Task_2

July 23, 2022

[1]:
```
#importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
import pandas_datareader as pdr
import tensorflow as tf
import math
from sklearn.metrics import mean_squared_error
```

[2]:
```
stock_data = pd.read_csv('stock.csv')
```

[3]:
```
stock_data
```

[3]:
```
            Date    Open    High     Low    Last   Close  \
0     2018-09-28  234.05  235.95  230.20  233.50  233.75
1     2018-09-27  234.55  236.80  231.10  233.80  233.25
2     2018-09-26  240.00  240.00  232.50  235.00  234.25
3     2018-09-25  233.30  236.75  232.00  236.25  236.10
4     2018-09-24  233.55  239.20  230.75  234.00  233.30
…            …       …       …       …       …       …
2030  2010-07-27  117.60  119.50  112.00  118.80  118.65
2031  2010-07-26  120.10  121.00  117.10  117.10  117.60
2032  2010-07-23  121.80  121.95  120.25  120.35  120.65
2033  2010-07-22  120.30  122.00  120.25  120.75  120.90
2034  2010-07-21  122.10  123.00  121.05  121.10  121.55

      Total Trade Quantity  Turnover (Lacs)
0                  3069914          7162.35
1                  5082859         11859.95
2                  2240909          5248.60
3                  2349368          5503.90
4                  3423509          7999.55
…                        …                …
2030                586100           694.98
2031                658440           780.01
2032                281312           340.31
```

```
2033                  293312           355.17
2034                  658666           803.56

[2035 rows x 8 columns]
```

[4]:
```python
print(stock_data)
```

```
            Date    Open    High     Low    Last   Close  \
0     2018-09-28  234.05  235.95  230.20  233.50  233.75
1     2018-09-27  234.55  236.80  231.10  233.80  233.25
2     2018-09-26  240.00  240.00  232.50  235.00  234.25
3     2018-09-25  233.30  236.75  232.00  236.25  236.10
4     2018-09-24  233.55  239.20  230.75  234.00  233.30
…            …     …     …     …     …     …
2030  2010-07-27  117.60  119.50  112.00  118.80  118.65
2031  2010-07-26  120.10  121.00  117.10  117.10  117.60
2032  2010-07-23  121.80  121.95  120.25  120.35  120.65
2033  2010-07-22  120.30  122.00  120.25  120.75  120.90
2034  2010-07-21  122.10  123.00  121.05  121.10  121.55

      Total Trade Quantity  Turnover (Lacs)
0                  3069914          7162.35
1                  5082859         11859.95
2                  2240909          5248.60
3                  2349368          5503.90
4                  3423509          7999.55
…                        …                …
2030                586100           694.98
2031                658440           780.01
2032                281312           340.31
2033                293312           355.17
2034                658666           803.56

[2035 rows x 8 columns]
```

[5]:
```python
stock_data.head()
```

[5]:
```
         Date    Open    High     Low    Last   Close  Total Trade Quantity  \
0  2018-09-28  234.05  235.95  230.20  233.50  233.75               3069914
1  2018-09-27  234.55  236.80  231.10  233.80  233.25               5082859
2  2018-09-26  240.00  240.00  232.50  235.00  234.25               2240909
3  2018-09-25  233.30  236.75  232.00  236.25  236.10               2349368
4  2018-09-24  233.55  239.20  230.75  234.00  233.30               3423509

   Turnover (Lacs)
0          7162.35
1         11859.95
2          5248.60
```

```
3          5503.90
4          7999.55
```

[6]:
```
stock_data.tail()
```

[6]:
```
            Date   Open    High     Low    Last   Close  Total Trade Quantity  \
2030  2010-07-27  117.6  119.50  112.00  118.80  118.65                586100
2031  2010-07-26  120.1  121.00  117.10  117.10  117.60                658440
2032  2010-07-23  121.8  121.95  120.25  120.35  120.65                281312
2033  2010-07-22  120.3  122.00  120.25  120.75  120.90                293312
2034  2010-07-21  122.1  123.00  121.05  121.10  121.55                658666

      Turnover (Lacs)
2030           694.98
2031           780.01
2032           340.31
2033           355.17
2034           803.56
```

[7]:
```
stock_data.shape
```

[7]: (2035, 8)

[8]:
```
print(stock_data.keys())
```

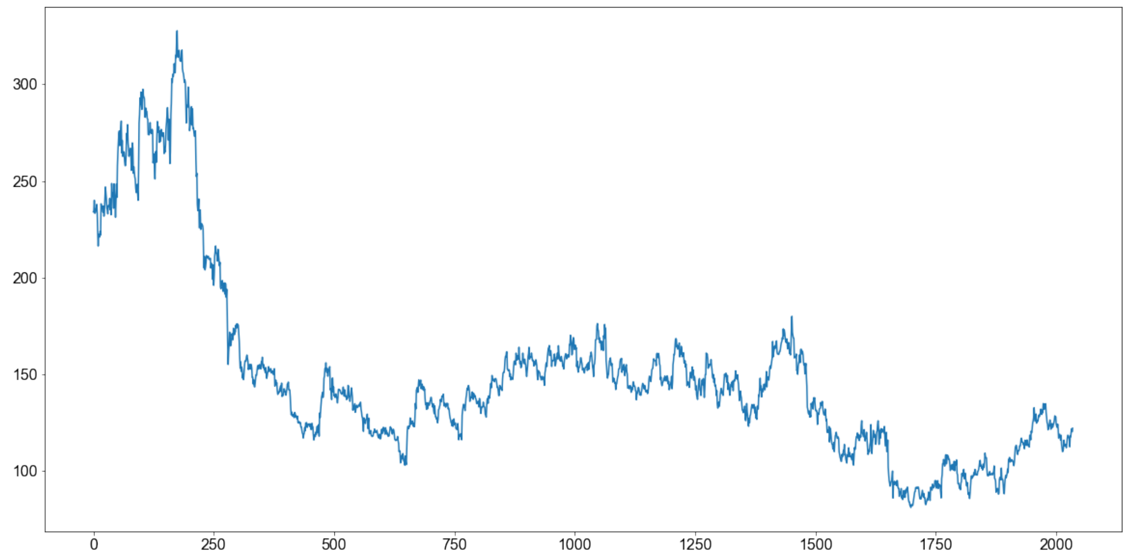
```
Index(['Date', 'Open', 'High', 'Low', 'Last', 'Close', 'Total Trade Quantity',
       'Turnover (Lacs)'],
      dtype='object')
```

[9]:
```
stock_data.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
```

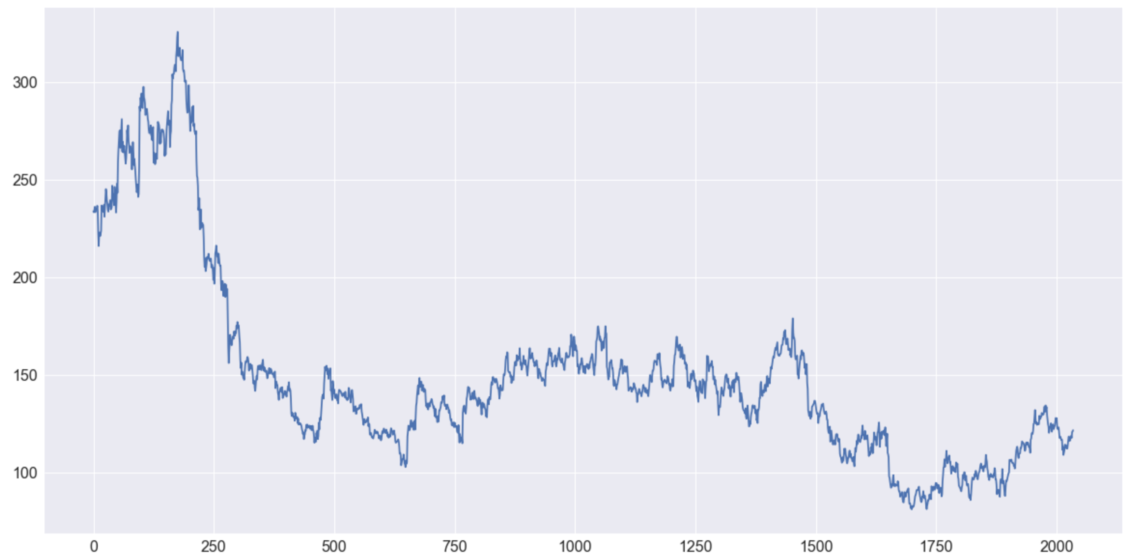
[10]:

```
stock_data['Open'].plot(figsize=(20,10), fontsize = 16)
plt.style.use("seaborn")
plt.show()
```



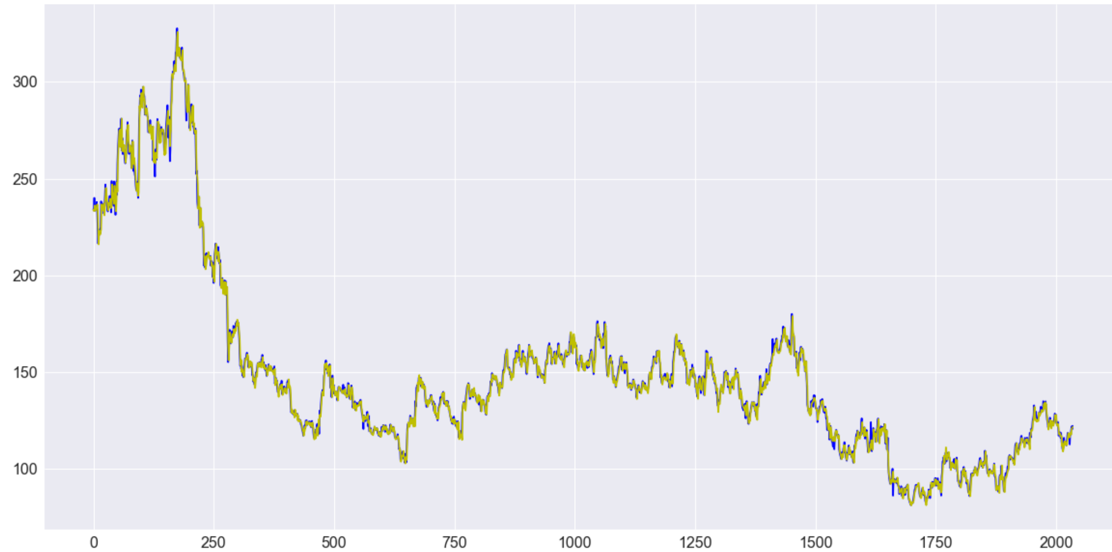
[11]:

```
stock_data['Close'].plot(figsize=(20,10), fontsize = 16)
plt.style.use("seaborn")
plt.show()
```



[12]:

```
stock_data['Open'].plot(figsize=(20,10), fontsize = 16, color="b")
plt.style.use("seaborn")
stock_data['Close'].plot(figsize=(20,10), fontsize = 16, color="y")
plt.style.use("seaborn")
plt.show()
```

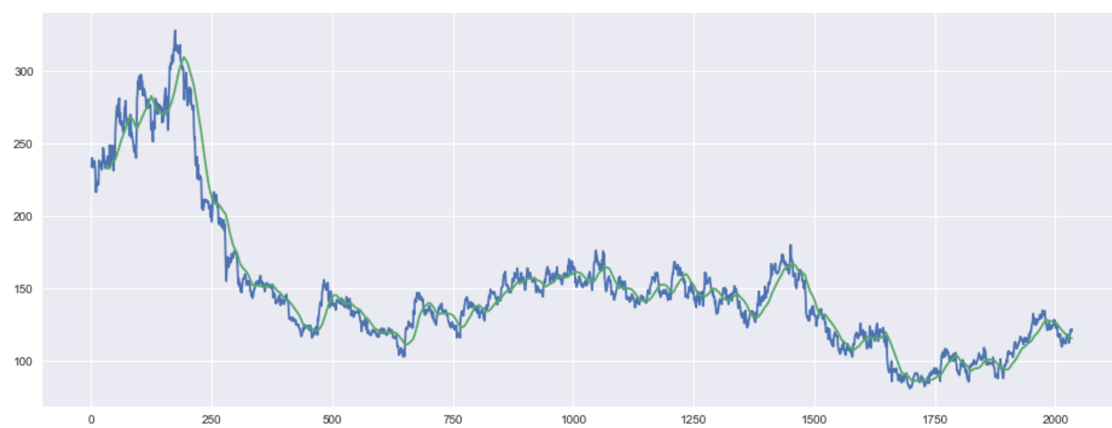


[13]:

```
stock_data['Open'].plot(figsize=(16,6))
stock_data.rolling(window=30).mean()['Close'].plot()
```

```
C:\Temp\ipykernel_18560\1620442995.py:2: FutureWarning: Dropping of nuisance
columns in rolling operations is deprecated; in a future version this will raise
TypeError. Select only valid columns before calling the operation. Dropped
columns were Index(['Date'], dtype='object')
  stock_data.rolling(window=30).mean()['Close'].plot()
```

[13]: <AxesSubplot:>



[14]:
```
plt.figure(figsize=(7,5))
sns.histplot(stock_data['Close'],color='red')
plt.title('Distribution of the Close Price', fontsize=16)
plt.xlabel('Closing Price', fontsize=15)
plt.ylabel('Density', fontsize=15)
plt.show()
```

[15]:
```
plt.figure(figsize=(6,5))
sns.distplot(stock_data['Close'],color='blue')
plt.title('Distribution of the Close Price', fontsize=16)
plt.xlabel('Closing Price', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.show()
```

```
C:\Users\Sushan Shivagiri\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
```

```
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

0.0.1 7 Days rolling mean

[16]:
```
stock_data.rolling(7).mean().head(21)
```

```
C:\Temp\ipykernel_18560\758619304.py:1: FutureWarning: Dropping of nuisance
columns in rolling operations is deprecated; in a future version this will raise
TypeError. Select only valid columns before calling the operation. Dropped
columns were Index(['Date'], dtype='object')
  stock_data.rolling(7).mean().head(21)
```

[16]:

	Open	High	Low	Last	Close	\
0	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	

```
6   235.200000  237.557143  231.135714  234.414286  234.307143
7   235.750000  238.028571  231.607143  234.700000  234.492857
8   235.550000  238.200000  231.485714  235.071429  234.971429
9   233.185714  237.728571  230.171429  234.928571  234.928571
10  230.764286  235.864286  227.407143  232.842857  233.007143
11  229.185714  233.892857  225.135714  230.321429  230.535714
12  227.400000  233.628571  224.092857  228.507143  228.735714
13  225.264286  231.814286  222.042857  226.871429  227.028571
14  223.278571  229.778571  219.857143  224.792857  225.028571
15  221.685714  227.864286  217.707143  222.750000  223.000000
16  223.792857  228.078571  217.607143  221.242857  221.535714
17  226.600000  230.914286  220.807143  223.414286  223.542857
18  228.671429  232.964286  223.392857  226.028571  226.157143
19  230.507143  233.271429  225.028571  228.350000  228.157143
20  232.342857  235.157143  226.971429  229.928571  229.814286

    Total Trade Quantity  Turnover (Lacs)
0                    NaN              NaN
1                    NaN              NaN
2                    NaN              NaN
3                    NaN              NaN
4                    NaN              NaN
5                    NaN              NaN
6           3.274848e+06      7652.388571
7           3.209831e+06      7509.724286
8           2.936693e+06      6879.075714
9           3.527693e+06      8241.347143
10          3.845060e+06      8883.934286
11          3.857272e+06      8846.257143
12          4.159956e+06      9494.928571
13          4.141448e+06      9429.222857
14          4.016310e+06      9099.654286
15          3.995196e+06      8989.585714
16          3.591903e+06      8043.774286
17          3.687891e+06      8406.114286
18          3.665725e+06      8431.457143
19          2.866757e+06      6629.497143
20          2.889922e+06      6706.281429
```

1 Stock Prediction Model

1.1 Preprocess

[17]:
```
#data cleaning
stock_data.isna().any()
```

```
[17]: Date                    False
      Open                    False
      High                    False
      Low                     False
      Last                    False
      Close                   False
      Total Trade Quantity    False
      Turnover (Lacs)         False
      dtype: bool
```

```
[18]: train_set=stock_data['Open']
      train_set=pd.DataFrame(train_set)
```

```
[19]: # Feature Scaling
      from sklearn.preprocessing import MinMaxScaler
      sc = MinMaxScaler(feature_range = (0, 1))
      train_set_scaled = sc.fit_transform(np.array(train_set).reshape(-1,1))
```

```
[20]: train_set_scaled
```

```
[20]: array([[0.6202352 ],
             [0.62226277],
             [0.64436334],
             …,
             [0.16504461],
             [0.15896188],
             [0.16626115]])
```

```
[21]: print(train_set_scaled)
```

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

```
[22]: ##splitting test and train data
      training_size=int(len(train_set_scaled)*0.65)
      test_size=len(train_set_scaled)-training_size
```

```
[23]: train_data,test_data=train_set_scaled[0:training_size,:
       ↪],train_set_scaled[training_size:len(train_set_scaled),:1]
```

```
[24]: training_size,test_size
```

```
[24]: (1322, 713)
```

[25]:
```
#convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    Xdata, Ydata = [], []
    for i in range(len(dataset)-time_step-1):
        x = dataset[i:(i+time_step), 0]
        Xdata.append(x)
        Ydata.append(dataset[i + time_step, 0])
    return np.array(Xdata), np.array(Ydata)
```

[26]:
```
time_step = 100
X_train, Y_train = create_dataset(train_data, time_step)
X_test, Y_test = create_dataset(test_data, time_step)
```

[27]:
```
print(X_train)
```

```
[[0.6202352  0.62226277 0.64436334 … 0.85908354 0.84549878 0.87145174]
 [0.62226277 0.64436334 0.61719384 … 0.84549878 0.87145174 0.84225466]
 [0.64436334 0.61719384 0.61820762 … 0.87145174 0.84225466 0.83515815]
 …
 [0.31589619 0.32846715 0.3215734  … 0.27047851 0.26277372 0.27716951]
 [0.32846715 0.3215734  0.33819951 … 0.26277372 0.27716951 0.24756691]
 [0.3215734  0.33819951 0.33292782 … 0.27716951 0.24756691 0.26094891]]
```

[28]:
```
print(X_test.shape), print(Y_test.shape)
```

```
(612, 100)
(612,)
```

[28]: (None, None)

[29]:
```
print(X_train.shape), (Y_train.shape)
```

```
(1221, 100)
```

[29]: (None, (1221,))

[30]:
```
#reshaping
X_train =np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test =X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

2 Building the Recurrent Neural Network

RNN LSTM Model

[31]:
```
#Importing keras libraries and packages
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
```

2.0.1 Initializing the Recurrent Neural Network

[32]:
```
regressor = Sequential()
```

2.1 Adding the LSTM Layer and some Dropout regularization

[33]:
```
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.
 ↪shape[1], 1)))
regressor.add(Dropout(0.2))
```

[34]:
```
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
```

[35]:
```
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
```

[36]:
```
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))
```

[37]:
```
print("Dropout is called as regularization, which is used to reduce overfitting.
 ↪")
```

```
Dropout is called as regularization, which is used to reduce overfitting.
```

[38]:
```
regressor.summary()
```

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

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 50)           20200

 dropout_1 (Dropout)         (None, 100, 50)           0

 lstm_2 (LSTM)               (None, 100, 50)           20200

 dropout_2 (Dropout)         (None, 100, 50)           0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

=================================================================
Total params: 71,000
Trainable params: 71,000
```

```
Non-trainable params: 0
_________________________________________________________________
```

3 Optimizer

3.1 Adaptive Movement Estimation - Adam

3.1.1 Types of optimizer can greatly affects how fast the algorithm converges to minimum value.

[39]:
```
#compiling
regressor.add(Dense(10))
regressor.compile(optimizer = 'adam', loss= 'mean_squared_error')
```

[40]:
```
#RNN and Training set fitting
regressor.fit(X_train, Y_train, epochs = 100, batch_size = 64, verbose=1)
```

```
Epoch 1/100
20/20 [==============================] - 11s 214ms/step - loss: 0.0499
Epoch 2/100
20/20 [==============================] - 5s 220ms/step - loss: 0.0112
Epoch 3/100
20/20 [==============================] - 4s 212ms/step - loss: 0.0083
Epoch 4/100
20/20 [==============================] - 4s 204ms/step - loss: 0.0065
Epoch 5/100
20/20 [==============================] - 5s 249ms/step - loss: 0.0062
Epoch 6/100
20/20 [==============================] - 5s 235ms/step - loss: 0.0061
Epoch 7/100
20/20 [==============================] - 4s 210ms/step - loss: 0.0054
Epoch 8/100
20/20 [==============================] - 4s 205ms/step - loss: 0.0050
Epoch 9/100
20/20 [==============================] - 4s 203ms/step - loss: 0.0049
Epoch 10/100
20/20 [==============================] - 4s 203ms/step - loss: 0.0042
Epoch 11/100
20/20 [==============================] - 4s 207ms/step - loss: 0.0049
Epoch 12/100
20/20 [==============================] - 4s 205ms/step - loss: 0.0042
Epoch 13/100
20/20 [==============================] - 4s 202ms/step - loss: 0.0039
Epoch 14/100
20/20 [==============================] - 4s 202ms/step - loss: 0.0040
Epoch 15/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0037
Epoch 16/100
```

```
20/20 [==============================] - 4s 186ms/step - loss: 0.0043
Epoch 17/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0044
Epoch 18/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0035
Epoch 19/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0034
Epoch 20/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0031
Epoch 21/100
20/20 [==============================] - 4s 186ms/step - loss: 0.0034
Epoch 22/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0031
Epoch 23/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0029
Epoch 24/100
20/20 [==============================] - 4s 185ms/step - loss: 0.0033
Epoch 25/100
20/20 [==============================] - 4s 183ms/step - loss: 0.0029
Epoch 26/100
20/20 [==============================] - 4s 186ms/step - loss: 0.0025
Epoch 27/100
20/20 [==============================] - 4s 194ms/step - loss: 0.0027
Epoch 28/100
20/20 [==============================] - 4s 184ms/step - loss: 0.0024
Epoch 29/100
20/20 [==============================] - 4s 182ms/step - loss: 0.0025
Epoch 30/100
20/20 [==============================] - 4s 184ms/step - loss: 0.0024
Epoch 31/100
20/20 [==============================] - 4s 186ms/step - loss: 0.0024
Epoch 32/100
20/20 [==============================] - 4s 186ms/step - loss: 0.0024
Epoch 33/100
20/20 [==============================] - 4s 193ms/step - loss: 0.0026
Epoch 34/100
20/20 [==============================] - 4s 197ms/step - loss: 0.0025
Epoch 35/100
20/20 [==============================] - 4s 198ms/step - loss: 0.0024
Epoch 36/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0024
Epoch 37/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0022
Epoch 38/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0020
Epoch 39/100
20/20 [==============================] - 4s 193ms/step - loss: 0.0021
Epoch 40/100
```

```
20/20 [==============================] - 4s 190ms/step - loss: 0.0021
Epoch 41/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0020
Epoch 42/100
20/20 [==============================] - 4s 224ms/step - loss: 0.0019
Epoch 43/100
20/20 [==============================] - 4s 211ms/step - loss: 0.0021
Epoch 44/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0026
Epoch 45/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0019
Epoch 46/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0020
Epoch 47/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0018
Epoch 48/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0016
Epoch 49/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0019
Epoch 50/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0016
Epoch 51/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0017
Epoch 52/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0018
Epoch 53/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0017
Epoch 54/100
20/20 [==============================] - 4s 184ms/step - loss: 0.0026
Epoch 55/100
20/20 [==============================] - 4s 184ms/step - loss: 0.0021
Epoch 56/100
20/20 [==============================] - 4s 205ms/step - loss: 0.0017
Epoch 57/100
20/20 [==============================] - 5s 236ms/step - loss: 0.0016
Epoch 58/100
20/20 [==============================] - 5s 237ms/step - loss: 0.0017
Epoch 59/100
20/20 [==============================] - 5s 242ms/step - loss: 0.0020
Epoch 60/100
20/20 [==============================] - 5s 239ms/step - loss: 0.0025
Epoch 61/100
20/20 [==============================] - 5s 237ms/step - loss: 0.0018
Epoch 62/100
20/20 [==============================] - 4s 220ms/step - loss: 0.0016
Epoch 63/100
20/20 [==============================] - 5s 231ms/step - loss: 0.0014
Epoch 64/100
```

```
20/20 [==============================] - 4s 220ms/step - loss: 0.0014
Epoch 65/100
20/20 [==============================] - 5s 227ms/step - loss: 0.0016
Epoch 66/100
20/20 [==============================] - 4s 194ms/step - loss: 0.0017
Epoch 67/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0016
Epoch 68/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0013
Epoch 69/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0016
Epoch 70/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0014
Epoch 71/100
20/20 [==============================] - 4s 189ms/step - loss: 0.0013
Epoch 72/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0015
Epoch 73/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0014
Epoch 74/100
20/20 [==============================] - 4s 186ms/step - loss: 0.0015
Epoch 75/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0013
Epoch 76/100
20/20 [==============================] - 4s 201ms/step - loss: 0.0013
Epoch 77/100
20/20 [==============================] - 4s 195ms/step - loss: 0.0012
Epoch 78/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0013
Epoch 79/100
20/20 [==============================] - 4s 195ms/step - loss: 0.0012
Epoch 80/100
20/20 [==============================] - 4s 200ms/step - loss: 0.0013
Epoch 81/100
20/20 [==============================] - 4s 194ms/step - loss: 0.0014
Epoch 82/100
20/20 [==============================] - 4s 194ms/step - loss: 0.0012
Epoch 83/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0013
Epoch 84/100
20/20 [==============================] - 4s 197ms/step - loss: 0.0012
Epoch 85/100
20/20 [==============================] - 4s 199ms/step - loss: 0.0013
Epoch 86/100
20/20 [==============================] - 4s 212ms/step - loss: 0.0012
Epoch 87/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0014
Epoch 88/100
```

```
20/20 [==============================] - 4s 192ms/step - loss: 0.0012
Epoch 89/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0011
Epoch 90/100
20/20 [==============================] - 4s 187ms/step - loss: 0.0010
Epoch 91/100
20/20 [==============================] - 4s 192ms/step - loss: 0.0011
Epoch 92/100
20/20 [==============================] - 4s 192ms/step - loss: 0.0013
Epoch 93/100
20/20 [==============================] - 4s 193ms/step - loss: 0.0012
Epoch 94/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0012
Epoch 95/100
20/20 [==============================] - 4s 192ms/step - loss: 0.0012
Epoch 96/100
20/20 [==============================] - 4s 194ms/step - loss: 0.0013
Epoch 97/100
20/20 [==============================] - 4s 192ms/step - loss: 0.0012
Epoch 98/100
20/20 [==============================] - 4s 190ms/step - loss: 0.0012
Epoch 99/100
20/20 [==============================] - 4s 188ms/step - loss: 0.0011
Epoch 100/100
20/20 [==============================] - 4s 191ms/step - loss: 0.0012
```

[40]: <keras.callbacks.History at 0x22c2740c430>

4 Making the prediction and visualizing the results

[41]:
```
tf.__version__
```

[41]: '2.9.1'

[42]:
```
train_predict=regressor.predict(X_train)
```

```
39/39 [==============================] - 5s 43ms/step
```

[43]:
```
test_predict=regressor.predict(X_test)
```

```
20/20 [==============================] - 1s 41ms/step
```

[44]:
```
#transform to original form
train_predict=sc.inverse_transform(train_predict)
test_predict=sc.inverse_transform(test_predict)
```

[45]:
```
y_pred = regressor.predict(X_test)
```

```
20/20 [==============================] - 1s 41ms/step
```

```
[46]: y_pred
```

```
[46]: array([[0.3324863 , 0.3313312 , 0.3319041 , …, 0.33122027, 0.33148885,
         0.33138138],
        [0.32892781, 0.32764468, 0.32828352, …, 0.3277564 , 0.32814696,
         0.32778877],
        [0.32437682, 0.3230249 , 0.32371116, …, 0.32333308, 0.32384223,
         0.32324842],
        …,
        [0.15653126, 0.15617089, 0.15730938, …, 0.15684131, 0.1550341 ,
         0.1538467 ],
        [0.15659226, 0.15625528, 0.15736169, …, 0.15689993, 0.15519105,
         0.15394503],
        [0.15862687, 0.15842235, 0.15945096, …, 0.15892482, 0.1572892 ,
         0.15609416]], dtype=float32)
```

5 Plotting

```
[47]: plt.figure(figsize=(30,10))
plt.plot(10**(np.array(Y_test)), color='red')
plt.plot(10**(y_pred), color='Blue')
plt.suptitle('Actual Vs. Predicted Close Price', fontsize=30)
plt.legend(['Actual','Predicted'], fontsize=20)
plt.xlabel('No of Test Data', fontsize=15)
plt.ylabel('Closing Price', fontsize=15)
plt.grid()
```


```
[ ]:
```