School of Electronic Engineering and Computer Science

MSc Computing and Information Systems
Project Report 2019

Machine Learning Stock Prediction

Theodore Sanfred Hinanto

December 2019

Acknowledgements

I would like to thank my supervisor Dr Nikos Tvelekos of the School of Electronic Engineering and Computer Science at Queen Mary University of London (QMUL); the inventors of Python, Anaconda, Jupyter, Anaconda, NumPy, Scikit-learn, Tensorflow, Pandas, Matplotlib, Seaborn, ARIMA, Linear Regression, RNN, LSTM, Prophet, and World Trading Data; and QMUL for providing Data Science class which provided me the first step to learn about Machine Learning.

I must express my gratitude to God and my parents who have been providing me continuous support and encouragement throughout my study. This thesis would not be possible without them.

Thank you.

Theodore Hinanto

ABSTRACT

Ever since the advent of Artificial Intelligence, financial advisers, stockbrokers, and even researchers have been attempting different approaches to exploit the predictive prowess of statistical and neural network techniques. The availability of high-performance computers and the sheer size of data generated daily make AI the go-to choice for making highly accurate predictions in a relatively short time.

This project aims to compare the performance of Linear Regression, Long Short-Term Memory (LSTM), Prophet, and Autoregressive Integrated Moving Average (ARIMA), with a manual double simple moving average (SMA) backtesting algorithm.

High-quality historical data will be required to develop a valid predictive algorithm. WorldTrading– a platform that provides high-quality and historical forex, index, and stock price data is used with a dedicated API (Application Programming Interface) to feed the algorithm with training, validation, and test data in the ratio 60-20-20 respectively.

After developing the predictive model, a standard statistical error measurement technique, Root Mean Square Error (RMSE) is used to measure the disparity between predicted values and the test set. This project focuses on predicting the trend rather than the exact stock price of a selected company. This aims to help the end-user to know which approach is favorable and also, to make buying and selling decisions in the stock market.

The results obtained established that of the four approaches used, Prophet performs best in predicting stock dipping and rising trends. Comparing Prophet with a backtesting algorithm proves that AI-based methods do not outperform the manual double SMA Backtesting algorithm developed.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1

1. INTRODUCTION

Predicting stock market prices has been a widely researched topic (Chang et al., 2013) as highly accurate models can lead to significant profit for investors and stockbrokers. The availability of historical and real-time data is also a contributing factor to the increased interest in this field. Developing a model that guarantees a high-profit margin is difficult as stock price data are non-stationary and chaotic, owing to the "wavy" nature of the stock market.

Machine Learning (ML) and Neural Networks (NNs) are subsets of Artificial Intelligence that can learn relationships between different types of data (Gilbert & Karahalios, 2010). ML algorithms are majorly based on classical statistical techniques. An example is the Autoregressive Integrated Moving Average (ARIMA) framework that will be used in this project. Artificial Neural Networks, as introduced by Alan Turing in 1948 (Alan, 1948), are inspired by biological neuron activities in the brain and are tasked with learning patterns in a given data to develop models that detect complex relationships between input and output data to extract features, classify items, or predict future occurrences. Examples of Neural Network architectures are Convolutional Neural Networks, Generative Adversarial Networks and Long Short-Term Memory (LSTM) - which will be used as one of the frameworks for comparative analysis in this project.

A platform where buyers and sellers of ownership claims to businesses (otherwise called shares) converge to trade units of these "*share*s" is described as a stock market. Several commercial companies trade units of their company shares through stock exchanges for interested buyers, also known as shareholders. Prices of these shares dip (fall) or rise, depending on the influx of

buyers or the number of shareholders that sell their shares; this is in turn, a factor of company policies, change in leadership, innovations, and other financial or economic news around the company (Robert & Hsinchun, 2009). This irrational behavior has made profit-seeking stockbrokers, and investors rely on the predictive ability of AI architectures as they have been proven to learn intricate patterns between input and output data (Mohit & Sachchidanand, 2015).

To develop any meaningful predictive algorithm, the model needs to be fed with adequate data. For this project, primary stock price data of up to 5 years provided by WorldTrading.com was used. As common with big data, there are missing rows in the data. These missing data are handled, as shown in the third chapter of this report.

1.1. AIM

The specific aim of this project is to perform comparative analysis using four AI based architectures– LSTM, Prophet, Linear Regression and ARIMA with a manual "double Simple Moving Average" (double SMA) backtesting algorithm.

1.2. OBJECTIVES

This project will achieve its stated aim by following the following objectives:

- develop AI-based predictive algorithms using LSTM, Prophet, and ARIMA;
- develop a manual double SMA backtesting algorithm;
- calculate the Root Mean Square Error (RMSE) of all developed algorithms; and
- Perform a comparative analysis to detect the best performing algorithm.

1.3. MOTIVATION

It has been widely believed that since AI-based architectures are excellent for time-series predictions, they should be excellent in forecasting stock prices given necessary historical stock price data of opening price, closing price, daily highest, and daily lowest prices. This project shows that AI architectures tend to predict trends better than the exact stock price.

The assumption that machine learning algorithm generally performs better than traditional Simple Moving Average (SMA) techniques can be detrimental if it is solely adopted by a broker or investor. This research gives a practical basis to test if that assumption is correct or not. The justification for this research comes from the fact that stock prices are volatile, and they can easily, in little to no time, cause significant losses.

CHAPTER 2

2. LITERATURE REVIEW

Several researchers and investors have developed one AI-based trading system or the other. Only a few, if not none, have been deployed for real-life applications as they haven't surpassed the profit-making capabilities of top broking firms. Due to competitiveness and the drive to make maximum profit, highly successful models developed by researchers might not be published. This section summarizes some recent approaches to stock price prediction and the result obtained from the algorithms, theoretical background of the AI architectures that were used, and a description of the dataset used to train and validate the algorithms.

2.1. THEORETICAL BACKGROUND

The techniques employed in this project are the offspring of AI. They are developed with a technical background based on some theoretical principles. This section describes the basic working principles of these architectures.

2.1.1. AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA)

ARIMA is a statistical analysis technique based on regression analysis that is used to predict future trends in time series data (Box & Jenkins, 1970). An ARIMA model has three components.

- AutoRegressive (AR): This refers to the component of the model that indicates a moving variable that regresses on its previous values. This is represented with the variable ‘p’ in the ARIMA equation, as shown in equation 2.1.
- Integrated (I): ARIMA makes time-series observations stationery. This component represents the differencing of raw observations, which means that new data values are replaced by the difference between previous values and the new values. This component is denoted with ‘d’.
- Moving Average (MA): combines the dependency between a residual error and an observation from a moving average model applied to lagged observations. Denoted with ‘q’. This is the size of the moving window.

$$y'_t = I + \alpha_1 y'_{t-1} + \alpha_2 y'_{t-2} + \cdots + \alpha_p y'_{t-p} + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \cdots + \theta_q e_{t-q}$$ (Walter, 1988) 2.1

Equation 2.2 shows the lagged p data points are the predictors for the autoregressive component of the model. The lagged q errors denote the moving average part, which is differenced. The prediction is the differenced y_t. This is termed the ARIMA (p, d, and q) model. ARIMA approximates the coefficients α and θ for a given p, d, q, to learn from the training data in a time series data. Specifying p, d, q is done by trying out different permutations and assessing the model’s performance.

2.1.2. LINEAR REGRESSION

Linear Regression is a supervised machine learning algorithm that models the relationship between a dependent variable with one or more independent variables. Regression models have an aim of predicting values built on independent variables. It is commonly used to establish a relationship between variables and forecasting (Francis,

1886). Different regression models differ based on the kind of relationship between the dependent and independent variables, considering and the number of independent variables.

Linear regression executes the task to predict a dependent variable (y) based on a given independent variable (x). So, the regression technique finds out a linear relationship between x (input) and y (output), hence, termed Linear Regression.

$y = \theta_1 + \theta_2 . x$ (George & Alan, 2003) 2.2

During model training, the parameters will be defined as:

x = input training data (uni-variate – one input variable (parameter))

y = labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

θ_1 = intercept

θ_2 = coefficient of x

Once the best θ_1 and θ_2 values are achieved, the best fit line will be obtained. While using the model for final prediction, the value of y for the input value of x will be predicted.

2.1.3. PROPHET

The Prophet architecture is a time series prediction library developed by Facebook to simplify how time-series data are modeled and aid in providing better performing models (Sean & Letham, 2017). Unlike the three tunable components in ARIMA, Prophet offers more intuitive parameters that are easy to tune.

Prophet uses a time series model that is decomposable, and it has three main model components: trend, seasonality, and holidays, as shown in equation 2.3.

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \quad \text{(Sean \& Letham, 2017)} \qquad 2.3$$

Where g(t) is the piecewise linear or logistic growth curve for modeling non-periodic changes in time series, s(t) represents seasonality changes, h(t) shows the consequence of holidays and ε_t is the error term that accounts for any unusual change that the model does not accommodate.

Prophet uses time as a regressor and tries to fit several linear and non-linear functions of time as components.

A piece-wise linear curve can be fitted over the trend or the non-periodic part of the time series to model the trend. This technique ensures that missing data and spikes do not have an abrupt effect on the model.

Prophet relies on Fourier series to fit and forecast the effects of seasonality and to deliver a flexible model. Seasonal effects are estimated by equation 2.4

$$s(t) = \sum_{n-1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)\right) \quad \text{(Sean \& Letham, 2017)} \qquad 2.4$$

P denotes the period in days

Prophet creates an allowance for a custom list of past events and those that might occur in the future. Windows are created around such days and are considered separately for additional parameters to be fitted into the model to cater to the effect of holidays and events.

2.1.4. LONG SHORT-TERM MEMORY (LSTM)

LSTM as proposed by (Sepp & Jurgen, 1997) is a type of neural network architecture that has feedback connections. LSTM networks have been made known to learn long-term dependencies more quickly than the simple recurrent architectures. LSTM blocks contain one or more self-connected memory cell(s) and three multiplicative units identified as the input, output, and forget gates. The cell remembers values over arbitrary time intervals, and the three gates regulate the flow of information into and out of the cell.

These multiplicative units provide continuous analogs of write, read, and reset operations for the cells. They also allow LSTM memory cells to store and access information over long periods, thereby extenuating the vanishing gradient problem (Graves, 2012).

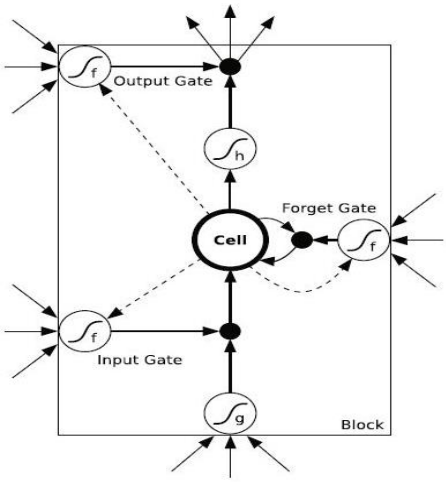

Figure 1: An LSTM Cell **(Graves, 2012)**

The LSTM cell internal state is updated as follows:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \quad 2.5$$

$$where\ f_i^{(t)} = forget\ gate\ (for\ time\ step\ 't' and\ cell\ 'i')$$

$$x^{(t)}\ and\ h^{(t)}\ are\ current\ input\ and\ hidden\ layer\ vector\ respectively$$

$$b_i^f, U_{i,j}^f, W_{i,j}^f\ are\ biases, input\ weight\ and\ recurrent\ weight\ for\ the\ forget\ gate$$

(Ian et al., 2016)

2.1.5. MANUAL BACKTESTING ALGORITHM

Backtesting is a vital step when building trading strategies. It serves as a baseline model in measuring the viability of developed models. The backtesting of trading strategies is vital for stockbrokers to judge if a particular strategy is profitable under specific circumstances. It enables the users to learn how a trading strategy is likely to perform in the marketplace. It also provides the users with the opportunity to improve a trading strategy (Albert et al., 2008).

A well-conducted backtest that produces positive results guarantees traders that the strategy is primarily sound and will likely return profits when implemented in reality. A well-conducted backtest that produces poor outcomes will alert traders to modify or reject the strategy.

The particular backtesting technique used in this project uses a Simple Moving Average (SMA) calculation to operate a buy and hold strategy and it is to be implemented any time the specified conditions of the strategy are met.

2.2. AI-BASED STOCK PREDICTION TECHNIQUES

In one of the recent standout papers on stock prediction using LSTM, (Xiongwen et al., 2018) used multi-stock high-dimensional historical data instead of a single stock index with an LSTM network possessing embedded layer, and another LSTM network with automatic encoder for data vectorization and predict the stock market. Their experiment indicates that the LSTM network with embedded layers performs slightly better with an accuracy of 57.2%, while the LSTM with automatic encoder's level of accuracy was measured at 56.9%. These accuracy levels are not good enough for real-life applications as they are only accurate approximately half of the time. It can lead to severe capital loss.

In a breakthrough classical paper in stock market prediction presented by (Takashi et al., 1990), several modular neural network models were developed for the Tokyo Stock Exchange Prices Indexes (TOPIX). These models can be used to detect the timing for when to buy and sell stocks. They learned the connections between numerous economic and technical indexes. Figure 2 illustrates the block diagram of the prediction system.

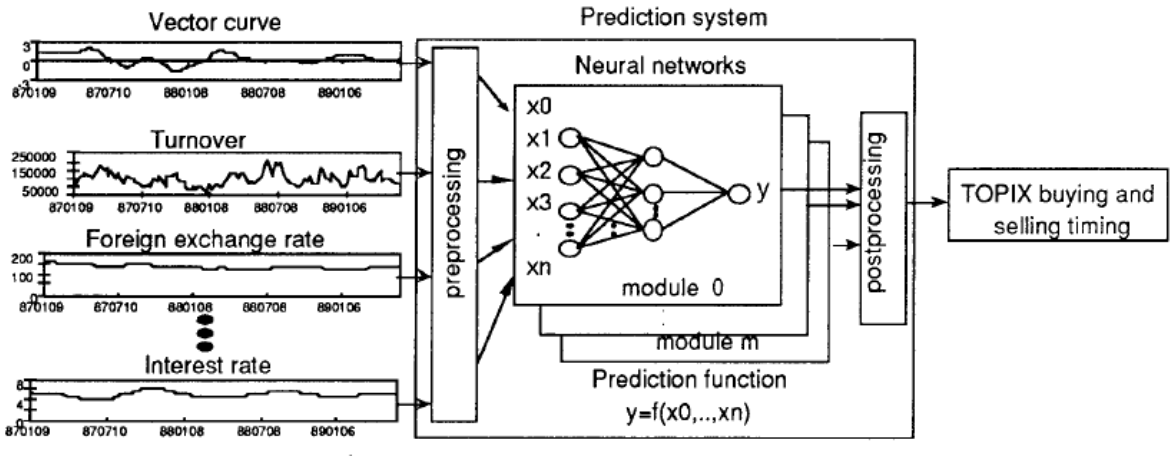

Figure 2: Block diagram of the TOPIX predictive algorithm **(Takashi et al., 1990)**

The input indexes are not limited to just stock prices. Other factors considered include vector curve, turnover, interest rate, foreign exchange rate, and New York Dow-Jones average.

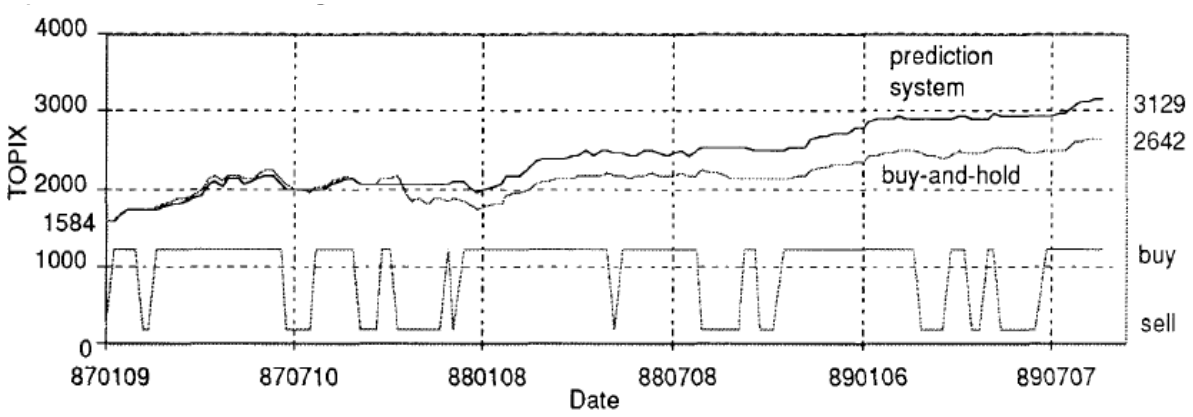

Figure 3: Performance of the TOPIX predictive algorithm compared with Buy and Hold

(Takashi et al., 1990)

Figure 3 shows how well the prediction system performs compared with a standard buy and hold policy.

Using ARIMA to predict the value of stocks at any level requires concrete evidence to show that the type of data being used possesses an autoregressive component. (Jeffrey & Kyper, 2011) Inferred that daily stock price index comprises of autoregressive components; therefore, one can predict returns on stock. The research also used ARIMA-Intervention to analyze their data that span over a long period where stock prices changed during fluctuating temporal economic activities. The developed model from this research is given below:

$$0.500009Y_t = 9.937475 + 0.492162\,Y_{t-1} + (2.3e^{-9})V_t - 2.324061\,I_t \text{ (Takashi et al., 1990) 2.6}$$

Where "I_t" is an intervention or dummy variable and V_t is the Volume

2.3. DATASET

World trading is a platform that provides historical and near real-time stock and forex prices. They provide API endpoints that provide relevant stock and forex market data. The data is imported from their database in JSON (JavaScript Object Notation) format, and a python script was written to convert it into a Pandas (a data science framework that can be used for handling and processing data) data frame.

This project uses the Apple Incorporation stock index (AAPL). Apple Inc. is one of the fastest-growing companies in the past decade, and the choice for selecting it for this research is a result of the rising and falling trends it has experienced over the years, capturing a regular stock market performance.

Figure 4 shows a sample of the trading data provided by WorldTrading. It is a 6- column data, and its rows represent an entire calendar day.

	open	close	high	low	volume
2019-10-30	244.76	243.26	245.30	241.21	31130522
2019-10-29	248.97	243.29	249.75	242.57	35709867
2019-10-28	247.42	249.05	249.25	246.72	24143241
2019-10-25	243.16	246.58	246.73	242.88	18369296
2019-10-24	244.51	243.58	244.80	241.81	17916255

Figure 4: Sample of Data Obtained From World Trading

The values in each column have its significance, as explained below:

- Open: This daily opening price of a stock is the price of the first trade of that particular stock on a given day. The opening price is a significant indicator of that day's trading activity, especially for those interested in evaluating short-term effects.
- Close: Closing price signifies the price at which a stock is traded when the exchange closes for the day. Comparing this with the opening price shows how stock rises or falls in a day.
- High: This column holds the value for the day's highest price of a particular stock.
- Low: Signifies the value for the day's lowest price.

The graph in figure 5 shows the closing value of AAPL plotted against each calendar day. The time frame in this graph is from January 1, 2015, to October 31, 2019. This plot gives a general overview of how stock prices behave.

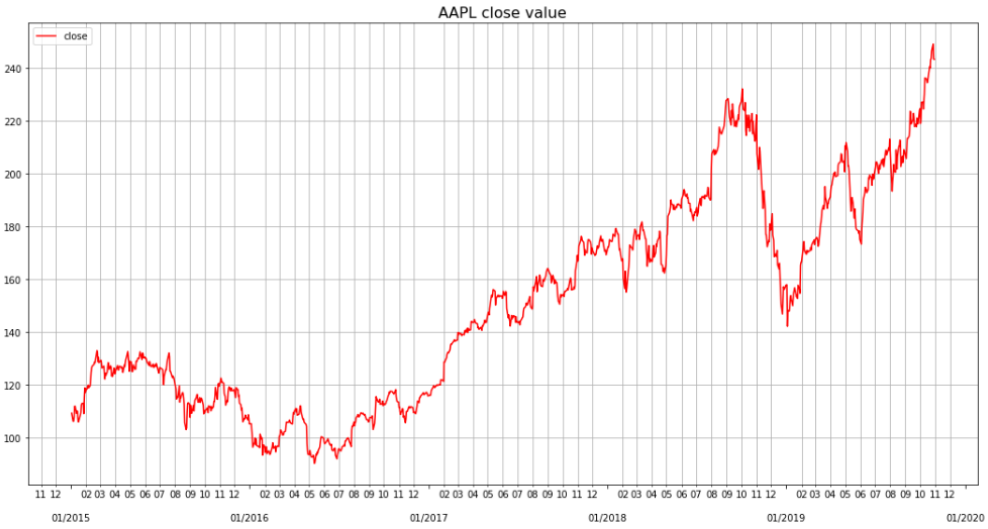

Figure 5: AAPL close value from 2015 to 2019

The dataset has 1758 rows as it covers each trading day from January 1st, 2015, to November 1st, 2019.

CHAPTER 3

3. METHODOLOGY

Several steps are necessary to develop a machine learning or neural network model. Such steps include data acquisition and cleaning, data visualization, developing the model, fitting the model, and deducting inference from it. To actualize this stock market prediction project, all these techniques are required and carried out using the python programming language and the Jupyter Notebook IDE. These tools, libraries and steps will be described in the sections that follow.

3.1. TOOLS & LIBRARIES

3.1.1. ANACONDA

Anaconda is a free and open-source distribution that offers easy ways to perform Python and R tasks like data science, neural networks, and machine. It is the industry standard for developing, testing, and training models as well as analyzing and visualizing data.

3.1.2. JUPYTER NOTEBOOK

The Jupyter Notebook is an open-source web application that allows you to create and share records that contain live software, words, thoughts, and information from your browser. Uses include: cleaning and change of data, numerical computations, visualization, specific AI tasks and much more.

3.1.3. PYTHON

Python was the programming language of choice for this task. This was a simple choice for numerous reasons.

- As a script, Python has behind it a massive network. With an excursion to Stack Overflow, any problems that may be experienced can be overcome effectively. Python is one of the most popular dialects on the web, making it possible that every question would be answered directly.
- Python has a range of impressive tools prepared for practical figuration. For starters, packages such as NumPy, Pandas, and SciPy are openly available. For example, existing libraries can significantly decrease the amount of code that a project requires. (Jason, 2016)

3.1.4. NUMPY

NumPy is a python module that gives logical and more elevated level numerical computation. In most of the programming languages, we can't utilize numerical reflections, for example, f(x) as it would influence the semantics and the sentence structure of the code. In any case, by utilizing NumPy we can adventure such capacities in our code.

NumPy's cluster type increases the Python language with a productive information structure utilized for numerical tasks. NumPy additionally gives essential numerical schedules and contains a powerful N-dimensional array. For example, NumPy provides a method for discovering Eigenvectors.

3.1.5. SCIKIT-LEARN

Scikit-learn is a Python programming language free AI library. This carries various operations such as support vector machine, random forests, angle boosting, k-means, and so on. It is intended to interoperate primarily with the NumPy and SciPy numerical and logical libraries of Python.

Scikit-learn is written in Python to a large extent, with some center calculations written in Python for performance. Support vector machines are performed by a Python wrapper around LIBSVM, i.e., by a comparable wrapper around LIBLINEAR, calculated relapse and direct aid vector machines.

3.1.6. TENSORFLOW

TensorFlow is an open-source programming library that uses knowledge flow graphs for numerical calculation. The adaptable design allows you to send calculations to at least one CPU or GPU with a single API in a work area, server or cell phone. TensorFlow was initially created by researchers and developers taking a shot at the Google Brain Team during Google's Machine Intelligence inquiry about collaboration for purposes of leading AI and developing deep neural systems, but the architecture is broad enough to be applicable in a wide range of different areas as well.

TensorFlow is the second-age framework for Google Brain. While reference usage runs on single gadgets, TensorFlow can run on numerous CPUs and GPUs (with discretionary expansions of CUDA and SYCL). TensorFlow is available on 64 Linux, MacOS, Windows, and mobile stages like Android and iOS.

3.1.7. PANDAS

In PC programming, Pandas is a software library composed of the Python programming language for information control and analysis. It is free software with three conditions provided under the BSD permit. The Pandas library has the following features:

- Data Frame objects for data manipulation with integrated indexing.
- Tools for reading and writing data in several file formats.
- Data alignment and ability to handle missing or corrupted data.
- Reshaping data sets which involves data structure column insertion and deletion .
- Slicing, fancy indexing, sub-setting, and data filtration of large data sets.
- Data set merging and joining.
- Time series-functionalities

3.1.8. MATPLOTLIB

Matplotlib is a library for the data visualization using the Python programming language and its numerical calculation increase in NumPy. It provides an item-arranged API for, for instance, Tkinter, wxPython, Qt and GTK+ to use a versatile GUI toolbox to integrate plots into applications. There is also a procedural "pylab" interface that is based on a state machine (like OpenGL), worked closely to MATLAB, although it is limited in its use. SciPy is using Matplotlib.

3.1.9. SEABORN

Seaborn is a numerical analysis library built in Python. It is based on matplotlib and is coupled with panda’s data structures. Seaborn offers the following functionalities:

- A dataset-oriented API for exploratory relationships between several variables
- Specialized maintenance for using categorical variables to display observations.
- Options for visualizing uni-variate and bivariate distributions and for comparison among data subsets.
- Automatic approximation and plotting of linear regression models for various types of dependent variables
- Ability to conveniently view the overall structure of complex datasets.
- High-level abstractions for organizing multi-plot grids that lets one easily build complex conceptions.
- Tools for choosing color palettes that dependably disclose patterns in data

3.2. DATA HANDLING

The data required for this project was acquired using an API from WorldTrading. WorldTrading provides a platform where researchers and data analysts can access a plethora of historical, intraday, real-time stock and forex indexes.

The API uses Hypertext Transfer Protocol (HTTP) to receive, send, delete, and post data. It offers interoperability between the internet and computer systems. To access this feature from the IDE, a token that is given upon signup with WorldTrading is required. The API takes 5 query parameters; API-token, sort by, symbol, sort order, and output. The API returns a JSON data format as the output.

This acquired JSON data needs to be converted to a Pandas (an easy to use data analysis tool for python) Data frame to allow easy data manipulation. Pandas also provide a method to deal with missing values. All acquired data are resampled and interpolated to cater for

missing data. A plot for the closing price of AAPL stock was plotted as shown in Figure 5 to give a glimpse of what the data looks like.

```
#specify company name and token to access WorldTrading API
company_name = 'AAPL'
token = "YFl1bfK9zAddFzcsOgenstMkFg3Vl84lb6rtf80vAbQ2yj1ldqM4lv75cE5q"
```

```
#specify the co,pany symbol and the function to be imported
funct = "history"
sym = "AAPL"
```

```
#specify the API url
url = (f"https://api.worldtradingdata.com/api/v1/history?symbol={company_name}&date_from=2015-01-01&sort=newest&api_token={token}
```

```
######### Extract data from the url given ##############

#convert response to python json list
data = requests.get(url).json()[funct]
```

```
# Make dataframe from the json file

df = pd.DataFrame(data)
df = df.T
```

```
#convert list to dataframe with columns names and order

df = pd.DataFrame(columns=['Date','Open','High','Low','Close'])
for k,v in data.items():
    date = datetime.strptime(k, '%Y-%m-%d')
    data_row = [date.date(),float(v['open']),float(v['high']),
                float(v['low']),float(v['close'])]
    df.loc[-1,:] = data_row
    df.index = df.index + 1
```

```
# Convert Date column to datetime
df.loc[:, 'Date'] = pd.to_datetime(df['Date'],format='%Y-%m-%d')
```

```
# copy the dataframe to another dataframe

dfCopy = df.copy()
```

Figure 6: Import Data from WorldTrading and Convert JSON to Data Frame

Cell 1 and 2 in figure 6 assigns the company name, token, function to be loaded from the API and the company symbol to four different variables instead of assigning them directly in the URL. This makes changing these variables easier in case of testing other companies or changing the token. Cell 3 extracts data from the given URL in JSON format while cell 4, 5 and 6 converts, sorts, and index the JSON format into a workable pandas Data frame. Cell 7 creates a variable that holds the data frame after cleaning the imported data.

Handling missing value

```
### filling with missing value with spline interpolation


#df_f = df_f.resample('D')

upsampled = dfCopy['Close'].resample('D').interpolate().astype(float)
interpolated_1 = upsampled.interpolate(method='spline', order=2).astype(float)
```

```
df_final = pd.DataFrame(interpolated_1)
df_final.head(5)
```

```
########## split the dataset with train set and test set

df_t = df_final.head(1500)
df_test = df_final.tail(258)
```

Figure 7: Handling Missing Data in-code

Figure 7 shows how the program handles missing data. Cell 1 does a spline interpolation to fill in values for missing trading days while cell 2 creates a variable that holds the final version of the cleaned data. Cell 3 splits the dataset into train and test sets.

3.3. IMPLEMENTATION AND INFERENCING

To achieve the specified objectives of this project, three AI-based models were developed and compared with a manual backtesting algorithm. The subsections that follow give a summary that describes each developed model.

3.3.1. AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA)

Developing ARIMA models in python has been simplified by statsmodel– a Python module that offers an estimation of statistical models using functions and classes, as well as conducting statistical data exploration and tests.

ARIMA models are straightforward to develop because they only need to be enclosed in a “for loop” and fed with values for the p, d, and q parameters.

```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

```
## making arima model with history data

history = df_t['Close'].values.tolist()
test = df_test['Close'].values.tolist()
predictions = list()


for t in tqdm(range((df_test.shape[0]))):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(yhat)
    #print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
```

Figure 8: ARIMA Model Development

Figure 8 is presents how ARIMA model was developed. Cell 1 contains the import statements to introduce the Libraries needed to develop this model. Cell 2 shows how the model was developed. The process is enclosed in a for-loop because it uses the immediate past value in making new predictions; hence it needs to be updated after every prediction step.

Figure 9 shows how the plot to visualize the ARIMA predictions was obtained while figure 10 is the resulting prediction.

```
#plt.plot(list(range(258)),test,label='original')
#plt.plot(list(range(258)),predictions,label='prediction')

plt.figure(figsize=(10,8))
plt.plot_date(df_test.index,test,'r-',label='original',color='blue')
plt.plot_date(df_test.index,predictions,'r-',label='Prediction',color='red')
plt.title('Predicted Stock Value with Arima model')
plt.xlabel('Date')
plt.ylabel('Close value')
ax.xaxis.set_minor_locator(dates.MonthLocator())
ax.xaxis.set_minor_formatter(dates.DateFormatter('%m'))
plt.legend()
plt.show()
```

Figure 9: Plotting ARIMA Predictions for the Daily Close Values

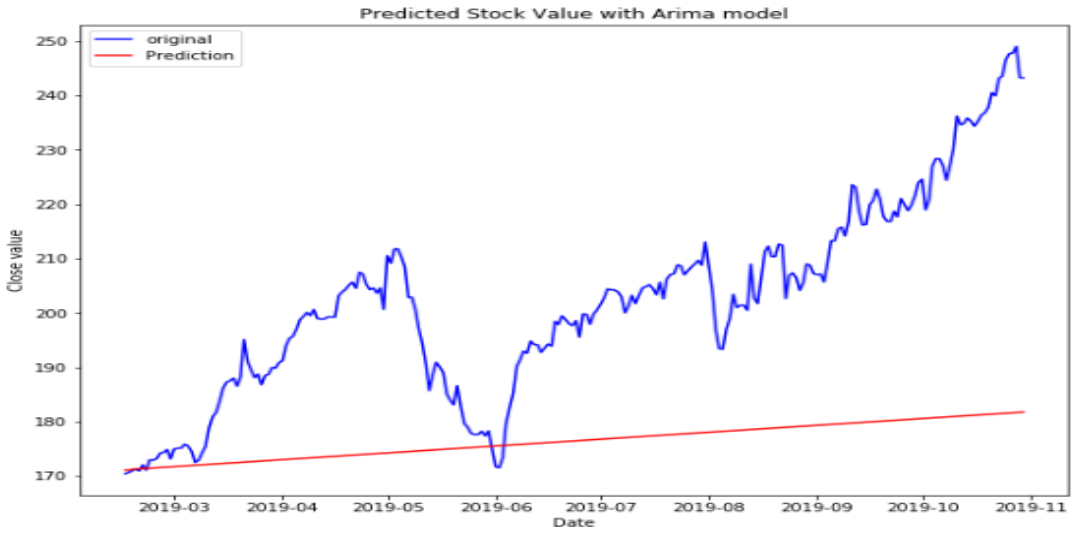

Figure 10: Plot of the ARIMA model performance

3.3.2. LONG SHORT-TERM MEMORY (LSTM)

LSTM is a type of recurrent artificial neural network that solves the problem of a vanishing gradient. It learns long term dependencies so that weights of values that occurred in previous time steps are still seen as necessary when developing the model.

There are different approaches to developing LSTM models. Such methods include; Vanilla LSTM, Bidirectional LSTM, ConvLSTM, CNN LSTM, and Stacked LSTM. This project uses Vanilla LSTM, which comprises of a single hidden LSTM layer, and an output layer used in making predictions.

The LSTM model developed in this project is a sequential model developed with Keras. Keras is a high-end python library that provides an interface for neural network model development.

The shape of the input is a significant factor needed in defining an LSTM model; that is what input the model should expect from each sample in relation to the number of features and the time steps. The Adam version of stochastic gradient descent is used in fitting the model, and the loss function selected is the mean squared error or ‘mse‘ (Mean Square Error) for optimization. After the model has been defined, it is fitted to make predictions over 10 epochs.

Figure 11 shows how the LSTM model was trained and fitted using the Sequential() method from the Keras Library. The LSTM model has a hidden layer and a dense layer and uses Mean Absolute Error (MAE) as its loss function while using ADAM as its optimizer. The model was fitted over 10 epochs.

```
# design LSTM network
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(70, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(tf.keras.layers.Dense(1))
model.compile(loss='mae', optimizer='adam')
```

```
# fit network with 10 epochs

history = model.fit(train_X, train_y_1, epochs=10, validation_data=(test_X, test_y_1), verbose=2,shuffle=False)
## make a prediction
yhat = model.predict(test_X)
print(len(yhat))
```

Figure 11: Training and Fitting the LSTM Model

```
def make_forecasts(model, n_batch, test, n_lag, n_seq):
    forecasts = list()
    for i in range(len(test)):
        X, y = test[i, 0:((3*n_lag))], test[i, ((3*n_lag)):]
        # make forecast
        forecast = forecast_lstm(model, X, n_batch)
        # store the forecast
        forecasts.append(forecast)
    return forecasts
```

Figure 12: Making Forecasts with LSTM

To make predictions using the LSTM model over a certain time steps, the method needs to be enclosed in a for-loop as shown in figure 12.

Figure 13 shows the dataset set aside for testing the model developed, while figure 14 represents a plot of predicted values using our LSTM model.

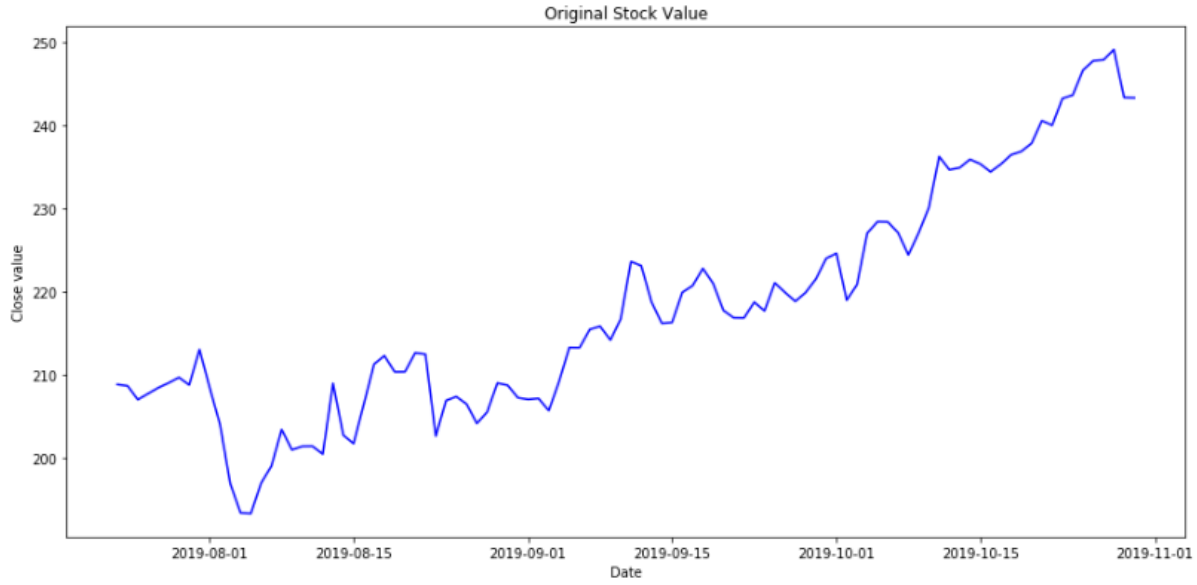

Figure 13: Original Values for the Test Set

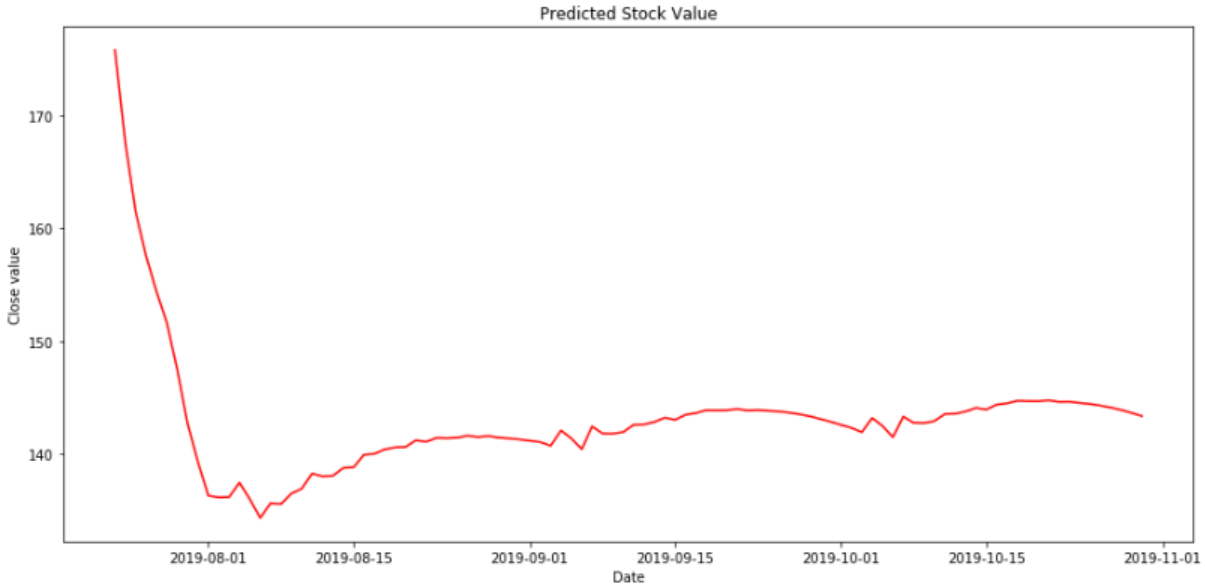

Figure 14: Predicted Values for the Test Set

3.3.3. PROPHET

Prophet is a Facebook library for making time-series predictions. It is capable of forecasting data frames in the format of date stamp and the column with the values to be predicted. The training set has the shape (1500, 2), which specifies a 2-column data with 1500 instances.

```
##### Importing library

from fbprophet import Prophet
```

```
df_pr.shape
```

```
# Making with daily seasonaly True

model = Prophet(daily_seasonality=True)
model.fit(df_pr)
```

```
df_test.shape
```

```
df_test.head()
```

```
####### Making prediction for next 100 days

num_days = df_test.shape[0]

future = model.make_future_dataframe(periods=num_days)
forecast = model.predict(future)

print (forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())
```

Figure 15: Predicting with Prophet

The process of developing a predictive model with Prophet starts by importing the library as shown in the first cell of figure 15, compiling the model and then making predictions as presented in cell 3 and 6 respectively.

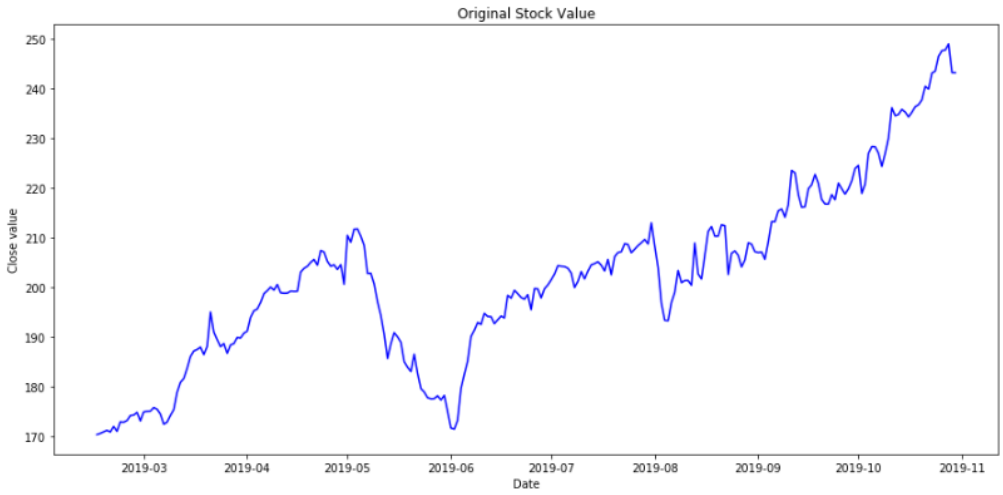

Figure 16: Original Values of the Test Set

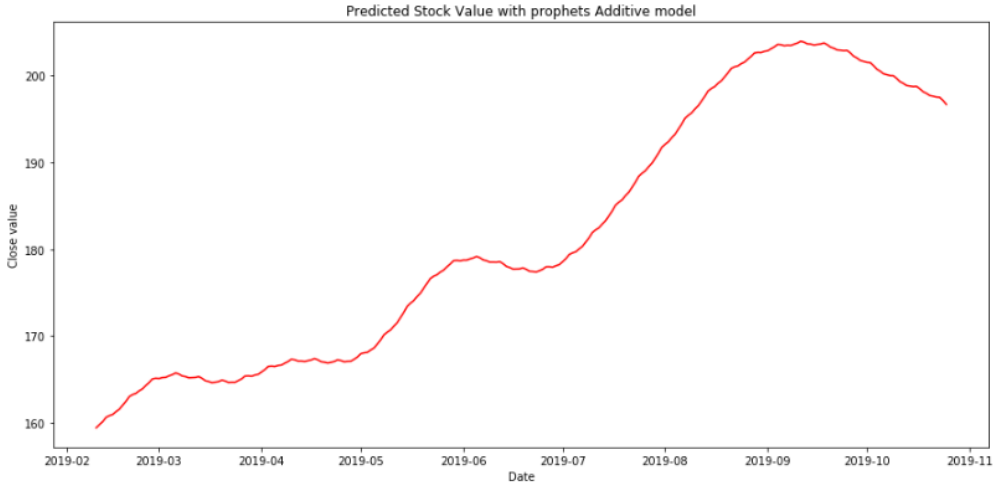

Figure 17: Predicted Values Using Prophet

3.3.4. LINEAR REGRESSION

The linear regression model developed will try to draw a regression line across the stock closing values and make predictions based on the inference obtained. As shown in figure 18, the method employed allows the user specify the number of points in the regression line, plot the regression line and make predictions from it.

```
# Specify the day you are interested in
day = pd.Timestamp(date(2018, 8, 20))

# Specify the maximum N you want to plot (If Nmax2 is too large it gets very cluttered)
Nmax2 = 5

df_temp = cv[cv['Date'] <= day]
plt.figure(figsize=(12, 8), dpi=80)
plt.plot(range(1,Nmax2+2), df_temp[-Nmax2-1:]['Close'], 'bx-')
plt.plot(Nmax2+1, df_temp[-1:]['Close'], 'ys-')
legend_list = ['Close', 'actual_value']

# Plot the linear regression lines and the predictions
color_list = ['r', 'g', 'k', 'y', 'm', 'c', '0.75']
marker_list = ['x', 'x', 'x', 'x', 'x', 'x', 'x']
regr = LinearRegression(fit_intercept=True) # Create linear regression object
for N in range(5, Nmax2+1):
    # Plot the linear regression lines
    X_train = np.array(range(len(df_temp['Close'][-N-1:-1]))) # e.g. [0 1 2 3 4]
    y_train = np.array(df_temp['Close'][-N-1:-1]) # e.g. [2944 3088 3226 3335 3436]
    X_train = X_train.reshape(-1, 1)
    y_train = y_train.reshape(-1, 1)
    regr.fit(X_train, y_train)            # Train the model
    y_est = regr.predict(X_train)         # Get linear regression line
    plt.plot(range(Nmax2+1-N,Nmax2+2),
             np.concatenate((y_est, np.array(df_temp['est_N'+str(N)][-1:]).reshape(-1,1))),
             color=color_list[N%len(color_list)],
             marker=marker_list[N%len(marker_list)])
    legend_list.append('est_N'+str(N)+'_lr')

    # Plot the predictions
    plt.plot(Nmax2+1,
             df_temp['est_N'+str(N)][-1:],
             color=color_list[N%len(color_list)],
             marker='o')
    legend_list.append('est_N'+str(N))


plt.grid()
plt.xlabel('timestep')
plt.ylabel('Price')
plt.legend(legend_list, bbox_to_anchor=(1.05, 1))
matplotlib.rcParams.update({'font.size': fontsize})
```

Figure 18: Predicting with Linear Regression

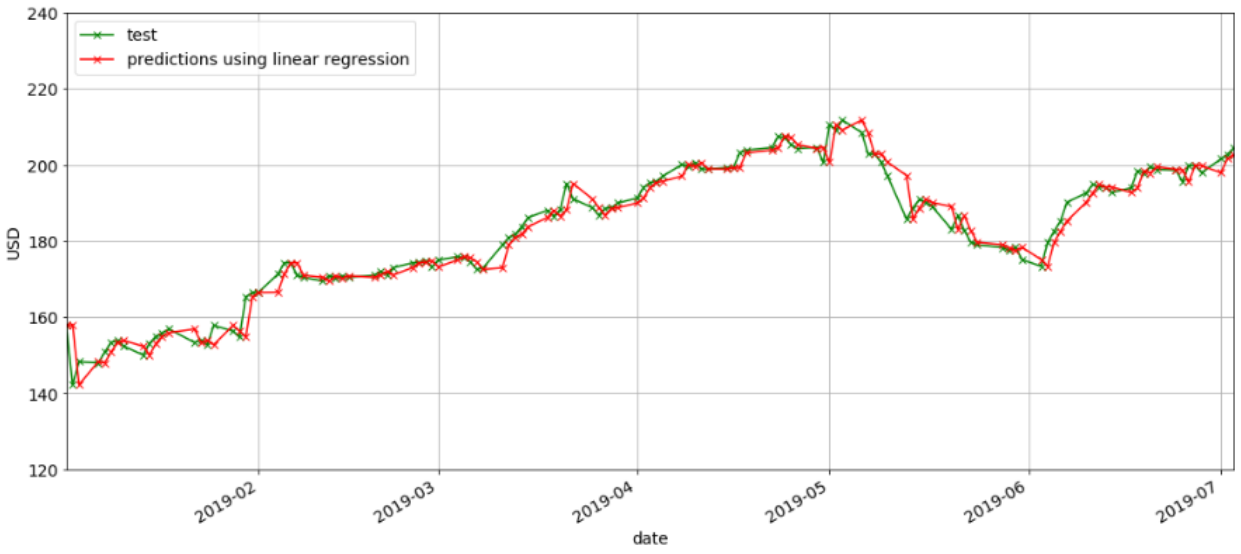

Figure 19: Linear Regression Performance

This method is somewhat visually accurate, but it guarantees losses when practically implemented. This type of predictive technique is not suitable for stock price or stock trend prediction.

3.3.5. MANUAL BACKTESTING ALGORITHM

One of the objectives of this research is to show how well machine learning and neural network algorithms perform when compared with a manual backtesting approach. The backtesting technique used in this project is the Simple Moving Average (SMA) approach.

The Simple Moving Average (SMA) of a stock is the arithmetic mean of that stock. It is calculated by summing the latest prices, then dividing that by the number of time frames in the calculation average. Short-term averages react rapidly to variation in the price of the stock, while long-term averages are sluggish to react. SMA is an investigative tool used to identify stock market trends and the potential for a change in a recognized trend. The SMA serves as a technical indicator for determining if a stock price will reverse or continue a bear or bull trend. It has been established in literature by (Albert et al., 2008) that buy and hold is not a bad performing strategy, especially in big firms.

The formula for SMA is given as:

$$SMA = \frac{A_1 + A_2 + \cdots + A_n}{n} \quad 3.1$$

```
from backtesting import Strategy
from backtesting.lib import crossover


class SmaCross(Strategy):

    # Define the two MA lags as *class variables*
    # for later optimization
    n1 = 15
    n2 = 60

    def init(self):
        # Precompute two moving averages
        self.sma1 = self.I(SMA, self.data.Close, self.n1)
        self.sma2 = self.I(SMA, self.data.Close, self.n2)

    def next(self):
        # If sma1 crosses above sma2, buy the asset
        if crossover(self.sma1, self.sma2):
            self.buy()

        # Else, if sma1 crosses below sma2, sell it
        elif crossover(self.sma2, self.sma1):
            self.position.close()

from backtesting import Backtest

bt = Backtest(df, SmaCross, cash=10000, commission=.002)
bt.run()
```

Figure 20: Implementation of the Backtesting Algorithm

The manual SMA strategy is developed by importing Strategy and crossover modules from the python backtesting library. We defined two moving average lags 15 and 60 and then pre-compute these lags. The primary strategy is to buy assets when the first SMA crosses over the second SMA and sell asset when the first SMA goes below the second.

3.4. MODEL EVALUATION AND INFERENCING

To evaluate our developed models, we use a classical statistical technique, and the root mean square error (RMSE), which is given by equation 3.2.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(A_t - F_t)^2} \quad \text{(Cort \& Kenji, 2005)} \quad 3.2$$

Where A_t is the actual value, F_t is the forecasted value, n is the number of observations, and t denotes the specific time step under consideration.

The root-mean-square error (RMSE) is a commonly used tool to measure the differences between values predicted by a model and real acquired values. RMSE is a measure of accuracy, and it is used to compare forecasting errors of several developed models for a specific dataset and not between datasets because it is scale-dependent. RMSE is always a positive number, and a value of 0 would indicate a perfect fit to the data. In general, the lower the RMSE, the better the performance of the model. RMSE is the square root of the average of squared errors. The RMSE is more punishing of errors.

Of all the models developed, Prophet performed best. It does not predict the exact stock price but gives a "beyond average performance" prediction of trends in the AAPL stock. Table 1 shows the performance of all the developed models using the RMSE error measurement technique on the test set.

S/N	Model	RMSE on Test Set
1	ARIMA	886.35543451713
2	LSTM	5818.542114685997
3	Prophet	552.9824255758182

Table 1: RMSE Values of all Developed Models

CHAPTER 4

4. CONCLUSION AND RECOMMENDATION

The findings and discussions of this project “Stock Market Analysis and Prediction System” will be presented in this chapter with closing remarks as well as some recommendations for future work that can be implemented to make this project better.

4.1.1. CRITICAL ANALYSIS

The performance of the developed neural network and machine learning models are not at the level of implementation for real-life trading. During the course of developing the models in this project, we used linear regression, which just forecasts values in previous time steps as its prediction. This method is visually accurate, but it assures losses when practically implemented. This type of predictive technique is not suitable for neither stock price nor stock trend prediction.

Though our LSTM, ARIMA, and Prophet predictions perform better than the linear regression model, their performance does not meet up with the classical SMA strategy.

The SMA algorithm was provided with 10000 units (of money) for trading with a commission of 0.2%. Figure 21 shows a breakdown of how the model performed.

Start	2015-01-02 00:00:00
End	2019-11-01 00:00:00
Duration	1764 days 00:00:00
Exposure [%]	50.2268
Equity Final [$]	14615.1
Equity Peak [$]	14615.1
Return [%]	46.1506
Buy & Hold Return [%]	132.101
Max. Drawdown [%]	-24.0338
Avg. Drawdown [%]	-4.50758
Max. Drawdown Duration	566 days 00:00:00
Avg. Drawdown Duration	63 days 00:00:00
# Trades	10
Win Rate [%]	50
Best Trade [%]	25.8061
Worst Trade [%]	-9.79097
Avg. Trade [%]	2.18697
Max. Trade Duration	187 days 00:00:00
Avg. Trade Duration	89 days 00:00:00
Expectancy [%]	8.89181
SQN	0.51867
Sharpe Ratio	0.190265
Sortino Ratio	0.854927
Calmar Ratio	0.0909959
_strategy	SmaCross

Figure 21: Breakdown of SMA performance

The SMA started on January 2, 2015, and ended on November 1st, 2019, with a total of 1764 trading days. Using the buy and hold technique it boasts a return of 132.101% having traded (buy + sell) 10 times.

4.1.2. APPRAISAL

This Double SMA strategy provides a performance better than what automatic algorithmic approaches can offer. Therefore, we conclude that given this particular stock

data we used (opening, closing, highest, and lowest daily prices), the manual approach works better.

4.1.3. FURTHER WORK

Improving current models or finding out other models that might result better could also be done if I had the time to do so. However, taking into account other variables that can show a company's performance such as debt status, information search company policy, accumulated profit, profile management, volume of word of mouth, emotional valence, opinion polarization and other economic and technical details (David & Frank, 2015) might produce better performing AI-based models. Outside factors such as interest rate, inflation rate, and exchange rate might also have an effect in the stock market. Making available more relevant data for all these methods will make them more robust and possess more information at their disposal for accurate prediction.

REFERENCES

Alan, T.M., 1948. *Intelligent Machinery*. NPL. Mathematics Division.

Albert, S., Zuoquan, X. & Xun, Y.Z., 2008. Thou shalt buy and hold. *Quantitative Finance*, 8(8), pp.765-76.

Box, G.E.P. & Jenkins, G.M., 1970. *Time Series Analysis*. 1st ed. Holden-Day.

Chang, S.V. et al., 2013. A review of stock market prediction with Artificial neural network (ANN). In *IEEE International Conference on Control System, Computing and Engineering*. Penang, 2013. IEEE.

Cort, W.J. & Kenji, M., 2005. Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in Assessing Average Model Performance. *Climate Research*, 30, pp.79-82.

David, G. & Frank, S., 2015. Social Signals and Algorithmic Trading of Bitcoin. *Royal Society Open Science*, 2(9), pp.1-13.

Francis, G., 1886. Regression Towards Mediocrity in Hereditary Stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15, pp.246-63.

George, S.A.F. & Alan, L.J., 2003. *Linear Regression Analysis*. 2nd ed. New Jersey: Wiley & Sons Inc.

Gilbert, E. & Karahalios, K., 2010. Widespread Worry and the Stock Market. In *International AAAI Conference on Weblogs and Social Media*., 2010.

Graves, A., 2012. *Supervised Sequence Labelling With Recurrent Neural Networks*. 1st ed. Berlin: Springer.

Ian, G., Yoshua, B. & Aaron, C., 2016. *Deep Learning*. MIT Press.

Jason, B., 2016. *Machine Learning Mastery with Python*. 14th ed. Melbourne.

Jeffrey, J.E. & Kyper, E., 2011. ARIMA Modeling With Intervention to Forecast and Analyze Chinese Stock Prices. *International Journal of Engineering Business Management*, 3(3), pp.53-58.

Mohit, S. & Sachchidanand, S., 2015. In pursuit of the best Artificial Neural Network for Predicting the Most Complex Data. In *International Conference on Communication, Information & Computing Technology*. Mumbai, 2015. IEEE.

Robert, S.P. & Hsinchun, C., 2009. Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFinText System. *ACM Transactions on Information Systems*, 27(2), pp.12:1-12:19.

Sean, T.J. & Letham, B., 2017. Forecasting at Scale. *The American Statistician*, 72(1), pp.37-45.

Sepp, H. & Jurgen, S., 1997. Long Short-Term Memory. *Neural Computation*, 9, p.1735–1780.

Takashi, K., Kazuo, A., Morio, Y. & Masakazu, T., 1990. Stock Market Prediction System with Modular Neural Networks. In *International Joint Conference on Neural Networks*. San Diego, 1990. IEEE.

Walter, E., 1988. Arima and Cointegration Tests of PPP under Fixed and Flexible Exchange Rate Regimes. *The Review of Economics and Statistics*, 70(3), pp.504-08.

Xiongwen, P. et al., 2018. An Innovative Neural Network Approach for Stock Market Prediction. *The Journal of Supercomputing*, pp.1-21.