Stock Price Prediction

-Stock Price Prediction using machine learning helps you discover the future value of company stock and other financial assets traded on an exchange.

-The entire idea of predicting stock prices is to gain significant profits.

-Predicting how the stock market will perform is a hard task to do. There are other factors involved in the prediction, such as physical and psychological factors, rational and irrational behavior, and so on.

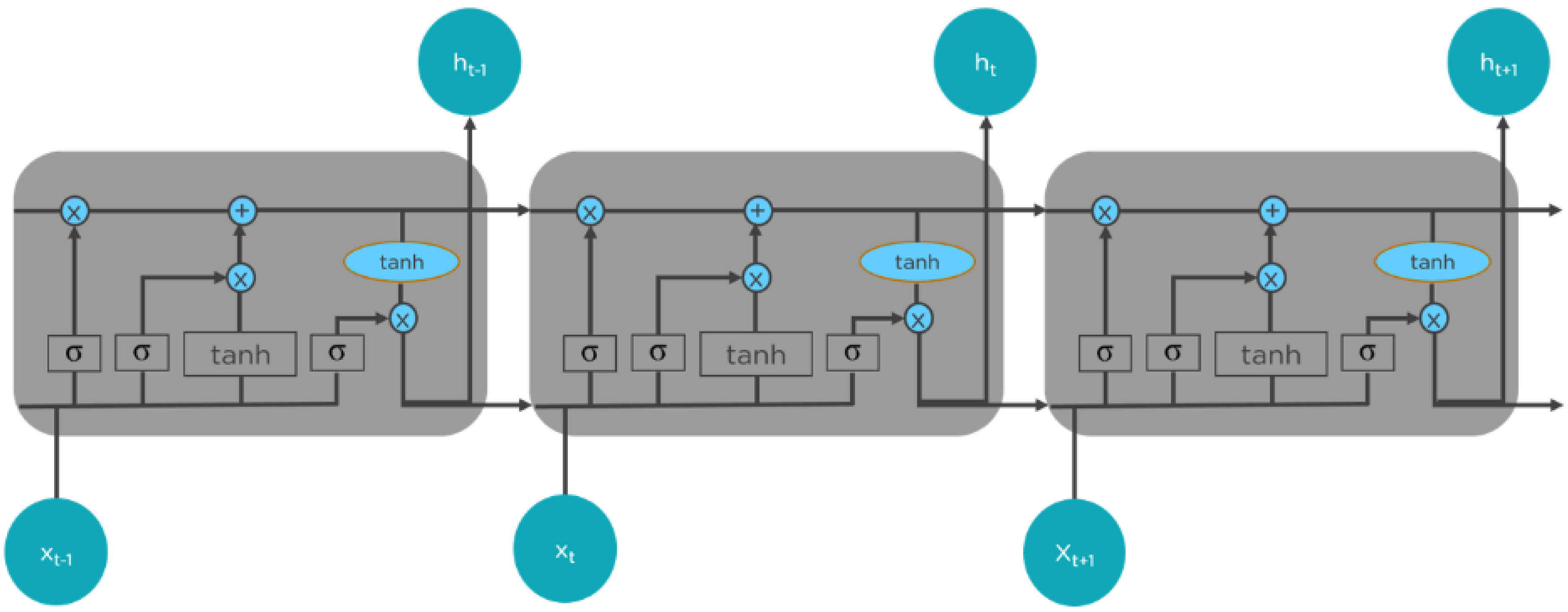
-All these factors combine to make share prices dynamic and volatile. This makes it very difficult to predict stock prices with high accuracy.

LSTM

Long Short Term Memory Network is used for building model to predict the stock prices of Google.

LTSMs are a type of Recurrent Neural Network for learning long-term dependencies.

It is commonly used for processing and predicting time-series data.



LOADING DATASET

```
import pandas as pd
df=pd.read_csv("/content/Google_Stock_Price_Train.csv",index_col="Date",parse_dates=True)
df
```

Date	Open	High	Low	Close	Volume
2012-01-03	325.25	332.83	324.97	663.59	7,380,500
2012-01-04	331.27	333.87	329.08	666.45	5,749,400
2012-01-05	329.83	330.75	326.89	657.21	6,590,300
2012-01-06	328.34	328.77	323.68	648.24	5,405,900
2012-01-09	322.04	322.29	309.46	620.76	11,688,800
...	...	...	...	...	...
2016-12-23	790.90	792.74	787.28	789.91	623,400
2016-12-27	790.68	797.86	787.66	791.55	789,100
2016-12-28	793.70	794.23	783.20	785.05	1,153,800
2016-12-29	783.33	785.93	778.92	782.79	744,300
2016-12-30	782.75	782.78	770.41	771.82	1,770,000

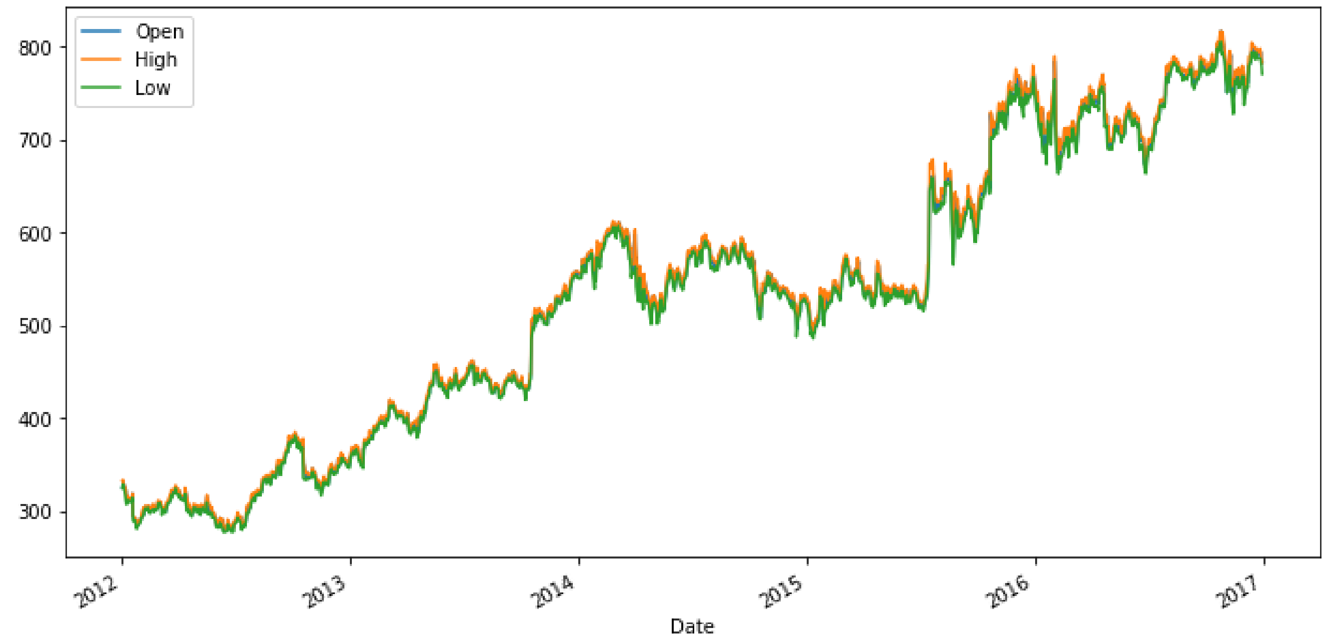
1258 rows × 5 columns

```
df1=pd.read_csv("/content/Google_Stock_Price_Test.csv")
df1
```

	Date	Open	High	Low	Close	Volume
0	1/3/2017	778.81	789.63	775.80	786.14	1,657,300
1	1/4/2017	788.36	791.34	783.16	786.90	1,073,000
2	1/5/2017	786.08	794.48	785.02	794.02	1,335,200
3	1/6/2017	795.26	807.90	792.20	806.15	1,640,200
4	1/9/2017	806.40	809.97	802.83	806.65	1,272,400
5	1/10/2017	807.86	809.13	803.51	804.79	1,176,800
6	1/11/2017	805.00	808.15	801.37	807.91	1,065,900
7	1/12/2017	807.14	807.39	799.17	806.36	1,353,100
8	1/13/2017	807.48	811.22	806.69	807.88	1,099,200
9	1/17/2017	807.08	807.14	800.37	804.61	1,362,100
10	1/18/2017	805.81	806.21	800.99	806.07	1,294,400
11	1/19/2017	805.12	809.48	801.80	802.17	919,300
12	1/20/2017	806.91	806.91	801.69	805.02	1,670,000
13	1/23/2017	807.25	820.87	803.74	819.31	1,963,600
14	1/24/2017	822.30	825.90	817.82	823.87	1,474,000
15	1/25/2017	829.62	835.77	825.06	835.67	1,494,500
16	1/26/2017	837.81	838.00	827.01	832.15	2,973,900
17	1/27/2017	834.71	841.95	820.44	823.31	2,965,800
18	1/30/2017	814.66	815.84	799.80	802.32	3,246,600
19	1/31/2017	796.86	801.25	790.52	796.79	2,160,600

```
df.plot(figsize=(12,6))
```

```
<AxesSubplot:xlabel='Date'>
```




```
len(df)
```

```
1258
```

```
train=df.iloc[:,0:1]
```

```
train
```

	Open
Date	
2012-01-03	325.25
2012-01-04	331.27
2012-01-05	329.83
2012-01-06	328.34
2012-01-09	322.04
...	...

```
train.shape
```

```
(1258, 1)
```

```
test=df1.iloc[:,1:2]
```

```
test
```

	Open
0	778.81
1	788.36
2	786.08
3	795.26
4	806.40
5	807.86
6	805.00
7	807.14
8	807.48
9	807.08
10	805.81
11	805.12
12	806.91
13	807.25
14	822.30
15	829.62
16	837.81
17	834.71
18	814.66
19	796.86

```
test.shape
```

```
(20, 1)
```

NORMALIZING THE DATASET

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(train)
scaled_train=scaler.transform(train)
scaled_test=scaler.transform(test)
```

```
scaled_train[:10]
```

```
array([[0.08581368],
       [0.09701243],
       [0.09433366],
       [0.09156187],
       [0.07984225],
       [0.0643277 ],
       [0.0585423 ],
       [0.06568569],
       [0.06109085],
       [0.06639259]])
```

```
scaled_test
```

```
array([[0.92955205],
       [0.94731751],
       [0.94307612],
       [0.96015329],
       [0.98087655],
       [0.98359253],
       [0.97827219],
       [0.98225314],
       [0.98288563],
       [0.98214153],
       [0.979779  ],
       [0.97849542],
       [0.98182528],
       [0.98245777],
       [1.01045465],
       [1.02407173],
       [1.03930724],
       [1.03354044],
       [0.99624228],
       [0.9631297 ]])
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
n_inputs=3
n_features=1
generator=TimeseriesGenerator(scaled_train,scaled_train,length=n_inputs,batch_size=1)
```

```
X,y=generator[1]
print("Input",X)
print("Generated:",y)
```

```
Input [[[0.09701243]
  [0.09433366]
  [0.09156187]]]
Generated: [[0.07984225]]
```

```
n_inputs=20
generator=TimeseriesGenerator(scaled_train,scaled_train,length=n_inputs,batch_size=1)
```

Building the Model by Importing the Libraries and Adding Different Layers to LSTM

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

```
model=Sequential()
model.add(LSTM(100,activation='relu',return_sequences=True,input_shape=(n_inputs,n_features))
model.add(LSTM(120,activation='relu'))
model.add(Dense(1))
```

Fitting the Model

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

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_4 (LSTM)               (None, 20, 100)           40800     
                                                                 
 lstm_5 (LSTM)               (None, 120)               106080    
                                                                 
 dense_2 (Dense)             (None, 1)                 121       
                                                                 
=================================================================
Total params: 147,001
Trainable params: 147,001
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(generator,epochs=10,batch_size=2)
```

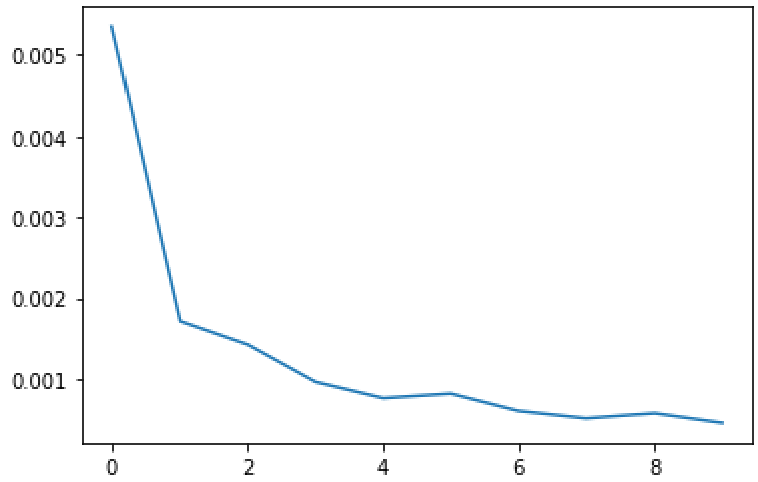
```
Epoch 1/10
1238/1238 [==============================] - 31s 23ms/step - loss: 0.0053
Epoch 2/10
1238/1238 [==============================] - 29s 23ms/step - loss: 0.0017
Epoch 3/10
1238/1238 [==============================] - 29s 23ms/step - loss: 0.0014
Epoch 4/10
1238/1238 [==============================] - 28s 23ms/step - loss: 9.6389e-04
Epoch 5/10
1238/1238 [==============================] - 29s 24ms/step - loss: 7.6372e-04
Epoch 6/10
1238/1238 [==============================] - 29s 23ms/step - loss: 8.1921e-04
Epoch 7/10
1238/1238 [==============================] - 28s 23ms/step - loss: 6.0493e-04
Epoch 8/10
1238/1238 [==============================] - 29s 23ms/step - loss: 5.1383e-04
Epoch 9/10
1238/1238 [==============================] - 28s 23ms/step - loss: 5.7640e-04
Epoch 10/10
1238/1238 [==============================] - 28s 23ms/step - loss: 4.5729e-04
<keras.callbacks.History at 0x7f63b2a00640>
```

```
loss_per_epoch=model.history.history["loss"]
loss_per_epoch
```

```
[0.005348223727196455,
 0.0017143437871709466,
 0.0014249780215322971,
 0.0009638919145800173,
 0.0007637225207872689,
 0.000819212116766721,
 0.0006049296353012323,
 0.000513831153512001,
 0.0005764021771028638,
 0.00045728866825811565]
```

```
import matplotlib.pyplot as plt
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

```
[<matplotlib.lines.Line2D at 0x7f63aec3cf70>]
```




```
last_train_batch=scaled_train[-20:]
```

```
last_train_batch=last_train_batch.reshape(1,n_inputs,n_features)
```

```
model.predict(last_train_batch)
```

```
1/1 [==============================] - 0s 311ms/step
array([[0.96708816]], dtype=float32)
```

```
scaled_test
```

```
array([[0.92955205],
       [0.94731751],
       [0.94307612],
       [0.96015329],
       [0.98087655],
       [0.98359253],
       [0.97827219],
       [0.98225314],
       [0.98288563],
       [0.98214153],
       [0.979779  ],
       [0.97849542],
       [0.98182528],
       [0.98245777],
       [1.01045465],
       [1.02407173],
       [1.03930724],
       [1.03354044],
       [0.99624228],
       [0.9631297 ]])
```

```
scaled_train[-20:]
```

```
array([[0.86589404],
       [0.89030062],
       [0.90335962],
       [0.89642086],
       [0.91777662],
       [0.93176576],
       [0.94114145],
       [0.95762334],
       [0.96413424],
       [0.96402262],
       [0.96971501],
       [0.95077759],
       [0.96294367],
       [0.96123223],
       [0.95475854],
       [0.95204256],
       [0.95163331],
       [0.95725128],
       [0.93796041],
       [0.93688146]])
```

```
test_predictions=[]
first_eval_batch=scaled_train[-n_inputs:]
current_batch=first_eval_batch.reshape(1,n_inputs,n_features)
```

```
import numpy as np
for i in range(len(test)):
  current_pred=model.predict(current_batch)
  test_predictions.append(current_pred)
  current_batch=np.append(current_batch[:,1:,:],[current_pred],axis=1)
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 38ms/step
```

```
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
```

```
test_predictions
```

```
[array([[0.96708816]], dtype=float32),
 array([[0.994685]], dtype=float32),
 array([[1.0219613]], dtype=float32),
 array([[1.0486861]], dtype=float32),
 array([[1.0747463]], dtype=float32),
 array([[1.1003034]], dtype=float32),
 array([[1.1255462]], dtype=float32),
 array([[1.1506323]], dtype=float32),
 array([[1.1756778]], dtype=float32),
 array([[1.200766]], dtype=float32),
 array([[1.2259517]], dtype=float32),
 array([[1.2512852]], dtype=float32),
 array([[1.2767829]], dtype=float32),
 array([[1.3023021]], dtype=float32),
 array([[1.3277144]], dtype=float32),
 array([[1.3529316]], dtype=float32),
 array([[1.3779018]], dtype=float32),
 array([[1.4025879]], dtype=float32),
 array([[1.4269595]], dtype=float32),
 array([[1.4509915]], dtype=float32)]
```

```
predictions=current_batch
predictions
```

```
array([[[0.96708816],
        [0.99468499],
        [1.02196133],
        [1.04868615],
        [1.07474625],
        [1.10030341],
        [1.12554622],
        [1.15063226],
        [1.17567778],
        [1.20076597],
        [1.22595167],
        [1.2512852 ],
        [1.27678287],
        [1.30230212],
        [1.32771444],
        [1.35293162],
        [1.37790179],
        [1.40258789],
        [1.42695951],
        [1.45099151]]])
```

```
prediction=predictions.reshape(20,1)
```

```
y_pred=scaler.inverse_transform(prediction)
y_pred
```

```
array([[ 798.98791285],
       [ 813.82286549],
       [ 828.48553329],
       [ 842.85172504],
       [ 856.86059474],
       [ 870.59910188],
       [ 884.16862439],
       [ 897.65387888],
       [ 911.11734545],
       [ 924.60375342],
       [ 938.14258059],
       [ 951.76086961],
       [ 965.46739975],
       [ 979.18552876],
       [ 992.84617609],
       [1006.40192094],
       [1019.82488759],
       [1033.09514648],
       [1046.19635668],
       [1059.11499684]])
```

Calculating the metrics

```
from sklearn.metrics import mean_squared_error,r2_score
print("Mean Squarred Error",mean_squared_error(scaled_test,prediction))
print("r2_score",r2_score(scaled_test,prediction))
```

```
Mean Squarred Error 0.06913200816247353
r2_score -90.91662970343795
```

```
test["Open"]
```

```
0     778.81
1     788.36
2     786.08
3     795.26
4     806.40
5     807.86
6     805.00
7     807.14
8     807.48
9     807.08
10    805.81
11    805.12
12    806.91
13    807.25
14    822.30
15    829.62
16    837.81
17    834.71
18    814.66
19    796.86
Name: Open, dtype: float64
```

```
y_pred1=y_pred.flatten()
```

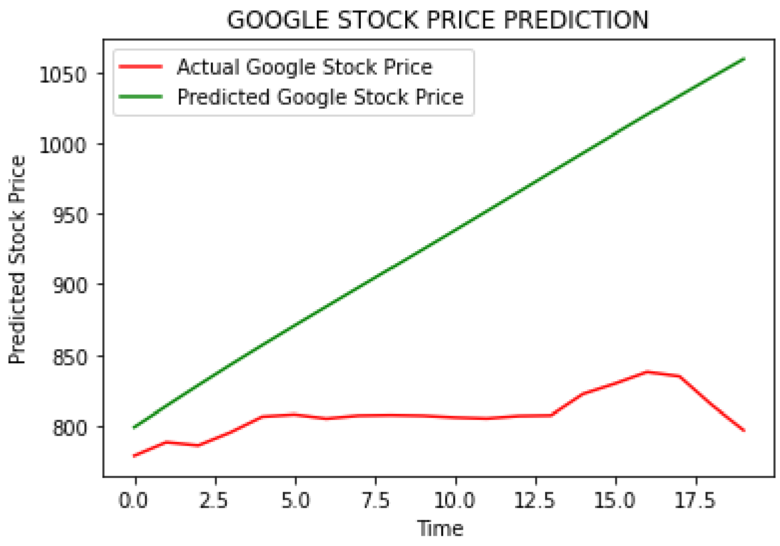
Displaying the actual and predicted values

```
df=pd.DataFrame({"Actual":test["Open"],"Predicted":y_pred1})
df
```

	Actual	Predicted
0	778.81	798.987913
1	788.36	813.822865
2	786.08	828.485533
3	795.26	842.851725
4	806.40	856.860595
5	807.86	870.599102
6	805.00	884.168624
7	807.14	897.653879
8	807.48	911.117345
9	807.08	924.603753
10	805.81	938.142581
11	805.12	951.760870
12	806.91	965.467400
13	807.25	979.185529
14	822.30	992.846176
15	829.62	1006.401921
16	837.81	1019.824888
17	834.71	1033.095146
18	814.66	1046.196357
19	796.86	1059.114997

```
import matplotlib.pyplot as plt
plt.plot(test,color="red",label="Actual Google Stock Price")
plt.plot(y_pred1,color="green",label="Predicted Google Stock Price")
plt.title("GOOGLE STOCK PRICE PREDICTION")
plt.xlabel("Time")
plt.ylabel("Predicted Stock Price")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f63b27ed280>
```



✓ 0s completed at 3:13 PM