



School of Computer Science and Engineering

Predicting Stock Movement Using Market Correlation Networks

CSE3021 – Social and Information Networks a Project Report

Submitted to:

Dr Jayaram

Submitted by:

Harsh Sharma (19BCE1464)

Shreeharsh Pandey (19BCE1818)

In partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

June 2021

TABLE OF CONTENTS

Page

1. Abstract.....	3
2. Introduction.....	4
3. Feasibility study.....	5
4. About the dataset.....	7
5.Design and flow of models.....	8
6. Module list	
6.1. Exploratory Data Analysis (EDA).....	9
6.2. Data Pre-Processing.....	12
6.3 Building Various Models.....	14
7. Algorithms.....	15
8.Risk Analysis.....	19
9.Implementation.....	20
9.1.Analysis.....	20
9.2.Visualization.....	22
9.3.Comparing the results.....	31
10. Results.....	33
11.Conclusion.....	34
12.References.....	35
13.List of figures.....	36
14.Appendix code.....	38

ABSTRACT

The aim of Predicting Stock Movement is to provide the customers with measured predicted data and analysis of the stock market. This project will use a correlation network to deal with all functionalities a stock market will encounter. This project is best suited for stock movements which have precise previous data which can be analysed to predict the future movements. This project helps to track and analyse the current and previous stock movements and predict the future ones. It Stores the data of stock movements happening. While working on this project it also helps us to know our stocks, learn machine learning properly and understand the models of Stock Prediction.

INTRODUCTION

The goal for this project is to discern whether network properties of financial markets can be used to predict market dynamics. Building on previous work involving networks derived from market price correlations, we augment basic price correlation networks with additional information (revenue, sentiment, and news-ow). Our intuition is that these alternative networks will capture relationships beyond price correlations (e.g., business model exposures) that could eventually enhance downstream predictive models. The final insight we aim to provide is a prediction of future market behaviour based on features that incorporate both standard trading information (price, volume, etc.) and market network characteristics (centrality, clustering coefficient, etc.).

The project methodology comprises three components: structural, analytical, and predictive. In the structural component, we filter the data to find and visualize the underlying structural motifs of the network. In the analytical component, additional metrics are computed for graphs built from the full dataset, and we do statistical testing to see whether our graph features have predicting power for stock prices. These two components use correlations of both prices and the newly introduced news/sentiment variables when building networks. They also featurize properties of our market correlation networks for sub-periods of years or quarters to see how these networks change over time. Finally, the predictive component incorporates features/metrics generated by the structural and analytical components into a recurrent neural network (RNN) to predict binary market movements (up/down) over a future period of interest.

FEASIBILITY STUDY

Feasibility Study Depending on the results of the initial investigation a more detailed feasibility study is prepared so as to evaluate if the project is viable or not. “FEASIBILITY STUDY” is a test of system proposal according to its workability, impact of the organization, ability to meet needs and effective use of the resources. It focuses on these major questions:

1. What are the user’s needs and how does the proposed system meet them?
2. What resources are available with the user?
3. Is it worth solving the problem?

During feasibility analysis for this project, following primary areas of interest were considered. Investigation and generating ideas about the new solution gives the following results.

Technical feasibility

A study of resource availability that may affect the ability to achieve an acceptable system. This evaluation determines whether the technology needed for the proposed system is available or not.

This is concerned with specifying equipment and software that will successfully satisfy the user requirement.

During this project the

1. Software requirement

1.1 Any basic Operating System like Linux, Windows or MAC

1.2 Google Colab

2. Hardware requirement

2.1 8GB RAM for faster training of the neural network

The technical feasibility is the most important part of this project as having the required system only can enable the use of the proposed solution.

Economic feasibility

Economic justification is generally the “Bottom Line” consideration for most systems. Economic justification includes a broad range of concerns that includes cost benefit analysis. In this we weigh the cost and the benefits associated with the proposed solution and if it suits the basic purpose of the organization generating revenues. Economic feasibility for this project is not required

Operational Feasibility

It is mainly related to human organizations and political aspects. The system is operationally feasible as it is very easy to operate. It only needs basic information about any OS platform.

Schedule feasibility

Time evaluation is the most important consideration in the development of a project. The time schedule required for the development of this project is very important since more development time affects machine time, cost and causes delay in the development of other systems. The model can be developed in the considerable amount of time

ABOUT DATASET

The dataset consists of details about the stock market prices and are listed below:

The types of stock movements are:

1. Date
2. Open
3. Close
4. High
5. Low
6. Volume

File descriptions

- train.csv - the training set, contains details of stock price with their labels (has 1259rows * 6 columns)
- test.csv - the test set for which the stock market price will be predicted (has 21rows * 6 columns)

Datasets:

- <https://www.kaggle.com/akram24/google-stock-price-train>
- <https://www.kaggle.com/akram24/google-stock-price-test>

DESIGN AND FLOW OF MODULES:

Following are the modules in which we have divided are project:

- Forming the strategy for making prediction model and setting the definite approach.
- Analysing the data
- Visualizing the data using several types of plots
- Creating a train and test dataset from the available data and making a feature extraction model.
- Implementing different ML models and algorithms and doing a comparative analysis to find out where the accuracy is best.

MODULE LIST

1. Exploratory Data Analysis (EDA):

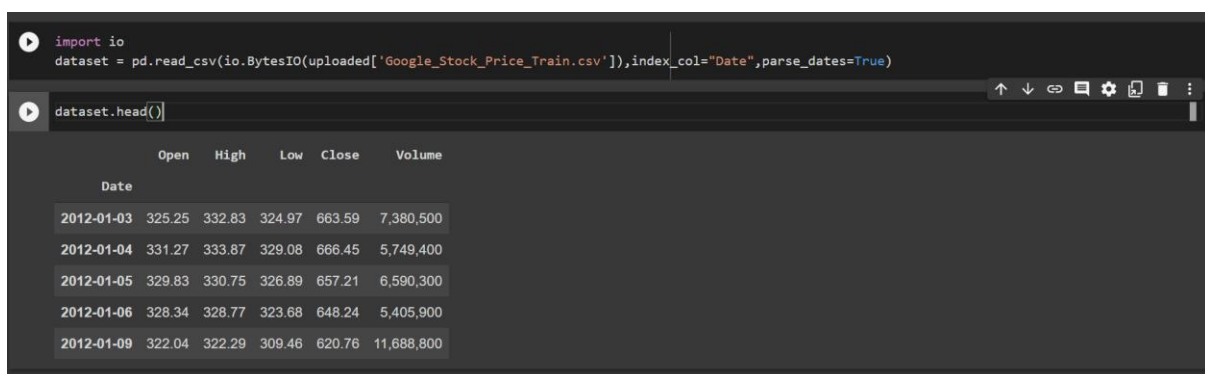
Exploratory Data Analysis is one of the important steps in the data analysis process. Here, the focus is on making sense of the data in hand — things like formulating the correct questions to ask to your dataset, how to manipulate the data sources to get the required answers, and others.

- First, we will import the required libraries

```
[ ] #Importing the Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime
```

FIGURE 1: IMPORTING LIBRARIES

- Load the csv dataset and check the first five rows as default from it.



```
import io
dataset = pd.read_csv(io.BytesIO(uploaded['Google_Stock_Price_Train.csv']), index_col="Date", parse_dates=True)

dataset.head()
```

Date	Open	High	Low	Close	Volume
2012-01-03	325.25	332.83	324.97	663.59	7,380,500
2012-01-04	331.27	333.87	329.08	666.45	5,749,400
2012-01-05	329.83	330.75	326.89	657.21	6,590,300
2012-01-06	328.34	328.77	323.68	648.24	5,405,900
2012-01-09	322.04	322.29	309.46	620.76	11,688,800

FIGURE 2: LOADING DATASET

- Get the overall information of the dataset used.

```
[ ] dataset.isna().any()

Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool

[ ] dataset.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1258 entries, 2012-01-03 to 2016-12-30
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Open    1258 non-null     float64
1   High    1258 non-null     float64
2   Low     1258 non-null     float64
3   Close   1258 non-null     object
4   Volume  1258 non-null     object
dtypes: float64(3), object(2)
memory usage: 59.0+ KB
```

FIGURE 3: INFORMATION REGARDING DATASET USED

- Plot the graph of one of the columns, here, OPEN to show the correctness



FIGURE 4: CORRECTNESS GRAPH

```
[16] dataset['Close: 30 Day Mean'] = dataset['Close'].rolling(window=30).mean()
dataset[['Close', 'Close: 30 Day Mean']].plot(figsize=(16,6))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc4046898d0>

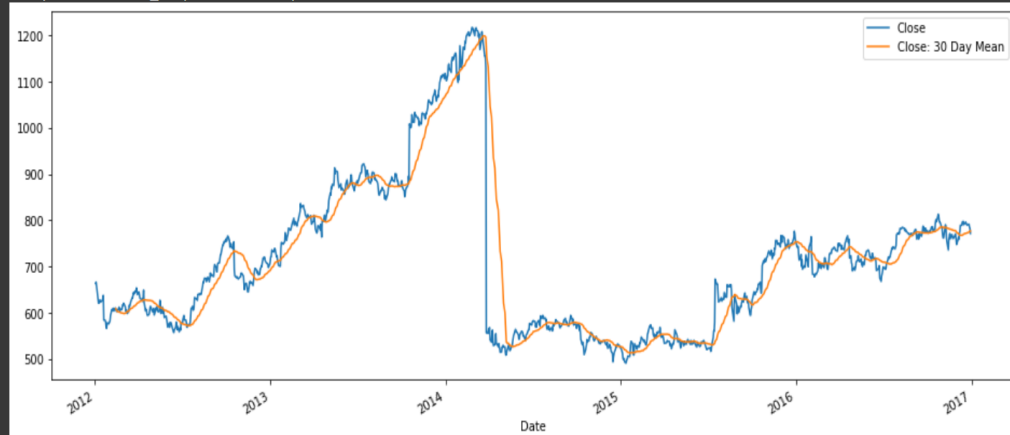


FIGURE 5: GRAPH REPRESENTATION

```
[17] # Optional specify a minimum number of periods
dataset['Close'].expanding(min_periods=1).mean().plot(figsize=(16,6))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc40463b590>

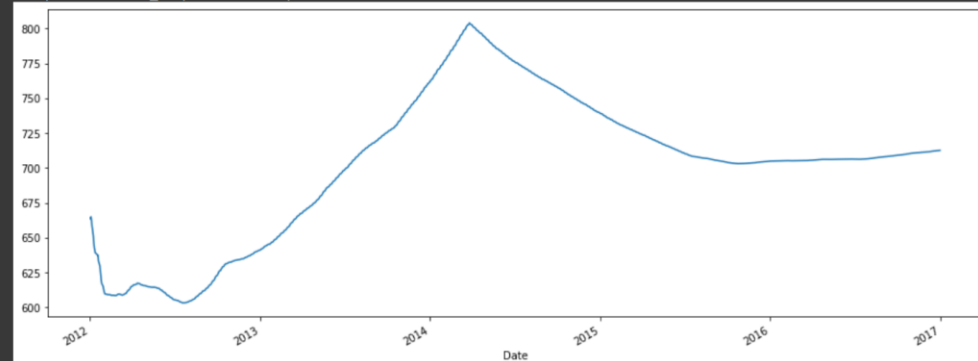


FIGURE 6: GRAPH REPRESENTATION

2. Data Pre-processing:

- Remove any punctuation marks from the columns

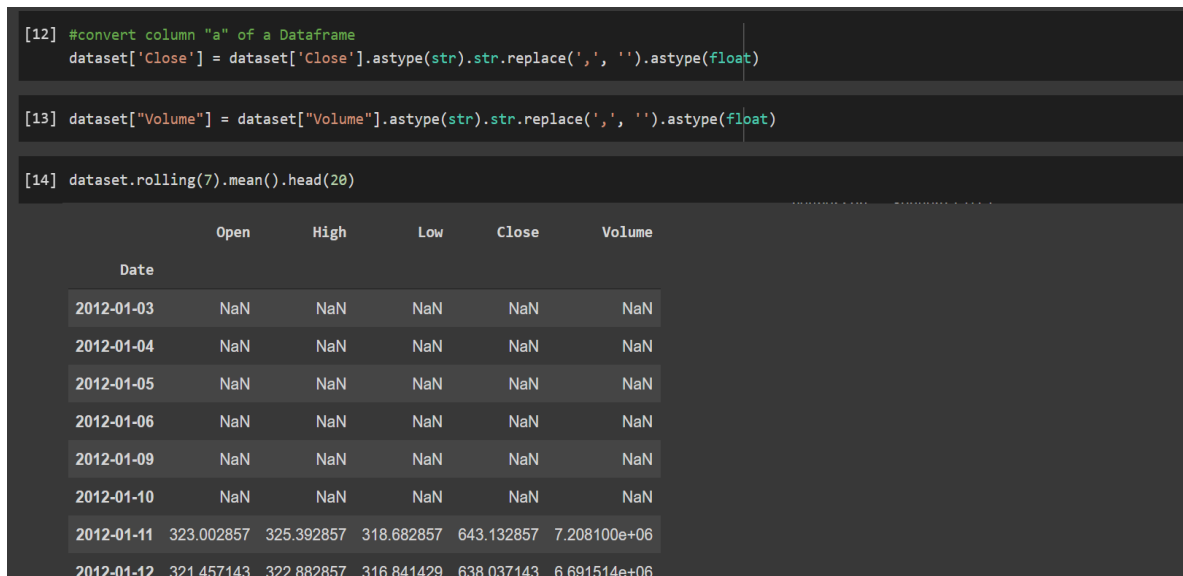


FIGURE 7: REMOVAL OF PUNCTUATION MARKS

- Plotted Open and Close Column

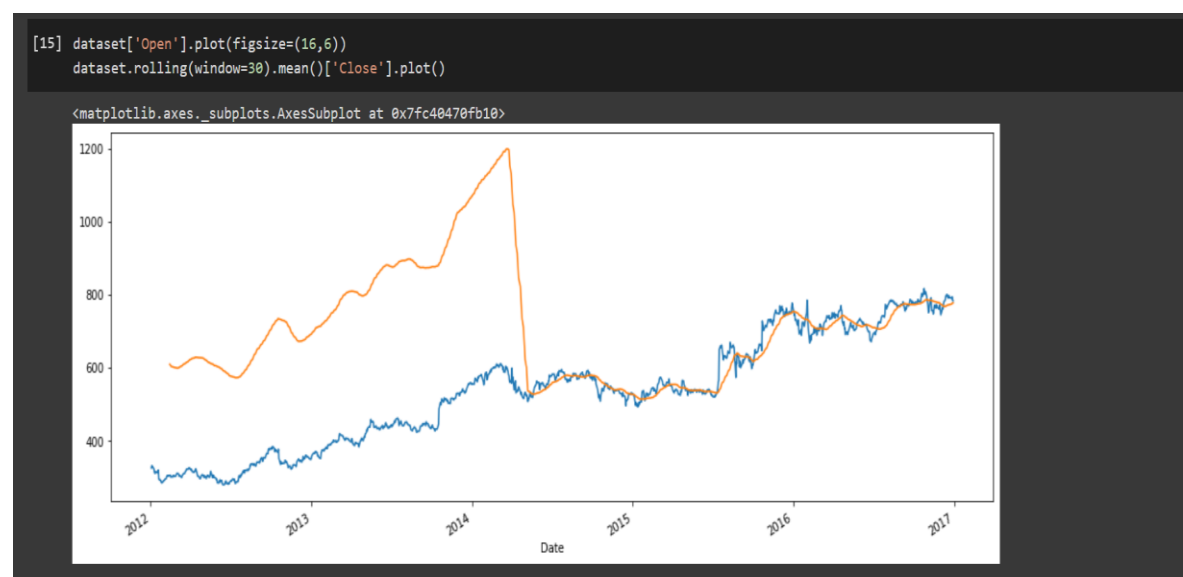


FIGURE 8: OPEN AND CLOSE REPRESENTATION

- Data Cleaning and setting the training data frame

```
[18] training_set=dataset['Open']  
      training_set=pd.DataFrame(training_set)
```

```
[19] #Data Cleaning  
      dataset.isna().any()
```

```
Open           False  
High           False  
Low            False  
Close          False  
Volume         False  
Close: 30 Day Mean  True  
dtype: bool
```

FIGURE 9: DATA CLEANING

```
[20] # Feature Scaling  
      from sklearn.preprocessing import MinMaxScaler  
      sc = MinMaxScaler(feature_range = (0, 1))  
      training_set_scaled = sc.fit_transform(training_set)
```

```
[ ] # Creating a data structure with 60 timesteps and 1 output  
    X_train = []  
    y_train = []  
    for i in range(60, 1258):  
        X_train.append(training_set_scaled[i-60:i, 0])  
        y_train.append(training_set_scaled[i, 0])  
    X_train, y_train = np.array(X_train), np.array(y_train)  
  
    # Reshaping  
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

FIGURE 10: FEATURE SCALING

3. Building Various Models:

We use various models to predict the stock movements in the market. The models that we have used are as follows:

LSTM:

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behaviour required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

ARIMA:

An ARIMA model is a class of statistical models for analysing and forecasting time series data. It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skilful time series forecasts. It's a model used in statistics and econometrics to measure events that happen over a period of time. The model is used to understand past data or predict future data in a series.

FBPROPHET:

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is an additive regression model with a piecewise linear or logistic growth curve trend. It includes a yearly seasonal component modelled using Fourier series and a weekly seasonal component modelled using dummy variables.

ALGORITHMS

LSTM Model:

In order to build the LSTM, we need to import a couple of modules from Keras:

- Sequential for initializing the neural network.
- Dense for adding a densely connected neural network layer
- LSTM for adding the Long Short-Term Memory layer
- Dropout for adding dropout layers that prevent overfitting

We add the LSTM layer and later add a few Dropout layers to prevent overfitting. We add the LSTM layer with the following arguments:

- 50 units which is the dimensionality of the output space
- return_sequences=True which determines whether to return the last output in the output sequence, or the full sequence
- input_shape as the shape of our training set.

When defining the Dropout layers, we specify 0.2, meaning that 20% of the layers will be dropped. Thereafter, we add the Dense layer that specifies the output of 1 unit. After this, we compile our model using the popular adam optimizer and set the loss as the mean_squared_error. This will compute the mean of the squared errors. Next, we fit the model to run on 100 epochs with a batch size of 32.

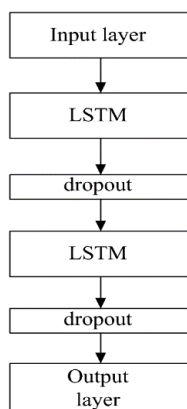


Diagram 1: Flow algorithm of LSTM

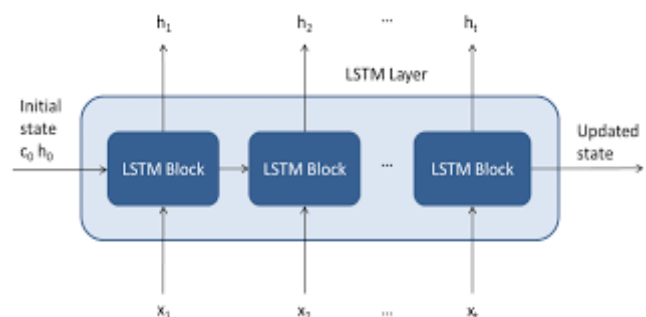


Diagram 2: Block diagram

Arima Model:

The time series model is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

- **Level:** The average value in the series.
- **Trend:** The increasing or decreasing value in the series.
- **Seasonality:** The repeating short-term cycle in the series.
- **Noise:** The random variation in the series.

First, we need to check if a series is stationary or not because time series analysis only works with stationary data.

Let's divide the data into a training (70 %) and test (30%) set. For this tutorial we select the following ARIMA parameters: $p=4$, $d=1$ and $q=0$.

A rolling forecasting procedure is required given the dependence on observations in prior time steps for differencing and the AR model. To this end, we re-create the ARIMA model after each new observation is received.

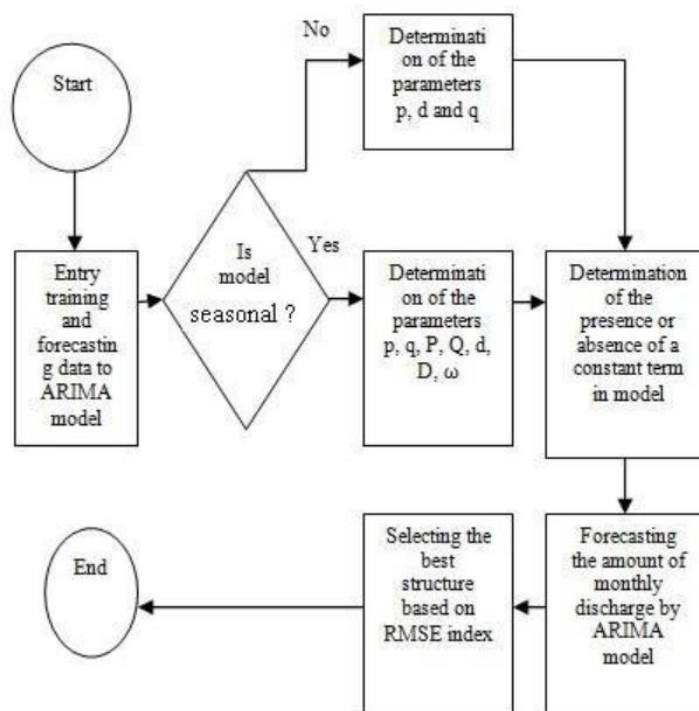


Diagram 3: Flow Diagram of ARIMA

FBPROPHET MODEL:

For implementing in Python, it is needed to have some of the libraries installed: Pandas, Matplotlib, FB Prophet (we can install it through! `pip install fbprophet`).

Prophet works under the specific format that is the data frame should have only 'ds' and 'y' as names ds for dates and y for observations.

After renaming the columns now the next thing to do is to change the data to date time format using Pandas as Prophet algorithm uses DateTime format.

After the model is created it is fitted with the data(*data_to_use* here) like in other Deep Learning/Machine Learning Models i.e.

```
model.fit(data_to_use)
```

One of the interesting things about Prophet is that with *make_future_dataframe* function Periods for the future prediction can be provided.

When the models are loaded with future dates, the 'y' component values get predicted.

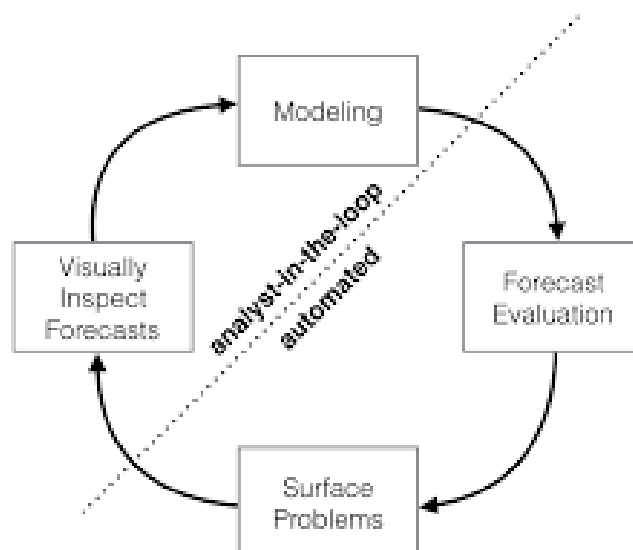


Diagram 4: Flow Diagram of Prophet

Adam Optimizer

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

Tikhonov Regularization

Tikhonov regularization, named for Andrey Tikhonov, is a method of regularization of ill-posed problems. Also known as ridge regression, it is particularly useful to mitigate the problem of multicollinearity in linear regression, which commonly occurs in models with large numbers of parameters. In general, the method provides improved efficiency in parameter estimation problems in exchange for a tolerable amount of bias (see bias-variance trade-off).

Risk Analysis

Identifying the Risk

The potential risks involved while doing this project were the time limit and the CPU/GPU/TPU requirement to train the vast number of neural networks and machine learning models. With a consumer grade computer, it would be not be possible to train all the models hence most of the training process has to be done on cloud platforms like google Colab or Aws sage maker. Even these online cloud platforms come their own personal risks like the RAM getting overloaded with data due to the vast amount of training data.

Probability of Risk affecting the project

Although google Colab has the necessary support to run the models even in free version but still due to the sheer size of the corpus there is some chance that colab may crash due to excessive load on the RAM.

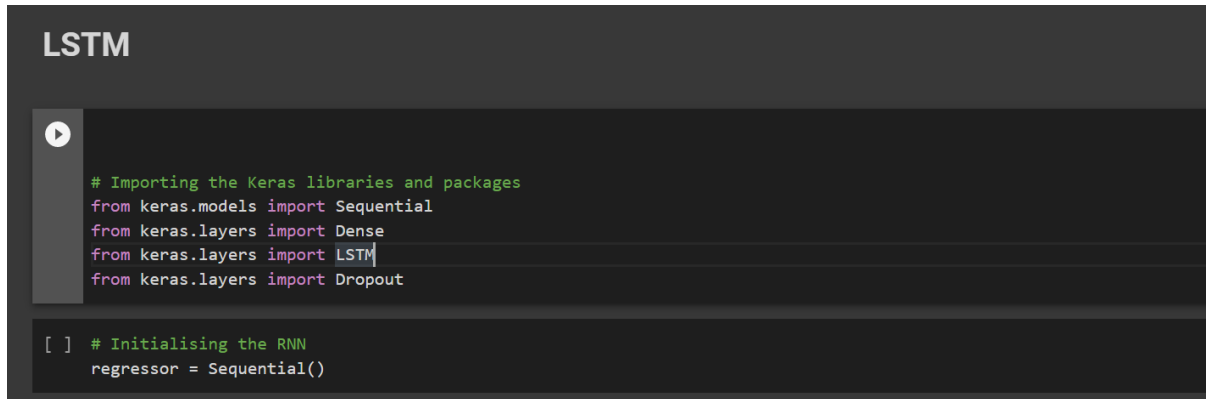
Impact on the project

The impact of the above-mentioned risk is quite minimal as if the RAM does not support the entire training data at a time, then measures can be taken to feed chunks of training data and then go for an ensemble model after training each individual model. This will be better overall and could lead to faster training with lesser RAM consumption

IMPLEMENTATION

1. Analysis

- Importing the libraries and initializing the RNN

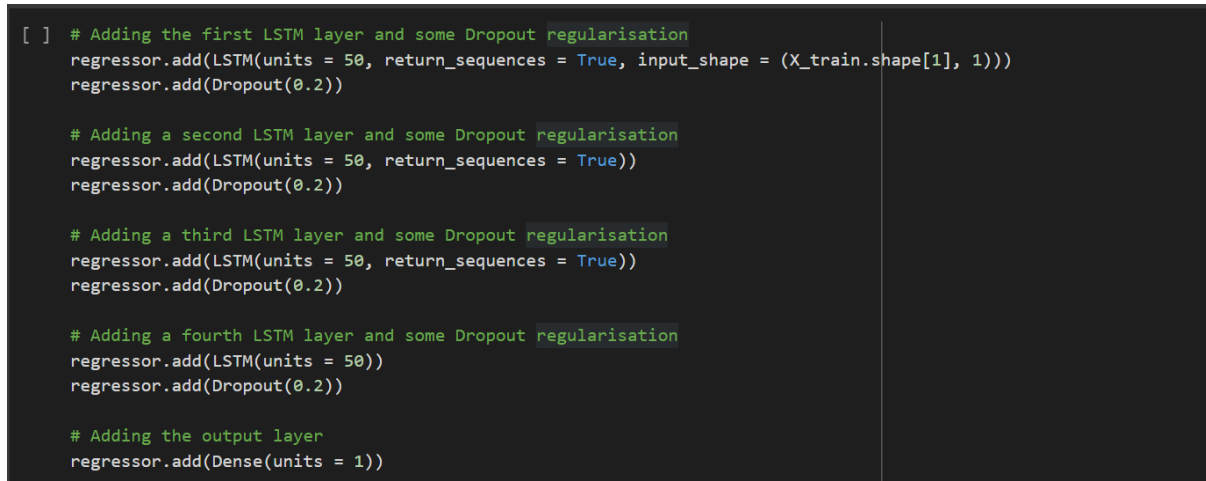


```
LSTM

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

[ ] # Initialising the RNN
regressor = Sequential()
```

FIGURE 11: IMPORTING REQUIRED LIBRARIES



```
[ ] # Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))
```

FIGURE 12: LSTM MODEL IMPLEMENTATION

- Compiling the RNN and using adam optimizer

```
[ ] # Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)

Epoch 1/100
38/38 [=====] - 28s 115ms/step - loss: 0.0775
Epoch 2/100
38/38 [=====] - 4s 114ms/step - loss: 0.0057
Epoch 3/100
38/38 [=====] - 4s 116ms/step - loss: 0.0051
Epoch 4/100
38/38 [=====] - 4s 116ms/step - loss: 0.0047
Epoch 5/100
38/38 [=====] - 4s 115ms/step - loss: 0.0050
Epoch 6/100
38/38 [=====] - 4s 116ms/step - loss: 0.0048
Epoch 7/100
38/38 [=====] - 4s 113ms/step - loss: 0.0057
Epoch 8/100
38/38 [=====] - 4s 115ms/step - loss: 0.0047
Epoch 9/100
38/38 [=====] - 4s 114ms/step - loss: 0.0045
Epoch 10/100
38/38 [=====] - 4s 114ms/step - loss: 0.0045
```

FIGURE 13: COMPILE OF RNN

- Importing the Test data set from Kaggle

```
[ ] from google.colab import files
    uploaded = files.upload()

Choose Files Google_Stock_Price_Test.csv
• Google_Stock_Price_Test.csv(application/vnd.ms-excel) - 1029 bytes, last modified: 4/13/2021 - 100% done
Saving Google_Stock_Price_Test.csv to Google_Stock_Price_Test.csv
```

FIGURE 14: DATASET IMPORT

- Making predictions and visualizing the data

Importing Test dataset and extracting information about it.

```
[ ] # Getting the real stock price of 2017
import io
dataset_test= pd.read_csv(io.BytesIO(uploaded['Google_Stock_Price_Test.csv']),index_col="Date",parse_dates=True)

[ ] real_stock_price = dataset_test.iloc[:, 1:2].values

[ ] dataset_test.head()
```

Date	Open	High	Low	Close	Volume
2017-01-03	778.81	789.63	775.80	786.14	1,657,300
2017-01-04	788.36	791.34	783.16	786.90	1,073,000
2017-01-05	786.08	794.48	785.02	794.02	1,335,200
2017-01-06	795.26	807.90	792.20	806.15	1,640,200
2017-01-09	806.40	809.97	802.83	806.65	1,272,400

FIGURE 15: PREDICTION IMAGE

```
[ ] dataset_test.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 20 entries, 2017-01-03 to 2017-01-31
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Open    20 non-null        float64
1   High    20 non-null        float64
2   Low     20 non-null        float64
3   Close   20 non-null        float64
4   Volume  20 non-null        object
dtypes: float64(4), object(1)
memory usage: 960.0+ bytes
```

```
[ ] dataset_total = pd.concat((dataset['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
    X_test.append(inputs[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))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
[ ] predicted_stock_price=pd.DataFrame(predicted_stock_price)
predicted_stock_price.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype
---  -
0   0       20 non-null        float32
dtypes: float32(1)
memory usage: 208.0 bytes
```

FIGURE 16: PREDICTION IMAGE

2. Visualization

```
[ ] # Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```

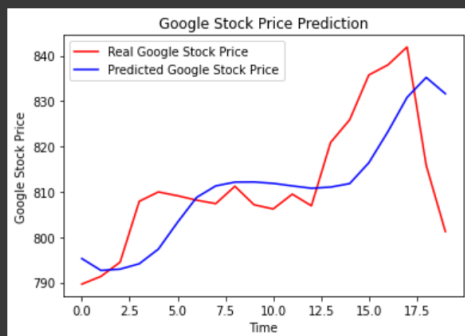


FIGURE 17: PREDICTION

ARIMA MODEL:

▼ ARIMA MODEL

```
[ ] import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlib

matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'

[ ] df1 = pd.read_csv('Google_Stock_Price_Train.csv')
stock=df1
```

```
[ ] stock.head()
```

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

```
[ ] stock['Date'].min()
```

'1/10/2012'

```
[ ] stock['Date'].max()
```

'9/9/2016'

```
[ ] stock.Date = pd.to_datetime(stock.Date, format='%Y%m%d', errors='ignore')
```

```
[ ] cols = ['High', 'Low', 'Open', 'Volume', 'Open']
stock.drop(cols, axis=1, inplace=True)
stock = stock.sort_values('Date')
```

```
[ ] stock.isnull().sum()
```

```
Date      0
Close     0
dtype: int64
```

```
[ ] stock = stock.groupby('Date')['Close'].sum().reset_index()
```

FIGURE 18: ARIMA MODEL IMPLEMENTATION

```
[ ] stock.head()

      Date    Close
0  1/10/2012  621.43
1  1/10/2013  739.45
2  1/10/2014  1,127.09
3  1/11/2012   624.25
4  1/11/2013   737.96

[ ] stock = stock.set_index('Date')
stock.index

Index(['1/10/2012', '1/10/2013', '1/10/2014', '1/11/2012', '1/11/2013',
      '1/11/2016', '1/12/2012', '1/12/2015', '1/12/2016', '1/13/2012',
      ...,
      '9/6/2016', '9/7/2012', '9/7/2016', '9/8/2014', '9/8/2015', '9/8/2016',
      '9/9/2013', '9/9/2014', '9/9/2015', '9/9/2016'],
      dtype='object', name='Date', length=1258)
```

FIGURE 19: ARIMA MODEL WORKING

```
[ ] #y = stock['Close'].resample('M').mean()
stock.index = pd.to_datetime(stock.index)

monthly_mean = dataset.Close.resample('M').mean()

[ ] monthly_mean['2015:']

      Date
2015-01-31  511.014500
2015-02-28  536.517895
2015-03-31  558.184091
2015-04-30  539.304286
2015-05-31  535.239000
2015-06-30  532.915909
2015-07-31  590.093636
2015-08-31  636.838095
2015-09-30  617.934762
2015-10-31  663.592727
2015-11-30  735.388500
2015-12-31  755.354545
2016-01-31  718.495789
2016-02-29  702.689000
2016-03-31  727.056818
2016-04-30  736.833810
2016-05-31  712.182857
```

FIGURE 20: ARIMA MODEL OUTPUT

```
[ ] Date
2015-01-31  511.014500
2015-02-28  536.517895
2015-03-31  558.184091
2015-04-30  539.304286
2015-05-31  535.239000
2015-06-30  532.915909
2015-07-31  590.093636
2015-08-31  636.838095
2015-09-30  617.934762
2015-10-31  663.592727
2015-11-30  735.388500
2015-12-31  755.354545
2016-01-31  718.495789
2016-02-29  702.689000
2016-03-31  727.056818
2016-04-30  736.833810
2016-05-31  712.182857
2016-06-30  706.486364
2016-07-31  725.719500
2016-08-31  775.793478
2016-09-30  773.619524
2016-10-31  788.449524
2016-11-30  766.640952
2016-12-31  782.171429
Freq: M, Name: Close, dtype: float64
```

FIGURE 21: OUTPUT



FIGURE 22: ARIMA MODEL VISUALISATION

```
[ ] l_param = []
    l_param_seasonal=[]
    l_results_aic=[]
    for param in pdq:
        for param_seasonal in seasonal_pdq:
            try:
                mod = sm.tsa.statespace.SARIMAX(monthly_mean,
                                                order=param,
                                                seasonal_order=param_seasonal,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                results = mod.fit()

                print('ARIMA({}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))

                l_param.append(param)
                l_param_seasonal.append(param_seasonal)
                l_results_aic.append(results.aic)
            except:
                continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:947.9358112063479
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:728.3327441619533
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:667.892501433547
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
"Check mle_retvals", ConvergenceWarning)
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1273.6769627907347
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:682.6056466797982
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:643.5104207545708
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:500.8108412156643
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
"Check mle_retvals", ConvergenceWarning)
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1653.2773872139562
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:864.3980908376452
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:662.0591427960572
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:612.2826347128226
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:2626.360289535644
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:639.1130093429332
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:592.2800204749811
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:473.09224502684077
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1541.5241975728206
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:678.4076883552631
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:548.8657190393873
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:577.2586059171712
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
"Check mle_retvals", ConvergenceWarning)
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1218.3338341306521
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:560.0495437316682
```

```
[ ] minimum=l_results_aic[0]
    for i in l_results_aic[1:]:
        if i < minimum:
            minimum = i
    i=l_results_aic.index(minimum)

[ ] mod = sm.tsa.statespace.SARIMAX(monthly_mean,
                                    order=l_param[i],
                                    seasonal_order=l_param_seasonal[i],
                                    enforce_stationarity=False,
                                    enforce_invertibility=False)

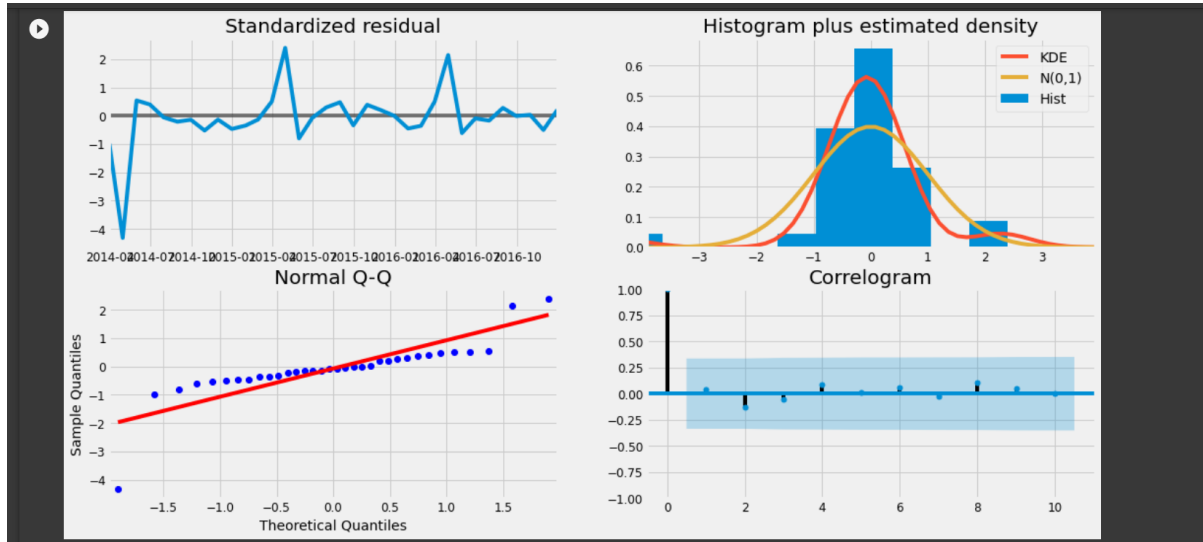
    results = mod.fit()

    print(results.summary().tables[1])

=====
              coef    std err          z      P>|z|      [0.025    0.975]
-----
ar.L1          0.2009      0.291      0.689      0.491      -0.370      0.772
ar.S.L12       -0.4455      0.072     -6.148      0.000      -0.588     -0.303
sigma2        1.469e+04    2468.015      5.952      0.000    9852.929    1.95e+04
=====
```

FIGURE 23: GETTING OPEN AND CLOSE VALUE

```
results.plot_diagnostics(figsize=(16, 8))
plt.show()
```



```
[ ] pred = results.get_prediction(start=pd.to_datetime('2015-10-31'), dynamic=False)
pred_ci = pred.conf_int()

ax = monthly_mean['2013:'].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.4, figsize=(14, 7))

ax.fill_between(pred_ci.index,
               pred_ci.iloc[:, 0],
               pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('close price')
plt.legend()

plt.show()
```

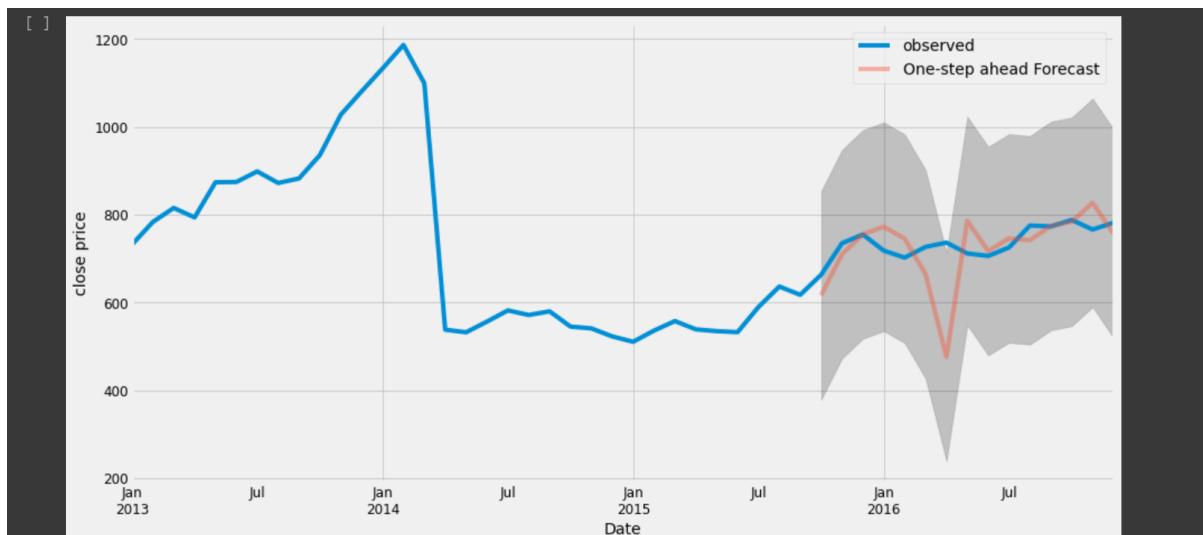


FIGURE 24: GRAPHICAL ANALYSIS

```
[ ] y_forecasted = pred.predicted_mean
y_truth = monthly_mean['2015-10-31:']

# Compute the mean square error
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))

The Mean Squared Error of our forecasts is 6037.39

[ ] print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))

The Root Mean Squared Error of our forecasts is 77.7

[ ] pred_uc = results.get_forecast(steps=100)
pred_ci = pred_uc.conf_int()

ax = monthly_mean.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
               pred_ci.iloc[:, 0],
               pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('close price')

plt.legend()
plt.show()
```

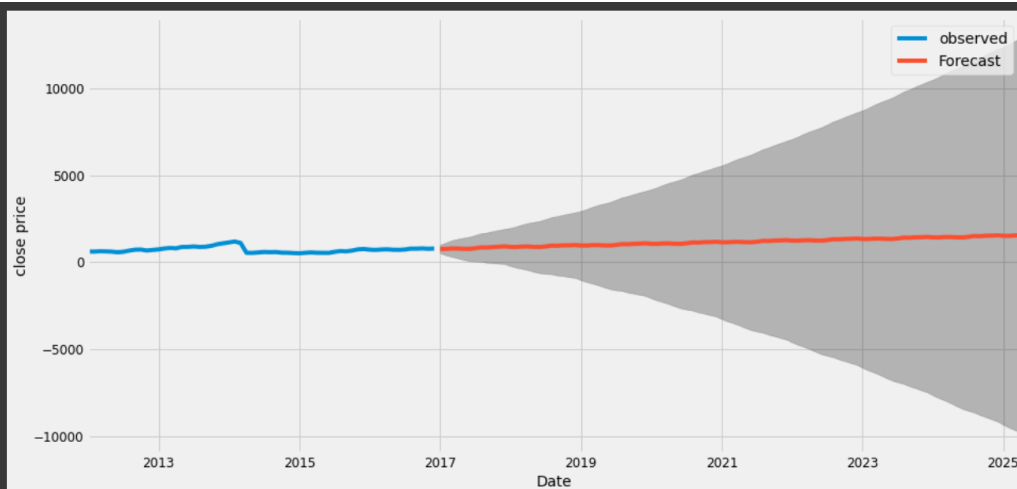


FIGURE 25: GRAPHICAL REPRESENTATION OF MODEL

PROPHET MODEL:

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas_datareader as web
import warnings
!pip install fbprophet
import fbprophet

Requirement already satisfied: fbprophet in /usr/local/lib/python3.7/dist-packages (0.7.1)
Requirement already satisfied: Cython>=0.22 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (0.29.23)
Requirement already satisfied: cmdstanpy>=0.9.5 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (0.9.5)
Requirement already satisfied: pystan>=2.14 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (2.19.1.1)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (1.19.5)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (1.1.5)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (3.2.2)
Requirement already satisfied: LunarCalendar>=0.0.9 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (0.0.9)
Requirement already satisfied: convertdate>=2.1.2 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (2.3.2)
Requirement already satisfied: holidays>=0.10.2 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (0.10.5.2)
Requirement already satisfied: setuptools-git>=1.2 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (1.2)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (2.8.1)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.7/dist-packages (from fbprophet) (4.41.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=1.0.4->fbprophet) (2018.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.0.0->fbprophet) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.0.0->fbprophet) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.0.0->fbprophet) (2.4.7)
Requirement already satisfied: ephemeris>=3.7.5.3 in /usr/local/lib/python3.7/dist-packages (from LunarCalendar>=0.0.9->fbprophet) (3.7.7.1)
Requirement already satisfied: pymeeus<1,>=0.3.13 in /usr/local/lib/python3.7/dist-packages (from convertdate>=2.1.2->fbprophet) (0.5.11)
Requirement already satisfied: hijri-converter in /usr/local/lib/python3.7/dist-packages (from holidays>=0.10.2->fbprophet) (2.1.1)
Requirement already satisfied: korean-lunar-calendar in /usr/local/lib/python3.7/dist-packages (from holidays>=0.10.2->fbprophet) (0.2.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from holidays>=0.10.2->fbprophet) (1.15.0)
```

```
[ ] from google.colab import files
    uploaded = files.upload()
    data = pd.read_csv("INFY.NS.csv")
    data.head()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving INFY.NS.csv to INFY.NS.csv

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-04-13	629.000000	651.700012	626.049988	637.400024	622.156494	8330276.0
1	2020-04-15	651.400024	653.299988	635.599976	639.049988	623.767029	11963447.0
2	2020-04-16	619.950012	634.900024	603.500000	623.849976	608.930481	17783287.0
3	2020-04-17	644.000000	646.000000	626.000000	628.750000	613.713379	8878320.0
4	2020-04-20	640.000000	661.000000	639.000000	653.299988	637.676270	13322913.0

```
[ ] plt.style.use("fivethirtyeight")
    plt.figure(figsize=(16,8))
    plt.title("Google Closing Stock Price")
    plt.plot(data["Close"])
    plt.xlabel("Date", fontsize=14)
    plt.ylabel("Close Price", fontsize=14)
    plt.show()
```



FIGURE 26: IMPLEMENTATION AND GRAPH

```
[ ] data = data[["Date", "Close"]]
    data = data.rename(columns = {"Date": "ds", "Close": "y"})
    data.head()
```

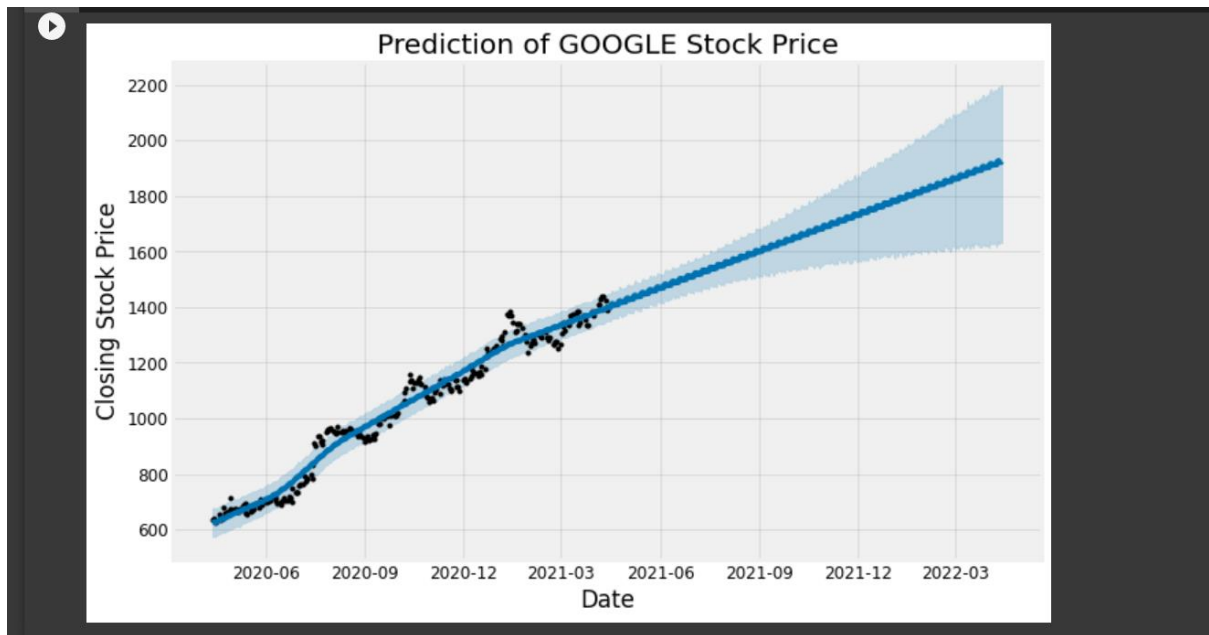
	ds	y
0	2020-04-13	637.400024
1	2020-04-15	639.049988
2	2020-04-16	623.849976
3	2020-04-17	628.750000
4	2020-04-20	653.299988

```
[ ] from fbprophet import Prophet
    m = Prophet(daily_seasonality=False)
    m.fit(data)
```

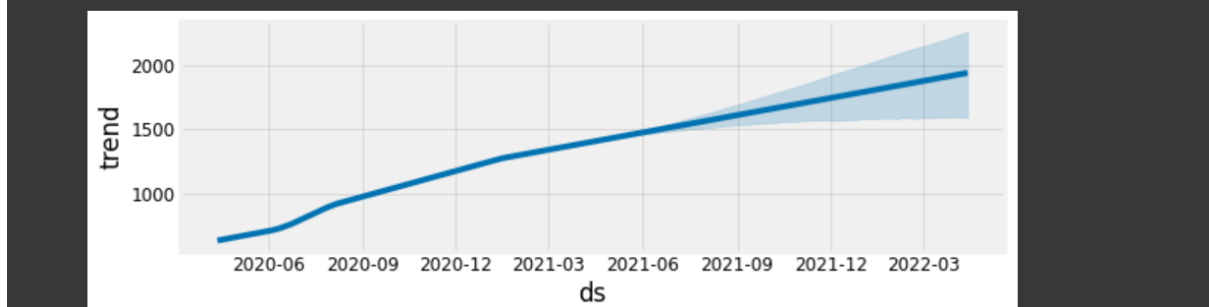
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7fc393de3e90>

```
[ ] future = m.make_future_dataframe(periods=365)
    predictions=m.predict(future)
    m.plot(predictions)
    plt.title("Prediction of GOOGLE Stock Price")
    plt.xlabel("Date")
    plt.ylabel("Closing Stock Price")
    plt.show()
```

FIGURE 27 PROPHET MODEL IMPLEMENTATION



```
[ ] m.plot_components(predictions)
    plt.show()
```



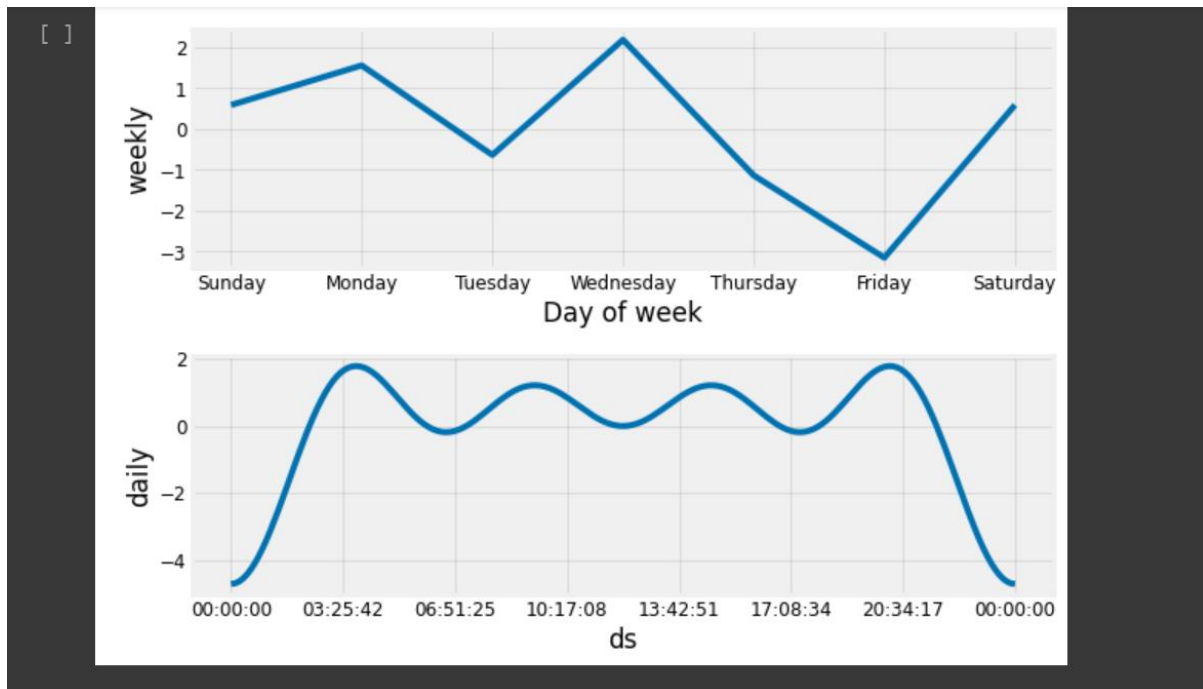
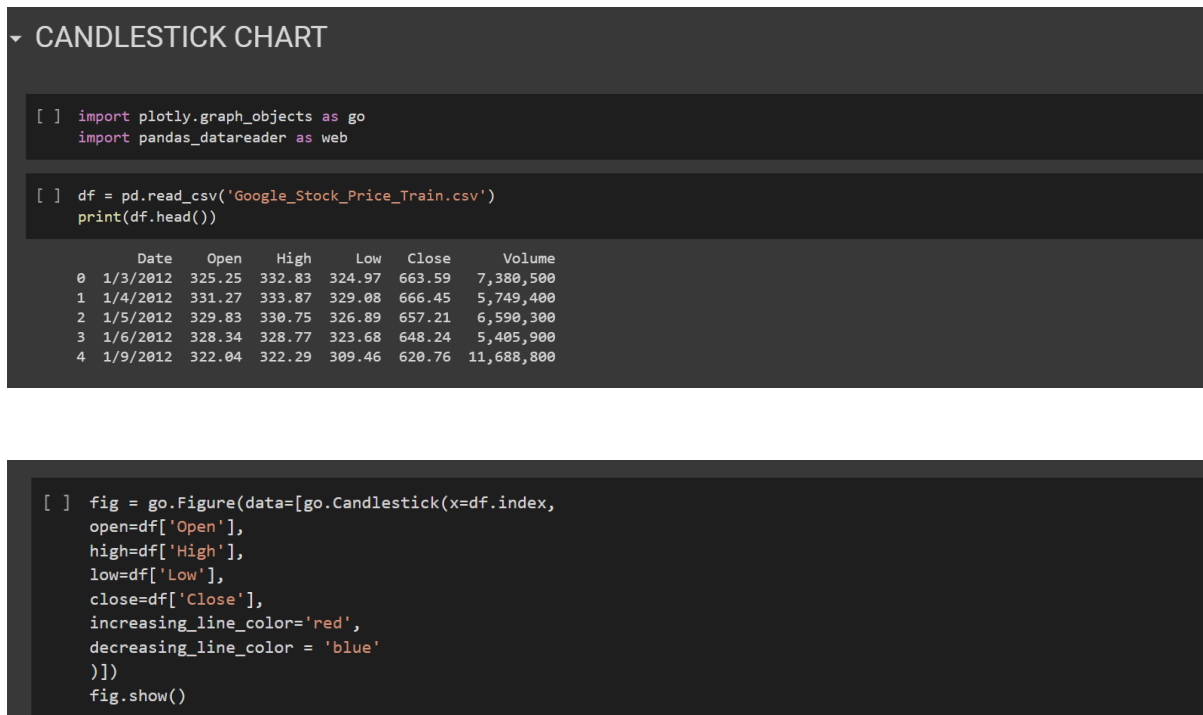


FIGURE 28: PROPHET MODEL OUTCOME

3. COMPARING THE RESULTS

CANDLE STICK CHART:



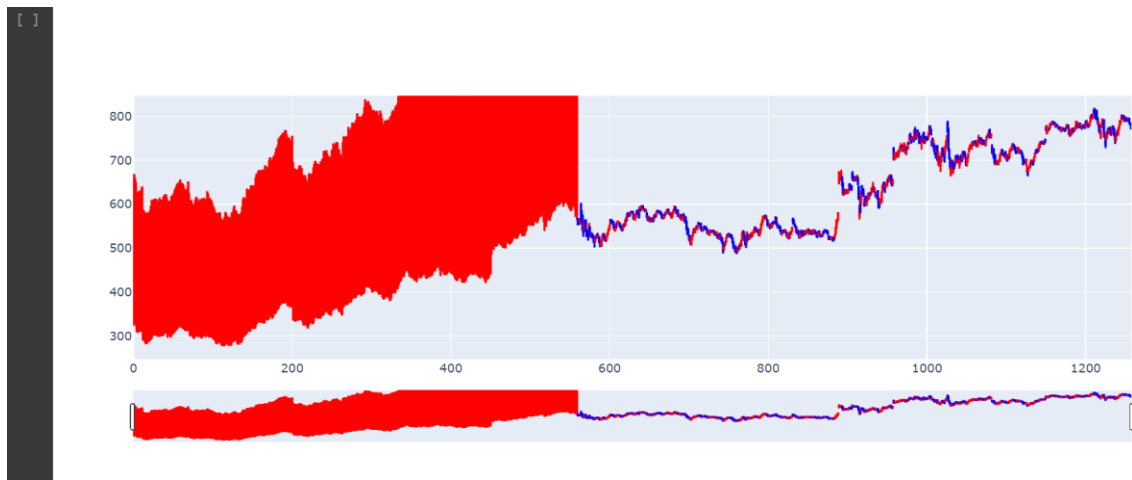


FIGURE 29: CANDLESTICK CHART ANALYSIS

RESULTS

On trying out all the three models we get to know that the most effective model is Prophet Model. It gives approximately same value thus predicting the prices to the nearest value.

We see that ARIMA outperforms LSTM and even more powerful Prophet model while making the predictions. From the experiment, we can see that SARIMAX model forecasting has better accuracy than the Prophet model forecasting. The RMSE for the SARIMAX model was around 8% while Prophet Model had RMSE of 11.4%.

Facebook Prophet is a good package to use because the syntax is very similar to SKlearn and easy to plot in Plotly for trend analysis.

Below is the demonstration of the same:

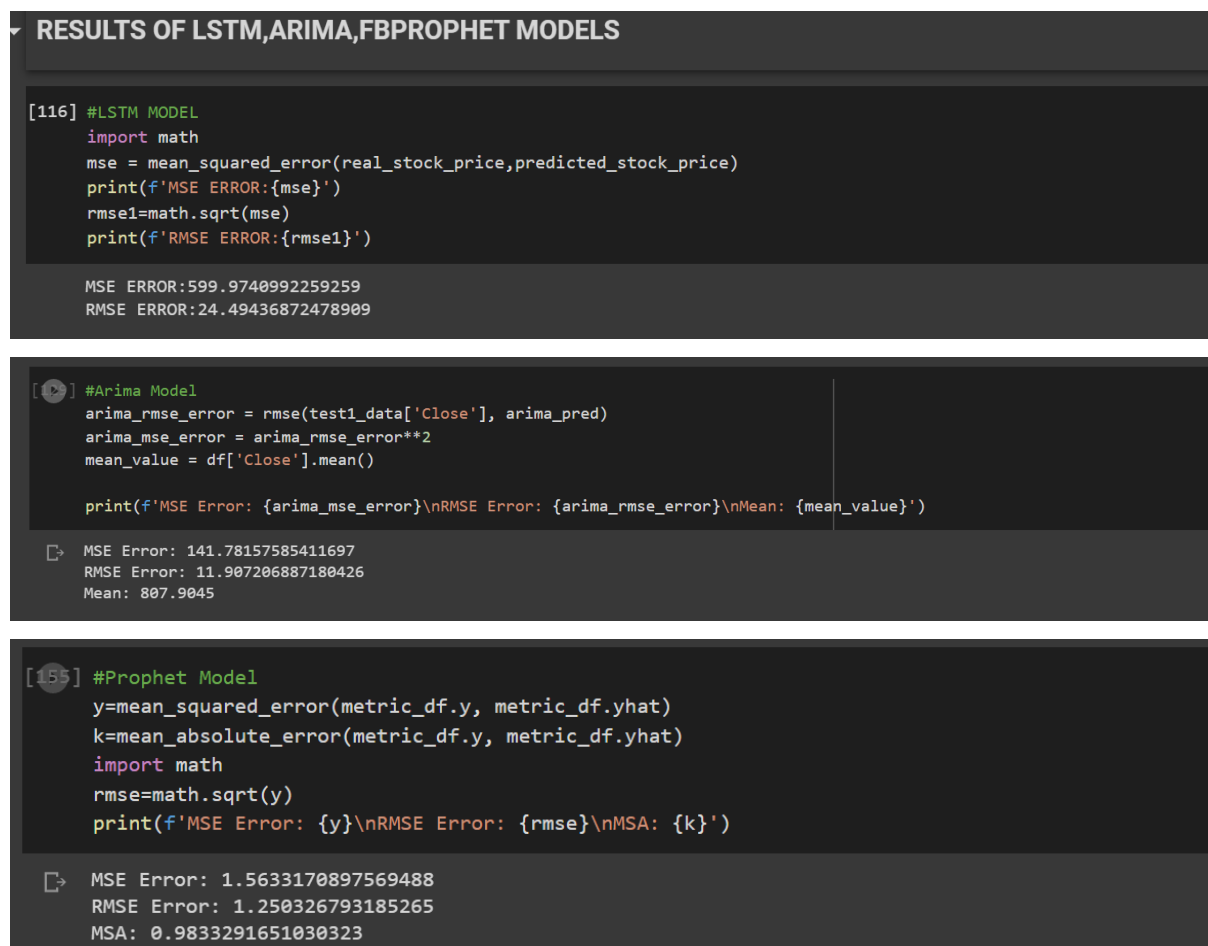


FIGURE 30: RESULTS AND COMPARISON

CONCLUSION

Stock markets are hard to monitor and require plenty of context when trying to interpret the movement and predict prices. LSTMs perform better as they are able to keep track of the context-specific temporal dependencies between stock prices for a longer period of time while performing predictions. An analysis of the results also indicates that both models give better accuracy when the size of the dataset increases. With more data, more patterns can be fleshed out by the model, and the weights of the layers can be better adjusted. At its core, the stock market is a reflection of human emotions. Pure number crunching and analysis have their limitations; a possible extension of this stock prediction system would be to augment it with a news feed analysis from social media platforms such as Twitter, where emotions are gauged from the articles. This sentiment analysis can be linked with the LSTM to better train weights and further improve accuracy.

In conclusion, I would like to say that the popularity of stock market trading is growing extremely rapidly which is encouraging researchers to find out new methods for the prediction using new techniques, the forecasting technique is not only helpful to researchers but also helps investors or any person dealing with stock market in order to help predict the stock indices a forecasting model with good accuracy is required in this work we have used one of the most precise forecasting technology is using RNN and LSTM which helps investors analysts or any person interested in the stock market by providing them a good knowledge of the future situation of the stock market in such little project at least we were able to get the trend right.

REFERENCES

- [1] Tse, C. K., Liu, J., and Lau, F. C.M., 2010. A network perspective of the stock market. *Journal of Empirical Finance* 17 (2010), p. 659-667.
- [2] Boginski, V., Butenko, S., and Pardalos, P. M., 2004. Statistical analysis of financial networks. *Computational Statistics & Data Analysis* 48 (2005), p. 431-443.
- [3] Giles, C., Lawrence, S., and Tsoi, A., 2001. Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Inference. *Machine Learning*, Volume 44, Number 12, July/August, pp. 161-183.
- [4] Timmermann, A., and Granger, C. W.J., 2004. Efficient market hypothesis and forecasting. *International Journal of Forecasting*, Volume 20, Issue 1, January-March 2004, p. 15-27.
- [5] Hochreiter, S., and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation* 9, p. 1735-1780.
- [6] Hermans, M., and Schrauwen, B., 2013. Training and Analyzing Deep Recurrent Neural Networks. NIPS 2013. http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2013_5166.pdf

LIST OF FIGURES

Figure No.	Explanation	Page No.
Figure 1	Importing Libraries	9
Figure 2	Loading Dataset	9
Figure 3	Information Regarding Dataset Used	10
Figure 4	Correctness Graph	10
Figure 5	Graph Representation	11
Figure 6	Graph Representation	11
Figure 7	Removal of Punctuation Marks	12
Figure 8	Open and Close Representation	12
Figure 9	Data Cleaning	13
Figure 10	Feature Scaling	13
Figure 11	Importing required libraries	20
Figure 12	LSTM model implementation	20
Figure 13	Compilation of RNN	21
Figure 14	Dataset Import	21
Figure 15	Prediction Image	21

Figure 16	Prediction Image	22
Figure 17	Prediction	22
Figure 18	ARIMA model implementation	23
Figure 19	ARIMA model working	24
Figure 20	ARIMA model output	24
Figure 21	Output	24
Figure 22	ARIMA model visualization	25
Figure 23	Getting open and close value	26
Figure 24	Graphical analysis	27
Figure 25	Graphical representation of model	28
Figure 26	Implementation and graph	29
Figure 27	Prophet model implementation	30
Figure 28	Prophet model outcome	31
Figure 29	Candlestick Analysis	32
Figure 30	Results and comparisons	33
Diagram 1	Flow algorithm of LSTM	15
Diagram 2	Block diagram	15
Diagram 3	Flow diagram of ARIMA	16

Diagram 4	Flow Diagram of Prophet	17
-----------	-------------------------	----

APPENDIX CODE:

https://colab.research.google.com/drive/1nKialsTMdYCu4p_UOVQIM3ZY_oIdklHm?usp=sharing