BY::
:SHYAM.M [411621104050]
:TANUJ KUMAR [411621104052]
: SHIBYA VISVAISWARAN.S [411621104049]
: VADDIBOINA SANTHOSH [411621104054]
: POOVA RAGAN[411621104040]

Stock prediction

[machine learning:::long short term memory]

Introduction::

“Stock market prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange.”

The efficient-market hypothesis indicates that the prices ofstock reflect all information, available currently and hence, any changes in price that are not based on information thatrecently came into light, are consequently unpredictable.

Thepeople who oppose this hypothesis therefore, possess myriadmethods and technologies which calculatedly allow them to gain information about the future price of stock. Stock market price data is tediously voluminous and extremely volatile.

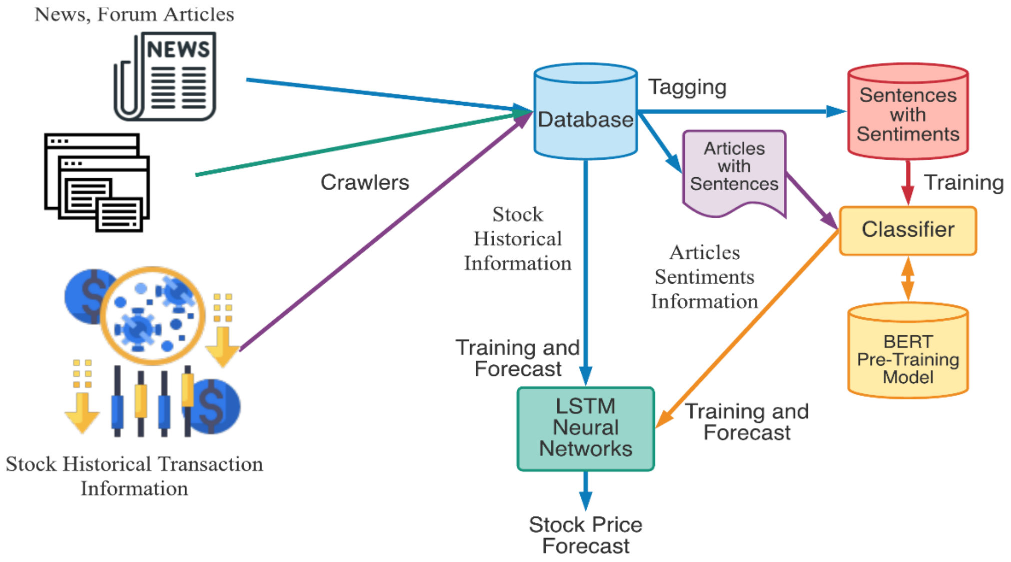
Stock market trading is an extremely complicatedand ever-changing system where people will either gain a fortune or lose their entire life savings. In this work, anattempt to build a Time-Series prediction model, to predictstock prices. This paper takes the current stock valuesfrom the data sets gathered

We use regression models in python or LSTM model the data. We run a comprehensive search algorithm on the data sets and create a summary table based on the output. We plot the values on a chart and apply regression and clustering techniques

LSTM(LONG SHORT TERM MEMORY)

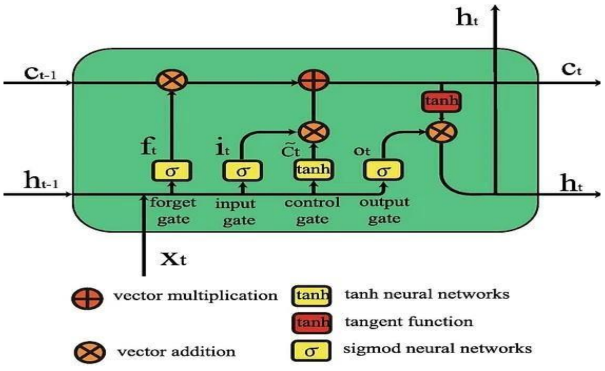
Understanding Long Short Term Memory Network for Stock Price Prediction
LSTM is a Recurrent Neural Network that works on data sequences, learning to retain only relevant information from a time window. New information the network learns is added to a “memory” that gets updated with each timestep based on how significant the new sample seems to the model. Over the years, LSTM has revolutionized speech and handwriting recognition, language understanding, forecasting, and several other applications that have become the new normal today.

A standard LSTM cell comprises of three gates: the input, output, and forget gate. These gates learn their weights and determine how much of the current data sample should be remembered and how much of the past learned content should be forgotten.



As seen in the equations below, i, f, and o represent the three gates: input, forget, and output. C is the cell state that preserves the learneddata, which is given as output h. Allof this is computed for each timestamp t, considering the learned data from timestamp (t-1).

$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$
$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$
$$h_t = \tanh(C_t) * o_t$$



The forget gate decides what information and how much of it canbe erased from the current cell state, while the input gate decides what will be added to the current cell state. The output gate, used in the final equation, controls the magnitude of output computed by the first two gates.

So, as opposed to standard feed-forward neural nets,LSTMs have the potential to remember or erase portions of the past data windows actively. Its featureof reading and training on windows (or timesteps) of data makes its training unique

Exploratory Data Analysis (EDA):

a. Distribution of price over time: Visualize the sales trends over the available time period.
b. Seasonality and trends: Identify any seasonal patterns or long-term trends in the data.
c. Correlations: Analyze correlations between features, especially with the target variable (price).
d. Outliers: Identify and investigate outliers that may affect your forecasting accuracy.
e. Store/item analysis: Explore sales patterns at the store and item levels.

To begin this exploratory analysis, first import libraries and define functions for plotting the data using `matplotlib`

```
In [1]: from mpl_toolkits.mplot3d import Axes3D
        from sklearn.preprocessing import StandardScaler
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        import pandas as pd
```

Data Preprocessing:

- Data cleaning: Check for missing values, duplicate records, and outliers in the dataset.
- Handle missing data by either removing, imputing, or interpolating missing values based on the nature of the data and the impact of missing values on your analysis.
- Check for and remove duplicate records if any are found in the dataset.
- Identify and deal with outliers that may adversely affect the accuracy of your forecasting model. You can consider techniques such as trimming, winsorizing, or transforming the data.
- Convert categorical variables into numerical format, either by using one-hot encoding, label encoding, or other suitable methods.
- Explore and visualize the data to gain insights into its distribution, trends, and potential patterns that may inform your forecasting model.
- Split the dataset into training and validation sets, typically reserving a portion of the data for model evaluation.

Model Evaluation:

- Accuracy: I used accuracy as a fundamental metric for classification tasks. It measures the ratio of correctly predicted instances to the total number of instances. A higher accuracy indicates better model performance in correctly classifying data points.
- Root Mean Square Error (RMSE): For regression tasks, I calculated RMSE to assess the model's predictive accuracy. RMSE quantifies the average deviation between the predicted values and the actual values. Lower RMSE values signify better predictive accuracy.

$$RMSE = \sqrt{\frac{1}{N} * \sum_{t=1}^{N} (At - Ft)^2}$$

- Mean Absolute Error (MAE): MAE is another metric for regression tasks. It calculates the average absolute difference between predicted and actual values. Like RMSE, lower MAE values indicate better model performance in terms of prediction accuracy.

$$MAPE = \frac{1}{N} * \sum_{t=1}^{N} \left| \frac{At - Ft}{At} \right|$$

Model Interpretability:

- Address the first point related to model interpretability. Discuss how you ensured that your chosen models are interpretable. This could involve explaining feature importance, visualization of results, or any other techniques you used to make the models transparent and explainable.

Model Development:

- Discuss the second point about model development. Describe the steps you took to develop the models, including data splitting, training, and validation.
- Explain any challenges you encountered during model development and how you overcame them.

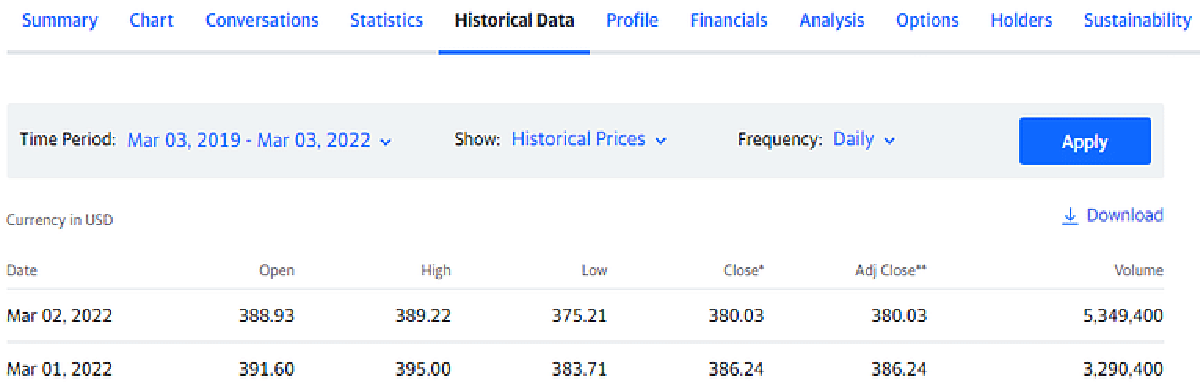
Prediction and Results:

- Present the results of your time series forecasting models. Include performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and any others that are relevant.

Downloading the Stock Prices Dataset for Project on Machine Learning Price Prediction of Stocks

Go to finance.yahoo.com/ and search the company whose data you want to seek for stock price prediction. For our example, we will look at the Netflix (NFLX) stock over 3 years.

Going to finance.yahoo.com/quote/NFLX/history?p=NFLX in the "Historical Data" section, we see the stock data listed each day. We can filter out the time for which we wish to analyze and download the CSV file using the download button on the right.



Time Period: Mar 03, 2019 - Mar 03, 2022

Show: Historical Prices

Frequency: Daily

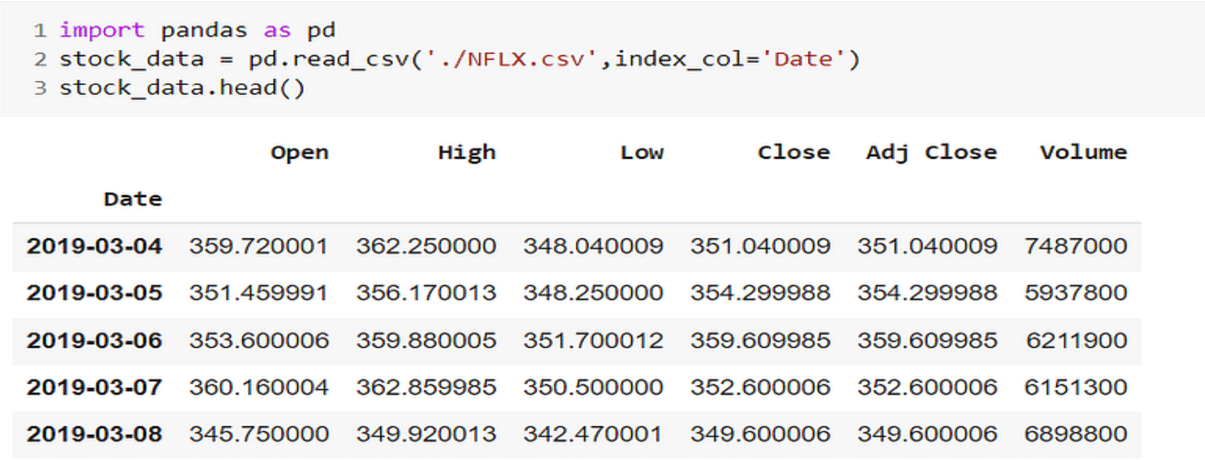
Currency in USD

Date	Open	High	Low	Close*	Adj Close**	Volume
Mar 02, 2022	388.93	389.22	375.21	380.03	380.03	5,349,400
Mar 01, 2022	391.60	395.00	383.71	386.24	386.24	3,290,400

The download CSV file will contain the data for Open, High, Low, Close, Adj Close, Volume for each date, as shown in the image above.

Loading the Stock Prices Dataset

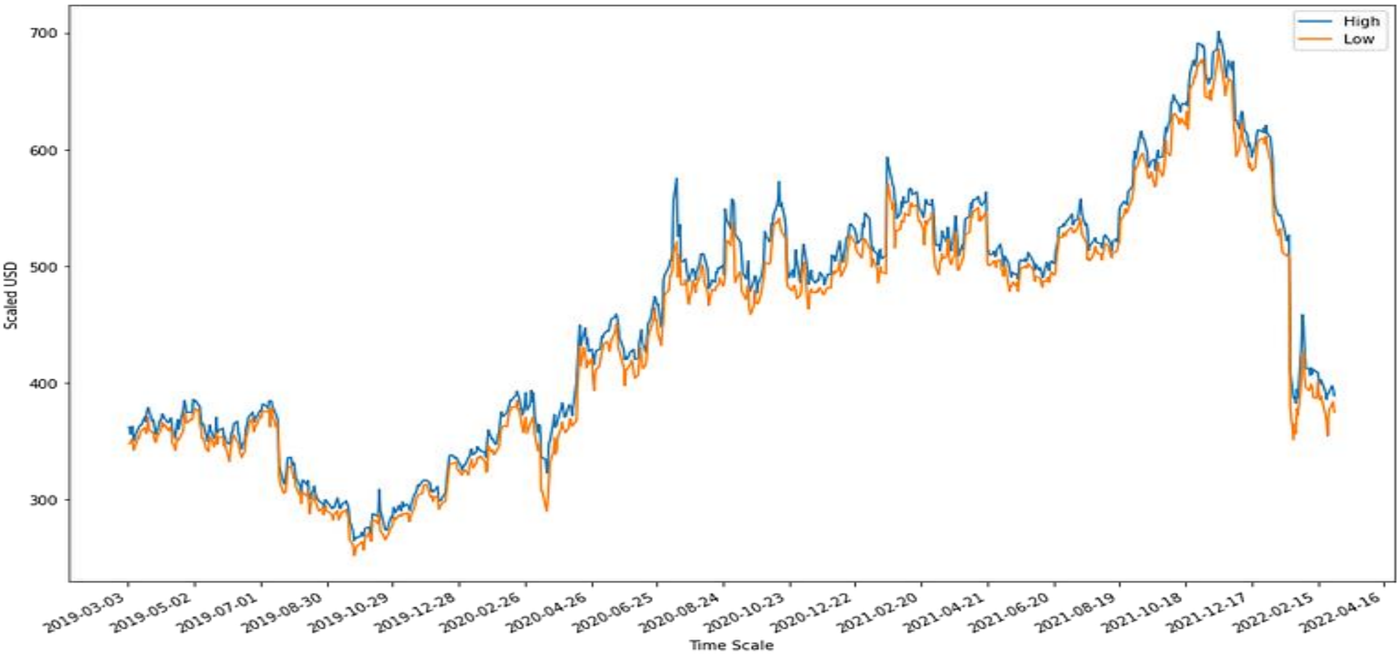
Load the CSV file as a DataFrame using Pandas. Since the data is indexed by date (each row represents data from a different date), we can also index our DataFrame by the date column. We have taken the data from March 2019 to March 2022.



```python
import pandas as pd
stock_data = pd.read_csv('./NFLX.csv',index_col='Date')
stock_data.head()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2019-03-04	359.720001	362.250000	348.040009	351.040009	351.040009	7487000
2019-03-05	351.459991	356.170013	348.250000	354.299988	354.299988	5937800
2019-03-06	353.600006	359.880005	351.700012	359.609985	359.609985	6211900
2019-03-07	360.160004	362.859985	350.500000	352.600006	352.600006	6151300
2019-03-08	345.750000	349.920013	342.470001	349.600006	349.600006	6898800

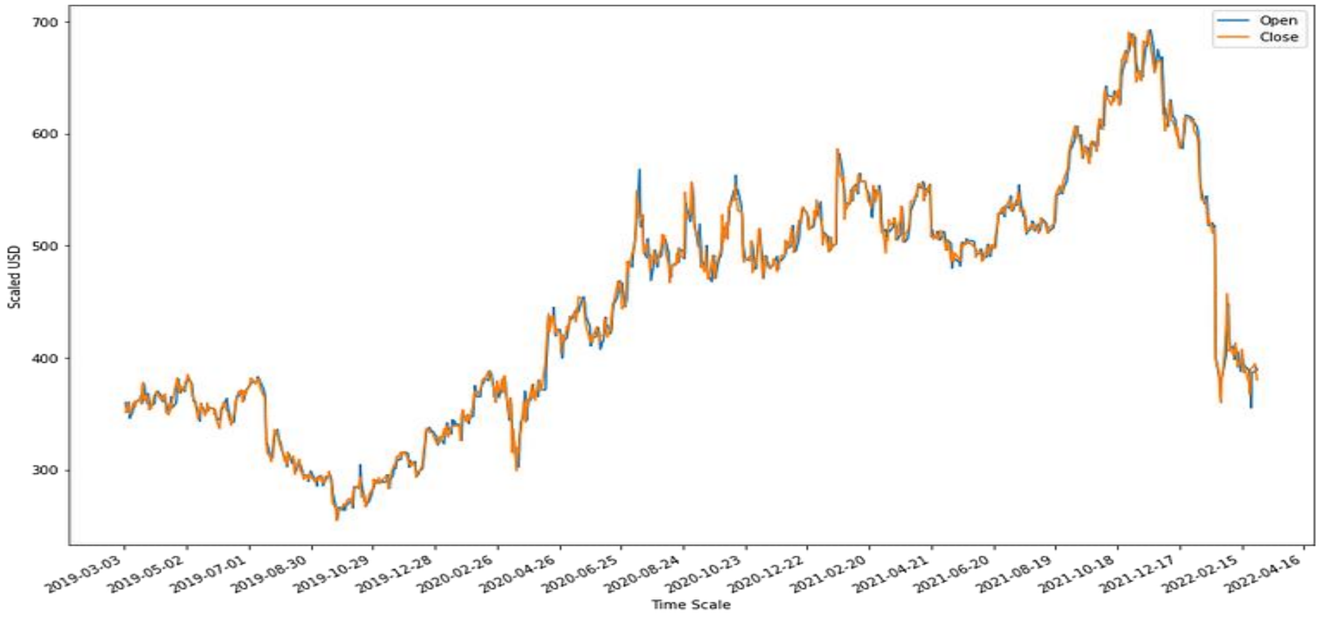
Plotting the High and Low points of Netflix stock over 3 years, we see the below graph.



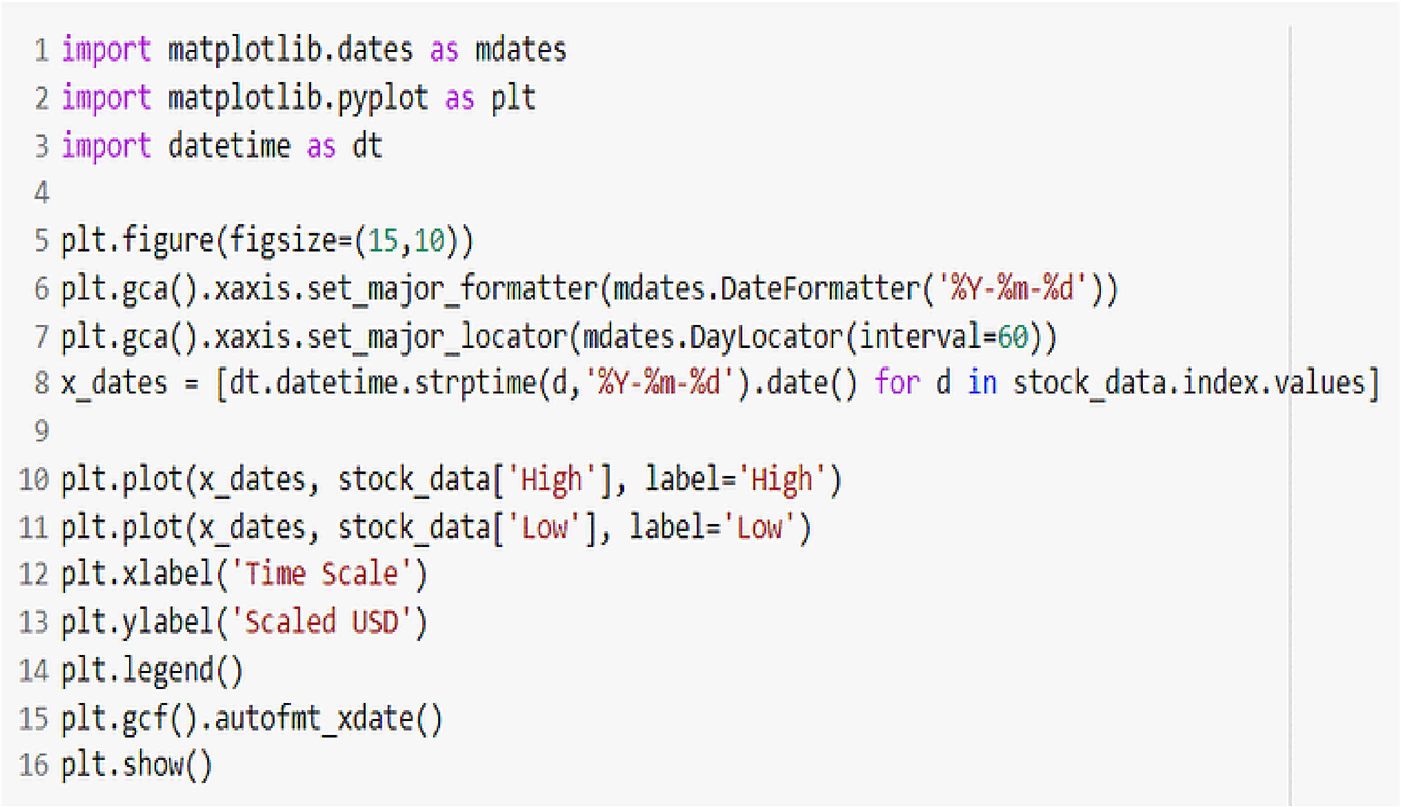
As noticeable, around March 2020, we see a sudden drop in the price, after which it reports steady growth until recently.

It will be challenging for a model in the stock prediction using machine learning project to correctly estimate the rapid changes that we can see in March 2020 and February 2022. We will focus on evaluating the model performance in predicting the more recent values after training it on the past data.

Similarly, plotting the Open and Close value of the stock for each day gives equivalent observations.



The code for plotting these graphs is as shown below. We use matplotlib to plot the DataFrame columns directly against the Date index column. To make things flexible while plotting against dates, lines 6-8 convert our date strings into datetime format and plot them cleanly and legibly. The interval parameter in line 7 defines the interval in days between each tick on the date axis.

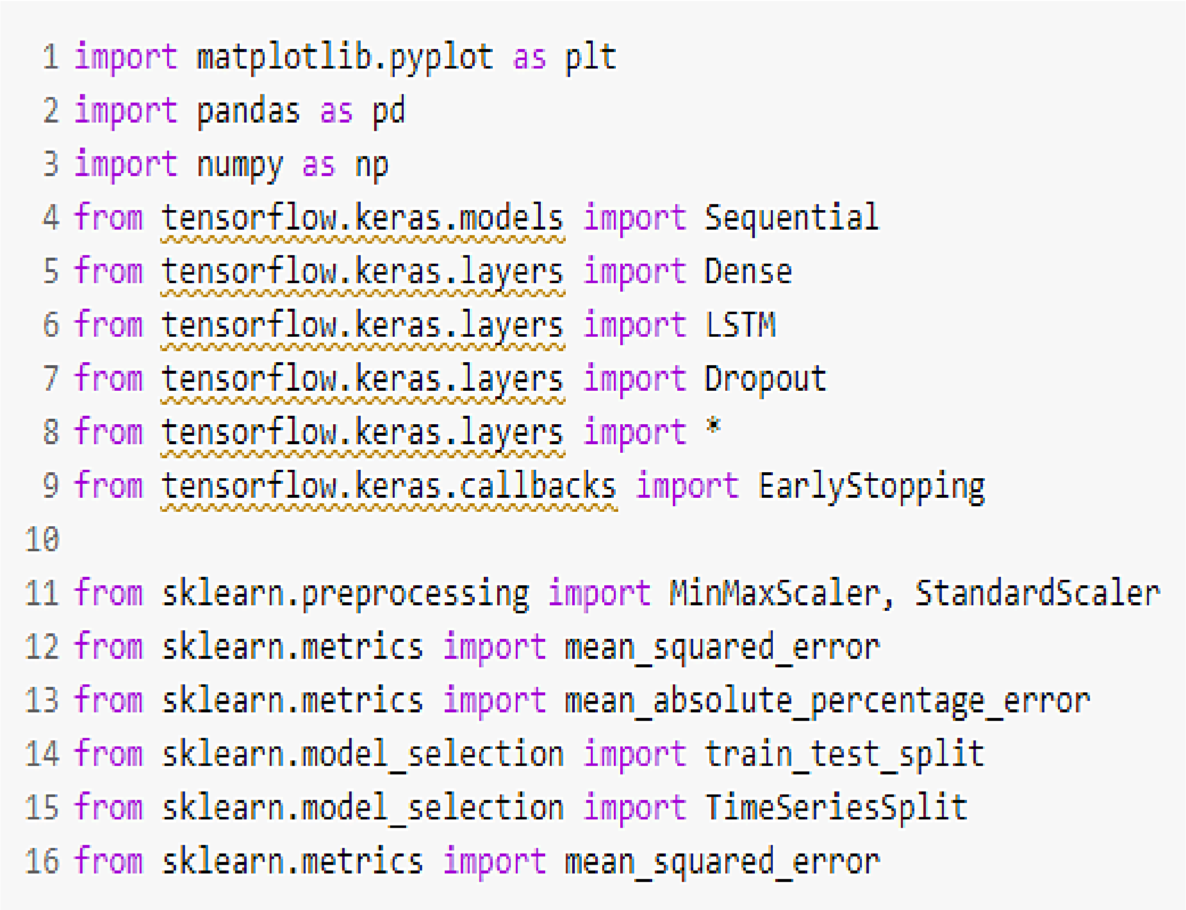


```
 1 import matplotlib.dates as mdates
 2 import matplotlib.pyplot as plt
 3 import datetime as dt
 4 
 5 plt.figure(figsize=(15,10))
 6 plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
 7 plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=60))
 8 x_dates = [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in stock_data.index.values]
 9 
10 plt.plot(x_dates, stock_data['High'], label='High')
11 plt.plot(x_dates, stock_data['Low'], label='Low')
12 plt.xlabel('Time Scale')
13 plt.ylabel('Scaled USD')
14 plt.legend()
15 plt.gcf().autofmt_xdate()
16 plt.show()
```

We will use the Open, High, and Low columns to predict the Closing value of the Netflix stock for the next day.

Importing the Libraries for Stock Price Prediction Project

We will be building our LSTM models using Tensorflow Keras and preprocessing our stock prediction machine learning data using scikit-learn.

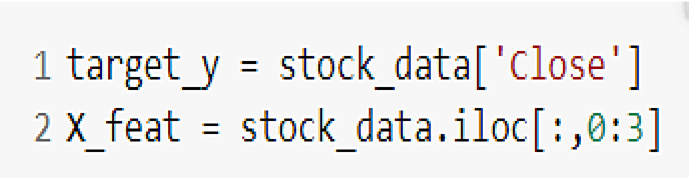


```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
```

Data Preprocessing for Stock Market Prediction using Machine Learning

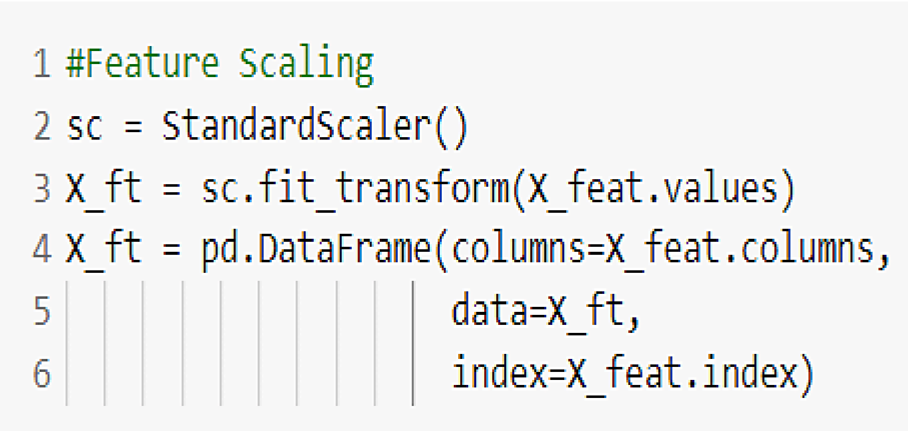
As with any other machine learning model, it is always good to normalize or rescale the data within a fixed range when dealing with real data. This will avoid features with larger numeric values to unjustly interfere and bias the model and help achieve rapid convergence in the machine learning stock prediction project.

First, we define the features and the target as discussed above.



```python
target_y = stock_data['Close']
X_feat = stock_data.iloc[:,0:3]
```

Next, we use a StandardScaler to rescale our values between -1 and 1.



```
#Feature Scaling
sc = StandardScaler()
X_ft = sc.fit_transform(X_feat.values)
X_ft = pd.DataFrame(columns=X_feat.columns,
                    data=X_ft,
                    index=X_feat.index)
```

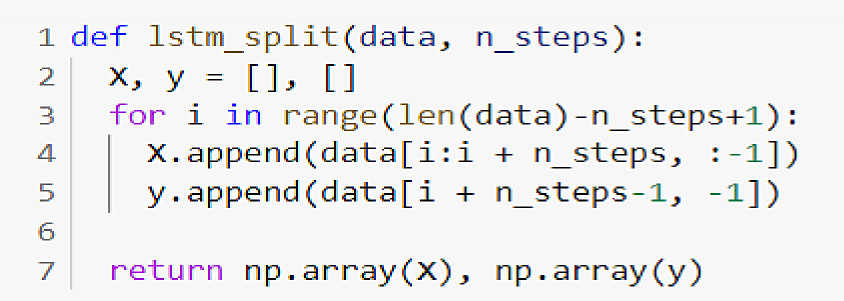
The change in values from using either of these methods would not be much, so we stick to StandardScaler.

We have 757 data samples in the dataset.

So, the next step would be to split it into training and testing sets. For instance, the LSTM will take 10 data samples to predict the 10th one by weighing the first nine input samples in one step. So, we need a different approach than the train_test_split provided by scikit-learn.

Let splitting function called lstm_split() which will make windows of size “n_steps” starting from the first sample of data and ending at n_steps’th sample (if n_steps=10, then the 10th sample) from the end. We understand the latter part because, for each time step, LSTM will take n_steps-1 samples for training and predict the last sample. Loss calculation is done based on the error in this prediction. So if n_steps=10, you cannot use the last 9 samples to predict anything because the “10th” data point for the current step does not exist in the dataset.

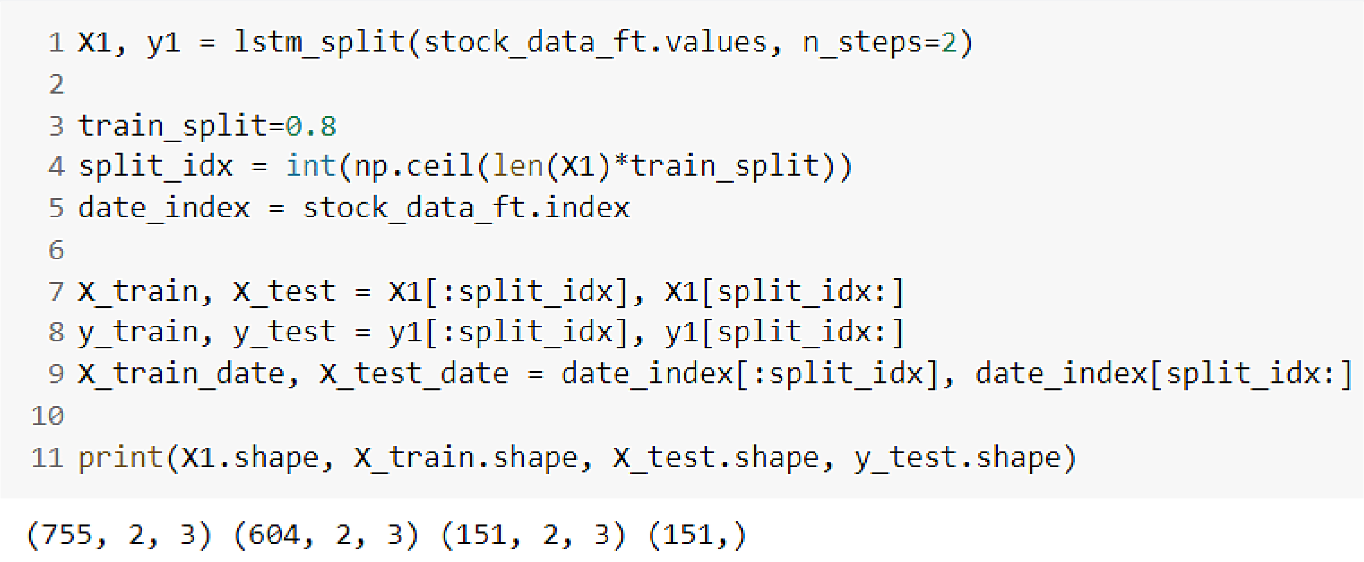
The function below takes the entire data and creates windows of size n_steps starting from the beginning. The target y will contain the target value corresponding to the n_steps’th index. So if n_steps is 10, the first element in X will have features from 10 data samples, and y will contain the target of the 10th data sample.



```
def lstm_split(data, n_steps):
    X, y = [], []
    for i in range(len(data)-n_steps+1):
        X.append(data[i:i + n_steps, :-1])
        y.append(data[i + n_steps-1, -1])

    return np.array(X), np.array(y)
```

Train and Test Sets for Stock Price Prediction Project

We split our data into training and testing sets. Shuffling is not permitted in time-series datasets. In the beginning, we take two steps worth of past data to predict the current value. Thus, the model will look at yesterday’s and today’s values to predict today’s closing price.



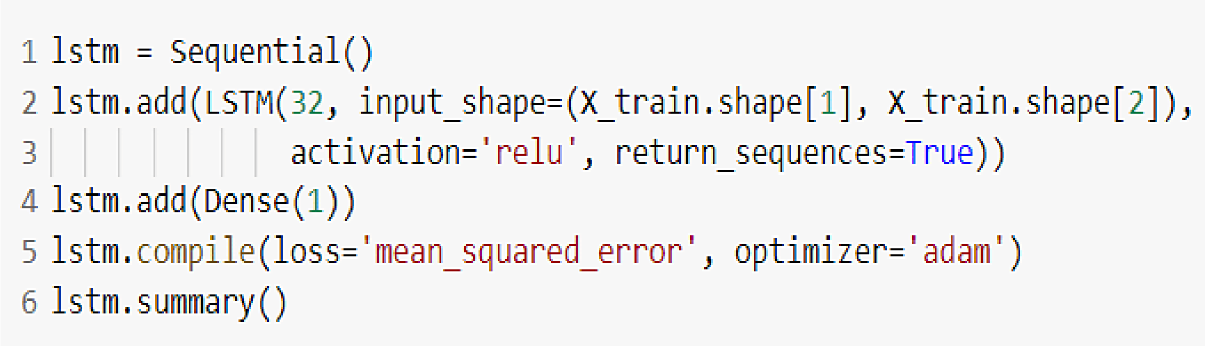
```
X1, y1 = lstm_split(stock_data_ft.values, n_steps=2)

train_split=0.8
split_idx = int(np.ceil(len(X1)*train_split))
date_index = stock_data_ft.index

X_train, X_test = X1[:split_idx], X1[split_idx:]
y_train, y_test = y1[:split_idx], y1[split_idx:]
X_train_date, X_test_date = date_index[:split_idx], date_index[split_idx:]

print(X1.shape, X_train.shape, X_test.shape, y_test.shape)
```

```
(755, 2, 3) (604, 2, 3) (151, 2, 3) (151,)
```

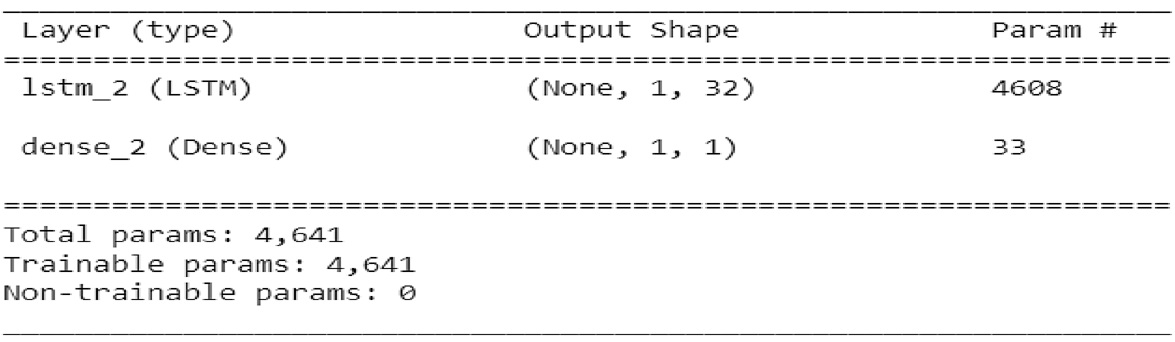
Note above that the size of X1 is n_steps less than that of the original dataset. As we explained above, you cannot use the last two samples of the original set during training or prediction as we do not have their corresponding ground truth values.

Stock Prediction Machine Learning Project- Building the LSTM model

We will use the Sequential and LSTM modules provided by Tensorflow Keras to build a simple, single-unit LSTM model.

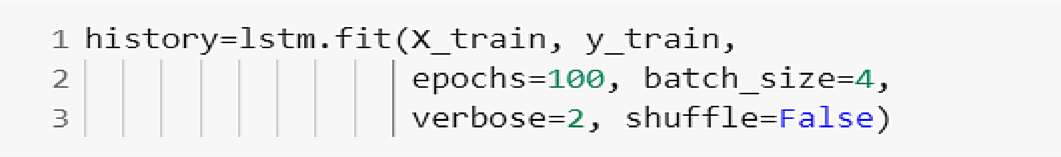


```
lstm = Sequential()
lstm.add(LSTM(32, input_shape=(X_train.shape[1], X_train.shape[2]),
              activation='relu', return_sequences=True))
lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error', optimizer='adam')
lstm.summary()
```

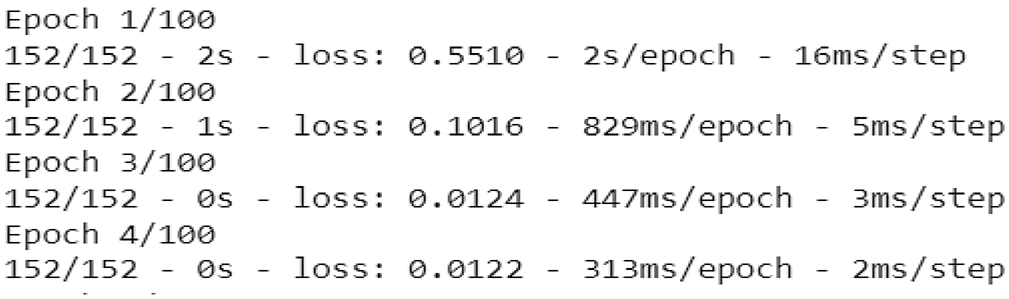


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 1, 32)             4608

 dense_2 (Dense)             (None, 1, 1)              33

=================================================================
Total params: 4,641
Trainable params: 4,641
Non-trainable params: 0
_________________________________________________________________
```

Now we can fit this simple model to the training data.

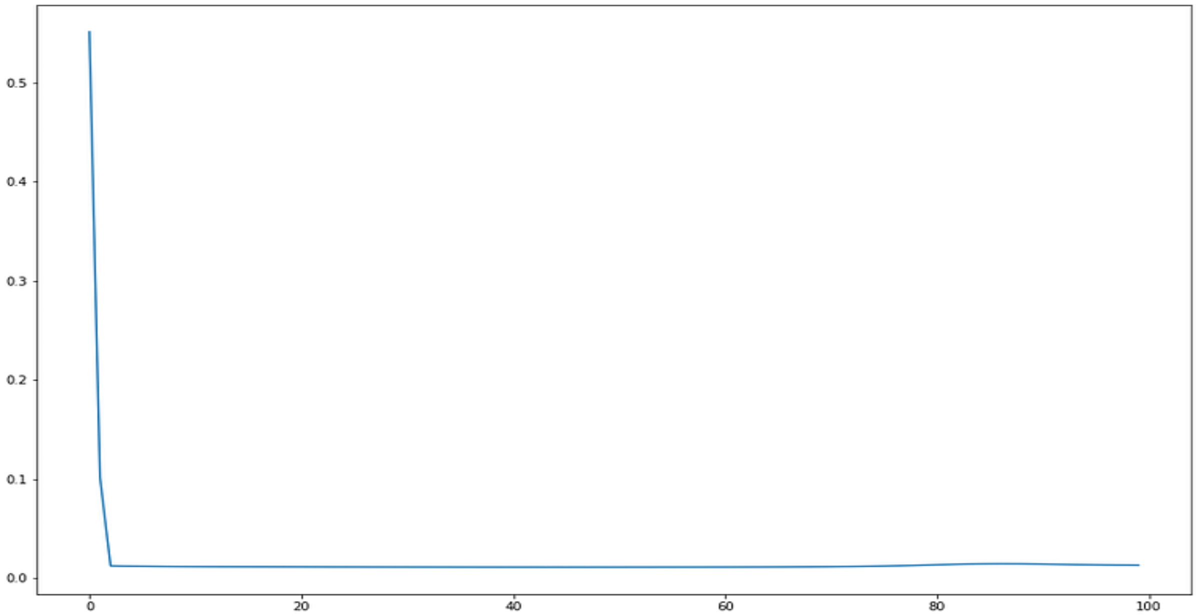


```
history=lstm.fit(X_train, y_train,
                 epochs=100, batch_size=4,
                 verbose=2, shuffle=False)
```



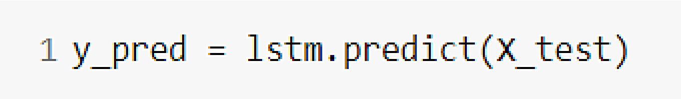
```
Epoch 1/100
152/152 - 2s - loss: 0.5510 - 2s/epoch - 16ms/step
Epoch 2/100
152/152 - 1s - loss: 0.1016 - 829ms/epoch - 5ms/step
Epoch 3/100
152/152 - 0s - loss: 0.0124 - 447ms/epoch - 3ms/step
Epoch 4/100
152/152 - 0s - loss: 0.0122 - 313ms/epoch - 2ms/step
```

Given the simplicity of the model and the data, we note that the loss reduction stagnates after only 20 epochs. You can observe this by plotting the training loss against the number of epochs, and LSTM does not learn much after 10-20 epochs.



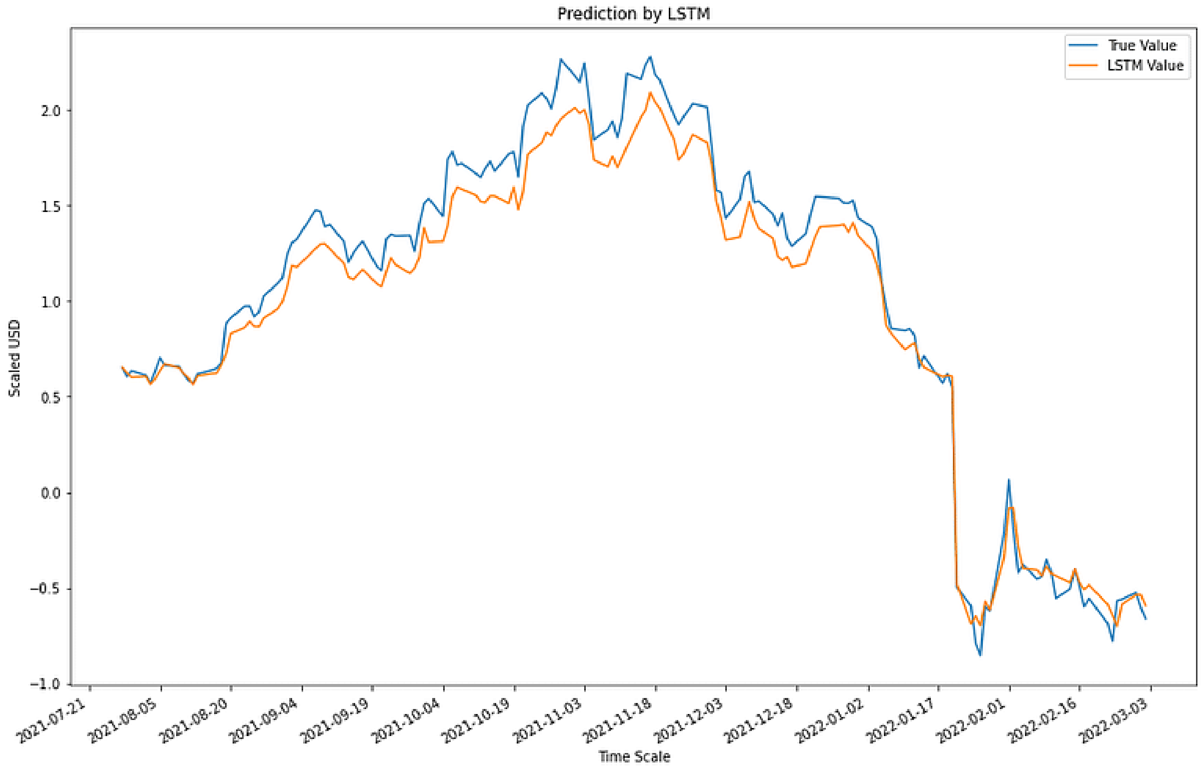
Machine Learning Stock Prediction Project- Performance Evaluation on Test Set

Nevertheless, we can check the performance of our model on a test set as below.



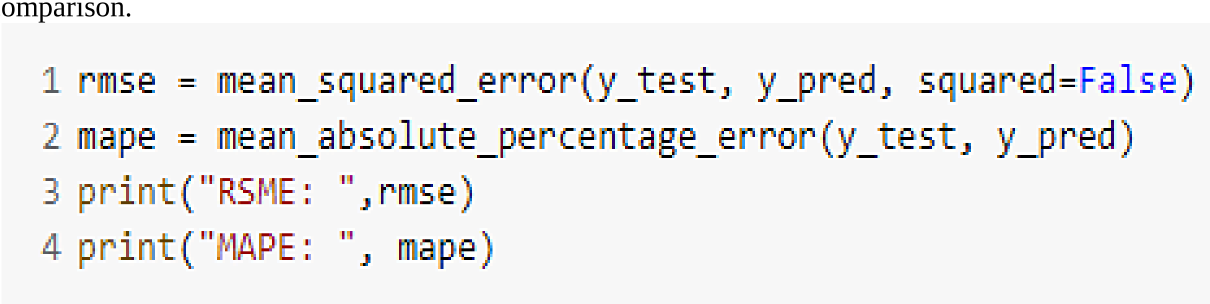
```
y_pred = lstm.predict(X_test)
```

To evaluate, first, we plot the curve for true values and overlap it with that for the predicted values.

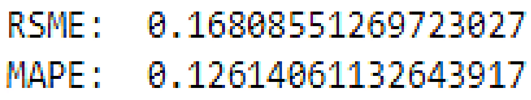


Thus, we can see that LSTM can emulate the trends of the stock prices to a certain extent. Based on the recent dip in prices, it has also fit the dropping curve well.

As we decided earlier, we can also check the RMSE and MAPE values to evaluate the performance. We will use these values for future comparison.

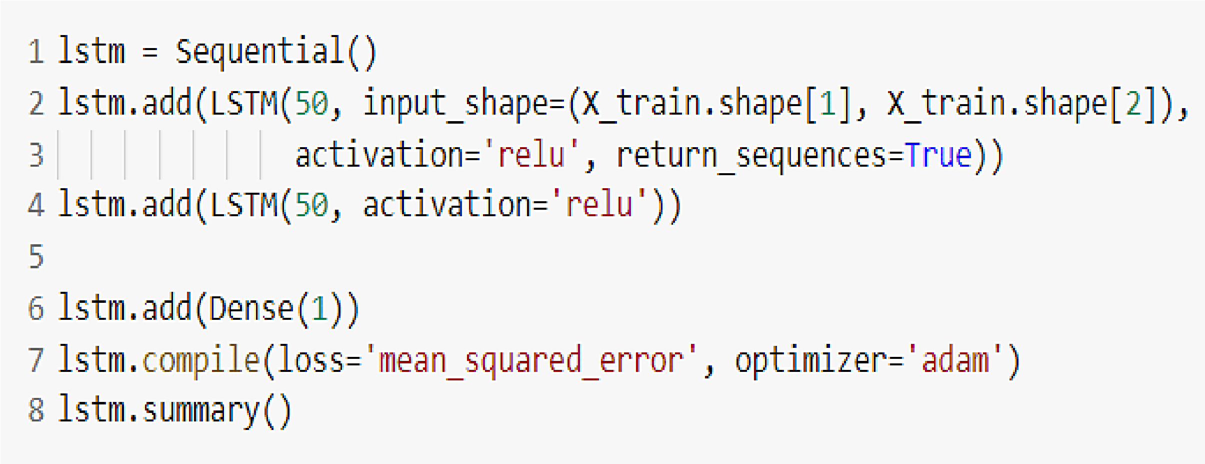


```
rmse = mean_squared_error(y_test, y_pred, squared=False)
mape = mean_absolute_percentage_error(y_test, y_pred)
print("RSME: ",rmse)
print("MAPE: ", mape)
```



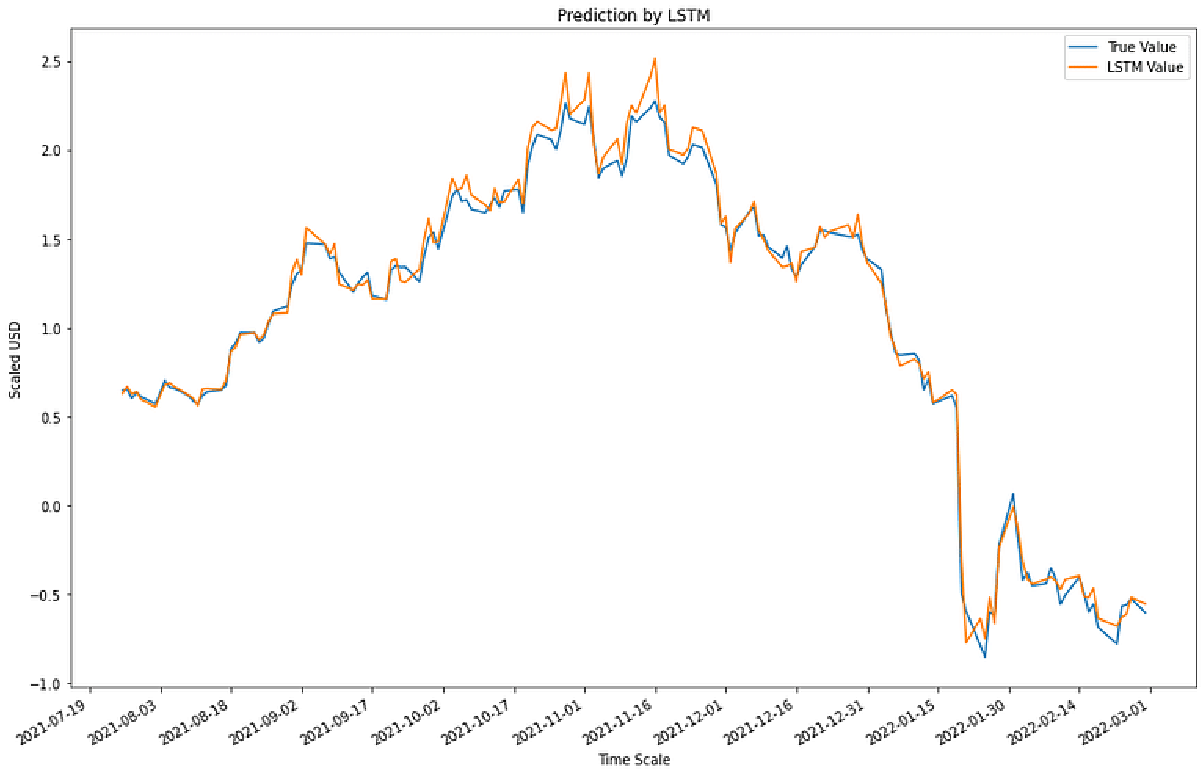
```
RSME:  0.16808551269723027
MAPE:  0.12614061132643917
```

Let’s try to get better results with the same dataset but a deeper LSTM model.

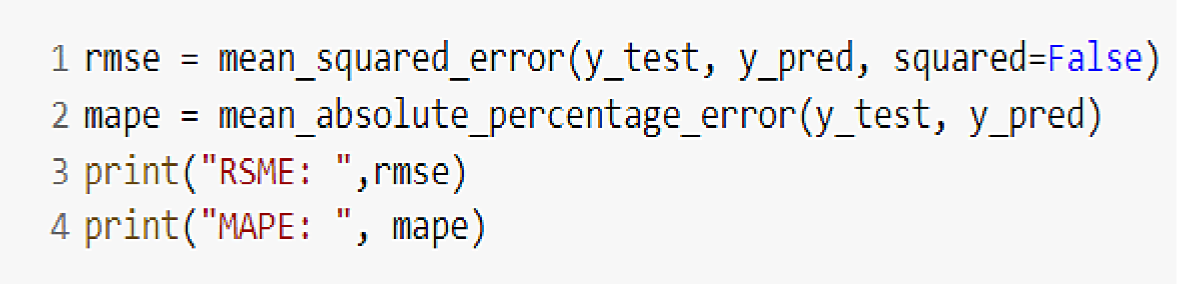


```
lstm = Sequential()
lstm.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]),
                            activation='relu', return_sequences=True))
lstm.add(LSTM(50, activation='relu'))

lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error', optimizer='adam')
lstm.summary()
```

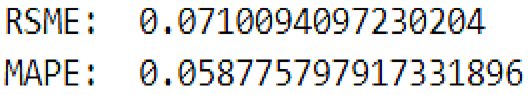
We added another LSTM layer and increased the number of LSTM units per layer to 50.



Moreover, the RMSE and MAPE values are better too.



```
rmse = mean_squared_error(y_test, y_pred, squared=False)
mape = mean_absolute_percentage_error(y_test, y_pred)
print("RSME: ",rmse)
print("MAPE: ", mape)
```

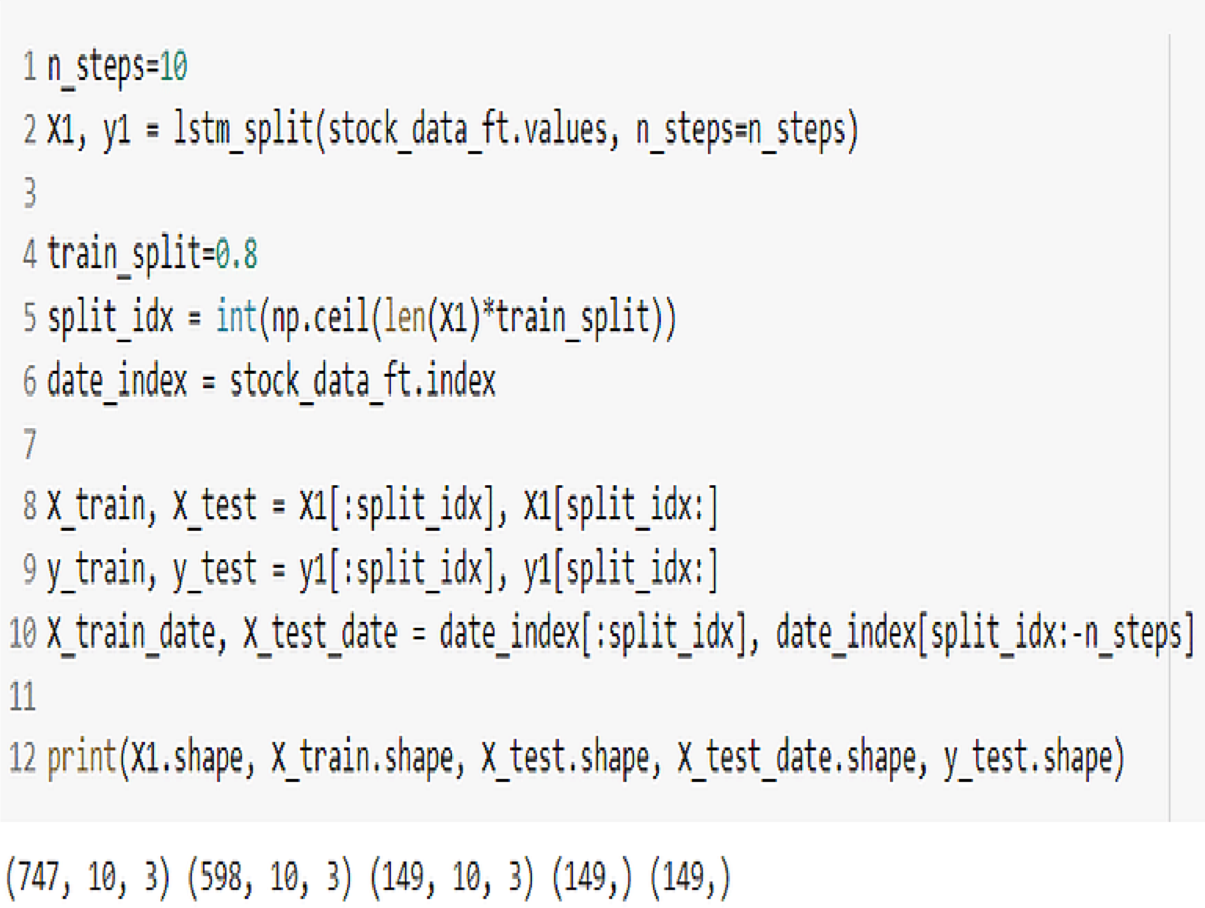


```
RSME:  0.07100940972302O4
MAPE:  0.058775797917331896
```

Thus we observe substantial improvement by adding another LSTM layer to the model. However, further adding even more layers would not be fruitful as the model might overfit or stagnate during training.

Now we will try fitting the same model but with increased time steps. We’ll try for n_steps=10.

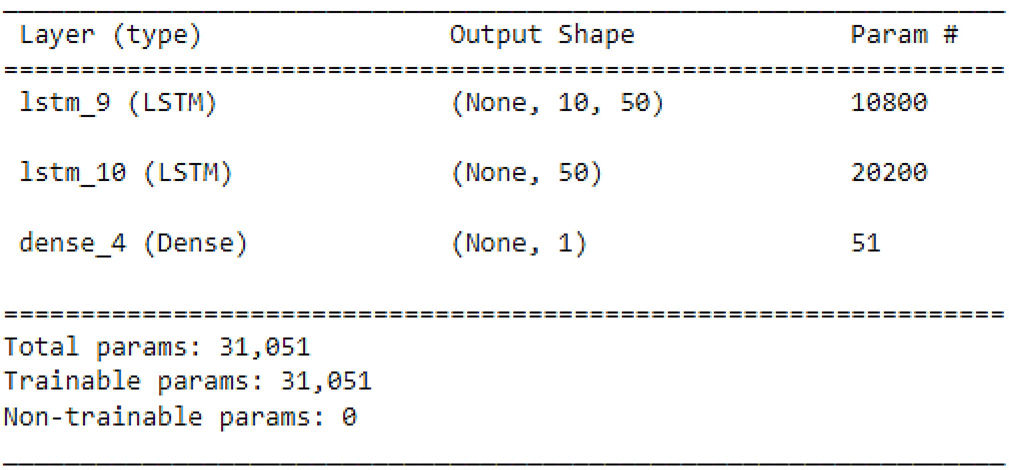
We change the value in the block below and rerun the entire process with the same model as before.



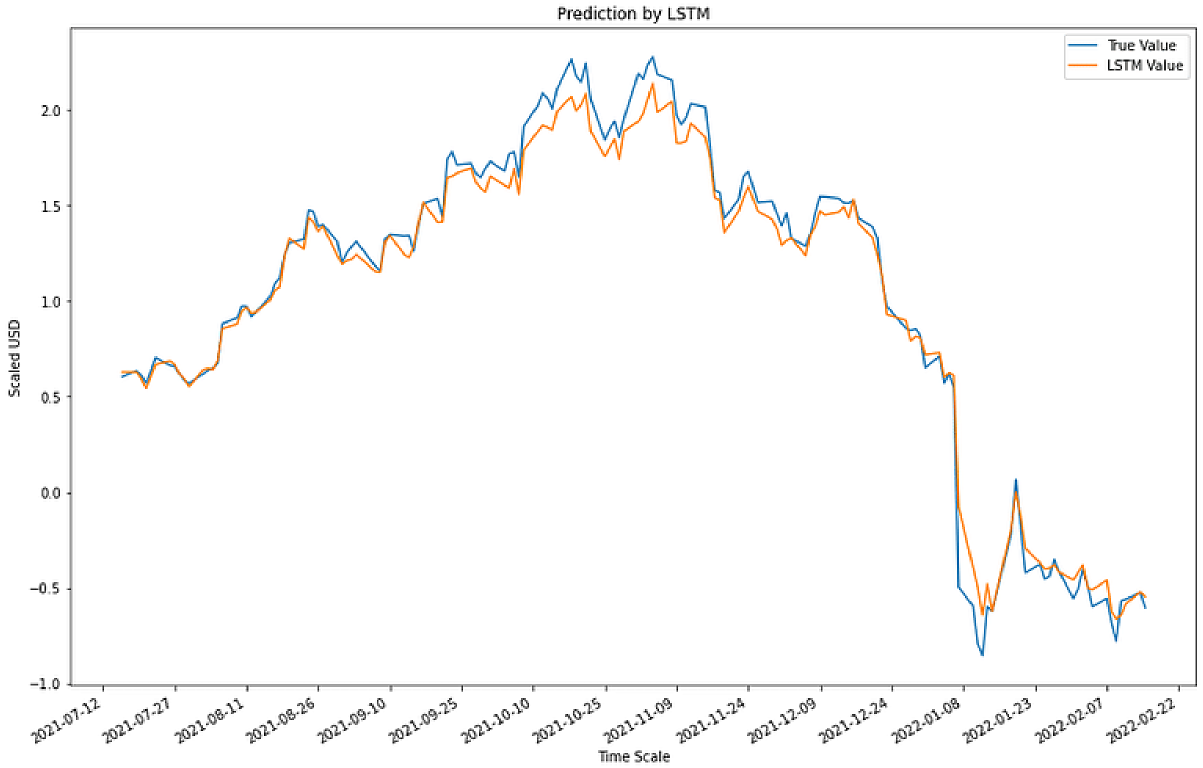
```
n_steps=10
X1, y1 = lstm_split(stock_data_ft.values, n_steps=n_steps)

train_split=0.8
split_idx = int(np.ceil(len(X1)*train_split))
date_index = stock_data_ft.index

X_train, X_test = X1[:split_idx], X1[split_idx:]
y_train, y_test = y1[:split_idx], y1[split_idx:]
X_train_date, X_test_date = date_index[:split_idx], date_index[split_idx:-n_steps]

print(X1.shape, X_train.shape, X_test.shape, X_test_date.shape, y_test.shape)
```

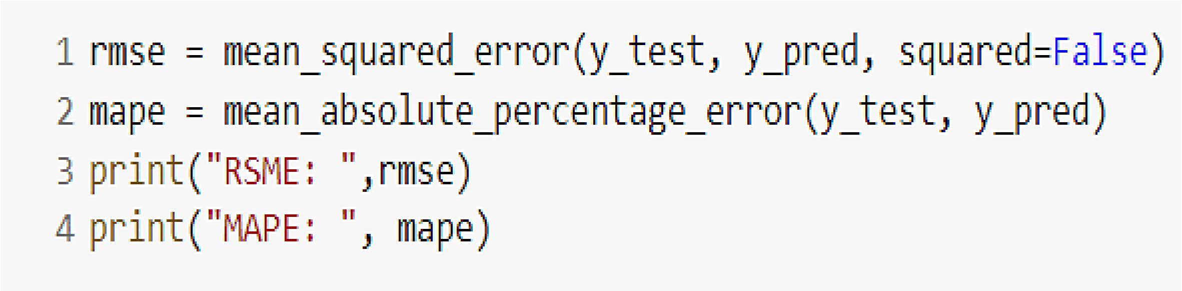
```
(747, 10, 3) (598, 10, 3) (149, 10, 3) (149,) (149,)
```

The model we used above:

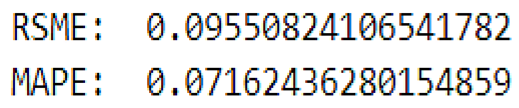


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_9 (LSTM)               (None, 10, 50)            10800

 lstm_10 (LSTM)              (None, 50)                20200

 dense_4 (Dense)             (None, 1)                 51

=================================================================
Total params: 31,051
Trainable params: 31,051
Non-trainable params: 0
_________________________________________________________________
```





```
rmse = mean_squared_error(y_test, y_pred, squared=False)
mape = mean_absolute_percentage_error(y_test, y_pred)
print("RSME: ",rmse)
print("MAPE: ", mape)
```



```
RSME:  0.09550824106541782
MAPE:  0.07162436280154859
```

We note that LSTM was able to achieve decent RMSE and MAPE values despite the data complexity. Further, we note that creating even deeper networks did not help improve the test performance of the stock price prediction model.

Before we conclude, as promised earlier, let’s look at how better or worse LSTMs perform compared with statistical techniques such as SMA and EMA.