B.Tech. Project Report

CSPE-40

On

STOCK PRICE PREDICTOR

By

Sandeep Kumar Meena(11710147)

Suraj Kumar Chaudhary(11710150)

Ajay(11710173)

Group No.: 6

Under the Supervision of

Prof. S.K. Jain, Professor

DEPARTMENT OF COMPUTER ENGINEERING,

NATIONAL INSTITUTE OF TECHNOLOGY KURUKSHETRA –

136119, HARYANA (INDIA)

April-May, 2021

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech Project Work (**CSPE-40)**report entitled **"**Stock Price Prediction**",** in partial fulfillment of the requirements for the award of the **B. Tech (Computer Engineering)** is an authentic record of our own work carried out during a period from January, 2021 to April, 2021 under the supervision of Prof. Sanjay Jain, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidate
Sandeep Kumar Meena (11710147)
Suraj Kumar Chaudhary (11710150)
Ajay (11710173)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signature of Supervisor

Prof. Sanjay Jain

Professor

TABLE OF CONTENTS

Abstract

Stock price prediction is the one of the complex problem of Machine Learning. As it depend on the no. of factors. In this project we tried to predict the closing value of a stock price. We did the technical analysis of various approach for stock price prediction proposed in the past. After learning about the available approaches for prediction we implemented them and compared their result using mean square error as metrics .After implementation we can see how one model is better that the other by comparing their mean square error. Among them we found LSTM giving the best result. And then we improved the LSTM model by fine tuning its various parameter so that it can give more accurate result.

We are using time series data of stocks for stock price prediction. LSTM is one of the popular algorithm for time series data for classification and prediction as it is able to store the past important information and forgets non-important information. In this way we implement various algorithms for stock price prediction and are able to get more accurate results using LSTM ant its improve model.

1. Introduction

What is stock Price?

A stock price is the value given for every share issued by a publicly traded company. It reflects the company's value, which the public is willing to pay for a piece of the company. Stock price value can rise and fall, based on a variety of factors in the global landscape and within the company itself.

Problem Statement:

The challenge of this project is to accurately predict the future closing value of the stock price of a company across a period of time. For this we will be using LSTM (Deep Learning model) with other machine model as benchmark model. We will be using S&P 500 past prices as the dataset.

Goal:

1. Explore stock prices.
2. Implement a basic model using Linear regression.
3. Implement other machine learning model like KNN,SVM,RNN,DNN.
4. Implement LSTM for predicting price using time series data.
5. Improve LSTM to increase accuracy.
6. Compare results of implemented models.

Performance Matrices:

For measuring performance of models we will be using Mean Squared Error(MSE) and Root Mean Squared Error(RMSE), which can be calculated as the difference between the predicted and actual closing price of the targeted stock price. We will also be calculating the delta between the performance of benchmark model(Linear Regression Implementation) and primary model(LSTM implementation)

2. Motivation:

Stock price is a place where a person can earn fortune, if he knows how the stock price of a company will behave in future. But it is impossible to correctly predict the future stock price of a company. As we know that various experts try to predict the future stock value, using the various factors, but the question is can we make a machine learn to predict the future stock value using the factors humans use. Today all Big fin-tech company and hedge funds are building and using some kind of financial model to better understand the market and tries to predict how the market will behave.

Through this project we tried to use Deep learning Models, LSTM a neural network algorithm to predict the stock price. We will we using the RNN (specifically LTM variant) as they are most popular and used for data with time frames. By using the results of these models we will we able to get some king of idea, how a company stock price will behave , as it depend on the companies past stock values.

3. Literature survey:

Analysis of stock market is broadly divided into two parts:

- Fundamental analysis
- Technical analysis

Fundamental analysis : It involves analyzing the company's profitability on the basis of its current business environment and financial performance.

Technical Analysis: It involves reading the charts and using statistical figures to identify the trends in the stock market.

For this project we will be focused on the the technical analysis. We will be using various ML Algorithms to predict stock price,here we will be using data if Alphabet incorporation.

3.1 Different Algo for stock price prediction:

Following are the various Algorithms through which we can predict stock price

1). Linear Regression:

It is an approach for predictive modeling to showcase the relation between a scalar dependent variable 'Y' (for stock price it is close attribute) and one or more independent variable 'X' (in our case trading day attribute),

Its equation is given by;

$Y = bX + a$

Y=predicted close vector

X=Day vector

b = weight, a= bias

Initially with random weight and bias we draw the graph, after that we calculate the mean square error . We will change the wt and bias value until we achieve the minimum error.

2). k-Nearest Neighbor Classifier (KNN) :

KNN is a machine learning algorithm that is quite easy to implement . This technique can be used for the stock price prediction problem , where the problem of stock prediction can be

mapped into a similarity based classification. Both the test data as well as the historical stock data are mapped into a set of vectors. Each stock feature is represented by a N dimension mapped vector. Afterwards, the decision is made by computing a similarity metric like Euclidean distance . KNN is termed as a lazy learning technique that doesn't build a function or model previously, instead yields the k closest records of the training data set that are the most similar to the test. Then, a voting is done among the selected k records to determine the class label having the majority voting and then it is assigned to the query record.

Stock market closing price prediction can be computed using KNN as follows:

a) Determine nearest neighbors number , k.

b) Compute the distance between the query record and the training samples .

c) Sort all training records using their distance values.

d) Majority voting will be used for the class labels of k nearest neighbors, and it will be assigned as a prediction value of the query record

3) Support Vector Machine (SVM) :

Data classification technique - SVM , is a supervised machine learning model , which is used for two - group classification problems . After training SVM models with sets of labeled training data for each different category, they can categorize new text.

SVM creates a decision boundary which shares most points of one category on one side of the boundary while points of the other category on the other side of the boundary.

Examine an n-dimensional feature vector $x = (X1,...,Xn)$.

A linear boundary (hyperplane) can be defined as $\beta0 + \beta1X1 + ... + \beta nXn = \beta0 + n \sum i=1 \beta iXi = 0$

Which gives , elements of one category have sum greater than 0, while elements of other category will have sum less than 0.

Consider labeled examples,

$\beta0 + \sum n\ i=1\ \beta iXi = y,,$

where y is the label. Here ,for our simplicity, $y \in \{-1,1\}$.

We can also write the hyperplane equation with the help of inner products.$y = \beta0 + \sum \alpha iyix(i) * x$

where ∗is the inner product operator. It is to be noted that the inner product is weighted by its label only . The optimal hyperplane should maximize the distance from the plane to any other point , termed as margin. The more the margin the better the data is splitted. Nevertheless , since it might not be a perfect differentiation, we often add error variables ε1...εn and keep their sum below some budget for instance B. The most important element is that only the points which are closest to the boundary are considered for hyperplane selection , all other points can be termed as irrelevant. These points are the support vectors, and the hyperplane is termed as a SVC(Support Vector Classifier) for the reason that it places each support vector in one class or the other class .

4). Deep Neural Network (DNN) :

DNN - Deep Neural Network is an artificial neural network having multiple layers between the input and the output layers. DNN have the following components: weights,neurons, synapses functions, and biases .These components function just like the human brain and can be trained just like any other algorithm of Machine Learning .For instance , a Deep Neural Network is trained to recognize different dog breeds will transverse through the given image and calculate the probability that the dog in the given image belongs to a certain breed. We can review the results and then select which probability the network should display (define some threshold, etc.) and then return the proposed label. Each of the mathematical manipulations like this is considered a layer, and a complex Deep Neural Network has a large number of layers, so the name --**deep**-- networks. DNNs can also model non - linear complex relationships. DNN architectures produce compositional models where the object is shown as a layered constitution of primitives. The extra added layers allows composition of features from lower layers, possibly modeling complex data with fewer units compared to a similarly performing shallow network. For example , it was proved that DNNs approximate sparse multivariate polynomials exponentially easier than with shallow networks .Deep Neural Networks are usually the feed forward networks where data flows from the input layer to the output layer without looping back. Initially, the DNN creates a web of virtual neurons and random numerical values, or more specifically -weights, are assigned to the connections between them. The inputs and the weights are multiplied and then it returns an output between 0 and 1. If that network failed to accurately recognize a particular pattern, then an algorithm would adjust the weights between the connections . In this manner , the algorithm

can make certain parameters more weighted , until the model determines the correct mathematical manipulation to completely process the data.

5). Recurrent Neural Network (RNN) :

RNN - Recurrent Neural Network , is a Neural Network where the output from the last step are provided as the input to the active step , All the outputs and inputs are independent of each other in traditional neural networks, but in the cases where it is required to predict the next word of a sentence, the knowledge of some previous words are required so , there is a need to remember some of the the previous words . This is the reason why RNN comes into the picture , which solved the issue using the concept of Hidden Layer. The foremost important characteristic of RNN is its Hidden state, which remembers some of the information about a sequence.

RNN also has a --**memory**-- which helps in remembering all the information about what has been performed. Same parameters are used for each of the inputs because it performs the same task on all the hidden or input layers to give the output. This helps in reducing the complexity of the parameters, which makes it different from other neural networks.

Advantages of RNN

a).RNN has memory so it remembers most of the information through time , which makes it useful for time series prediction.

b).RNNs are also used with convolutional layers to expand the functional pixel neighborhood.

Disadvantages of RNN

a). It suffers from the Gradient vanishing and exploding problem .

b). Training of RNN is a quite difficult task.

c). Using activation functions like relu or tanh , It cannot process quite long sequences .

6) Long-Short-Term Memory(LSTM):

LSTM belong to the family of deep learning algorithms. It is a type of recurrent network as there are feedback connection in its architecture. It has capability to process the entire sequence which is an advantage over traditional networks. It is able to store past information that is important and it forgets the information that is not important, which make its working very well. LSTM architecture consist of the cell, input gate output gate and forget gate.

The three gates regulates the flow of info. In and out of the cell and the cell remembers value over arbitrary time intervals. The cell keeps the track of dependencies of input sequence elements. The input gate controls how much new value is added to the cell ,the forget gate control the extent to which a value remain in the cell and the output gate controls how much the value in the cell is used to compute the output activation of LSTM unit.

LSTM is very popular for it use on time series data for processing , classification and making prediction. The reason it is very popular for time series applications is because in a time series there can be several lags of unknown duration between important events.

3.2 Analysis:

Data Exploration:

The data used for this project is of Alphabet incorporation from Jan,2009 to 2021. This data is time series data and is maintained in oldest to newest form. Our goal was to predict the future closing price of the given stock data, after training. All data is downloaded from the Yahoo finance API for the ease of reusability and reproducibility.

We have to made prediction for the closing (Adjusted closing) price of data. Since Yahoo finance already adjusted the closing price for us, we just needed to predict the "CLOSE" price of a stock.

The downloaded dataset is of the following form:

Date	**Open**	**High**	**Low**	**Close**	**Volume**
2021-04-21	2285.25	2295.32	2258.57	2293.29	1196500
2021-04-22	2293.23	2303.762	2256.45	2267.92	1054800
2021-04-23	2283.47	2325.82	2278.21	2315.3	1433500

Table: The whole data can be found out in 'Google.csv' in the project root folder

Note: We did not observe any abnormality in datasets, i.e, no feature is empty and does not contains any incorrect value as negative values.

The mean, standard deviation, maximum and minimum of the dataset is as follows:

Feature	**Open**	**High**	**Low**	**Close**	**Volume**
Mean	726.0125	732.90134	719.2970	726.3473	3263475.6961
Std	449.6357	455.0209	445.2686	450.3963	2652420.3858
Max	2307.8898	2325.8200	2287.8449	2315.3000	29760734
Min	191.1584	193.7985	189.5395	191.1385	7922

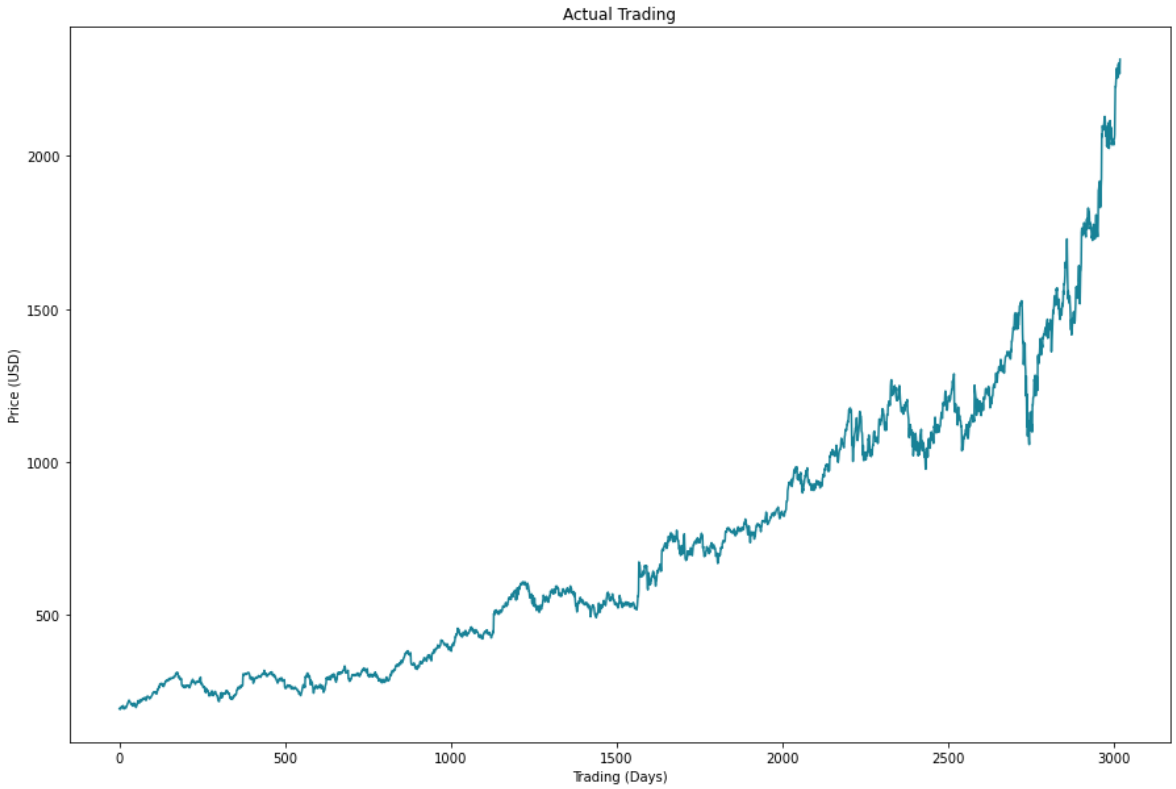
We can conclude from the dataset that high and low value are not the important features of the dataset. As for any day it does not matter what was the highest or what was the lowest price of a stock on any particular day. What matters is the what is the opening and closing price of a stock on any day. As if the closing price is greater than the opening price the we can earn profit otherwise it leads to the losses. Also volume is also the important feature, as for a rising market there is a rising volume means if with the increase in the price ,volume decreases it shows the lack of interest. This result in warning of potential reversal. Price drop on large volume shows that their is something fundamental changes in the stock.

Therefore we have removed Date, High and low features from data set at preprocessing step. The mean, standard deviation, maximum and minimum of the preprocessed data is as follows:

	Mean	**Std**	**Max**	**Min**
Open	0.2526	0.2124	0.9999	0.0
Close	0.2519	0.2120	1.0	0.0
Volume	0.1094	0.0891	0.9999	0.0

3.3 Visualization of dataset:

To visualize the data we have used matplotlib library .We have plotted closing price of the data with the no of days available. Below we can see the plotted data:



X-axis: Represents Tradings Days
Y-axis: Represents Closing Price In USD

From the above data we can se there is a continuous growth in Alphabet stock price. The major fall is in between 2700-2800.

4. Proposed Approach:

The Goal of this project is to study time-series data and explore as many options as to accurately predict the stock price. Through our research we came to know about various machine learning algorithms like linear regression , KNN,SVM etc which we can use for stock price prediction. But they are not as efficient which we will be showing through our implementation. After these we came to know about **Recurrent Neural Nets (**RNN) which are specifically used for pattern and sequence learning. But RNNs have vanishing gradient descent problem which does not allow it to learn from past data, which was expected. This problem was solved by **Long-Short Term MemoryNetworks** also known as LSTM. These are special king of RNN which are capable of learning long-term dependencies. Through the implementation of LSTM we will be seeing it improved result over other models.

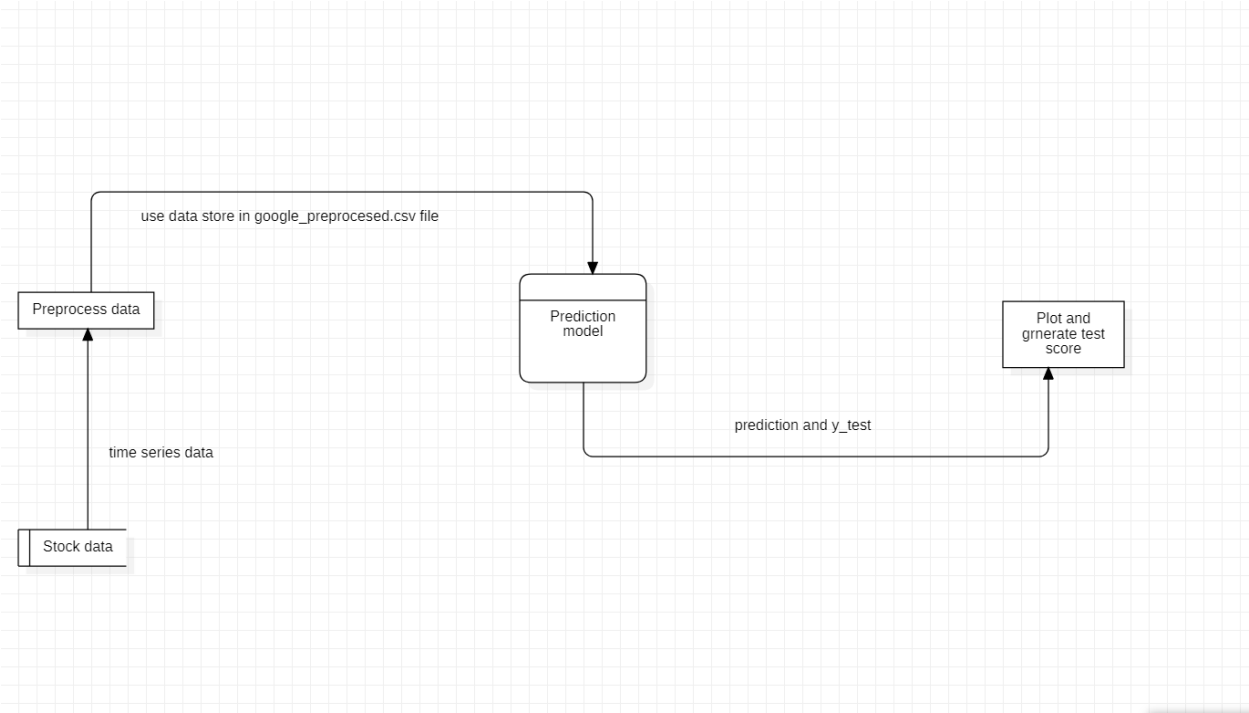
For implementing the architecture of neural Network and optimizing the model we can fine tune the following set of parameters:

- Input Parameters
    - Preprocessing and Normalization (see Data Preprocessing Section)
- Neural Network Architecture
    - Number of Layers (how many layers of nodes in the model; used 3)
    - Number of Nodes (how many nodes per layer; tested 1,3,8, 16, 32, 64, 100,128)
- Training Parameters
    - Training / Test Split (how much of dataset to train versus test model on; kept constant at 82.95% and 17.05% for benchmarks and lstm model)
    - Validation Sets (kept constant at 0.05% of training sets)

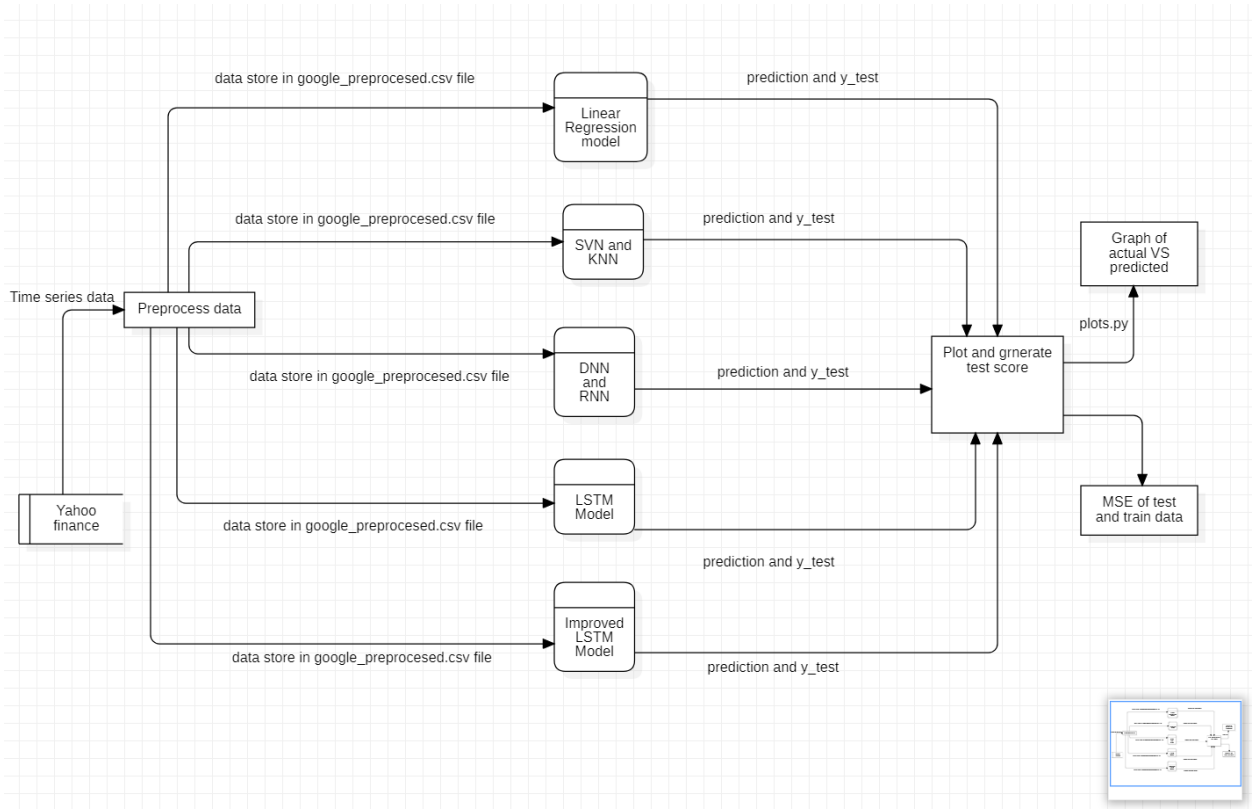
- Batch Size (how many time steps to include during a single training step; kept at for basic lstm model and at 512 for improved lstm model)
- Optimizer Function (which function to optimize by minimizing error; used “Adam” throughout)
- Epochs (how many times to run through the training process; kept at 1 for base model and at 20 for improved LSTM)

5. **DFD**

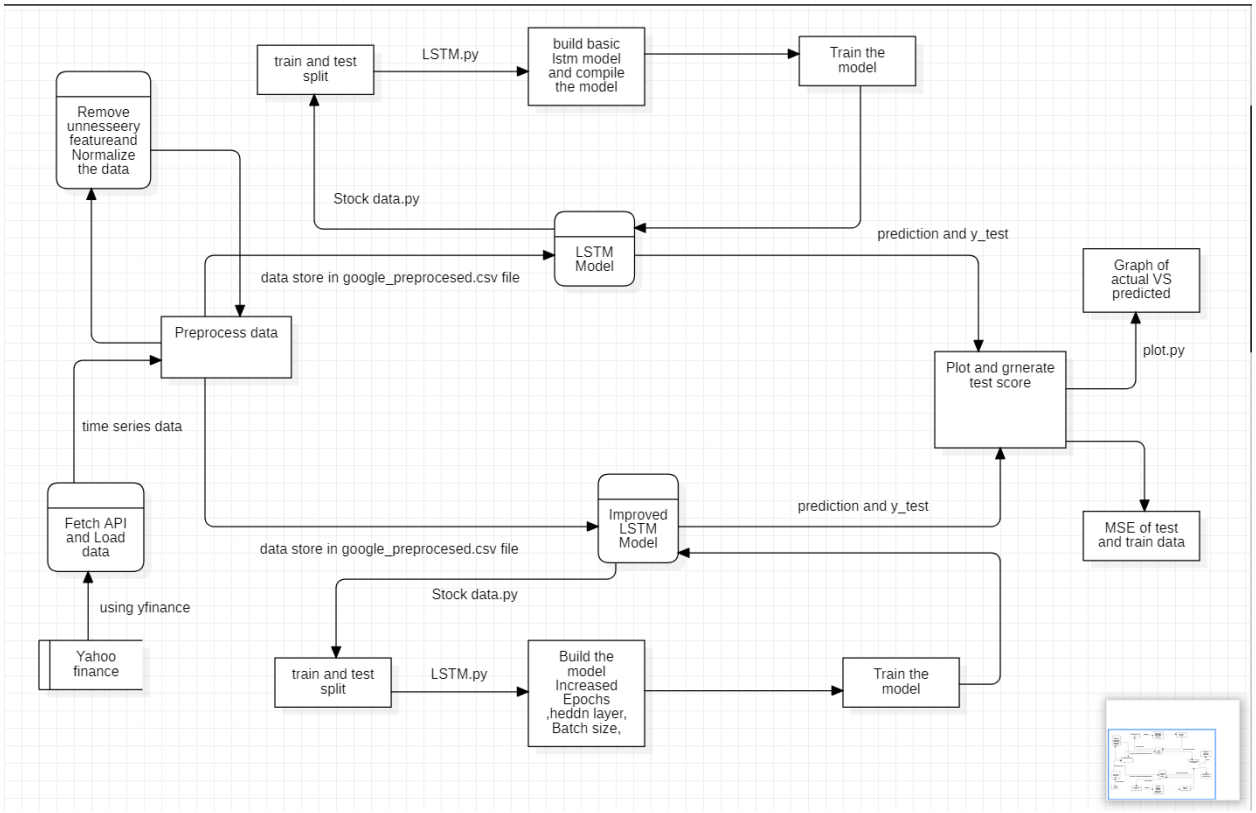
5.1 Level 0 DFD



5.2 Level 1 DFD:



5.3 Level 2 DFD



6. Implementation Details:

Starting with Set up of infrastructure:

- For this project we will be using jupyter Notebook which came with anaconda edition of python.
- Incorporate all required Libraries (Keras, Tensor flow, Pandas, Matplotlib, Sklearn, Numpy)
- We will be y finance for downloading data from yahoo finance API.

After Setting Infrastructure for our project we will be preparing dataset:

- At first we will be downloading the required data from yahoo finance using yfinance API. The downloaded data is of following format.
- After that we will remove the high and low feature from acquired data using data pre_process.py file. The data will be in the following format.
- After that we will normalize the data using seven different function like standard scalar, minmax scalar, Maxabs scalr etc. The plot from these functions is similar but resulted range of data are different. The MinMaxscalar give the data in range of 0 to 1 ,which we find useful as it can be used with different models easily.Following is the result from min max scalar.
- After normalizing the data we will store the data in google_preprocessed.csv file for future reusability and implementing different models.

After preparing the data we will be implementing different models using same data for predicting stock price and after that we will comparing their results using mean square error as metrices and see which model is performing better than the others

Following are the implementation detail of Different models:

1. **Linear Regression Model:**
    - At first we will load the data from google_preprocessed.csv file.
    - Split the data into test and train pair using Stock_data.py file. It will create the test(80%) and train dataset for linear regression model.

- After that we will call LinearResgressionModel.py file which will build the model for the project.
- Predict the price for test dataset using predict_price function in LinearResgressionModel.py
- Finally calculate the test score and plot the result of Linear Regression Model.

2. **Support Vector Machine (SVM) Model**
    - Split the data from google_preprosessed file into test and train dataset
    - Build the model
    - Plot the prediction and calculatr test error
3. **K-NearesrNeighbor (KNN) Model**
    - Split the data into test and train dataset
    - Build and train the model
    - Calculate rmse value and plot the prediction vs actual graph
4. **Deep Neural Network (DNN) Model**
    - Split the dataset into test and train dataset
    - Create a simple DNN Model with one hidden layer
    - Fit Model
    - Evaluate Model. Print the error metrics and plot the model result for train dataset.
5. **Recurrent Neural Network (RNN) Model**
    - Use the data from google_preprocesed.csv file and split the data in train and test dataset
    - Built a RNN Model with two hidden Layer.
    - Fit Model
    - Plot out error metrics and generate model loss plot
6. **Long-Short-Term Mmory (LSTM) Model**
    - Import keras libraries for smooth implementation of lstm.
    - Split train and test data sets and Unroll train and test data for lstm model using stock_data.py file.
    - Build a basic Long-Short Term Memory model using lstm.py file.
    - Train the model

- Make prediction using test data
- Plot the test result and get the test score

7. **Improve LSTM Model:**
    - Split into train and test model. Here same set of training and testing daset is used for improved LSTM which was used for basic LSTM model.
    - Build an improved LSTM Model by calling function defined in lstm.py file.This will create the improved LSTM Model for the project.The function uses keras long short term memory liabrary to implement LSTM Model.

      We have increased batch_size from 1 to 512 and epockhs from 1 to 20 for improved LSTM Model. We also increased the hidden layer from 100 to 128 and added a drop out of 0.2 to all the layers.
    - Train the model
    - Predict the price for the given datasets
    - calculate the test score and plot the results of improved LSTM Model

Refinement:

For improving the LSTM we had to fine tune the parameter of LSTM so that we can get better prediction. We did improvement by testing and analyzing each parameter and in the last we select the final value for each parameter.
For improving the LSTM we made the following changes:

- No. of hidden node increases from100 to 128.
- Batch size increases from 1 to 512.
- Dropout of 0.2 is added to the each layer of LSTM
- Epochs increased from 1 to 20
- Verbose added =2
- Prediction made with the batch size

The Mean square error is improved from 0.00604687 to 0.00331067 for training sets

7. Results & Observations:

With each model we have improved our prediction and reduced mean square error significantly

- **For Linear Regression Model**

  Train score: 0.2295 MSE (0.4790 RMSE)

Test Score: 0.16925723 MSE (0.41140883 RMSE)

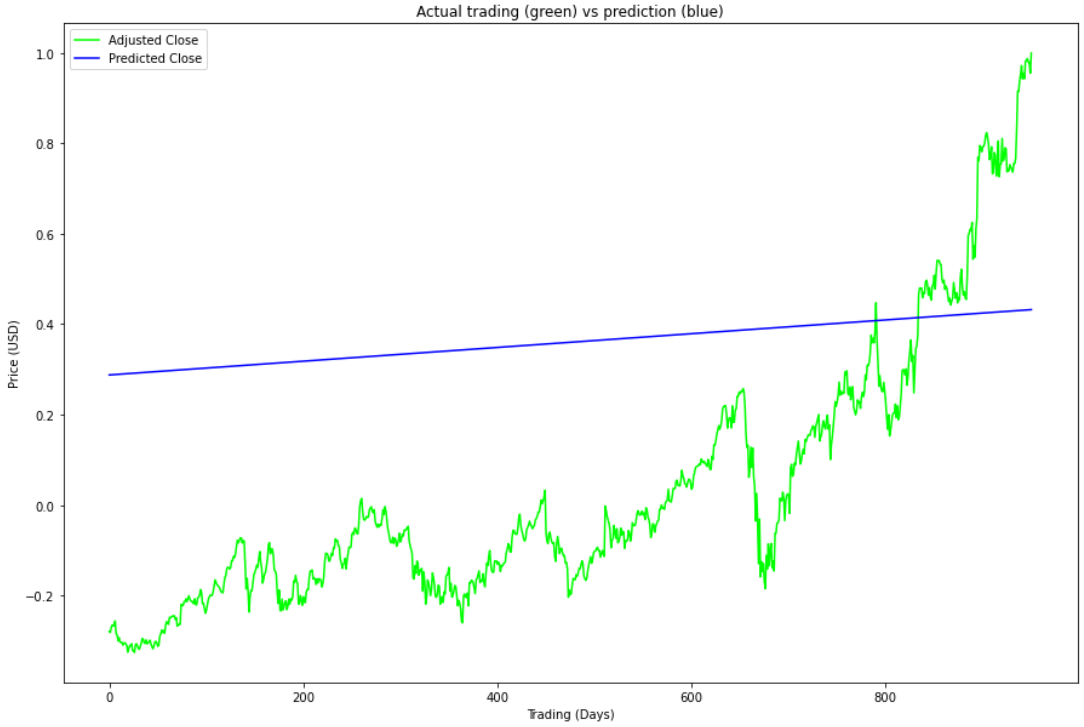


Fig : Plot for Linear Regression Model

- **For Support Vector Machine (SVM) Model**

  RMSE for test data: 0.6789937199116884

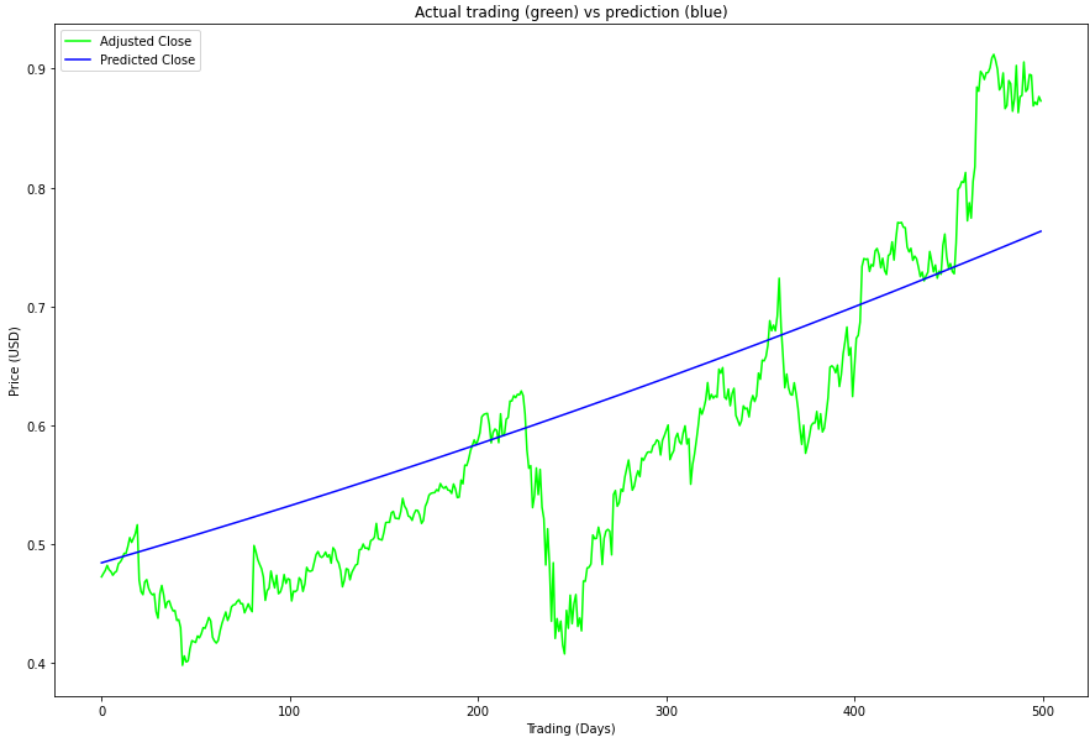


Fig : Plot for SVM Model

- **For k-Nearest Neighbor Classifier (KNN) Model:**

RMSE for train data: 0.23067884456887236

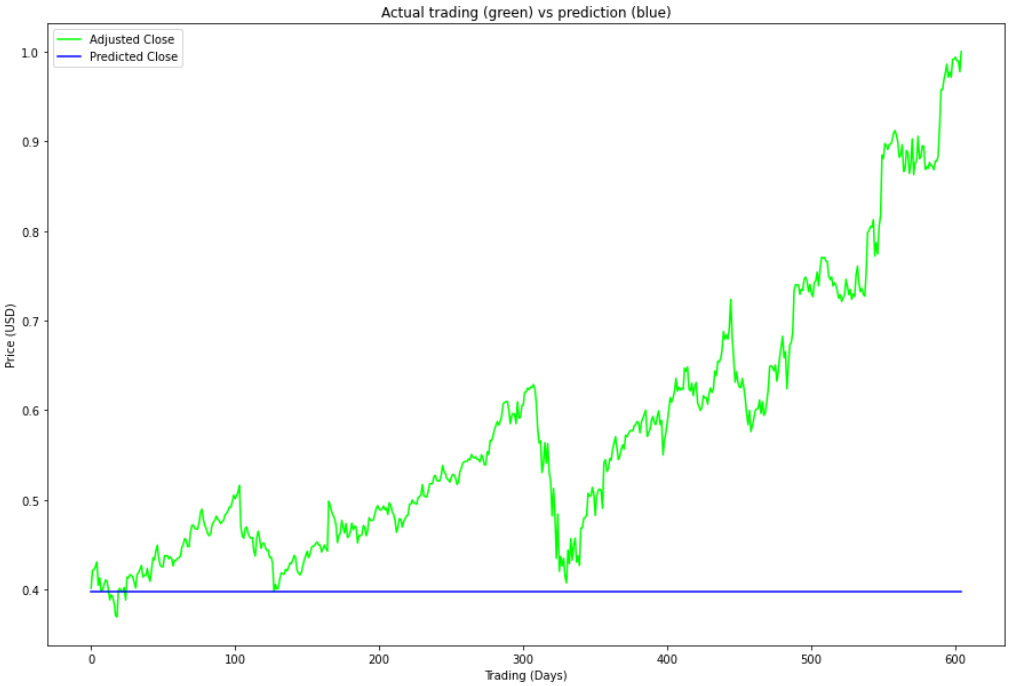


Fig : Plot for KNN Model

- **For Deep Neural Network (DNN)Model**

  RMSE for train data: 0.08

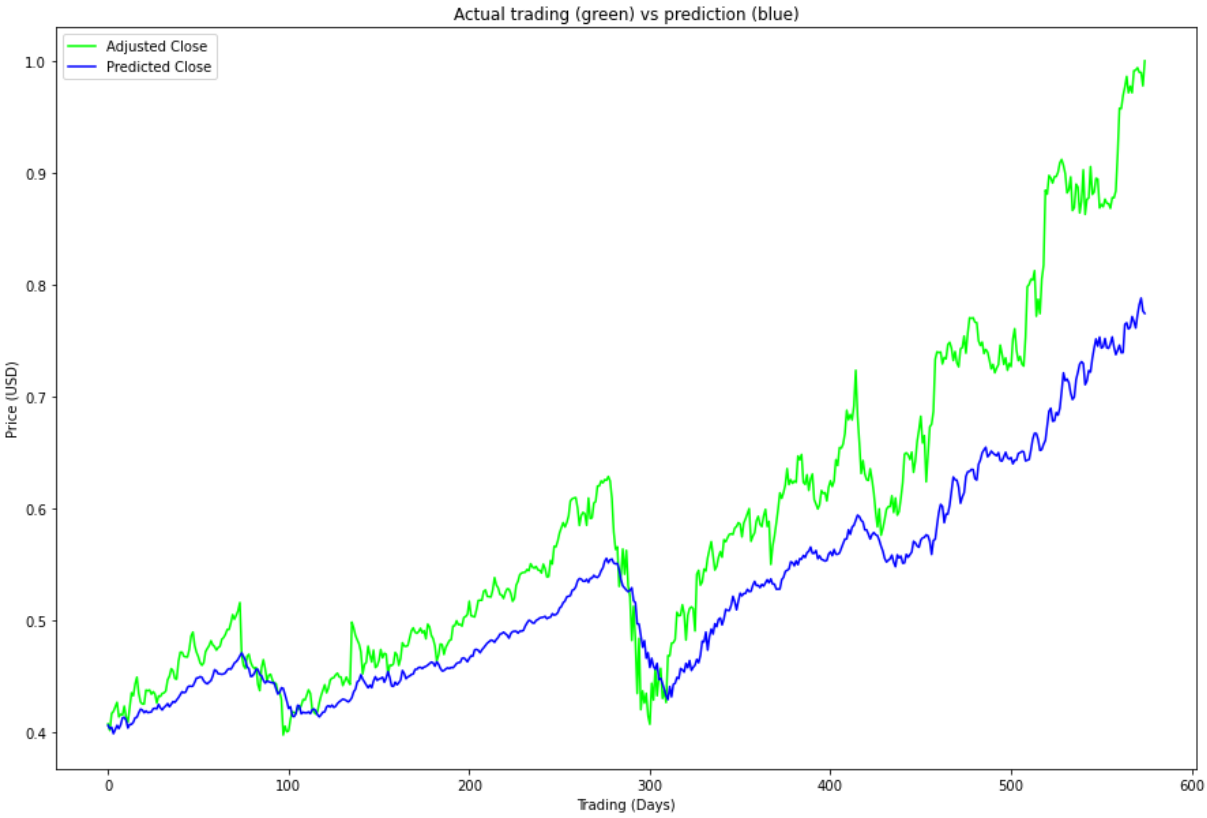


Fig : Plot for DNN Model

- **For Recurrent Neural Network Model**

  RMSE for train data: 0.03

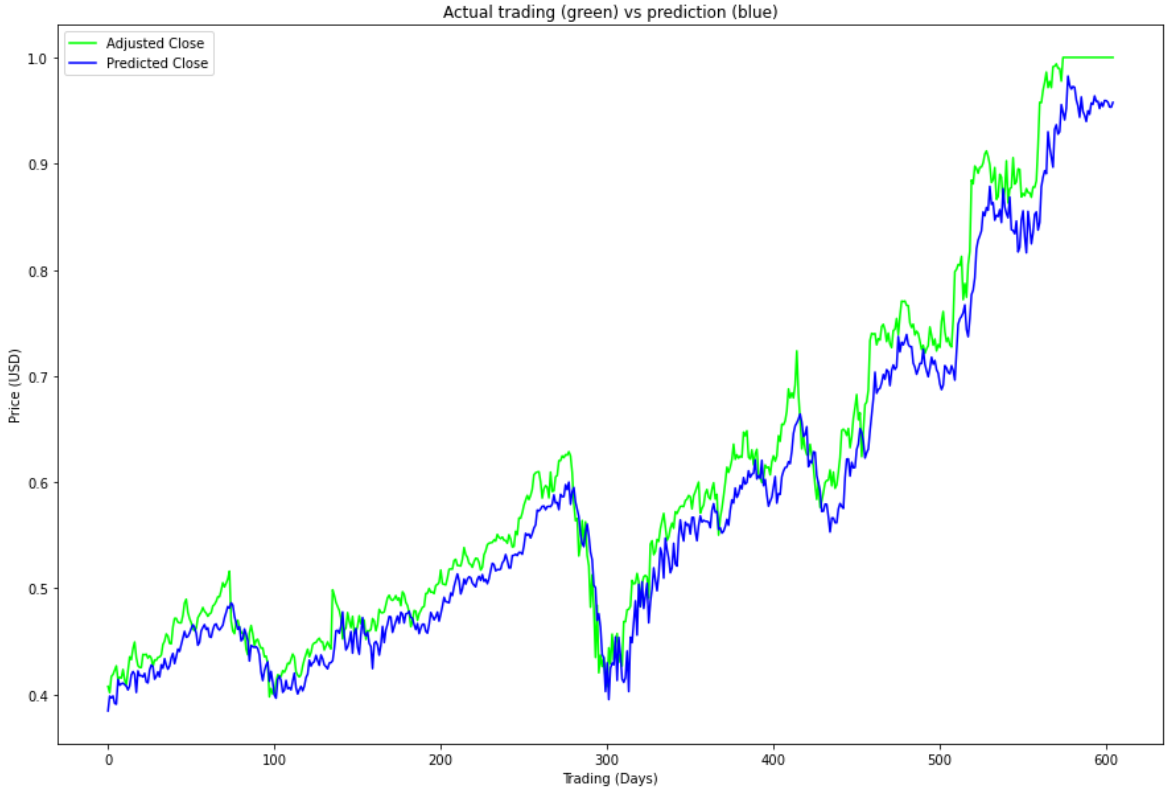


Fig : Plot for Basic RNN Model

- **For Long-Short-Term Memory(LSTM) Model**

  Train Score: 0.00025910 MSE (0.01609657 RMSE)

  Test Score: 0.00604687 MSE (0.07776165 RMSE)

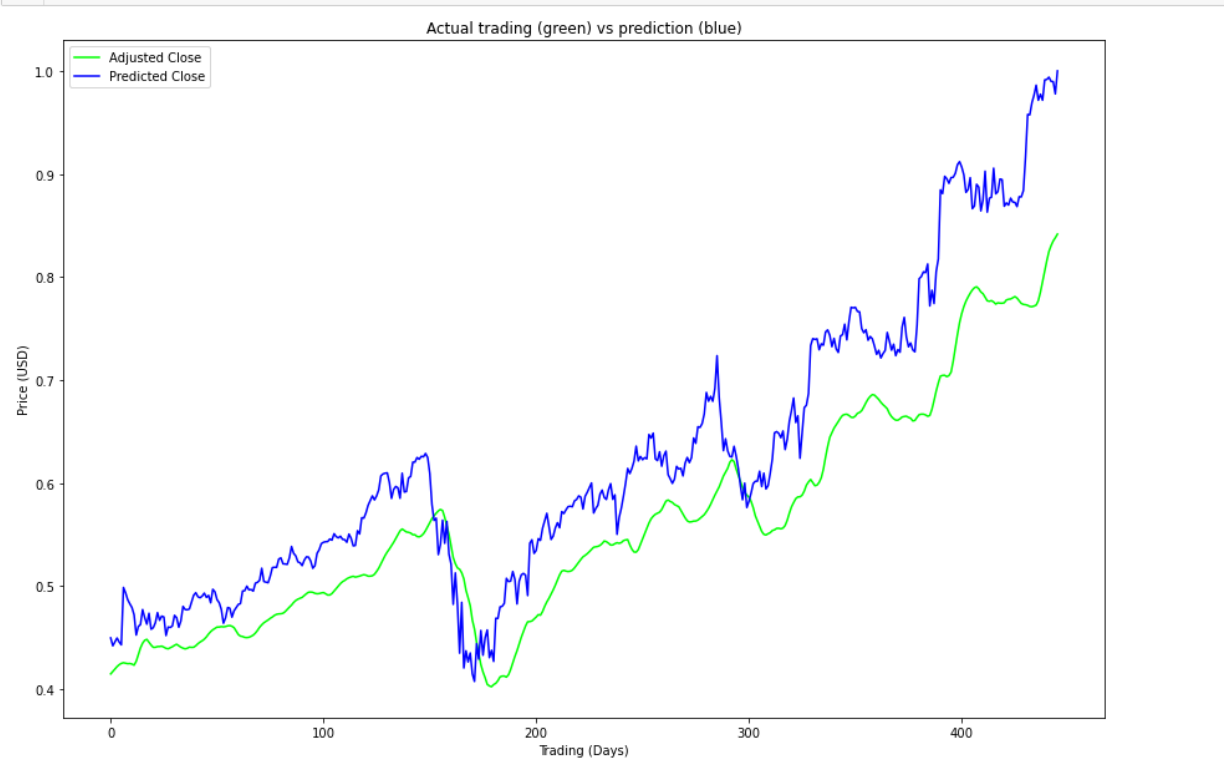


Fig : Plot for Basic LSTM Model

- For improved LSTM Model:

  Train Score: 0.00019930 MSE (0.01411734 RMSE)

  Test Score: 0.00331067 MSE (0.05753839 RMSE

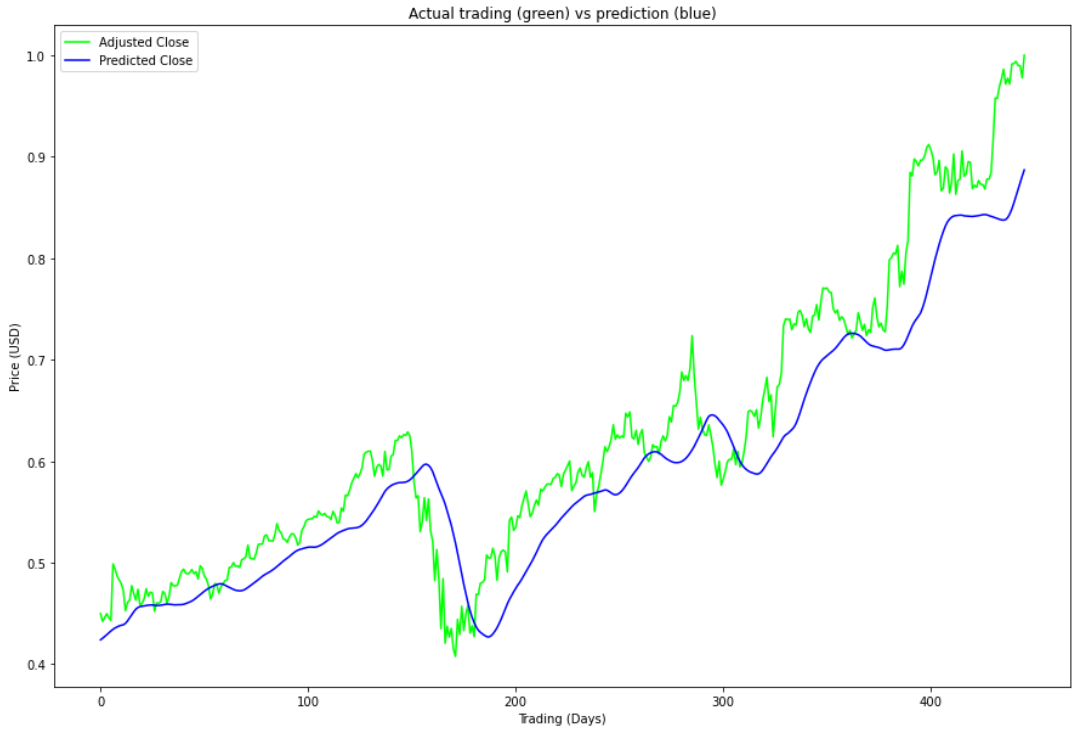


Fig : Plot for Improved LSTM Model

8. Conclusion & Future Plans:

In this project we tried to predict the future closing price of a stock using Technical analysis. For this we use various Machine Learning models. We started with linear regression model whose RMSE value was 0.4790 which is not so good. After that we get better result from RNN Model with rmse value 0.03 And in our last model(improved LSTM) we get the rmse value of 0.01411734 and MSE value 0.00019930 which is significant improvement over our previous models.

We started this project to learn about completely new algorithms like Long Short Term Memory(LSTM) and to explore real time series data. From above results we learned that how LSTM is most suitable for predicting stock prices for time series data as it produced least MSE

Although we get good result from above model but we can't completely depend on above model for predicting stock price as it does not include fundamental analysis factors. So we can still improve the model by adding the factors like textual analysis

So we can say that using advance deep learning models like LSTM we can get some picture of how the stock prices will act. But for getting the whole picture we should include other non technical factors as well.

For improving this project we can add following feature in future:

- For improving the result we can add other noon-technical factors for predicting stock price.
- There is no user interaction provide in this project. So We can definitely add an UI so that user can check the value of future dates.
- Here we used stock of Alphabet Incorporation only. We can surely add more companies in the list so that we can make this project more comprehensive.

9. References

[1] Anita Yadava, C K Jhaa , Aditi Sharan , Optimizing LSTM for time series prediction in Indian stock market,16 july,2020

[2] Elijah Joseph , Amit Mishra Forecast on Close Stock Market Prediction using Support Vector Machine (SVM), 2february 2019

[3] Guangyu Ding, Liangxi Qin, Study on the prediction of stock price based on the associated network model of LSTM, 30 november 2019, pp 1307-1317

[4] Hiransha M ,Gopalakrishnan E.A,Vijay Krishna Menon,Soman K.P, NSE Stock Market Prediction Using Deep Learning Models,2017,pp 1353-1361

[5] Murtaza Roondiwala1, Harshal Patel, Shraddha Varma, Predicting Stock Prices Using LSTM,4 April 2017,pp 1754-1756

[6] Khalid Alkhatib,Hassan Najadat2 ,Ismail Hmeidi ,Mohammed K. Ali Shatnawi , Stock Price Prediction Using *K*-Nearest Neighbor (*k*NN) Algorithm,2 March 2013

[7] colah's blog,Understanding LSTM Networks,August 27, 2015

[8] Vaibhav Kumar, Devlopers cornerHands-On Guide To LSTM Recurrent Neural Network For Stock Market Prediction, 27 March 2020

[9] Yang Lyla, TDS, A Quick Deep Learning Recipe: Time Series Forecasting with Keras in Python, 29 March 2020

Appendix A

Source code

data_preprocess.py

This python file contains the code for removing the unnecessary features and normalise the rest of the features using different scalar functions.

```
import pandas as pd
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import PowerTransformer


def delete_data(data):

    # parameter 'data' containing records of all the stock prices with columns as  ['Date','Open','Hi
gh','Low','Close','Volume']
    # returns a DataFrame with columns as  ['index','Open','Close','Volume']

    # Define columns to keep as it is
    item = list()
    open = list()
    close = list()
    volume = list()

    # Loop through the stock data and copy the required features
    index = 0
    for ind in range(len(data)):
        item.append(index)
        open.append(data['Open'][ind])
        close.append(data['Close'][ind])
        volume.append(data['Volume'][ind])
        index += 1

    # Create a data frame for stock data
    stocks = pd.DataFrame()

    # Assign fetures to data frame
    stocks['Item'] = item
```

```
    stocks['Open'] = open
    stocks['Close'] = pd.to_numeric(close)
    stocks['Volume'] = pd.to_numeric(volume)

    # return the new updated data
    return stocks


def normalize_data(data):

    # Initializing MinMax Scalar
    scaler = MinMaxScaler()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_StandardScalar(data):

    # Initialize a scaler, then apply it to the features
    scaler = StandardScaler()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_MinMaxScaler(data):

    # Initialize a scaler, then apply it to the features
    scaler = MinMaxScaler()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_MaxAbsScaler(data):
```

```
    # Initialize a scaler, then apply it to the features
    scaler = MaxAbsScaler()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_RobustScaler(data):

    # Initialize a scaler, then apply it to the features
    scaler = RobustScaler()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_Normalizer(data):

    # Initialize a scaler, then apply it to the features
    scaler = Normalizer()
    numerical = ['Open', 'Close', 'Volume']
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_QuantileTransformer(data):

    # Initialize a scaler, then apply it to the features
    scaler = QuantileTransformer()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data


def get_normalised_data_PowerTransformer(data):
```

```
    # Initialize a scaler, then apply it to the features
    scaler = PowerTransformer()
    numerical = ['Open', 'Close', 'Volume']

    # Applying scalar to features
    data[numerical] = scaler.fit_transform(data[numerical])

    return data
```

stock_data.py

This python file contain the code for splitting, scaling and unrolling the dataset for different models.

```
import numpy as np
import math


def scale_range(x, input_range, target_range):

    range = [np.amin(x), np.amax(x)]
    x_std = (x - input_range[0]) / (1.0*(input_range[1] - input_range[0]))
    x_scaled = x_std * \
        (1.0*(target_range[1] - target_range[0])) + target_range[0]
    return x_scaled, range


def train_test_split_linear_regression(stocks):
    # Create numpy arrays for features and targets
    feature = []
    label = []

    # Convert dataframe columns to numpy arrays for scikit learn
    for index, row in stocks.iterrows():
        # print([np.array(row['Item'])])
        feature.append([(row['Item'])])
        label.append([(row['Close'])])

    # Regularize the feature and target arrays and store min/max of input data for rescaling later
    feature_bounds = [min(feature), max(feature)]
    feature_bounds = [feature_bounds[0][0], feature_bounds[1][0]]
    label_bounds = [min(label), max(label)]
    label_bounds = [label_bounds[0][0], label_bounds[1][0]]
```

```
    feature_scaled, feature_range = scale_range(
        np.array(feature), input_range=feature_bounds, target_range=[-1.0, 1.0])
    label_scaled, label_range = scale_range(
        np.array(label), input_range=label_bounds, target_range=[-1.0, 1.0])

    # Define Test/Train Split 80/20
    split = .315
    split = int(math.floor(len(stocks['Item']) * split))

    # Set up training and test sets
    X_train = feature_scaled[:-split]
    X_test = feature_scaled[-split:]

    y_train = label_scaled[:-split]
    y_test = label_scaled[-split:]

    return X_train, X_test, y_train, y_test, label_range


def train_test_split_lstm(stocks, prediction_time=1, test_data_size=450, unroll_length=50):
    # training data
    test_data_cut = test_data_size + unroll_length + 1

    x_train = stocks[0:-prediction_time - test_data_cut].to_numpy()
    y_train = stocks[prediction_time:-test_data_cut]['Close'].to_numpy()

    # test data
    x_test = stocks[0 - test_data_cut:-prediction_time].to_numpy()
    y_test = stocks[prediction_time - test_data_cut:]['Close'].to_numpy()

    return x_train, x_test, y_train, y_test


def unroll(data, sequence_length=24):
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
    return np.asarray(result)
```

LinearRegressionModel.py

This python file contains the code for builing the linear regression model and predicting the price.

```
import numpy as np
import stock_data as sd
from sklearn import linear_model


def price_prediction(lr_model, feature, range):
    x = np.reshape(feature, (feature.shape[0], 1))  # test features
    predicted_price = lr_model.predict(x)
    scaled_prediction, _ = sd.scale_range(
        predicted_price, input_range=[-1.0, 1.0], target_range=range)

    return scaled_prediction.flatten()


def build_model(X, y):
    # instantiating linear regression model
    model = linear_model.LinearRegression()
    X = np.reshape(X, (X.shape[0], 1))  # feature dataset
    y = np.reshape(y, (y.shape[0], 1))  # label dataset
    model.fit(X, y)  # model fitting
    return model
```

knnModel.py

This python file contains the code for builing the K-Nearest Neighbour model and predicting the price.

```
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV

def build_model():
    # instantiating knn model
    #using gridsearch to find the best parameter
    params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
    knn = neighbors.KNeighborsRegressor()
    model = GridSearchCV(knn, params, cv=5)
    return model
```

dnnModel.py

This python file contains the code for builing the Deep Neural Network model and predicting the price.

```
from keras.models import Sequential
from keras.layers import Dense

import matplotlib.pyplot as plt

def build_model():
    # instantiating dnn model
    # setup look_back window
    look_back = 30

    model=Sequential()
    model.add(Dense(units=32, input_dim=look_back, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error',  optimizer='adam',metrics = ['mse', 'mae'])
    return model

def model_loss_plot(history):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Test Loss')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(loc='upper right')
    plt.show();
```

rnnModel.py

This python file contains the code for builing the Recurrent Neural Network model and predicting the price.

```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

def build_model():
    # instantiating rnn model
    #create window size as look_back=30
    look_back = 30

    model=Sequential()
    model.add(SimpleRNN(units=32, input_shape=(1,look_back), activation="relu"))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error',  optimizer='adam',metrics = ['mse', 'mae'])
    return model

def model_loss_plot(history):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Test Loss')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(loc='upper right')
    plt.show();
```

lstm.py

This python file contains the code for builing the Long Short Term Memory model and predicting the price.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM


def build_basic_model(input_dim, output_dim, return_sequences):
    model = Sequential()
    model.add(LSTM(
        units=output_dim,
        input_shape=(None, input_dim),
        return_sequences=return_sequences))

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

    model.add(Dense(
        units=1))
    model.add(Activation('linear'))

    return model


def build_improved_model(input_dim, output_dim, return_sequences):
    model = Sequential()
    model.add(LSTM(
        units=output_dim,
        input_shape=(None, input_dim),
        return_sequences=return_sequences))

    model.add(Dropout(0.2))

    model.add(LSTM(
        128,
        return_sequences=False))

    model.add(Dropout(0.2))
```

```
    model.add(Dense(
        units=1))
    model.add(Activation('linear'))

    return model
```

plots.py

This python file contains the code for plotting the actual vs prediction graph for different models.

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15, 10)


def price(x):
    # returns the price upto 2 decimal digits and dollar sign added in the beginning
    return '$%1.2f' % x


def prediction_plot(actual, prediction):
    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Add labels
    plt.ylabel('Price (USD)')  # depicts the y-axis label
    plt.xlabel('Trading (Days)')  # depicts the x-axis label

    # Plot actual and predicted close values

    plt.plot(actual, '#00FF00', label='Adjusted Close')
    plt.plot(prediction, '#0000FF', label='Predicted Close')

    # Sets title of the plot
    ax.set_title('Actual trading (green) vs prediction (blue)')
    ax.legend(loc='upper left')

    # Prints the plot against stock and its closing value
    plt.show()


def basic_plot(stocks):
```

```
    fig, ax = plt.subplots()
    ax.plot(stocks['Item'], stocks['Close'], '#178196')

    ax.format_ydata = price
    ax.set_title('Actual Trading')

    # Add labels
    plt.ylabel('Price (USD)')  # depicts the y-axis label
    plt.xlabel('Trading (Days)')  # depicts the x-axis label

    # Prints the plot against stock and its closing value
    plt.show()


def lstm_prediction_plot(actual, prediction):
    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Add labels
    plt.ylabel('Price (USD)')  # depicts the y-axis label
    plt.xlabel('Trading (Days)')  # depicts the x-axis label

    plt.plot(actual, '#00FF00', label='Adjusted Close')
    plt.plot(prediction, '#0000FF', label='Predicted Close')

    # Sets title of the plot
    ax.set_title('Actual trading (green) vs prediction (blue)')
    ax.legend(loc='upper left')

    # Prints the plot against stock and its closing value
    plt.show()
```

stock_price_predictor.ipynb

In this ipynb file, we fetch the data from yfinance api, preprocess the fetched data and implement the different machine learning and deep learning model for predicting stock price using RMSE as metrics for comparison.

```
#!/usr/bin/env python
# coding: utf-8

# # Stock price predictor
#

# ## Fetch the Data
#
#
#    **Step 1 :** Function to get historical data from yahoo finance

# In[1]:


import pandas as pd
import yfinance as yf
def get_stock_data(stock_code, period):
    alphabet = yf.Ticker(stock_code)
    col_names = ['Date','Open','High','Low','Close','Volume']
    #df = pd.DataFrame(yf.download("GOOG",start = "2005-01-01",end = "2017-06-30"))
    df = pd.DataFrame(alphabet.history(period='12y'))
    df = df.filter(col_names)
    return df


#  **Step 2:** get the stock data of desired firm from [Yahoo Finance](https://in.finance.yahoo.com/).

# In[2]:


data = get_stock_data('GOOG','12y') #Last 12 years
print(data)


# **Step 3:** Write the data to a csv file.

# In[3]:
```

```
#data.to_csv('google.csv',index = False)


# ## Preprocess the data
#
# Now we will preprocess the data i.e removing unneccessary features and normalising the rest.
#
# **Step 1 :** Import necessary libraries

# In[4]:


import pandas as pd
import numpy as np
print(data.head())

print("\n")
print("Open   ---
 mean :", np.mean(data['Open']),  "  \t Std: ", np.std(data['Open']),  "  \t Max: ", np.max(data['Ope
n']),  "  \t Min: ", np.min(data['Open']))
print("High   ---
 mean :", np.mean(data['High']),  "  \t Std: ", np.std(data['High']),  "  \t Max: ", np.max(data['High
']),  "  \t Min: ", np.min(data['High']))
print("Low    ---
 mean :", np.mean(data['Low']),   "  \t Std: ", np.std(data['Low']),   "  \t Max: ", np.max(data['Low
']),   "  \t Min: ", np.min(data['Low']))
print("Close  ---
 mean :", np.mean(data['Close']), "  \t Std: ", np.std(data['Close']), "  \t Max: ", np.max(data['Clos
e']), "  \t Min: ", np.min(data['Close']))
print("Volume ---
 mean :", np.mean(data['Volume']),"  \t Std: ", np.std(data['Volume']),"  \t Max: ", np.max(data['
Volume']),"  \t Min: ", np.min(data['Volume']))


# **Step 2 :** Remove data that is no longer required i.e Date and High value

# In[5]:


import data_preprocess as dpp
stocks = dpp.delete_data(data)

#Print the dataframe head and tail
print(stocks.head())
print("---")
print(stocks.tail())
```

In[6]:

```
import plots

plots.basic_plot(stocks)
```

Step 3 : Normalise the data using different scalar function

In[7]:

```
import copy
Stocks1=copy.copy(stocks)
Stocks1 = dpp.get_normalised_data_StandardScalar(Stocks1)
print(Stocks1.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks1['Open']),  "  \t Std: ", np.std(Stocks1['Open']),  "  \t Max: ", np.max(St
ocks1['Open']),  "  \t Min: ", np.min(Stocks1['Open']))
print("Close  ---
 mean :", np.mean(Stocks1['Close']), "  \t Std: ", np.std(Stocks1['Close']), "  \t Max: ", np.max(St
ocks1['Close']), "  \t Min: ", np.min(Stocks1['Close']))
print("Volume ---
 mean :", np.mean(Stocks1['Volume']),"  \t Std: ", np.std(Stocks1['Volume']),"  \t Max: ", np.max
(Stocks1['Volume']),"  \t Min: ", np.min(Stocks1['Volume']))
print("\n\n")
Stocks2=copy.copy(Stocks1)


Stocks3=copy.copy(Stocks2)
Stocks3 = dpp.get_normalised_data_MaxAbsScaler(Stocks3)
print(Stocks3.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks3['Open']),  "  \t Std: ", np.std(Stocks3['Open']),  "  \t Max: ", np.max(St
ocks3['Open']),  "  \t Min: ", np.min(Stocks3['Open']))
print("Close  ---
 mean :", np.mean(Stocks3['Close']), "  \t Std: ", np.std(Stocks3['Close']), "  \t Max: ", np.max(St
ocks3['Close']), "  \t Min: ", np.min(Stocks3['Close']))
print("Volume ---
 mean :", np.mean(Stocks3['Volume']),"  \t Std: ", np.std(Stocks3['Volume']),"  \t Max: ", np.max
(Stocks3['Volume']),"  \t Min: ", np.min(Stocks3['Volume']))
```

```
print("\n\n")
Stocks4=copy.copy(Stocks3)
Stocks4 = dpp.get_normalised_data_RobustScaler(Stocks4)
print(Stocks4.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks4['Open']),  "  \t Std: ", np.std(Stocks4['Open']),  "  \t Max: ", np.max(St
ocks4['Open']),  "  \t Min: ", np.min(Stocks4['Open']))
print("Close  ---
 mean :", np.mean(Stocks4['Close']), "  \t Std: ", np.std(Stocks4['Close']), "  \t Max: ", np.max(St
ocks4['Close']), "  \t Min: ", np.min(Stocks4['Close']))
print("Volume ---
 mean :", np.mean(Stocks4['Volume']),"  \t Std: ", np.std(Stocks4['Volume']),"  \t Max: ", np.max
(Stocks4['Volume']),"  \t Min: ", np.min(Stocks4['Volume']))
print("\n\n")
Stocks5=copy.copy(Stocks4)
Stocks5 = dpp.get_normalised_data_Normalizer(Stocks5)
print(Stocks5.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks5['Open']),  "  \t Std: ", np.std(Stocks5['Open']),  "  \t Max: ", np.max(St
ocks5['Open']),  "  \t Min: ", np.min(Stocks5['Open']))
print("Close  ---
 mean :", np.mean(Stocks5['Close']), "  \t Std: ", np.std(Stocks5['Close']), "  \t Max: ", np.max(St
ocks5['Close']), "  \t Min: ", np.min(Stocks5['Close']))
print("Volume ---
 mean :", np.mean(Stocks5['Volume']),"  \t Std: ", np.std(Stocks5['Volume']),"  \t Max: ", np.max
(Stocks5['Volume']),"  \t Min: ", np.min(Stocks5['Volume']))
print("\n\n")
Stocks6=copy.copy(Stocks5)
Stocks6 = dpp.get_normalised_data_QuantileTransformer(Stocks6)
print(Stocks6.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks6['Open']),  "  \t Std: ", np.std(Stocks6['Open']),  "  \t Max: ", np.max(St
ocks6['Open']),  "  \t Min: ", np.min(Stocks6['Open']))
print("Close  ---
 mean :", np.mean(Stocks6['Close']), "  \t Std: ", np.std(Stocks6['Close']), "  \t Max: ", np.max(St
ocks6['Close']), "  \t Min: ", np.min(Stocks6['Close']))
print("Volume ---
 mean :", np.mean(Stocks6['Volume']),"  \t Std: ", np.std(Stocks6['Volume']),"  \t Max: ", np.max
(Stocks6['Volume']),"  \t Min: ", np.min(Stocks6['Volume']))
print("\n\n")
Stocks7=copy.copy(Stocks6)
Stocks7 = dpp.get_normalised_data_PowerTransformer(Stocks7)
```

```
print(Stocks7.head())
print("\n")
print("Open   ---
 mean :", np.mean(Stocks7['Open']),  "  \t Std: ", np.std(Stocks7['Open']),  "  \t Max: ", np.max(St
ocks7['Open']),  "  \t Min: ", np.min(Stocks7['Open']))
print("Close  ---
 mean :", np.mean(Stocks7['Close']), "  \t Std: ", np.std(Stocks7['Close']), "  \t Max: ", np.max(St
ocks7['Close']), "  \t Min: ", np.min(Stocks7['Close']))
print("Volume ---
 mean :", np.mean(Stocks7['Volume']),"  \t Std: ", np.std(Stocks7['Volume']),"  \t Max: ", np.max
(Stocks7['Volume']),"  \t Min: ", np.min(Stocks7['Volume']))
print("\n\n")
stocks = dpp.normalize_data(stocks)
print(stocks.head())
print("\n")
print("Open   ---
 mean :", np.mean(stocks['Open']),  "  \t Std: ", np.std(stocks['Open']),  "  \t Max: ", np.max(stock
s['Open']),  "  \t Min: ", np.min(stocks['Open']))
print("Close  ---
 mean :", np.mean(stocks['Close']), "  \t Std: ", np.std(stocks['Close']), "  \t Max: ", np.max(stocks
['Close']), "  \t Min: ", np.min(stocks['Close']))
print("Volume ---
 mean :", np.mean(stocks['Volume']),"  \t Std: ", np.std(stocks['Volume']),"  \t Max: ", np.max(sto
cks['Volume']),"  \t Min: ", np.min(stocks['Volume']))


# **Step 4 :** Visualize the data again

# In[8]:


plots.basic_plot(Stocks1)
plots.basic_plot(Stocks2)
plots.basic_plot(Stocks3)
plots.basic_plot(Stocks4)
plots.basic_plot(Stocks5)
plots.basic_plot(Stocks6)
plots.basic_plot(Stocks7)
plots.basic_plot(stocks)
print(stocks.head())


# **Step 5:** Log the normalised data for future resuablilty

# In[9]:
```

```
stocks.to_csv('google_preprocessed.csv',index= False)


#
#
# ### Machine Learning Models
#
# In the following section we will implement various learning models for predicting stock prices.
#
# #### Linear Regression Model

# **Step 1:** Load the preprocessed data

# In[10]:


import math
import pandas as pd
import numpy as np
from IPython.display import display
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit

import plots as plts
import stock_data as sd
import LinearRegressionModel

stocks = pd.read_csv('google_preprocessed.csv')
display(stocks.head())


# **Step 2:** Split data into train and test pair

# In[11]:


X_train, X_test, y_train, y_test, label_range= sd.train_test_split_linear_regression(stocks)

print("x_train", X_train.shape)
print("y_train", y_train.shape)
print("x_test", X_test.shape)
print("y_test", y_test.shape)
```

Step 3: Train a Linear regressor model on training set and get prediction

In[12]:

```
model = LinearRegressionModel.build_model(X_train,y_train)
```

Step 4: Get prediction on test set

In[13]:

```
predictions = LinearRegressionModel.price_prediction(model,X_test, label_range)
```

Step 5: Plot the predicted values against actual

In[14]:

```
plts.prediction_plot(y_test,predictions)
```

Step 6: measure accuracy of the prediction

In[15]:

```
trainScore = mean_squared_error(X_train, y_train)
print('Train Score: %.4f MSE (%.4f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = mean_squared_error(predictions, y_test)
print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))
```

SVM Model

In[16]:

```
#svm model
import stock_data as sd
from sklearn import svm
```

```
X_train, X_test, y_train, y_test = sd.train_test_split_ml(stocks,0.80)
clf = svm.SVR(kernel='poly')
X_train=X_train.reshape(-1,1)
X_test=X_test.reshape(-1,1)

clf.fit(X_train, y_train) #train
accuracy = clf.score(X_test, y_test) #test Accuracy squared error for linreg
prtt = clf.predict(X_test)

print(accuracy)
plots.lstm_prediction_plot(y_test,prtt)


# ### KNN Model

# In[17]:


#KNN Model
import stock_data as sd
import knnModel

X_train, X_test, y_train, y_test = sd.train_test_split_ml(stocks,0.80)

X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)

model = knnModel.build_model()

#fit the model and make predictions
model.fit(X_train,y_train)
preds = model.predict(X_test)

rms=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(preds)),2)))
print(rms)
plots.lstm_prediction_plot(y_test,preds)


# ## Deep Learning Models :-
#
# ### DNN Model

# In[18]:


#DNN Model
```

```
import stock_data as sd
import dnnModel

trainX, trainY, testX, testY = sd.train_test_split_dnn(stocks,0.80)

model=dnnModel.build_model()
past=model.fit(trainX,trainY, epochs=25, batch_size=30, verbose=1, validation_data=(testX,test
Y),shuffle=False)

train_score = model.evaluate(trainX, trainY, verbose=0)


# #### DNN Model Loss Plot

# In[19]:


print('Train Root Mean Squared Error(RMSE): %.2f; Train Mean Absolute Error(MAE) : %.2f '
% (np.sqrt(train_score[1]), train_score[2]))
test_score = model.evaluate(testX, testY, verbose=0)
print('Test Root Mean Squared Error(RMSE): %.2f; Test Mean Absolute Error(MAE) : %.2f '
% (np.sqrt(test_score[1]), test_score[2]))
dnnModel.model_loss_plot(past)


# #### DNN Model Prediction Plot

# In[20]:


plots.lstm_prediction_plot(testY, model.predict(testX))


# ### RNN Model

# In[21]:


#RNN Model
import stock_data as sd
from sklearn.metrics import mean_absolute_error
import rnnModel

trainX, trainY, testX, testY = sd.train_test_split_rnn(stocks,0.80)

# reshape input to be [samples, window size, features]
```

```
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

model=rnnModel.build_model()
past=model.fit(trainX,trainY, epochs=25, batch_size=30, verbose=1, validation_data=(testX,test
Y),shuffle=False)

train_predict = model.predict(trainX)
test_predict = model.predict(testX)


# #### RNN Model Loss Plot

# In[22]:


print('Train Root Mean Squared Error(RMSE): %.2f; Train Mean Absolute Error(MAE) : %.2f '
      % (np.sqrt(mean_squared_error(trainY, train_predict[:,0])), mean_absolute_error(trainY, trai
n_predict[:,0])))
print('Test Root Mean Squared Error(RMSE): %.2f; Test Mean Absolute Error(MAE) : %.2f '
      % (np.sqrt(mean_squared_error(testY, test_predict[:,0])), mean_absolute_error(testY, test_pr
edict[:,0])))
rnnModel.model_loss_plot(past)

testY=testY.flatten()
test_predict=test_predict.flatten()


# #### RNN Model Prediction Plot

# In[23]:


plots.lstm_prediction_plot(testY, test_predict)


#
# ## Long-Sort Term Memory Model
#
# In this section we will use LSTM to train and test on our data set.

# ### Basic LSTM Model
#
# First lets make a basic LSTM model.

# **Step 1 :** import keras libraries for smooth implementaion of lstm
```

In[24]:

```
import math
import pandas as pd
import numpy as np
from IPython.display import display

from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.metrics import mean_squared_error
from sklearn.model_selection import StratifiedKFold

import lstm, time #helper libraries

import plots as plts
import stock_data as sd
import LinearRegressionModel

stocks = pd.read_csv('google_preprocessed.csv')
stocks_data = stocks.drop(['Item'], axis =1)

display(stocks_data.head())
```

Step 2 : Split train and test data sets and Unroll train and test data for lstm model

In[25]:

```
X_train, X_test,y_train, y_test = sd.train_test_split_lstm(stocks_data, 5)

unroll_length = 50
X_train = sd.unroll(X_train, unroll_length)
X_test = sd.unroll(X_test, unroll_length)
y_train = y_train[-X_train.shape[0]:]
y_test = y_test[-X_test.shape[0]:]

print("x_train", X_train.shape)
print("y_train", y_train.shape)
print("x_test", X_test.shape)
print("y_test", y_test.shape)
```

Step 3 : Build a basic Long-Short Term Memory model

In[26]:

```
# build basic lstm model
model = lstm.build_basic_model(input_dim = X_train.shape[-
1],output_dim = unroll_length, return_sequences=True)

# Compile the model
start = time.time()
model.compile(loss='mean_squared_error', optimizer='adam')
print('compilation time : ', time.time() - start)
```

Step 4: Train the model

In[27]:

```
model.fit(
    X_train,
    y_train,
    batch_size=1,
    epochs=1,
    validation_split=0.05)
```

Step 5: make prediction using test data

In[28]:

```
predictions = model.predict(X_test)
```

Step 6: Plot the results

In[29]:

```
plts.lstm_prediction_plot(y_test,predictions)
#plts.lstm_prediction_plot(predictions,y_test)
```

In[30]:

```
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.8f MSE (%.8f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))


# # Improved LSTM Model
# *Step 1:  Build an improved LSTM model*

# In[31]:


# Set up hyperparameters
batch_size = 100
epochs = 5

# build improved lstm model
model = lstm.build_improved_model( X_train.shape[-
1],output_dim = unroll_length, return_sequences=True)

start = time.time()
#final_model.compile(loss='mean_squared_error', optimizer='adam')
model.compile(loss='mean_squared_error', optimizer='adam')
print('compilation time : ', time.time() - start)


# In[32]:


model.fit(X_train,
          y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=2,
          validation_split=0.05
         )


# In[33]:


# Generate predictions
predictions = model.predict(X_test, batch_size=batch_size)
```

```
# In[34]:


plts.lstm_prediction_plot(y_test,predictions)


# In[35]:


trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.8f MSE (%.8f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))


# In[36]:


range = [np.amin(stocks_data['Close']), np.amax(stocks_data['Close'])]

#Calculate the stock price delta in $

true_delta = testScore*(range[1]-range[0])
print('Delta Price: %.6f - RMSE * Adjusted Close Range' % true_delta)


# In[37]:


import data_preprocess as dpp

data = pd.read_csv('googl.csv')

stocks = dpp.delete_data(data)

stocks = dpp.normalize_data(stocks)

stocks = stocks.drop(['Item'], axis = 1)
#Print the dataframe head and tail
print(stocks.head())

X = stocks[:].to_numpy()
Y = stocks[:]['Close'].to_numpy()
```

```
X = sd.unroll(X,1)
Y = Y[-X.shape[0]:]

print(X.shape)
print(Y.shape)

# Generate predictions
predictions = model.predict(X)

#get the test score
testScore = model.evaluate(X, Y, verbose=0)
print('Test Score: %.4f MSE (%.4f RMSE)' % (testScore, math.sqrt(testScore)))
```