NAME: JAYANTH RAGAVAN M

BATCH-JULY-AUG

COURSE: DATA SCIENCE(SELF PLACED)

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
import os
from datetime import datetime

import warnings
warnings.filterwarnings("ignore")
```

```
data = pd.read_csv('./s_p_stock/all_stocks_5yr.csv')
print(data.shape)
print(data.sample(7))
```

Output:

(619040, 7)

	date	open	high	low	close	volume	Name
449309	2014-10-30	86.16	87.000	85.670	86.940	5560308	PG
382759	2013-07-25	100.39	101.920	100.000	100.710	3285061	MON
29309	2014-07-03	61.95	62.200	61.830	62.010	773696	AKAM
303701	2016-12-09	53.54	54.115	53.390	53.830	2826688	IP
137693	2016-01-04	132.46	132.460	130.935	131.860	411964	COO
53947	2013-02-19	76.84	77.360	76.270	77.300	5854522	APA
349007	2014-07-10	29.33	29.835	29.255	29.775	916948	LNT

Since the given data consists of a date feature, this is more likely to be an ‘object’ data type.

```
data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619040 entries, 0 to 619039
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    619040 non-null  object
 1   open    619029 non-null  float64
 2   high    619032 non-null  float64
 3   low     619032 non-null  float64
 4   close   619040 non-null  float64
 5   volume  619040 non-null  int64
 6   Name    619040 non-null  object
dtypes: float64(4), int64(1), object(2)
memory usage: 33.1+ MB
```

Whenever we deal with the date or time feature, it should always be in the DateTime data type. Pandas library helps us convert the object date feature to the DateTime data type.

```
data['date'] = pd.to_datetime(data['date'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619040 entries, 0 to 619039
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    619040 non-null  datetime64[ns]
 1   open    619029 non-null  float64
 2   high    619032 non-null  float64
 3   low     619032 non-null  float64
 4   close   619040 non-null  float64
 5   volume  619040 non-null  int64
 6   Name    619040 non-null  object
dtypes: datetime64[ns](1), float64(4), int64(1), object
memory usage: 33.1+ MB
```

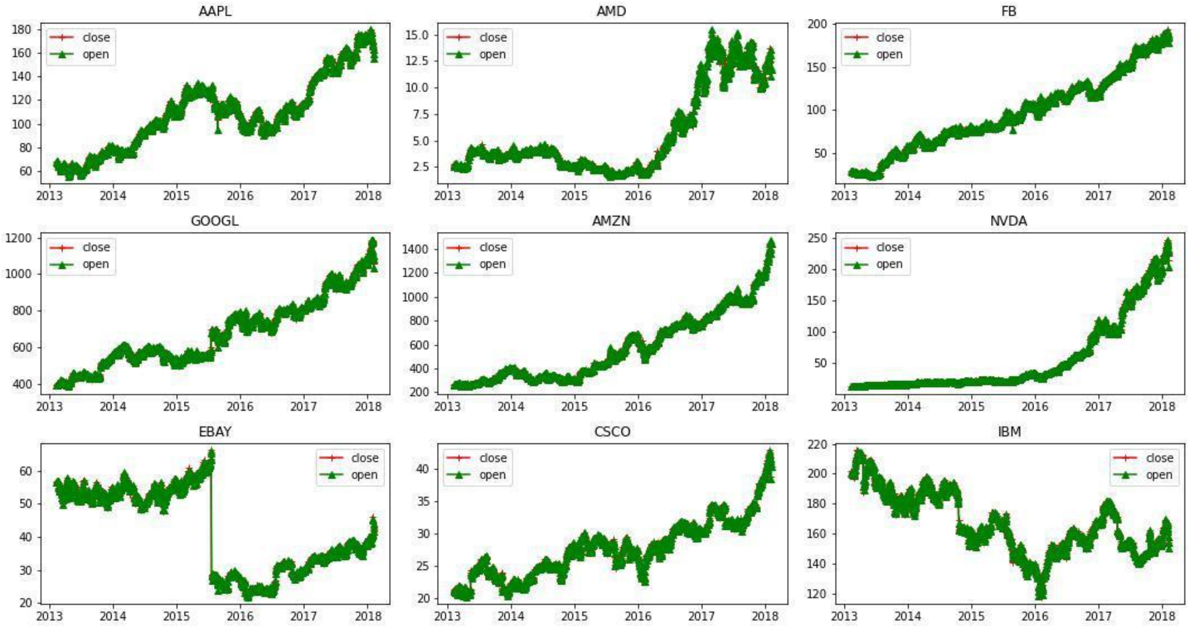
Exploratory Data Analysis

EDA also known as Exploratory Data Analysis is a technique that is used to analyze the data through visualization and manipulation. For this project let us visualize the data of famous companies such as Nvidia, Google, Apple, Facebook, and so on. First, let us consider a few companies and visualize the distribution of open and closed Stock prices through 5 years.

```
data['date'] = pd.to_datetime(data['date'])
```

```python
# date vs open
# date vs close
plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['close'], c="r", label="close", marker="+")
    plt.plot(c['date'], c['open'], c="g", label="open", marker="^")
    plt.title(company)
    plt.legend()
    plt.tight_layout()
```

Output:

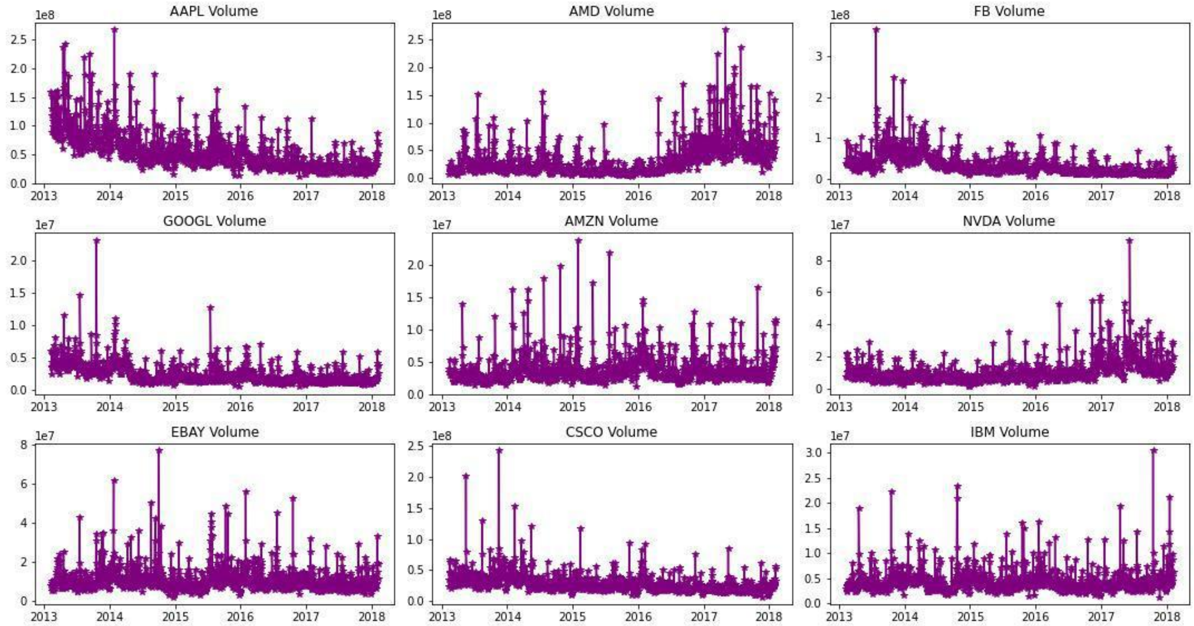


Analyzing Close and Open prices for stocks of 9 different country

Now let's plot the volume of trade for these 9 stocks as well as a function of time.

```python
plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['volume'], c='purple', marker='*')
    plt.title(f"{company} Volume")
    plt.tight_layout()
```

Output:

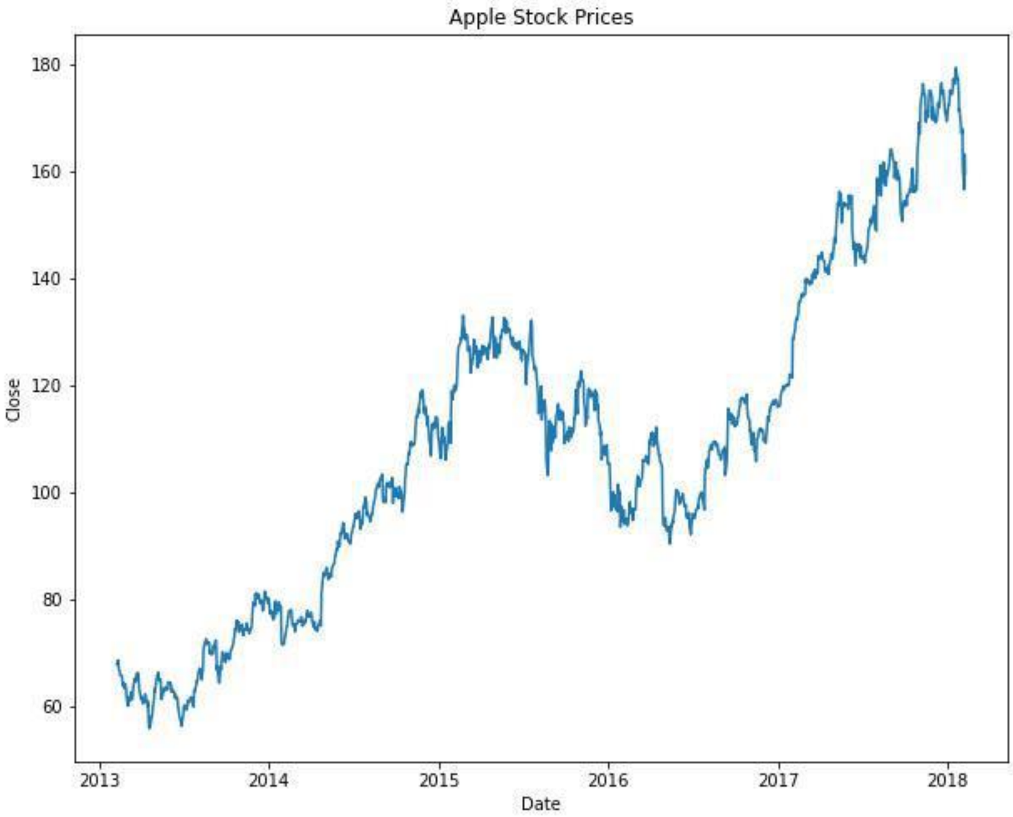


Analyzing volume for stocks of 9 different country

Now let's analyze the data for Apple Stocks from 2013 to 2018.

```
apple = data[data['Name'] == 'AAPL']
prediction_range = apple.loc[(apple['date'] > datetime(2013,1,1))
 & (apple['date']<datetime(2018,1,1))]
plt.plot(apple['date'],apple['close'])
plt.xlabel("Date")
plt.ylabel("Close")
plt.title("Apple Stock Prices")
plt.show()
```

Output:



The overall trend in the prices of the Apple Stocks

Now let's select a subset of the whole data as the training data so, that we will be left with a subset of the data for the validation part as well.

```
close_data = apple.filter(['close'])
dataset = close_data.values
training = int(np.ceil(len(dataset) * .95))
print(training)
```

Output:
1197

Now we have the training data length, next applying scaling and preparing features and labels that are x_train and y_train.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
```

```
scaled_data = scaler.fit_transform(dataset)

train_data = scaled_data[0:int(training), :]
# prepare feature and labels
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
model = keras.models.Sequential()
model.add(keras.layers.LSTM(units=64,
                            return_sequences=True,
                            input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(units=64))
model.add(keras.layers.Dense(32))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(1))
model.summary
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 64)            16896

 lstm_1 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 52,033
Trainable params: 52,033
Non-trainable params: 0
_________________________________________________________________
```

Model summary to analyze the architecture of the model

While compiling a model we provide these three essential parameters:

-

```
model.compile(optimizer='adam',
              loss='mean_squared_error')
history = model.fit(x_train,
                    y_train,
                    epochs=10)
```

Output:

```
Epoch 1/10
36/36 [==============================] - 4s 37ms/step - loss: 0.0334
Epoch 2/10
36/36 [==============================] - 1s 34ms/step - loss: 0.0095
Epoch 3/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0092
Epoch 4/10
36/36 [==============================] - 1s 35ms/step - loss: 0.0081
Epoch 5/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0073
Epoch 6/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0073
Epoch 7/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0071
Epoch 8/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0071
Epoch 9/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0069
Epoch 10/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0065
```

Progress of model training epoch by epoch

For predicting we require testing data, so we first create the testing data and then proceed with the model prediction.

```
test_data = scaled_data[training - 60:, :]
x_test = []
y_test = dataset[training:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

# predict the testing data
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# evaluation metrics
mse = np.mean(((predictions - y_test) ** 2))
print("MSE", mse)
```

```
print("RMSE", np.sqrt(mse))
```

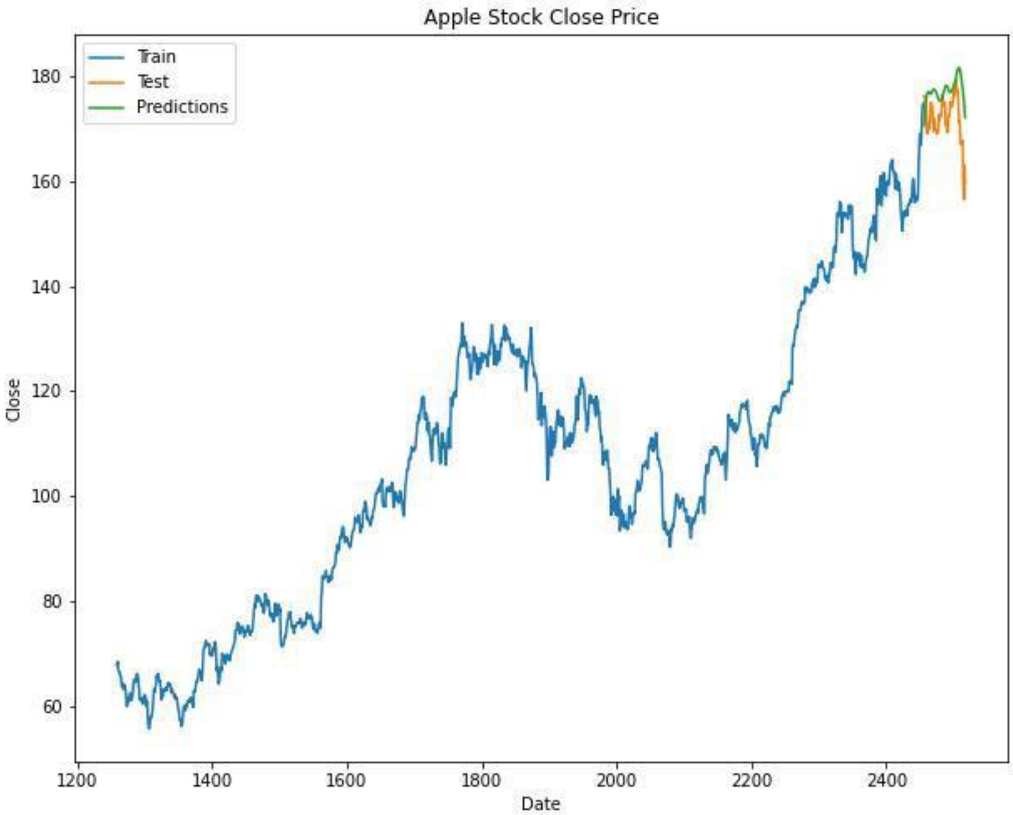
Output:

```
2/2 [==============================] - 1s 13ms/step

MSE 46.06080444818086

RMSE 6.786811066191607
```

Now that we have predicted the testing data, let us visualize the final results.

```
train = apple[:training]
test = apple[training:]
test['Predictions'] = predictions

plt.figure(figsize=(10, 8))
plt.plot(train['Date'], train['Close'])
plt.plot(test['Date'], test[['Close', 'Predictions']])
plt.title('Apple Stock Close Price')
plt.xlabel('Date')
plt.ylabel("Close")
plt.legend(['Train', 'Test', 'Predictions'])
```

Output:



Apple Stock Close Price
Train
Test
Predictions
Close
Date
180
160
140
120
100
80
60
1200
1400
1600
1800
2000
2200
2400