# Predicting the Sales Volume of the Google Stock Price Dataset

## Data Preprocessing

## Dataset Importing

I used Google colabs as the IDE for designing the model and analysis. In google colabs we can import data in 2 ways as importing dataset directly to colabs environment or mounting google drive to colabs and uploading the dataset to google drive. I used to 2nd method as i don't need to upload dataset again and again when session is over.

I used pandas **read_csv** method to import data by passing several arguments as follows,

1. giving location of dataset
2. make date column as index column of data frame.
3. Sep as '**,**' as csv store data by using comma separator.
4. thousand as '**,**' as volume data is present in object and used thousand separators.
5. parse_dates as True to parse dates in date column for data splitting purpose.

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

```
import pandas as pd
dataset = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/data/Google_Stock_Price.csv', header=0, index_col=0,  sep=',' , thousands=',',parse_dates = True)
print(dataset.head(10))
```

```
              Open    High     Low   Close    Volume
Date
2012-01-03  325.25  332.83  324.97  663.59   7380500
2012-01-04  331.27  333.87  329.08  666.45   5749400
2012-01-05  329.83  330.75  326.89  657.21   6590300
2012-01-06  328.34  328.77  323.68  648.24   5405900
2012-01-09  322.04  322.29  309.46  620.76  11688800
2012-01-10  313.70  315.72  307.30  621.43   8824000
2012-01-11  310.59  313.52  309.40  624.25   4817800
2012-01-12  314.43  315.26  312.08  627.92   3764400
2012-01-13  311.96  312.30  309.37  623.28   4631800
2012-01-17  314.81  314.81  311.67  626.86   3832800
```

*Figure 1 Snapshot of Dataset import and Data frame after import.*

## Handling/Checking for Missing Values

I check whether any null values are present on dataset using info () method. I got 1258 Datetime Index entries from 2012-01-03 to 2016-12-30 and 5 columns as shown in Figure 2 and there was not any null value present.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1258 entries, 2012-01-03 to 2016-12-30
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    1258 non-null   float64
 1   High    1258 non-null   float64
 2   Low     1258 non-null   float64
 3   Close   1258 non-null   float64
 4   Volume  1258 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 59.0 KB
```

*Figure 2 Snapshot of info () method on dataset.*

I used isnull() method to confirm the result that come from info() method so I could verify there wasn't any null or missing values in dataset as shown in Figure 3.

```
missing_values = dataset.isnull().mean()*100
missing_values.sum()
```

```
0.0
```

*Figure 3 missing value checking using isnull() method.*

To check whether any duplicate value present in dataset I use duplicated().any() method and none of duplicated values were found as per Figure 4.

```
isDuplicate = dataset.duplicated().any()
if isDuplicate :
  print("Is there any duplicate value present in Google_Stock_Price dataset : YES")
else :
 print("Is there any duplicate value present in Google_Stock_Price dataset : NO")
```

```
Is there any duplicate value present in Google_Stock_Price dataset : NO
```

*Figure 4 checking for duplicates in dataset.*

## Handling /Checking Outliers

I use z score approach to check whether any outlier data present in dataset and use score approach to remove outliers. In z score approach data point with z score that greater than 3 or less than -3 consider as outliers. I use abs and stats library provide by scipy to convert data to z score as shown in Figure 5.

```
from scipy import stats

z = np.abs(stats.zscore(dataset))
print(z)
```

```
[[1.37285476 1.34065332 1.35579373 0.29801768 1.85777584]
 [1.33320882 1.33385362 1.32848349 0.28065141 1.14012166]
 [1.34269224 1.35425271 1.34303566 0.33675781 1.51010229]
 ...
 [1.71221829 1.67605855 1.68906636 0.43950216 0.88185833]
 [1.64392454 1.62179174 1.66062649 0.42577916 1.06203085]
 [1.64010483 1.6011965  1.60407899 0.35916799 0.6107416 ]]
```

*Figure 5 converting data points to z score.*

I check for any data points that present in z score with higher than 3 and found there are 33 data points that have z score that are greater than 3. And no data point with z score less than -3 as show in Figure 6.

```
print(np.where(z > 3))
```

```
(array([  4,  11,  12,  15,  69,  70,  71,  95, 138, 182, 184, 185, 201,
       202, 264, 324, 387, 451, 521, 522, 533, 537, 538, 539, 540, 541,
       543, 544, 545, 546, 547, 553, 889]), array([4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 4]))
```

```
print(np.where(z < -3))
```

```
(array([], dtype=int64), array([], dtype=int64))
```

*Figure 6 Detecting outliers.*

Then I removed data points with z score that grater than 3 as per Figure 7 and Figure 8 shows boxplot analysis of data before and after removing outliers in dataset. After removing outliers there was 1225 data points.

```
dataset_new = dataset[(z < 3).all(axis=1)]
dataset_new.shape
```

```
(1225, 5)
```

*Figure 7 removing outliers.*

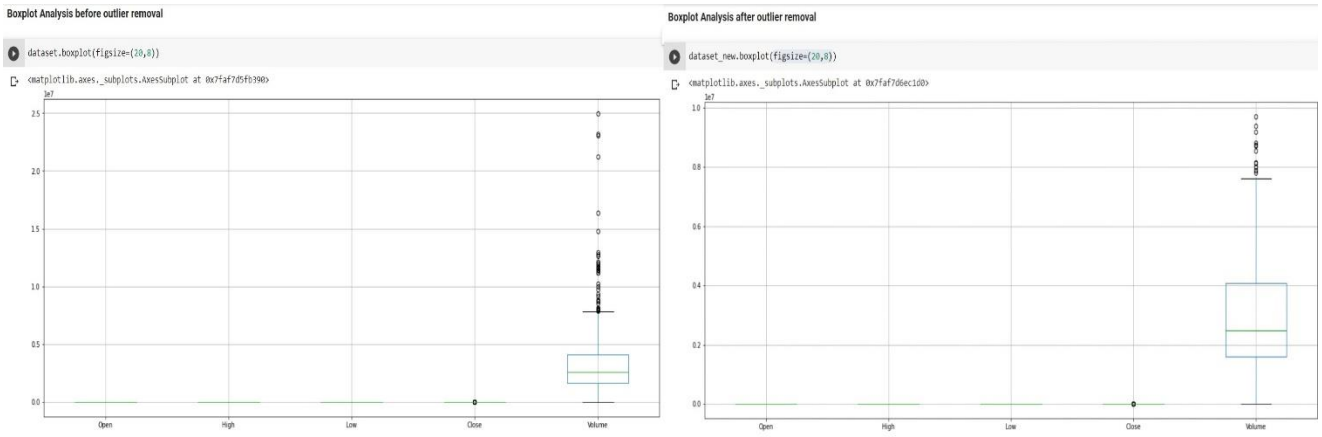



*Figure 8 boxplot analysis of data before and after removing outliers.*

## Principal Components Analysis

Principle component analysis is used to find which components that can explain almost all the variance in data set. As per figure 9 we can say with use open and High we can explain variation but as we need to predict volume, we must consider what are the significant features and which kind of model should use for that purpose.

```
pca = PCA(n_components = 5)
pca.fit(dataset_new)
ratio_sum = pca.explained_variance_ratio_.cumsum()
ratio_sum
```

```
array([0.99999998, 0.99999999, 1.        , 1.        , 1.        ])
```

*Figure 9 PCA Analysis.*

## Feature Analysis

I have done correlation analysis and time series plot analysis to identify the significant and independent features.

### *Correlation Analysis*

I used seaborn sns libraries to plot a heatmap to visualize the correlation between features. All high , open and low has positive correlation of 1 and all these feature have -.063 correlation with volume and close has nearly 0.2 correlation with all others as shown in figure 10.

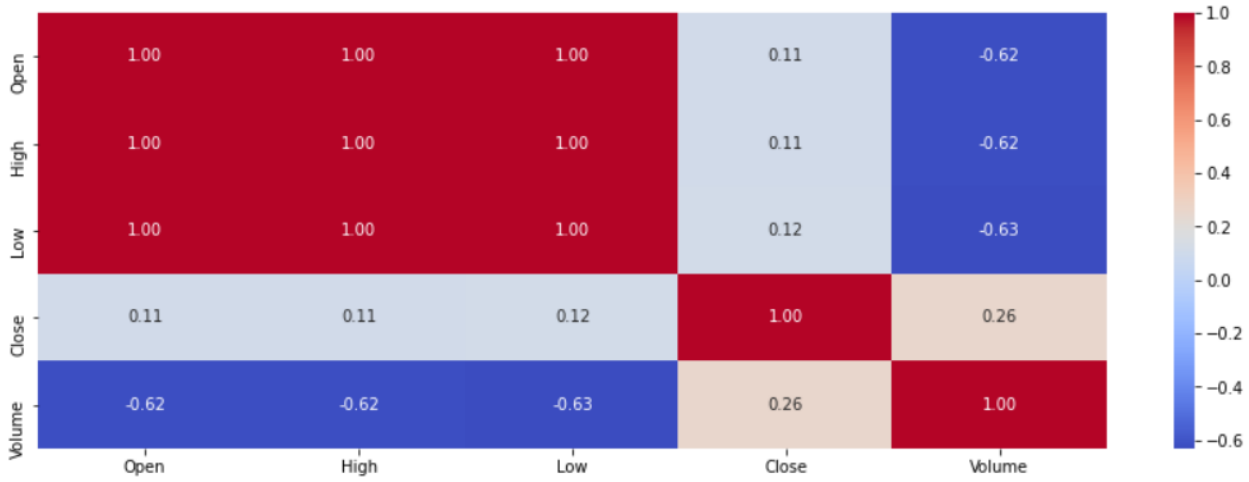



*Figure 10 Correlation between features.*

## *Time Series Plot Analysis*

I plot timeseries plot for each feature and see any anomaly or a stationarity in data. So as per figure 11 we can see all high, low, open variation are simar with time but close have high value that expected high value according to stock value variations, so reliability of Close stock feature is quite problematic.

```
dataset_new[['High','Open','Low','Close']].plot.line()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faf735e7090>
```

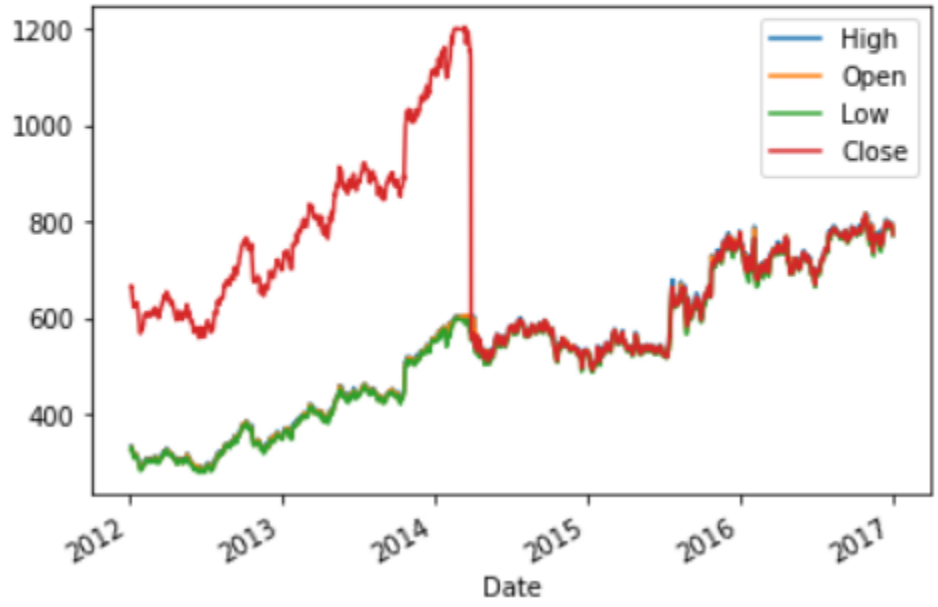



*Figure 11 variations of features High, Low, Open and Close.*

```
dataset_new[['Volume']].plot.line()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8cb0af6d0>
```

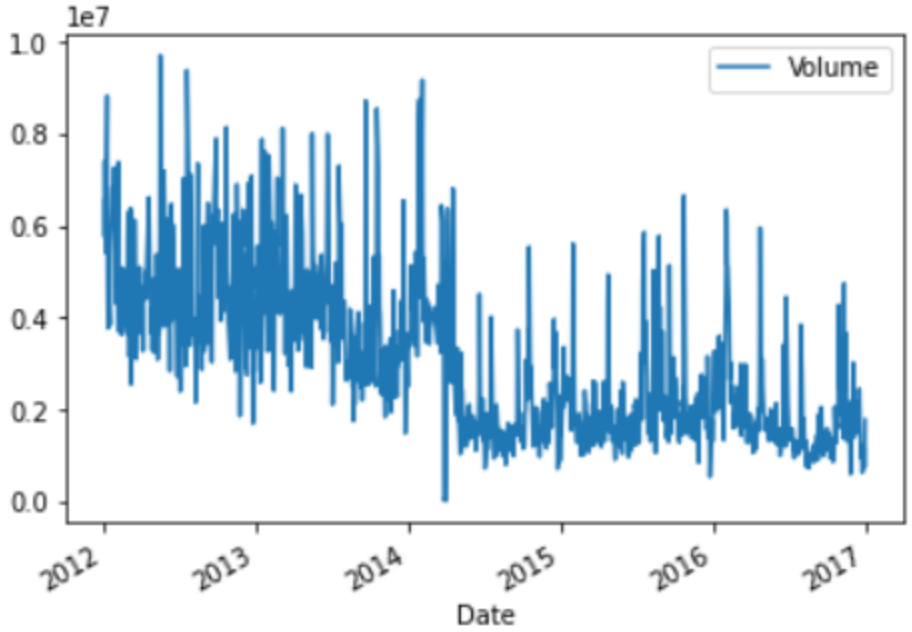



*Figure 12 Variation of volume with time.*

## Model Selection

As we need to Predict the Sales Volume and when compared to other features volume has less correlation with other data and volume is a timeseries data so we can use volume as a Sequence prediction by using supervised learning approach to Predict the Sales Volume so for that we use LSTM Model to design our Neural network model.so I use Univariate Time Series Forecasting concept to design the LSTM model to predict the task.

## Preparing Dataset for LSTM Model

I sorted the dataset with date index to make sure all the data is aligning with time (figure 13).

```
series = series.sort_index()
```

```
print(series)
```

```
              Open    High     Low   Close     Volume
Date
2012-01-03  325.25  332.83  324.97  663.59  7380500.0
2012-01-04  331.27  333.87  329.08  666.45  5749400.0
2012-01-05  329.83  330.75  326.89  657.21  6590300.0
2012-01-06  328.34  328.77  323.68  648.24  5405900.0
2012-01-10  313.70  315.72  307.30  621.43  8824000.0
...            ...     ...     ...     ...        ...
2016-12-23  790.90  792.74  787.28  789.91   623400.0
2016-12-27  790.68  797.86  787.66  791.55   789100.0
2016-12-28  793.70  794.23  783.20  785.05  1153800.0
2016-12-29  783.33  785.93  778.92  782.79   744300.0
2016-12-30  782.75  782.78  770.41  771.82  1770000.0

[1225 rows x 5 columns]
```

*Figure 13 Sorting data according to time*

Main part of Data preparation is to splitting dataset to train, validation, and test sets so for that I split data for training from 2012/01/01 to 2015/12/31 which has 973 records which is nearly 80 % of data and split data for validation from 2016/01/01 to 2016/06/30 which has 125 records which is nearly 10 % of data the I split data for testing 2016/01/01 to 2016/12/31 which has 127 records which is nearly 10 % of data as per figure 14. I only extracted volume data for each train, validation and test and convert it to 2d np array.

```
import datetime as dt

train_start = dt.date(2012,1,1)
train_end = dt.date(2015,12,31)
train_data = series.loc[train_start:train_end].iloc[:, 4:5].values

validation_start = dt.date(2016,1,1)
validation_end = dt.date(2016,6,30)
validation_data = series.loc[validation_start:validation_end].iloc[:, 4:5].values

test_start = dt.date(2016,7,1)
test_end = dt.date(2016,12,31)
test_data = series.loc[test_start:test_end].iloc[:, 4:5].values

print(train_data.shape,validation_data.shape,test_data.shape)
```

```
(973, 1) (125, 1) (127, 1)
```

*Figure 14 Data splitting for training, validation, and testing.*

## Feature Scaling

As we could see there are big and different range of values in Volume data, so I decide to use normalization as feature scaling. I used Min Max Scaler that Sklearn for this task and range of scalar was ser to 0 to 1.1st I fit and transform the training dataset and then apply transformations to both validation and testing data sets. (Figure 15)

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))

training_set_scaled = sc.fit_transform(train_data)
validation_set_scaled = sc.transform(validation_data)
test_set_scaled = sc.transform(test_data)

print(training_set_scaled.shape,validation_set_scaled.shape,test_set_scaled.shape)
```

```
(973, 1) (125, 1) (127, 1)
```

*Figure 15 Feature Scaling.*

## Creating a Training and Validation data structures with (30 / 40 / 50) timesteps

### *Concept of Timesteps*

LSTM models are designed to analyze data sequence by considering a previous value of sequence or set of values in sequence. So, when we are designing a LSTM model architecture, we need prepare all training, validation, and testing data structures with a relevant timestep. Timestep simply means number of data points which are used to predict the variable of the next datapoint. In this model I have tasted model using 3 different timestep as 30, 40 and 50 to check whether which timestep provide high accuracy of prediction. (Figure 16)

**Preparing Traning and Validation Dataset**

```
[31] X_train = []
     y_train = []
     for i in range(30, 973):
         X_train.append(training_set_scaled[i-30:i, 0])
         y_train.append(training_set_scaled[i, 0])
     X_train, y_train = np.array(X_train), np.array(y_train)
```

```
     X_validation = []
     y_validation = []
     for i in range(30, 125):
         X_validation.append(validation_set_scaled[i-30:i, 0])
         y_validation.append(validation_set_scaled[i, 0])
     X_validation, y_validation = np.array(X_validation), np.array(y_validation)
```

```
[33] print(X_train.shape)
```

```
     (943, 30)
```

```
     print(X_validation.shape)
```

```
     (95, 30)
```

*Figure 16 preparing Training and validation test according to timesteps.*

As LSTM model need training set and validation set as 3D array, we need to reshape it. (Figure 17)

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

```
(943, 30, 1)
```

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

```
(95, 30, 1)
```

*Figure 17 Reshaping 2D array to 3D array*

## Building Model

1st I was designed a 4-layer LSTM Model with each 50 unit and dropout regularizations with .2 was designed and tested but it would not provide good result so I fine tune model with 96,65,55 and 34 units and same dropout regularizations. LSTM mainly have exploding and vanishing gradient issue, so I use tanh as activation fiction. Dropout regularization is added to avoid overfitting of the model. Danse layer is a fully connected to all the output from previous layer as I need one value as prediction in this neural network, I set it to 1 neuron. (Figure 18)

```
regressor.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_8 (LSTM)                (None, 30, 96)            37632
_________________________________________________________________
dropout_8 (Dropout)          (None, 30, 96)            0
_________________________________________________________________
lstm_9 (LSTM)                (None, 30, 65)            42120
_________________________________________________________________
dropout_9 (Dropout)          (None, 30, 65)            0
_________________________________________________________________
lstm_10 (LSTM)               (None, 30, 55)            26620
_________________________________________________________________
dropout_10 (Dropout)         (None, 30, 55)            0
_________________________________________________________________
lstm_11 (LSTM)               (None, 34)                12240
_________________________________________________________________
dropout_11 (Dropout)         (None, 34)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 35
=================================================================
Total params: 118,647
Trainable params: 118,647
Non-trainable params: 0
_________________________________________________________________
```

*Figure 18 Summary of Model with 30 timesteps*

## Training the Model

I pass both X_Train and Y_Train as training data set and X_Validation and Y_Validation as validating dataset and set epoch to 100 and batch size to 32. (Figure 19)

```
history = regressor.fit(X_train, y_train, epochs = 100, batch_size = 32,validation_data=(X_validation, y_validation))
```

```
Epoch 1/100
30/30 [==============================] - 12s 141ms/step - loss: 0.0542 - mae: 0.1773 - val_loss: 0.0085 - val_mae: 0.0760
Epoch 2/100
30/30 [==============================] - 2s 76ms/step - loss: 0.0189 - mae: 0.0995 - val_loss: 0.0087 - val_mae: 0.0709
Epoch 3/100
30/30 [==============================] - 2s 76ms/step - loss: 0.0171 - mae: 0.0952 - val_loss: 0.0108 - val_mae: 0.0836
```

*Figure 19 Training the Model.*

## Testing the Model

In order to test the model, I had to prepare test data according timesteps and reshape the X_Test array to make it 2D to 3D array. (Figure 20)

```
inputs = test_set_scaled
X_test = []
y_test = []
for i in range(30, 127):
    X_test.append(inputs[i-30:i, 0])
    y_test.append(inputs[i, 0])
X_test = np.array(X_test)
y_test = np.array(y_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
X_test.shape
```

```
(97, 30, 1)
```

*Figure 20 preparing Test Data.*

## Making the predictions

## Getting the predicted stock volume

To make prediction I use predict method in sequence model and pass the prepared X_Test array. The take the result prediction and through inverse_tranform method set the prediction real volume values. (Figure 21)

```
predicted_value = regressor.predict(X_test)
predicted_stock_volume = sc.inverse_transform(predicted_value)
```

*Figure 21 predicting stock volume.*

## Fixing 1-lag issue in prediction

When I was design and evaluate model with 50, 40 and 30 respective timesteps I got a R2 accuracy of 22% to 26% but when I visualize the Real stock test data vs predict data from model they was right shift on graph so when I searching the issue this cause I found that I used sequence prediction as supervise learning so what really happen was $0^{th}$ position value of real stock volume array is correspond to $1^{st}$ position value of predicted volume array so I had to shift the prediction array in order to evaluate and visualize properly.

Prediction array was a 2D array so I could not easily shift it using shift function provide by scipy.ndimage.interpolation library so I used ravel() function to flatten the array and shift the array to left by applying -1 and replacing the missing last value by the previous last value of array and the reshape it to 2D Array. (Figure 22)

```
from scipy.ndimage.interpolation import shift

predicted_stock_volume = predicted_stock_volume.ravel()
array_length = len(predicted_stock_volume)
last_element = predicted_stock_volume[array_length - 1]
predicted_stock_volume = shift(predicted_stock_volume, -1, cval=last_element)
predicted_stock_volume = np.reshape(predicted_stock_volume,(predicted_stock_volume.shape[0], 1))
```

*Figure 22 Fixing 1-lag issue.*

Preparing real stock volume value for evaluation and visualizing purpose is shown in Figure 23.

```
stock_volume = y_test.reshape(-1,1)
real_stock_volume = sc.inverse_transform(stock_volume)
```

*Figure 23 Preparing real stock volume.*

## Evaluating Model performance

I have evaluated the model for 3 types of time steps and for both on and before fixing 1 lag transformation.

### Evaluate the model for 100 epochs and 50 timesteps.

*Before Fixing 1-lag transform of the sequence*

```
from numpy import sqrt
mse, mae = regressor.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
```

```
MSE: 0.004, RMSE: 0.066, MAE: 0.048
```

```
from sklearn.metrics import r2_score

print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume))
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
R2 Score:  0.2266817793371212

R2 Score:  0.2266817528952294
```

*Figure 24 evaluation results of 50 timesteps before fixing 1-lag issue.*

We can observe that accuracy of model is 22.66 % according to $R^2$ matrix which is very low. All mse, rmse and mae calculate for scaled values through model.

*After Fixing 1-lag transform of the sequence*

```
from numpy import sqrt
from sklearn.metrics import r2_score

# mse, mae = regressor.evaluate(X_test, y_test, verbose=0)
# print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))

d = real_stock_volume - predicted_stock_volume
mse = np.mean(d**2)
mae = np.mean(abs(d))

print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
print('')
print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume))
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
MSE: 99696452131.470, RMSE: 315747.450, MAE: 213090.209

R2 Score:  0.8113194498033985

R2 Score:  0.2520010330020507
```

*Figure 25 evaluation results of 50 timesteps after fixing 1-lag issue.*

After fixing issue of 1-lag transform of the sequence we got accuracy of 81.13%. But as we did not change fix 1-lag issue in scaled data we can see the accuracy for that is 25.20%. So, we can conclude that after fixing 1-lag issue accuracy of model with 100 epoch and 50 timesteps increase by 55.93%.

## Evaluate the model for 100 epochs and 40 timesteps.

*Before Fixing 1-lag transform of the sequence*

```
from numpy import sqrt
mse, mae = regressor.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
```

```
MSE: 0.004, RMSE: 0.062, MAE: 0.043
```

```
from sklearn.metrics import r2_score

print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume))
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
R2 Score:  0.26572243029176523

R2 Score:  0.26572244001043244
```

*Figure 26 evaluation results of 40 timesteps before fixing 1-lag issue.*

We can observe that accuracy of model is 26.57 % according to $R^2$ matrix which is very low. All mse, rmse and mae calculate for scaled values through model.

*After Fixing 1-lag transform of the sequence*

```
d = real_stock_volume - predicted_stock_volume
mse = np.mean(d**2)
mae = np.mean(abs(d))

print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
print('')
print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume)
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
MSE: 119892307813.247, RMSE: 346254.686, MAE: 244835.437

R2 Score:  0.7594437077291704

R2 Score:  0.1883833956672405
```

*Figure 27 evaluation results of 40 timesteps after fixing 1-lag issue.*

After fixing issue of 1-lag transform of the sequence we got accuracy of 75.94%. But as we didn't change fix 1-lag issue in scaled data we can see the accuracy for that is 18.83%. So, we can conclude that after fixing 1-lag issue accuracy of model with 100 epoch and 40 timesteps increase by 57.11%

## Evaluate the model for 100 epochs and 30 timesteps.

*Before Fixing 1-lag transform of the sequence*

```
from numpy import sqrt
mse, mae = regressor.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
```

```
MSE: 0.004, RMSE: 0.062, MAE: 0.043
```

```
from sklearn.metrics import r2_score

print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume))
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
R2 Score:  0.26572243029176523

R2 Score:  0.26572244001043244
```

*Figure 28 evaluation results of 30 timesteps before fixing 1-lag issue.*

We can observe that accuracy of model is 26.57 % according to $R^2$ matrix which is very low. All mse, rmse and mae calculate for scaled values through model.

*After Fixing 1-lag transform of the sequence*

```
d = real_stock_volume - predicted_stock_volume
mse = np.mean(d**2)
mae = np.mean(abs(d))

print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
print('')
print('R2 Score: ', r2_score(real_stock_volume, predicted_stock_volume))
print('')
print('R2 Score: ', r2_score(stock_volume, predicted_value))
```

```
MSE: 92912440336.196, RMSE: 304815.420, MAE: 197445.470

R2 Score:  0.80945283637062

R2 Score:  0.2746503347466841
```

*Figure 29 evaluation results of 30 timesteps after fixing 1-lag issue.*

After fixing issue of 1-lag transform of the sequence we got accuracy of 80.94%. But as we didn't change fix 1-lag issue in scaled data we can see the accuracy for that is 27.46%. So, we can conclude that after fixing 1-lag issue accuracy of model with 100 epoch and 30 timesteps increase by 53.48%

**Conclusion**: Model for 100 epochs and 50 timesteps after fixing the 1-lag issue gives the highest accuracy as 81.13% as R2 Score.

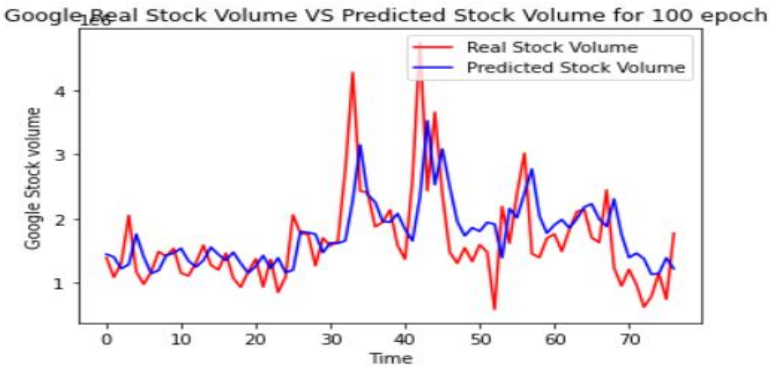
## Visualizing the Results

### Real Stock Volume VS Predicted Stock Volume line charts.

*For 50 timesteps*

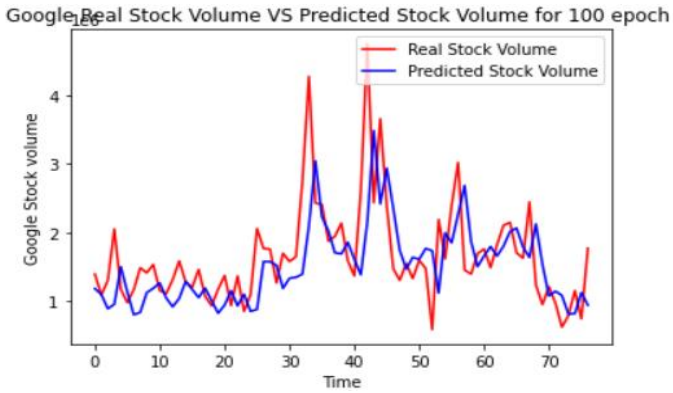
Before changing the LSTM Network Composition

When LSTM layer with 50,50,50,50 units with 4 layers and before fixing the lag-1 issue.

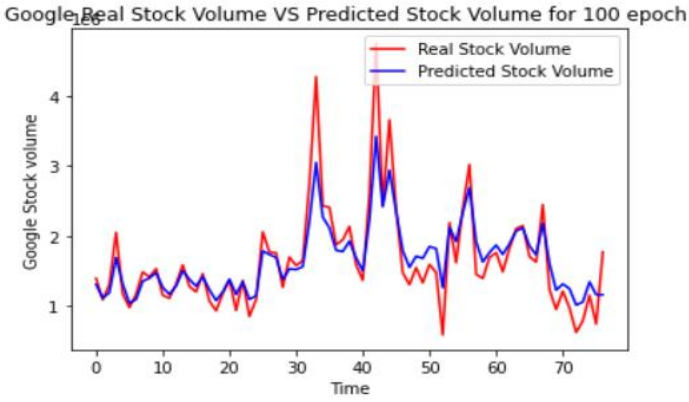



After changing the LSTM Network Composition

Before Fixing 1-lag transform of the sequence

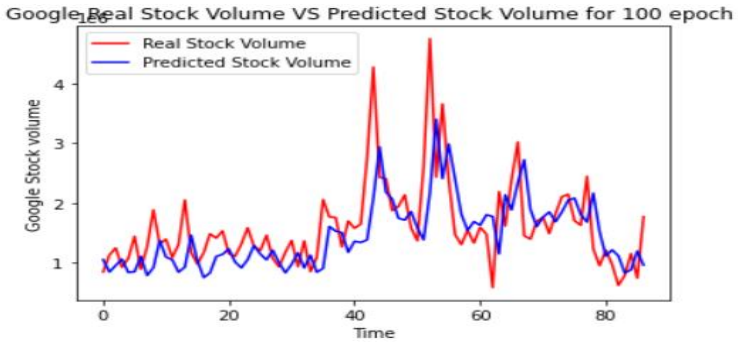



After Fixing 1-lag transform of the sequence

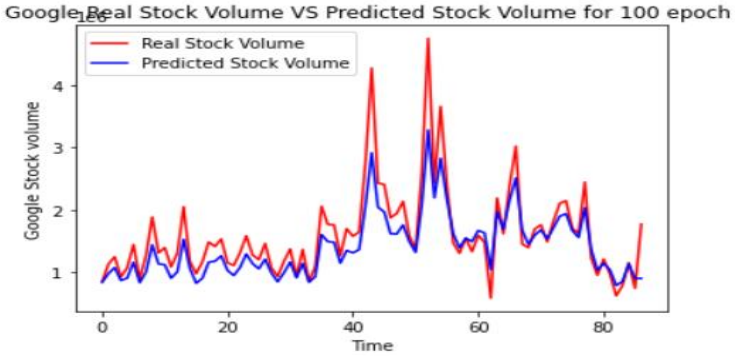


*For 40 timesteps*

Before Fixing 1-lag transform of the sequence

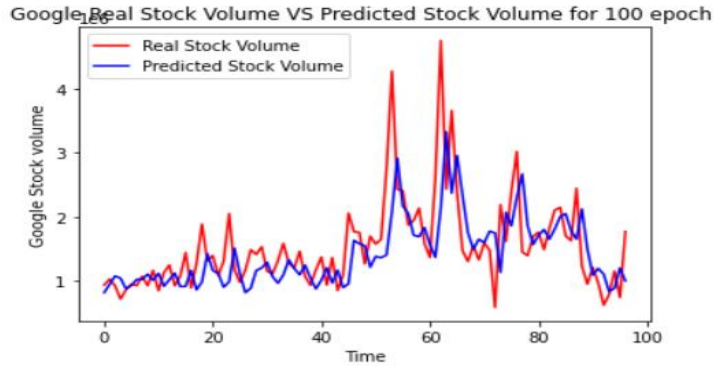



After Fixing 1-lag transform of the sequence

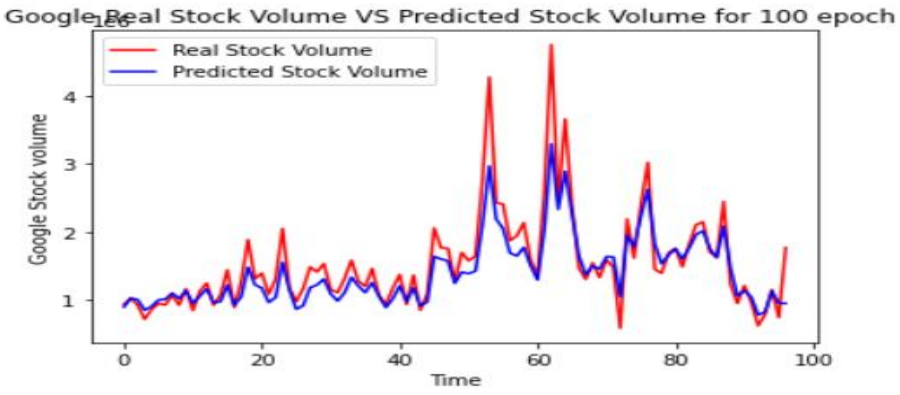



*For 30 timesteps*

Before Fixing 1-lag transform of the sequence

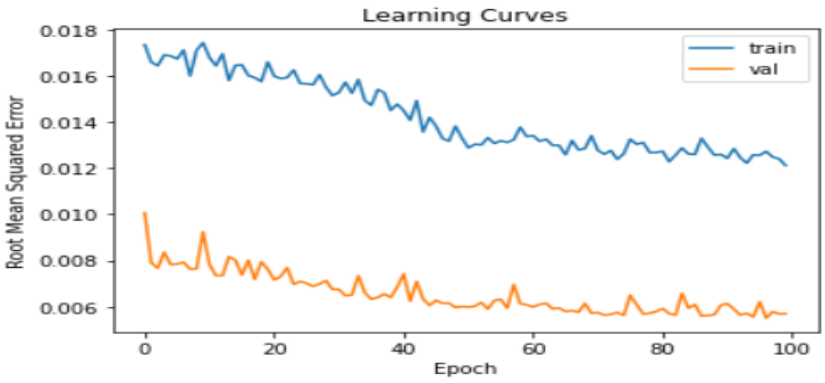


After Fixing 1-lag transform of the sequence

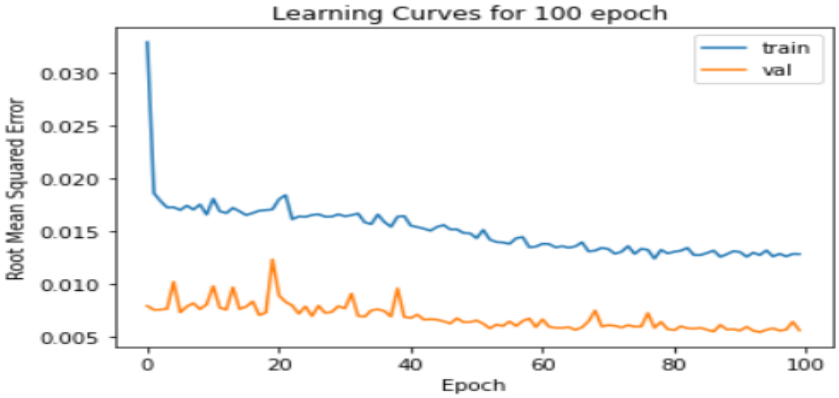



## Learning Curves.

*For 50 timesteps*




*For 40 timesteps*



*For 30 timesteps*