Importing Dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Google.csv to Google (2).csv
```

```
import io
```

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv(io.BytesIO(uploaded['Google.csv']))
```

Observing Data

```
df.describe()
#df.head()
```

	Open	High	Low	Close	Adj Close	Volume
count	4041.000000	4041.000000	4041.000000	4041.000000	4041.000000	4.041000e+03
mean	533.983149	538.995819	528.658860	533.999060	533.999060	6.909802e+06
std	383.007917	386.590237	379.488087	383.326004	383.326004	7.895987e+06
min	49.644646	50.920921	48.028027	50.055054	50.055054	5.206000e+05
25%	241.211212	243.688690	238.873871	241.036041	241.036041	1.844600e+06
50%	342.592590	345.795807	338.598602	342.177185	342.177185	4.191600e+06
75%	791.979980	798.000000	786.200012	790.460022	790.460022	8.702600e+06
max	1699.520020	1726.099976	1660.189941	1717.390015	1717.390015	8.215110e+07

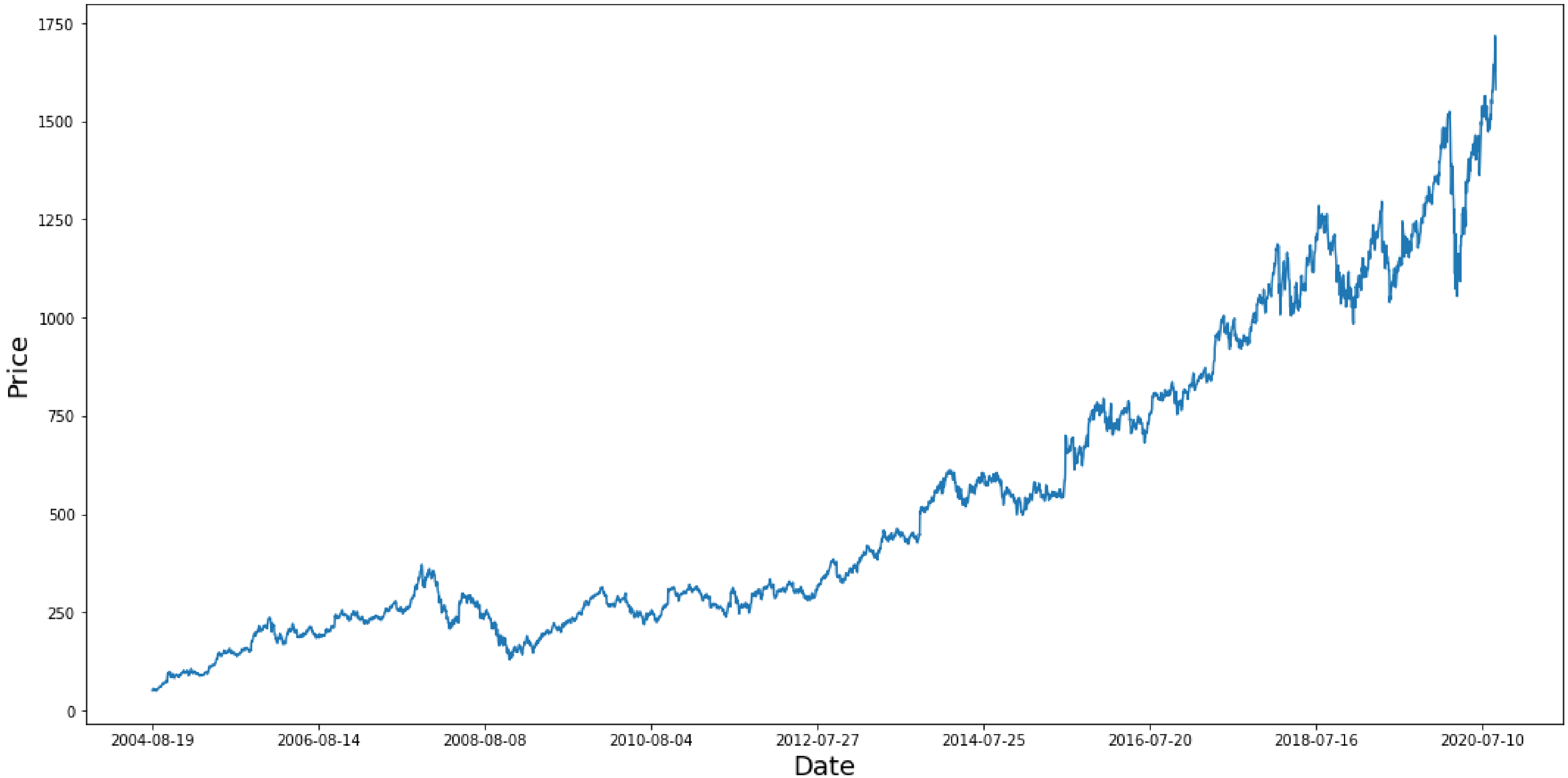
Sorting on the basis of Date

```
df = df.sort_values('Date')
df.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
4036	2020-08-31	1643.569946	1644.500000	1625.329956	1629.530029	1629.530029	1321100
4037	2020-09-01	1632.160034	1659.219971	1629.530029	1655.079956	1655.079956	1133800
4038	2020-09-02	1668.010010	1726.099976	1660.189941	1717.390015	1717.390015	2476100
4039	2020-09-03	1699.520020	1700.000000	1607.709961	1629.510010	1629.510010	3180200
4040	2020-09-04	1609.000000	1634.989990	1537.970093	1581.209961	1581.209961	2792533

Plotting the Google Stock Price Data

```
plt.figure(figsize = (18,9))
plt.plot(range(df.shape[0]),df['Close'])
plt.xticks(range(0,df.shape[0],500),df['Date'].loc[::500])
plt.xlabel('Date',fontsize=18)
plt.ylabel('Price',fontsize=18)
plt.show()
```



```
df.shape
```

```
(4041, 7)
```

Taking Closing Price of Stocks as Data

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

```
array([[  50.220219],
       [  54.209209],
       [  54.754753],
       ...,
       [1717.390015],
       [1629.51001 ],
       [1581.209961]])
```

Scaling the Data in range (0 , 1)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0, 1))
data_scaled = scaler.fit_transform(data)
data_scaled
```

```
array([[9.90592795e-05],
       [2.49149397e-03],
       [2.81868917e-03],
       ...,
       [1.00000000e+00],
       [9.47293131e-01],
       [9.18324717e-01]])
```

```
training_data_len = int(len(data)*0.80)
```

```
#print(training_data_len)
train_data = data_scaled[0 : training_data_len ,  :]

#print(train_data.shape)
#print(test_data.shape)
```

Creating X and Y Dataset

X will contain data of past 100 days and Y will contain the data of next day

```
def creating_X_Y_Dataset(data) :
      time_step = 100
      xData = []
      yData = []
      for i in range(time_step , len(train_data)) :
            xData.append(data[i-time_step : i , 0])
            yData.append(data[i , 0])
      return np.array(xData) , np.array(yData)

x_train , y_train = creating_X_Y_Dataset(train_data)
print(x_train.shape)
y_train.shape
```

```
(3132, 100)
(3132,)
```

Reshaping the data

Data can not be fed into LSTM without reshaping it in 3D array/matrix

```
x_train = np.reshape(x_train , (x_train.shape[0] , x_train.shape[1] , -1))
```

```
x_train = np.reshape(x_train , (x_train.shape[0] , x_train.shape[1] , 1))
print(x_train.shape)
y_train.shape
```

```
(3132, 100, 1)
(3132,)
```

Creating Model with Layers of LSTM

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

model = Sequential()

model.add(LSTM(units = 50, activation = "relu", return_sequences = True, input_shape = (x_train.shape[1], 1)))      # input shape is
model.add(Dropout(rate = 0.1))

model.add(LSTM(units = 50, activation = "relu", return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(rate = 0.1))

model.add(LSTM(units = 50, activation = "relu" , return_sequences = False))

model.add(Dense(units = 1))

model.compile(optimizer = "adam", loss = "mean_squared_error")

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
dropout_2 (Dropout)          (None, 100, 50)           0
```

```
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 100, 50)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

Fitting Training Data in the Model

```
model.fit(x_train , y_train , batch_size = 50 , epochs = 80 , verbose=1)
```

```
Epoch 1/80
63/63 [==============================] - 13s 154ms/step - loss: 0.0212
Epoch 2/80
63/63 [==============================] - 10s 152ms/step - loss: 2.9740e-04
Epoch 3/80
63/63 [==============================] - 9s 151ms/step - loss: 2.2615e-04
Epoch 4/80
63/63 [==============================] - 10s 153ms/step - loss: 2.0285e-04
Epoch 5/80
63/63 [==============================] - 10s 157ms/step - loss: 2.1694e-04
Epoch 6/80
63/63 [==============================] - 10s 156ms/step - loss: 1.9569e-04
Epoch 7/80
63/63 [==============================] - 10s 157ms/step - loss: 2.0499e-04
Epoch 8/80
63/63 [==============================] - 10s 157ms/step - loss: 1.8847e-04
Epoch 9/80
63/63 [==============================] - 10s 156ms/step - loss: 2.2689e-04
Epoch 10/80
63/63 [==============================] - 10s 156ms/step - loss: 1.7739e-04
```

```
Epoch 11/80
63/63 [==============================] - 10s 157ms/step - loss: 1.8411e-04
Epoch 12/80
63/63 [==============================] - 10s 155ms/step - loss: 1.6446e-04
Epoch 13/80
63/63 [==============================] - 10s 158ms/step - loss: 1.5981e-04
Epoch 14/80
63/63 [==============================] - 10s 157ms/step - loss: 1.8844e-04
Epoch 15/80
63/63 [==============================] - 10s 156ms/step - loss: 1.5531e-04
Epoch 16/80
63/63 [==============================] - 10s 157ms/step - loss: 1.4740e-04
Epoch 17/80
63/63 [==============================] - 10s 158ms/step - loss: 1.4182e-04
Epoch 18/80
63/63 [==============================] - 10s 157ms/step - loss: 1.6373e-04
Epoch 19/80
63/63 [==============================] - 10s 159ms/step - loss: 1.6247e-04
Epoch 20/80
63/63 [==============================] - 10s 160ms/step - loss: 1.3326e-04
Epoch 21/80
63/63 [==============================] - 10s 161ms/step - loss: 1.4682e-04
Epoch 22/80
63/63 [==============================] - 10s 158ms/step - loss: 1.6006e-04
Epoch 23/80
63/63 [==============================] - 10s 158ms/step - loss: 1.4902e-04
Epoch 24/80
63/63 [==============================] - 10s 160ms/step - loss: 1.3139e-04
Epoch 25/80
63/63 [==============================] - 10s 157ms/step - loss: 1.4545e-04
Epoch 26/80
63/63 [==============================] - 10s 156ms/step - loss: 1.3248e-04
Epoch 27/80
63/63 [==============================] - 10s 157ms/step - loss: 1.3968e-04
Epoch 28/80
63/63 [==============================] - 10s 157ms/step - loss: 1.2045e-04
Epoch 29/80
63/63 [==============================] - 10s 158ms/step - loss: 1.3149e-04
Epoch 30/80
```

Test Data Formation

```
time_step = 100
test_data = data_scaled[training_data_len-time_step : len(data_scaled) , : ]
x_test = []
y_test = data[training_data_len :  , :]
for i in range(time_step , len(test_data)) :
      x_test.append(test_data[i-time_step : i , 0])

x_test = np.array(x_test)
x_test.shape
```

```
(809, 100)
```

Reshaping the Test data in 3D array

LSTM takes 3D array as input

```
x_test = np.reshape(x_test , (x_test.shape[0] , x_test.shape[1] , 1))
print(x_test.shape)
y_test.shape
```

```
(809, 100, 1)
(809, 1)
```

Predicting Values

Prediction done on x_test and then inverse_transform used to unscale it as it has to be compared with original data

```
scaled_predict = model.predict(x_test)
predict = scaler.inverse_transform(scaled_predict)
predict
```

```
array([[ 944.27875],
       [ 945.04675],
       [ 946.45715],
       [ 948.7739 ],
       [ 950.38916],
       [ 949.48004],
       [ 947.8381 ],
       [ 944.29333],
       [ 939.23157],
       [ 932.96423],
       [ 927.4977 ],
       [ 922.89685],
       [ 920.2582 ],
       [ 919.92487],
       [ 921.2807 ],
       [ 924.5045 ],
       [ 928.64087],
       [ 933.4051 ],
       [ 937.94183],
       [ 942.577  ],
       [ 947.2338 ],
       [ 951.38086],
       [ 954.8617 ],
       [ 957.88934],
       [ 958.1663 ],
       [ 956.2646 ],
       [ 952.2804 ],
       [ 947.88837],
       [ 942.91046],
       [ 938.19574],
       [ 934.2941 ],
       [ 930.83887],
       [ 928.4027 ],
       [ 926.8988 ],
       [ 926.02075],
       [ 925.28723],
       [ 923.4515 ],
       [ 921.41797],
       [ 920.12866],
       [ 919.45197],
```

```
       [ 919.649  ],
       [ 919.1608 ],
       [ 917.9893 ],
       [ 916.04034],
       [ 915.14514],
       [ 915.30475],
       [ 915.712  ],
       [ 915.6619 ],
       [ 915.0349 ],
       [ 914.5955 ],
       [ 914.93604],
       [ 916.6334 ],
       [ 918.8443 ],
       [ 920.3173 ],
       [ 921.06714],
       [ 921.8255 ],
       [ 921.9702 ],
       [ 921.8096 ],
       [ 921.73804]
```

Root Mean Squared Error Between Actual And Predicted Values

```
RMSE = np.sqrt(np.mean(predict - y_test)**2)
RMSE
```

```
45.25778572731092
```

Plotting Actual and Predicted Stock Values

```
plt.figure(figsize=(14,5))
plt.plot(y_test,color = '#113377', label = 'Actual Stock Price')
plt.plot(predict, color = '#993311', label = 'Predicted Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time In Increasing Order')
plt.ylabel('Google Stock Price')
```

```
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```