CS575 Project_IBM dataset

Name- Vipin Gupta

Roll- 2011MT22

Downloading & Exporting the dataset

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

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

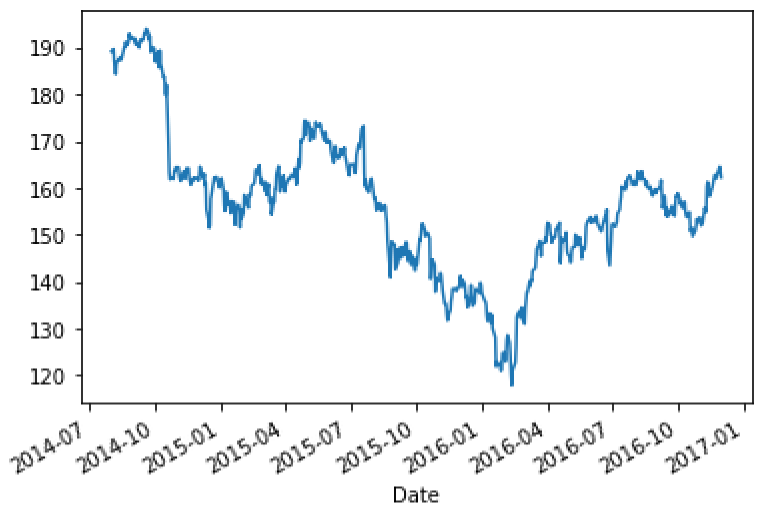
```
#print first few lines of data
print(ibm.head())
```

```
                  High         Low  ...     Volume   Adj Close
Date                                ...
2014-08-01  191.500000  188.860001  ...  5181100.0  143.561371
2014-08-04  189.949997  188.600006  ...  2125900.0  143.933304
2014-08-05  189.199997  186.440002  ...  3307900.0  142.005493
2014-08-06  186.880005  184.440002  ...  3847000.0  141.982544
2014-08-07  186.679993  183.580002  ...  2708600.0  140.707535

[5 rows x 6 columns]
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc694f68690>
```



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
```

```
#ADF Test

def adf_test(atr):

    #Perform Dickey-Fuller test:
    timeseries = ibm[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                  -2.279273
p-value                          0.178740
#Lags Used                       0.000000
Number of Observations Used    588.000000
Critical Value (1%)             -3.441520
Critical Value (5%)             -2.866468
Critical Value (10%)            -2.569394
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

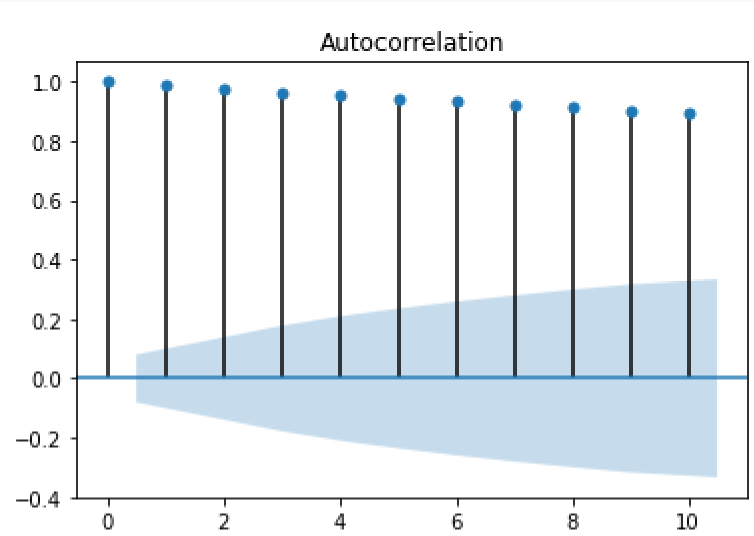
so we cant reject the null hypothesis so the time series is non stationary.

```
#KPSS Test

def kpss_test(atr):
    timeseries = ibm[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')
```

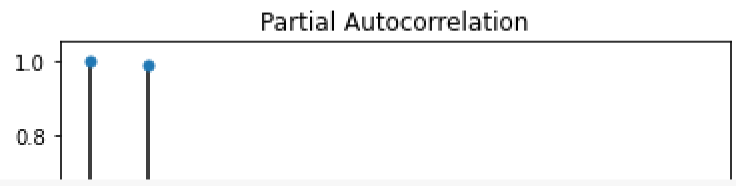
```
Results of KPSS Test for  Close
Test Statistic            1.268862
p-value                   0.010000
Lags Used                19.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py: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(ibm['Close'].dropna(), lags=10)
plt.show()
```

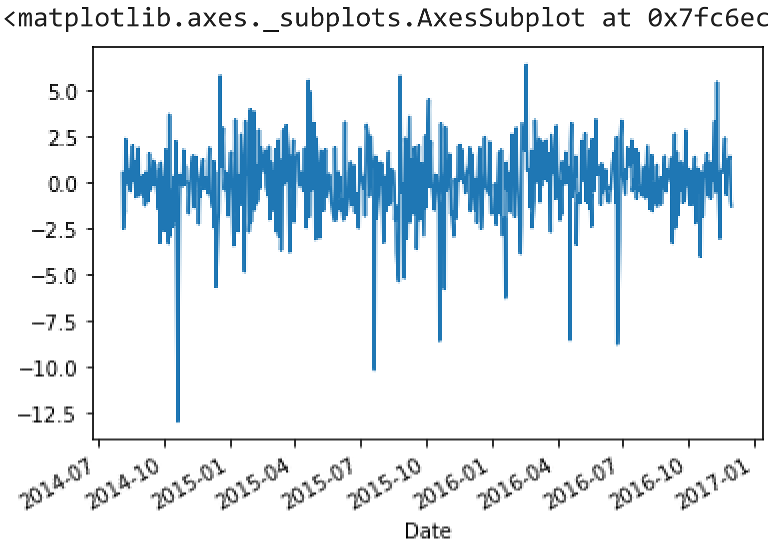



```
# PACF Test of differenced data
plot_pacf(ibm['Close'].dropna(), lags=10)
plt.show()
```




```
#Differencing to make data as stationary

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6ec551fd0>
```




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

```
Results of Dickey-Fuller Test for  diff

Test Statistic                 -1.843371e+01
p-value                         2.166547e-30
#Lags Used                      1.000000e+00
Number of Observations Used     5.860000e+02
Critical Value (1%)            -3.441558e+00
Critical Value (5%)            -2.866485e+00
Critical Value (10%)           -2.569403e+00
dtype: float64
```

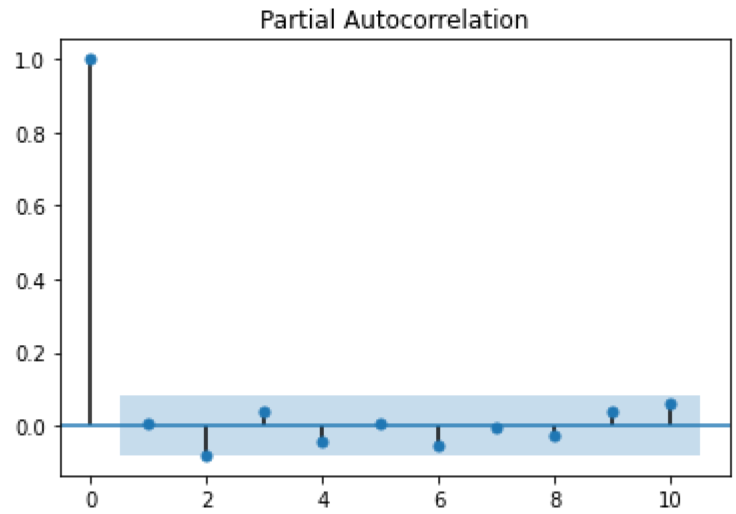
```
# KPSS Test of differenced data
kpss_test('diff')
```

```
Results of KPSS Test for  diff
Test Statistic            0.251866
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(ibm['diff'].dropna(), lags=10)
plt.show()
```

Autocorrelation

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



Exponential

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

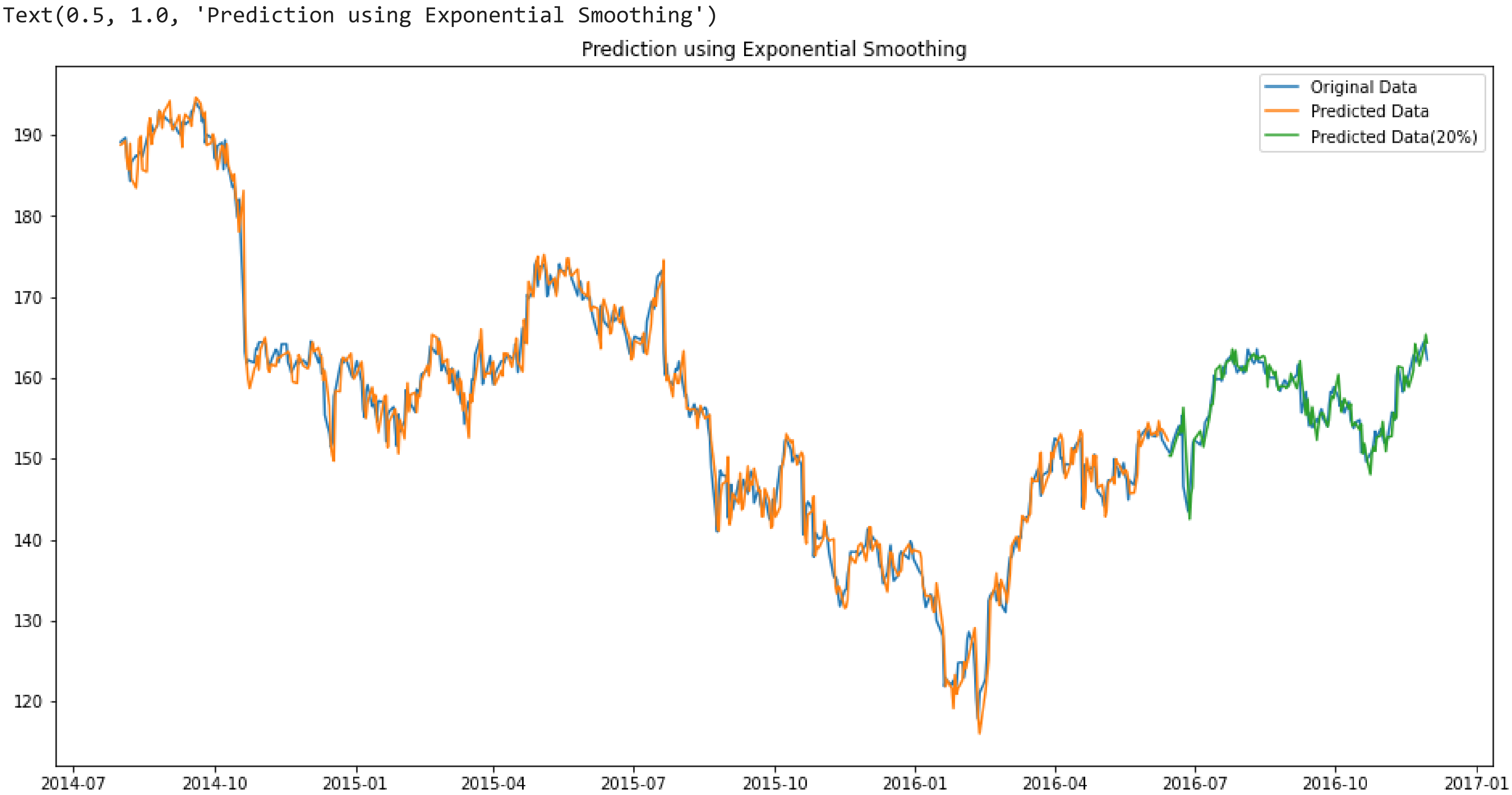
```
n = int(len(ibm["Close"])*0.8)
data = ibm['Close'].to_numpy()
train2 = data[:n]
test2 = data[n:]
date = (ibm.index)
```

```
Exp_model = ExponentialSmoothing(ibm.Close,trend='mul',seasonal='mul',seasonal_periods=4)
ibm['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],ibm.Pred_Exp[:n], label='Predicted Data')
plt.plot(date[n:],ibm.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(ibm.Pred_Exp[n:])),2)))
print('MSE value using Exponential Smoothing model: ',rmse2)
```

```
MSE value using Exponential Smoothing model:  3.4172334666694675
```

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('IBM_stock.csv')
df.head()
```

	Date	High	Low	Open	Close	Volume	Adj Close
0	2014-08-01	191.500000	188.860001	190.500000	189.149994	5181100.0	143.561371
1	2014-08-04	189.949997	188.600006	189.350006	189.639999	2125900.0	143.933304
2	2014-08-05	189.199997	186.440002	188.750000	187.100006	3307900.0	142.005493
3	2014-08-06	186.880005	184.440002	185.360001	185.970001	3847000.0	141.982544
4	2014-08-07	186.679993	183.580002	186.639999	184.300003	2708600.0	140.707535

```
# 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
```

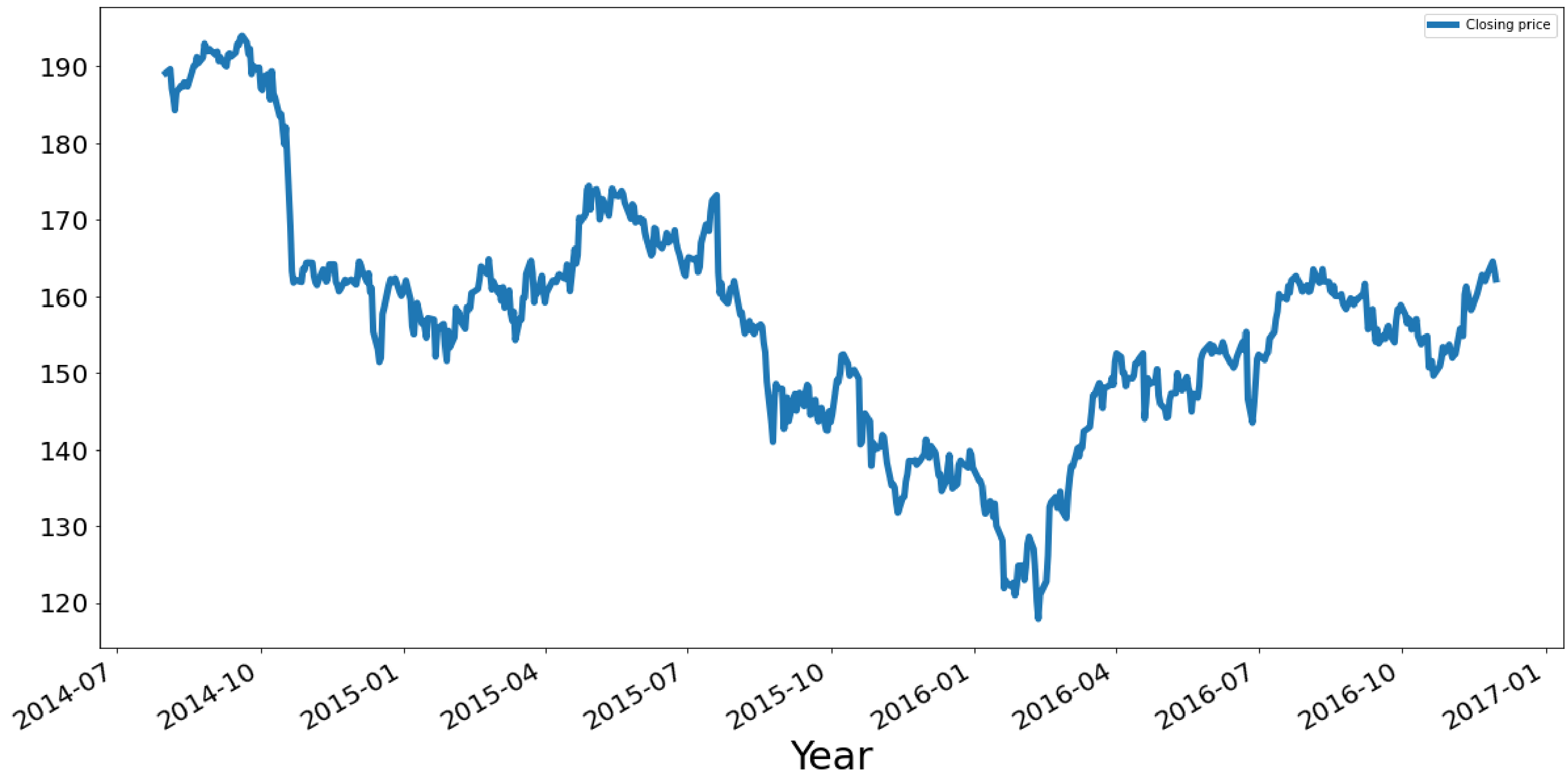
```
# Making Date column to be the index
df.columns=['Date','Closing price']
df.set_index('Date', inplace=True)
df.head()
```

Closing price

Date

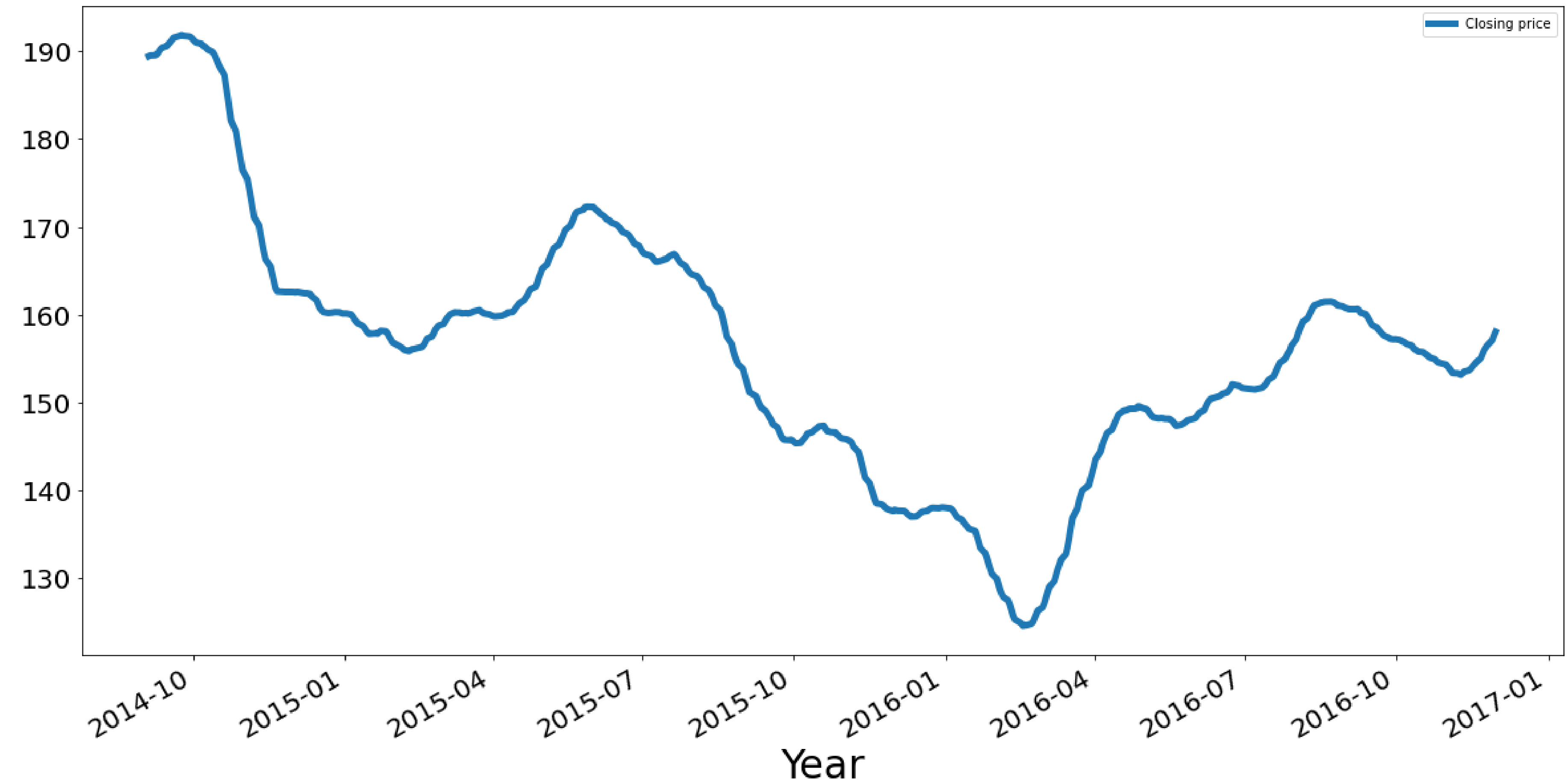
```
# Plot
df.plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
```

```
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
```

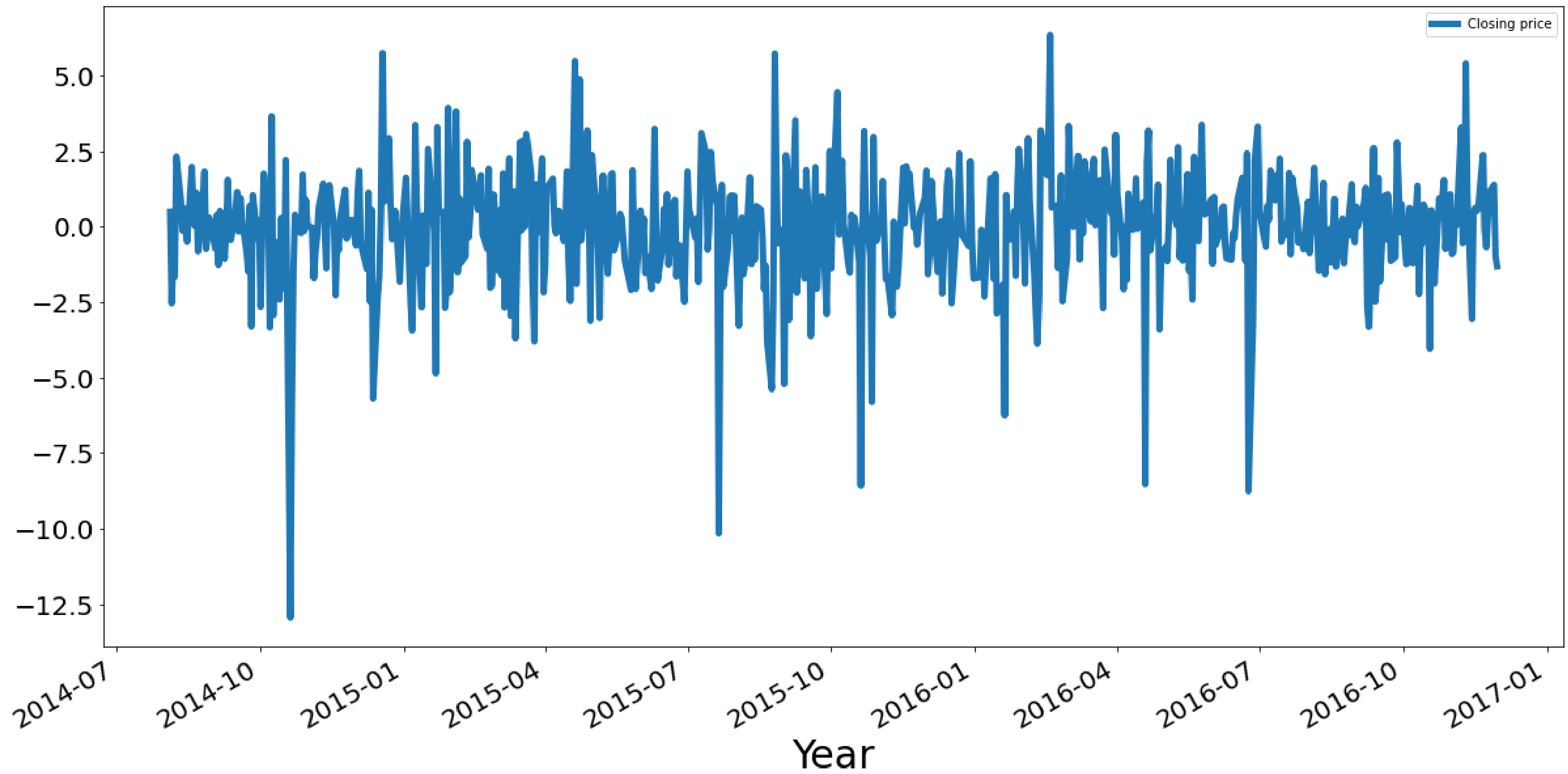
```
Text(0.5, 0, 'Year')
```




```
# We can see that there is no specific seasonality here
# Removing trend
df.diff().plot(figsize=(20,10), linewidth=5,fontsize=20);
```

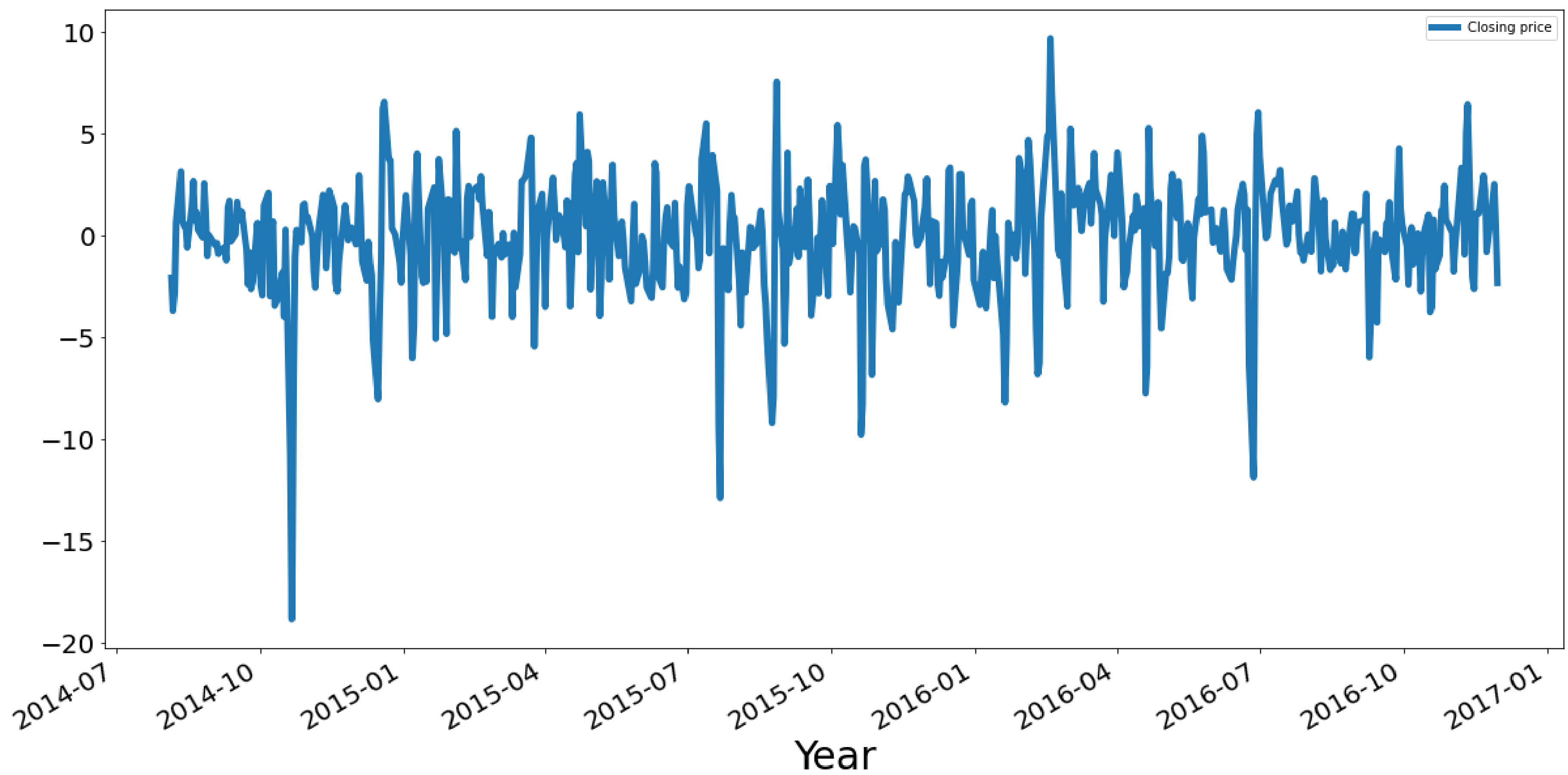
```
plt.xlabel('Year', fontsize=30)
```

Text(0.5, 0, 'Year')



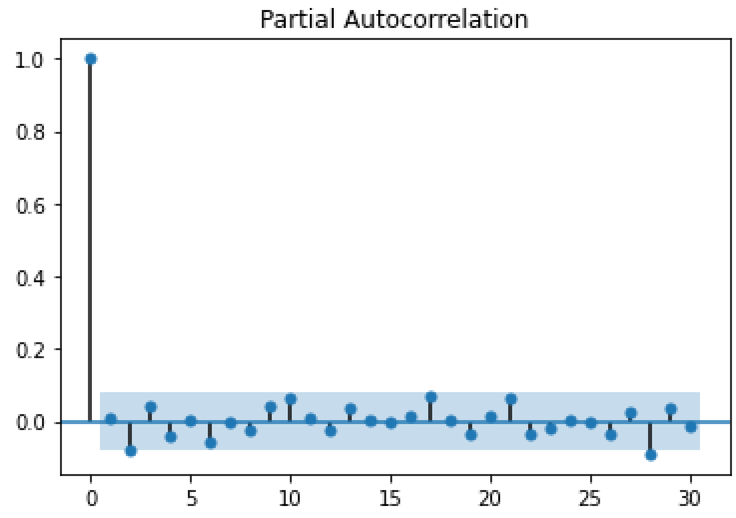
```
# 2nd order differencing
df.diff(periods=2).plot(figsize=(20,10), linewidth=5,fontsize=20);
plt.xlabel('Year', fontsize=30)
```

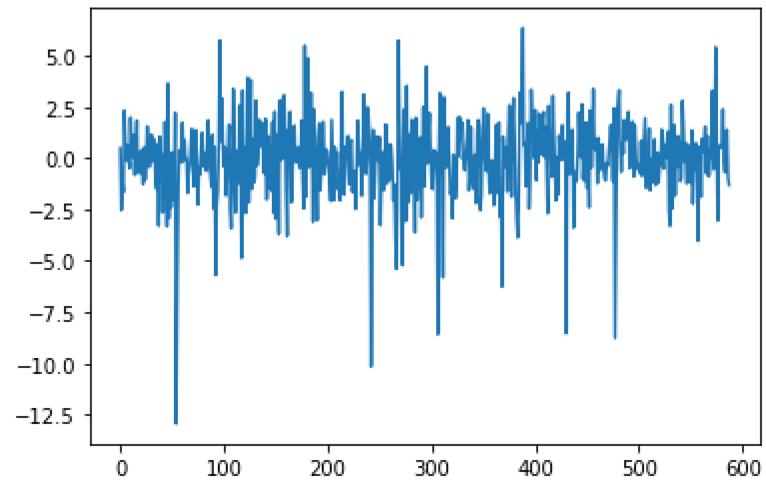
Text(0.5, 0, 'Year')

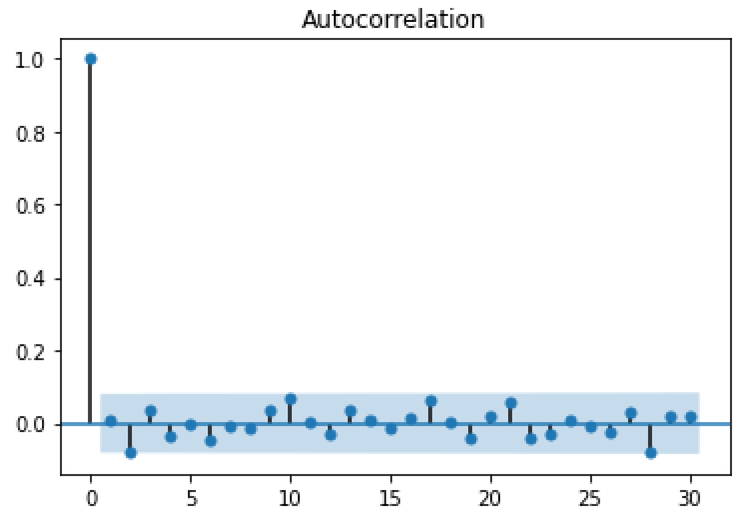


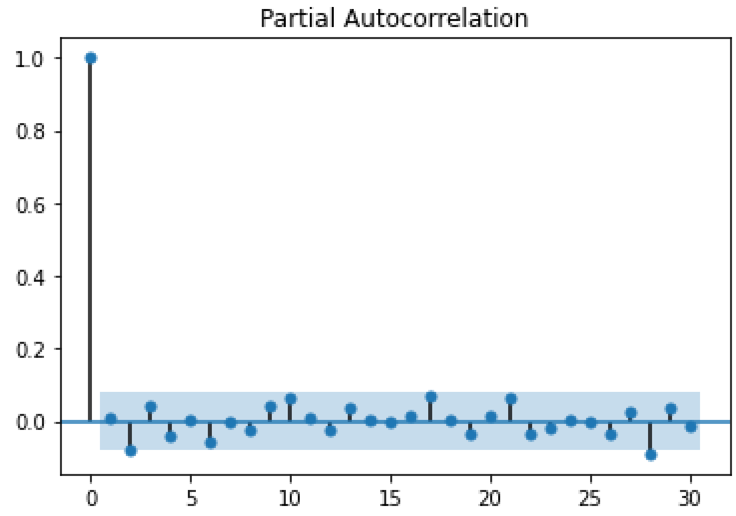
```
# Let's take a look at its auto-corelation plots
# Before that we'll have to do manual differencing
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

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

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









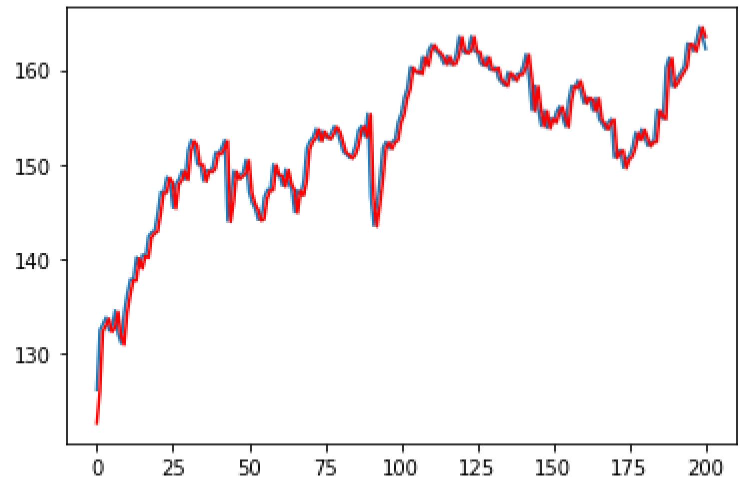

```
# Forecasting
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARIMA
df = df.astype(np.float64)
Y = df.values
size = int(len(Y)*0.66)
train, test = Y[0:size], Y[size:len(Y)]
```

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

for t in range(len(test)):
    model = ARIMA(history, order=(0,1,0))
    model_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=122.568396, expected=126.099998
predicted=125.937498, expected=132.449997
predicted=132.304239, expected=133.080002
predicted=132.936233, expected=133.770004
predicted=133.628367, expected=132.399994
predicted=132.255223, expected=132.800003
predicted=132.656619, expected=134.500000
predicted=134.361294, expected=132.029999
predicted=131.885391, expected=131.029999
predicted=130.883231, expected=134.369995
predicted=134.232010, expected=136.300003
predicted=136.167214, expected=137.800003
predicted=137.671306, expected=137.800003
predicted=137.671628, expected=140.149994
predicted=140.027799, expected=139.070007
predicted=138.945430, expected=140.410004
predicted=140.289061, expected=140.190002
predicted=140.068814, expected=142.360001
predicted=142.244470, expected=142.779999
predicted=142.665787, expected=142.960007
predicted=142.846518, expected=144.789993
predicted=144.681268, expected=147.039993
predicted=146.937035, expected=147.089996
predicted=146.987411, expected=148.630005
predicted=148.531416, expected=148.100006
predicted=148.000370, expected=145.399994
predicted=145.294062, expected=147.949997
predicted=147.850480, expected=148.399994
predicted=148.301801, expected=149.330002
predicted=149.234281, expected=148.410004
predicted=148.312306, expected=151.449997
predicted=151.359806, expected=152.520004
predicted=152.432582, expected=152.070007
predicted=151.981722, expected=150.000000
predicted=149.907007, expected=150.020004
predicted=149.927279, expected=148.250000
predicted=148.153310, expected=149.350006
predicted=149.256138, expected=149.250000
predicted=149.156118, expected=149.630005
predicted=149.537235, expected=151.229996
predicted=151.141190, expected=151.160004
predicted=151.071242, expected=151.720001
predicted=151.632752, expected=152.529999
predicted=152.444836, expected=144.000000
predicted=143.895244, expected=146.110001
predicted=146.010371, expected=149.300003
predicted=149.207971, expected=148.500000
predicted=148.406336, expected=148.809998
predicted=148.717262, expected=149.080002
predicted=148.988098, expected=150.470001
predicted=150.381489, expected=147.070007
predicted=146.973934, expected=145.940002
predicted=145.841574, expected=145.270004
predicted=145.170277, expected=144.130005
predicted=144.027919, expected=144.250000
predicted=144.148416, expected=146.470001
predicted=146.373658, expected=147.289993
predicted=147.195714, expected=147.339996
```

```
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
```

    3.1982815526323543

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('IBM_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))
```

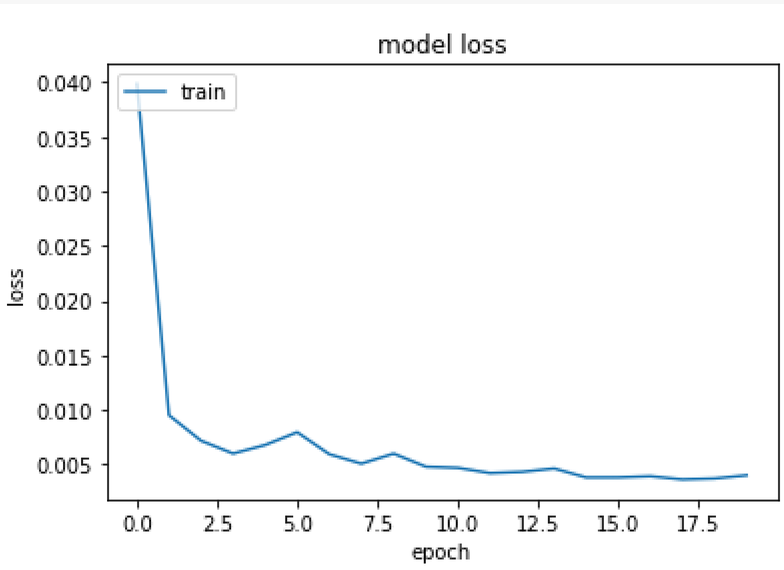
```
model.add(Dropout(0.2))

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 - 9s - loss: 0.0399
Epoch 2/20
17/17 - 3s - loss: 0.0095
Epoch 3/20
17/17 - 3s - loss: 0.0072
Epoch 4/20
17/17 - 3s - loss: 0.0060
Epoch 5/20
17/17 - 3s - loss: 0.0068
Epoch 6/20
17/17 - 3s - loss: 0.0079
Epoch 7/20
17/17 - 3s - loss: 0.0059
Epoch 8/20
17/17 - 3s - loss: 0.0051
Epoch 9/20
17/17 - 3s - loss: 0.0060
Epoch 10/20
17/17 - 3s - loss: 0.0048
Epoch 11/20
17/17 - 3s - loss: 0.0047
Epoch 12/20
17/17 - 3s - loss: 0.0042
Epoch 13/20
17/17 - 3s - loss: 0.0043
Epoch 14/20
17/17 - 3s - loss: 0.0046
Epoch 15/20
17/17 - 3s - loss: 0.0038
Epoch 16/20
17/17 - 3s - loss: 0.0038
Epoch 17/20
17/17 - 3s - loss: 0.0039
Epoch 18/20
17/17 - 3s - loss: 0.0036
Epoch 19/20
17/17 - 3s - loss: 0.0037
Epoch 20/20
17/17 - 3s - loss: 0.0040
```

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('IBM_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 = 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
```

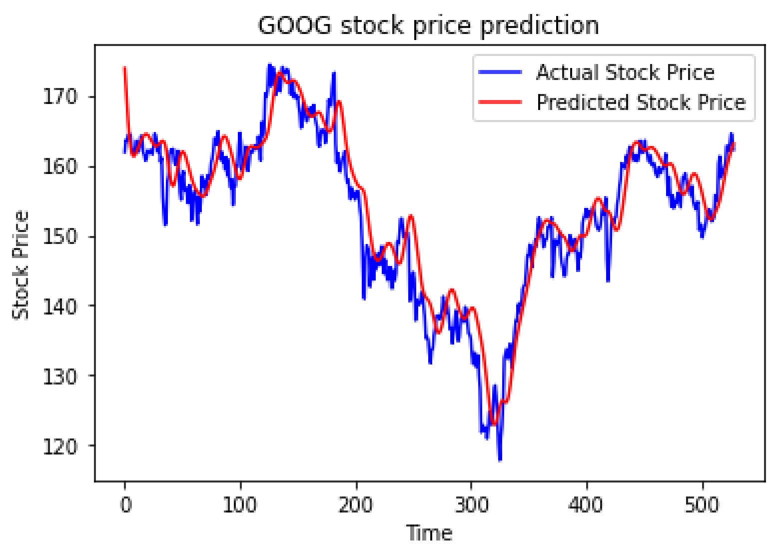
```
(529, 60, 1)
```

```
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
```

```
15.992904502728457
```