Minor Project - Stock Price Prediction

Introduction :

Utilize Machine Learning techniques to estimate the stock value using the Long Short-term Memory(LSTM) Networks.italicized text

```
# Importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout
```

```
# Importing data
df = pd.read_excel('/content/drive/MyDrive/1729258-1613615-Stock_Price_data_set_(1).xlsx')
df.head()
```

	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

```
# Check shape of the dataset
df.shape
```

```
(1009, 7)
```

```
# Info of the dataset
df.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   datetime64[ns]
 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: datetime64[ns](1), float64(5), int64(1)
memory usage: 55.3 KB
```

```
# Description of the dataset
df.describe()
```

```
# Sum of null values
df.isnull().sum()
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

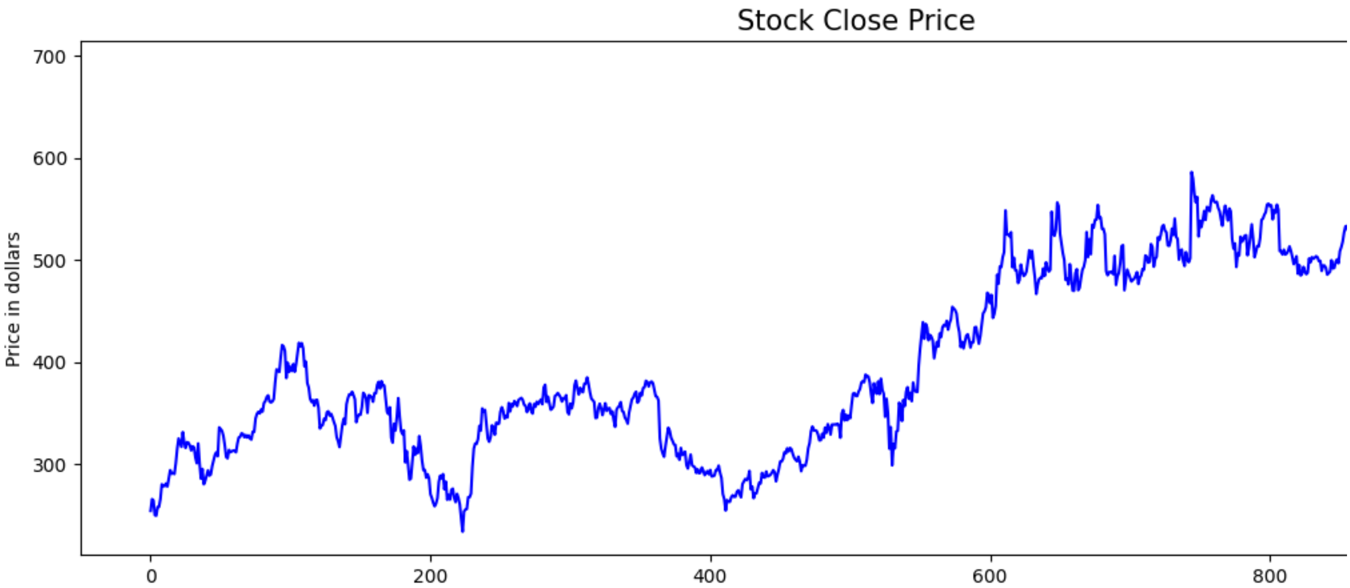
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.322400e+06

```
# Looking for the unique values
df.nunique()
```

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

Exploratory Data Analysis

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




```
# Splitting the data into training and testing sets

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

print(df_train.shape)
print(df_test.shape)
```

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

```
# Checking the output of training & testing sets
df_train.head()
```

	Close

```
df_test.head()
```

	Close
706	476.619995
707	482.880005
708	485.000000
709	491.359985
710	490.700012

```
# Scaling the data
scaler = MinMaxScaler(feature_range=(0,1))
```

```
df_train_array = scaler.fit_transform(df_train)
df_train_array
```

```
       [1.        ],
       [0.98850231],
       [0.90454647],
       [0.87448476],
       [0.84649951],
       [0.82533242],
       [0.76483721],
       [0.76905199],
       [0.75116998],
       [0.81231598],
       [0.77472338],
       [0.73238919],
       [0.73164533],
       [0.78553945],
       [0.79737819],
       [0.73365975],
       [0.74131464],
       [0.77168628],
       [0.79576658],
       [0.8045372 ],
       [0.82483655],
       [0.91000099],
       [0.83422694],
       [0.88874092],
       [0.84293552],
       [0.93215974],
       [0.92326522],
       [0.94697373],
       [0.9481204 ],
       [0.99237623],
       [0.95320303],
       [0.95472158],
       [0.92016608],
       [0.91994912],
       [0.9035237 ],
       [0.79080794],
       [0.77896929],
       [0.78842163],
       [0.78829764],
       [0.79043605],
       [0.78209935],
       [0.83779093],
       [0.74955837],
       [0.77552919],
       [0.78513655],
       [0.81529123],
       [0.86738779],
       [0.87039387],
       [0.73331889],
       [0.7635045 ],
       [0.79610754],
       [0.7837419 ],
       [0.77156229],
       [0.75997153],
       [0.76471321],
       [0.76830823],
       [0.77723377],
       [0.78829764]])
```

```
# Chekcking the shape of scaled array
df_train_array.shape
```

```
(706, 1)
```

```
# Preparing the training data

X_train = []
y_train = []

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

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

```
# 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))
```

```
# 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)
_________________________________________________________________
```

```
# 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 )
```

```
19/19 - 4s - loss: 0.0079 - 4s/epoch - 233ms/step
Epoch 32/50
19/19 - 6s - loss: 0.0065 - 6s/epoch - 337ms/step
Epoch 33/50
19/19 - 5s - loss: 0.0069 - 5s/epoch - 259ms/step
Epoch 34/50
19/19 - 5s - loss: 0.0075 - 5s/epoch - 249ms/step
Epoch 35/50
19/19 - 6s - loss: 0.0068 - 6s/epoch - 328ms/step
Epoch 36/50
19/19 - 4s - loss: 0.0065 - 4s/epoch - 235ms/step
Epoch 37/50
19/19 - 6s - loss: 0.0066 - 6s/epoch - 293ms/step
Epoch 38/50
19/19 - 5s - loss: 0.0070 - 5s/epoch - 273ms/step
Epoch 39/50
19/19 - 4s - loss: 0.0064 - 4s/epoch - 232ms/step
Epoch 40/50
19/19 - 6s - loss: 0.0073 - 6s/epoch - 337ms/step
Epoch 41/50
19/19 - 4s - loss: 0.0070 - 4s/epoch - 232ms/step
Epoch 42/50
19/19 - 4s - loss: 0.0063 - 4s/epoch - 235ms/step
Epoch 43/50
19/19 - 6s - loss: 0.0055 - 6s/epoch - 337ms/step
Epoch 44/50
19/19 - 4s - loss: 0.0064 - 4s/epoch - 237ms/step
Epoch 45/50
19/19 - 5s - loss: 0.0072 - 5s/epoch - 242ms/step
Epoch 46/50
19/19 - 6s - loss: 0.0063 - 6s/epoch - 327ms/step
Epoch 47/50
19/19 - 4s - loss: 0.0062 - 4s/epoch - 233ms/step
Epoch 48/50
19/19 - 6s - loss: 0.0065 - 6s/epoch - 299ms/step
Epoch 49/50
19/19 - 5s - loss: 0.0061 - 5s/epoch - 273ms/step
Epoch 50/50
19/19 - 4s - loss: 0.0056 - 4s/epoch - 233ms/step
```

```
df_test.head()
```

	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 'df_train.tail() to df_test.head()' as mentioned below:

```
df_train.tail()
```

	Close
701	479.100006
702	480.630005
703	481.790009
704	484.670013
705	488.239990

```
# Append testing & training data
past_100_days = df_train.tail(100)
```

```
final_df = past_100_days.append(df_test, ignore_index=True)
```

```
# Scaling the data

input_data = scaler.fit_transform(final_df)
input_data
```

```
       [1.        ],
       [0.97087267],
       [0.96117349],
       [0.90213563],
       [0.8866532 ],
       [0.89939448],
       [0.92153382],
       [0.916112  ],
       [0.85002566],
       [0.77734274],
       [0.77342681],
       [0.73023284],
       [0.76204102],
       [0.80086754],
       [0.80839788],
       [0.7569505 ],
       [0.75893843],
       [0.73755232],
       [0.71776254],
       [0.73899808],
       [0.69688844],
       [0.68384582],
       [0.70496095],
       [0.73863664],
       [0.7667098 ],
       [0.76625809],
       [0.76333622],
       [0.75607704],
       [0.75556485],
       [0.76023381],
       [0.73116659],
       [0.71589503],
       [0.69715961],
       [0.62598275],
       [0.58311989],
       [0.54628149],
       [0.54263673],
       [0.54561891],
       [0.53471479],
       [0.48043617],
       [0.49998492],
       [0.45513413],
       [0.47037555],
       [0.44745321],
       [0.11385882],
       [0.08268316],
       [0.02024158],
       [0.        ],
       [0.08132775],
       [0.07427927],
       [0.20313866],
       [0.29347268],
       [0.21018706],
       [0.13825716],
       [0.15202266]])
```

```
# Checking shape of the input_data
input_data.shape
```

```
(403, 1)
```

```
# 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,)
```

```
# Making Predictions

y_pred = model.predict(X_test)
print(y_pred.shape)
```

```
10/10 [==============================] - 2s 123ms/step
(303, 1)
```

```
# Checking y_test
y_test
```

```
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,
       0.48359888, 0.49498485, 0.49621975, 0.43703124, 0.4592909 ,
       0.49221355, 0.52829911, 0.48528567, 0.43121774, 0.44685075,
       0.46462244, 0.46293565, 0.48784592, 0.54134154, 0.54510671,
       0.5567337 , 0.56414345, 0.58700567, 0.58920447, 0.58158385,
       0.58444524, 0.54314893, 0.57086046, 0.56278795, 0.58658392,
       0.57191482, 0.44941109, 0.44904964, 0.4393204 , 0.45362806,
       0.4393204 , 0.44224218, 0.44971232, 0.46317649, 0.45004361,
       0.43218165, 0.41079544, 0.42124757, 0.43416967, 0.3825115 ,
       0.4077833 , 0.37736077, 0.38242114, 0.40263257, 0.38928882,
       0.38127652, 0.38555379, 0.42763338, 0.4162475 , 0.43133825,
       0.42663932, 0.42971167, 0.43422988, 0.43106717, 0.41983186,
       0.42031381, 0.39076474, 0.40675919, 0.40651826, 0.39968073,
       0.37986081, 0.3842585 , 0.38877671, 0.42227178, 0.39820472,
       0.39974094, 0.4176029 , 0.42492238, 0.41356665, 0.44917015,
       0.46097769, 0.47700229, 0.50414169, 0.52209411, 0.52350972,
       0.50757557, 0.52363014, 0.52495549, 0.54802858, 0.53091965,
       0.51525649, 0.53097977, 0.53498597, 0.54513686, 0.56703517,
       0.55197447, 0.51390099, 0.519835  , 0.51612995, 0.46365854,
       0.45805591, 0.46902005, 0.47227321, 0.47956253, 0.48073731,
       0.46552605, 0.47552637, 0.46823705, 0.45519443, 0.47486361,
       0.49757525, 0.48450249, 0.4827554 , 0.47031543, 0.45995366,
       0.4548932 , 0.47055627, 0.47658055, 0.47956253, 0.48847853,
       0.55426373, 0.56381216, 0.58324049, 0.58348133, 0.56592069,
       0.57357146, 0.60007825, 0.62194641, 0.63101297, 0.66980934,
       0.68932794, 0.6952921 , 0.74402849, 0.74204037, 0.71640704,
       0.71996134, 0.69155689, 0.65682703, 0.67221901, 0.68315309,
       0.69173762, 0.64980869, 0.64291096, 0.69565354, 0.70351518,
       0.70089464, 0.70164767, 0.67517085, 0.72098555, 0.75496257,
       0.76342667, 0.73390756, 0.82866952, 0.84159153, 0.81975352,
       0.82219335, 0.80526514, 0.79893972, 0.81345826, 0.82562723,
       0.80903031, 0.8381878 , 0.84129039, 0.79954219, 0.8839422 ,
       0.91894342, 0.93966677, 0.93020879, 0.9133407 , 0.94686583,
       0.99584324, 0.96831224, 0.95792032, 0.98975866, 0.92984735,
       0.86153188, 0.87879156, 0.8924666 , 0.86511633, 0.89725598,
       0.97264973, 0.96277001, 0.98707799, 1.        , 0.97087267,
       0.96117349, 0.90213563, 0.8866532 , 0.89939448, 0.92153382,
       0.916112  , 0.85002566, 0.77734274, 0.77342681, 0.73023284,
       0.76204102, 0.80086754, 0.80839788, 0.7569505 , 0.75893843,
       0.73755232, 0.71776254, 0.73899808, 0.69688844, 0.68384582,
       0.70496095, 0.73863664, 0.7667098 , 0.76625809, 0.76333622,
       0.75607704, 0.75556485, 0.76023381, 0.73116659, 0.71589503,
       0.69715961, 0.62598275, 0.58311989, 0.54628149, 0.54263673,
       0.54561891, 0.53471479, 0.48043617, 0.49998492, 0.45513413,
```

```
# Checking y_pred
y_pred
```

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

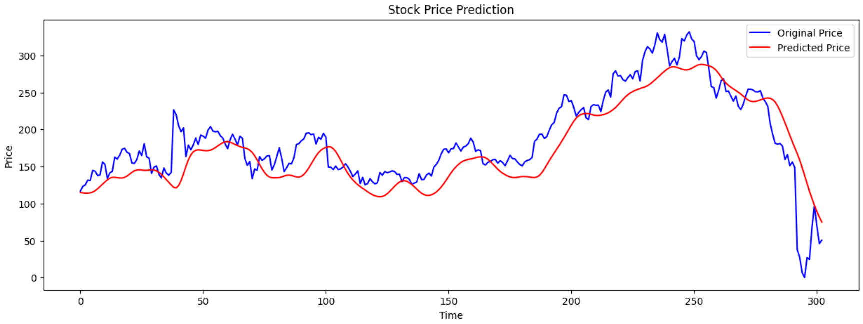
```
# Scaling the data
scaler.scale_
```

```
array([0.00301214])
```

```
scale_factor = 1/0.00301214
y_pred = y_pred * scale_factor
y_test = y_test * scale_factor
```

```
# 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 :

Above graph shows the relation between Actual price(Blue Line) and Predicted price(Red Line) of stock for the mentioned dataset.*italicized text*