

google-stock-prediction-1

November 29, 2023

```
[1]: import pandas as pd
      from matplotlib import pyplot
```

```
[2]: from pandas import read_csv
      data=pd.read_csv('C:\TSAdataset\GOOG_data.
      ↪csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
      data.head()
```

```
[2]:
```

	Open_Price	Close_Price	Adj_Close_Price	Volume
Date				
2019-06-14	1086.420044	1085.349976	1085.349976	1111500
2019-06-17	1086.280029	1092.500000	1092.500000	941600
2019-06-18	1109.689941	1103.599976	1103.599976	1386700
2019-06-19	1105.599976	1102.329956	1102.329956	1338800
2019-06-20	1119.989990	1111.420044	1111.420044	1262000

1 Ploting data in time series format

```
[3]: import plotly.express as px
      figure=px.line(data,x=data.index,y="Close_Price",title="Time Series_
      ↪Analysis(line plot)")
      figure.show()
```

2 Extracting time series analysis component

```
[4]: #De-Trend
```

```
[5]: import matplotlib.pyplot as plt
      import warnings
      warnings.filterwarnings('ignore')
      %matplotlib inline
```

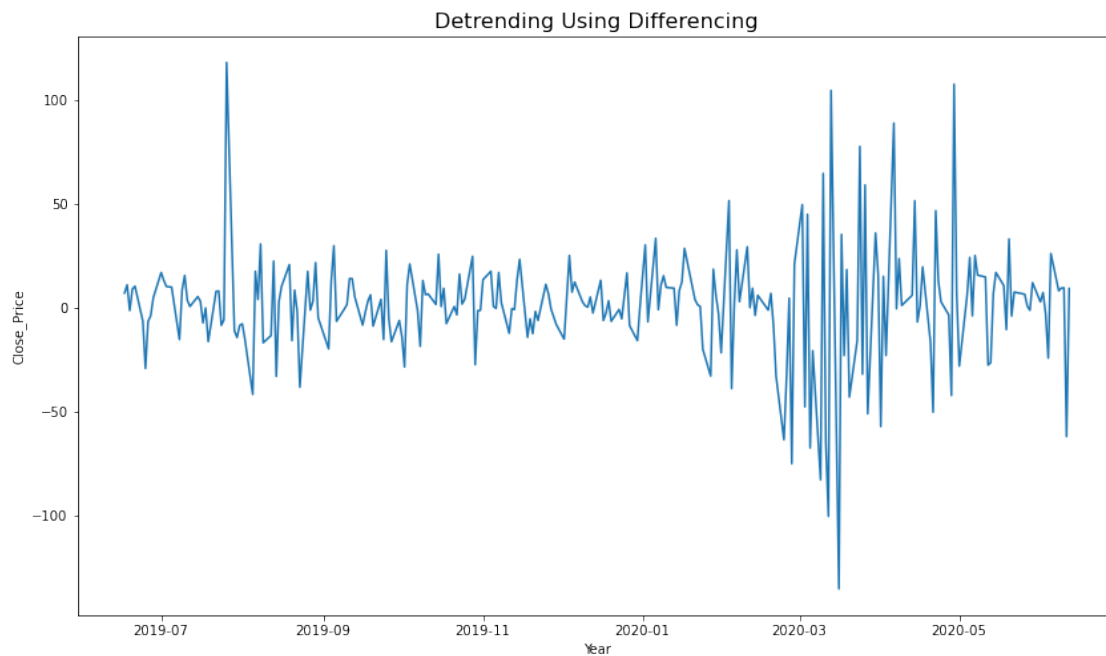
```
[6]: df=pd.read_csv('C:\TSAdataset\GOOG_data.
      ↪csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
      df.head()
```

```
[6]:
```

Date	Open_Price	Close_Price	Adj_Close_Price	Volume
2019-06-14	1086.420044	1085.349976	1085.349976	1111500
2019-06-17	1086.280029	1092.500000	1092.500000	941600
2019-06-18	1109.689941	1103.599976	1103.599976	1386700
2019-06-19	1105.599976	1102.329956	1102.329956	1338800
2019-06-20	1119.989990	1111.420044	1111.420044	1262000

```
[7]: diff=df.Close_Price.diff()
```

```
[8]: plt.figure(figsize=(14,8))
plt.plot(diff)
plt.title('Detrending Using Differencing',fontsize=16)
plt.xlabel('Year')
plt.ylabel('Close_Price')
plt.show()
```



```
[9]: #De-Seasonality
```

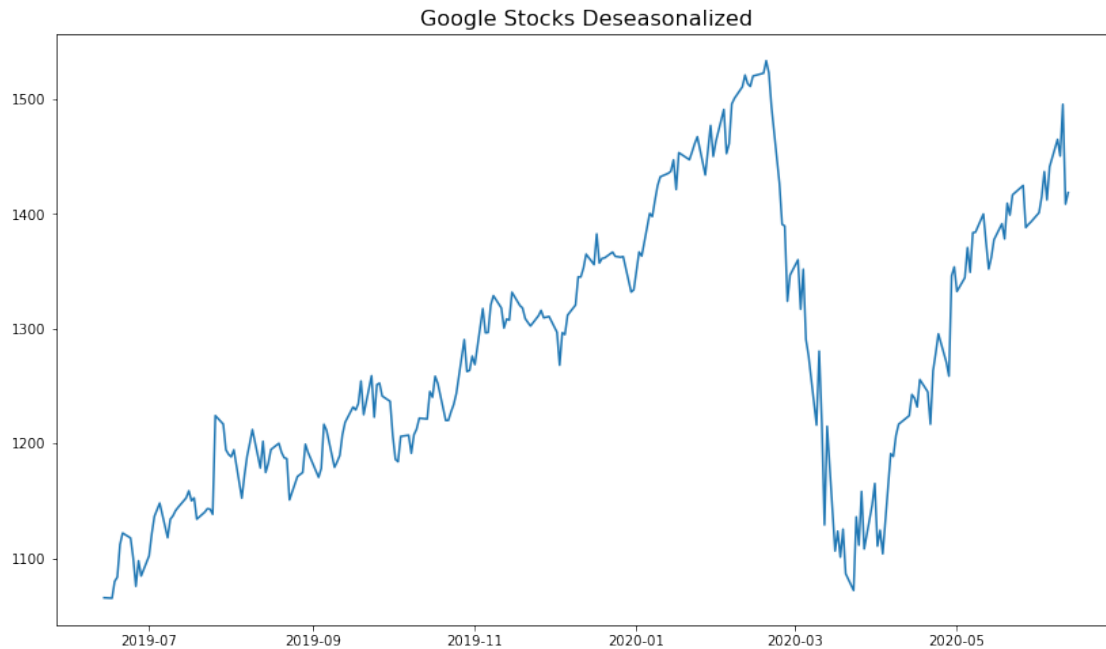
```
[10]: from statsmodels.tsa.seasonal import seasonal_decompose
```

```
[11]: result_mul = seasonal_decompose(df['Close_Price'], model='multiplicative',
    ↪period=30)
```

```
[12]: deseasonalized = df['Close_Price'].values / result_mul.seasonal
```

```
[13]: plt.figure(figsize=(14,8))
plt.plot(deseasonalized)
plt.title('Google Stocks Deseasonalized', fontsize=16)
plt.plot()
```

```
[13]: []
```



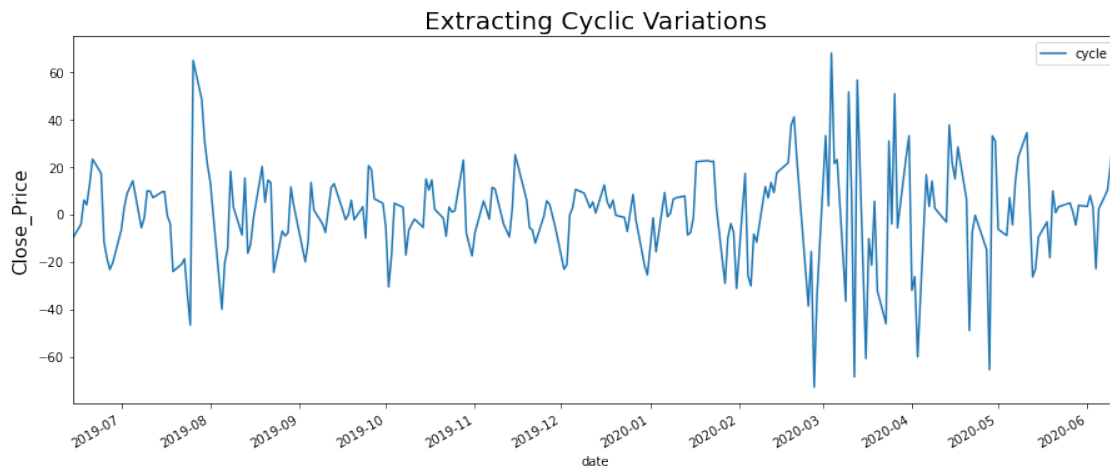
```
[14]: #Detecting cyclic variations
```

```
[15]: from statsmodels.tsa.filters.hp_filter import hpfilter
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
[16]: df=pd.read_csv('C:\TSAdataset\GOOG_data.
↵csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
```

```
[17]: GOOG_cycle,GOOG_trend = hpfilter(df['Close_Price'], lamb=100)
df['cycle'] =GOOG_cycle
df['trend'] =GOOG_trend
df[['cycle']].plot(figsize=(15,6)).autoscale(axis='x',tight=True)
plt.title('Extracting Cyclic Variations', fontsize=20)
plt.xlabel('date')
plt.ylabel('Close_Price', fontsize =15)
```

```
plt.show()
```



```
[18]: # Detecting irregularities in GOOG_data
```

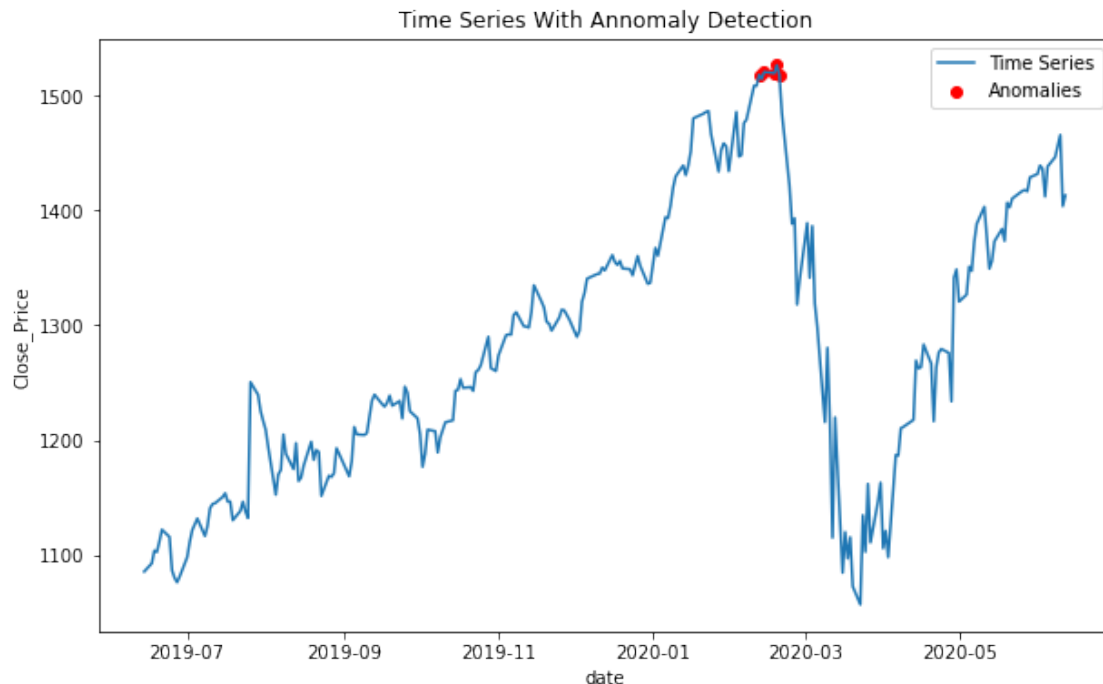
```
[19]: #I have used Z-score method to identify anomalies  
# set threshold to 2
```

```
[20]: z_scores = (df['Close_Price'] - df['Close_Price'].mean()) / df['Close_Price'].  
      ↪std()
```

```
[21]: z_score_threshold = 2.0
```

```
[22]: anomalies=df[abs(z_scores) > z_score_threshold]
```

```
[23]: plt.figure(figsize=(10,6))  
plt.plot(df.index,df['Close_Price'],label='Time Series')  
plt.scatter(anomalies.  
      ↪index,anomalies['Close_Price'],color='red',label='Anomalies')  
plt.xlabel('date')  
plt.ylabel('Close_Price')  
plt.title('Time Series With Anomaly Detection')  
plt.legend()  
plt.show()
```



3 Check whether data is stationary or non stationary

```
[24]: from pandas import read_csv
      from statsmodels.tsa.stattools import adfuller
```

```
[25]: df=pd.read_csv('C:\TSAdataset\GOOG_data.
      ↪csv',header=0,index_col=0,parse_dates=True)
      df.head()
      df1=df.drop(['Volume', 'Open_Price','Adj_Close_Price'], axis=1)
```

```
[26]: result=adfuller(df1)
```

```
[27]: print('ADF statistic: %f'% result[0])
```

ADF statistic: -2.497410

```
[28]: print('Critical values :')
      for key,value in result[4].items():
          print('\t%s: %.3f' % (key,value))
```

Critical values :

1%:-3.458
5%:-2.874
10%:-2.573

```
[29]: if result[0]>result[4]["5%"]:
        print("reject H0 - Time Series Is Stationary")
    else:
        print("failed to reject H0 - Time Series Is Non Stationary")
```

reject H0 - Time Series Is Stationary

4 Implement Auto Regressive model for time series

```
[30]: from pandas import read_csv
        from matplotlib import pyplot
        from statsmodels.tsa.ar_model import AutoReg
        from sklearn.metrics import mean_squared_error
        from math import sqrt
```

```
[31]: #load the dataset

df=pd.read_csv('C:\TSAdataset\GOOG_data.
↪csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
google_stock=df['Close_Price']
```

```
[32]: #split data into training and testing sets

train_size = int(len(google_stock) * 0.8)
train, test = google_stock[:train_size], google_stock[train_size:]
```

```
[33]: #fit AR model

lags = 10
model = AutoReg(train, lags=lags)
model_fit = model.fit()
```

```
[34]: #predict using the trained AR model

start = len(train)
end = len(train) + len(test) - 1
predictions = model_fit.predict(start=start, end=end, dynamic=False)
```

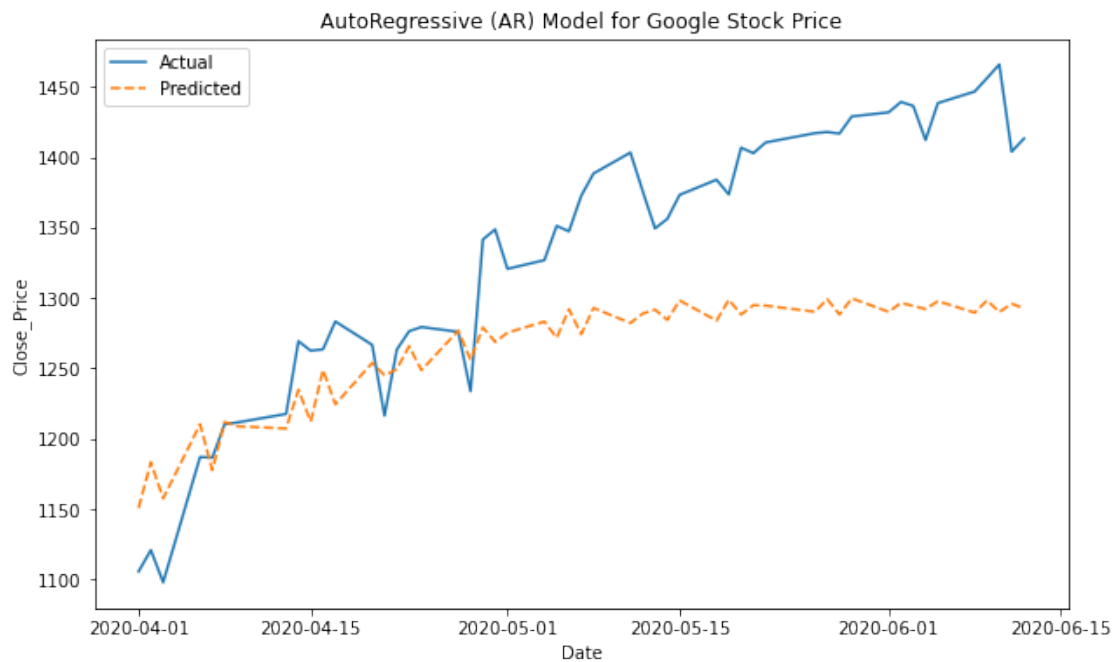
```
[35]: #calculate mean squared error

mse = mean_squared_error(test, predictions)
print(f"Mean Squared Error: {mse}")
```

Mean Squared Error: 8215.914903466022

```
[36]: #plot actual and predicted values
```

```
plt.figure(figsize=(10, 6))
plt.plot(test.index, test.values, label='Actual')
plt.plot(test.index, predictions, label='Predicted', linestyle='--')
plt.legend()
plt.title('AutoRegressive (AR) Model for Google Stock Price')
plt.xlabel('Date')
plt.ylabel('Close_Price')
plt.show()
```



5 Validate the model

```
[37]: import numpy as np
import pandas as pd
from sklearn.model_selection import TimeSeriesSplit
from statsmodels.tsa.ar_model import AutoReg
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
[38]: #load the dataset
```

```
df=pd.read_csv('C:\TSAdataset\GOOG_data.
↪ csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
```

```
google_stock=df['Close_Price']
```

```
[39]: #number of splits for cross validation
```

```
n_splits=5
```

```
[40]: # Initialize TimeSeriesSplit
```

```
tscv = TimeSeriesSplit(n_splits=n_splits)
```

```
mse_scores = []
```

```
# To store Mean Squared Error
```

```
↪ (MSE) scores
```

```
[41]: #perform cross validation
```

```
for train_idx, test_idx in tscv.split(google_stock):
```

```
    train, test = google_stock.iloc[train_idx], google_stock.iloc[test_idx]
```

```
    lags = 10
```

```
    model = AutoReg(train, lags=lags)
```

```
    model_fit = model.fit()
```

```
    start = len(train)
```

```
    end = len(train) + len(test) - 1
```

```
    predictions = model_fit.predict(start=start, end=end)
```

```
    mse = mean_squared_error(test, predictions)
```

```
    mse_scores.append(mse)
```

```
[42]: #Plot the actual and predicted values for the current fold
```

```
plt.figure(figsize=(10, 6))
```

```
plt.plot(test.index, test.values, label='Actual')
```

```
plt.plot(test.index, predictions, label='Predicted', linestyle='--')
```

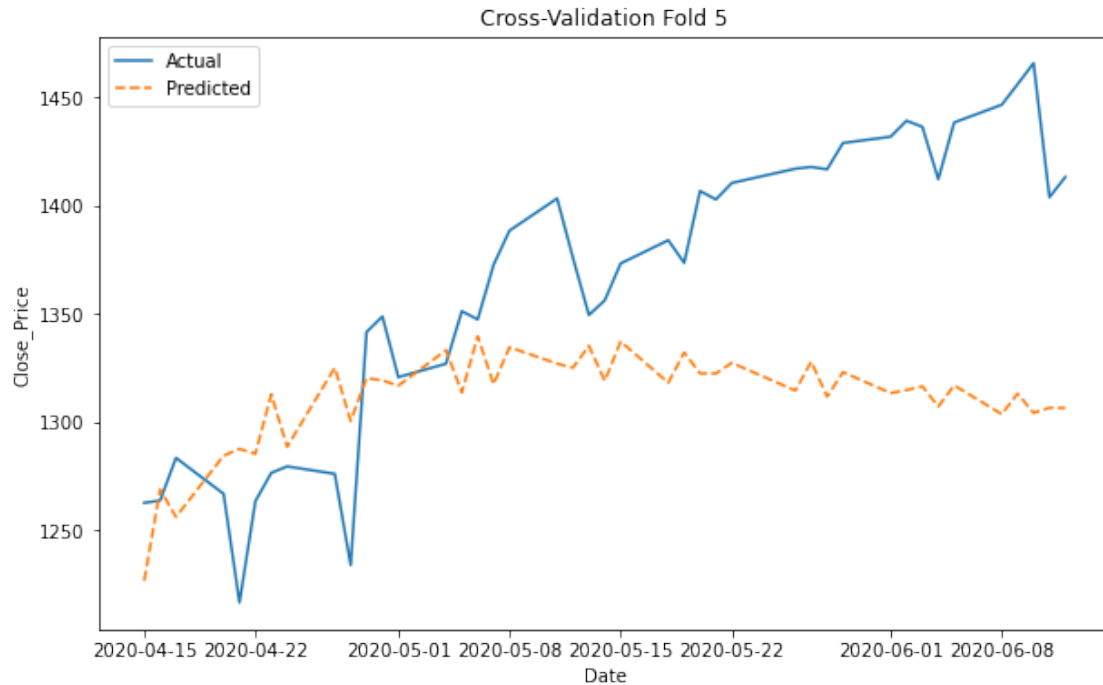
```
plt.legend()
```

```
plt.title(f'Cross-Validation Fold {len(mse_scores)}')
```

```
plt.xlabel('Date')
```

```
plt.ylabel('Close_Price')
```

```
plt.show()
```

```
[43]: # Calculate the average MSE across folds

average_mse = np.mean(mse_scores)
print(f"Average Mean Squared Error: {average_mse}")
```

Average Mean Squared Error: 30472.09208227747

6 Use the model to forecast future values

```
[44]: import pandas as pd
from statsmodels.tsa.ar_model import AutoReg
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
```

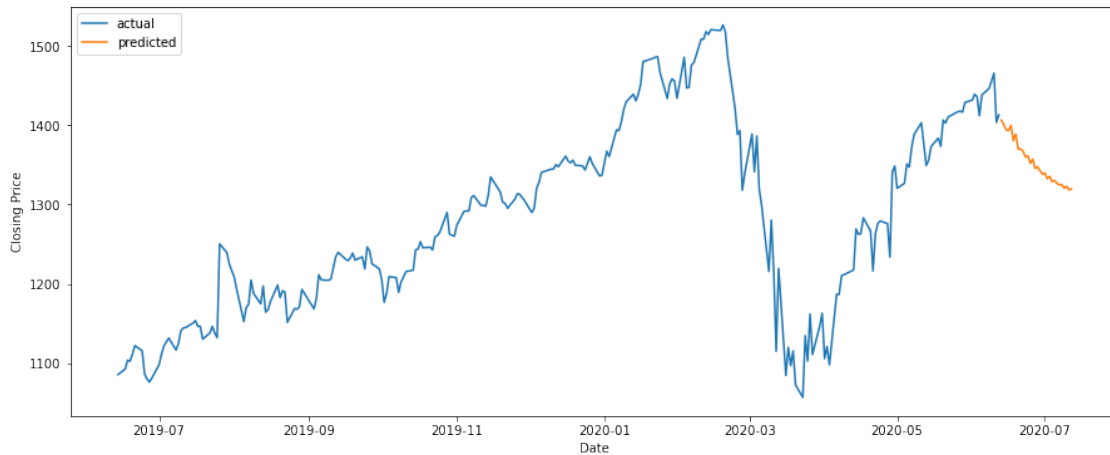
```
[45]: df=pd.read_csv('C:\TSAdataset\GOOG_data.
↪ csv',header=0,index_col=0,parse_dates=True,infer_datetime_format=True)
closing_prices=df['Close_Price']
```

```
[46]: order = 10 # AR order (p)
model = AutoReg(closing_prices, lags=order)
model_fit = model.fit()
```

```
[47]: forecast_periods = 30 # Number of future periods to forecast
```

```
forecast_index = pd.date_range(start=closing_prices.index[-1] +  
    ↪timedelta(days=1), periods=forecast_periods, freq='D')  
forecast = model_fit.forecast(steps=forecast_periods)
```

```
[48]: plt.figure(figsize=(15,6))  
plt.plot(closing_prices.index, closing_prices, label='Historical')  
plt.plot(forecast_index, forecast, label='Forecast')  
plt.xlabel('Date')  
plt.ylabel('Closing Price')  
plt.legend(["actual", "predicted"], loc ="upper left")  
plt.show()
```



```
[ ]:
```

```
[ ]:
```