Project 1:- Stock Market Forecasting

For the stock market forecasting I first tried different algorithms, especially ARIMA, LSTM and combination of algorithms like DBN for feature extraction and LSTM to perform better.

Even after tuning the algorithm the best was performed by the LSTM. Following is the algorithm of it.

Code:-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

Importing the libraries which will be needed.

```
class Stock_Price_Forecasting_LSTM:
    def __init__(self, stock_data_path):
        self.stock_data = pd.read_csv(stock_data_path)
        print(self.stock_data.head())
        self.min_max_scaler = MinMaxScaler(feature_range=(0, 1))
        self.X_training_data = None
        self.X_testing_data = None
        self.y_training_data = None
        self.y_test_data = None
        self.lstm_model = None
```

Reading the csv file and initialing variables which will be used throughout the class.

Read the csv data using the pandas.

Declaring min max scaler and creating variables of lstm model, training data and test data.

```
    def stock_data_cleaning(self, time_steps):
        self.stock_data['Date'] =
pd.to_datetime(self.stock_data['Date'])
        self.stock_data.sort_values('Date', inplace=True)
        daily_closing_price =
self.stock_data['Close'].values.reshape(-1, 1)
        scaled_daily_closing_price =
self.min_max_scaler.fit_transform(daily_closing_price)
        X_data, y_data =
self.load_stock_dataset(scaled_daily_closing_price, time_steps)
        X_data = X_data.reshape(X_data.shape[0], X_data.shape[1], 1)
        train_size = int(len(X_data) * 0.97)
        self.X_training_data, self.X_testing_data =
X_data[:train_size], X_data[train_size:]
        self.y_training_data, self.y_test_data =
y_data[:train_size], y_data[train_size:]
        print("After cleaning X training
data"+str(self.X_training_data))
```

Preparing the data in the proper format.

Removing variables which are not required and splitting the data into test and train format.

The assignment states that the prediction should be of last 2 months and training data should be of 5 years.

So the training data is of 5 years and testing data is of 2 months.

Changing the shape of it and converting it into the required format.

```
    def load_stock_dataset(self, data, time_steps):
        X_data, y_data = [], []
        for i in range(len(data) - time_steps):
            X_data.append(data[i:(i + time_steps), 0])
            y_data.append(data[i + time_steps, 0])
        return np.array(X_data), np.array(y_data)
```

This function will divide the code into required format and convert it into the required format. This data is later on returned and made into the required format,

```
    def lstm_stock_model(self):
        lstm_model = Sequential()
        lstm_model.add(LSTM(units=132, return_sequences=True,
input_shape=(self.X_training_data.shape[1], 1)))
        lstm_model.add(Dropout(0.2))
        lstm_model.add(LSTM(units=110, return_sequences=False))
        lstm_model.add(Dropout(0.2))
        lstm_model.add(Dense(units=1))
        lstm_model.compile(optimizer='adam',
loss='mean_squared_error')
        self.lstm_model = lstm_model
        print(self.lstm_model.summary())
```

After tuning the model this model perform best. So the basic summary of the LSTM model is as follows. This model has 3 layers and gets one particular output.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_18 (LSTM)              (None, 60, 132)           70752

 dropout_18 (Dropout)        (None, 60, 132)           0

 lstm_19 (LSTM)              (None, 110)               106920

 dropout_19 (Dropout)        (None, 110)               0

 dense_9 (Dense)             (None, 1)                 111

=================================================================
Total params: 177783 (694.46 KB)
Trainable params: 177783 (694.46 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
  def lstm_model_training(self, epochs=80, batch_size=16,
validation_split=0.1):
        lstm_stock_market_trained_model =
self.lstm_model.fit(self.X_training_data, self.y_training_data,
epochs=epochs, batch_size=batch_size,
validation_split=validation_split)
        return lstm_stock_market_trained_model
```

The above code is used to train the data of designed LSTM model. This model undergoes in the batch of 16 and validation split is 0.1 and number of epochs is 110.

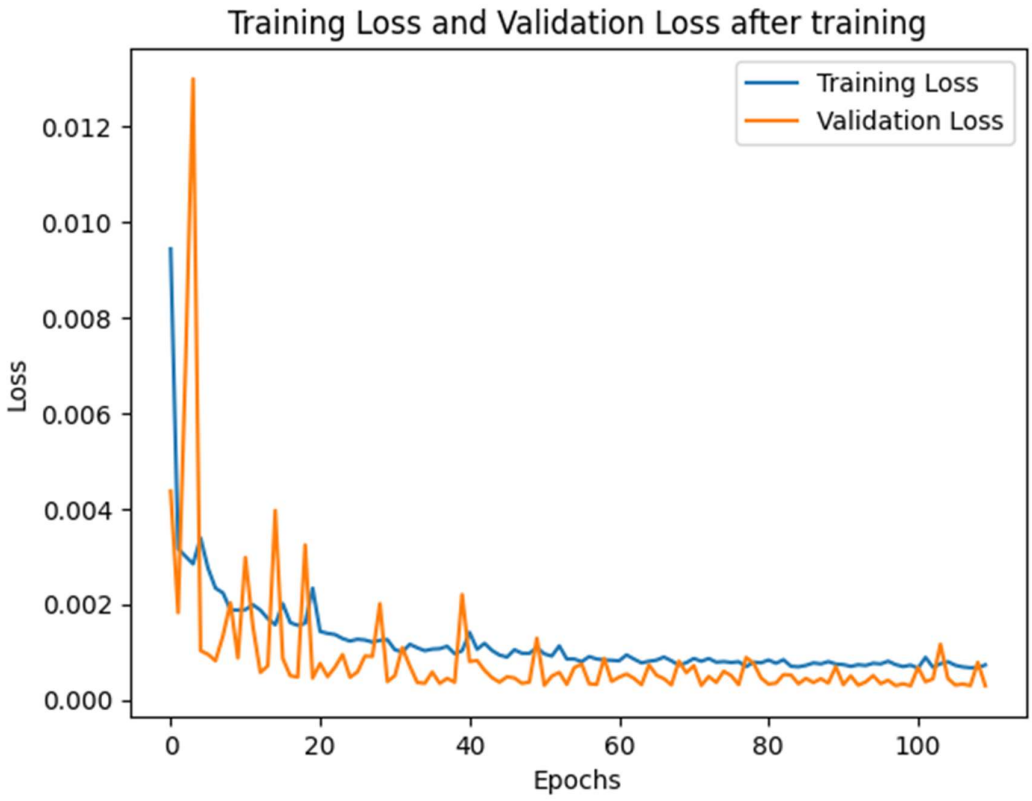
```
Epoch 98/110
63/63 [==============================] - 1s 9ms/step - loss: 7.3518e-04 - val_loss: 2.9526e-04
Epoch 99/110
63/63 [==============================] - 1s 9ms/step - loss: 6.9383e-04 - val_loss: 3.3365e-04
Epoch 100/110
63/63 [==============================] - 1s 9ms/step - loss: 7.2435e-04 - val_loss: 2.9099e-04
Epoch 101/110
63/63 [==============================] - 1s 10ms/step - loss: 6.6187e-04 - val_loss: 6.7564e-04
Epoch 102/110
63/63 [==============================] - 1s 13ms/step - loss: 8.8651e-04 - val_loss: 3.7534e-04
Epoch 103/110
63/63 [==============================] - 1s 13ms/step - loss: 6.8010e-04 - val_loss: 4.3724e-04
Epoch 104/110
63/63 [==============================] - 1s 10ms/step - loss: 7.5334e-04 - val_loss: 0.0012
Epoch 105/110
63/63 [==============================] - 1s 9ms/step - loss: 7.9542e-04 - val_loss: 4.4339e-04
Epoch 106/110
63/63 [==============================] - 1s 9ms/step - loss: 7.2005e-04 - val_loss: 3.0608e-04
Epoch 107/110
63/63 [==============================] - 1s 10ms/step - loss: 6.8611e-04 - val_loss: 3.2883e-04
Epoch 108/110
63/63 [==============================] - 1s 9ms/step - loss: 6.6947e-04 - val_loss: 2.9495e-04
Epoch 109/110
63/63 [==============================] - 1s 9ms/step - loss: 6.7453e-04 - val_loss: 7.8192e-04
Epoch 110/110
63/63 [==============================] - 1s 9ms/step - loss: 7.2747e-04 - val_loss: 2.9171e-04
```

```
def rsme_mae(self):
        lstm_train_prediction =
self.lstm_model.predict(self.X_training_data)
        lstm_test_predictions =
self.lstm_model.predict(self.X_testing_data)
        lstm_train_prediction =
self.min_max_scaler.inverse_transform(lstm_train_prediction)
        y_training_data =
self.min_max_scaler.inverse_transform([self.y_training_data])
        lstm_test_predictions =
self.min_max_scaler.inverse_transform(lstm_test_predictions)
        y_test_data =
self.min_max_scaler.inverse_transform([self.y_test_data])

        trianed_data_rsme_result =
np.sqrt(mean_squared_error(y_training_data[0],
lstm_train_prediction[:, 0]))
        test_data_rsme_result =
np.sqrt(mean_squared_error(y_test_data[0], lstm_test_predictions[:,
0]))
        trianed_data_mae_result =
mean_absolute_error(y_training_data[0], lstm_train_prediction[:, 0])
```

```
        tested_data_mae_result = mean_absolute_error(y_test_data[0],
lstm_test_predictions[:, 0])

        return trianed_data_rsme_result, test_data_rsme_result,
trianed_data_mae_result, tested_data_mae_result, y_training_data[0],
lstm_train_prediction[:, 0], y_test_data[0],
lstm_test_predictions[:, 0]
```

```
def graph_training_loss_validation_loss(self, lstm_predicted_stock):
        plt.plot(lstm_predicted_stock.history['loss'],
label='Training Loss')
        plt.plot(lstm_predicted_stock.history['val_loss'],
label='Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.title('Training Loss and Validation Loss after
training')
        plt.legend()
        plt.show()
```

Following function provides the Training RMSE score, Testing RMSE score, Training MAE score and Testing MAE score. This function is also prints the loss training loss and validation loss.




```
35/35 [==============================] - 1s 5ms/step
2/2 [==============================] - 0s 7ms/step
Training data root mean square error (RMSE): 1.229837425806187
Testing data root mean square error (RMSE): 1.1668959818843025
Training data Mean Absolute Error (MAE): 0.9172184561571866
Testing data Mean Absolute Error (MAE): 0.9484270105957027
```

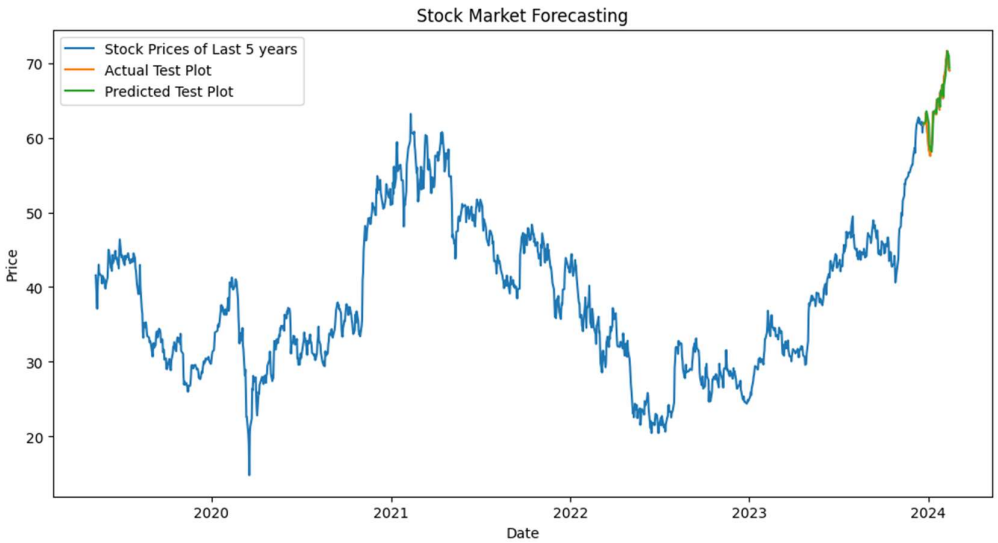
```
def lstm_stock_market_forecasting_plot(self, test_actual,
test_predicted):
        plt.figure(figsize=(12, 6))
        plt.plot(self.stock_data['Date'][-len(test_actual):],
test_actual, label='Actual Test plot')
        plt.plot(self.stock_data['Date'][-len(test_predicted):],
test_predicted, label='Predicted Test plot')
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.title('Prediction and Actual plot')
        plt.legend()
        plt.show()
```

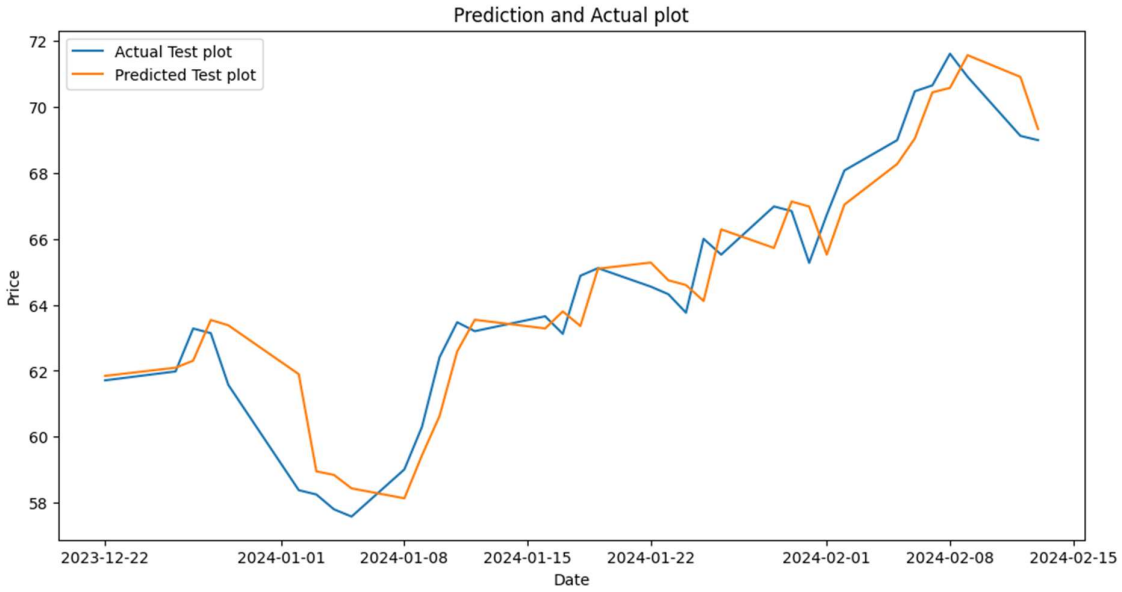
Following code is used for the to plot the prediction. This is done using the matplotlib function. Here x axis has year and y axis has the value of the stock.

```
 def five_year_stock_price_plot(self, test_actual, test_predicted):
        plt.figure(figsize=(12, 6))
        plt.plot(self.stock_data['Date'][-(5 * 365):],
self.stock_data['Close'][-(5 * 365):], label='Stock Prices of Last 5
years')
        plt.plot(self.stock_data['Date'][-len(test_actual):],
test_actual, label='Actual Test Plot')
        plt.plot(self.stock_data['Date'][-len(test_predicted):],
test_predicted, label='Predicted Test Plot')
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.title('Stock Market Forecasting')
        plt.legend()
        plt.show()
```

It will plot 5 years of the stock data.

Output:-





How To Run:-

1) If you are trying to run the on VaibhavParikh_RL.py

Then write following in the command prompt

```
pip install pandas numpy torch transformers scikit-learn
```

If you are running it on the VaibhavParikh_RL.ipynb file then on first line already pip import has already been written. You just need to run it.

2) In command prompt just write
   python VaibhavParikh_RL.py

or just run the entire VaibhavParikh_RL.ipymb