Name = Maalika Maini

Internship at Let's grow more

In [2]:
```python
#Importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [3]:
```python
#Importing dataset
url='https://raw.githubusercontent.com/mwitiderrick/stockprice/master/NSE-TATAGLOBAL.csv'
dataset_train=pd.read_csv(url)
training_set=dataset_train.iloc[:,1:2].values
```

In [39]:
```python
#Checking dataset
dataset_train.head()
```

Out[39]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

In [5]:
```python
dataset_train.describe()
```

Out[5]:

	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
count	2035.000000	2035.000000	2035.000000	2035.000000	2035.00000	2.035000e+03	2035.000000
mean	149.713735	151.992826	147.293931	149.474251	149.45027	2.335681e+06	3899.980565
std	48.664509	49.413109	47.931958	48.732570	48.71204	2.091778e+06	4570.767877
min	81.100000	82.800000	80.000000	81.000000	80.95000	3.961000e+04	37.040000
25%	120.025000	122.100000	118.300000	120.075000	120.05000	1.146444e+06	1427.460000
50%	141.500000	143.400000	139.600000	141.100000	141.25000	1.783456e+06	2512.030000
75%	157.175000	159.400000	155.150000	156.925000	156.90000	2.813594e+06	4539.015000
max	327.700000	328.750000	321.650000	325.950000	325.75000	2.919102e+07	55755.080000

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

Out[7]:

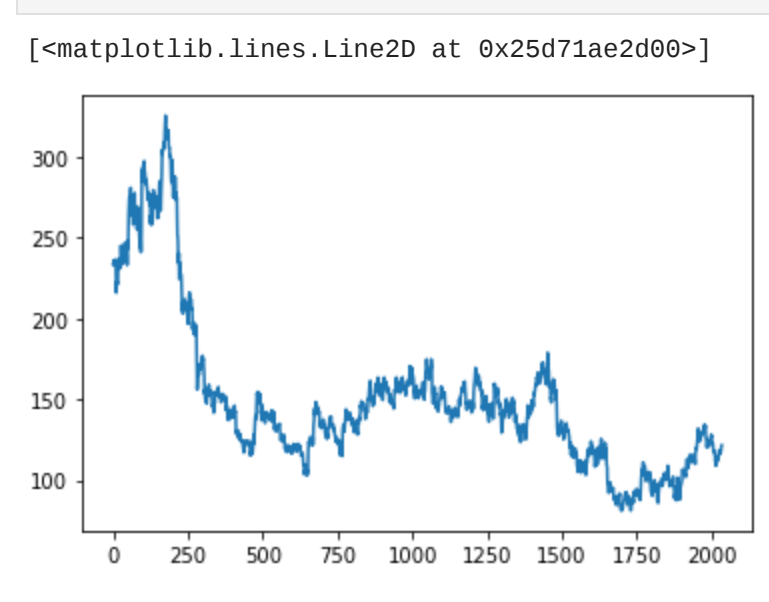
	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
2030	2010-07-27	117.6	119.50	112.00	118.80	118.65	586100	694.98
2031	2010-07-26	120.1	121.00	117.10	117.10	117.60	658440	780.01
2032	2010-07-23	121.8	121.95	120.25	120.35	120.65	281312	340.31
2033	2010-07-22	120.3	122.00	120.25	120.75	120.90	293312	355.17
2034	2010-07-21	122.1	123.00	121.05	121.10	121.55	658666	803.56

In [8]:
```python
data_close=dataset_train['Close']
data_close
```

Out[8]:
```
0       233.75
1       233.25
2       234.25
3       236.10
4       233.30
         ...
2030    118.65
2031    117.60
2032    120.65
2033    120.90
2034    121.55
Name: Close, Length: 2035, dtype: float64
```

In [9]:
```python
plt.plot(data_close)
```

Out[9]: [<matplotlib.lines.Line2D at 0x25d71ae2d00>]



In [10]:
```python
scaler=MinMaxScaler(feature_range=(0,1))
data_close=scaler.fit_transform(np.array(data_close).reshape(-1,1))
```

In [11]:
```python
data_close.shape
```

Out[11]: (2035, 1)

In [12]:
```python
print(data_close)
```

```
[[0.62418301]
 [0.62214052]
 [0.62622549]
 ...
 [0.1621732 ]
 [0.16319444]
 [0.16584967]]
```

Break dataset into train and test range

In [13]:
```python
training_size=int(len(data_close)*0.75)
test_size=len(data_close)-training_size
train_data,test_data=data_close[0:training_size,:],data_close[training_size:len(data_close),:1]
```

In [14]:
```python
training_size
```

Out[14]: 1526

In [15]:
```python
test_size
```

Out[15]: 509

In [16]:
```python
print(train_data)
```

```
[[0.62418301]
 [0.62214052]
 [0.62622549]
 ...
 [0.18831699]
 [0.18811275]
 [0.17034314]]
```

In [17]:
```python
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]
        dataX.append(a)
        dataY.append(dataset[i+time_step,0])
    return np.array(dataX),np.array(dataY)
```

In [18]:
```python
time_step=100
x_train,y_train=create_dataset(train_data,time_step)
x_test,y_test=create_dataset(test_data,time_step)
```

In [19]:
```python
print(x_train)
```

```
[[0.62418301 0.62214052 0.62622549 ... 0.83455882 0.86213235 0.85273693]
 [0.62214052 0.62622549 0.63378268 ... 0.86213235 0.85273693 0.87111928]
 [0.62622549 0.63378268 0.62234477 ... 0.85273693 0.87111928 0.84497549]
 ...
 [0.32271242 0.3247549  0.32148693 ... 0.1997549  0.2001634  0.20506536]
 [0.3247549  0.32148693 0.32352941 ... 0.2001634  0.20506536 0.2005719 ]
 [0.32148693 0.32352941 0.3255719  ... 0.20506536 0.2005719  0.18831699]]
```

In [20]:
```python
print(y_train)
```

```
[0.87111928 0.84497549 0.84027778 ... 0.2005719  0.18831699 0.18811275]
```

In [21]:
```python
print(x_train.shape),print(y_train.shape)
```

```
(1425, 100)
(1425,)
```

Out[21]: (None, None)

In [23]:
```python
#Reshape the input to be[samples,time steps,features] which is the requirement of 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)
```

In [25]:
```python
#Create the LSTM Model
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense
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')
```

In [26]:
```python
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
_________________________________________________________________
```

In [27]:
```python
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=50,batch_size=64,verbose=1)
```

```
Epoch 1/50
23/23 [==============================] - 8s 157ms/step - loss: 0.0188 - val_loss: 0.0069
Epoch 2/50
23/23 [==============================] - 3s 114ms/step - loss: 0.0022 - val_loss: 6.9264e-04
Epoch 3/50
23/23 [==============================] - 3s 134ms/step - loss: 0.0014 - val_loss: 0.0015
Epoch 4/50
23/23 [==============================] - 3s 140ms/step - loss: 0.0012 - val_loss: 0.0010
Epoch 5/50
23/23 [==============================] - 3s 119ms/step - loss: 0.0012 - val_loss: 9.1364e-04
Epoch 6/50
23/23 [==============================] - 3s 113ms/step - loss: 0.0011 - val_loss: 9.1555e-04
Epoch 7/50
23/23 [==============================] - 3s 114ms/step - loss: 0.0011 - val_loss: 8.4672e-04
Epoch 8/50
23/23 [==============================] - 2s 108ms/step - loss: 0.0011 - val_loss: 8.0291e-04
Epoch 9/50
23/23 [==============================] - 3s 110ms/step - loss: 0.0010 - val_loss: 6.1762e-04
Epoch 10/50
23/23 [==============================] - 2s 108ms/step - loss: 9.7239e-04 - val_loss: 9.3562e-04
Epoch 11/50
23/23 [==============================] - 2s 106ms/step - loss: 9.1661e-04 - val_loss: 7.3695e-04
Epoch 12/50
23/23 [==============================] - 3s 121ms/step - loss: 8.6949e-04 - val_loss: 6.6815e-04
Epoch 13/50
23/23 [==============================] - 3s 115ms/step - loss: 8.3378e-04 - val_loss: 6.6442e-04
Epoch 14/50
23/23 [==============================] - 3s 129ms/step - loss: 0.0010 - val_loss: 9.1277e-04
Epoch 15/50
23/23 [==============================] - 3s 113ms/step - loss: 7.8544e-04 - val_loss: 7.6324e-04
Epoch 16/50
23/23 [==============================] - 2s 103ms/step - loss: 7.3302e-04 - val_loss: 6.6436e-04
Epoch 17/50
23/23 [==============================] - 2s 102ms/step - loss: 6.9438e-04 - val_loss: 6.4604e-04
Epoch 18/50
23/23 [==============================] - 2s 105ms/step - loss: 6.7790e-04 - val_loss: 6.7985e-04
Epoch 19/50
23/23 [==============================] - 2s 102ms/step - loss: 6.4056e-04 - val_loss: 7.4193e-04
Epoch 20/50
23/23 [==============================] - 3s 127ms/step - loss: 6.2274e-04 - val_loss: 8.0925e-04
Epoch 21/50
23/23 [==============================] - 3s 113ms/step - loss: 6.1512e-04 - val_loss: 4.9217e-04
Epoch 22/50
23/23 [==============================] - 3s 112ms/step - loss: 6.1569e-04 - val_loss: 7.2961e-04
Epoch 23/50
23/23 [==============================] - 3s 116ms/step - loss: 5.8832e-04 - val_loss: 5.3389e-04
Epoch 24/50
23/23 [==============================] - 3s 128ms/step - loss: 5.6516e-04 - val_loss: 5.3436e-04
Epoch 25/50
23/23 [==============================] - 3s 124ms/step - loss: 5.3576e-04 - val_loss: 5.7864e-04
Epoch 26/50
23/23 [==============================] - 3s 114ms/step - loss: 5.3997e-04 - val_loss: 7.5812e-04
Epoch 27/50
23/23 [==============================] - 3s 121ms/step - loss: 5.1882e-04 - val_loss: 6.6916e-04
Epoch 28/50
23/23 [==============================] - 3s 119ms/step - loss: 5.5042e-04 - val_loss: 5.1381e-04
Epoch 29/50
23/23 [==============================] - 3s 111ms/step - loss: 4.8503e-04 - val_loss: 4.6686e-04
Epoch 30/50
23/23 [==============================] - 2s 105ms/step - loss: 5.4497e-04 - val_loss: 4.1982e-04
Epoch 31/50
23/23 [==============================] - 2s 102ms/step - loss: 4.7207e-04 - val_loss: 4.8263e-04
Epoch 32/50
23/23 [==============================] - 2s 102ms/step - loss: 4.7300e-04 - val_loss: 5.6892e-04
Epoch 33/50
23/23 [==============================] - 3s 112ms/step - loss: 4.9516e-04 - val_loss: 4.8350e-04
Epoch 34/50
23/23 [==============================] - 3s 121ms/step - loss: 4.5017e-04 - val_loss: 4.8008e-04
Epoch 35/50
23/23 [==============================] - 2s 103ms/step - loss: 4.2279e-04 - val_loss: 5.0640e-04
Epoch 36/50
23/23 [==============================] - 3s 113ms/step - loss: 4.0887e-04 - val_loss: 5.7316e-04
Epoch 37/50
23/23 [==============================] - 3s 115ms/step - loss: 4.5720e-04 - val_loss: 3.5597e-04
Epoch 38/50
23/23 [==============================] - 2s 107ms/step - loss: 3.8156e-04 - val_loss: 3.8102e-04
Epoch 39/50
23/23 [==============================] - 2s 105ms/step - loss: 3.8003e-04 - val_loss: 3.9708e-04
Epoch 40/50
23/23 [==============================] - 3s 113ms/step - loss: 3.9434e-04 - val_loss: 6.3063e-04
Epoch 41/50
23/23 [==============================] - 3s 110ms/step - loss: 3.6937e-04 - val_loss: 4.3684e-04
Epoch 42/50
23/23 [==============================] - 2s 103ms/step - loss: 3.5608e-04 - val_loss: 3.2193e-04
Epoch 43/50
23/23 [==============================] - 3s 117ms/step - loss: 3.8909e-04 - val_loss: 3.9870e-04
Epoch 44/50
23/23 [==============================] - 3s 110ms/step - loss: 3.2491e-04 - val_loss: 2.8401e-04
Epoch 45/50
23/23 [==============================] - 2s 108ms/step - loss: 3.3732e-04 - val_loss: 3.5370e-04
Epoch 46/50
23/23 [==============================] - 2s 103ms/step - loss: 3.6264e-04 - val_loss: 3.0418e-04
Epoch 47/50
23/23 [==============================] - 3s 117ms/step - loss: 3.7454e-04 - val_loss: 3.3577e-04
Epoch 48/50
23/23 [==============================] - 3s 109ms/step - loss: 3.6171e-04 - val_loss: 3.6922e-04
Epoch 49/50
23/23 [==============================] - 2s 102ms/step - loss: 3.6446e-04 - val_loss: 3.2862e-04
Epoch 50/50
23/23 [==============================] - 3s 110ms/step - loss: 3.8122e-04 - val_loss: 2.5012e-04
```

Out[27]: <keras.callbacks.History at 0x25d7a798640>

In [28]:
```python
#lets predict and check performance matrics
train_predict = model.predict(x_train)
test_predict = model.predict(x_test)
```

In [29]:
```python
#Transform back to original form
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

In [30]:
```python
#Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train, train_predict))
```

Out[30]: 163.0915721058262

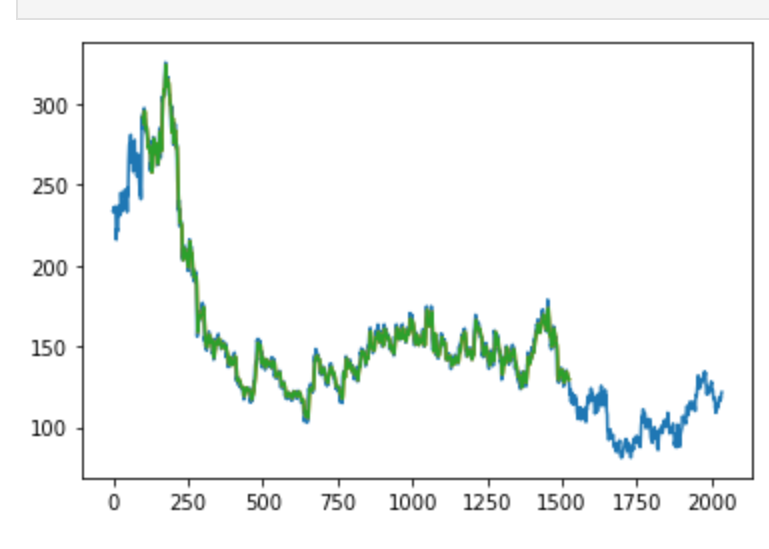
In [31]:
```python
#Test Data RMSE
math.sqrt(mean_squared_error(y_test,test_predict))
```

Out[31]: 105.7597552358849

In [32]:
```python
#Plotting
#Shift train prediction for plotting
look_back=100
trainPredictPlot = np.empty_like(data_close)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict

#Shift test prediction for plotting
testPredictPlot = np.empty_like(data_close)
testPredictPlot[:,:]=np.nan
testPredictPlot[look_back:len(train_predict)+look_back,:] = train_predict

#Plot baseline and predictions
plt.plot(scaler.inverse_transform(data_close))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [33]:
```python
len(test_data),x_test.shape
```

Out[33]: (509, (408, 100, 1))

In [34]:
```python
x_input=test_data[409:].reshape(1,-1)
x_input.shape
```

Out[34]: (1, 100)

In [35]:
```python
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [36]:
```python
temp_input
```

Out[36]:
```
[0.12479575163398693,
 0.1384803921568627,
 0.14011437908496732,
 0.1388888888888889,
 0.13541666666666663,
 0.14011437908496732,
 0.13807189542483655,
 0.130718954248366,
 0.130718954248366,
 0.12867647058823528,
 0.11846405228758167,
 0.14644607843137253,
 0.1480800653594771,
 0.159109477124183,
 0.15992647058823523,
 0.1578839869281045,
 0.16441993464052285,
 0.178921568627451,
 0.17933006535947704,
 0.19260620915032678,
 0.2081290849673203,
 0.18974673202614378,
 0.18055555555555558,
 0.1823937908496731,
 0.17708333333333326,
 0.17810457516339862,
 0.18055555555555558,
 0.17810457516339862,
 0.17851307189542487,
 0.196078431372549,
 0.18913398692810457,
 0.1895424836601307,
 0.19403594771241828,
 0.19444444444444442,
 0.20200163398692816,
 0.19771241830065356,
 0.19934640522875813,
 0.19873366013071891,
 0.19975490196078427,
 0.21282679738562094,
 0.21568627450980393,
 0.2044526143790849,
 0.21772875816993464,
 0.21098856209150318,
 0.21425653594771243,
 0.1975081699346406,
 0.1881127450980392,
 0.17851307189542487,
 0.17381535947712412,
 0.16033496732026142,
 0.16564542483660127,
 0.1711601307189542,
 0.17422385620915026,
 0.1803513071895424,
 0.1740196078431372,
 0.16278594771241828,
 0.1697303921568627,
 0.17810457516339862,
 0.1703431372549019,
 0.16830065359477125,
 0.17279411764705876,
 0.1754493464052288,
 0.18382352941176472,
 0.19138071895424835,
 0.18913398692810457,
 0.1909722222222222,
 0.17238562091503273,
 0.16830065359477125,
 0.16932189542483655,
 0.1717728758169934,
 0.16156045751633985,
 0.14971405228758167,
 0.15032679738562088,
 0.1519607843137255,
 0.1472630718954248,
 0.14501633986928103,
 0.14603758169934639,
 0.12479575163398693,
 0.13112745098039214,
 0.11397058823529405,
 0.11907679738562088,
 0.12377450980392157,
 0.13562091503267976,
 0.12908496732026142,
 0.1345996732026144,
 0.12806372549019607,
 0.13031045751633985,
 0.12724673202614373,
 0.13521241830065356,
 0.1452205882352941,
 0.15257352941176466,
 0.14848856209150324,
 0.14338235294117646,
 0.14562908496732024,
 0.1523692810457516,
 0.15400326797385622,
 0.14971405228758167,
 0.16217320261437906,
 0.16319444444444448,
 0.16584967320261434]
```

In [ ]:
```python
#Indicate the prediction of next 30 days
lst_output=[]
n_steps=100
nextNumberofDays=30
i=0
while(i<nextNumberofDays):
    if(len(temp_input)>100):
        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))
        yhat=model.predict(x_input,verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
    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)
```

In [37]:
```python
new=np.arange(1,101)
predict=np.arange(101,131)
```

In [38]:
```python
len(data_close)
```

Out[38]: 2035

In [ ]: