REPORT – STOCK PRICE PREDICTION

End-Semester Evaluation

Submitted by:

(8024320023) – Ayush Susheel

(8024320018) – Anurag Karmakar

(8024320006) – Abhinav Kumar

(8024320021) – Atul Garg

ME(CSE) First Year Course Work

Submitted to:

"Mrs. Divisha Garg"

Doctoral Researcher

Computer Science and Engineering Department

TIET, Patiala

November, 2024

In the project majorly 4 Steps has to be followed in a sequence

- Data Visualization
- Data Pre-Processing
- Model Implementation
- Model Evaluation

So hereby,

This report consists of the 1st two steps as mentioned below :-

Step1) Data Visualization

- The very 1st step is to import the necessary libraries.
- Here we have used the matplotlib, seaborn and plotly libraries for data visualization.

Importing Libraries

[3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sb
import plotly.express as px


import warnings
warnings.filterwarnings('ignore')
```

• Understanding the Dataset

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	30-08-2019	59.924999	59.924999	59.190151	59.404999	59.337475	22596000
3	03-09-2019	58.851501	59.344501	58.16	58.419498	58.353096	29598000
4	04-09-2019	58.835499	59.174	58.549999	59.070499	59.003357	21378000
5	05-09-2019	59.5765	60.652	59.5765	60.569	60.500153	28162000
6	06-09-2019	60.406502	60.60075	60.126099	60.246498	60.178017	21442000
7	09-09-2019	60.200001	61	59.631001	60.220501	60.15205	29438000
8	10-09-2019	59.7575	60.5	59.729	60.299999	60.231457	25202000
9	11-09-2019	60.170502	61.130001	60.110001	61.008499	60.939152	26140000
10	12-09-2019	61.215	62.092999	61.151001	61.712502	61.642357	34518000
11	13-09-2019	61.567501	62.043999	61.350498	61.978001	61.907551	26028000
1249	15-08-2024	162.210007	163.520004	161.490005	163.169998	163.169998	18392500
1250	16-08-2024	163.410004	166.949997	163.080002	164.740005	164.740005	16853100
1251	19-08-2024	167	168.470001	166.089996	168.399994	168.399994	13100800
1252	20-08-2024	168.740005	170.410004	168.660004	168.960007	168.960007	12622500
1253	21-08-2024	166.990005	168.639999	166.570007	167.630005	167.630005	15269600
1254	22-08-2024	169.039993	169.419998	165.029999	165.490005	165.490005	19123800
1255	23-08-2024	166.550003	167.949997	165.660004	167.429993	167.429993	14281600
1256	26-08-2024	168.154999	169.380005	166.320007	167.929993	167.929993	11990300
1257	27-08-2024	167.610001	168.244995	166.160004	166.380005	166.380005	13718200
1258	28-08-2024	166.779999	167.389999	163.279999	164.5	164.5	15208700
1259	29-08-2024	166.059998	167.630005	161.981995	163.399994	163.399994	17125000
1260	30-08-2024	164.22	165.28	163.41	165.11	164.89	18498800

- We have considered “Google Stock Price” dataset from the years 30th August,2019 - 30th August,2024.
- In the dataset we have features like “Date”, Opening Price as “Open”, Closing Price as “Close” , Highest Price on the specific date as “High” , Lowest Price on the specific date as “Low” , Adjusted Closing Price after all the dividend and split factor as “Adj Close” and the amount of stock bought on specific date as “Volume”.
- Now after understanding the dataset it’s time to visualize the numeric data so that it can understood more clearly and precisely.

- This we did using the matplotlib library (plotted in graph,histogram) form.
- We also used the Plotly library which helps in visualizing data more clearly.

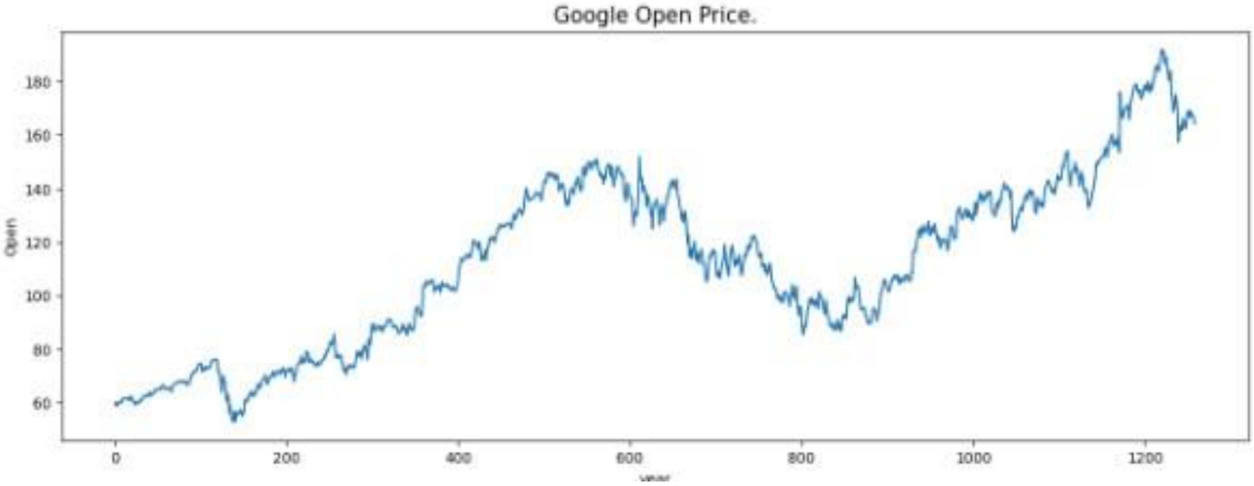
- **<u>Visualizing Data for the Feature "OPEN" in a Graph</u>**

Step 1) Data Visualization

"Open Price"

using matplotlib(Graph,Histogram) and Plotly

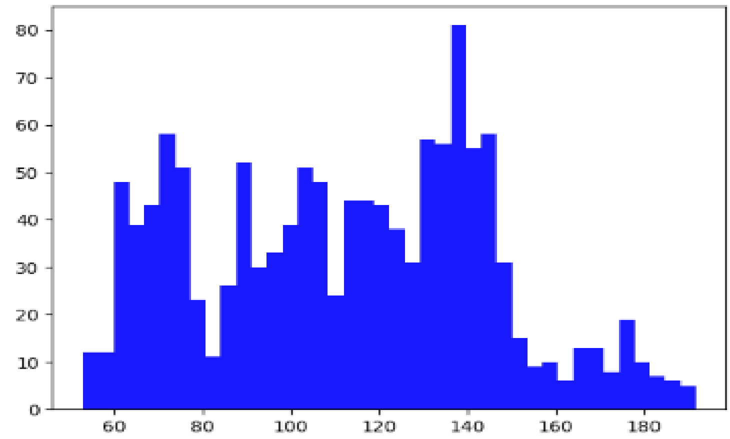
```python
# 1)Open Price

plt.figure(figsize=(15,5))
df['Open'].plot()
# plt.plot(df['Open'])
plt.title('Google Open Price.', fontsize=15)
plt.xlabel("year")
plt.ylabel('Open')
plt.show()
```



- **<u>Visualizing Data for the Feature "OPEN" in Histogram Form</u>**

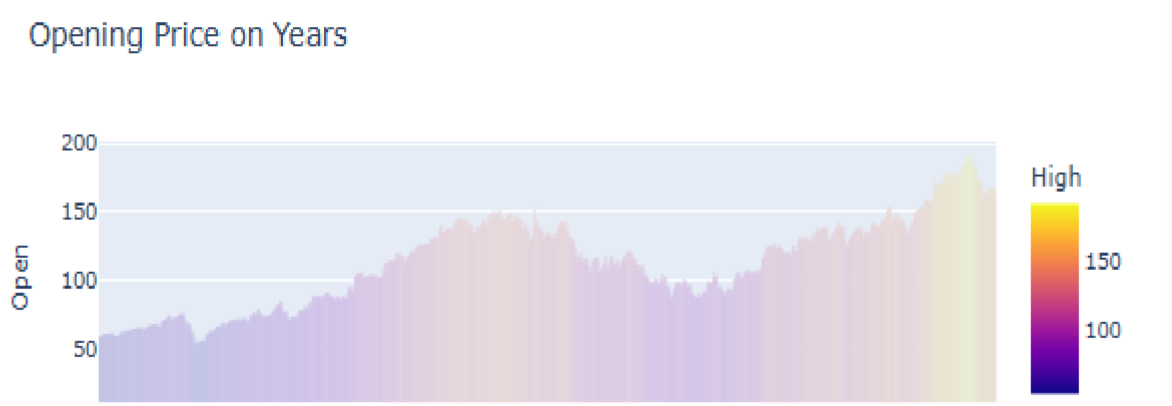
```
[87]: #using matplotlib
plt.hist(df['Open'],bins= 40,color='blue',alpha=0.9)

 # bins:- width of the data points,aplha = transparency of the chart 0-1 range
```

```
[87]: (array([12., 12., 48., 39., 43., 58., 51., 23., 11., 26., 52., 30., 33.,
        39., 51., 48., 24., 44., 44., 43., 38., 31., 57., 56., 81., 55.,
        58., 31., 15.,  9., 10.,  6., 13., 13.,  8., 19., 10.,  7.,  6.,
         5.]),
 array([ 52.8255   ,  56.2986125,  59.771725 ,  63.2448375,  66.71795  ,
         70.1910625,  73.664175 ,  77.1372875,  80.6104   ,  84.0835125,
         87.556625 ,  91.0297375,  94.50285  ,  97.9759625, 101.449075 ,
        104.9221875, 108.3953   , 111.8684125, 115.341525 , 118.8146375,
        122.28775  , 125.7608625, 129.233975 , 132.7070875, 136.1802   ,
        139.6533125, 143.126425 , 146.5995375, 150.07265  , 153.5457625,
        157.018875 , 160.4919875, 163.9651   , 167.4382125, 170.911325 ,
        174.3844375, 177.85755  , 181.3306625, 184.803775 , 188.2768875,
        191.75     ]),
 <BarContainer object of 40 artists>)
```



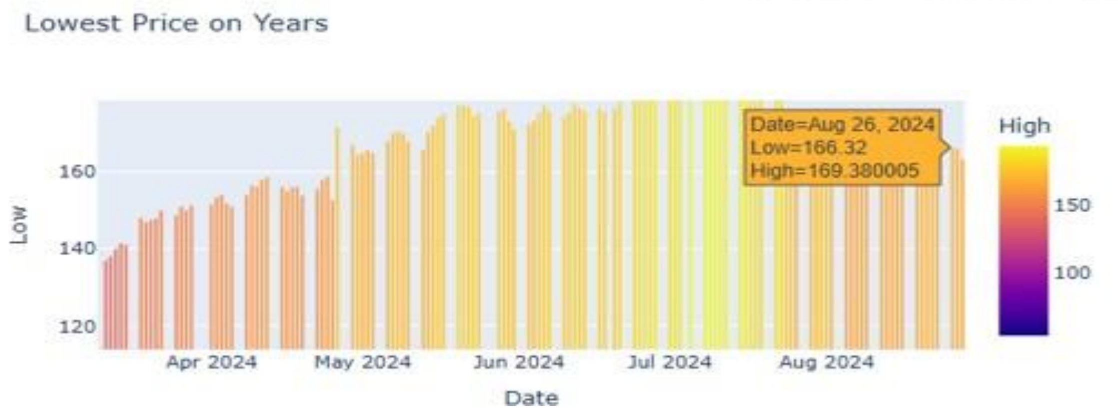
- **<u>Visualizing Data for the Feature “OPEN” using Plotly</u>**

```
[12]: # using plotly
fig = px.bar(df,x='Date',y='Open',title="Opening Price on Years",color="High")
fig.show()
```



- **<u>Visualizing Data for the Feature “LOW” using Plotly</u>**

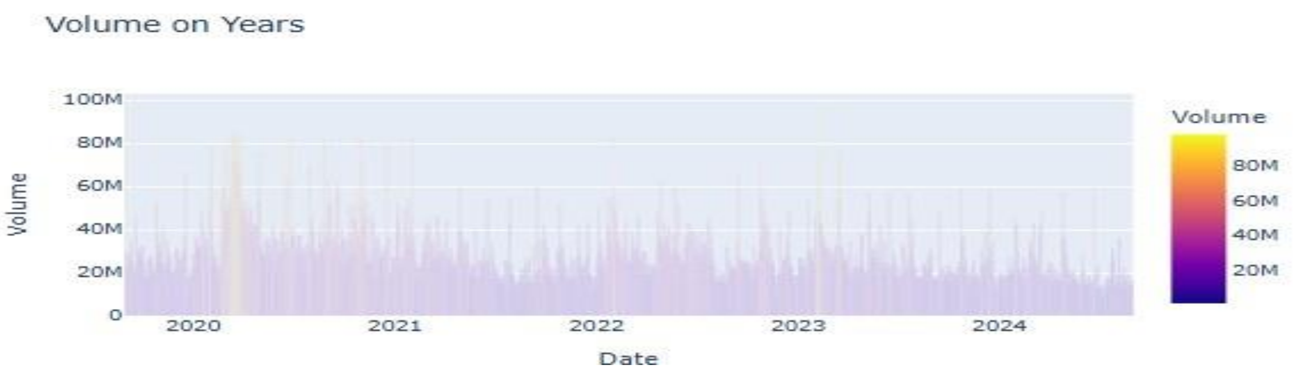
```
# 3)Low Price

# plt.figure(figsize=(15,5))
# plt.plot(df['High'])
# plt.title('Google High Price.', fontsize=15)
# plt.ylabel('Price in Dollars.')
# plt.show()


fig = px.bar(df,x='Date',y='Low',title="Lowest Price on Years",color="High")
fig.show()
```



- **<u>Visualizing Data for the Feature "VOLUME" using Plotly</u>**

"Volume"

[89]:
```
# 6)Volume

# plt.figure(figsize=(15,5))
# plt.plot(df['High'])
# plt.title('Google High Price.', fontsize=15)
# plt.ylabel('Price in Dollars.')
# plt.show()


fig = px.bar(df,x='Date',y='Volume',title="Volume on Years",color="Volume")
fig.show()
```



- Like these the visualizations are performed for all the features mentioned below :-
  - Open
  - High
  - Low
  - Close
  - AdjClose
  - Volume

Now after visualization comes the second step,

Step2) Data Preprocessing

- In Data preprocssing various steps are involved which are very important to perform because this data is going to be used further in the model implementation.
- These Steps are :-
  - ✓ Handle the missing values
  - ✓ Data Transformation
  - ✓ Data Normalization
  - ✓ Outlier Removal
  - ✓ Feature Selection

- In our Project **we have used Data Normalization, Outlier Removal** method because we found the no missing values were present in the dataset, there is no need to transform the data as our data is already in numeric form if it was in categorical form then we need to use transformation.

- ## Data Normalization

  - ✓ We found that the 2 features “Adj Close” and “Close” have similar values therefore we are dropping the feature as we should not provide the duplicate value to the model

  - ✓ As we are going to predict the “CLOSE” feature values therefore we are scaling it in the range 0-1 using the MinMaxScaler.

- ## Removing the duplicate values :-

  - ✓ Using the drop function we are dropping the feature “Adj Close”

  - ✓ Also providing the axis value as 1 means dropping the column value and if we provide 0 it means drop the row value.

```
df = df.drop(['Adj Close'], axis=1)
```

```
df.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

```
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume'], dtype='object')
```

- **<u>Scale down the values (MinMaxScaler):-</u>**

  ✓ 1st we are creating a new dataframe which consists only "CLOSE" column values.

  ✓ Then converting it into a numpy array.

```
[99]: #create a new dataframe with only the close column

      data = df.filter(['Close'])

      # converting the dataframe to a numpy array
      dataset = data.values
```

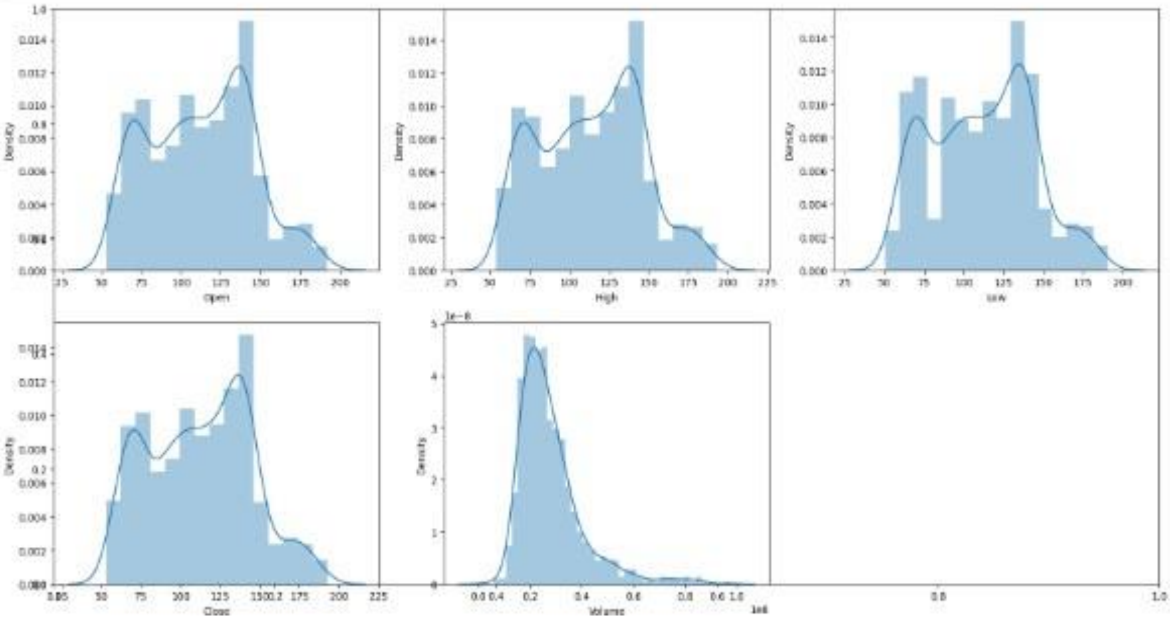
- ✓ Using the MinMaxScaler we are scaling down the values in the range 0 - 1

```
[101]: #scale the data

       scaler = MinMaxScaler(feature_range=(0,1))
       scaled_data = scaler.fit_transform(dataset)

       scaled_data

[101]: array([[0.04701455],
              [0.03996665],
              [0.04462234],
              ...,
              [0.79861114],
              [0.79074434],
              [0.80297361]])
```

- ## Removing the Outliers from the Dataset

✓ 1st finding the outliers by visualizing the values of all features

```
# "First find outliers via visualization"

features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sb.distplot(df[col])
plt.show()
```

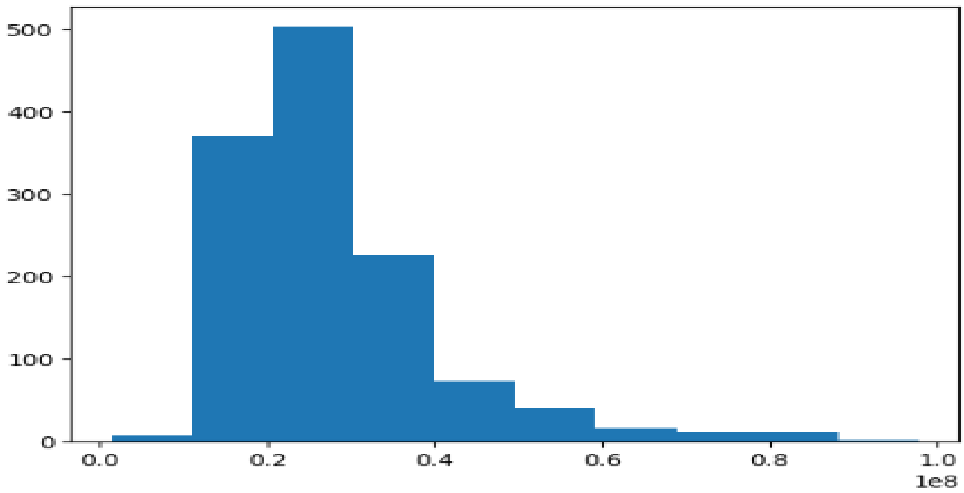


✓ We found that the data is not normally distributed in the feature “VOLUME”

(1)

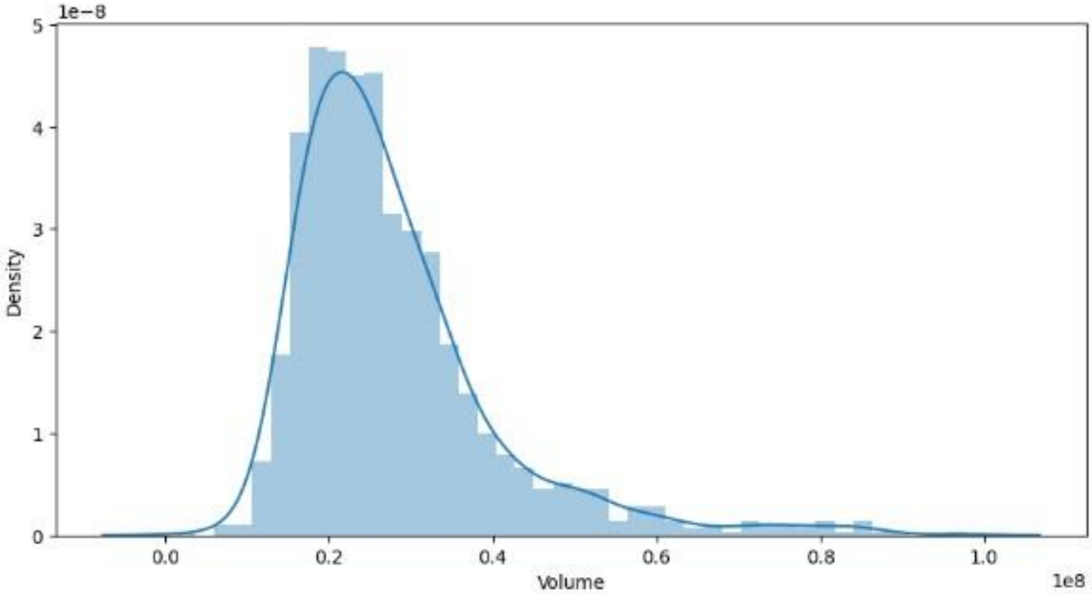
"Visualizing Volume"

```
1]: plt.hist(df['Volume'], bins = 10)
    plt.show()
```



(2) Using Plotly and Seaborn visualizing the feature “VOLUME”

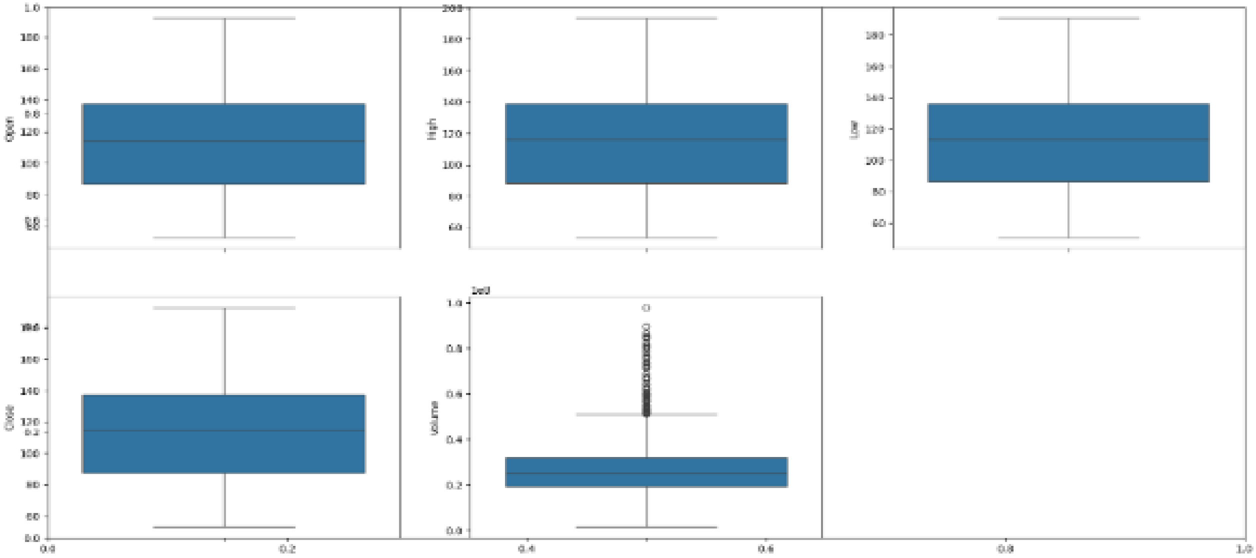
```
[169]: plt.subplots(figsize=(10,5))
       sb.distplot(df['Volume'])

[169]: <Axes: xlabel='Volume', ylabel='Density'>
```



(3) Using Box Plot to visualize the outliers

Creating "Box - Plot"

```
[ ]: plt.subplots(figsize=(20,10))
     for i, col in enumerate(features):
       plt.subplot(2,3,i+1)
       sb.boxplot(df[col])
     plt.show()
```



✓ After finding the outliers remove them from the dataset using the quantile method

"Removing Outliers from Volume"

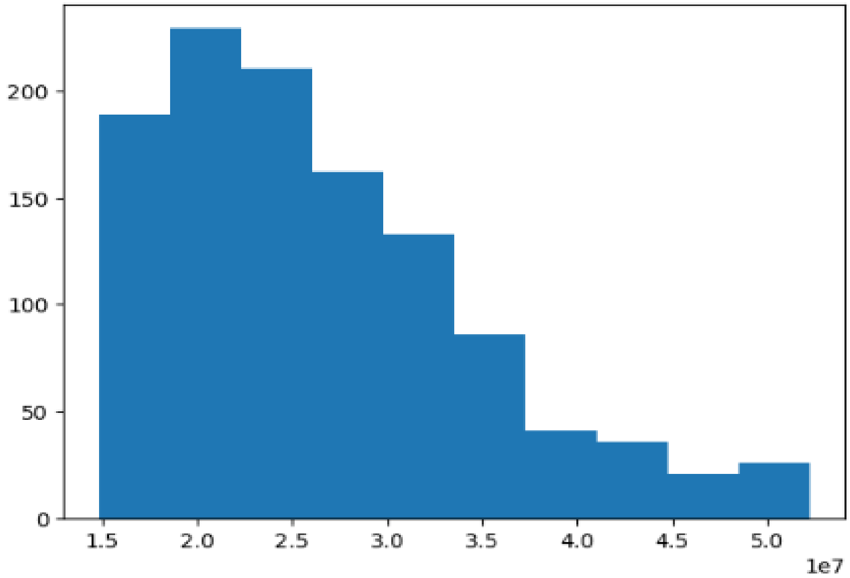
```
[65]: lowerLimit = df['Volume'].quantile(0.05) #means 5%
      print("\nLower Limit is:- " + str(lowerLimit))
      # df[df['Volume'] < lowerLimit]

      upperLimit = df['Volume'].quantile(0.95) #means 95%
      print("\nUpper Limit is:- " + str(upperLimit) + "\n")
      # df[df['Volume'] > upperLimit]

      dataset_with_values_btw_lower_upper = df[ (df['Volume'] > lowerLimit) & (df['Volume'] < upperLimit) ]
```

```
Lower Limit is:- 14855400.0

Upper Limit is:- 52219000.0
```

- ✓ After successfully removing the outliers we are visualizing the dataset to make sure the outliers are removed perfectly.

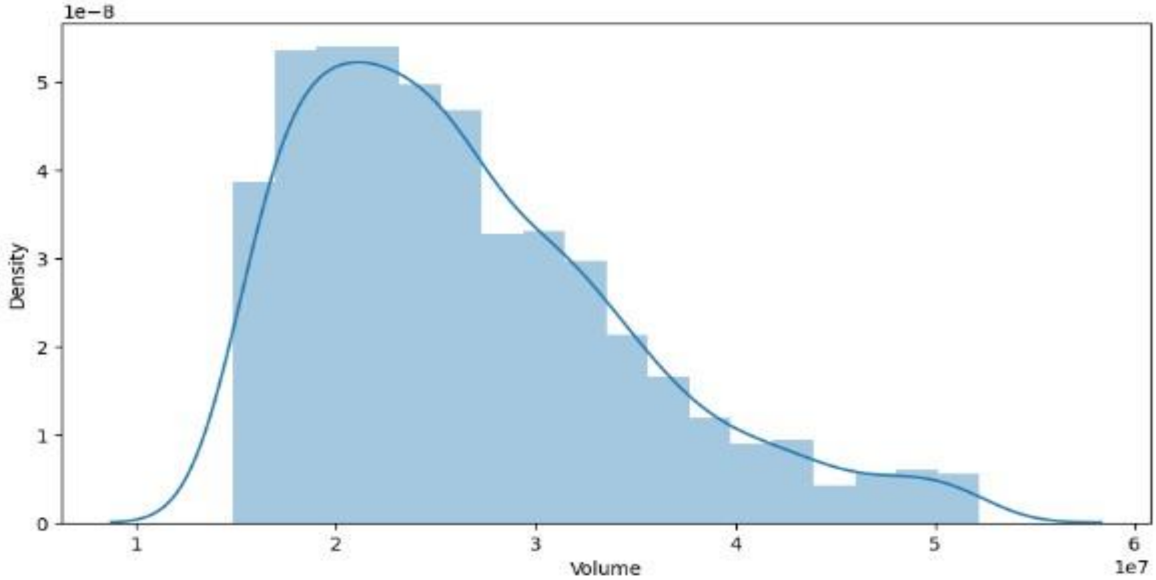
Visualizing Updated DataSet

```
9]:
plt.hist(dataset_with_values_btw_lower_upper['Volume'], bins = 10)
#bins:- width of the data points,aplha = transparency of the chart
plt.show()
```



- ✓ Successfully removed the outliers.

```
[73]: plt.subplots(figsize=(10,5))
      sb.distplot(dataset_with_values_btw_lower_upper['Volume'])

[73]: <Axes: xlabel='Volume', ylabel='Density'>
```



Step 3) Model Implementation

- Now, as we started with the implementation part 1st we calculate the moving average for 250 days, 100 days and will compare both of them via graph (as this concept helps us to see the stock closing price will go down or up)

Here, google_data is our data frame via which we are selecting the Close Feature

Moving Average

```
[42]: google_data['MA_for_250_days'] = google_data['Close'].rolling(250).mean()
```

```
[44]: google_data['MA_for_250_days']
```

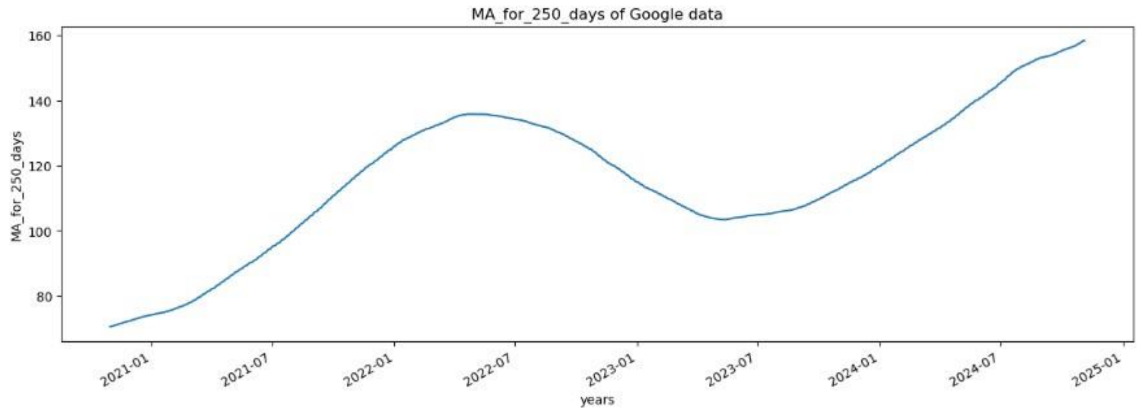
```
[44]: Date
2019-11-05          NaN
2019-11-06          NaN
2019-11-07          NaN
2019-11-08          NaN
2019-11-11          NaN
                  ...
2024-10-29    157.65508
2024-10-30    157.84936
2024-10-31    158.02580
2024-11-01    158.19492
2024-11-04    158.35184
Name: MA_for_250_days, Length: 1258, dtype: float64
```

```
[47]: google_data['MA_for_250_days'][0:250].tail()

# 2020-10-30          NaN
# 2020-11-02          NaN
# 2020-11-03          NaN :---> 248th row
# 2020-11-04          NaN :---> 249th row
# 2020-11-05    70.646793    :----------> this is my 250th row  (so 0 to 249th rows Moving Average value will be zero)
```

```
[47]: Date
2020-10-26          NaN
2020-10-27          NaN
2020-10-28          NaN
2020-10-29          NaN
2020-10-30    70.503353
Name: MA_for_250_days, dtype: float64
```

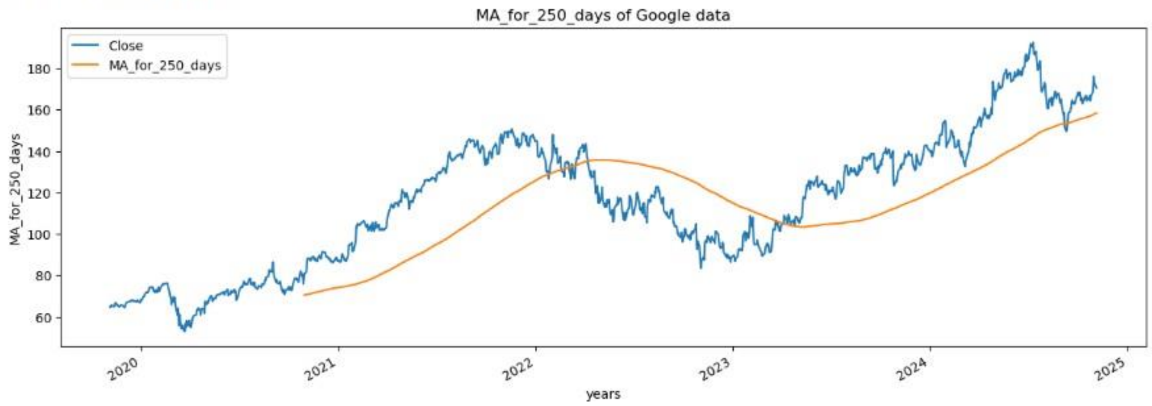
- **Plotting the Graph for Moving Average of 250 days**

```
[55]: plot_graph((15,5), google_data['MA_for_250_days'], 'MA_for_250_days')
```



- **Comparing Moving average for 250 days with Original Stock closing price**

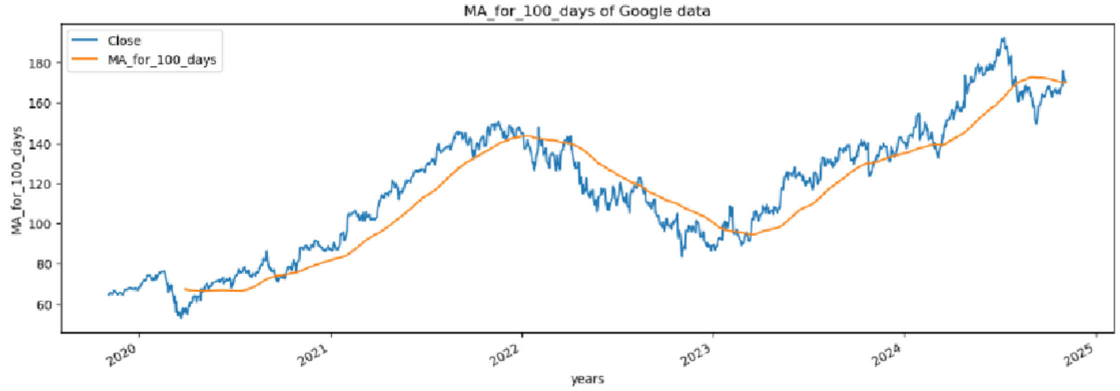
```
[57]: plot_graph((15,5), google_data[['Close','MA_for_250_days']], 'MA_for_250_days')
<Figure size 640x480 with 0 Axes>
```



- **<u>Calculating Moving Avg for 100 days and Comparing it with Original Stock closing price</u>**

```
[21]: google_data['MA_for_100_days'] = google_data['Close'].rolling(100).mean()

      plot_graph((15,5), google_data[['Close','MA_for_100_days']], 'MA_for_100_days')
```

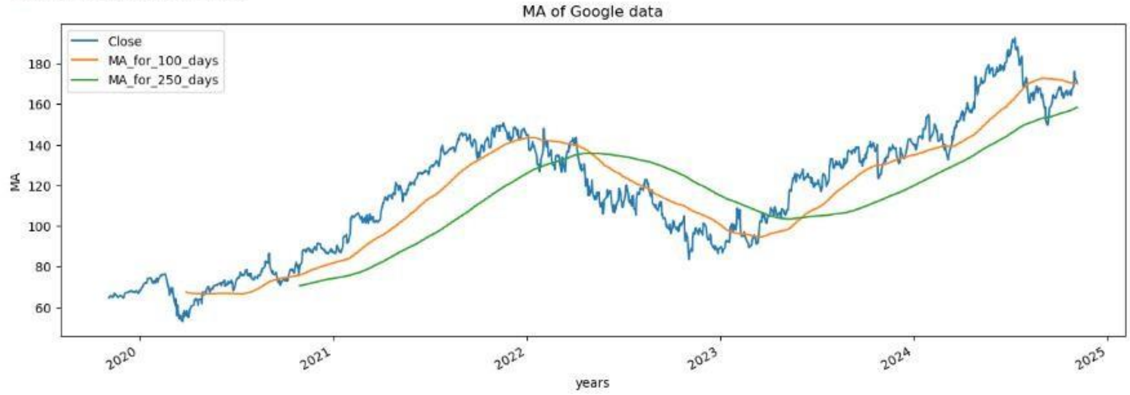
```
<Figure size 640x480 with 0 Axes>
```



- **<u>Comparing both MA's</u>**

```
[61]: plot_graph((15,5), google_data[['Close','MA_for_100_days', 'MA_for_250_days']], 'MA')

      #here we can observe that mera MA for 100 days is good compared to MA of 250 days!!!
```

```
<Figure size 640x480 with 0 Axes>
```



- **Now, creating our x and y data via which we will calculate the values of our Moving Average**

```
]: # creating x and y data as list
   # from the scaled data i will now extracting x,y data
   # Moving Avg concept i'll use here
   # 100 days of previous data i'll take as input training data for each of the pricing dataset
   # 1 to 100 rows of data as input training data to predict 101th row value
   # and for 102th row i'll take data fro 2 to 101 till 1258 rows

   x_data = []
   y_data = []

   #taking previous 100 days means previous 99 days will have null values!
   for i in range(100, len(scaled_data)):
       x_data.append(scaled_data[i-100:i])
       y_data.append(scaled_data[i]) #we will predict x data using y data

   import numpy as np
   x_data, y_data = np.array(x_data), np.array(y_data)
```

```
]: x_data[0],y_data[0]
```

```
]: (array([[0.08417782],
           [0.08409554],
           [0.09019588],
           [0.09109337],
           [0.0867381 ],
           [0.08659864],
           [0.08631257],
           [0.09112556],
           [0.09949652],
           [0.09442964],
           [0.09255592],
           [0.08811831],
           [0.08751043],
           [0.08536138],
           [0.08941995],
           [0.0918729 ],
```

- **Now comes the train and test split part where we used 70% of data for training and 30% of data for testing**

```
[84]: #taking 70% as training data and 30% as test data
      int(len(x_data)*0.7)

[84]: 810

[86]: 1258 - 100 - int(len(x_data)*0.7) #-100 bec we have removed the 1st 100 values bcz they are null

[86]: 348

[88]: splitting_len = int(len(x_data)*0.7)

      x_train = x_data[:splitting_len]

      y_train = y_data[:splitting_len]

      x_test = x_data[splitting_len:]

      y_test = y_data[splitting_len:]

[90]: print(x_train.shape)
      print(y_train.shape)
      print(x_test.shape)
      print(y_test.shape)

      (810, 100, 1)
      (810, 1)
      (348, 100, 1)
      (348, 1)
```

- **<u>We used Sequential Model and "LSTM" (Long Short- Term Memory) layers in order to find the predicted values.</u>**

```
[92]: from keras.models import Sequential
      from keras.layers import Dense, LSTM
```

```
[94]: model = Sequential()  #model is my object and sequential() is the imported model which transfers the data from input layers to the series of layers and t

      model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1],1)))#my 1st layer is LSTM and no. of neurons are 128 and return sequence values

      model.add(LSTM(64,return_sequences=False))#this is my 2nd layer with 64 neurons and return sequnce as FALSE bcx here i'm not giving as a sequence

      model.add(Dense(25))#dense layer with 25 neurons

      model.add(Dense(1)) # last layer with single neuron  this is my output neuron and output layer
```

```
[96]: #optimizer is just to optimize the running of the model ADAM is just an algo,
      # and loss value as :--> this is a matrix MEAN SQUARE ERROR

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

```
[98]: #using the  training data to fit in the model

      model.fit(x_train, y_train, batch_size=1, epochs = 10) #bath size will basically segregate the input data i.e training data into some batches the default
```

```
Epoch 1/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 33s 35ms/step - loss: 0.0038
Epoch 2/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 29s 36ms/step - loss: 0.0011
Epoch 3/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 30s 37ms/step - loss: 7.0068e-04
Epoch 4/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 29s 36ms/step - loss: 4.9358e-04
Epoch 5/10
810/810 ━━━━━━━━━━━━━━━━━━━━ 29s 36ms/step - loss: 6.5988e-04
```

Here in the LSTM model,

- We added 2 LSTM layers and 2 Dense layers with the neurons as mentioned above.
  As it is used to control the output shape of the LSTM layer and whether it returns the entire sequence of outputs or just the last output

- When this parameter is set to **True**, the LSTM layer will return the full sequence of outputs at each time step for each input in the sequence.

- When this parameter is set to **False**, the LSTM layer only returns the output at the final time step of the sequence.

- ## <u>Here is the model summary</u>

```
[100]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 128)	66,560
lstm_1 (LSTM)	(None, 64)	49,408
dense (Dense)	(None, 25)	1,625
dense_1 (Dense)	(None, 1)	26

```
Total params: 352,859 (1.35 MB)
Trainable params: 117,619 (459.45 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 235,240 (918.91 KB)
```

- Layer Type basically tells the type

- Output Shape tells that how the data transforms when it passes during the network

- And Param means the no. of parameters trained in each layer

- **<u>Predicting the Values using predict() method and passing x_test as a parameter to it.</u>**

```
[102]: #predicting the stock data
       #taking the X data and predicting
       predictions = model.predict(x_test)

       11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step

[104]: predictions

[104]: array([[0.50879323],
              [0.5075314 ],
              [0.49013165],
              [0.5084764 ],
              [0.50260425],
              [0.47551528],
              [0.4752919 ],
              [0.48981792],
              [0.48220223],
              [0.48828593],
              [0.48521414],
              [0.49915785],
              [0.48727116],
              [0.4818209 ],
              [0.45986778],
              [0.46600795],
              [0.47932446],
              [0.514527 ],
```

```
[106]: #as all my values are btw 0 - 1 i need to apply inverse transform method to get my original data

       inv_predictions = scaler.inverse_transform(predictions)

       inv_predictions

              [121.365616],
              [123.93075 ],
              [123.10966 ],
              [119.32182 ],
              [119.290596],
```

- After predicting the values we are now doing inverse transform so that it comes to its proper decimal values as all the data before inverse transform is in range 0-1.

- Also doing the inverse transform in the y test data

```
[108]: # now taking the Y - Test data with which we have to compare
       # y data is also in range 0 - 1 therefore i'll apply inverse transform here also

       inv_y_test = scaler.inverse_transform(y_test)

       inv_y_test
```

```
        [155.53999329],
        [158.36999512],
        [158.99000549],
        [160.27999878],
        [160.80999756],
        [163.24000549],
        [164.63999939],
        [163.07000732],
        [163.63999939],
        [162.99000549],
        [163.83000183],
        [165.28999329],
        [167.19000244],
        [168.41999817],
        [167.30999756],
        [167.21000671],
        [168.55999756],
        [164.38999939],
```

Step 4) Model Evaluation

[112]:
```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

 # inv_y_test =  actual stock prices
# inv_predictions = predicted stock prices

# Mean Absolute Error (MAE)
mae = mean_absolute_error(inv_y_test, inv_predictions)

# Mean Squared Error (MSE)
mse = mean_squared_error(inv_y_test, inv_predictions)

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Mean Absolute Percentage Error (MAPE)
mape = np.mean(np.abs((inv_y_test - inv_predictions) / inv_y_test)) * 100

# R-squared (R²)
r2 = r2_score(inv_y_test, inv_predictions)

# Print all metrics
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"\nMean Squared Error (MSE): {mse:.4f}")
print(f"\nRoot Mean Squared Error (RMSE): {rmse:.4f}")
print(f"\nMean Absolute Percentage Error (MAPE): {mape:.2f}%")
print(f"\nR-squared (R²): {r2:.4f}")
```

```
Mean Absolute Error (MAE): 2.0845

Mean Squared Error (MSE): 7.5954

Root Mean Squared Error (RMSE): 2.7560

Mean Absolute Percentage Error (MAPE): 1.37%
```

- Now Calculating the **Evaluation Metrics** one by one as :-

- **MAE = 2.08**
- **MSE = 7.5**
- **RMSE = 2.75**
- **MAPE = 1.37**
- **R2 = 0.97**

- ## <u>Comparing the original values along with the predicted values</u>

```
R-squared (R²): 0.9785
```

[114]:
```
ploting_data = pd.DataFrame(
 {
  'original_test_data': inv_y_test.reshape(-1), #reshape(-1) means converting our data in 1 dimensional array as our Y data is in 2D form we are conver
    'predictions': inv_predictions.reshape(-1)  #creating dictionary
 } ,
    index = google_data.index[splitting_len+100:]
)
ploting_data.head()
```

[114]:

Date	original_test_data	predictions
2023-06-20	123.849998	123.975052
2023-06-21	121.260002	123.798615
2023-06-22	123.870003	121.365616
2023-06-23	123.019997	123.930748
2023-06-26	119.089996	123.109657

[116]:
```
print(ploting_data)
```

```
            original_test_data  predictions
Date
2023-06-20          123.849998   123.975052
2023-06-21          121.260002   123.798615
2023-06-22          123.870003   121.365616
2023-06-23          123.019997   123.930748
2023-06-26          119.089996   123.109657
```

[118]:
```
pd.set_option('display.max_rows', None)  # Show all rows
pd.set_option('display.max_columns', None)  # Show all columns

print(ploting_data)

# Optionally, reset to default after viewing
# pd.reset_option('display.max_rows')
# pd.reset_option('display.max_columns')
```

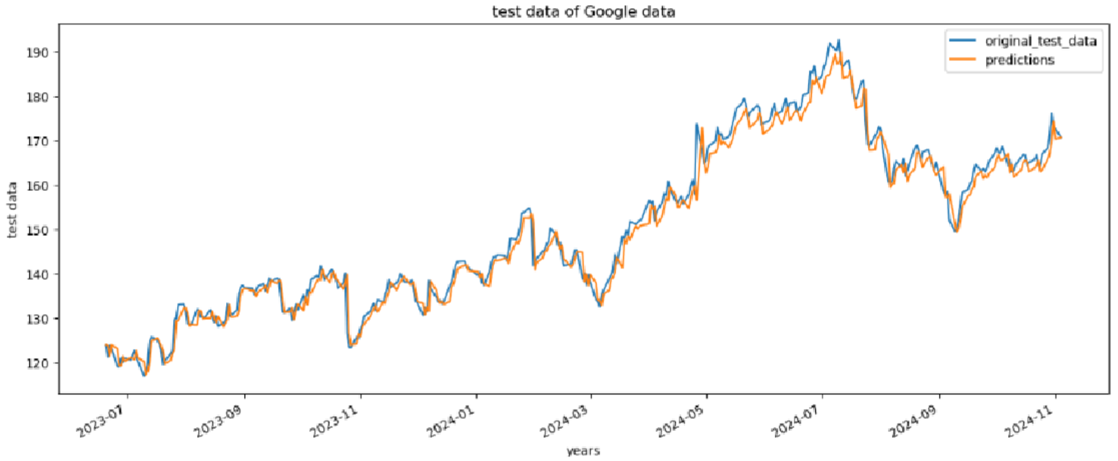
```
            original_test_data  predictions
Date
2023-06-20          123.849998   123.975052
2023-06-21          121.260002   123.798615
2023-06-22          123.870003   121.365616
2023-06-23          123.019997   123.930748
2023-06-26          119.089996   123.109657
2023-06-27          119.010002   119.321823
2023-06-28          121.080002   119.290596
2023-06-29          120.010002   121.321754
2023-06-30          120.970001   120.256859
2023-07-03          120.559998   121.107536
2023-07-05          122.629997   120.678009
2023-07-06          120.930000   122.627747
2023-07-07          120.139999   120.965637
2023-07-10          116.870003   120.203537
2023-07-11          117.709999   117.133858
2023-07-12          119.620003   117.992432
```

- **<u>Plotting the graph of Actual vs Predicted Values</u>**

2024-10-24	164.529999	163.059647
2024-10-25	166.990005	163.341049
2024-10-28	168.339996	165.845764
2024-10-29	171.139999	166.957962
2024-10-30	176.139999	169.627579
2024-10-31	172.690002	174.403336
2024-11-01	172.649994	170.322571
2024-11-04	170.679993	170.725510

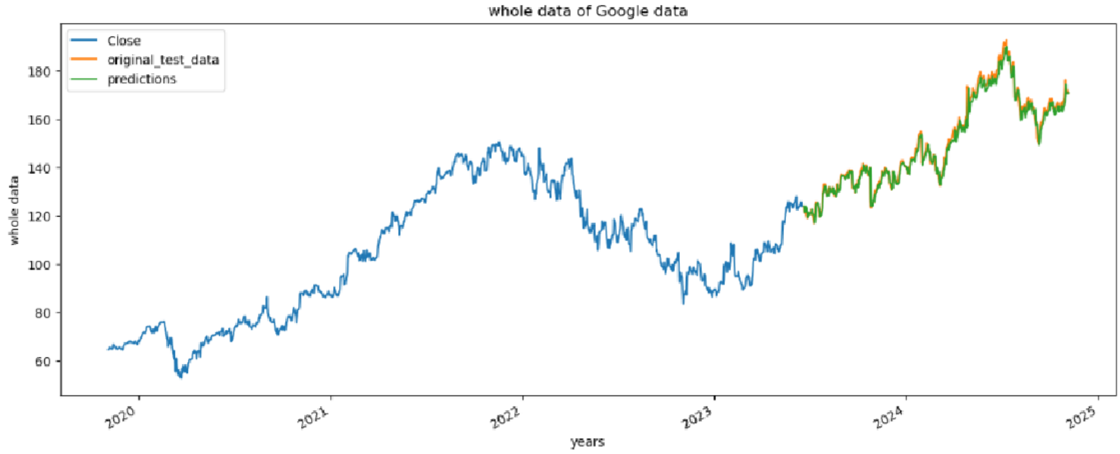
```
[120]: plot_graph((15,6), ploting_data, 'test data')
```

<Figure size 640x480 with 0 Axes>




```
[122]: plot_graph((15,6), pd.concat([close_price[:splitting_len+100],ploting_data], axis=0), 'whole data')
```

<Figure size 640x480 with 0 Axes>



- ## <u>Predicting the next 30 days values</u>

  - Now for predicting the next 30 days values we are using the y_test values

```
[162]: print("x train ",len(x_train))

       print("y train ",len(y_train))

       print("x test",len(x_test))

       print("y test",len(y_test))

x train  810
y train  810
x test 348
y test 348

[166]: x_input=y_test[248:].reshape(1,-1) #as my y test was 348 therefore i have to give 248 here i.e. for predicting say 6th november stock price i have to tal
       x_input.shape

[166]: (1, 100)

[170]: # converting my x_input in a list format
       temp_input=list(x_input)
       temp_input=temp_input[0].tolist()

       temp_input #these are my previous 100 days valuess

[170]: [0.8861466655848778,
        0.8978036866339834,
        0.9007358656178115,
        0.8840726427427996,
        0.8930837187297216,
        0.9113201914897298,
        0.9151105266153079,
        0.9493667142686519,
        0.9478648318959526,
        0.9585207435294409,
```

- Here basically we are reshaping the data again and again when we are adding the predicted values in the final output list, we are starting with 1 index ahead each time therefore we are taking values from [1 :] till end.

```
[194]: # demonstrate prediction for next 10 days
from numpy import array

lst_output=[]
n_steps=100
i=0

#this condition will keep on going untill we complete 30 loops
while(i<30):

    if(len(temp_input)>100):
        # print(temp_input)
        x_input=np.array(temp_input[1:])   #now i'm adding my yhat value here also kyuki ab jo values aai wo aai 101 values then jaise [1,2,3,4,5....100
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)                      #again doing reshape
        x_input = x_input.reshape((1, n_steps, 1))
        print(x_input)
        yhat = model.predict(x_input, verbose=0)           #again doing the predictions
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())   #adding my yhat values to final input
        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)) #in the very 1st step the code execution will come here and yhaa pe previous 100 days data idhr paas ki


        yhat = model.predict(x_input, verbose=0)

        print(yhat[0])

        temp_input.extend(yhat[0].tolist())
```

- The previous values are being given as an input and the new value as output is then added each time to the output list.

```
        temp_input.extend(yhat[0].tolist())

        print(len(temp_input))

        lst_output.extend(yhat.tolist()) #adding my yhat value to the final output value

        i=i+1


print(lst_output)   # list_output mei sab values aaegi
```

```
 [0.61943692]
 [0.61756301]
 [0.61572516]
 [0.61392254]
 [0.61215425]
 [0.61041945]]]
29 day output [[0.6087172]]
[0.6916232709815167, 0.6949845280362197, 0.7102889309370364, 0.7345328240632045, 0.754771843028357, 0.7592059043848842, 0.7684314102826071, 0.77222174
54081852, 0.7896001711510798, 0.7996123565536191, 0.7883844135753884, 0.7924607643733379, 0.7878122731060095, 0.7938195843475331, 0.8042608478209474,
0.8178489384382639, 0.8266453662651116, 0.8187070945800139, 0.8179920008367676, 0.8276465848053656, 0.7978244585085488, 0.8071930268047756, 0.78831282
78138183, 0.7891709839555683, 0.7987542004118691, 0.8118416271967404, 0.8157749155961858, 0.8146307437820646, 0.7986826146502988, 0.8025445355374472,
0.8079082296726583, 0.8152028842514434, 0.7984680756148614, 0.7988256770488027, 0.8164186418271349, 0.8260732257957326, 0.8460977057254477, 0.88185566
66268542, 0.857182695429811, 0.8568965706328033, 0.8428079253076781, 0.82955002784729, 0.8182655572891235, 0.8086355328559875, 0.8002366423606873, 0.7
926733493804932, 0.7856684923171997, 0.7790589928627014, 0.7727612853050232, 0.7667360305786133, 0.7609639167785645, 0.7554307579994202, 0.75012195110
32104, 0.7450212240219116, 0.7401109933853149, 0.7353742718696594, 0.73079514503479, 0.7263595461845398, 0.7220554947853088, 0.7178730368614197, 0.713
8038277626038, 0.7098411917686462, 0.7059794664382935, 0.702213704586029, 0.6985397934913635, 0.6949542164802551, 0.6914538741111755, 0.68803566694259
64, 0.684697151184082, 0.6814358234405518, 0.6782492399215698, 0.6751352548599243, 0.6720917224884033, 0.6691166162490845, 0.666208028793335, 0.663363
9335632324, 0.6605824828147888, 0.6578619480133057, 0.6552005410194397, 0.6525965332984924, 0.6500483751296997, 0.6475544571876526, 0.645113110542297
4, 0.6427230834960938, 0.6403828263282776, 0.6380900681320190, 0.6358461380004883, 0.6336470842361450, 0.6314925551414490, 0.6293813586235046, 0.62731242
```

- Inverse Transforming the values as they are in the range 0 – 1

```
[180]: next_30days_values = scaler.inverse_transform(lst_output)

       next_30days_values
```

```
[180]: array([[168.8261541 ],
              [167.24825784],
              [165.90170113],
              [164.72729265],
              [163.66972494],
              [162.69024276],
              [161.76604305],
              [160.88544088],
              [160.04293553],
              [159.23582661],
              [158.46213054],
              [157.71980537],
              [157.00657579],
              [156.31998314],
              [155.65765209],
              [155.01735736],
              [154.39713199],
              [153.79530078],
              [153.21047186],
              [152.64147841],
              [152.08738698],
              [151.54740577],
              [151.02084305],
              [150.50712373],
              [150.00575607],
              [149.51630669],
              [149.03834219],
              [148.57152086],
              [148.11549266],
              [147.66991587]])
```

- As the last value of our dataset was 4th November 2024, so after that date we will predict the values

```
[182]: google_data.tail()
```

[182]:

Date	Open	High	Low	Close	Adj Close	Volume	MA_for_250_days	MA_for_100_days
2024-10-29	169.384995	171.860001	168.660004	171.139999	171.139999	28916100	157.65508	170.2278
2024-10-30	182.410004	183.789993	175.744995	176.139999	176.139999	49698300	157.84936	170.2297
2024-10-31	174.720001	178.419998	172.559998	172.690002	172.690002	32801900	158.02580	170.1903
2024-11-01	171.539993	173.820007	170.309998	172.649994	172.649994	21752900	158.19492	170.1349
2024-11-04	171.240005	171.919998	169.485001	170.679993	170.679993	16194000	158.35184	170.0461

- We can either use this method also in which we are passing the last 60 days values in order to predict next 30 days values.

```
[240]: sequence_length = 60

# Use the last sequence of the scaled dataset as the input for prediction
last_60_days = y_test[-sequence_length:]  # Last 60 days
last_60_days = last_60_days.reshape((1, sequence_length, 1))  # Reshape for LSTM

# Predict the next 30 days
predicted_prices = []
for _ in range(30):
    next_day_prediction = model.predict(last_60_days)[0, 0]
    predicted_prices.append(next_day_prediction)

    # Append the prediction to the input sequence and reshape
    last_60_days = np.append(last_60_days[:, 1:, :], [[[next_day_prediction]]], axis=1)

# Transform predictions back to the original scale
predicted_prices = scaler.inverse_transform(np.array(predicted_prices).reshape(-1, 1))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

- These are the next 30 days predicted values

```
[242]: predicted_prices

[242]: array([[168.82614],
        [167.24825],
        [165.90169],
        [164.72726],
        [163.66971],
        [162.69022],
        [161.76602],
        [160.88542],
        [160.04292],
        [159.23582],
        [158.46211],
        [157.7198 ],
        [157.00656],
        [156.31996],
        [155.65764],
        [155.01733],
        [154.39713],
        [153.79527],
        [153.21042],
        [152.64143],
        [152.08736],
        [151.54738],
        [151.02081],
        [150.5071 ],
        [150.00572],
        [149.51628],
        [149.03831],
        [148.57149],
        [148.11543],
        [147.66986]], dtype=float32)
```

- From the datetime library we are now setting the date values along with their predicted prices.

```python
from datetime import datetime, timedelta

# Example predicted prices from LSTM
predicted_prices #actual predictions
predicted_prices = np.array(predicted_prices).reshape(-1)

# Define a starting date (manual specification)
start_date = datetime(2024, 11, 4)  # desired start date

# Generate future dates for predictions
future_dates = [start_date + timedelta(days=i) for i in range(1, len(predicted_prices) + 1)]

# Combine dates and predicted prices
predicted_df = pd.DataFrame({'Date': future_dates, 'Predicted Price': predicted_prices})

# Print the DataFrame
print(predicted_df)

# # Optionally, save to a CSV
# predicted_df.to_csv('predicted_prices.csv', index=False)
```

```
         Date  Predicted Price
0  2024-11-05       168.826141
1  2024-11-06       167.248245
2  2024-11-07       165.901688
3  2024-11-08       164.727264
4  2024-11-09       163.669708
5  2024-11-10       162.690216
6  2024-11-11       161.766022
7  2024-11-12       160.885422
8  2024-11-13       160.042923
9  2024-11-14       159.235825
10 2024-11-15       158.462112
11 2024-11-16       157.719803
12 2024-11-17       157.006561
13 2024-11-18       156.319962
```

```
14 2024-11-19        155.657639
15 2024-11-20        155.017334
16 2024-11-21        154.397125
17 2024-11-22        153.795273
18 2024-11-23        153.210419
19 2024-11-24        152.641434
20 2024-11-25        152.087357
21 2024-11-26        151.547379
22 2024-11-27        151.020813
23 2024-11-28        150.507095
24 2024-11-29        150.005722
25 2024-11-30        149.516281
26 2024-12-01        149.038315
27 2024-12-02        148.571487
28 2024-12-03        148.115433
29 2024-12-04        147.669861
```

```
[220]: import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(predicted_df['Date'], predicted_df['Predicted Price'], label='Predicted Prices', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Stock Price Prediction for Next 30 Days')
plt.legend()
plt.grid()
plt.show()
```

- After predicting the value's we are then plotting a graph to visualize the data more clearly.