GOOGLE STOCK PRICE PREDICTION

USING LONG SHORT TERM MEMORY

GIRI PRASATH S
8-2-2022

Abstract:

The art of forecasting the stock prices has been a difficult task for many of the researchers and analysts. In fact, investors are highly interested in the research area of stock price prediction. For a good and successful investment, many investors are keen in knowing the future situation of the stock market.

Good and effective prediction systems for stock market help traders, investors, and analyst by providing supportive information like the future direction of the stock market. In this work, we present a Long Short-Term Memory (LSTM) approach to predict stock market indices. The dataset of Google stock for the past 10 years is used to forecast the information of the future direction of Google stock.

Dataset- https://github.com/giri00777/Google-Stock-Price-Prediction/blob/main/GOOGL%20(1).csv

Code-https://github.com/giri00777/Google-Stock-Price-Prediction/blob/main/Google%20Stock%20price%20prediction%20(1).ipynb

ACKNOWLEDGE

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of my capstone project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, I have fortunate to have Mr. ANBU JOEL as my mentor. He has readily shared his immense knowledge in data analytics and guide me in a manner that the outcome resulted in enhancing my data skills.

I certify that the work done by me for conceptualizing and completing this project is original and authentic.

Date: 03/08/2022 Name: Giri Prasath S

CERTIFICATION OF COMPLETION

I certify that the project titled "Google Stock Price Prediction" was undertaken and completed.

(2nd August 2022).

Mentor Mr. ANBU JOEL

Date: 2nd August, 2022

Place: Trichy.

Table Of Content

INTRODUCTION

Data analysis have been used in all business for data-driven decision making. In share market, there are many factors that drive the share price, and the pattern of the change of price is not regular. This is why it is tough to take a robust decision on future price. Artificial Neural Network (ANN) has the capability to learn from the past data and make the decision over future. Deep learning networks. such as Recurrent Neural Network (RNN) etc. works great with multivariate time series data. We train our model from the past stock data and calculate the future price of that stock. This future price use to calculate the future growth of a company. Moreover, we found a future growth curve from different companies. Thus we can analyze and investigate the similarity of one company's future curve over another. Stock price of a listed company in a stock exchange varies every time an order is placed for sell or buy and a transaction completes. An exchange collects all sell bids with expected price per stock and all buy bids with or without a price limit and a buy sell transaction is committed when both bids have a match i.e. selling bid price is same with buying bid price of some buy-bid Fame in 1970 proposed efficient market hypothesis which says that in an efficient market the effect all market events are already incorporated in stock prices hence it is not possible to predict using past events or prices. The stock price of a company depends on many intrinsic as well as extrinsic attributes. Macro-economic conditions too play an important role in growth or decline of a sector as a whole. Some of the intrinsic factors could be company's net profit, liabilities, demand stability, competition in market, technically advanced assembly line, surplus cash for adverse situations, stakes in raw material supplier and finished product distributors etc.

LSTM Architecture

An overview of Recurrent Neural Network (RNN)

Classic RNNs have short memory, and were neither popular nor powerful for this exact reason. But a recent major improvement in Recurrent Neural Networks gave rise to the popularity of LSTMs (Long Short Term Memory RNNs) which has completely changed the playing field.

Predicted stock price for Google's trending Stock data

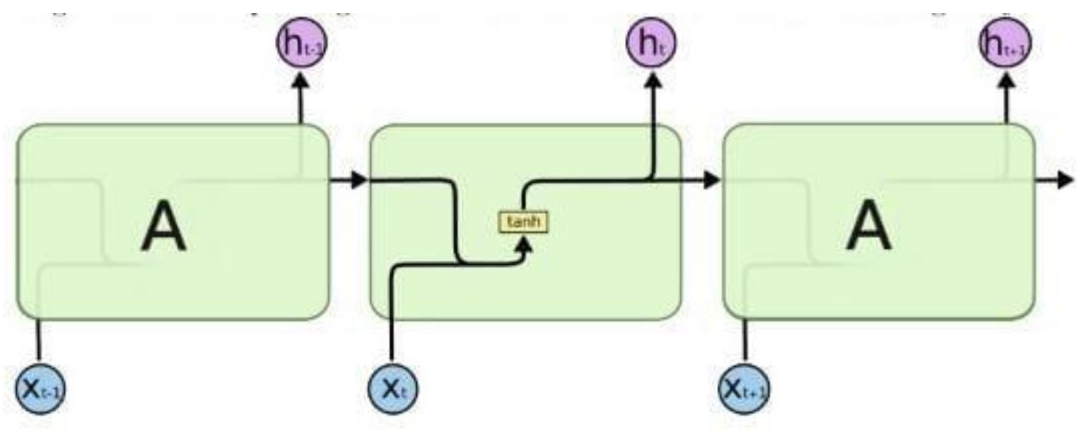
1. Successfully predicted Google stock price by using last 5 year's data of Google stock price.
2. Use of RNN(LSTM) was implemented alongside with Keras framework producing some good results.

Technology Used: Deep Learning , RNN , Machine Learning , LSTM

Initially (at time step t) for some input Xt the RNN generates an output of ht. In the next time step (t+1) the RNN takes two input Xt+1 and ht to generate the output ht+1. A loop allows information to be passed from one step of the network to the next. RNNs are not free from limitations though. When the ‘context’ is from near past it works great towards the correct output. But when an RNN has to depend on a distant ‘context’ (i.e. something learned long past) to produce.

LSTM Networks

Hoch Reiter Schmid Huber [10] introduced a special type of RNN which is capable of learning long term dependencies. Later on many other researchers improved upon this pioneering work in [11] [12] [13] [14]. LSTMs are perfected over the time to mitigate the long-term dependency issue. The evolution and development of LSTM from RNNs are explained in [15] [16]. Recurrent neural networks are in the form of a chain of repeating modules of the neural network. In standard RNNs, this repeating module has a simple structure.



LSTMs follow this chain-like structure, however the repeating module has a different structure. Instead of having a single neural network layer, there are four layers, interacting in a very special way. In every line represents an entire feature vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

The Working of LSTM

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is like a conveyor belt. This runs straight down the entire chain, having some minor linear interactions. LSTM has the ability to add or remove information to the cell state, controlled by structures called gates. Gates are used for optionally let information through. Gates are composed of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between 0 and 1, describing how much of each component should be let through. A value of 0 means "let nothing through," while a value of 1 means "let everything through! "An LSTM has three of these gates, to protect and control the cell state The first step of LSTM is to decide what information are to be thrown out from the cell state. It is made by a sigmoid layer called the "forget gate layer." It looks at ht, ht−1and txt, and outputs a number between 00 and 11 for each number in the cell state ct−1ct−1. A 11 represents "completely keep this" while a 00 represents "completely remove this." In the next step it is decided what new information are going to be stored in the cell state. It has two parts. First, a sigmoid layer called the "input gate layer" decides which values are to be updated. Thereafter, a tanh layer creates a vector of new candidate values, that could be added to the state. In the next step, these two are combined to create an update to the state. It is now time to update the old cell state, ct−1ct−1, into the new cell state . Then we add it C tic C t. This is the new candidate values, scaled by how much we decide to update each state value. Finally, we need to decide on the output. The output will be a filtered version of the cell state. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh-tanh (to push the values to be between 11 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

CONCLUSION

In present, there are several models to predict the stock market but they are less accurate. We proposed a model that uses RNN and LSTM to predict the trend in stock prices that would be more accurate. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. In this work by increasing the Epochs and batch size, the accuracy of prediction is more. In proposed method, we are using a test data that is used to predict which gives results that are more accurate with the test data. The proposed method is capable of tracing and prediction of stock market and the prediction will produce higher and accurate results. In our above model will be more useful to stock analysts, Business analysts, Stock Market Investors.

In [1]:

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
data=pd.read_csv(r"C:\Users\91638\Downloads\GOOGL.csv",na_values="None")
```

In [3]:

```
data
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	44659096
1	2004-08-20	50.555557	54.594597	50.300301	54.209209	54.209209	22834343
2	2004-08-23	55.430431	56.796799	54.579578	54.754753	54.754753	18256126
3	2004-08-24	55.675674	55.855858	51.836838	52.487488	52.487488	15247337
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	9188602
...	...	...	...	...	...	...	...
4426	2022-03-18	2668.489990	2724.879883	2645.169922	2722.510010	2722.510010	2223100
4427	2022-03-21	2723.270020	2741.000000	2681.850098	2722.030029	2722.030029	1341600
4428	2022-03-22	2722.030029	2821.000000	2722.030029	2797.360107	2797.360107	1774800
4429	2022-03-23	2774.050049	2791.770020	2756.699951	2765.510010	2765.510010	1257700
4430	2022-03-24	2784.000000	2832.379883	2755.010010	2831.439941	2831.439941	1317900

4431 rows × 7 columns

In [4]:

```
data1=data['Close']
data1
```

Out[4]:

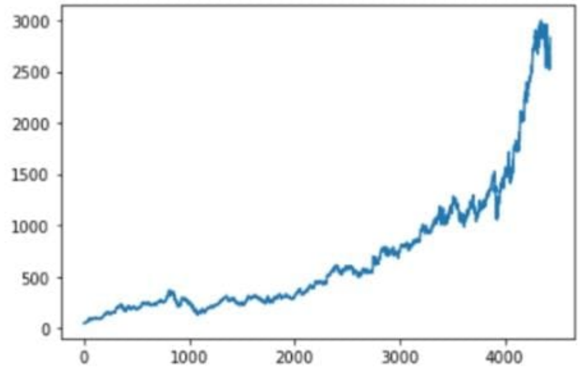
```
0         50.220219
1         54.209209
2         54.754753
3         52.487488
4         53.053055
            ...
4426    2722.510010
4427    2722.030029
4428    2797.360107
4429    2765.510010
4430    2831.439941
Name: Close, Length: 4431, dtype: float64
```

In [5]:

```
import matplotlib.pyplot as plt
plt.plot(data1)
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x28fb70866a0>]
```



In [6]:

```
import numpy as np
```

In [7]:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
data1=scaler.fit_transform(np.array(data1).reshape(-1,1))
```

In [8]:

```
data1
```

Out[8]:

```
array([[5.60505519e-05],
       [1.40975800e-03],
       [1.59489433e-03],
       ...,
       [9.32328062e-01],
       [9.21519383e-01],
       [9.43893427e-01]])
```

In [9]:

```
data1.ndim
```

Out[9]:

```
2
```

In [10]:

```
training_size=int(len(data1)*0.65)
test_size=len(data1)-training_size
train_data,test_data=data1[0:training_size,:],data1[training_size:len(data1),:1]
```

In [11]:

```
training_size,test_size
```

Out[11]:

```
(2880, 1551)
```

In [12]:

```
train_data
```

Out[12]:

```
array([[5.60505519e-05],
       [1.40975800e-03],
       [1.59489433e-03],
       ...,
       [2.31975250e-01],
       [2.32032936e-01],
       [2.26531908e-01]])
```

In [13]:

```
import numpy
# convert an array of values into a dataset matrix
```

```
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]    ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

In [14]:

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

In [15]:

```
print(X_train.shape), print(y_train.shape)
```

```
(2779, 100)
(2779,)
```

Out[15]:

```
(None, None)
```

In [16]:

```
print(X_test.shape), print(ytest.shape)
```

```
(1450, 100)
(1450,)
```

Out[16]:

```
(None, None)
```

In [17]:

```
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 [18]:

```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [19]:

```
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 [20]:

```
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 [21]:

```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=3,batch_size=64,verbose
```

4/16

```
Epoch 1/3
44/44 [==============================] - 33s 484ms/step - loss: 8.9526e-04 - val_loss:
020
Epoch 2/3
44/44 [==============================] - 18s 405ms/step - loss: 4.1611e-05 - val_loss: 6.1
518e-04
Epoch 3/3
44/44 [==============================] - 17s 395ms/step - loss: 3.6007e-05 - val_loss: 5.7
361e-04
```

Out[21]:

```
<keras.callbacks.History at 0x28fc209e5e0>
```

In [22]:

```
import tensorflow as tf
```

In [23]:

```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

In [24]:

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [25]:

```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[25]:

```
356.54854677666646
```

In [26]:

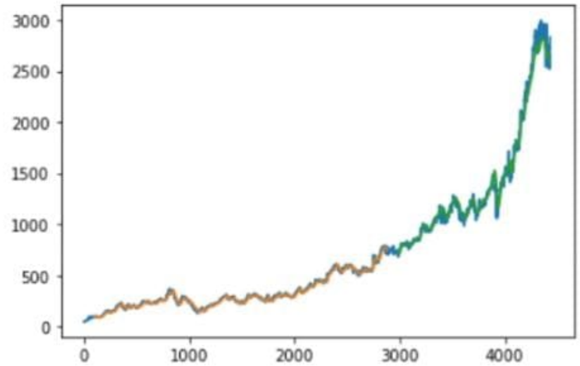
```
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[26]:

```
1539.122901118341
```

In [27]:

```
### Plotting
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(data1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(data1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(data1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [28]:
```
len(test_data)
```
Out[28]:
```
1551
```
In [29]:
```
x_input=test_data[1451:].reshape(1,-1)
x_input.shape
```
Out[29]:
```
(1, 100)
```
In [30]:
```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```
In [31]:
```
temp_input
```
Out[31]:
```
[0.956958823481945,
 0.9700954727495689,
 0.9780059965935639,
 0.9893373053170964,
 0.9933044148390155,
 0.9945193535220263,
 0.9937557913770747,
 0.9732244537017088,
 0.9723624636452197,
 0.9921234455087096,
 0.9905895272125209,
 0.9866800825825106,
 0.9878406702333218,
 1.0,
 0.9938100592658408,
 0.9759970062201122,
 0.97246760275897,
 0.9747616858576066,
 0.9480404077874425,
 0.9707606897870554,
 0.9461026699791092,
 0.9403607091192273,
 0.9533548532566145,
 0.9468085672321521,
 0.9546376478409617,
 0.982563591119997,
 0.9887875005281388,
 0.9850681180542793,
 0.9875318816295717,
 0.9727696801605071,
```

```
0.9067639747413561,
0.9323280618244906,
0.9215193825434963,
0.9438934267794395]
```

In [32]:

```python
# demonstrate prediction for next 30 days
from numpy import array

lst_output=[]
n_steps=100
i=0
while(i<30):

    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())
        i=i+1
    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)
```

```
[0.86387885]
101
1 day input [0.97009547 0.978006   0.98933731 0.99330441 0.99451935 0.99375579
 0.97322445 0.97236246 0.99212345 0.99058953 0.98668008 0.98784067
 1.         0.99381006 0.97599701 0.9724676  0.97476169 0.94804041
 0.97076069 0.94610267 0.94036071 0.95335485 0.94680857 0.95463765
 0.98256359 0.9887875  0.98506812 0.98753188 0.97276968 0.95974157
 0.97694044 0.96339309 0.94493189 0.94413096 0.95679254 0.97676397
 0.98016777 0.98688705 0.97861007 0.97839291 0.97530809 0.96615554
 0.96710237 0.96308431 0.91812238 0.91793571 0.91297769 0.92419351
 0.9314321  0.94293309 0.92363359 0.92969801 0.90606147 0.90007858
 0.88780044 0.86773747 0.8708087  0.8445489  0.86014592 0.8585985
 0.88809573 0.90134779 0.91723321 0.98752169 0.95419646 0.95557429
 0.92780096 0.92914481 0.94402922 0.92385754 0.89441798 0.90285793
 0.91020506 0.91787125 0.88258451 0.86808702 0.86408594 0.84898098
 0.88361618 0.89561933 0.89967468 0.89291803 0.8963795  0.89181851
 0.87829154 0.84077186 0.84569938 0.888564   0.88184133 0.86447277
 0.8378703  0.85990838 0.88761726 0.89140789 0.90692686 0.90676397
 0.93232806 0.92151938 0.94389343 0.86387885]
1 day output [[0.8658021]]
2 day input [0.978006   0.98933731 0.99330441 0.99451935 0.99375579 0.97322445
 0.97236246 0.99212345 0.99058953 0.98668008 0.98784067 1.
 0.99381006 0.97599701 0.9724676  0.97476169 0.94804041 0.97076069
 0.94610267 0.94036071 0.95335485 0.94680857 0.95463765 0.98256359
 0.9887875  0.98506812 0.98753188 0.97276968 0.95974157 0.97694044
 0.96339309 0.94493189 0.94413096 0.95679254 0.97676397 0.98016777
 0.98688705 0.97861007 0.97839291 0.97530809 0.96615554 0.96710237
 0.96308431 0.91812238 0.91793571 0.91297769 0.92419351 0.9314321
 0.94293309 0.92363359 0.92969801 0.90606147 0.90007858 0.88780044
```

```
 0.86773747 0.8708087  0.8445489  0.86014592 0.8585985  0.88809573
 0.90134779 0.91723321 0.98752169 0.95419646 0.95557429 0.92780096
 0.92914481 0.94402922 0.92385754 0.89441798 0.90285793 0.91020506
 0.91787125 0.88258451 0.86808702 0.86408594 0.84898098 0.88361618
 0.89561933 0.89967468 0.89291803 0.8963795  0.89181851 0.87829154
 0.84077186 0.84569938 0.888564   0.88184133 0.86447277 0.8378703
 0.85990838 0.88761726 0.89140789 0.90692686 0.90676397 0.93232806
 0.92151938 0.94389343 0.86387885 0.86580211]
2 day output [[0.86745065]]
3 day input [0.98933731 0.99330441 0.99451935 0.99375579 0.97322445 0.97236246
 0.99212345 0.99058953 0.98668008 0.98784067 1.         0.99381006
 0.97599701 0.9724676  0.97476169 0.94804041 0.97076069 0.94610267
 0.94036071 0.95335485 0.94680857 0.95463765 0.98256359 0.9887875
 0.98506812 0.98753188 0.97276968 0.95974157 0.97694044 0.96339309
 0.94493189 0.94413096 0.95679254 0.97676397 0.98016777 0.98688705
 0.97861007 0.97839291 0.97530809 0.96615554 0.96710237 0.96308431
 0.91812238 0.91793571 0.91297769 0.92419351 0.9314321  0.94293309
 0.92363359 0.92969801 0.90606147 0.90007858 0.88780044 0.86773747
 0.8708087  0.8445489  0.86014592 0.8585985  0.88809573 0.90134779
 0.91723321 0.98752169 0.95419646 0.95557429 0.92780096 0.92914481
 0.94402922 0.92385754 0.89441798 0.90285793 0.91020506 0.91787125
 0.88258451 0.86808702 0.86408594 0.84898098 0.88361618 0.89561933
 0.89967468 0.89291803 0.8963795  0.89181851 0.87829154 0.84077186
 0.84569938 0.888564   0.88184133 0.86447277 0.8378703  0.85990838
 0.88761726 0.89140789 0.90692686 0.90676397 0.93232806 0.92151938
 0.94389343 0.86387885 0.86580211 0.86745065]
3 day output [[0.86864936]]
4 day input [0.99330441 0.99451935 0.99375579 0.97322445 0.97236246 0.99212345
 0.99058953 0.98668008 0.98784067 1.         0.99381006 0.97599701
 0.9724676  0.97476169 0.94804041 0.97076069 0.94610267 0.94036071
 0.95335485 0.94680857 0.95463765 0.98256359 0.9887875  0.98506812
 0.98753188 0.97276968 0.95974157 0.97694044 0.96339309 0.94493189
 0.94413096 0.95679254 0.97676397 0.98016777 0.98688705 0.97861007
 0.97839291 0.97530809 0.96615554 0.96710237 0.96308431 0.91812238
 0.91793571 0.91297769 0.92419351 0.9314321  0.94293309 0.92363359
 0.92969801 0.90606147 0.90007858 0.88780044 0.86773747 0.8708087
 0.8445489  0.86014592 0.8585985  0.88809573 0.90134779 0.91723321
 0.98752169 0.95419646 0.95557429 0.92780096 0.92914481 0.94402922
 0.92385754 0.89441798 0.90285793 0.91020506 0.91787125 0.88258451
 0.86808702 0.86408594 0.84898098 0.88361618 0.89561933 0.89967468
 0.89291803 0.8963795  0.89181851 0.87829154 0.84077186 0.84569938
 0.888564   0.88184133 0.86447277 0.8378703  0.85990838 0.88761726
 0.89140789 0.90692686 0.90676397 0.93232806 0.92151938 0.94389343
 0.86387885 0.86580211 0.86745065 0.86864936]
4 day output [[0.8693471]]
5 day input [0.99451935 0.99375579 0.97322445 0.97236246 0.99212345 0.99058953
 0.98668008 0.98784067 1.         0.99381006 0.97599701 0.9724676
 0.97476169 0.94804041 0.97076069 0.94610267 0.94036071 0.95335485
 0.94680857 0.95463765 0.98256359 0.9887875  0.98506812 0.98753188
 0.97276968 0.95974157 0.97694044 0.96339309 0.94493189 0.94413096
 0.95679254 0.97676397 0.98016777 0.98688705 0.97861007 0.97839291
 0.97530809 0.96615554 0.96710237 0.96308431 0.91812238 0.91793571
 0.91297769 0.92419351 0.9314321  0.94293309 0.92363359 0.92969801
 0.90606147 0.90007858 0.88780044 0.86773747 0.8708087  0.8445489
 0.86014592 0.8585985  0.88809573 0.90134779 0.91723321 0.98752169
 0.95419646 0.95557429 0.92780096 0.92914481 0.94402922 0.92385754
 0.89441798 0.90285793 0.91020506 0.91787125 0.88258451 0.86808702
 0.86408594 0.84898098 0.88361618 0.89561933 0.89967468 0.89291803
 0.8963795  0.89181851 0.87829154 0.84077186 0.84569938 0.888564
 0.88184133 0.86447277 0.8378703  0.85990838 0.88761726 0.89140789
 0.90692686 0.90676397 0.93232806 0.92151938 0.94389343 0.86387885
 0.86580211 0.86745065 0.86864936 0.8693471 ]
5 day output [[0.8695663]]
6 day input [0.99375579 0.97322445 0.97236246 0.99212345 0.99058953 0.98668008
 0.98784067 1.         0.99381006 0.97599701 0.9724676  0.97476169
 0.94804041 0.97076069 0.94610267 0.94036071 0.95335485 0.94680857
```

```
 0.91793571 0.91297769 0.92419351 0.9314321  0.94293309 0.92363359
 0.92969801 0.90606147 0.90007858 0.88780044 0.86773747 0.8708087
 0.8445489  0.86014592 0.8585985  0.88809573 0.90134779 0.91723321
 0.98752169 0.95419646 0.95557429 0.92780096 0.92914481 0.94402922
 0.92385754 0.89441798 0.90285793 0.91020506 0.91787125 0.88258451
 0.86808702 0.86408594 0.84898098 0.88361618 0.89561933 0.89967468
 0.89291803 0.8963795  0.89181851 0.87829154 0.84077186 0.84569938
 0.888564   0.88184133 0.86447277 0.8378703  0.85990838 0.88761726
 0.89140789 0.90692686 0.90676397 0.93232806 0.92151938 0.94389343
 0.86387885 0.86580211 0.86745065 0.86864936 0.8693471  0.86956632
 0.86936867 0.86883026 0.868029   0.86703473 0.86590695 0.86469305
 0.86342931 0.86214215 0.86084974 0.85956466 0.85829407 0.85704267
 0.85581201 0.85460311 0.85341513 0.85224783 0.85109985 0.8499704
 0.84885848 0.84776336 0.8466841  0.84562033]
28 day output [[0.84457153]]
29 day input [0.97276968 0.95974157 0.97694044 0.96339309 0.94493189 0.94413096
 0.95679254 0.97676397 0.98016777 0.98688705 0.97861007 0.97839291
 0.97530809 0.96615554 0.96710237 0.96308431 0.91812238 0.91793571
 0.91297769 0.92419351 0.9314321  0.94293309 0.92363359 0.92969801
 0.90606147 0.90007858 0.88780044 0.86773747 0.8708087  0.8445489
 0.86014592 0.8585985  0.88809573 0.90134779 0.91723321 0.98752169
 0.95419646 0.95557429 0.92780096 0.92914481 0.94402922 0.92385754
 0.89441798 0.90285793 0.91020506 0.91787125 0.88258451 0.86808702
 0.86408594 0.84898098 0.88361618 0.89561933 0.89967468 0.89291803
 0.8963795  0.89181851 0.87829154 0.84077186 0.84569938 0.888564
 0.88184133 0.86447277 0.8378703  0.85990838 0.88761726 0.89140789
 0.90692686 0.90676397 0.93232806 0.92151938 0.94389343 0.86387885
 0.86580211 0.86745065 0.86864936 0.8693471  0.86956632 0.86936867
 0.86883026 0.868029   0.86703473 0.86590695 0.86469305 0.86342931
 0.86214215 0.86084974 0.85956466 0.85829407 0.85704267 0.85581201
 0.85460311 0.85341513 0.85224783 0.85109985 0.8499704  0.84885848
 0.84776336 0.8466841  0.84562033 0.84457153]
29 day output [[0.84353715]]
[[0.8638788461685181], [0.8658021092414856], [0.8674506545066833], [0.8686493635177612],
[0.869347095489502], [0.8695663213729858], [0.8693686723709106], [0.8688302636146545], [0.
868028998374939], [0.8670347332954407], [0.8659069538116455], [0.8646930456161499], [0.863
4293079376221], [0.862142145636975], [0.8608497381210327], [0.8595646619796753], [0.85829
40697669983], [0.857042670249939], [0.8558120131492615], [0.854603111743927], [0.853415131
5689087], [0.8522478342056274], [0.8510998487472534], [0.8499704003334045], [0.84885847568
51196], [0.8477633595466614], [0.8466840982437134], [0.845620334148407], [0.84457153081893
92], [0.8435371518135071]]
```

In [38]:

```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

In [39]:

```
import matplotlib.pyplot as plt
```

In [40]:

```
len(data1)
```

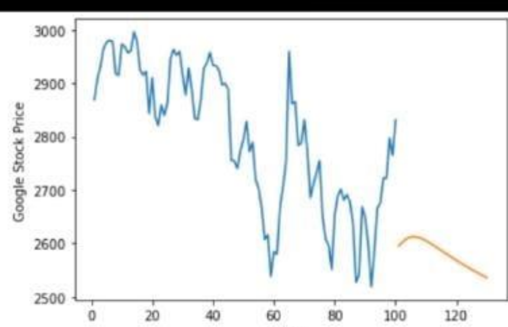
Out[40]: 4431

In [41]:

```
plt.plot(day_new,scaler.inverse_transform(data1[4331:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
```

Out[41]: Text(0, 0.5, 'Google Stock Price')

[redacted]



15/16