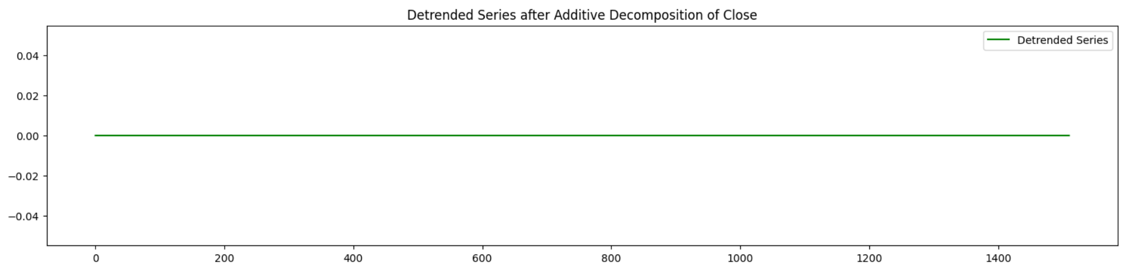
[17]:
```
# # Checking stationarity of the residuals of Additive Decomposition using ADF␣
↪test

# for col in data.columns[1:]:
#     print(f'Checking stationarity of Residuals of {col} column of Additive␣
↪Decomposition')
#     decomposition_additive =␣
↪seasonal_decompose(data[col],model='additive',period=period)
#     residuals = decomposition_additive.resid.dropna()
#     adf = adfuller(residuals)
#     print('ADF Statistic:', adf[0])
#     print('p-value:', adf[1])

#     if adf[1] < 0.05:
#         print("The residuals are stationary")
#     else:
#         print("The residuals are non-stationary")
#     print('-'*100)
```

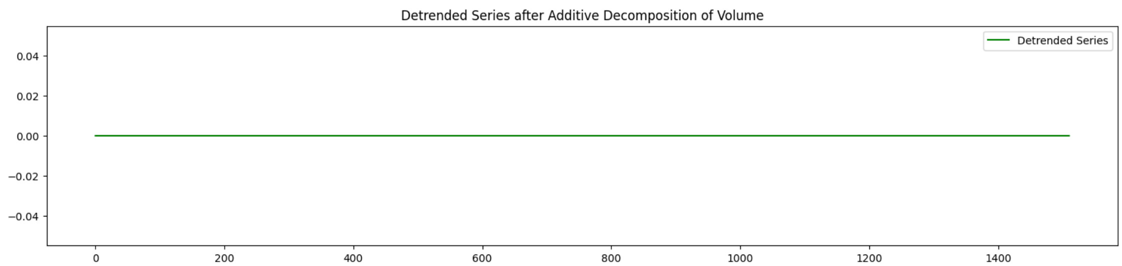
```
[18]: # De-Trending series after removing trend and seasonality

for col in data.columns[1:]:
    # Perform additive decomposition
    decomposition_additive = seasonal_decompose(data[col], model='additive',␣
 ↪period=period)

    # Detrended series calculation
    detrended_series = data[col] - decomposition_additive.trend -␣
 ↪decomposition_additive.seasonal

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

    # Plot the detrended series
    plt.subplot(3, 1, 3)
    plt.plot(detrended_series, label='Detrended Series', color='green')
    plt.title(f'Detrended Series after Additive Decomposition of {col}')
    plt.legend()
    plt.tight_layout()
```









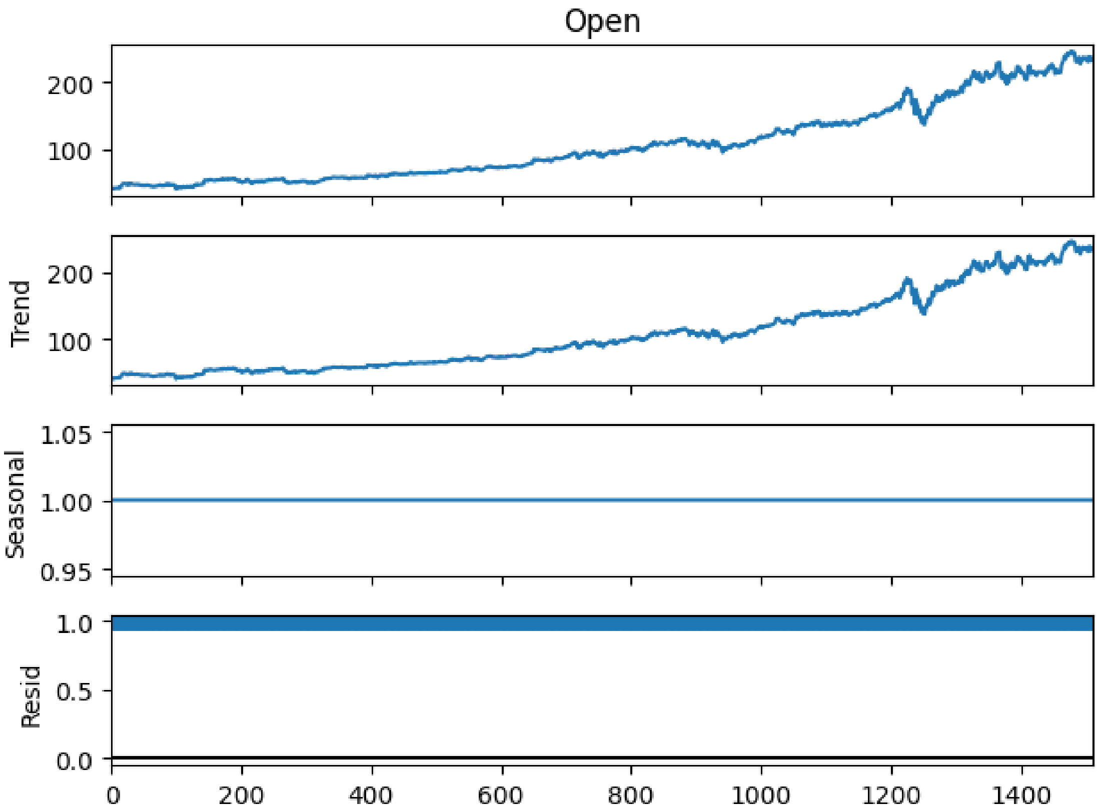


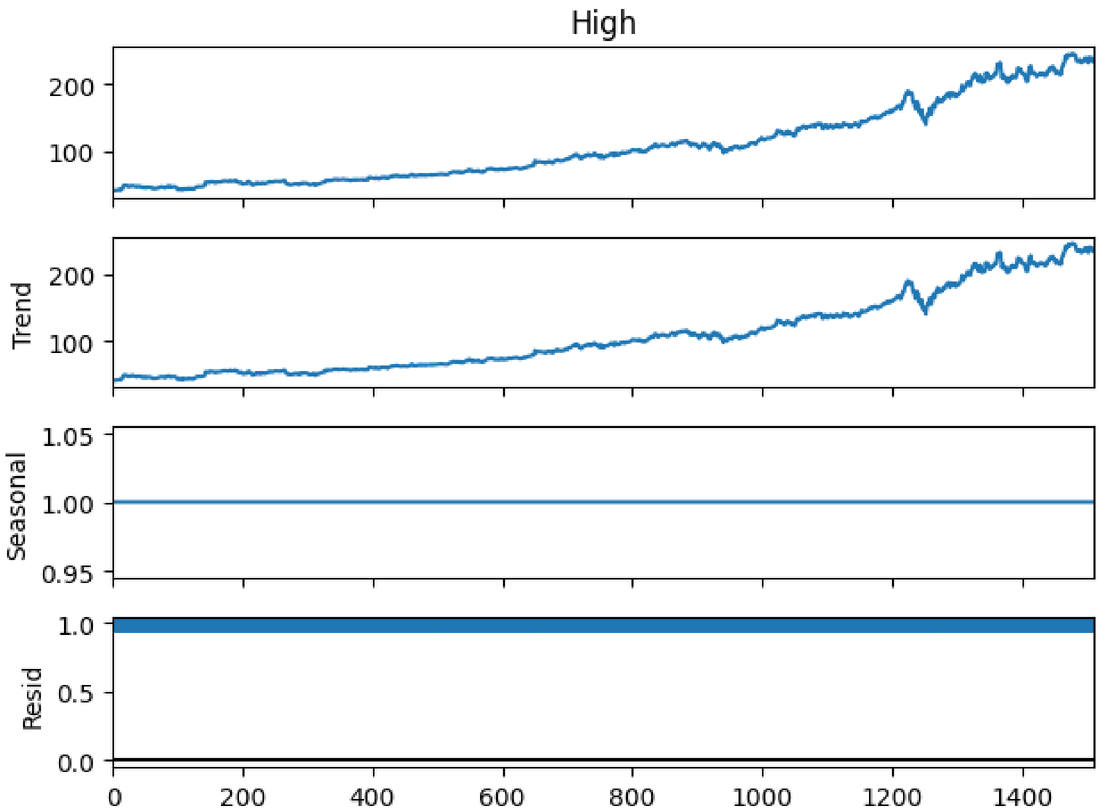
1.2 Muliplicative Decomposisiton

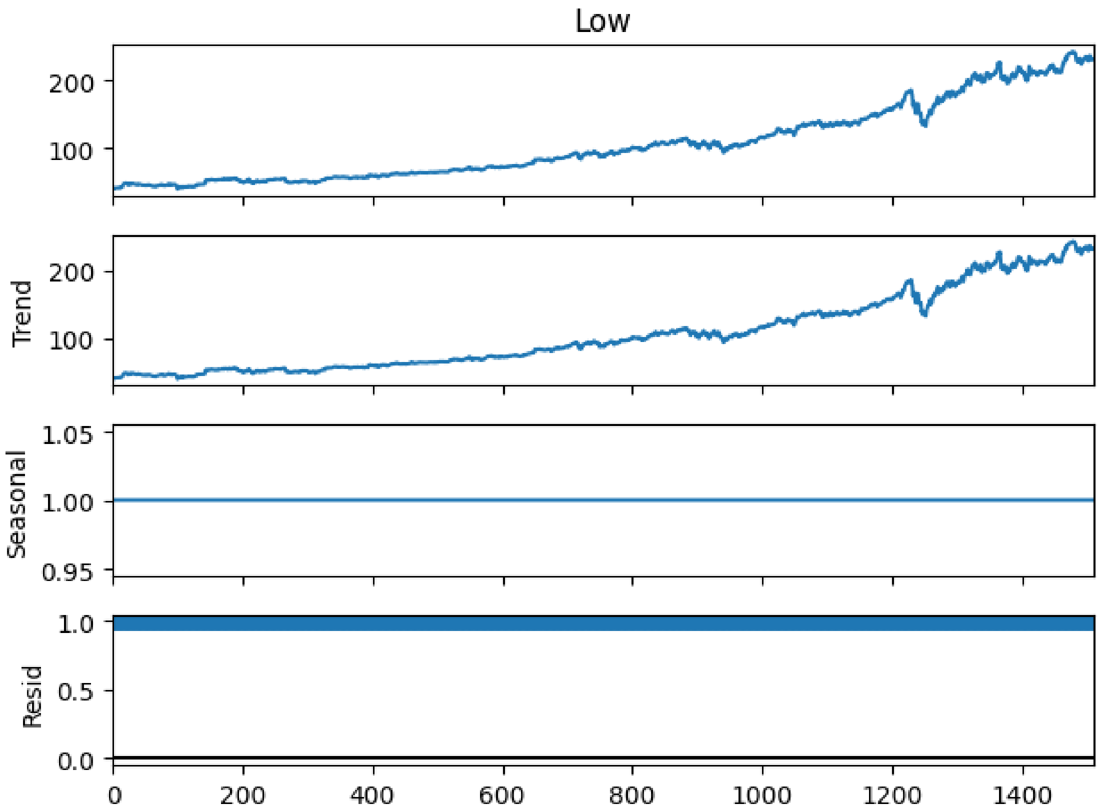
[19]:
```
### Multiplicate Model

plt.figure(figsize=figsize)
for col in data.columns[1:]:
    print(f'Muliplicative decomposisiton plot of {col} column')
    decomposition_multiplicative =␣
  ↪seasonal_decompose(data[col],model='multiplicative',period=1)
    decomposition_multiplicative.plot()
```

```
Muliplicative decomposisiton plot of Open column
Muliplicative decomposisiton plot of High column
Muliplicative decomposisiton plot of Low column
Muliplicative decomposisiton plot of Close column
Muliplicative decomposisiton plot of Volume column
```

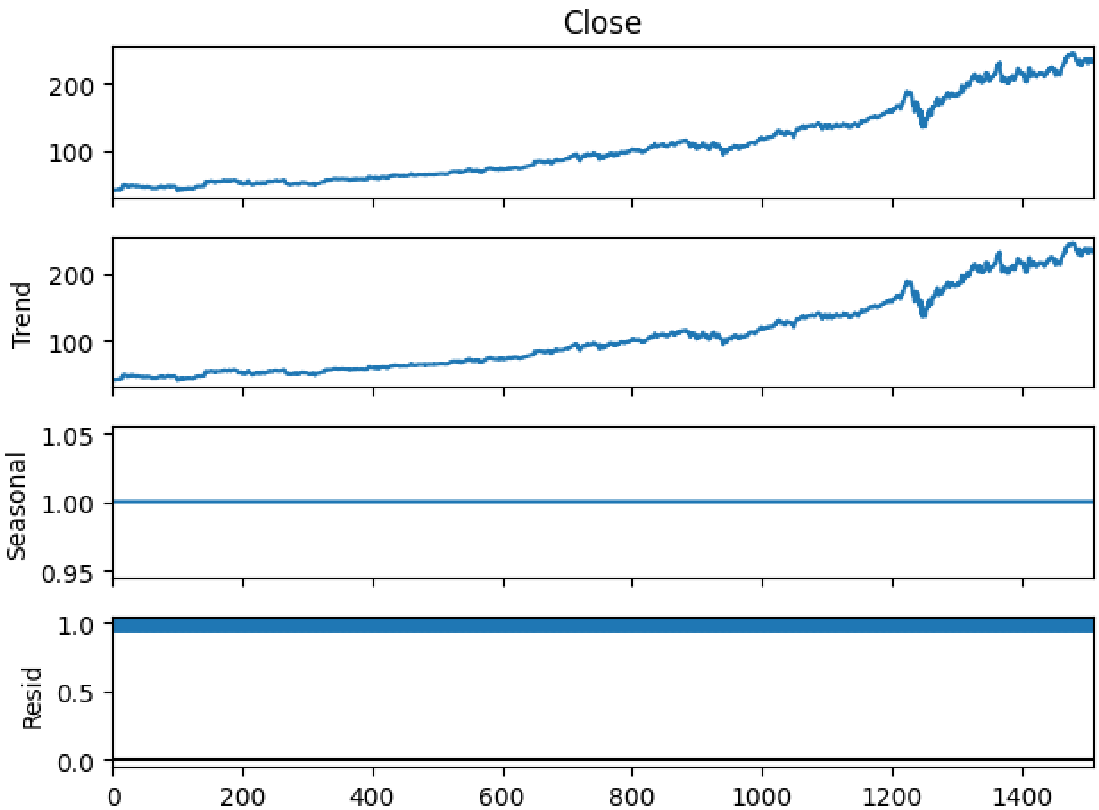
<Figure size 1000x500 with 0 Axes>

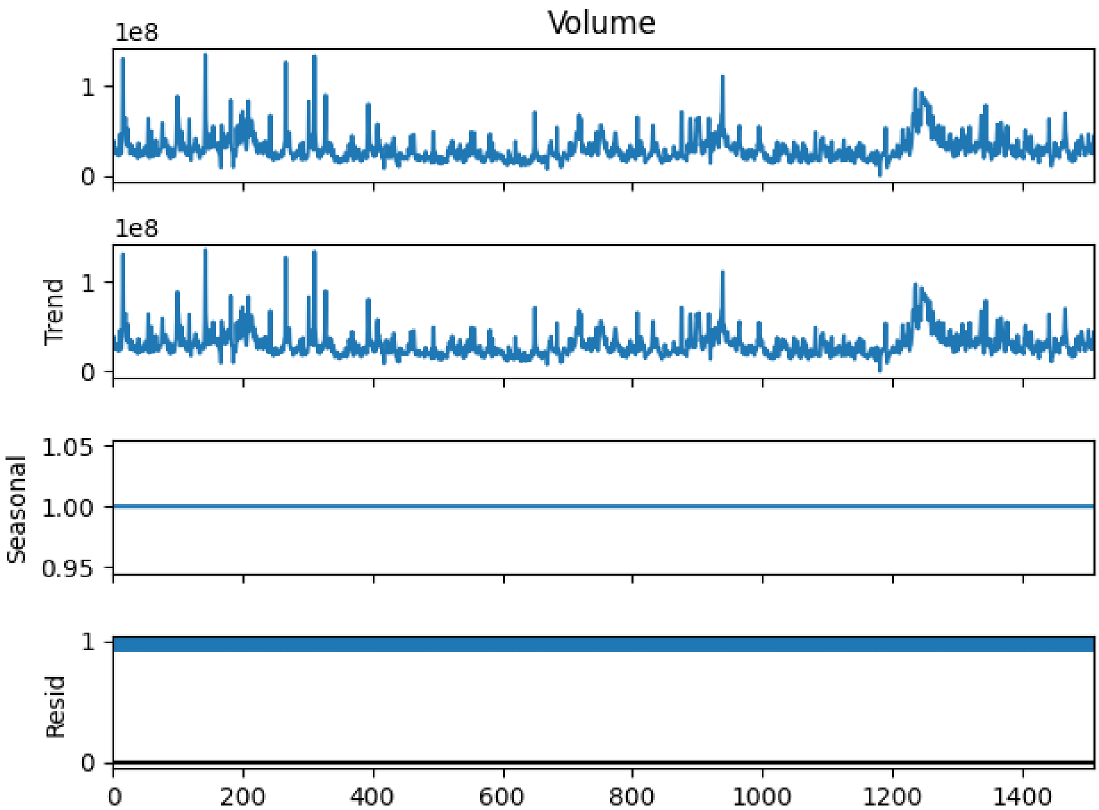



High
Trend
Seasonal
Resid
200
100
1.05
1.00
0.95
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400



Low
200
100
Trend
200
100
Seasonal
1.05
1.00
0.95
Resid
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400

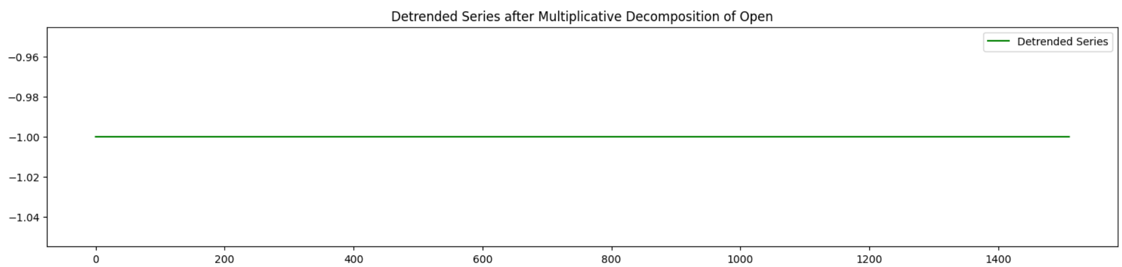

Close
200
100
Trend
200
100
Seasonal
1.05
1.00
0.95
Resid
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400

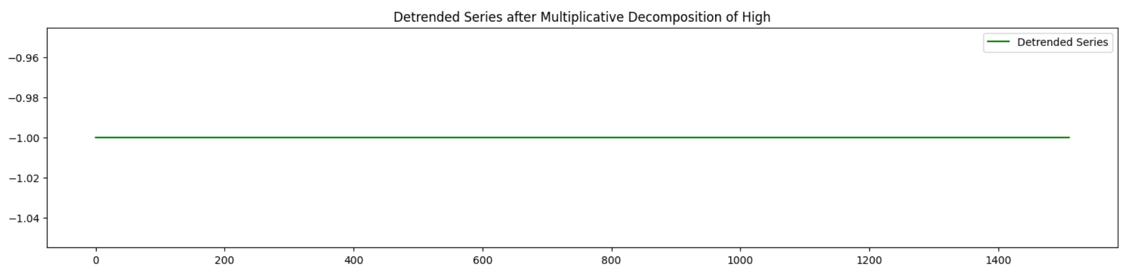


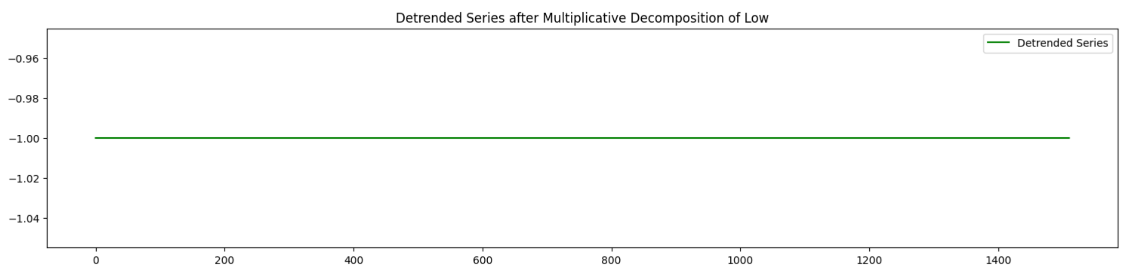
[20]:
```
# # Checking stationarity of the residuals of Multiplicative Decomposition␣
 ↪using ADF test

# for col in data.columns[1:]:
#     print(f'Checking stationarity of Residuals of {col} column of␣
 ↪multiplicative Decomposition')
#     decomposition_multiplicative =␣
 ↪seasonal_decompose(data[col],model='multiplicative',period=period)
#     residuals = decomposition_multiplicative.resid.dropna()
#     adf = adfuller(residuals)
#     print('ADF Statistic:', adf[0])
#     print('p-value:', adf[1])

#     if adf[1] < 0.05:
#         print("The residuals are stationary")
#     else:
#         print("The residuals are non-stationary")
#     print('-'*100)
```

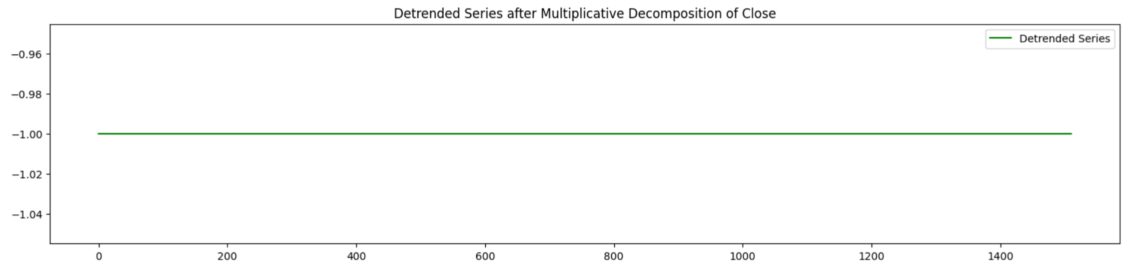
[21]:

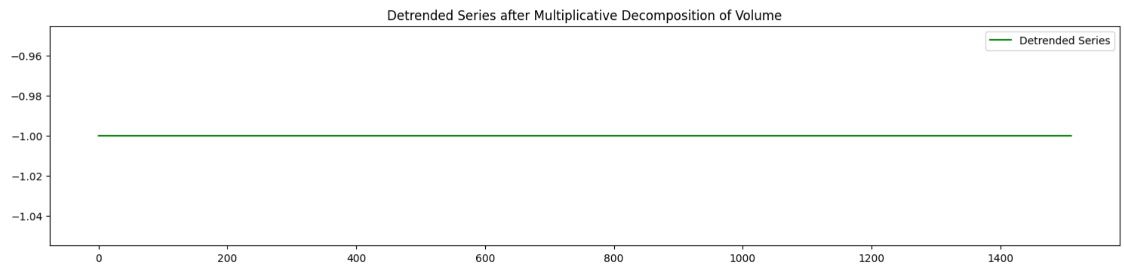
```python
# De-Trending series after removing trend and seasonality

for col in data.columns[1:]:
    # Perform additive decomposition
    decomposition_multiplicative = seasonal_decompose(data[col],␣
 ↪model='multiplicative', period=period)

    # Detrended series
    detrended_series = data[col] - decomposition_multiplicative.trend -␣
 ↪decomposition_multiplicative.seasonal

     # Visualization
    plt.figure(figsize=(15, 10))
    plt.subplot(3, 1, 3)
    plt.plot(detrended_series, label='Detrended Series', color='green')
    plt.title(f'Detrended Series after Multiplicative Decomposition of {col}')
    plt.legend()
    plt.tight_layout()
```











1.3 Time series forecasting:

Split the dataset into training and testing sets. Implement a forecasting model (e.g., ARIMA,SARIMA, Prophet) on the training set to forecast future stock prices.

Evaluate the performance of the forecasting model using appropriate evaluation metrics.

Visualize the actual vs. predicted stock prices on the testing set.

1.3.1 Splitting the data set into train and test

[22]:
```
train_data = data.iloc[: int(0.8*len(data)),:]
train_data
```

[22]:

	Date	Open	High	Low	Close	Volume
0	2015-04-01 16:00:00	40.60	40.76	40.31	40.72	36865322
1	2015-04-02 16:00:00	40.66	40.74	40.12	40.29	37487476
2	2015-04-06 16:00:00	40.34	41.78	40.18	41.55	39223692
3	2015-04-07 16:00:00	41.61	41.91	41.31	41.53	28809375
4	2015-04-08 16:00:00	41.48	41.69	41.04	41.42	24753438
…	…	…	…	…	…	…
1203	2020-01-10 16:00:00	162.82	163.22	161.18	161.34	20733946
1204	2020-01-13 16:00:00	161.76	163.31	161.26	163.28	21637007
1205	2020-01-14 16:00:00	163.39	163.60	161.72	162.13	23500783
1206	2020-01-15 16:00:00	162.62	163.94	162.57	163.18	21417871

```
1207 2020-01-16 16:00:00  164.35  166.24  164.03  166.17  23865360

[1208 rows x 6 columns]
```

[23]:
```python
test_data = data.iloc[int(0.8*len(data)): ,:]
test_data
```

[23]:
```
                     Date    Open    High     Low   Close    Volume
1208 2020-01-17 16:00:00  167.42  167.47  165.43  167.10  34371659
1209 2020-01-21 16:00:00  166.68  168.19  166.43  166.50  29517191
1210 2020-01-22 16:00:00  167.40  167.49  165.68  165.70  24138777
1211 2020-01-23 16:00:00  166.19  166.80  165.27  166.72  19680766
1212 2020-01-24 16:00:00  167.51  167.53  164.45  165.04  24918117
…                     …     …     …     …     …     …
1506 2021-03-25 16:00:00  235.30  236.94  231.57  232.34  34061853
1507 2021-03-26 16:00:00  231.55  236.71  231.55  236.48  25479853
1508 2021-03-29 16:00:00  236.59  236.80  231.88  235.24  25227455
1509 2021-03-30 16:00:00  233.53  233.85  231.10  231.85  24792012
1510 2021-03-31 16:00:00  232.91  239.10  232.39  235.77  43623471

[303 rows x 6 columns]
```

1.4 ——————————————————————-ARIMA———————————————————

1.4.1 Box-Cox transformation to stablize variance

[24]:
```python
# Applying Box-Cox transformation
lambda_values = {}

for col in train_data.columns[1:-1]:  # Non-stationary series
    train_data['BoxCox_' + col], lambda_value = boxcox(train_data[col])
    lambda_values[col] = lambda_value

print('Lambda values of different columns:', lambda_values)
train_data.head()
```

```
Lambda values of different columns: {'Open': -0.2860837812706928, 'High':
-0.29471552262107537, 'Low': -0.2853855602472242, 'Close': -0.2908854221227933}
```

[24]:
```
                  Date   Open   High    Low  Close    Volume  BoxCox_Open  \
0 2015-04-01 16:00:00  40.60  40.76  40.31  40.72  36865322     2.283951
1 2015-04-02 16:00:00  40.66  40.74  40.12  40.29  37487476     2.284463
2 2015-04-06 16:00:00  40.34  41.78  40.18  41.55  39223692     2.281722
3 2015-04-07 16:00:00  41.61  41.91  41.31  41.53  28809375     2.292438
4 2015-04-08 16:00:00  41.48  41.69  41.04  41.42  24753438     2.291361

   BoxCox_High  BoxCox_Low  BoxCox_Close
```

```
0      2.255381    2.283901      2.268258
1      2.255216    2.282255      2.264640
2      2.263638    2.282775      2.275102
3      2.264672    2.292404      2.274939
4      2.262920    2.290134      2.274042
```

1.4.2 Determine Stationarity (d parameter)

[25]:

```
for col in ['BoxCox_Open','BoxCox_High','BoxCox_Low','BoxCox_Close']:
    print(f'Checking stationarity of {col} column')
    train_data[col + '_first_diff'] = train_data[col].diff()
    adf = adfuller(train_data[col + '_first_diff'].dropna())
    print('ADF Statistic:', adf[0])
    print('p-value:', adf[1])

    if adf[1] < 0.05:
        print(f"The '{col + '_first_diff'}' column series is stationary")
        d = 1
        print(f"since series became stationary in first differencing, the value␣
↪of, d={d}")
    else:
        print(f"The '{col + '_first_diff'}' column series is non-stationary")
    print('-'*100)

train_data
```

```
Checking stationarity of BoxCox_Open column
ADF Statistic: -13.634787774349094
p-value: 1.6964799527612692e-25
The 'BoxCox_Open_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of BoxCox_High column
ADF Statistic: -24.887503686957956
p-value: 0.0
The 'BoxCox_High_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of BoxCox_Low column
ADF Statistic: -14.429639071357878
p-value: 7.66857407731591e-27
The 'BoxCox_Low_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
Checking stationarity of BoxCox_Close column
```

```
ADF Statistic: -14.321117210776181
p-value: 1.140835758236616e-26
The 'BoxCox_Close_first_diff' column series is stationary
since series became stationary in first differencing, the value of, d=1
--------------------------------------------------------------------------------
--------------------
```

[25]:

	Date	Open	High	Low	Close	Volume	\
0	2015-04-01 16:00:00	40.60	40.76	40.31	40.72	36865322	
1	2015-04-02 16:00:00	40.66	40.74	40.12	40.29	37487476	
2	2015-04-06 16:00:00	40.34	41.78	40.18	41.55	39223692	
3	2015-04-07 16:00:00	41.61	41.91	41.31	41.53	28809375	
4	2015-04-08 16:00:00	41.48	41.69	41.04	41.42	24753438	
…	…	…	…	…	…	…	
1203	2020-01-10 16:00:00	162.82	163.22	161.18	161.34	20733946	
1204	2020-01-13 16:00:00	161.76	163.31	161.26	163.28	21637007	
1205	2020-01-14 16:00:00	163.39	163.60	161.72	162.13	23500783	
1206	2020-01-15 16:00:00	162.62	163.94	162.57	163.18	21417871	
1207	2020-01-16 16:00:00	164.35	166.24	164.03	166.17	23865360	

	BoxCox_Open	BoxCox_High	BoxCox_Low	BoxCox_Close	\
0	2.283951	2.255381	2.283901	2.268258	
1	2.284463	2.255216	2.282255	2.264640	
2	2.281722	2.263638	2.282775	2.275102	
3	2.292438	2.264672	2.292404	2.274939	
4	2.291361	2.262920	2.290134	2.274042	
…	…	…	…	…	
1203	2.681201	2.637213	2.682483	2.654209	
1204	2.679678	2.637336	2.682599	2.656929	
1205	2.682015	2.637731	2.683267	2.655322	
1206	2.680915	2.638193	2.684494	2.656790	
1207	2.683377	2.641286	2.686582	2.660904	

	BoxCox_Open_first_diff	BoxCox_High_first_diff	BoxCox_Low_first_diff	\
0	NaN	NaN	NaN	
1	0.000512	-0.000165	-0.001646	
2	-0.002741	0.008422	0.000521	
3	0.010716	0.001034	0.009628	
4	-0.001077	-0.001752	-0.002270	
…	…	…	…	
1203	0.001408	0.001370	0.000218	
1204	-0.001523	0.000123	0.000116	
1205	0.002337	0.000395	0.000667	
1206	-0.001100	0.000462	0.001227	
1207	0.002462	0.003093	0.002088	

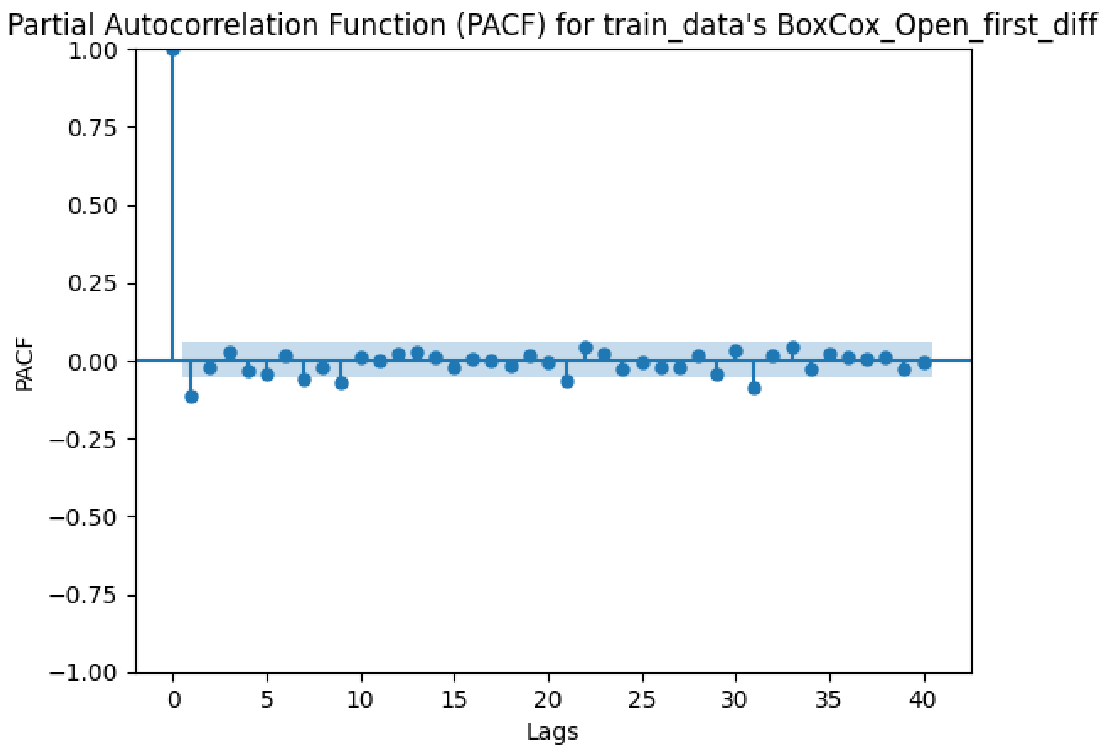
BoxCox_Close_first_diff

```
0                            NaN
1                      -0.003617
2                       0.010462
3                      -0.000163
4                      -0.000897
…                            …
1203                   -0.001056
1204                    0.002720
1205                   -0.001607
1206                    0.001468
1207                    0.004114

[1208 rows x 14 columns]
```

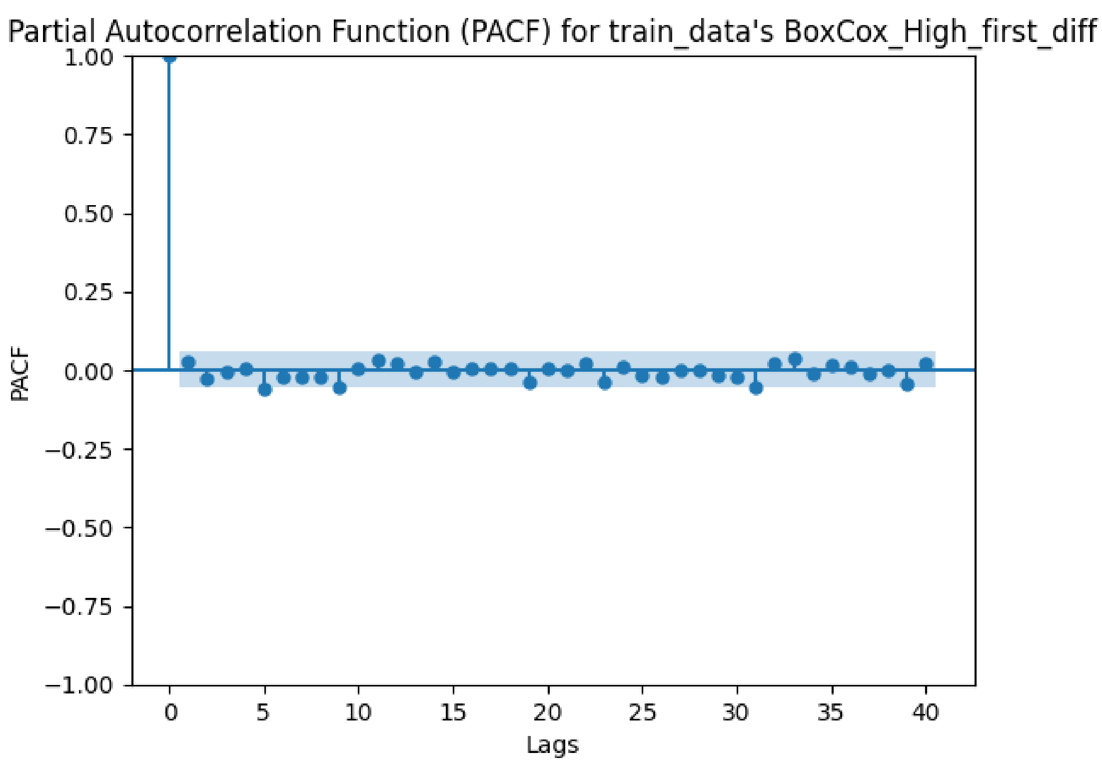
1.4.3 Determine parameter p

[26]:
```
for col in ['BoxCox_Open_first_diff', 'BoxCox_High_first_diff',␣
↪'BoxCox_Low_first_diff', 'BoxCox_Close_first_diff']:
    plt.figure(figsize=(10, 5))
    plot_pacf(train_data[col].dropna(), lags=40)
    plt.title(f"Partial Autocorrelation Function (PACF) for train_data's {col}")
    plt.xlabel('Lags')
    plt.ylabel('PACF')
    plt.show()
```

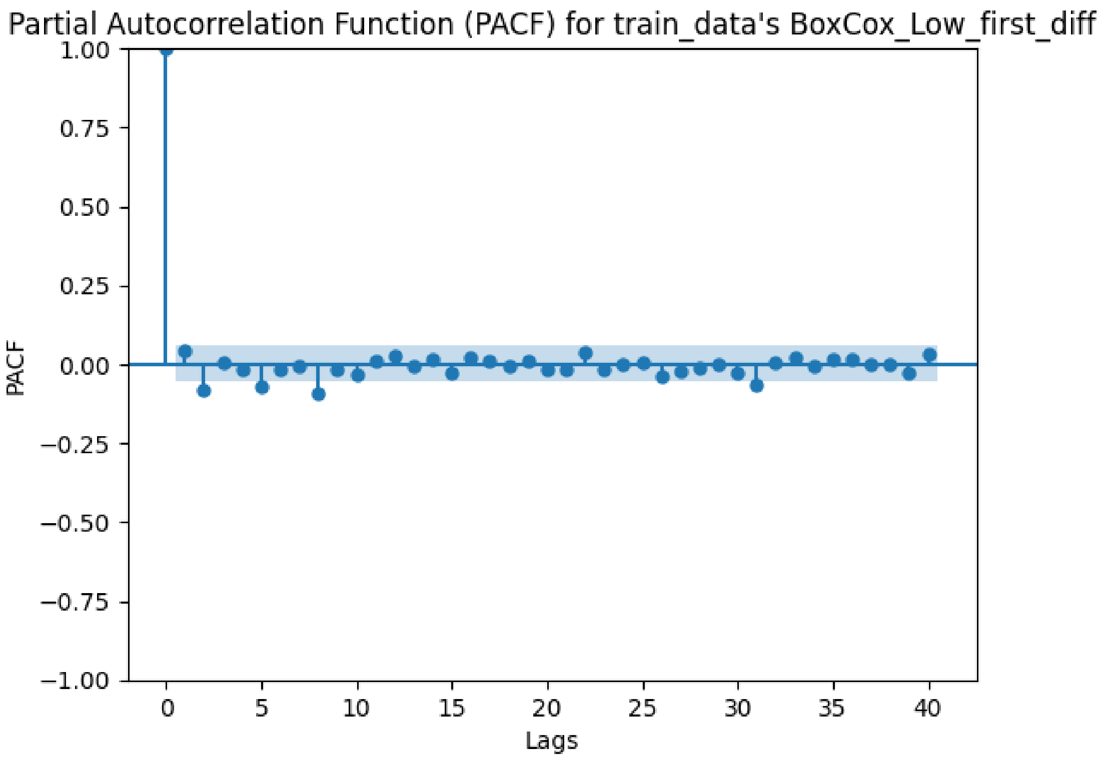
```
<Figure size 1000x500 with 0 Axes>
```



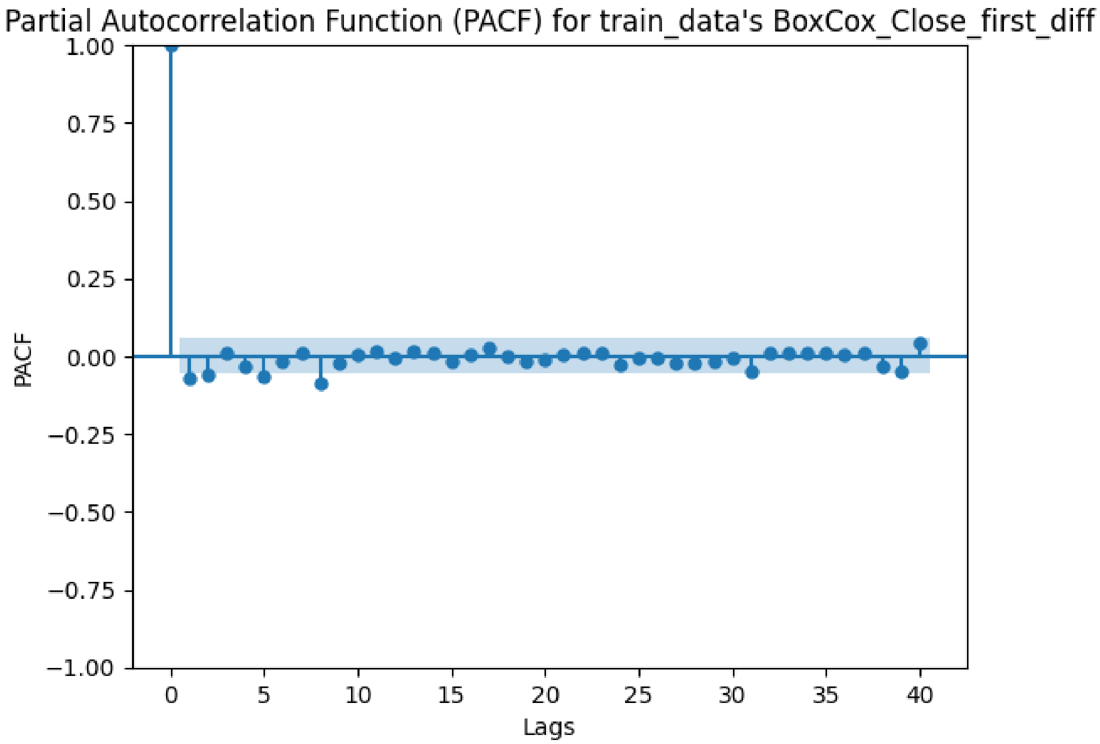
<Figure size 1000x500 with 0 Axes>



<Figure size 1000x500 with 0 Axes>



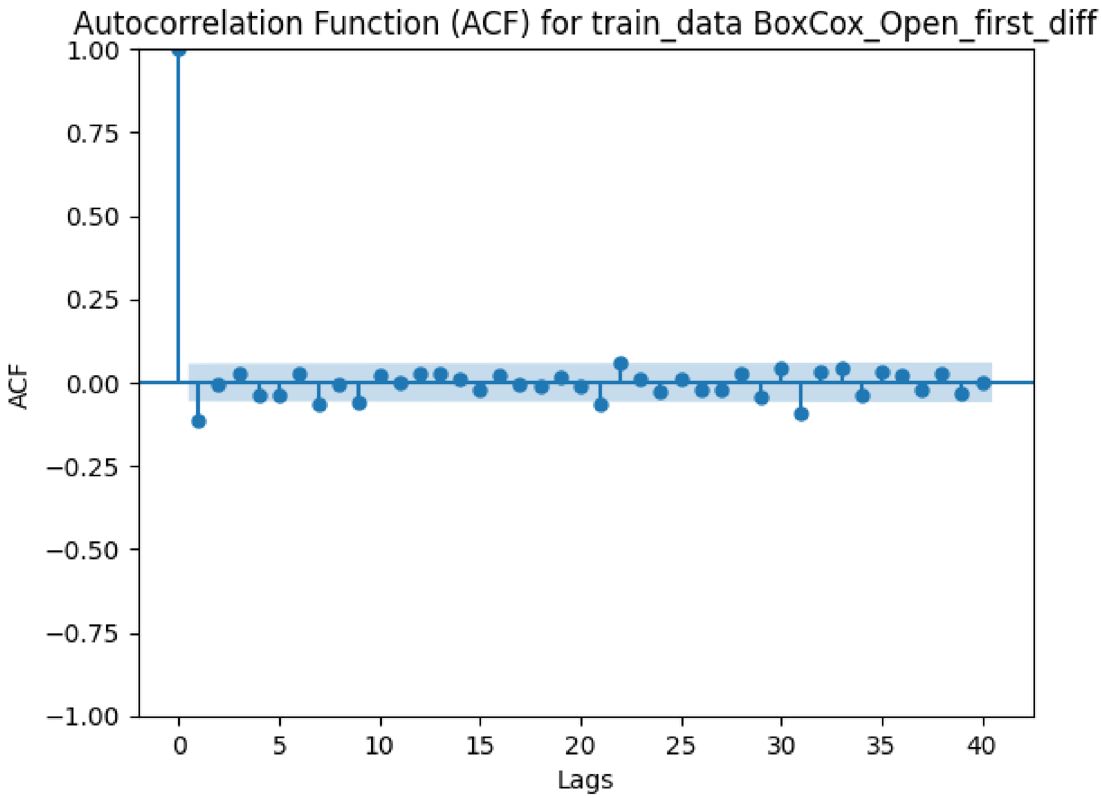
<Figure size 1000x500 with 0 Axes>



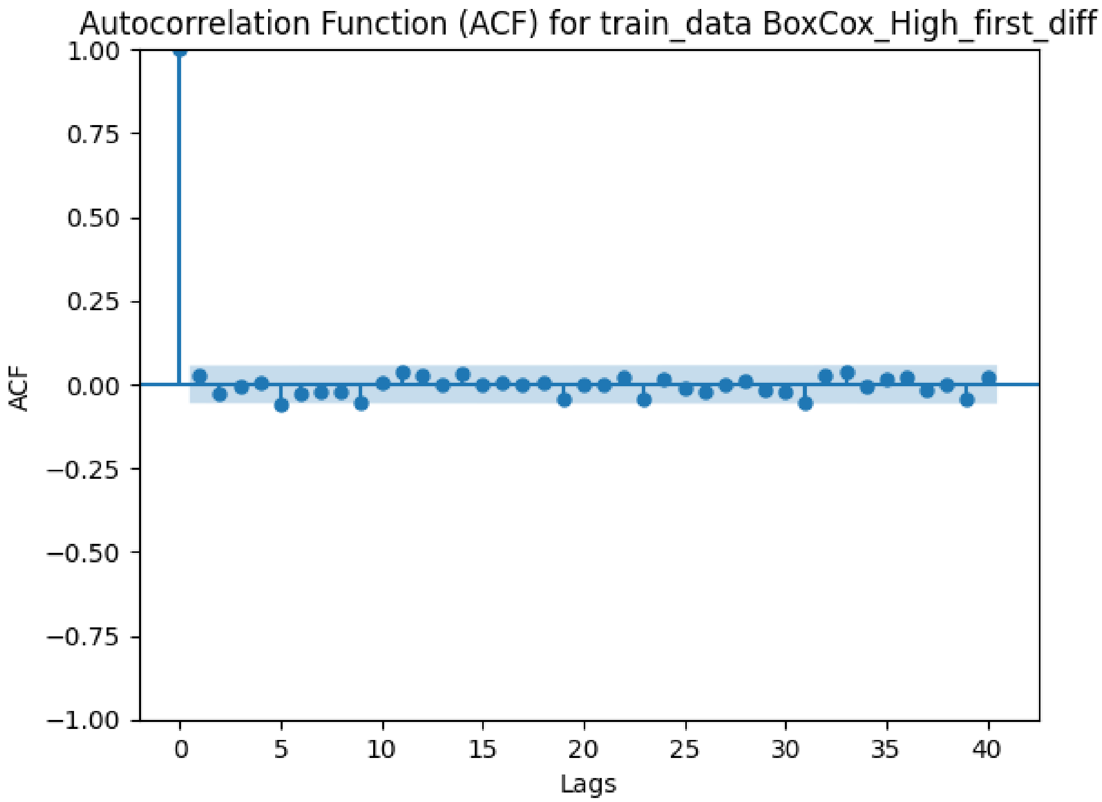
1.4.4 Determine parameter q

[27]:
```
for col in ['BoxCox_Open_first_diff', 'BoxCox_High_first_diff',␣
↪'BoxCox_Low_first_diff', 'BoxCox_Close_first_diff']:
    plt.figure(figsize=(10, 5))
    plot_acf(train_data[col].dropna(), lags=40)  # Plot ACF for the column
    plt.title(f'Autocorrelation Function (ACF) for train_data {col}')
    plt.xlabel('Lags')
    plt.ylabel('ACF')
```

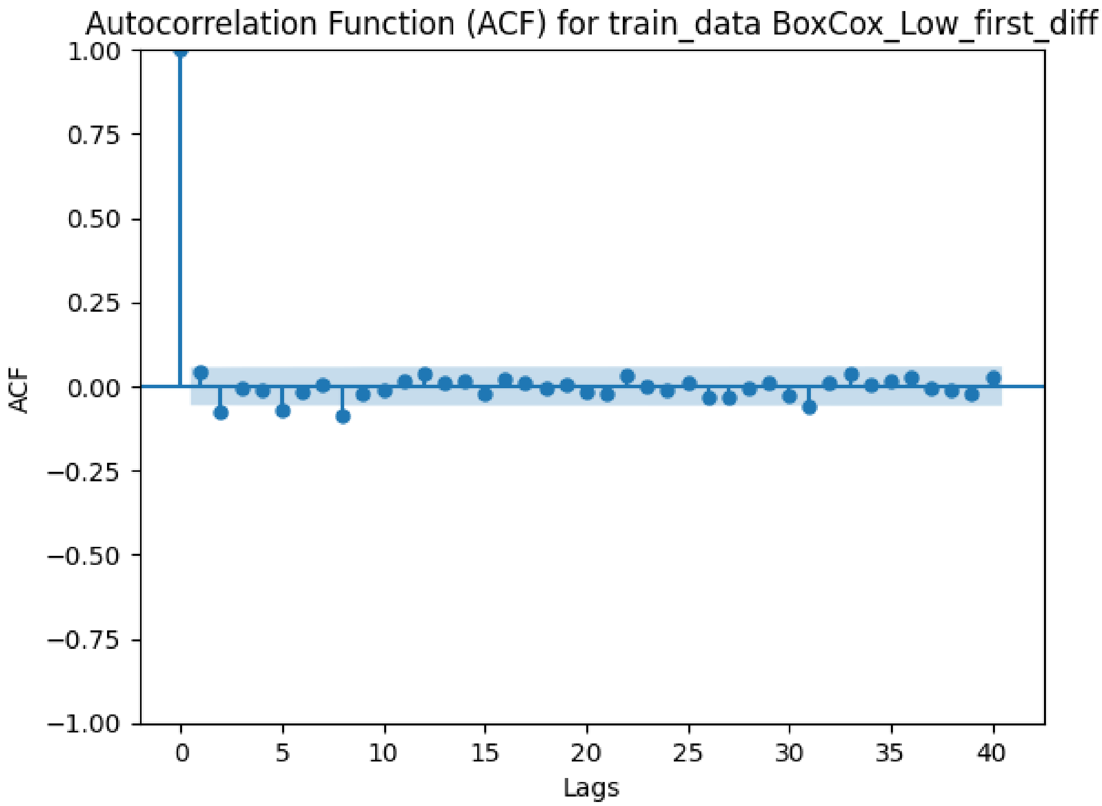
```
<Figure size 1000x500 with 0 Axes>
```



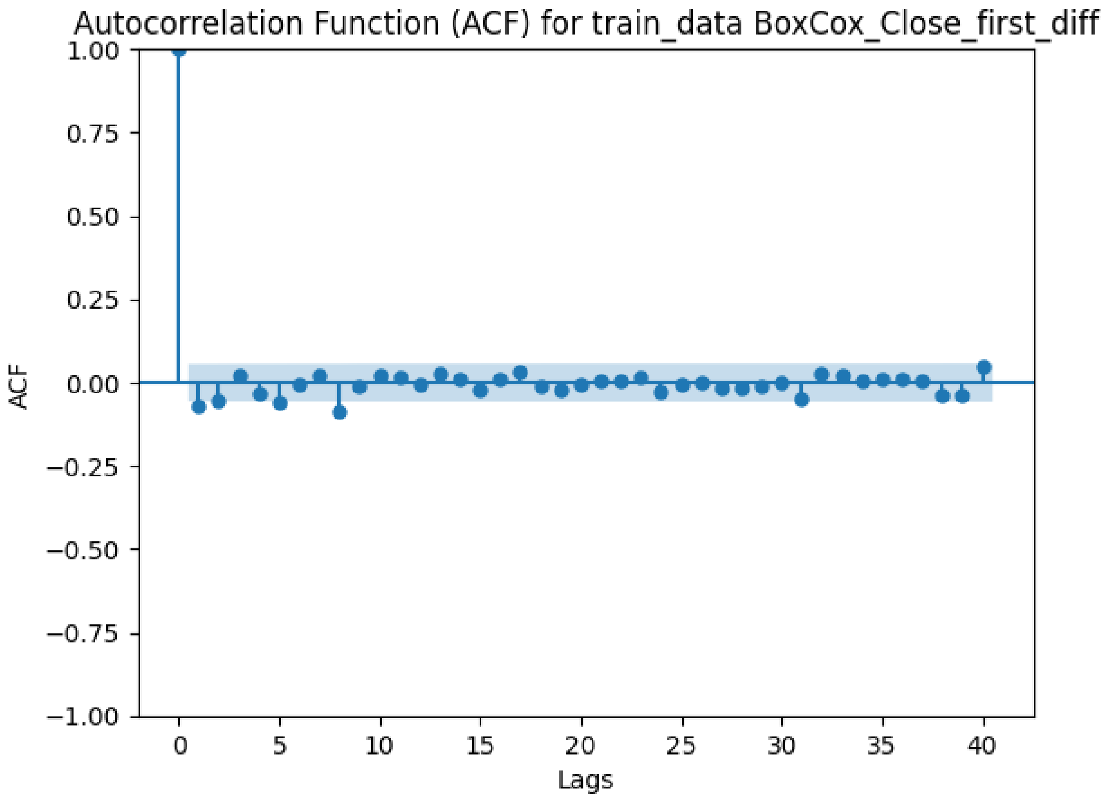

```
<Figure size 1000x500 with 0 Axes>
```



<Figure size 1000x500 with 0 Axes>




```
<Figure size 1000x500 with 0 Axes>
```



1.4.5 Using Auto-Arima to determine p,d,q automatically and doing forecasting - ARIMA

[28]:
```
def inverse_boxcox(y, lambda_value):
        if lambda_value == 0:
            return np.exp(y)  # If lambda is 0, use the exponential function
        else:
            return np.power((y * lambda_value + 1), 1 / lambda_value)


for col in ['BoxCox_Close']:
    print(f"Fitting ARIMA model for: {col}")
    model = auto_arima(train_data[col].dropna(), seasonal=False, stepwise=True,␣
 ↪trace=True)
    print(model.summary())
    print('+*'*75)
    print('\n\n')

    # Forecast future values
    n_periods = len(test_data) # Number of periods to forecast
```

```python
    forecast, conf_int = model.predict(n_periods=n_periods,␣
↪return_conf_int=True)
    forecast_original = inverse_boxcox(forecast, lambda_values['Close'])

# Evaluation of forecast
mae = mean_absolute_error(test_data['Close'], forecast_original)
mse = mean_squared_error(test_data['Close'], forecast_original)
rmse = np.sqrt(mse)
mape = np.mean(np.abs((test_data['Close'] - forecast_original) /␣
↪test_data['Close'])) * 100

# Print evaluation metrics
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Percentage Error (MAPE):", mape)
```

```
Fitting ARIMA model for: BoxCox_Close
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-9804.025, Time=0.87 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-9801.722, Time=0.20 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-9805.532, Time=0.15 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-9806.220, Time=0.24 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-9796.382, Time=0.10 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-9807.376, Time=0.86 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-9806.348, Time=0.42 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-9806.167, Time=0.37 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-9808.162, Time=0.46 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-9806.232, Time=0.98 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-9804.019, Time=1.10 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-9800.571, Time=0.18 sec

Best model:  ARIMA(0,1,2)(0,0,0)[0] intercept
Total fit time: 5.956 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1208
Model:               SARIMAX(0, 1, 2)   Log Likelihood                4908.081
Date:                Mon, 21 Oct 2024   AIC                          -9808.162
Time:                        12:20:24   BIC                          -9787.778
Sample:                             0   HQIC                         -9800.486
                               - 1208
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0003      0.000      2.943      0.003       0.000       0.001
```

```
ma.L1          -0.0722      0.020     -3.645      0.000      -0.111      -0.033
ma.L2          -0.0565      0.022     -2.588      0.010      -0.099      -0.014
sigma2       1.717e-05   3.13e-07     54.830      0.000    1.66e-05    1.78e-05
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
3539.38
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               0.50   Skew:
0.46
Prob(H) (two-sided):                  0.00   Kurtosis:
11.34
================================================================================
===

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



Mean Absolute Error (MAE): 11.690434695339892
Mean Squared Error (MSE): 207.9251387968375
Root Mean Squared Error (RMSE): 14.419609523036243
Mean Absolute Percentage Error (MAPE): 5.953771532880906
```

1.4.6 Using Auto-Arima to determine p,d,q and P,D,Q & m automatically and doing forecasting - SARIMA

[29]:
```
def inverse_boxcox(y, lambda_value):
    if lambda_value == 0:
        return np.exp(y)
    else:
        return np.power((y * lambda_value + 1), 1 / lambda_value)


for col in ['BoxCox_Close']:
    print(f"Fitting seasonal ARIMA model for: {col}")

    model = auto_arima(train_data[col].
 ↪dropna(),seasonal=True,m=1,stepwise=True,trace=True)

    print(model.summary())
    print('+*'*100)
```

```python
    print('\n\n')

    # Forecasting
    n_periods = len(test_data)  # Number of periods to forecast
    forecast, conf_int = model.predict(n_periods=n_periods,␣
↪return_conf_int=True)

    # Inverting Box-Cox transformation to get the original scale
    forecast_original = inverse_boxcox(forecast, lambda_values['Close'])

    # Evaluation of forecast
    mae = mean_absolute_error(test_data['Close'], forecast_original)
    mse = mean_squared_error(test_data['Close'], forecast_original)
    rmse = np.sqrt(mse)
    mape = np.mean(np.abs((test_data['Close'] - forecast_original) /␣
↪test_data['Close'])) * 100

    # Evaluation metrics
    print("Mean Absolute Error (MAE):", mae)
    print("Mean Squared Error (MSE):", mse)
    print("Root Mean Squared Error (RMSE):", rmse)
    print("Mean Absolute Percentage Error (MAPE):", mape)
```

```
Fitting seasonal ARIMA model for: BoxCox_Close
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-9804.025, Time=0.62 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-9801.722, Time=0.20 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-9805.532, Time=0.14 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-9806.220, Time=0.24 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-9796.382, Time=0.09 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-9807.376, Time=0.86 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-9806.348, Time=0.43 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-9806.167, Time=0.50 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-9808.162, Time=0.56 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-9806.232, Time=0.98 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-9804.019, Time=1.12 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-9800.571, Time=0.17 sec

Best model:  ARIMA(0,1,2)(0,0,0)[0] intercept
Total fit time: 5.913 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1208
Model:               SARIMAX(0, 1, 2)   Log Likelihood                4908.081
Date:                Mon, 21 Oct 2024   AIC                          -9808.162
Time:                        12:20:30   BIC                          -9787.778
Sample:                             0   HQIC                         -9800.486
```

```
                                  - 1208
Covariance Type:                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0003      0.000      2.943      0.003       0.000       0.001
ma.L1         -0.0722      0.020     -3.645      0.000      -0.111      -0.033
ma.L2         -0.0565      0.022     -2.588      0.010      -0.099      -0.014
sigma2      1.717e-05   3.13e-07     54.830      0.000    1.66e-05    1.78e-05
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
3539.38
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               0.50   Skew:
0.46
Prob(H) (two-sided):                  0.00   Kurtosis:
11.34
================================================================================
===

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



Mean Absolute Error (MAE): 11.690434695339892
Mean Squared Error (MSE): 207.9251387968375
Root Mean Squared Error (RMSE): 14.419609523036243
Mean Absolute Percentage Error (MAPE): 5.953771532880906
```

1.5 PROPHET Forecasting

[30]:
```python
# Prepare the training data for Prophet
df_prophet_train = train_data[['Date', 'Close']].rename(columns={'Date': 'ds',␣
↪'Close': 'y'})

# Fit the model
model = Prophet()
model.fit(df_prophet_train)

# Create future DataFrame for predictions
```

```
future = model.make_future_dataframe(periods=len(test_data) + 365)

# Make predictions
forecast = model.predict(future)

# Ensure 'Date' in test_data is in datetime format
test_data['Date'] = pd.to_datetime(test_data['Date'])  # Convert to datetime if␣
↪not already

# Normalize dates to only use the date part (removing time)
test_data['Date'] = test_data['Date'].dt.date
forecast['ds'] = forecast['ds'].dt.date  # This line must come after forecast␣
↪is created

# Filter the forecast to only include the test dates
forecast_test = forecast[forecast['ds'].isin(test_data['Date'])]

# Print lengths for comparison
print("Length of test_data:", len(test_data))
print("Length of forecast_test:", len(forecast_test))
```

```
12:20:31 - cmdstanpy - INFO - Chain [1] start processing
12:20:32 - cmdstanpy - INFO - Chain [1] done processing
```

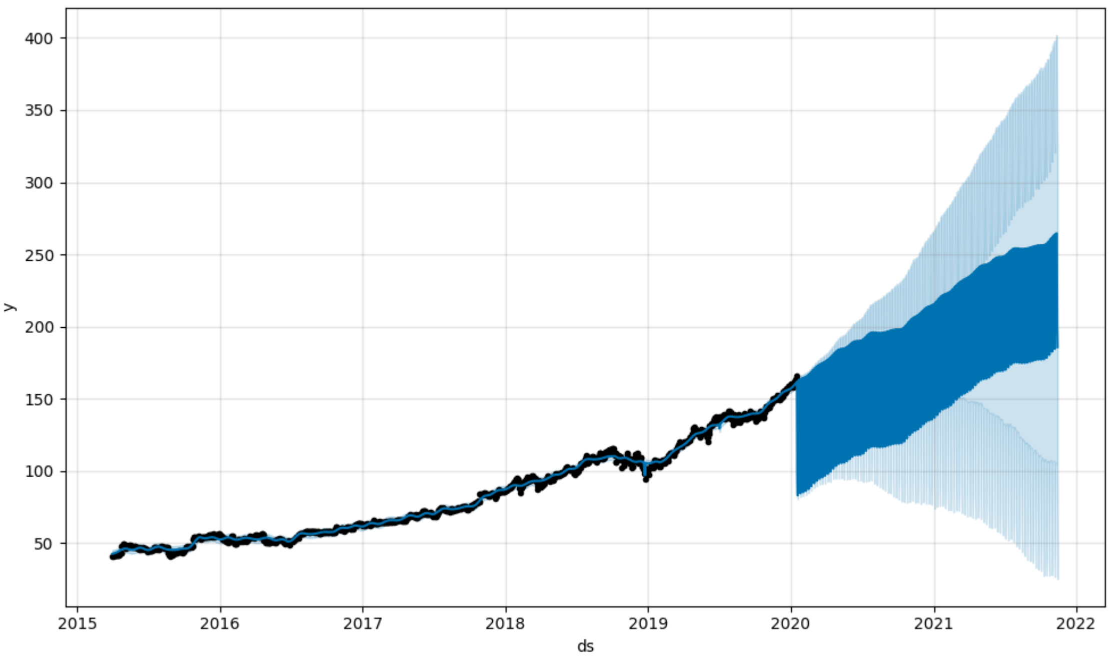
```
Length of test_data: 303
Length of forecast_test: 303
```

[31]:

```
# Filtering the forecast to only include the test dates
forecast_test = forecast[forecast['ds'].isin(test_data['Date'])]

# Reset index
forecast_test.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)

# evaluation
mae = mean_absolute_error(test_data['Close'], forecast_test['yhat'])
mse = mean_squared_error(test_data['Close'], forecast_test['yhat'])
rmse = np.sqrt(mse)
mape = np.mean(np.abs((test_data['Close'] - forecast_test['yhat']) /␣
↪test_data['Close'])) * 100

# Print evaluation metrics
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Percentage Error (MAPE):", mape)

# 'ds' in forecast in datetime format
```

```python
forecast['ds'] = pd.to_datetime(forecast['ds'])

# Plot the forecast
fig1 = model.plot(forecast)
```

```
Mean Absolute Error (MAE): 10.356701403596318
Mean Squared Error (MSE): 174.0310234345922
Root Mean Squared Error (RMSE): 13.19208184611482
Mean Absolute Percentage Error (MAPE): 5.324451893479434
```



1.5.1 Task

Further analysis:

Implement additional time series models or techniques to improve the forecasting performance, and Compare the performance of different models and techniques.

Discuss the strengths and weaknesses of each model or technique in the context of the Microsoft Stocks dataset

Performance Summary:

```
Prophet outperformed ARIMA and seasonal ARIMA models across all metrics:
MAE (10.36) is lower compared to ARIMA (11.69), indicating more accurate predictions on averag
MSE and RMSE are also lower, showing that Prophet has less variance in its errors.
MAPE (5.32%) is better than the ARIMA models (5.95%), meaning that Prophet had more accurate pe
```

ARIMA (AutoRegressive Integrated Moving Average)

```
Strengths:
    Good for short-term forecasting
    Handles non-stationary data
    Can tune the AR, MA, and differencing terms to fit the data well.
    Widely understood and used
Weaknesses:
    ARIMA assumes a linear structure in the data and may not capture more complex, nonlinear pa
    it's less flexible than other models like Prophet for handling seasonal components.
    Not suitable for long-term forecasting.
```

PROPHET

```
Strengths:
    Handles seasonality well
    Robust to missing data
    The model is designed to be robust to outliers
    Prophet has a user-friendly API and does not require extensive

Weaknesses:
    Less accurate for short-term predictions
    Prophet assumes that trends and seasonal components are additive
    it offers less flexibility for fine-tuning
```

SARIMA (Seasonal ARIMA)

```
Strengths:
    Handles seasonality
    SARIMA allows you to fine-tune parameters to capture both trends and seasonal components.
    Works for stationary and non-stationary data

Weaknesses:
    Finding the right parameters for SARIMA (p, d, q, P, D, Q) can be challenging and time-cons
    Like ARIMA, SARIMA assumes a linear relationship between variables and struggles with captu
```

1.5.2 Explaining the approach

```
> Importing the neccessary libraries
> Explore the dataset, Summary statistic of the dataset
> Check for missing values and handle them appropriately,Visualize the stock prices over time u
> Check if the time series is stationary or non-stationary using ADF test and KPSS.
> Determining time period for seasonality using fourier transformation
> Determing the presence of seasonility by differencing the time series and plotting ACF and PA
> Perform time series decomposition to extract the trend, seasonality, and residual components
> Doing Additive and Multiplicative Decomposition to analyse trend, seasonality and residuals.
> De-Trending series after removing trend and seasonality(if present)
> Splitting the data set into train and test 80 % train data and 20 % test data
> Applying ARIMA : Box-Cox transformation of the time series to stablize variance,Determine Sta
```

```
> Using Auto-Arima to determine p,d,q automatically and doing forecasting - ARIMA
> Using Auto-Arima to determine p,d,q and P,D,Q & m automatically and doing forecasting - SARI
> Forecasting using PROPHET and plotting the forecast
> Compare the performance of ARIMA/SARIMA and PROPHET. The performance of prophet is slightly
```