Stock Market Prediction HACKATHON

FTSE

In [1]:
```python
### Data Collection
import pandas_datareader as pdr
import pandas as pd
key=""
```

In [3]:
```python
df_nifty = pd.read_csv("C:\\Users\\Intel\\Downloads\\ash.csv")
```

In [4]:
```python
df_nifty
```

Out[4]:

	Date	Open Price	Close Price	High Price	Low Price	Volume
0	20-Nov-2017	7380.68	7389.46	7397.39	7350.37	582349888
1	21-Nov-2017	7389.46	7411.34	7422.18	7367.86	665962560
2	22-Nov-2017	7411.34	7419.02	7460.91	7410.38	802857600
3	23-Nov-2017	7419.02	7417.24	7422.73	7373.31	549163072
4	24-Nov-2017	7417.24	7409.64	7425.19	7389.54	490732544
...	...	...	...	...	...	...
1253	14-Nov-2022	7318.04	7385.17	7413.82	7317.57	716288320
1254	15-Nov-2022	7385.17	7369.44	7413.01	7344.85	929093824
1255	16-Nov-2022	7369.44	7351.19	7394.42	7341.59	763112256
1256	17-Nov-2022	7351.19	7346.54	7354.07	7294.75	751494464
1257	18-Nov-2022	7346.54	7385.52	7423.51	7343.50	843610752

1258 rows × 6 columns

In [5]:
```python
df = df_nifty
```

In [6]:
```python
df.head()
```

Out[6]:

	Date	Open Price	Close Price	High Price	Low Price	Volume
0	20-Nov-2017	7380.68	7389.46	7397.39	7350.37	582349888
1	21-Nov-2017	7389.46	7411.34	7422.18	7367.86	665962560
2	22-Nov-2017	7411.34	7419.02	7460.91	7410.38	802857600
3	23-Nov-2017	7419.02	7417.24	7422.73	7373.31	549163072
4	24-Nov-2017	7417.24	7409.64	7425.19	7389.54	490732544

In [7]:
```python
df.tail()
```

Out[7]:

	Date	Open Price	Close Price	High Price	Low Price	Volume
1253	14-Nov-2022	7318.04	7385.17	7413.82	7317.57	716288320
1254	15-Nov-2022	7385.17	7369.44	7413.01	7344.85	929093824
1255	16-Nov-2022	7369.44	7351.19	7394.42	7341.59	763112256
1256	17-Nov-2022	7351.19	7346.54	7354.07	7294.75	751494464
1257	18-Nov-2022	7346.54	7385.52	7423.51	7343.50	843610752

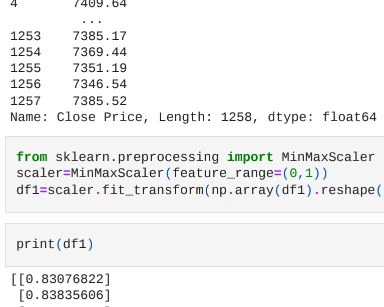
In [8]:
```python
df1=df.reset_index()["Close Price"]
```

In [9]:
```python
df1
```

Out[9]:
```
0       7389.46
1       7411.34
2       7419.02
3       7417.24
4       7409.64
          ...
1253    7385.17
1254    7369.44
1255    7351.19
1256    7346.54
1257    7385.52
Name: Close Price, Length: 1258, dtype: float64
```

In [10]:
```python
import matplotlib.pyplot as plt
plt.plot(df1)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x23b85ec18a0>]
```



In [11]:
```python
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler
```

In [12]:
```python
import numpy as np
```

In [13]:
```python
df1
```

Out[13]:
```
0       7389.46
1       7411.34
2       7419.02
3       7417.24
4       7409.64
          ...
1253    7385.17
1254    7369.44
1255    7351.19
1256    7346.54
1257    7385.52
Name: Close Price, Length: 1258, dtype: float64
```

In [14]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [15]:
```python
print(df1)
```

```
[[0.83076822]
 [0.83835606]
 [0.84101943]
 ...
 [0.81749643]
 [0.81588384]
 [0.82940185]]
```

In [16]:
```python
df1
```

Out[16]:
```
array([[0.83076822],
       [0.83835606],
       [0.84101943],
       ...,
       [0.81749643],
       [0.81588384],
       [0.82940185]])
```

In [17]:
```python
import numpy
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-time_step-1):
                a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                dataX.append(a)
                dataY.append(dataset[i + time_step, 0])
        return numpy.array(dataX), numpy.array(dataY)
```

In [18]:
```python
time_step = 50
X_train, y_train = create_dataset(df1, time_step)
```

In [19]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [20]:
```python
import tensorflow as tf
```

In [21]:
```python
import math
from sklearn.metrics import mean_squared_error
```

In [22]:
```python
print(X_train.shape)
```

```
(1207, 50)
```

In [23]:
```python
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
```

In [24]:
```python
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [25]:
```python
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(50,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [26]:
```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 50)            10400

 lstm_1 (LSTM)               (None, 50, 50)            20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [27]:
```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 50)            10400

 lstm_1 (LSTM)               (None, 50, 50)            20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

In [28]:
```python
model.fit(X_train,y_train,epochs=95,batch_size=64,verbose=1)
```

```
Epoch 1/95
19/19 [==============================] - 6s 62ms/step - loss: 0.1269
Epoch 2/95
19/19 [==============================] - 1s 56ms/step - loss: 0.0127
Epoch 3/95
19/19 [==============================] - 1s 74ms/step - loss: 0.0081
Epoch 4/95
19/19 [==============================] - 2s 101ms/step - loss: 0.0065
Epoch 5/95
19/19 [==============================] - 1s 53ms/step - loss: 0.0056
Epoch 6/95
19/19 [==============================] - 1s 56ms/step - loss: 0.0051
Epoch 7/95
19/19 [==============================] - 2s 118ms/step - loss: 0.0049
Epoch 8/95
19/19 [==============================] - 2s 81ms/step - loss: 0.0048
Epoch 9/95
19/19 [==============================] - 1s 52ms/step - loss: 0.0046
Epoch 10/95
19/19 [==============================] - 1s 72ms/step - loss: 0.0045
Epoch 11/95
19/19 [==============================] - 2s 118ms/step - loss: 0.0043
Epoch 12/95
19/19 [==============================] - 1s 66ms/step - loss: 0.0039
Epoch 13/95
19/19 [==============================] - 1s 52ms/step - loss: 0.0038
Epoch 14/95
19/19 [==============================] - 2s 97ms/step - loss: 0.0036
Epoch 15/95
19/19 [==============================] - 2s 116ms/step - loss: 0.0033
Epoch 16/95
19/19 [==============================] - 1s 55ms/step - loss: 0.0036
Epoch 17/95
19/19 [==============================] - 2s 120ms/step - loss: 0.0032
Epoch 18/95
19/19 [==============================] - 2s 83ms/step - loss: 0.0031
Epoch 19/95
19/19 [==============================] - 2s 82ms/step - loss: 0.0029
Epoch 20/95
19/19 [==============================] - 2s 118ms/step - loss: 0.0030
Epoch 21/95
19/19 [==============================] - 1s 55ms/step - loss: 0.0030
Epoch 22/95
19/19 [==============================] - 2s 107ms/step - loss: 0.0027
Epoch 23/95
19/19 [==============================] - 2s 100ms/step - loss: 0.0027
Epoch 24/95
19/19 [==============================] - 1s 63ms/step - loss: 0.0025
Epoch 25/95
19/19 [==============================] - 2s 116ms/step - loss: 0.0024
Epoch 26/95
19/19 [==============================] - 1s 75ms/step - loss: 0.0026
Epoch 27/95
19/19 [==============================] - 2s 86ms/step - loss: 0.0025
Epoch 28/95
19/19 [==============================] - 2s 119ms/step - loss: 0.0026
Epoch 29/95
19/19 [==============================] - 1s 54ms/step - loss: 0.0026
Epoch 30/95
19/19 [==============================] - 2s 112ms/step - loss: 0.0022
Epoch 31/95
19/19 [==============================] - 2s 93ms/step - loss: 0.0022
Epoch 32/95
19/19 [==============================] - 1s 73ms/step - loss: 0.0022
Epoch 33/95
19/19 [==============================] - 2s 118ms/step - loss: 0.0022
Epoch 34/95
19/19 [==============================] - 1s 66ms/step - loss: 0.0020
Epoch 35/95
19/19 [==============================] - 2s 90ms/step - loss: 0.0020
Epoch 36/95
19/19 [==============================] - 2s 113ms/step - loss: 0.0019
Epoch 37/95
19/19 [==============================] - 1s 52ms/step - loss: 0.0022
Epoch 38/95
19/19 [==============================] - 2s 110ms/step - loss: 0.0020
Epoch 39/95
19/19 [==============================] - 2s 94ms/step - loss: 0.0019
Epoch 40/95
19/19 [==============================] - 1s 61ms/step - loss: 0.0020
Epoch 41/95
19/19 [==============================] - 2s 115ms/step - loss: 0.0020
Epoch 42/95
19/19 [==============================] - 1s 74ms/step - loss: 0.0019
Epoch 43/95
19/19 [==============================] - 2s 81ms/step - loss: 0.0019
Epoch 44/95
19/19 [==============================] - 2s 117ms/step - loss: 0.0018
Epoch 45/95
19/19 [==============================] - 1s 54ms/step - loss: 0.0018
Epoch 46/95
19/19 [==============================] - 2s 102ms/step - loss: 0.0017
Epoch 47/95
19/19 [==============================] - 2s 102ms/step - loss: 0.0016
Epoch 48/95
19/19 [==============================] - 1s 54ms/step - loss: 0.0015
Epoch 49/95
19/19 [==============================] - 2s 114ms/step - loss: 0.0016
Epoch 50/95
19/19 [==============================] - 2s 83ms/step - loss: 0.0015
Epoch 51/95
19/19 [==============================] - 1s 74ms/step - loss: 0.0016
Epoch 52/95
19/19 [==============================] - 2s 117ms/step - loss: 0.0014
Epoch 53/95
19/19 [==============================] - 1s 62ms/step - loss: 0.0014
Epoch 54/95
19/19 [==============================] - 2s 110ms/step - loss: 0.0014
Epoch 55/95
19/19 [==============================] - 2s 108ms/step - loss: 0.0015
Epoch 56/95
19/19 [==============================] - 2s 81ms/step - loss: 0.0015
Epoch 57/95
19/19 [==============================] - 2s 128ms/step - loss: 0.0014
Epoch 58/95
19/19 [==============================] - 1s 56ms/step - loss: 0.0014
Epoch 59/95
19/19 [==============================] - 2s 106ms/step - loss: 0.0014
Epoch 60/95
19/19 [==============================] - 2s 101ms/step - loss: 0.0013
Epoch 61/95
19/19 [==============================] - 1s 62ms/step - loss: 0.0013
Epoch 62/95
19/19 [==============================] - 2s 117ms/step - loss: 0.0013
Epoch 63/95
19/19 [==============================] - 1s 75ms/step - loss: 0.0014
Epoch 64/95
19/19 [==============================] - 2s 87ms/step - loss: 0.0013
Epoch 65/95
19/19 [==============================] - 2s 124ms/step - loss: 0.0012
Epoch 66/95
19/19 [==============================] - 1s 55ms/step - loss: 0.0012
Epoch 67/95
19/19 [==============================] - 2s 125ms/step - loss: 0.0011
Epoch 68/95
19/19 [==============================] - 2s 120ms/step - loss: 0.0011
Epoch 69/95
19/19 [==============================] - 1s 57ms/step - loss: 0.0012
Epoch 70/95
19/19 [==============================] - 2s 126ms/step - loss: 0.0011
Epoch 71/95
19/19 [==============================] - 2s 121ms/step - loss: 0.0010
Epoch 72/95
19/19 [==============================] - 1s 56ms/step - loss: 0.0010
Epoch 73/95
19/19 [==============================] - 2s 120ms/step - loss: 0.0012
Epoch 74/95
19/19 [==============================] - 2s 82ms/step - loss: 0.0011
Epoch 75/95
19/19 [==============================] - 2s 87ms/step - loss: 9.6764e-04
Epoch 76/95
19/19 [==============================] - 2s 123ms/step - loss: 0.0011
Epoch 77/95
19/19 [==============================] - 2s 85ms/step - loss: 0.0010
Epoch 78/95
19/19 [==============================] - 1s 79ms/step - loss: 0.0011
Epoch 79/95
19/19 [==============================] - 2s 118ms/step - loss: 0.0010
Epoch 80/95
19/19 [==============================] - 1s 59ms/step - loss: 0.0011
Epoch 81/95
19/19 [==============================] - 2s 103ms/step - loss: 9.8395e-04
Epoch 82/95
19/19 [==============================] - 2s 119ms/step - loss: 9.0263e-04
Epoch 83/95
19/19 [==============================] - 1s 69ms/step - loss: 8.8888e-04
Epoch 84/95
19/19 [==============================] - 2s 93ms/step - loss: 0.0012
Epoch 85/95
19/19 [==============================] - 2s 113ms/step - loss: 0.0012
Epoch 86/95
19/19 [==============================] - 1s 52ms/step - loss: 9.8113e-04
Epoch 87/95
19/19 [==============================] - 2s 116ms/step - loss: 9.6233e-04
Epoch 88/95
19/19 [==============================] - 2s 118ms/step - loss: 8.8754e-04
Epoch 89/95
19/19 [==============================] - 1s 57ms/step - loss: 8.8490e-04
Epoch 90/95
19/19 [==============================] - 2s 105ms/step - loss: 0.0012
Epoch 91/95
19/19 [==============================] - 2s 117ms/step - loss: 9.5200e-04
Epoch 92/95
19/19 [==============================] - 1s 67ms/step - loss: 9.1483e-04
Epoch 93/95
19/19 [==============================] - 2s 95ms/step - loss: 8.4803e-04
Epoch 94/95
19/19 [==============================] - 2s 118ms/step - loss: 8.7411e-04
Epoch 95/95
19/19 [==============================] - 2s 81ms/step - loss: 9.0721e-04
```

Out[28]:
```
<keras.callbacks.History at 0x23ba437ed40>
```

In [29]:
```python
import tensorflow as tf
```

In [30]:
```python
tf.__version__
```

Out[30]:
```
'2.10.0'
```

In [31]:
```python
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
```

```
38/38 [==============================] - 2s 20ms/step
```

In [32]:
```python
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
```

In [33]:
```python
### Calculate RMSE performance metrics
import math
ytrain  = scaler.inverse_transform(y_train.reshape(-1,1))
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(ytrain,train_predict))
```

Out[33]:
```
85.03587725450421
```

In [ ]:

In [34]:
```python
len(df1)
```

Out[34]:
```
1258
```

In [35]:
```python
x_input=df1[len(df1)-50:].reshape(1,-1)
x_input.shape
```

Out[35]:
```
(1, 50)
```

In [ ]:

In [ ]:

In [36]:
```python
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [37]:
```python
# demonstrate prediction for next 10 days
from numpy import array

lst_output=[]
n_steps=50
i=0
while(i<5):

    if(len(temp_input)>50):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

```
[0.81463796]
51
1 day input [0.85974975 0.82951976 0.79187185 0.79352606 0.7777851  0.76251925
 0.77811802 0.75102651 0.70215636 0.70297133 0.69036191 0.69757522
 0.65464218 0.65888    0.66406456 0.72568977 0.71395428 0.69475926
 0.69261607 0.68159497 0.65590451 0.63541594 0.6437806  0.64673529
 0.66804575 0.67376784 0.66969302 0.67625435 0.68520856 0.70055764
 0.70038078 0.71515072 0.72126122 0.71223765 0.7284884  0.76026509
 0.74569282 0.76112167 0.81182635 0.7997406  0.80187338 0.79844359
 0.82587149 0.80600022 0.82928047 0.82382541 0.81749643 0.81588384
 0.82940185 0.81463796]
1 day output [[0.8132578]]
2 day input [0.82951976 0.79187185 0.79352606 0.7777851  0.76251925 0.77811802
 0.75102651 0.70215636 0.70297133 0.69036191 0.69757522 0.65464218
 0.65888    0.66406456 0.72568977 0.71395428 0.69475926 0.69261607
 0.68159497 0.65590451 0.63541594 0.6437806  0.64673529 0.66804575
 0.67376784 0.66969302 0.67625435 0.68520856 0.70055764 0.70038078
 0.71515072 0.72126122 0.71223765 0.7284884  0.76026509 0.74569282
 0.76112167 0.81182635 0.7997406  0.80187338 0.79844359 0.82587149
 0.80600022 0.82928047 0.82382541 0.81749643 0.81588384 0.82940185
 0.81463796 0.81325781]
2 day output [[0.80664396]]
3 day input [0.79187185 0.79352606 0.7777851  0.76251925 0.77811802 0.75102651
 0.70215636 0.70297133 0.69036191 0.69757522 0.65464218 0.65888
 0.66406456 0.72568977 0.71395428 0.69475926 0.69261607 0.68159497
 0.65590451 0.63541594 0.6437806  0.64673529 0.66804575 0.67376784
 0.66969302 0.67625435 0.68520856 0.70055764 0.70038078 0.71515072
 0.72126122 0.71223765 0.7284884  0.76026509 0.74569282 0.76112167
 0.81182635 0.7997406  0.80187338 0.79844359 0.82587149 0.80600022
 0.82928047 0.82382541 0.81749643 0.81588384 0.82940185 0.81463796
 0.81325781 0.80664396]
3 day output [[0.8000198]]
4 day input [0.79352606 0.7777851  0.76251925 0.77811802 0.75102651 0.70215636
 0.70297133 0.69036191 0.69757522 0.65464218 0.65888    0.66406456
 0.72568977 0.71395428 0.69475926 0.69261607 0.68159497 0.65590451
 0.63541594 0.6437806  0.64673529 0.66804575 0.67376784 0.66969302
 0.67625435 0.68520856 0.70055764 0.70038078 0.71515072 0.72126122
 0.71223765 0.7284884  0.76026509 0.74569282 0.76112167 0.81182635
 0.7997406  0.80187338 0.79844359 0.82587149 0.80600022 0.82928047
 0.82382541 0.81749643 0.81588384 0.82940185 0.81463796 0.81325781
 0.80664396 0.8000198 ]
4 day output [[0.79479563]]
[[0.8146379590034485], [0.8132578134536743], [0.8066439628601074], [0.8000198006629944], [0.7947956323623657]]
```

In [38]:
```python
future_5 = scaler.inverse_transform(lst_output)
future_5
```

Out[38]:
```
array([[7342.94743306],
       [7338.96770056],
       [7319.89626554],
       [7300.7950964 ],
       [7285.73089365]])
```

In [39]:
```python
future_5_upper = future_5 + 0.05*future_5
future_5_upper
```

Out[39]:
```
array([[7710.09480472],
       [7705.91608559],
       [7685.89107882],
       [7665.83485122],
       [7650.01743834]])
```

In [40]:
```python
future_5_lower = future_5 - 0.05*future_5
future_5_lower
```

Out[40]:
```
array([[6975.80006141],
       [6972.01931553],
       [6953.90145227],
       [6935.75534158],
       [6921.44434897]])
```

In [41]:
```python
future_ftse = pd.DataFrame(future_5,columns=['future_ftse'])
future_ftse['future_upper'] = future_5_upper
future_ftse['future_lower'] = future_5_lower
future_ftse
```

Out[41]:

	future_ftse	future_upper	future_lower
0	7342.947433	7710.094805	6975.800061
1	7338.967701	7705.916086	6972.019316
2	7319.896266	7685.891079	6953.901452
3	7300.795096	7665.834851	6935.755342
4	7285.730894	7650.017438	6921.444349

In [160…
```python
future_ftse.to_csv('stock_pred_ftse.csv')
```

In [ ]:

In [ ]:

In [1]:

```python
import pandas_datareader as pdr
import pandas as pd
import datetime as datetime
```

In [2]:

```python
fulldf = pdr.get_data_yahoo('^NDX')
```

In [3]:

```python
df_nifty = fulldf
df_nifty
```

Out[3]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2017-11-20	6324.589844	6301.879883	6319.569824	6308.609863	1811750000	6308.609863
2017-11-21	6380.069824	6336.259766	6336.910156	6378.629883	1891960000	6378.629883
2017-11-22	6391.160156	6371.750000	6384.129883	6386.120117	1589150000	6386.120117
2017-11-24	6410.770020	6389.399902	6393.330078	6409.290039	872110000	6409.290039
2017-11-27	6420.209961	6392.009766	6409.520020	6405.970215	1781550000	6405.970215
...	...	...	...	...	...	...
2022-11-14	11863.820312	11669.099609	11728.110352	11700.940430	5004060000	11700.940430
2022-11-15	12024.950195	11735.400391	12006.450195	11871.150391	5617310000	11871.150391
2022-11-16	11796.990234	11673.139648	11767.419922	11699.089844	4585190000	11699.089844
2022-11-17	11737.610352	11519.379883	11521.639648	11676.860352	4354360000	11676.860352
2022-11-18	11794.679688	11579.639648	11791.849609	11677.019531	4175420000	11677.019531

1259 rows × 6 columns

In [4]:

```python
from statsmodels.tsa.stattools import adfuller
```

In [5]:

```python
close = df_nifty['Close']
result = adfuller(close)
print(result[1])
```

```
0.6264015951871802
```

In [6]:

```python
df = df_nifty
```

In [7]:

```python
df['Close_1'] = df['Close']-df['Close'].shift(1)
```

In [8]:

```python
df
```

Out[8]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2017-11-20	6324.589844	6301.879883	6319.569824	6308.609863	1811750000	6308.609863	NaN
2017-11-21	6380.069824	6336.259766	6336.910156	6378.629883	1891960000	6378.629883	70.020020
2017-11-22	6391.160156	6371.750000	6384.129883	6386.120117	1589150000	6386.120117	7.490234
2017-11-24	6410.770020	6389.399902	6393.330078	6409.290039	872110000	6409.290039	23.169922
2017-11-27	6420.209961	6392.009766	6409.520020	6405.970215	1781550000	6405.970215	-3.319824
...	...	...	...	...	...	...	...
2022-11-14	11863.820312	11669.099609	11728.110352	11700.940430	5004060000	11700.940430	-116.069336
2022-11-15	12024.950195	11735.400391	12006.450195	11871.150391	5617310000	11871.150391	170.209961
2022-11-16	11796.990234	11673.139648	11767.419922	11699.089844	4585190000	11699.089844	-172.060547
2022-11-17	11737.610352	11519.379883	11521.639648	11676.860352	4354360000	11676.860352	-22.229492
2022-11-18	11794.679688	11579.639648	11791.849609	11677.019531	4175420000	11677.019531	0.159180

1259 rows × 7 columns

In [9]:

```python
close_1 = df['Close_1'].dropna()
result_1 = adfuller(close_1)
print(result_1[1])
```

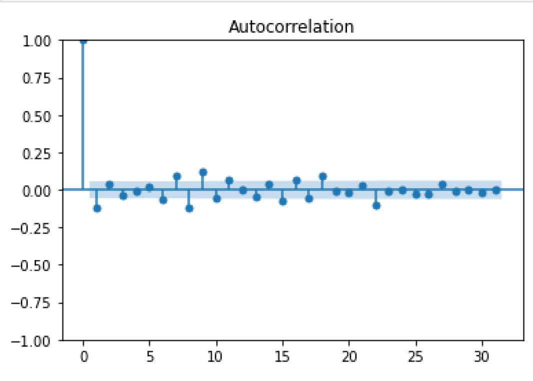
```
6.4094648027209035e-18
```

In [10]:

```python
from statsmodels.graphics.tsaplots import  plot_acf,plot_pacf
```

In [11]:

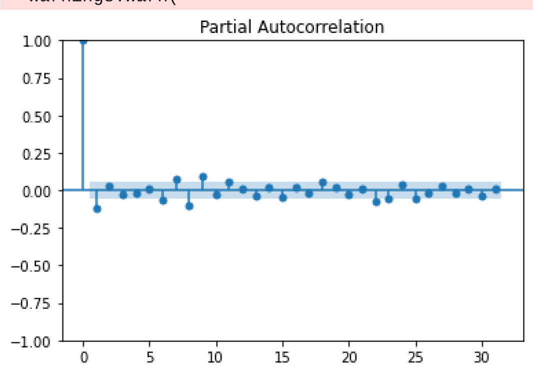
```python
acf = plot_acf(df['Close_1'].dropna());
```



In [12]:

```python
pacf = plot_pacf(df['Close_1'].dropna());
```

```
C:\Users\Intel\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\graphics\tsaplots.py:348: FutureWarning: The default method 'yw' can produce PACF values outsi
de of the [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



In [13]:

```python
df.shape
```

Out[13]:

```
(1259, 7)
```

In [14]:

```python
## test train split
train_data = df[:-100]
train_data
```

Out[14]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2017-11-20	6324.589844	6301.879883	6319.569824	6308.609863	1811750000	6308.609863	NaN
2017-11-21	6380.069824	6336.259766	6336.910156	6378.629883	1891960000	6378.629883	70.020020
2017-11-22	6391.160156	6371.750000	6384.129883	6386.120117	1589150000	6386.120117	7.490234
2017-11-24	6410.770020	6389.399902	6393.330078	6409.290039	872110000	6409.290039	23.169922
2017-11-27	6420.209961	6392.009766	6409.520020	6405.970215	1781550000	6405.970215	-3.319824
...	...	...	...	...	...	...	...
2022-06-23	11729.620117	11515.480469	11622.089844	11697.679688	5258200000	11697.679688	169.969727
2022-06-24	12112.820312	11812.179688	11826.950195	12105.849609	9468130000	12105.849609	408.169922
2022-06-27	12175.980469	11965.669922	12157.929688	12008.240234	5047860000	12008.240234	-97.609375
2022-06-28	12133.870117	11633.129883	12021.339844	11637.769531	5433220000	11637.769531	-370.470703
2022-06-29	11710.209961	11537.719727	11619.009766	11658.259766	5650860000	11658.259766	20.490234

1159 rows × 7 columns

In [15]:

```python
test_data=df[-100:]
test_data
```

Out[15]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2022-06-30	11650.959961	11322.860352	11532.320312	11503.719727	5654520000	11503.719727	-154.540039
2022-07-01	11592.849609	11378.629883	11472.629883	11585.679688	4865730000	11585.679688	81.959961
2022-07-05	11781.740234	11366.070312	11419.339844	11779.900391	5053570000	11779.900391	194.220703
2022-07-06	11941.309570	11727.360352	11807.080078	11852.589844	4851170000	11852.589844	72.689453
2022-07-07	12137.709961	11897.509766	11913.730469	12109.049805	4685050000	12109.049805	256.459961
...	...	...	...	...	...	...	...
2022-11-14	11863.820312	11669.099609	11728.110352	11700.940430	5004060000	11700.940430	-116.069336
2022-11-15	12024.950195	11735.400391	12006.450195	11871.150391	5617310000	11871.150391	170.209961
2022-11-16	11796.990234	11673.139648	11767.419922	11699.089844	4585190000	11699.089844	-172.060547
2022-11-17	11737.610352	11519.379883	11521.639648	11676.860352	4354360000	11676.860352	-22.229492
2022-11-18	11794.679688	11579.639648	11791.849609	11677.019531	4175420000	11677.019531	0.159180

100 rows × 7 columns

In [16]:

```python
training_set = close[:-100]
testing_set  = close[-100:]
```

In [17]:

```python
testing_set
```

Out[17]:

```
Date
2022-06-30    11503.719727
2022-07-01    11585.679688
2022-07-05    11779.900391
2022-07-06    11852.589844
2022-07-07    12109.049805
                  ...
2022-11-14    11700.940430
2022-11-15    11871.150391
2022-11-16    11699.089844
2022-11-17    11676.860352
2022-11-18    11677.019531
Name: Close, Length: 100, dtype: float64
```

In [18]:

```python
from pmdarima import auto_arima

import warnings
warnings.filterwarnings("ignore")

stepwise_fit = auto_arima(close,
                          trace = True,
                          error_action ='ignore',
                          suppress_warnings = True)

# To print the summary
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=16530.924, Time=1.30 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=16541.812, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=16526.539, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=16527.516, Time=0.20 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=16540.577, Time=0.02 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=16527.645, Time=0.09 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=16527.438, Time=0.20 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=16524.668, Time=0.60 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=16530.384, Time=0.83 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=16529.447, Time=0.27 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=16528.847, Time=0.11 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=16532.658, Time=0.50 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=16523.552, Time=0.16 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=16526.322, Time=0.09 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=16526.562, Time=0.05 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=16529.405, Time=0.23 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=16529.953, Time=0.31 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=16525.508, Time=0.03 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=16528.335, Time=0.14 sec
 ARIMA(3,1,0)(0,0,0)[0]             : AIC=16527.813, Time=0.06 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=16531.407, Time=0.42 sec

Best model:  ARIMA(2,1,1)(0,0,0)[0]
Total fit time: 5.705 seconds
```

Out[18]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1259
Model:	SARIMAX(2, 1, 1)	**Log Likelihood**	-8257.776
Date:	Sun, 20 Nov 2022	**AIC**	16523.552
Time:	22:07:11	**BIC**	16544.101
Sample:	0	**HQIC**	16531.275
	- 1259		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.0361	0.054	-19.087	0.000	-1.143	-0.930
ar.L2	-0.1334	0.019	-7.190	0.000	-0.170	-0.097
ma.L1	0.9214	0.052	17.560	0.000	0.819	1.024
sigma2	2.962e+04	801.475	36.954	0.000	2.8e+04	3.12e+04

Ljung-Box (L1) (Q):	0.06	**Jarque-Bera (JB):**	423.33
Prob(Q):	0.81	**Prob(JB):**	0.00
Heteroskedasticity (H):	6.50	**Skew:**	-0.52
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.65

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

In [47]:

```python
import itertools
p=range(0,8)
q=range(0,8)
d=range(0,3)
pdq_com = list(itertools.product(p,d,q))
len(pdq_com)
```

Out[47]:

```
192
```

In [20]:

```python
import statsmodels.api as sm
```

In [21]:

```python
rmse=[]
orderl=[]
```

In [22]:

```python
model = sm.tsa.arima.ARIMA(training_set, order=(0,1,0)).fit()
pred = model.predict(start = len(training_set),end= len(training_set) + len(testing_set)-1,dynamic=False)
err = math.sqrt(mean_squared_error(testing_set, pred))
err
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [22], in <module>
      1 model = sm.tsa.arima.ARIMA(training_set, order=(0,1,0)).fit()
      2 pred = model.predict(start = len(training_set),end= len(training_set) + len(testing_set)-1,dynamic=False)
----> 3 err = math.sqrt(mean_squared_error(testing_set, pred))
      4 err

NameError: name 'math' is not defined
```

In [217…

```python
for pdq in pdq_com:
        model = sm.tsa.arima.ARIMA(training_set, order=pdq).fit()
        pred = model.predict(start = len(training_set),end= len(training_set) + len(testing_set)-1,dynamic=False)
        err = math.sqrt(mean_squared_error(testing_set, pred))
        orderl.append(pdq)
        rmse.append(err)
```

In [218…

```python
len(rmse)
```

Out[218]:

```
192
```

In [219…

```python
results = pd.DataFrame(index=orderl,data=rmse,columns=['RMSE'])
results.sort_values(by = 'RMSE')
```

Out[219]:

	RMSE
(5, 2, 3)	666.259570
(0, 2, 1)	667.186487
(7, 2, 5)	667.331312
(4, 2, 7)	667.472986
(1, 2, 0)	667.483422
...	...
(0, 0, 3)	4259.332098
(0, 0, 4)	4259.523810
(0, 0, 2)	4267.902299
(0, 0, 1)	4275.928942
(0, 0, 0)	4288.172304

192 rows × 1 columns

In [ ]:

In [34]:

```python
import statsmodels.api as sm
model = sm.tsa.arima.ARIMA(close, order=(2,1,1)).fit()
```

In [35]:

```python
close
```

Out[35]:

```
Date
2017-11-20     6308.609863
2017-11-21     6378.629883
2017-11-22     6386.120117
2017-11-24     6409.290039
2017-11-27     6405.970215
                  ...
2022-11-14    11700.940430
2022-11-15    11871.150391
2022-11-16    11699.089844
2022-11-17    11676.860352
2022-11-18    11677.019531
Name: Close, Length: 1259, dtype: float64
```

In [36]:

```python
close
```

Out[36]:

```
Date
2017-11-20     6308.609863
2017-11-21     6378.629883
2017-11-22     6386.120117
2017-11-24     6409.290039
2017-11-27     6405.970215
                  ...
2022-11-14    11700.940430
2022-11-15    11871.150391
2022-11-16    11699.089844
2022-11-17    11676.860352
2022-11-18    11677.019531
Name: Close, Length: 1259, dtype: float64
```

In [37]:

```python
pred = model.predict(start = len(close),end= len(close)+4,dynamic=False)
pred
```

Out[37]:

```
1259    11682.541339
1260    11676.798825
1261    11682.012287
1262    11677.376427
1263    11681.484384
Name: predicted_mean, dtype: float64
```

In [1]:

```python
import pandas_datareader as pdr
import pandas as pd
import datetime as datetime
```

In [2]:

```python
fulldf = pdr.get_data_yahoo('^NSEI')
```

In [31]:

```python
df_nifty = fulldf
df_nifty
```

Out[31]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2017-11-21	10358.700195	10315.049805	10329.250000	10326.900391	186100.0	10326.900391
2017-11-22	10368.700195	10309.549805	10350.799805	10342.299805	157600.0	10342.299805
2017-11-23	10374.299805	10307.299805	10358.450195	10348.750000	153000.0	10348.750000
2017-11-24	10404.500000	10362.250000	10366.799805	10389.700195	129200.0	10389.700195
2017-11-27	10407.150391	10340.200195	10361.049805	10399.549805	141900.0	10399.549805
...	...	...	...	...	...	...
2022-11-14	18399.449219	18311.400391	18376.400391	18329.150391	301400.0	18329.150391
2022-11-15	18427.949219	18282.000000	18362.750000	18403.400391	250900.0	18403.400391
2022-11-16	18442.150391	18344.150391	18398.250000	18409.650391	219300.0	18409.650391
2022-11-17	18417.599609	18312.949219	18358.699219	18343.900391	200500.0	18343.900391
2022-11-18	18394.599609	18209.800781	18382.949219	18307.650391	198800.0	18307.650391

1230 rows × 6 columns

In [32]:

```python
from statsmodels.tsa.stattools import adfuller
```

In [33]:

```python
close = df_nifty['Close']
result = adfuller(close)
print(result[1])
```

```
0.8990818986137608
```

In [34]:

```python
df = df_nifty
```

In [35]:

```python
df['Close_1'] = df['Close']-df['Close'].shift(1)
```

In [36]:

```python
df
```

Out[36]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2017-11-21	10358.700195	10315.049805	10329.250000	10326.900391	186100.0	10326.900391	NaN
2017-11-22	10368.700195	10309.549805	10350.799805	10342.299805	157600.0	10342.299805	15.399414
2017-11-23	10374.299805	10307.299805	10358.450195	10348.750000	153000.0	10348.750000	6.450195
2017-11-24	10404.500000	10362.250000	10366.799805	10389.700195	129200.0	10389.700195	40.950195
2017-11-27	10407.150391	10340.200195	10361.049805	10399.549805	141900.0	10399.549805	9.849609
...	...	...	...	...	...	...	...
2022-11-14	18399.449219	18311.400391	18376.400391	18329.150391	301400.0	18329.150391	-20.548828
2022-11-15	18427.949219	18282.000000	18362.750000	18403.400391	250900.0	18403.400391	74.250000
2022-11-16	18442.150391	18344.150391	18398.250000	18409.650391	219300.0	18409.650391	6.250000
2022-11-17	18417.599609	18312.949219	18358.699219	18343.900391	200500.0	18343.900391	-65.750000
2022-11-18	18394.599609	18209.800781	18382.949219	18307.650391	198800.0	18307.650391	-36.250000

1230 rows × 7 columns

In [37]:

```python
close_1 = df['Close_1'].dropna()
result_1 = adfuller(close_1)
print(result_1[1])
```

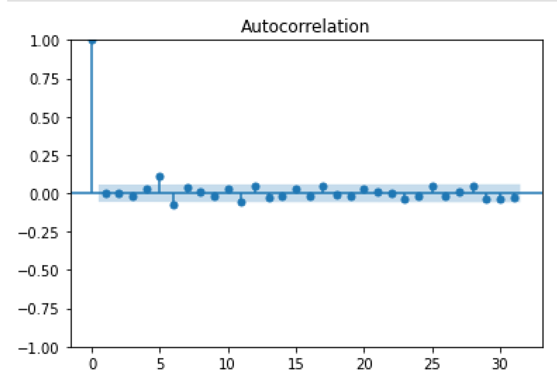
```
7.218623989040169e-23
```

In [38]:

```python
from statsmodels.graphics.tsaplots import  plot_acf,plot_pacf
```

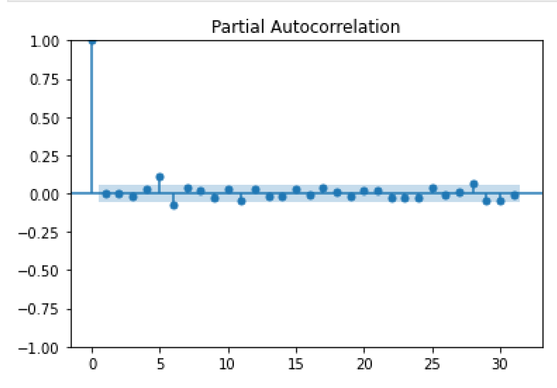
In [39]:

```python
acf = plot_acf(df['Close_1'].dropna());
```



In [40]:

```python
pacf = plot_pacf(df['Close_1'].dropna());
```



In [41]:

```python
df.shape
```

Out[41]:

```
(1230, 7)
```

In [42]:

```python
## test train split
train_data = df[:-100]
train_data
```

Out[42]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2017-11-21	10358.700195	10315.049805	10329.250000	10326.900391	186100.0	10326.900391	NaN
2017-11-22	10368.700195	10309.549805	10350.799805	10342.299805	157600.0	10342.299805	15.399414
2017-11-23	10374.299805	10307.299805	10358.450195	10348.750000	153000.0	10348.750000	6.450195
2017-11-24	10404.500000	10362.250000	10366.799805	10389.700195	129200.0	10389.700195	40.950195
2017-11-27	10407.150391	10340.200195	10361.049805	10399.549805	141900.0	10399.549805	9.849609
...	...	...	...	...	...	...	...
2022-06-17	15400.400391	15183.400391	15272.650391	15293.500000	342600.0	15293.500000	-67.099609
2022-06-20	15382.500000	15191.099609	15334.500000	15350.150391	260000.0	15350.150391	56.650391
2022-06-21	15707.250000	15419.849609	15455.950195	15638.799805	262800.0	15638.799805	288.649414
2022-06-22	15565.400391	15385.950195	15545.650391	15413.299805	220900.0	15413.299805	-225.500000
2022-06-23	15628.450195	15367.500000	15451.549805	15556.650391	259200.0	15556.650391	143.350586

1130 rows × 7 columns

In [43]:

```python
test_data=df[-100:]
test_data
```

Out[43]:

	High	Low	Open	Close	Volume	Adj Close	Close_1
Date							
2022-06-24	15749.250000	15619.450195	15657.400391	15699.250000	219600.0	15699.250000	142.599609
2022-06-27	15927.450195	15815.500000	15926.200195	15832.049805	210900.0	15832.049805	132.799805
2022-06-28	15892.099609	15710.150391	15757.450195	15850.200195	251900.0	15850.200195	18.150391
2022-06-29	15861.599609	15687.799805	15701.700195	15799.099609	444900.0	15799.099609	-51.100586
2022-06-30	15890.000000	15728.849609	15774.500000	15780.250000	306000.0	15780.250000	-18.849609
...	...	...	...	...	...	...	...
2022-11-14	18399.449219	18311.400391	18376.400391	18329.150391	301400.0	18329.150391	-20.548828
2022-11-15	18427.949219	18282.000000	18362.750000	18403.400391	250900.0	18403.400391	74.250000
2022-11-16	18442.150391	18344.150391	18398.250000	18409.650391	219300.0	18409.650391	6.250000
2022-11-17	18417.599609	18312.949219	18358.699219	18343.900391	200500.0	18343.900391	-65.750000
2022-11-18	18394.599609	18209.800781	18382.949219	18307.650391	198800.0	18307.650391	-36.250000

100 rows × 7 columns

In [44]:

```python
training_set = close[:-100]
testing_set  = close[-100:]
```

In [45]:

```python
testing_set
```

Out[45]:

```
Date
2022-06-24    15699.250000
2022-06-27    15832.049805
2022-06-28    15850.200195
2022-06-29    15799.099609
2022-06-30    15780.250000
                  ...
2022-11-14    18329.150391
2022-11-15    18403.400391
2022-11-16    18409.650391
2022-11-17    18343.900391
2022-11-18    18307.650391
Name: Close, Length: 100, dtype: float64
```

In [46]:

```python
from pmdarima import auto_arima

import warnings
warnings.filterwarnings("ignore")

stepwise_fit = auto_arima(close,
                          trace = True,
                          error_action ='ignore',
                          suppress_warnings = True)

# To print the summary
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=15795.458, Time=0.13 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=15787.433, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=15789.431, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=15789.437, Time=0.06 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=15787.773, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=15791.437, Time=0.09 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.370 seconds
```

Out[46]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1230
Model:	SARIMAX(0, 1, 0)	**Log Likelihood**	-7891.716
Date:	Sun, 20 Nov 2022	**AIC**	15787.433
Time:	21:38:49	**BIC**	15797.661
Sample:	0	**HQIC**	15791.281
	- 1230		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	6.4937	4.427	1.467	0.142	-2.183	15.171
sigma2	2.213e+04	452.751	48.876	0.000	2.12e+04	2.3e+04

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2280.06
Prob(Q):	0.96	**Prob(JB):**	0.00
Heteroskedasticity (H):	3.87	**Skew:**	-0.83
Prob(H) (two-sided):	0.00	**Kurtosis:**	9.46

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

In [47]:

```python
import itertools
p=range(0,8)
q=range(0,8)
d=range(0,3)
pdq_com = list(itertools.product(p,d,q))
len(pdq_com)
```

Out[47]:

```
192
```

In [20]:

```python
import statsmodels.api as sm
```

In [21]:

```python
rmse=[]
orderl=[]
```

In [217…

```python
for pdq in pdq_com:
        model = sm.tsa.arima.ARIMA(training_set, order=pdq).fit()
        pred = model.predict(start = len(training_set),end= len(training_set) + len(testing_set)-1,dynamic=False)
        err = math.sqrt(mean_squared_error(testing_set, pred))
        orderl.append(pdq)
        rmse.append(err)
```

In [218…

```python
len(rmse)
```

Out[218]:

```
192
```

In [219…

```python
results = pd.DataFrame(index=orderl,data=rmse,columns=['RMSE'])
results.sort_values(by = 'RMSE')
```

Out[219]:

	RMSE
(5, 2, 3)	666.259570
(0, 2, 1)	667.186487
(7, 2, 5)	667.331312
(4, 2, 7)	667.472986
(1, 2, 0)	667.483422
...	...
(0, 0, 3)	4259.332098
(0, 0, 4)	4259.523810
(0, 0, 2)	4267.902299
(0, 0, 1)	4275.928942
(0, 0, 0)	4288.172304

192 rows × 1 columns

In [ ]:

In [48]:

```python
import statsmodels.api as sm
model = sm.tsa.arima.ARIMA(close, order=(5,2,3)).fit()
```

In [49]:

```python
close
```

Out[49]:

```
Date
2017-11-21    10326.900391
2017-11-22    10342.299805
2017-11-23    10348.750000
2017-11-24    10389.700195
2017-11-27    10399.549805
                  ...
2022-11-14    18329.150391
2022-11-15    18403.400391
2022-11-16    18409.650391
2022-11-17    18343.900391
2022-11-18    18307.650391
Name: Close, Length: 1230, dtype: float64
```

In [50]:

```python
org = fulldf[-5:]['Close']
org
```

Out[50]:

```
Date
2022-11-14    18329.150391
2022-11-15    18403.400391
2022-11-16    18409.650391
2022-11-17    18343.900391
2022-11-18    18307.650391
Name: Close, dtype: float64
```

In [51]:

```python
pred = model.predict(start = len(close),end= len(close)+4,dynamic=False)
pred
```

Out[51]:

```
1230    18277.452790
1231    18314.283280
1232    18311.127785
1233    18307.379861
1234    18334.528998
Name: predicted_mean, dtype: float64
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: