CS575 Project_Facebook dataset

Name- Vipin Gupta

Roll- 2011MT22

Downloading & Exporting the dataset

```
from pandas_datareader import data as pdr
from datetime import datetime
```

```
#download data
fb = pdr.DataReader('FB', 'yahoo', start=datetime(2014, 8, 1), end=datetime(2016, 11, 30))
```

```
#print first few lines of data
print(fb.head())
```

```
                 High        Low       Open      Close    Volume  Adj Close
Date
2014-08-01  73.220001  71.550003  72.220001  72.360001  43535000  72.360001
2014-08-04  73.879997  72.360001  72.360001  73.510002  30777000  73.510002
2014-08-05  73.589996  72.180000  73.199997  72.690002  34986000  72.690002
2014-08-06  73.720001  71.790001  72.019997  72.470001  30986000  72.470001
2014-08-07  74.000000  72.699997  73.000000  73.169998  38141000  73.169998
```

```
#export and save as csv files
fb.to_csv('Facebook_stock.csv', sep=',')
```

```
#Visulaizing the close data
import matplotlib.pyplot as plt
fb["Close"].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dfe4978d0>
```

Statistical analysis like ACF, PACF, ADF, KPSS Test

```
#Importing libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller,kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecat
  import pandas.util.testing as tm
```

```
#ADF Test

def adf_test(atr):

    #Perform Dickey-Fuller test:
    timeseries = fb[atr].dropna()
    print ('Results of Dickey-Fuller Test for ',atr,'\n')
    dftest = adfuller(timeseries, 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)
```

```
#apply adf test on the series
adf_test('Close')
```

```
Results of Dickey-Fuller Test for  Close

Test Statistic                  -0.966644
p-value                          0.765235
#Lags Used                       6.000000
Number of Observations Used    582.000000
Critical Value (1%)             -3.441636
Critical Value (5%)             -2.866519
Critical Value (10%)            -2.569422
dtype: float64
```

The p value obtained is greater than significance level of 0.05 and test statistic is higher than any of the critical values

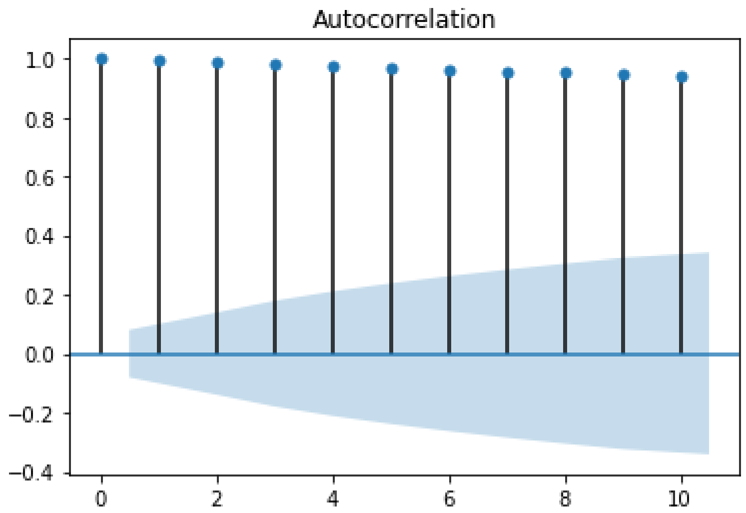
so we cant reject the null hypothesis so the time series is non stationary.

```
#KPSS Test

def kpss_test(atr):
    timeseries = fb[atr].dropna()
    print ('Results of KPSS Test for ',atr)
    kpsstest = kpss(timeseries, regression='c')
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print (kpss_output)

kpss_test('Close')
```

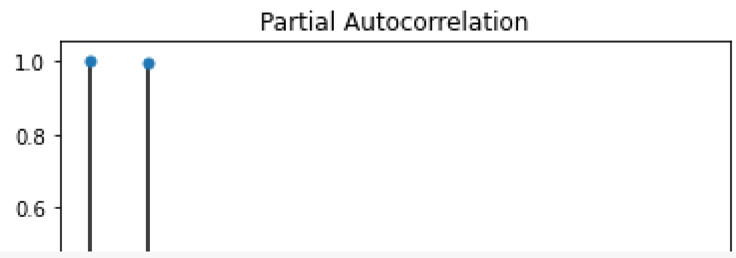
```
Results of KPSS Test for  Close
Test Statistic            2.941636
p-value                   0.010000
Lags Used                19.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:1685: FutureWarning: The behavior of using lags=Nor
  warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:1709: InterpolationWarning: p-value is smaller thar
  warn("p-value is smaller than the indicated p-value", InterpolationWarning)
```

The p value is significant less than 0.05 hence we can reject the null hypothesis so series is non stationary

```
# ACF Test of differenced data
plot_acf(fb['Close'].dropna(), lags=10)
plt.show()
```

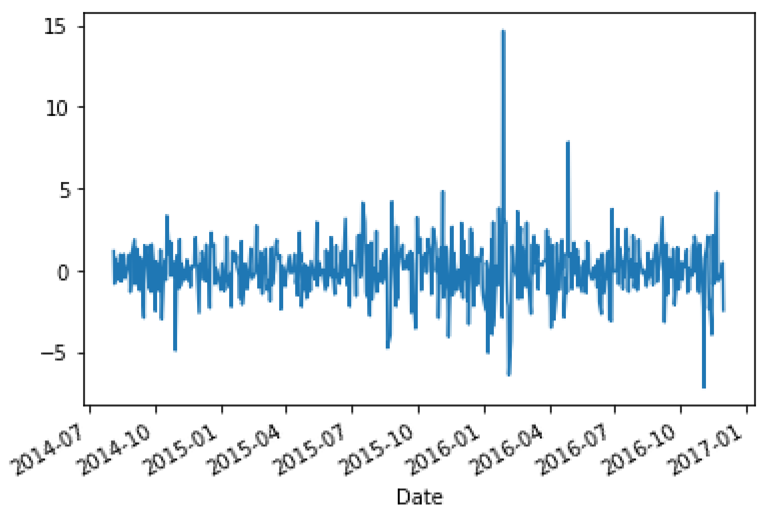



```
# PACF Test of differenced data
plot_pacf(fb['Close'].dropna(), lags=10)
plt.show()
```




```
#Differencing to make data as stationary

#Differencing the data
fb['diff'] = fb['Close'].diff(periods=1)

#Visulaizing the differenced data
fb["diff"].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8de5d39890>
```




```
# ADF Test of differenced data
adf_test('diff')
```

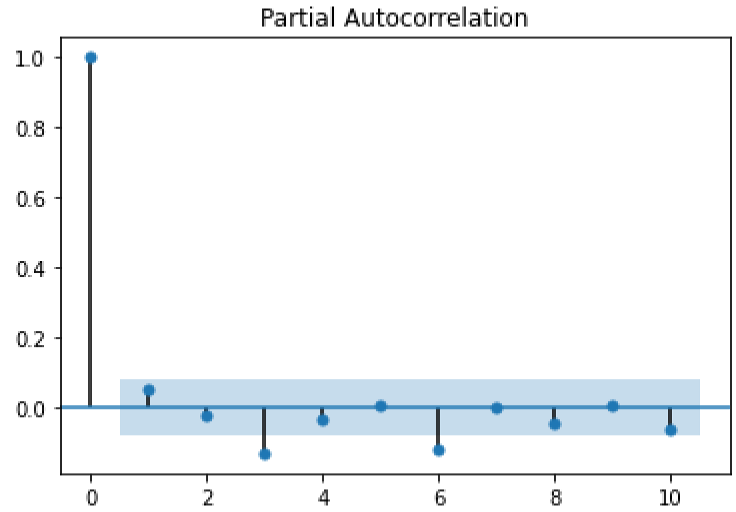
```
Results of Dickey-Fuller Test for  diff

Test Statistic                 -1.191784e+01
p-value                         5.114835e-22
#Lags Used                      5.000000e+00
Number of Observations Used     5.820000e+02
Critical Value (1%)            -3.441636e+00
Critical Value (5%)            -2.866519e+00
Critical Value (10%)           -2.569422e+00
dtype: float64
```

```
# KPSS Test of differenced data
kpss_test('diff')
```

```
Results of KPSS Test for  diff
Test Statistic             0.058162
p-value                    0.100000
Lags Used                 19.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:1685: FutureWarning: The behavior of using lags=Nor
  warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:1711: InterpolationWarning: p-value is greater thar
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

```
# ACF Test of differenced data
plot_acf(fb['diff'].dropna(), lags=10)
plt.show()
```


```
# PACF Test of differenced data
plot_pacf(fb['diff'].dropna(), lags=10)
plt.show()
```



Exponential

```
import numpy as np
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

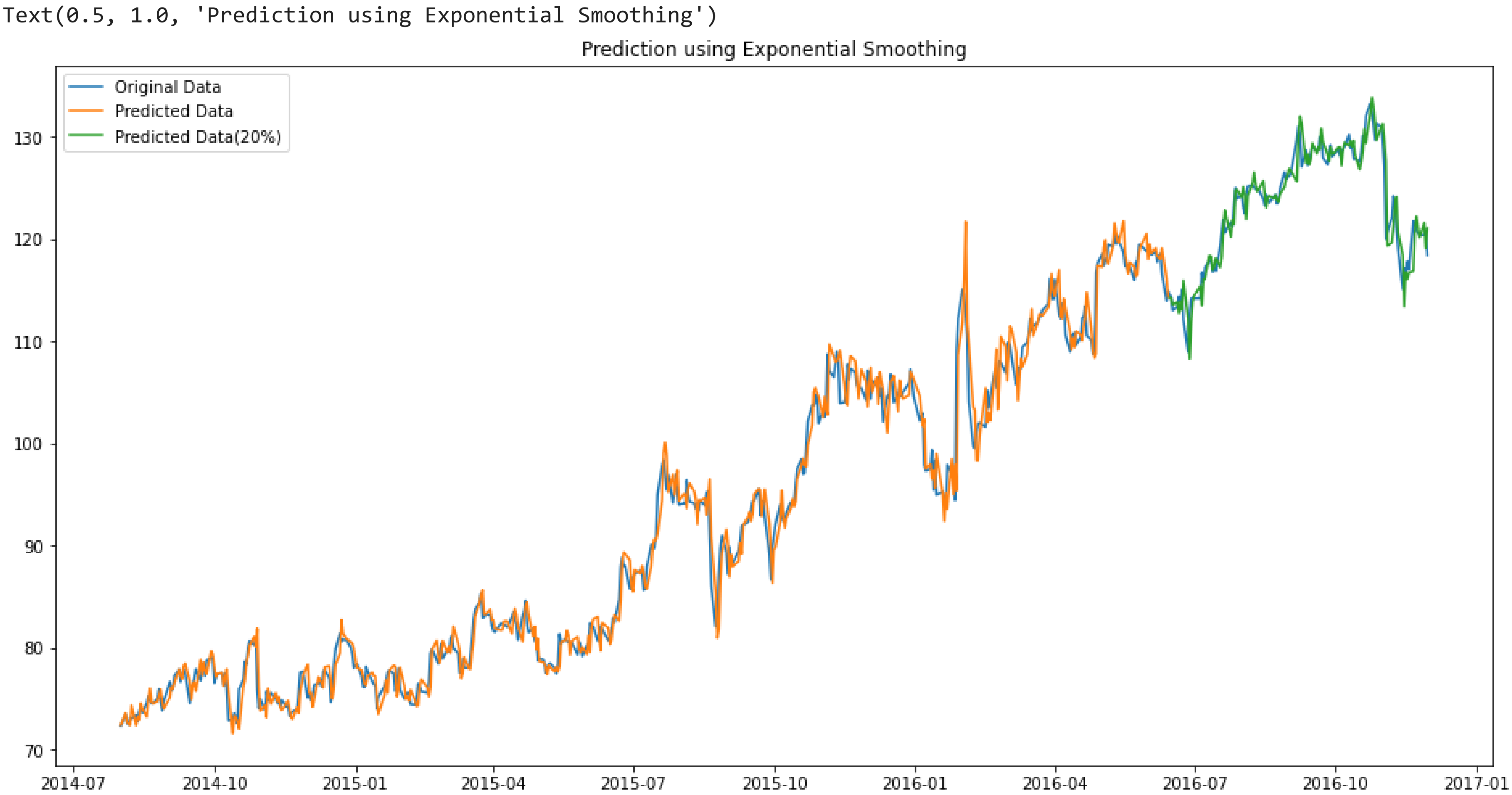
```
n = int(len(fb["Close"])*0.8)
data = fb['Close'].to_numpy()
train2 = data[:n]
test2 = data[n:]
date = (fb.index)
```

```
Exp_model = ExponentialSmoothing(fb.Close,trend='mul',seasonal='mul',seasonal_periods=4)
fb['Pred_Exp'] = Exp_model.fit(smoothing_level = 0.9,smoothing_slope= 0.1,smoothing_seasonal = 0.2).fittedvalues.shift(0)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarning: A date index has been provi
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
plt.figure(figsize=(16,8))
plt.plot(date,data, label='Original Data')
plt.plot(date[:n],fb.Pred_Exp[:n], label='Predicted Data')
plt.plot(date[n:],fb.Pred_Exp[n:], label='Predicted Data(20%)')
plt.legend()
plt.title('Prediction using Exponential Smoothing')
```

```
Text(0.5, 1.0, 'Prediction using Exponential Smoothing')
```




```
#Calculation of MSE for comparing the model
```

```
rmse2 = (np.mean(np.power((np.array(test2)-np.array(fb.Pred_Exp[n:])),2)))
print('MSE value using Exponential Smoothing model: ',rmse2)
```

```
MSE value using Exponential Smoothing model:  3.126790071926512
```

ARIMA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import lag_plot
from pandas import datetime
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: FutureWarning: The pandas.datetime class is deprecated
  """
```

```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Importing data
df = pd.read_csv('Facebook_stock.csv')
df.head()
```

	Date	High	Low	Open	Close	Volume	Adj Close
0	2014-08-01	73.220001	71.550003	72.220001	72.360001	43535000	72.360001
1	2014-08-04	73.879997	72.360001	72.360001	73.510002	30777000	73.510002
2	2014-08-05	73.589996	72.180000	73.199997	72.690002	34986000	72.690002
3	2014-08-06	73.720001	71.790001	72.019997	72.470001	30986000	72.470001
4	2014-08-07	74.000000	72.699997	73.000000	73.169998	38141000	73.169998

```
# Extracting the required columns
df = df[['Date', 'Close']]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 589 entries, 0 to 588
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    589 non-null    object 
 1   Close   589 non-null    float64
dtypes: float64(1), object(1)
memory usage: 9.3+ KB
```

```
# Changing the Date column to proper DateTime object
df.Date = pd.to_datetime(df.Date)
df.info()
```

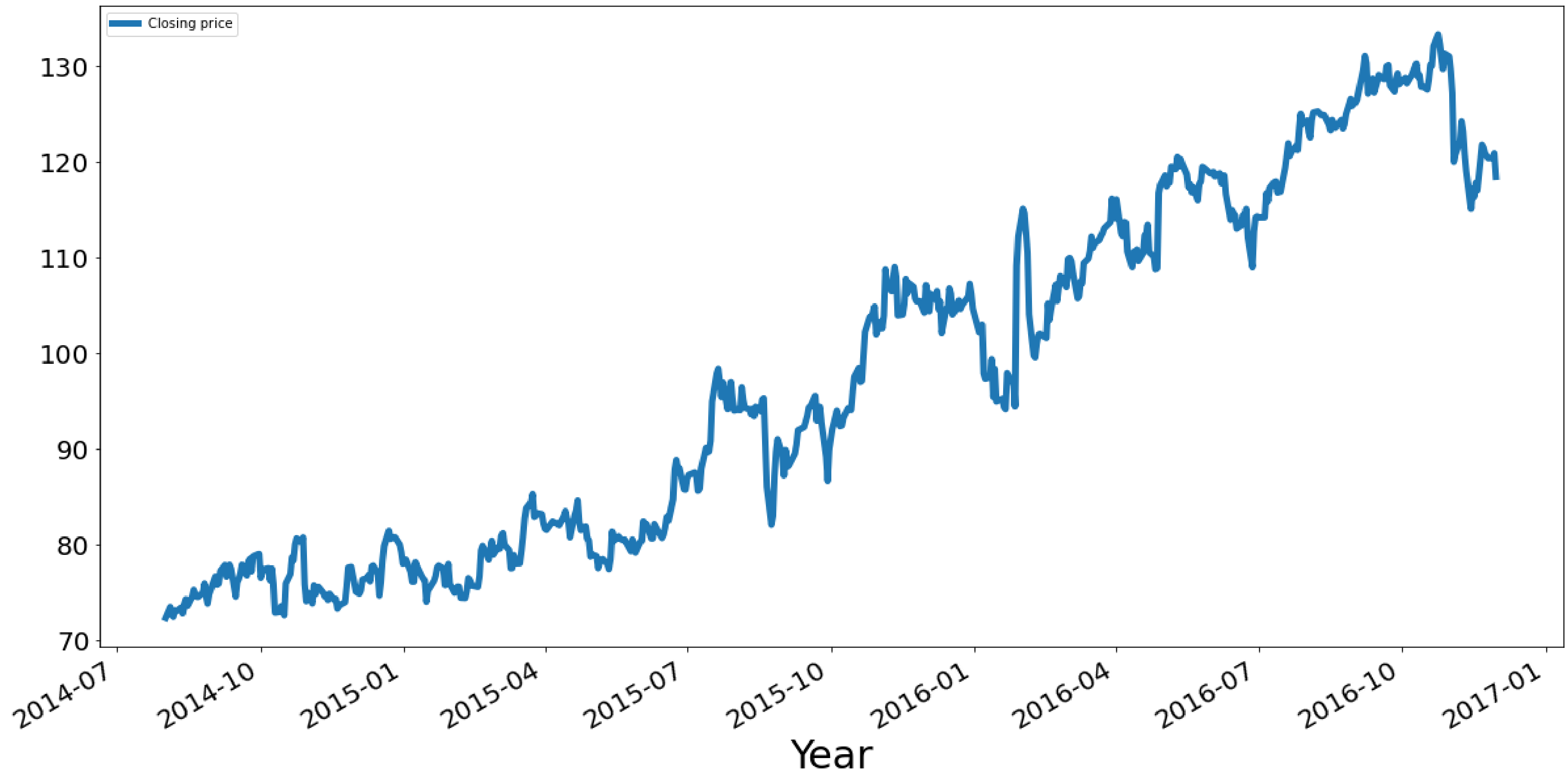
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 589 entries, 0 to 588
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Date    589 non-null    datetime64[ns]
 1   Close   589 non-null    float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 9.3 KB
```

```
# Making Date column to be the index
df.columns=['Date','Closing price']
df.set_index('Date', inplace=True)
df.head()
```

	Closing price
Date	
2014-08-01	72.360001

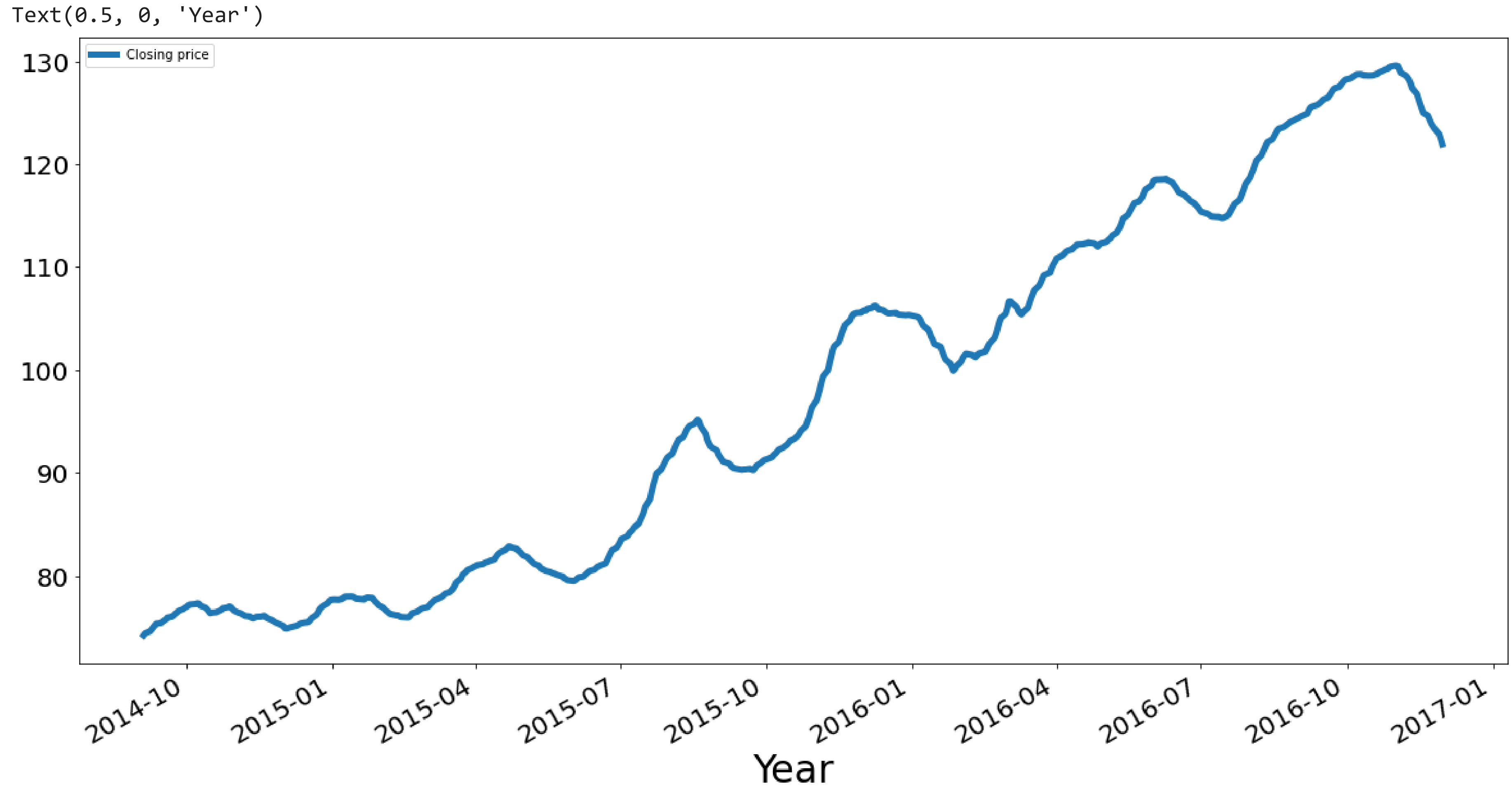
```
# Plot
df.plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
```

```
Text(0.5, 0, 'Year')
```




```
# Seeing the trend more clearly
df.rolling(24).mean().plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
# Overall a rise here
```

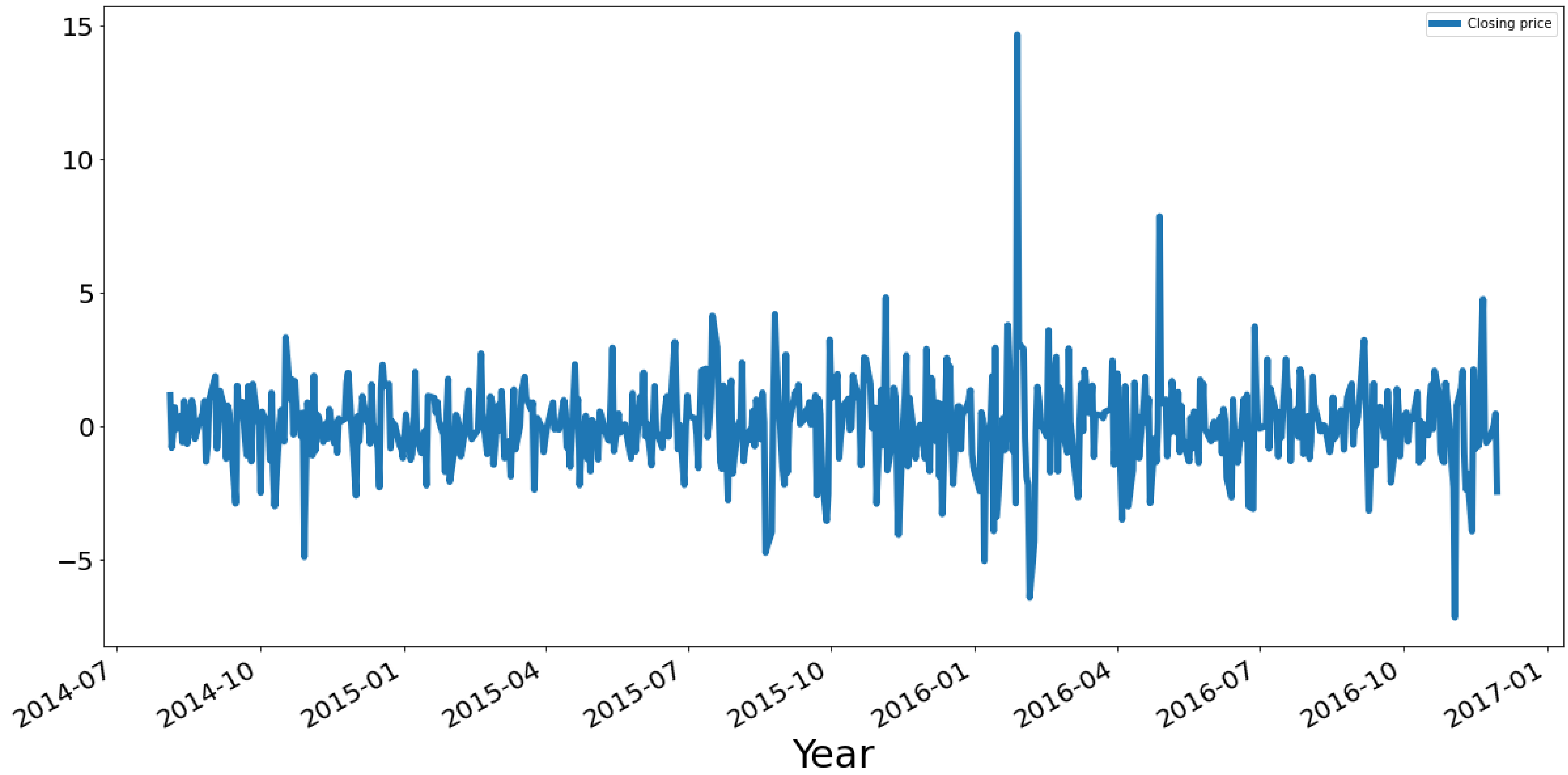
```
Text(0.5, 0, 'Year')
```




```
# We can see that there is no specific seasonality here
```

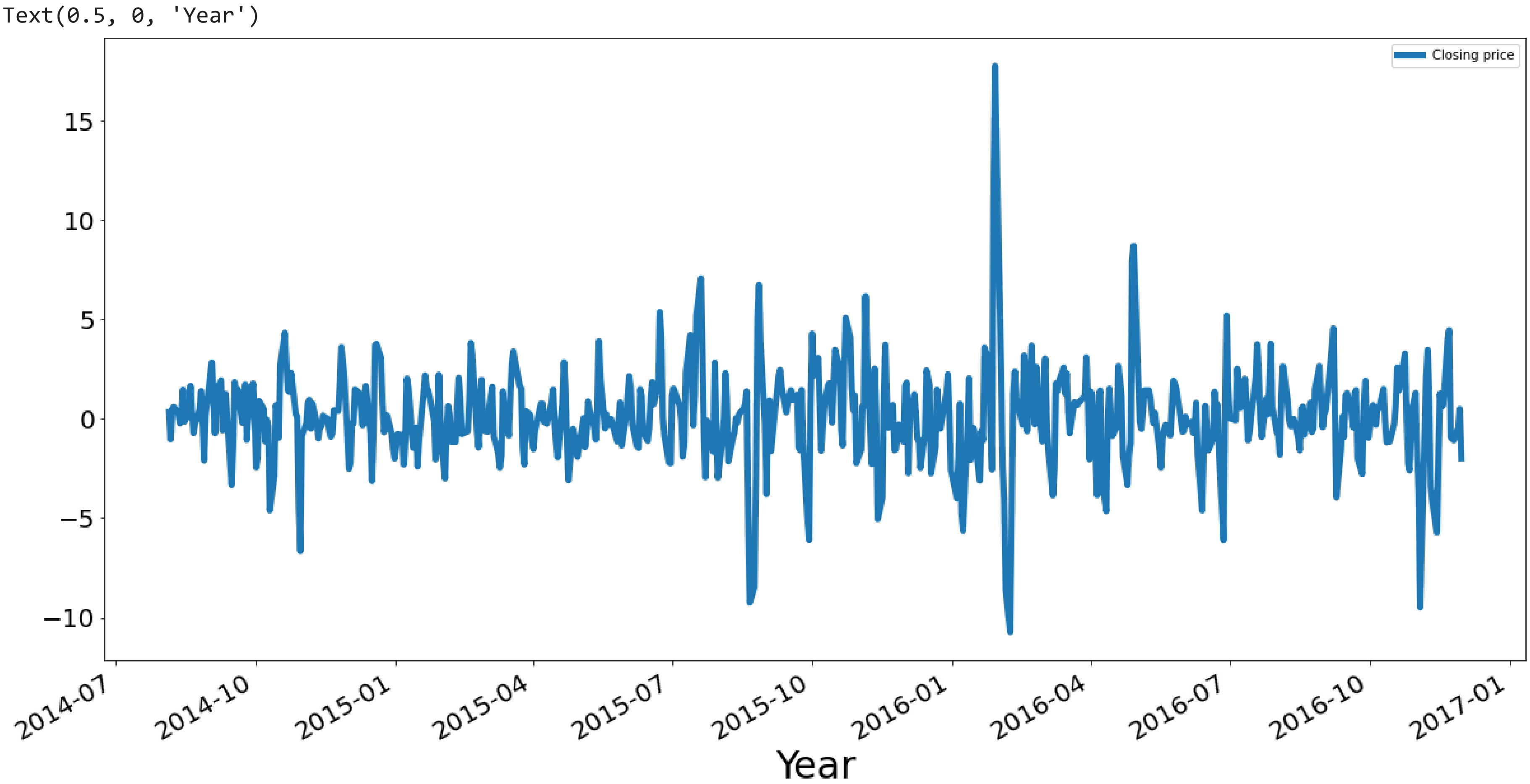
```
# Removing trend
df.diff().plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
```

```
Text(0.5, 0, 'Year')
```



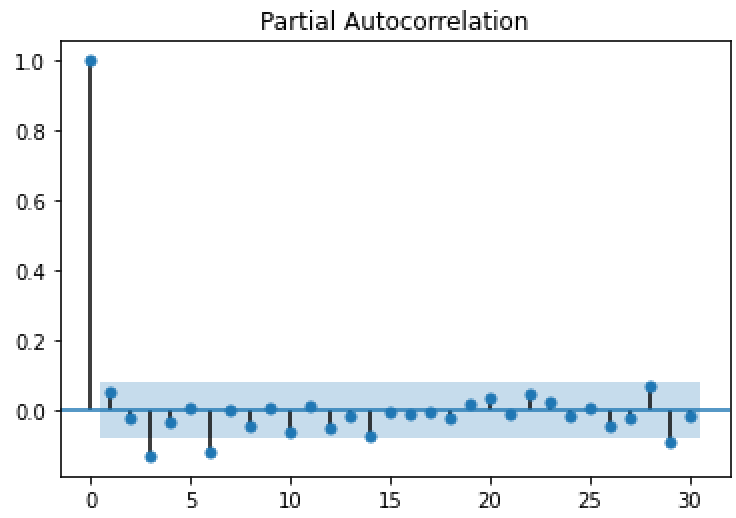

```
# 2nd order differencing
df.diff(periods=2).plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
```

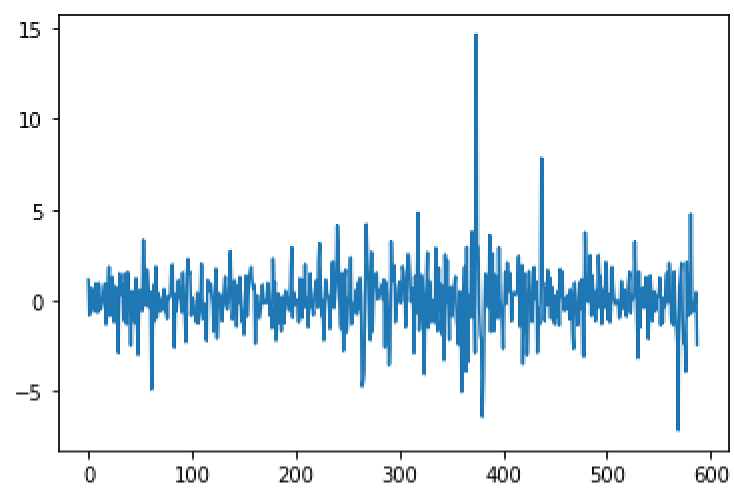
```
Text(0.5, 0, 'Year')
```

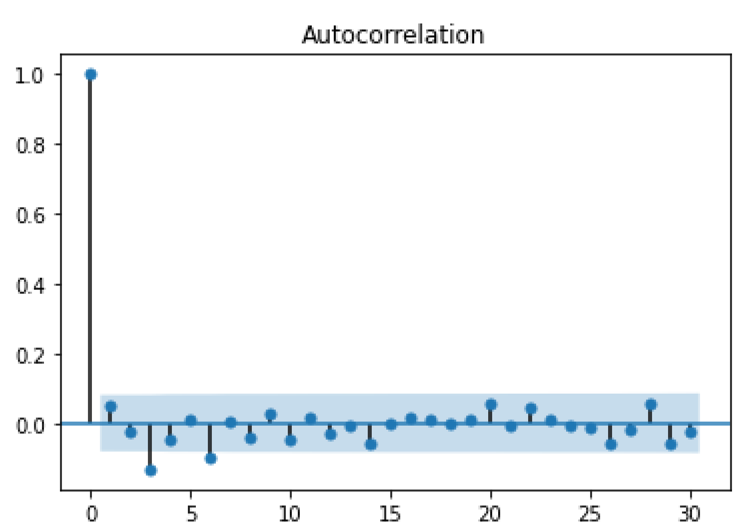


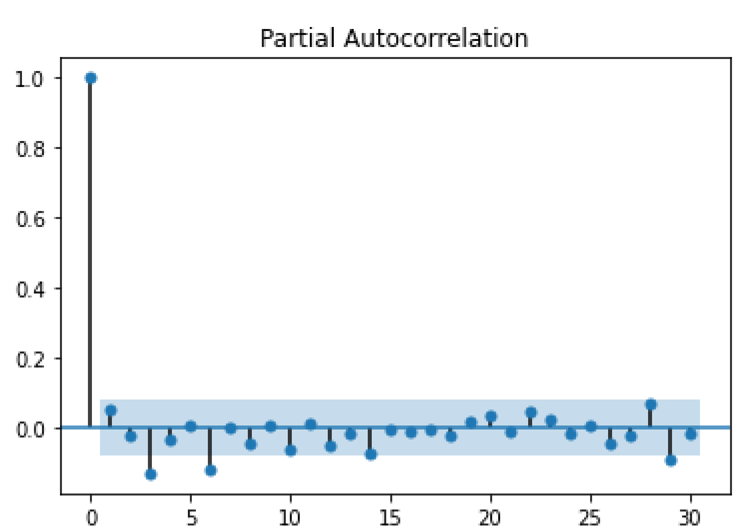

```
# Let's take a look at its auto-corelation plots
# Before that we'll have to do manual differencing
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

def difference(data, lag):
    diff= []

    for i in range(lag, len(data)):
        value = data[i] - data[i-lag]
```

```
        value = data[i] - data[i-lag]
        diff.append(value)
    return pd.Series(diff)

df_close = df['Closing price']
X = df_close.values
diff = difference(X,1)
plt.plot(diff)
df_diff = pd.DataFrame(diff)
plot_acf(df_diff, lags=30)
plot_pacf(df_diff, lags=30)
```









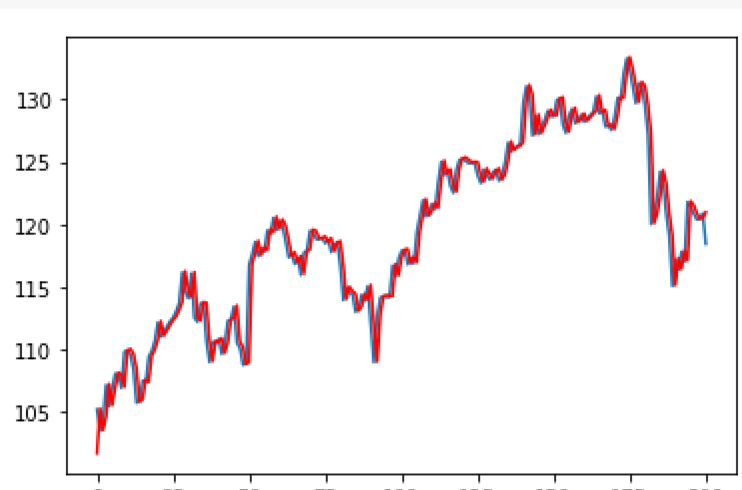

```
# Forecasting
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARIMA
df = df.astype(np.float64)
Y = df.values
size = int(len(Y)*0.66)
train, test = Y[0:size], Y[size:len(Y)]
```

```
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=(0,1,0))
```

```
    model = ARIMA(history, order=(0,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    pred = output[0]
    predictions.append(pred)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f'%(pred, obs))
```

```
predicted=101.685582, expected=105.199997
predicted=105.284636, expected=103.470001
predicted=103.549976, expected=104.570000
predicted=104.652589, expected=107.160004
predicted=107.249006, expected=105.459999
predicted=105.544438, expected=106.879997
predicted=106.967834, expected=108.070000
predicted=108.160634, expected=107.919998
predicted=108.010023, expected=106.919998
predicted=107.007271, expected=109.820000
predicted=109.914357, expected=109.949997
predicted=110.044444, expected=109.580002
predicted=109.673285, expected=108.389999
predicted=108.480074, expected=105.730003
predicted=105.813220, expected=105.930000
predicted=106.013508, expected=107.510002
predicted=107.597223, expected=107.320000
predicted=107.406534, expected=109.410004
predicted=109.501485, expected=109.889999
predicted=109.982438, expected=110.669998
predicted=110.764126, expected=112.180000
predicted=112.277598, expected=111.019997
predicted=111.114520, expected=111.449997
predicted=111.545338, expected=111.849998
predicted=111.946081, expected=112.250000
predicted=112.346820, expected=112.540001
predicted=112.637289, expected=113.050003
predicted=113.148288, expected=113.690002
predicted=113.789593, expected=116.139999
predicted=116.245240, expected=114.699997
predicted=114.801532, expected=114.099998
predicted=114.199855, expected=116.059998
predicted=116.164293, expected=112.550003
predicted=112.645694, expected=112.220001
predicted=112.314681, expected=113.709999
predicted=113.807985, expected=113.639999
predicted=113.737588, expected=110.629997
predicted=110.720257, expected=108.989998
predicted=109.076186, expected=110.610001
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimiza
  "Check mle_retvals", ConvergenceWarning)
predicted=110.699789, expected=110.510002
predicted=110.599346, expected=110.839996
predicted=110.929903, expected=109.639999
predicted=109.726899, expected=110.449997
predicted=110.538578, expected=112.290001
predicted=112.382646, expected=112.419998
predicted=112.512730, expected=113.440002
predicted=113.534875, expected=110.559998
predicted=110.648016, expected=110.099998
predicted=110.186757, expected=108.760002
predicted=108.843488, expected=108.889999
predicted=108.973592, expected=116.730003
predicted=116.831305, expected=117.580002
predicted=117.683009, expected=118.570000
predicted=118.675022, expected=117.430000
predicted=117.532200, expected=118.059998
predicted=118.163391, expected=117.809998
```

```
plt.plot(test)
plt.plot(predictions, color='red')
plt.show()
```



```
#Calculation of MSE for comparing the model
difference_array = np.subtract(test, predictions)
squared_array = np.square(difference_array)
mse = squared_array.mean()
mse
```

```
2.4418359598771464
```

LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout
```

importing the training data

```
data = pd.read_csv('Facebook_stock.csv')
```

choosing the close column

```
data["Close"]=pd.to_numeric(data.Close,errors='coerce') #turning the Close column to numeric
data = data.dropna() #romeving the NA values
trainData = data.iloc[:,4:5].values #selecting only the closing prices for training
```

scaling the values in the range of 0-1 for best preformances

```
sc = MinMaxScaler(feature_range=(0,1))
trainData = sc.fit_transform(trainData)
trainData.shape
```

```
(589, 1)
```

preparing the data for LSTM

since its a time series problem we took 60 as timestep for our learning : given 60 closing values as an input data the 61st value is our output

```
X_train = []
y_train = []

for i in range (60,589): #60 : timestep // 1149 : length of the data
    X_train.append(trainData[i-60:i,0])
    y_train.append(trainData[i,0])

X_train,y_train = np.array(X_train),np.array(y_train)
```

ps : LSTM take a 3D tensor (seq_len,timestep,batch_size)

```
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1)) #adding the batch_size axis
X_train.shape
```

```
(529, 60, 1)
```

building the model

```
model = Sequential()

model.add(LSTM(units=100, return_sequences = True, input_shape =(X_train.shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))
```

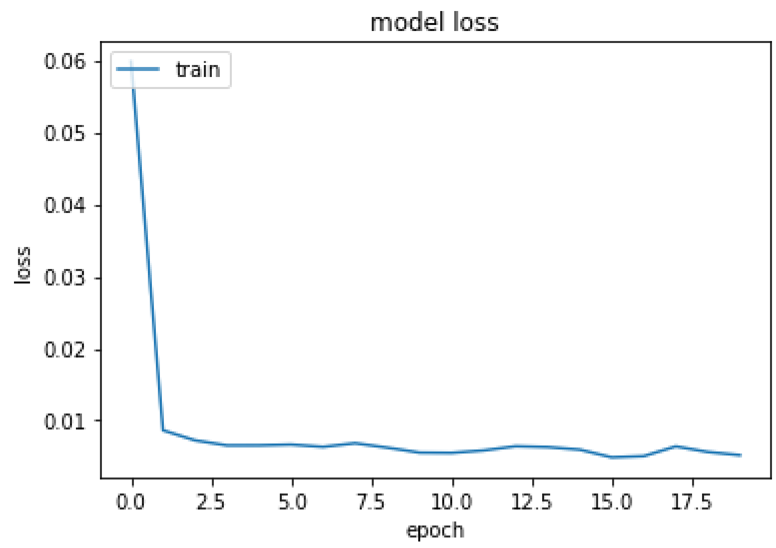
```
model.add(LSTM(units=100, return_sequences = False))
model.add(Dropout(0.2))

model.add(Dense(units =1))
model.compile(optimizer='adam',loss="mean_squared_error")
```

```
hist = model.fit(X_train, y_train, epochs = 20, batch_size = 32, verbose=2)
```

```
Epoch 1/20
17/17 - 10s - loss: 0.0598
Epoch 2/20
17/17 - 3s - loss: 0.0086
Epoch 3/20
17/17 - 3s - loss: 0.0072
Epoch 4/20
17/17 - 3s - loss: 0.0065
Epoch 5/20
17/17 - 3s - loss: 0.0065
Epoch 6/20
17/17 - 3s - loss: 0.0067
Epoch 7/20
17/17 - 3s - loss: 0.0063
Epoch 8/20
17/17 - 3s - loss: 0.0068
Epoch 9/20
17/17 - 3s - loss: 0.0062
Epoch 10/20
17/17 - 3s - loss: 0.0055
Epoch 11/20
17/17 - 3s - loss: 0.0055
Epoch 12/20
17/17 - 3s - loss: 0.0058
Epoch 13/20
17/17 - 3s - loss: 0.0064
Epoch 14/20
17/17 - 3s - loss: 0.0063
Epoch 15/20
17/17 - 3s - loss: 0.0059
Epoch 16/20
17/17 - 3s - loss: 0.0049
Epoch 17/20
17/17 - 3s - loss: 0.0050
Epoch 18/20
17/17 - 3s - loss: 0.0064
Epoch 19/20
17/17 - 3s - loss: 0.0056
Epoch 20/20
17/17 - 3s - loss: 0.0052
```

ploting the training loss

```
plt.plot(hist.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



testing the model on new data

```
testData = pd.read_csv('Facebook_stock.csv') #importing the test data
testData["Close"]=pd.to_numeric(testData.Close,errors='coerce') #turning the close column to numerical type
testData = testData.dropna() #droping the NA values
testData = testData.iloc[:,4:5] #selecting the closing prices for testing
y_test = testData.iloc[60:,0:].values #selecting the labels
```

```
#input array for the model
inputClosing = testData.iloc[:,0:].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep,length): #doing the same preivous preprocessing
    X_test.append(inputClosing_scaled[i-timestep:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
X_test.shape
```

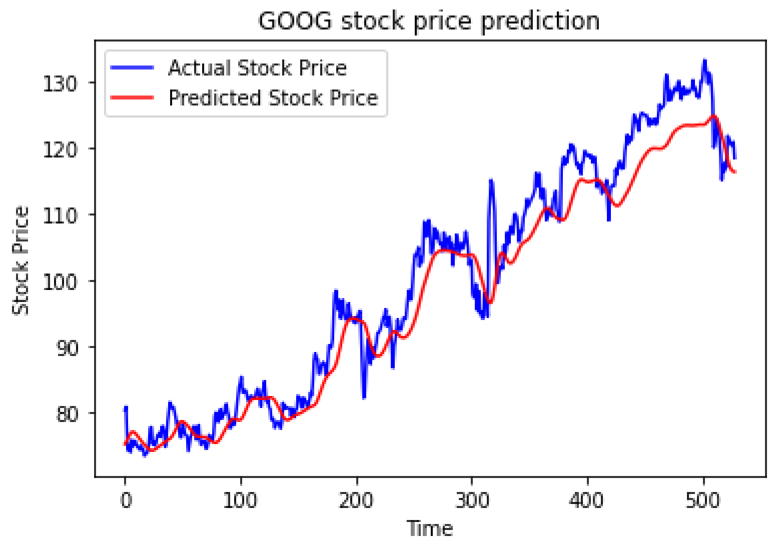
```
(529, 60, 1)
```

```
y_pred = model.predict(X_test) #predicting the new values
```

```
predicted_price = sc.inverse_transform(y_pred) #inversing the scaling transformation for ploting
```

ploting the results

```
plt.plot(y_test, color = 'blue', label = 'Actual Stock Price')
plt.plot(predicted_price, color = 'red', label = 'Predicted Stock Price')
plt.title('GOOG stock price prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```




```
#Calculation of MSE for comparing the model
difference_array = np.subtract(y_test, predicted_price)
squared_array = np.square(difference_array)
mse = squared_array.mean()
mse
```

```
20.561681311768684
```