STOCK PRICE PREDICTION USING ML MODEL

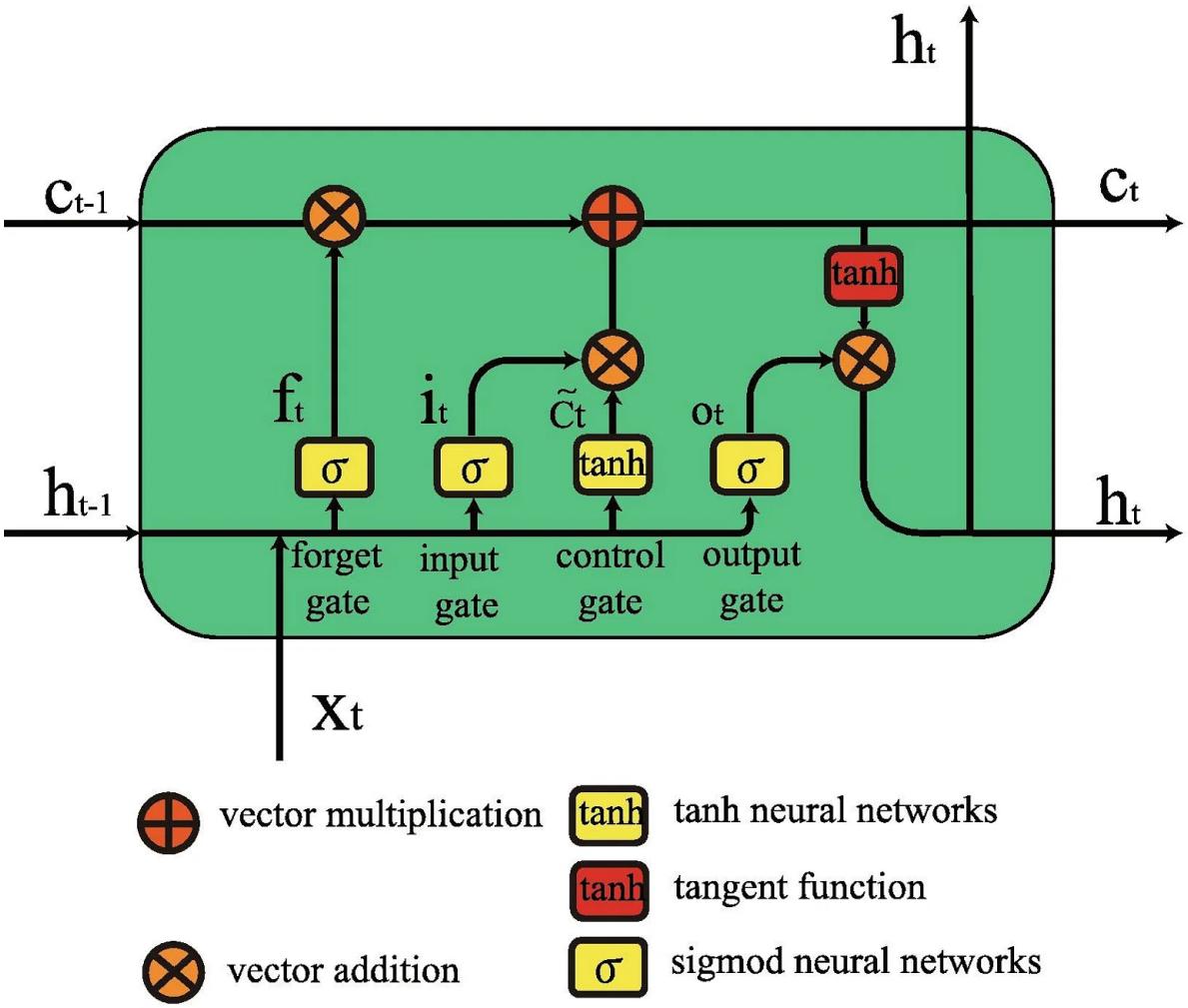
Stock Price Prediction using machine learning is the process of predicting the future value of a stock traded on a stock exchange for reaping profits. With multiple factors involved in predicting stock prices, it is challenging to predict stock prices with high accuracy, and this is where machine learning plays a vital role.

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. This simple structure is an improvement over the previous and similar RNN model.

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



The forget gate decides what information and how much of it can be 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.

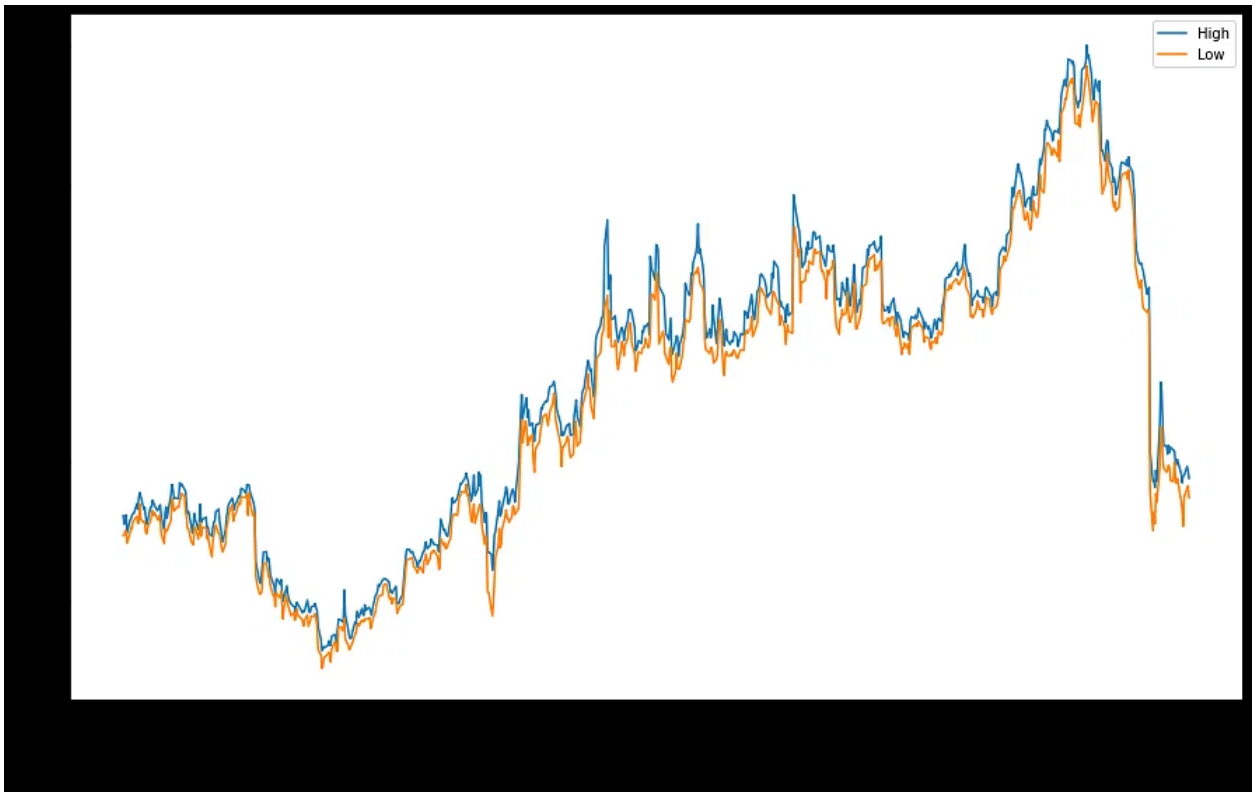
CODE:

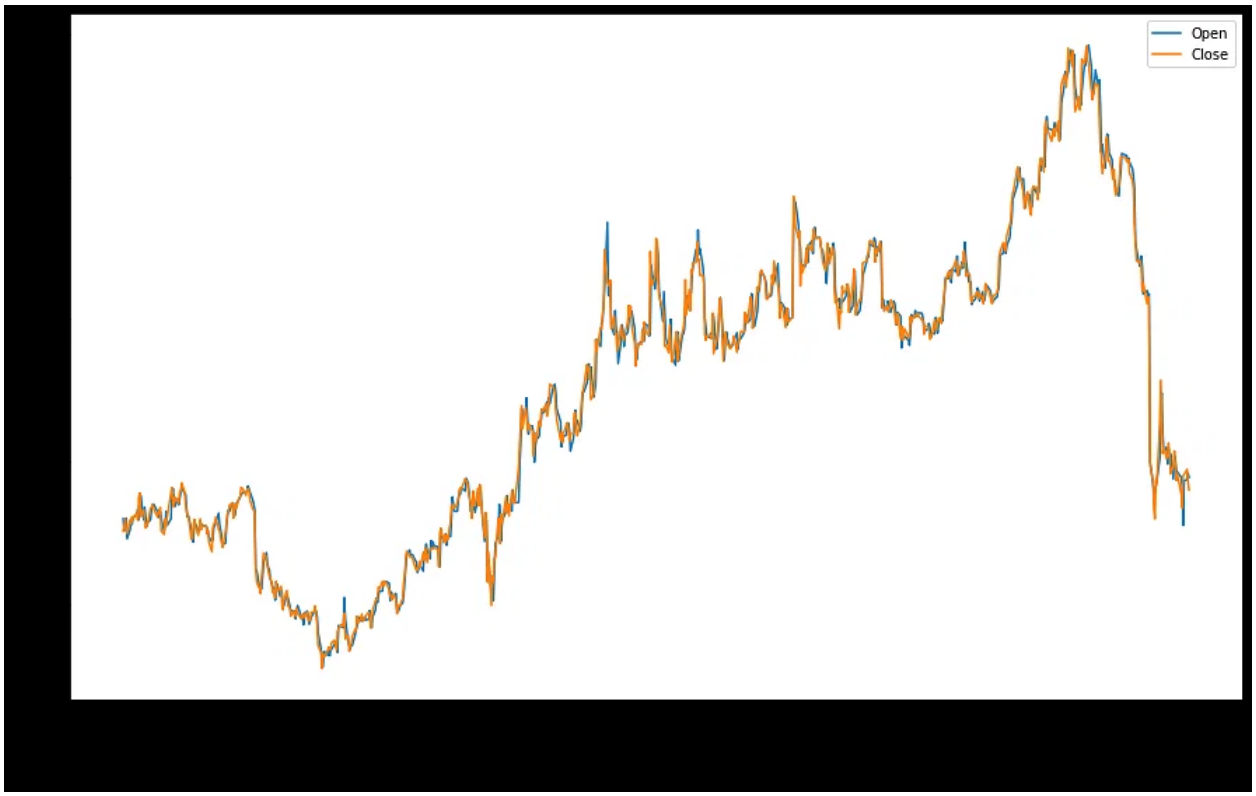
```
import pandas as pd
stock_data = pd.read_csv('./NFLX.csv',index_col='Date')
stock_data.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
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

```
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import datetime as dt

plt.figure(figsize=(15,10))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=60))
x_dates = [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in stock_data.index.values]

plt.plot(x_dates, stock_data['High'], label='High')
plt.plot(x_dates, stock_data['Low'], label='Low')
plt.xlabel('Time Scale')
plt.ylabel('Scaled USD')
plt.legend()
plt.gcf().autofmt_xdate()
plt.show()
```

```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
```






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

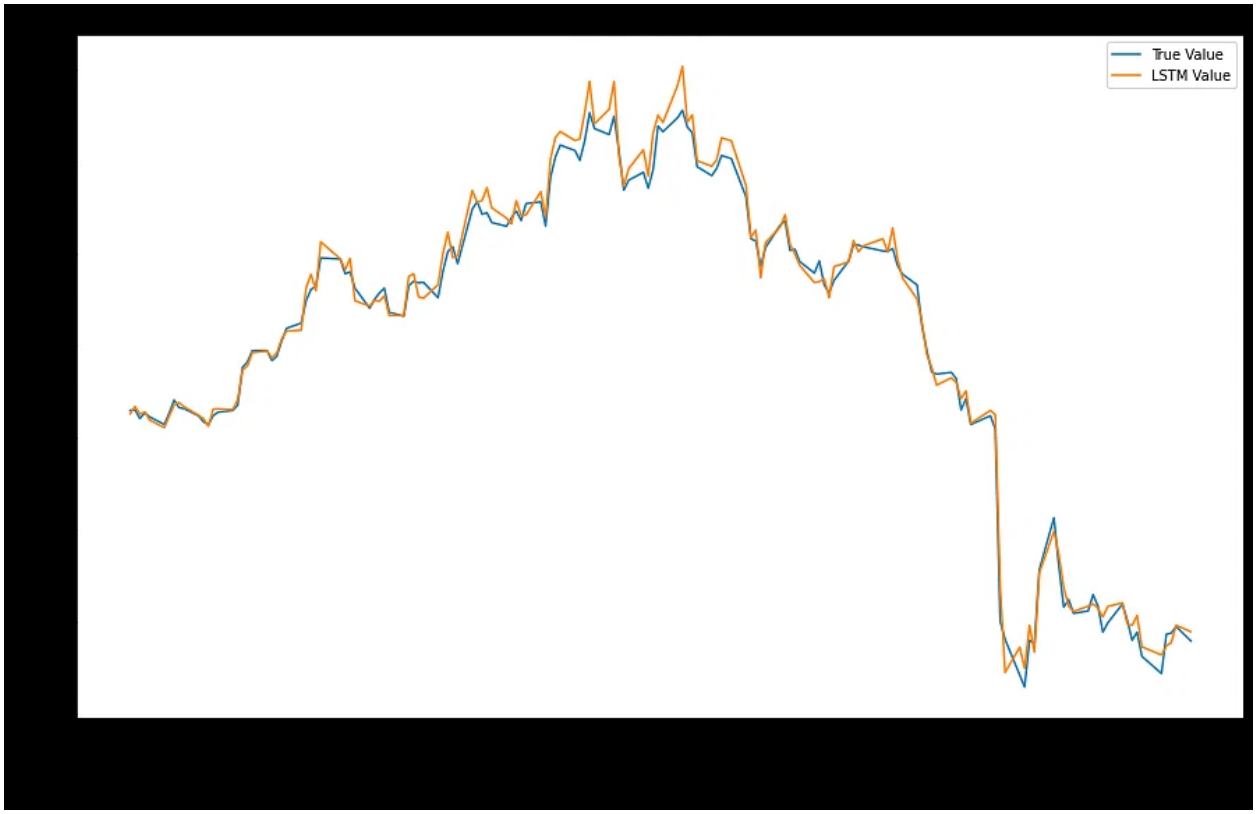
```
#Feature Scaling
sc = StandardScaler()
X_ft = sc.fit_transform(X_feat.values)
X_ft = pd.DataFrame(columns=X_feat.columns,
                    data=X_ft,
                    index=X_feat.index)
```

```python
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)
```

```python
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,)
```

```python
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()
```




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

```
RSME:  0.0710094097230204
MAPE:  0.058775797917331896
```

Hence an accuracy of RMSE 93% was achieved!!