*Vol-1 Issue-2 2023*
*Scientific Research Journal of Science, Engineering and Technology*
ISSN: 2584-0584, Peer Reviewed Journal

# MATHEMATICAL MODEL FOR STOCK PRICE PREDICTION USING LSTM NETWORKS IN PYTHON JUPYTER NOTEBOOK

**Dr Vivek Parkash**

*Assistant Prof. of Mathematics,*

*Dyal Singh College, Karnal (Haryana), India*

**ABSTRACT**

*Long short-term memory, called LSTM for short, is a kind of neural network technology with applications in deep learning and artificial intelligence. By combining python code in a jupyter notebook with LSTM networks, which stands for "long short-term memory," I hope to accurately predict future price movements for TCS stocks that are listed on the NSE. It will be decided whether the expected change in the price of TCS stock was similar to the actual change. Prices from the previous several days of trade will also be used to set opening prices for the following 20 trading days.*

**Keywords:** *yahoo finance, long short-term memory networks, Keras, pandas, data frame, transfer learning, neural network*

---

## Introduction

An artificially evolved neural network utilized in AI and supervised learning [6] is called long short-term memory. A supervised learning, sequencing neural net which can remember information between training rounds is called a Long Short-Term Memory Structure. It is a subset of RNNs that can overcome the receding gradient issue often encountered by RNNs. The LSTM algorithm, created by Hochreiter and Schmidhuber, is an improvement over prior deep learning and recurrent neural network methods ([7]). The Keras package allows for LSTM to be implemented in Python. (Shipra Saxena,2021).

Let's look at long-term, short-term memory network (LSTM) operations. Algorithms are utilized during the process of machine learning. They have memory cells that store the assumptions of the earlier section using intrinsic parameters, and these prognostications are used as input to anticipate the values of the following sequence. Additionally, they have synaptic connections that are used to store the predictions of the importance of the previous step. This is similar to using assumptions in a recurrence to get the following set of forecasts. To get things started, we are now loading the data on the price movement of TCS stock. Yahoo Finance is what we're going to be utilizing for this purpose. Yahoo Finance is a repository of stock price data. The following libraries and codes are used as input in the jupyter notebook ([1],[4]).
After input, jupyter notebook yields the output:

```
jupyter LSTM PYTHON Last Checkpoint: 21 hours ago (unsaved changes)          Logout
File  Edit  View  Insert  Cell  Kernel  Widgets  Help          Trusted | Python 3 (ipykernel)

In [141]: import yfinance as yf
          import matplotlib.pyplot as plt
          import numpy as np

In [31]:
          stock_symbol = 'TCS.NS'
```

We have imported TCS stock data for three years daily, from 2020 till March 2023. After using the pandas dataframe, we get [6],

```
In [31]:
          stock_symbol = 'TCS.NS'

In [88]:  #last 3 years data with interval of 1 day
          data = yf.download(tickers=stock_symbol,period='3y',interval='1d')

          [*********************100%***********************]  1 of 1 completed

In [89]:  type(data)

Out[89]:  pandas.core.frame.DataFrame

In [90]:  data.head()
```

```
In [90]:  data.head()

Out[90]:
```

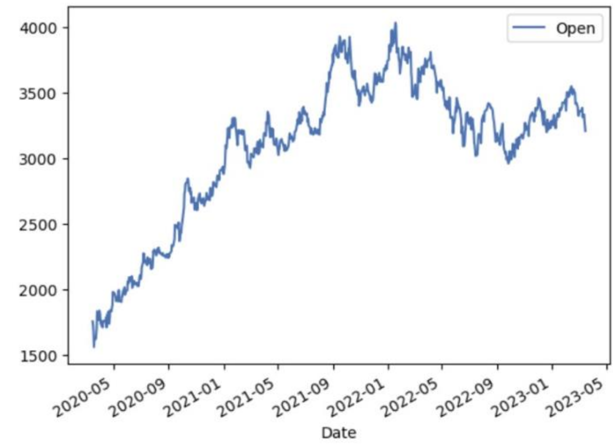
| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2020-03-16 | 1755.000000 | 1842.250000 | 1675.849976 | 1696.400024 | 1595.482544 | 7844271 |
| 2020-03-17 | 1730.000000 | 1731.000000 | 1623.150024 | 1658.000000 | 1559.366699 | 5713248 |
| 2020-03-18 | 1676.800049 | 1713.550049 | 1627.750000 | 1654.400024 | 1555.980957 | 7258778 |
| 2020-03-19 | 1559.699951 | 1685.449951 | 1546.750000 | 1636.349976 | 1550.249634 | 5135111 |
| 2020-03-20 | 1630.000000 | 1869.000000 | 1627.000000 | 1797.449951 | 1702.872559 | 8547498 |

```
In [91]:  type(data)

Out[91]:  pandas.core.frame.DataFrame

In [36]:  len(data)

Out[36]:  1237
```

The open, high, and low values of TCS are shown here, along with the data for the close, adj. close, and volume.

Now, for the data prediction, I have considered the available prices of TCS so that after the prediction model, we can fetch the forecast of general costs for the next 20 days.

```
In [143]: opn = data[['Open']]

In [144]: opn.plot()

Out[144]: <AxesSubplot:xlabel='Date'>
```

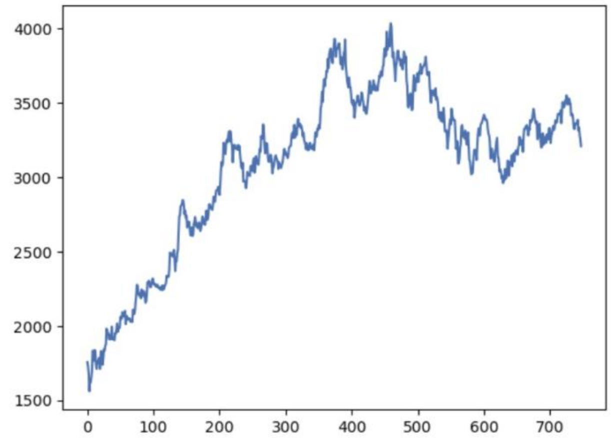



I convert this data to a NumPy array using the dot values function.

```
In [97]: ds = opn.values

In [98]: ds

Out[98]: array([[1755.        ],
                [1730.        ],
                [1676.80004883],
                [1559.69995117],
                [1630.        ],
                [1620.        ],
                [1653.05004883],
                [1700.        ],
                [1831.59997559],
                [1820.        ],
                [1766.        ],
                [1837.40002441],
                [1825.90002441],
                [1740.        ],
                [1710.        ],
                [1760.        ],
                [1750.44995117],
                [1761.        ],
                [1785.        ],
```

To plot it, I import matplotlib.

```
In [99]: plt.plot(ds)
Out[99]: [<matplotlib.lines.Line2D at 0x26276bf3250>]
```




We have used the min-max scaler function to assign values between 0 and 1. Let us understand with the help of an example:
Suppose we have two values, x and y, where x is 0 to 50 and y is 100 to 500. Here LSTN will be more inclined towards the greater value y. So, to normalize this feature, we use the normalization function. This function will assign a value between 0 and 1 corresponding to any matter we give [2].

We have taken 80 % of the data as test data and 20% as train data.

```
In [204]: from sklearn.preprocessing import MinMaxScaler

In [205]: normalizer = MinMaxScaler(feature_range=(0,1))
          ds_scaled = normalizer.fit_transform(np.array(ds).reshape(-1,1))

In [101]: len(ds_scaled), len(ds)
Out[101]: (1237, 748)

In [206]: train_size = int(len(ds_scaled)*0.80)
          test_size = len(ds_scaled) - train_size

In [207]: train_size,test_size
Out[207]: (598, 150)

In [208]: #Splitting data between train and test
          ds_train, ds_test = ds_scaled[0:train_size,:], ds_scaled[train_size:len(ds_scaled),:1]

In [209]: len(ds_train),len(ds_test)
Out[209]: (598, 150)
```

For creating a time series dataset for LSTM mode, we take 120 days' price as a single data record for training.

```
In [234]: #X[120,140,160,180,200] : Y[220]
          def create_ds(dataset,step):
              Xtrain, Ytrain = [], []
              for i in range(len(dataset)-step-1):
                  a = dataset[i:(i+step), 0]
                  Xtrain.append(a)
                  Ytrain.append(dataset[i + step, 0])
              return np.array(Xtrain), np.array(Ytrain)

In [211]:
          time_stamp = 120
          X_train, y_train = create_ds(ds_train,time_stamp)
          X_test, y_test = create_ds(ds_test,time_stamp)

In [212]: X_train.shape,y_train.shape

Out[212]: ((477, 120), (477,))

In [213]: X_test.shape, y_test.shape

Out[213]: ((29, 120), (29,))
```

Now we reshape the data to fit into our LSTM model, and for Creating the LSTM model using Keras, we input the code and get the output:

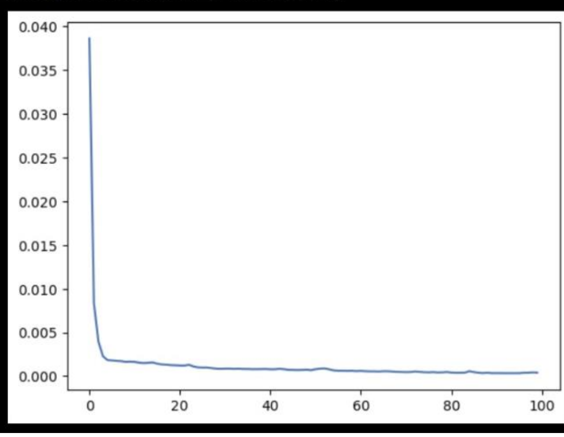
```
In [214]:
          X_train = X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
          X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

In [215]: from keras.models import Sequential
          from keras.layers import Dense, LSTM

In [150]:
          model = Sequential()
          model.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1],1)))
          model.add(LSTM(units=50,return_sequences=True))
          model.add(LSTM(units=50))
          model.add(Dense(units=1,activation='linear'))
          model.summary()

          Model: "sequential_2"
          _________________________________________________________________
           Layer (type)                Output Shape              Param #
          =================================================================
           lstm_6 (LSTM)               (None, 120, 50)           10400

           lstm_7 (LSTM)               (None, 120, 50)           20200

           lstm_8 (LSTM)               (None, 50)                20200

           dense_2 (Dense)             (None, 1)                 51

          =================================================================
          Total params: 50,851
          Trainable params: 50,851
          Non-trainable params: 0
```

We are implementing the model with The Adam methodology is a kind of spontaneous gradient descent that is predicated on the adaptive estimate and mean squared error loss function as below:

```
In [216]: model.compile(loss='mean_squared_error',optimizer='adam')
          model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=64)

          Epoch 1/100
          8/8 [==============================] - 8s 285ms/step - loss: 0.0075 - val_loss: 0.0079
          Epoch 2/100
          8/8 [==============================] - 1s 126ms/step - loss: 0.0036 - val_loss: 6.0041e-04
          Epoch 3/100
          8/8 [==============================] - 1s 121ms/step - loss: 0.0013 - val_loss: 0.0012
          Epoch 4/100
          8/8 [==============================] - 1s 122ms/step - loss: 0.0011 - val_loss: 4.4437e-04
          Epoch 5/100
          8/8 [==============================] - 1s 118ms/step - loss: 6.6024e-04 - val_loss: 3.1845e-04
          Epoch 6/100
          8/8 [==============================] - 1s 120ms/step - loss: 6.0370e-04 - val_loss: 3.8974e-04
          Epoch 7/100
          8/8 [==============================] - 1s 121ms/step - loss: 5.1632e-04 - val_loss: 2.6726e-04
          Epoch 8/100
          8/8 [==============================] - 1s 121ms/step - loss: 5.1639e-04 - val_loss: 2.7871e-04
          Epoch 9/100
```

The below plot shows that loss has decreased quite a lot, and the model has been trained well.

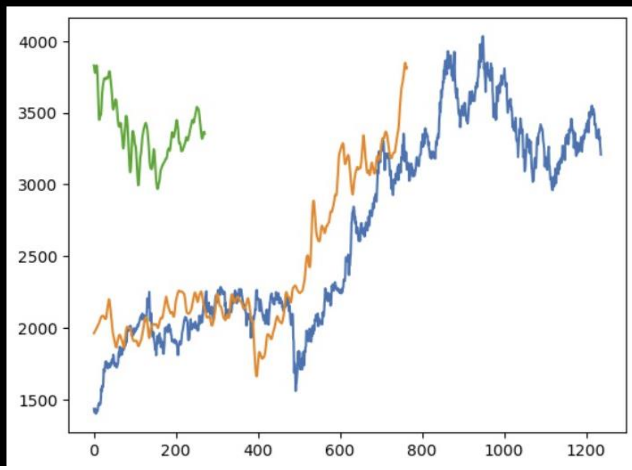
```
In [152]: loss = model.history.history['loss']
          plt.plot(loss)

Out[152]: [<matplotlib.lines.Line2D at 0x2620b9c4850>]
```




```
In [217]: train_predict = model.predict(X_train)
          test_predict = model.predict(X_test)

          15/15 [==============================] - 2s 32ms/step
          1/1 [==============================] - 0s 45ms/step

In [218]: train_predict = normalizer.inverse_transform(train_predict)
          test_predict = normalizer.inverse_transform(test_predict)

In [66]:  #Comparing using visuals
          plt.plot(normalizer.inverse_transform(ds_scaled))
          plt.plot(train_predict)
          plt.plot(test_predict)
```

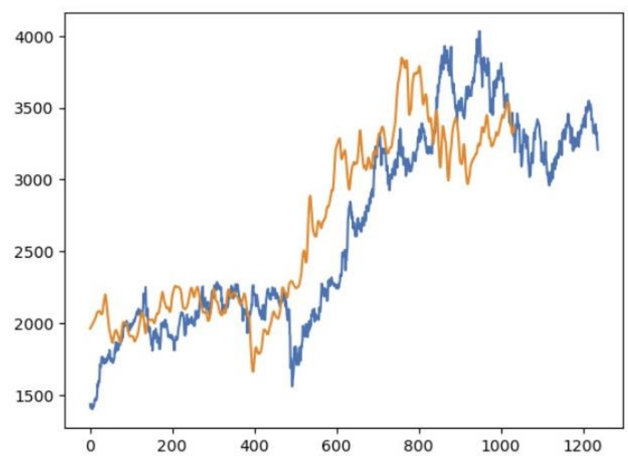
The below plot shows that the blue curve is the actual data graph, an orange angle is the predicted graph of train data, and a green curve is the expected curve of test data of TCS stock:

```
In [66]: plt.plot(normalizer.inverse_transform(ds_scaled))
         plt.plot(train_predict)
         plt.plot(test_predict)

Out[66]: [<matplotlib.lines.Line2D at 0x262759e2a00>]
```




After combining the predicted data to create uniform data visualization (graph without discontinuity)

```
In [219]: type(train_predict)

Out[219]: numpy.ndarray

In [220]: test = np.vstack((train_predict,test_predict))

In [69]: plt.plot(normalizer.inverse_transform(ds_scaled))
         plt.plot(test)

Out[69]: [<matplotlib.lines.Line2D at 0x2627598a040>]
```



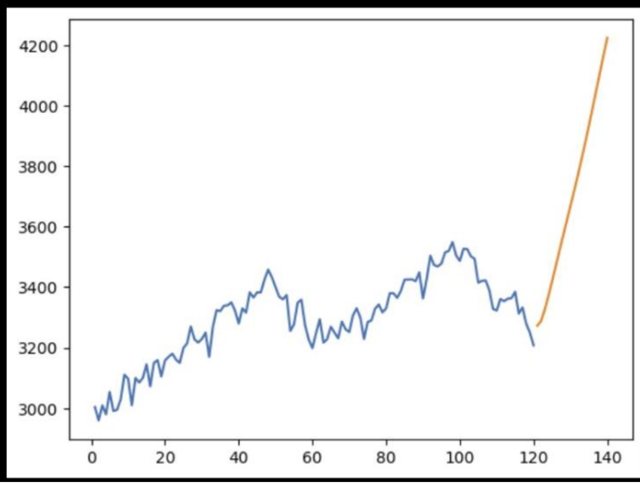

Now fetching the last 120 days' records, creating a list of the previous 120 data, and predicting the next 20 days' prices using the current data we have:

```
In [221]: len(ds_test)
Out[221]: 150

In [222]: fut_inp = ds_test[250:]

In [223]: fut_inp = fut_inp.reshape(1,-1)

In [224]: tmp_inp = list(fut_inp)

In [226]: fut_inp.shape
Out[226]: (1, 0)

In [227]: tmp_inp = tmp_inp[0].tolist()
```

```
In [167]: lst_output=[]
          n_steps=102
          i=0
          while(i<20):

              if(len(tmp_inp)>102):
                  fut_inp = np.array(tmp_inp[1:])
                  fut_inp=fut_inp.reshape(1,-1)
                  fut_inp = fut_inp.reshape((1, n_steps,1))
                  yhat = model.predict(fut_inp, verbose=0)
                  tmp_inp.extend(yhat[0].tolist())
                  tmp_inp = tmp_inp[1:]
                  lst_output.extend(yhat.tolist())
                  i=i+1
              else:
                  fut_inp = fut_inp.reshape((1, n_steps,1))
                  yhat = model.predict(fut_inp, verbose=0)
                  tmp_inp.extend(yhat[0].tolist())
                  lst_output.extend(yhat.tolist())
                  i=i+1


          print(lst_output)

          [[0.7108184099197388], [0.7165108919143677], [0.7305189371109009], [0.7476314902305603], [0.7661316990852356], [0.7852297425270
          081], [0.8044221997261047], [0.8234397172927856], [0.8422471284866333], [0.860995888710022], [0.8799392580986023], [0.899342298
          5076904], [0.9194039106369019], [0.9402101635932922], [0.9617148041725159], [0.9837561845779419], [1.00609290599823], [1.028449
          296951294], [1.0505621433258057], [1.0722135305404663]]
In [229]: len(ds_scaled)
Out[229]: 748

In [230]: plot_new=np.arange(1,121)
          plot_pred=np.arange(121,141)
```

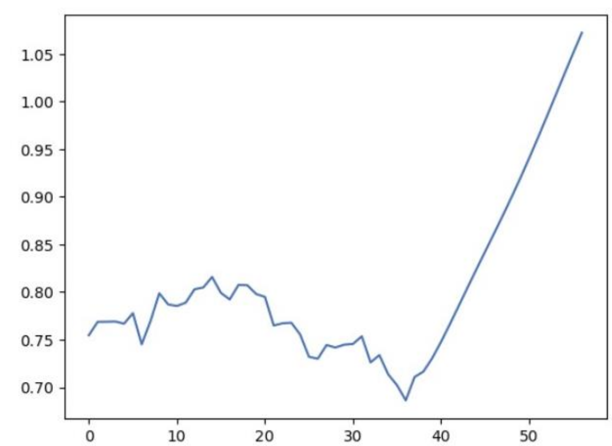
The orange curve below shows the predicted curve:

```
In [197]: plt.plot(plot_new, normalizer.inverse_transform(ds_scaled[1117:]))
          plt.plot(plot_pred, normalizer.inverse_transform(lst_output))

Out[197]: [<matplotlib.lines.Line2D at 0x2620fbdd490>]
```



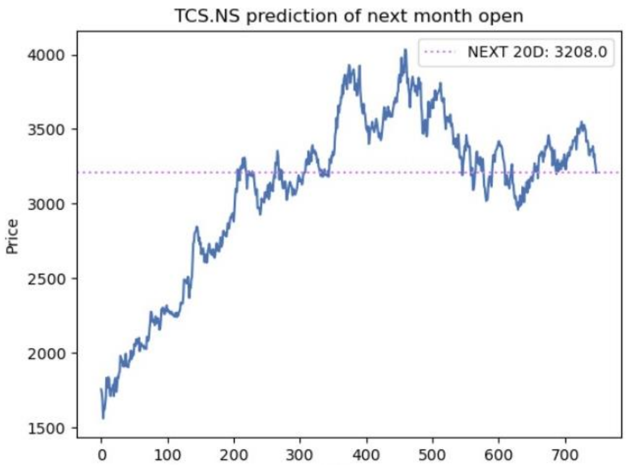

There is some gap between the blue curve and the beginning of the orange turn. After making it continuous,

```
In [231]: ds_new = ds_scaled.tolist()

In [232]: len(ds_new)

Out[232]: 748

In [200]: ds_new.extend(lst_output)
          plt.plot(ds_new[1200:])

Out[200]: [<matplotlib.lines.Line2D at 0x2620fc41730>]
```



```
In [233]: #Creating final data for plotting
          final_graph = normalizer.inverse_transform(ds_new).tolist()
```

```
In [235]: final results with predicted value after 20 Days
          inal_graph,)
          l("Price")
          l("Time")
          ("{0} prediction of next month open".format(stock_symbol))
          ne(y=final_graph[len(final_graph)-1], color = 'violet', linestyle = ':', label = 'NEXT 20D: {0}'.format(round(float(*final_graph[l
          d()

Out[235]: &lt;matplotlib.legend.Legend at 0x2620fc17a90&gt;
```




## Conclusion

So, our model has successfully predicted stock TCS move for the next 20 days. This graph shows how well the share has moved during the prediction period. This above-described model is for TCS stock. This model can be applied to any other stock. All we must do is to import the corresponding stock data from yahoo finance. We can change different parameters accordingly and tweak the parameters to get better results.

**Disclaimer: Always consult your financial advisor before applying this model in a live market.**

## References

[1] https://www.analyticsvidhya.com/blog/2021/12/stock-price-prediction-using-lstm/
[2] https://towardsdatascience.com/lstm-for-google-stock-price-prediction-e35f5cc84165
[3] https://www.datacamp.com/tutorial/lstm-python-stock-market
[4] https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm
[5]HongjuYan and Hongbing Ouyang. Financial time series prediction based on deep learning. Wireless Personal Communications, 102(2):683–700, 2018.
[6] https://en.wikipedia.org/wiki/Long_short-term_memory
[7]https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/