INTERNSHIP

MINOR PROJECT REPORT

ON

STOCK PRICE PREDICTION

Submitted to

CORIZO

Empowering Tomorrow's Leaders

Submitted by –

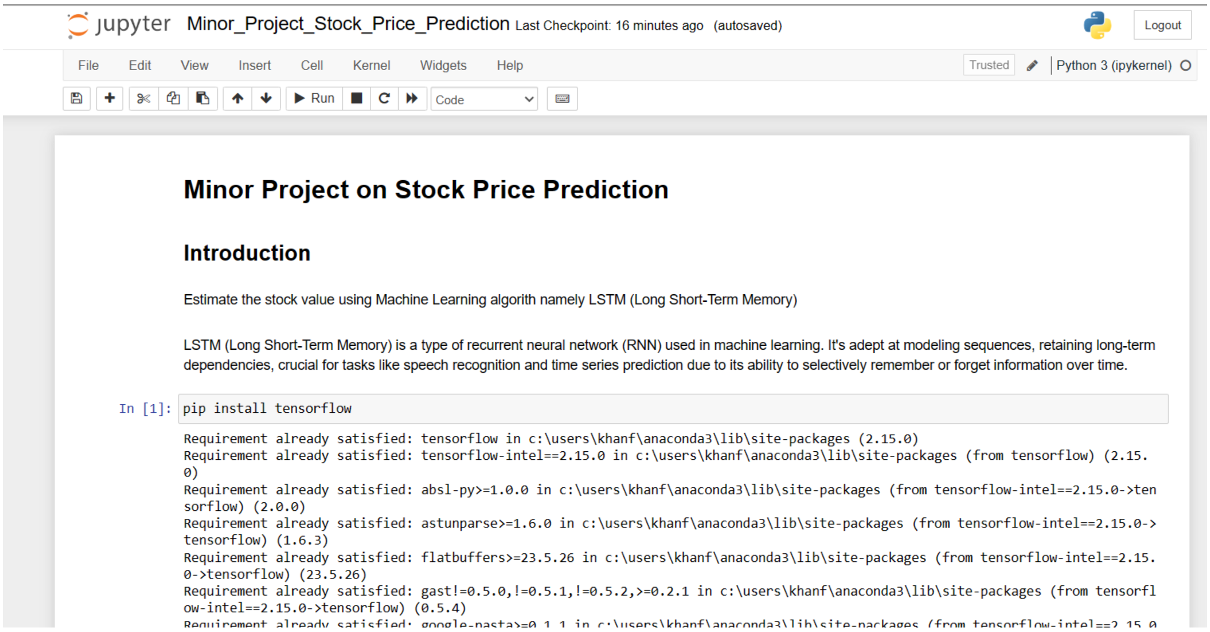
Faraz Khan

Table of Contents

STOCK PRICE PREDICTION

Introduction

The objective of this project was to create a Machine Learning model to predict stock prices using Long Short-Term Memory (LSTM) neural networks.



Executive Summary

The report focuses on utilizing Long Short-Term Memory (LSTM) neural networks for stock price prediction. LSTM, a type of recurrent neural network (RNN), demonstrates its efficacy in capturing temporal dependencies within financial data. By leveraging historical stock prices, volume, and other relevant indicators, the LSTM model forecasts future stock movements. Challenges including market volatility and data limitations are discussed. The study showcases LSTM's potential for predictive accuracy in financial markets, providing valuable insights for investors.

Data Overview

Open - The price the stock opened at.

High - The highest price during the day.

Low - The lowest price during the day.

Close - The closing price on the trading day.

Adj Close - The price of the stock after paying off the dividends.

Volume - How many shares were traded.

STEPS IN DATA ANALYSIS:

Data Collection:

Acquired the stock price dataset from Corizo, the source of the dataset during internship. The dataset encompassed historical stock price data for the desired stock(s) and covered a specific timeframe.

Data Exploration

1. Import Libraries.
2. Conducted preliminary data exploration to understand the dataset.
3. Displayed basic information about the dataset (data types, columns, etc.).
4. Reviewed the first few rows of the dataset to understand its structure.
5. Visualized closing prices over time to identify trends and patterns.
6. Obtained statistical summaries (mean, min, max, etc.) for the dataset.

In [2]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
WARNING:tensorflow:From C:\Users\khanf\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_
cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
```

In [4]:
```
# Load your stock data
data = pd.read_csv('Stock data.csv')
data.head()
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

In [5]:
```
# check shape
data.shape
```

Out[5]: (1009, 7)

In [6]:
```
# Info of dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1009 non-null   object 
 1   Open       1009 non-null   float64
 2   High       1009 non-null   float64
 3   Low        1009 non-null   float64
 4   Close      1009 non-null   float64
 5   Adj Close  1009 non-null   float64
 6   Volume     1009 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 55.3+ KB
```

In [7]:
```
#Description of the dataset
data.describe()
```

Out[7]:

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.009000e+03
mean	419.059673	425.320703	412.374044	419.000733	419.000733	7.570685e+06
std	108.537532	109.262960	107.555867	108.289999	108.289999	5.465535e+06
min	233.919998	250.649994	231.229996	233.880005	233.880005	1.144000e+06
25%	331.489990	336.299988	326.000000	331.619995	331.619995	4.091900e+06
50%	377.769989	383.010010	370.880005	378.670013	378.670013	5.934500e+06
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.322400e+06
max	692.349976	700.989990	686.090027	691.690002	691.690002	5.890430e+07

In [8]:
```
# Sum of null values
data.isnull().sum()
```

Out[8]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [9]:
```
# Unique values
data.nunique()
```

Out[9]:
```
Date         1009
Open          976
High          983
Low           989
Close         988
Adj Close     988
Volume       1005
dtype: int64
```

In [10]:
```
plt.figure(figsize=(15,5))
plt.plot(data['Close'], color="blue")
plt.title('Stock Close Price', fontsize=15)
plt.ylabel('Price in dollars')
plt.show()
```

Data Pre-processing

- Selected the 'Close' column as the target for prediction.
- Normalized the data using Min-Max scaling to bring values within a range of 0 to 1.
- Created sequences of data to feed into the LSTM model.
- Split the dataset into training and testing sets for model validation.

In [11]:
```python
# Splitting the data into training and testing sets

data_train = pd.DataFrame(data['Close'][0:int(len(data)*0.70)])          #70% used as a training data
data_test = pd.DataFrame(data['Close'][int(len(data)*0.70):int(len(data))])   #30% used as a testing data

print(data_train.shape)
print(data_test.shape)
```

```
(706, 1)
(303, 1)
```

In [12]:
```python
# Checking the output of training & testing sets
data_train.head()
```

Out[12]:

	Close
0	254.259995
1	265.720001
2	264.559998
3	250.100006
4	249.470001

In [13]:
```python
data_test.head()
```

Out[13]:

	Close
706	476.619995
707	482.880005
708	485.000000
709	491.359985

In [14]:
```python
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [15]:
```python
data_train_array = scaler.fit_transform(data_train)
data_train_array
```

```
       [0.14383118],
       [0.15430629],
       [0.17063873],
       [0.18622741],
       [0.17035982],
       [0.17370686],
       [0.19893391],
       [0.21628912],
       [0.23358229],
       [0.24102022],
       [0.22902655],
       [0.31667028],
       [0.31189757],
       [0.3062572 ],
       [0.29097836],
       [0.2628382 ],
       [0.22667117],
       [0.22276632],
       [0.24824127],
       [0.24136117],
```

In [16]:
```python
# Chekcking the shape of scaled array
data_train_array.shape
```

Out[16]: (706, 1)

Model Building:

- ✓ Constructed an LSTM neural network.
- ✓ Defined the architecture with multiple LSTM layers followed by a Dense layer.
- ✓ Compiled the model using the Adam optimizer and Mean Squared Error (MSE) loss function.
- ✓ Trained the model on the training data with a specified number of epochs and batch size.

In [17]:
```
# Preparing the training data

x_train = []
y_train = []

for i in range(100,data_train_array.shape[0]):
    x_train.append(data_train_array[i-100:i])
    y_train.append(data_train_array[i,0])

x_train,y_train = np.array(x_train),np.array(y_train)
```

In [18]:
```
# Building model of 4 LSTM network followed by Dropout layout

model = Sequential()

model.add(LSTM(units=50, activation = 'relu', return_sequences = True, input_shape = (x_train.shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=60, activation = 'relu', return_sequences = True))
model.add(Dropout(0.3))

model.add(LSTM(units=80, activation = 'relu', return_sequences = True))
model.add(Dropout(0.4))

model.add(LSTM(units=120, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(units = 1))
```

```
WARNING:tensorflow:From C:\Users\khanf\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is d
eprecated. Please use tf.compat.v1.get_default_graph instead.
```

In [19]:
```
# Checking the summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 60)           26640

 dropout_1 (Dropout)         (None, 100, 60)           0

 lstm_2 (LSTM)               (None, 100, 80)           45120

 dropout_2 (Dropout)         (None, 100, 80)           0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 178761 (698.29 KB)
Trainable params: 178761 (698.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [20]:
```
#Compiling & fitting the model

model.compile(optimizer = 'adam', loss = 'mean_squared_error')
hist = model.fit(X_train,y_train, epochs = 50, batch_size = 32, verbose = 2 )
```

```
Epoch 26/50
19/19 - 4s - loss: 0.0096 - 4s/epoch - 188ms/step
Epoch 27/50
19/19 - 4s - loss: 0.0096 - 4s/epoch - 193ms/step
Epoch 28/50
19/19 - 4s - loss: 0.0098 - 4s/epoch - 192ms/step
Epoch 29/50
19/19 - 4s - loss: 0.0107 - 4s/epoch - 196ms/step
Epoch 30/50
19/19 - 4s - loss: 0.0110 - 4s/epoch - 200ms/step
Epoch 31/50
19/19 - 4s - loss: 0.0101 - 4s/epoch - 200ms/step
Epoch 32/50
19/19 - 4s - loss: 0.0090 - 4s/epoch - 201ms/step
Epoch 33/50
19/19 - 4s - loss: 0.0081 - 4s/epoch - 206ms/step
Epoch 34/50
19/19 - 4s - loss: 0.0085 - 4s/epoch - 205ms/step
Epoch 35/50
```

In [21]:
```
data_test.head()
```

Out[21]:

	Close
706	476.619995
707	482.880005
708	485.000000
709	491.359985
710	490.700012

For prediction, we need testing data and if we look the test data from above table. We can say that we need previous days data for prediction. Hence, for prediction append the 'data_train.tail() to data_test.head()' as mentioned below

In [22]:
```
data_train.tail()
```

Out[22]:

	Close
701	479.100006
702	480.630005
703	481.790009
704	484.670013
705	488.239990

Model Evaluation

- ✓ Generated predictions on the test dataset using the trained model.
- ✓ Transformed the predicted and actual values to their original scale.
- ✓ Calculated the Root Mean Squared Error (RMSE) as an evaluation metric to measure the model's performance.

```
In [23]: # Append testing & training data
         past_100_days = data_train.tail(100)

In [24]: final_data = past_100_days.append(data_test, ignore_index=True)

C:\Users\khanf\AppData\Local\Temp\ipykernel_15468\2758746569.py:1: FutureWarning: The frame.append method is deprecated and wil
l be removed from pandas in a future version. Use pandas.concat instead.
  final_data = past_100_days.append(data_test, ignore_index=True)

In [25]: # Scaling the data

         input_data = scaler.fit_transform(final_data)
         input_data

        [0.43097681],
        [0.4459773 ],
        [0.56938454],
        [0.49941261],
        [0.4975451 ],
        [0.49266545],
        [0.5051056 ],
        [0.40148794],
        [0.42986233],
        [0.39278291],
        [0.39193951],
        [0.35507087],
        [0.36371579],
        [0.40950024],
        [0.38799362],
        [0.3758547 ],
        [0.37983066],
        [0.3891081 ],
        [0.4184463 ],

In [26]: # Checking shape of the input_data
         input_data.shape

Out[26]: (403, 1)

In [27]: # Preparing the testing data
         x_test = []
         y_test = []

         for i in range(100,input_data.shape[0]):
             x_test.append(input_data[i-100:i])
             y_test.append(input_data[i,0])

         x_test,y_test = np.array(x_test), np.array(y_test)
         print(x_test.shape)
         print(y_test.shape)

         (303, 100, 1)
         (303,)

In [28]: # Making Predictions

         y_pred = model.predict(x_test)
         print(y_pred.shape)

         10/10 [==============================] - 1s 58ms/step
         (303, 1)
```

```
In [29]: # Checking y_test
         y_test

Out[29]: array([0.35217924, 0.37103526, 0.37742098, 0.39657814, 0.39459021,
                0.43639862, 0.43278411, 0.41513293, 0.41751255, 0.47013471,
                0.46073667, 0.4033254 , 0.42588629, 0.43230216, 0.49013517,
                0.48218326, 0.49739453, 0.52170251, 0.52637129, 0.50968392,
                0.50492488, 0.46621878, 0.46468256, 0.48019515, 0.51558778,
                0.49667165, 0.54528743, 0.49146052, 0.48525552, 0.42407899,
                0.44938103, 0.45392929, 0.41989216, 0.40528327, 0.44606766,
                0.42519346, 0.41651858, 0.42793452, 0.68267123, 0.66309233,
                0.61890412, 0.59363241, 0.60914481, 0.49272575, 0.53887156,
                0.52016629, 0.54019691, 0.5676676 , 0.54143199, 0.57971616,
                0.57558954, 0.56694472, 0.60053014, 0.61414507, 0.59607223,
                0.59284922, 0.59513848, 0.57724637, 0.56784832, 0.54375121,
                0.52435321, 0.56161335, 0.58348133, 0.56326999, 0.5396246 ,
                0.57513783, 0.56664358, 0.48495438, 0.45661014, 0.47197207,
                0.40251206, 0.44200125, 0.43627821, 0.49206299, 0.47688187,
```

```
In [30]: # Checking y_pred
         y_pred
```

```
                [0.58036655],
                [0.57724136],
                [0.5756183 ],
                [0.57621723],
                [0.57949984],
                [0.5848959 ],
                [0.5913933 ],
                [0.598042  ],
                [0.60372335],
                [0.6075498 ],
                [0.6086911 ],
                [0.6066068 ],
                [0.60227436],
                [0.59741   ],
                [0.59300554],
                [0.58910984],
                [0.58634335],
                [0.58492374],
                [0.58307993],
                [0.5789026 ],
```

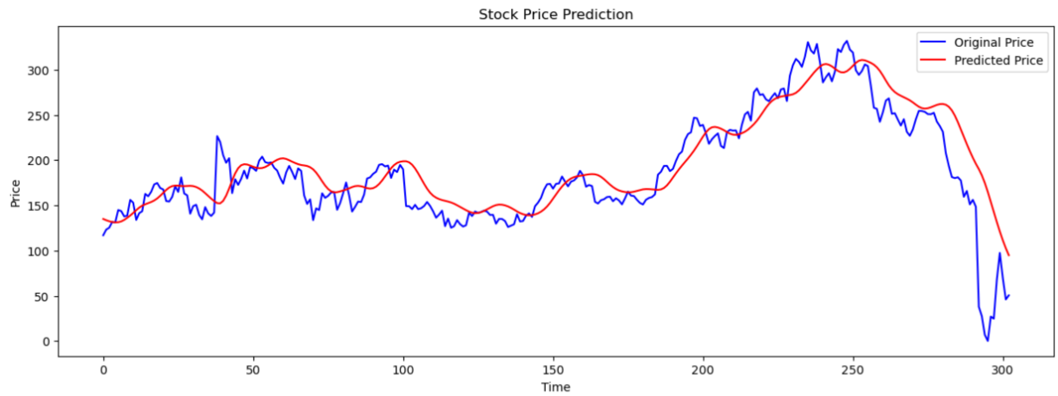
From above y_test & y_pred, we can't recognize how they are matching. hence, for that we need to scale the data.

```
In [31]: # Scaling the data
         scaler.scale_

Out[31]: array([0.00301214])
```

```
In [32]: scale_factor = 1/0.00301214
         y_pred = y_pred * scale_factor
         y_test = y_test * scale_factor
```

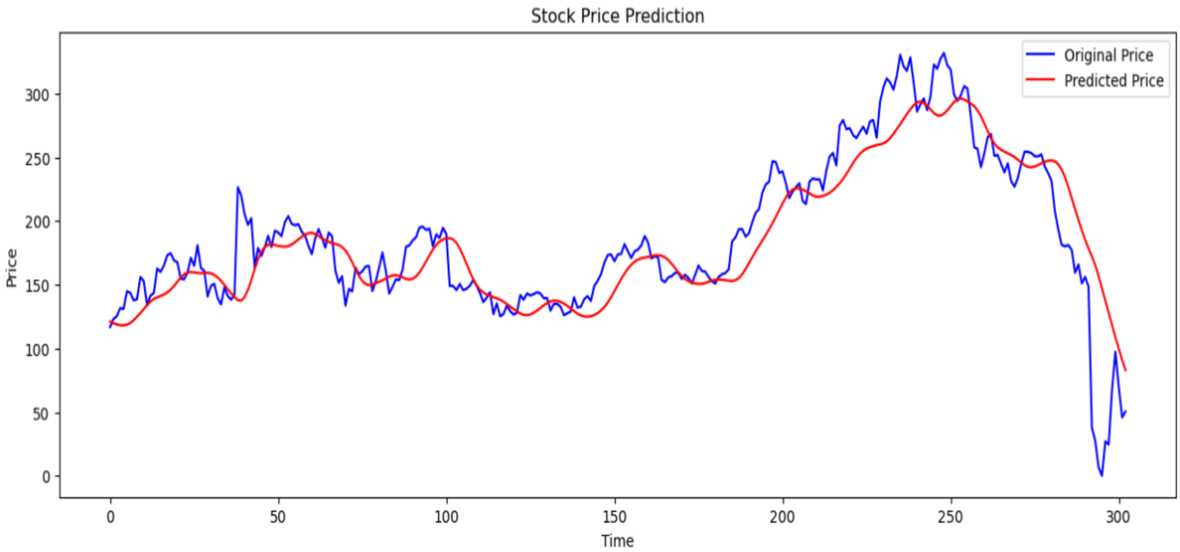
```
In [33]: # Plotting graph for the result
         plt.figure(figsize = (15,5))
         plt.plot(y_test,'b',label = 'Original Price')
         plt.plot(y_pred,'r',label = 'Predicted Price')
         plt.title('Stock Price Prediction')
         plt.xlabel('Time')
         plt.ylabel('Price')
         plt.legend()
         plt.show()
```



Data Visualization

Visualized the predicted stock prices against the actual prices to visualize the model's accuracy.

In [33]:
```
# Plotting graph for the result
plt.figure(figsize = (15,5))
plt.plot(y_test,'b',label = 'Original Price')
plt.plot(y_pred,'r',label = 'Predicted Price')
plt.title('Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Conclusion

- The LSTM model demonstrated the ability to predict stock prices based on historical data.
- The visualization of actual vs. predicted prices provided an intuitive understanding of the model's performance.
- Above graph shows the relation between Actual price (Blue Line) and Predicted price (Red Line) of stock for the mentioned dataset.