UNIVERSITY OF WEST ATTICA

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

Stock Prices Forecasting using Machine Learning Techniques

Undergraduate:

Mr. Emmanouil-Andreas A. Loukaidis
Registration Number: 50106854

Supervisor:

Dr. Grigorios E. Koulouras
Assistant Professor

ATHENS-EGALEO, NOVEMBER 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

Πρόβλεψη της Τιμής των Μετοχών χρησιμοποιώντας Τεχνικές Μηχανικής Μάθησης

Φοιτητής:

Εμμανουήλ-Ανδρέας Α. Λουκαΐδης
Αριθμός Μητρώου: 50106854

Επιβλέπων Καθηγητής:

Δρ. Γρηγόριος Ε. Κουλούρας
Επίκουρος Καθηγητής

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΝΟΕΜΒΡΙΟΣ 2020

ΔΗΛΩΣΗ ΠΕΡΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ ΚΑΙ ΛΟΓΟΚΛΟΠΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπόγραφα ότι η παρούσα εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα αποκλειστικά και ότι είμαι ο αποκλειστικός συγγραφέας του κειμένου της.

Η εργασία μου δεν προσβάλλει οποιασδήποτε μορφής δικαιώματα πνευματικής ιδιοκτησίας, προσωπικότητας ή προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής ή λογοκλοπής.

Κάθε βοήθεια που έλαβα για την ολοκλήρωση της εργασίας είναι αναγνωρισμένη και αναφέρεται λεπτομερώς στο κείμενό της. Ειδικότερα, έχω αναφέρει ευδιάκριτα μέσα στο κείμενο και με την κατάλληλη παραπομπή όλες τις πηγές δεδομένων, κώδικα προγραμματισμού Η/Υ, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών που χρησιμοποιήθηκαν, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης, και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Επιπλέον, όλες οι πηγές που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης κατά τα διεθνή πρότυπα.

Τέλος δηλώνω ενυπόγραφα ότι αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της είναι προϊόν λογοκλοπής.

Ημερομηνία 04/11/2020
Εμμανουήλ Ανδρέας Α. Λουκαΐδης

[illegible]

Digitally signed by
ΕΜΜΑΝΟΥΗΛ
ΑΝΔΡΕΑΣ ΛΟΥΚΑΪΔΗΣ
Date: 2020.11.05
10:42:20 +02'00'

Dedication

This study is wholeheartedly dedicated to my beloved family, who has been my source of inspiration and who continually provide their moral, spiritual, emotional, and financial support.

It is also dedicated to all the people who are nearby me in my toughest times and will be in those to come.

Acknowledgments

I would like to acknowledge my supervisor, Mr. Koulouras Grigorios, for his guidance through each stage of the process and for inspiring my interest in the field of informatics. I would also want to thank a Ph.D. Candidate, Mr. Zantalis Fotios, who also gave me a great deal of assistance. In addition, I would like to thank my sister Georgia Loukaidou, which is an exceptional assistant and the most prominent Ph.D. Candidate in Astrophysics.

Abstract

Predicting stock behavior is an issue that has been studied by many scientists and engineers, and it forms today's cutting-edge research. Great interest can be found in the combination of the financial science and the IT science, as the correct prediction of stock shares is of great economic importance for the apparent reason that an application could be profitable for either the creator of the customer. Using computing power today, which has reached unprecedented levels of power, we can build more complex and computationally powerful applications than ever before. A typical example for approximately the past ten years is the exponential growth in the development and use of "Artificial Intelligence" applications. More and more people see the prospects and the countless applications that such models in the real world could give, as countless companies are already providing these services, such as the colossal companies Google and Amazon. Therefore, using Python programming language and its libraries like SciKit Learn and Keras, we can create stock behavior, prediction models. More specifically, we will study linear models of the SciKit Learn library such as Linear Regression, Tree Decision, and Deep Learning models of the Keras library, such as Long Short-Term Memory, which is a more accurate model than the previous models in terms of stock forecasting. The purpose of the current thesis is to analyze these forecasting techniques and learn how they can be applied in the financial data and become familiar with the Python programming language and its capabilities in data analysis and processing.

Keywords

Machine Learning, Neural Networks, Python, Stock Market Forecasting, Regression, Deep Learning.

Περίληψη

Η πρόβλεψη της συμπεριφοράς των μετοχών είναι ένα ζήτημα το οποίο έχει μελετηθεί από πολλούς επιστήμονες και μηχανικούς και αποτελεί έρευνα αιχμής. Τεράστιο ενδιαφέρον προκαλεί ο συνδυασμός του οικονομικού τομέα και του τομέα της πληροφορικής, καθώς έχει μεγάλη οικονομική σημασία η σωστή πρόβλεψη των μετοχών για τον ευνόητο λόγο που μπορεί να προβεί μία εφαρμογή κερδοφόρα για τον δημιουργό ή για τον πελάτη. Χρησιμοποιώντας την υπολογιστική δύναμη στις μέρες μας, η οποία έχει φτάσει σε πρωτόγνωρα επίπεδα δύναμης, μπορούμε να φτιάξουμε εφαρμογές πιο περίπλοκες και υπολογιστικά πιο δυνατές από ποτέ. Χαρακτηριστικό παράδειγμα τα τελευταία περίπου 10 χρόνια είναι η εκθετική αύξηση στην εξέλιξη και χρήση εφαρμογών «Τεχνητής Νοημοσύνης». Όλο και περισσότερος κόσμος βλέπει την προοπτική και τις αμέτρητες εφαρμογές που σου δίνουν τέτοιου είδους μοντέλα στον πραγματικό κόσμο, καθώς ακολουθούν και αμέτρητες εταιρείες όπως οι εταιρείες κολοσσοί Google και Amazon. Χρησιμοποιώντας λοιπόν τη γλώσσα προγραμματισμού Python και τις βιβλιοθήκες της όπως η SciKit Learn και η Keras μπορούμε να δημιουργήσουμε μοντέλα πρόβλεψης συμπεριφοράς των μετοχών. Πιο συγκεκριμένα θα δούμε γραμμικά μοντέλα της βιβλιοθήκης SciKit Learn όπως το Linear Regression και το Tree Decision καθώς και μοντέλα Deep Learning της βιβλιοθήκης Keras όπως το Long Short-Term Memory, το οποίο είναι ακριβέστερο από τα υπόλοιπα μοντέλα όσον αφορά την πρόβλεψη μετοχών. Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάλυση αυτών των τεχνικών πρόβλεψης και το πώς εφαρμόζονται πάνω στον οικονομικό τομέα, αλλά και η εξοικείωση με τη γλώσσα προγραμματισμού Python και τις δυνατότητές της πάνω στην ανάλυση και επεξεργασία δεδομένων.

Λέξεις – κλειδιά

Machine Learning, Νευρωνικά Δίκτυα, Python, Πρόβλεψη Μετοχών, Regression, Βαθιά Μάθηση.

Table of Contents

List of Tables

List of Figures

Dictionary

AI: Artificial Intelligence

ANN: Artificial Neural Network

API: Application Programming Interface

CEO: Chie Executive Officer

CPU: Central Processing Unit

CSV: Comma Separated Values

GDPR: General Data Protection Regulation

GPU: Graphics Processing Unit

GUI: Graphical User Interface

IDE: Integrated Development Environment

LSTM: Long Short-Term Memory

MLP: MultiLayer Perceptron

MSE: Mean Squared Error

NLP: Natural Language Processing

NN: Neural Network

RELU: REctified Linear Unit

RMSE: Root Mean Squared Error

RNN: Recurrent Neural Network

SQL: Structured Query Language

TPU: Tensor Processing Unit

1 CHAPTER 1 – Preface

1.1 The Subject of this Thesis

Year by year, computational power accomplishes more significant leaps forward than ever before. As a result, it has become easily accessible to more and more people all over the world. For example, a computer that was quite expensive and powerful for 2015 should have lower performance capabilities compared to a cheaper entry-level computer in 2020. For many years, the Informatics domain called Machine Learning was quite challenging to be studied because it demanded substantial computational power and utilizes a vast amount of data to operate. Nowadays, with the emerge of Cloud Services such as Google Colab, which enables all the developers to use Google's hardware, it is easier than ever before to study and build Machine Learning models and run them and obtain real-time results. This thesis tries to dive deeper into the making of Neural Networks (NN) and try to implement them in real-world tasks. Specifically, this current research focuses on Stock prediction, in which many companies have already implemented their unique formulas to help them predict the rise or fall of their stocks. Many companies and services widely use Machine Learning algorithms throughout the world wide web. Such as Google uses its algorithms to predict all possible google searches or how Netflix uses its algorithms to predict what movies to suggest to suit every personal taste.

The potential behind Machine Learning algorithms and what they can achieve is limitless. This is the main factor that leads to studying and expanding the knowledge in data analysis and building Machine Learning algorithms to predict whatever a human mind could imagine. A simple Google search will prove that over the past five years Artificial Intelligence (AI) domain is one of the most popular domains that people are interested in Informatics and has made considerable improvements in many different ways.

1.2 Aims and Objectives

The main objective of this current thesis is to learn a relatively new and widely used programming language called Python and try to learn up to an extent, how to build Machine Learning model algorithms and predict with as high accuracy as possible Stock market prices. Besides building a proper Machine Learning algorithm, a considerable amount of time was spent to make its performance better for specific data types. So hyper tuning and optimizing the code was a big issue of this research leading to a certain point where the accuracy level of the model predictions was satisfactory. This project's more abstract goal is the making of a stock prediction bot that could use Machine Learning algorithms and trade stocks automatically with real money and try to make a profit, which is, of course, already implemented by other programmers. However, it would be the most reasonable cause of forecasting stocks. Ultimately by combining economics, Python's programming tools for data analysis, and Machine Learning Techniques, assisted to a great extent in building a decent code that aims to predict the Stock Prices.

1.3 Implementation

The methodology of the thesis was relatively simple and similar to most informatics projects. The first step was to find what programming language will be used to code, as there are plenty of different programming languages where a developer can code to build Machine Learning algorithms. The best choice was Python because it is a high-level and general-purpose programming language with enormous libraries for building Machine Learning algorithms. The next move was to find the Integrated Development Environment (IDE) to compose the code, which was also an easy task because Google Colab can give anyone the power to run massive programs that simple consumer computers could not handle. Also, Jupyter was a simple IDE that was helpful in the first steps of the research, as it allows the developer to execute the code line by line, learning this way the algorithm more comfortably and robustly. After choosing all the programming tools, the next step was to find what programming libraries will be used and therefore are necessary to build such a program. A small online search can easily lead to SciKit Learn and Keras libraries, which will be used for the current thesis models. Of course, there are many libraries to use, but these are mainly widely common, and for that reason, many programmers have already tried them. This fact was also taken into serious consideration because human feedback for optimizing and fixing the code could also assist this research. Having all the tools ready to do the desirable research, the need for the data has emerged. There are many stock data online in many formats, but the most common one has been used in this research, where both the stock’s Closing price and Date time were available, as those are the essential features. Yahoo-Finance was the primary source for using the data either in an online form or downloaded as a Comma Separated Values (CSV) file containing them. Now comes the most significant part of the project, hours upon hours of optimization, bug fixing, and tuning the model to a more accurate one with as much accuracy as possible. After the model provided stable and high accuracy predictions, it is time to analyze the data.

Preferably having an economic knowledge background would be ideal for interpreting the plots correctly and taking the most out of the predictions.

1.4 Contribution to Knowledge

Machine Learning as an Informatics domain is relatively new to the world and still has plenty of space to be improved, get more precise and efficient. Notably, in economics, Machine Learning algorithms have been present only in recent years, and still, there is much work to be done to say that reliable results have been achieved. Resulting from this thesis and its objectives, a unique code structure has been implemented and been merged from multiple other Machine Learning applications from other programmers. For example, StackOverflow is a well-known forum for programmers to meet and share ideas, bug fixing techniques, and coding examples. Using this free online help from several programmers worldwide leads to making a unique code that does an essential prediction task with its features. By testing three primary Machine Learning models, Linear Regression, Decision Tree Regressor, and RNNs (specifically LSTMs), this research concluded that the last-mentioned model works the best in terms of accuracy. If the developer-stock trader wants a high accuracy model, then a Deep Learning model is the go-to; if not, then fast-made and straightforward algorithms such as Linear Regression and Decision Tree, if used correctly, can be useful to some extent. Another observation that was made is that for every specific stock price, the model must be changed. For example, having a NN model working fine for Apple’s stocks does not necessarily mean that it will have similarly high accuracy predictions with Tesla’s stocks.

1.5 Structure

In the following chapter, the literature review of this thesis will be mentioned. Along with this review, notable mentions for the online sources and courses used will be made. Additionally, there will be referred to the already existing stock prediction models and the reason why Machine Learning techniques were chosen for predicting the Stock Prices.

The sequent chapter can be described as one of the main chapters, as it describes the Machine Learning field as brief as possible. It contains an overview of Machine Learning types and the reasons behind Machine Learning choice in general as a tool. Some of the essential types of algorithms will also be analyzed more thoroughly. The union of economics and Machine Learning is the topic of this research, and it unfolds in this chapter. Moreover, Deep Learning cannot be absent since it is a crucial sub-chapter to Machine Learning. Examples and features of Deep Learning will be mentioned to show some difference from direct Machine Learning models.

Following up with an introduction to Python language and its attributes in detail alongside why someone should choose Python programming language over plenty of others. Most of its advantages are mentioned and why many people select such a programming language for Machine Learning tasks. Also, there will be a discussion about the chosen IDEs and their importance in completing this project with the also indispensable use of Google Colab service. Along with the services and tools mentioned, there will be a closer look at Python's two main libraries for Machine Learning tasks, Keras and SciKit Learn.

Chapter five is where all the project unfolds. Explicatory sub-chapters for Linear Regression, Decision Tree, and Recurrent Networks show the use and mechanics of these Machine Learning methods in more detail. LSTM networks will be further analyzed as they are specially selected for this project. In conclusion, the implementations of this thesis will also be inserted in this chapter with some comments and clarifications.

The next chapter will be about the comparison and interpretation of the results. Step by step explanation of the outcome will be conducted, and after the comparison of the models, a decision of which implementation worked better and why will be made.

Closing chapters include a more general project review with a more comprehensive perspective of the Machine Learning algorithm results. For a smooth closure, future projects and prospects of this research and Machine Learning in economics, in general, will be discussed and be accompanied by some general thoughts about Machine Learning and some comments on the ethical part of this whole Machine Learning skyrocketing.

2 CHAPTER 2 – Literature Review

2.1 Introduction

In the past few years many books and papers have studied Machine Learning algorithms and their use in real-life situations. Starting off with this thesis, a literature review must be addressed, since finding useful resources, was the key of building up this research. In this chapter, all the books and the literature in general that were used as guidelines will be analyzed and summarized. Additionally, a mention to all the existing stock prediction approaches will be made, and the methodology that was chosen to make this research feasible.

2.2 Stock Prediction Common Approaches

Two main analytical approaches are common and described by Tsai and Hsiao (2010) in their research paper extensively. The first one is the Fundamental analysis. This analysis believes that every stock has its intrinsic value. The methodology is simple if the share prices are lower than its intrinsic value; it translates to the fact that the stock is currently undervalued. Hence, the stock trader should buy this stock. If the share's value is higher than the intrinsic, then the opposite transaction should be made. Fundamental analysis, in general, is the process of analyzing the information included in the financial statements. Such financial data are the balance sheets, the company's annual report, and income statements.

Furthermore, some widely used financial ratios for Stock Price predictions are the current ratio, return on assets, liabilities ratio, and more. Macroeconomics data and economic factors, in general, have a significant impact on this methodology and are many analysts consider them. The second methodology is Technical analysis, which is also known as "charting." Alongside Fundamental analysis, Technical analysis has been widely used for years by most analysts. Its primary process is studying a stock's historical price and volume movements by using charts to predict the price movements. So, searching for trends and patterns of an investment instrument's price, volume, breadth, and exchanging activities includes most of the relevant information needed for a stock trader to make a decision and determine its value. *(Tsai & Hsiao, 2010)*

2.3 Stock Prediction using Machine Learning

In this thesis, the selection of an LSTM network, which is considered by many researchers as the most successful RNN, was made in order to predict Stock Prices. Two methodologies are already mentioned about stock prediction. This research's main point is that these methodologies are human-made and pretty old-fashioned even though they are reliable in their predictions. As Machine Learning is rising over the years, taking the LSTM networks will help work with sequence prediction problems. Therefore, building a prediction model with primary NN layers, in addition to LSTM Layers, made an exceptional model with high accuracy. More details about the features of the model will be further mentioned later. Moreover, simple Linear Regression and Decision Tree Machine Learning models have been applied to the same datasets to compare the basic AI models with more complicated Deep Learning ones.

Some numerous researchers and developers have built countless models for such a prediction. As with every problem, there is not only one solution; with stock prediction, there are many methodologies, economic factors, and models to predict significantly. LSTM networks Decision Trees, and Linear Regression are not the only ways to build an AI model for stock prediction, but rather the methodologies chosen to be studied. Studying these methods make someone understand the difference between Artificial Neural Networks (ANN) and simple Machine Learning models.

2.4 Sources used for this research

Before starting to mention the literature used for the theoretical part of this project, the sources that assisted in the making of the coding and building of the models must be addressed. First of all, Udemy was the site that was used for learning the fundamentals of Machine Learning and Python. Mainly, the course “Python for Data Science and Machine Learning Bootcamp” played a significant role in doing the project. Furthermore, as every developer in every informatic field uses the same tool, google search, and especially StackOverflow and GitHub were the main two sites that were used for code comparison, optimizing, and bug-fixing. There are countless occasions where StackOverflow was the key-solution for a code error. There are many examples online or inside papers and books to get aid from, although building the project, much trial-and-error as a method was used along with the brute-forcing method for optimizing the code.

When it comes to the theoretical and explanatory part of the thesis, there are several books and papers mentioned and used throughout. Some of them are; "Introduction to Machine Learning with Python," written by Andreas C. Müller and Sarah Guido, "Hands-On Machine Learning with Scikit-Learn 7 TensorFlow" written by Aurélien Géron, and 'Machine Learning in Action" by Peter Harrington. These books were the source of many examples and facts that are used inside this thesis. These books had a large amount of information needed to comprehend the Machine Learning and Python merge to a better extent. The paper "Various Frameworks and Libraries of Machine Learning and Deep Learning: A Survey" from Zhaobin Wang et al. was an interesting article for the frameworks and libraries chosen for the implementation. In the economic part of the thesis, "Decision Support Systems" from Chih-Fong Tsai and Yu-Chieh Hsiao played an essential role in understanding the junction between economics and AI algorithms.

2.5 In Conclusion

Concluding this chapter, the general literature review of this research was analyzed and explained. It started concerning the already existing stock predictions methods without the assistance of AI algorithms, along with the chosen implementation. This implementation, of course, includes Machine Learning techniques and programming. A small overview of the books and papers used as a source of the theoretical part was also mentioned. Supplementary mentions of sites and online courses were mentioned due to their significant importance in the Machine Learning's first steps journey.

3 CHAPTER 3 – Machine Learning

3.1 Introduction

In this chapter, the field of Machine Learning will be addressed, which is the main subject of this thesis. Machine Learning is all about analyzing big data that are humanly impossible to do and then extract every single drop of knowledge. Machine Learning combines statistics, artificial intelligence, and computer science to calculate using complicated methods and predict these data outcomes. As it was mentioned earlier, the application of Machine Learning algorithms in our lives is getting more significant. From automated cars that can predict the road obstacles or human reactions on the go, to what food to order or which clothes or shoes to buy. Machine Learning also recognizes faces in photos, and categorize them, or suggest them. Many companies have already applied such algorithms to their products. Take as an example, Facebook, Amazon, Netflix, or Uber that almost every service that they offer contains numerous Machine Learning models.

Besides commercial applications, Machine Learning significantly impacted scientific problems such as understanding stars, finding distant Earth-like planets, discovering new particles, analyzing DNAs, and providing personalized cancer treatments. *(Müller & Guido, 2016)*

3.2 About Machine Learning

When most people hear the phrase "Machine Learning," they picture a more intelligent robot than humans that will ultimately conquer the world. However, this is not true but rather a fantasy. Machine Learning has been in our lives for quite a few years in some specialized applications, such as Image recognition that identifies characters, images, or the spam filters that have been around since the 1990s that helped flag unwanted emails (Figure 3.1).

Machine Learning, as an abstract definition, is the science of programming computers so they can learn from data. For example, our email spam filter can learn to flag a spam email, given examples of spam emails and nonspam emails. Alternatively, Machine Learning algorithms learn our preferences in YouTube videos and suggest them in the news feed.

In the field of data science, a standard algorithm consists of a sequence of processing steps. In Machine Learning, algorithms are built to find patterns and sequences among massive amounts of data and make decisions based on what the programmer wants to have as an outcome. The more data the algorithm has to "learn from," the more accurate it becomes.

The procedure of building a Machine Learning algorithm for a specific task is similar in every case. First of all, the programmer has to study the problem and gather the data that the algorithm needs to process. These are the most critical parts of the model because if they are not on point, the model will have solutions that might be far from what the programmer intended to be. So, cleaning up the data and setting clear goals for the program helps build the algorithm's foundations and makes it more robust and more apparent to its purpose. After the data collection has been completed, the part of training the Machine Learning algorithm comes. With current programming libraries, it is relatively simple and not as much complicated as someone would think. The program's dataset is called the training set, and every single data is called training sample. After the program has learned from the data, it is time to validate new data that the algorithm has never seen before and come to a solution. When the training has been completed comes the evaluation of the solution and the analysis of the errors. If the solution is close to the predictions or the desired outcome, usually measured with accuracy error metrics, the programmer is ready to launch it. If the solution had many errors or was far from the expectations, a repeat procedure of analyzing the errors must be done. The programmer must then re-write the algorithm correcting the data, the Machine Learning model parameters, and whatever else is needed until the achievement of the accuracy level that has been established at the beginning.

In order to build the model with as high accuracy as possible, it is crucial from the perspective of the programmer to understand which outcome is considered suitable—for example, building a Machine Learning model that can predict-classify with high accuracy whether the patient is ill or not has specific demands regarding the outcome of the algorithm that the programmer should take into consideration. The critical point of the whole algorithm is to be useful and precise. Classifying a patient as ill while being healthy is not a mistake that significantly impacts the patient's life because the patient will be flagged ill, then checked by a human doctor, and then realize that it was a program's mistake. Conversely, an ill patient classified as healthy is an error where the programmer should minimize because this patient will remain unchecked and has more chances of spreading the illness. In summarizing, the higher the accuracy the model has, the better it is, but not always. In specific cases like this, minimizing the possibility of letting an ill patient unchecked is more crucial than classifying a healthy patient as ill. These decisions are human-made only, and algorithms cannot distinguish between these two outcomes.

To summarize, Machine Learning helps with problems that require a lot of tuning and long lists of rules, while Machine Learning algorithms are somewhat smaller in rules and coding and, in most cases, perform better. Also, complex problems that are impossible to be solved by simple programs can only be solved by Machine Learning techniques. The flexibility of Machine Learning makes it easier than any other program to adapt to new data regarding the size and still perform outstandingly. *(Géron, 2017)*

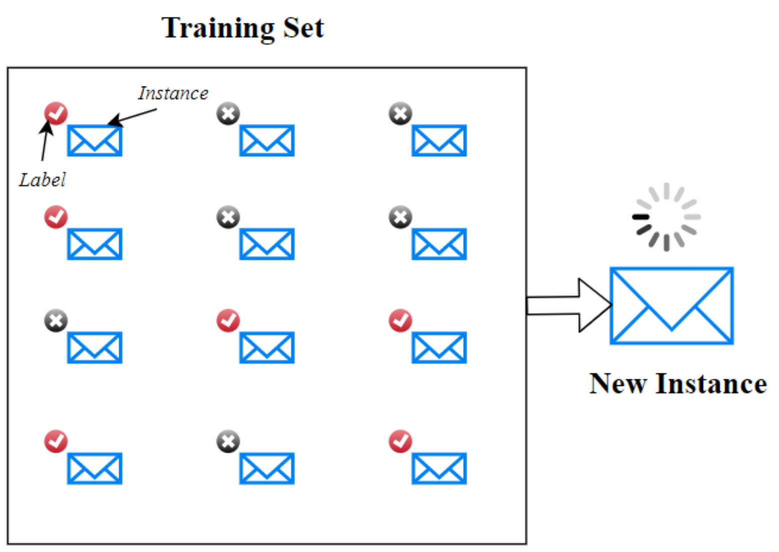

Figure 3.1 - Labeled training for email classification. ***(Source: Géron, 2017)***

3.3 Why use Machine Learning

Most reasons for choosing Machine Learning as an economic field approach have already been addressed at some point. However, the main reason is that Machine Learning is one of the biggest things to invest time and money in Informatics nowadays. A statement could be made that Machine Learning is the future and one of the leading "ingredients" in every service that is bought or used by people and, additionally, in every tool that is used by consumers (e.g., translation tools). Companies and programmers use Machine Learning algorithms even in things that human imagination cannot comprehend, such as smartphones that use these AI algorithms to interpret particular unique use of phones and ultimately make them more efficient in power and energy. Focusing on being "the next big thing," Machine Learning has a bright future in terms of upgrading, optimizing, and making it even more accessible for every programmer out there. So, a broad claim is that Machine Learning is in its "junior years."

During this period, a lesson was taught that major companies such as Google and Facebook have basically based all their products on those AI algorithms by monitoring their customers' usage. For many years this was unknown for the majority of people using products that included these algorithms. The emerge of General Data Protection Regulation (GDPR) across the world brought out from the dark these implementations showing us two things. First, online privacy should be more concerned with regulations and be protected, and secondly, how much significant impact these algorithms had in the real-world and needed to be controlled to some extent.

The raw truth is that any feature of Machine Learning or AI algorithm, helpful or controversial as it might be, is a one-way solution for major tasks-problems. Usually, the tasks with enormous data, complicated structures, and complicated solutions can be almost impossible for any human or even program to come to a proper solution. This is where all the Super-Computers (and not only) using their enormous computational power "shine" as they perform these types of codes because, frankly, they make human lives better in so many different ways (Figure 3.2).

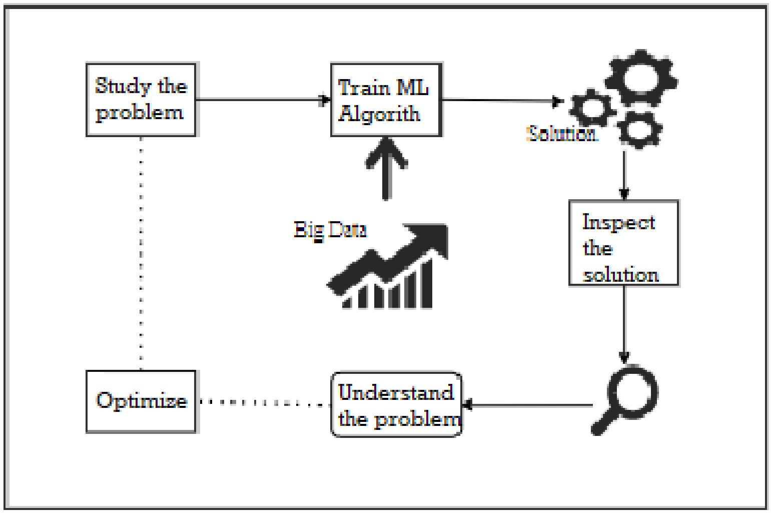

Figure 3.2 - Machine Learning can help humans learn *(Source: Géron, 2017)*.

3.4 Types of Machine Learning

There is a wide variety of Machine Learning systems that can be classified into three main categories. The first category is whether they are trained with human supervision (supervised, unsupervised, reinforcement learning). The second category is whether they can learn on-the-go (online or batch learning). Moreover, at last, there is a difference in whether they are predictive models by extracting patterns from data or compare new data points to already known data points (instance-based, model-based learning). These categories are not exclusive, and many combinations and sub-categories also exist. For example, as mentioned earlier in Chapter 3.2, a spam filter is a combination of an online, model-based, supervised learning system. Further analysis will occur to the three types used in this project, Supervised learning with batches of data on a model-based architecture *(Géron, 2017).*

3.4.1 Supervised Learning

Machine Learning algorithms can be sorted regarding the amount and the type of supervision they have during training. The most common types are Supervised, Unsupervised, Semisupervised, and Reinforcement Learning. During the Stock Prices Prediction of this research, Supervised Learning was used, which is the one that will be analyzed the most. From its definition, Supervised Learning means that the programmer feeds the model with data that include the desired solutions. It is also one of the most commonly used types of Machine Learning. Once more, the example of spam emails AI algorithm will be mentioned to understand better Supervised Learning. The prediction of whether the email is spam or not is the outcome of this task. At first, the model is fed with data that are usually in input/output pairs. The model is built from them by compromising the training set. In simpler words, the model is trained with emails that are already known to be spam and emails that are non-spam. In the end, by giving the model never-seen-before data, it makes accurate predictions. Usually, there is some effort to make the learning data efficient or “clean” for the model to work with, but this will not affect the model that will automate and speed up the process independently. There are two widely known methodologies used in implementing Supervised Machine Learning algorithms, Regression and Classification. This separation of emails is categorized as a classification methodology.

The Supervised Learning task of this thesis, the other most common methodology is used, called regression. An easy way to distinguish between these two types is to ask whether there is a continuity in the output, which in this specific case, prices that can take multiple values are being processed. In contrast, classification has distinct classes that the values must be inside them (e.g., whether this object is blue, red, green, or yellow; in simpler words, there is no in-between) *(Müller & Guido, 2016; Géron, 2017).*

3.4.2 Batch Learning

Another split of categories for Machine Learning algorithms is Batch and Online Learning. In Online Learning, the system is trained incrementally and continuously by continually giving the model small data batches. Usually, this method requires standard computational power. This research could efficiently work with this learning type by updating on-the-go about Stock Prices and building a solid model. However, the Batch Learning option is chosen, which means that the model was trained offline. Offline training can use more data than Online Learning and unavoidably more computational resources like CPU, RAM, HDD/SSD Storage, Networking. Except for computational resources, offline training is usually a massive and time-consuming procedure. The advantage of this approach is that it does not always need to be done frequently, and in many cases, a weekly or even monthly update of the model with new data is just fine. For example, whenever there are new types of spam emails with new features to recognize them, the programmer takes the model offline from use and re-train it old and new data adding this new "knowledge." When the reevaluation of spam filtering completes, it is time for the model to go online, continuing its purpose *(Géron, 2017).*

3.4.3 Model-based Learning

One of the examples that are used for predicting the stocks is Linear Regression. This particular algorithm is called model-based learning. Linear Regression uses the training data to find parameters and patterns and make the linear model fit best to the data; this is the training procedure. Linear Regression tries to fit most to the probably noisy data and apply its linear model to predict possible future values. For a use such as stock prediction, there is little linearity throughout the stock string. This effect happens because of the simple fact that Stocks cannot be described with a basic linear model and unavoidably see in future chapters that this model did not work as efficiently and precisely. More difficult hand-made models could be made for more precise prediction, but that complicates at an immense rate the procedure of trying to interpret the Stocking behavior *(Géron, 2017).*

3.5 Machine Learning on Economics

While Machine Learning techniques can be applied in every science field, a specific one will be focused on; economics. While someone could say that predicting the economy cannot be trusted on a computer might be true to some extent because of the physical factors that machines cannot comprehend and take into consideration. For example, a company's CEO gets sick, and instantly the shareholders start selling their shares in fear that the company will lose its integrity. Nevertheless, Machine Learning is a fantastic helping tool.

For many years, economists had used data sets as large as they could calculate in order to predict and make assumptions about product prices, employees' wages, and inflation. Machine Learning came in to play with enormous data sets compared to the ones that humans could manage. This fact leads to economists being able to test and try their models at an incredible speed. Also, Machine Learning techniques can help experts by allowing the design of systematic model-selection processes preventing them from choosing an inappropriate model. The combination of Machine Learning and econometrics can also help estimate more realistic and complex models due to these artificial networks' enormous computational capabilities.

The specific task that was selected for this thesis is stock prediction using Machine learning algorithms and NNs. Everyone would expect that no matter how good the model works, it cannot predict precisely the outcome in the future, especially in the world of the stock market where the only thing that happens is numbers plunging and rising with no specific lead explaining why. Without degrading the use of stock forecasting, many economists and companies repeatedly use robust models that give meaning to an endless string of values. Combining these AI algorithms with economic background, tipsters in the stock market, and of course, strong deterministic capabilities can make a "deadly composition" for a stock investor.

3.6 From Machine Learning to Deep Learning

Machine Learning is the general name for almost every AI algorithm that exists. One specific domain that has recently seen massive attention from many people around the world is Deep Learning. This is a sub-category of the Machine Learning genre and is commonly known as Neural Networks or NNs. Deep learning algorithms are often used in specific cases and show better prediction performance than other basic Machine Learning models. The most basic methods are Multilayer Perceptrons (MLPs) (Figure 3.3) for regression and classification that can also set the structure for more complicated learning models. It is widely common to describe MLPs as generalizations of linear models that include numerous stages of processing. The model consists of input features shown as nodes in Figure 3.3, and the weights are the connections between input, the hidden layers, and predictions. Computing the series of weighted sums has no mathematical difference than the sum of one weighted sum. So, after computing a weighted sum for each hidden unit, a nonlinear function is applied to the outcome (e.g., in the specific project, rectifying nonlinearity was used, also known as a rectified linear unit or RELU). What RELU does is that it cuts off values below zero and has been one of the latest and most effective activation functions. These nonlinear functions' importance allows the NN to learn much more complicated functions that a simple linear model could achieve *(Géron, 2017)*.

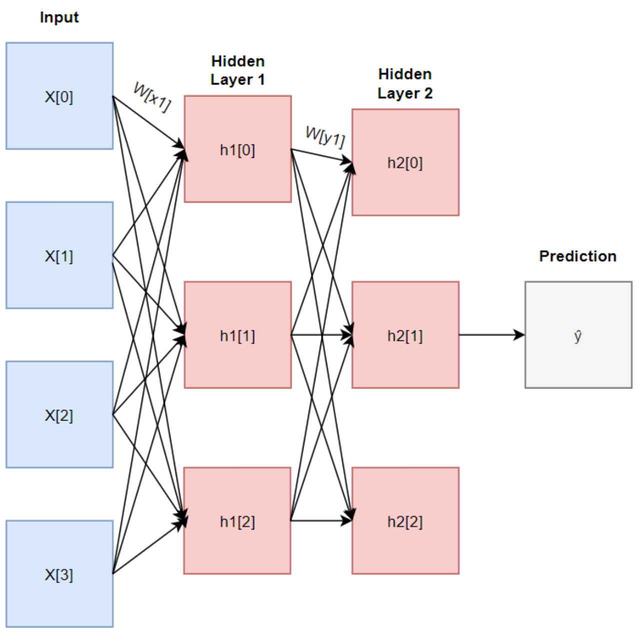

Figure 3.3 - A multilayer perceptron with two hidden layers *(Source: Müller & Guido, 2016)*.

3.7 In Conclusion

This chapter consists of a brief overview of Machine Learning and some of its features, while this is the main topic of this thesis. The advantages and the features that make Machine Learning distinguish between other non-AI algorithms were also mentioned as well as the reason why these algorithms are everywhere around us. Some types of Machine Learning were introduced to this chapter and precisely the types used in the project. Notable mention of the relevance between Machine Learning and economics was a vital topic to analyze since this thesis is about Stock Prices Forecasting. Finally, some features of NNs were shown, and Deep Learning models, specifically MLPs, were analyzed up to some extent.

4 CHAPTER 4 – Services and Tools

4.1 Introduction

This chapter contains information about the services and tools that were used throughout the making of the project. A mention of the languages that exist for making Machine Learning programs will be made and then explain Python's choice. Python also has some exciting libraries that can help the code to get more straightforward. Except for the standard libraries, Python's Machine Learning libraries and their possibilities are critical for the project, and they will be further analyzed. Additionally, the services that are chosen to compile and execute these algorithms will be explained thoroughly.

4.2 Programming Languages for Machine Learning

Searching for a suitable programming language for building a Machine Learning algorithm is not as simple as it seems. Four main programming languages can be found with a simple search for Machine Learning and Data Science. These programming languages such as R, C++, Python, and Java, are the programmers' most preferable choices, and in fact, this is not accidentally.

R language is considered one of the tops for data science applications. It has been around for some decades with an open-source community, making it a top choice for many Machine Learning programmers. On the contrary to most programming languages, R intends to use non-programmers or people with little programming experience, such as analysts, data miners, and more. The vital point is that it supports friendly IDEs like Jupyter and provides various tools to manage our libraries and draw complex graphs. In conclusion, R's most crucial advantage is that it is designed to handle data-processing easily compared to other programming languages. Hence, it is not as versatile as Python, for instance. It is an ideal language for data exploration and managing massive dataset with the data visualization-orientated language.

Java has also been around for quite some time, and programmers worldwide consider it one of the most useful and significant programming languages. Given that fact, Java is also a well-known programming language for building Machine Learning models. If a programmer wants to find a simple programming language to use, Java is not one of them for sure, while many users around the world have criticized for its complexity and the slow-paced coding.

C++ is, of course, a mainstream programming language. Due to that, companies and programmers are looking to add more Machine Learning to exist projects using C++ and C, respectively, making it very popular yet tricky. However, it is another versatile, powerful, and irritating language, similar to Java. Where C++ shines is its versatility while being a lower-level language than most Machine Learning languages, thus more straightforward for the computer to "read." Most programmers could agree that C++ is the best programming language for Machine Learning applications but not much preferable in learning Machine Learning.

4.2.1 Why Python?

One of the many programming languages that are common and widely used for Machine Learning applications is Python. It is undoubtedly the most used programming language these days, from data scientists to Machine Learning applications. Python, while being elegant, lightweight, and straightforward as a language yet is so powerful. It is so easy to build complicated programs with simple code lines that make Python stand out as a new trend for programmers, almost in every field of Informatics.

Python is a prevalent programming language at the moment, so many examples are available right away online, making it easy to learn and adapt to it fast. Except for the broad public, Python is popular in the scientific, financial, and statistic communities. Many high-level languages allow us to do matrix math, an essential operation for Machine Learning tasks, such as MATLAB. To use such a language, we would need a large amount of money to take the license to use it legally, so Python was a slippery slope. At the same time, high-level Python language aids programmers accomplish the task as efficiently and transparently as possible. Having all these advantages, Python must have drawbacks also, and it does. Python is not one of the best performance programming languages such as Java and C. Of course, this is for many a minor thing to consider, but for larger-scale projects is something to keep in mind. *(Harrington, 2012)*

4.2.2 Python Libraries

Python has some strong libraries that make data analysis seem relatively easy. Pandas, NumPy, and Matplotlib are the three most useful libraries included in the making of this project.

Starting with NumPy, one of the fundamental libraries for scientific computing and Machine Learning algorithms in Python. Its primary function is handling multidimensional arrays, high-level mathematical functions, and random number generators. SciKit Learn is a Machine Learning library, which was also selected for this project, that takes in data in the form of NumPy arrays, making NumPy's presence obligatory to the libraries list. Every data processed must be converted to a NumPy array before used. NumPy has a class named ndarray, an N-dimensional array with elements of the same type.

Pandas library was the solution for data handling and analysis. It is based on a data structure named DataFrame originating from R's programming language DataFrame. DataFrame can be described as a table similar to an Excel spreadsheet, supporting SQL-like queries and joins of tables. Pandas' DataFrame might seem similar to NumPy's array, but in reality, DataFrames can have multiple data types in the same column (e.g., integers, dates, strings). Another critical feature of the Pandas library is that it can insert or "read" many file formats and databases, like SQL, Excel files, and CSVs.

When plotting for the results comes, Matplotlib is an easy choice for the coding. It includes functions for making visualizations such as histograms, charts, and scatter plots. Inside Jupyter Notebook, Matplotlib can immediately show the figures using the *%matplotlib inline* command, making it directly accessible to the viewer. Matplotlib is not the only library existing in Python for figures; Seaborn is another notable one with more sophisticated figures. Nevertheless, Matplotlib is used by the majority of programmers, and it is relatively easy. Notable mention of pyplot's iplot function must be made because it provides an interactive figure for a better view of the figures' information. This interactive plot could be handy for presentations, and in some cases, it can be used to scrutinize complicated plots.

4.2.3 Python Tools and Frameworks

Anaconda is a free and open-source distribution of the Python programming language that most data scientists, analysts, and Machine Learning programmers use and prefer. It comes with a considerable number of packages automatically installed and much more available to add afterward. Anaconda's advantage is that it has "conda" package manager that analyses the current environment, including what is already installed, and all together, it finds how to install a compatible set of dependencies to make them all operational. Such a feature is quite helpful for beginners with a small amount of experience.

Inside Anaconda's Navigator GUI (Graphical User Interface) multiple applications are available by default, such as Jupyter Notebook, Spyder, RStudio, and more. The application that was most used for this research was Jupyter. Jupyter Notebook is an interactive environment for running code in the local browser of the machine. This particular integrated development environment (IDE) was chosen because of its ease of use. It is an excellent tool for data analysis, and it supports line-by-line execution for better comprehension of the programming language and code handling in general. One extra feature of Jupyter is that it makes it easy to conflate code, text, figures, and images all in one, making it readable and easy for someone else to start reading it from scratch.

A famous framework that is a top choice for many Machine Learning projects is Google's second-generation open-source artificial intelligence learning system named TensorFlow. Tensor is an N-dimensional array, and Flow means that the calculation is based on the data flow diagram. In general, it is a system that delivers complex data structures to AI NNs for analysis and processing. A computation expressed using TensorFlow can be executed with almost no differentiation in almost every platform ranging from computers to phones, tablets, and even thousands of computational devices such as CPUs and GPUs, respectively. TensorFlow can support many applications, but it mainly focuses on deep NNs and serves as a research platform for Machine Learning systems. Another notable mention is that TensorFlow has a large community that can help a newcomer learn to use this framework effectively. In this research, TensorFlow was used for the backend to the Keras API (Application Programming Interface) and was not used solely for the making of the code. The reason that TensorFlow was not selected is that it is a low-level library for NNs. A partial difficulty using TensorFlow was to avoid since there was no need to make low-level changes when building the model. Ultimately a high-level API was selected named Keras. *(Wang et al., 2019)*

4.3 Services

As mentioned earlier, Machine Learning algorithms in order to be executed, they need a significant amount of computational power. Such power might not be accessible by every programmer who wants to build an AI algorithm. This is where cloud services come to assist in performance. FloydHub and Google Colab are cloud services that provide the user with and IDE, in both cases, Jupyter, and the choice of CPU or GPU acceleration. In most cases, these services might be the only thing that a programmer needs to start building a model. This research was based on Google Colab because it is entirely free, even using GPU acceleration for faster results. Moreover, made by trustworthy Google gives the programmer the consistency that is needed. With the combination of the ability to share the code instantly using Google Drive, a massive project among many people can be implemented.

4.3.1 Google Colab

Google Collaboratory is a product from Google Research that was publicly released in 2017. As mentioned earlier, it is a cloud-based IDE for python to enable Machine Learning with storage on the cloud. Its use is free as long the user has a Google account. An unlimited amount of computational power, such as multiple GPUs, is, of course, for the shake of everyone limited to consumers. If anyone might need such enormous power, it can be enabled by acquiring a Colab Pro account. Based on Jupyter, Google Colab has the most common dependencies, such as already mentioned, NumPy, Pandas, and even TensorFlow pre-installed. There are many small features to consider, such as the vital, in most cases, the built-in "Search Stack Overflow" button that pops up whenever an error occurs. Considering the drawbacks of this service, while using the Colab, the programmer cannot avoid storing the notebooks in Google Drive, which some people might find as a downside. Another drawback is that Colab interrupts long-running background computations after a relatively short period (e.g., 12 hours if a GPU is used). Building and training massive Deep Learning models that require a significant amount of time to execute are suggested to run locally. As datasets get bigger than ever, Google Drive's 15GB of free space might seem a little space to work with, eventually buying more space being the solution, increasing the project's cost.

4.4 Machine Learning Libraries of Python

Additional to the basic but rather essential Python libraries, some libraries enable the programmer to build a Machine Learning model from scratch. All libraries mentioned earlier have strong popularity among programmers and multiple online sources with plenty of examples. Pytorch is a Python package with GPU-accelerated tensor calculation and the ability to build a unique dynamic NN. Theano is another Deep Learning library that has been around since 2007. According to Theano's official introduction, Theano is a low-level Python library similar to TensorFlow that allows us to define, optimize, and evaluate mathematical expressions, including multidimensional arrays, using CPU or GPU acceleration. Anyhow, Pytorch, Theano, TensorFlow, and Keras are altogether Machine Learning libraries that support GPU acceleration and Deep Learning or NNs implementations. SciKit Learn, on the contrary, is a more conservative library that will be analyzed further in the next sub-chapter. SciKit Learn was first chosen for its simplicity and forwardness of making simple regression Machine Learning algorithms before diving deeper into Deep Learning models. Keras' choice came as a Deep Learning, and GPU-acceleration features were available to make more complicated NNs for Stock Price Forecasting. In Table 4.1, the most widely Machine Learning libraries are mentioned along with their popularity in GitHub *(Wang et al., 2019)*.

Table 4.1 - Deep Learning frameworks in GitHub statistics *(Source: Wang et al., 2019)*.

Framework	*Stars*	*Forks*	*Contributors*
Caffe	15.057	9338	222
Keras	**10.875**	**10.875**	**327**
MX Net	7471	2764	250
Torch	6163	1793	113
Deeplearning4j	5090	1970	103
TensorFlow	**4505**	**667**	**573**
Chainer	1983	512	96
CNTK	9063	2144	100
Theano	5352	1868	271
Lasagne	2749	761	55
Neon	2633	573	52

4.4.1 SciKit Learn

SciKit Learn was first made public by data scientist David Cournapeau in 2007 as a more inclined framework to use Python as part of the API interface. Other dependencies such as NumPy or SciPy are open-source libraries explicitly designed for Machine Learning use. Like most open-source projects, SciKit Learn is currently maintained by programmers that use the framework due to conservation costs. The primary functions of SciKit Learn are divided into classification, regression, clustering, dimensionality reduction, model selection, and data preprocessing. As mentioned earlier, SciKit Learn cannot build ANNs, Deep Learning models, or large-scale projects because it lacks GPU-acceleration, making it hard to count solely on CPU performance. SciKit Learn shines in its regression algorithm that covers most developers while providing precise and helpful use case references. Comparing to other libraries, this particular is easy to use, and the environment is equally effortless to build almost all the mainstream algorithms for Machine Learning. Another drawback is that it has low-flexibility and is usually suitable for dealing with medium to small-sized datasets only. In conclusion, SciKit Learn is an instantaneous library, easy to use, and applies in most Machine Learning algorithms; thus, many developers characterize it as an entry-level Machine Learning framework for beginners to learn the process of building AI algorithms (Wang et al., 2019).

4.4.2 Keras

Keras is one of the most used Deep Learning frameworks and has TensorFlow to its backend. Keras is an industry-strength framework that can scale to large clusters of GPUs or even an entire Tensor Processing Unit (TPU) pod. This framework is concentrated on defining layers for NNs while not having to deal with tensors. It is easy to handle the library since it is a high-level API. In general, Keras is like a wrapper to TensorFlow, making it useful for the making of instant implementations of AI algorithms. It uses the programming language of Python to operate, which makes it even more user-friendly. Searching in GitHub or StackOverflow, anyone can find that Keras has many contributors and developers who continuously share ideas and codes. The choice of Keras for building NNs was made by taking into consideration the ease of use while being strong in terms of performance.

4.5 In Conclusion

This chapter was all about introducing the “Services and Tools” for a developer to make a Machine Learning project. All the existing programming languages for building an AI algorithm were mentioned and the choice of this research. The choice of Python was also explained and compared to other programming languages for a clear view. Libraries and tools that are essential and used for the stock prediction project were mentioned and analyzed. Another notable mention was Google Colab’s contribution to the computational power needed to make this project feasible. Closing the chapter, Python’s Machine Learning libraries were mentioned, especially the chosen ones, Keras and SciKit Learn, with their advantages and unique features for assisting the developer.

5 CHAPTER 5 – Machine Learning Implementations of this Thesis

5.1 Introduction

In this particular chapter, the implementations that were used for doing this thesis will be addressed. An introduction and a general view of Linear Regression, Decision Tree Regressor, and Recurrent Neural Networks will be presented. A specific category of Recurrent Neural Networks will also be analyzed, LSTM models since it was the RNN choice for this research, and why such an RNN was chosen and what it has to offer to stock prediction. After the introduction, all the implementations and their results will be viewed and explained.

5.2 Linear Regression

Linear models are widely used in practice, and many pieces of research have been done in past decades. With the use of a linear function, linear models make a prediction. For regression, the general formula for a linear model as mentioned by the researchers Müller and Guido looks as follows:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b \; (1)$$

In the equation, $x[0]$ to $x[p]$ indicates the features (the particular example the number of features is p) of a single data point. W and b are the model's parameters that are learned through the training process. The character $\hat{y}$ is for the prediction of the model. For a single feature dataset, the equation (1) takes this form:

$$\hat{y} = w[0] * x[0] + b \; (2)$$

This equation (2) can be linked with the equation for a line where $w[0]$ is the line's slope, and b is the y-axis offset. Linear Regression is the simplest yet the most common linear method for regression used by many researchers and developers in countless applications. What this method does is that it finds the parameters w and b that accumulate the mean squared error (MSE) between prediction's values) and the actual regression values y that are included in the training set. The MSE is a sum of the squared differences between the actual values and the model's predictions. Linear Regression has a limited complexity since it has no parameters. This characteristic of Linear Regression could be described as an advantage but not in every case. A simple example of Linear Regression application can be seen in Figure 5.1, where the

Linear Regression tries to fit the best possible line between the data points *(Müller & Guido, 2016).*

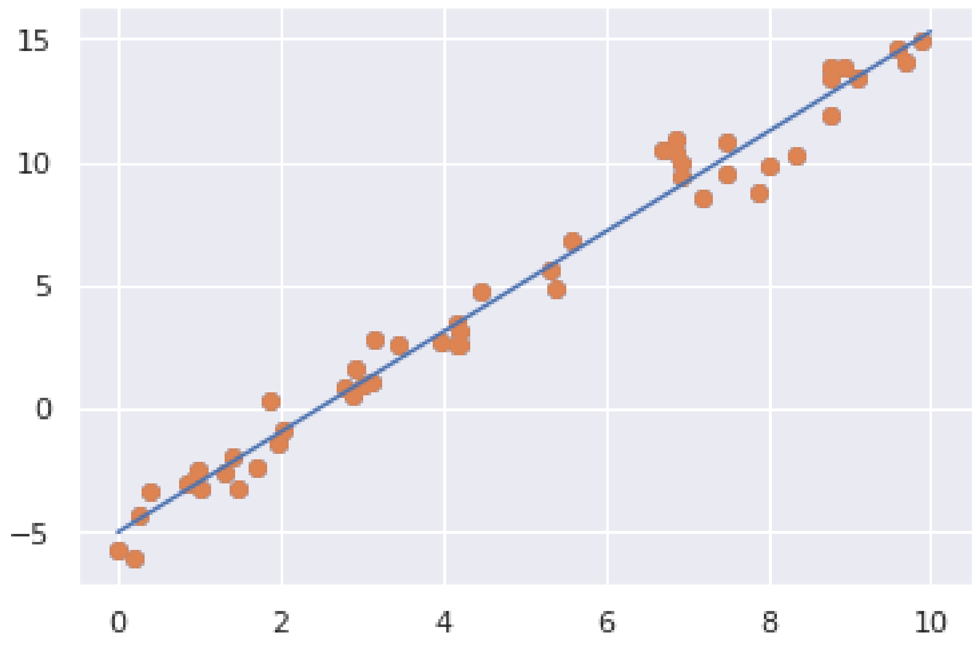

Figure 5.1 - Linear Regression fit.

5.2.1 The Implementation

The complete coding for the Linear Regression implementation will be deployed for better convenience in Appendix A section. So, an overview of the code will be addressed for better understanding. First of all, in order for the program to operate, the essential and useful libraries are imported. NumPy and Pandas are installed for the data analysis part of the project. Afterward, all necessary imports from the SciKit Learn library are also installed for the Linear Regression models and all the error metrics needed. Matplotlib and Plotly are included for the making of the plots. Since all libraries are installed, the data must be inserted into the program. The CSV files for individual companies are downloaded from Yahoo Finance and include all the necessary data for the prediction model; specific statistic and numeric features are available in Table 5.1. The "Close" column in the particular example includes the last price at which a stock trades during a regular trading session. Before starting to use it, such a value has to be converted to float type. Afterward, a new column is created and shifted "F" days forward, including the targeted "Close" prices. "F" translates to the number of days that want the model to predict. The data is then split to feature data set “X,” and the target data set “Y” after being converted to NumPy arrays. A second split of the data into 75% percent of training data points and 25% of testing points must be executed before creating the model. As mentioned, with few code lines, the model of Linear Regression is created with inputs of the training parts of the “X” and “Y” dataset. After the model has been created, its predictions are plotted, as seen in Figure 5.2, compared to the actual values. (Note: the implementation was done with a total of 256 stock price points of Netflix shares)

Table 5.1 - Information about Netflix CSV file

	Open	**High**	**Low**	**Close**	**Volume**
count	254.000000	254.000000	254.000000	254.000000	2.540000E+02
mean	371.885157	378.668819	365.765336	372.772992	7.685473E+06
std	76.348062	77.895003	74.679086	76.540696	4.509276E+06
min	255.710007	265.000000	252.279999	254.589996	2.019300E+06
25%	300.890007	305.680007	295.007492	299.822495	4.920875E+06
50%	363.539994	370.944992	356.449997	363.534988	6.447200E+06
75%	435.560005	441.442497	427.247490	434.425011	8.762150E+06
max	567.97998	575.369995	520.960022	548.729980	3.825890E+07

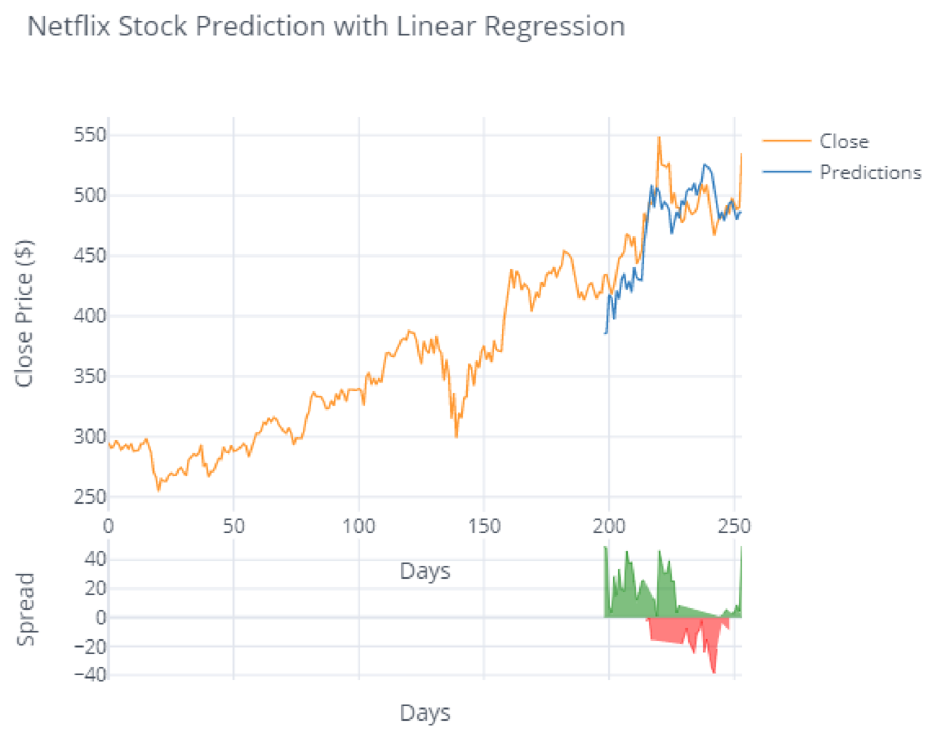

Figure 5.2 - Linear Regression, Predictions & Residuals.

A specific code was also used to determine the “sweet spot” for the number of days in the future that the model predicts with higher accuracy, as seen in Figure 5.3 and Figure 5.4, respectively. It is evident that the R^2 score is more consistent and immensely better after 50 days. In addition to the MAE results, they show a “window” of 15-20 days between 40- and 60-days periods that the model has significantly less error. By considering these errors, an assumption that the model will predict with higher accuracy for 50-60 days in the future.

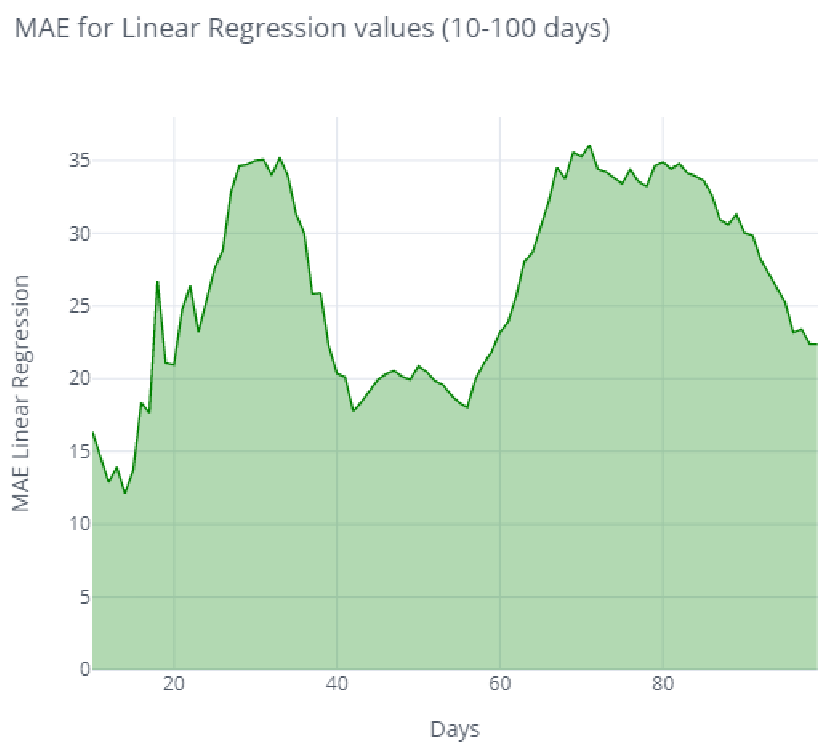

Figure 5.3 - Mean Absolut Error for future days set (Linear Regression).

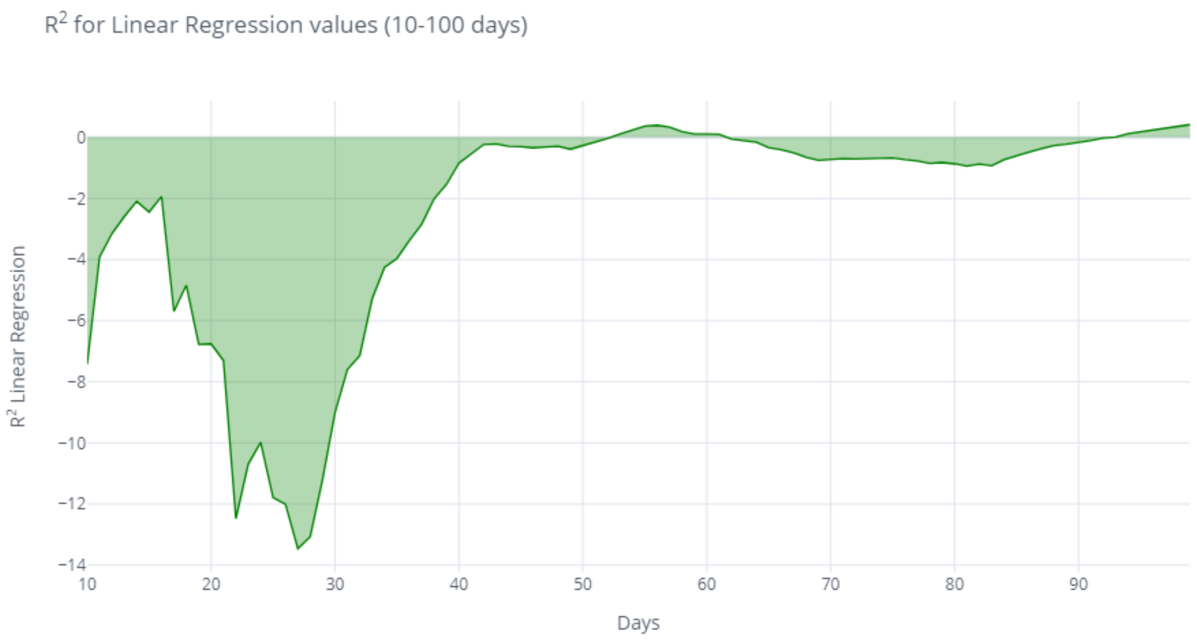

Figure 5.4 - R^2 score for future days set (Linear Regression).

5.3 Decision Tree

Decision Trees are widely used for classification and regression problems. They have a hierarchy of if/else questions leading to a decision. It might sound effortless as a task, and it is. All is needed is an example of how a Decision Tree works, and everyone without any knowledge of Machine Learning or even mathematics will understand. Imagine a task where the developer wants to distinguish or even select one specific human among many. Every human has its characteristics, how tall he/she is, how much bodyweight, facial characteristics, hair color, and many more. These are like attributes of the human, making each person look like a table of data. Decision Tree works precisely as it says in its name, by making decisions in a tree/hierarchical order. For example, the developer wants to narrow down to a single person by asking questions about their appearance and characteristics (e.g., is he/she blond/e?). The answers to those questions are binary (yes or no). Question after question, from all the data available, the developer makes accurate questions and eliminates unwanted results coming ultimately at the desired output. In Figure 5.5, such an example is presented.

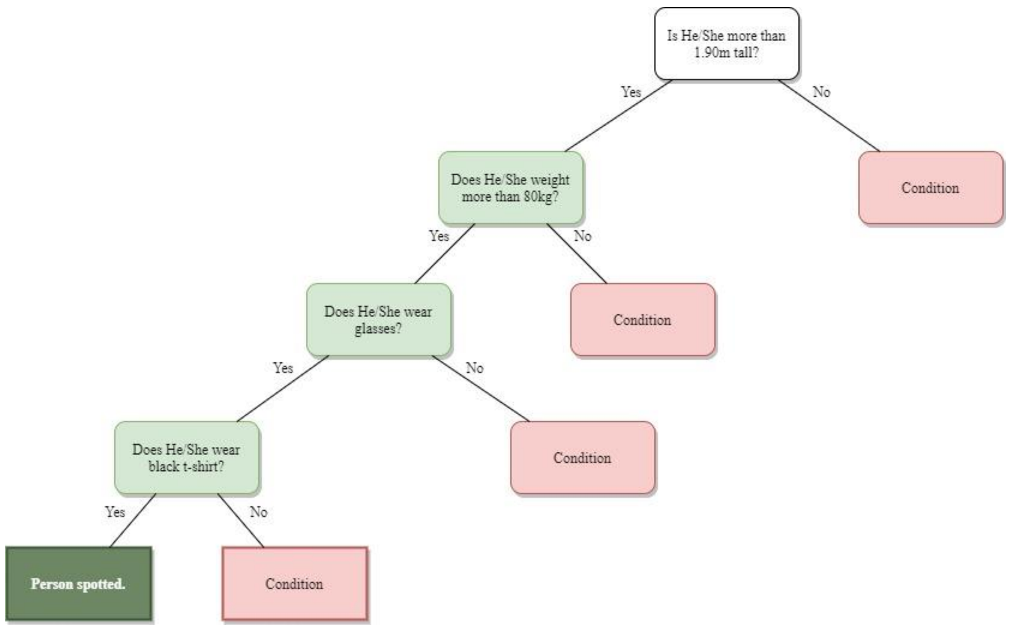
Is He/She more than 1.90m tall?
Yes
No
Does He/She weight more than 80kg?
Condition
Yes
No
Does He/She wear glasses?
Condition
Yes
No
Does He/She wear black t-shirt?
Condition
Yes
No
Person spotted.
Condition

Figure 5.5 - A Decision Tree for finding a person.

Furthermore, these questions help the model classify and categorize the data; for example, “Does this animal fly?” If yes, then it is classified as a bird. This example could be named as a Classification Tree problem. In the Regression Tree, similar architecture is executed in the background. This analysis is when the predicted outcome can be considered a real number (e.g., the price of a house or a Stock price).

5.3.1 The Implementation

Regarding the implementation of the Decision Tree, the procedure of building the program is precisely identical to the Linear Regression one. Appendix B includes the code for the Decision Tree Regressor implementation for further understanding. Since chapter 5.2.1 explained the program's data analysis, there is no need to repeat the same steps. After the data is ready, the Decision Tree model is built and plotted, as seen in Figure 5.6. The same data is used; every detail of the data can be seen once more in Table 5.1. (Note: the implementation was done with a total of 256 stock price points of Netflix shares)

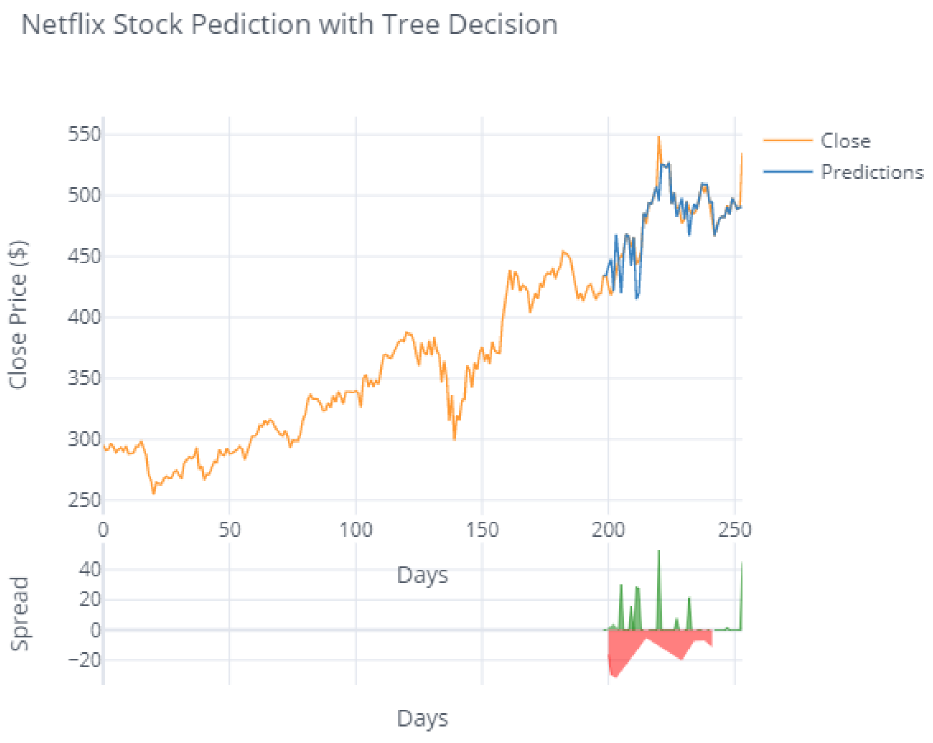

Figure 5.6 - Tree Decision, Predictions & Residuals.

Similar to Linear Regression, a specific code was also used to determine the “sweet spot” for the number of days in the future that the model predicts with higher accuracy, as seen in Figure

5.7 and Figure 5.8, respectively. Once more, the R^2 score is more consistent and immensely better after 50 days. In addition to the MAE results, despite having more "noisy data" compared to Linear Regression MAE errors, they also show a "window" of 15-20 days between 40- and 60-days periods that the model has significantly less error. By considering these errors, an assumption that the model will predict with higher accuracy for 50-60 days in the future.

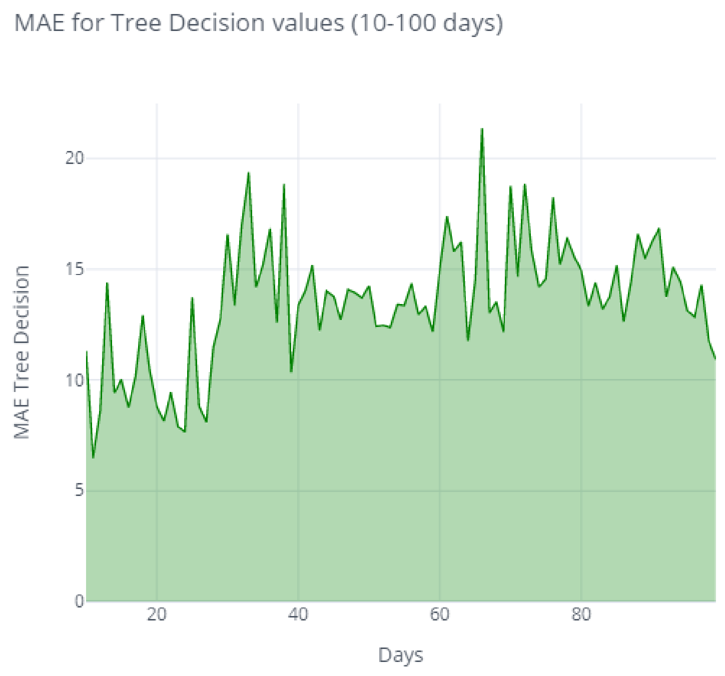

Figure 5.7 - Mean Absolut Error for future days set (Tree Decision).

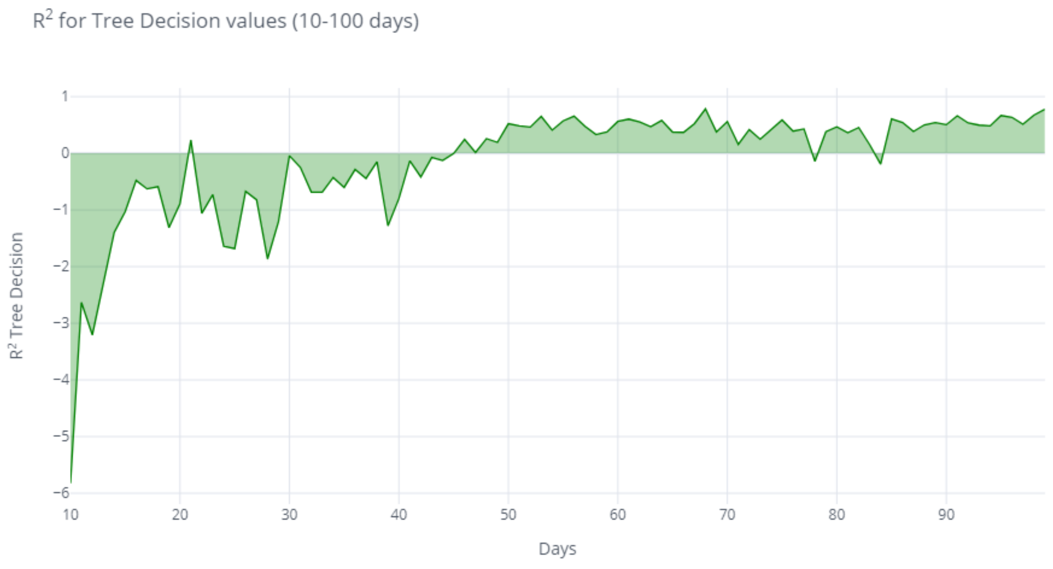

Figure 5.8 - R^2 for future days set (Tree Decision).

5.4 Recurrent Neural Network

Humans are used to predicting the future, from when the car is taking a turn, and the driver automatically steers the driving wheel to when a student makes coffee and anticipates its smell. Recurrent Neural Networks or RNN, for short, is a class of nets that can predict the future up to a point, of course. It is suitable to analyze time-series data such as Stock Prices and eventually suggest when to buy or sell a share. This is the main reason why RNNs were chosen for implementing such an algorithm in the current thesis. Additionally, to analyzing time series, RNNs are useful for natural language processing systems (NLP) such as automatic translation, speech-to-text converter, or even sentiment analysis. Another capability of RNN is that it can generate sentences, images (e.g., project DRAW), and even melodies (e.g., Google's Magenta project), giving it the ability to have surprising creativity. Plain NNs have already mentioned that are feedforward, and the activations flow only in one direction, from the input layer to the output. RNN is almost identical to a simple NN with the differentiator with connections pointing backward from output to the input layer, as seen in Figure 5.9. At each iteration, over time, the RNN is fed with the input (x) and the outcome from the previous step (ŷ). This procedure, along with the activation forms already mentioned (e.g., activation of RELU), make RNN a strong Machine Learning technique for predictions *(Géron, 2017)*.

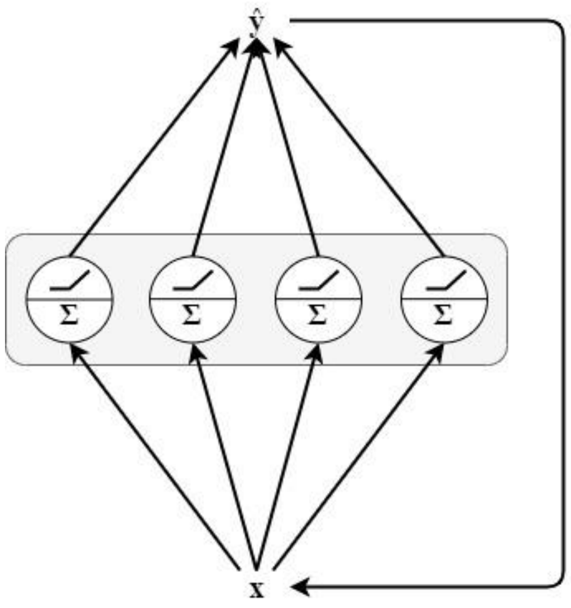

Figure 5.9 - A layer of a recurrent neurons *(Source: Géron, 2017)*.

5.4.1 Long Short-Term Memory

The main problem that RNNs face is that they have vanishing or exploding gradients. This is where Long Short-Term Memory (LSTM) comes to fight this issue. This so-called cell was first proposed in the distant 1997 has been gradually improved over the years by several researchers. An LSTM is compared to a black box and can be used as a bare cell except for having better performance. The training procedure of an LSTM layer is faster than other archetypes, and it will also detect long-term dependencies in the data. Such a feature is a critical factor for time series like stock predictions. In simple words, an LSTM cell can learn to identify essential data points in the input layer, store it in the long-term state, learn to preserve it for as long as needed, and ultimately learn when to extract it. This procedure makes LSTM amazingly successful at identifying long-term motives in time series, massive texts, and audio recordings. The magic that happens inside an LSTM cell is shown in the Figure 5.10 where h(t) stands for the short-term state and c(t) stand as the long-term state *(Géron, 2017)*.

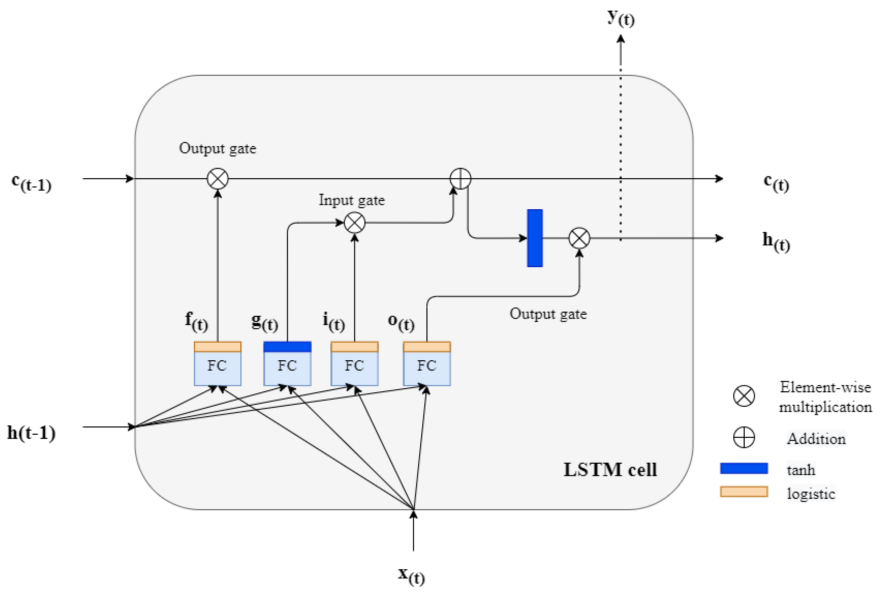

Figure 5.10 - LSTM cell (Source: ***Géron, 2017*****).**

5.4.2 The Implementation

Before starting building the code, all the necessary libraries must be installed; the most important ones are NumPy, Pandas, matplotlib, TensorFlow, Keras, and the error metrics from SciKit Learn library. Further details about which imports have been used can be seen in Appendix C part where the whole program lies. In order to gather the data, a different approach was used since the whole program was launched through Google Colab; that approach was the web.Datareader (reading the data online and not from a CSV file). The source of the data was once more Yahoo Finance. The column that is needed for the model to predict is the “Close price,” which was the only column selected from the dataset. After converting it to a NumPy array, all the values were normalized from 0 to 1. This helps the model adjust better to the values and eliminates some occasional significant differences between prices that can interfere with the model. Once the dataset is scaled, it is split between the “X” train and “Y” train datasets. “X” train and “Y” train datasets are also converted to NumPy arrays, and the “X” train is reshaped to a 3D dataset since the LSTM demands a 3D input. The same procedure happens for the test data that will be used later on.

Upon finishing with fixing and splitting the data, it is time for the model to take a form. The model is Sequential and consists of four hidden layers. The first two are LSTM layers with fifty neurons, respectively, while the second one has the RELU activation function. The other two hidden layers are straight Dense layers, with the first one having twenty-five neurons and the second one having one, and both are having RELU activation functions. Also, a dropout function is called between layers to make the training harder and, therefore, significantly better. After the model is built, it is compiled with the “adam” as an optimizer and the pair of MSE and RMSE (Root Mean Squared Error) for errors. For the training process, a batch size of two-hundred was chosen with ten-percent validation split in a total of fifty epochs of training. After the train, the test data is used as input for the prediction of the stock prices. The reverse of the scaling to the data comes after the prediction and must not be forgotten. Having the data and the predictions in their original form, they are plotted and compared, as seen in Figure 5.11 and with a closer view of the residuals in Figure 5.12. (Note: the implementation was done with a total of 2006 stock price points of Apple shares)

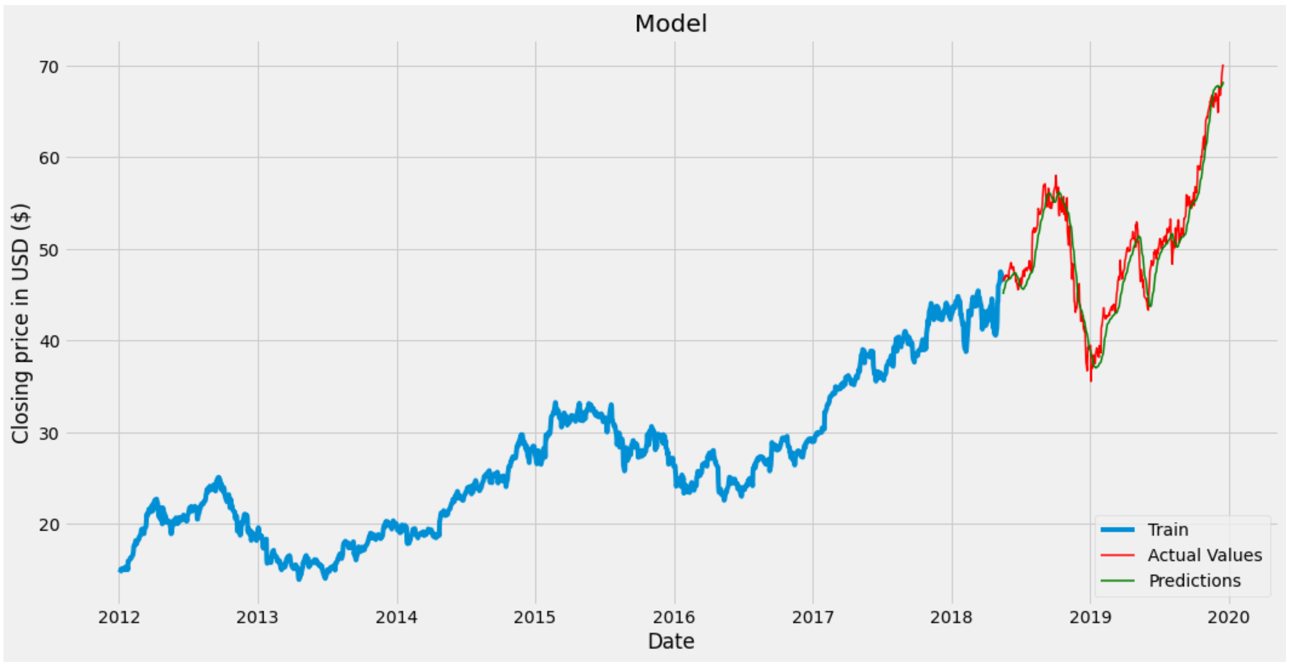

Figure 5.11 - LSTM Stock Price Predictions.

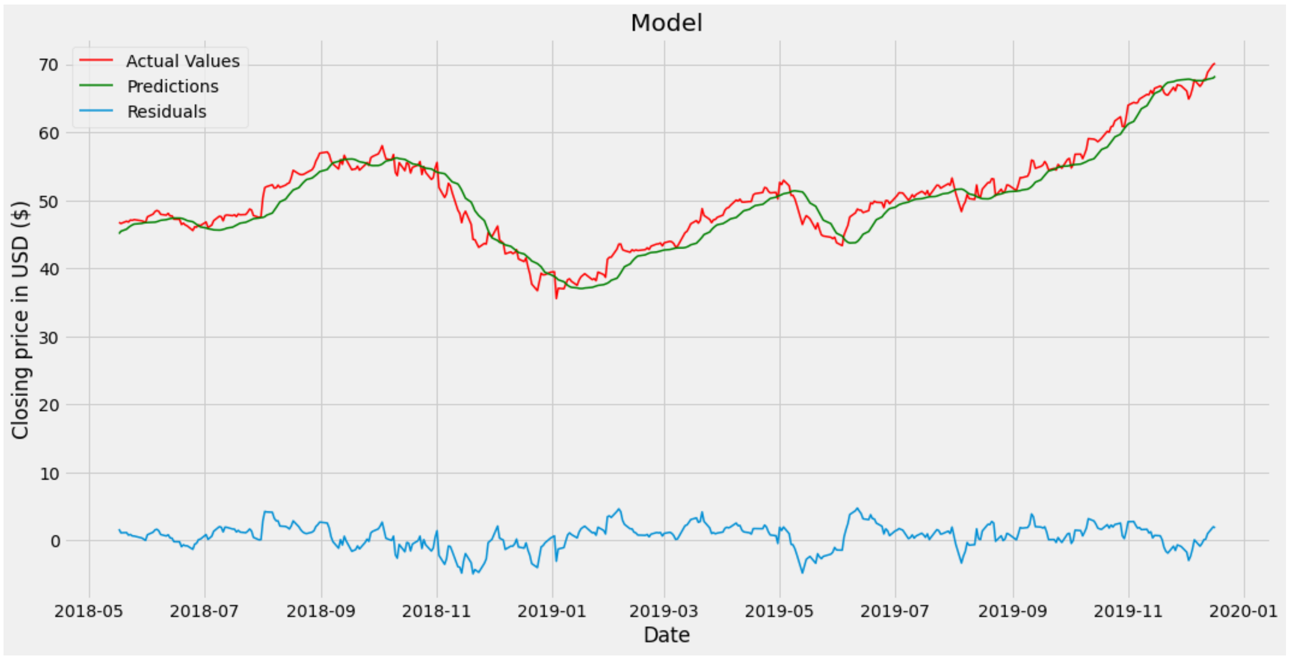

Figure 5.12 - Residuals of the Predictions and Actual Values.

Figure 5.13 played a significant role in the training process, and the reason must be address. The plot shows the MSE and RMSE errors through fifty epochs of training. This plot is crucial because it translates whether the model was trained correctly or not. After the tenth epoch, MSE error flattens, and RMSE after the fortieth epoch is also flattened significantly in the particular graph. There are three diagnoses for this graph. The first one is the Underfit Learning Curves; this happens when the model cannot obtain a sufficiently low error value on the training set. Another underfitting situation in the model is when the loss is decreasing and continues to decrease at the end of the last epoch, meaning that it needs either more epochs or a change in the model's structure. Such a diagnosis can be identified from the Learning Curve of training loss. The second diagnosis is the Overfit of Learning Curves that happens when the model is more complicated than it should be, leading to overfitting the data, which essentially means following the errors and the noise too closely. Overfitting can be identified if the validation loss plot decreases up to a point and then starts increasing again. Usually, the training loss of an overfitted model will decrease while the validation losses will stay similar through epochs or even higher, meaning the training is overfitting. The final diagnosis of these Learning Curves is a good fit; as an example, Figure 5.13 demonstrates its case. The validation and training losses decrease at the same rate. In addition to that, the losses flatten out at an acceptable small value, meaning that the training was made successful. This was the method to keep track of the LSTM model's training that made the stock predictions.

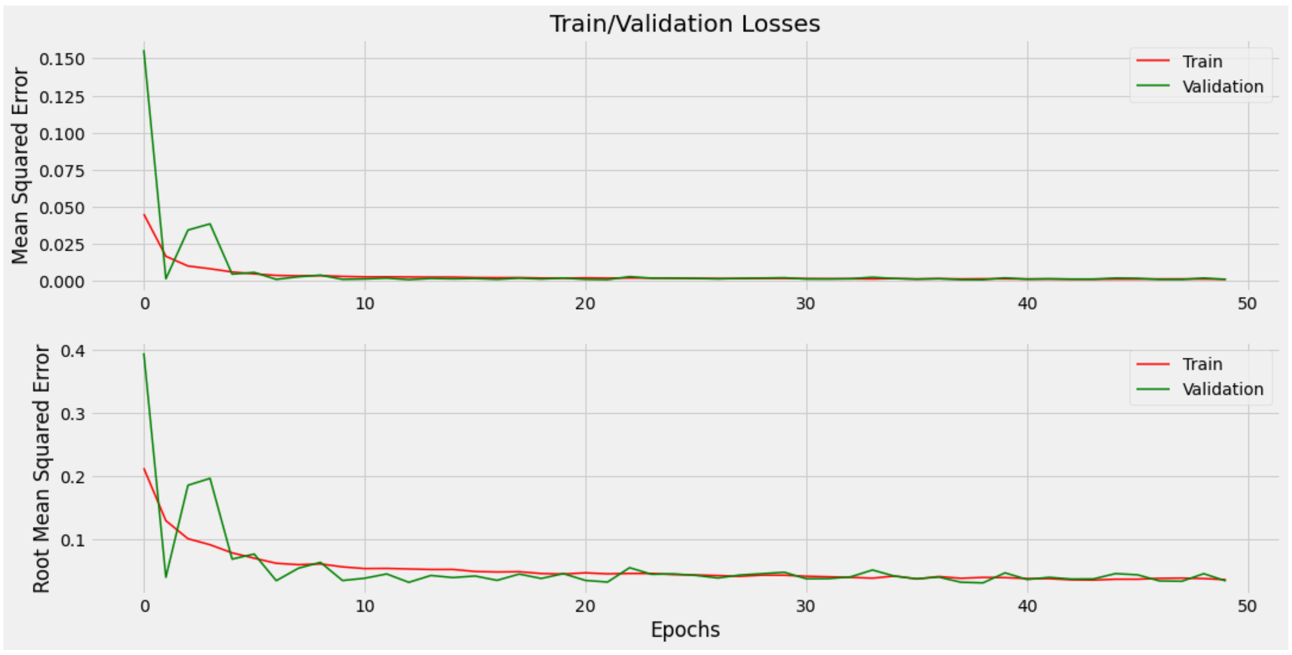

Figure 5.13 - Training and Validation Learning Curves.

5.5 In Conclusion

By making a recap to this chapter, all the implementations were analyzed thoroughly. The results were shown graphically, along with more details about the code of each methodology. Besides the implementations, the theoretical background of these three implementations was also mentioned for further explanation. Overall, it was an important chapter that includes the main projects presenting them separately before their comparison in the next chapter.

6 CHAPTER 6 – Comparison of the Implementations

6.1 Introduction

Since a reference to all the implementations has been made, it is time to see which one is better and why. This chapter will first focus on the differences between every single implementation. Afterward, a discussion around the best methodology for better accuracy will be made, along with its advantages and disadvantages. The role of this chapter, in general, is a more generic way to compare and see different possible solutions and their results.

6.2 Results Comparison

Besides the visual interpretation, the predictions are used inside some selected regression error metrics for better control of the model's efficiency. Such errors selected are Mean Absolute Error (MAE), coefficient of determination or R^2 score, and max error. Mean Absolute error is a risk metric corresponding to the expected value of the absolute error loss and is calculated as:

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \; (3)$$

Where $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value. MAE output is a non-negative floating point with the best value being 0.0. R^2 score represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of how good is the fit and therefore how likely it is for unseen samples to be predicted by the model. Its best score is 1 and worse is 0, though it can get negative values meaning that the model could be arbitrarily worse. If $\hat{y}$ is the predicted value of the i-th sample and y_i is the corresponding true value for a total of n-samples, R^2 is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n} * (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} * (y_i - \overline{y})^2} \; (4)$$

$$\text{Where } \overline{y} = \frac{1}{n} \sum_{i=1}^{n} * y_i \; (5) \text{ and } \sum_{i=1}^{n} * (y_i - \hat{y}_{i_i})^2 = \sum_{i=1}^{n} * \epsilon_i^2 \; (6)$$

Max error computes the maximum residual error, and as simple error as it seems, it should be a necessary tool to evaluate the model since it predicts stock prices. With that error, high divergence at certain points can be found. The higher the max error is the less trustworthy the predictions are, meaning low values on max errors are preferable. If ŷ is the predicted value of the i-th sample and y_i is the corresponding true value for a total of n-samples, max error is defined as:

$$\text{Max Error}(y, \hat{y}) = max(|y_i - \hat{y}_i|)(7)$$

In order to compare the three implementations, there will be a series of testing for various percentages of future days predictions, and they will be presented in Table 6.1 for MAE error and in Table 6.2 for R^2 error. The test is on Apple's stock prices from '04-01-2010' to '27-08-2020', a total of 2682 data points, and it is the same for all three methods for equality of the test.

Table 6.1 - MAE for all three implementations from 5% to 40% of the data.

MAE	5%	10%	15%	20%	25%	30%	35%	40%
Linear Regression	35.819	74.661	49.03	36.772	29.851	41.877	42.375	48.468
Tree Decision	10.355	22.524	15.671	10.37	9.857	13.793	16.052	16.984
LSTM	3.728	2.591	2.993	2.482	2.719	3.740	3.987	2.051

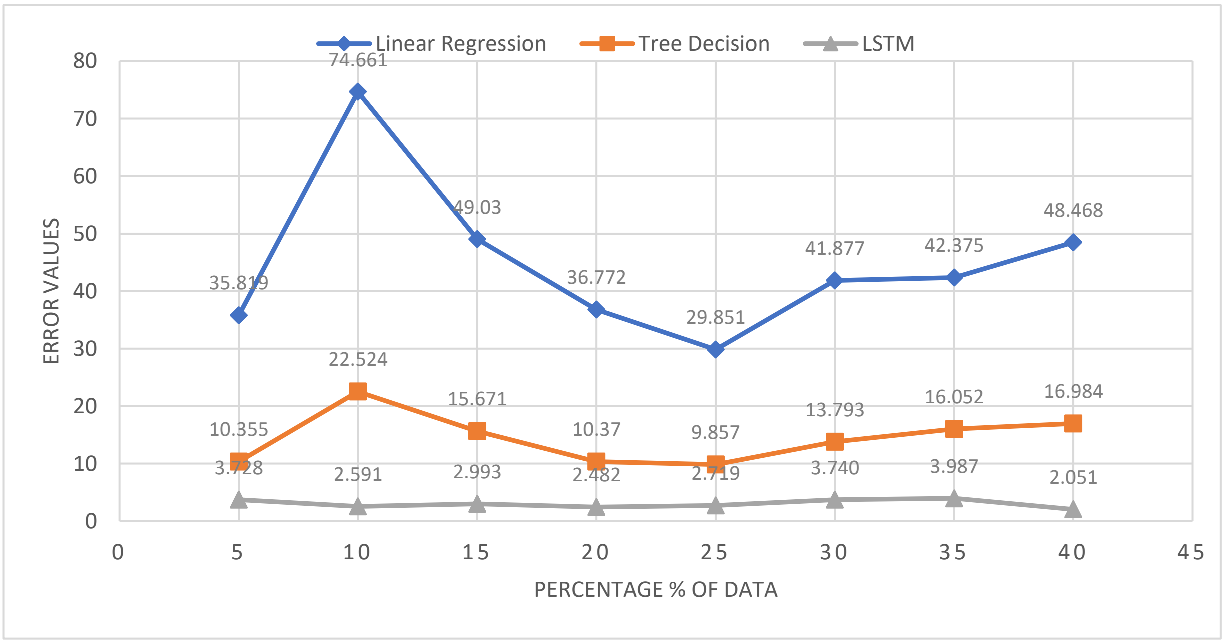

Figure 6.1 - Graph of MAE score over Data Percentage for Prediction on Apple Stocks.

Table 6.2 - R^2 for all three implementations from 5% to 40% of the data.

R^2	5%	10%	15%	20%	25%	30%	35%	40%
Linear Regression	0.5250	-0.8040	-0.0060	0.5040	0.6270	0.4080	0.1240	0.1960
Tree Decision	0.9330	0.4140	0.6680	0.9170	0.9210	0.8160	0.6880	0.6540
LSTM	0.9241	0.9562	0.9627	0.9685	0.9655	0.9278	0.9155	0.9800

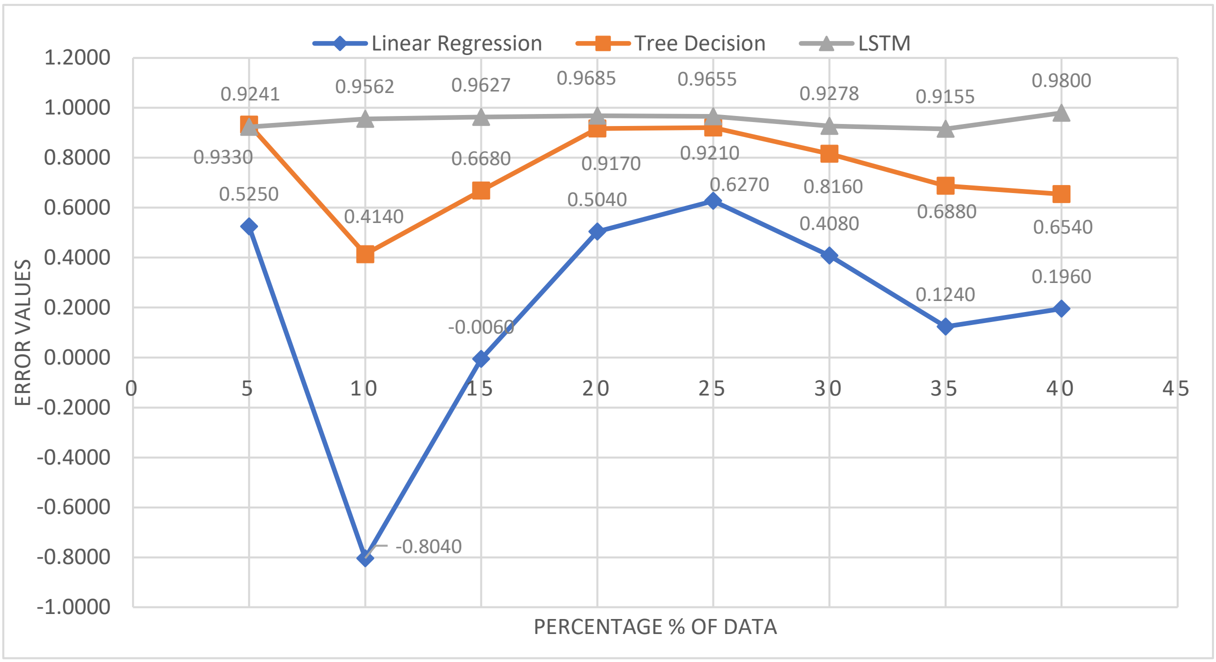

Figure 6.2 - Graph of R^2 score over Data Percentage for Prediction on Apple Stocks.

Both graphs show clearly that the worse implementation of all three is Linear Regression. In Figure 6.1, it has higher MAE values, which is, of course, an indicator of a flawed prediction model. In Figure 6.2, it has the most inconsistent plot with, once more, the worst values for R^2 errors. In second place comes Tree Decision, as can be observed in both Figures that it competes for the LSTM network results. It has surprisingly low enough MAE scores and values relatively close to the ideal "1.00" R^2 score. LSTM marks are outstanding and consistent throughout all the testing, declaring the clear winner between these three methods.

6.3 Best Implementation

It is evident that an ANN implementation easily dominates over the Linear Regression and Tree Decision models. LSTM results are more consistent and far better in comparison. In Table 6.2 LSTM at forty percent of future prediction, it scored "0.98" in the R^2 error metric, which implies that the model predicted the values almost entirely. Such results were expected since this implementation had to be tuned for optimizing the predictions. It learned the data at a satisfying level and delivered even more satisfying results. It was concluding that from all three implementations, the most accurate and reliable one is the LSTM network used in this project. This specific RNN played a significant role in memorizing the time series patterns as it did a great job in predicting and finding better patterns for future prices. Anyone who watches this specific research must not forget that the results can vary widely if the data is changed or some small parameters from the model. So, considering these results, anyone could adapt this implementation to a new one, and why not even a better one.

6.4 In Conclusion

This chapter mainly made a comparison between the implementations that were executed in this research. For this comparison to be made, a series of testing of the models and their predicting capabilities was conducted and analyzed in graphs and tables. By summarizing the results, LSTM overcame the other implementations, making it the best possible method to predict stock prices.

7 CHAPTER 7 – Summary

7.1 Project Review

This project overall showed what level of power the computers in recent years have achieved. With more computational potential than ever before, Machine Learning "came to conquer the world." Specifically, the junction between the economic field and Machine Learning created a great program that "predicts" the future. While it might not be an actual prediction, but all this tremendous computational power will undoubtedly assist at a great rate all developers and researchers.

As a project in general, it was quite challenging to be made correctly and on point. However, it is not considered a challenging project for someone with above-mediocre knowledge of programming skills and Machine Learning algorithms. There was a vast amount of different possible ways to predict stock prices. Linear Regression, Tree Decision Regressor, and LSTM networks seemed to work pretty well as individuals. Considering the project as finished, all objectives have been achieved from learning and exploring Python's programming capabilities, especially in Machine Learning territories, to learning all the possible ways to solve regression tasks with NNs. Anyone who wants to see Keras and SciKit Learn libraries' power can quickly execute the code that lies around in the Appendix section.

Generally speaking, the program itself stands alone, astonishing well. However, of course, "perfect is the enemy of the good," this stands for this project too, and there are probably many ways to get better. For example, the data in the Linear Regression and Decision Tree implementations were not normalized. In the LSTM network, there were not put weight-initializers in the first hidden layer. Both tools, normalization and weight initialization, should help the models perform grater. As already seen, they were not included in the making of the models since, when tested, they harmed the accuracy of the models. Further exploration of their association with the models could be done, at a future level, for optimizing the code.

7.2 Results

Learning how to use valuable sources and tools was the best result possible. Now, in terms of the project, the results should be noticeable. Machine Learning is an extraordinary field where the sky is the limit for every new developer that wants to make an application for any prediction to make.

Specifically, during this thesis, Linear Regression came up significantly shorter than other implementations in terms of success and accuracy. The main reason is that Linear Regression uses patterns that simple enough and incapable of following the fast transition of the stock prices. In many cases and many stock examples (as seen in Figure 6.1 and Figure 6.2), Linear Regression has a “disappointing” accuracy score. Tree Decision, despite the fact this it is not a Deep Learning method, found better patterns in the time series data and had significantly better predictions compared to Linear Regression. If someone had to choose from those two implementations, Tree Decision should be the go-to.

After experiencing the possibilities of simple Machine Learning models, a further “dive” to Deep Learning was a necessary step to discover and learn NNs in a better way. From a small online search, LSTM networks were undoubtedly the most common implementation for time-series tasks and stock prediction specifically. Compared to Linear Regression and Decision Tree, NNs have far more hyper-tuning, hence better possibilities to be improved and optimized. After many trials and tests, a certain level of accuracy was achieved for a specific stock price (Apple’s stock price). A notable mention is that when tested to different stocks, the same implementation that had excellent error scores had significantly less accurate than expected. One possible reason is the fact that every single stock as time-series graph has a different slope, ups and downs, history, and in general, a completely different form. A model that can predict fast transitions and significant changes in prices can perform worse in a more straightforward stock share where there are few price changes, and vice versa.

7.3 Alternate Projects

It is evident for countless reasons that Machine Learning has a vast future ahead of it. This project could be described as a basis for a bigger stock price prediction program, even though it can get even better with future implementations and different methodologies. An alternate project that can be built using this stock prediction model is a trading bot. There are several available APIs that can make such a project feasible. A trading bot can use a Machine Learning model to predict the prices and act accordingly. This trading bot could be a “money-making” machine that uses a starting fund and tries to increase it are any given rate. So, the better the prediction model is, the more accurate transactions the bot will make. While Machine Learning techniques cannot be used alone, they can be combined with other stock trading methodologies to make a more accurate mixture.

Another possibility for stock prediction is making a model that can consider more input values than just price points. A model can be made that takes into consideration an overall view of the company. This could happen by making a bot that analyzes blogs and news sites, whether something crucial happened in the real world. For example, as history has shown, whenever the CEO position is changed, instability and uncertainty are the two facts that make the shareholders sell due to the fall of the stock price. In opposition, taking the example of Apple, every September is programmed to release its new products; such an event has high chances to raise its shared values as it deploys new technology to the public. There is much useful information that can assist in the stock prediction since stock prices are based on real-world happenings and not virtual ones.

8 CHAPTER 8 – Epilogue

8.1 General thoughts and remarks

Artificial Intelligence, in general, has been around for decades. For example, Stephen Hawking used one of the first machines that use a Machine Learning algorithm for text prediction. Even though such a machine made his communication ability better while using AI algorithms, he stated that *“artificial intelligence could be the worst event in the history of our civilization.”* That incident shows to the developers the importance of Machine Learning algorithms and the caution needed to build such algorithms. Stock prediction cannot be harmful to humankind, but there are countless implementations for Machine Learning that can get out of control. So, every developer should be aware of the risks that might hide behind sophisticated AI algorithms and also that there must be effective management.

8.2 Ethics on Machine Learning

There is a particular part of Machine Learning that needs to be addressed before the closure of this research, and this is the ethical part of it. After seeing all the good and beneficial parts of Machine Learning, some concerns have to be expressed to raise every reader’s caution. Machine Learning, of course, is considered a necessary tool to help humanity advance and thrive. One big problem is that AI can get uncontrollable after a point. That undoubtedly means not robots conquering the world, but the nature of a machine reading patterns better than humans. In the past few years, some documentaries have also explained their concerns about this issue. An alarming fact is that there are Machine Learning algorithms that are currently active and are so complicated that, in some way, operate on their own, based on the given results that the developers wanted. Of course, these results are the customers and their precious money where money can sometimes be the customer’s attention and not real money. Complicated AI algorithms have learned human patterns and how a human reacts at an even emotional level. It might sound strange or even uncanny, but it is somewhat true that the algorithm “reads” the customer and tries its best to indirectly “force” the customer to use the product-service more.

AI algorithms, of course, help humans to categorize their preferable clothes, music, videos to watch, photos, and much more. However, without control, this could lead the human to get addicted to that service. Uncontrollable use of these algorithms with the sole intent of the money has had an enormous impact on countless people. There is no secret that younger ages are more prone to addiction since their usage of, for example, social media is remarkably high. There have also been psychological studies supported by statistical data that indicate a clear relation between the addiction to digital connections and the rise in depression or even suicide rates, especially among teenagers in the USA. Nevertheless, there is no regulation regarding "how much someone can use a service" since it is free and accessible to almost everyone. This leads to the companies creating more and more addictive products using these AI algorithms to satisfy their customers' needs.

All these facts are not a statement for removing the use of AI. Instead, it is a grievance to the developers whose only intention, on purpose or not, is to have the customer's attention without a limit. There should be moderation in terms of desired customer attention. Developers, researchers, and companies should not forget that these AI algorithms are here to assist and not overcome humans' presence. Wisely and sparingly used, Machine Learning algorithms can thrive with humans and not over humans.

Bibliography – References – Online Sources

Géron, A. (2019). Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems. Beijing: O'Reilly Media.

Harrington, P. (2012). Machine learning in action. Shelter Island, NY: Manning.

Müller, A. C., & Guido, S. (2018). Introduction to machine learning with Python: A guide for data scientists. Sebastopol, CA: O'Reilly Media.

Tsai, C. F., & Hsiao, Y. C. (2010). Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. Decision Support Systems, 50(1), 258-269.

Wang, Z., Liu, K., Li, J., Zhu, Y., & Zhang, Y. (2019). Various Frameworks and Libraries of Machine Learning and Deep Learning: A Survey. Archives of Computational Methods in Engineering, 1–24. https://doi.org/10.1007/s11831-018-09312-w

Appendix A

Linear Regression

```
#Install the libraries and the dependencies
#Import Data analysis libraries
import numpy as np
import pandas as pd
#Import Machine Learning Libraries
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
#Import simple plot library
import matplotlib.pyplot as plt
#Import iPlot libraries
from plotly import __version__
import cufflinks as cf
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
init_notebook_mode(connected=True)
#Import error library
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error

#Use cufflinks offline
cf.go_offline()

#Jupyter interactive plot
%matplotlib inline

#Load the data
netflix = pd.read_csv('NFLX.csv')

#Print out statistical features of the dat set
print('"Statistical features of Netflix"')
print(netflix.describe())
print()

#Print out the file's rows and columns
print("Netflix's data set rows,columns")
print(netflix.shape)
print()

#Convert object type to float
netflix['Close'] = pd.to_numeric(netflix['Close'], errors='coerce')

#Plot the stock data
netflix.iplot(x='Date',y='Close',title='Netflix Stock',xTitle='Date',yTitle='Close Price ($)',
fill=True,colors=['green'],theme='white')

#Get the close price
netflix = netflix[['Close']]

#Create a variable to predict the 'x' days out into the future 10%
future_days = 56

#Create a new column (target) shifted 'x' units/days up
netflix['Prediction']= netflix[['Close']].shift(-future_days)

#Create the feature data set (X) and convert it to a numpy array and remove the last 'x' rows/
days
Xnetflix = np.array(netflix.drop(['Prediction'],axis=1)[:-future_days])
```

```

#Create the target data set (Y) and convert it to a numpy array and get all of the target values except the last 'x' rows/days
Ynetflix = np.array(netflix['Prediction'])[:-future_days]

#Split the data into 75% training and 25% testing
x_train_netflix, x_test_netflix, y_train_netflix, y_test_netflix = train_test_split(Xnetflix, Ynetflix, test_size=0.25)

#Create the Linear Regression model
lr_netflix = LinearRegression().fit(x_train_netflix, y_train_netflix)

#Get the last 'x' rows of the feature data set
x_future_netflix = netflix.drop(['Prediction'],1)[:-future_days]
x_future_netflix = x_future_netflix.tail(future_days)
x_future_netflix = np.array(x_future_netflix)
x_future_netflix

#Show the model Linear Regression prediction
lr_prediction_netflix = lr_netflix.predict(x_future_netflix)
print('Linear regression prediction preview')
print(lr_prediction_netflix)
print()

#Compute the mean absolute error for Linear Regression predictions
mae_lr_netflix = mean_absolute_error(netflix['Close'].iloc[-(future_days+1):-1],lr_prediction_netflix.tolist())
print('Mean absolute error for Linear Regression')
print(round(mae_lr_netflix,3))
print()

#Compute the r2 score for Linear Regression predictions
r2_lr_netflix = r2_score(netflix['Close'].iloc[-(future_days+1):-1],lr_prediction_netflix.tolist())
print('r2 score for Linear Regression')
print(round(r2_lr_netflix,3))
print()

#Compute the max error for Linear Regression predictions
maxerr_lr_netflix = max_error(netflix['Close'].iloc[-(future_days+1):-1],lr_prediction_netflix.tolist())
print('max error for Linear Regression')
print(round(maxerr_lr_netflix,3))
print()

#Visualize the data of linear Regression
predictions2_netflix = lr_prediction_netflix
valid2_netflix = netflix[Xnetflix.shape[0]:]
valid2_netflix['Predictions'] = predictions2_netflix
lr_final_netflix=pd.concat([netflix['Close'],valid2_netflix['Predictions']],axis=1)
lr_final_netflix[['Close','Predictions']].iplot(kind='spread',theme='white',title='Netflix Stock Prediction with Linear Regression ',xTitle='Days',yTitle='Close Price ($)')
```

Appendix B

Tree Decision

```
#Install the libraries and the dependencies
#Import Data analysis libraries
import numpy as np
import pandas as pd
#Import Machine Learning Libraries
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
#Import simple plot library
import matplotlib.pyplot as plt
#Import iPlot libraries
from plotly import __version__
import cufflinks as cf
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
init_notebook_mode(connected=True)
#Import error library
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error

#Use cufflinks offline
cf.go_offline()

#Jupyter interactive plot
%matplotlib inline

#Load the data
netflix = pd.read_csv('NFLX.csv')

#Print out statistical features of the dat set
print('"Statistical features of Netflix"')
print(netflix.describe())
print()

#Print out the file's rows and columns
print("Netflix's data set rows,columns")
print(netflix.shape)
print()

#Convert object type to float
netflix['Close'] = pd.to_numeric(netflix['Close'], errors='coerce')

#Plot the stock data
netflix.iplot(x='Date',y='Close',title='Netflix Stock',xTitle='Date',yTitle='Close Price ($)',
fill=True,colors=['green'],theme='white')

#Get the close price
netflix = netflix[['Close']]

#Create a variable to predict the 'x' days out into the future 10%
future_days = 56

#Create a new column (target) shifted 'x' units/days up
netflix['Prediction']= netflix[['Close']].shift(-future_days)

#Create the feature data set (X) and convert it to a numpy array and remove the last 'x' rows/
days
Xnetflix = np.array(netflix.drop(['Prediction'],axis=1)[:-future_days])
```

```

#Create the target data set (Y) and convert it to a numpy array and get all of the target values except the last 'x' rows/days
Ynetflix = np.array(netflix['Prediction'])[:-future_days]

#Split the data into 75% training and 25% testing
x_train_netflix, x_test_netflix, y_train_netflix, y_test_netflix = train_test_split(Xnetflix, Ynetflix, test_size=0.25)

#Create the decision Tree Regressor model
tree_netflix = DecisionTreeRegressor().fit(x_train_netflix,y_train_netflix)

#Get the last 'x' rows of the feature data set
x_future_netflix = netflix.drop(['Prediction'],1)[:-future_days]
x_future_netflix = x_future_netflix.tail(future_days)
x_future_netflix = np.array(x_future_netflix)
x_future_netflix

#Show the model Tree Prediction
tree_prediction_netflix = tree_netflix.predict(x_future_netflix)
print('Tree prediction preview')
print(tree_prediction_netflix)
print()

#Compute the mean absolute error for Tree Regression predictions
mae_tree_netflix = mean_absolute_error(netflix['Close'].iloc[-(future_days+1):-1],tree_prediction_netflix.tolist())
print('Mean absolute error for Tree Decision')
print(round(mae_tree_netflix,3))
print()

#Compute the r2 score for Tree Regression predictions
r2_tree_netflix = r2_score(netflix['Close'].iloc[-(future_days+1):-1],tree_prediction_netflix.tolist())
print('r2 score for Tree Decision')
print(round(r2_tree_netflix,3))
print()

#Compute the max error for Tree Regression predictions
maxerr_tree_netflix = max_error(netflix['Close'].iloc[-(future_days+1):-1],tree_prediction_netflix.tolist())
print('max error for Tree Decision')
print(round(maxerr_tree_netflix,3))
print()

#Visualize the data of Tree Decision
predictions1_netflix = tree_prediction_netflix
valid1_netflix = netflix[Xnetflix.shape[0]:]
valid1_netflix['Predictions'] = predictions1_netflix
tree_final_netflix=pd.concat([netflix['Close'],valid1_netflix['Predictions']],axis=1)
tree_final_netflix[['Close','Predictions']].iplot(kind='spread',theme='white',title='Netflix Stock Pediction with Tree Decision',xTitle='Days',yTitle='Close Price ($)')
```

Appendix C

LSTM

```
#Import all libraries
import math
import numpy as np
import pandas as pd
import pandas_datareader as web
import matplotlib.pyplot as plt
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
import tensorflow as tf
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras import backend as K
from sklearn.metrics import r2_score
from sklearn.preprocessing import MinMaxScaler
plt.style.use('fivethirtyeight')

#Check if GPU is running on Colab
tf.test.gpu_device_name()

#Read our stock data from yahoo finance
df = web.DataReader('AAPL',data_source='yahoo',start='2012-01-01',end='2019-12-17')
df = df.drop(['Volume', "Adj Close", "Low"], axis=1)

#Create a dataframe with only the closing price
data = df.filter(['Close'])

#Make it a numpy array
dataset = data.values

#Get the number of rows to train the model on/ i.e to 80% of the data are going to be trained

training_data_len = math.ceil(len(dataset)*0.8)

#Scale-normalize our data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

#Create the scaled training data set
train_data = scaled_data[0:training_data_len,:]
#Split the data to the x train y train data
x_train = []
y_train = []

#Use the last 60 days
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i,0])
    y_train.append(train_data[i,0])

#Convert to numpy arrays too
x_train, y_train = np.array(x_train), np.array(y_train)

#Reshape the data because the LSTM model want 3-
d data and x_train.shape(80% of data,60 days)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

#Create the test data set
#Create the new array with the last 60 values
test_data = scaled_data[training_data_len-60:, :]

```

```
#Create the x_test and y_test data sets
x_test = []
y_test = dataset[training_data_len:,:]

#Use last 60 days
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i,0])

#Convert to numpy array
x_test = np.array(x_test)

#Reshape
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))

#Build the model with 4 layers
d=0.2
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(Dropout(d))
model.add(LSTM(50, return_sequences=False, activation='relu'))
model.add(Dropout(d))
model.add(Dense(25,activation='relu'))
model.add(Dropout(d))
model.add(Dense(1,activation='relu'))

#Compile the model
model.compile(optimizer='adam',loss='mse',metrics=[tf.keras.metrics.RootMeanSquaredError()])

#Model specs
bs = 200
ep = 50
vs = 0.1

#Train the model
history = model.fit(x_train, y_train, batch_size=bs, epochs=ep,validation_split=vs)

#Save metrics
losses = history.history

#Get the predictions from the models for closing prices
predictions = model.predict(x_test)

#Reverse scale values
predictions = scaler.inverse_transform(predictions)

#Set our data for the plot
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

#Plot the closing price
plt.figure(figsize=(16,8))
plt.title('Close price along Time')
plt.plot(df['Close'],linewidth=3)
plt.xlabel('Date')
plt.ylabel('Closing price in USD ($)')
plt.show()
df.shape

#Plot the metrics to see if our model was built correctly
plt.figure(figsize=(16,8))
plt.subplot(2,1,1)
plt.title('Train/Validation Losses')
plt.plot(losses['loss'],color='red',linewidth=1.5)
plt.plot(losses['val_loss'],color='green',linewidth=1.5)
plt.ylabel('Mean Squared Error')
plt.legend(['Train','Validation'], loc=0)
plt.subplot(2,1,2)
plt.plot(losses['root_mean_squared_error'],color='red',linewidth=1.5)
plt.plot(losses['val_root_mean_squared_error'],color='green',linewidth=1.5)
plt.xlabel('Epochs')
plt.ylabel('Root Mean Squared Error')
plt.legend(['Train','Validation'], loc=0)
plt.show()

#Vizualize the prediction of our model
plt.figure(figsize=(16,8))
plt.title('Model')
plt.plot(train['Close'])
plt.plot(valid['Close'],color='red',linewidth=1.5)
plt.plot(valid['Predictions'],color='green',linewidth=1.5)
plt.xlabel('Date')
plt.ylabel('Closing price in USD ($)')
plt.legend(['Train','Actual Values','Predictions'], loc='lower right')
plt.show()

#Visualize residuals
plt.figure(figsize=(16,8))
plt.title('Model')
plt.plot(valid['Close'],color='red',linewidth=1.5)
plt.plot(valid['Predictions'],color='green',linewidth=1.5)
plt.plot(valid['Close']-valid['Predictions'],linewidth=1.5)
plt.xlabel('Date')
plt.ylabel('Closing price in USD ($)')
plt.legend(['Actual Values','Predictions','Residuals'], loc=0)
plt.show()

#Custom R2-score metrics for prediction evaluation
print()
print('R2 score is =')
print(r2_score(valid['Close'],predictions))
print()
```