Stock Market Prediction And Forecasting Using Stacked LSTM

1.Import : required library

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import io
import requests
import datetime
```

2. Read Dataset

```
df = pd.read_csv("NSE-TATAGLOBAL.csv")
df.head()

#shape of data
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2035 entries, 0 to 2034
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  2035 non-null   object
 1   Open                  2035 non-null   float64
 2   High                  2035 non-null   float64
 3   Low                   2035 non-null   float64
 4   Last                  2035 non-null   float64
 5   Close                 2035 non-null   float64
 6   Total Trade Quantity  2035 non-null   int64
 7   Turnover (Lacs)       2035 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 127.3+ KB
```

3.Gathering information about the data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2035 entries, 0 to 2034
```

```
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Date                  2035 non-null   object 
 1   Open                  2035 non-null   float64
 2   High                  2035 non-null   float64
 3   Low                   2035 non-null   float64
 4   Last                  2035 non-null   float64
 5   Close                 2035 non-null   float64
 6   Total Trade Quantity  2035 non-null   int64  
 7   Turnover (Lacs)       2035 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 127.3+ KB
```

```
df.describe()
```

	Open	High	Low	Last	Close	Total Trade Quantity	T
count	2035.000000	2035.000000	2035.000000	2035.000000	2035.00000	2.035000e+03	2035
mean	149.713735	151.992826	147.293931	149.474251	149.45027	2.335681e+06	3899
std	48.664509	49.413109	47.931958	48.732570	48.71204	2.091778e+06	4570
min	81.100000	82.800000	80.000000	81.000000	80.95000	3.961000e+04	37
25%	120.025000	122.100000	118.300000	120.075000	120.05000	1.146444e+06	1427
50%	141.500000	143.400000	139.600000	141.100000	141.25000	1.783456e+06	2512
75%	157.175000	159.400000	155.150000	156.925000	156.90000	2.813594e+06	4539

```
df.dtypes
```

```
Date                     object
Open                    float64
High                    float64
Low                     float64
Last                    float64
Close                   float64
Total Trade Quantity      int64
Turnover (Lacs)         float64
dtype: object
```

4.Data Cleaning

```
from numpy.lib.function_base import percentile
#Total percentage of data is missing

missing_values_count = df.isnull().sum()
```

```
total_cells = np.product(df.shape)

total_missing = missing_values_count.sum()

percentage_missing = (total_missing/total_cells)*100

print(percentage_missing)
```

```
0.0
```

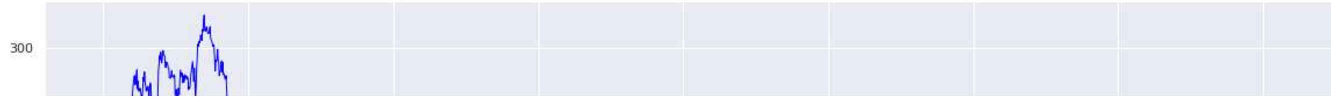
```
NAN = [(c,df[c].isnull().mean()*100)for c in df]
NAN = pd.DataFrame(NAN,columns=['column_name','percentage'])
NAN
```

	column_name	percentage
0	Date	0.0
1	Open	0.0
2	High	0.0
3	Low	0.0
4	Last	0.0
5	Close	0.0
6	Total Trade Quantity	0.0
7	Turnover (Lacs)	0.0

5.Data Visualization

```
sns.set(rc={'figure.figsize':(20,5)})
df['Open'].plot(linewidth = 1, color='blue')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efecbe76650>
```



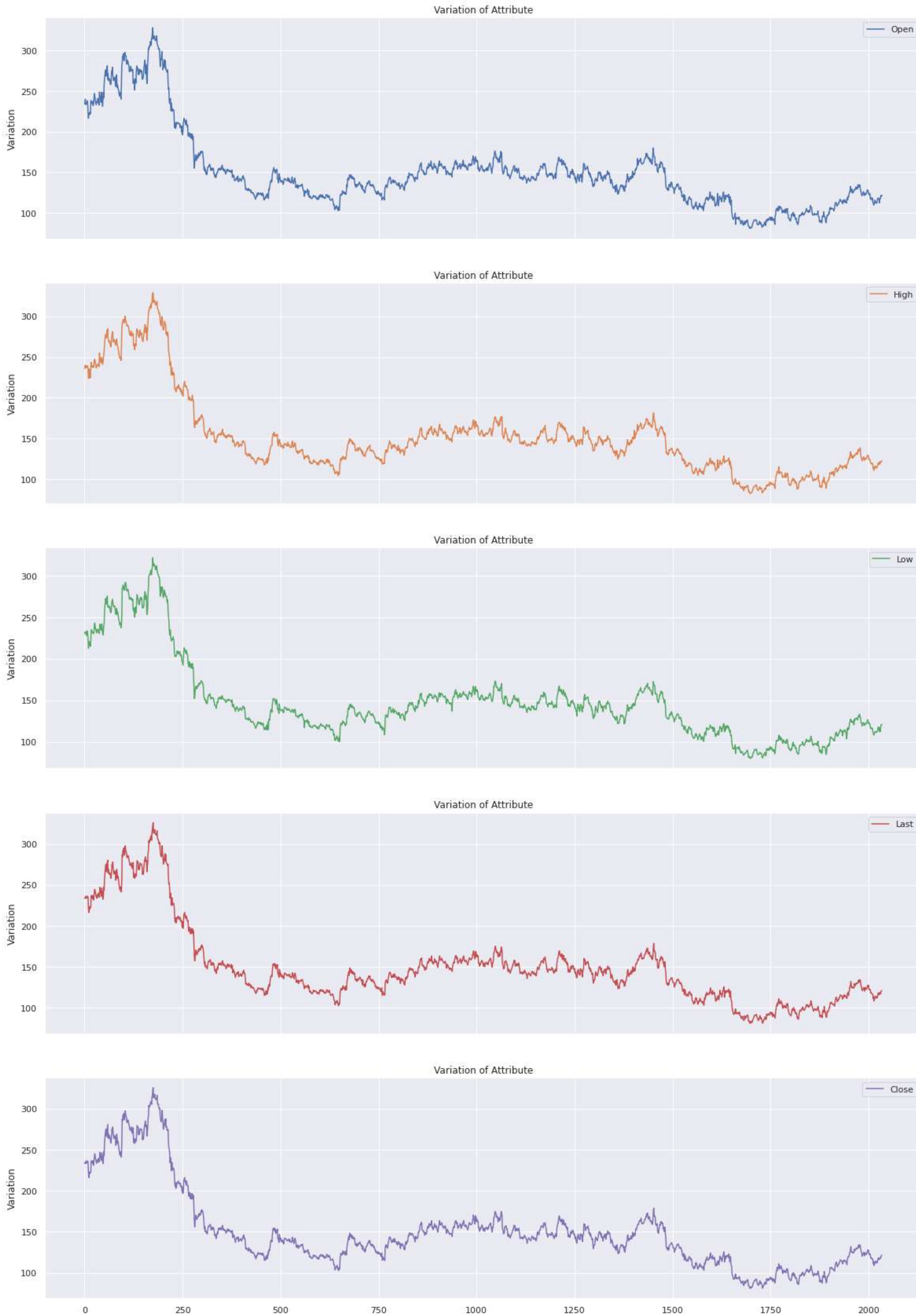

```
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Last', 'Close', 'Total Trade Quantity',
       'Turnover (Lacs)'],
      dtype='object')
```



```
cols_plot = ['Open','High',"Low","Last","Close"]
axes = df[cols_plot].plot(alpha = 1, figsize=(20,30),subplots = True)

for ax in axes:
  ax.set_ylabel("Variation")
  ax.set_title("Variation of Attribute")
```


Variation of Attribute
Open
Variation
Variation of Attribute
High
Variation
Variation of Attribute
Low
Variation
Variation of Attribute
Last
Variation
Variation of Attribute
Close
Variation
100
150
200
250
300
0
250
500
750
1000
1250
1500
1750
2000

Sort the dataset on date time and filter “Date” and “Open” columns

```
df['Date']=pd.to_datetime(df.Date,format ="%Y-%m-%d")
df.index = df['Date']
df
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
Date								
2018-09-28	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
2018-09-27	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2018-09-26	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
2018-09-25	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
2018-09-24	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55
...	...	...	...	...	...	...	...	...
2010-07-27	2010-07-27	117.60	119.50	112.00	118.80	118.65	586100	694.98
2010-07-26	2010-07-26	120.10	121.00	117.10	117.10	117.60	658440	780.01

```
del df['Date']
```

```
df.dtypes
```

```
Open                    float64
High                    float64
Low                     float64
Last                    float64
Close                   float64
Total Trade Quantity      int64
Turnover (Lacs)         float64
dtype: object
```

6. 7 day rolling mean

```
df.rolling(7).mean().head(10)
```

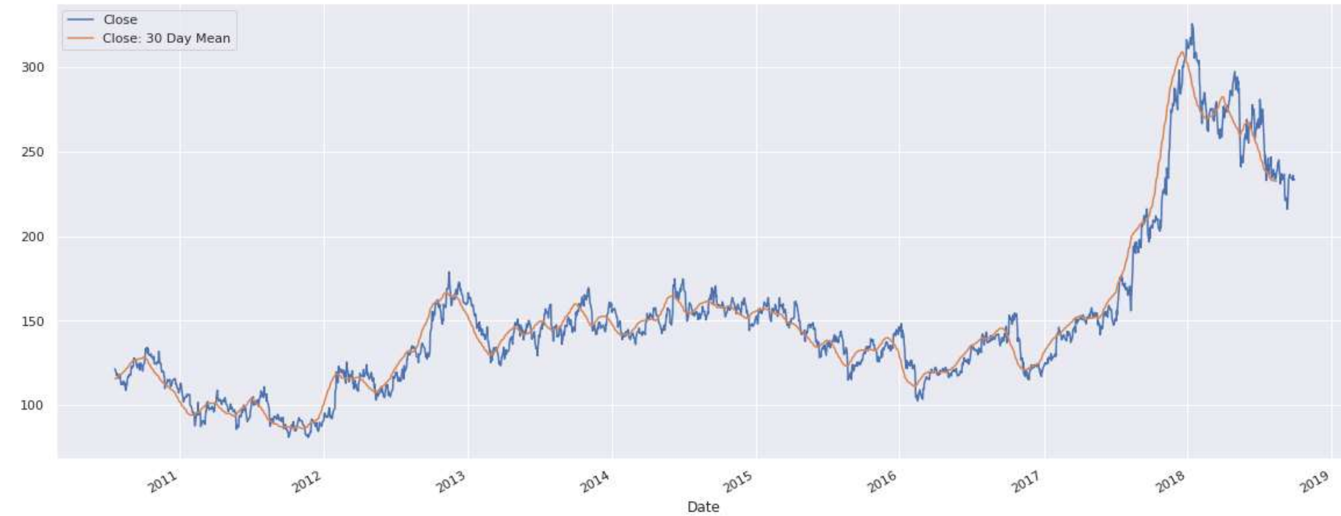
	Open	High	Low	Last	Close	Total Trade Quantity	Turnove (Lacs
Date							
2018-09-28	NaN	NaN	NaN	NaN	NaN	NaN	Na
2018-09-27	NaN	NaN	NaN	NaN	NaN	NaN	Na
2018-09-26	NaN	NaN	NaN	NaN	NaN	NaN	Na
2018-09-25	NaN	NaN	NaN	NaN	NaN	NaN	Na
2018-09-24	NaN	NaN	NaN	NaN	NaN	NaN	Na
2018-09-21	NaN	NaN	NaN	NaN	NaN	NaN	Na

```
df['Open'].plot(figsize=(20,8),alpha=1)
df.rolling(window=30).mean()['Close'].plot(alpha=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efecbe6c410>
```

```
df['Close: 30 Day Mean'] = df['Close'].rolling(window=30).mean()
df[['Close','Close: 30 Day Mean']].plot(figsize=(20,8),alpha=1)
```

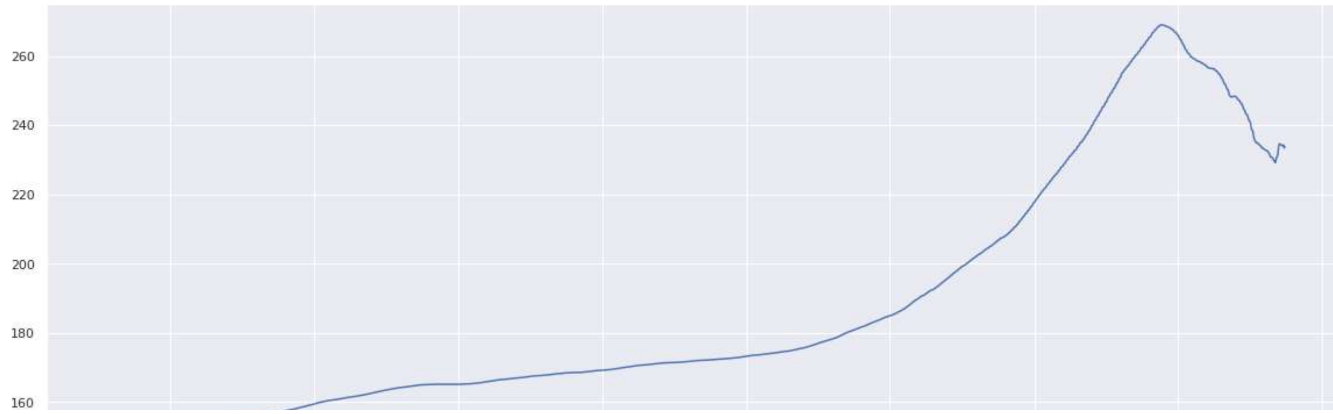
```
<matplotlib.axes._subplots.AxesSubplot at 0x7efecbd8a810>
```



Optional specify a minimum numbe2of periods

```
df['Close'].expanding(min_periods=1).mean().plot(figsize=(20,8),alpha=1)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7efeca104290>

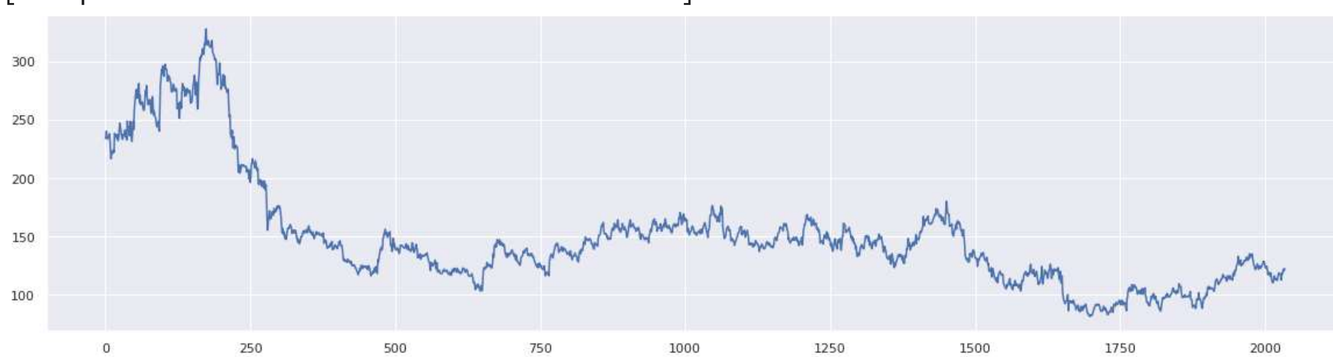



```
df2 = df.reset_index()['Open']
df2
```

```
0       234.05
1       234.55
2       240.00
3       233.30
4       233.55
         ...
2030    117.60
2031    120.10
2032    121.80
2033    120.30
2034    122.10
Name: Open, Length: 2035, dtype: float64
```

```
plt.plot(df2)
```

[<matplotlib.lines.Line2D at 0x7efec9dff6d0>]



6.LSTM are sensitive to the scale of the data. so we apply MinMax scaler

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
df2 = scaler.fit_transform(np.array(df2).reshape(-1,1))

print(df2)
```

```
[[0.6202352 ]
 [0.62226277]
 [0.64436334]
 ...
 [0.16504461]
 [0.15896188]
 [0.16626115]]
```

7.splitting dataset into train and test split

```
train_size = int(len(df2)*0.75)
test_size = len(df2)-train_size

train_data, test_data = df2[0:train_size,:],df2[train_size:len(df2),:1]
```

```
train_size, test_size
```

```
(1526, 509)
```

```
train_data, test_data
```

```
           [0.00587997],
           [0.00973236],
           [0.013382  ],
           [0.01784266],
           [0.01520681],
           [0.02270884],
           [0.03264396],
           [0.02676399],
           [0.02392539],
           [0.0148013 ],
           [0.04217356],
           [0.04257908],
           [0.03892944],
           [0.03649635],
           [0.04906732],
           [0.04622871],
           [0.04115977],
           [0.04379562],
           [0.04420114],
           [0.05616383],
           [0.05515004],
           [0.05636659],
           [0.0405515 ],
           [0.04521492],
```

```
       [0.05352798],
       [0.05636659],
       [0.03994323],
       [0.04927007],
       [0.04379562],
       [0.04744526],
       [0.04420114],
       [0.03122466],
       [0.02007299],
       [0.06042174],
       [0.06893755],
       [0.09184915],
       [0.09164639],
       [0.10077048],
       [0.09610706],
       [0.08678021],
       [0.08880779],
       [0.10948905],
       [0.1107056 ],
       [0.09164639],
       [0.09935118],
       [0.10989457],
       [0.10827251],
       [0.1054339 ],
       [0.10056772],
       [0.10056772],
       [0.09083536],
       [0.07765612],
       [0.08049473],
       [0.08880779],
       [0.08941606],
       [0.08069749],
       [0.09042985],
       [0.07866991],
```

8.convert an array of values into a dataset matrix

```
from matplotlib import numpy
def create_dataset(dataset, time_step=1):
  train_X, train_Y = [], []
  for i in range(len(dataset)-time_step-1):
    a = dataset[i:(i+time_step), 0]        ## i = 0, 0,1,2,3,4,5------99, 100
    train_X.append(a)
    train_Y.append(dataset[i+time_step,0])
  return numpy.array(train_X), numpy.array(train_Y)
```

9.reshape into X=t,t+1,t+2,t+3 and Y=t+4

```
import numpy
```

```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)

X_test, ytest=create_dataset(test_data, time_step)

print(X_train.shape), print(y_train.shape)
```

```
(1425, 100)
(1425,)
(None, None)
```

10. 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)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)
```

11. Create the Stacked LSTM model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM


model = Sequential()

model.add(LSTM(50, return_sequences=True, input_shape=(100,1)))

model.add(LSTM(50, return_sequences=True))

model.add(LSTM(50))

model.add(Dense(1))

model.compile(loss='mean_squared_error',optimizer='adam')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 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
_________________________________________________________________
```

```
model.fit(X_train, y_train, validation_data=(X_test,ytest),epochs=100,batch_size=64, verbose=
```

```
Epoch 1/100
23/23 [==============================] - 10s 216ms/step - loss: 0.0222 - val_loss: 0.
Epoch 2/100
23/23 [==============================] - 4s 162ms/step - loss: 0.0026 - val_loss: 6.9
Epoch 3/100
23/23 [==============================] - 4s 161ms/step - loss: 0.0016 - val_loss: 0.0
Epoch 4/100
23/23 [==============================] - 4s 160ms/step - loss: 0.0015 - val_loss: 9.8
Epoch 5/100
23/23 [==============================] - 4s 160ms/step - loss: 0.0015 - val_loss: 0.0
Epoch 6/100
23/23 [==============================] - 4s 161ms/step - loss: 0.0014 - val_loss: 0.0
Epoch 7/100
23/23 [==============================] - 4s 163ms/step - loss: 0.0013 - val_loss: 0.0
Epoch 8/100
23/23 [==============================] - 4s 159ms/step - loss: 0.0014 - val_loss: 0.0
Epoch 9/100
23/23 [==============================] - 4s 181ms/step - loss: 0.0013 - val_loss: 8.2
Epoch 10/100
23/23 [==============================] - 4s 157ms/step - loss: 0.0012 - val_loss: 0.0
Epoch 11/100
23/23 [==============================] - 4s 155ms/step - loss: 0.0011 - val_loss: 8.4
Epoch 12/100
23/23 [==============================] - 4s 195ms/step - loss: 9.7680e-04 - val_loss:
Epoch 13/100
23/23 [==============================] - 4s 162ms/step - loss: 9.6979e-04 - val_loss:
Epoch 14/100
23/23 [==============================] - 4s 157ms/step - loss: 8.9067e-04 - val_loss:
Epoch 15/100
23/23 [==============================] - 4s 157ms/step - loss: 9.4233e-04 - val_loss:
Epoch 16/100
23/23 [==============================] - 4s 158ms/step - loss: 8.6299e-04 - val_loss:
Epoch 17/100
23/23 [==============================] - 4s 157ms/step - loss: 8.9846e-04 - val_loss:
Epoch 18/100
23/23 [==============================] - 4s 160ms/step - loss: 9.1836e-04 - val_loss:
Epoch 19/100
23/23 [==============================] - 4s 160ms/step - loss: 7.9959e-04 - val_loss:
Epoch 20/100
23/23 [==============================] - 4s 160ms/step - loss: 7.8691e-04 - val_loss:
Epoch 21/100
23/23 [==============================] - 4s 157ms/step - loss: 7.4535e-04 - val_loss:
Epoch 22/100
23/23 [==============================] - 4s 156ms/step - loss: 7.4684e-04 - val_loss:
```

```
Epoch 23/100
23/23 [==============================] - 4s 156ms/step - loss: 7.3388e-04 - val_loss:
Epoch 24/100
23/23 [==============================] - 4s 158ms/step - loss: 6.9437e-04 - val_loss:
Epoch 25/100
23/23 [==============================] - 4s 158ms/step - loss: 7.0839e-04 - val_loss:
Epoch 26/100
23/23 [==============================] - 4s 154ms/step - loss: 6.8030e-04 - val_loss:
Epoch 27/100
23/23 [==============================] - 4s 159ms/step - loss: 7.5250e-04 - val_loss:
Epoch 28/100
23/23 [==============================] - 4s 159ms/step - loss: 6.7548e-04 - val_loss:
Epoch 29/100
```

```
import tensorflow as tf
```

12. Lets Do the prediction and check performance metrics

```
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
45/45 [==============================] - 2s 29ms/step
13/13 [==============================] - 0s 28ms/step
```

```
train_predict = scaler.inverse_transform(train_predict)

test_predict = scaler.inverse_transform(test_predict)
```

13.Calculate RMSE performance metrics

```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
163.6141908605129
```

14.Test data RMSE

```
math.sqrt(mean_squared_error(ytest,test_predict))
```

```
106.9854155573461
```

15.Shift Train prediction for plotting

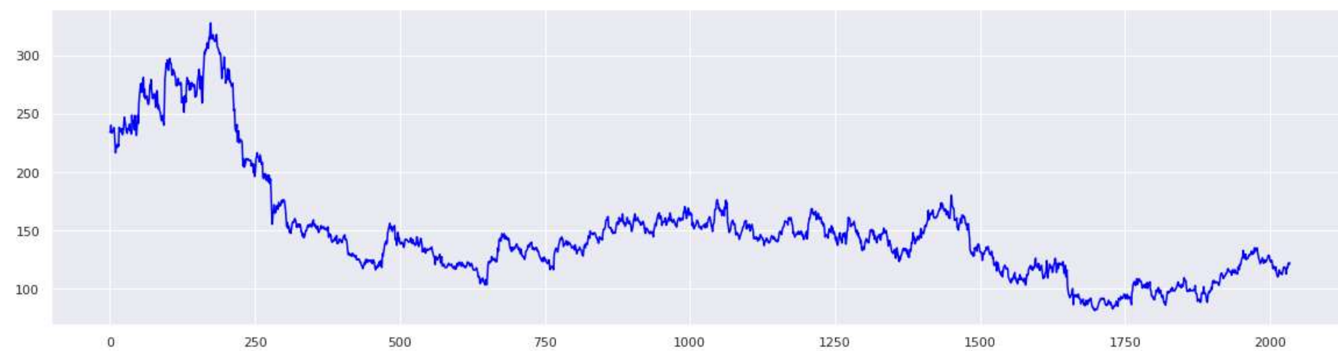
```
look_back = 100
trainPredictplot = numpy.empty_like(df2)
trainPredictplot[:,:] = numpy.nan
trainPredictplot[len(train_predict)+(look_back*2)+1:len(df2)-1,:] = test_predict
```

16.Shift test predication for plotting

```
testPredictplot = numpy.empty_like(df2)
testPredictplot[:,:] = numpy.nan
testPredictplot[len(train_predict)+(look_back*2)+1:len(df2)-1, :] = test_predict
```

17.Plot baseline and predications

```
pred = scaler.inverse_transform(df2)
plt.plot(pred,color='blue')
plt.show()
```




```
plt.plot(trainPredictplot, color='red')
plt.show()
plt.plot(testPredictplot, color = 'green')
plt.show()
```


```
plt.plot(pred,color = 'blue')
plt.plot(trainPredictplot, color='red')
plt.plot(testPredictplot, color = 'green')
plt.show()
```


```
len(test_data)
```

    509