A Systematic Approach To Financial Modelling Using Machine Learning

By: Antonio Jesus Cefalo Yaghmour

Student ID: UP828398

Supervisor: Dr. Alexander Gegov

School of Computing

Final-year Project

Project Type: PJE40

May 2021

Abstract

Machine learning modelling is a way to predict and test data, commonly used by traders and investors, using artificial intelligence. There has been a significant increase in the number of users within the stock market industry over the years, leading to more people finding and creating different methods and techniques to evaluate the financial market accordingly. In this context, machine learning algorithms can be used to make stock predictions and compare them to each other.

The research in this report has shown that it is worth integrating machine learning algorithms and determining the stock prediction horizon for the software implementation. Particularly, the aim of this report is a short term prediction of the price of a certain stock for the coming end-of-day considering a certain number of days in the past. To achieve this aim, several machine learning algorithms have been chosen for training and testing based on a particular dataset.

This report uses accuracy and efficiency as performance evaluation indicators for the following machine learning learning algorithms: Linear Regression (LR), Polynomial Regression (PR), Random Forest (RF) and Long Short-Term Memory Neural Network (LSTM). For this purpose, these algorithms are simulated for making predictions about stock prices from the historical daily prices of Nasdaq-traded stocks and ETFs Exchange (NASDAQ) using the Amazon (AMZN) dataset. The accuracy of the algorithms is quantified and compared in terms of training and testing errors using four different simulation runs by means of the Root Mean Square Error (RMSE). The efficiency of the algorithms is quantified and compared in terms of computational time for both training and testing using the same four simulation runs by means of duration in minutes, seconds and milliseconds. The simulation runs are in terms of an established benchmark approach and three novel approaches.

The best algorithm for the training Root Mean Square Error (RMSE) is Polynomial Regression (PR) having an average error of 2.93115E-4 whilst the worst is LSTM Neural Network (LSTM) having an average error of 2.69158E-3 between all simulation runs. More importantly, for the testing Root Mean Square Error (RMSE), the best algorithm is Linear Regression (LR) having an average error of 9.55991E-3 whilst the worst is Polynomial Regression (PR) having an average error of 0.60096 between all simulation runs.

Finally, some of the novel approaches perform better than the benchmark approach in terms of accuracy during training and testing which highlights their benefits. In particular, some of the novel approaches applied with the Long Short-Term Memory Neural Network (LSTM) outperform the corresponding benchmark approach. Likewise, some of the novel approaches applied with Linear Regression (LR) perform identically with the corresponding benchmark approach. This improvement of accuracy is achieved at the expense of only marginal loss of efficiency.

Acknowledgements

Firstly, I would like to take this opportunity to thank my project supervisor, Dr Alexander Gegov, for all the assistance, support and feedback throughout the project processes; Especially during the final few months, in which he was able to not only meet up in a consistent basis but giving me the exact tools and help needed to succeed in this project.

I would not have been able to complete this project without the support of my mother Elvira Yaghmour who has been on my side since day one in order for me to continue pushing through.

To Leslie Botha and Heidi Botha who have been like parents to me all the way since I started my university career, huge thank you for always being there when I needed it the most without failure. I hope I can make you guys proud.

A special mention to Fakruddin Mohammed who has mentored and guided me to the completion of this project. I couldn’t thank you enough for your time given to me, I will always be thankful.

Contents

Chapter 1

Introduction

1.1 Problem & Background

This chapter presents an overview of machine learning and modelling. It also states the aim, objectives, outcomes, benefits of the research described in this project and introduces the overall structure of the report.

Machine learning based modelling is an effective approach that can be used by people who would like to gain a deeper understanding of the stock market. Nowadays, with the popularity of the stock market investments increasing, this approach generates an impact and more attraction to individuals that are looking for new ways of financial development and economic growth. That is why machine learning based modelling has been implemented in many software applications in recent years for the purpose of helping investors by means of stock price prediction. As mentioned in (flatworldsolutions.com)

Machine learning is concerned with the investigation and computer modelling of learning processes in all of their forms. It is currently organised around three main research areas:
(1) Task-oriented studies, also known as the engineering method, are the creation and study of learning processes to increase performance in a predetermined series of tasks.
(2) Cognitive simulation is the study and simulation of human learning processes using computers.
(3) Theoretical analysis—a non-domain-specific theoretical exploration of the space of potential learning methods and algorithms.

In addition to (1), (2), (3) above, machine learning includes also exploration of alternative learning approaches, such as the discovery of various induction algorithms, the scope and limitations of some methods, the knowledge that must be accessible to the learner, the issue of dealing with incomplete training data, and the development of general techniques applicable in several task domain. All of these areas of exploration for machine learning are equally important as (1), (2), (3) Marcus Thorström (2017)

Machine learning has become important and useful in recent years due to the wide variety of applications as well as remarkable capacity to adjust and provide solutions to complex problems in a timely, accurate, and productive manner. The iterative aspect of machine learning is important because models will evolve independently as they are exposed to new data. They use previous computations to generate

consistent, repeatable decisions and outcomes. It's an old science that's gaining new traction thanks to the increased availability of large data sets (Shelley Elmblad, 2021)

This report goes through the implementation, design and thought process of creating a system that would handle specific amounts of datasets to determine the prediction of stocks. By creating a software system where several machine learning algorithms are combined and analysed simultaneously, the report provides meaning to large data to get a better understanding of how the accuracy and efficiency of these algorithms can be affected and how important it is for the specific outcome expected.

These days, potential investors want to get not only an accurate but also an efficient prediction of stock prices depending on their individual circumstances and preferences. Therefore, it is important to test some established machine learning algorithms and see how they work for stock prediction in terms of both accuracy and efficiency. In this context, it is desirable to increase the accuracy of the algorithms without compromising their efficiency.

This report looks at the increase of prediction accuracy for stock prices making use of large data. However, the latter can compromise the efficiency of the models. For this reason, this project is focused on achieving higher accuracy by using more data without compromising the efficiency of the solution as much.

1.2 Aim

The aim of this report is to systematically explore suitability and applicability of several popular machine learning algorithms for financial modelling of the stock market by means of a research focused engineering project. This aim is achieved through the following objectives.

1.3 Objectives

1. To evaluate accuracy and efficiency of stock market prediction in a complementary way
2. To visualise results from algorithm training and testing in an integrated way
3. To execute several machine learning algorithms in a simultaneous way
4. To develop a software tool for stock price prediction
5. To implement several established machine learning algorithms in this software tool

The first 3 objectives have a research focus and the last 2 have an engineering focus.

1.4 Outcomes & Benefits

Although, an exact prediction of the outcomes in a research might be impossible to state this early in the report chapter, a quite precise as to the nature and scope of the outcomes and as to who might benefit is attempted.

Expected outcomes are:

- Produce stock price prediction software which meets all the objectives from section 1.3 and benefits investors on the stock market
- Identify weaknesses and strengths of different machine learning algorithms and how they can complement each other

Potential benefits are:

- Attract more investors to help companies grow and make a bigger contribution to the economy
- Help users and employees of the stock market industry to understand better stock price prediction

1.5 Thesis Organization

This report shows in detail the functionality and performance characteristics of a specific software developed for financial modelling. Different chapters are shown throughout the report where each of the chapters to follow next have a sequence of the previous.

Chapter 1, the introduction section, where it clearly states the problem & background of the research, aims and objectives of the report as well as outcomes and benefits of the software.

Chapter 2, the literature review section, focuses on conducting research introducing different machine learning algorithms, software artefacts, a conclusion and a summary section of the chapter. This chapter is a less technical description but more of the research.

Chapter 3, the methodology section, which provides a more detailed description of the research, shows how the management of the project was developed throughout. In more technical details the research part of the algorithms used and software methodology of the approach chosen, conclusion and summary of the chapter.

Chapter 4, the requirements section, which includes the research and software strands of the artefact as well as the summary of the chapter.

Chapter 5, the design section, shows machine learning charts for the specific algorithms chosen and diagrams of the software implemented for the design model and summary of the chapter.

Chapter 6, the implementation section, shows screenshots of the graph results, a software artefact part explaining the code and summary of the chapter.

Chapter 7, the testing and evaluation section, shows a benchmark approach of the results of each algorithm and evaluates requirements met from a previous chapter as well as the summary of the chapter.

Chapter 8, the conclusion section, mentions contributions being accomplished, dissemination, further research that can take place, wider context, general reflections.

The report finishes with references containing relevant information sources used and appendices section with supplementary material for the project.

Chapter 2

Literature Review

2.1 Introduction

In order to achieve the aims and objectives stated in the previous chapter, a detailed literature review on the chosen research topic is presented in the current chapter. Many useful softwares for stock prediction exist on the market to help not only investors but businesses that are also involved, machine learning being the popular tool used to run these softwares. People have implemented machine learning into their stock market prediction as a strong helpful tool to generate more into their investments results. Machine learning is usually implemented by means of a wide range of methods and algorithms. Some of these methods and algorithms are reviewed in section 2.2 of this chapter in the context of their applicability for stock price prediction This chapter also reviews popular software artefacts for stock price prediction that use different machine learning methods and algorithms in section 2.3.

2.2 Machine Learning Algorithms For Stock Prediction

There are different types of machine learning artefacts available for stock prediction on the market (Hiransha et al., 2018). However, depending upon preferences and goals different software artefacts use particular algorithms. This section discusses on a general scale, what types of algorithms there are out there based on research that people have used for their stock prediction software in order to decide what algorithms to choose for the development of the software in this report. Stock price prediction is usually considered in the context of two main groups that are highlighted in sections 2.2.1-2.2.2 below whether it is short term or long term. As machine learning algorithms are fairly generic and used for both short and long term prediction, they are then described in sections 2.2.3-2.2.5 within a general overview.

2.2.1 Short-Term Prediction

Short-Term predictions are linked to short-term investments which are held for less than a year. Short-term investments are also named by many people as temporary investments. Many short-term investments are typically sold by many after a period of only 3-12 months as people with not a lot of experience expect quicker returns in a short time. Some common examples of short term investments include money market accounts, high-yield savings accounts, government bonds, and treasury bills. Normally, short term investments are of highly liquid assets (Troy Segal, 2021).

Referring to financial assets, short-term investments have a few additional requirements that are owned by a company. Recorded in a separate account and listed in a particular asset section of the organization, short-term investments in this context are investments that a company has made available for individuals, employees or investors in general that are expected to be sold within one year as (Katelyn Peters, 2021) stated in investopedia.com

2.2.2 Long-Term Prediction

Long-Term predictions are linked to long-term investment which are considered investments held for more than 1 year and 1 day. A long-term investment is an account on the asset side of a company's balance sheet that represents the company's investments, including stocks, bonds, real estate and cash (Claire Boyte-White, 2021)

The long-term investment has an obvious and big difference from the short-term investment. As mentioned, short-term investments will be sold quickly, whereas long-term investments will be held for a longer period of time, sometimes could even never be sold. Long-term investors are willing to consider risks that come with it, just for an uncertain potential higher return. You must be patient for a longer period of time. It is advisable of the individual having enough capital to afford a quantity that will not be touched for a while as (Alexandra Twin, 2021) stated.

2.2.3 Regression Analysis

Regression Analysis (RA), is a generally useful methodology for expectation and forecasting, where its implementation has a generous cover within the machine learning field. Second, in certain circumstances regression analysis can be utilized to deduce causal connections between the dependent and independent factors. Critically, regressions on their own just uncover connections between a dependent variable and independent in a fixed dataset. To utilize regression for prediction or to derive causal connections, respectively, a researcher must carefully justify why existing relationships have predictive power for a new context or why a relationship between two variables has a causal interpretation (Nirbhey Singh, 2017). Some regression analysis methods are discussed in the following paragraphs.

Linear Regression (LR), was developed in the statistics field, but it is used in machine learning as well. Linear regression builds a model that assumes a linear relationship between the input variables (x) and the output variable (y) along with a corresponding coefficient for each input variable. Machine learning uses gradient descent to update these coefficients to reduce the error in forecasting the output variable. (Nirbhey Singh, 2017)

Polynomial Regression (PL), is a slightly more advanced regression algorithm than linear regression. It models the relationship between a dependent(y) and independent variable (x) as nth degree polynomial. If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a non-linear dataset, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased. So for such cases, where data points are arranged in a non-linear fashion, we need the Polynomial Regression model (Nirbhey Singh, 2017)

Random Forest Regression (RF), is a supervised learning algorithm for classification and regression purposes that consists of a large number of decision trees that operate as an ensemble. An ensemble method combines the predictions from multiple decision tree algorithms together to make more accurate predictions than any individual model would. Random forest constructs a multitude of decision trees with a training set and outputs the class for a classification problem or prediction for a regression problem. (Katherine Gail Nowadly and Sohyun Jung, 2020)

2.2.4 Artificial Neural Network

Artificial Neural Network (ANN), is a machine learning algorithm that resembles the workings of the human brain. ANN discovers a new pattern out of investigating a relationship 42 between the inputs and outputs. ANN has three important elements in its structure: input layer, hidden layer, and output layer. The input layer receives the information into the model, the hidden layer computes the patterns among the input data, and the output layer obtains the result. In a neural network, there are multiple parameters and hyperparameters that affect the performance of the model. Each node in the network is assigned weights that can be interpreted as the impact that node has on the node of the next layer. Depending on the weighted sum value, an activation function defines whether a given node should be activated or not. Finally, based on the result, the model adjusts the weights of the neural network to optimize the following various cost minimization functions. (Murkute Amod, Tanuja Sarode, 2015). Some common types of artificial neural networks are discussed in the following paragraphs.

Feed Forward Neural Network (FFNN), the information only moves in one direction from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice. Feed-forward neural network has no memory of the input it receives and is bad at predicting what is coming next. Because a feed-forward network only considers the current input, it has no notion of order in time. It simply can not remember anything about what happened in the past except its training. (Niklas Donges, 2019)

Recurrent Neural Network (RNN), is a class of neural networks that is helpful in modeling sequence data. Derived from feedforward networks, RNN exhibits similar behavior to how human brains function. Recurrent neural networks are of big use to produce predictive results in sequential data that other algorithms can not do (Murkute Amod, Tanuja Sarode, 2015).

Long Short-Term Memory Neural Network (LSTM Neural Network), is very powerful in sequence prediction problems because it is able to store past information. This is important because the previous price of a stock is crucial in predicting its future price. (D. M. Q. Nelson et. al, 2017)

2.2.5 Other Machine Learning Methods

Support Vector Machine (SVM), is a supervised learning algorithm that is used for classification. It is based on the structural risk minimization principle, which sets a hyperplane that maximizes the margin of separation between different classes and minimizes the expected error of a learning machine. Through training the data, one unique hyperplane, called optimize hyperplane, is created to separate the data. Since most real data is not linearly distributed, a kernel function is created to classify the non linearly distributed data. The kernel function is applied on each data instance to map the original nonlinear instances into a high dimensional space, where they become separable. By calculating the distances from 41 the kernel function to the instances, SVM can be used as a regression method to predict the future values too. (Vapnik, 1998)

Distributed Decision Tree (DDT), is a machine learning method that finds patterns inside of data and makes a rule to classify said data. This method classifies instances into branch-like segments, hence its nomenclature. The decision tree structure has internal nodes representing a feature and branches representing a decision rule, with each leaf node representing the outcome. The decision tree algorithm starts by selecting the best attribute using attribute selection measures to split the data. The attribute selected becomes a decision node and breaks the dataset into smaller subsets. By repeating this process recursively for each instance, a tree is formed. At each node of the decision tree, the information entropy gain generated by the split is used to evaluate whether the variable is meaningful or not (S. Patil et al, 2019)

Some people use a combination of different machine learning algorithms for stock prediction. In accordance with different approaches in order to try and combine the advantages that each of these algorithms can offer (Marcus Thorström, 2017), (Mehtab, S. and Sen, J, 2021)

2.3 Software Artefacts For Stock Prediction

This section considers first some existing software artefacts for stock prediction and focuses on their development approach and their functionality attributes. These two aspects are then discussed in a more general software engineering context in subsection 2.3.1- 2.3.2. There are many different programming languages and software environments to choose from, four different approaches for implementation will be reviewed for stock prediction.

Software using Python language
“Modeling high-frequency limit order book dynamics with support vector machines”. This project uses Support Vector Machines (SVM) for tracking the dynamics of foreign exchange markets. However, applying SVMs in financial markets is still something new. This artefact makes use of “multi-class SVM” methods to capture the dynamics in high-frequency limit order book data and forecast movements of the mid-price over short time intervals. NASDAQ stock market data is used to show the multi-class SVM models built in. This paper not only predicts various metrics with high accuracy, but also delivers predictions efficiently. In addition, experiments with simple trading strategies using these signals were observed to produce positive payoffs with controlled risk (Kercheval, A. N., Zhang, Y. 2013)

Software using R language
“Big Data Prediction of Stock Prices in R”. This project aims to introduce a variety of methods for predicting stock prices. Predicting techniques used are Holt Winter Filtering, ARIMA model, Neural Networks, Polynomial & Linear generalization trends, using the Mean Absolute Error for the calculations. However, for predicting the stock prices accurately, a compound pull of 25 prediction techniques was carried out. The installation process in order to get it up and running was time consuming and probably confusing for people that want to learn stock prediction using machine learning for the first time (Aumann.D, 2019)

Software using MATLAB
“Matlab Module for Stock Market Prediction using Simple Neural Network”. Using Genetic Algorithm and Neural Networks, (Shikhar, 2018) stated that Genetic Algorithms are more suited for optimization problems and was used to optimize the parameters of the Neural Network for more accurate predictions Also, this software implemented long-term prediction by using 8000 iterations for training for different numbers of years, 5, 10 and 19 years (Shikhar, 2018)

Software using Jupyter Notebook
“Jupyter notebook which illustrates moving average crossover strategy and neural network predictions for stock trading”. This software discusses three strategies for decision making in the stock market using short-term prediction by training a neural network on historical data, the results using this platform were successfully achieved

as the results stated that it is very difficult to establish all of the factors influencing the market and to feed them into a neural network for training (github.com/jer805/)

Most softwares use short-term prediction with an iteration number for training the model of 10 and number of days to run of 50. Additionally, most trading systems that use machine learning are automated systems that can result in anomalies such as duplicated orders and missing orders ("Automated Trading Systems: The Pros & Cons", 2019).

2.3.1 Development Approach

Waterfall model: has five clearly defined phases being requirements gathering, system design, implementation, verification and finally, maintenance (Ahmed. 2012). Each stage is developed in the order mentioned. Every stage is completed sequentially, it respects every step as it comes, the project then moves on to the next phase in the development lifecycle. Never concerned with scrums, iterations, or revisions, the waterfall model provides a simple, clear and rigid structure for development. This simple structure makes it an attractive candidate to consider, as it does give the solid structure needed for the project. However, one key issue with this approach is its rigidity (Wysocki, C. 2006). When a project is more complex means that the project could face challenges that might require fixing along the way and backtracking in order to address them. Royce (1987) notes that due to the testing phase being a late phase in the project, design flaws or missing requirements can cause a substantial delay.

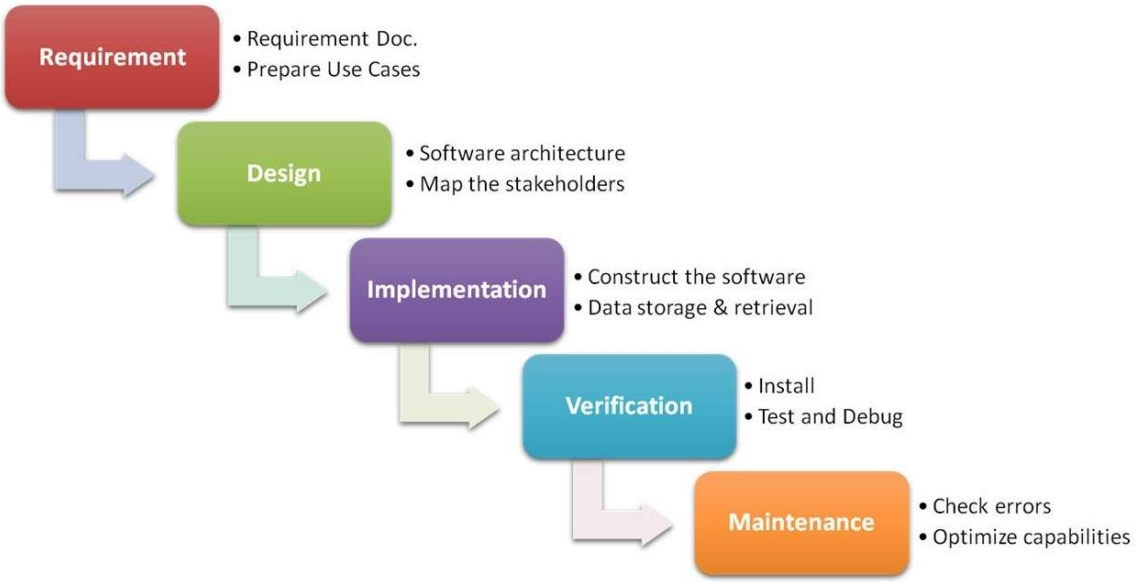


Figure 2.1: Waterfall Methodology Model

Agile model: is a software development methodology that uses an approach that evaluates the requirements of the project throughout the development lifecycle (Matharu et al. 2015). The advantage of this methodology is that the client is very engaged in the development of the software. agile development may not be suitable for large teams (Matharu et al. 2015). Thankfully, this is not relevant to this project as a singular author develops it. Secondly, agile methodologies benefit mostly from highly skilled individuals(Matharu et al. 2015). Agile methodologies can be difficult to efficiently execute by inexperienced teams, as they rely on implicit knowledge rather than documentation.

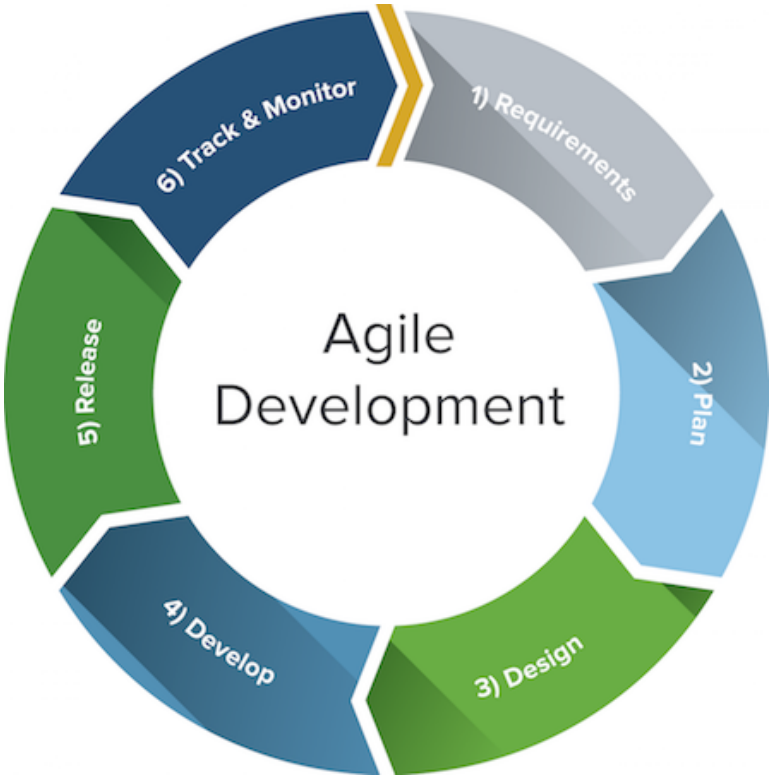


Figure 2.2: Agile Methodology Model

Spiral model: is a more complex development approach where reducing risk is the main strength of this model. The model lifecycle adds a lot of hazard investigation, where important plans are made in case specific danger occurs. This strategy is useful for huge scope activities and ventures that have a high danger as fundamental danger examination is led to relieve these dangers. The weakness of this current model technique is that it incorporates a single point of failure, that being the danger investigation as any disappointment in the danger examination period of the venture can lead to the whole project failing. Furthermore, the model is very complex in contrast with the Waterfall and Agile Model (Boehm, B. W. 1995)

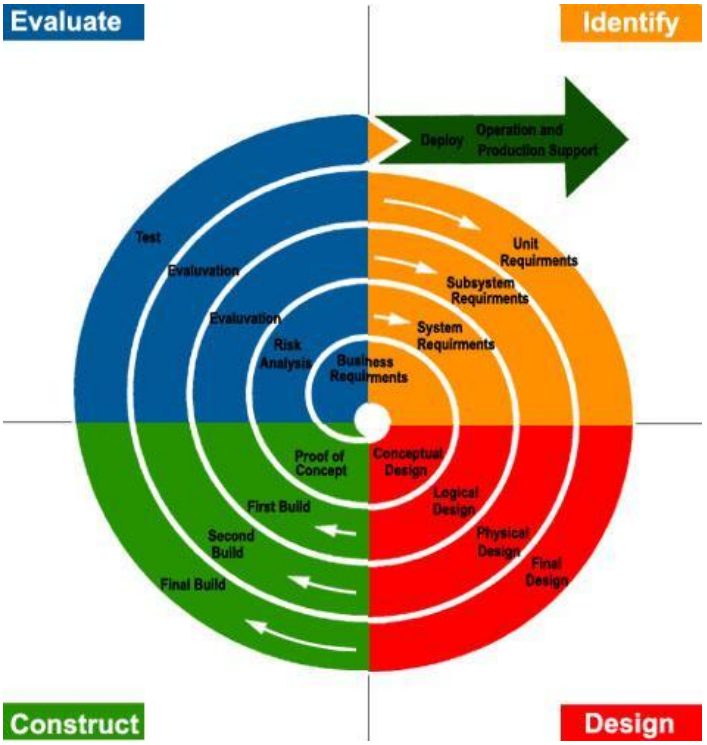


Figure 2.3: Spiral Methodology Model

2.3.2 Quality Attributes

Quality attributes are the attributes that describe the quality of the software. There are several software quality attributes some of which will be reviewed in the paragraphs below from different textbooks (Gomaa, H. 2011), (Barbacci, M. 1995)

Functionality
This attribute decides the agreement of a product driven framework with real prerequisites and specifications. Most Software Testing experts see this trait as essential and a first prerequisite of an advanced application, and would thus advocate the presentation of tests that evaluate the ideal usefulness of a framework in the underlying phases of Software Testing drives.

Maintainability
Programming to be written in such a manner with the goal that it can develop to meet the changing requirements of clients. This is a basic trait since programming change is an unavoidable necessity of a changing business climate.

Availability
Is a demonstrative concerning whether an application will execute accordingly the tasks doled out to perform. Incorporates certain ideas that can be related with the software such as dependability, security, integrity, performance, reliability, and confidentiality. Likewise, good availability implementation demonstrates that a software artefact will fix any working issues so that assistance blackout periods would not surpass a particular time esteem.

Interoperability
Programming driven frameworks could be needed to liaise and tackle certain tasks. Interoperability represents the capacity of two softwares to take part in the trading of data by means of specific interfaces. In this way, software quality assurance engineers should look at the interoperability property as far as both syntactic and semantic interoperability.

Performance
This attribute relates to the capacity of a product driven framework to adjust to timing necessities. From a testing perspective, it suggests that software testing engineers should check whether the framework reacts to different occasions inside characterized time limits. These occasions may happen as clock occasions, measuring interferences, messages and demands from various clients and others.

Testability
Programming testability demonstrates how well a product driven framework permits software testing experts to direct tests in accordance with predefined measures. This trait likewise evaluates the straightforwardness with which software quality

assurance designers can create test measures for a system and its different segments. Architects can evaluate the testability of a framework by utilizing different strategies like embodiment, interfaces, designs and low coupling

Security
This attribute estimates the capacity of a software to capture and obstruct unapproved activities that might actually harm the software. The characteristic accepts significance since security signifies the capacity of the software to shield information and guard data from unapproved access. Security additionally incorporates approval and confirmation strategies, assurance against network assaults, information encryption and such different dangers. It is basic for software testing experts to often improve security and keep an eye on the software.

Usability
Each software is intended for convenience to achieve certain assignments. Usability within a system means the straightforwardness with which clients can execute tasks with ease on the framework; it additionally shows the sort of client support given by the framework. Moreover, software quality assurance engineers should test programming to check whether the software upholds diverse accessibility kinds of control for individuals with incapacities. Usability is a critical attribute for a software to be used by many.

These are some of the most important software quality attributes to be considered within a development for stock prediction.

2.4 Conclusion

This section describes the choices made for this project based on the literature review analysis introduced in the sections 2.2 - 2.3 above.

Prediction type: Short term prediction has been chosen for the development of the software artefact. The reason for this choice is that the stock market is a very dynamic market and nowadays investors want to make profit quicker, they don't want to wait for too long for the investment return to happen and most importantly they can sell their shares at any time without a potential need of credit that they can’t hold in stocks for a long period of time.

Machine learning algorithms: Linear regression (LR), polynomial regression (PL), random forest (RF) and long short-term memory (LSTM) neural networks are the machine learning algorithms choices for the stock prediction in this project. The reason for these particular choices is that they have been more widely used than the others for stock prediction and they work well for the short term prediction.

Programming language: Python programming language has been chosen for the successful implementation of different algorithms within this project. The reason for this choice is for its widely successful use in machine learning for the prediction of

stocks as well as the increase in its popularity over the years. In contrast, the other languages reviewed in section 2.3 are more difficult to implement and less efficient.

Software environment: Jupyter Notebook has been chosen for the programming environment of the software artefact. The reason for this choice is that Jupyter Notebook is mostly used when it comes to stock price prediction development, It also provides an easy built in mechanism for users to approach and develop software more efficiently as it allows you to combine code, images, plots, and comments in one place which covers the needs for the development of the software part of this project.

Development approach: Waterfall development approach has been chosen for the life cycle development of the software artefact. The reason for this choice is that this approach is suitable for this project which had clearly defined and determined goals from the start. Also, the stock price prediction process has a sequential structure with clearly defined individual stages and this structure is easy to implement using the waterfall approach.

Quality attributes: Usability and maintainability are the choices of the quality attributes for the softwares reviewed and the ones to consider for the software artefact to be developed. The reason for these choices is that they are the most critical attributes when developing software for stock price prediction. The software will have these attributes because having a good structure with separate segments and very useful comments within the code means that the software is then usable, anyone can use and maintain it and perhaps update it in the future easily.

2.5 Summary

Two main strands were successfully reviewed based on research. The discussion of different types of machine learning algorithms for stock prediction being used in section 2.2 and several software artefact for stock prediction that uses those algorithms in section 2.3.

Based on this review, justified choices were made in the conclusion section in this chapter for the development approach. This section is particularly crucial for the remainder of this report as it identifies what exactly will be discussed and implemented in the following chapters. Development approach, software environment, language chosen, software quality attributes (usability, maintainability), algorithms to be used and short term prediction are all important selections made for the continued success as it also gives a good logical flow throughout this report.

The following chapter (3), presents a more in depth analysis of the choices made in the conclusion section.

Chapter 3

Project Management & Methodology

3.1 Introduction

This chapter describes in more technical detail the choices made in the previous chapter, in section 2.4. Some choices are presented in sections 3.3 in terms of the machine learning algorithms to be used and review of their accuracy and efficiency performance. The rest of the choices made are presented in section 3.4 for the software methods considered. These sections are preceded by section 3.2 which presents a well defined plan and clearly specifies the approach taken towards this project.

3.2 Overall Project Management

The project management approach for this specific project contains three core sections and are done by the waterfall development life cycle requirements which are: First, project research for the research and understanding of the theory part. Secondly, the software development approach taken for the implementation of the artefact and finally, the documentation for the final finalisation of the write-up. Time allocation to each section can be seen in the figure 3.1. With the majority of the time allocated to the project research. As figure 3.1 shows, all activities have been planned intentionally in a very structured way with a high level of detail.

Task Order	Task name	Duration	Start Date	End Date
1	Complete Project Execution	8 months, 21 days	01/10/2020	21/05/2021
2	Project Research	8 months	01/10/2020	01/05/2021
3	Machine Learning Research	1 month	01/10/2020	01/11/2020
4	Stock Prediction Research	1 month	01/11/2020	01/12/2020
5	Algorithms for Stock Prediction Research	3 months, 15 days	01/12/2020	15/02/2021
6	Literature Review Completion	2 months, 15 days	15/02/2021	01/05/2021
7	Software Development	4months	01/11/2020	01/03/2021
8	Software Design	15 days	01/11/2020	15/11/2020
9	Software Implementation	2 months	15/11/2020	15/01/2021
10	Software Testing	1 month	15/01/2021	15/02/2021
11	Software Evaluation	15 days	15/02/2021	01/03/2021
12	Documentation	3 months, 21 days	01/02/2021	21/05/2021
13	Report Design	1 month	01/02/2021	01/03/2021
14	Report Write-up	2 months	01/03/2021	01/05/2021
15	Report Adjustments	10 days	01/05/2021	10/05/2021
16	Report Completion	11 days	10/05/2021	21/05/2021

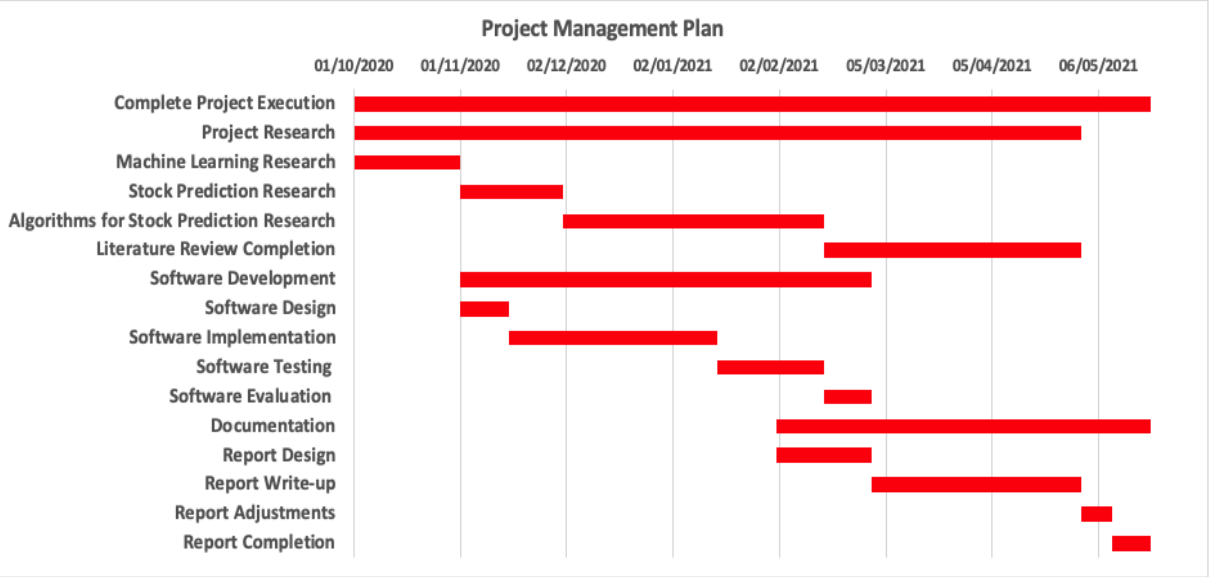


Figure 3.1: Project Management

3.3 Research Methodology

Four different machine learning methodologies for stock prediction from many possibles on the market described in section 2.2 were selected. Those are: linear regression (LR), polynomial regression (PL), random forest (RF) and Long short-term memory (LSTM) neural networks, in that particular order. Where each of them will be explained in more detail in the following subsections. Finally, a detailed discussion of those algorithms in terms of accuracy and efficiency is also presented.

3.3.1 Linear Regression

This algorithm was the first kernel to be used in a Support Vector Machine (SVM) and has no impact on the dimensionality of the data as opposed to the other kernels. The linear kernel is related to a linear equation. The definition of the linear kernel is: K(xi , xj) = x T i xj

In linear regression, a linear equation is fitted to a data set and minimizes the squared error between estimates and actual values. In this implementation, the ordinary least squares (OLS) solution is used where a matrix X is computed to give the best estimate. The form of the regression can be seen in (Equation 1), where the shape of ***b*** is a ***1 × n*** matrix, the shape of ***a*** is a ***n × p*** matrix and the ***X*** is a 1 × n matrix (Mehtab, S. 2021)

(Equation 3.1), ***b = aX***

The objective of ordinary least squares (OLS) is to minimize the euclidean distance of the estimation of X as can be seen in (Equation 2), where $||...||^2$ is the euclidean distance.

(Equation 3.2), ***min(||b − aX||²)***

A common equation used for linear regression is $y = \beta_0 + \beta_1 x_1$, figure 3.2 shows how the algorithm can perform in a graphical way based on data fitted into the algorithm.

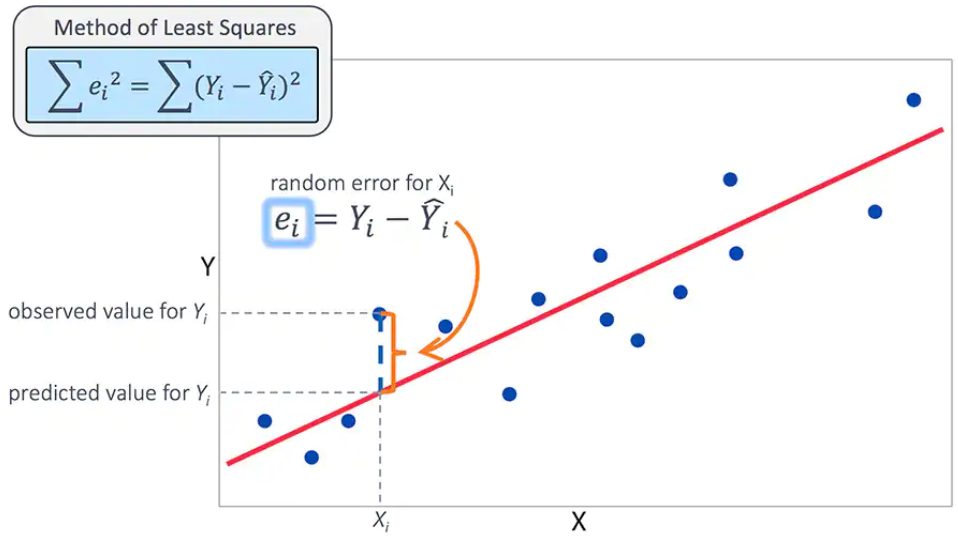


Figure 3.2: Graph plot - Linear Regression

3.3.2 Polynomial Regression

Very similar to the Multiple Linear regression (MLR) but instead of the different variables $x_1, x_2, x_3 \ldots x_n$ as shown in equation3.3, Polynomial has the same variable x_1 but with different powers of that variable for example: $x_1 + x_1^2 + x_1^3 \ldots x_1^n$ using two or more explanatory variables as seen in equation 3.4 below.

(Equation 3.3), $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_n x_n + \ldots$

(Equation 3.4), $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \ldots \beta_n x_1^n + \ldots$

The equation in order to conduct a Polynomial regression is shown in equation 3.5 where β_0 is the constant coefficient, $\beta_1 x_1$ the simple linear regression described in section 3.3.1 and $\beta_2 x_1^2$ is what gives the parabolic effect to the curve as shown in figure 3.3

(Equation 3.5), $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2$

Polynomial Regression can be very useful when managing multiple data and has its own use cases such as describing how diseases, pandemics or epidemics are spread across the population. In this case, figure 3.3 shows how the algorithm performs perfectly in a graphical way based on the equations already described.

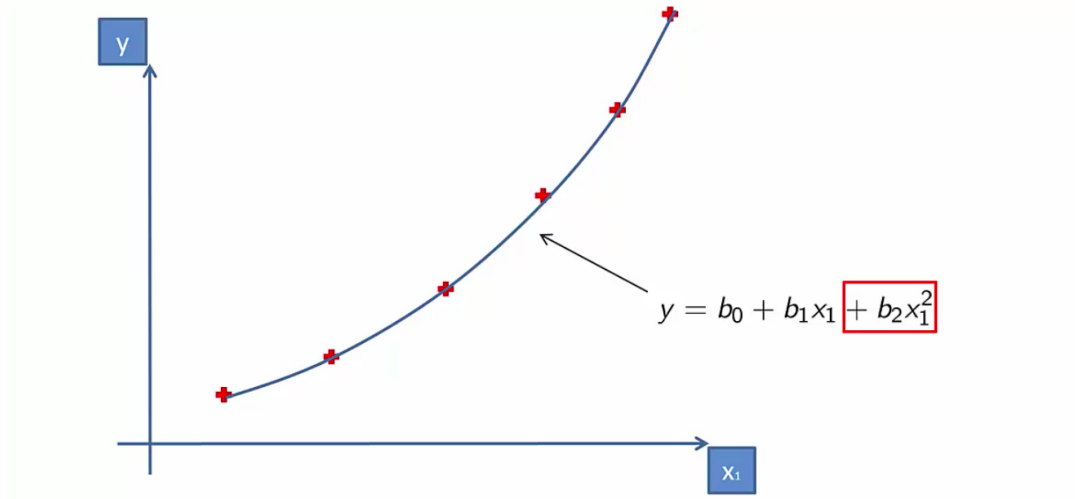


Figure 3.3: Graph plot - Polynomial Regression

3.3.3 Random Forest Regression

Rationally, the combination of multiple learning decision trees increases the accuracy being the purpose of this bagging technique which is to average noisy and unbiased decision trees to create more lower variance trees as more data is processed. The figure 3.4 shows a sample set matrix ***S*** of training samples that the algorithm will process to create a classification model, where f are the features, ***C1*** and ***Cn*** refers to multiple features and a training class to be created.

$$S = \begin{pmatrix} fA1 & fB1 & fC1 & C1 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ fAn & fBn & fCn & Cn \end{pmatrix}$$

Figure 3.4: Random Forest Regression matrix

From the sample set matrix, many subsets can be created with random values as shown in figure 3.5, the subsets (***S1***, ***S2***) start at a random value (***fA12, fA2)*** and subset ***Sm*** refers to the M^{th} number of subsets created. These subsets are called decision trees in which they create different variations of the main sample set matrix for a final prediction result.

$$S1 = \begin{pmatrix} fA12 & fB12 & fC12 & C12 \\ fA15 & fB15 & fC15 & C15 \\ . & . & . & . \\ . & . & . & . \\ fA35 & fB35 & fC35 & C35 \end{pmatrix} \quad S2 = \begin{pmatrix} fA2 & fB2 & fC2 & C2 \\ fA6 & fB6 & fC6 & C6 \\ . & . & . & . \\ . & . & . & . \\ fA20 & fB20 & fC20 & C20 \end{pmatrix}$$

$$Sm = \begin{pmatrix} fA4 & fB4 & fC4 & C4 \\ fA9 & fB9 & fC9 & C9 \\ . & . & . & . \\ . & . & . & . \\ fA12 & fB12 & fC12 & C12 \end{pmatrix}$$

Figure 3.5: Random Forest Regression subsets

3.3.4 Long Short-Term Memory Neural Network

In a Long Short-Term Memory Neural Network (LSTM) the nodes are recurrent but they also have an internal state as working memory space which means information can be stored and retrieved over many time steps. Each LSTM cell unit as shown in figure 3.6, maintains a cell memory. For each time step the next LSTM cell unit can choose to read from it, write to it or reset the cell using an explicit gating mechanism.

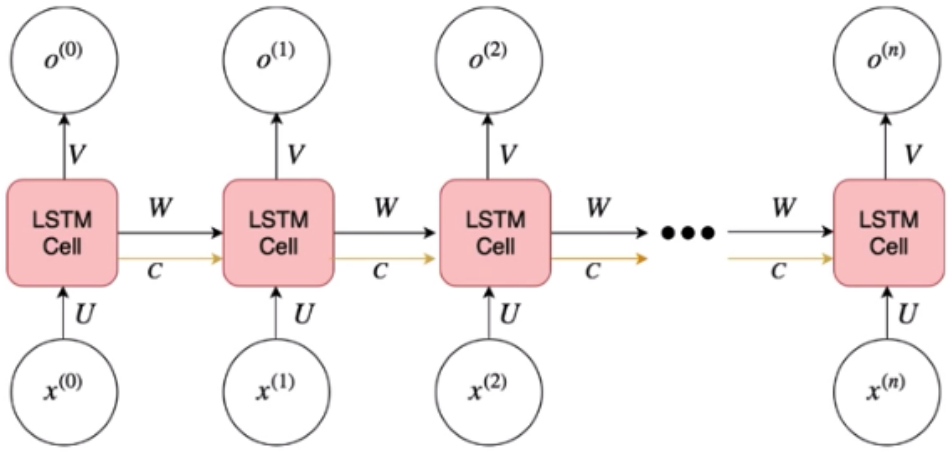


Figure 3.6: Long Short-Short Term Memory Neural Network

Each cell unit has three gates, the input gate which controls whether the memory cell is updated using the formula presented in the equation 3.6, the forget gate which controls if the memory cell is reset to 0 using the formula presented in the equation 3.7 and the output gate which controls whether the information of the current cell state is made visible using the formula presented in the equation 3.8

(Equation 3.6), $$i^{(t)} = \sigma\left(W^{i}\left[h^{(t-1)}, x^{(t)}\right] + b^{i}\right)$$

(Equation 3.7), $$f^{(t)} = \sigma\left(W^{f}\left[h^{(t-1)}, x^{(t)}\right] + b^{f}\right)$$

(Equation 3.8), $$o^{(t)} = \sigma\left(W^{o}\left[h^{(t-1)}, x^{(t)}\right] + b^{o}\right)$$

3.3.5 Research Method Attributes

Accuracy and efficiency are very critical performance indicators for stock price prediction, it is important for investors to get accurate predictions to know where to invest but also to get quick predictions as stocks is a very dynamic market. Is also key to consider these attributes in correlation with each other, complementing one another depending on investors preferences. In some cases efficiency may be more important than accuracy. If someone does not want to win too much but wants to win straight away, they are not interested so much in accuracy but they are interested in the efficiency of the predicted value. On the other hand others might want to win more regularly but not straight away leading more towards accuracy focus.

3.4 Software Methodology

The software to be developed is intended mainly to achieve the aim of this research focused project - which is to evaluate the efficiency and accuracy of the machine learning algorithms, as mentioned in section 3.3.5. This section explains different aspects of the software methodology chosen in chapter 2, section 2.4. These choices include the Waterfall development lifecycle methodology, the Python programming language and the Jupyter simulation environment. Finally, the quality attributes indicators Usability and Maintainability of the software for stock prediction are considered.

3.4.1 Waterfall

The history of the Waterfall model came originally from a paper called "*Managing the Development of Large Softwares Systems"* published in 1970 written by Dr Winston Royce. Developing software he realised that there were some patterns that he wanted to improve upon. He talks about the "Waterfall process" when it was not a common term to hear. However, it is not what he proposed as he mentions that it was not a good idea estating in the paper: "I believe in this concept, but the implementation described above is risky and invites failure", he intended to described his process for how software development should be carried out instead as shown in figure 3.6. He tried to find a way to deliver projects that were on time, met budget and scope figuring out how to make all work, the highest cons were coding and testing due to time. He was never going to find a solution working with all technology and arcade software as it was back then. (Royce, W. W. 1987, March)

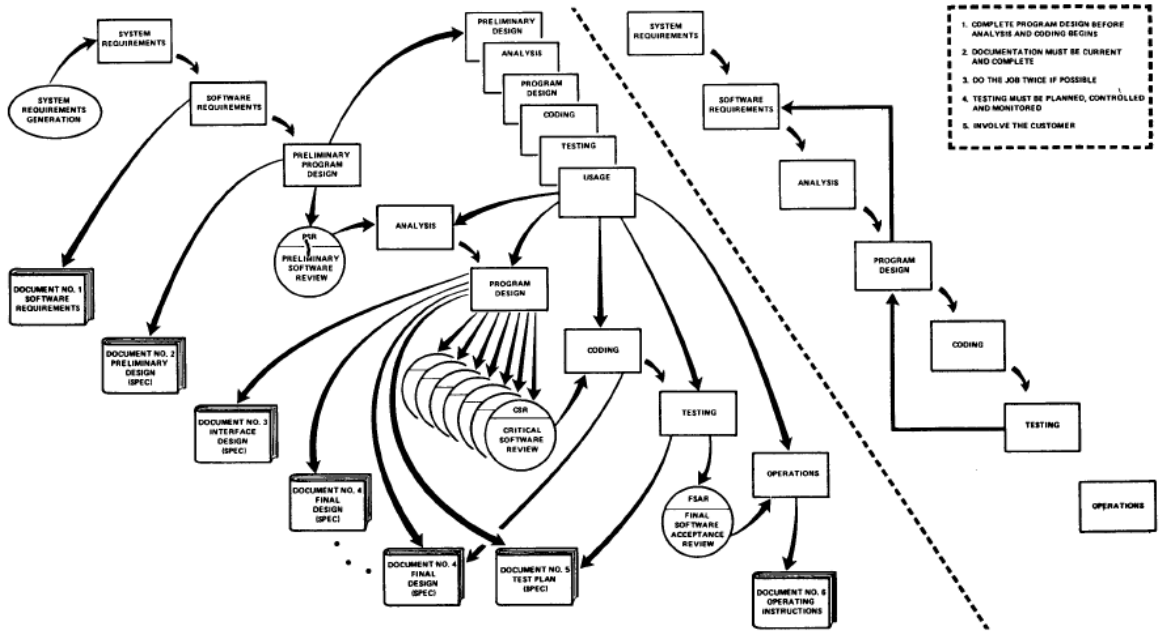


Figure 3.6: Waterfall model by Winston Royce

3.4.2 Python Language

Implemented in 1989 in the Netherlands by Guido van Rossum at Centrum Wiskunde & Informatica (CWI), was design to emphasys code readability and was conceived as an upgrade from the ABC language which he also helped creating, inspired by the complex mathematical language of sets Less Commonly Taught Languages (LCTL). Python programming language was also created to handle an interface with the amoebaOS. Guido van Rossum used the good features from the ABC programming language to create Python which would be modified to eventually remove all the flaws in the ABC language. Many companies and governments organization uses Python such as NASA to create a Workflow Automation System (WAS) which is prove to be cheap, fast and effective, Google for system administration tools, various code evaluation tools and Application Programming Interfaces, Facebook which uses 21% of its infrastructure code base providing thousands of libraries and endless code, Netflix to handle their critical application such as the central alert gateway (Data Flair Team (n.d)). Python is shown to have the ability to implement machine learning algorithms.

3.4.3 Jupyter Environment

By having to edit multiple files while developing big projects simultaneously other code editors work very well but are not good when testing or practicing certain concepts as they do not allow to go back to a specific cell after a run, it becomes time consuming to save a file and run everytime is needed. The Jupyter Notebook provides with the advantage of stretchable code cells that can be infinite, multiple commands can be included on specific cells. Very fast as it has an interactive shell and individual cells can be run in the notebook. Jupyter Notebook saves code, good for testing and learning concepts, and output can be seen quickly by running the command which is important in the implementation of machine learning for testing different algorithms and not losing much time being a very efficient computational environment. (Dataquest. 2020, August)

3.4.4 Software Artefact Attributes

Specifically for stock prediction software, Usability and Maintainability performance evaluation indicators are important. Having an efficient structure that allows easy navigation and is informative helps investors complete specific tasks, making those tasks completion a primary target in the development of the software which demonstrates a very usable software for the investors needs. On the other hand, having a good structure with separate segments and very useful comments in the code means that the software is maintained which helps investors use it with ease as it is well written and uncomplicated to understand as well as easy to expand it in the future.

Usability: User experience design focuses on a deep understanding of the users needs, wants and abilities. Usability design is the process of designing the software from the perspective of how it will eventually be used by an actual person. Differs from user experience because it considers the software usefulness in addition to everything else covered by user experience design. Any software should always be tested for Usability, no project is ever truly completed, every project needs to continually evolve and improve. User behaviour changes, when it does the software needs to change to change too (Abran, A. et.al 2003).

Maintainability: Developing big projects, especially with major calculations and plotting graphs such as the implementation of machine learning algorithms for stock prediction, a well maintained program is required. Implemented machine learning algorithms lead to complicated formulations and getting back to the code not knowing fully how it worked in the past could lead to time waste. Maintainability is measured by using comments in order to separate sections of code and be able to see what each segment achieves, where it starts and finishes, giving sensible names to the variables that actually refer to the data avoiding single letters, use procedures or functions to structure the code and eliminate duplicate code, using whitespace to make sections of a program easier to see (Coleman, D. et.al 1994).

3.5 Conclusion

Stock prices are usually influenced by a wide range of financial indicators. Therefore, these prices are characterized by dynamic fluctuations which make their accurate prediction for investment purposes a very challenging task.

In order to address the above challenges, this project introduces several academic novelties and related contributions. The latter are based on the research methodology discussed in section 3.2 above. In particular, these novelties and contributions are as follows:

1) Use of data sets from the past that are substantially longer than the ones used in known existing works. This is done for the purpose of improving the accuracy of the chosen machine learning methods and algorithms while maintaining their efficiency. This novelty is aimed at achieving the first research objective stated in chapter 1, section 1.3, about the evaluation of these methods and algorithms in a complementary way.
2) Graphical presentation of the results from the chosen machine learning methods and algorithms that is more meaningful than the ones used in known existing works. This is done for the purpose of improving the understanding of these methods and algorithms. This novelty is aimed at achieving the second research objective stated in chapter 1, section 1.3, about visualisation of training and testing results in an integrated way.
3) Execution of the chosen machine learning methods and algorithms that is more user friendly than the ones used in known existing works. This is done for the purpose of improving the perception of these methods and algorithms by the user. This novelty is aimed at achieving the third research objective stated in chapter 1, section 1.3, about the execution of several algorithms in a simultaneous way.

3.6 Summary

Two main strands were technically described successfully. The discussion in separate sections for individual types of machine learning algorithms for stock prediction being used for the software, as well as a comparison of accuracy and efficiency as two main system quality attributes for those algorithms in the research methodology. Development approach, programming language chosen, software environment and importance of software quality attributes for a software were discussed.

The following chapter (4), discusses what the software to be developed is expected to do and its functionality.

Chapter 4

Requirements

4.1 Introduction

The research and software methodology introduced in chapter 3 are essential for the specification requirements of the research experiment and the software artefact of this project. These requirements help in the achievement of the objectives introduced in chapter 1. The current chapter explains how the software performs in terms of different modules introduced in the research experiment section 4.2, and the specific elements and functions required by the software in the software requirement section 4.3.

4.2 Research Experiment Specification

This section introduces the requirements for the research experiment with the chosen machine learning algorithms. Each of these algorithms should be simulated using three independent modules which are discussed in this section. These modules should interact in a way whereby each module performs appropriately for the achievement of specific tasks for the algorithms. The first module deals with the selection of the input features for the machine learning algorithms. The second module focuses on the processing of the features by these algorithms. Finally, the third module deals with the presentation of the modelling results from the algorithms.

4.2.1 Input Module Specification

This module should allow investors to select input variables such as number of days in the past with actual stock values and number of training iterations for the chosen machine learning algorithms. These input variables are important as they correlate directly with the actual stock values for the next day in the past. The upper limit of the variation ranges for these input variables should be 100 for the number of days in the past and 20 for the number of iterations. These upper limits are higher than the short term prediction works reviewed in chapter 2. The purpose of this is to explore the impact on the computational accuracy and efficiency of the chosen machine learning algorithms.

4.2.2 Processing Module Specification

This module should process the input variables from the previous module in section 4.2.1 in order to mapped them into output variables. These output variables are the actual and predicted stock values for the next day in the past as well as the training and testing errors for the chosen machine learning algorithms. Each machine

learning algorithm should be executed simultaneously during a single run to facilitate the analytical calculation of the output variables. The purpose is to simulate the computational accuracy and efficiency of these algorithms.

4.2.3 Output Module Specification

This module should map the analytical values of the output variables from the previous module in section 4.2.2 into a graphical presentation. This presentation should be done simultaneously for the chosen machine learning algorithms in regards to the actual and predicted stock values for the next day in the past as well as the training and testing errors for these algorithms. The results for each machine learning algorithm should be visualised simultaneously during a single run to enable investors to make better investment decisions. These decisions are based on the evaluation of the computational accuracy and efficiency of these algorithms.

4.3 Software Requirements Specification

The requirements for the development of the software are divided into two main groups which are applicable to this software. These requirements are explained in terms of functional and non-functional. Each main group is evaluated based on two aspects, ‘must-have’ and ‘should-have’. The requirements needed by the software are labeled as ‘must-have’ or ‘may-have’ depending on whether they are compulsory or optional for the functionality of the software. These requirements follow from the quality attributes discussed in section 2.3.2.

4.3.1 Functional Requirements

Must Have

Dataset Integration
The software must integrate a ‘read file’ implementation within the source code, where it is able to take different datasets from the NASDAQ-traded stock market to successfully read and use the data to analyse the final prediction.

Days Input
The software must allow the user to input a variation of the number of days in the past from the particular dataset used, depending upon preferences and goals they want to achieve. Whether they want more accuracy in the prediction of the results or to test the software capabilities.

Iterations Input
The software must allow the user to input a variation of algorithm iterations numbers depending upon preferences and goals they want to achieve. Whether they want a faster result or longer training for the machine learning algorithms.

Graphical Representation
The software must present simultaneously in a graphical way to the user, the results obtained by the machine learning algorithms chosen. Each algorithm must have an individual graph result as well as a combination of all in the same graph.

May Have

Interactivity
The software may present in a friendly way a clear understanding of the software behaviour allowing the user to have a satisfactory experience of what the software can produce depending on their preferences.

Response Time
The software may respond in the average time taken for every run requested. The results from the chosen machine learning algorithms must be presented in a considerable period of time without the software crashing or stopping.

4.3.2 Non-Functional Requirements

Must Have

Usability
The software must ease the ability of an user to interact with the design product to achieve their own needs effectively, efficiently and satisfy the overall experience of use.

Reliability
The software must describe the most vulnerable points and solutions, analized similar existing applications and evaluate typical problems. Performing well under low latency conditions, being able to obtain the results wanted with the smallest delay possible.

Maintainability
The software must define the time taken for a solution to be fixed. The performance improvement, implementing new qualities or covering the software needs, demonstrates the repair probability for a period of time which must be targeted within the same day.

Performance
The software must assess how the software reacts to the users actions. Content loading speed is a priority, the speed of reaction from when the user enters an input to the end of the process to obtain a defined result.

May Have

Documentation
The software may have reliable documentation for any interested party that wants to continue the development of the software in the future in order to easily maintain, review, revert, and update without much information needed.

Portability
The software may be considered portable. Meaning is able to compile and be executed on different operating systems such as (Windows, Mac or Linux) without creating any problems or errors. Different operating systems should not limit the software ability.

4.4 Summary

This chapter discussed the specification of the modules for the research experiment in this project with regard to the simulation of the chosen machine learning algorithms. It also described the functional and non-functional requirements for the software in terms of two main aspects, ‘must-have’ and ‘may-have’.

The following chapter (5), presents the design of the software to be developed in this project.

Chapter 5
Design

5.1 Introduction

Based on the description of the different modules explained and the requirements needed for the development of the software in chapter 4, a presented design of the software can be achieved. This chapter introduces a general language independent design context of the software. For each of the machine learning algorithms, the design is explained in terms of flowcharts in section 5.2, and to reflect how the different modules are connected to each other, the use of UML is presented in section 5.3 for each of the modules.

5.2 Machine Learning Algorithms

The chosen machine learning algorithms are explained in terms of their specific design in order to simulate these algorithms within the software. Those designs are presented in a flowchart format which shows a general design type of level in terms of block segments. Each block segment has a task assigned that processes information from the previous block. The behaviour of each algorithm depends on the first block segment which relies on the user input in order to behave in a certain way. The following subsection presents those designs for the machine learning algorithms chosen, Linear Regression (LR), Polynomial Regression (PR), Random Forest Regression(RF), Long Short-Term Memory, in that particular order.

5.2.1 Linear Regression

The design for the Linear Regression (LR) algorithm is described in the flowchart diagram in figure 5.1. Each block segment presented in this diagram correlates to one another as an accurate implementation of the algorithm behaviour in the software simulation. There are three different methods that can be implemented when using the Linear Regression (LR) model, Bisquare, Least Square or Least Absolute Residual method; this will depend upon the experimental data points used that differs significantly from the rest of the data points on the graph. In the development of this software, using Linear Regression algorithm for stock prediction, plenty of data in the past is required, therefore the Least Square method for the linear fit is used to find the slope and intercept by minimizing residue for even closer prediction results. The weight referred to in the diagram is the array for (X, Y) observations. The general formula for Linear Regression can be found and explained in section 3.3.1

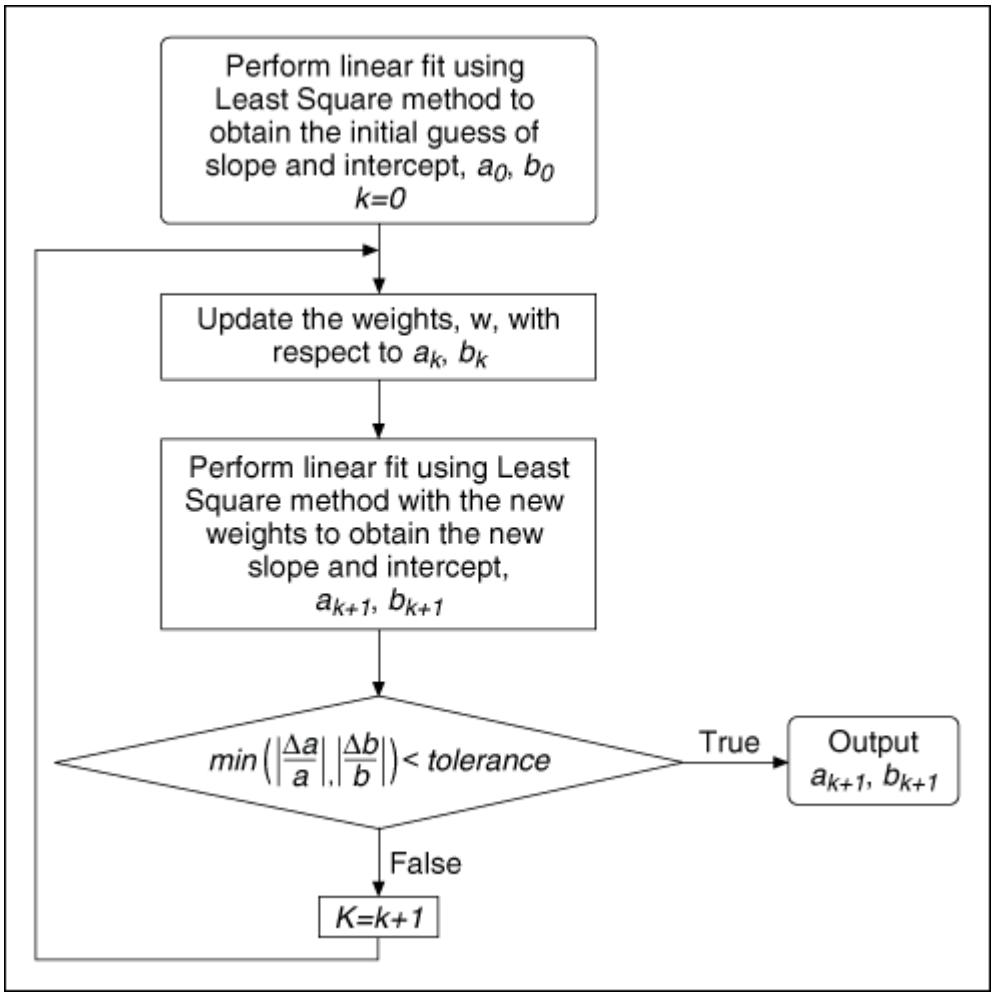


Figure 5.1: Flowchart diagram - Linear Regression

5.2.2 Polynomial Regression

The design for the Polynomial Regression (PR) algorithm is described on the flowchart diagram in figure 5.2. The block segment presented in this diagram is an accurate implementation of the algorithm behaviour in the software simulation. The Polynomial Regression (PL) algorithm is very similar to the Linear Regression (LR) algorithm with the only difference that Polynomial Regression (PL) gets the Polynomial Coefficients using the Least Square method instead. In the development of this software, using Polynomial Regression algorithm for stock prediction, plenty of data in the past is required, therefore the Least Square method for the polynomial fit is used to find the Polynomial Coefficients by minimizing residue for even closer prediction results. The weight referred to in the diagram is the array for (X, Y) observations. The general formula for Polynomial Regression can be found and explained in section 3.3.2

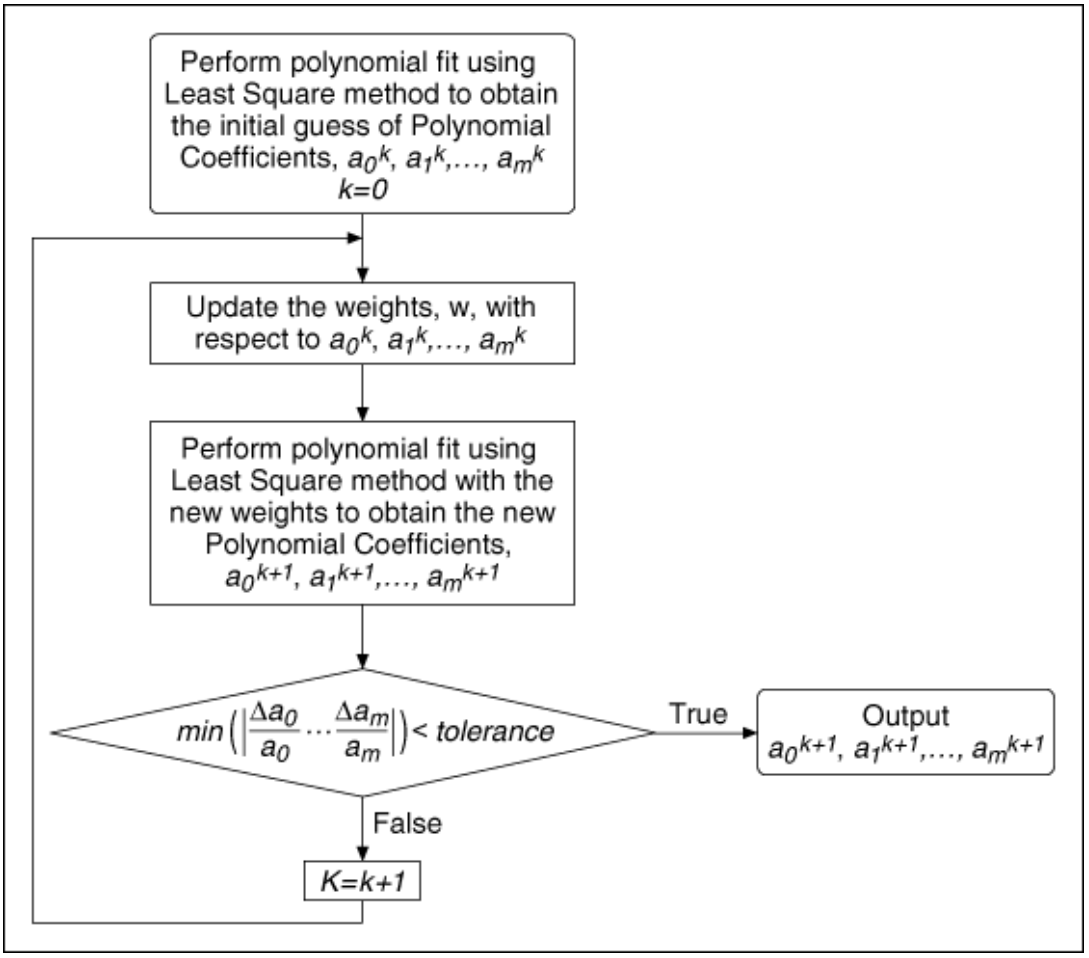


Figure 5.2: Flowchart diagram - Polynomial Regression

5.2.3 Random Forest Regression

The design for the Random Forest Regression (RF) algorithm is described on the flowchart diagram in figure 5.3. The block segment presented in this diagram is an accurate implementation of the algorithm behaviour in the software simulation. The RF model takes input vectors from the given dataset represented as (n observations) while predictions obtained shows as (m predictors). A number of regression trees is created being fitted with different training data as “k trees”, resamplying randomly the original values from the dataset to each regression tree created, for this reason majority of the data is used many times for each tree. However, based on the data fitted, each regression tree performs single predictions shown as “Pred 1” “Pred 2”, “Pred k” subsequently every prediction is then average for a unique estimation.

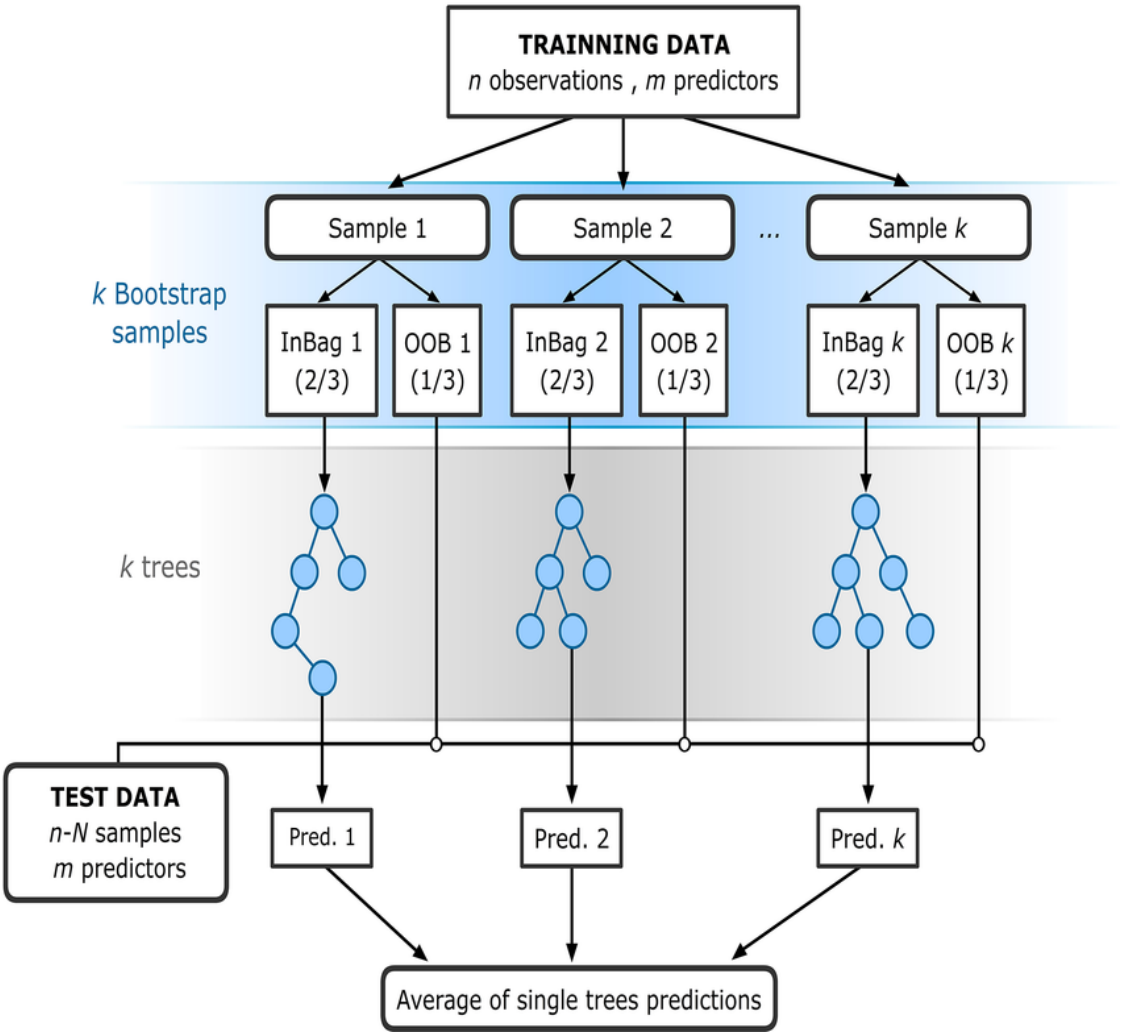


Figure 5.3: Flowchart diagram - Random Forest Regression

5.2.4 Long Short-Term Memory Neural Network

The design for the Long Short-Term Memory Neural Network (LSTM) algorithm is described on the flowchart diagram in figure 5.4. The block segment presented in this diagram is an accurate implementation of the algorithm behaviour in a software simulation. The decisions made by this model are based on its three different inputs: the current input, previous output and previous memory shown in the figure 5.4 legend. These inputs are all used in the node calculations, the result of the calculations are used not only to provide an output but also update the memory. The gate parameters control the flow of information within the node, in particular how much the safe state information is used as an input to the calculation. These gate parameters are weights and biases which means the behaviour depends on the inputs

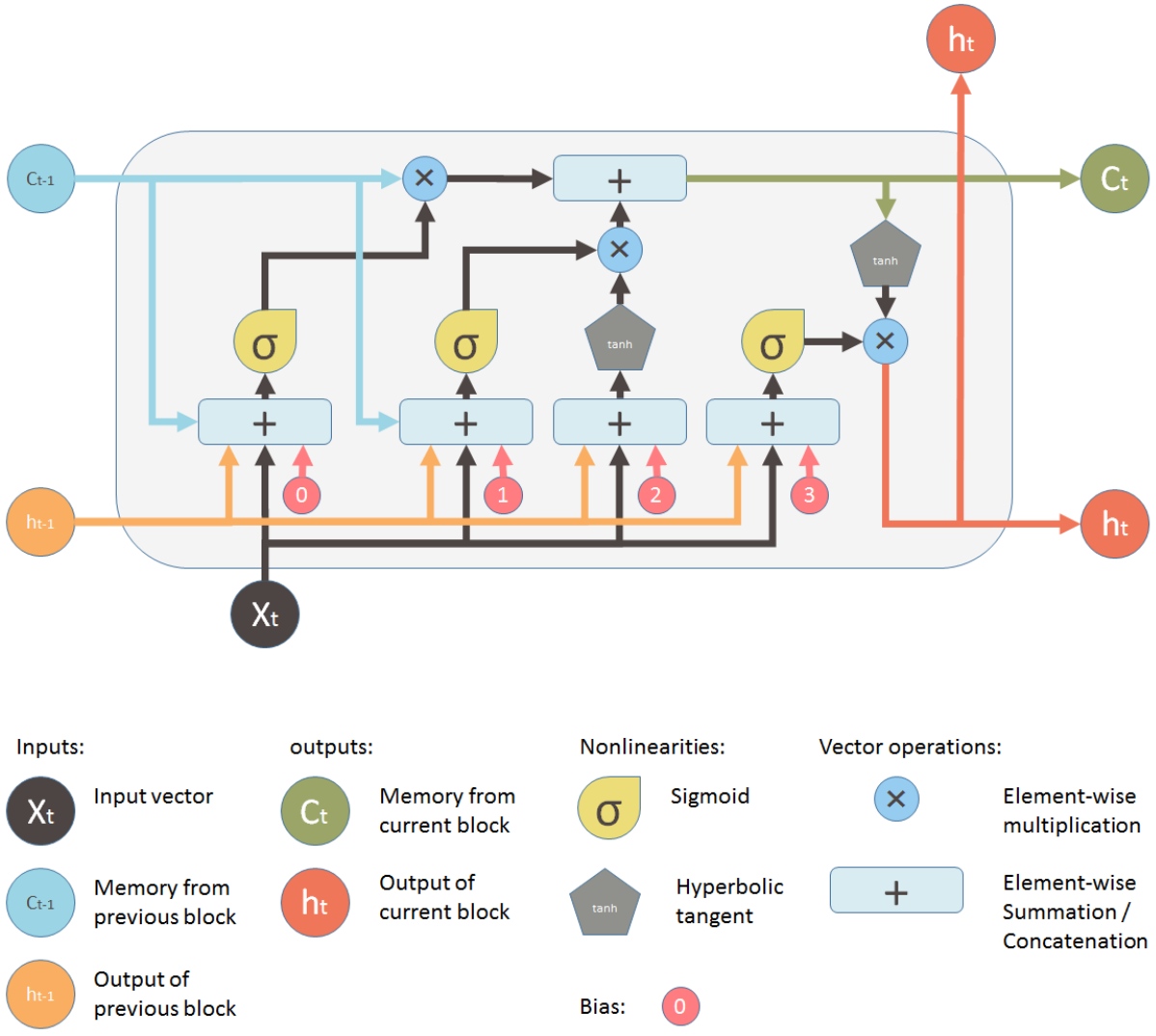


Figure 5.4: Flowchart diagram - Long Short-Term Memory Neural Network

5.3 Software Artefact

This section introduces the structure of the program in terms of different modules and how they work together without making reference to the source code. It formalises in a visual format how the features of the input, processing and output modules of the program communicate with each other for each of the chosen machine learning algorithms. Each module is presented using a UML diagram and a relevant explanation is given on how each segment within the diagram operates. It is important be able to test and compare the four different machine learning algorithms fairly for different scenarios

5.3.1 Input Module Design

The design of the input module is described on the UML diagram in figure 5.5. The user presented in this diagram is an actual investor who uses the software. Depending upon their individual preferences with regards to the accuracy and efficiency of the stock prediction and the desired compromise between these two conflicting objectives, the investor is able to specify the number of training iterations and number of days in the past used in the simulation of the machine learning algorithms.

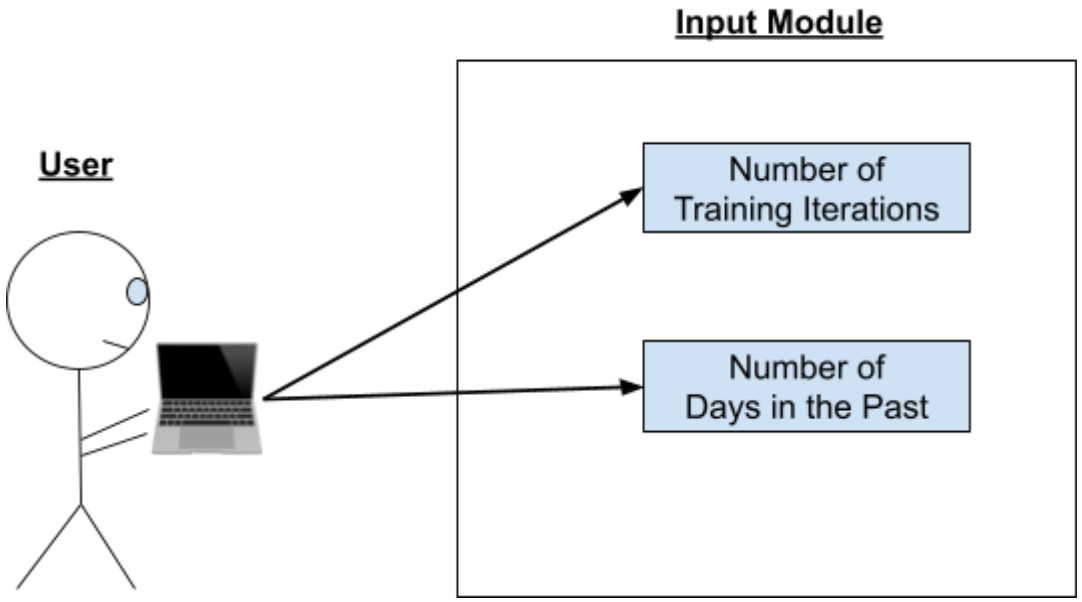


Figure 5.5: UML diagram - Input Module Design

5.3.2 Processing Module Design

The design of the processing module is described on the UML diagram in figure 5.6. The user is not involved in this module therefore is not presented in the diagram as they do not communicate with the back-end side of the process. This module is the interface between the input and output modules that performs analytical calculations as decision support for investors. This decision support is based on the demonstration of the computational accuracy and efficiency of the chosen machine learning algorithms. Depending upon different input choices from the user made in the input module, the time taken for processing can vary to achieve a good compromise between the accuracy and efficiency of the generated output. Dotted lines correspond to any computations that the back-end performs.

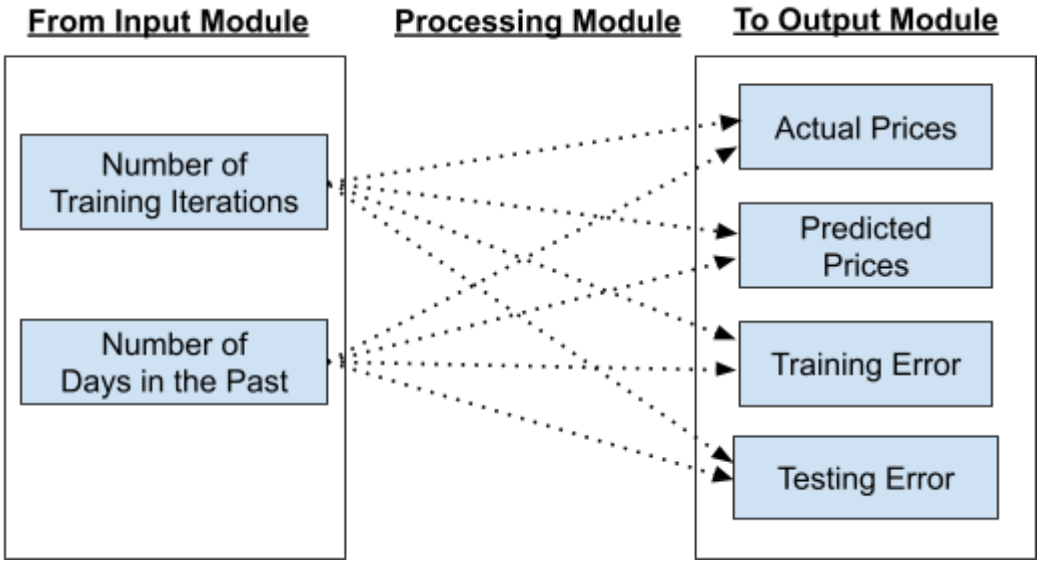


Figure 5.6: UML diagram - Processing Module Design

5.3.3 Output Module Design

The design of the output module is described on the UML diagram in figure 5.7. The user presented in this diagram is an investor waiting for the processing module to be completed in order to obtain a result. Once the analytical values from the processing module have been derived, the output of the operations is presented graphically and simultaneously for the all chosen machine learning algorithms which makes these results easier to analyse. The actual stock prices are taken from the available dataset and the predicted stock prices are calculated by these algorithms. In addition, the training and testing errors of the algorithms are presented in terms of Root Mean Squared Error.

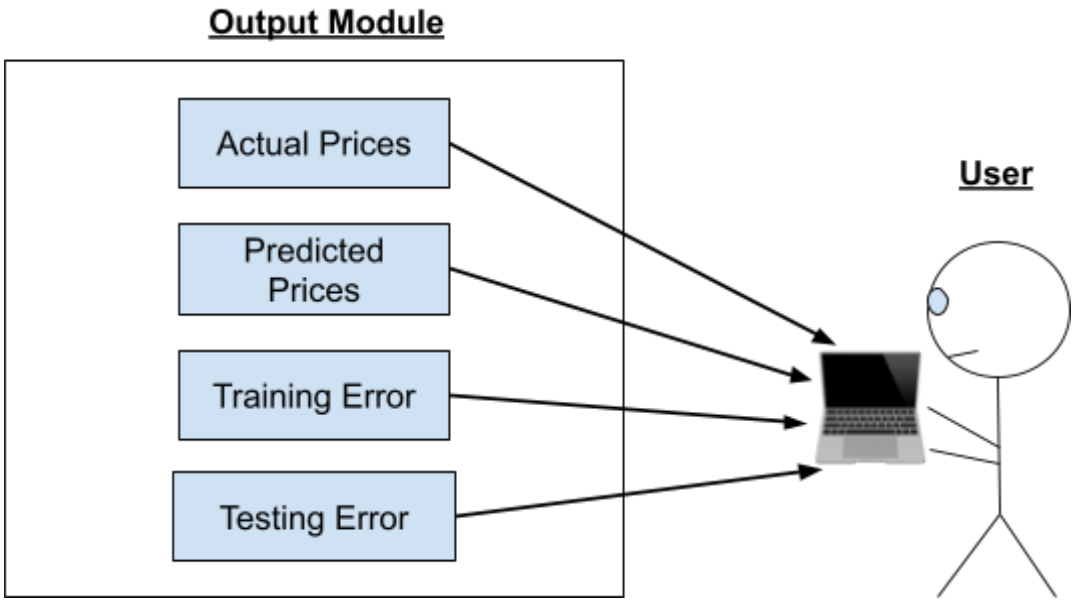


Figure 5.7: UML diagram - Output Module Design

5.4 Summary

This chapter has looked at the description of machine learning algorithms used in terms of flowcharts in section 5.2 and the associated software modules for their application in terms of UML diagrams in section 5.3. This description has been done in a programming language independent design context.

The following chapter (6), focuses on the programming based implementation of the design modules introduced in this chapter.

Chapter 6
Implementation

6.1 Introduction

Based on the description designs of the different machine learning algorithms chosen and software modules behaviours presented in chapter 5, the implementation of the software can be carried out in this chapter. Using the historical daily prices from the NASDAQ-traded stock market, the multinational technology enterprise Amazon has been chosen for the implementation of its dataset within the software development of results. In section 6.2, results of four scenarios are presented graphically for each machine learning algorithm, one of the scenarios being the benchmark approach and the other scenarios being novel approaches. These novel approaches are expected to improve the accuracy of the algorithms and have some impact on their efficiency. Section 6.3 presents a clear explanation of code segments developed within the software.

6.2 Research Results

This section presents screenshots of the results for each of the machine learning algorithms chosen in terms of actual and predicted stock values for different combinations of numbers of training iterations and days in the past as well as a table with partial data points for each algorithm. Sections 6.2.1, 6.2.2, 6.2.3 and 6.2.4 present the above results for Linear Regression (LR), Polynomial Regression (PR), Random Forest Regression (RF) and Long Short-Term Memory Neural Networks (LSTM), respectively. The benchmark scenario, based on existing research, is implemented with 10 training iterations and 50 days in the past. The three novel scenarios, 10 iterations and 100 days, 20 iterations and 50 days, and 20 iterations and 100 days, are also implemented as part of this research project.

6.2.1 Linear Regression

This algorithm works very well for all scenarios based on the closeness of the two curves on the corresponding screenshots that represent actual and predicted stock prices. The rough visual observation of the screenshots suggests that the variation of the number of training iterations and days in the past does not have a substantial impact on the very good prediction accuracy of the algorithm. The general conclusion from the above observation is confirmed analytically on the corresponding tables for the first five days from the dataset used. The orange curve on the graph represents the prediction values and the blue curve shows the actual values from the dataset even though this curve is not very visible.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

	Actual	LinearRegression_Pred
0	0.100093	0.100123
1	0.099180	0.100238
2	0.099143	0.098866
3	0.098364	0.099967
4	0.098235	0.099017

Table 6.1: Actual price vs Linear Regression Prediction

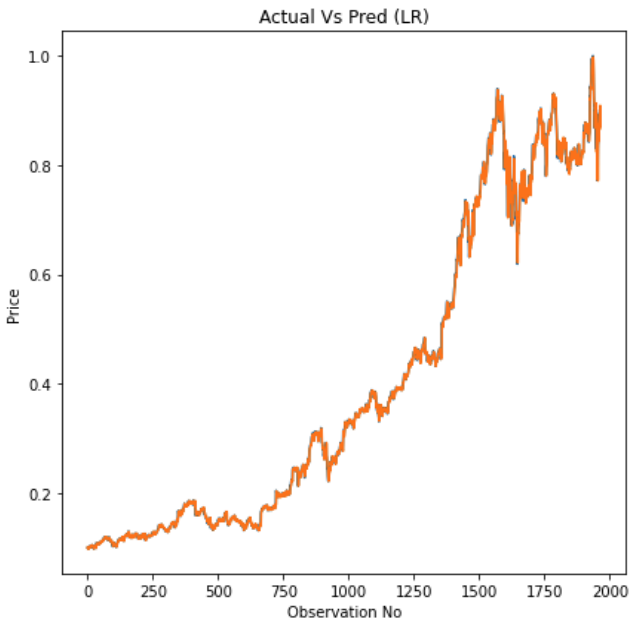


Figure 6.1: Actual price vs Linear Regression Prediction

Novel, first time application scenarios:

Run 2: 10 training iterations and 100 days in the past.

	Actual	LinearRegression_Pred
0	0.110177	0.110112
1	0.109762	0.110109
2	0.111445	0.109497
3	0.110569	0.110922
4	0.112662	0.111000

Table 6.2: Actual price vs Linear Regression Prediction

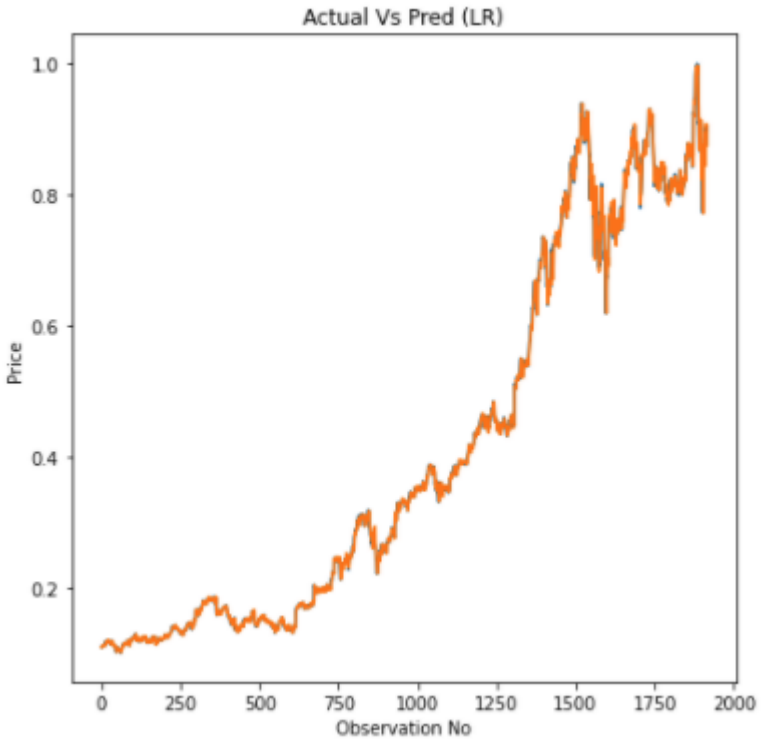


Figure 6.2: Actual price vs Linear Regression Prediction

Run 3: 20 training iterations and 50 days in the past.

	Actual	LinearRegression_Pred
0	0.100093	0.100123
1	0.099180	0.100238
2	0.099143	0.098866
3	0.098364	0.099967
4	0.098235	0.099017

Table 6.3: Actual price vs Linear Regression Prediction

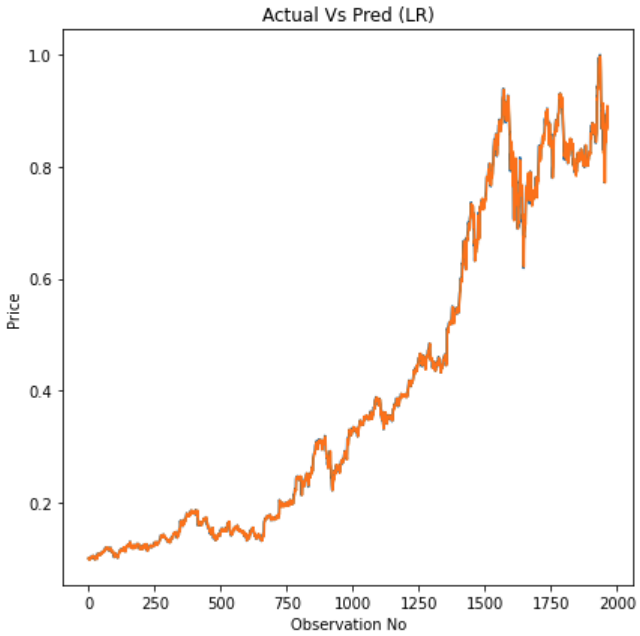


Figure 6.3: Actual price vs Linear Regression Prediction

Run 4: 20 training iterations and 100 days in the past.

	Actual	LinearRegression_Pred
0	0.110177	0.110112
1	0.109762	0.110109
2	0.111445	0.109497
3	0.110569	0.110922
4	0.112662	0.111000

Table 6.4: Actual price vs Linear Regression Prediction

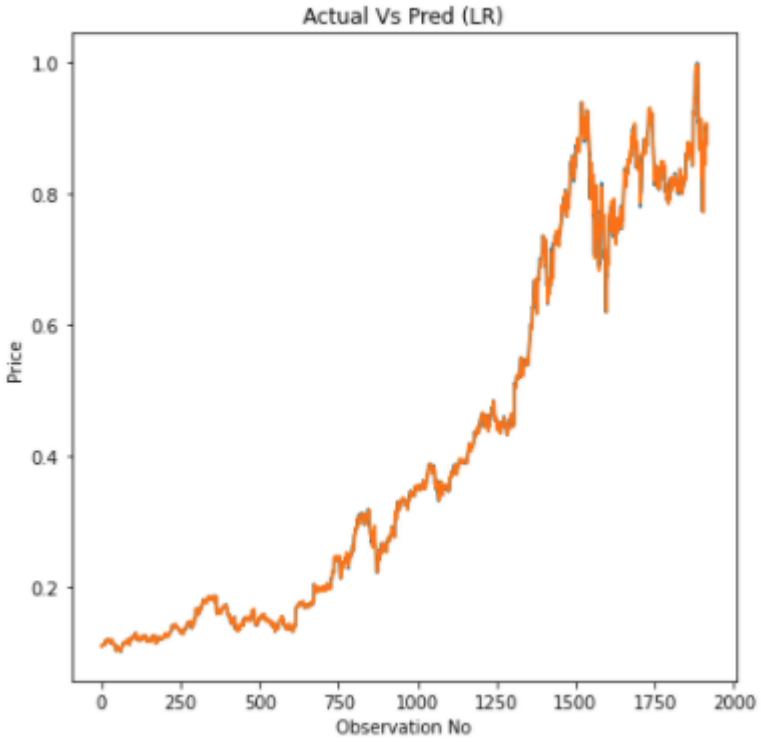


Figure 6.4: Actual price vs Linear Regression Prediction

6.2.2 Polynomial Regression

This algorithm does not work very well for all scenarios based on the closeness of the two curves on the corresponding screenshots that represent actual and predicted stock prices. The rough visual observation of the screenshots suggests that the variation of the number of training iterations and days in the past does not have a substantial impact on the poor prediction accuracy of the algorithm. However, the general conclusion from the above observation is not confirmed analytically on the corresponding tables for the first five days from the dataset used. The orange curve on the graph represents the prediction values and the blue curve shows the actual values from the dataset even though this curve is not visible.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

	Actual	Polynomial_Pred
0	0.100093	0.100540
1	0.099180	0.100277
2	0.099143	0.103137
3	0.098364	0.107803
4	0.098235	0.099681

Table 6.5: Actual price vs Polynomial Regression Prediction

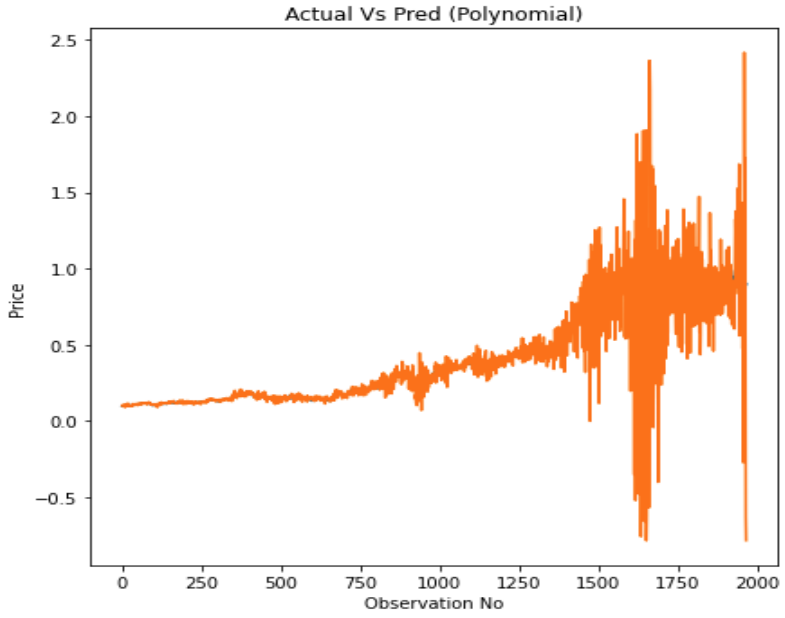


Figure 6.5: Actual price vs Polynomial Regression Prediction

Novel, first time application scenarios:

Run 2: 10 training iterations and 100 days in the past.

	Actual	Polynomial_Pred
0	0.110177	0.085887
1	0.109762	0.196852
2	0.111445	0.086760
3	0.110569	0.121058
4	0.112662	0.130694

Table 6.6: Actual price vs Polynomial Regression Prediction

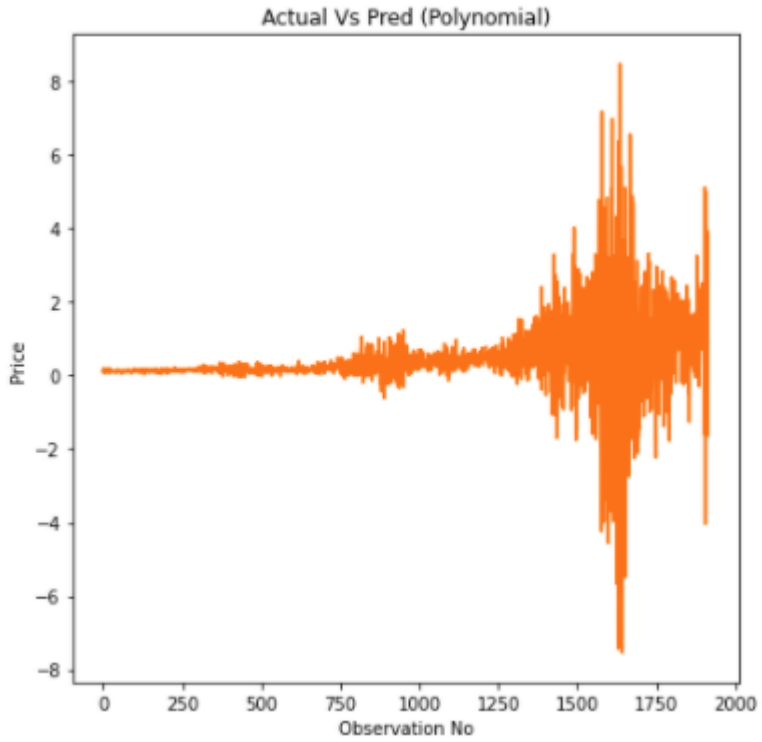


Figure 6.6: Actual price vs Polynomial Regression Prediction

Run 3: 20 training iterations and 50 days in the past.

	Actual	Polynomial_Pred
0	0.100093	0.100540
1	0.099180	0.100277
2	0.099143	0.103137
3	0.098364	0.107803
4	0.098235	0.099681

Table 6.7: Actual price vs Polynomial Regression Prediction

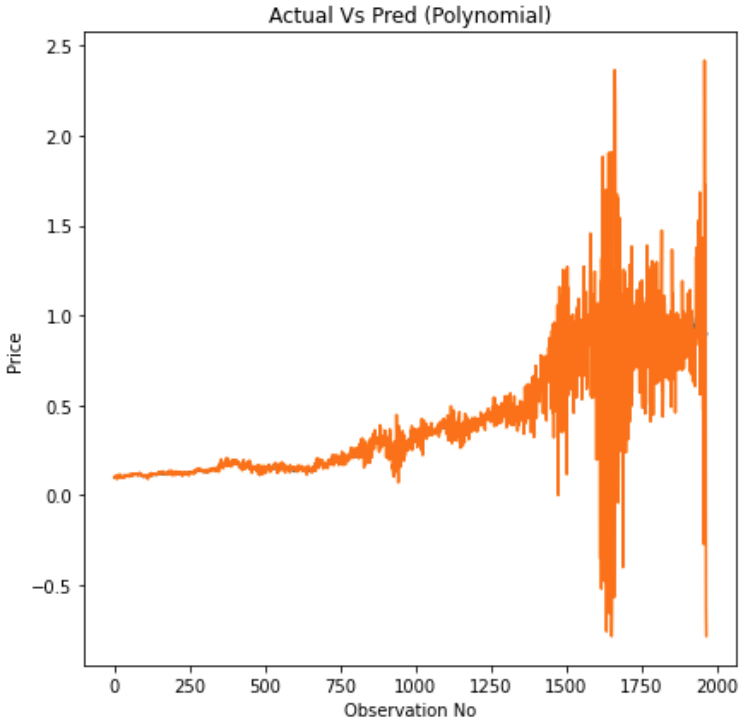


Figure 6.7: Actual price vs Polynomial Regression Prediction

Run 4: 20 training iterations and 100 days in the past.

	Actual	Polynomial_Pred
0	0.110177	0.085887
1	0.109762	0.196852
2	0.111445	0.086760
3	0.110569	0.121058
4	0.112662	0.130694

Table 6.8: Actual price vs Polynomial Regression Prediction

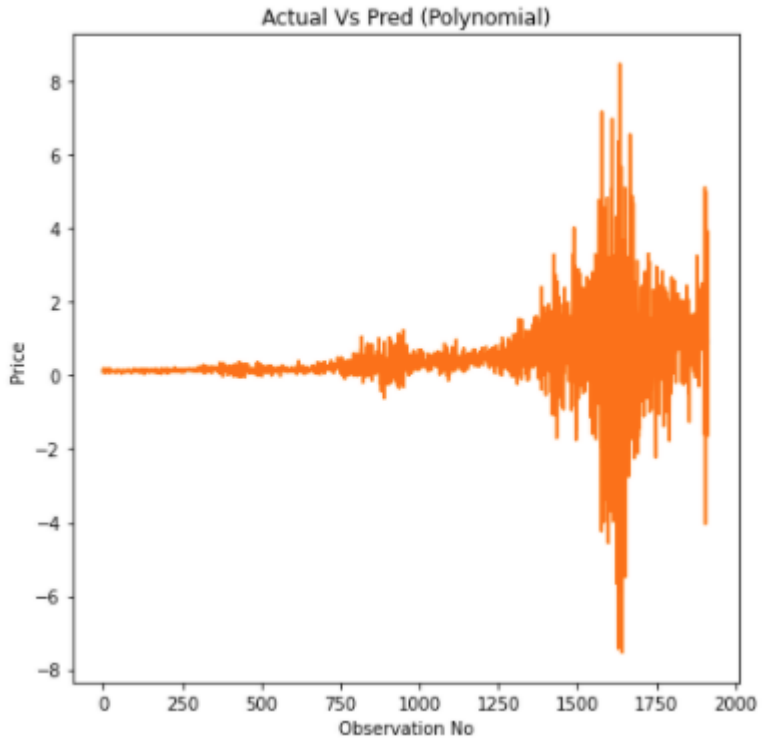


Figure 6.8: Actual price vs Polynomial Regression Prediction

6.2.3 Random Forest Regression

This algorithm does not work well for all scenarios based on the closeness of the two curves on the corresponding screenshots that represent actual and predicted stock prices. The rough visual observation of the screenshots suggests that the variation of the number of training iterations and days in the past does not have a substantial impact on the very poor prediction accuracy of the algorithm. However, the general conclusion from the above observation is not confirmed analytically on the corresponding tables for the first five days from the dataset used. The orange curve on the graph represents the prediction values and the blue curve shows the actual values from the dataset.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

	Actual	RandomForest_Pred
0	0.100093	0.096559
1	0.099180	0.096710
2	0.099143	0.096362
3	0.098364	0.096820
4	0.098235	0.096314

Table 6.9: Actual price vs Random Forest Prediction

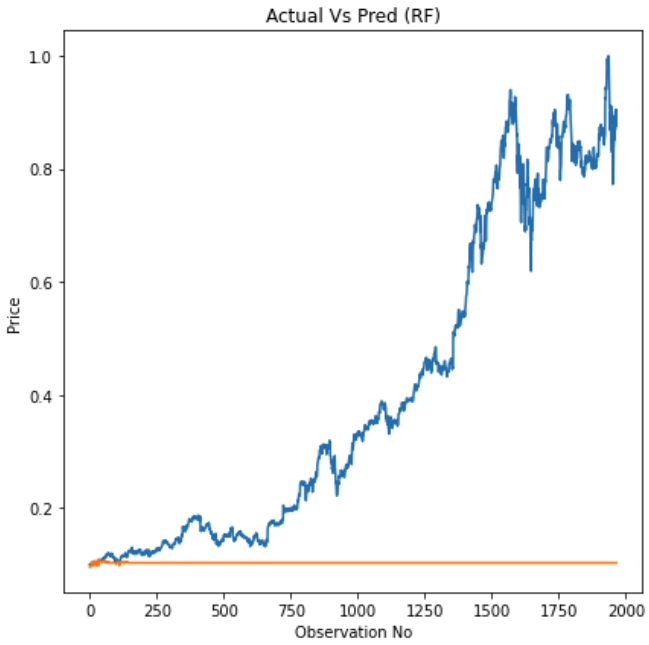


Figure 6.9: Actual price vs Random Forest Prediction

Novel, first time application scenarios:

Run 2: 10 training iterations and 100 days in the past.

	Actual	RandomForest_Pred
0	0.110177	0.105576
1	0.109762	0.105373
2	0.111445	0.105131
3	0.110569	0.105028
4	0.112662	0.105820

Table 6.10: Actual price vs Random Forest Prediction

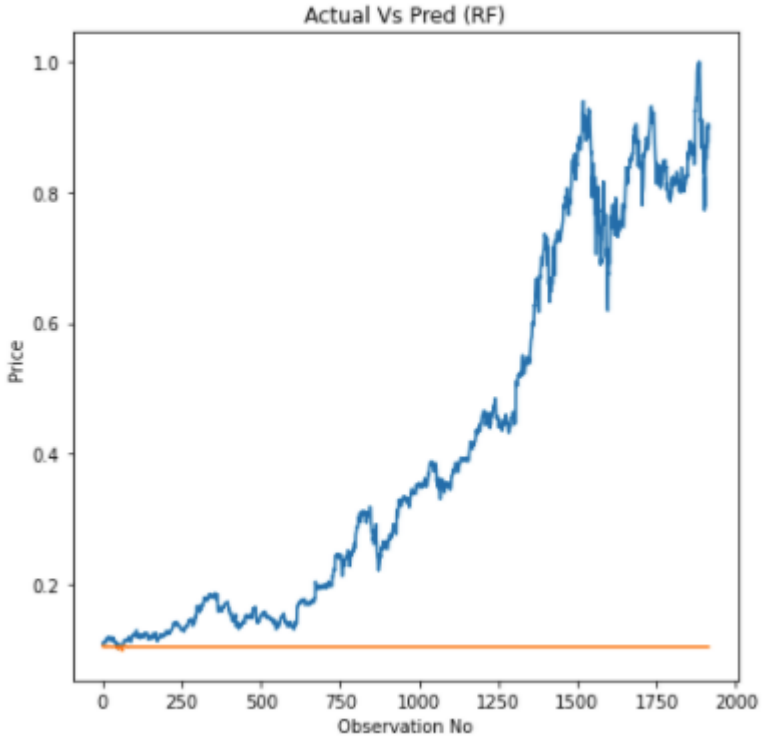


Figure 6.10: Actual price vs Random Forest Prediction

Run 3: 20 training iterations and 50 days in the past.

	Actual	RandomForest_Pred
0	0.100093	0.096559
1	0.099180	0.096710
2	0.099143	0.096362
3	0.098364	0.096820
4	0.098235	0.096314

Table 6.11: Actual price vs Random Forest Prediction

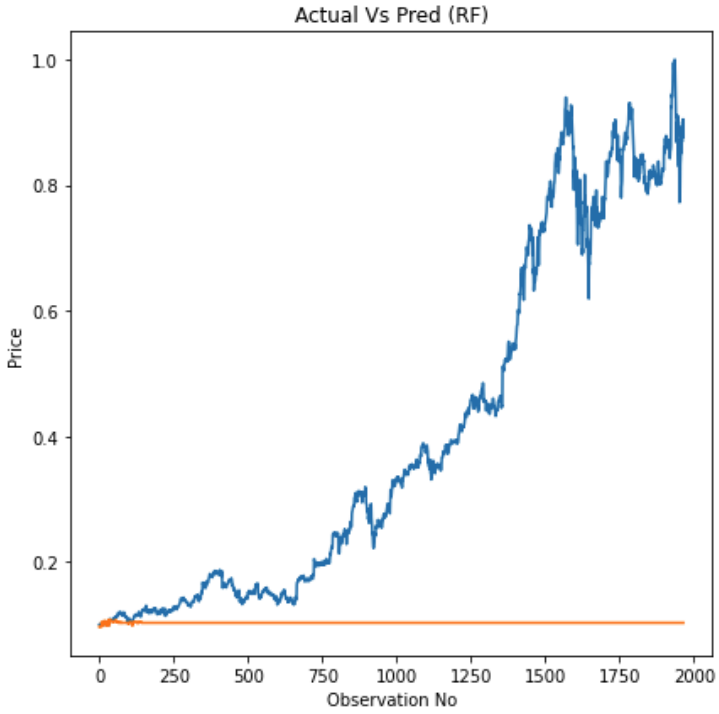


Figure 6.11: Actual price vs Random Forest Prediction

Run 4: 20 training iterations and 100 days in the past.

	Actual	RandomForest_Pred
0	0.110177	0.105576
1	0.109762	0.105373
2	0.111445	0.105131
3	0.110569	0.105028
4	0.112662	0.105820

Table 6.12: Actual price vs Random Forest Prediction

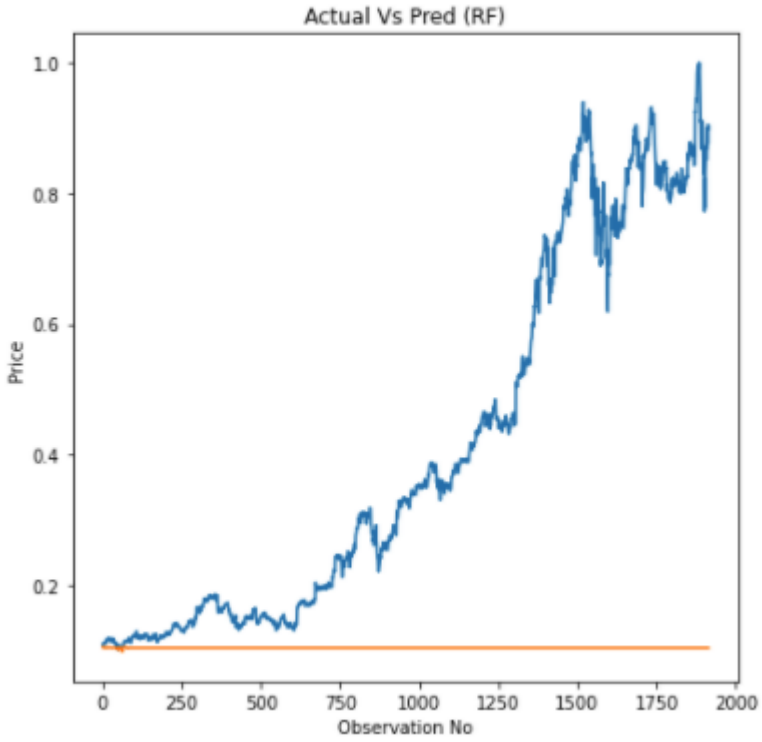


Figure 6.12: Actual price vs Random Forest Prediction

6.2.4 Long Short-Term Memory Neural Network

This algorithm works well for all scenarios based on the closeness of the two curves on the corresponding screenshots that represent actual and predicted stock prices. The rough visual observation of the screenshots suggests that the variation of the number of training iterations and days in the past do not have a substantial impact on the good prediction accuracy of the algorithm. The general conclusion from the above observation is confirmed analytically on the corresponding tables for the first five days from the dataset used. The orange curve on the graph represents the prediction values and the blue curve shows the actual values from the dataset.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

	Actual	LSTM_Pred
0	0.100093	0.094903
1	0.099180	0.094955
2	0.099143	0.095110
3	0.098364	0.095325
4	0.098235	0.095550

Table 6.13: Actual price vs LSTM Neural Network Prediction

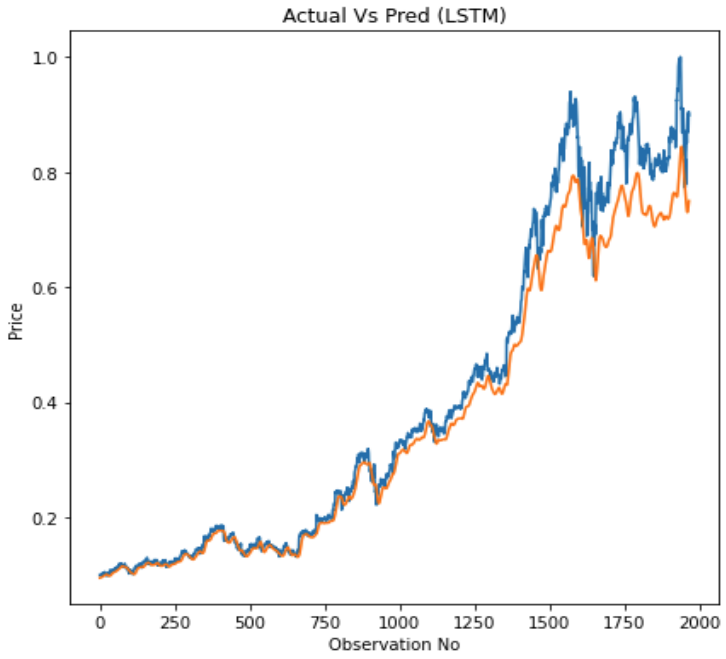


Figure 6.13: Actual price vs LSTM Neural Network Prediction

Novel, first time application scenarios:

Run 2: 10 training iterations and 100 days in the past.

	Actual	LSTM_Pred
0	0.110177	0.106206
1	0.109762	0.106494
2	0.111445	0.106794
3	0.110569	0.107120
4	0.112662	0.107453

Table 6.14: Actual price vs LSTM Neural Network Prediction

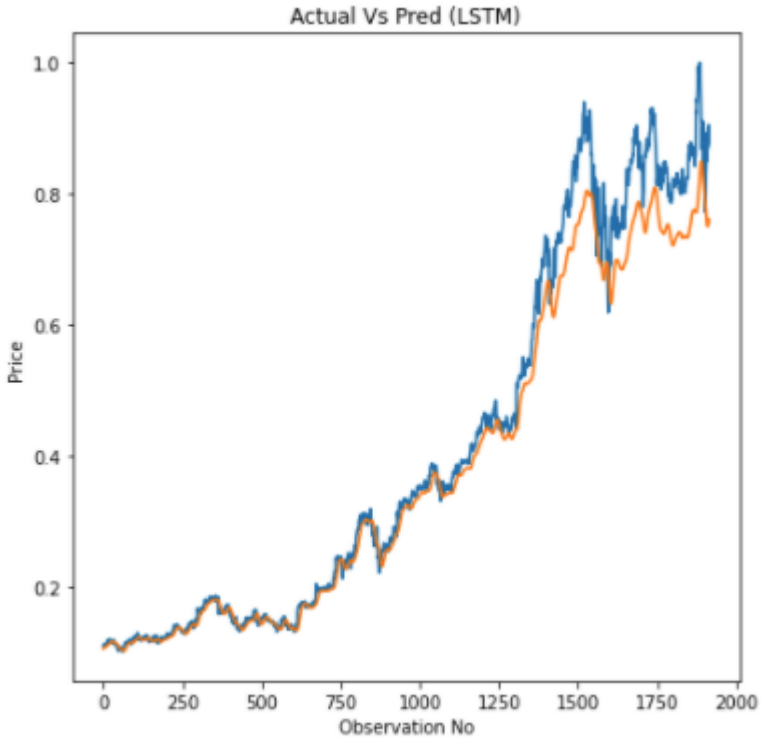


Figure 6.14: Actual price vs LSTM Neural Network Prediction

Run 3: 20 training iterations and 50 days in the past.

	Actual	LSTM_Pred
0	0.100093	0.094903
1	0.099180	0.094955
2	0.099143	0.095110
3	0.098364	0.095325
4	0.098235	0.095550

Table 6.15: Actual price vs LSTM Neural Network Prediction

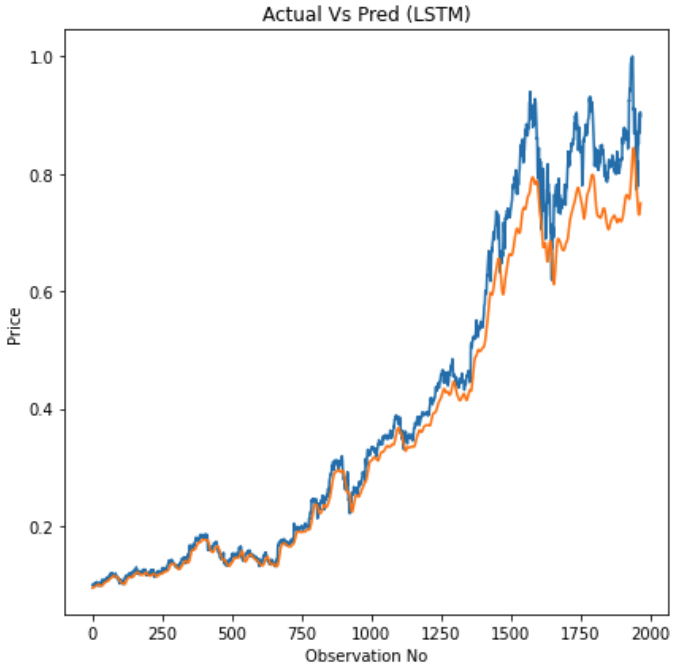


Figure 6.15: Actual price vs LSTM Neural Network Prediction

Run 4: 20 training iterations and 100 days in the past.

	Actual	LSTM_Pred
0	0.110177	0.106206
1	0.109762	0.106494
2	0.111445	0.106794
3	0.110569	0.107120
4	0.112662	0.107453

Table 6.16: Actual price vs LSTM Neural Network Prediction

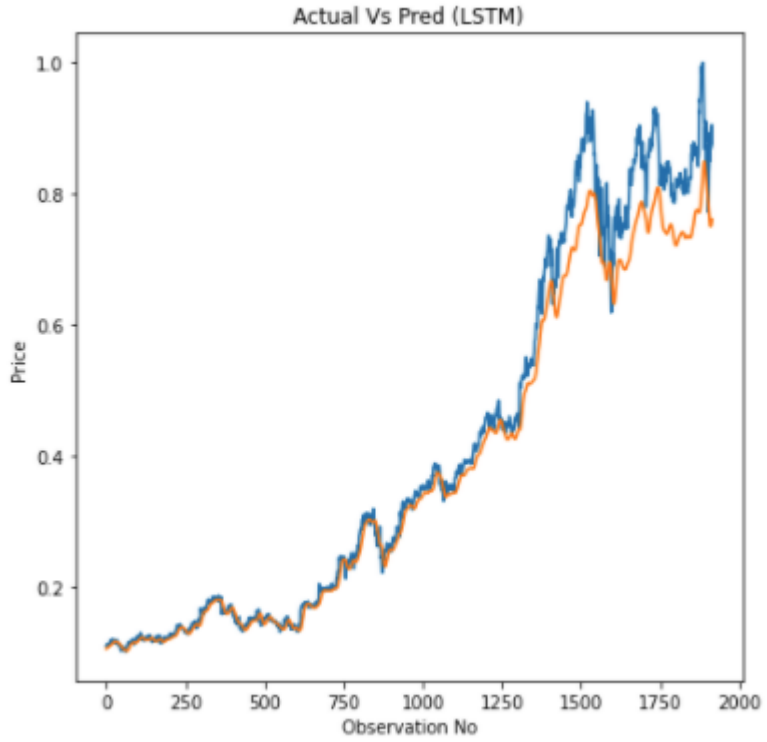


Figure 6.16: Actual price vs LSTM Neural Network Prediction

6.3 Software Artefact

Apart from all the imported libraries used in the implementation of the software artefact, all other segments of code are written by the author of this report. These imported libraries include: "*numpy", "pandas", "seaborn", "sklearn.preprocessing", "MinMaxScaler", "matplotlib.pyplot", "tensorflow.compat.v1", "math"* and *"mean_squared_error from sklearn.metrics"*. All of these libraries are used for mathematical calculations, data analysis, tables and graphical representations. However, specific libraries were used for the implementation of each algorithm. For the Linear Regression implementation: *"LinearRegression"* from "*sklearn.linear_model"*, "*statsmodels.api", "dmatrices from patsy"* for the coefficients of the linear regression model, for the Polynomial Regression implementation: *"PolynomialFeatures from sklearn.preprocessing"* and *"r2_score from sklearn.metrics"*, for the Random Forest Regression implementation: *"RandomForestRegressor from sklearn.ensemble"* and finally for the LSTM Neural Network implementation: *"Sequential from tensorflow.keras.models", "Dense from tensorflow.keras.layers"* and *"LSTM from tensorflow.keras.layers"* were used. Some of these libraries are visible in the following screenshots for each algorithm by the *'import'* and *'from'* commands, also an explanation of these screenshots is presented. The complete source code can be found in the appendix C.

Linear Regression

As shown in figure 6.17, the implementation of the Linear Regression algorithm starts by first importing the required library *"LinearRegression"* in order to create the coefficients of the algorithm. The second segment of code holds a call to the actual algorithm imported previously into a variable *"lr"*. The "*X*" and "*Y*" training size values are calculated for the creation of the model, the number of days in the past imputed is also used. Similarly, the "*X*" and "*Y*" testing size values are calculated for the creation of the model, the number of days in the past imputed is also used. Finally, the *"X"* and *"Y"* values for only training are fitted into the model and a call of the *"LinearRegression()"* is made for the creation and training of the algorithm.

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
print(X_train.shape)
lr_X_train = X_train.reshape(X_train_size, time_step)
print(lr_X_train.shape)
```

```
(3641, 100, 1)
(3641, 100)
```

```
print(y_train.shape)
lr_y_train = y_train.reshape(y_train_size)
print(lr_y_train.shape)
```

```
(3641,)
(3641,)
```

```
print(X_test.shape)
lr_X_test = X_test.reshape(X_test_size, time_step)
print(lr_X_test.shape)
```

```
(1915, 100, 1)
(1915, 100)
```

```
print(ytest.shape)
lr_y_test = ytest.reshape(y_test_size)
print(lr_y_test.shape)
```

```
(1915,)
(1915,)
```

```

```

```
lr.fit(lr_X_train,lr_y_train)
```

```
LinearRegression()
```

Figure 6.17: Linear Regression code segments

Polynomial Regression

As shown in figure 6.18, the implementation of the Polynomial Regression algorithm starts by first importing the required libraries in order to perform calculations and create the algorithm. The second segment, *"PolynomialFeatures"* requires a degree=2 by default to create a matrix of values and to be fitted afterwards. The *"LinearRegression()"* function is saved in the variable *"p_lr"* representing the polynomial regression, then the model passes the values required for the creation of the polynomial algorithm in the parameters. The following segments show how the *"X_test_TRANSF"* , *"p_lr_test_predict"* , *"p_lr_train_predict"* variables hold specific values to be used for the table representation of results. Finally, the table is created with three different columns, the "Actual" values of the dataset, the "predicted" values of the algorithm and the difference in "error" values. The table shows the first five values by default using the *".head()"* command.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

```
nb_degree = 2

polynomial_features = PolynomialFeatures(degree = nb_degree)

X_TRANSF = polynomial_features.fit_transform(lr_X_train)
```

```
p_lr = LinearRegression()

p_lr.fit(X_TRANSF, lr_y_train)
```

```
LinearRegression()
```

```
X_test_TRANSF = polynomial_features.fit_transform(lr_X_test)
p_lr_test_predict = p_lr.predict(X_test_TRANSF)
p_lr_train_predict = p_lr.predict(X_TRANSF)
```

```
p_lr_compare = pd.concat([pd.DataFrame(lr_y_test, columns=['Actual']),
                          pd.DataFrame(p_lr_test_predict, columns=['Polynomial_Pred'])],axis=1)
p_lr_compare['error'] = p_lr_compare['Actual']-p_lr_compare['Polynomial_Pred']

p_lr_compare.head()
```

Figure 6.18: Polynomial Regression code segments

Random Forest Regression

As shown in figure 6.19, the implementation of the Random Forest Regression algorithm starts by first importing the required library in order to perform calculations and create the algorithm. The *"RandomForestRegressor"* model takes a number of trees *"n_estimators"* which is 100 by default, and *"random_state=0"* is required. *"regressor.fit"* creates a forest of trees from the training set for *"lr_X_train"* and *"lr_y_train"*. The variables *"rf_test_predict"* and *"rf_train_predict"* hold the required values in order to perform the prediction of the algorithm. Finally, the table is created with three different columns, the "Actual" values of the dataset, the "predicted" values of the algorithm and the difference in "error" values. The table shows the first five values by default using the *".head()"* command.

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=100, random_state=0)
regressor.fit(lr_X_train, lr_y_train)
rf_test_predict = regressor.predict(lr_X_test)
rf_train_predict = regressor.predict(lr_X_train)
```

```
rf_compare = pd.concat([pd.DataFrame(lr_y_test, columns=['Actual']),
                        pd.DataFrame(rf_test_predict, columns=['RandomForest_Pred'])],axis=1)
rf_compare['error'] = rf_compare['Actual']-rf_compare['RandomForest_Pred']
rf_compare.head()
```

Figure 6.19: Random Forest Regression code segments

LSTM Neural Network

As shown in figure 6.20, the implementation of the LSTM Neural Network algorithm starts by first importing the required libraries in order to perform calculations and create the stacked LSTM algorithm model. The model variable holds the *"Sequential()"* function. To stack the LSTM layers, it is required to set the *"return_sequences"* argument to "True" first in order to provide a 3D array and return one output for each input time step. The arguments "*loss"* and "*optimizer=Adam"* are required for compiling the model. Finally, the array is created with three different columns, having the first column with "Layer (type)" set to be "*lstm", "lstm_1", "lstm_2"* and *"dense",* the second with the output shape values and third with *"parameter"* values of the model.The table shows the 3D array results by using the *".summary()"* command.

```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(time_step,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.20: LSTM Neural Network code segments

6.4 Summary

This chapter has looked at the implementation of each of the machine learning algorithms chosen in terms of actual and predicted values graphically for benchmark and novel approaches in section 6.2. Review the implemented source code for each algorithm and how they work in section 6.3.

The following chapter (7), focuses on the testing and evaluation phase of the software implemented in this chapter.

Chapter 7
Testing & Evaluation

7.1 Introduction

Based on the implementation of the results for the chosen machine learning algorithms and their graphical presentation in the previous chapter, testing and evaluation are carried out in this chapter. The evaluation compares the novel approaches with the benchmark approach for every simulation run of each machine learning algorithm. The comparison is done using the Root Mean Square Error (RMSE) separately for the training and testing of the algorithms in terms of accuracy and the computational time for both training and testing in terms of efficiency. The presentation of the Root Mean Square Error (RMSE) for training and testing for the algorithms is shown in graphs and tables in section 7.2. The requirements met by the software artefact are detailed in section 7.3 in line with their specification in section 4.3.

7.2 Research Experiment Results

This section presents graphical results in terms of Root Mean Square Error (RMSE) using the dataset introduced in section 6.1. This dataset is split into training and testing sets whereby training is used to fit the machine learning algorithms to the training set and testing is used to validate the algorithms with the testing set. Representation of the training and testing graph curves is shown in section 7.2.1 for every simulation run for each algorithm. Comparative evaluation of these training and testing sets results in terms of accuracy and efficiency is presented in section 7.2.2 in terms of the proposed novel approaches and the benchmark approach for the machine learning algorithms.

7.2.1 Algorithms Training and Testing

This section shows graphically Root Mean Square Error (RMSE) for every simulation run for each algorithm, first on individual graphs and then integrated together on a single graph. A precise percentage for each set is used. For training the software uses 65% of the data to fit the algorithms and 35% of the data is used to test the algorithms. Each algorithm is described with four graphs, one for the benchmark approach and three for the novel approaches. According to the graphs, the orange curve refers to the training error while the green curve refers to the testing error. The actual values of the datasets used are presented in a blue curve, whose visibility varies in some graphs.

Linear Regression

This algorithm works very well for all scenarios based on the variation range (0,1) of the two parts of the curves on the corresponding screenshots that represent training and testing Root Mean Square Error (RMSE) for stock prices. The visual observation of the screenshots confirms the conclusion from the rough observation in section 6.2.1 that the number of training iterations and days in the past does not have a substantial impact on the very good prediction accuracy of the algorithm.

Benchmark scenario:
Run 1: 10 training iterations and 50 days in the past.

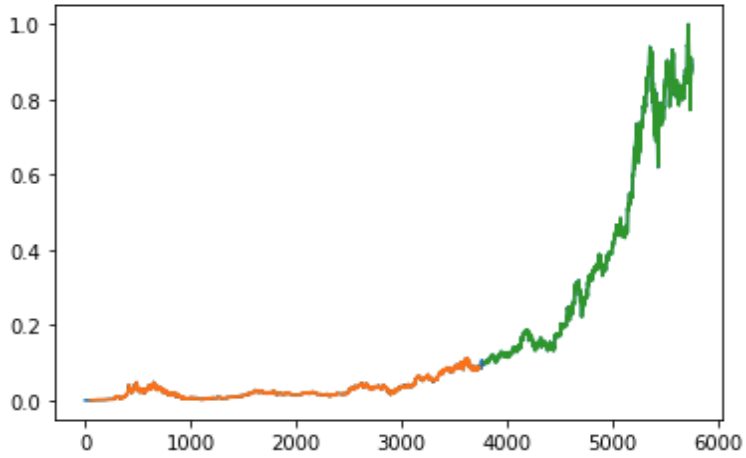


Figure 7.1: Training and Testing for Linear Regression

Novel, first time application scenarios:

Run 2: 10 training iterations and 100 days in the past.

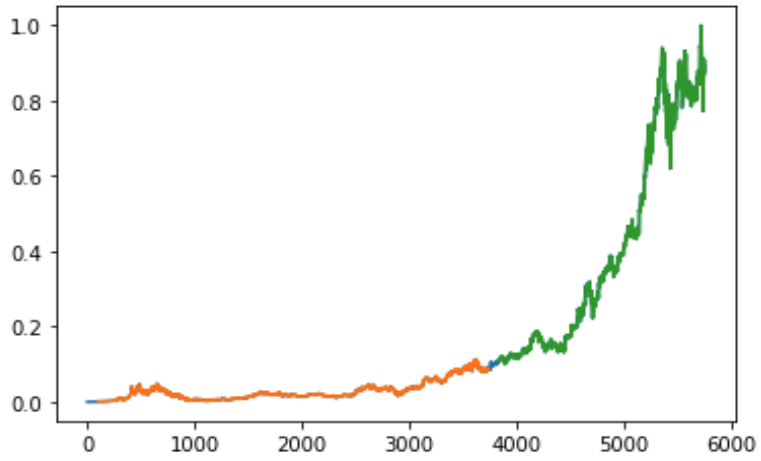


Figure 7.2: Training and Testing for Linear Regression

Run 3: 20 training iterations and 50 days in the past.

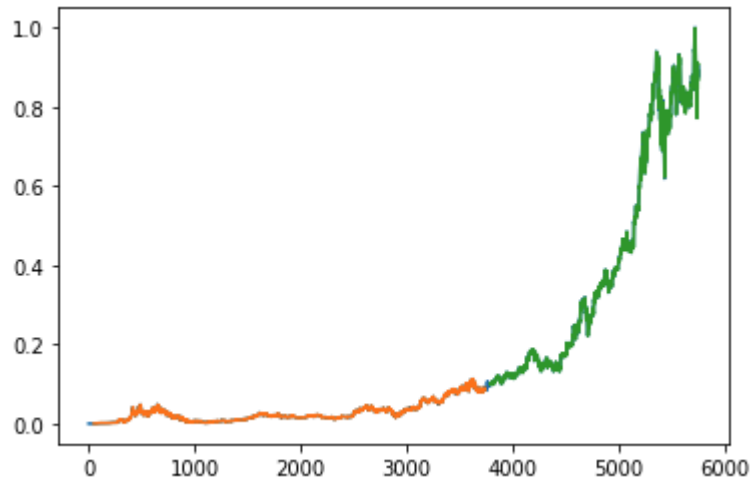


Figure 7.3: Training and Testing for Linear Regression

Run 4: 20 training iterations and 100 days in the past.

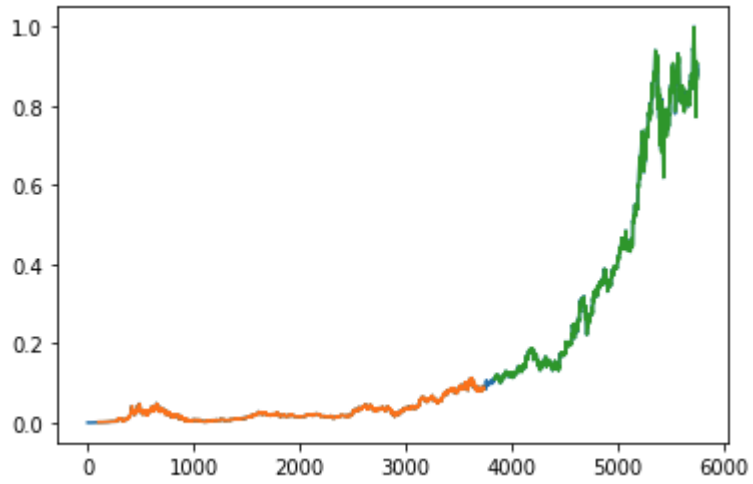


Figure 7.4: Training and Testing for Linear Regression

Polynomial Regression

This algorithm does not work very well for all scenarios based on the variation range (0,1) of the two parts of the curves on the corresponding screenshots that represent training and testing Root Mean Square Error (RMSE) for stock prices. The visual observation of the screenshots confirms the conclusion from the rough observation in section 6.2.2 that the number of training iterations and days in the past does not have a substantial impact on the poor prediction accuracy of the algorithm.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

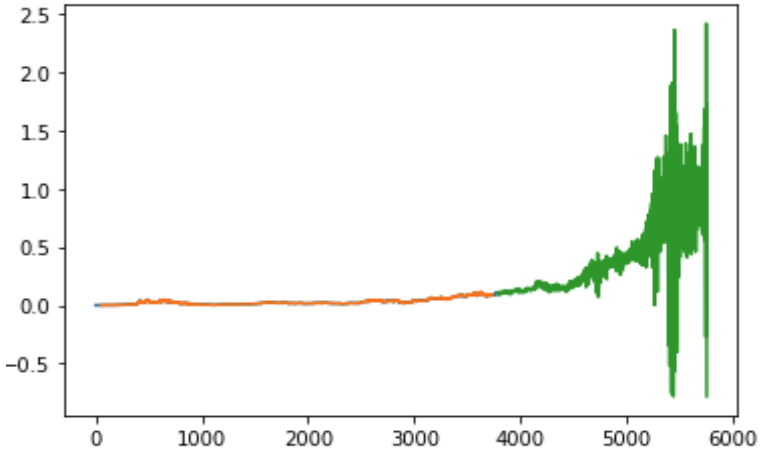


Figure 7.5: Training and Testing for Polynomial Regression

Novel, first time application scenarios

Run 2: 10 training iterations and 100 days in the past.

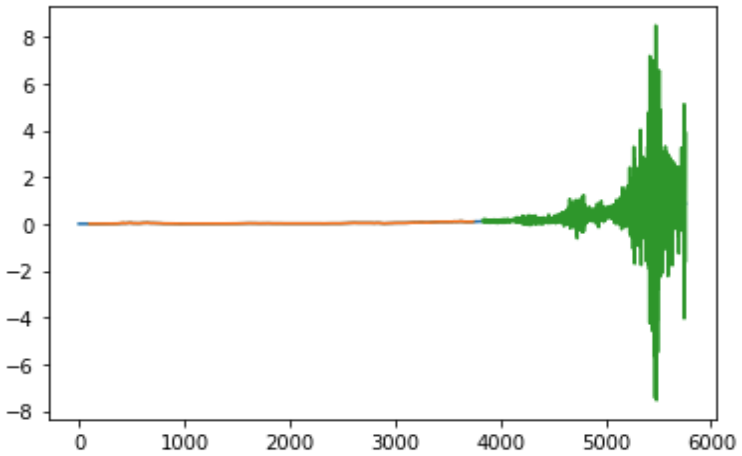


Figure 7.6: Training and Testing for Polynomial Regression

Run 3: 20 training iterations and 50 days in the past.

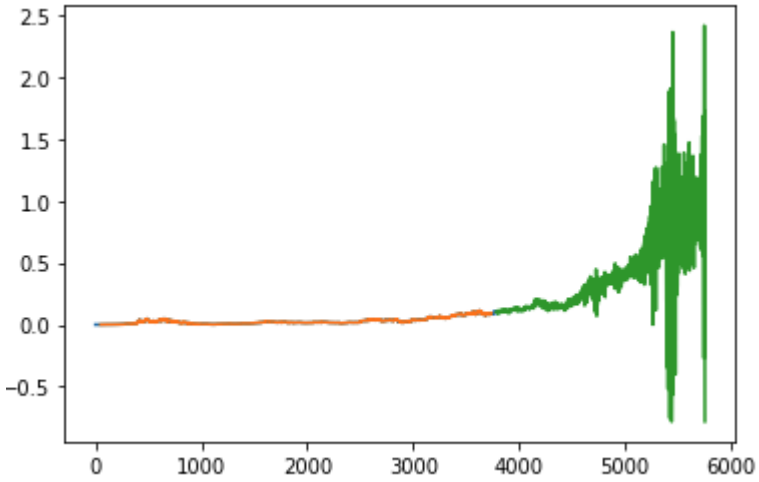


Figure 7.7: Training and Testing for Polynomial Regression

Run 4: 20 training iterations and 100 days in the past.

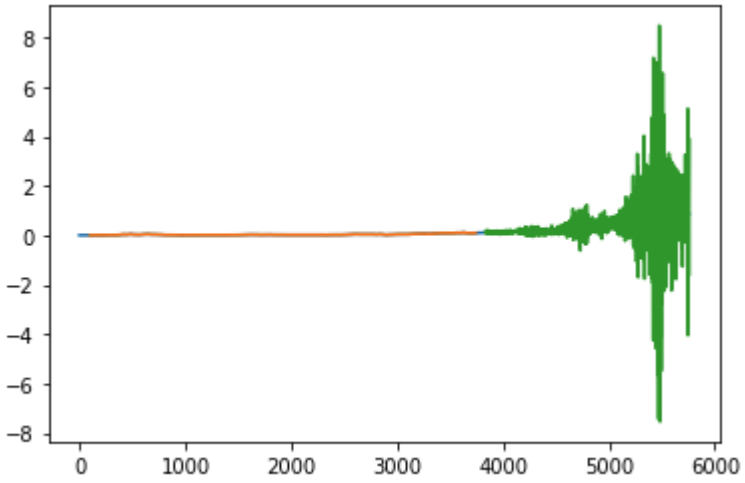


Figure 7.8: Training and Testing for Polynomial Regression

Random Forest Regression

This algorithm does not work well for all scenarios based on the variation range (0,1) of the two parts of the curves on the corresponding screenshots that represent training and testing Root Mean Square Error (RMSE) for stock prices. The visual observation of the screenshots confirms the conclusion from the rough observation in section 6.2.3 that the number of training iterations and days in the past does not have a substantial impact on the very poor prediction accuracy of the algorithm.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

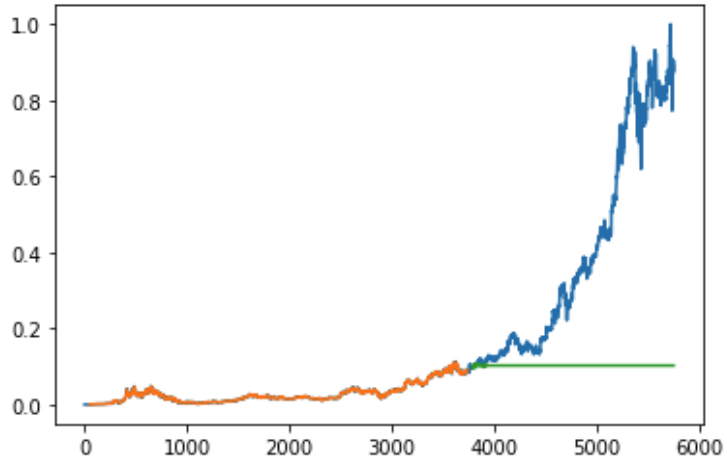


Figure 7.9: Training and Testing for Random Forest Regression

Novel, first time application scenarios

Run 2: 10 training iterations and 100 days in the past.

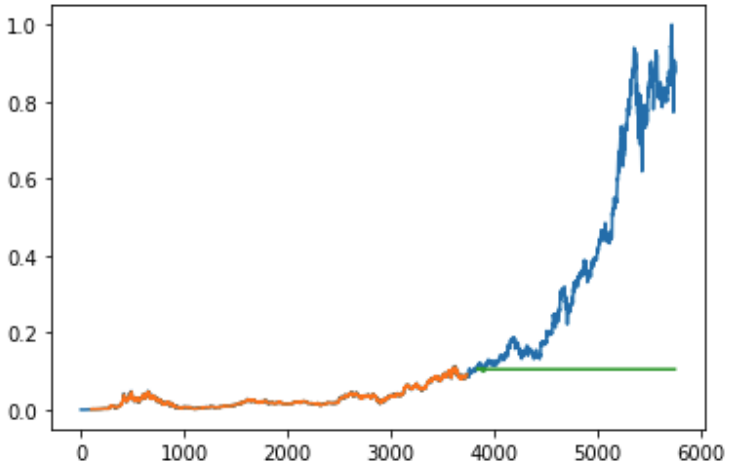


Figure 7.10: Training and Testing for Random Forest Regression

Run 3: 20 training iterations and 100 days in the past.

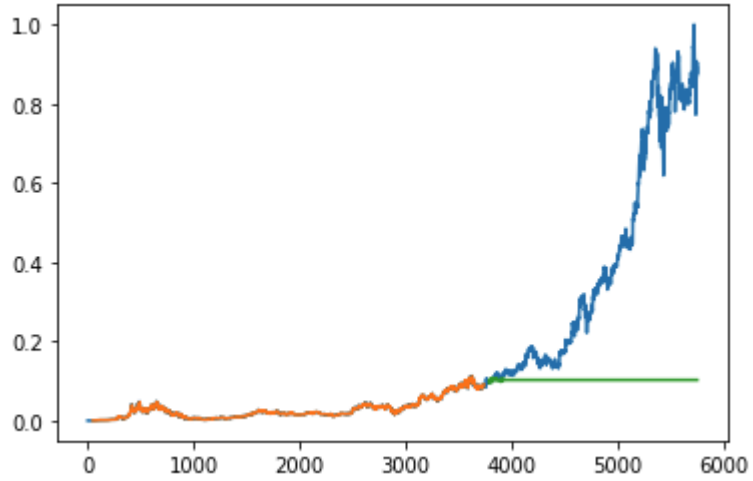


Figure 7.11: Training and Testing for Random Forest Regression

Run 4: 20 training iterations and 100 days in the past.

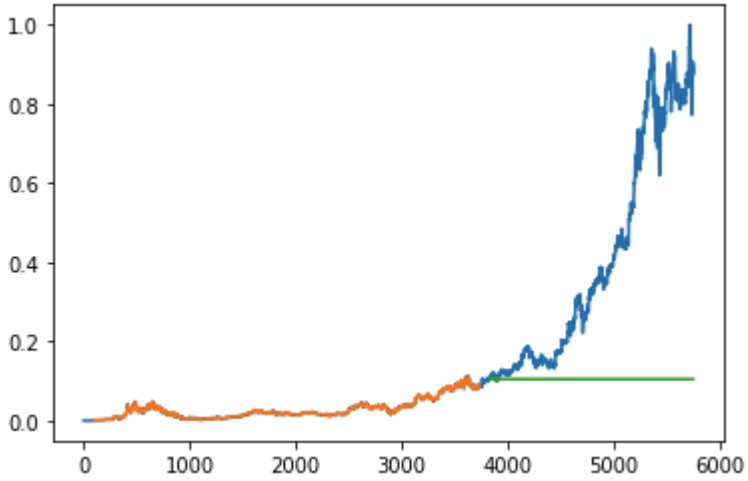


Figure 7.12: Training and Testing for Random Forest Regression

Long Short-Term Memory Neural Network

This algorithm works well for all scenarios based on the variation range (0,1) of the two parts of the curves on the corresponding screenshots that represent training and testing Root Mean Square Error (RMSE) for stock prices. The visual observation of the screenshots confirms the conclusion from the rough observation in section 6.2.4 that the number of training iterations and days in the past does not have a substantial impact on the good prediction accuracy of the algorithm.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

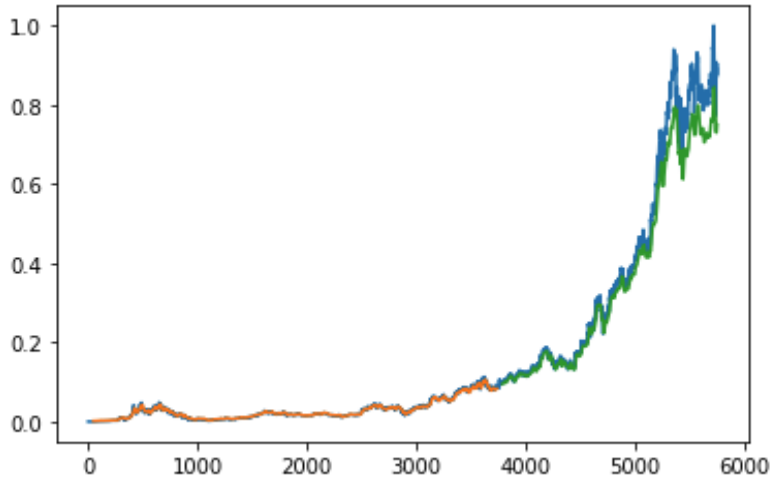


Figure 7.13: Training and Testing for LSTM Neural Network

Novel, first time application scenarios

Run 2: 10 training iterations and 100 days in the past.

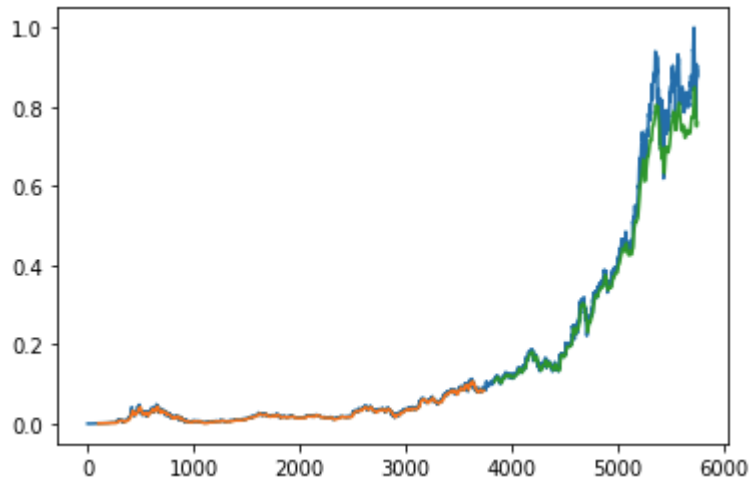


Figure 7.14: Training and Testing for LSTM Neural Network

Run 3: 20 training iterations and 50 days in the past.

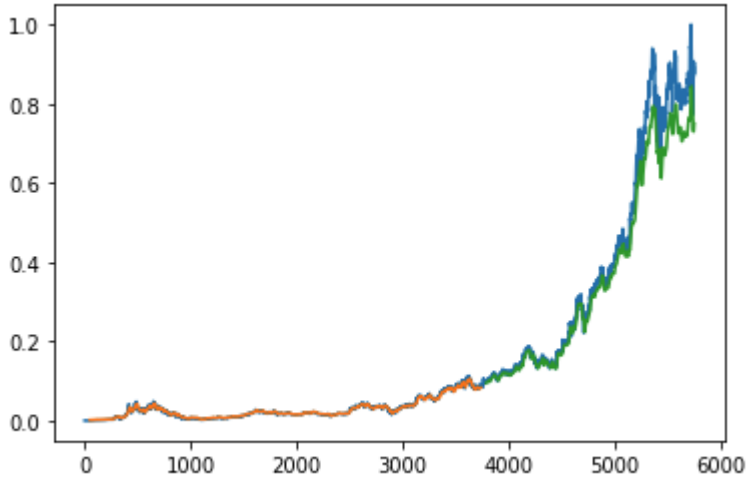


Figure 7.15: Training and Testing for LSTM Neural Network

Run 4: 20 training iterations and 100 days in the past.

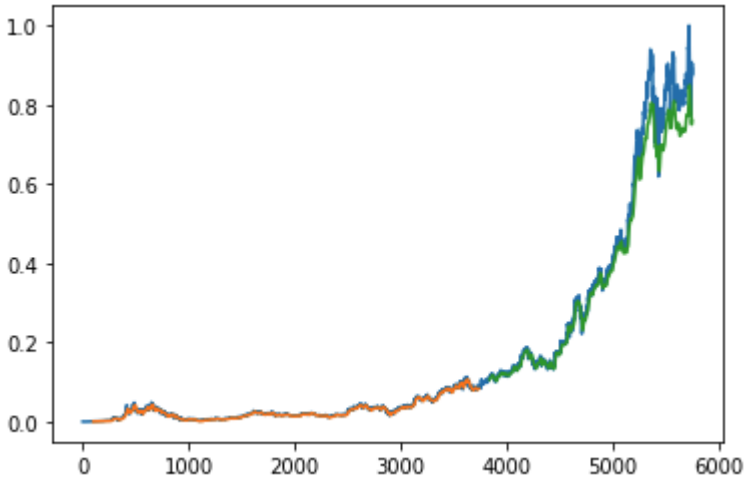


Figure 7.16: Training and Testing for LSTM Neural Network

Integrated representation

The following graph shows the testing Root Mean Square Error (RMSE) results for all the algorithms in an integrated way for each run. A legend for each algorithm can be found at the top right or top left corner of each graph identifying what particular colour represents each algorithm within the graph. These legends are presented as “LSTM” for Neural Network algorithm, “LR” for Linear Regression algorithm, “Poly” for Polynomial Regression algorithm and “RF” for Random Forest algorithm respectively.

Benchmark scenario

Run 1: 10 training iterations and 50 days in the past.

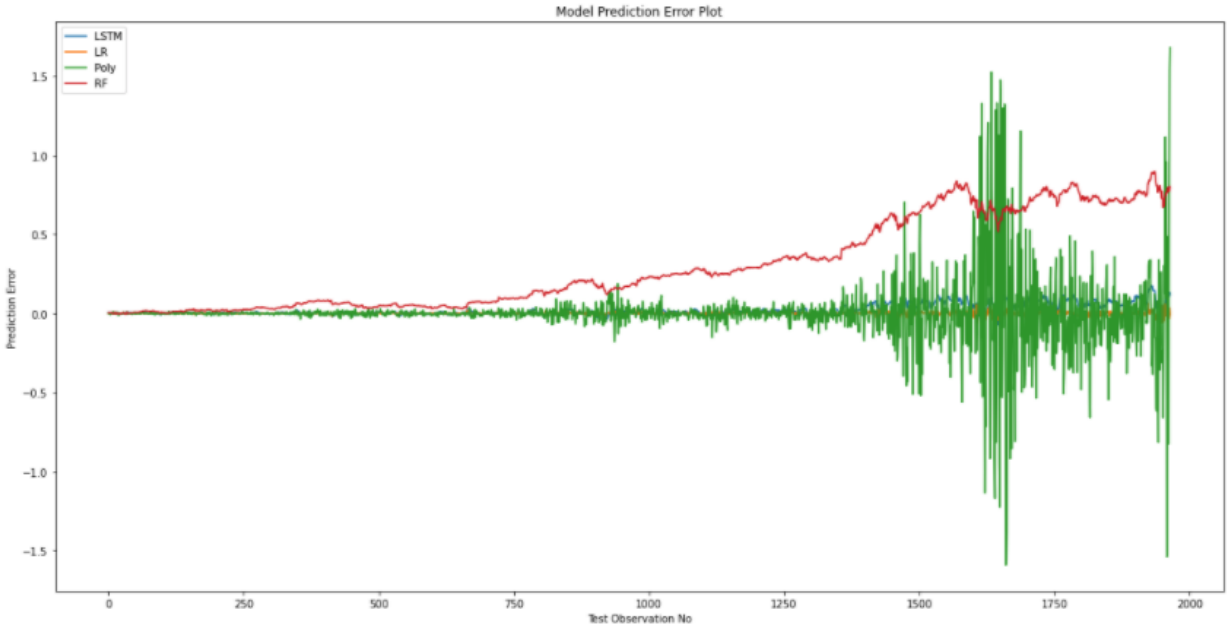


Figure 7.16: Integrated representation for Testing (1)

Novel, first time application scenarios

Run 2: 10 training iterations and 100 days in the past.

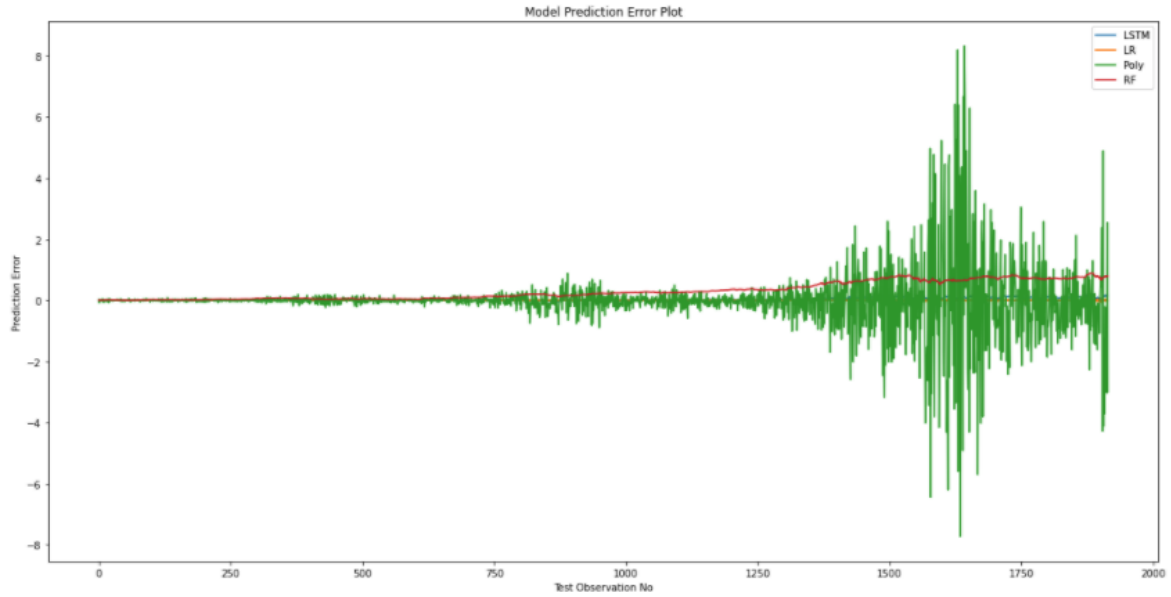


Figure 7.16: Integrated representation for Testing (2)

Run 3: 20 training iterations and 50 days in the past.

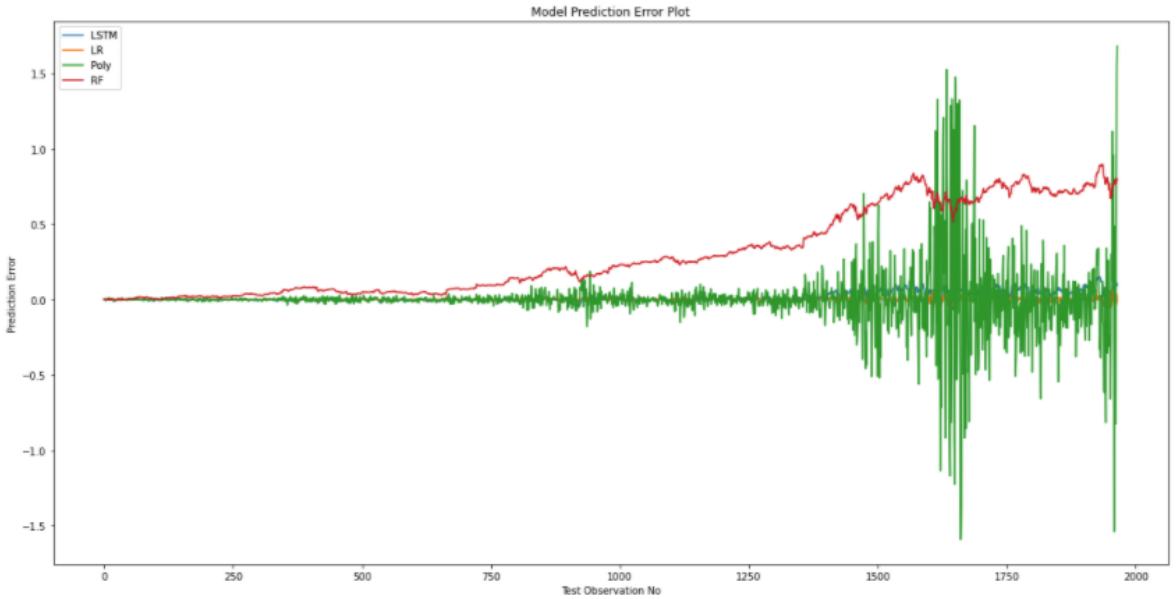


Figure 7.16: Integrated representation for Testing (3)

Run 4: 20 training iterations and 100 days in the past.

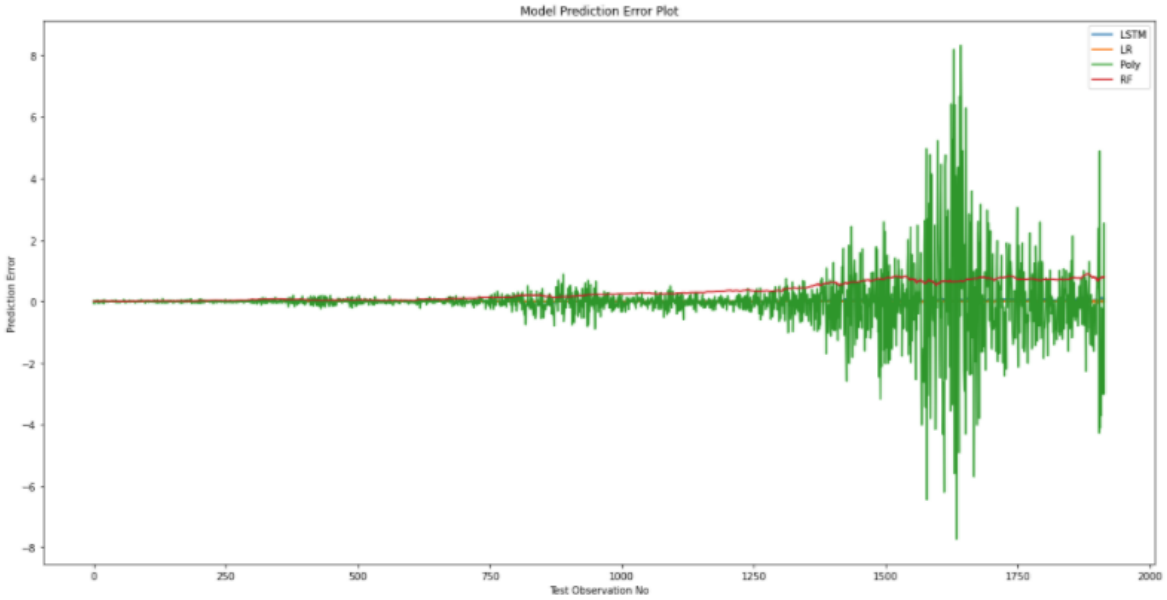


Figure 7.16: Integrated representation for Testing (4)

7.2.2 Algorithms Evaluation

This section presents three tables of 5x5 with Root Mean Square Error result evaluations and time taken performed by the machine learning for the different runs. These tables are presented in terms of training error, testing error and computational time. The best algorithm performance for training is Polynomial Regression (PR) and for testing is Linear Regression (LR) as they both maintained a great overall accuracy in their results. The worst algorithm performance for training is LSTM Neural Network and for testing is Polynomial Regression (PR) as they did not maintain a great overall accuracy in their results, these results are compared based on the benchmark run 1 with run 2. However, LSTM Neural Network is the only algorithm which made an improvement from run 1 upto run 4 for training and testing results. The best efficiency in terms of computational time was achieved in run 3.

Training

Root Mean Square Error (RMSE) for training is a way of representing a performance evaluation metric for accuracy during the learning process. The figures shown in table 7.1 present two clear observations. The first observation, which indicates run 1 for the benchmark approach, has the same values as run 3, similarly for run 2 and run 4 except for the LSTM Neural Network algorithm. Second observation, indicates how the accuracy changes with the number of days in the past for each algorithm between run 1 and run 2. Algorithms such as Linear Regression (LR) and Polynomial Regression (PR) increased in their accuracy results in run 2 compared to run 1, whilst Random Forest (RF) and LSTM Neural Network decreased in their accuracy results in run 2 compared to run 1. However, LSTM Neural Network increased its overall accuracy results in run 4 compared to runs 1, 2 and 3. In conclusion, based on these observations, the number of training iterations does not change the algorithms accuracy in most runs, however the increase of the number of days in the past changes the accuracy for all of the algorithms. For clarity, table 7.1 shows the best results for each algorithm highlighted in bold.

Trainning: Root Mean Square Error (RMSE) values				
Simulation scenarios / Machine Learning Algorithms	Run 1: Benchmark 10 number of trainning iterations 50 days in the past	Run 2: Novel 10 number of trainning iterations 100 days in the past	Run 3: Novel 20 number of trainning iterations 50 days in the past	Run 4: Novel 20 number of trainning iterations 100 days in the past
Linear Regression (LR)	0.0011333929498220	**0.0011254118145283**	0.0011333929498220	**0.0011254118145283**
Polynomial Regression (PR)	0.0005874331528703	**2.104837013662E-14**	0.0005874331528703	**2.104837013662E-14**
Random Forest Regression (RF)	**0.0004571049811023**	0.0004578849487166	**0.0004571049811023**	0.0004578849487166
LSTM Neural Network	0.0025748614649406	0.0028083088037128	0.0025748614649406	**0.0024926140940913**

Table 7.1: Root Mean Squared Error for Training

Testing

Root Mean Squared Error (RMSE) for testing is a way of representing a performance evaluation metric for accuracy during the validation process. The figures shown in table 7.2 present two clear observations. The first observation, indicates run 1 for the benchmark approach, having the same values as run 3, similarly for run 2 and run 4 except for the LSTM Neural Network algorithm. Second observation, indicates how the accuracy changes with the number of days in the past for each algorithm between run 1 and run 2. Algorithms such as Linear Regression (LR), Polynomial Regression (PR) and Random Forest (RF) decreased in their accuracy results in run 2 compared to run 1, however LSTM Neural Network increased its accuracy results in runs 1, 2 and 4. In conclusion, based on these observations, the number of training iterations does not change the algorithms accuracy in most runs, however the increase of the number of days in the past changes the accuracy for all of the algorithms. For clarity, table 7.2 shows the best results for each algorithm highlighted in bold.

Testing: Root Mean Square Error (RMSE) values				
Simulation scenarios / Machine Learning Algorithms	Run 1: Benchmark **10** number of trainning iterations **50** days in the past	Run 2: Novel **10** number of trainning iterations **100** days in the past	Run 3: Novel **20** number of trainning iterations **50** days in the past	Run 4: Novel **20** number of trainning iterations **100** days in the past
Linear Regression (LR)	**0.0094332258511968**	0.009686760360715	**0.0094332258511968**	0.009686760360715
Polynomial Regression (PR)	**0.2046464791621710**	0.997284063926984	**0.2046464791621710**	0.997284063926984
Random Forest Regression (RF)	**0.4082935685876730**	0.412106095354398	**0.4082935685876730**	0.412106095354398
LSTM Neural Network	0.055764914121696 1	0.049497095549923	0.055764914121696 1	**0.033478711736450**

Table 7.2: Root Mean Squared Error for Testing

Computational Time

Computational time Is a specific way of representing a performance evaluation metric for efficiency. The figures shown in table 7.3 are approximate times for the simulation runs performed by the software. The main observation from this table indicates how the efficiency changes with the number of days in the past between run 1 to run 2 and run 3 to run 4. In conclusion, based on this observation, the increase in the number of days in the past has a bigger impact in the time efficiency than the number of training iterations. For each run of the 4 algorithms the computational time is divided by 4 to obtain a raw estimate of computational time for each individual algorithm. The computational time in this table includes both training and testing times. These times can not be measured separately because the software gives both results for every algorithm simultaneously. For clarity, table 7.3 shows the best results for each algorithm highlighted in bold.

Computational time values (measured in minutes)				
Simulation scenarios / Machine Learning Algorithms	Run 1: Benchmark **10** number of trainning iterations **50** days in the past	Run 2: Novel **10** number of trainning iterations **100** days in the past	Run 3: Novel **20** number of trainning iterations **50** days in the past	Run 4: Novel **20** number of trainning iterations **100** days in the past
Linear Regression (LR)	1:42.63 minutes	3:13.27 minutes	2:28.51 minutes	4:27.32 minutes
Polynomial Regression (PR)	1:42.63 minutes	3:13.27 minutes	2:28.51 minutes	4:27.32 minutes
Random Forest Regression (RF)	1:42.63 minutes	3:13.27 minutes	2:28.51 minutes	4:27.32 minutes
LSTM Neural Network	1:42.63 minutes	3:13.27 minutes	2:28.51 minutes	4:27.32 minutes

Table 7.3: Computational time

7.3 Software Artefact

The software artefact has been tested using an incremental modular approach. Each module has been tested first individually with check points for essential program arguments and then gradually with the addition of the other tested modules. In this case, the modules represent individual code segments as the ones discussed in section 6.3.

By means of code segments tested, a reliable software artefact is implemented to also meet the specific requirements required. Tables with “met” and “partially met” for functional and non-functional requirements introduced in section 4.3 are shown in figures 7.4 and 7.5. All compulsory (Must Have) requirements have been fully met and all optional (May Have) requirements have been partially met. In conclusion, all requirements that are essential for the software artefact functionality have been fully met and the ones partially met are only desirable for this functionality.

Funtional Requirements	Met	Partially Met
Must Have		
Dataset Integration	✅	
Days Input	✅	
Iterations Input	✅	
Graphical Representation	✅	
May Have		
Interactivity		✅
Response Time		✅

Table 7.4: Functional Requirements

Non-Funtional Requirements	Met	Partially Met
Must Have		
Usability	✅	
Reliability	✅	
Maintainability	✅	
Performance	✅	
May Have		
Documentation		✅
Portability		✅

Table 7.5: Non-Functional Requirements

7.4 Summary

This chapter has looked at the testing and evaluation for each of the machine learning algorithms. Section 7.2 presented curves graphically for these algorithms in terms of Root Mean Square Error (RMSE) and integrated representation for training and testing which archives the objective 2 from section 1.3, also evaluated these two indicators in every simulation run for each machine learning algorithm. Section 7.3, showed a table of the requirements met from chapter 5.

The following chapter (8), refers to the conclusion phase which presents pros, cons, and achievements obtained in the implementation of this project.

Chapter 8
Conclusion

8.1 Achievements

The aim of this research focused engineering project has been successfully achieved. In particular, this report has systematically explored the suitability and applicability of several popular machine learning algorithms for financial modelling of the stock market by means of a research focused engineering project. This aim has been fully met through the achievement of all planned objectives.

Objective 1, to evaluate accuracy and efficiency of stock market prediction in a complementary way, has been achieved by exploring the relationship between these two performance indicators, as shown in section 7.2.2.

Objective 2, to visualise results from algorithm training and testing in an integrated way, has been achieved by presenting the Root Mean Square Error for these two stages in the same graph, as shown in section 7.2.1

Objective 3, to execute several machine learning algorithms in a simultaneous way, has been achieved by running these algorithms at the same time and showing their results together, as shown in sections 6.2.1-6.2.4 and section 7.2.1.

Objective 4, to develop a software tool for stock price prediction, has been achieved by writing source code for four machine learning algorithms, as shown in section 6.3 and appendix C.

Objective 5, to implement several established machine learning algorithms in this software tool, has been achieved by compiling and executing the source code for these machine learning algorithms, as shown in section 6.3

8.2 Contributions

This research focused engineering project has made several contributions in terms of academic novelty and performance evaluation. Each of the four machine learning algorithms considered for this project has been evaluated comparatively. In particular, each algorithm has been simulated and tested for four different simulation runs whereby one of these runs is a benchmark approach and the other three runs are novel approaches. In this context, the benchmark approach uses a low number of training iterations and days in the past in line with most other works in the field of short term prediction for stock prices. In contrast, the novel approaches use a higher number of training iterations and/or days in the past in order to take advantage of the ability of machine learning algorithms to perform better by using longer learning and

larger datasets. Finally, the prediction accuracy of these machine learning algorithms has been evaluated in relation to their computational efficiency in order to provide more flexibility to individual investors depending on their preferences.

8.3 Dissemination

The main research results from this project will be submitted for possible publication as a journal article and/or a conference paper such as the IEEE Transactions on Artificial Intelligence and the IEEE Symposium Series in Computational Intelligence. More details about this journal and conference are given below:

1- https://cis.ieee.org/publications/ieee-transactions-on-artificial-intelligence
2- https://attend.ieee.org/ssci-2021/

The research results of this project will also be submitted to the school of computing showcase at the University of Portsmouth.

8.4 Further Research

There are a number of future studies that can be undertaken for this research focus project which would be beneficial for anyone that wants to expand this project.

1. To evaluate more days in the past from different companies datasets for a performance comparison.
2. To run the software with more than 2 different numbers of training iteration for an even closer accuracy evaluation.
3. To add more company datasets from the NASDAQ stock market and compare the results of each company.
4. To combine datasets from different stock exchanges.
5. To use different machine learning algorithms for stock prediction such as Support Vector Regression (SVR), Support Vector Machines (SVM) and Back Propagation Neural Network (BPNN).
6. To use different financial indicators such as currency exchange rate.

8.5 Wider Context

The use of machine learning algorithms is very useful and has a lot of potential in the financial sector, predicting more general financial indicators, not just for individual investors but also for general economic business. It is also widely used in multiple other fields. Healthcare sector for medical diagnosis and disease. Agriculture sector to help farmers increase the productivity of new weeds and monitoring soil. Transportation sector to manage the traffic circulation and implementation of

self-driving vehicles. Those are just a few of the common sectors where machine learning has added a significant value in their productivity.

A webinar from the IEEE Computational Intelligence Society by Roy S. Freedman was attended by author within the development of this project which referred to “Understanding the Complexity of Financial Systems of Systems”.

8.6 General Reflections

The development of this chosen research project was conducted as planned, taking in consideration regular meetings with the supervisor and having good dynamic planning played a big part to ensure that the implementation of this project is in the best possible state and contains the successful results expected in the end. The difference from the project initiation document in appendix A, is that the actual work done in this research focus project is a more in depth investigation on machine learning algorithms. In regards to the Ethics form from appendix B for this particular project, as no external individuals were involved.

The time taken for the development of this project is something to be aware of in the future if starting again. Perfection for each section was a priority therefore things took longer to do leading to more time consuming. When developing a software, new ways to improve the current version will always be the case, therefore prioritising what is mandatory is advisable.

References

Abran, A., Khelifi, A., Suryn, W., & Seffah, A. (2003). Usability meanings and interpretations in ISO standards. *Software quality journal*, *11*(4), 325-338.

Alexandra Twin (2021, April). “Long-Term Investments”, Retrieved from: https://www.investopedia.com/terms/l/longterminvestments.asp

Arévalo, A., Niño, J., Hernández, G., & Sandoval, J. (2016). High-Frequency Trading Strategy Based on Deep Neural Networks. Intelligent Computing Methodologies Lecture Notes in Computer Science, 424436. doi:10.1007/978-3-319-42297-8_40

Arora, A., et al.: Deep Learning with H2O (2015) learning process 20. Zeiler, M.D.: ADADELTA: An Adaptive Learning Rate Method, 6 (2012)

Aumann, D. (2019, September 9). *R-stock Prediction*. github. https://github.com/daumann/r-stockPrediction

Barbacci, M., Klein, M., Longstaff, T., Weinstock, C. (1995). Quality Attributes. In *Computer Programming and Software.* (pp. 11-56). Carnegie Mellon University. https://apps.dtic.mil/sti/pdfs/ADA307888.pdf

Boehm, B. W. (1995). A spiral model of software development and enhancement. In *Readings in Human–Computer Interaction* (pp. 281-292). Morgan Kaufmann.

Bollen, J., & Mao, H. (2011). Twitter Mood as a Stock Market Predictor. Computer,44(10), 91-94. doi:10.1109/mc.2011.323

Chuqing Zhang, Han Zhang, Xiaoting Hu. (2019) A Contrastive Study of Machine Learning on Funding Evaluation Prediction. *IEEE Access* 7, pages 106307-106315

Claire Boyte-White (2021, April). “Understanding Long-term vs ShortTerm Capital Gains”, Retrieved from: https://www.investopedia.com/articles/personal-finance/101515/comparing-longterm-vs-shortterm-capital-gain-tax-rates.asp

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer*, *27*(8), 44-49.

Data Flair Team (n.d). “16 Facts about Python Programming that every Geek should know". Retrieved from Data Flair: https://data-flair.training/blogs/facts-about-python-programming/

Dataquest (2020, August 24). “How to Use Jupyter Notebook in 2020: A Beginner’s Tutorial”. Retrieved from: https://www.dataquest.io/blog/jupyter-notebook-tutorial/

Ding, X., Zhang, Y., Liu, T., & Duan, J. (2014). Using Structured Events to Predict Stock Price Movement: An Empirical Investigation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1148

Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. The Journal of Finance, 25(2), 383. doi:10.2307/2325486

Gomaa, H. (2011). Software Quality Attributes. In *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures* (pp. 357-368). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511779183.022

Hiransha, M., Gopalakrishnan, E. A., Krishna, V., & Soman, K.P. (2018). NSE Stock Market Prediction Using Deep-Learning Models. *Procedia Computer Science*, *132*(2018), 1351-1362. https://doi.org/10.1016/j.procs.2018.05.050

Horne, J. C., & Parker, G. G. (1967). The RandomWalk Theory: An Empirical Test. Financial Analysts Journal, 23(6), 87-92. doi:10.2469/faj.v23.n6.87

Huseyin Ince & Theodore B. Trafalis (2008) Short term forecasting with support vector machines and application to stock price prediction, International Journal of General Systems, 37:6, 677-687, DOI: 10.1080/03081070601068595

Jingyi Shen, M. Omair Shafiq. (2020) Short-term stock market price trend prediction using a comprehensive deep learning system. *Journal of Big Data* 7:1

Katelyn Peters (2021, January). “Short-Term Gain”, Retrieved from: https://www.investopedia.com/terms/s/short-term-gain.asp

Kercheval, A. N., & Zhang, Y. (2013, October 24). *Modeling high-frequency limit order book dynamics with support vector machines*. math.fsu.edu. https://www.math.fsu.edu/~aluffi/archive/paper462.pdf

Katherine Gail Nowadly and Sohyun Jung, (2020). “Nowadly using machine learning approaches to improve long range demand forecasting capstone”. Retrieved from: https://dspace.mit.edu/bitstream/handle/1721.1/126503/scm2020-nowadly-using-machine-learning-approaches-to-improve-long-range-demand-forecasting-capstone.pdf?sequence=1&isAllowed=y

Kaustubh Khare, Omkar Darekar, Prafull Gupta and V. Z. Attar, "Short term stock price prediction using deep learning", *Recent Trends in Electronics Information & Communication Technology (RTEICT) 2017 2nd IEEE International Conference on*, pp. 482-486, 2017.

Li, X., Huang, X., Deng, X., & Zhu, S. (2014). Enhancing quantitative intra-day stock return prediction by integrating both market news and stock prices information. Neurocomputing, 142, 228-238. doi:10.1016/j.neucom.2014.04.043

Marcus Thorström (2017). Applying machine learning to key performance indicators. Chalmers University of Technology and University of Gothenburg. 4, 25-36

Mark L. Mitchell and J. Harold Mulherin The Journal of Finance Vol. 49, No. 3, Papers and Proceedings Fifty-Fourth Annual Meeting of the American Finance Association, Boston, Massachusetts, January 3-5, 1994 (Jul., 1994), pp. 923-950

Matharu, G. S., Mishra, A., Singh, H. and Upadhyay, P., 2015. Empirical study of agile software development methodologies: A comparative analysis. ACM SIGSOFT Software Engineering Notes, 40 (1), 1–6.

Mehtab, S. and Sen, J. (2021) “A time series analysis-based stock price prediction using machine learning and deep learning models”, International Journal of Business Forecasting and Marketing Intelligence (IJBFMI), Inderscience Publishers. (In Press)

Murkute Amod, Tanuja Sarode, “Forecasting market price of stock using artificial neural networks”, International Journal of Computer Applications, 124 (12) (2015), pp. 11-15

Nelson, D. M. Q., A. C. M. Pereira and R. A. de Oliveira, "Stock market's price movement prediction with LSTM neural networks," 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 1419-1426, doi: 10.1109/IJCNN.2017.7966019.

Nirbhey, S., Neeha K., Deepali V., Vidhi S (April 2017). Stock Prediction using Machine Learning a Review Paper. Journal of Computer Applications (0975 – 8887) Volume 163 –

Patil, S and Kulkarni, U. "Accuracy Prediction for Distributed Decision Tree using Machine Learning approach," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 1365-1371, doi: 10.1109/ICOEI.2019.8862580.

Royce, W. W. (1987, March). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering* (pp. 328-338).

Saqib Aziz, Michael M. Dowling, Helmi Hammami, Anke Piepenbrink. (2019) Machine Learning in Finance: A Topic Modeling Approach. *SSRN Electronic Journal*.

Schumaker, R. P., & Chen, H. (2009). A quantitative stock prediction system based on financial news. Information Processing & Management, 45(5), 571-583. doi:10.1016/j.ipm.2009.05.001

Shikhar, S. (2018, August 5). *Matlab Module for Stock Market Prediction using Simple NN*. github. https://github.com/Shikhar1998/Stock-Market-Prediction-using-Neural-Networks-and-Genetic-Algorithm

Simon, J. (2020, February 23). *Jupyter notebook which illustrates moving average crossover strategy and neural network predictions for stock trading*. github. https://github.com/jer805/stock-prediction/blob/master/Stocks%20and%20Neural%20Networks.ipynb

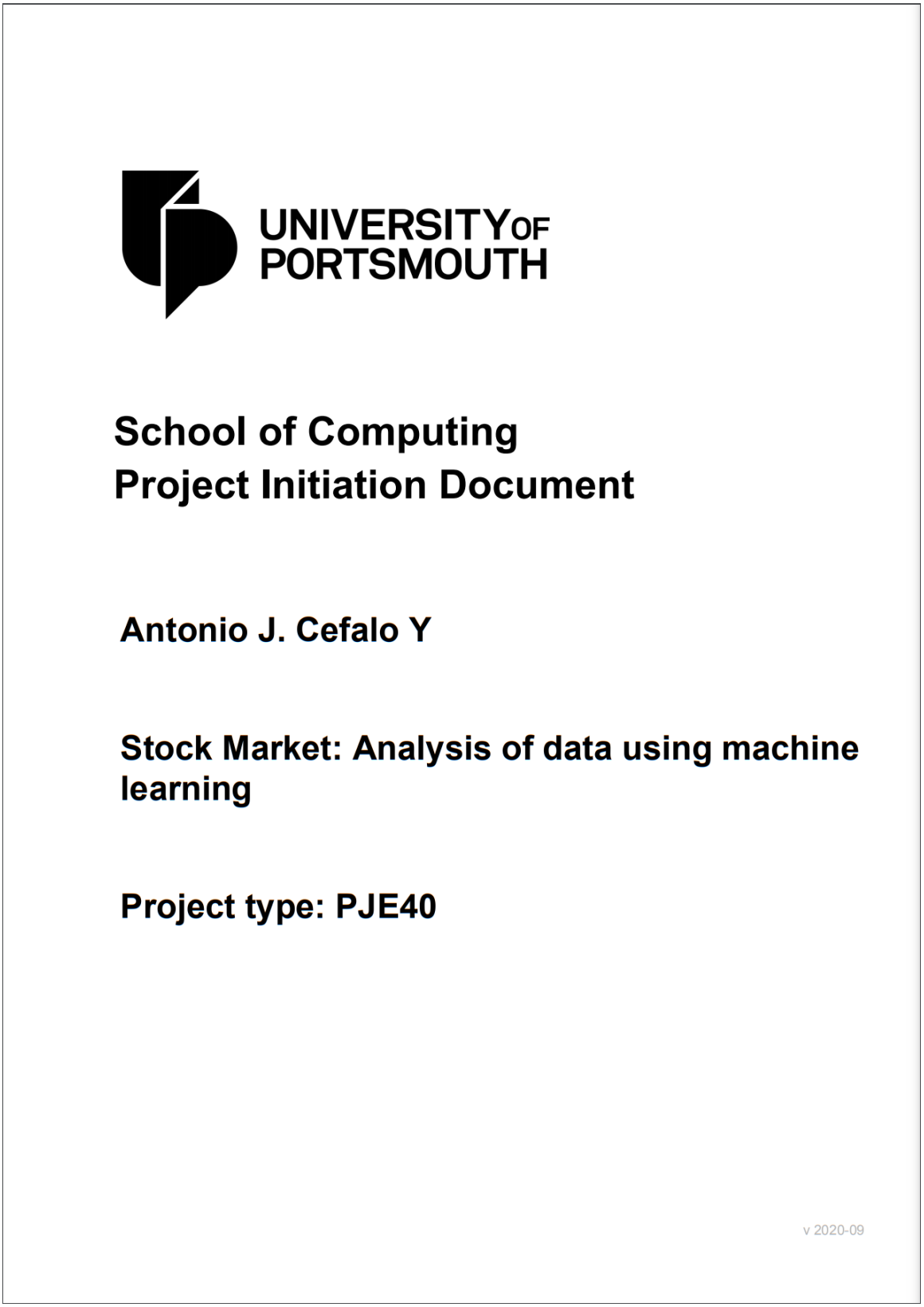
Singh, Aishwarya. (26 July 2019) “Predicting the Stock Market Using Machine Learning and Deep Learning.” Analytics Vidhya, article. www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learningtechniques-python/.

Vijh, M., Chandola, D., Tikkiwal, V.A., Kumar, A.Hide details, Stock Closing Price Prediction using Machine Learning Techniques, Procedia Computer Science, Volume 167, 2020

Weng, B., Ahmed, M. A., & Megahed, F. M. (2017). Stock market one-day ahead movement prediction using disparate data sources. Expert Systems with Applications,79, 153-163. doi:10.1016/j.eswa.2017.02.041

Appendices

A: Project Initiation Document



1. Basic details

Student name:	Antonio Jesus Cefalo Yaghmour
Draft project title:	Stock Market: Analysis of data using machine learning
Course:	Software Engineering
Project supervisor:	Alex Gegov
Client organisation:	
Client contact name:	

2. Degree suitability

As a Software Engineering student we learn different approaches to solve a problem using algorithms when talking about technology. In this specific project an implemented algorithm will be used in order to analyse data from the Stock Market and compare them.

3. Outline of the project environment and problem to be solved

The users/ clients for this project are people that are involved and interested in learning the financial market. Someone that has never invested before will like to know further about a specific company to give them a piece of mind on what to expect. The program would be use to show future possible values of a company by comparing their previous data. This project is required for the clients/ users with or without knowledge to help them make the right decision in their specific investment.

4. Project aim and objectives

The overall aim of the project is to develop the right software using applied financial systems. The objectives for the project includes reviewing different literatures about the project, researching in depth different markets that have been already developed, gathering the requirements of the project, make a project plan, start creating a software design, developing the software and testing.

5. Project deliverables

A stock market analyser will be developed producing test code which will be provided combined with the final report.Implementing the software however, will produce the final report where requirements, design specifications, test strategies, user guides, that are useful outside of the project report.

6. Project constraints

Due to the time frame, just a specific stock market will be analysed. However the market more suitable for this project will be in depth taking into account.

7. Project approach

First of all a deep look into existing research used to predict the stock market. To establish the requirements a Waterfall Model Software methodology will be used. Running previous developments of similar approach will be considered methods/algorithms for predicting trends in the stock market. That will be of massive help as having a deep understanding of how they work and also how accurate they are in order to perfect the software to be developed in this project and therefore apply new skills needed

8. Literature review plan

Looking at existing algorithms used for predicting future data will be the starting point in order to have a better understanding and have a clear approach to the research in this project.

9. Facilities and resources

Taking in consideration different program languages to be tested first, will then define how accurate the approach to the final solution can be. Different other resources such as Microsoft products, GitHub and online forums will be use to gather information needed for the development of this project. The project requires a computer in order to run the software successfully and to run test accurately.

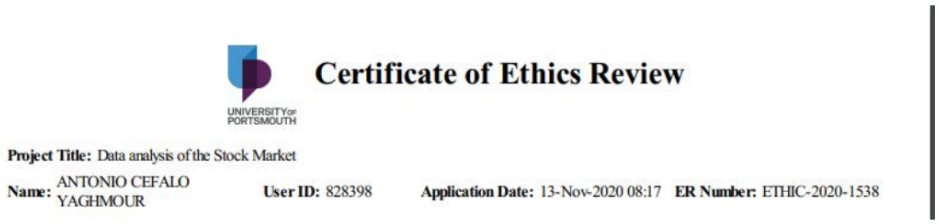
10. Log of risks

Description	Impact	Likelihood	Mitigation	First indicator
Data loss- caused from work not being saved correctly	*Technical*	*Likely*	*Turn on Auto-Save were applicable*	*Work that had been completed suddenly becoming missing*

11. Project plan

- Create a Gantt Chart
- Literature Review
- Requirements
- System Requirements
- Design Specification
- Development
- Testing
- Write Report

12. Legal, ethical, professional, social issues (mandatory)



UNIVERSITY OF PORTSMOUTH

Certificate of Ethics Review

Project Title: Data analysis of the Stock Market

Name: ANTONIO CEFALO YAGHMOUR **User ID:** 828398 **Application Date:** 13-Nov-2020 08:17 **ER Number:** ETHIC-2020-1538

B: Ethics Certificate

Certificate of Ethics Review

Project Title: Data analysis of the Stock Market

Name: ANTONIO CEFALO YAGHMOUR **User ID:** 828398 **Application Date:** 13-Nov-2020 08:17 **ER Number:** ETHIC-2020-1538

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **SOC**
What is your primary role at the University?: **UndergraduateStudent**
What is the name of the member of staff who is responsible for supervising your project?: **Alex Gegov**
Is the study likely to involve human subjects (observation) or participants?: **No**
Are there risks of significant damage to physical and/or ecological environmental features?: **No**
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**
Does the project involve animals in any way?: **No**
Could the research outputs potentially be harmful to third parties?: **No**
Could your research/artefact be adapted and be misused?: **No**
Does your project or project deliverable have any security implications?: **No**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature:  **Date:** 13/11/2020

C: Source Code

+*In[1]:*+
[source, ipython3]
----
import numpy as np
#import tensorflow as tf
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline

+*In[2]:*+
[source, ipython3]
----
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

+*In[3]:*+
[source, ipython3]
----
df = pd.read_csv('AMZN.csv',index_col='Date')
df.index = pd.to_datetime(df.index)
df.head(5)

[[step-2---understand-the-data]]
== Step 2 - Understand the Data

+*In[4]:*+
[source, ipython3]
----
df.isnull().sum()

+*In[5]:*+
[source, ipython3]
----
df.info()

+*In[6]:*+
[source, ipython3]

```
df.describe()
```

+*In[7]:*+
[source, ipython3]
```
df['Close'].head()
```

+*In[8]:*+
[source, ipython3]
```
df['Close'].tail()
```

+*In[9]:*+
[source, ipython3]
```
plt.scatter(df.index,df['Close'])
```

+*In[10]:*+
[source, ipython3]
```
import math
```

+*In[11]:*+
[source, ipython3]
```
plt.hist(df['Close'])
```

+*In[12]:*+
[source, ipython3]
```
df1 = df['Close']
```

+*In[13]:*+
[source, ipython3]
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

+*In[14]:*+
[source, ipython3]
----
print(df1)

+*In[15]:*+
[source, ipython3]

##splitting dataset into train and test split
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]

+*In[16]:*+
[source, ipython3]
----
training_size,test_size

+*In[17]:*+
[source, ipython3]
----
import numpy
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
                                                    dataX, dataY = [], []
                                                for i in range(len(dataset)-time_step-1):

                                a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100

                                                                    dataX.append(a)

                                                                dataY.append(dataset[i + time_step, 0])
                                                    return numpy.array(dataX), numpy.array(dataY)

[[parameter---1]]
=== Parameter - 1

+*In[18]:*+
[source, ipython3]
----
time_step = 100

+*In[19]:*+
[source, ipython3]
----
reshape into X=t,t+1,t+2,t+3 and Y=t+4

X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)

+*In[20]:*+
[source, ipython3]
----
print(X_train.shape), print(y_train.shape)

+*In[21]:*+
[source, ipython3]
----
print(X_test.shape), print(ytest.shape)

+*In[22]:*+
[source, ipython3]

X_train_size = X_train.shape[0]
y_train_size = y_train.shape[0]

X_test_size = X_test.shape[0]
y_test_size = ytest.shape[0]

print("X-train size", X_train_size)
print("y-train size", y_train_size)
print("")
print("X-test size", X_test_size)
print("y-test size", y_test_size)

+*In[23]:*+
[source, ipython3]

# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

+*In[24]:*+
[source, ipython3]

Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

+*In[25]:*+
[source, ipython3]
---
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(time_step,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')

+*In[26]:*+
[source, ipython3]
---
model.summary()

+*In[27]:*+
[source, ipython3]
---
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=10,batch_size=64,ve
rbose=1)

+*In[28]:*+
[source, ipython3]

Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)

+*In[29]:*+
[source, ipython3]
---
lsrm_y_test = ytest.reshape(y_test_size)
lstm_compare = pd.concat([pd.DataFrame(lsrm_y_test, columns=['Actual']),
                    pd.DataFrame(test_predict, columns=['LSTM_Pred'])],axis=1)
lstm_compare['error'] = lstm_compare['Actual']-lstm_compare['LSTM_Pred']

```
lstm_compare.head()
```

+*In[30]:*+
[source, ipython3]

```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
lstm_rmse_train = math.sqrt(mean_squared_error(y_train,train_predict))
print('Train - LSTM RMSE:', lstm_rmse_train)
```

+*In[31]:*+
[source, ipython3]

```
### Test Data RMSE
lstm_rmse_test = math.sqrt(mean_squared_error(ytest,test_predict))
print('Test - LSTM RMSE:', lstm_rmse_test)
```

+*In[32]:*+
[source, ipython3]

```
### Plotting
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
#plt.plot(scaler.inverse_transform(df1))
plt.plot(df1)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

+*In[33]:*+
[source, ipython3]

```
len(test_data)
```

```
---

  +*In[34]:*+
[source, ipython3]
----
x_input=test_data[y_train_size+1:].reshape(1,-1)
x_input.shape
----

  +*In[35]:*+
[source, ipython3]
----
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
----

 +*In[36]:*+
[source, ipython3]
----
day_new=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+1+30) #next 30 days prediction
----

  +*In[37]:*+
[source, ipython3]
----
import matplotlib.pyplot as plt
----

+*In[38]:*+
[source, ipython3]
----
len(df1)
----

 +*In[39]:*+
[source, ipython3]
----
from sklearn.linear_model import LinearRegression
----

+*In[40]:*+
[source, ipython3]
----
lr = LinearRegression()
----
```

+*In[41]:*+
[source, ipython3]
----
print(X_train.shape)
lr_X_train = X_train.reshape(X_train_size, time_step)
print(lr_X_train.shape)

+*In[42]:*+
[source, ipython3]
----
print(y_train.shape)
lr_y_train = y_train.reshape(y_train_size)
print(lr_y_train.shape)

+*In[43]:*+
[source, ipython3]
----
print(X_test.shape)
lr_X_test = X_test.reshape(X_test_size, time_step)
print(lr_X_test.shape)

+*In[44]:*+
[source, ipython3]
----
print(ytest.shape)
lr_y_test = ytest.reshape(y_test_size)
print(lr_y_test.shape)

+*In[45]:*+
[source, ipython3]
----
lr.fit(lr_X_train,lr_y_train)

+*In[46]:*+
[source, ipython3]

Lets Do the prediction and check performance metrics
lr_train_predict=lr.predict(lr_X_train)
lr_test_predict=lr.predict(lr_X_test)

+*In[47]:*+
[source, ipython3]
----
lr_compare = pd.concat([pd.DataFrame(lr_y_test, columns=['Actual']),
pd.DataFrame(lr_test_predict,
columns=['LinearRegression_Pred'])],axis=1)
lr_compare['error'] = lr_compare['Actual']-lr_compare['LinearRegression_Pred']
lr_compare.head()

+*In[48]:*+
[source, ipython3]
----
lr_rmse_train = math.sqrt(mean_squared_error(lr_y_train,lr_train_predict))
print('Train - Linear Regression RSME:', lr_rmse_train)

+*Out[48]:*+

Train - Linear Regression RSME: 0.0011254118145283014

[[rmse-of-test-data-using-linear-regression]]
=== RMSE of Test Data using Linear Regression

+*In[49]:*+
[source, ipython3]

Test Data RMSE
lr_rmse_test = math.sqrt(mean_squared_error(lr_y_test,lr_test_predict))
print('Test - Linear Regression RSME:', lr_rmse_test)

+*In[50]:*+
[source, ipython3]

Plotting
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(lr_train_predict)+look_back, :] =
lr_train_predict.reshape(X_train_size,1)
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan

```
testPredictPlot[len(lr_train_predict)+(look_back*2)+1:len(df1)-1,  :]  =
lr_test_predict.reshape(X_test_size,1)
# plot baseline and predictions
#plt.plot(scaler.inverse_transform(df1))
plt.plot(df1)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

+*In[51]:*+
[source, ipython3]

```
import statsmodels.api as sm

import pandas as pd

from patsy import dmatrices
```

+*In[52]:*+
[source, ipython3]

Linear Regression Model Coefficients

[[polynomial-regression]]
= Polynomial Regression

+*In[53]:*+
[source, ipython3]

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

+*In[54]:*+
[source, ipython3]

```
nb_degree = 2

polynomial_features = PolynomialFeatures(degree = nb_degree)

X_TRANSF = polynomial_features.fit_transform(lr_X_train)
```

+*In[55]:*+

[source, ipython3]
----
p_lr = LinearRegression()

p_lr.fit(X_TRANSF, lr_y_train)

+*In[56]:*+
[source, ipython3]

X_test_TRANSF = polynomial_features.fit_transform(lr_X_test)
p_lr_test_predict = p_lr.predict(X_test_TRANSF)
p_lr_train_predict = p_lr.predict(X_TRANSF)

+*In[57]:*+
[source, ipython3]
----
p_lr_compare = pd.concat([pd.DataFrame(lr_y_test, columns=['Actual']),
                                                    pd.DataFrame(p_lr_test_predict,
columns=['Polynomial_Pred'])],axis=1)
p_lr_compare['error'] = p_lr_compare['Actual']-p_lr_compare['Polynomial_Pred']

p_lr_compare.head()

  +*In[58]:*+
[source, ipython3]
----
pr_rmse_train = math.sqrt(mean_squared_error(lr_y_train,p_lr_train_predict))
print('Train - Polynomial Regression RSME:', pr_rmse_train)

+*In[59]:*+
[source, ipython3]

Test Data RMSE
pr_rmse_test = math.sqrt(mean_squared_error(lr_y_test,p_lr_test_predict))
print('Test - Polynomial Regression RSME:', pr_rmse_test)

+*In[60]:*+
[source, ipython3]

Plotting
# shift train predictions for plotting
look_back=time_step

```
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(p_lr_train_predict)+look_back, :] =
p_lr_train_predict.reshape(X_train_size,1)
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(p_lr_train_predict)+(look_back*2)+1:len(df1)-1, :] =
p_lr_test_predict.reshape(X_test_size,1)
# plot baseline and predictions
#plt.plot(scaler.inverse_transform(df1))
plt.plot(df1)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
----

+*In[61]:*+
[source, ipython3]
----
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=100, random_state=0)
regressor.fit(lr_X_train, lr_y_train)
rf_test_predict = regressor.predict(lr_X_test)
rf_train_predict = regressor.predict(lr_X_train)
----

[[rmse-of-train-data-using-random-forest-regression]]
=== RMSE of Train Data using Random Forest Regression

+*In[62]:*+
[source, ipython3]
----
rf_rmse_train = math.sqrt(mean_squared_error(lr_y_train,rf_train_predict))
print('Train - Random Forest RSME:', rf_rmse_train)
----

+*In[63]:*+
[source, ipython3]
----
### Test Data RMSE
rf_rmse_test = math.sqrt(mean_squared_error(lr_y_test,rf_test_predict))
print('Test - Random Forest RSME:', rf_rmse_test)
----

+*In[64]:*+
```

[source, ipython3]
---
type(rf_test_predict)

+*In[65]:*+
[source, ipython3]
---
rf_compare = pd.concat([pd.DataFrame(lr_y_test, columns=['Actual']),
pd.DataFrame(rf_test_predict,
columns=['RandomForest_Pred'])],axis=1)
rf_compare['error'] = rf_compare['Actual']-rf_compare['RandomForest_Pred']
rf_compare.head()

+*In[66]:*+
[source, ipython3]

Plotting
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(rf_train_predict)+look_back, :] =
rf_train_predict.reshape(X_train_size,1)
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(rf_train_predict)+(look_back*2)+1:len(df1)-1, :] =
rf_test_predict.reshape(X_test_size,1)
# plot baseline and predictions
#plt.plot(scaler.inverse_transform(df1))
plt.plot(df1)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()

+*In[67]:*+
[source, ipython3]
---
plt.figure(figsize=(20,10))
plt.plot(lstm_compare['error'])
plt.plot(lr_compare['error'])
plt.plot(p_lr_compare['error'])
plt.plot(rf_compare['error'])
plt.legend(['LSTM','LR','Poly','RF'])

```
plt.title('Model Prediction Error Plot')
plt.xlabel('Test Observation No')
plt.ylabel('Prediction Error')
plt.plot()
```

+*In[68]:*+
[source, ipython3]

```
fig, axs = plt.subplots(2, 2, figsize=(15,15))
axs[0, 0].plot(lstm_compare['Actual'])
axs[0, 0].plot(lstm_compare['LSTM_Pred'])
axs[0, 0].set_title('Actual Vs Pred (LSTM)')
axs[0, 0].set_xlabel('Observation No')
axs[0, 0].set_ylabel('Price')

axs[0, 1].plot(lr_compare['Actual'])
axs[0, 1].plot(lr_compare['LinearRegression_Pred'])
axs[0, 1].set_title('Actual Vs Pred (LR)')
axs[0, 1].set_xlabel('Observation No')
axs[0, 1].set_ylabel('Price')

axs[1, 0].plot(p_lr_compare['Actual'])
axs[1, 0].plot(p_lr_compare['Polynomial_Pred'])
axs[1, 0].set_title('Actual Vs Pred (Polynomial)')
axs[1, 0].set_xlabel('Observation No')
axs[1, 0].set_ylabel('Price')

axs[1, 1].plot(rf_compare['Actual'])
axs[1, 1].plot(rf_compare['RandomForest_Pred'])
axs[1, 1].set_title('Actual Vs Pred (RF)')
axs[1, 1].set_xlabel('Observation No')
axs[1, 1].set_ylabel('Price')
plt.plot()
```

+*In[69]:*+
[source, ipython3]

```
print("LSTM - RSME", lstm_rmse_test)
print("LR - RSME", lr_rmse_test)
print("PR - RSME", pr_rmse_test)
print("RF - RSME", rf_rmse_test)
```
