# Assignment 3 - Problem 2

**Note:**This notebook serves as our report. Suitable comments and explanations with visualizations were prodcued where and when needed.

**Note:**
We have provided two files (Train_RNN.py and Test_RNN.py).

The trained model is already saved in the models folder.

In the Data folder, we have the two csv files for train and test (train_data_RNN.csv, test_data_RNN.csv) along with final.csv file (the new dataset with 12 features and 1 target value).

We have two pickle files which have the Minmax object saved(used in the Test_RNN.py file for scaling the test data).

---

## Recurrent Neural Networks for Regression

---

- Designing RNN for stock price prediction.
- Predict the next day open price using the past 3 days Open, High, and Low Prices, and Volume.

In [1]:
```
#Importing necessary libraries
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from subprocess import check_output
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM,GRU,SimpleRNN
from keras.models import Sequential
from sklearn.model_selection import  train_test_split
import time
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
from numpy import newaxis
import math
from sklearn.metrics import mean_squared_error
```

In [2]:
```
tf.random.set_seed(221)
np.random.seed(221)
os.environ['PYTHONHASHSEED']=str(221)
```

### Dataset Creation

In [3]:
```
# Reading and storing the "q2_dataset.py" given
stocks_dataset =  pd.read_csv('q2_dataset.csv', header=0,parse_dates=['Date'])
stocks_dataset.columns = ['Date', 'close','volume','open','high','low']
stocks_dataset['close'] = stocks_dataset['close'].replace('[\$,]', '', regex=True).astype(float)
stocks_dataset=stocks_dataset.sort_values(by='Date',ascending=False)
stocks_dataset
```

Out[3]:

| | Date | close | volume | open | high | low |
|---|---|---|---|---|---|---|
| **0** | 2020-07-08 | 381.37 | 29272970 | 376.72 | 381.50 | 376.36 |
| **1** | 2020-07-07 | 372.69 | 28106110 | 375.41 | 378.62 | 372.23 |
| **2** | 2020-07-06 | 373.85 | 29663910 | 370.00 | 375.78 | 369.87 |
| **3** | 2020-07-02 | 364.11 | 28510370 | 367.85 | 370.47 | 363.64 |
| **4** | 2020-07-01 | 364.11 | 27684310 | 365.12 | 367.36 | 363.91 |
| **...** | ... | ... | ... | ... | ... | ... |
| **1254** | 2015-07-15 | 126.82 | 33559770 | 125.72 | 127.15 | 125.58 |
| **1255** | 2015-07-14 | 125.61 | 31695870 | 126.04 | 126.37 | 125.04 |
| **1256** | 2015-07-13 | 125.66 | 41365600 | 125.03 | 125.76 | 124.32 |
| **1257** | 2015-07-10 | 123.28 | 61292800 | 121.94 | 123.85 | 121.21 |
| **1258** | 2015-07-09 | 120.07 | 78291510 | 123.85 | 124.06 | 119.22 |

1259 rows × 6 columns

- Creating a new data frame with past 3 days Open, High, and Low prices, and Volume
- Included next day's opening price as Target

In [4]:
```
df = pd.DataFrame(columns=['open1', 'high1', 'low1', 'volume1',
                           'open2', 'high2', 'low2', 'volume2',
                           'open3', 'high3', 'low3', 'volume3','target'])
```

```
l= len(stocks_dataset.index)
i=0
while i < l-3:

    df.loc[i, 'open1'] = stocks_dataset["open"][i+3]
    df.loc[i, 'high1'] = stocks_dataset["high"][i+3]
    df.loc[i, 'low1'] = stocks_dataset["low"][i+3]
    df.loc[i, 'volume1'] = stocks_dataset["volume"][i+3]

    df.loc[i, 'open2'] = stocks_dataset["open"][i+2]
    df.loc[i, 'high2'] = stocks_dataset["high"][i+2]
    df.loc[i, 'low2'] = stocks_dataset["low"][i+2]
    df.loc[i, 'volume2'] = stocks_dataset["volume"][i+2]

    df.loc[i, 'open3'] = stocks_dataset["open"][i+1]
    df.loc[i, 'high3'] = stocks_dataset["high"][i+1]
    df.loc[i, 'low3'] = stocks_dataset["low"][i+1]
    df.loc[i, 'volume3'] = stocks_dataset["volume"][i+1]

    df.loc[i, 'target'] = stocks_dataset["open"][i]

    i =i+1

df.to_csv('final.csv', mode='w+', header=True, index=False)
final_dataset =  pd.read_csv('final.csv', header=0)
final_dataset
```

Out[4]:

| | open1 | high1 | low1 | volume1 | open2 | high2 | low2 | volume2 | open3 | high3 | low3 | volume3 | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 367.85 | 370.47 | 363.64 | 28510370 | 370.00 | 375.78 | 369.87 | 29663910 | 375.41 | 378.62 | 372.23 | 28106110 | 376.72 |
| **1** | 365.12 | 367.36 | 363.91 | 27684310 | 367.85 | 370.47 | 363.64 | 28510370 | 370.00 | 375.78 | 369.87 | 29663910 | 375.41 |
| **2** | 360.08 | 365.98 | 360.00 | 35055820 | 365.12 | 367.36 | 363.91 | 27684310 | 367.85 | 370.47 | 363.64 | 28510370 | 370.00 |
| **3** | 353.25 | 362.17 | 351.28 | 32661520 | 360.08 | 365.98 | 360.00 | 35055820 | 365.12 | 367.36 | 363.91 | 27684310 | 367.85 |
| **4** | 364.41 | 365.32 | 353.02 | 51314210 | 353.25 | 362.17 | 351.28 | 32661520 | 360.08 | 365.98 | 360.00 | 35055820 | 365.12 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1251** | 125.72 | 127.15 | 125.58 | 33559770 | 127.74 | 128.57 | 127.35 | 35987630 | 129.08 | 129.62 | 128.31 | 45970470 | 130.97 |
| **1252** | 126.04 | 126.37 | 125.04 | 31695870 | 125.72 | 127.15 | 125.58 | 33559770 | 127.74 | 128.57 | 127.35 | 35987630 | 129.08 |
| **1253** | 125.03 | 125.76 | 124.32 | 41365600 | 126.04 | 126.37 | 125.04 | 31695870 | 125.72 | 127.15 | 125.58 | 33559770 | 127.74 |
| **1254** | 121.94 | 123.85 | 121.21 | 61292800 | 125.03 | 125.76 | 124.32 | 41365600 | 126.04 | 126.37 | 125.04 | 31695870 | 125.72 |
| **1255** | 123.85 | 124.06 | 119.22 | 78291510 | 121.94 | 123.85 | 121.21 | 61292800 | 125.03 | 125.76 | 124.32 | 41365600 | 126.04 |

1256 rows × 13 columns

- Splitting the data randomly into 70% for training and 30% for testing
- Saving the training and testing data as ‘train_data_RNN.csv’ and ‘test_data_RNN.csv’ respectively

In [5]:
```
train,test = train_test_split(final_dataset, test_size=0.30, random_state=42)
train.to_csv('train_data_RNN.csv', index=False, mode='w+')
test.to_csv('test_data_RNN.csv', index=False, mode='w+')
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

## Data Exploartion

- Exploring the types, dimensions/shape of the data
- Plotting the Target opening price in order to observe the trend of the data
- Plotting the Volume in order to observe if there are any irregularities in the data

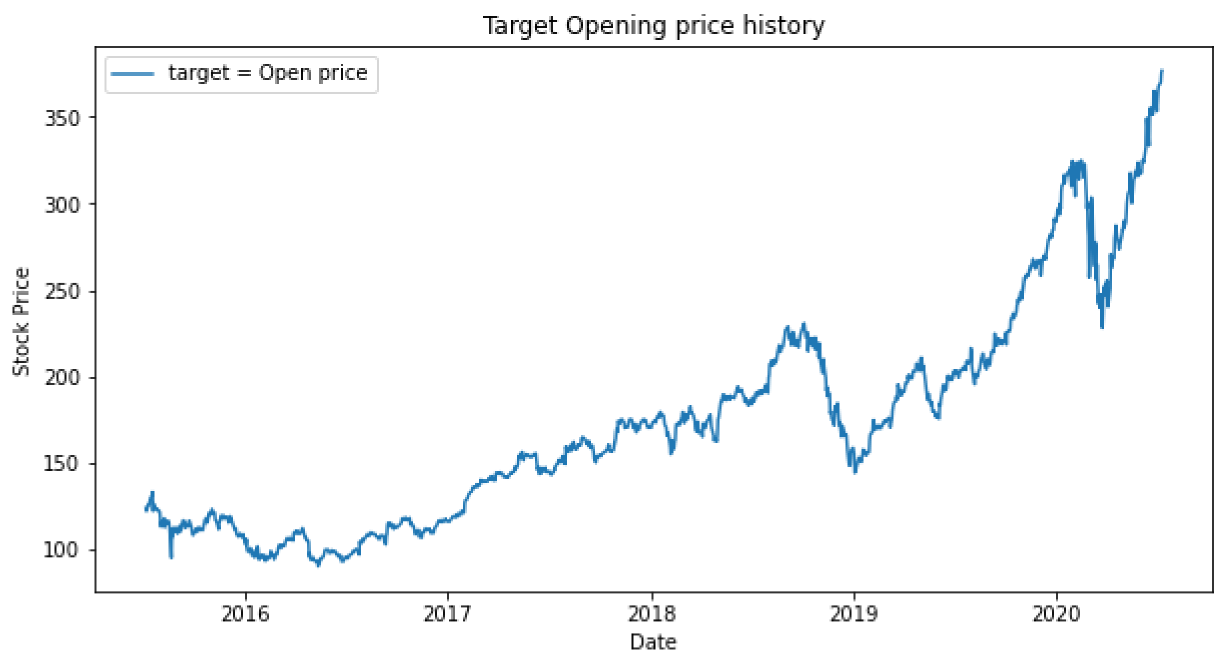
In [6]:
```
train.shape, test.shape
```

Out[6]:
```
((879, 13), (377, 13))
```

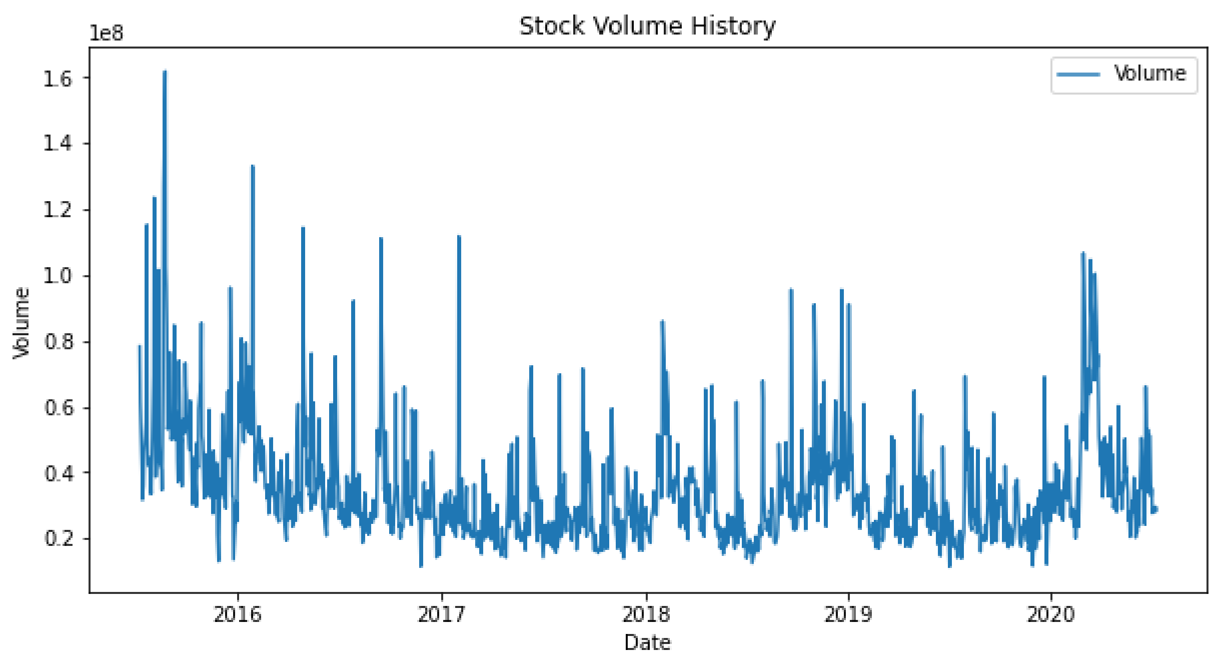
In [7]:
```
final_dataset.dtypes
```

Out[7]:
```
open1       float64
high1       float64
low1        float64
volume1       int64
open2       float64
high2       float64
low2        float64
volume2       int64
open3       float64
high3       float64
low3        float64
volume3       int64
target      float64
dtype: object
```

In [8]:
```
#Target pirce history plot
plt.figure(figsize=(10, 5))
plt.plot(stocks_dataset['Date'].values[::-1],stocks_dataset["open"].values[::-1])  #Reversing the plot as the given data has lates
plt.title('Target Opening price history')
plt.ylabel('Stock Price')
plt.xlabel('Date')
plt.legend(['target = Open price'], loc='upper left')
plt.show()
```



In [9]:
```
#Volume plot
plt.figure(figsize=(10, 5))
plt.plot(stocks_dataset['Date'].values[::-1],stocks_dataset["volume"].values[::-1]) #Reversing the plot as the given data has late
plt.title('Stock Volume History')
plt.ylabel('Volume')
plt.xlabel('Date')
plt.legend(['Volume'], loc='upper right')
plt.show()
```



## Data preprocessing

- Checking if there are any Null data in the given dataset

In [10]:
```
final_dataset.isna().sum()
```

Out[10]:
```
open1      0
high1      0
low1       0
volume1    0
open2      0
high2      0
low2       0
volume2    0
open3      0
high3      0
low3       0
volume3    0
target     0
dtype: int64
```

- Normalization of Training Data

In [11]:
```
#MinMaxScaler
train_dataset =  pd.read_csv('train_data_RNN.csv')
y_train=train_dataset[['target']]
x_train=train_dataset.drop(columns=['target'])
```

In [12]:
```
y_train.shape
```

Out[12]:
```
(879, 1)
```

In [13]:
```
#MinMaxObject = MinMaxScaler(feature_range=(0,1))
MinMaxObject_x = MinMaxScaler(feature_range=(0,1))
MinMaxObject_y = MinMaxScaler(feature_range=(0,1))
```

```
norm_x_train = MinMaxObject_x.fit_transform(x_train)
norm_y_train = MinMaxObject_y.fit_transform(y_train)
```

In [14]:
```
# t_x_train=norm_x_train
# t_x_train=t_x_train.reshape(t_x_train.shape[0],t_x_train.shape[1] , 1)
# t_x_train.shape
```

In [15]:
```
# Splitting each normalized data sample into 3 arrays (Timestamps: t, t-1, and t-2)
# This will convert the data as 3 timestamps data with 4 features in each timestamp
dummy_x_train=[]
for i in range(len(norm_x_train)):
    z=np.split(norm_x_train[i], 3)
    dummy_x_train.append(np.array(z))
new_x_train=np.array(dummy_x_train)
new_x_train.shape
```

Out[15]:
```
(879, 3, 4)
```

## Model Implementation - LSTM

There exists exploding gradient problem and vanishing gradient problem in a neyral network with a back propagation. In a Recurrent Neural Network these probelms become even worse as we are not just finding gradients with different layers but also through time. A special type of RNN, LSTM - Long Short Term Memory overcomes these issues with their cell states, also these have additional units(memory cells) which help in storing memory for longer periods. Hence LSTM has been considered for the implementation.

**- LSTM Model 1**

LSTM Model 1 is designed with the following parameters:

Units/Neurons = 200, Layers = 2, Epochs = 100, Batch size = 32

Optimizer = Adam, loss = mean_squared_error

In [16]:
```
model_lstm_1 = Sequential()
#adding 1st lstm layer
model_lstm_1.add(LSTM(units = 200, return_sequences = True, input_shape = (new_x_train.shape[1], 4)))
model_lstm_1.add(Dropout(rate = 0.2))

##adding 2nd lstm layer: 200 neurons
model_lstm_1.add(LSTM(units = 200, return_sequences = False))
model_lstm_1.add(Dropout(rate = 0.2))
'''
#adding 3rd lstm layer
model_lstm_1.add(LSTM(units = 200, return_sequences = False))
model_lstm_1.add(Dropout(rate = 0.2))
'''
##adding output layer
model_lstm_1.add(Dense(units = 1))

model_lstm_1.compile(optimizer='adam', loss='mean_squared_error')

model_lstm_1.fit(new_x_train, norm_y_train, epochs=100, batch_size=32, verbose=0)
```

Out[16]:
```
<keras.callbacks.History at 0x1fe649c2b80>
```

In [17]:
```
train_predict_1 = model_lstm_1.predict(new_x_train)
train_predict_1 = MinMaxObject_y.inverse_transform(train_predict_1)
loss_train = mean_squared_error(y_train,train_predict_1)
print("Loss on the training data of LSTM Model 1 (MSE) = " +str(loss_train))
```

```
Loss on the training data of LSTM Model 1 (MSE) = 13.865860269927666
```

**- LSTM Model 2**

LSTM Model 2 is designed with the following parameters:

Units/Neurons = 200, Layers = 2, Epochs = 500, Batch size = 64

Optimizer = Adam, loss = mean_squared_error

In [18]:
```
model_lstm_2 = Sequential()
#adding 1st lstm layer
model_lstm_2.add(LSTM(units = 200, return_sequences = True, input_shape = (new_x_train.shape[1], 4)))
model_lstm_2.add(Dropout(rate = 0.2))

##adding 2nd lstm layer: 200 neurons
model_lstm_2.add(LSTM(units = 200, return_sequences = False))
model_lstm_2.add(Dropout(rate = 0.2))
'''
#adding 3rd lstm layer
model_lstm_2.add(LSTM(units = 200, return_sequences = False))
model_lstm_2.add(Dropout(rate = 0.2))
'''
##adding output layer
model_lstm_2.add(Dense(units = 1))

model_lstm_2.compile(optimizer='adam', loss='mean_squared_error')

model_lstm_2.fit(new_x_train, norm_y_train, epochs=500, batch_size=64, verbose=0)
```

Out[18]:
```
<keras.callbacks.History at 0x1fe6bd26e80>
```

In [19]:
```
# Prediction
train_predict_2 = model_lstm_2.predict(new_x_train)
```

```
train_predict_2 = MinMaxObject_y.inverse_transform(train_predict_2)
#Loss Calculation on training data
loss_train = mean_squared_error(y_train,train_predict_2)
print("Loss on the training data of LSTM Model 2 (MSE) = " +str(loss_train))
```

```
Loss on the training data of LSTM Model 2 (MSE) = 10.151389068409598
```

- **LSTM Model 3**

LSTM Model 3 is designed with the following parameters:

Units/Neurons = 200, Layers = 2, Epochs = 1000, Batch size = 64

Optimizer = Adam, loss = mean_squared_error

In [20]:
```
model_lstm_3 = Sequential()
#adding 1st lstm layer
#model_lstm_3.add(LSTM(units = 200, return_sequences = True, input_shape = (new_x_train.shape[1], 4)))
model_lstm_3.add(LSTM(units = 200, return_sequences = True, input_shape = (new_x_train.shape[1], 4)))
model_lstm_3.add(Dropout(rate = 0.2))

##adding 2nd lstm layer: 200 neurons
model_lstm_3.add(LSTM(units = 200, return_sequences = False))
model_lstm_3.add(Dropout(rate = 0.2))
'''
#adding 3rd lstm layer
model_lstm_3.add(LSTM(units = 200, return_sequences = False))
model_lstm_3.add(Dropout(rate = 0.2))
'''
##adding output layer
model_lstm_3.add(Dense(units = 1))
model_lstm_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
lstm_4 (LSTM)               (None, 3, 200)            164000

dropout_4 (Dropout)         (None, 3, 200)            0

lstm_5 (LSTM)               (None, 200)               320800

dropout_5 (Dropout)         (None, 200)               0

dense_2 (Dense)             (None, 1)                 201

=================================================================
Total params: 485,001
Trainable params: 485,001
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
model_lstm_3.compile(optimizer='adam', loss='mean_squared_error')

model_lstm_3.fit(new_x_train, norm_y_train, epochs=1000, batch_size=64, verbose=0)
```

Out[21]:
```
<keras.callbacks.History at 0x1fe0b450070>
```

In [22]:
```
train_predict = model_lstm_3.predict(new_x_train)
train_predict = MinMaxObject_y.inverse_transform(train_predict)
loss_train = mean_squared_error(y_train,train_predict)
print("Loss on the training data of LSTM Model 3 (MSE)= " +str(loss_train))
```

```
Loss on the training data of LSTM Model 3 (MSE)= 8.615814735999862
```

Out of three implementations with different parameters, mean squared error loss was less for the third LSTM Model, hence finalized the hyperparameters as - Units/Neurons = 200, Layers = 2, Epochs = 1000, and Batch size = 64

## Saving the trained model

In [23]:
```
from tensorflow.keras.models import load_model

model_lstm_3.save('model\Group_55_RNN_model.hdf5')
```

- **Normalization of Test Data**

In [24]:
```
# Reading the "test_data_RNN.csv" which is created above
test_dataset =  pd.read_csv('test_data_RNN.csv')
y_test=test_dataset[['target']]
x_test=test_dataset.drop(columns=['target'])
y_test.shape

norm_x_test = MinMaxObject_x.transform(x_test) #MinMaxScaler
norm_y_test = MinMaxObject_y.transform(y_test)

test_x_test=norm_x_test
test_x_test=test_x_test.reshape(test_x_test.shape[0],test_x_test.shape[1] , 1)
test_x_test.shape

#Splitting the data into arrays - each sample has 3 timestamps of 4 features each
#This step is carried out so as to provide the input to the LSTM model appropriately
dummy_x_test=[]
for i in range(len(norm_x_test)):
    k=np.split(norm_x_test[i], 3)
    dummy_x_test.append(np.array(k))
```

```
new_x_test=np.array(dummy_x_test)
new_x_test.shape
```

Out[24]:
```
(377, 3, 4)
```

## Evaluate the Model

In [25]:
```
value_predicted = model_lstm_3.predict(new_x_test)
```

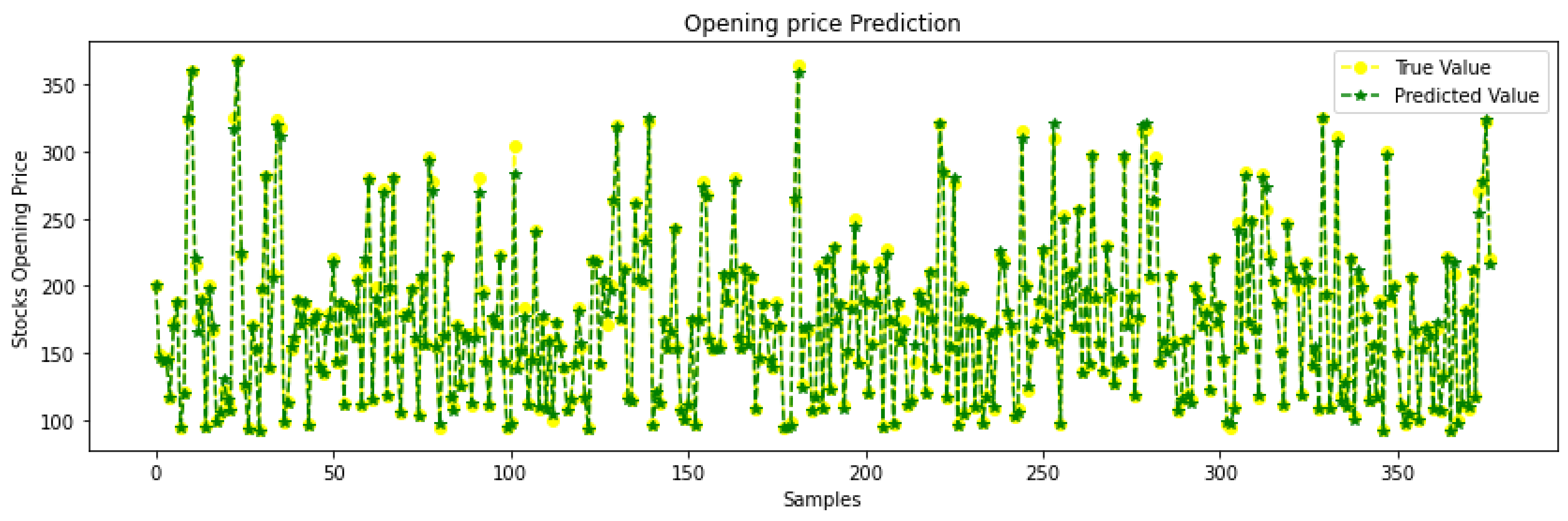
### Loss calculation on Test data

In [26]:
```
test_predict = MinMaxObject_y.inverse_transform(value_predicted)
rmse_test = math.sqrt(mean_squared_error(y_test,test_predict))
loss_test = mean_squared_error(y_test,test_predict)
print("The performance metric considered is Mean squared Error")
#print("Root Mean squared error of LSTM Model 3 = " +str(rmse_test))
print("Loss on the testing data of LSTM Model 3 = " +str(loss_test))
```

```
The performance metric considered is Mean squared Error
Loss on the testing data of LSTM Model 3 = 8.767859272557532
```

### Plot of True and Predicted Values

In [27]:
```
plt.figure(figsize=(14, 4))
plt.plot(y_test.values[::-1], linestyle='--',color='yellow',marker='o')
plt.plot(test_predict[::-1],linestyle='--', color= 'green',marker='*')
plt.legend(['True Value','Predicted Value'], loc='upper right')
plt.title("Opening price Prediction")
plt.xlabel("Samples")
plt.ylabel("Stocks Opening Price")
plt.show()
```



## Model Implementation - Gated Recurrent Unit(GRU)

In [28]:
```
model_GRU_1 = Sequential()
model_GRU_1.add(GRU(units=200, return_sequences=True, input_shape=(new_x_train.shape[1], 4)))
model_GRU_1.add(Dropout(rate = 0.2))
model_GRU_1.add(GRU(units=200))
model_GRU_1.add(Dropout(rate = 0.2))
model_GRU_1.add(Dense(units=1))
model_GRU_1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 3, 200)            123000
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 200)            0
_________________________________________________________________
gru_1 (GRU)                  (None, 200)               240600
_________________________________________________________________
dropout_7 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 201
=================================================================
Total params: 363,801
Trainable params: 363,801
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
model_GRU_1.compile(optimizer='adam', loss='mean_squared_error')
model_GRU_1.fit(new_x_train, norm_y_train, epochs=100, batch_size=32, verbose=0)
```

Out[29]:
```
<keras.callbacks.History at 0x1fe1ac7bbb0>
```

In [30]:
```
train_predict_GRU_1 = model_GRU_1.predict(new_x_train)
train_predict_GRU_1 = MinMaxObject_y.inverse_transform(train_predict_GRU_1)
loss_train_GRU_1 = mean_squared_error(y_train,train_predict_GRU_1)
print("Loss on the training data of GRU Model 1 = " +str(loss_train_GRU_1))
```

```
Loss on the training data of GRU Model 1 = 9.750507704292891
```

## Summary

**Dataset Creation:**

> The dataset given (q2_dataset.csv) has 4 features for each date. We are required to convert this dataset by using the latest 3 days as the features and the next day's opening price as the target.
>
> For doing the same, we have considered the 1st sample on the dataset as the Target sample(t) and the following 3 data samples as the Input data (t-1, t-2, and t-3).
>
> We have created a dataframe with Open, High, Low and Volume columns for all the 3 days. Assigned the feature values of timestamps t-1, t-2, and t-3 to the 1st sample data point in the new dataframe.
>
> We have assigned the (t) sample's Open price to the Target price column in the newly created Dataframe.
>
> Repeated the process for all the data points and created a new dataset with 12 features in each sample and a target price column.
>
> Final Dataset dimension = 1256 rows × 13 columns
>
> This created dataset is randomly splitted into 70% for training and 30% for testing and saved as 'train_data_RNN.csv' and 'test_data_RNN.csv' respectively

**Preprocessing:**

> Train and Test data are normalized so as to scale the stock prices between (0,1). Normalization is performed as-
>
> - This helps in avoiding intensive calculations.
> - The models learning happens quicker due to normalization of the data.
> - In order to have all the features (Open, High, Low prices, and Volume) of the data weighed equally and to avoid false prioritisation of data input to the network.

**Finding the Best Network:**

> - We have explored two Recurrent neural networks - Long Short-Term Memory (LSTM) and Gated recurrent units (GRU). We have done performance evaluation on both these RNN's and have finalized with LSTM due to better training and less loss incurred during our training.
>
> - Usually for long sequences of data, LSTM model performs better when compared to GRU model.
>
> - With GRU network, we have tuned the network(Units/Neurons, # of epochs, Batch size) and could produce the least loss when we have the following parameters: Number of epochs = 100, Batch size = 32
>
> - Training loss of LSTM = 8.6, Training loss of GRU = 9.7. As LSTM is giving us less loss when compared to GRU, so we have choosen **LSTM as our final model.**
>
> - LSTMs have the ability to store past information which would be crucial for the future price prediction of stocks.
>
> - In order to determine the best configuration for the LSTM, we have varied the parameters of the LSTM. Following are the parameters varied - number of layers, number of epochs, and batch size.
>
> - In our implementations, we have carried out various combinations of the hyperparameters. In our code above,we have displayed 3 models with the variations in the parameters (Layers, Epochs and Batch size).
>
> - As we increased the epochs and batch size, we have noticed the Loss(MSE) kept decreasing giving us a better trained network. But, when we increased the number of layers, we have observed that the loss increased. For this reason, we have limited to 2 layers in our network.
>
> - Increasing the number of epochs above 1000, we have observed that the loss was increasing as the network was overfitting.
>
> - Based on the above observations, we have choosen our network which was best fitting for the given data.

**Architecture of our Final Network:**

> - We have finalized the **model_lstm_3 (3rd LSTM model)**
>
> - Following is the architecture of our LSTM network -
>   1. LSTM Layer 1: Units/Neurons = 200, Activation fucntion- "tanh", Recurrent Activation function- "Sigmoid"
>   2. Dropout Layer 1: 20% dropout
>   3. LSTM Layer 2: Units/Neurons = 200, Activation fucntion- "tanh", Recurrent Activation function- "Sigmoid"
>   4. Dropout Layer 2: 20% dropout
>   5. Output Layer: Outputs the predicted stock price (o/p dimension = 1)

> - Optimizer used is "Adam" so as to handle sparse gradients and to train the nework efficiently.
> - Loss function choosen is "Mean Squared Error".
> - **Number of epochs = 1000, Batch size = 64** (We have finalized these values after evaluating the loss and the performance of other combinations).

**Output of the Training loop:**

> The loss encountered during the training phase of the model is **8.6**

**Output from Testing:**

> The loss encountered during the testing phase of the model is **8.7**
> The output plot of True vs Predicted values clearly depicts that majority of the predictions made by the finalized LSTM model are accurate. The predicted values lags behind the actual/true value when there are spikes in the prices. Apart from that, the model could replicate the upward and downward trends in the Opening price prediction.

**More days for features:**

> When we incorporate more days for features i.e.,stacking more timestamps data together, the output price predicted would be more accurate as the model would have reference to more data samples. This would allow the model to train well with the abnormal spikes in the prices which would decrease the loss factor further and improve the prediction of the opening price of the stock.