Forecasting of stock Prices using ARIMA, RNN, LSTM

Team: *Quad Squad*

Sri Nikhitha Boddapati - 16322565

Sukumar Bodapati - 16326105

Pavan Kumar Jonnadula - 16324822

Vamsi Alapaty - 18230326

Source Code Link:

https://github.com/Srinikhitha98/BigDataProject/tree/main/Source%20Code

Video Link:

https://github.com/Srinikhitha98/BigDataProject/tree/main/Video

Introduction:

In today's world, we have multiple models to forecast the stock market.

- Out of all the techniques we have, which strategies are the most effective?
- Is it better to try to predict stock prices or to trade strategies based on stock price movement?
- Can we predict and tell when we can buy a stock using these models?
- What parameters can we use for various situations?

DataSet:

In this project, we are using Amazon Stock data.

Data Exploration:

```
hist_data.reset_index(inplace=True)
hist_data.head(3)
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	1997-05-15	2.437500	2.500000	1.927083	1.958333	72156000	0	0.0
1	1997-05-16	1.968750	1.979167	1.708333	1.729167	14700000	0	0.0
2	1997-05-19	1.760417	1.770833	1.625000	1.708333	6106800	0	0.0

As we are dealing with the prediction, if we examine the closing prices and dates. The goal of time series analysis is to locate closing prices. The reason for this is that closing prices, as opposed to opening or average pricing, more accurately reflect how business was doing. So, consider the Close data for prediction.

Let's have a look at how many days we have in this dataset and visualize the data to consider training and testing samples.

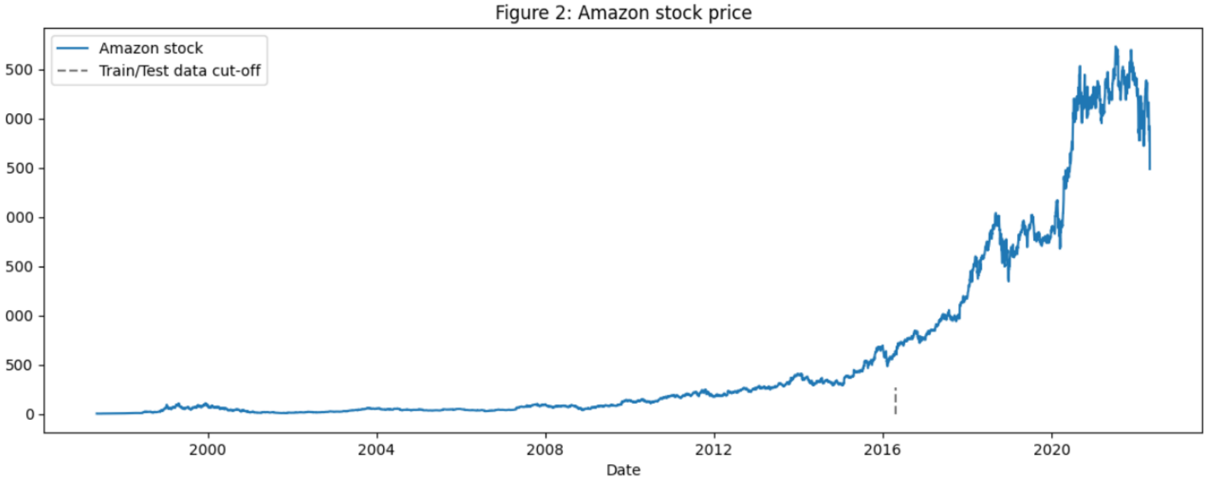
```
print('There are {} number of days in the dataset.'.format(df.shape[0]))
```

```
There are 6282 number of days in the dataset.
```

In this dataset, it has 6282 number of days in this dataset.

Visualization:

Figure 2: Amazon stock price



From the above graphical observations, most of the Stock price from 2015 is exponential growing. Amazon's business really took off after 2010 and peaked around 2015, therefore the testing will be very interesting. For training, normalize the training data so that related data points across time are condensed to a single data point, allowing the model to train in a way that anticipates exponential growth beyond 2015.

Feature Extraction:

Designed the below indicators for the feature Extraction:

- **Momentum:** This indicator will be useful in predicting future stock prices as well as determining whether we should acquire the stock. Momentum is possibly the simplest and most straightforward oscillator (financial analysis instrument) to comprehend and apply. It is the measurement of the rate of change in price movement for a certain item, as well as the speed or velocity of price movements.
  Momentum(m)=Latest Price -Closing price (at Number of days ago)

- **Bollinger bands:** For Quantitative Analysis of Stock market, this indicator is useful. These are used to characterize the trading range of a financials' by defining the current high and low values in a market. They are measure of volatility and a Moving Average (MA) line, an upper band, and a lower band make up the bands. MA simply adds and subtracts standard deviation to get the upper and lower bands.

- **EMA:** The exponential moving average (EMA) is a better variant of the simple moving average (SMA). Moving averages simply average out the data over a period, allowing us to see how the company's closing price has changed over time. For example, if the price was 32,33,45,1 for four days (the company downs on the fourth day), the average would be 32. Now, because 32 is a lower-than-average number, it suggests that 45 was a fluke and that the company was always losing money.

$$EMA(t)EMA(t0)=(1-\alpha)EMA(t-1)+\alpha\ p(t)=p(t0)$$

where $\alpha=1L+1$ and length of window is $\alpha=2M$

For Generating Technical Indicators:

```python
[ ] def get_technical_indicators(dataset): #function to generate feature technical indicators


        # Create 7 and 21 days Moving Average
        dataset['ma7'] = dataset['Close'].rolling(window = 7).mean()
        dataset['ma21'] = dataset['Close'].rolling(window = 21).mean()

        #Create MACD
        dataset['26ema'] = dataset['Close'].ewm(span=26).mean()
        dataset['12ema'] = dataset['Close'].ewm(span=12).mean()
        dataset['MACD'] = (dataset['12ema']-dataset['26ema'])

        #Create Bollinger Bands
        dataset['20sd'] = dataset['Close'].rolling(window = 20).std()
        dataset['upper_band'] = (dataset['Close'].rolling(window = 20).mean()) + (dataset['20sd']*2)
        dataset['lower_band'] = (dataset['Close'].rolling(window = 20).mean()) - (dataset['20sd']*2)


        #Create Exponential moving average
        dataset['ema'] = dataset['Close'].ewm(com=0.5).mean()

        #Create Momentum
        dataset['momentum'] = (dataset['Close']/100)-1



        return dataset
```

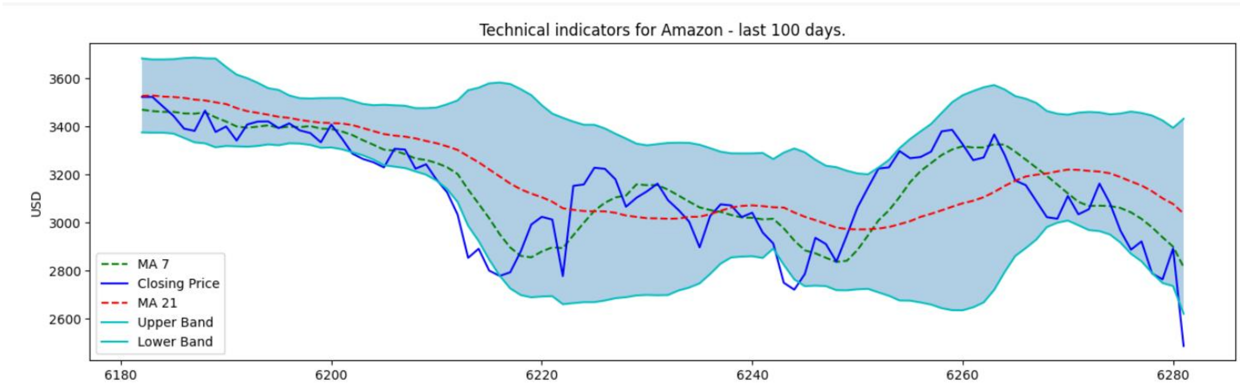
Calculated indicators:

```python
[ ] dataset_TI_df = get_technical_indicators(hist_data)
```

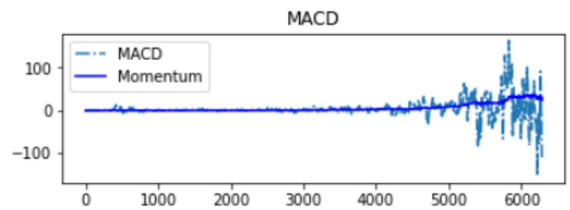
```python
[ ] dataset_TI_df.head()
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	ma7	ma21	26ema	12ema	MACD	20sd	upper_band	lower_band	ema	momentum
0	1997-05-15	2.437500	2.500000	1.927083	1.958333	72156000	0	0.0	NaN	NaN	1.958333	1.958333	0.000000	NaN	NaN	NaN	1.958333	-0.980417
1	1997-05-16	1.968750	1.979167	1.708333	1.729167	14700000	0	0.0	NaN	NaN	1.839343	1.834201	-0.005142	NaN	NaN	NaN	1.786458	-0.982708
2	1997-05-19	1.760417	1.770833	1.625000	1.708333	6106800	0	0.0	NaN	NaN	1.792272	1.785075	-0.007197	NaN	NaN	NaN	1.732372	-0.982917
3	1997-05-20	1.729167	1.750000	1.635417	1.635417	5467200	0	0.0	NaN	NaN	1.748422	1.737834	-0.010589	NaN	NaN	NaN	1.666927	-0.983646
4	1997-05-21	1.635417	1.645833	1.375000	1.427083	18853200	0	0.0	NaN	NaN	1.673903	1.653404	-0.020499	NaN	NaN	NaN	1.506370	-0.985729

Visualizing the indicators:



The above graph indicates the technical indicators that we designed for the last 100 days.



The threshold between MACD and momentum is displayed in this graph. As can be seen, momentum gives the MACD an average value between the peak values and the highest or lowest values. MACD is dependent on the above calculated moving average features.

ARIMA Model:

One of the most prominent strategies for predicting future values of time series data was Autoregressive Integrated Moving Average (ARIMA) (in the pre-neural networks ages). Let's try it out and see whether it turns out to be a useful predictive feature.

Key aspects of this model:

- *'AR'* stands for autoregression. The dependent relationship between an observation and a set of lagged observations is used in this model.
- *'I'* stand for "integrated." To make the time series steady, differencing raw observations (e.g. subtracting an observation from an observation from the preceding time step) is used.
- *'MA'* stands for Moving Average. A model that takes advantage of the relationship between an observation and the residual error from a moving average model when applied to lagged observations.

Parameters used in this model:

- *'p'*: The lag order, or the number of lag observations incorporated in the model.
- *'d'*: The degree of differencing is the number of times the raw observations are differenced.
- *'q'*: The order of moving average, also known as the size of the moving average window.

```
[ ] from statsmodels.tsa.arima_model import ARIMA
    from pandas import DataFrame
    from pandas import datetime

    series = data_FT['Close']
    model = ARIMA(series, order=(5, 1, 0))
    model_fit = model.fit(disp=0)
    print(model_fit.summary())
```

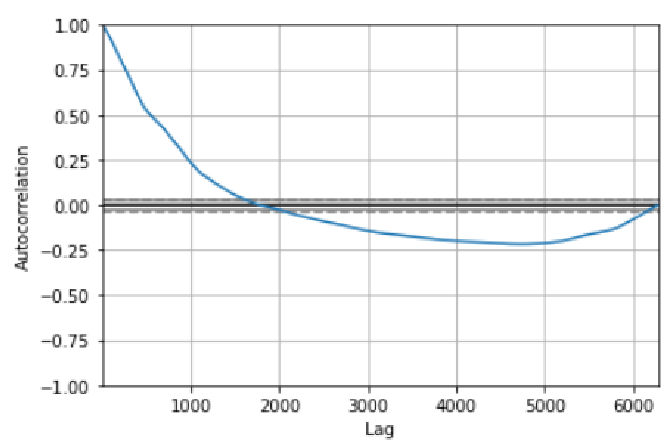
Define the model and set the lag value to 5 for regression and used difference order of 1 to make stationary time series and moving average model to zero.

Model Results:

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.Close   No. Observations:                 6281
Model:                 ARIMA(5, 1, 0)   Log Likelihood              -28379.150
Method:                       css-mle   S.D. of innovations             22.183
Date:                Mon, 02 May 2022   AIC                          56772.300
Time:                        13:29:40   BIC                          56819.517
Sample:                             1   HQIC                         56788.660

=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const             0.3997      0.257      1.554      0.120      -0.104       0.904
ar.L1.D.Close    -0.0406      0.013     -3.129      0.002      -0.066      -0.015
ar.L2.D.Close    -0.0088      0.013     -0.678      0.498      -0.034       0.017
ar.L3.D.Close    -0.0490      0.013     -3.773      0.000      -0.074      -0.024
ar.L4.D.Close     0.0169      0.013      1.292      0.196      -0.009       0.042
ar.L5.D.Close    -0.0065      0.013     -0.502      0.616      -0.032       0.019
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -2.0482           -0.0000j            2.0482           -0.5000
AR.2           -0.0596           -2.6999j            2.7005           -0.2535
AR.3           -0.0596           +2.6999j            2.7005            0.2535
AR.4            2.3713           -2.1445j            3.1971           -0.1170
AR.5            2.3713           +2.1445j            3.1971            0.1170
-----------------------------------------------------------------------------
```

Summary of the model:

```
plt.show()
```



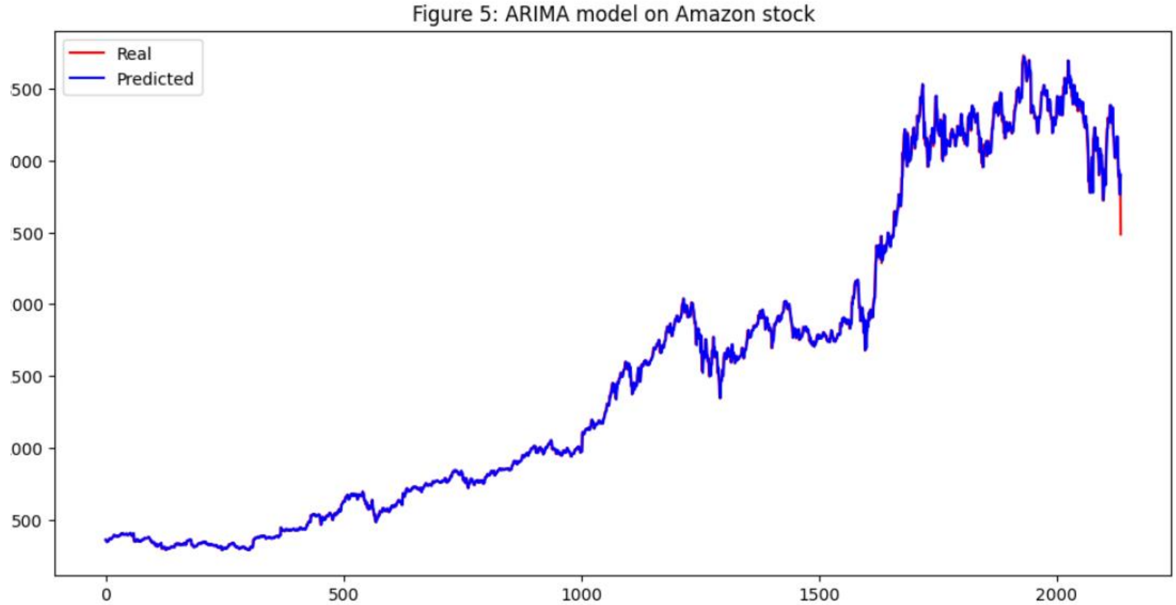

```
<Figure size 800x560 with 0 Axes>
```

- We found that 5 was a decent starting point for the AR parameter of the model.
- Except for the last two, most P-values are greater than 0.05, according to the ARIMA summary. The model working is pretty good.

- In the statistics of the model the difference between Bayesian information criterion and Akaike information criterion is very low which depicts that this model is good.
- In the autocorrelation below, we can see that there is a positive correlation with the first 0-to-500 lags, which is possibly significant for the first 250 lags.

Visualization of ARIMA model:

Figure 5: ARIMA model on Amazon stock



If we observe the above graph, the model is very good. The real and predicted values are closer.

LSTM Model:

The LSTM algorithm excels in forecasting stock market data. We'll first try to forecast closing prices with only one feature, Open (which has the strongest connection to closing prices), and then with many features (using some form of one-hot encoding) to see what we can come up with.

Cleaned the dataset and count the features, samples:

```
print('Total dataset has {} samples, and {} features.'.format(dataset_lstm_df.shape[0], \
                                                              dataset_lstm_df.shape[1]))
```

```
Total dataset has 6282 samples, and 19 features.
```

Normalized the data:

```
] #normalise
  from sklearn.preprocessing import MinMaxScaler
  scaler = MinMaxScaler(feature_range = (0, 1))
  open_training = scaler.fit_transform(open_training)
  #convert to right shape
  features_set_1 = []
  labels_1 = []
  for i in range(60,450):
      features_set_1.append(open_training[i-60:i, 0])
      labels_1.append(open_training[i, 0])
```

Creating label and feature set:

```
] features_set_1, labels_1 = np.array(features_set_1), np.array(labels_1)
  features_set_1 = np.reshape(features_set_1, (features_set_1.shape[0], features_set_1.shape[1], 1))
```

LSTM Model:

```
[ ] #training it
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(features_set_1.shape[1],1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(units = 1))
    model.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics = ['mean_absolute_error'])
    model.fit(features_set_1, labels_1, epochs = 100, batch_size = 32,validation_data = (features_set_1, labels_1))
```

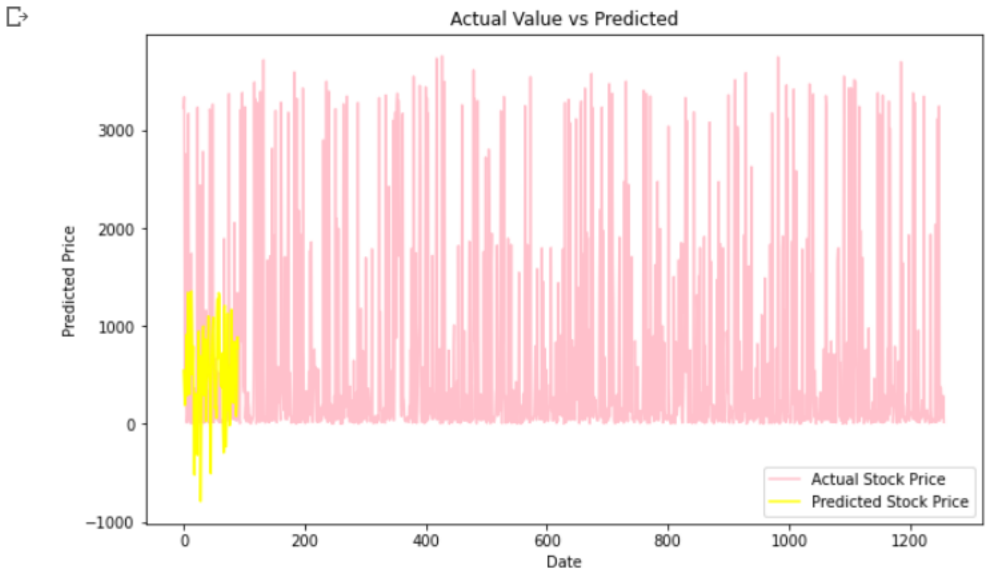
Used 100 epochs of Open training data to try to predict Open. Because this is more of a regression concern, utilized MSE and mean absolute error instead of accuracy. Because there was some overfitting, there we need to normalize the data.

```
Epoch 100/100
13/13 [==============================] - 1s 42ms/step - loss: 0.0426 - mean_absolute_error: 0.1567 - val_loss: 0.0406 - val_mear
<keras.callbacks.History at 0x7f8a3f4ad2d0>
```

MAE was: ***0.1567***

This suggests that for all 2265 datapoints, the average difference between input and output is 0.1567.

Visualization of the model:



If we observe, this model is not so great.

LSTM with multiple features:

In the previous, we have considered only one feature. In this we will try to predict using multiple features.

	Open	Close	High	Low
0	2.437500	1.958333	2.500000	1.927083
1	1.968750	1.729167	1.979167	1.708333
2	1.760417	1.708333	1.770833	1.625000
3	1.729167	1.635417	1.750000	1.635417
4	1.635417	1.427083	1.645833	1.375000

The above data is used to train the model.

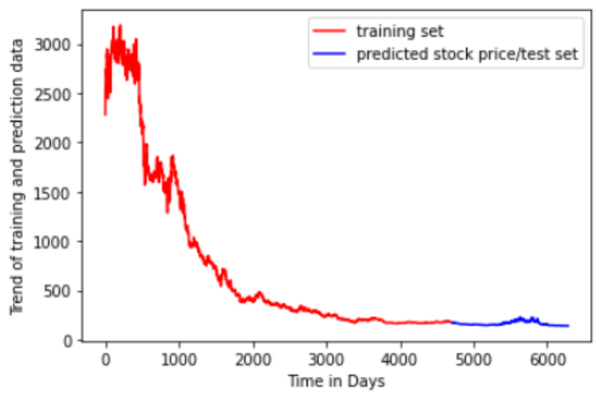
Model:

```
[ ] # LSTM MODEL
    model = Sequential()
    model.add(LSTM(32, input_shape=(1, step_size), return_sequences = True))
    model.add(LSTM(16))
    model.add(Dense(1))
    model.add(Activation('linear'))
```

```
[ ] # MODEL COMPILING AND TRAINING
    model.compile(loss='mean_squared_error', optimizer='adagrad',metrics = ['mae']) # Try mae, adam, adagr
    model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)
```

With one feature, the mean absolute error is smaller than the previous model. The error is just under 0.0026. As a result, the training model should be quite like the testing model.

Prediction:



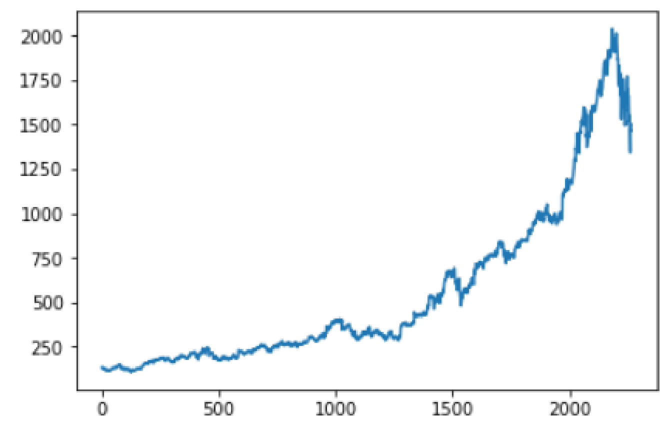
Predicting the next day values:

```
[ ] # PREDICT FUTURE VALUES
    last_val = testPredict[-1]
    last_val_scaled = last_val/last_val
    next_val = model.predict(np.reshape(last_val_scaled, (1,1,1)))
    print("Last Day Value:", np.asscalar(last_val))
    print("Next Day Value:", np.asscalar(last_val*next_val))
    # print np.append(last_val, next_val)

    Last Day Value: 139.40843200683594
    Next Day Value: 119.3088150024414
```

LSTM (Vanilla) Model:

We merely normalized the closing prices before splitting it into train and test datasets. We basically used a simple lookback window to give all similar data the same movement (movement is simply the same kind of normalization applied to the data points) and input them into price points. Have changed the hyper parameters from the past mode to give the best output.

Data visualization for close values:



Normalize the data so that prices are reduced to normalized values, and instead of actual prices, we can forecast stock movement.

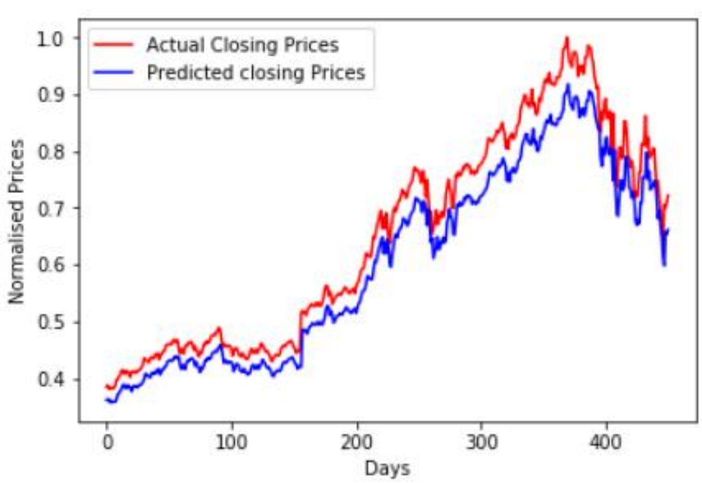
Model:

```
[ ] #Step 2 Build Model
    model = Sequential()

    model.add(LSTM(
        input_dim=1,
        output_dim=50,
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        100,
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=1))
    model.add(Activation('relu'))

    start = time.time()
    model.compile(loss='mse', optimizer='rmsprop', metrics=['mae'])
    print ('compilation time : ', time.time() - start)
```

Error:

```
Epoch 40/40
1719/1719 [==============================] - 0s 95us/step - loss: 1.1486e-04 - mean_absolute_error: 0.0081 - val_loss: 5.5542e-04 - val_mean_absolute_error: 0.02
<keras.callbacks.History at 0x7ff31164e208>
```

Visualization of the model:



When we normalize the prices and estimate the stock price movement, LSTM works wonderfully for predicting Closing Prices.

```
mae = model.evaluate(trainX,
    trainY,
    batch_size=128,verbose=1)
```

```
1810/1810 [==============================] - 0s 28us/step
[0.00017531904821587696, 0.01169782763974772]
```

The error is too low i.e., ***0.000175.***

Conclusion:

In this project, used ARIMA, LSTM, LSTM with multiple features to forecast multiple models of the time series OHLCV data, perform feature extraction, hyper parameter tweaking, and train on ARIMA, LSTM. Based on the above conclusions, LSTM works well.

From this model we can assume that the next day value as **119.30**

Future work:

- Stock prediction can be done using GAN. We tried to implement using TimeGAN but due to complex architecture and time constraints, we couldn't complete it.
- We can predict the next day values for different datasets from Day 1 to till today.
- This model can be integrated into any web application where we can show the predict of next day stock value for all the trending stocks (Creating the stock market website).