SVKM'S NMIMS, MPSTME, Shirpur Campus

FINAL REPORT

FINAL REPORT

PROJECT: **STOCK MARKET FORECAST**

A report submitted in partial fulfilment of the requirements of 5 Years Integrated MBA (Tech) Program of Mukesh Patel School of Technology Management & Engineering, NMIMS.

Faculty Mentor:
Ms Varsha Nemade
(Assistant Professor)
MPSTME,
NMIMS Shirpur

Submitted By:
Dipanshu Agarwal – N201
Riya Airen – N204
Saurabh Ajit – N205
Campus Course: MBATECH CE
Batch: 2017 – 2022

ABSTRACT

Predicting the Stock Market has been the bane and goal of investors since its existence. Everyday billions of dollars are traded on the exchange, and behind each dollar is an investor hoping to profit in one way or another. Entire companies rise and fall daily based on the behaviour of the market. Should an investor be able to accurately predict market movements, it offers a tantalizing promises of wealth and influence. In the real world, the stock market predictions can be categorized in 2 parts, Fundamental Analysis and Technical Analysis.

Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. psychological, rational and irrational behaviour, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.

In this undertaking, we will be creating supervised machine learning models which will help us to somewhat predict the price value of stocks/security of a company i.e. State Bank of India to be specific. The Model will be using Time-Series Analysis, Time series is a set of observations or data points taken at specified time usually at equal intervals and it's used to predict the future values based on the previous observed values.

The stock data for the past 9 years (2011-2019) has been collected and trained using different model with different parameters. The models were applied to dataset to forecast the value of stock of Sate bank of India (SBI).

INDEX

INTRODUCTION

Stock Market prediction has always had a certain appeal for researchers. While numerous scientific attempts have been made, no method has been discovered to accurately predict stock price movement. The difficulty of prediction lies in the complexities of modelling market dynamics. Even with a lack of consistent prediction methods, there have been some mild successes. Stock Market research encapsulates two elemental trading philosophies; Fundamental and Technical approaches. In Fundamental analysis, Stock Market price movements are believed to derive from a security's relative data. Fundamentalists use numeric information such as earnings, ratios, and management effectiveness to determine future forecasts. In Technical analysis, it is believed that market timing is key. Technicians utilize charts and modelling techniques to identify trends in price and volume. These later individuals rely on historical data in order to predict future outcomes.

The stock market appears in the news every day. Every time it reaches a new high or a new low. The rate of investment and business opportunities in the Stock market can increase if an efficient algorithm could be devised to predict the short term price of an individual stock.

OBJECTIVES:

In the past decades, there is an increasing interest in predicting markets among economists, policymakers, academics and market makers. The objective of the proposed work is to study and implement the supervised learning algorithm to predict the stock price.

TECHNICAL OBJECTIVE

The technical objectives will be implemented in Python. The system must be able to access a list of historical prices. It must calculate the estimated price of stock based on the historical data. It must also provide an instantaneous visualization of the market index.

- To add to the academic understanding of stock market prediction
- This project will focus exclusively on predicting the daily trend (price movement) of individual stocks
- The project will also analyse the accuracies of these predictions
- The project will also compare the accuracy percentage between the different models implemented.

LITERATURE REVIEW

Contreras et al. [3] used ARIMA models to predict next day electricity prices; they have found two ARIMA models to predict hourly prices in the electricity markets of Spain and California. The Spanish model needs 5 hours to predict future prices as opposed to the 2 hours needed by the Californian model.

Kumar et al. [4] used ARIMA model to forecast daily maximum surface ozone concentrations in Brunei Darussalam. They have found that ARIMA (1,0,1) was suitable for the surface 03 data collected at the airport in Brunei Darussalam.

Tsitsika et al. [5] used ARIMA model to forecast pelagic fish production. The final model selected were of the form AR[MA (1,0,1) and AR[MA (0,1,1).

Azad et al. [6] used ARIMA model in forecasting Exchange Rates of Bangladesh. By using Box-Jenkins methodology they tried to find out the best model for forecasting.

Debadrita Banerjee et al.[7] has collected data on the monthly closing stock indices of sensex for six years(2007-2012) and based on these she has tried to develop an appropriate model which would help her to forecast the future unobserved values of the Indian stock market indices. This study offers an application of ARIMA model based on which she predicts the future stock indices which have a strong influence on the performance of the Indian economy. To establish the model she applied the validation technique with the observed data of sensex of 2013.

TIME-SERIES

It is simply a series of data points ordered in time. In a time series, time is often the independent variable and the goal is usually to make a forecast for the future.

The Time-Series generated may have any of the 3 properties:-

AUTO-CORRELATION

Refers to the similarity between observations as a function of the time lag between them. For example, in the graph below the first and the 24th value have a high autocorrelation similarly for the 12th and 36th value

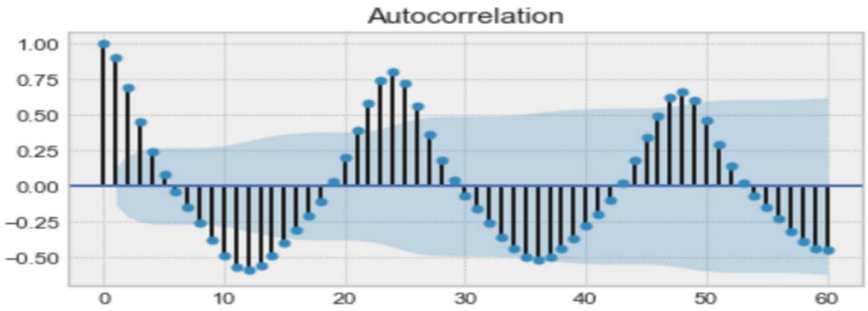

Figure 1: EXAMPLE OF AN AUTO-CORRELATION PLOT

SEASONALITY

Refers to periodic functions, for example electricity consumption is high during the day and low during night, or online sales increase during Christmas before slowing down again. Seasonality can also be derived from an **Auto-Correlation Plot** if it has a sinusoidal shape.

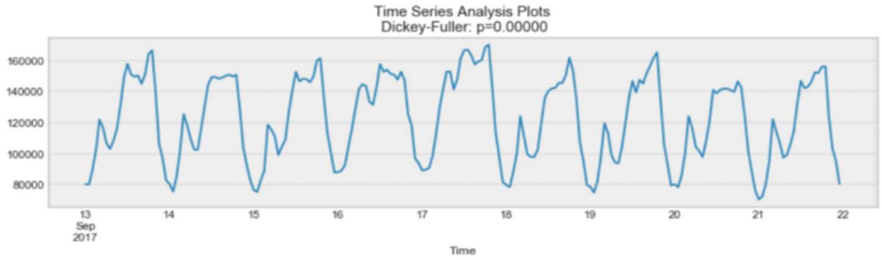

Figure 2: EXAMPLE OF SEASONALITY

STATIONARY

Refers to an important characteristic of time-series. A time-series is said to be stationary if its statistical properties do not change over time. In other words, it has constant mean and variance, and co-variance is independent of time.

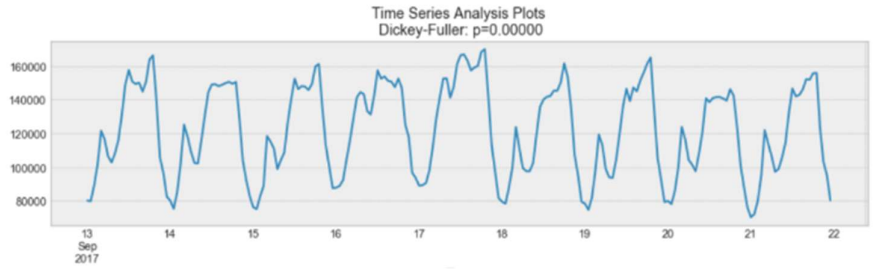

Figure 3: EXAMPLE OF SEASONALITY

In the above, we see that the process above is stationary. The mean and variance do not vary over time.

Often **Stock Prices** are not a stationary process, since we might see a growing trend, or its volatility might increase over time i.e. variance is ever-changing.

Ideally, we want to have a stationary time-series for modelling. Of course, not all of them are stationary, but we can make different transformations to make them stationary.

MAPE Value

The mean absolute percentage error (MAPE) is the mean or average of the absolute percentage errors of forecasts. Error is defined as actual or observed value minus the forecasted value. Percentage errors are summed without regard to sign to compute MAPE. This measure is easy to understand because it provides the error in terms of percentages. Also, because absolute percentage errors are used, the problem of positive and negative errors cancelling each other out is avoided. Consequently, MAPE has managerial appeal and is a measure commonly used in forecasting. The smaller the MAPE the better the forecast.

The mean absolute percentage error (MAPE) can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. Where At is the actual value and Ft is the forecast value, this is given by:

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

The MAPE is also sometimes reported as a percentage, which is the above equation multiplied by 100. The difference between A_t and F_t is divided by the actual value A_t again. The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points n. multiplying by 100% makes it a percentage error.

The mean absolute percentage error (MAPE) is the most common measure used to forecast error, and works best if there are no extremes to the data (and no zeros).

Although MAPE is straightforward to calculate and easy to interpret, there are a couple potential drawbacks to using it:

1. Since the formula to calculate absolute percent error is |actual-forecast| / |actual| this means that it will be undefined if any of the actual values are zero.

2. MAPE should not be used with low volume data. For example, if the actual demand for some item is 2 and the forecast is 1, the value for the absolute percent error will be |2-1| / |2| = 50%, which makes it seem like the forecast error is quite high, despite the forecast only being off by one unit.

RMSE Value

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values of these differences. These deviations are called residuals when the calculations are performed over the data sample that was used for estimation and are called errors (or prediction errors) when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. RMSD is a measure of accuracy.

We can compare any predicted value with an actual measurement (observed value).

- Predicted value
- Observed value

Root mean square error takes the difference for each observed and predicted value.

You can swap the order of subtraction because the next step is to take the square of the difference. This is because the square of a negative value will always be a positive value. But just make sure that you keep the same order throughout.

After that, divide the sum of all values by the number of observations. Finally, we get a RMSE value. Here's what the **RMSE Formula** looks like:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (P_i - O_i)^2}{n}}$$

FUNDAMENTALS OF TRADING

Figure 4: A SAMPLE OF HOW THE TECHNICAL CHART IN BSE LOOKS LIKE

In the above graph,

We see a peculiar representation technique being used which is known as a candlestick, a candlestick is a type of price chart used in technical analysis that displays the high, low, open, and closing prices of a security for a specific period.

The Color of the Bar i.e. Red or Green denotes that the stock closed on a Lower price or a Higher Price respectively on that particular day.

There is a third state which is known as a consolidated state, in which the number of buyers and sellers for a security are the same in the market. In this case, it is denoted using a simple horizontal line.

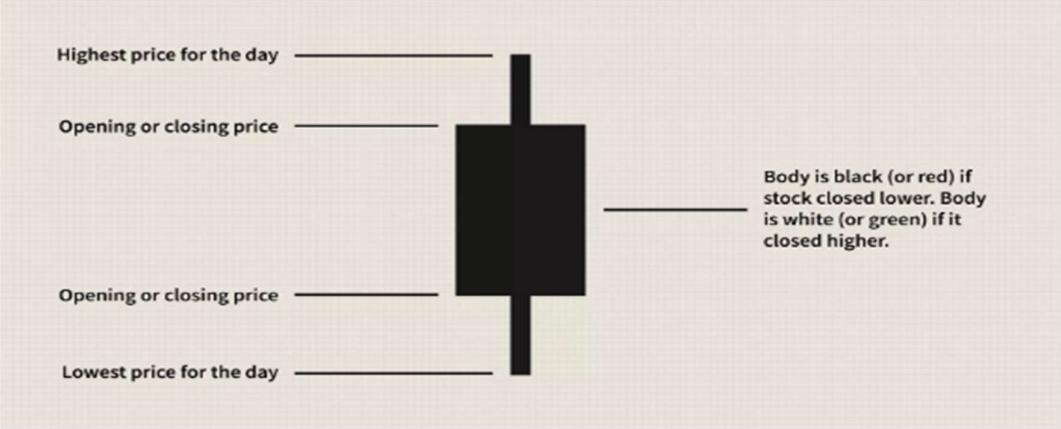

Figure 5: TECHNICALITITES OF A CANDLESTICK FIGURE

The Main Factors Which Bring About A Huge Change In The Variance And Mean Of The Security Prices Are:-

1. **Buyer and Seller: -** From the colours, we can also conclude that Red or Black means there are more number of buyers in the market as they will try to drive the price of security down so as to buy it at a low cost, and Green or White means that the numbers of seller of that particular security is more as they will try to drive the price higher so as to earn maximum profit on selling the security.
2. **IPO: -** stand for Initial Public Offering. When the news media report that a company is "going public," this **means** that company is making an **initial public offering**. This **means** that the company is offering its shares for sale to the public for the first time.
3. **Pandemics: -** External shocks can derail economic trends and abruptly alter market sentiment. Not all risk is economic policy or monetary
4. **Merger and Acquisitions: -** Mergers and acquisitions are transactions in which the ownership of companies, other business organizations, or their operating units are transferred or consolidated with other entities.
5. **Import and Export: -** An economic condition that occurs when a country is importing more goods than it is exporting is called trade deficit. It is also referred to as net exports. A trade deficit is calculated by deducting value of goods being exported from the goods being imported by a country. It is mentioned in the currency that a particular country uses. An extended trade deficit can adversely impact a country's economy and its stock markets. A country with extended trade deficit means it's into debt. The investors take notice of such financial parameters and also take note of plunge in spending

on locally produced goods. This hurts domestic producers and their share prices. This reduces demand in the domestic stock markets and result in decline of market.

6. **Government Issues/Changes to Laws: -** The laws and ordinance passed by the government may stimulate the market or it may even bring a pause on the Foreign Investments happening in the market on the basis of the current situations of the global economy and trade treaties.

TREND ANALYSIS

Trend analysis is a technique used in technical analysis that attempts to predict the future stock price movements based on recently observed trend data. Trend analysis is based on the idea that what has happened in the past gives traders an idea of what will happen in the future.

A trend is the general direction the market is taking during a specified period of time. Trends can be both upward and downward, relating to bullish and bearish markets, respectively. While there is no specified minimum amount of time required for a direction to be considered a trend, the longer the direction is maintained, the more notable the trend.

TYPES OF TRENDS

- BULLISH OR UPWARD TREND

  A bull market is a rising market. In a bull market, investors are positive. Trend line is a line that is drawn over pivot highs or under pivot lows to show the prevailing direction of price. Trendlines are a visual representation of support and resistance in any time frame. Since in a bullish trend orders are placed upwards it is very important to determine the lower trendline of an upward movement. Break through a lower trend line signals that a general trend weakens or changes its previous direction.

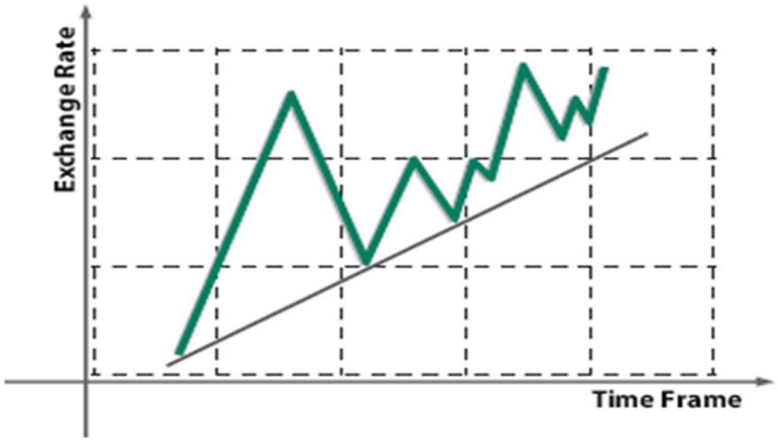

Figure 6: BULLISH TREND

- BEARISH OR DOWNTREND

  A prolonged period in which investment prices fall, accompanied by widespread pessimism. A bear market is a market showing a lack of confidence. Lower trend line is a line that is drawn under pivot lows to show the prevailing direction of price. Bullish trendline is a visual representation of a resistance. Since in a bearish trend orders are placed downwards it is very important to determine the upper trendline. Break through an upward trend line signals that a general trend weakens or changes its previous direction.

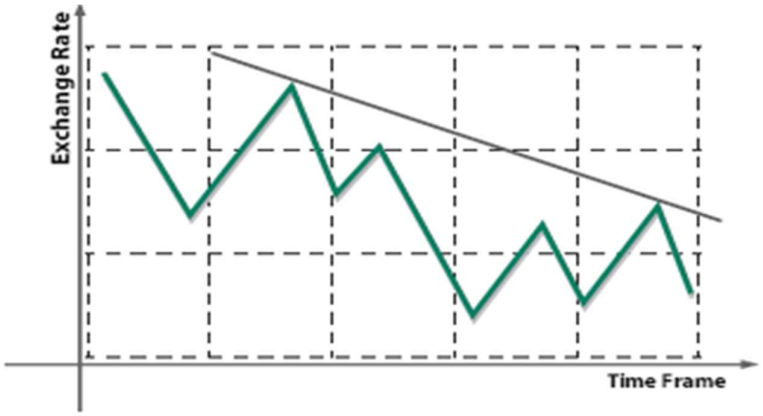

Figure 7: BEARISH TREND

- SIDEWAYS TREND

  A sideways market occurs where the price trend of a certain trading instrument has been experiencing neither an uptrend nor a downtrend. Sideways trend is generally a result of the price traveling between strong levels of support and resistance. Sometimes, when a price moves in a reverse direction once the upward or downward trend is over, there are no certain price directions at the market. In this case, price fall or rise movements in a vertical direction are not expected. As a rule, when the sideways trend ends an uptrend or downtrend may begin.

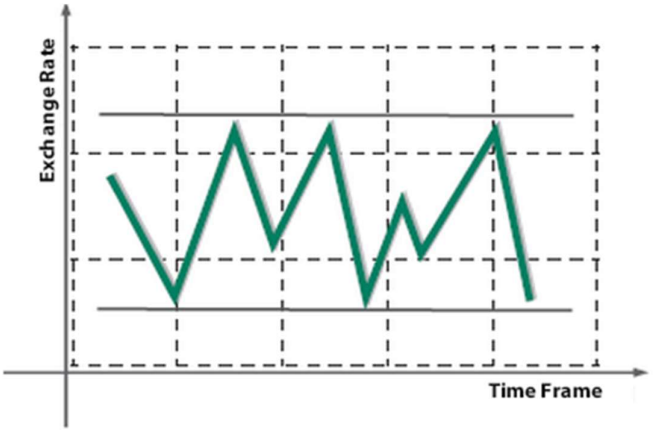

Figure 8: SIDEWAYS TREND

What Does Trend Analysis Tell You?

Trend analysis tries to predict a trend, such as a bull market run, and ride that trend until data suggests a trend reversal, such as a bull-to-bear market. Trend analysis is helpful because moving with trends, and not against them, will lead to profit for an investor.

A trend is the general direction the market is taking during a specified period of time. Trends can be both upward and downward, relating to bullish and bearish markets, respectively. While there is no specified minimum amount of time required for a direction to be considered a trend, the longer the direction is maintained, the more notable the trend.

Trend analysis is the process of trying to look at current trends in order to predict future ones and is considered a form of comparative analysis. This can include attempting to determine whether a current market trend, such as gains in a particular market sector, is likely to continue, as well as whether a trend in one market area could result in a trend in another. Though an analysis may involve a large amount of data, there is no guarantee that the results will be correct.

Examples of Trend Analysis

In order to begin analyzing applicable data, it is necessary to first determine which market segment will be analyzed. An example of sectors can include a focus on a particular industry, such as the automotive or pharmaceuticals sector, as well as a particular type of investment, such as the bond market. Once the sector has been selected, it is possible to examine the general performance of the sector. This can include how the sector was affected by internal and external forces. For example, changes in a similar industry or the creation of a new governmental regulation would qualify as forces impacting the market. Analysts then take this data and attempt to predict the direction the market will take moving forward.

Trend Trading Strategies

Trend traders attempt to isolate and extract profit from trends. There are many different trend trading strategies using a variety of indicators:

- **Moving Averages:** These strategies involve entering into long positions when a short-term moving average crosses over above a long-term moving average, and entering short positions when a short-term moving average crosses below a long-term moving average.
- **Momentum Indicators:** These strategies involve entering into long positions when a security is trending with strong momentum and exiting long positions

when a security loses momentum. Often times, the relative strength index (RSI) is used in these strategies.

- **Trendlines & Chart Patterns:** These strategies involve entering long positions when a security is trending higher and placing a stop-loss below key trendline support levels. If the stock starts to reverse, the position is exited for a profit.

Indicators can simplify price information, as well as provide trend trade signals or warn of reversals. Indicators can be used on all time frames, and have variables that can be adjusted to suit each trader's specific preferences. Combine indicator strategies, or come up with your own guidelines, so entry and exit criteria are clearly established for trades. Each indicator can be used in more ways than outlined. If you like an indicator, research it further, and most importantly, test it out before using it to make live trades.

Limitations of Trend Analysis

Critics of trend analysis, and technical trading in general, argue that markets are efficient, and so they already price in all available information. That means that history does not necessarily need to repeat itself, and that the past does not predict the future. Adherents of fundamental analysis, for example, analyze the financial condition of companies using financial statements and economic models to predict future price. For these types of investors, day-to-day stock movements follow a random walk that cannot be interpreted as patterns or trends.

METHODOLOGY

We will be using a 6-step approach to build each model then we will compare the accuracy for each.

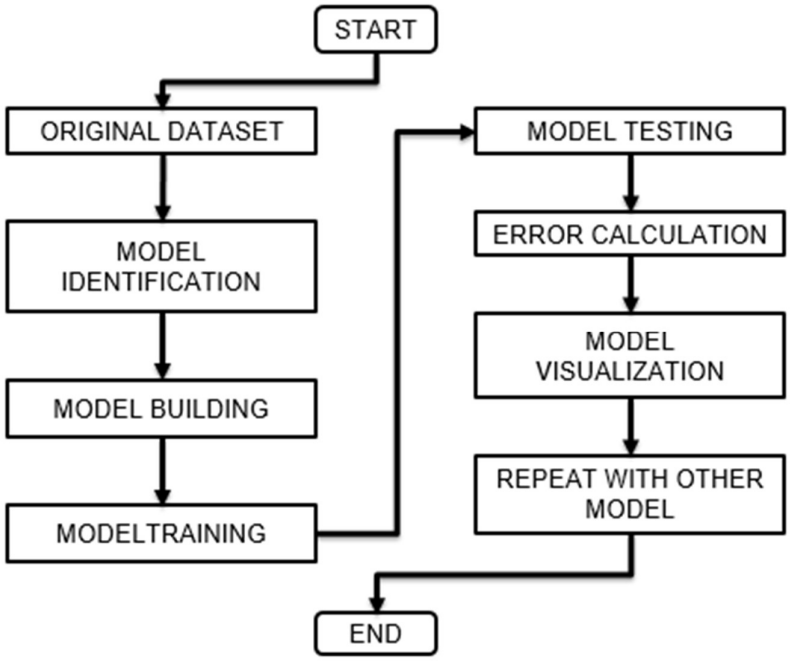

Figure 9: FLOWCHART DEFINING THE APPROACH OF OUR PROJECT

PYTHON

It has been chosen as the language of choice for this project. This was an easy decision for the multiple reasons.

1. Python as a language has an enormous community behind it. Any problems that might be encountered can be easily solved with a trip to Stack Overflow. Python is among the most popular languages on the site which makes it very likely there will be a direct answer to any query.
2. Python has an abundance of powerful tools ready for scientific computing. Packages such as Numpy, Pandas, and Scikit are freely available and well documented. Packages such as these can dramatically reduce, and simplify the code needed to write a given program. This makes iteration quick.
3. Python as a language is forgiving and allows for programs that look like pseudo code. This is useful when pseudocode given in academic papers needs to be implemented and tested. Using Python, this step is usually reasonably trivial.

However, Python is not without its flaws. The language is dynamically typed and packages are notorious for Duck Typing. This can be frustrating when a package method returns something that, for example, looks like an array rather than being an actual array. Coupled with the fact that standard Python documentation does not explicitly state the return type of a method, this can lead to a lot of trials and error testing that would not otherwise happen in a strongly typed language. This is an issue that makes learning to use a new Python package or library more difficult than it otherwise could be.

IMPORTING THE NECESSARY LIBRARIES

```
# importing libraries
import pandas as pd
import numpy as np
%matplotlib inline
from matplotlib import pyplot as plt
from pyramid.arima import auto_arima
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from fbprophet import Prophet
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
```

Figure 10: SCREENSHOT OF CODE, IMPORTING LIBRARIES

NUMPY

It is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

PANDAS

It is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays

and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

SCIKIT-LEARN

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machine, random forest, gradient boosting, k-means etc. It is mainly designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM .i.e., logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR.

PMDARIMA

Pmdarima (originally pyramid-arima, for the anagram of 'py' + 'arima') is a statistical library designed to fill the void in Python's time series analysis capabilities. It brings R's belove to Python, auto-arima making an even stronger case for why you don't need R for data science. It is 100% Python + Cython and does not leverage any R code, and implements a single, easy-to-use scikit-learn-esque estimator. Pmdarima wraps statsmodels under the hood, but is designed with an interface that's familiar to users coming from a scikit-learn background.

PROPHET

Implements 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 robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet follows the sklearn model API. We create an instance of the Prophet class and then call its fit and predict methods.

The input to Prophet is always a dataframe with two columns: ds and y. The ds (datestamp) column should be of a format expected by Pandas, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp. The y column must be numeric, and represents the measurement we wish to forecast.

KERAS

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

UNDERSTANDING THE DATASET

We'll be using a dataset from https://www.moneycontrol.com (you can find historical data for various stocks here) and for this particular project, we have used the data for 'STATE BANK OF INDIA'. We have collected data from January 2011 to December 2019 and after a successful implementation and training of our model, we will use the respective accuracies of the model to draw inferences to find the best suitable model out there which can be used for stock market prediction. The Data collected has been stored in the form of a CSV file; in Comma-Separated Values format.

IMPORTING THE DATASET

```
df = pd.read_csv('https://raw.githubusercontent.com/dipanshuagarwal/Stock-Market-Forecasting/master/Dataset/dataset11-19.csv')

print(df.head())
print('\n Shape of the data:')
print(df.shape)
```

Figure 11: SCREENSHOT OF CODE, IMPORTING THE DATASET

```
         Date   Open   High     Low   Close   Volume
0  31-12-2019  334.9  336.3  332.60  333.70   744106
1  30-12-2019  337.8  337.9  332.45  334.25   860732
2  27-12-2019  333.6  338.3  333.05  337.25  1249101
3  26-12-2019  333.0  333.2  328.05  329.85   660165
4  24-12-2019  332.7  334.2  329.25  331.30   750326

 Shape of the data:
(2226, 6)
```

Figure 12: SCREENSHOT OF OUTPUT OF CODE IN FIG. 11

There are multiple variables in the dataset – date, open, high, low, close, and volume.

- The columns Open and Close represent the starting and final price at which the stock is traded on a particular day.
- High and Low represent the maximum and minimum of the share for the day.
- Volume is the number of shares bought or sold in the day.

Another important thing to note is that the market is closed on weekends and public holidays.

The profit or loss calculation is usually determined by the closing price of a stock for the day, hence we will consider the closing price as the target variable.

CLEAN THE DATASET

In this we will be setting the Date variable as the Index Variable so that it can be used as a point of reference from now on, as well we will be creating a copy of dataset in sorted on which the work will be performed so that any new feature created does not affect the original data.

```
# setting the index as date
df['Date'] = pd.to_datetime(df.Date,format='%d-%m-%Y')
df.index = df['Date']

#sorting
data = df.sort_index(ascending=True, axis=0)
```

Figure 13: SCREENSHOT OF CODE, CLEANING THE DATASET

PLOT THE DATASET

Plot the target variable (Close) to understand how it's shaping up in our data

```
plt.figure(figsize=(16,8))
plt.plot(df['Close'], label='Close Price history')
```

Figure 14: SCREENSHOT OF CODE, PLOTTING THE GRAPH OF CLOSE

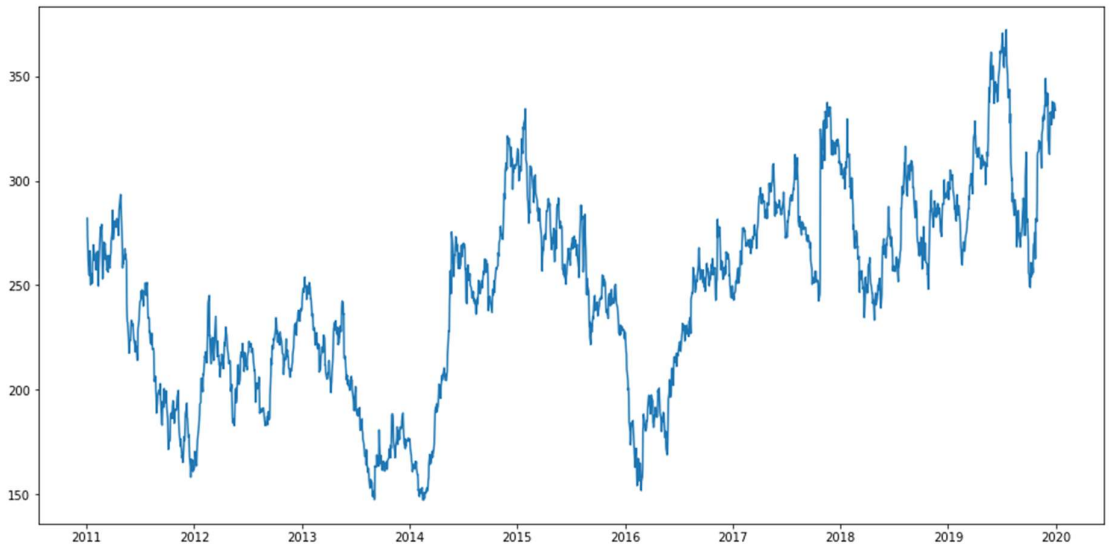

Figure 15: SCREENSHOT OF OUTPUT OF CODE IN FIG 14.

MODEL SELECTION

We selected a total of 6 models for implementation and training & testing of our Dataset. We will be comparing the Root Mean Squared Error value and Mean Absolute Percentage Error values of the same.

Moving Average Model: - 'Average' is easily one of the most common things we use in our day-to-day lives. For instance, calculating the average marks to determine overall performance, or finding the average temperature of the past few days to get an idea about today's temperature – these all are routine tasks we do on a regular basis. So this is a good starting point to use on our dataset for making predictions.

The predicted closing price for each day will be the average of a set of previously observed values. Instead of using the simple average, we will be using the moving average technique which uses the latest set of values for each prediction. In other words, for each subsequent step, the predicted values are taken into consideration while removing the oldest observed value from the set. Here is a simple figure that will help you understand this with more clarity.

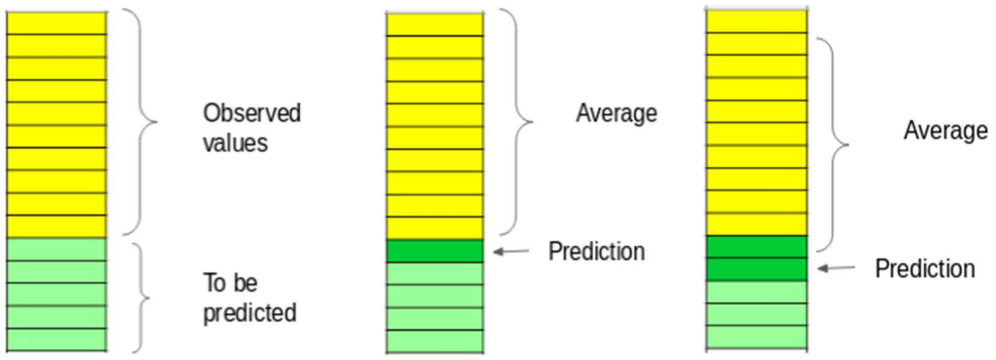

Figure 16: PICTORIAL DEPICTION OF HOW MOVING AVERAGE LOOKS LIKE

We will implement this technique on our dataset. The first step is to create a dataframe that contains only the Date and Close price columns, then split it into train and validation sets to verify our predictions.

```
#creating dataframe with date and the target variable
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

# NOTE: While splitting the data into train and validation set, we cannot use random splitting since that will destroy the time component.
# So here we have set the last year's data into validation and the 8 years' data before that into train set.
# splitting into train and validation
train = new_data[:1981]
valid = new_data[1981:]

# shapes of training set
print('\n Shape of training set:')
print(train.shape)

# shapes of validation set
print('\n Shape of validation set:')
print(valid.shape)

# In the next step, we will create predictions for the validation set and check the RMSE using the actual values.
# making predictions
preds = []
for i in range(0,valid.shape[0]):
    a = train['Close'][len(train)-245+i:].sum() + sum(preds)
    b = a/245
    preds.append(b)
```

Figure 17: SCREENSHOT OF CODE, IMPLEMENTION OF MOVING AVERAGE

```
 Shape of training set:
(1981, 2)

 Shape of validation set:
(245, 2)
```

Figure 18: OUTPUT OF MOVING AVERAGE FIGURE 17

```
# checking the results (RMSE value)
ma_rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-preds),2)))
print('\n RMSE value on validation set:')
print(ma_rms)

ma_mape=(np.mean((np.array(valid['Close'])-preds)))
print('\n MAPE value on validation set:')
print(ma_mape)
```

Figure 19: CLACULATION OF VALUES

```
 RMSE value on validation set:
44.46086937909403

 MAPE value on validation set:
33.3288560620692
```

Figure 20: OUTPUT OF CODE IN FIG 19

Just checking the RMSE does not help us in understanding how the model performed. Let's visualize this to get a more intuitive understanding. So here is a plot of the predicted values along with the actual values.

```
valid['Predictions'] = 0
valid['Predictions'] = preds
plt.figure(figsize=(16,8))
plt.plot(train['Close'])
plt.plot(valid[['Predictions', 'Close']])
```

Figure 21: PLOTTING OF PREICTED VERSUS ACTUAL

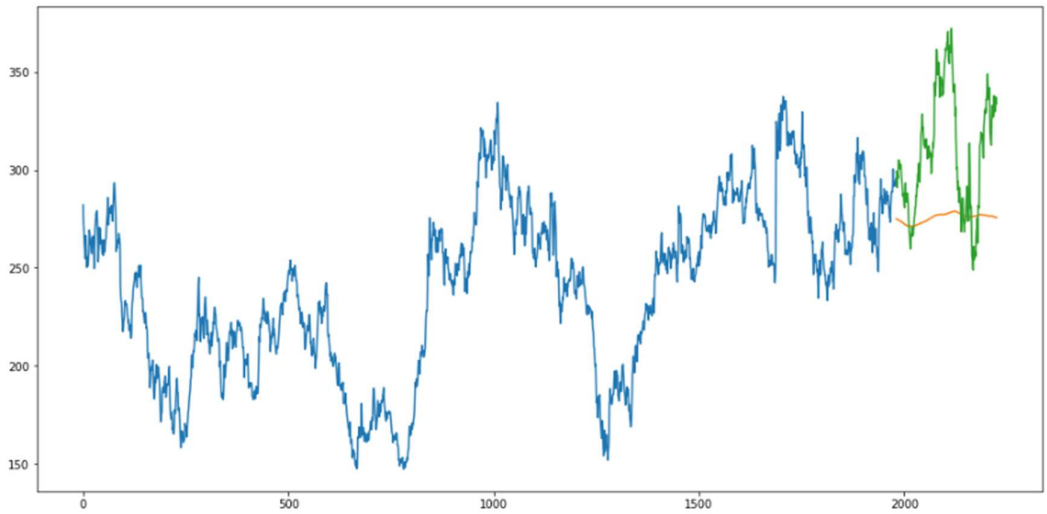

Figure 22: GRAPH DEPICTING PREDICTED (ORANGE) VS ACTUAL (GREEN)

The RMSE value is close to 44 but the results are not very promising (as you can gather from the plot). The predicted values are of the same range as the highest of observed values in the train set.

Linear Regression: - The most basic machine learning algorithm that can be implemented on this data is linear regression. The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable.

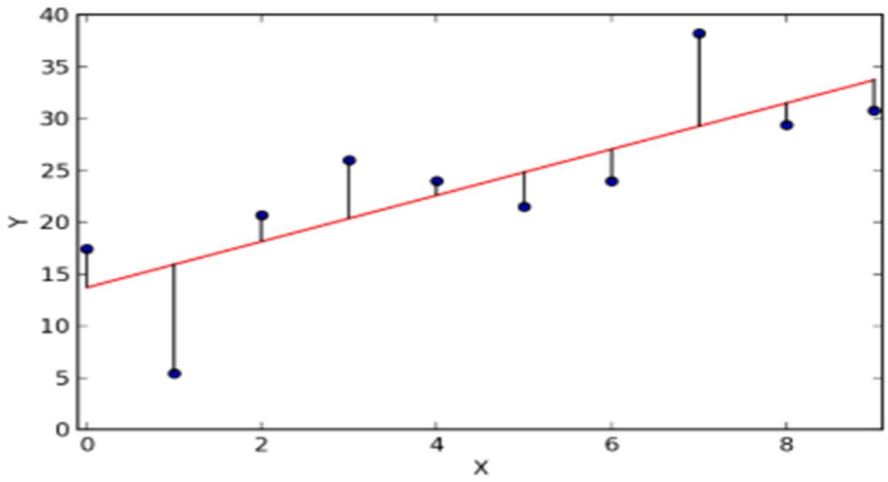

Figure 23: HOW LINEAR REGRESSION LOOKS LIKE WHEN IT WORKS

For our problem statement, we do not have a set of independent variables. We have only the dates instead. Let us use the date column to extract features like – day, month, year, mon/Friday etc. and then fit a linear regression model.

```
#creating a separate dataset
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

#create features
from fastai.structured import add_datepart
add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)  #elapsed will be the time stamp

new_data['mon_fri'] = 0
for i in range(0,len(new_data)):
    if (new_data['Dayofweek'][i] == 0 or new_data['Dayofweek'][i] == 4):
        new_data['mon_fri'][i] = 1
    else:
        new_data['mon_fri'][i] = 0

#split into train and validation
train = new_data[:1981]
valid = new_data[1981:]

x_train = train.drop('Close', axis=1)
y_train = train['Close']
x_valid = valid.drop('Close', axis=1)
y_valid = valid['Close']

#implement linear regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)

#make predictions
preds = model.predict(x_valid)
```

Figure 24: SCREENSHOT OF CODE, IMPLEMENTION OF LINEAR REGRESSION

This creates features such as:

'Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear', 'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', 'Is_year_end', and 'Is_year_start'.

Our hypothesis is that the first and last days of the week could potentially affect the closing price of the stock far more than the other days. So I have created a feature that identifies whether a given day is Monday/Friday or Tuesday/Wednesday/Thursday.

```
lr_rms=np.sqrt(np.mean(np.power((np.array(y_valid)-np.array(preds)),2)))
print('\n RMS value on validation set:')
print(lr_rms)

lr_mape=np.mean(np.array(np.array(y_valid)-np.array(preds)))
print('\n MAPE value on validation set:')
print(lr_mape)
```

Figure 25: Calculation of Error Values

```
 RMS value on validation set:
38.43329798565597

 MAPE value on validation set:
25.71709195169567
```

Figure 26: Error Values

The RMSE value is still higher than the desired, which clearly shows that linear regression has performed poorly. Let's look at the plot and understand why linear regression has not done well:

```
valid['Predictions'] = 0
valid['Predictions'] = preds

valid.index = new_data[1981:].index
train.index = new_data[:1981].index

plt.figure(figsize=(16,8))
plt.plot(train['Close'])
plt.plot(valid[['Predictions', 'Close']])
```

Figure 27: PLOTTING OF GRAPH

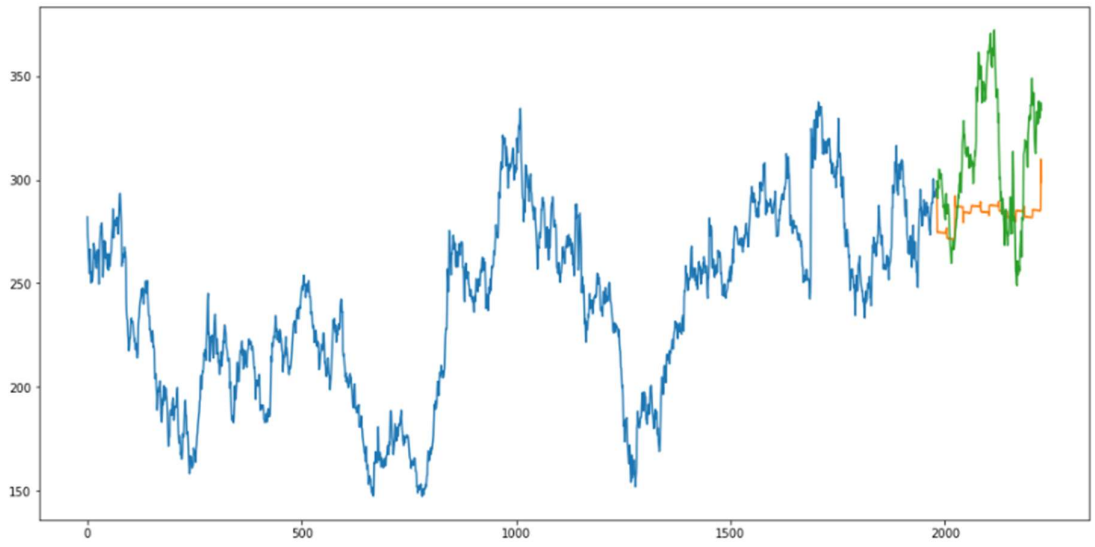

Figure 28: GRAPH OF PREDICTED (ORANGE) VS ACTUAL (GREEN)

Linear regression is a simple technique and quite easy to interpret, but there are a few obvious disadvantages. One problem in using regression algorithms is that the model overfits to the date and month column. Instead of taking into account the previous values from the point of prediction, the model will consider the value from the same *date* a month ago, or the same *date/month* a year ago.

K-Nearest Neighbors: -Another interesting ML algorithm that one can use here is K-NN (K-Nearest Neighbours). Based on the independent variables, K-NN finds the similarity between new data points and old data points.

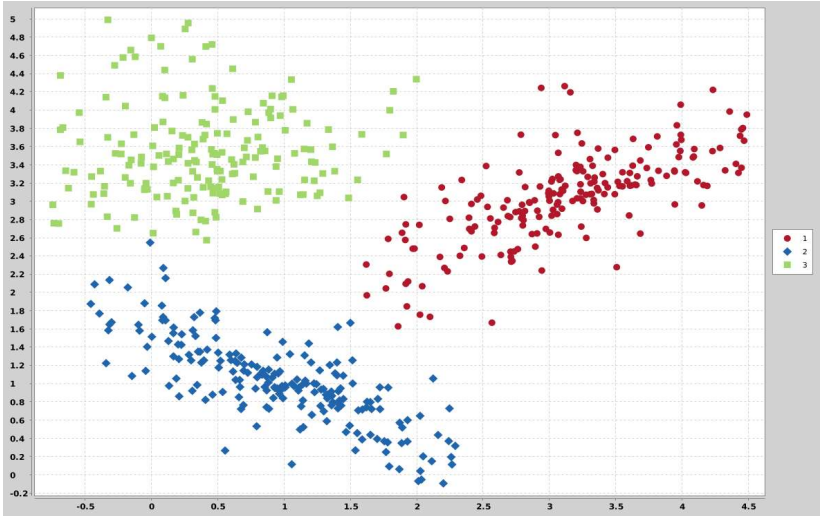

Figure 29: DEPICTING HOW K-NEAREST NEIGBORS LOOKS LIKE

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

#creating a separate dataset
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

#create features
from fastai.structured import add_datepart
add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)  #elapsed will be the time stamp

new_data['mon_fri'] = 0
for i in range(0,len(new_data)):
    if (new_data['Dayofweek'][i] == 0 or new_data['Dayofweek'][i] == 4):
        new_data['mon_fri'][i] = 1
    else:
        new_data['mon_fri'][i] = 0

#split into train and validation
train = new_data[:1981]
valid = new_data[1981:]

x_train = train.drop('Close', axis=1)
y_train = train['Close']
x_valid = valid.drop('Close', axis=1)
y_valid = valid['Close']

```

```
#scaling data
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_valid_scaled = scaler.fit_transform(x_valid)
x_valid = pd.DataFrame(x_valid_scaled)

#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)

#fit the model and make predictions
model.fit(x_train,y_train)
preds = model.predict(x_valid)
```

Figure 30: IMPLEMENTATION OF K-NN

```
knn_rms=np.sqrt(np.mean(np.power((np.array(y_valid)-np.array(preds)),2)))
print('\n RMS value on validation set:')
print(knn_rms)

knn_mape=np.mean(np.array(np.array(y_valid)-np.array(preds)))
print('\n MAPE value on validation set:')
print(knn_mape)
```

Figure 31: CALCULATION OF ERROR VALUES

```
 RMS value on validation set:
94.68023402139535

 MAPE value on validation set:
87.07895691609978
```

Figure 32: ERROR VALUES

```
valid['Predictions'] = 0
valid['Predictions'] = preds

plt.figure(figsize=(16,8))
plt.plot(valid[['Close', 'Predictions']])
plt.plot(train['Close'])
```

Figure 33: PLOT THE GRAPH

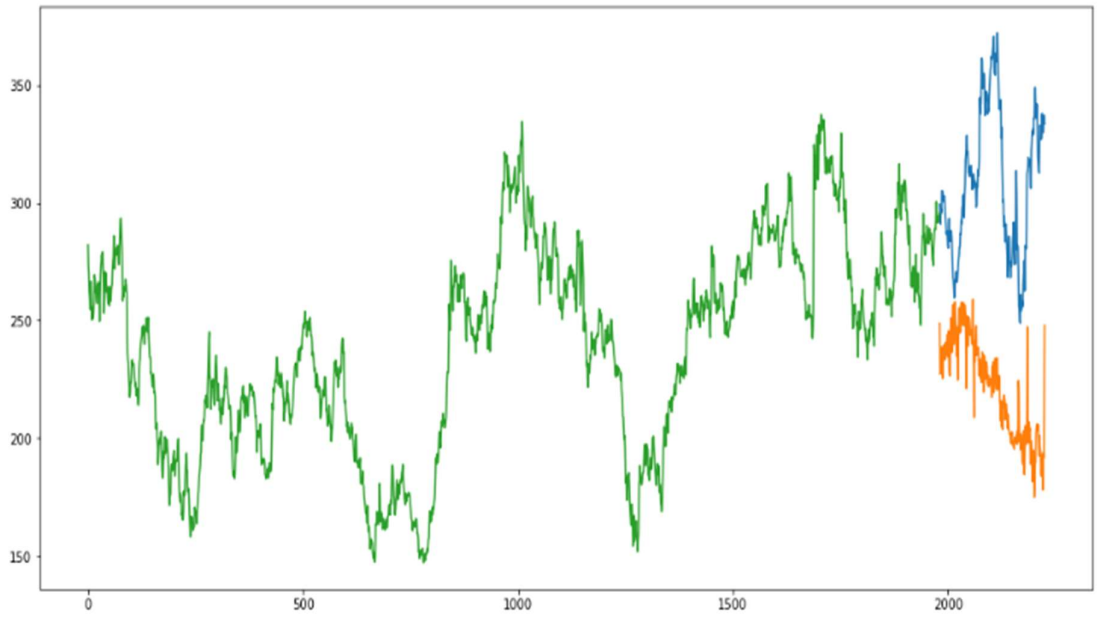

Figure 34: GRAPH SHOWING THE PREDICTED (ORANGE) VS ACTUAL (BLUE)

The RMSE value is almost similar to the linear regression model and the plot shows the same pattern. We can safely say that regression algorithms have not performed well on this dataset.

fbPhrophet: - There are a number of time series techniques that can be implemented on the stock prediction dataset, but most of these techniques require a lot of data preprocessing before fitting the model. Prophet, designed and pioneered by Facebook, is a time series forecasting library that requires no data preprocessing and is extremely simple to implement. The input for Prophet is a dataframe with two columns: date and target (ds and y). Prophet tries to capture the seasonality in the past data and works well when the dataset is large.

```
#creating dataframe
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

new_data['Date'] = pd.to_datetime(new_data.Date,format='%d-%m-%Y')
new_data.index = new_data['Date']

#preparing data
new_data.rename(columns={'Close': 'y', 'Date': 'ds'}, inplace=True)

#train and validation
train = new_data[:1981]
valid = new_data[1981:]

#fit the model
model = Prophet()
model.fit(train)

#predictions
close_prices = model.make_future_dataframe(periods=len(valid))
forecast = model.predict(close_prices)

forecast_valid = forecast['yhat'][1981:]
```

Figure 35: IMPLEMENTATION OF fbPROPHET MODEL

```
#rmse
p_rms=np.sqrt(np.mean(np.power((np.array(valid['y'])-np.array(forecast_valid)),2)))
print('\n RMS value on validation set:')
print(p_rms)

p_mape=abs(np.mean(np.array(np.array(forecast_valid)-np.array(valid['y']))))
print('\n MAPE value on validation set:')
print(p_mape)
```

Figure 36: CALCULATION OF ERROR VALUES

```
 RMS value on validation set:
50.31837619277186

 MAPE value on validation set:
39.030956806155714
```

Figure 37: ERROR VALUES

```
valid['Predictions'] = 0
valid['Predictions'] = forecast_valid.values

plt.figure(figsize=(16,8))
plt.plot(train['y'])
plt.plot(valid[['Predictions', 'y']])
```

Figure 38: PLOTTING OF GRAPH

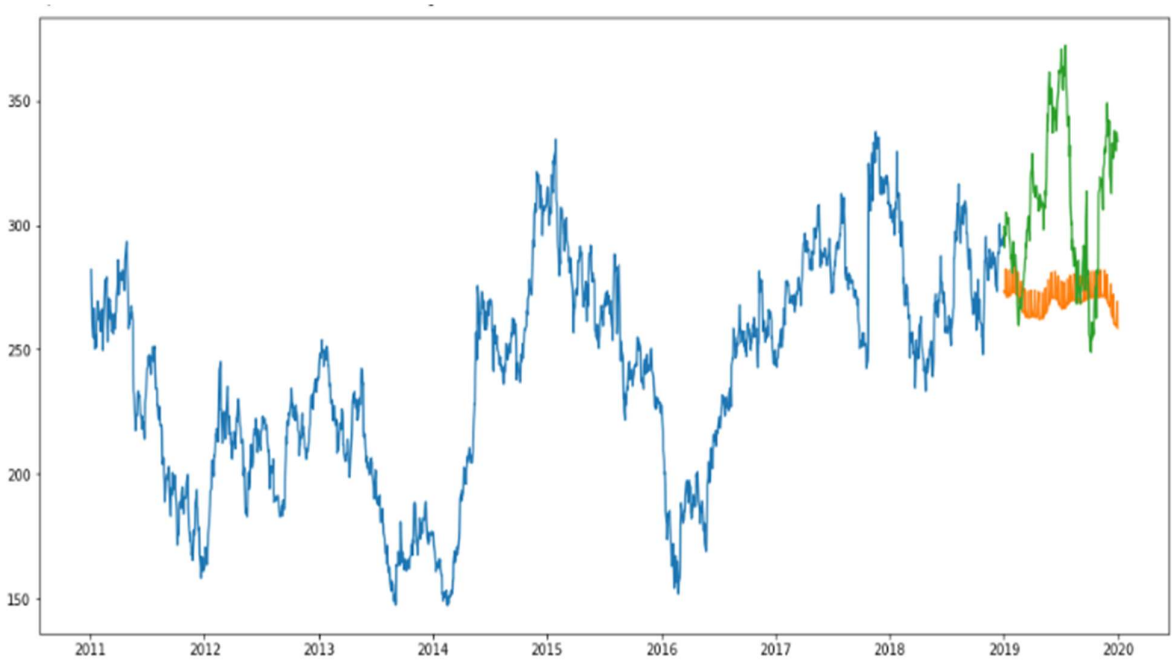

Figure 39: GRAPH DEPICITING PREDICTED (ORANGE) VS ACTUAL (GREEN)

Prophet (like most time series forecasting techniques) tries to capture the trend and seasonality from past data. This model usually performs well on time series datasets, but fails to live up to its reputation in this case.

AUTO-ARIMA: - ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:

p (past values used for forecasting the next value)
q (past forecast errors used to predict the future values)
d (order of differencing)

Parameter tuning for ARIMA consumes a lot of time. So we will use auto ARIMA which automatically selects the best combination of (p, q, and d) that provides the least error.

```
train = data[:1981]
valid = data[1981:]

training = train['Close']
validation = valid['Close']

model = auto_arima(training, start_p=1, start_q=1,max_p=3, max_q=3, m=12,start_P=0,
                   seasonal=True,d=1, D=1, trace=True,error_action='ignore',suppress_warnings=True)
model.fit(training)

forecast = model.predict(n_periods=245)
forecast = pd.DataFrame(forecast,index = valid.index,columns=['Prediction'])
```

Figure 40: IMPLEMENTATION OF AUTO-ARIMA

```
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 1, 12); AIC=11933.830, BIC=11961.754, Fit time=28.180 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 12); AIC=13170.028, BIC=13181.198, Fit time=0.343 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 12); AIC=12647.224, BIC=12669.563, Fit time=4.806 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 12); AIC=11933.118, BIC=11955.457, Fit time=23.496 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(1, 1, 1, 12); AIC=11932.170, BIC=11960.094, Fit time=21.904 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(1, 1, 0, 12); AIC=12646.561, BIC=12668.900, Fit time=5.304 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(1, 1, 2, 12); AIC=11936.583, BIC=11970.092, Fit time=53.648 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 12); AIC=13156.360, BIC=13173.115, Fit time=1.864 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(2, 1, 2, 12); AIC=11933.705, BIC=11972.799, Fit time=40.879 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(1, 1, 1, 12); AIC=11932.936, BIC=11966.444, Fit time=34.467 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 12); AIC=11941.345, BIC=11963.684, Fit time=15.444 seconds
Fit ARIMA: order=(0, 1, 2) seasonal_order=(1, 1, 1, 12); AIC=11932.823, BIC=11966.332, Fit time=27.634 seconds
Fit ARIMA: order=(1, 1, 2) seasonal_order=(1, 1, 1, 12); AIC=11933.463, BIC=11972.556, Fit time=40.744 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(2, 1, 1, 12); AIC=11934.170, BIC=11967.678, Fit time=46.315 seconds
Total fit time: 345.036 seconds
```

Figure 41: OUTPUT OF MODEL FIT FUNCTION

```
a_rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-np.array(forecast['Prediction'])),2)))
print('\n RMS value on validation set:')
print(a_rms)

a_mape=np.mean(np.array(np.array(forecast['Prediction']-np.array(valid['Close']))))
print('\n MAPE value on validation set:')
print(a_mape)
```

Figure 42: CALCULATION OF ERROR VALUES

```
 RMS value on validation set:
31.84104405718795

 MAPE value on validation set:
7.353536006014123
```

Figure 43: ERROR VALUES

```
plt.figure(figsize=(16,8))
plt.plot(train['Close'])
plt.plot(valid['Close'])
plt.plot(forecast['Prediction'])
```

Figure 44: PLOTTING THE GRAPH

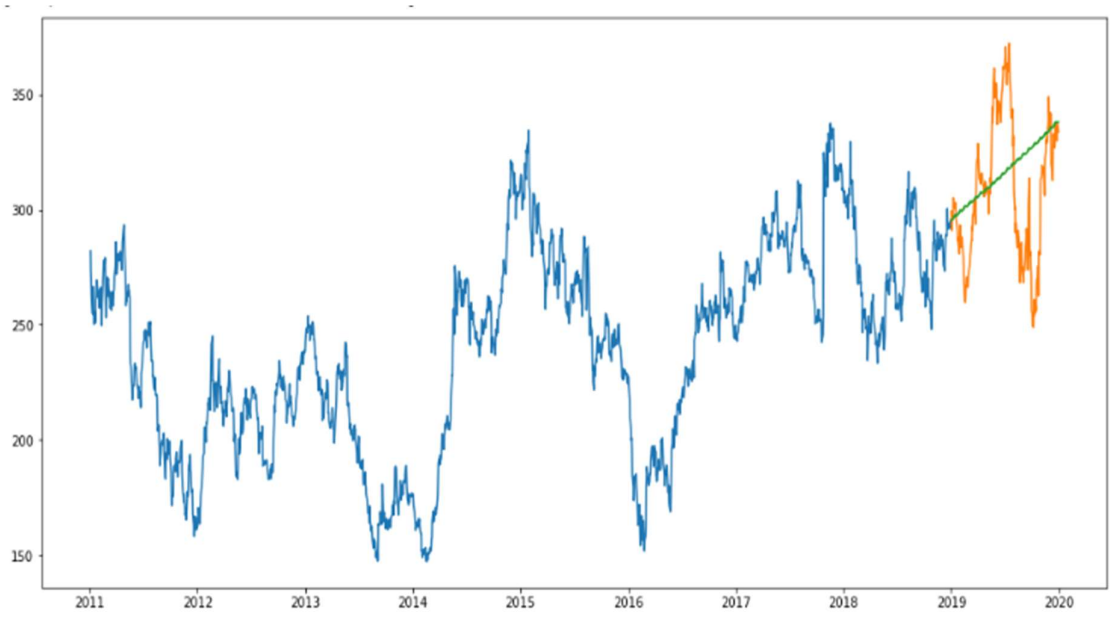

Figure 45: GRAPH DEPICTING PREDICTED (ORANGE) VS ACTUAL (GREEN)

As we saw earlier, an auto ARIMA model uses past data to understand the pattern in the time series. Using these values, the model captured an increasing trend in the series. Although the predictions using this technique are far better than that of the previously implemented machine learning models, these predictions are still not close to the real values.

As it's evident from the plot, the model has captured a trend in the series, but does not focus on the seasonal part.

As it turns out, stock prices do not have a particular trend or seasonality. It highly depends on what is currently going on in the market and thus the prices rise and fall. Hence forecasting techniques like ARIMA, SARIMA and Prophet would not show good results for this particular problem.

Let us go ahead and try another advanced technique – Long Short Term Memory (LSTM).

LONG SHORT TERM MEMORY: - Long Short Term Memory (LSTM) are widely used for sequence prediction problems and have proven to be extremely effective. The reason they work so well is because LSTM is able to store past information that is important, and forget the information that is not. LSTM has three gates:

- The input gate: The input gate adds information to the cell state

- The forget gate: It removes the information that is no longer required by the model
- The output gate: Output Gate at LSTM selects the information to be shown as output

```
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

#setting index
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)

#creating train and test sets
dataset = new_data.values

train = dataset[0:1981,:]
valid = dataset[1981:,:]

#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

x_train, y_train = [], []
for i in range(60,len(train)):
    x_train.append(scaled_data[i-60:i,0])
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

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

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))

```

```
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=32, batch_size=50, verbose=2)

#predicting 246 values, using past 60 from the train data
inputs = new_data[len(new_data) - len(valid) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    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))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

Figure 46: IMPLEMENTATION OF LSTM MODEL

```
Data with input dtype object was converted to float64 by MinMaxScaler.

Epoch 1/32
 - 4s - loss: 0.0198
Epoch 2/32
 - 3s - loss: 0.0027
Epoch 3/32
 - 3s - loss: 0.0024
Epoch 4/32
 - 3s - loss: 0.0022
Epoch 5/32
 - 3s - loss: 0.0020
Epoch 6/32
 - 3s - loss: 0.0019
Epoch 7/32
 - 3s - loss: 0.0017
Epoch 8/32
 - 3s - loss: 0.0017
Epoch 9/32
 - 3s - loss: 0.0016
Epoch 10/32
 - 3s - loss: 0.0015
Epoch 11/32
 - 3s - loss: 0.0015
Epoch 12/32
 - 3s - loss: 0.0014
Epoch 13/32
 - 3s - loss: 0.0013
Epoch 14/32
 - 3s - loss: 0.0013
Epoch 15/32
 - 3s - loss: 0.0012
Epoch 16/32
 - 3s - loss: 0.0012
Epoch 17/32
 - 3s - loss: 0.0012
Epoch 18/32
 - 3s - loss: 0.0012
Epoch 19/32
 - 3s - loss: 0.0011
Epoch 19/32
 - 3s - loss: 0.0011
Epoch 20/32
 - 3s - loss: 0.0010
Epoch 21/32
 - 3s - loss: 0.0011
Epoch 22/32
 - 3s - loss: 0.0010
Epoch 23/32
 - 3s - loss: 9.2008e-04
Epoch 24/32
 - 3s - loss: 8.7276e-04
Epoch 25/32
 - 3s - loss: 8.6261e-04
Epoch 26/32
 - 3s - loss: 8.5241e-04
Epoch 27/32
 - 3s - loss: 8.2616e-04
Epoch 28/32
 - 3s - loss: 8.0957e-04
Epoch 29/32
 - 3s - loss: 7.8188e-04
Epoch 30/32
 - 3s - loss: 8.4848e-04
Epoch 31/32
 - 3s - loss: 7.4284e-04
Epoch 32/32
 - 3s - loss: 7.4666e-04
```

Figure 47: OUTPUT OF MODEL TARINING

```
1 lstm_rms=np.sqrt(np.mean(np.power((valid-closing_price),2)))
2 print('\n RMS value on validation set:')
3 print(lstm_rms)
4
5 lstm_mape=abs(np.mean(valid-closing_price))
6 print('\n MAPE value on validation set:')
7 print(lstm_mape)
```

Figure 48: CALCULATION OF ERROR VALUES

```
 RMS value on validation set:
7.3527248670339995

 MAPE value on validation set:
1.0907925103635212
```

Figure 49: ERROR VALUES

```
#for plotting
train = new_data[:1981]
valid = new_data[1981:]
valid['Predictions'] = closing_price
plt.figure(figsize=(16,8))
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
```

Figure 50: PLOTTING THE GRAPH

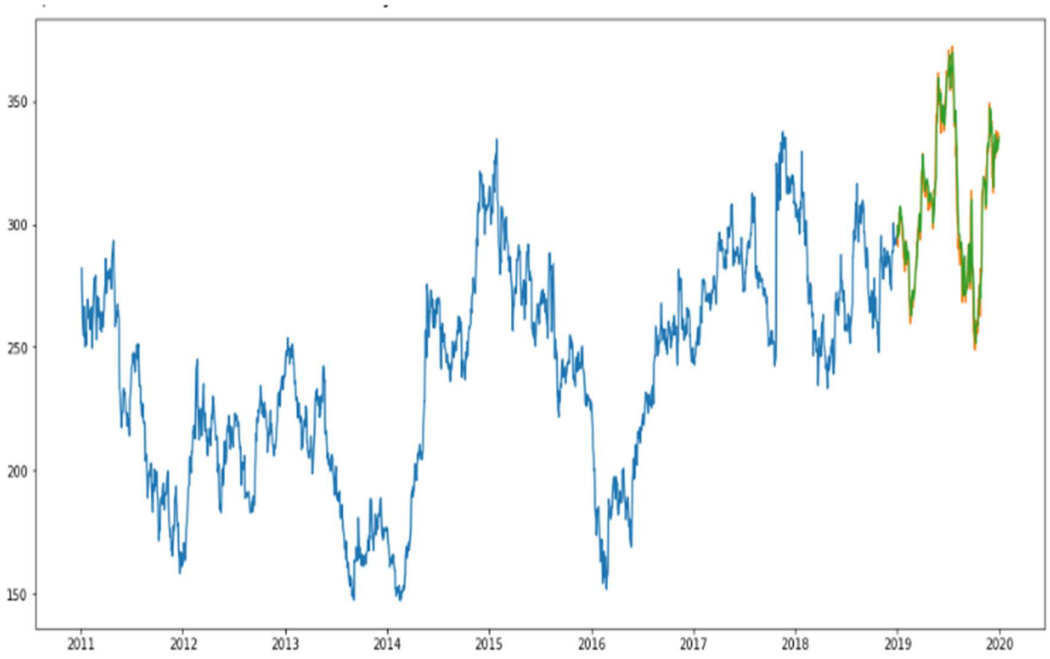

Figure 51: GRAPH DEPICTING THE PREDICTED (ORANGE) AND ACTUAL (GREEN)

Wow! The LSTM model can be tuned for various parameters such as changing the number of LSTM layers, adding dropout value or increasing the number of epochs.

CONCLUSION

For comparison of all the models till now we will plot the values of RMSE and MAPE obtained for each of the model. Then we will derive further conclusions based on the values, such as the optimal model out of all the implanted ones, which of the models are same as to that of their predicting natures and such on.

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.grid(True)
ax.set_title('Mape Values')
mods = ['Moving Avg', 'Linear Regr.', 'K-NN', 'fbProphet', 'Auto-Arima','LSTM']
maval = [ma_mape,lr_mape,knn_mape,p_mape,a_mape,lstm_mape]
ax.bar(mods,maval)
plt.show()

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.grid(True)
ax.set_title('RMS Values')
mods = ['Moving Avg', 'Linear Regr.', 'K-NN', 'fbProphet', 'Auto-Arima','LSTM']
rmval = [ma_rms,lr_rms,knn_rms,p_rms,a_rms,lstm_rms]
ax.bar(mods,rmval)
plt.show()
```

Figure 52: PLOTTING GRAPH FOR ALL THE RMSE AND MAPE VALUES

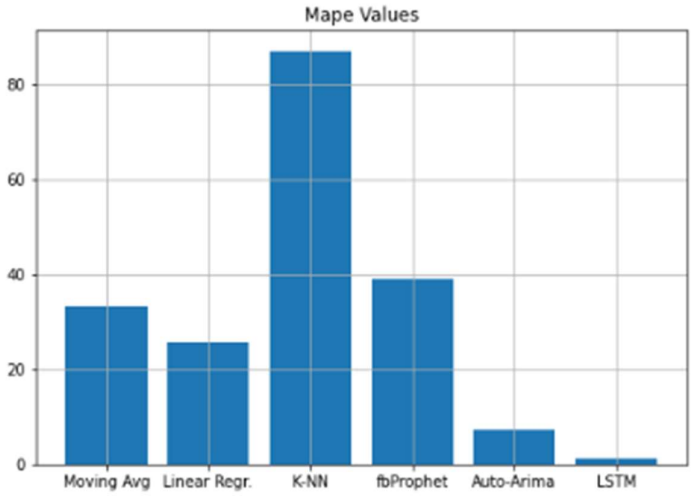

Figure 53: GRAPH OF COMPARISON OF MAPE VALUES

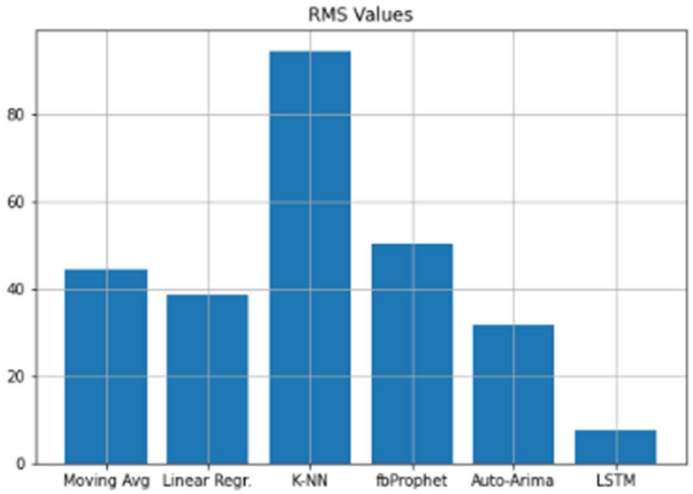

Figure 54: GRAPH OF COMPARISON OF RMS VALUES

Table 1: COMPARITIVE STUDY OF RMSE & MAPE VALUES (ODER BY EFFICIENCY)

MODEL NAME	RMSE VALUE	MAPE VALUE
Long Short Term Memory	7.35	1.09
Auto Arima	31.84	7.35
Linear Regression	38.43	25.72
Moving Average	33.33	44.46
fbProphet	50.32	39.03
K-Nearest Neighbors	94.68	87.08

As we can see in Table 1, LSTM is the most efficient model which was implemented on our particular dataset, because as LSTM is an R-NN model, which is able to selectively work on the past data i.e. only on the needed values which may effect on the trend and seasonality, thus showing a much more effective outcome and very less loss in the predictions.

The second best model implemented was the Auto-Arima model, the Arima model was able to effectively capture the trend which was present in our dataset, but it could not work with the seasonality in order to minimize the loss. We can still improve the efficiency of the arima model if we were to hyper-tune its p, d, and q variables, but even then it may not perform upto the standards of the LSTM model on an equal no of hyper-tuned parameters.

The KNN and Linear Regression models did not work well with our particular time-series dataset, since we were only focused on the variables, DATE and CLOSE, and we were predicting the CLOSE so it had only DATE as a viable option for feature extraction, and we all know that from a DATE, one can only study it TIME, DAY, MONTH, WEEK and YEAR. There are not a lot of different features which can be worked upon thus, they fail to even predict the trend much less the seasonality.

The Prophet model which was specifically designed the Facebook, to work with TIME-SERIES, has failed in our case because, a Stock Market Dataset may be a Time-Series Dataset, but at the end of the day, it does not follows as strict trend or seasonality, as the other Time-Series, because there are other factors also present in the real-world which may bring about a drastic change in Stock Prices.

End Note:

The LSTM model was the best model implemented out of all to work with Time-Series. But are the predictions from LSTM enough to identify whether the stock price will increase or decrease? Certainly not!

As we know, stock price is affected by the news about the company and other factors like demonetization or merger/demerger of the companies. There are certain intangible factors as well which can often be impossible to predict beforehand.

Time series forecasting is a very intriguing field to work with, as we have realized during the time spent on this project. There is a perception in the community that it's a complex field, and while there is a grain of truth in there, it's not so difficult once you get the hang of the basic techniques.

REFERENCES

1. A 6 Step Field Guide for Building Machine Learning Projects by Daniel Bourke
2. Introduction to Time Series Forecasting With Python by Jason Brownlee
3. Contreras, 1. Espinola, R.NogaJes, F1.and conejo,AJ.(2003) "ARIMA models to predict next day electricity prices", IFEE transactions on power system, vo1.18, noJ,pp: I 014-1 020.
4. Kumar; K Yadav;A.KSingh, M.P; Hassan and H.Jain,V.K(2004)"Forecasting Daily Maximum Surface Ozone".
5. Tsitsika,E.V;Maravelias,C.D& Haralatous,J. (2007)"Modelling and forecasting pelagic fish production using univariate and multivariate ARIMA models". Fisheries science volume 73,pp:979-988.
6. Datta K.(2011)"ARIMA forecasting of Inflation in the Bangladesh Economy",The IUP journal of bank management,voI.X,No.4,pp-7-15.
7. D. Banerjee, "Forecasting of Indian stock market using time-series ARIMA model," 2014 2nd International Conference on Business and Information Management (ICBIM), Durgapur, 2014, pp. 131-135, doi: 10.1109/ICBIM.2014.6970973.
8. “Introduction to Time Series Analysis and Forecasting” by Douglas C. Montgomery, Cheryl L. Jennings, and Murat Kulahci
9. “Time Series Analysis: Forecasting and Control” by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung
10. Chris Chatfield, "The analysis of time series An introduction"