s048-s046-s036-s028-atsa-project

October 22, 2023

[27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

[28]:
```
df = pd.read_csv('TATAE.csv', parse_dates=['Date'], index_col='Date')
```

[29]:
```
df
```

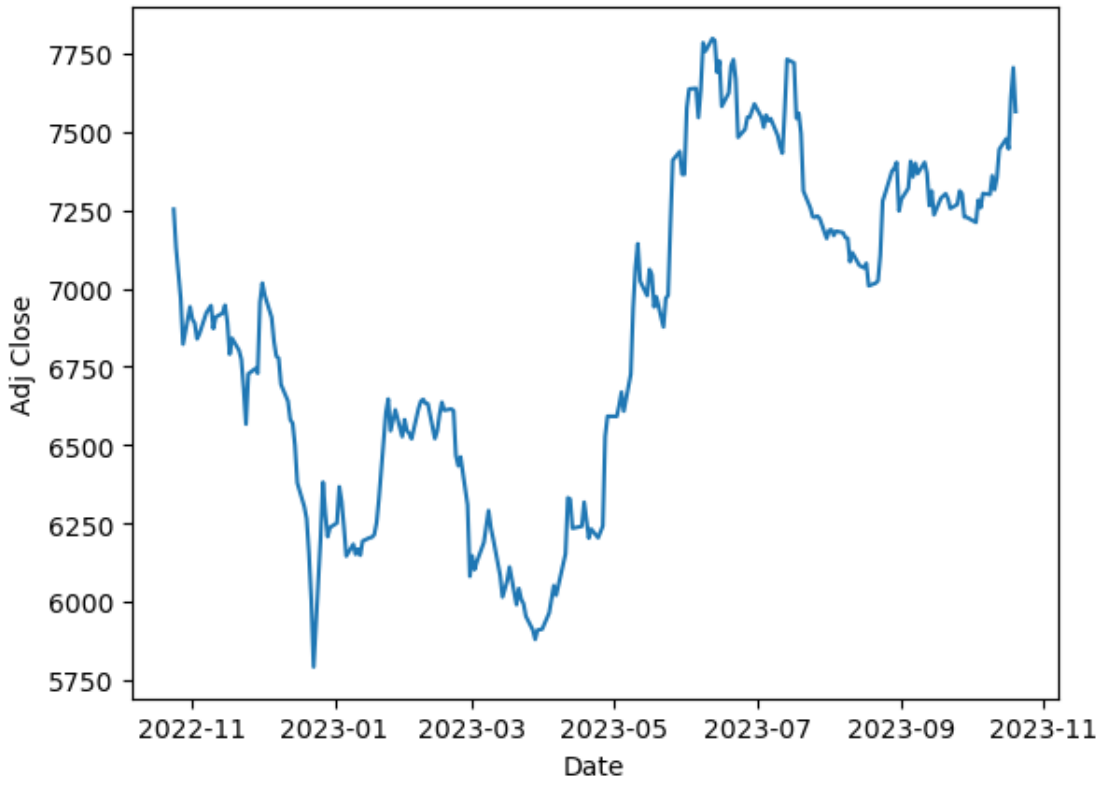
[29]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-10-24	7360.000000	7365.000000	7272.000000	7309.299805	7252.444824	66855
2022-10-25	7344.950195	7344.950195	7175.250000	7186.399902	7130.500977	222853
2022-10-27	7219.850098	7248.000000	7000.000000	7023.100098	6968.471191	360043
2022-10-28	7021.399902	7025.299805	6840.000000	6875.600098	6822.118652	459903
2022-10-31	6910.000000	7091.950195	6900.000000	6995.850098	6941.433105	378242
…	…	…	…	…	…	…
2023-10-16	7490.000000	7490.000000	7385.000000	7477.049805	7477.049805	
2023-10-17	7496.950195	7510.000000	7400.000000	7445.350098	7445.350098	
2023-10-18	7485.000000	7694.399902	7480.500000	7608.899902	7608.899902	
2023-10-19	7608.899902	7758.899902	7569.950195	7704.049805	7704.049805	
2023-10-20	7736.600098	7738.000000	7501.000000	7564.399902	7564.399902	

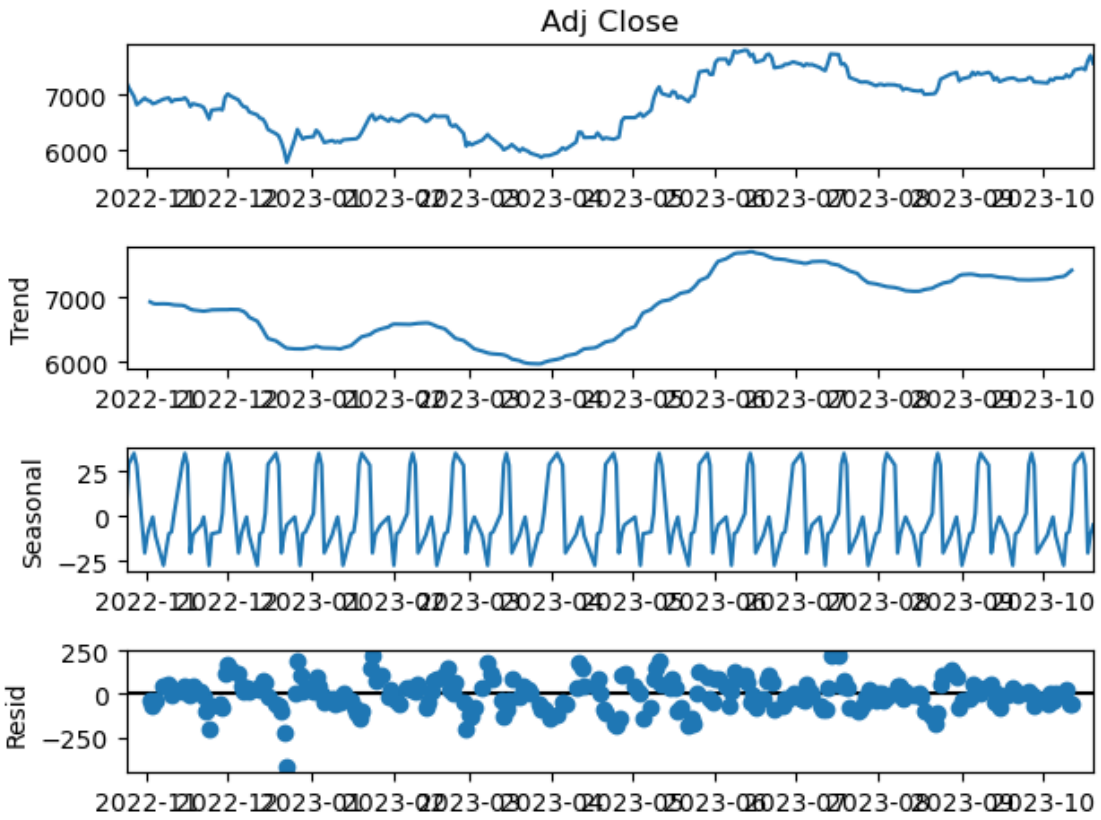
```
…              …
2023-10-16  100722
2023-10-17  112435
2023-10-18  497848
2023-10-19  282308
2023-10-20  119490

[247 rows x 6 columns]
```

[30]:
```
sns.lineplot(data=df,x=df.index,y=df['Adj Close'])
```

[30]:
```
<Axes: xlabel='Date', ylabel='Adj Close'>
```



[31]:
```
import statsmodels.api as sm
df1 = sm.tsa.seasonal_decompose(df['Adj Close'], model='additive', period=12)
df1.plot();
```



[32]:
```
#performing the rolling staistics test
df['rollMean'] = df['Adj Close'].rolling(window=12).mean()#rolling on 12 months
df['rollStd'] = df['Adj Close'].rolling(window=12).std()
df.head(13)
```

[32]:

Date	Open	High	Low	Close	Adj Close	\
2022-10-24	7360.000000	7365.000000	7272.000000	7309.299805	7252.444824	
2022-10-25	7344.950195	7344.950195	7175.250000	7186.399902	7130.500977	
2022-10-27	7219.850098	7248.000000	7000.000000	7023.100098	6968.471191	
2022-10-28	7021.399902	7025.299805	6840.000000	6875.600098	6822.118652	
2022-10-31	6910.000000	7091.950195	6900.000000	6995.850098	6941.433105	
2022-11-01	7060.000000	7060.000000	6926.200195	6955.799805	6901.694336	
2022-11-02	6999.950195	7048.000000	6925.000000	6943.200195	6889.192871	
2022-11-03	6943.000000	6943.000000	6880.149902	6892.649902	6839.035645	
2022-11-04	6920.000000	6941.000000	6852.250000	6908.049805	6854.315918	
2022-11-07	6972.000000	7040.000000	6951.299805	6976.450195	6922.184082	
2022-11-09	7025.000000	7045.000000	6977.000000	6999.500000	6945.054688	
2022-11-10	6990.000000	6990.250000	6893.000000	6925.700195	6871.829102	
2022-11-11	7095.500000	7095.700195	6941.000000	6962.500000	6908.342285	

```
             Volume     rollMean     rollStd
Date
2022-10-24   66855           NaN         NaN
2022-10-25  222853           NaN         NaN
2022-10-27  360043           NaN         NaN
2022-10-28  459903           NaN         NaN
2022-10-31  378242           NaN         NaN
2022-11-01  237699           NaN         NaN
2022-11-02  212622           NaN         NaN
2022-11-03  189007           NaN         NaN
2022-11-04  197189           NaN         NaN
2022-11-07  142033           NaN         NaN
2022-11-09  228537           NaN         NaN
2022-11-10  104329  6944.856283  126.099723
2022-11-11  287465  6916.181071   80.773471
```

[33]:
```
from statsmodels.tsa.stattools import adfuller as adf
adfTest = adf(df['Adj Close'],autolag='AIC')
adfTest
```

[33]:
```
(-0.985044098961716,
 0.7586799638911629,
 2,
 244,
 {'1%': -3.457437824930831,
  '5%': -2.873459364726563,
  '10%': -2.573122099570008},
 2702.423578492418)
```

[34]:
```
#function to check stationerity
def test_stationery(daf,var):
    daf['rollMean'] = daf[var].rolling(window=12).mean()#rolling on 12 months
    daf['rollStd'] = daf[var].rolling(window=12).std()

    from statsmodels.tsa.stattools import adfuller as adf
    adfTest = adf(daf[var],autolag='AIC')
    stats = pd.Series(adfTest[0:4], index = ['teststatistics',␣
 ↪'pvalue','lags','no. of obsrvation'])
    print(stats)

    for key, values in adfTest[4].items():   #our data is not stationery
        print('criticality',key,":",values)

    sns.lineplot(data=daf,x=daf.index,y=var)
    sns.lineplot(data=daf,x=daf.index,y=daf.rollMean)
    sns.lineplot(data=daf,x=daf.index,y=daf.rollStd)
```

[35]:
```
new_data = df[['Adj Close']]
new_data.head()
```

[35]:

Date	Adj Close
2022-10-24	7252.444824
2022-10-25	7130.500977
2022-10-27	6968.471191
2022-10-28	6822.118652
2022-10-31	6941.433105

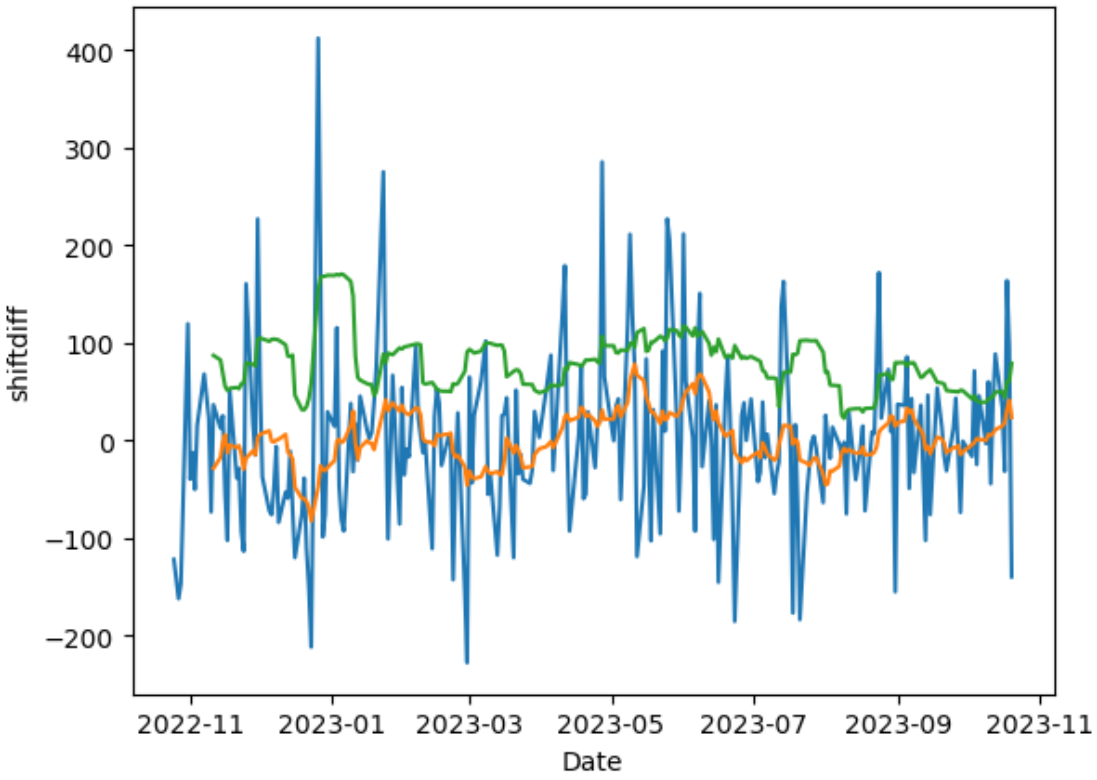
[36]:
```
new_data['shift'] = new_data['Adj Close'].shift()
new_data['shift'] = new_data['Adj Close'].shift()
new_data['shiftdiff'] = new_data['Adj Close']-new_data['shift']
new_data.head()
```

[36]:

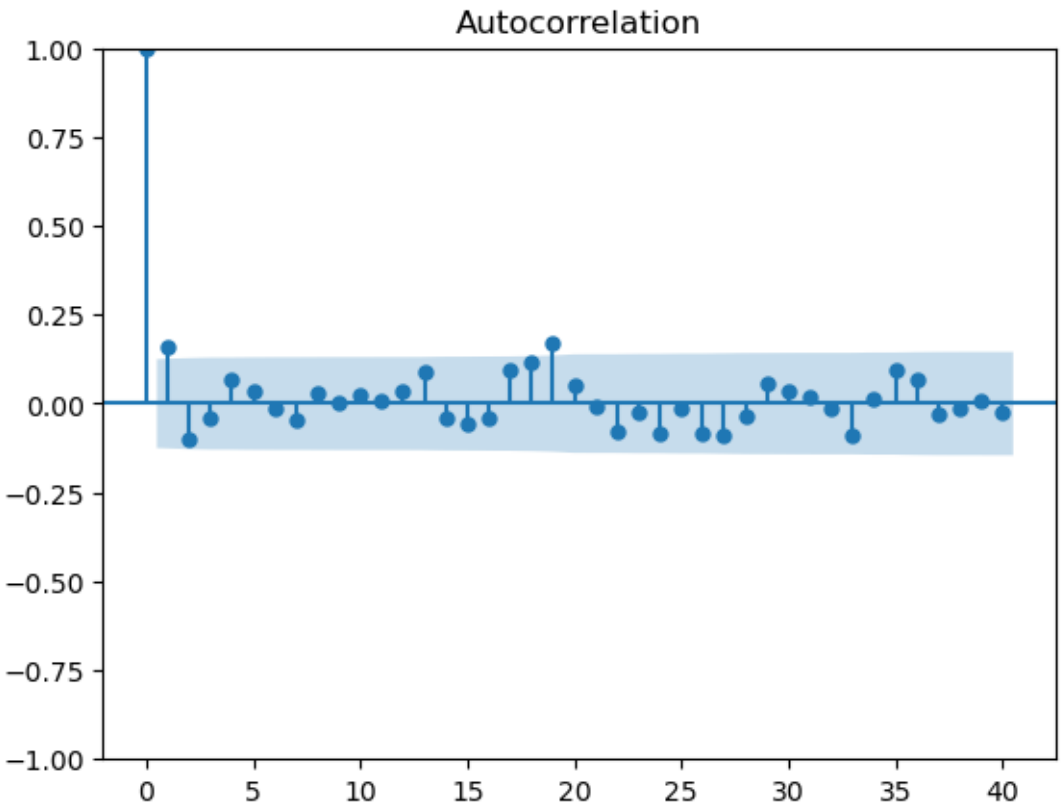
Date	Adj Close	shift	shiftdiff
2022-10-24	7252.444824	NaN	NaN
2022-10-25	7130.500977	7252.444824	-121.943847
2022-10-27	6968.471191	7130.500977	-162.029786
2022-10-28	6822.118652	6968.471191	-146.352539
2022-10-31	6941.433105	6822.118652	119.314453

[37]:
```
test_stationery(new_data.dropna(),'shiftdiff')
```

```
teststatistics      -1.165547e+01
pvalue               1.990377e-21
lags                 1.000000e+00
no. of obsrvation    2.440000e+02
dtype: float64
criticality 1% : -3.457437824930831
criticality 5% : -2.873459364726563
criticality 10% : -2.573122099570008
```



[38]:
```
plot_acf(new_data['shiftdiff'].dropna(),lags=40);
```



[39]:
```
df1 = new_data[['shiftdiff','Adj Close']].copy(deep=True)
```

[40]:
```
from sklearn.model_selection import train_test_split

train,test = train_test_split(
    df1, test_size=0.10, shuffle=False
)
```

[41]:
```
train
```

[41]:

Date	shiftdiff	Adj Close
2022-10-24	NaN	7252.444824
2022-10-25	-121.943847	7130.500977
2022-10-27	-162.029786	6968.471191
2022-10-28	-146.352539	6822.118652
2022-10-31	119.314453	6941.433105
…	…	…
2023-09-07	42.849610	7399.049805
2023-09-08	-32.849610	7366.200195
2023-09-11	35.949707	7402.149902

```
2023-09-12  -35.299804  7366.850098
2023-09-13 -102.450196  7264.399902

[222 rows x 2 columns]
```

[42]:
```
test.index
```

[42]:
```
DatetimeIndex(['2023-09-14', '2023-09-15', '2023-09-18', '2023-09-20',
               '2023-09-21', '2023-09-22', '2023-09-25', '2023-09-26',
               '2023-09-27', '2023-09-28', '2023-09-29', '2023-10-03',
               '2023-10-04', '2023-10-05', '2023-10-06', '2023-10-09',
               '2023-10-10', '2023-10-11', '2023-10-12', '2023-10-13',
               '2023-10-16', '2023-10-17', '2023-10-18', '2023-10-19',
               '2023-10-20'],
              dtype='datetime64[ns]', name='Date', freq=None)
```

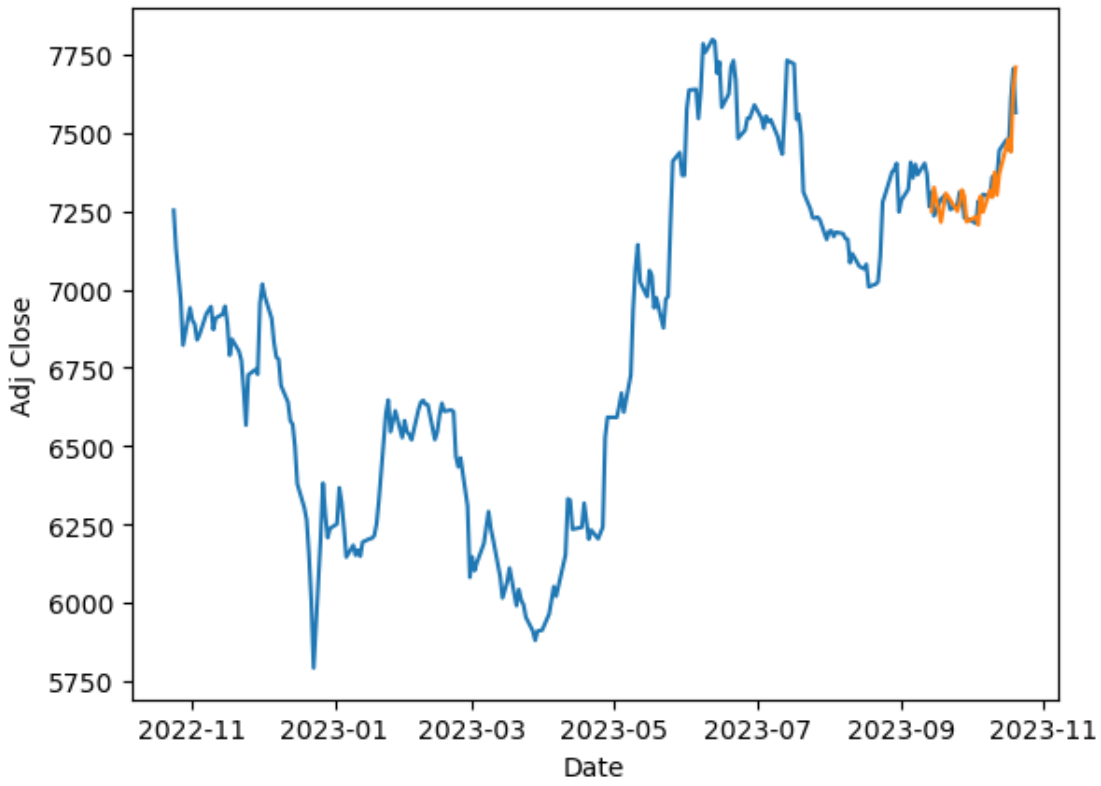
[43]:
```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Fitting ARIMA(1,1,1) model
arima_model = ARIMA(df['Adj Close'], order=(1,1,1))
arima_result = arima_model.fit()

# Displaying the model summary
arima_result.summary()

# Make predictions
start_date = test.index[0]
end_date = test.index[-1]
prediction = arima_result.predict(start_date,end_date)  # Use typ='levels' to␣
 ↪get actual values
```

[44]:
```
df1['arimaPred'] = prediction
df1.tail()
```

[44]:
```
             shiftdiff    Adj Close    arimaPred
Date
2023-10-16   33.299805  7477.049805  7455.168076
2023-10-17  -31.699707  7445.350098  7478.752003
2023-10-18  163.549804  7608.899902  7438.680898
2023-10-19   95.149903  7704.049805  7642.460023
2023-10-20 -139.649903  7564.399902  7708.538345
```

[45]:
```
sns.lineplot(data=df1, x=df1.index, y='Adj Close')
sns.lineplot(data=df1, x=df1.index, y='arimaPred')
```

[45]:
```
<Axes: xlabel='Date', ylabel='Adj Close'>
```



[46]:
```
# Fitting SARIMA(1,1,1)(1,1,1,12) model
sarima_model = SARIMAX(df['Adj Close'], order=(1,1,1),␣
 ↪seasonal_order=(1,1,1,12))
sarima_result = sarima_model.fit()

# Displaying the model summary
sarima_result.summary()


# Make predictions
prediction = sarima_result.predict(start=test.index[0], end=test.index[-1])
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  5.72626D+00    |proj g|=  6.13091D-02
```

```
At iterate    5    f=  5.70110D+00    |proj g|=  4.96948D-03

At iterate   10    f=  5.69680D+00    |proj g|=  2.16268D-02

 This problem is unconstrained.

At iterate   15    f=  5.65455D+00    |proj g|=  9.67435D-03

At iterate   20    f=  5.64959D+00    |proj g|=  1.53006D-03

At iterate   25    f=  5.64932D+00    |proj g|=  1.18834D-03

At iterate   30    f=  5.64922D+00    |proj g|=  3.56343D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     33     38      1     0     0   2.717D-05   5.649D+00
  F =   5.6492145850592141

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

[47]:
```
df1['sarimaxPred'] = prediction
df1
```

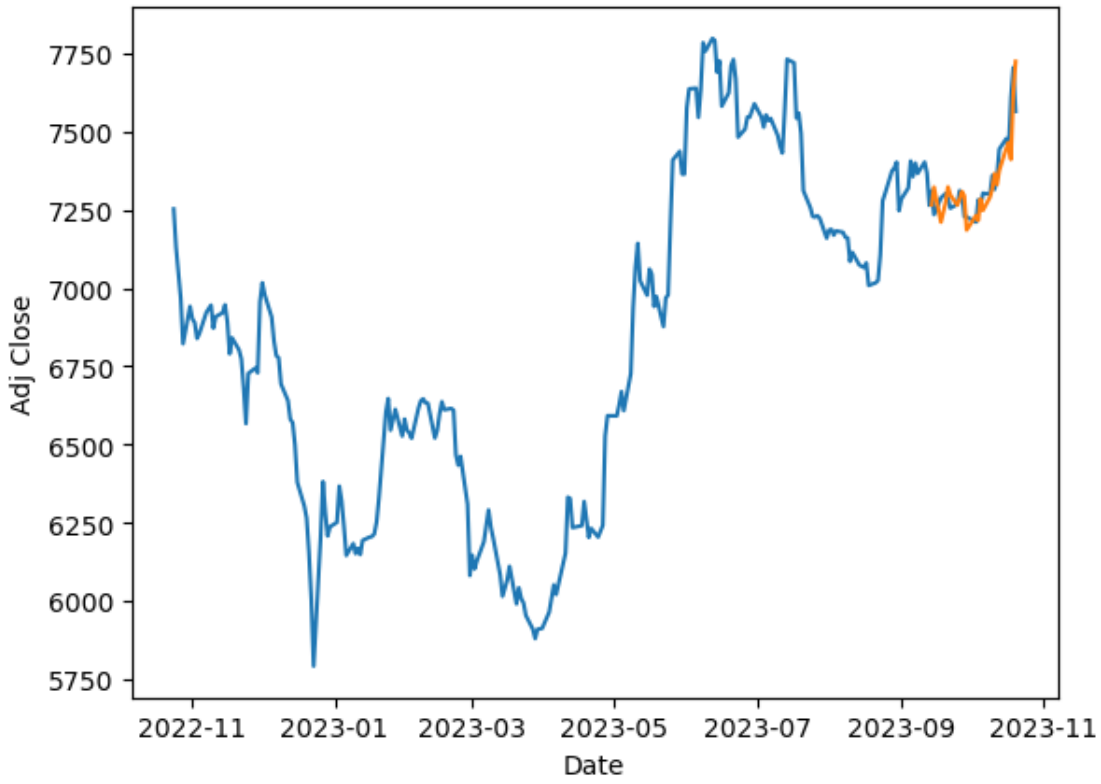
[47]:

Date	shiftdiff	Adj Close	arimaPred	sarimaxPred
2022-10-24	NaN	7252.444824	NaN	NaN
2022-10-25	-121.943847	7130.500977	NaN	NaN
2022-10-27	-162.029786	6968.471191	NaN	NaN
2022-10-28	-146.352539	6822.118652	NaN	NaN
2022-10-31	119.314453	6941.433105	NaN	NaN
…	…	…	…	…
2023-10-16	33.299805	7477.049805	7455.168076	7445.205805
2023-10-17	-31.699707	7445.350098	7478.752003	7465.119703
2023-10-18	163.549804	7608.899902	7438.680898	7411.093736

```
2023-10-19    95.149903  7704.049805  7642.460023  7624.252768
2023-10-20  -139.649903  7564.399902  7708.538345  7723.747377

[247 rows x 4 columns]
```

[48]:
```
sns.lineplot(data=df1, x=df1.index, y='Adj Close')
sns.lineplot(data=df1, x=df1.index, y='sarimaxPred')
```

[48]:
```
<Axes: xlabel='Date', ylabel='Adj Close'>
```



[53]:
```
futureDate = pd.DataFrame(pd.date_range(start = '2023-10-20', end =␣
 ↪'2023-11-20', freq='B'), columns = ['Dates'])
futureDate.set_index('Dates', inplace = True)
futureDate.head(10)
```

[53]:
```
Empty DataFrame
Columns: []
Index: [2023-10-20 00:00:00, 2023-10-23 00:00:00, 2023-10-24 00:00:00,
2023-10-25 00:00:00, 2023-10-26 00:00:00, 2023-10-27 00:00:00, 2023-10-30
00:00:00, 2023-10-31 00:00:00, 2023-11-01 00:00:00, 2023-11-02 00:00:00]
```

[54]:
```
import seaborn as sns

# Plot the actual data and in-sample predictions
sns.lineplot(data=df1, x= df1.index, y='Adj Close')
sns.lineplot(data=df1, x=df1.index, y='sarimaxPred')
sns.lineplot(data=df1, x=df1.index, y='arimaPred')

# Generate out-of-sample forecasts for SARIMA
forecast_values = sarima_result.forecast(steps=len(futureDate))

# Plot the SARIMA forecasts
sns.lineplot(x=futureDate.index, y=forecast_values, color='black',␣
 ↪label='SARIMA Forecast')
```

[54]:
```
<Axes: xlabel='Date', ylabel='Adj Close'>
```