Comparative Analysis of Machine Learning Models for Stock Price Analysis Across Different Dataset Sizes

by

Abir Alam Srabon
18101389
Mahmudul Hasan Abrar
18201165
Nimur Rahman
18101111
Washif Uddin Ahmed
18301204
Salamat Sajid Hridy
22241140

A thesis submitted to the School of Data and Sciences
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

School of Data and Sciences
Brac University
May 2023

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Abir

Abir Alam Srabon
18101389

[illegible]

Mahmudul Hasan Abrar
18201165

[illegible]

Nimur Rahman
18101111

Washif

Washif Uddin Ahmed
18301204

[illegible]

Salamat Sajid Hridy
22241140

Approval

The thesis/project titled "Comparative Analysis of Machine Learning Models for Stock Price Analysis Across Different Dataset Sizes" submitted by

1. Abir Alam Srabon(18101389)
2. Mahmudul Hasan Abrar(18201165)
3. Nimur Rahman(18101111)
4. Washif Uddin Ahmed(18301204)
5. Salamat Sajid Hridy(22241140)

Of Spring, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on May 17, 2023.

Examining Committee:

Supervisor:
(Member)

Moin Mostakim

Moin Mostakim
Senior Lecturer
School of Data and Sciences
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam, PhD
Associate Professor
School of Data and Sciences
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
School of Data and Sciences
Brac University

Ethics Statement

We thus declare that this thesis is based on the findings of our research. All other sources of information have been acknowledged in the text. This thesis has not been previously submitted, in whole or in part, to any other university or institute for the granting of any degree.

Abstract

The nature of the stock market has always been ambiguous as it constantly fluctuates for various factors. The regular fluctuations have always made it difficult for investors to invest. This paper compares six different machine learning models for stock price analysis: Explainable AI, Q-learning method, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network. Each model was evaluated using three different datasets consisting of 7000, 10000, and 14000 data points, respectively. The results of the experiments show that depending on the size of the dataset, the performance varies and the specific model used. In general, the deep learning models (LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network) outperformed the Explainable AI and Q-learning models in terms of predictive accuracy. However, the Explainable AI and Q-learning models had the advantage of being more interpretable and easier to understand, which may be desirable in certain applications. Overall, this study provides insights into the strengths and weaknesses of various machine learning models for stock price analysis and highlights the importance of choosing the right model for the specific task at hand. Future work concentrates on optimizing the performance of the models further or exploring the use of hybrid models that combine the strengths of multiple approaches.

Keywords: Machine Learning; Stock price analysis; Explainable AI; Q-learning; Bi-LSTM; Restricted Boltzmann Machine; LSTM; Deep Belief Network; Dataset size; Performance evaluation; Interpretable models; Predictive accuracy

Dedication

We would like to dedicate this thesis to our loving parents and all of the wonderful academics we met and learned from while obtaining our Bachelor's degree and especially our beloved supervisor Mr. Moin Mostakim.

Acknowledgement

First and foremost, we would like to express our gratitude to our Almighty for allowing us to conduct our research, put out our best efforts, and finish it. Second, we would like to express our gratitude to our supervisor, Mr. Moin Mostakim sir, for his input, support, advice, and participation in the project. We are grateful for his excellent supervision, which enabled us to complete our research effectively. Furthermore, we would like to express our gratitude to our faculty colleagues, family, and friends who guided us with kindness, inspiration, and advice. Last but not least, we are grateful to BRAC University for allowing us to do this research and for allowing us to complete our Bachelor's degree with it.

Table of Contents

List of Figures

Chapter 1

Introduction

The basic goal of statistical methods, a well-known investment approach, is to create a result that is uncorrelated to the stock market by speculating with various directions and expected returns in the form of positive. [9]. The distance [5] and cointegration procedures [8] are used in traditional statistical methods, and the relationship between the two is assessed using the distance metric and the test of cointegration, respectively. This statistical framework has been created in the past with a number of stock market prediction models [9], [14], [16] due to the significance of the models. The financial market is quite unpredictable, and predictions are frequently skewed due to a number of factors. Taking this into account, this article employs LSTM to create a multi-parametric model with time series data for stock market prediction.

In the world of finance, stock market forecasting is crucial because, if done correctly, it may limit losses and lower market risks while also providing significant benefits. Financial data are typically very volatile, melancholy, nonstationary, and nonlinear [6]. Because of this, the forecast is particularly troublesome and has recently received a lot of attention from academics and the financial industries [1], [13], [7]. The prediction approach is reliant on models for time series analysis. Similar to the autoregressive integrated moving average (ARIMA) [19], this model is exceptional. The generalized autoregressive conditional heteroskedasticity (GARCH) model [10] is another common model that is similar to ARIMA. These algorithms identify failure-prone linear series that capture nonlinear financial data trends. The biological neural networks that process data [25], [34], [33] are inspired by the computing paradigm known as the neural network, which can handle nonlinear problems and advance numerous fields of artificial intelligence, such as NLP [30] and computer vision [23]. Similarly to this, neural networks have gained popularity for forecasting time series data used in finance. Because it provides access to historical time series values through recurrent connections, the recurrent neural network (RNN) is one of the most well-known neural network techniques that has been employed extensively in the financial world [18], [26]. Unfortunately, RNN suffers from the vanishing gradient issue when utilized as a standard recurrent unit since backpropagation of time training is challenging. A recurrent unit called long short-term memory (LSTM) [3] was proposed as a solution to this issue. Long-term memory storage is made possible by the LSTM, which has memory cells with self-connections that can store temporal states of data and multiplicative gates that can govern the flow of information. Even on a wide variety of data, including machine translation [22], speech recognition [31], remaining life prediction [21], the virus spread prediction [29], etc., LSTM works well for sequential data modeling, such as stock market price.

1.1 Research Problem

Stock price analysis is a critical task for investors and financial professionals, as accurate predictions of stock prices can provide insights into the performance of companies and markets. Machine learning has emerged as a powerful tool for stock price analysis, as it can leverage large amounts of data and complex algorithms to identify patterns and trends in stock prices. However, the selection of the appropriate machine learning model and dataset size is essential for achieving accurate predictions. [32] [35]. Contrarily, the stock market contains complex data that may be influenced by a number of factors, with prior values being omitted. In order to forecast the stock market, it is crucial that recurrent units manage the multi-variable inputs. Thankfully, Guo et al. introduced the multi-variable LSTM in [32]. The management of the multi-variable inputs by recurrent units is essential for stock market forecasting. Fortunately, Guo et al.'s introduction of the multi-variable LSTM in article 06 saved the day. In this study, we compare Explainable AI, Q learning method, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network as six alternative machine learning models for stock price analysis. To assess the effectiveness of these models across various data sizes, we employ three distinct dataset sizes, ranging from 7000 to 14000 data points. The goal of our research is to identify the machine learning model or models that offer the best accurate stock price analysis forecasts, as well as how these models perform across various dataset sizes. Our tests demonstrate that in terms of predictive accuracy, the deep learning models—including LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network—outperform Explainable AI and Q-learning models. The Explainable AI and Q-learning models, however, offer the benefit of being easier to explain and comprehend, which may be preferable in some situations.

By addressing this research issue, our work seeks to shed light on the advantages and disadvantages of various machine learning models for stock price analysis and offer recommendations for the best models to choose for various applications and dataset sizes.

1.2 Research Objective

This study's main goal is to assess six machine learning models' effectiveness for stock price analysis while comparing their precision across various dataset sizes. In particular, we want to:

- Using three distinct dataset sizes (7000, 10000, and 14000 data points), assess the predictive performance of the Explainable AI, Q learning approach, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network models for stock price analysis.

- Find out which model(s) provide the best precise forecasts for stock price analysis and how their performance changes with the quantity of the dataset.
- Examine the advantages and disadvantages of several machine learning models for stock market analysis, as well as how their levels of usability and interpretability differ.

To accomplish these goals, we first preprocess the price information for stocks to eliminate any unusual or invalid values and make sure the data is in a machine learning-friendly format. The six machine learning models are then trained and tested using the three distinct dataset sizes, and their performance is assessed using metrics like mean squared error, root mean squared error, and R-squared.

Additionally, we compare the six predictive models, taking into consideration their effectiveness, interpretability, and usability. We assess each model's advantages and disadvantages and highlight how well it fits particular applications and use situations. Our study aims to shed light on the efficacy of various machine learning models for stock market prediction and assist in the selection of the most suitable models for particular dataset sizes and applications. Our research aims to add to the body of knowledge on machine learning and economics and offer insightful information to investors and financial experts.

Our research compares the precision and comprehensibility of six machine learning models for the price of stocks analysis across three distinct dataset sizes. By reaching this goal, we intend to offer insightful information on the benefits and drawbacks of various machine learning models and aid in the selection of the best models for particular applications and sample sizes. In order to help investors and financial professionals make wise decisions, our study intends to contribute to the expanding discipline of machine learning in financial affairs.

The rest of this paper is structured as follows. The related literature on the use of stock prediction models in statistical arbitrage is reviewed in Chapter 2. The models utilized in this paper are introduced in Chapter 3. The experimental results are described in Chapter 4. The pre-processing method and model implementation with performance assessment are illustrated in Chapter 5. The experimental results are displayed and examined in Chapter 6. The paper is concluded in Chapter 7 with a discussion of limitations and further research.

Chapter 2

Literature Review

In this research, the stock market prediction survey results have been taken into account.

2.1 Background

Over the past 20 years, the popularity of stock market forecasting in the research world has increased significantly. Major cases are discovered to use macroeconomic factors like stock returns as the only input for the first phase of the study with the utilization of non-linear slants as financial exchange record returns. For the reason that financial exchange returns are raucous, uncertain, confused, and nonlinear in nature, this strategy focuses on the nonlinear expectations of the stock market. Numerous functions, such as the binary threshold, the linear threshold, the hyperbolic sigmoid, and the brown functions, have been employed to forecast different values.

Since the stock market is a target market for the financial industry, predictions have been made using machine learning techniques. One of them is specific assessment, but it cannot consistently produce specific results, so it is crucial to develop methods for progressively more precise measurement. The backslide approach has its own limitations in light of the challenges and steps involved. This model behaves in a manner similar to the least squares approach, however it was fitted in an arbitrary sequence. For instance, by lessening a handicapped variant of the least squares setback work or the ”non-appearance of fit” in another standard. Again, fitting nonlinear models can be done using the least squares method.

The influence ratio of financial and technical analysis on stock market forecasting employing random forest, AI, and various man-made indicators is highly exceptional, as is well known. Researchers have spent time identifying the optimal technique and making future advancements in order to increase accuracy. As a result, there are several ways to apply the recent stock market predictions. It should come as no surprise that no model is ideal for financial analysis like stock market. Each methodology has its own usage restrictions. In the previously mentioned paper, the stock value estimation was completed using the self-assured Timberland estimating, which is being used to indicate the cost of the stock using fiscal extents structure the prospective quarter. This is one method for optically visualizing the incident by moving approaching it with the use of a clever model that uses erratic behaviour to forecast the future cost of the stock from recorded data. On the other hand, there are a variety of additional factors that affect the stock market and can cause changes. The accuracy of the stock value forecast model can be increased by using the cash related size in close proximity to a model that can strongly separate assumptions, such as the money-related authority's suspicions, the association's overall opinion, news from other sources, and even circumstances that cause the complete trade protection to alter.

[9] explains how challenging it is to produce stock value using a multi-source sample without being aware of protective trades. However, because to the internet's ability to connect academics, the process has become less challenging with the utilization of many variables over time. Using several approaches and options based on precise, reliable data, such as using a feeling analyzer to suggest a striking connection between people's feelings and how their passion for express stocks affects them, are the only ways that budgetary trade information can be adequately predicted. The web news also made it simple to access the stock market, which informed everyone of its recent ups and downs. The utilization of historical data analysis and its effects on stock market price prediction were also discussed in the article. The cost of the stock or offer might be anticipated using historical data and its examples, but in practice it is necessary to use counts to foresee the expenses. The traditional frameworks only care about the diversity of an element used for forecasting. The latter is frequently accomplished with the aid of Genetic Algorithms (GA) or Artificial Neural Networks (ANN's) [16], but these tools fail to build a long-distance relationship between their stock costs transient dependencies.

RautSushrut et al. [2] estimate the stock market price movement using stock index data and a supervised learning classifier. A statistical AI methodology has been modeled after the computational analytical approach. Thus, the employment of support vector machines has demonstrated a strategic method of stock price prediction. In order to anticipate stock market prices, Manoj et al. [4] developed an LSTM network that can handle a linear problem. LSTM with hidden layers was employed by Roondiwala et al. [14]. The typical neuron was replaced with a memory cell from the LSTM, which may assist the memory cell interact with other nodes that remote access the input time effectively and construct a dynamic data capture model with high forecasting accuracy. The NIFTY50 dataset was used for the study. The acquisition of data is a crucial step in the model-training process because the algorithm differs greatly depending on the dataset. The topic that helps to identify the stock market prediction problem was studied with ANN by Kim et al. [16]. They illustrate the massive amounts of data that the system keeps forgetting, which causes the neuron to break and makes their predictions inaccurate. This is frequently caused by two factors: loads are established self-assuredly, and the loads become closer to the system's terminus. In that paper, LSTM usage was suggested.

It is abundantly obvious from Selvin et althorough .'s analysis of financial data [32] that the prediction heavily depends on historical data and the reputation of the company. Additionally, the SVM technique has been used to lessen the issue. According to Loke et al. [6], the stock market's volatility should be taken into account when creating a model, with historical data analysis taking precedence and time series data explaining how to anticipate the stock market using conventional methods based on statistical methodologies. The limitations of their work have been addressed using machine learning and AI approaches in order to get results that are more accurate. Since the stock market is a dynamic data set, they came to the conclusion that there is no possible universal solution. According to Zhang et al. [27], stock market forecasting can be crucial in today's society. They demonstrated that predictions can be made with a high degree of accuracy if the data is accurate, can be obtained from reliable sources, and the approach is consistent with the data. RNN and LSTM were used by Xing et al. [22] to update the model with historical equity of share price. They took a certain attribute from the data and used it to make predictions. Opening price, day High, day Low, previous day o price, close price, and date of trading are the characteristics of shares. To predict the stock price of a certain time frame the suggested approach makes time series analysis. A CNN model was suggested by Prosky et al. [18] to evaluate the sentiment of the stock market. It is a more generalized form of the gated recurrent system, according to Xao [1]. The evanescent gradient problem that RNN has is solved by the LSTM that supposed to be less defected other than rest of the deep learning approaches like traditional feed forward neural network. With the help of K-means algorithm a short term stock market prediction has implemented with LSTM.

2.2 Related Work

This study analyzes the frameworks and statistical arbitrage of stock market forecasting models. Huck created a straightforward structure that is nonetheless adaptable for use in stock market forecasting [11]. The predicting, ranking, and trading processes make up the framework's main technique. It requires n stocks, and the function shows the data for stocks i and j at time t. The predictor creates a return projection for the following scale for each stock i. The dataset's time dimension is understood by the author using an Elman network as a predictor. A ranking of all stocks is produced using the $ELECTREIII$ outranking method based on these projected returns, with a matrix of predicted spreads serving as the decision matrix. The last m steps stock was sold out, and the top m steps stock was taken out during trade. Applying the $S\&P100$'s weekly return, which covers stock prices from 1992 to 2006, This study displays a superb outcome in terms of return and directional forecasting. Then, Huck conducted additional research on the subject and employed multi-step ahead forecasting, which was more flexible and realistic [12].

Similar but less complex framework was developed by Krauss et al. [15]. A predictor has been trained using the lag in stock input returns. then makes a prediction about the likelihood that a stock will one day outperform the market. Each stock has gone through this process once, with the stocks being arranged in descending order according to the prediction's probability. The high rank represents the future stock market success that is most anticipated. If the first k stocks are long and the final k stocks are short, the ranking can go up to $2k$ stocks. With a mean daily return of 0.43 percent and no concern for transaction costs, random forests [20] produce the greatest results when applied empirically to the stocks that made up the $S\&P500$ index from 1992 to 2015. Additionally, Fischer et al. created the work stated above using an LSTM network to forecast the direction of movement of data for the $S\&P500$ stock index that is out of sample [16]. LSTM achieves a daily return of 0.46 percent on the identical data set before transaction charges. For the same issue, Shen et al. [24] updated the gated recurrent unit (GRU) with the RNN. The margin-based loss has been minimized in the GRU support vector machine model, which was constructed with $softmax$ operation. They fared better than conventional GRU and SVM, according to that experiment. LSTM was also utilized to forecast stock market data by Lee et al. [17]. But they build diversified portfolios by establishing cutoffs for classifying companies as long or short. Based on investors' preferred degrees of risk, his system can create portfolios with those levels.

Only the lagged returns have been used as input in the above formula, which is based on closing price calculations. Since stock market data is generated in the financial sector, it is more complicated and subject to several influences. Because of this, if a prediction model is just built on previous pricing, it may not be accurate. In their prediction model, the authors [27], [28] employed a few technical indicators. They use PCA to minimize the input's dimension after adding the LSTM feature. However, there were no dense layers in the conventional LSTM to compute the inputs. As a result, in this study we not only add more variables as input features but also develop the LSTM-based prediction model.

Chapter 3

Model Architecture

The goal of this study is to use artificial neural networks to improve the precision of daily stock index price estimates. We obtain information from current market activity, where technical analysts utilize a type of safety analysis to predict price direction by examining past data. Because the gathered data comprises a range of values on various scales, the time series must first be fitted and normalized in order to enhance network training. There must first be a stage of preparation for the data. Technical analysis is used to locate the best technical indicators, whereas principal component analysis is used to identify features and reduce data. The NARX model is subsequently constructed using the PCA functional subset. After that, a data sample is trained using a serial-parallel architecture. After the training operations, the serial-parallel architecture is transformed into a parallelized network to carry out prediction tasks. The thorough work schedule for this study is shown in the accompanying graphic.

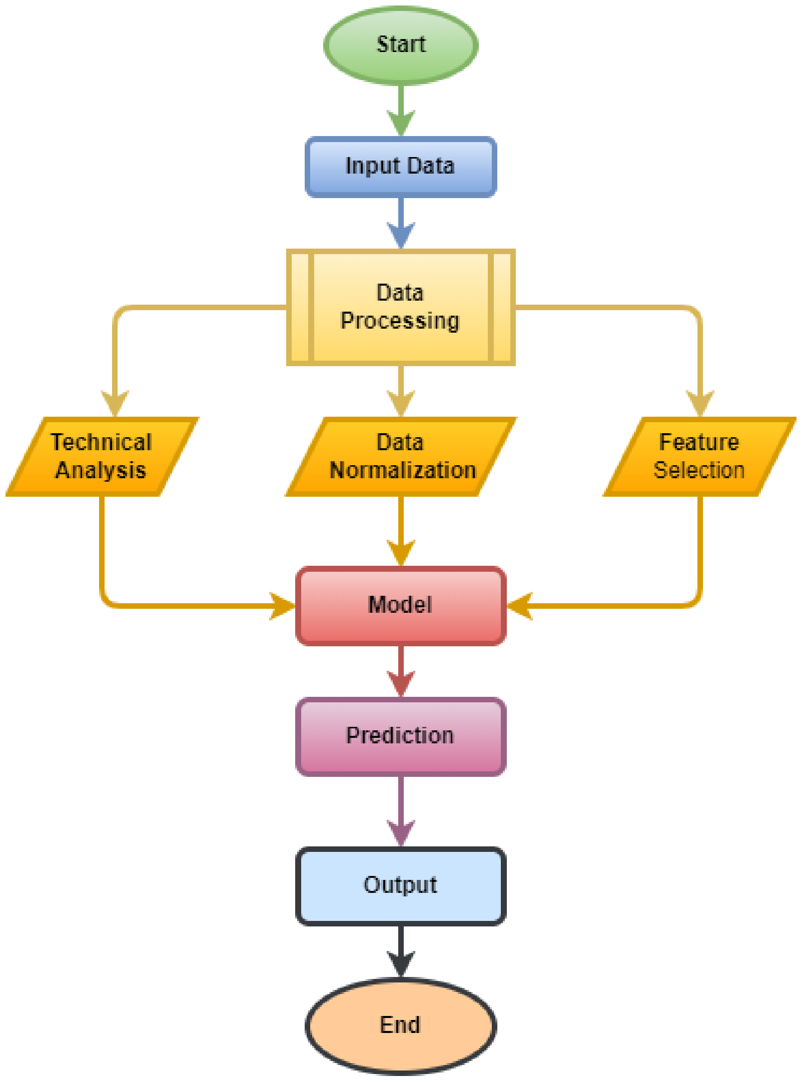


Figure 3.1: Work Plan

3.1 Explainable AI

The fields of artificial intelligence and machine learning have made substantial strides in a number of industries recently, including financial services, healthcare, and transportation. The lack of candor in these models, especially can make it challenging to grasp how they generate their forecasts and conclusions, which is an increasing worry that comes along with these advancements. Herein lies the value of Explainable AI (XAI). An emerging area of study called XAI aims to create machine learning algorithms that can justify their predictions and judgments. An extensive explanation of an XAI system's model architecture will be given in this part.

Our model is made up of three layers: the input layer, the hidden layer, and the output layer. The raw data is fed into the model in the input layer. This could comprise photos, text, or numerical data, based on the nature of the data. The neurons or nodes in the hidden layer do the computations essential to make projections. Finally, the projected output is produced by the output layer. A Decision Tree is a form of XAI model with a tree-like topology. A root node, branches, and leaves make up the structure. The root node symbolizes the complete dataset, and the branches reflect the data's features. The model's output or judgment is represented by the leaves.

The Decision Tree XAI model is based on a top-down approach, which means it divides the data into subgroups based on a set of principles or conditions. These rules are based on data characteristics involving age, income, or gender. The model then assesses each rule to see which one divides the data into discrete subgroups the best. After determining the rule, the data is divided into two subgroups based on the rule. This technique is performed for each subset recursively until a halting requirement is fulfilled. The stopping condition could be a maximum tree depth, a restricted number of observations per leaf, or a certain impurity reduction.

Furthermore, Decision Trees can handle both categorical and numerical data, making them suitable for a wide range of applications. They are also relatively fast and can handle large datasets.

3.2 Long Short Term Memory

To implement the PCA parameters manually it is very difficult and almost impossible to optimize. This research mitigates the problem using a more complex model so that the model can compute each past data point's significance and provide optimized predictions. Thus, the implementation of the Long Short Term Memory (LSTM) model provides the weight updation while training the ML model.

LSTM as an RNN provides the scope to work on data sequences and ease learning by retaining only the relevant information from the time scope. The extracted information from the network that is learned by the model is added to a memory that gets updated after each timestamp based on the significance of the new information to the sample.

The model implements the LSTM cell each with three gates illustrates as the input gate, the output gate, and the forget gate. The three gates combinedly perform to learn the weights and determine the ratio of a current data sample to be remembered and past learned context should be forgotten. The cell state C_t represents the short-term and the long-term internal memory of a cell.

3.2.1 Input Gate

The input gates have been implemented to select the new information to be added and stored in the current C_t. A *sigmoid* function is implemented to reduce the input vector (i_t) values.

$$i_t = \sigma(W_i.[h_t - 1, x_t] + b_i) \tag{3.1}$$

After that, a *tanh* function modifies each value between $[-1, 1]$ $(C - t)$. An element-by-element matrix has been multiplied by i_t and C_t which represents the information that requires to be added to the current cell.

$$\simeq C_t = \ tanh(W_C.[h_t - 1, x_t] + b_C) \tag{3.2}$$

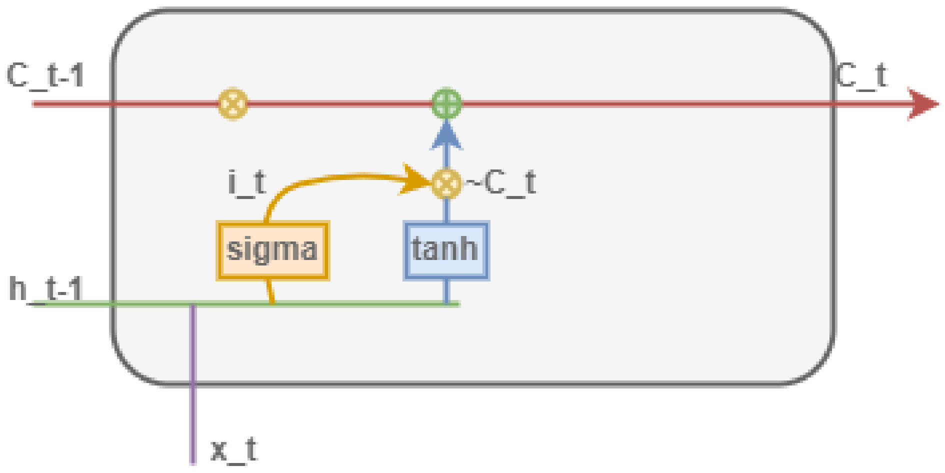


Figure 3.2: Input Gate

3.2.2 Output Gate

To control the output of the flowing to the next cell the output gate has been implemented. The output gate consists of a *sigmoid* function and then to filter the less important information a *tanh* function has been implemented. By this technique, the information that needs to pass through is kept. The output (o_t) is calculated by the following equation.

$$o_t = \sigma(W_o.[h_t - 1, x_t] + b_o) \tag{3.3}$$

And the required (h_t) is as follows.

$$h_t = \ o_t * tanh(C_t) \tag{3.4}$$

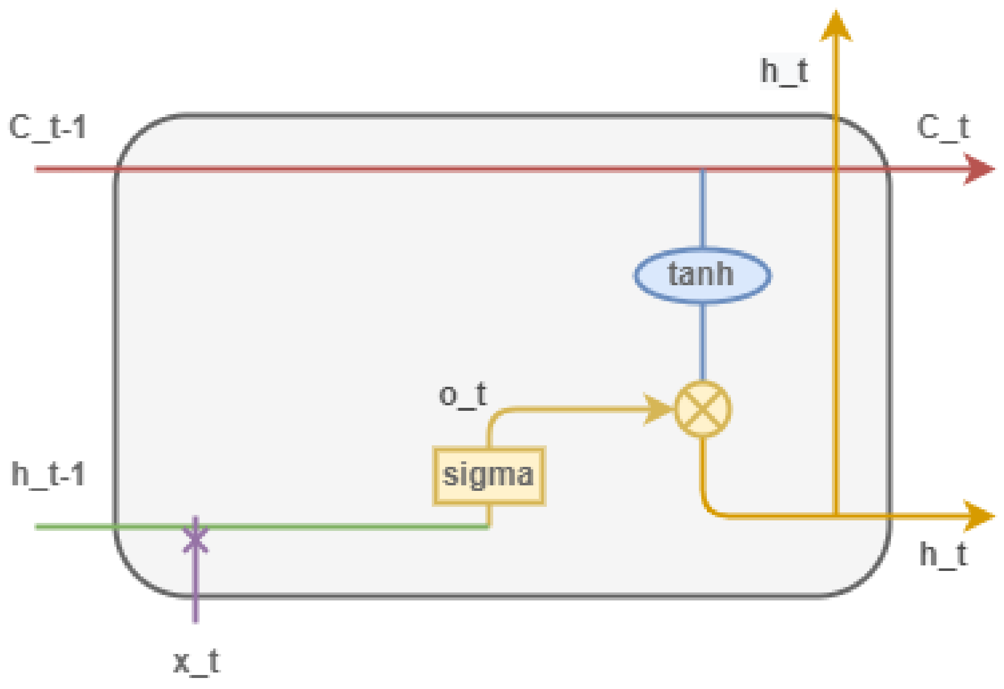


Figure 3.3: Output Gate

3.2.3 Forget Gate

The implementation of the forget gate confirms which information is to be forgotten by the model. The filtered information that the model can recognize as less important has been thrown away with this implementation of the forget gate. Mathematically, the forget gate has been implemented with the *sigmoid* function such as the output value range between $[0, 1]$ from the $C_t - 1$ state. 1 indicates the complete passing value and 0 defines the completely filtered-out values. The following equation has been implemented for the forget gate.

$$f_t = \sigma(W_f.[h_t - 1, x_t] + b_f) \tag{3.5}$$

The figure illustrates the functionality of forget gate in an LSTM cell.

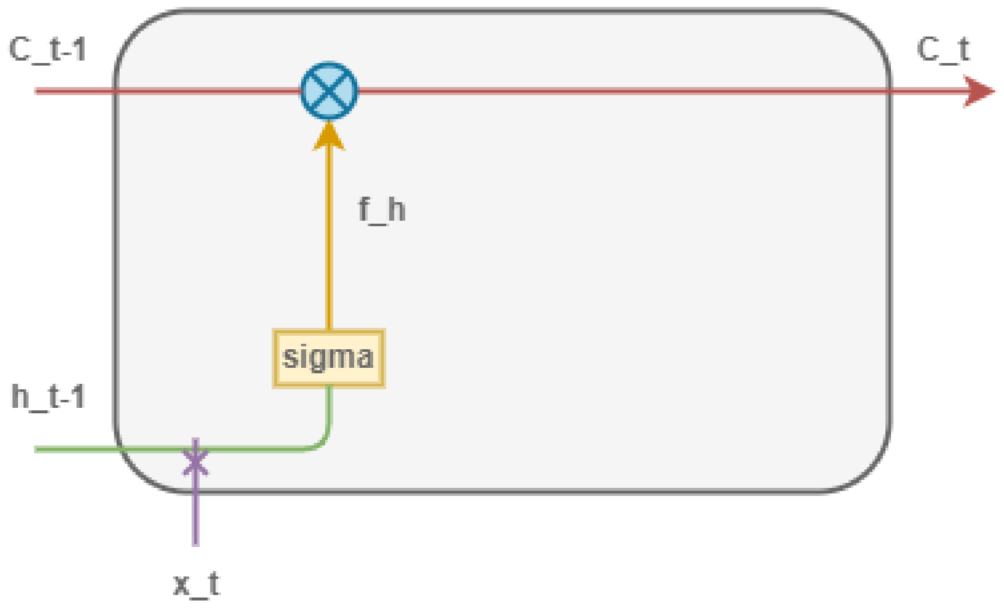


Figure 3.4: Forgate Gate

3.3 Q-Learning Mehtod

Q-Learning is a reinforcement learning method that iteratively improves its estimations of the parameters of state-action pairings to learn an optimal policy for an agent. The method operates by keeping a table of Q-values, that indicate the estimated accumulated reward that the agent will get by doing a specific action in a specific condition.

The Q-Learning algorithm begins by estimating the Q-values across all state-action pairs. The agent then engages with its surroundings by performing actions and monitoring the subsequent states and rewards. Based on this knowledge, the agent adjusts its Q-values using the formula:

$$Q(s, a) = Q(s, a) + [r + max(Q(s', a')) - Q(s, a)] \tag{3.6}$$

when Q(s, a) is the Q-value for the current state-action pair, r is the reward received for performing action an in state s, is the discount factor that determines the significance of additional rewards, and max(Q(s', a')) is the maximum Q-value for the state that follows s' and all possible actions a' that can be performed from s'.

The Q-Learning algorithm interacts with the environment indefinitely, adjusting its Q-values after every step until it reaches an optimal policy. For each state, the optimal policy is established by picking the action with the highest Q-value. One of the fundamental aspects of Q-Learning is the fact that it does not need an environment model, allowing it to learn from firsthand experience. This makes it ideal for applications with complicated and difficult-to-model environments.

However, Q-Learning may suffer from the exploration versus exploitation problem, in which the agent may become stuck in a suboptimal behavior because it is not researching sufficiently to identify the optimal policy. To balance exploration with exploitation, Softmax probing has been utilized.

3.4 Bi-LSTM

Bidirectional Long Short-Term Memory (Bi-LSTM) is a neural network architectural variant used in the processing of natural languages and speech recognition. It is a form of the recurrent neural network RNN for short, that can handle data sequences, such as text or audio, and model the relationships that exist between the sequence's constituents. The basic idea underlying Bi-LSTM is to employ two LSTMs, one that analyzes the input sequence forward and another which processes it backward. This enables the model to accurately represent both the past and future implications of each piece in the sequence, boosting its overall understanding of the sequence. The Bi-LSTM model is made up of many layers of LSTM cells, each with a number of hidden units. The model is fed a series of vectors, with each of the vectors indicating a different element in the order in which they appear. The input is sent into the Bi-LSTM's first layer, and the outcome of each LSTM cell in the subsequent layer is transmitted to the next layer.

The last LSTM layer's output is then sent to a fully connected layer, which maps the LSTM output to the resultant space. In a classification task, the output space can be a set of discontinuous labels, whereas, in a regression work, it can be a continuous value. The Bi-LSTM model makes use of two sets of hidden states, another for the forward LSTM and the other for the reverse LSTM. At each time step, the hidden states are updated by mixing the most recent input with the prior hidden state using a set of learned weights. The forward and backward LSTM outputs are concatenated at each time step to give a composite output that contains both the past and future implications of the present element in the series. This sum of the output is then transmitted to the Bi-LSTM's next layer.

3.5 Restricted Boltzmann Machine

RBM is a generative stochastic neural network system that is frequently employed for unsupervised learning tasks like dimensionality reduction, feature extraction, and collaborative filtering. The binary stochastic neuron, which may operate on values of 0 or 1, is the fundamental building unit of an RBM. The RBM model is divided into two layers: visible and hidden. The input data is represented by the visible layer, and the hidden layer is represented by an array of hidden variables that capture the fundamental structure of the data that was entered. The two layers are completely linked, with each visible neuron linked to each buried neuron and vice versa. Unsupervised training is used to learn the connection weights between neurons in the visible and hidden layers, which are typically learned via contrastive divergence or comparable gradient descent approaches. During training, the model adjusts the weights to minimize the reconstruction error, which is the disparity between the input data and the model's reconstructed output data.

The RBM model employs a probabilistic method for learning, with each neuron in the visible and hidden layers possessing a probability distribution that determines whether the neuron is active or inactive. The distribution of probabilities for each neuron is dependent on the model's energy function, which can be described as the sum of the neuron states and connection weights. Using Gibbs sampling or other similar techniques, the RBM model is trained by sampling from the joint distribution of the visible and hidden layers. The model changes the hidden layer based on the input data during training and then modifies the visible layer based on the newly generated hidden layer. This procedure is repeated until the model has learned the fundamental structure of the input data, at which time it is said to have converged. RBMs have the critical virtue of being "generative" models, which means they may be used to produce new data that is comparable to the input data. This is accomplished by sampling from the combined dispersion of the visible and hidden layers and then generating new output data from the sampled data.

3.6 Deep Belief Network

A Deep Belief Network (DBN) is a form of artificial neural network (ANN) composed of stacked layers of stochastic, latent variables known as Restricted Boltzmann Machines (RBMs). A typical DBN architecture may be defined as follows:

- The input layer, which contains the visible units that receive the input data.
- Multiple hidden layers, each of which is an RBM made up of hidden units and bias units. The hidden units learn to derive high-level characteristics from the provided information, while the bias units learn to predict the hidden units' prior likelihood of being active.
- The output layer, which is a linear regression layer that predicts the target value based on the activations of the hidden units in the top RBM.

A DBN's architecture may be represented as an accumulation of RBMs, with each RBM connected to the preceding and next layer by undirected edges. The interactions among the visible and hidden units in each RBM are weighted, and the weights are learned using the Contrastive Divergence (CD) unsupervised learning technique. Once the DBN has been pre-trained layer by layer with CD, it may be fine-tuned with supervised learning to further enhance the weights and biases of the whole network and fed into the set of data for analysis, such as classification or regression. Backpropagation and a loss function suited for the job at hand were used for fine-tuning.

3.7 Performance Evaluation

In this section, the metric to measure the performance of the implemented model has been discussed. As stock price prediction is a regression problem, the two evaluation metric has been used. One of the used metric is Root Mean Squared Error (RMSE) and the other is Mean Absolute Percentage Error (MAPE).

To calculate the error RMSE uses the difference between the actual (A_t) and predicted ($F - t$) price values and returns an absolute error measure for N timescale. The following equation has been implemented to calculate RMSE.

$$RMSE = \sqrt{\frac{1}{N} * \sum_{t=1}^{N} (A_t - F_t)^2} \tag{3.7}$$

Next, MAPE considers the error by looking at the true value and thus measures relatively how off the predicted values are from the truth value rather than taking the actual difference. It helps to deal with data with more fluctuations and keeps the error range in constant checking. The following formula has been implemented as the MAPE value.

$$MAPE = \frac{1}{N} * \sum_{t=1}^{N} \frac{A_t - F_t}{A_t} \tag{3.8}$$

Both of the metrics for measuring the error are implemented to determine how close or far the stock price is after the prediction from the real-world scenario.

Chapter 4

Dataset

The stock exchange is a complicated system that is impacted by a number of variables, including political stability, economic development, and global events. As a result, reliably forecasting stock prices is a difficult endeavor, and scholars have used a variety of ways to address this issue. In this paper, we propose a unique dataset containing data on stock markets from four countries: the United States, the United Kingdom, Russia, and Bangladesh. The dataset is divided into three subsets: 7000 data, 10,000 data, and 14000 data, providing a rich supply of information for constructing and testing prediction models.

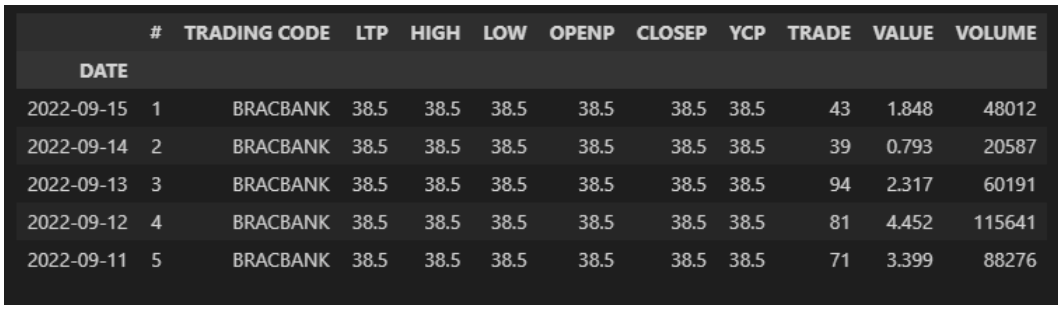
Opening and closing prices, daily high and low prices, trading volume, and other crucial financial indicators are all included in our dataset. These variables came from a variety of sources, including Internet financial databases, corporate reports, and news stories. The addition of data from many nations enhances the dataset's depth by giving a varied variety of political and economic circumstances to examine. This is especially essential since stock prices are frequently impacted by worldwide occurrences, and including data from different nations helps us to construct more reliable and applicable models. The uniqueness of our dataset rests in its comprehensiveness and the possibility it provides to stock market prediction researchers. Because of the dataset's vastness and variety, researchers may evaluate developments in the stock market across numerous nations and construct models that can effectively anticipate stock prices in diverse locations. Furthermore, our dataset may be used to investigate the influence of numerous factors on stock prices, such as economic policy and political instability. This form of study can be very beneficial for investors who want to invest in many nations but need to examine the risks involved.

Another notable benefit associated with our dataset is that it has the potential for application in deep learning model construction. The dataset's huge size and inclusion of several variables make it an ideal option for constructing deep learning models capable of understanding intricate correlations between diverse factors impacting stock prices. Furthermore, the incorporation of data from several nations allows for the development of models that can adapt effectively across diverse locations.

Finally, our dataset provides a once-in-a-lifetime chance for academics to design and test prediction models for stock market analysis. Its extensive breadth, wide variety of factors, and inclusion of data from numerous nations make it an ideal resource for researching stock market patterns, constructing prediction models, and studying the influence of a number of variables on stock prices. We hope that our dataset will help advance research in the discipline of forecasting the stock market and will help investors make educated decisions about investing in global stock markets.

4.1 Data Analysis

In this section, exploratory data analysis has been discussed with the Bangladeshi data only that is the stock market data of BRAC Bank. To understand the data set, first, it is required to visualize the data. For that purpose, the data frame has been visualized by the *pandas* library which is the most common data frame and manipulation library of Python. The following figure illustrates the data.



	#	TRADING CODE	LTP	HIGH	LOW	OPENP	CLOSEP	YCP	TRADE	VALUE	VOLUME
DATE											
2022-09-15	1	BRACBANK	38.5	38.5	38.5	38.5	38.5	38.5	43	1.848	48012
2022-09-14	2	BRACBANK	38.5	38.5	38.5	38.5	38.5	38.5	39	0.793	20587
2022-09-13	3	BRACBANK	38.5	38.5	38.5	38.5	38.5	38.5	94	2.317	60191
2022-09-12	4	BRACBANK	38.5	38.5	38.5	38.5	38.5	38.5	81	4.452	115641
2022-09-11	5	BRACBANK	38.5	38.5	38.5	38.5	38.5	38.5	71	3.399	88276

Figure 4.1: Data Set

From the figure above we can see the tabular view of our model, here based on date the data is shown. The LTP stands for the Last Trading Price and the YCP stands for the Yesterday's Closing Price.

Because of using *Pandas* library to read the CSV file as dataframe and since the data is indexed by date, the data can also be indexed by the date column. For the analysis the data has taken from September 2020 to September 2022. It will also allow the model to visualize the trends for the model to work with the unpredictable occurrences such as COVID-19 situation. After loading data to the dataframe, it is important to see the main trends of the dataset to understand the fluctuations. The following figure illustrates the HIGH and the LOW points of the BRAC Bank over the time scale.
As per the figure, it is visible that there are two sudden drop in the LOW price of the data, one is in May 2021 and another is in April 2022. Rest of the time it has the steady growth and downfall.

The challenging part for the ML model is to estimate the rapid changes of the data points correctly such as the values of the LOW in May 2021 and April 2022. This paper will focus on evaluating the model performance for predicting the recent values also another company share after training on the past data. Therefore, to mitigate the challenge it requires more parameters other than just the HIGH and the LOW values.

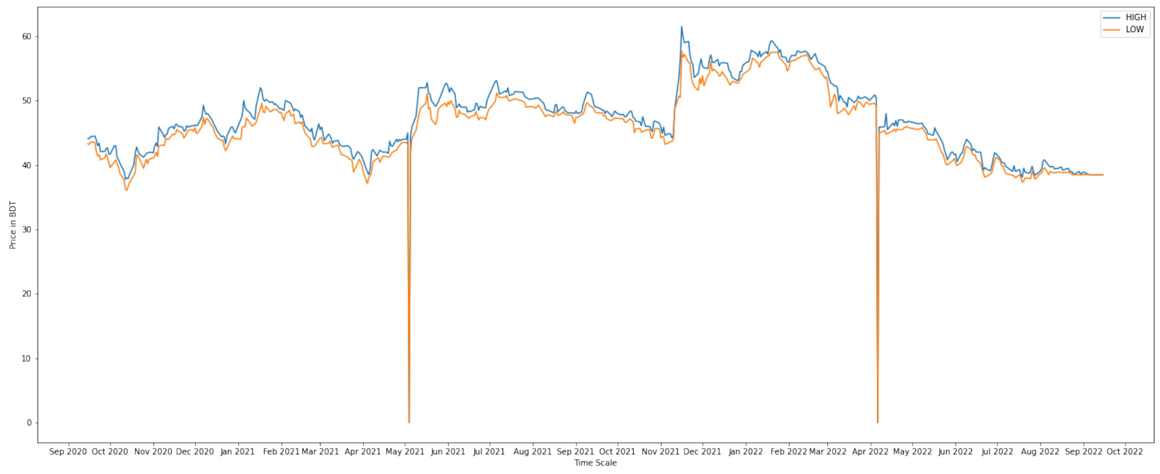


Figure 4.2: HIGH and LOW Points

Like the same way applied previously to see the HIGH and LOW value, the OPEN and the CLOSE value is observed to analyse the data more vividly. Here, the open price is named as OPENP and similarly the close price of the dataset is named as CLOSEP. Plotting the Open and Close values for the time scale is shown below.

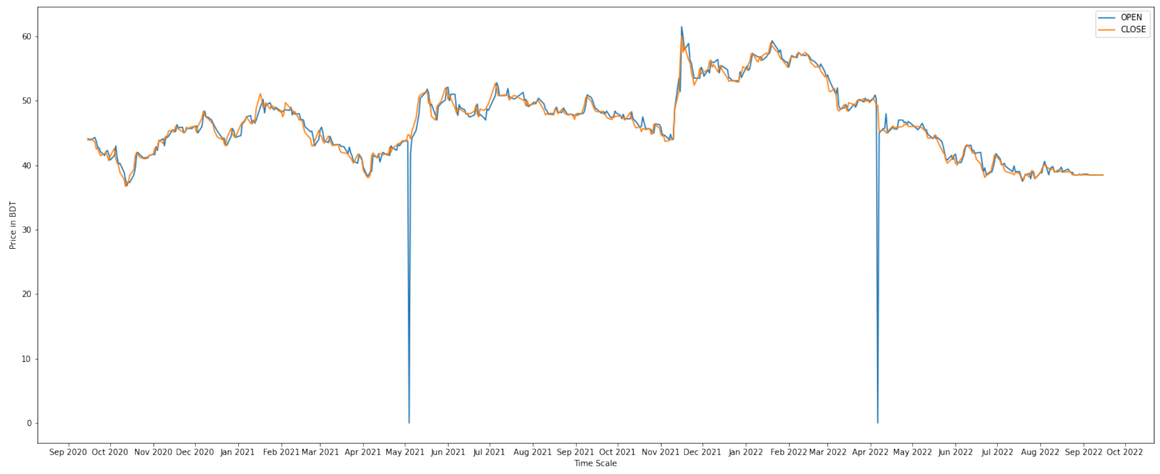


Figure 4.3: OPEN and CLOSE Price

Similar to LOW points, it is visible that OPEN price at May 2021 and April 2022 has gone down severely. As the high fluctuation can have impact on the model performance some other parameter has been taken under consideration.

Likewise, in the previous graphs, the Close price, and Yesterday's Close price are observed and illustrated in the following figure.

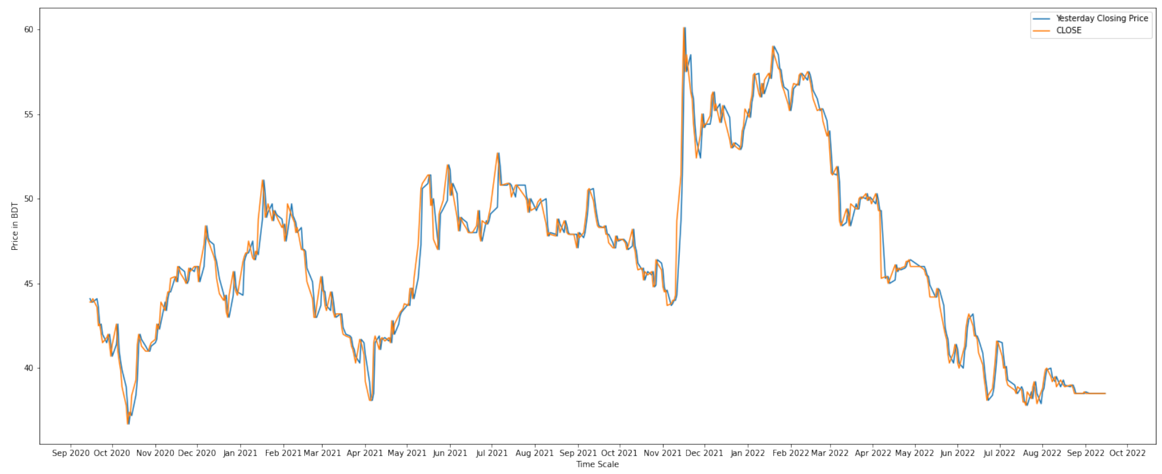


Figure 4.4: Close and Yesterday's Close Price

Here, it is clearly visible that there are no sudden fluctuations and it will help the model to train with good performance. The Volume of the data set should also need to visualize to understand the participation in the company by the traders which is another important aspect to determine the stock price for the next day. The following figure illustrates the volume of the BRAC Bank stock market data.

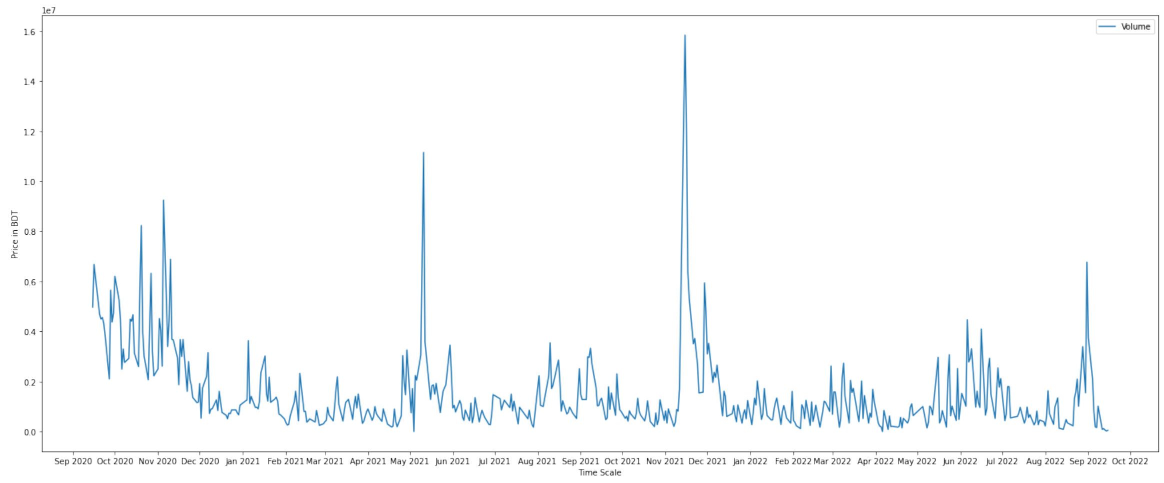


Figure 4.5: Volume

As seen in the figure, the volume is fluctuating more often. As the stock prices are unpredictable and contains high fluctuating values at time scale, it is important to get the right feature engineering processes to improve the model performance. The next section, the details analysis has been provided for the above analysis.

4.2 Stock Price as the Time Series Data

Despite the volatility, stock market price are not just a randomly generated numbers. Therefore, it is necessary to analyze the data as a sequence of discrete time data. Thus, the time series observation is granted as successive points in any given time scale over the time.

As the sequential nature of time series data, the sequence of information needs to aggregate. For the feature engineering the moving average is used that has the ability to smooth the short term fluctuations. This paper holds the 80% data to use for training purpose and 20% data for the prediction over trained values.

4.3 Fundamental Analysis

It is crucial to first look at the intrinsic values, such as tangible assets, investment statements, consumer behavior, and strategic objectives, for the fundamental analysis. For this to be an accurate predictor for the long-term investment, it depends on both historic data and current data. To calculate revenues, assets, liabilities, costs, and other financial metrics, historical and current data is crucial. However, in order to predict short-term market changes, it is necessary to analyze the stock market's technical elements, which are covered in the following part.

4.4 Technical Analysis

Technical analysis is the vital measure of the stock market price prediction with ML models. For this implementation, the measurable data from stock market activities has been thoroughly analyzed including stock prices, historical returns and the volume of historical trades. These quantitative information later used for identifying trading signals to identify the pattern of market movement of the stock market.

Technical analysis is been used to measure the short term trading purpose as the intention of the implementation is to predict the stock price for the short term instead of long term investment. For this paper, as a technical analysis, the sole focus is established on SMA and EMA and evaluated both to find the difference to see the model performance for each of them. LSTM as a deep learning RNN platform has utilized for the time series data and compared with the discussed technical analytical methods.

Chapter 5

Implementation

In this implementation, six different machine-learning models have been utilized for analyzing stock prices. These models include Explainable AI, Q learning method, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network. Three different datasets, containing 7000, 10000, and 14000 data points, respectively, have been used for evaluating the performance of these models. The aim of this study is to compare the accuracy and effectiveness of these models in predicting stock prices using different datasets. This analysis could provide valuable insights for investors and financial analysts.

5.1 Explainable AI

The implemented code aims to use the Decision Tree Regressor algorithm to predict the stock prices based on the given input parameters. The dataset used in this implementation contains stock market data from various countries like USA, UK, Russia, and Bangladesh with three different sizes: 7000, 10000, and 14000 data points.
The code begins with importing the necessary libraries such as *pandas*, *numpy*, *matplotlib*, and $scikit - learn$. Then the dataset is loaded using the $read\_csv()$ function from *pandas* and stored in a DataFrame called df. The $isnull()$ function is used to check if there are any missing values in the DataFrame, but this step has been commented out.
Next, the missing values are replaced with the mean of the corresponding column using the $fillna()$ function, but this step has also been commented out. Then, the *OpenInt* column is dropped using the $drop()$ function since it is not required for prediction.
The 'Date' column is converted to a datetime format using the $to\_datetime()$ function. The dataset is sorted based on the date using the $sort_values()$ function. The input features are extracted and stored in a DataFrame called X, and the target label is stored in a variable called y.
The dataset is split into training and testing sets using the $train\_test\_split()$ function from $scikit - learn$. The input features are standardized using the $StandardScaler()$ function. The $fit\_transform()$ function is used to calculate the mean and standard deviation of the training set and then transform the training set. The $transform()$ function is used to transform the testing set.
The Decision Tree Regressor algorithm is used to train the model with a maximum

depth of 4 and a random state of 42. The $fit()$ function is used to fit the model to the training data. The training and testing accuracy scores are calculated using the $score()$ function and printed to the console. The feature importance is calculated using the $feature\_importances$ attribute of the model, and the results are printed to the console. The feature importances are then plotted using the $bar()$ function from $matplotlib$.
The tree structure is visualized using the $plot\_tree()$ function from $scikit-learn$. The feature names are passed to the function to display the feature names on the tree nodes. The tree is plotted with a maximum depth of 3, and the plot is displayed using the $show()$ function from $matplotlib$.
In addition, the partial dependence plot is generated using the $partial\_dependence()$ function from $scikit - learn$. It is a graphical representation of the relationship between the target variable and a set of input variables, showing the marginal effect of the input variables on the predicted output. The partial dependence plot is not shown in the current implementation, but it can be generated by uncommenting the code.
In conclusion, the implemented code uses the Decision Tree Regressor algorithm to predict stock prices based on input features. The model is trained and tested using three different datasets containing stock market data from various countries. The feature importances and tree structure are also visualized, and partial dependence plots can also be generated. This implementation can be considered as a starting point for further exploration of the performance of other machine learning algorithms on the same dataset.

5.2 Q-Learning Method

The code consists of two parts. The first part implements a Q-learning agent for stock trading, while the second part implements a Q-learning algorithm for the same task but with a different implementation.
The first part of the code defines a class $QLearningAgent$, which is used to define an agent that can learn to trade stocks using Q-learning. The class has methods to initialize the agent, get the action to take based on the current state and $Q-values$, and learn from the reward obtained after taking an action. The agent is trained on a given dataset of stock prices, and the net worth of the agent is recorded over time. Finally, a graph of the net worth over time is plotted using Matplotlib.
The second part of the code implements the Q-learning algorithm for stock trading using a different approach. In this implementation, the state space is defined based on whether the stock price increased, decreased, or remained the same compared to the previous day. The action space consists of buying, selling, or holding the stock. The $Q - matrix$ is initialized with zeros, and the agent learns by updating the Q-values using the Bellman equation. The function is $q\_learning$ implements the Q-learning algorithm for a given number of episodes and returns the total rewards obtained during each episode.

5.3 LSTM

This model implements an LSTM-based deep learning model for time-series forecasting. Specifically, it trains a model to predict the future values of a stock market index based on its past values. The implementation uses the *Keras* library for building and training the model, and the *Pandas* and *NumPy* libraries for data processing.
The first step in the implementation is to load the dataset, which is in CSV format and contains the historical daily values of the stock market index. The data is sorted by date and set as the index of the Pandas DataFrame. The next step is to scale the data using the *MinMaxScaler*, which scales the values between 0 and 1. This is done to ensure that all the features have similar scales and to improve the convergence of the model during training.
The implementation divides the data into datasets for training and testing after scaling it. The training dataset comprises 80% of the data, with the remaining 20% in the testing dataset. To eliminate data leaks and to guarantee that the model is evaluated on unknown data, the split is done chronologically. The implementation also includes a method called $create_dataset$, which accepts a dataset and a time step as the inputs and outputs of the LSTM model's input and output data. The time step specifies how many previous values the model will use to forecast the upcoming value. The time step in this implementation is set to 100, which implies that the model predicts the following day's value based on the previous 100 days' results.
The input and output data returned by the *create_dataset* function are reshaped to a 3D format that is required by the LSTM layer in *Keras*. The LSTM layer takes a 3D input tensor with shape (*batch_size*, *time_steps*, *input_dim*), where *batch_size* is the number of samples in each batch, *time_steps* is the number of time steps in each sample, and *input_dim* is the number of features in each time step. In this implementation, the *batch_size* is not specified, and *Keras* automatically infers it from the input data.
The LSTM model is defined using the Sequential API in Keras. The model consists of three LSTM layers, each with 50 neurons, and a fully connected output layer with one neuron. The *return_sequences* parameter in the LSTM layers is set to True for the first two layers to return the sequence of hidden states, which is required by the next LSTM layer. The last LSTM layer does not return the sequence of hidden states since it is the final layer. The model is compiled using the mean squared error as the loss function and the Adam optimizer as the optimization algorithm.
The model is then trained on the training dataset for five epochs using a batch size of 64. The training loss and validation loss are printed after each epoch to monitor the training progress. Once the model is trained, it is used to make predictions on both the training and testing datasets. The predicted values are then transformed back to their original scale using the inverse transform function of the *MinMaxScaler*.
The root mean squared error (RMSE) performance measures are then computed for both the training and testing datasets. The root mean square error (RMSE) is a measure of the difference between anticipated and actual values, with smaller values indicating better performance. The RMSE is calculated using the *math.sqrt* and $mean_squared_error$ functions from the math and *sklearn.metrics* libraries, respectively.
Finally, the implementation plots the original and predicted values of the stock mar-

ket index for both the training and testing datasets using the *matplotlib* library. The training and testing RMSE values are also printed on the plot for easy comparison. Overall, this implementation is a simple yet effective example of using LSTM-based deep learning models for time-series forecasting. It showcases the use of Pandas and *NumPy* for data processing, *Keras* for building and training the model, and *matplotlib* for visualization.

5.4 Bi-LSTM

The implementation uses a Bidirectional Long Short-Term Memory (Bi-LSTM) model to predict stock prices based on historical data. The dataset used is loaded from a CSV file and preprocessed using *MinMaxScaler* to scale the closing prices between 0 and 1. The data is then split into training and testing sets, with 80% of the data used for training and the remaining 20% for testing. The training data is further split into input sequences of length 60 and corresponding output values.
The Bi-LSTM model is then built using the *Keras* library. The model architecture consists of two layers of 50 LSTM units each, with a dropout rate of 0.2 to prevent overfitting. The Bidirectional wrapper is used to make the LSTM layers process the input sequence both forwards and backwards, allowing the model to capture dependencies in both directions. The model is compiled using the Adam optimizer and mean squared error (MSE) loss function.
The model is then trained for three epochs on the training data with a batch size of 32. The model is used to generate recommendations on the data being tested once it has been trained. To determine the root mean squared error (RMSE), the projected values are inverse converted using the scaler and compared to the actual values.
In addition to the RMSE, the model's training and validation loss and mean absolute error (MAE) are also tracked and plotted using the model's history object. The model is retrained for two epochs with a larger batch size of 64, and the training and validation loss and MAE have been plotted again.
The final output of the implementation is two plots. The first plot shows the predicted and actual stock prices over time. The second plot shows the training and validation loss and MAE over the course of training. The plots provide a visual representation of the model's performance in predicting the stock prices and its training and validation error over time.
Overall, the implementation demonstrates the use of a Bi-LSTM model in predicting stock prices based on historical data. The model shows promise in accurately predicting stock prices, as demonstrated by the low RMSE and visually accurate predictions in the first plot. The training and validation loss and MAE plots also demonstrate the model's ability to learn and generalize to new data.

5.5 Restricted Boltzmann Machine

This section implements a Restricted Boltzmann Machine (RBM) to perform unsupervised learning on a dataset. RBMs are a type of artificial neural network that can be used for feature learning and dimensionality reduction.

The script first loads a dataset from a CSV file, preprocesses the data, scales the features, and splits the data into training and test sets. The RBM is defined with a certain number of visible and hidden units, and the weights and biases of the RBM are initialized. The energy function and negative log-likelihood are defined, and the optimizer is defined with a learning rate of 0.01. The RBM is trained for a certain number of epochs, with batches of data being fed through the model. The training loss is recorded for each epoch, and the weights, biases, and hidden units are plotted over time. Finally, the model is evaluated on the test set by calculating the mean squared error between the predicted and actual values.
The RBM is defined with a TensorFlow 2.0 backend, using the *Keras* API. The script uses other libraries such as *NumPy*, *Pandas*, and *Matplotlib* for data manipulation and visualization, as well as $scikit - learn$ for scaling and splitting the data.
The RBM contains a visible layer with a unit count equal to the number of features in the set of data and a hidden layer with a unit count defined by the user. The weights linking the visible and hidden layers are generated at random using a compressed normal distribution, and both layers' biases are set to zero. The energy function takes a visible unit vector and a hidden unit vector as input and computes the model's negative log-likelihood. The optimizer is used to reduce the negative log-likelihood, which optimizes the model's probability given the data.
The RBM is trained using batches of data, with the optimizer minimizing the negative log-likelihood of the model with respect to the weights and biases of the model. After each epoch of training, the training loss is recorded, and the weights, biases, and hidden units are plotted over time to monitor the training progress. Finally, the model is evaluated on the test set by computing the mean squared error between the predicted and actual values.

5.6 Deep Belief Network

This section implements a deep belief network (DBN) model implemented in TensorFlow using Python.
The data is first loaded from a CSV file and preprocessed by dropping unnecessary columns, converting date strings to Unix timestamps, and normalizing the features and labels. The data is then split into training and testing sets, and TensorFlow datasets are created for them.
The DBN model architecture is defined using the $tf.keras.Model$ API. The 'DBN' class contains a stack of dense layers with *ReLU* activation, followed by a single dense layer with linear activation for regression.
The loss function is defined as the mean squared error (MSE), and the gradient function is defined using $tf.GradientTape$. The mean absolute error (MAE) is the accuracy statistic.
The model is trained for a specified number of epochs using the Adam optimizer. The loss and accuracy values are stored in lists for each epoch and printed at the end of each epoch. Finally, the loss and accuracy values for each epoch are printed again.

Chapter 6

Result

In this study, we used six distinct machine learning models to assess and forecast the behavior of complex systems: Explainable AI, Q-Learning model, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep belief network. To assess the performance of these models, we employed three datasets of 7000, 10000, and 14000 data points, respectively.
The purpose of this research is to compare the efficacy of different models in order to discover which model works most effectively with each dataset. The models will be evaluated based on their performance.
The Explainable AI model is a newer technique that has gained traction in recent years because of its capability to explain how it makes judgments. The Q-Learning method is a reinforcement learning technique that has found widespread application in fields such as robotics and game theory. Deep learning models LSTM and Bi-LSTM have demonstrated performance in processing natural languages and recognition of speech challenges. Deep generative models such as the Restricted Boltzmann Machine and Deep belief network have been employed in unsupervised learning applications.
The results of this study will provide insights into the performance of these models in predicting complex systems and help us determine the best model for different types of datasets. This study will contribute to the growing body of research on machine learning and its applications in various fields.

6.1 Explainable AI

The implemented code aims to use the Decision Tree Regressor algorithm to predict the stock prices based on the given input parameters. The dataset used in this implementation contains stock market data from various countries like USA, UK, Russia, and Bangladesh with three different sizes: 7000, 10000, and 14000 data points. The model achieved 99% accuracy over the training and validation set.

6.1.1 Performance over 7000 data

The Decision Tree model's feature_importances_ property is used to determine the significance of each feature. These numbers show the relevance of each characteristic in making predictions. The feature importances are then shown using a bar plot built using *Matplotlib*, with the x-axis showing the feature names and the y-axis indicating their importance scores. The captured features for this dataset are shown in Figure 6.1.

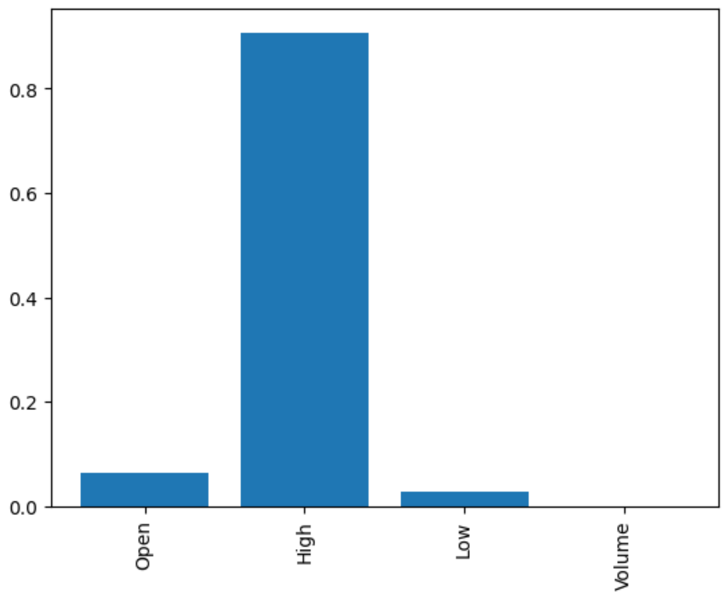


Figure 6.1: Feature Capture over 7000 data

The code includes two forms of decision tree visualizations. The first call to *plot_tree* provides a full picture of the whole decision tree, encompassing all its branches and nodes. The following *plot_tree* function call reduces the display to a maximum depth of three, resulting in a more compact depiction. The regression tree generated by the model for the 7000 dataset is shown in Figure 6.2.

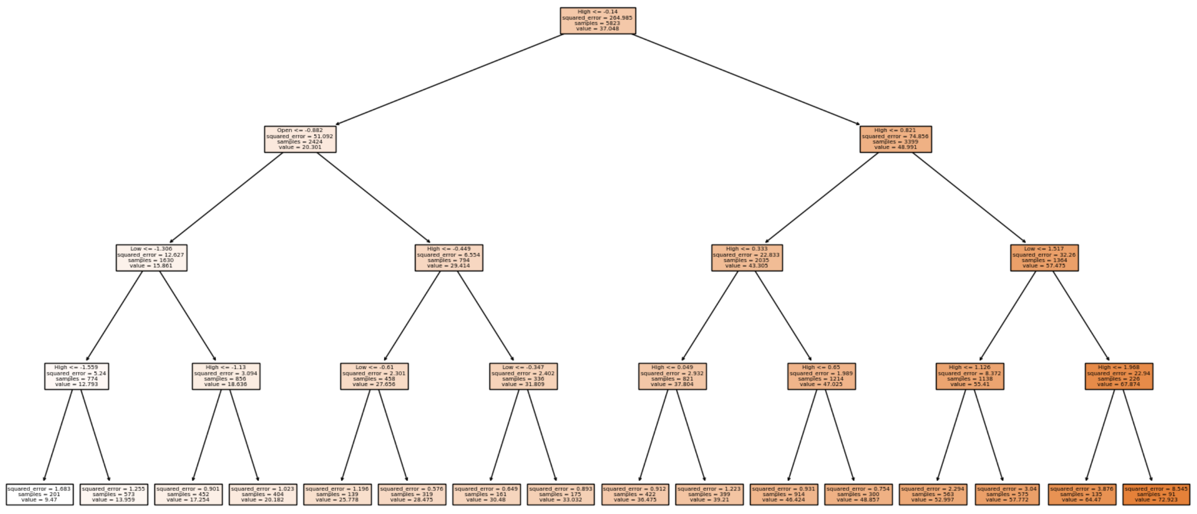


Figure 6.2: Regression Tree over 7000 data

6.1.2 Performance over 10000 data

The machine learning algorithm obtains more data points to extract information from as the dataset size grows, potentially leading to increased accuracy and more accurate feature significance assessments. A bigger dataset allows the model to catch additional trends and generalize to previously unknown data. As a result, the feature significance ratings may become more strong and representational of each feature's genuine relevance. The captured features for this dataset are shown in Figure 6.3.

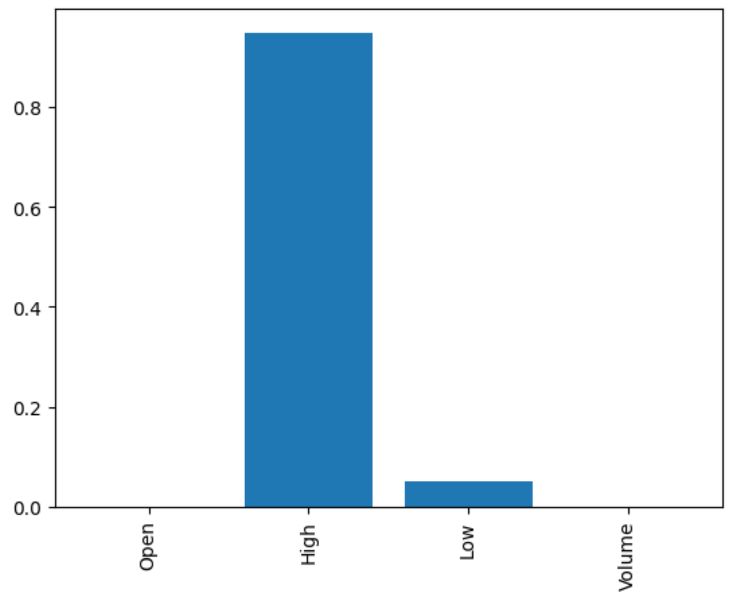


Figure 6.3: Feature Capture over 10000 data

A larger dataset often allows for improved model generalization. It lowers the danger of overfitting, which occurs when the model absorbs the training data rather than understanding the underlying patterns. A model that has been well-generalized is more likely to capture the real connections between features and the variable being studied, resulting in more accurate and dependable feature significance ratings. The regression tree generated by the model for the 10000 datasets is shown in Figure 6.4.

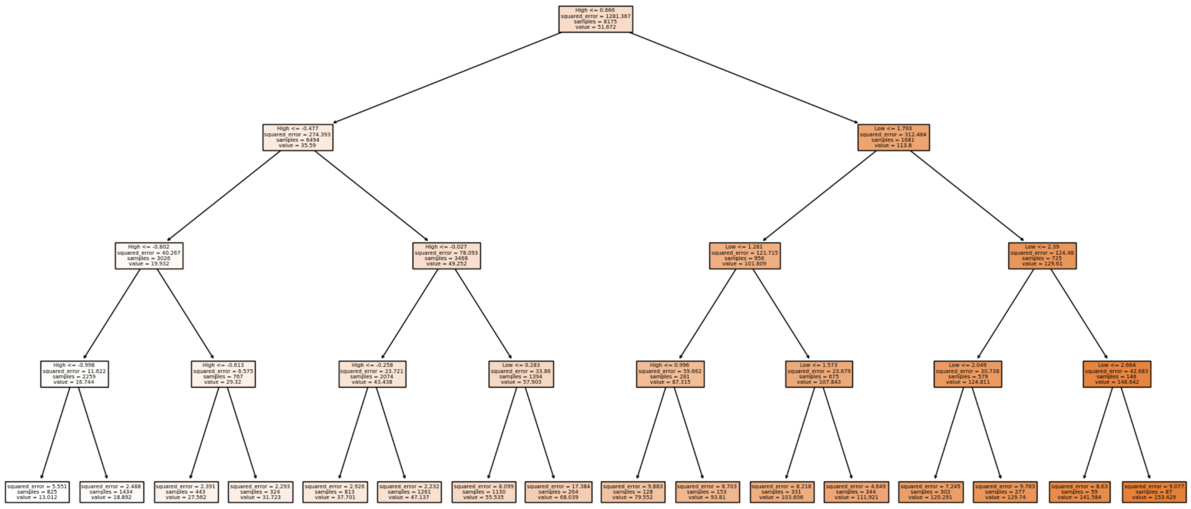

High <= 0.666 squared_error = 1281.367 samples = 8175 value = 51.672
High <= -0.477 squared_error = 274.393 samples = 6494 value = 35.59
Low <= 1.793 squared_error = 312.484 samples = 1681 value = 113.8
High <= -0.802 squared_error = 40.267 samples = 3026 value = 19.932
High <= -0.027 squared_error = 78.093 samples = 3468 value = 49.252
Low <= 1.281 squared_error = 121.715 samples = 956 value = 101.809
Low <= 2.39 squared_error = 124.48 samples = 725 value = 129.61
High <= -0.998 squared_error = 11.622 samples = 2259 value = 16.744
High <= -0.613 squared_error = 6.575 samples = 767 value = 29.32
High <= -0.258 squared_error = 23.721 samples = 2074 value = 43.438
Low <= 0.283 squared_error = 33.86 samples = 1394 value = 57.903
High <= 0.996 squared_error = 59.662 samples = 281 value = 87.315
Low <= 1.573 squared_error = 23.679 samples = 675 value = 107.843
Low <= 2.049 squared_error = 30.738 samples = 579 value = 124.811
Low <= 2.664 squared_error = 42.683 samples = 146 value = 148.642
squared_error = 5.551 samples = 825 value = 13.012
squared_error = 2.488 samples = 1434 value = 18.892
squared_error = 2.391 samples = 443 value = 27.562
squared_error = 2.293 samples = 324 value = 31.723
squared_error = 2.926 samples = 813 value = 37.701
squared_error = 2.232 samples = 1261 value = 47.137
squared_error = 8.099 samples = 1130 value = 55.535
squared_error = 17.384 samples = 264 value = 68.039
squared_error = 9.883 samples = 128 value = 79.552
squared_error = 8.703 samples = 153 value = 93.81
squared_error = 8.218 samples = 331 value = 103.606
squared_error = 4.649 samples = 344 value = 111.921
squared_error = 7.245 samples = 302 value = 120.291
squared_error = 9.783 samples = 277 value = 129.74
squared_error = 8.63 samples = 59 value = 141.584
squared_error = 9.077 samples = 87 value = 153.429

Figure 6.4: Regression Tree over 10000 data

6.1.3 Performance over 14000 data

Larger datasets, in general, help to maintain feature significance rankings. The influence of random changes and noise lessens as the dataset grows, enabling the model to zero in on more significant patterns. This consistency means that the relative relevance of traits is likely to remain relatively high as datasets get more extensive. However, further research must still confirm feature significance stability, including cross-validation or permutation significance. The captured features for this dataset are shown in Figure 6.5.

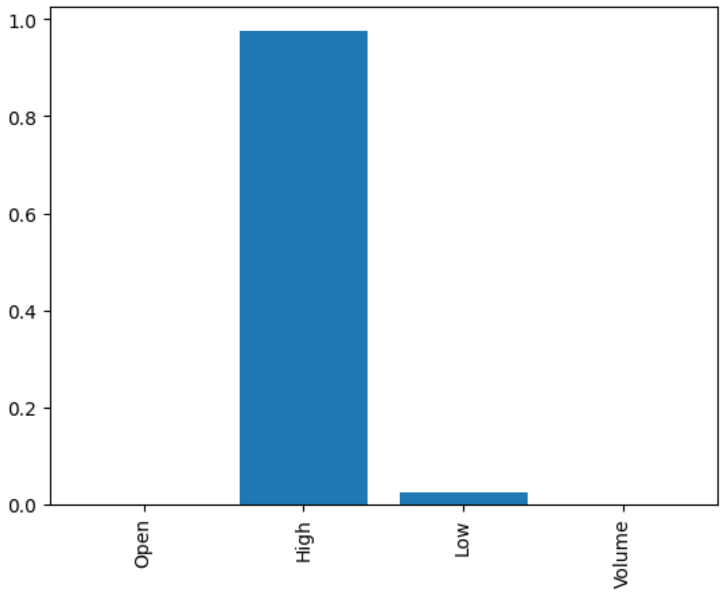
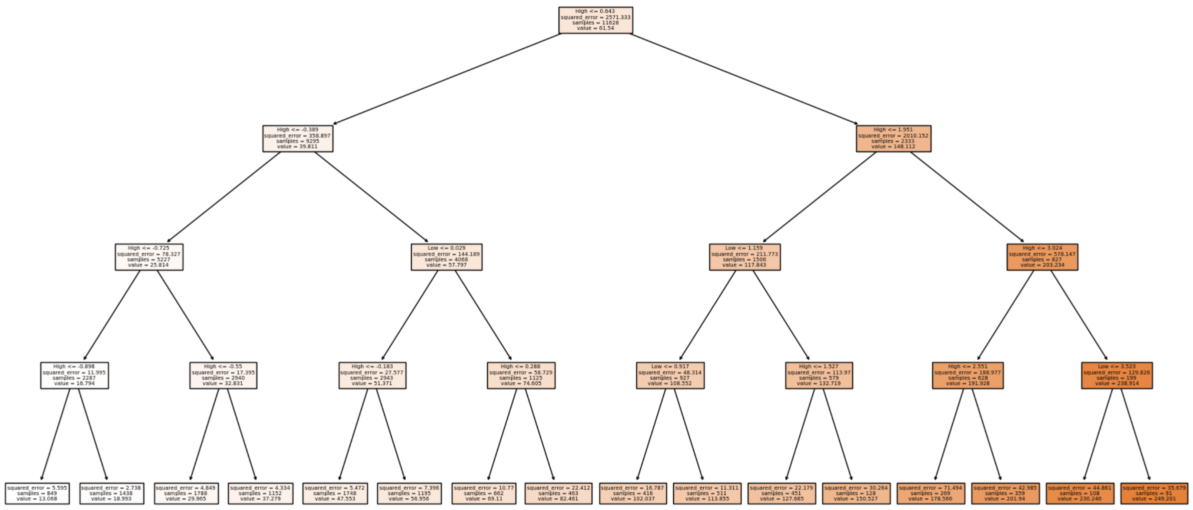


Figure 6.5: Feature Capture over 14000 data

Larger datasets may necessitate additional computational assets as well as time to develop and assess the model. Decision Tree scenarios, in particular, require more training time as the dataset becomes larger. It is critical to take into account the available computer power and modify resources accordingly. The regression tree generated by the model for the 14000 datasets is shown in Figure 6.6.



High <= 0.643 squared_error = 2571.333 samples = 11628 value = 61.54
High <= -0.389 squared_error = 358.897 samples = 9295 value = 39.811
High <= 1.951 squared_error = 2010.152 samples = 2333 value = 148.112
High <= -0.725 squared_error = 78.327 samples = 5227 value = 25.814
Low <= 0.029 squared_error = 144.189 samples = 4068 value = 57.797
Low <= 1.159 squared_error = 211.773 samples = 1506 value = 117.843
High <= 3.024 squared_error = 578.147 samples = 827 value = 203.234
High <= -0.898 squared_error = 11.995 samples = 2287 value = 16.794
High <= -0.55 squared_error = 17.395 samples = 2940 value = 32.831
High <= -0.183 squared_error = 27.577 samples = 2943 value = 51.371
High <= 0.288 squared_error = 58.729 samples = 1125 value = 74.605
Low <= 0.917 squared_error = 48.314 samples = 927 value = 108.552
High <= 1.527 squared_error = 113.97 samples = 579 value = 132.719
High <= 2.551 squared_error = 188.977 samples = 628 value = 191.928
Low <= 3.523 squared_error = 129.826 samples = 199 value = 238.914
squared_error = 5.595 samples = 849 value = 13.068
squared_error = 2.738 samples = 1438 value = 18.993
squared_error = 4.849 samples = 1788 value = 29.965
squared_error = 4.334 samples = 1152 value = 37.279
squared_error = 5.472 samples = 1748 value = 47.553
squared_error = 7.396 samples = 1195 value = 56.956
squared_error = 10.77 samples = 662 value = 69.11
squared_error = 22.412 samples = 463 value = 82.461
squared_error = 16.787 samples = 416 value = 102.037
squared_error = 11.311 samples = 511 value = 113.855
squared_error = 22.179 samples = 451 value = 127.665
squared_error = 30.264 samples = 128 value = 150.527
squared_error = 71.494 samples = 269 value = 178.566
squared_error = 42.985 samples = 359 value = 201.94
squared_error = 44.861 samples = 108 value = 230.246
squared_error = 35.679 samples = 91 value = 249.201

Figure 6.6: Regression Tree over 14000 data

6.1.4 Prediction

On the test set, the model's accuracy is calculated using the scoring method of the TransformedTargetRegressor. The prediction's coefficient of determination is represented by the score. The loss is calculated as well as the mean squared error (MSE) between the actual and anticipated values. Subsequently using Matplotlib, a scatter plot is constructed to demonstrate the connection between the actual outcomes (y_test) and the values that were predicted (y_pred). The x-axis shows the actual values, while the y-axis shows the expected values. Each scatter plot point represents a single data point from the experimental data set. The model's prediction is shown in the scatter plot in Figure 6.7.

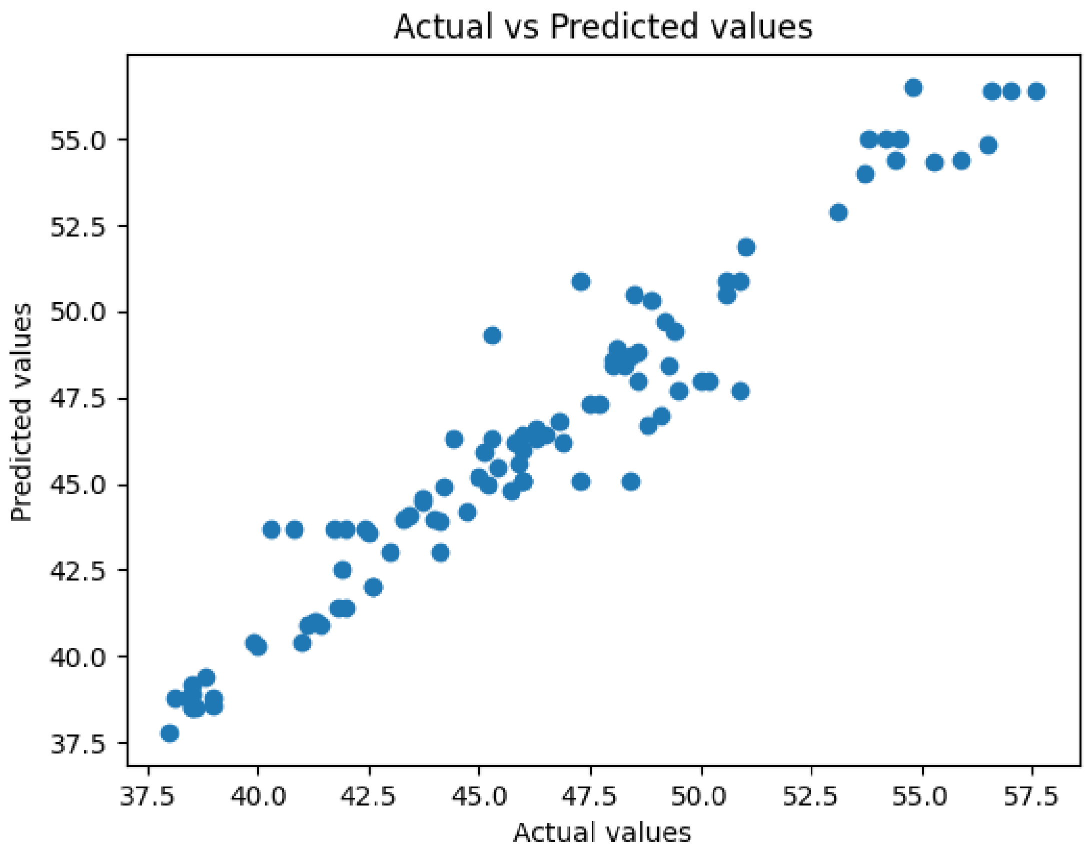


Figure 6.7: Actual vs Prediction

The code's charting section gives useful insights regarding the Decision Tree Regression model. The feature significance plot identifies the factors that are most significant for forecasting the target variable. A higher significance score denotes a greater influence on the forecasts. This data can be used to influence feature selection and additional research.
The decision tree visualization depicts the model's structure and decision-making process. Each node indicates a feature and its related split criterion, which leads to the following nodes or leaf nodes with anticipated values. The tree may be interpreted by readers to understand how various attributes influence the model's results. Readers can obtain a better grasp of the machine learning model's behavior, identify essential aspects, and evaluate the model's interpretability by evaluating these graphs.

6.2 Q-Learning Model

For this part, we will show the net worth, reward, state, and total reward gained by the model of the three distinctive datasets. To conduct the trading simulation utilizing the Q-Learning agent, use the run_simulation function. The function loops over each time step, acquiring the current state, deciding on an action based on the agent's decisions, modifying the agent's Q-values, and computing the reward. The net worth of the agent is monitored over time and saved in the net_worth_history list.

6.2.1 Performance over 7000 dataset

Following the completion of the simulation, the net_worth_history collection is visualized using Matplotlib. The x-axis is time (in days), while the y-axis is the agent's net value. At each time step, each point on the graph reflects the agent's net wealth. The net worth of the 7000 dataset generated by the model is shown in Figure 6.8.

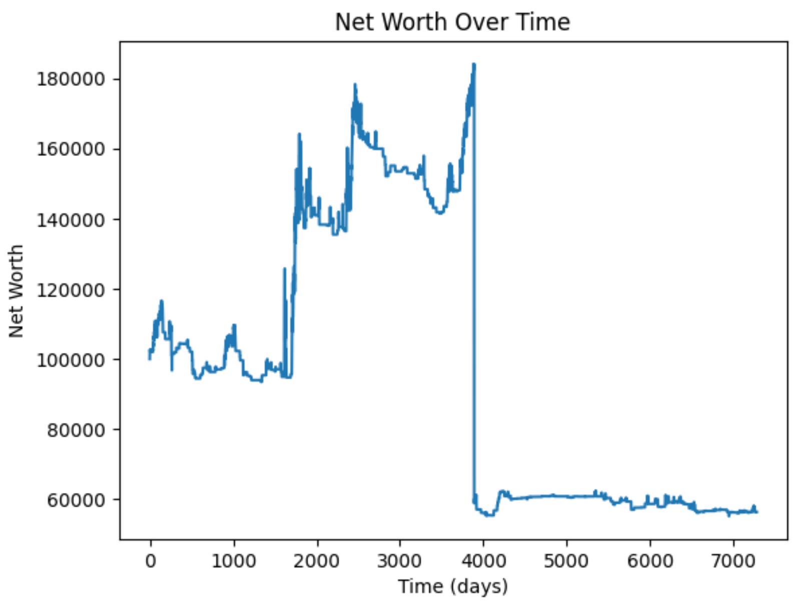


Figure 6.8: Net Worth over 7000 dataset

The get_reward function is designed to compute the instant reward depending on the current state and action selected. The reward is set to one if the state is 0 (price decline) and the action taken is 2 (buy low, sell high). The reward is set to one if the state is one (price raised) and the action performed is one (buy high, sell low). All other activities are rewarded with 0. The reward over time is shown in Figure 6.9.

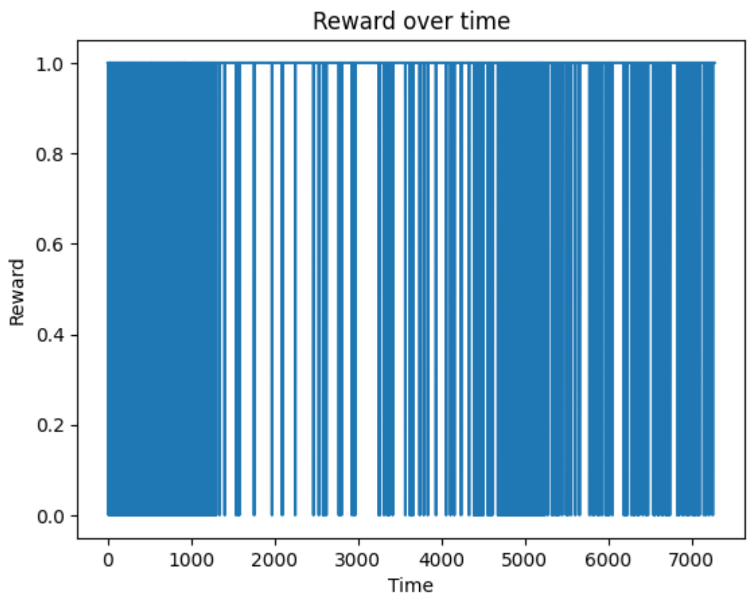


Figure 6.9: Rewards over 7000 dataset

The get_state method determines the present state based on pricing data at a particular time step. If the closing price is less than the initial price, the state is set to 0 (price reduction). When the closing price exceeds the initial price, the status is changed to 1 (price raised). Otherwise, if the costs stay unchanged, the status is set to 2 (price stayed unchanged). The state over time is shown in Figure 6.10.

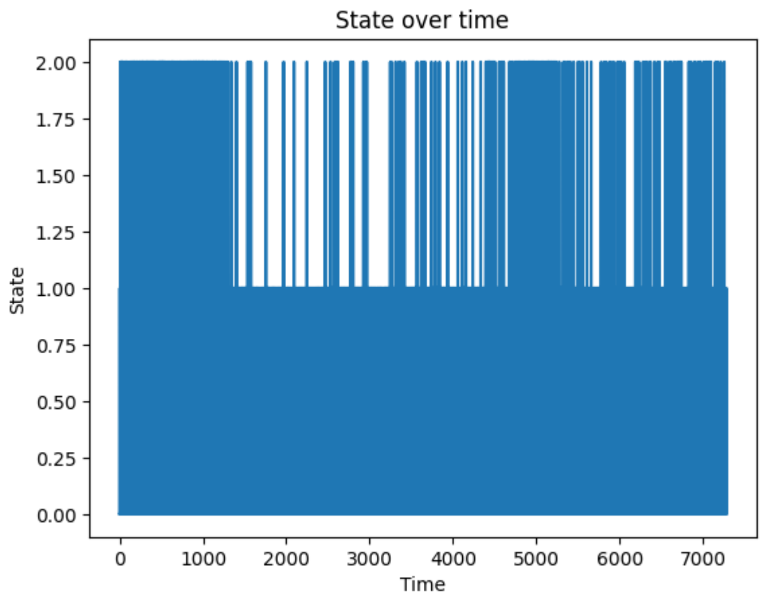


Figure 6.10: State over 7000 dataset

The total rewards per episode are shown in Figure 6.11.

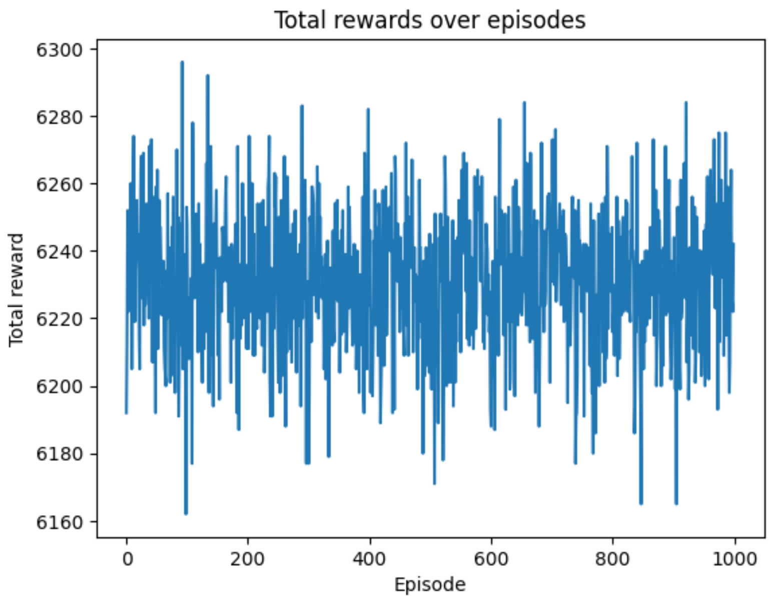


Figure 6.11: Total Rewards per Episode over 7000 dataset

6.2.2 Performance over 10000 datasets

The net worth curve's overall trend reveals whether or not the agent's trading technique is lucrative. An expanding curve indicates that the agent's trade actions provide positive returns, whereas a declining curve indicates negative returns. The volatility and swings in the net worth graph indicate the agent's trading efficiency variability. Sharp spikes or dips imply large gains or losses within certain time periods. The magnitude of these changes might reveal information about the risk associated with the approach to trading. The net worth of the 10000 datasets generated by the model is shown in Figure 6.12.

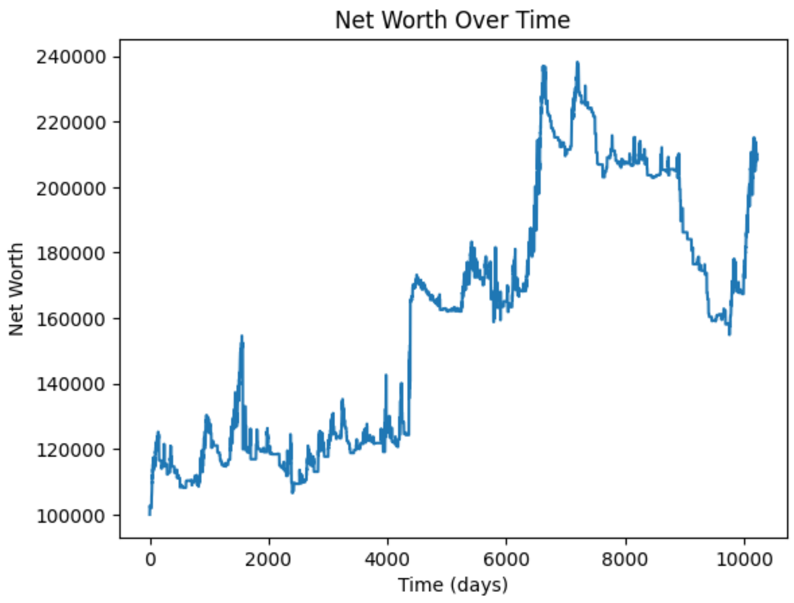


Figure 6.12: Net Worth over 10000 datasets

Matplotlib is used to plot the total prizes for each episode after the training process. The x-axis shows the number of episodes, while the y-axis reflects the total prize earned during the episode. This plot aids in visualizing the agent's learning progress across episodes. The reward over time is shown in Figure 6.13.

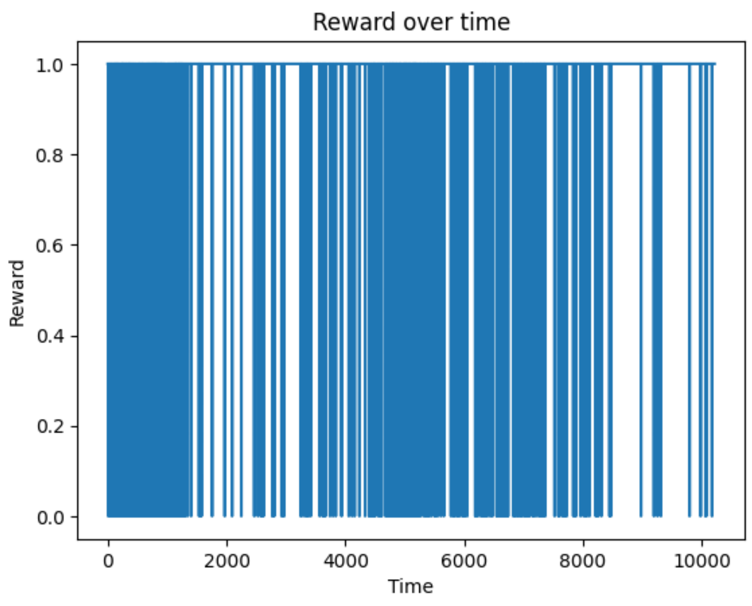


Figure 6.13: Rewards over 10000 dataset

Iterating over the complete dataset numerous times is often required while training a Q-Learning agent. Training time might grow correspondingly with more datasets. More episodes may be required for the agent to converge and learn optimal tactics. The state over time is shown in Figure 6.14.

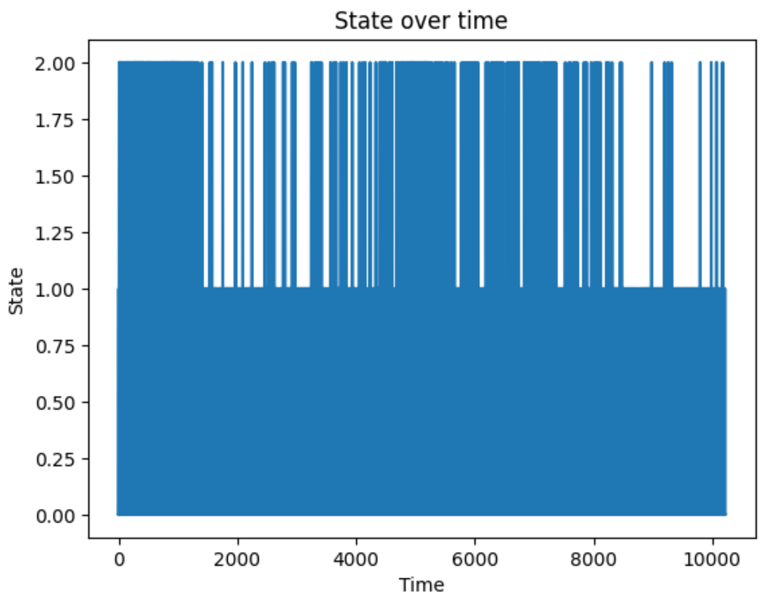


Figure 6.14: State over 10000 datasets

The total rewards per episode are shown in Figure 6.15.

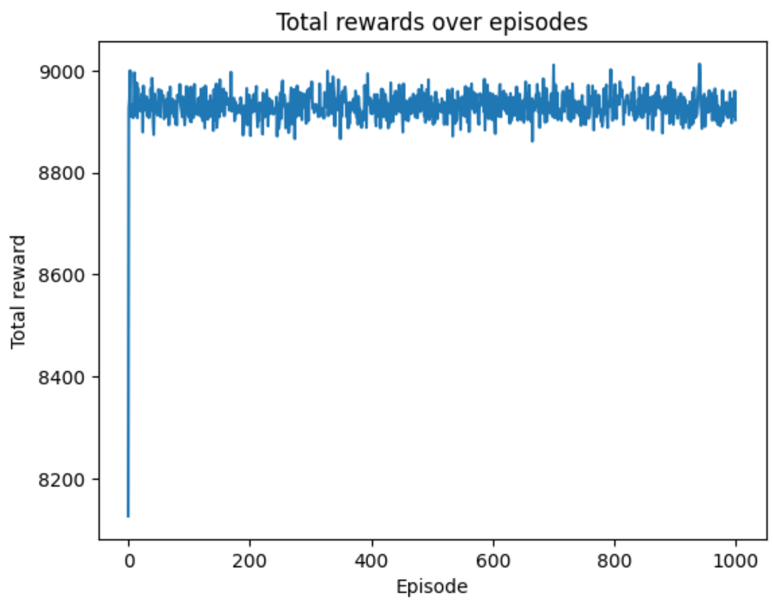


Figure 6.15: Total Rewards per Episode over 10000 dataset

6.2.3 Performance over 14000 datasets

The net worth curve may be used to assess the efficacy of the trading strategy employed by the Q-Learning agent. A continually ascending curve suggests a successful approach, however, a static or dropping curve may indicate that the agent's learning process has to be adjusted or improved. The net worth curve may also be used to analyze various trading techniques or to compare the agent's performance to a standard. Readers may evaluate the net worth rate of expansion and instability of alternative methods, as well as the agent's performance, by putting numerous curves on the identical graph. The net worth of the 14000 datasets generated by the model is shown in Figure 6.16.

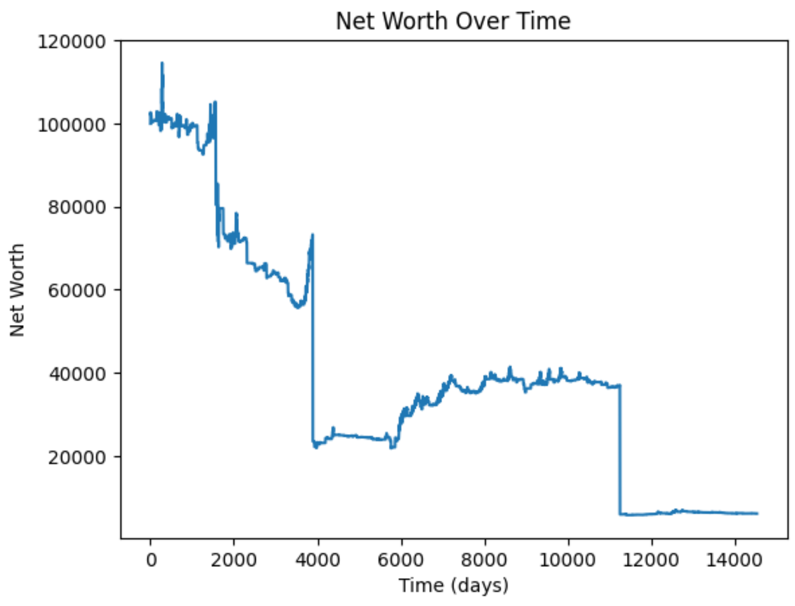


Figure 6.16: Net Worth over 14000 dataset

The reward over time is shown in Figure 6.17.

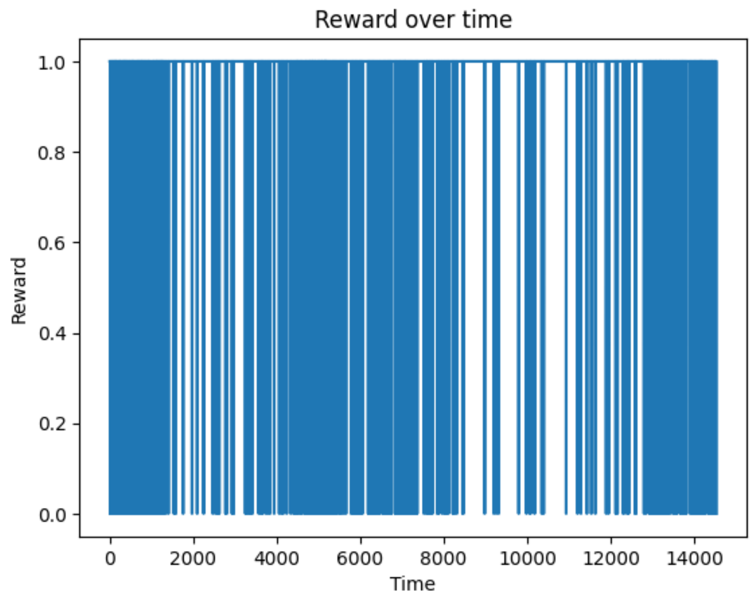


Figure 6.17: Rewards over 14000 dataset

The test rewards and states are displayed independently over time. The test rewards plot displays the rewards at each time step, indicating the agent's profitability throughout the test phase. The test states plot depicts the agent's impression of market circumstances as the state changes over time. The state over time is shown in Figure 6.18.

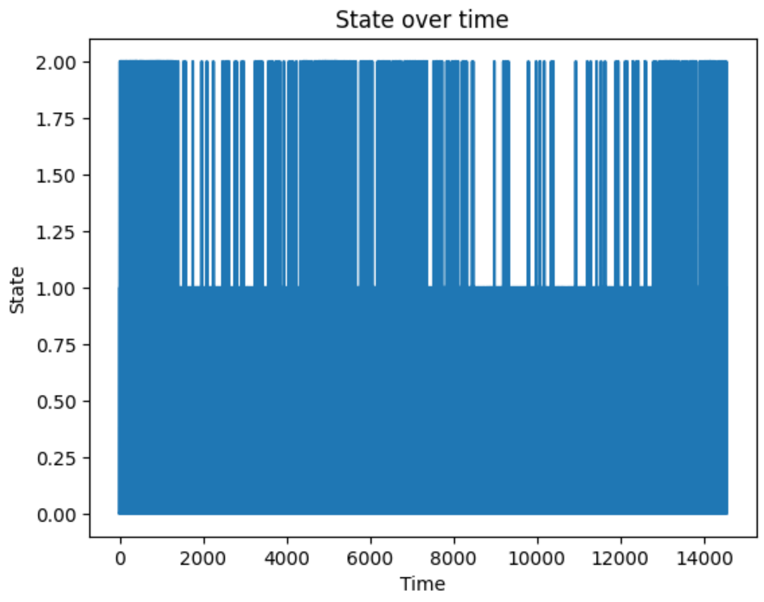


Figure 6.18: State over 14000 datasets

The performance of the trained agent is assessed using test data. The agent chooses behaviors depending on the Q-values it has learned and is rewarded accordingly. The test_rewards list stores the prizes gained during the testing period. The total rewards per episode are shown in Figure 6.19.

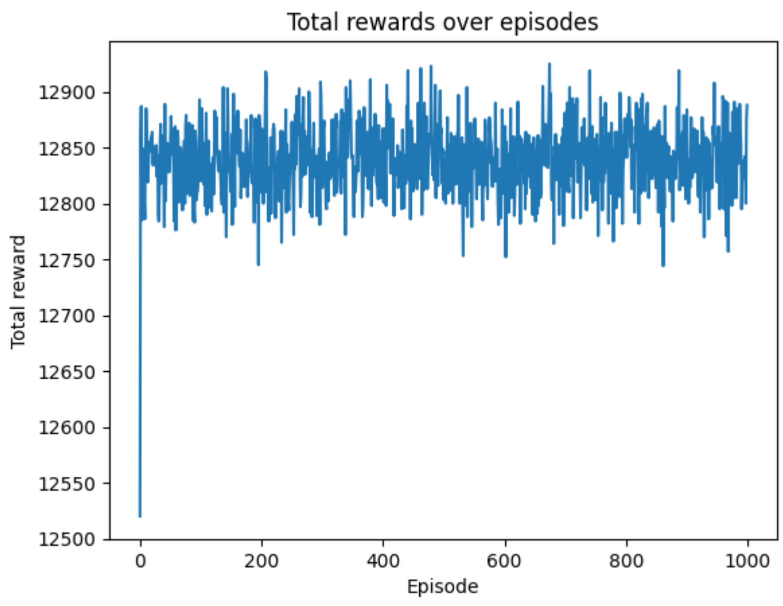


Figure 6.19: Total Rewards per Episode over 14000 dataset

6.3 LSTM

In this part, we will be discussing the prediction result generated by the LSTM models over three datasets and see the differences between them. The blue line is the original dataset prior to scaling. It displays the exact closing prices. The anticipated values for the training set are represented by the orange line. It demonstrates how effectively the model learns to suit the training data. The projected values for the test set are shown by the green line. It demonstrates how effectively the model can extrapolate and anticipate data that has not yet been observed. The visualization begins by generating an empty array with the same dimensions as the scaled data. The training set's anticipated values are allocated to the relevant spots in the empty array. The projected values for the test set are additionally allocated to the array's corresponding locations. Finally, the original dataset is shown alongside the projected values for the training set and the values that were anticipated for the test set.

6.3.1 Performance over 7000 dataset

The generated graph shows the actual vs prediction and the validation. The blue color informs the actual value, the orange color shows the prediction and the green color represents the validation. The model reflects the trends and patterns that appear in the data through the comparison of the blue line to the orange and green lines. The projected lines should ideally closely reflect the original data.

6.3.2 Performance over 10000 datasets

More trained data increasing the quantity of the dataset gives the model additional historical knowledge, potentially enhancing its capacity to detect trends and make precise forecasts. Longer Training Time: Because there are a greater number of samples to analyze with a greater amount of data, training the LSTM model might

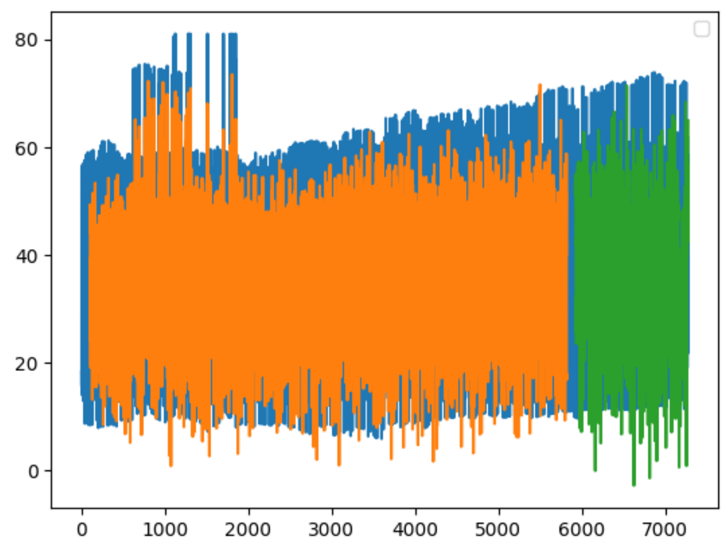


Figure 6.20: Actual vs Prediction vs Validation over 7000 dataset

take longer. The training time has risen proportionately to the quantity of the dataset. The generated graph shows the actual vs prediction and the validation. The blue color informs the actual value, the orange color shows the prediction and the green color represents the validation.

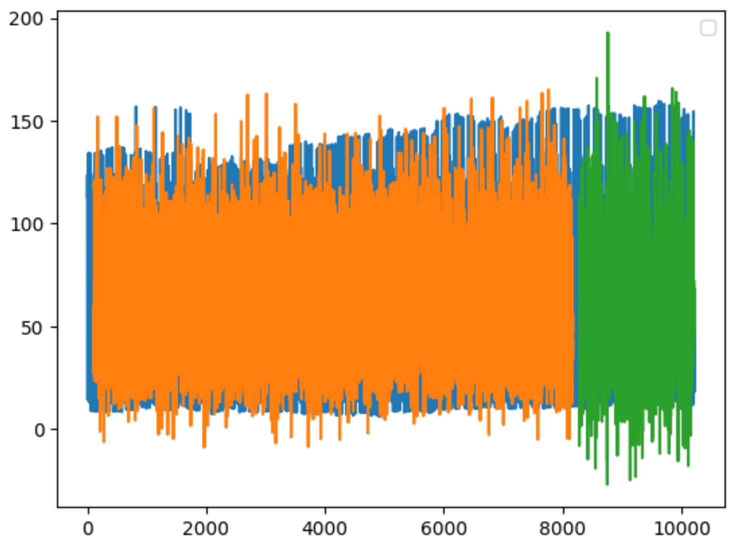


Figure 6.21: Actual vs Prediction vs Validation over 10000 dataset

6.3.3 Performance over 14000 datasets

A bigger dataset helps the model in generalizing to previously unknown data. It gives a more diversified set of patterns and variances, allowing the model to learn more robust time series representations. The generated graph shows the actual vs prediction and the validation. The blue color informs the actual value, the orange color shows the prediction and the green color represents the validation.

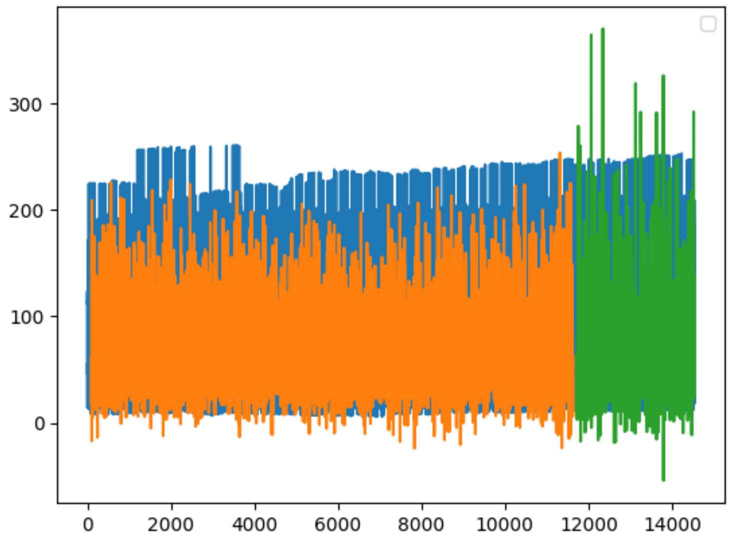


Figure 6.22: Actual vs Prediction vs Validation over 14000 dataset

6.4 Bi-LSTM

In this part, we will be discussing the prediction result generated by the Bi-LSTM models over three datasets and see the differences between them. The graphic shows two lines: The blue line depicts the test set's real stock values. The orange line reflects the model's expected stock values. By comparing the blue and orange lines, you can determine how effectively the model forecasts stock prices. In an ideal world, the forecast line would closely track the actual prices.

6.4.1 Performance over 7000 dataset

The generated graph shows the actual vs prediction. The blue color informs the actual value, and the orange color shows the prediction. The code also provides a training and validation loss plot as well as a training and validation MAE (Mean Absolute Error) plot. These charts give information about the algorithm's training progress and performance on unknown data. The training loss and MAE represent how well the model fits the training data, whereas the validation loss and MAE show how effectively the model extends to the validation data.

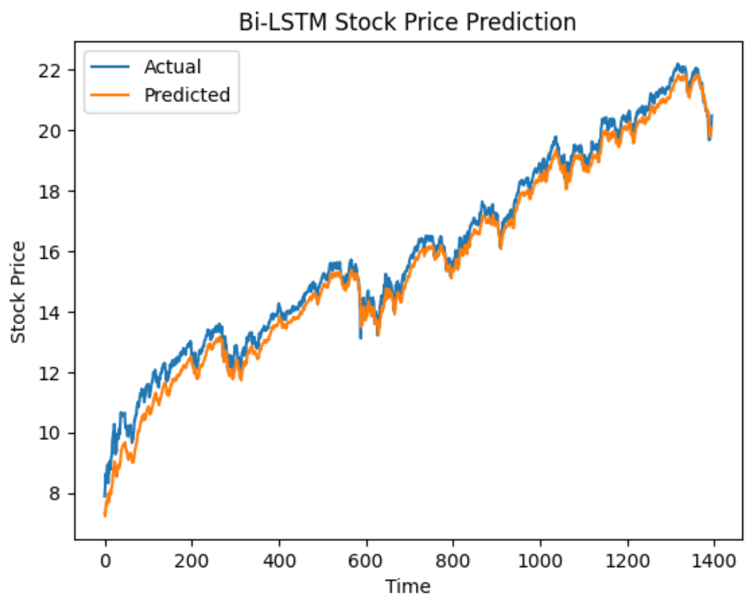


Figure 6.23: Actual vs Prediction over 7000 dataset

6.4.2 Performance over 10000 datasets

Increasing the dataset size adds more historical knowledge to the model, potentially boosting its capacity to detect trends and make accurate predictions. Because there are more samples to analyze with a bigger information set, developing the Bidirectional LSTM model might take longer. The training time will increase proportionately to the quantity of the dataset. The generated graph shows the actual vs prediction. The blue color informs the actual value, and the orange color shows the prediction.

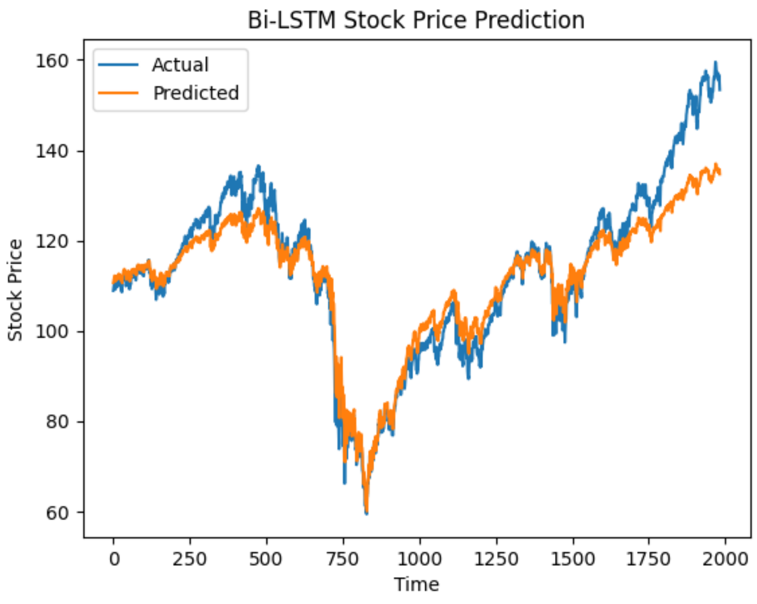


Figure 6.24: Actual vs Prediction over 10000 dataset

6.4.3 Performance over 14000 datasets

The generated graph shows the actual vs prediction. The blue color informs the actual value, and the orange color shows the prediction. A richer dataset can assist the model in generalizing to previously unknown data. It gives the model a wider range of trends and variances, allowing it to develop more accurate representations of stock price patterns.

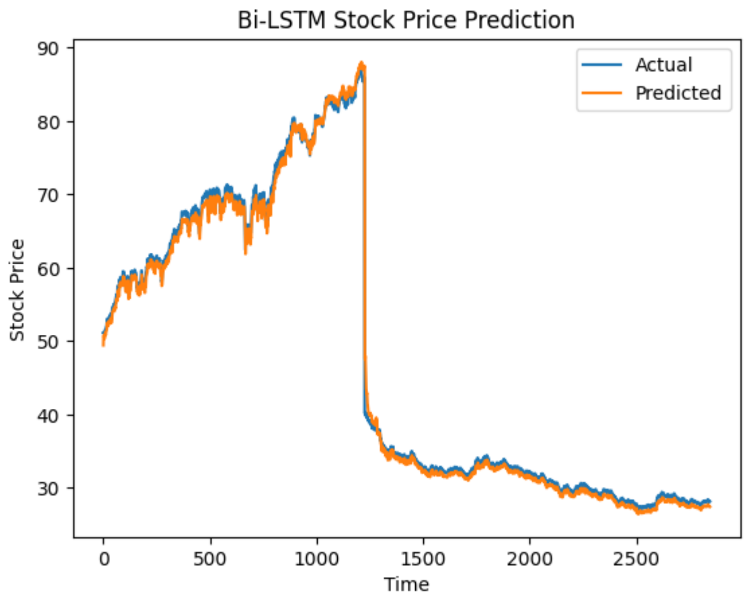


Figure 6.25: Actual vs Prediction over 14000 dataset

6.5 Restricted Boltzmann Machine

In this section, we will see the performance of the model in terms of various aspects such as actual vs prediction for a single time step, hidden bias over time, hidden unit over time, visible bias over time, visible unit over time, weight for a single hidden unit, and weight over time for different datasets.

6.5.1 Performance over 7000 dataset

This graphic examines the actual and anticipated stock prices from the test data for a single time step. The x-axis depicts time, while the y-axis depicts stock price values. The blue line depicts real stock values, while the orange line depicts expected market prices. The gray region between both lines represents the difference or mistake in pricing between the actual and forecasted prices. The plot gives an illustration of the model's capability to reliably anticipate stock prices. Here we have first observed the actual vs prediction for a single time step in Figure 6.26.
This graphic depicts the evolution of the biases of the RBM system's hidden units over time. The hidden units are represented by the x-axis, while the bias values are shown by the y-axis. The bias level for each concealed unit is represented by a different point on the plot. The figure enables the monitoring of the fluctuation

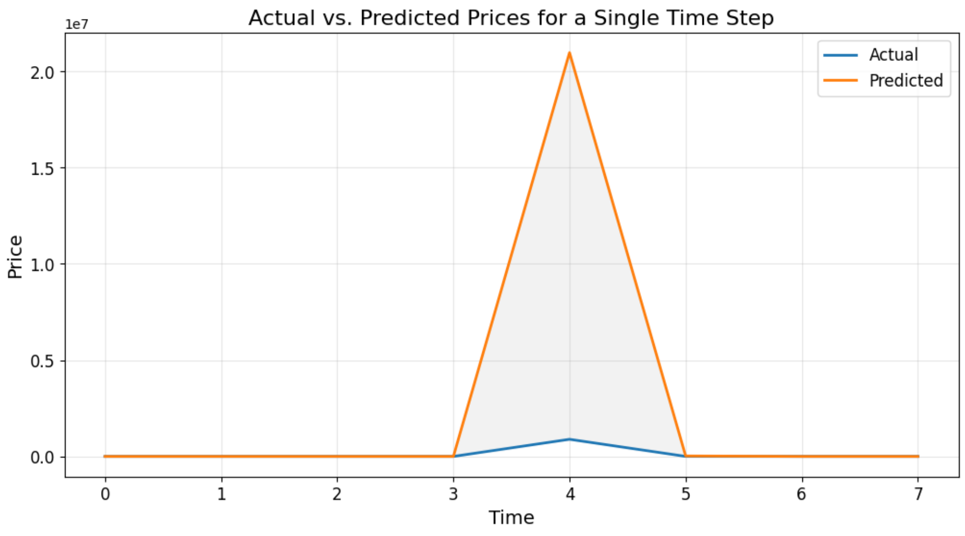


Figure 6.26: Actual vs Prediction over 7000 dataset

in bias values, which can alter the activation sequences of the RBM's hidden units. Then we observe the hidden bias in Figure 6.27.

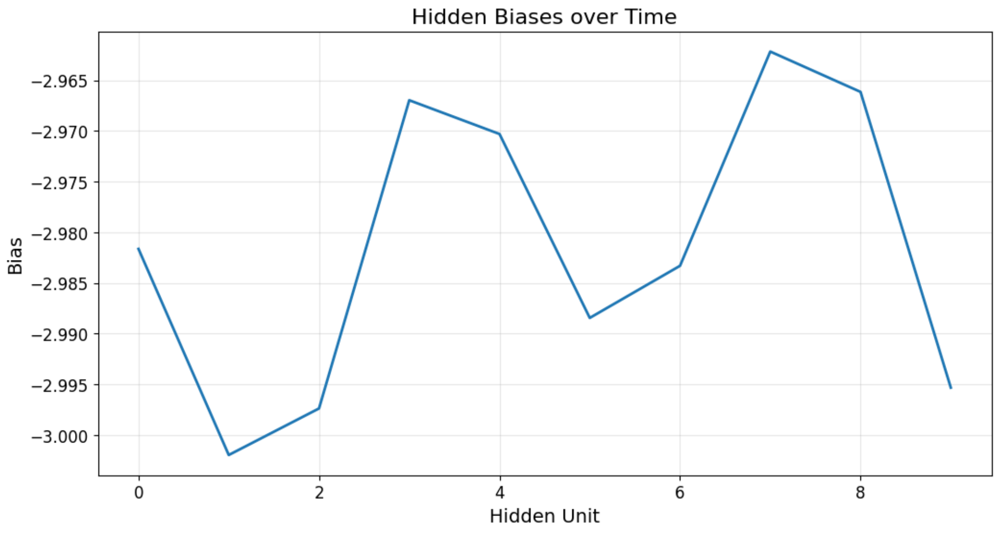


Figure 6.27: Hidden Bias over 7000 dataset

This graph depicts the activation of the RBM model hidden units over time. The x-axis indicates time, while the y-axis displays the activation values of the hidden units. Each line in the figure represents the activation values of a different hidden unit. The graphic facilitates observation of the changes and trends in the hidden unit activations over the data. Next, we see the hidden unit over time in Figure 6.28.

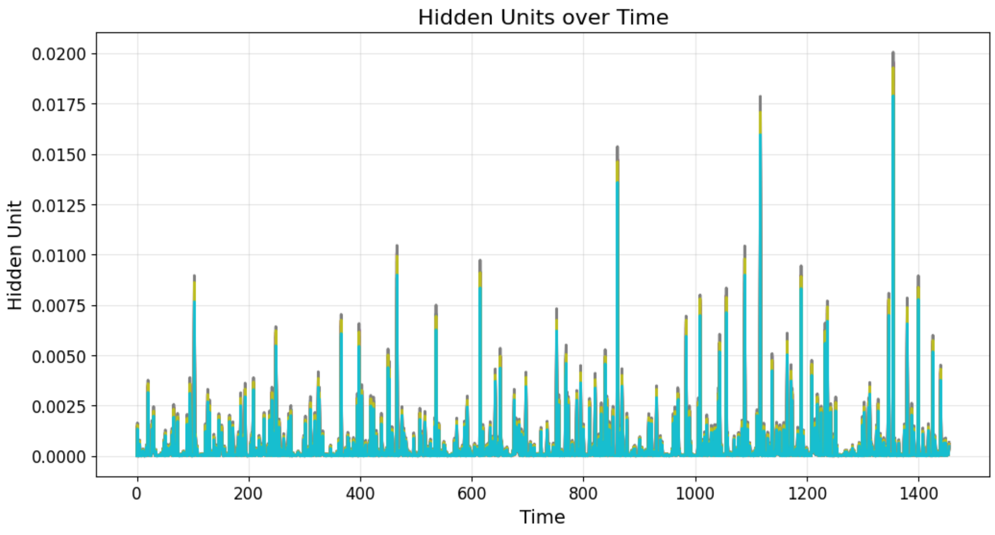


Figure 6.28: Hidden Unit over 7000 dataset

This graphic depicts the evolution of the biases of the RBM algorithm's visible units throughout epochs. The visible units are represented by the x-axis, while the bias values are shown by the y-axis. Each plot point indicates the bias value for a single visual unit. The graphic shows how the biases of visible units affect the RBM's creative process. After that, we illustrate the visible bias over time in the Figure

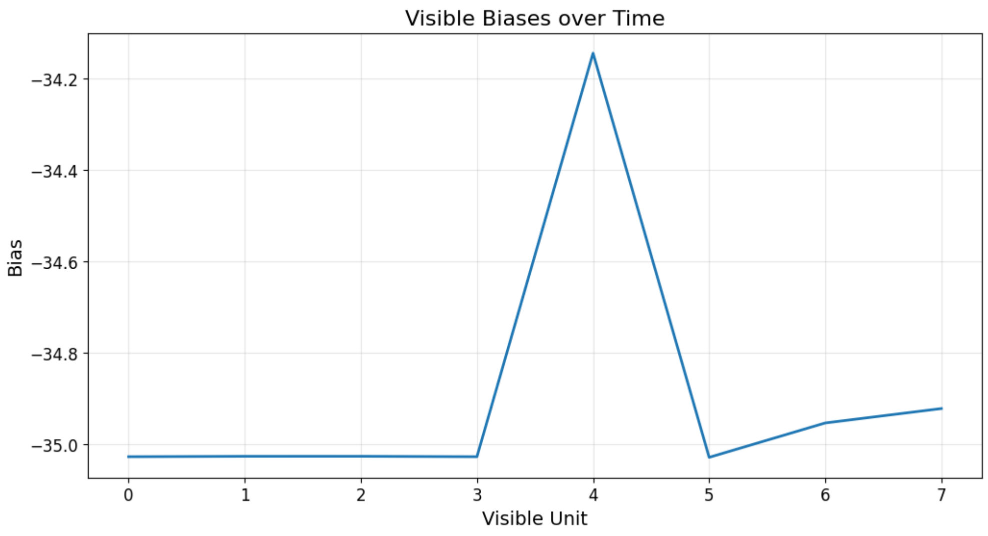


Figure 6.29: Visible Bias over 7000 dataset

The activation of visible units in the RBM architecture over a period of time is depicted in this graphic. The x-axis indicates time, while the y-axis displays the activation values of the visible units. Each line in the figure represents the activation levels of a particular visible unit. The graphic enables the examination of patterns and oscillations in the visible unit activations throughout the generating process of the RBM. Then, we observe the visible unit for the model.

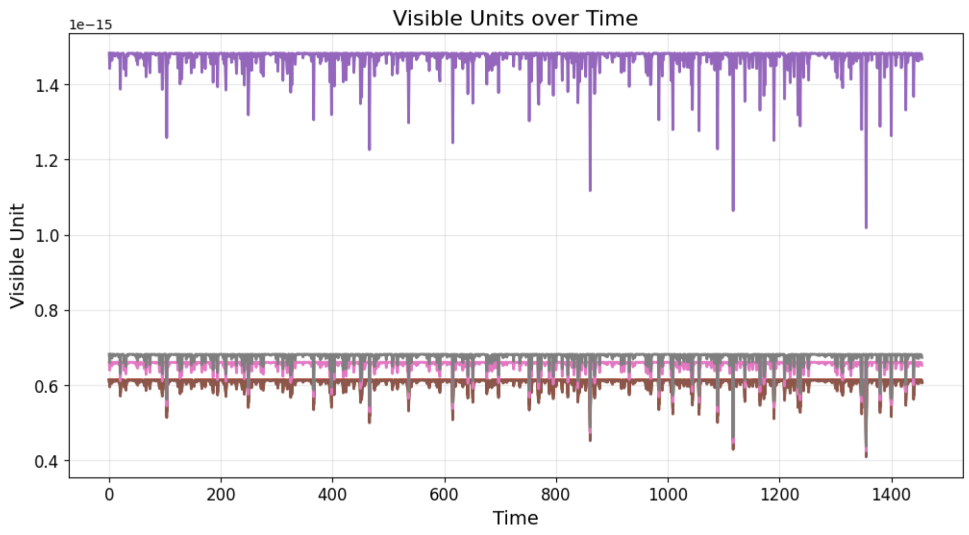


Figure 6.30: Visible Unit over 7000 dataset

This graphic depicts the evolution of the weights of the RBM model's links between visible and hidden units over time. The hidden units are represented by the x-axis, while the weight values are shown by the y-axis. Every segment in the figure represents the weights corresponding to a particular hidden unit. The figure allows you to see how the RBM's weights change over time and how they affect the model's learning procedure. Weight changes reveal how the RBM adapts its internal schema during training. Next, We will observe the weight for the single hidden unit and the overall weight over time in the following two figures.

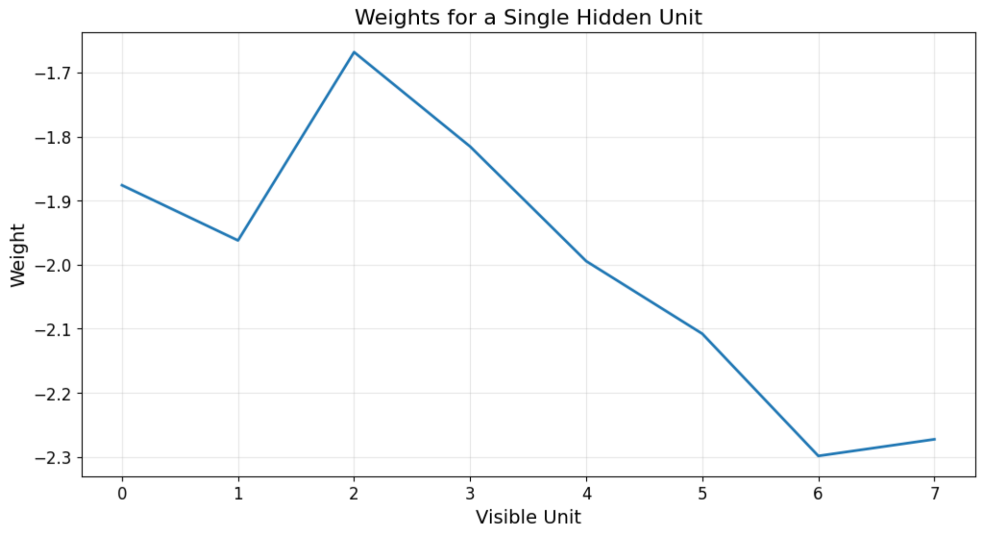


Figure 6.31: Weight for a single Hidden unit over 7000 dataset

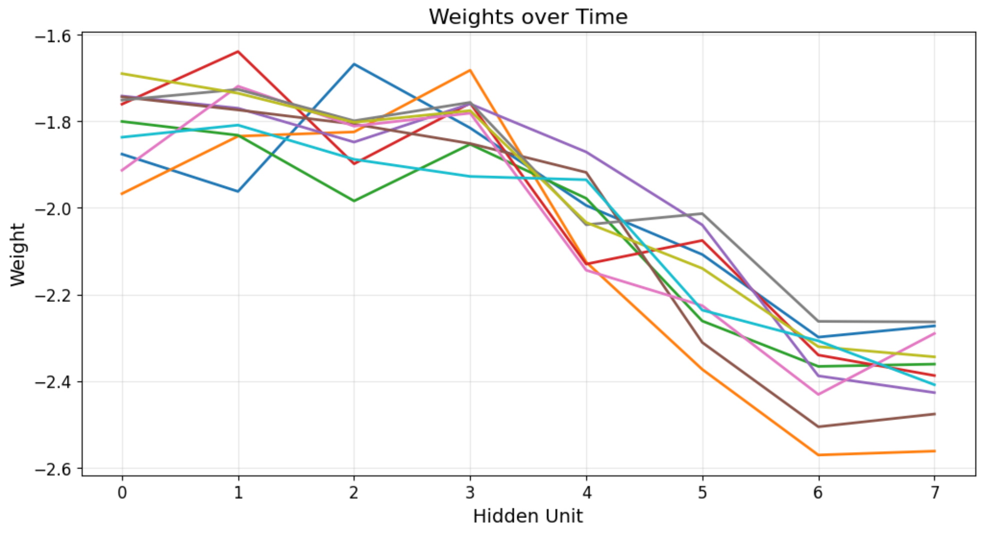


Figure 6.32: Weight over time over 7000 dataset

6.5.2 Performance over 10000 dataset

Here we have first observed the actual vs prediction for a single time step in Figure 6.33.

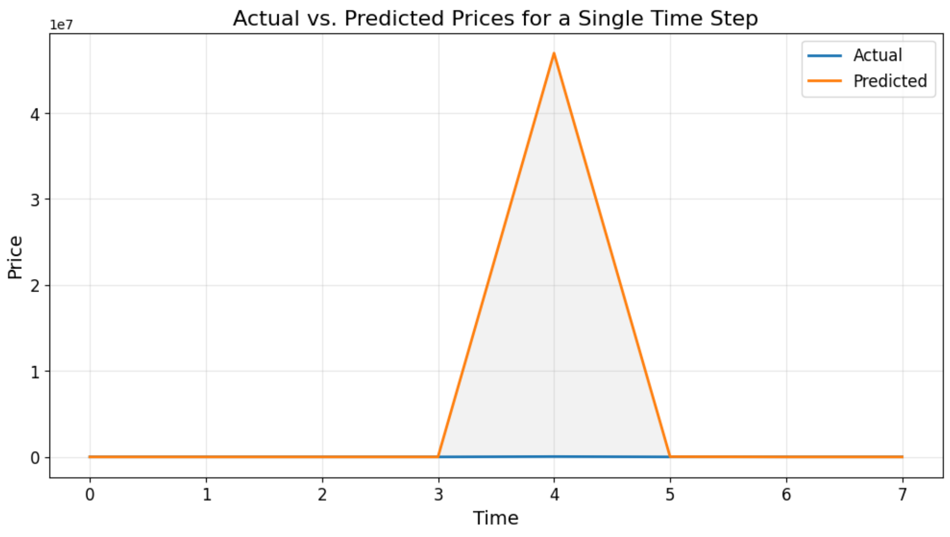


Figure 6.33: Actual vs Prediction over 10000 dataset

Then we observe the hidden bias in Figure 6.34.

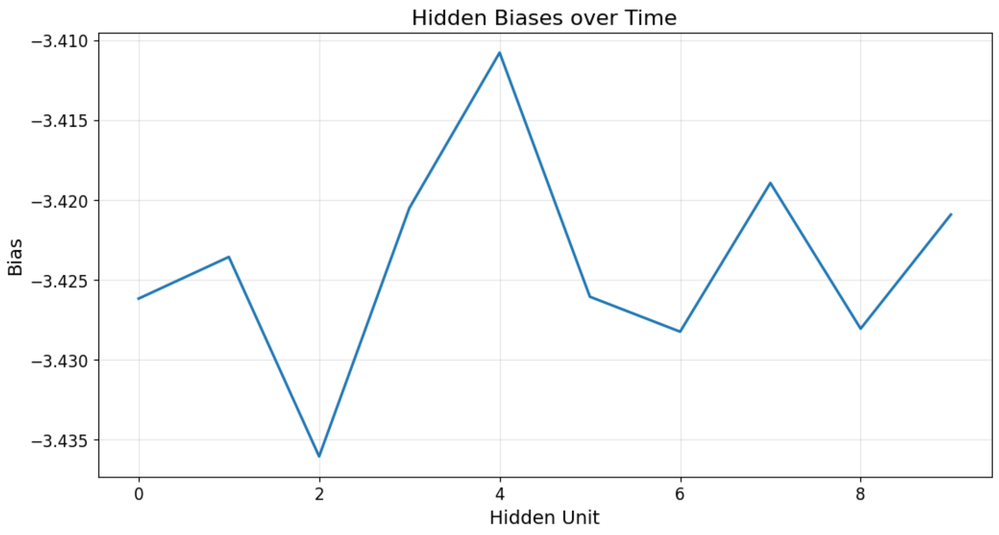


Figure 6.34: Hidden Bias over 10000 dataset

Next, we see the hidden unit over time in Figure 6.35.

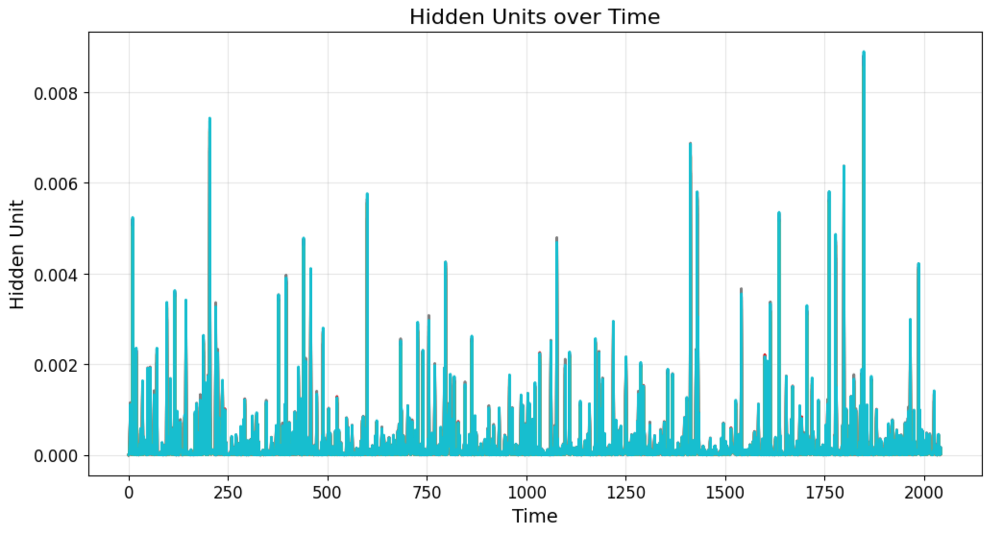


Figure 6.35: Hidden Unit over 10000 dataset

After that, we illustrate the visible bias over time in the Figure

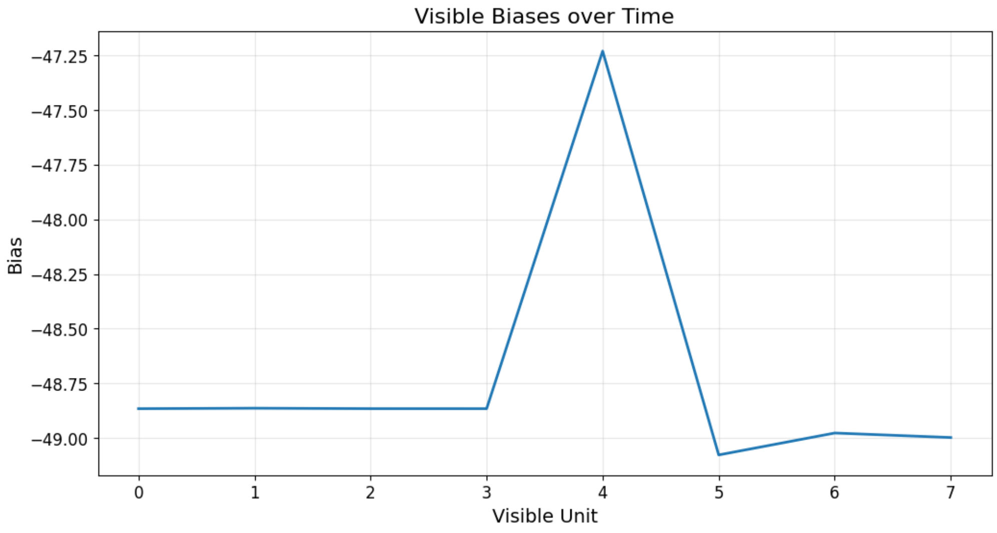


Figure 6.36: Visible Bias over 10000 dataset

Then, we observe the visible unit for the model.

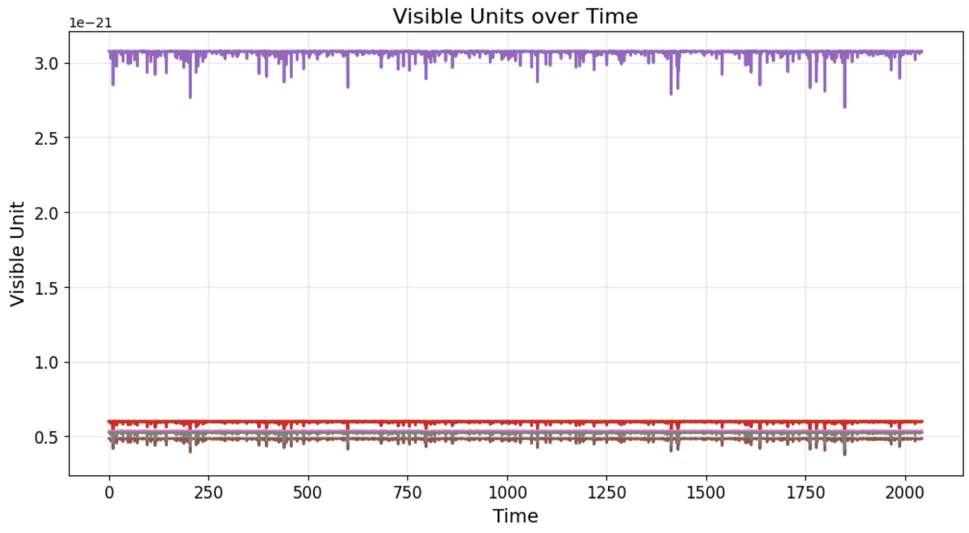


Figure 6.37: Visible Unit over 10000 dataset

Next, We will observe the weight for the single hidden unit and the overall weight over time in the following two figures.

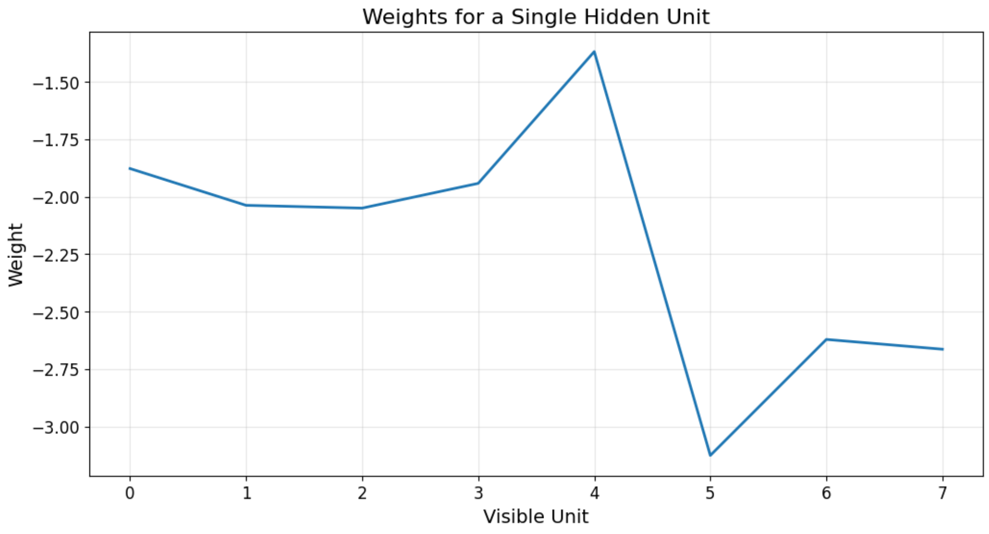


Figure 6.38: Weight for a single Hidden unit over 10000 dataset

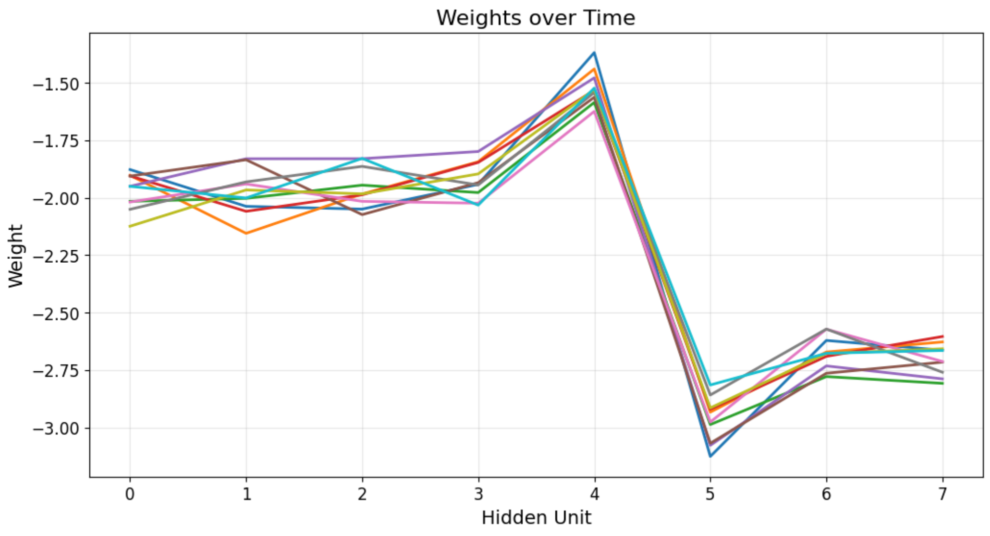


Figure 6.39: Weight over time over 10000 dataset

6.5.3 Performance over 14000 dataset

Here we have first observed the actual vs prediction for a single time step in Figure 6.40.

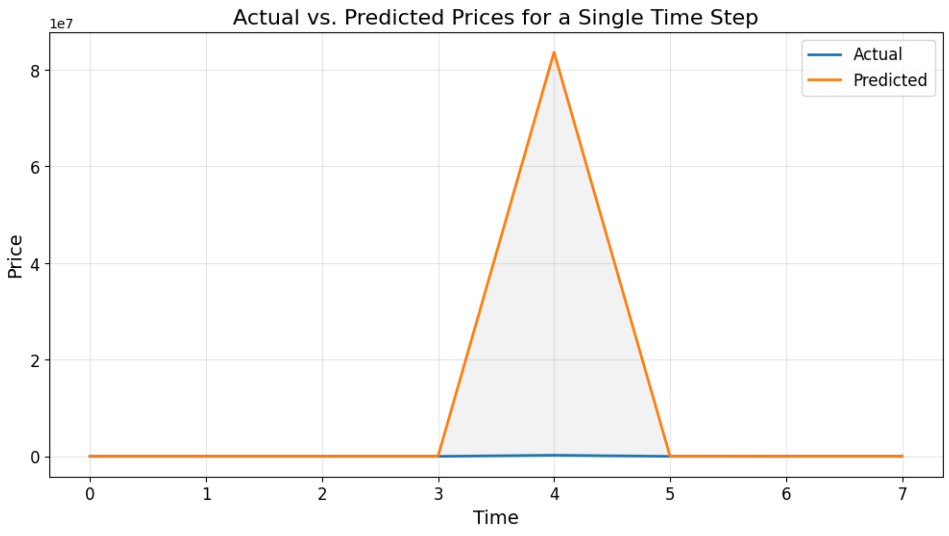


Figure 6.40: Actual vs Prediction over 14000 dataset

Then we observe the hidden bias in Figure 6.41.

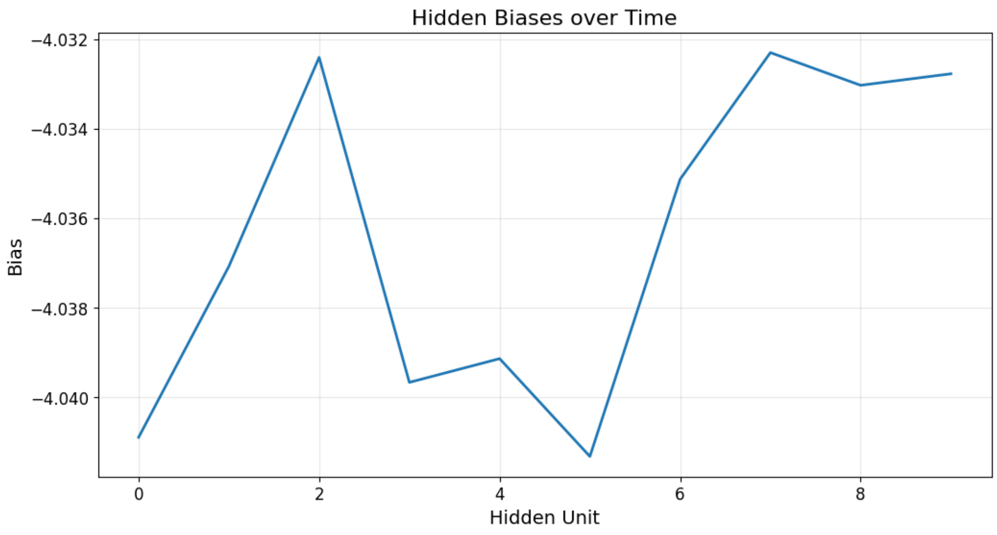


Figure 6.41: Hidden Bias over 14000 dataset

Next, we see the hidden unit over time in Figure 6.42.

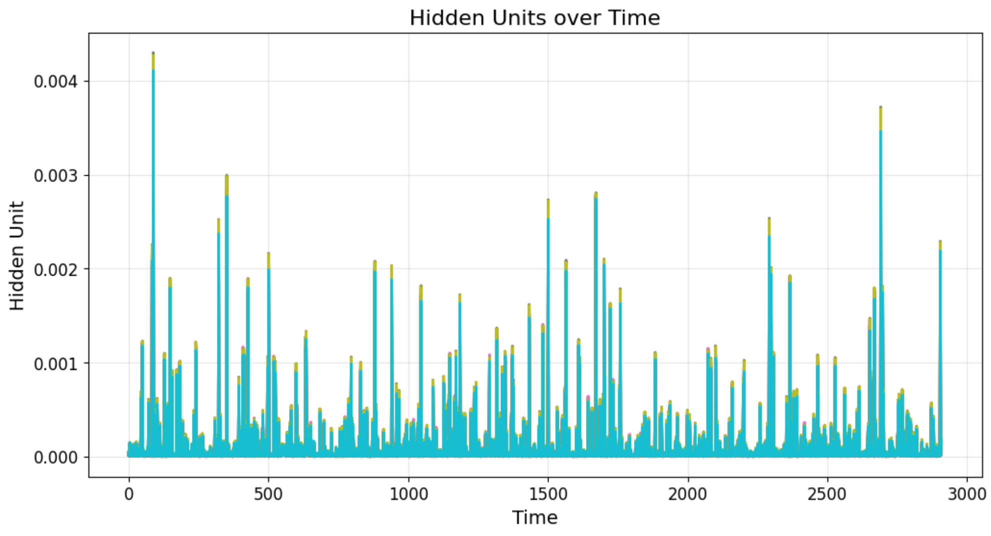


Figure 6.42: Hidden Unit over 14000 dataset

After that, we illustrate the visible bias over time in the Figure

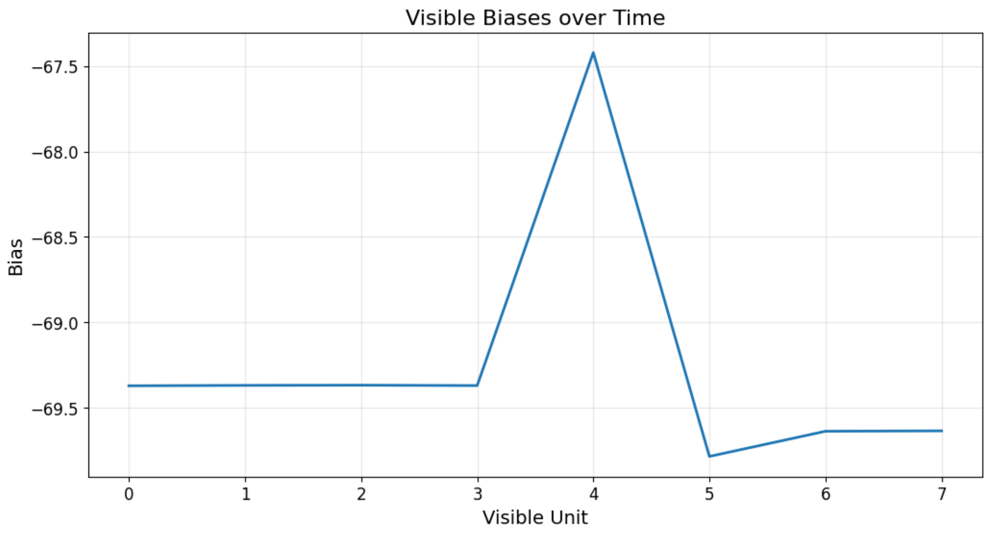


Figure 6.43: Visible Bias over 14000 dataset

Then, we observe the visible unit for the model.

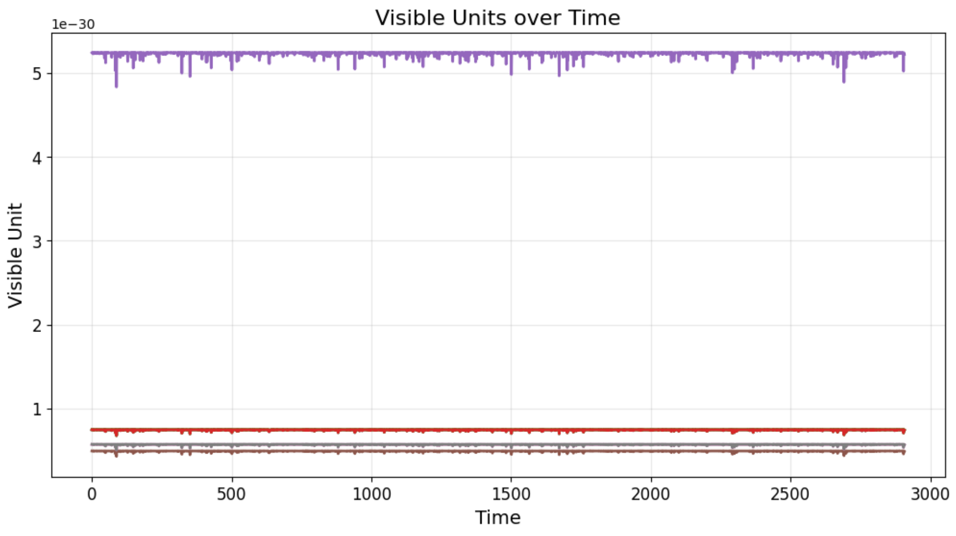


Figure 6.44: Visible Unit over 14000 dataset

Next, We will observe the weight for the single hidden unit and the overall weight over time in the following two figures.
With 10,000 and 14,000 data points, the weights plot shows more variability and patterns. The added information allows for a more detailed depiction of the interactions between visible and hidden units, resulting in more sophisticated weight patterns. The hidden biases plot, like the weights plot, shows more diverse and interesting patterns as the number of data points increases. To capture the intricacies and hidden characteristics contained in the broader dataset, the RBM model adjusts the hidden biases. When compared to the plot with 7,000 data points, the observable biases plot has more oscillations and corrections. With a larger dataset, the model may adjust the visible biases to more accurately reflect the statistical features of the data.
The hidden unit activations plot reveals more subtle dynamics and patterns with

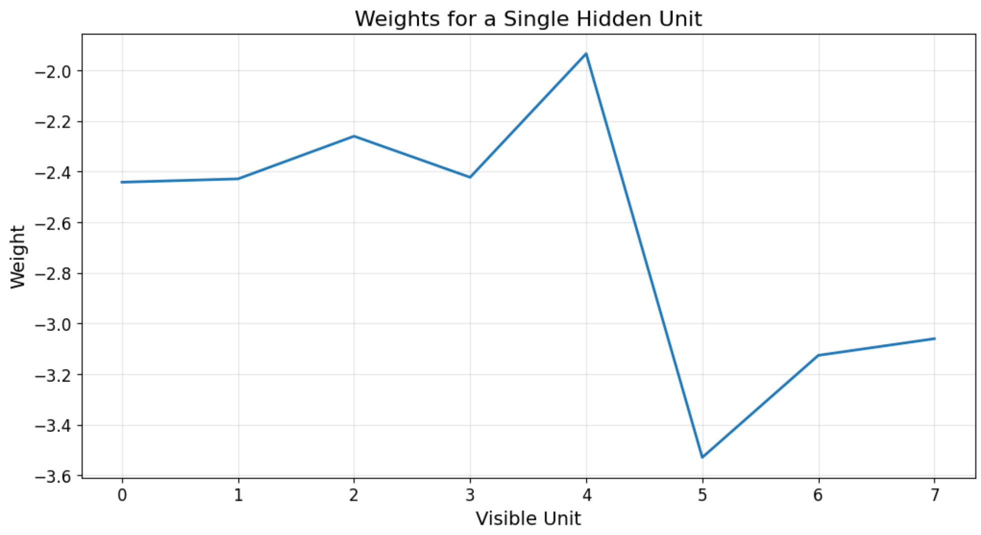


Figure 6.45: Weight for a single Hidden unit over 14000 dataset

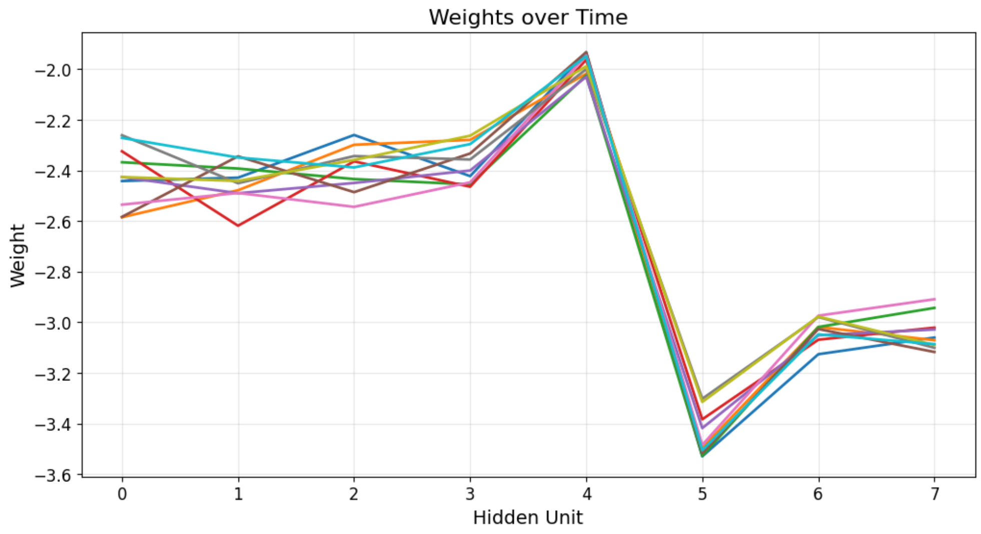


Figure 6.46: Weight over time over 14000 dataset

10,000 and 14,000 data points. The RBM model now has more data from which to learn, which might lead to more complex hidden unit activations. With additional data points, the visible unit activations plot shows more precise and complex patterns. Because of the bigger dataset, the RBM model can capture more complex changes in visible unit activations during the generating process. With additional data points, the model's predictions may become more accurate. The comparison plot of actual and forecasted stock prices shows a smaller difference or error between the two lines, suggesting higher predictive performance. In conclusion, increasing the dataset size to 10,000 or 14,000 data points can result in more detailed and accurate model representations, which leads to superior training loss convergence, weight patterns, biases corrections, and predictive performance. The extra information can give a more complete picture of the fundamental trends and fluctuations in stock price data.

6.6 Deep Belief Network

In this part, we discuss the Deep Belief Network model performance in the aspect of prediction and the loss and MAE with the three datasets. The plotting section of the code provides two graphs: "Training Metrics" and "Actual vs. Predicted Values". These graphs show how the DBN (Deep Belief Network) model performed when trained on stock price data.

6.6.1 Performance with 7000 dataset

"Actual vs. Predicted Values" shows a scatter plot of real stock prices vs projected stock prices. The x-axis shows the actual values, while the y-axis shows the expected values. Each data point in the graph represents a unique occurrence from the test dataset. The graph depicts how well the model's predictions match the actual values. The data points are represented by red plus markers ('r+'), and the transparency (alpha) is adjusted to 0.4 to show overlapping points. The first figure illustrates the actual vs prediction of the model in a scatter plot.

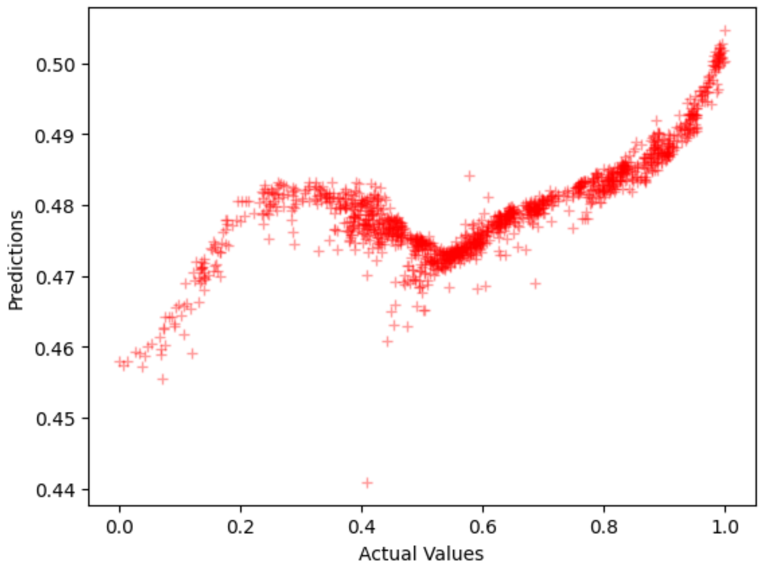


Figure 6.47: Actual VS Prediction over 7000 dataset

In ”Training Metrics,” there are two vertical subplots: one for loss and a second for Mean Absolute Error (MAE). The top subplot’s y-axis reflects the loss value, while the bottom subplot’s y-axis shows the MAE value. The number of epochs on the x-axis corresponds to the number of training iterations. To illustrate the model’s training progress, the loss, and MAE values are displayed against the total number of epochs. Next, the training matrices have been shown which represent the training loss and the training Mean Absolute Error of the model to illustrate the performance.

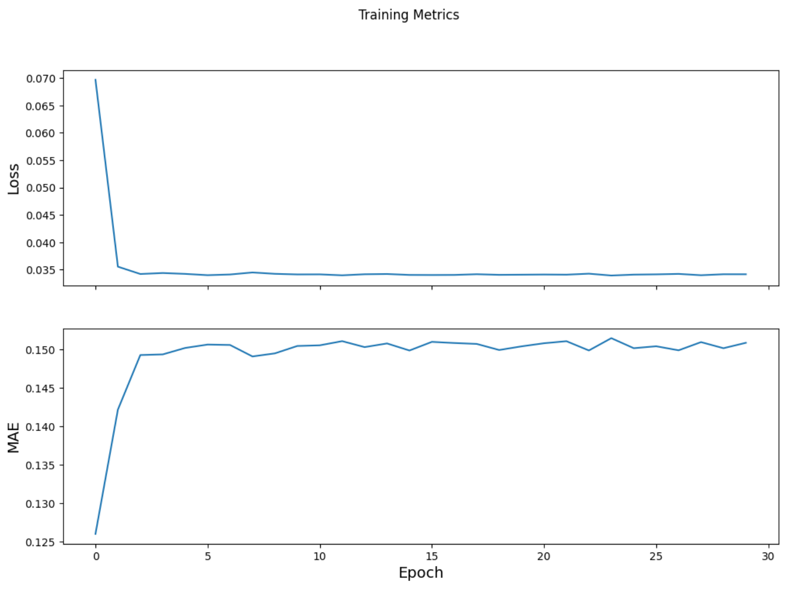


Figure 6.48: Loss and MAE over 7000 dataset

6.6.2 Performance with 10000 datasets

The first figure illustrates the actual vs prediction of the model in a scatter plot.

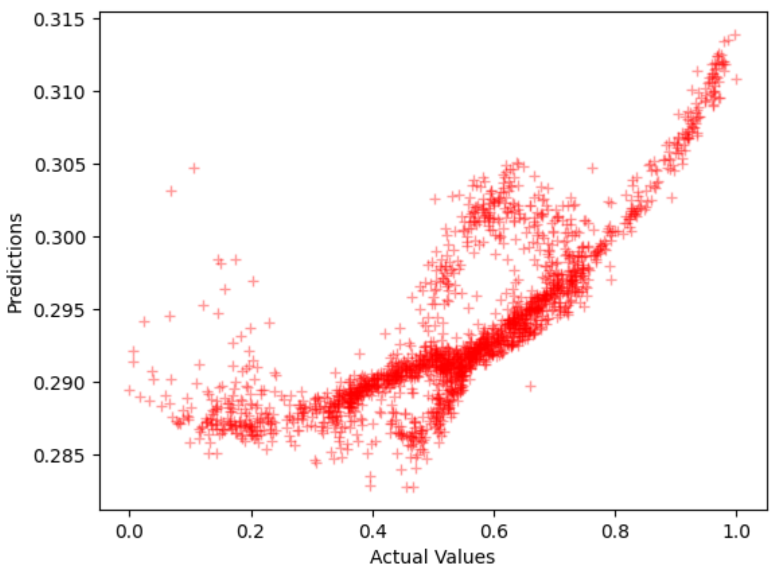


Figure 6.49: Actual VS Prediction over 10000 dataset

The plot in the top subplot depicts the training lost across the epochs. The difference between the actual and anticipated values is represented by the loss, with smaller values indicating a superior model performance. The graphic shows how the loss value varies over the selected number of epochs as the model trains. The declining trend of the loss implies that the model is improving its forecast accuracy with time. Next, the training matrices have been shown which represent the training loss and the training Mean Absolute Error of the model to illustrate the performance.

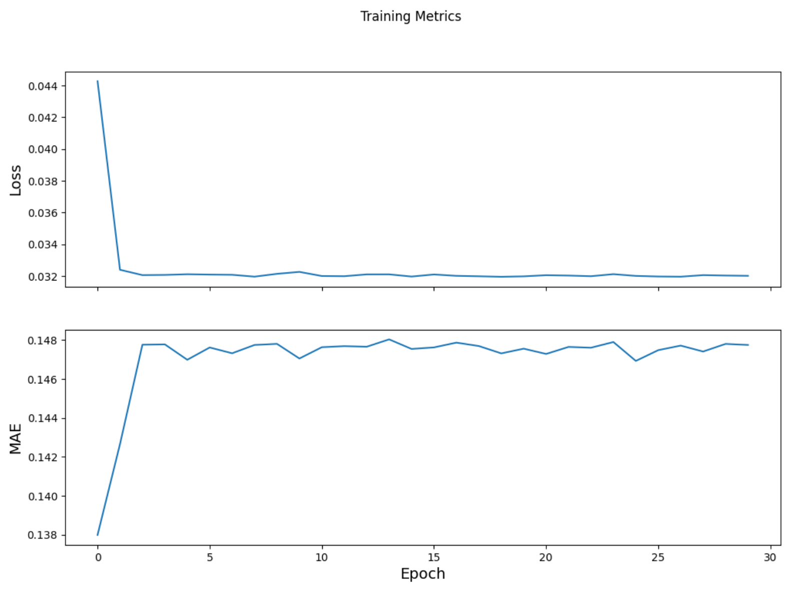


Figure 6.50: Loss and MAE over 10000 dataset

6.6.3 Performance with 14000 datasets

The first figure illustrates the actual vs prediction of the model in a scatter plot.

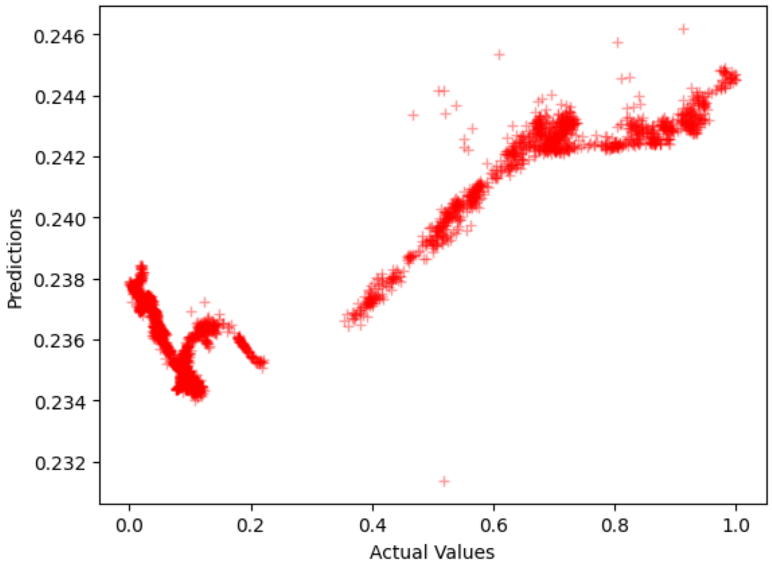


Figure 6.51: Actual VS Prediction over 14000 dataset

The plot in the bottom subplot illustrates the Mean Absolute Error (MAE) throughout the epochs. The average precise variance between the actual and anticipated values is measured by MAE, which provides insight into the model's absolute accuracy in forecasting. The graphic shows how the MAE value evolves throughout the selected number of epochs as the model trains. The MAE's declining trend implies that the predictions made by the model are becoming more accurate with time. Next, the training matrices have been shown which represent the training loss and the training Mean Absolute Error of the model to illustrate the performance.

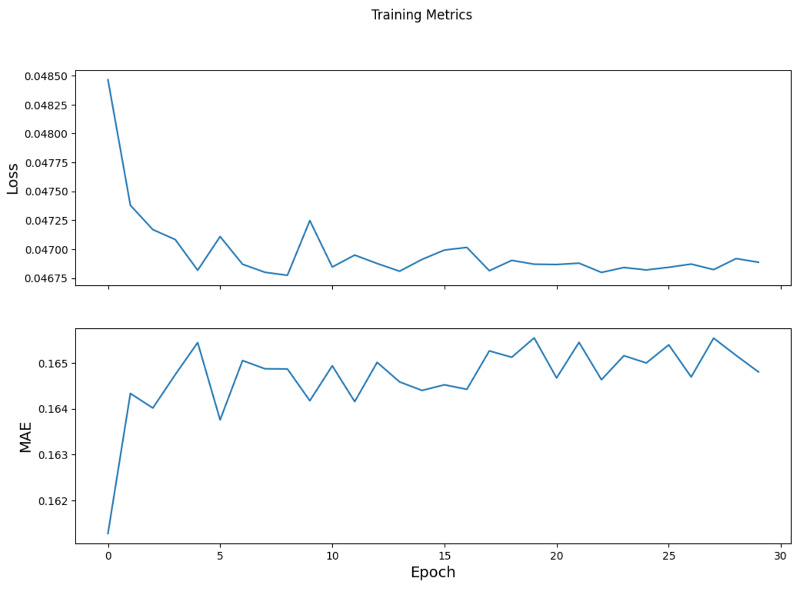


Figure 6.52: Loss and MAE over 14000 dataset

The Training Metrics graph tracks the model's training progress and shows how the loss and MAE values change over each epoch. The lowering loss and MAE numbers suggest that the model's prediction accuracy is increasing with time. The Actual vs. Predicted Values graph provides a visual representation of the performance of the model by comparing predicted to actual values. Overall, these graphs act as visual aids for assessing the DBN model's training success and evaluating its capacity for forecasting by comparing anticipated stock prices to actual values. The falling loss and MAE values show how well the model improvement in learning, whilst the scatter plot shows the model's accuracy in forecasting stock prices.

Chapter 7

Conclusion

In this research, we tested and compared the performance of six machine learning models for stock price analysis across different dataset sizes. Specifically, we evaluated the predictive accuracy of Explainable AI, Q-learning method, LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network models using three different dataset sizes. Our experiments showed that the deep learning models, including LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Network, outperformed the Explainable AI and Q-learning models in terms of predictive accuracy. However, the Explainable AI and Q-learning models have the advantage of being more interpretable and easier to understand.

Our research contributes to the growing body of knowledge in the field of machine learning and finance by providing valuable insights into the strengths and weaknesses of different machine learning models. The findings of our study can be used by investors and financial professionals to make informed decisions when selecting a machine-learning model for stock price analysis. For instance, our results suggest that deep learning models, especially LSTM, Bi-LSTM, Restricted Boltzmann Machine, and Deep Belief Networks, can provide more accurate predictions for stock price analysis than Explainable AI and Q-learning methods. However, in certain applications, such as those requiring interpretability and ease of use, Explainable AI and Q-learning methods may be more suitable.

Future studies can expand on these findings in a variety of ways. First, our study only employed three distinct dataset sizes; it would be fascinating to compare the performance of other machine learning models on bigger and smaller dataset sizes. Furthermore, we only utilized a single stock market dataset in our study; it would be interesting to compare the efficacy of various machine learning models on multiple stock market datasets to assess the generalizability of our results. Second, we used standard performance indicators such as mean squared error, root mean squared error, and R-squared coefficient to assess the performance of several machine learning models. Future research might look at using other measures to evaluate the success of machine learning models for stock price analysis. Third, our research looked at the accuracy and comprehension of several machine learning models for stock market prediction. Future research can look at other elements of machine learning models, such as their resilience to aberrations and capacity to deal with missing data.

Fourth, our research only looked at six machine-learning models for the price of stocks analysis. Future research might look into the performance of various machine learning models for stock price analysis, such as random forest models and Gradient Boosting Machines. Fifth, our research focuses on the use of machine learning models in stock price analysis. Future research might look at applying machine learning models to other financial applications like asset price analysis or foreign exchange rate prediction. Finally, the machine learning models in our study were fed just historical stock price data. Future research might look at using other forms of data as inputs, such as press releases or sentiment analysis on social media, in order to increase the forecasting accuracy of algorithms that use machine learning for the stock price analysis.

In summary, our study provides valuable insights into the strengths and weaknesses of different machine learning models for stock price analysis. Future research can build on our work by exploring other aspects of machine learning models and investigating their application to other financial applications and datasets.

Bibliography

[1] C. Granger, "Forecasting stock market prices," *International Journal of Forecasting*, vol. 8, pp. 3–13, Jun. 1991. DOI: 10.1016/0169-2070(92)90003-R.

[2] C. Cortes and V. Vapnik, "Support vector network," *Machine Learning*, vol. 20, pp. 273–297, Sep. 1995. DOI: 10.1007/BF00994018.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[5] W. Goetzmann, E. Gatev, and K. Rouwenhorst, "Pairs trading: Performance of a relative value arbitrage rule," *SSRN Electronic Journal*, vol. 19, Jan. 1999. DOI: 10.2139/ssrn.141615.

[6] R. Cont, "Empirical properties of asset returns: Stylized facts and statistical issues," *Quantitative Finance*, vol. 1, pp. 223–236, Jan. 2001. DOI: 10.1080/713665670.

[7] L. Feng, T. Dillon, and J. Liu, "Inter-transactional association rules for multi-dimensional contexts for prediction and their application to studying meteorological data," *Data Knowledge Engineering*, vol. 37, pp. 85–115, Apr. 2001. DOI: 10.1016/S0169-023X(01)00003-9.

[8] G. Vidyamurthy, "Pairs trading : Quantitative methods and analysis / g. vidyamurthy.," Jan. 2004.

[9] M. Avellaneda and J.-H. Lee, "Statistical arbitrage in the u.s. equities market," *Quantitative Finance*, vol. 10, pp. 761–782, Jul. 2008. DOI: 10.2139/ssrn.1153505.

[10] H. Shalit, D. Alberg, and R. Yosef, "Estimating stock market volatility using asymmetric garch models," *Applied Financial Economics*, vol. 18, pp. 1201–1208, Aug. 2008. DOI: 10.1080/09603100701604225.

[11] N. Huck, "Pairs selection and outranking: An application to the sp 100 index," *European Journal of Operational Research*, vol. 196, pp. 819–825, Jul. 2009. DOI: 10.1016/j.ejor.2008.03.025.

[12] N. Huck, "Pairs trading and outranking: The multi-step-ahead forecasting case," *European Journal of Operational Research*, vol. 207, pp. 1702–1716, Dec. 2010. DOI: 10.1016/j.ejor.2010.06.043.

[13] J. Agrawal, D. V. Chourasia, and A. Mittra, "State-of-the-art in stock prediction techniques," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 2, pp. 1360–1366, Jan. 2013.

[14] C.-F. Huang, C.-J. Hsu, C.-C. Chen, B. Chang, and C.-A. Li, "An intelligent model for pairs trading using genetic algorithms," *Computational intelligence and neuroscience*, vol. 2015, p. 939 606, Aug. 2015. DOI: 10.1155/2015/939606.

[15] C. Krauss, X. Do, and N. Huck, "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the sp 500," *European Journal of Operational Research*, vol. 259, Oct. 2016. DOI: 10.1016/j.ejor.2016.10.031.

[16] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, Dec. 2017. DOI: 10.1016/j.ejor.2017.11.054.

[17] S. I. Lee and S. J. Yoo, "A new method for portfolio construction using a deep predictive model," in *Proceedings of the 7th International Conference on Emerging Databases*, Springer, 2018, pp. 260–266.

[18] K. Al-Thelaya, E.-S. El-Alfy, and S. Mohammed, "Evaluation of bidirectional lstm for short-and long-term stock market prediction," Apr. 2018, pp. 151–156. DOI: 10.1109/IACS.2018.8355458.

[19] X. Zhan, Y. Li, R. Li, X. gu, O. Habimana, and W. Haozhao, "Stock price prediction using time convolution long short-term memory network: 11th international conference, ksem 2018, changchun, china, august 17–19, 2018, proceedings, part i," in Aug. 2018, pp. 461–468, ISBN: 978-3-319-99364-5. DOI: 10.1007/978-3-319-99365-2_41.

[20] S. Athey, J. Tibshirani, and S. Wager, "Generalized random forests," *Annals of Statistics*, vol. 47, pp. 1179–1203, Apr. 2019. DOI: 10.1214/18-AOS1709.

[21] F. Wang, X. Liu, G. Deng, X. Yu, H. Li, and Q. Han, "Remaining life prediction method for rolling bearing based on the long short-term memory network," *Neural Processing Letters*, vol. 50, Dec. 2019. DOI: 10.1007/s11063-019-10016-w.

[22] J. Zhang and C. Zong, "Deep learning for natural language processing," in Feb. 2019, pp. 111–138, ISBN: 978-3-030-06072-5. DOI: 10.1007/978-3-030-06073-2_5.

[23] S. Liang, Y. Khoo, and H. Yang, "Drop-activation: Implicit parameter reduction and harmonious regularization," *Communications on Applied Mathematics and Computation*, vol. 3, Oct. 2020. DOI: 10.1007/s42967-020-00085-3.

[24] Y. Liu, J. Trajkovic, H.-G. Yeh, and W. Zhang, "Machine learning for predicting stock market movement using news headlines," Nov. 2020, pp. 1–6. DOI: 10.1109/IGESSC50231.2020.9285163.

[25] F. Meng, X. Zeng, Z. Wang, and X. Wang, "Anti-synchronization of fractional-order chaotic circuit with memristor via periodic intermittent control," *Advances in Mathematical Physics*, vol. 2020, pp. 1–8, Jan. 2020. DOI: 10.1155/2020/5158489.

[26] V. Chang, X. Man, Q. Xu, and C.-H. Hsu, "Pairs trading on different portfolios based on machine learning," *Expert Systems*, vol. 38, May 2021. DOI: 10.1111/exsy.12649.

[27] Y. Gao, R. Wang, and E. Zhou, "Stock prediction based on optimized lstm and gru models," *Scientific Programming*, vol. 2021, 2021.

[28] Y. Hu, “Stock forecast based on optimized lssvm model,” *Computer science*, vol. 48, no. S1, pp. 151–157, 2021.

[29] S. Kumar, R. Sharma, T. Tsunoda, K. Thirumananseri, and A. Sharma, “Forecasting the spread of covid-19 using lstm network,” *BMC Bioinformatics*, vol. 22, Jun. 2021. DOI: 10.1186/s12859-021-04224-2.

[30] R. Chiong, Z. Fan, Z. Hu, and S. Dhakal, “A novel ensemble learning approach for stock market prediction based on sentiment analysis and the sliding window method,” *IEEE Transactions on Computational Social Systems*, vol. PP, pp. 1–11, Jan. 2022. DOI: 10.1109/TCSS.2022.3182375.

[31] F. Gao, J. Zhang, C. Zhang, X. Shuang, and C. Ma, “Long short-term memory networks with multiple variables for stock market prediction,” *Neural Processing Letters*, pp. 1–19, Oct. 2022. DOI: 10.1007/s11063-022-11037-8.

[32] T. Lees, S. Reece, F. Kratzert, *et al.*, “Hydrological concept formation inside long short-term memory (lstm) networks,” *Hydrology and Earth System Sciences*, vol. 26, pp. 3079–3101, Jun. 2022. DOI: 10.5194/hess-26-3079-2022.

[33] W. Wang, “Further results on mean-square exponential input-to-state stability of stochastic delayed cohen-grossberg neural networks,” *Neural Processing Letters*, Aug. 2022. DOI: 10.1007/s11063-022-10974-8.

[34] X. Gu, A. Metcalfe, N. Cook, C. Aldrich, and L. George, “Exploratory analysis of multivariate drill core time series measurements,” *ANZIAM Journal*, vol. 63, pp. C208–C230, Jan. 2023. DOI: 10.21914/anziamj.v63.17192.

[35] X. Gu, A. Metcalfe, N. Cook, C. Aldrich, and L. George, “Exploratory analysis of multivariate drill core time series measurements,” *ANZIAM Journal*, vol. 63, pp. C208–C230, Jan. 2023. DOI: 10.21914/anziamj.v63.17192.