University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Predictive Modeling and Analysis of Stock Price Movements Using NYSE Historical Data

Student Name and SRN:

Khubaib Ahmad 21031902

Supervisor: **Niall Miller**

Date Submitted: **29- 8 -2024**

Word Count: **6262**

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: **Khubaib Ahmad**

Student Name signature: **Khubaib**

Student SRN number: **21031902**

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ACKNOWLEDGEMENT

I acknowledge that this project report entitled "Predictive Modeling and Analysis of Stock Price Movements Using NYSE Historical Data" is a record of original work carried out by me, Khubaib Ahmad, under the guidance of Niall Miller. This work has not been submitted in part or full for any other degree or diploma at any other university or institution. All sources of information and data have been duly acknowledged.

ABSTRACT

For some years now, stock price forecasting has been an intricate sector that has fascinated many investors and experts. The use of machine learning models – Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), and Extreme Gradient Boosting (XGBoost) – to anticipate fluctuations in stock prices based on historical data from the New York Stock Exchange (NYSE) is investigated in this study. Comprehensive data preprocessing, feature engineering, and model training were employed to identify the complex dynamics associated with variations in stock prices.

LSTM became the best-performing model, making precise predictions for stock prices as demonstrated by R^2 of 0.9883. The recurrent neural network model was useful; however, it could not capture long-term dependencies in the data, which led to an R^2 of 0.6415. Powers of handling non-linear relationships were XGBoost but overfitting was seen where there was perfect performance on the training set and some reasonable level of error in the test set.

Currently, Artificial Intelligence (AI) is relied on for stock market predictions. Not only does this study make an important addition to the increasing number of works on financial forecasting using machine learning, but LSTM and other advanced models could also help enrich strategic and profitable trading practices. On the other hand, further improvements can be made to these models by adding more features such as technical indicators, macroeconomic determinants, and advanced architectures like stacked LSTMs or bidirectional LSTMs.

CONTENTS

CHAPTER 1 INTRODUCTION

1.1 BACKGROUND

Probably the most challenging domain for prediction is the dynamic nature of the stock market. For quite a long time, investors and analysts have been developing models to predict stock price movement to efficiently optimize returns against the risk undertaken. Of course, traditional tools of fundamental and technical analysis provided a background that helped in understanding market trends, but they always turned out to be poor in the face of the complexity and volatility of markets. In this sense, machine learning and artificial intelligence offer new opportunities for predictive modeling by availing tools that can process large amounts of data to uncover hidden trends and patterns that traditional methods might miss. The New York Stock Exchange is among the largest and most influential stock exchanges in the world, offering a rich dataset for analysis that contains daily stock prices, adjusted prices for splits, company descriptions, and financial metrics derived from SEC 10K filings. The data thereby opens up an opportunity to look into advanced predictive models that would be able to forecast stock prices and dictate relevant trading strategies. In this study, we focus on Long Short-Term Memory networks, Recurrent Neural Networks, and Extreme Gradient Boosting in stock price movement prediction. The two most common forms of neural networks in use for time series forecasting are LSTM and RNN. Being a variant of RNN, due to its design, it is particularly good at modeling long-term dependencies in sequential data; hence, it would be appropriate for time series forecasting of stock prices, wherein the past performance does affect the future trends in this case. RNNs, on the other hand, can learn from sequences of data but can sometimes be pretty bad at long-term dependencies; this itself is addressed by the special design in the architecture of LSTM. On account of its efficiency and performance, directly based on gradient boosting techniques, XGBoost has been among the strongest machine learning algorithms. Unlike neural networks, there is no native sequentially, so it fits pretty well-structured data. It can be used not only for solving classification but also for regression tasks. This makes XGBoost pretty effective in stock price prediction, specifically with historical price data features combined with financial metrics; it is an alternative way to model the complex factors that affect stock price movement.

The purpose of this project will be to use the NYSE dataset to develop LSTM, RNN, and XGBoost models for comparison in stock price forecasting. This will be done through exploratory data analysis, feature engineering, and back testing trading strategies to prove their accuracy and profitability in real trading scenarios. The results will contribute to the literature of fast-growing research in the application of machine learning techniques in finance, whereby opinions concerning the strengths and limitations of various predictive approaches usually contrast. Such findings can contribute to the further knowledge of the dynamics of the stock markets and the development of better tools for use in making more informed investment decisions. Through this work, I will attempt to bring about the development of models for predicting stock prices, more accurate and reliable by integrating advanced machine learning techniques with traditional financial analysis to offer significant support to strategic and profitable trading practices.

1.2 PROBLEM STATEMENT

Our study focuses on predicting stock price movements using past price information and financial metrics derived from the New York Stock Exchange. The study also considers which model of machine learning—Long Short-Term Memory, Recurrent Neural Networks, or Extreme Gradient

Boosting—is better at capturing the underlying trend of price movements. This project thus identifies which model can give the most accurate prediction, hence presenting useful insight to the investor and analyst.

1.3 OBJECTIVES

The objective of this work is to develop and evaluate predictive models of stock price movement from historical data on the NYSE. The specific objectives of the project are as follows:

- Do exploratory data analysis to look into the nature of historical stock price data and related financial metrics, understand the constitution of the dataset, identify the key trends, and detect anomalies.
- Data preprocessing means cleaning and preprocessing the dataset for missing values and outliers and merging different data sources. This is to ensure that high-quality data are available for use in training the model.
- Features engineering to enhance the predictive power of the models, such as moving averages, volatility metrics, and financial ratios.
- Implement and evaluate the performance of LSTM, RNN, and XGBoost models in predicting stock price movements. All these models will be trained with historical data, and their accuracy will be measured using suitable metrics.

1.4 SIGNIFICANCE OF STUDY

This is important research that contributes significantly to the analysis of financial markets with a view to integrating traditional financial analysis methods with machine learning methods, thus enabling the prediction of stock prices. A performance comparison between the LSTM, RNN, and XGBoost models provides critical information on which model is best in capturing the complexities surrounding the movement of stock prices. Using historical price data and financial metrics taken from the New York Stock Exchange, this research tries to channel the potential of machine learning in order to detect patterns and trends in the data that might have been overlooked by other approaches, yielding thereby more accurate predictions. Such findings can thus directly help in the elaboration of improved trading strategies by equipping investors and analysts with new and powerful tools to make more data-driven and informed decisions for higher returns while managing risks more effectively.

This research will also serve as a bridge between academic works and practical applications in the financial industry. It is expected that models investigated in this study may stimulate new financial product developments, for example, algorithmic trading systems and professional portfolio management tools. This also creates unparalleled educational value, since it is going to be a go-to reference for scholars and professionals alike who seek deeper knowledge in both finance and machine learning. Knowledge contribution in the field of financial data science will further help establish grounds for building future experts in this field by showing how such models can be utilized on real-world financial data.

1.5 THESIS LAYOUT

The thesis, therefore, unfolds in six primary chapters which are interlinked and complement each other for an all-round study in the application of advanced machine learning techniques in the prediction of stock price.

1.5.1 CHAPTER 1: THE INTRODUCTION

The first chapter may give a view of the study. The related background to this study describes the dynamic nature of the stock market. Moreover, there would be a need to have exact predictive models based on the understanding of stock prices to optimize returns. It talks about how machine learning can enhance traditional financial analysis with its ability to uncover hidden trends and patterns in big data. Whence, the problem statement lies in between the lines of which of these machine learning models, either LSTM, RNN, or XGBoost, would perform best in stock price predictions. In correspondence, the objectives and significance of this study are to put meaning in contributions of financial modeling, practical trading strategies, and educational value.

1.5.2 CHAPTER 2 – LITERATURE REVIEW

The second chapter provides a reader with existing research and the theories put forward for stock price prediction. The study grounds its work on the traditional methods that are the fundamental analysis, technical analysis, followed by the application of machine learning models in finance. It then subsequently moves on to previously published works with LSTM, RNN, and XGBoost, giving discussions on the pros and cons within the context of financial time series forecasting. Writing this literature review situates the current study within the predominant academic milieu. Chapter 3: Methodology Chapter 3 now delves into the methodology of the current study, opening with a thorough description of the dataset used. The dataset utilizes historical stock prices and financial metrics obtained from the New York Stock Exchange. Then, it discusses a preprocessing step of cleaning and preparing data, followed by feature selection and engineering to further bring improvement in the predictive power of the models. The chapter introduces the three machine learning models applied in this research: LSTM, RNN, and XGBoost, their architecture, parameters, and reasons for choosing these models.

1.5.3 CHAPTER 4: RESULTS AND DISCUSSION

Finally, Chapter 4 presents the empirical results of each model, that is, LSTM, RNN, and XGBoost, with respect to their performance on accuracy metrics in stock price movement predictions. In this chapter, each of the models is accorded a section, and the last section is devoted to an overall analysis by comparing the performance of all the models. Moreover, relative strengths and weaknesses for each approach are discussed together with the implications to financial forecasting.

1.5.4 CHAPTER 5: CONCLUSION

Chapter 5 concludes the findings of the study on effectiveness of different machine learning models in prediction of stock prices. This makes a discussion of the key implications that the results may have for both academic research and practical/financial applications. Possible areas for future work are also indicated in terms of model fine-tuning or adding additional data sources.

1.5.5 CHAPTER 6: REFERENCES

Chapter 6 consists of all the academic references, articles, and data sets utilized in tie-up to this thesis. This provides evidence that the work being presented is well-based concerning earlier research.

CHAPTER 2 LITERATURE REVIEW

The paper (Mohan, S., 2019) proposes a fusion of historical stock prices with deep learning models, namely based on the sentiment analysis of financial news for improving the accuracy in the prediction of stock prices. The dataset includes daily stock prices of S&P 500 companies over five years and more than 265,000 financial news articles. Applied models include ARIMA, Facebook Prophet, and RNN LSTM. RNN LSTM models, including both stock prices and news sentiment, gave the best performances. The lowest MAPE, hence the highest prediction accuracy, resulted from RNN-pp with stock prices and text polarity. The paper (Fenghua, W.E.N., 2014) uses SSA to decompose stock prices into trends, market fluctuation, and noise terms and feeds these into the support vector machine for price prediction. In this paper, the combination of SSA-SVM shows a better prediction accuracy than using SVM alone or combining SVM with other methods like EEMD-SVM. It uses the closing prices of the Shanghai Stock Exchange Composite Index from the year 2009 up to 2013. The results indicate that the SSA-SVM provides a good fit for the actual price trends. Combination predictions by SSA-SVM are more effective at capturing the features of stock price series for better predictive results in this respect. The paper (Vijh, M., 2020) predicts the stock closing price using Artificial Neural Network and Random Forest techniques. Ten years of historical data were taken for the five companies: Nike, Goldman Sachs, Johnson & Johnson, Pfizer, and JP Morgan. In this research, six new variables were developed from stock data for training the model. Models were evaluated on RMSE and MAPE. ANN showed better prediction accuracy with lower RMSE and MAPE values against RF. The paper (Pawar, K., 2019) discusses the use of RNN driven by LSTM cells in modeling stock prices for portfolio management. This work compares the LSTM-RNN model against traditional machine learning algorithms like Regression, SVM, Random Forest, and Forward Neural Networks. The model takes historical stock data from Yahoo Finance, normalizes it, and then splits it into a training set and a test set. Various LSTM architectures were run to come up with the best configuration. In this model, an LSTM RNN learned the trends of data and hence performed well in predicting stock prices compared to other traditional methods, so this approach benefits not only individual traders but also corporate investors. The paper (Zheng, 2017) addresses the problem of short-term stock price prediction from time series data for stock prices. Several machine learning models were implemented, such as Logistic Regression, Bayesian Network, Neural Network, and SVM. All the models achieved an accuracy of about 50-60%2. Technical indicators improved the accuracy in the prediction by up to 70%. The authors suggested using higher computational power and exploring more neural network models for better accuracy. This research work by (Oukhouya, H., 2024) will predict the daily prices of six international stock indices under study, using LSTM, XGBoost, and a hybrid model of the two. The models are trained and tested on five years of back data from six stock markets. The resulting hybrid model refers to the combination of LSTM residuals with XGBoost. The hyper-parameters of the said hybrid model have been optimized using a Grid Search. The model is actually doing better when combined with separate models and has predicted the stock price very successfully. The end results would be supportive to analysts and investors in decision-making, with exact stock price predictions. The paper (Shi, Z., 2022) proposes a hybrid model combining ARIMA, attention-based CNN-LSTM, and XGBoost for stock price prediction. First, a hybrid model in which ARIMA is used for the data sequence of the original stock price is proposed to preprocess the stock data. It would subsequently use attention-based CNN in the extraction of deep features and LSTM in the training of long-term dependencies. Thereafter, applying fine-tuning using XGBoost will improve the prediction accuracy. (Nabipour, 2020) focuses on the prediction of future values of groups related to the stock market with the help of different machine-learning algorithms. In this research, data have been collected from four groups: diversified financials, petroleum, non-metallic minerals, and basic metals, all from the Tehran Stock Exchange over 10 years. Algorithms used in this research work are decision tree, bagging, random forest, Adaboost, gradient boosting, XGBoost, ANN, RNN, and LSTM. LSTM had the highest accuracy among all models; however, the accuracy turned up very good for tree-based models, like Adaboost, Gradient Boosting, and XGBoost. The paper (Gao, P., 2020) is aimed at predicting stock index prices by four machine learning models: Multilayer Perceptron, Long Short Term Memory, Convolutional Neural Network, and the neural network endowed with attention. The dataset includes SP500, CSI300, and Nikkei225 indices data, which represent financial markets. The variables applied were seven inputs that contained daily trading data, technical indicators, and macroeconomic variables. From these models, the one with attention exhibited the best performance. All of the models performed better in developed markets compared to developing ones. The paper by (Rezaei, 2021) proposes deep learning-based novel hybrid algorithms: CEEMD-CNN-LSTM and EMD-CNN-LSTM for stock price prediction, along with frequency decomposition methods. Stock price prediction is complex due to the nonlinearity and high volatility of financial time series. LSTM and CNN models are integrated with Empirical Mode Decomposition and Complete Ensemble Empirical Mode Decomposition to improve the prediction accuracy. Results show that the proposed hybrid models are ahead of other ways, and CEEMD-CNN-LSTM performs better than EMD-CNN-LSTM.

CHAPTER 3 METHODOLOGY

3.1 DATASET DESCRIPTION

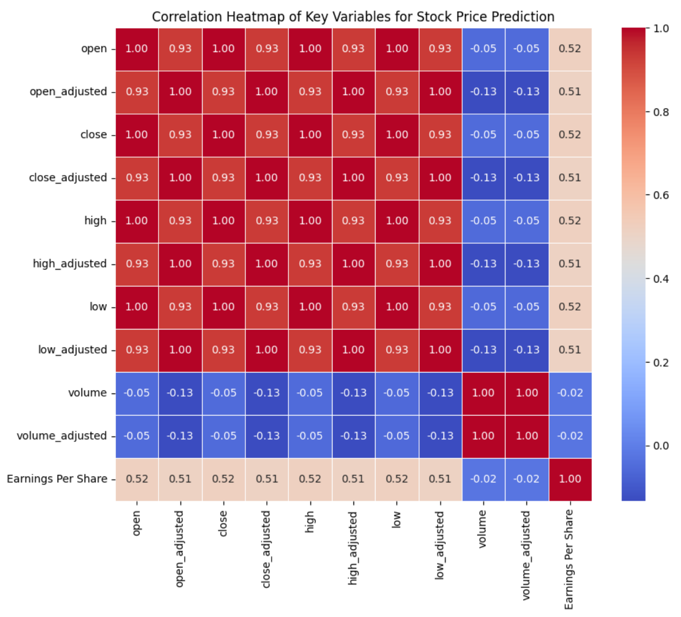
This project works with a dataset containing historical data from S&P 500 companies listed on the NYSE from 2010 to 2016, containing raw and adjusted daily stock prices, fundamental financial metrics, and a detailed description of each company within the file. The main files in this dataset are prices.csv, containing raw daily stock prices, and prices-split-adjusted.csv, which considers stock splits to appropriately analyze long-term prices. These price datasets form the basis of technical analysis and predictive modeling in the study of historical price trends and the formulation of trading strategies. Besides price data, the dataset includes fundamentals.csv, which has a bunch of financial metrics extracted from SEC 10K filings for the period from 2012 to 2016. The file contains metrics like earnings per Share, Return on Equity, and debt-to-equity ratios—key indicators in their full base for fundamental analysis. Therefore, this information could be used for assessing the financial status and performance of the companies in question and identifying undervalued stocks with a prediction of financial distress. Another important data file is securities.csv, which includes general descriptions of a firm, with a supplement of information on the sector classification. This file would be critical in a case where you would want to run an analysis on a sectoral front or possibly cluster firms around industry information, which in all probability would drive portfolio construction and risk management strategies. This dataset is very useful in all types of financial analyses, from short-term price predictions to long-term investment evaluations, because it includes both price and fundamental data.

3.2 EDA

3.2.1 CORRELATION MATRIX KEY SELECTED FEATURES

The correlation heatmap Figure 1 illustrates the relationships among key variables relevant to stock price prediction. The variables include various aspects of stock prices such as open, close, high, and low prices, and their adjusted counterparts, alongside trading volume metrics and earnings per share (EPS). Strong positive correlations are evident between open, close, high, and low prices, which is expected given their close interdependence in daily stock movements. The adjusted versions of these prices show similarly strong correlations among themselves. There is a moderate positive correlation between EPS and the stock price metrics, indicating that higher earnings per share tend to correlate with higher stock prices. Trading volume, both regular and adjusted, shows negligible correlation with the other variables, suggesting its relative independence from price movements and earnings per share in this context. This heatmap provides insights into which variables might be more influential or related in predicting stock price behavior, aiding in further analysis or modeling efforts.

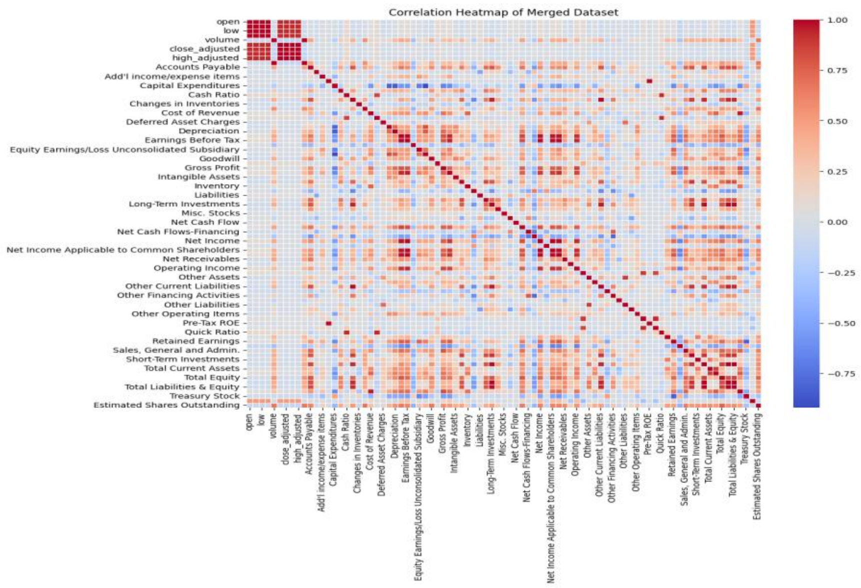
Figure 1 Strong Correlation



3.2.2 CORRELATION MATRIX OF ALL THE FEATURES IN THE DATASET

Based on this Figure 2 visualization we select those features that have impact on the close price either strong or small.

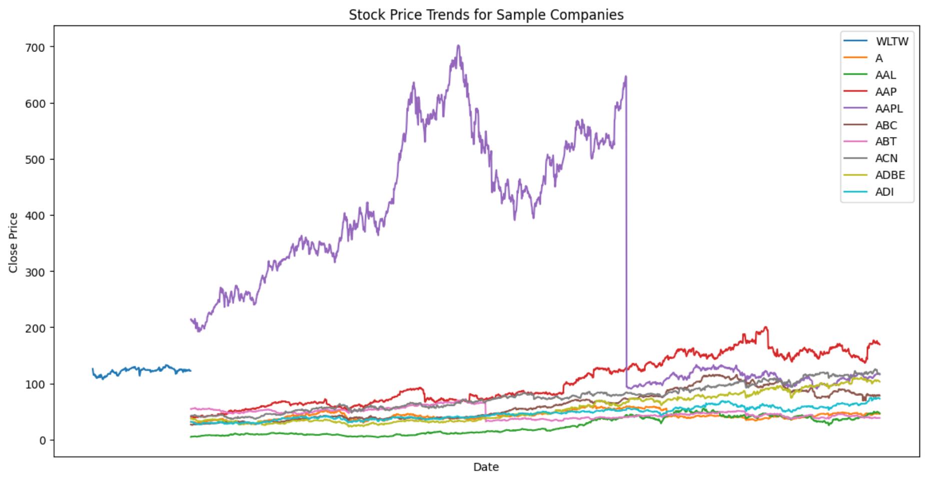
Figure 2 Correlation of Features



3.2.3 STOCK PRICE TRENDS

We selected a sample of ten companies from the dataset and plotted their daily closing prices over time shown by Figure 3. The line plot demonstrates the variations and trends in the closing prices for these companies, allowing for a comparative analysis. The x-axis represents the date, while the y-axis represents the closing price of the stocks. Each line corresponds to a different company's stock, with a legend indicating the respective symbols. This visualization provides insight into the selected companies' stock performance and market behavior, highlighting patterns and potential correlations in their price movements.

Figure 3 Stock Price Action



3.3 PRE-PROCESSING

Preprocessing, performing cleaning, combination, and transformation of the data, are very much necessary parts of the process to get the dataset ready for machine learning. We first load four key datasets, which incorporate stock prices, financial metrics, and firmescription information. The datasets when carefully merged finally give a single dataset, to which predictive modeling can be done. The union of datasets is generally based on common columns. Missing values in the merged dataset are identified and handled in such a way that the integrity of the data is maintained. Records with missing values on key features are dropped to ensure a robust dataset for modeling and free of inconsistencies. This is to avoid introducing bias in the model and to make sure the predictions are drawn from complete and accurate data. The next step, after handling missing values, is to prepare the data for modeling by applying feature scaling. Here, MinMaxScaler is used for normalizing numerical features in the dataset. Scaling is important when working with any machine learning algorithm sensitive to the magnitude of features, for example, neural networks. MinMaxScaler rescales the features so that all of them end up being in a similar kind of scale and hence end up having equal weightage with respect to the model; no one feature overshadows the others. Finally, the categorical features—for instance, the symbol column with the various stock tickers—have their processing carried out so they are made ready for modeling. This, for the most part, is translating them into numerical values. One common way of doing this is through LabelEncoder, which can be used to produce a unique integer for each category. This is very useful for algorithms that only accept numerical input: it effectively encodes the categorical information into the model. The end result will be a clean, scaled, and structured dataset that will be ready for the next stage of feature selection and training a model on it. Such attention to pre-processing guarantees that data used during the process of building the models is the best possible—this is why this way the predictions would be more accurate and reliable.

3.4 FEATURE SELECTION AND ENGINEERING

Feature selection and engineering are key techniques, either through setting or creating the most relevant variables necessary for any analysis to improve the predictive power of Machine Learning models. In the feature selection process, we will make thoughtful selections of preliminary important features based on their appropriateness to the task of stock price prediction and the representation in the dataset. The basic stock market indicators are chosen here: Open, Close, High, Low, Volume, and also a rich set of metrics in terms of financial health from fundamentals data. Such metrics as Earnings Before Interest and Tax, Net Income, and Operating Margin, together with several ratios capturing more information about a company's financial health, would be very strong predictors for stock prices. The first of these selections narrows the focus on only some of those variables to have the most impact, reducing dimensionality within the dataset, and thus the efficiency of the model. Then, feature engineering follows the selection of key features by creating new variables that capture more information from the data. Technical indicators are robust tools commonly used in predicting stock prices. They are derived from price and volume data to give a clue of a pattern, trend, and points of reversal. The Relative Strength Index, Moving Averages, Moving Average Convergence Divergence, and Average Directional Index are indicators calculated using the 'ta' library, which specializes in technical analysis. These indicators are computed for each stock symbol across varying time windows, thus giving a very rich set of features that reflect momentum, strength of trend, and volatility of the market. Further, the accumulation/distribution line is computed by hand to track money inflows and outflows from the stock, hence adding another dimension to the analysis.

This feature set is further enhanced by the computation of Bollinger Bands to capture volatility, a Stochastic Oscillator to measure momentum, and a Rate of Change to measure the speed of price movements. These indicators are very useful in spotting any potential buy or sell signals based on past price movements. We have also implemented Exponential Moving Averages, which weigh more on recent price trends. On-balance volume conveys the changes in prices related to volume. Then there is the Money Flow Index that measures buying pressure against selling pressure. Each of these indicators has a uniqueness according to the nature of the stock market and is hence very important for predictive modeling. Finally, after engineering these new features, we drop rows with any remaining missing values generated during the calculation of technical indicators, to make sure that the dataset is clean and ready for model training. Then, using MinMaxScaler, scale the selected and engineered features to normalize data. This will be particularly useful for models sensitive to feature scaling, like neural networks. The categorical symbol column is encoded to be prepared for model training using Label Encoder, turning it into its appropriate numerical format to feed into any machine learning algorithm. The supplied rich set of features, along with the domain-specific engineered features, provides a fairly good basis to build robust predictive models in an attempt to capture complex stock market behaviors.

3.5 MODEL

We will now discuss in detail the development and tuning of three different models, namely LSTM, Simple RNN, and XGBoost. Each of these models was constructed with a specific architecture best suited for yielding top performance on stock price prediction tasks. For fine tuning each model, an exhaustive hyper parameter search was performed to ensure that the models perform at their best.

3.5.1 LSTM MODEL

ARCHITECTURE AND WORKING:

The LSTM network is designed to learn the dependencies for a long term, thus it is very suitable for time series forecasting-for example, the prediction of stock prices. The architecture of the model includes, but is not limited to components such as the following:

- LSTM Layer: This is the heart of processing data sequentially. For this layer, the number of units-things in neural network terminology-was treated as a hyperparameter, which was tuned across different values to find the optimal setting. This LSTM layer should use a 'tanh' activation function that can be used to keep the gradient under control from exploding during backpropagation and keeps the output value within a decent range that maintains the stability of the network.
- Dropout Layer: To prevent overfitting, a dropout layer was added after the LSTM layer. Dropout randomly sets a fraction of the input units to zero at each update during training, forcing it to generalize better.
- Dense Output Layer: A dense layer containing one neuron was added on top to provide the stock price prediction. This layer will give out directly the predicted closing price.

TRAINING PROCESS:

The model was trained using the Adam optimizer due to its efficiency in most deep learning applications based on its adaptive learning rate. Two hyperparameters were optimized through a grid search: one is the number of units in the LSTM layer, and the second is the learning rate. This range was searched over to find out which configuration of the two aforementioned hyperparameters yields the minimum validation loss. EarlyStopping was also used during the training, which stopped training if no improvement in terms of validation loss was observed. It had a patience parameter equal to two epochs. The model was then trained for each different time step configuration of 1, 2, 5, and 10 steps, where the best overall model for each configuration was selected by its performance on the validation set.

PREDICTION:

The LSTM model, once trained, predicts stock prices by processing the sequence of the flow of previous price data. Due to its capability for sustaining and updating memory concerning what has occurred in the past, it is able to perform intelligent predictions of the future course of prices, considering both the recent trend and the long-run pattern of the data.

3.5.2 RNN MODEL

ARCHITECTURE AND WORKING:

The RNN model is also less complex compared to LSTM and was also explored in light of time series forecasting. The architecture of RNN models constitutes:

- RNN Layers: The two RNN layers in our model were each capable of handling sequences of data. The number of neurons to be used in these kinds of layers was regarded as a hyperparameter, for which we tried 100, 200, and 500 during the tuning process. In addition, these layers included the 'tanh' activation function in order to help with controlling the gradients by reducing the likelihood of exploding or vanishing gradients during training.
- Dropout Layer: Similar to what was done with the LSTM model, a dropout layer was added to reduce overfitting issues in this RNN by randomly dropping the units while training. Therefore, the next additional layer to the RNN is a dropout layer.
- Dense Output Layer: The final layer in this RNN is the dense layer responsible for generating the forecast of the stock price.

TRAINING PROCESS:

The RNN model was tuned by doing a grid search over a number of hyper-parameters: number of neurons in the RNN layers, learning rates, and the optimizer. Hence, Adamax was one form of Adam which is robust to high learning rates. Again, EarlyStopping was used to avoid over-fitting by saving the best model depending on the performance of the validation set.

PREDICTION:

The RNN model predicts stock prices by processing sequences of past price data after training. While RNNs are generally worse at dealing with long-term dependencies than LSTMs, they usually do well on shorter sequences and hence could often serve well for some specific tasks of time series forecasting.

3.5.3 XGBOOST MODEL

ARCHITECTURE AND WORKING:

Contrasting with the neural network models, a non-sequential learning algorithm was XGBoost. XGBoost is the most powerful gradient-boosting framework that really shines when working with structured data. The model developed in this work set up an XGBRegressor, whose objective was established as a string 'reg' because this minimizes the squared error during training.

TRAINING THE MODEL:

The grid search for the XGBoost model was done over a number of hyperparameters, including the number of estimators, a pseudo-parameter that controls the number of trees in the ensemble, and learning rate. Grid search was performed with 5-fold cross-validation in order to make sure that the model generalizes well and does not overfit to the training data. The best combination of hyperparameters was chosen with the highest value of the negative mean squared error. It will scale all the features in a similar range to improve model performance using MinMaxScaler feature scaling.

PREDICTION

XGBoost, unlike LSTM and RNN, is not designed to be applied inherently to sequences. Its main strength lies in the modeling of complex relationships in structured data. So it will make a prediction using historical price data and financial metrics by creating an ensemble of decision trees to predict the future stock price based on the pattern it learned from the data.

CHAPTER 4 RESULT AND DISCUSSION

4.1 LSTM MODEL

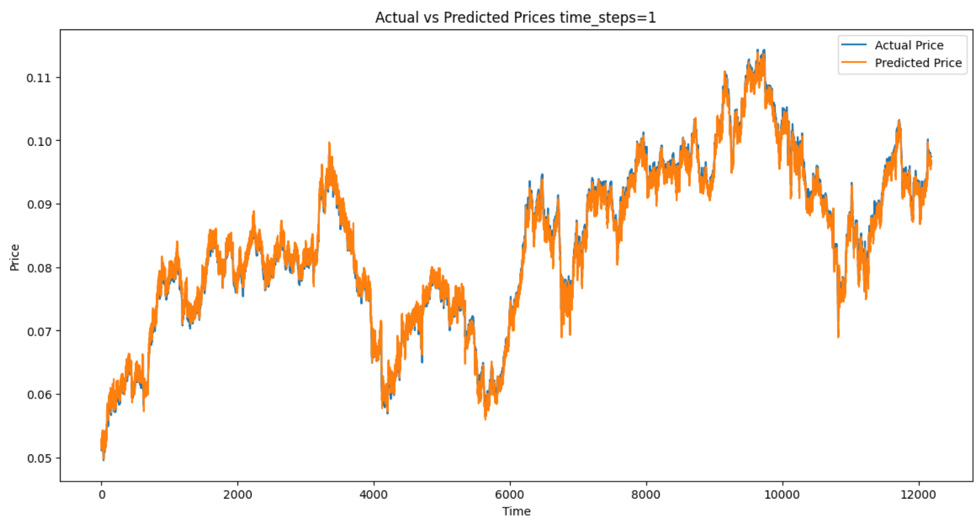
In this study, the performance of the LSTM model was tested for different time steps: 1, 2, 5, and 10. The best performance was obtained when using 1-day time steps. The results are presented below.

Table 1 LSTM EVALUATION MATRIC

Metric	Value
MAE	0.0012
MSE	0.0000
rRMSE	0.0171
R^2	0.9883

The performance of the LSTM for a time step of 1 day was the best in all configurations tested, Table 1 presents the performance of the LSTM timestamp 1 model. The low MAE and MSE already indicate that the model predictions are very near the real values, meaning minimal errors. It is also of a very low rRMSE of 0.0171; thus, high accuracy the model in capturing the underlying pattern of the stock prices. Knowing the high R^2 of 0.9883, it interprets that almost 99% of the variance in stock prices is explained through this model; thus, this extremely good fit and its prediction power are followed. The optimal hyperparameters to this model were 500 units with a learning rate of 1e-4 using the Adam optimizer. It means that this setting provided a good balance of model complexity with learning rate, thus enabling an LSTM to learn from the time patterns in this very data.

Figure 4 LSTM Prediction



4.2 RNN MODEL

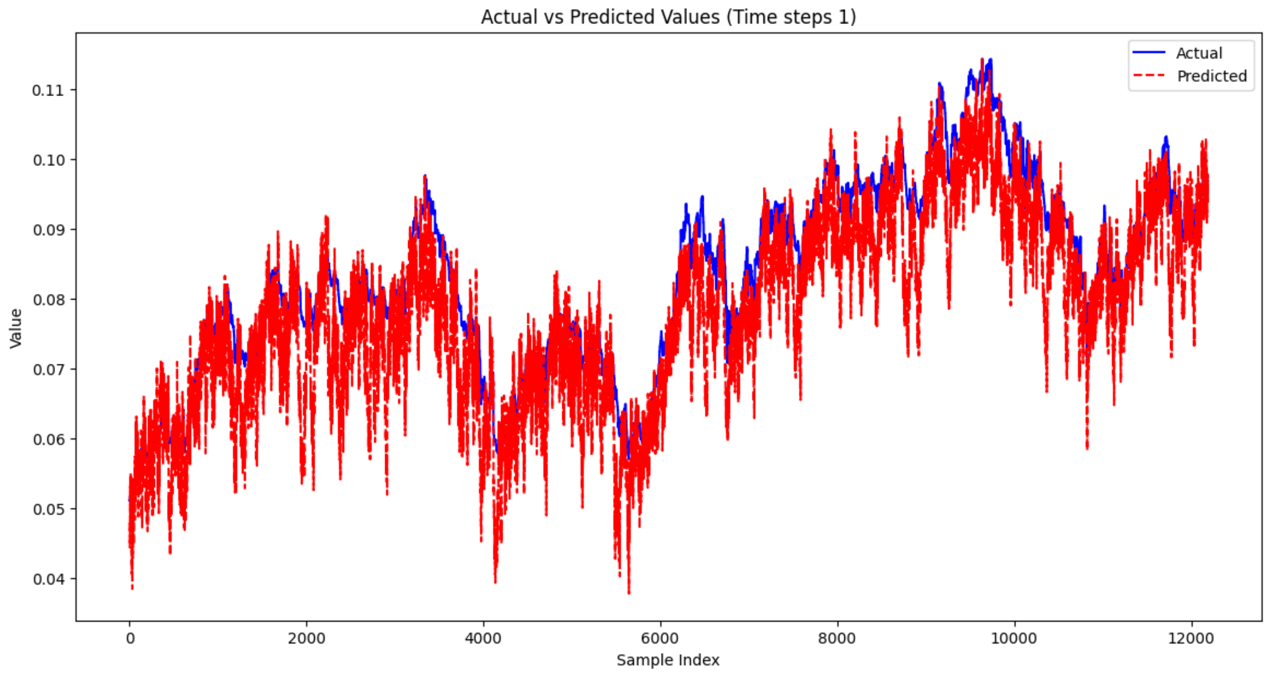
The Recurrent Neural Network (RNN) model was evaluated using the best parameters, which included 200 neurons, a learning rate of 0.0001, and the Adam optimizer. The results for the RNN model are presented by Table 2 as follows.

Table 2 RNN EVALUATION MATRIC

Metric	Value
MAE	0.0064
MSE	6.3102e-05
RMSE	0.0079
R^2	0.6415

The RNN model showed decent performance, but it was not as strong as the LSTM model. The Mean Absolute Error (MAE) and Mean Squared Error (MSE) were higher compared to the LSTM, indicating more prediction error. The Root Mean Squared Error (RMSE) of 0.0079 reflects a higher average error in predictions compared to the LSTM. The R^2 value of 0.6415 suggests that the RNN model explains approximately 64% of the variance, which is lower than the LSTM. The best parameters for the RNN model were 200 neurons, a learning rate of 0.0001, and the Adam optimizer. While the RNN model has some predictive power, it appears less effective than the LSTM in capturing long-term dependencies and patterns in the stock price data.

Figure 5 RNN Prediction



4.3 XGBOOST MODEL

Table 3 XGBOOST EVALUATION MATRIC

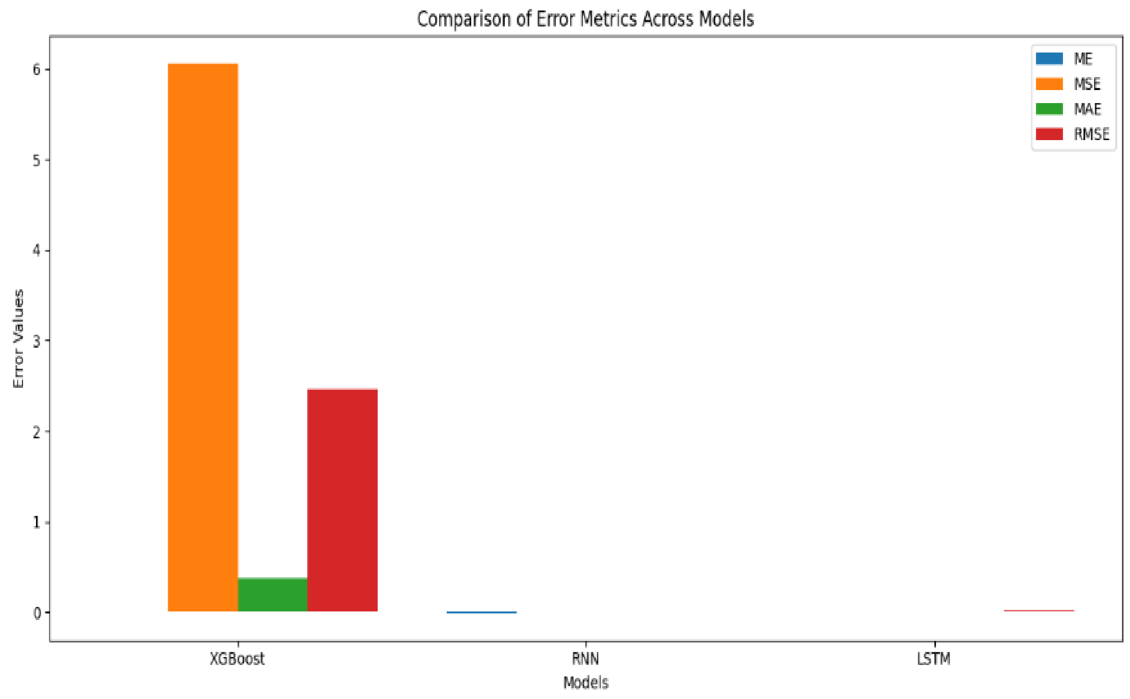
Metric	**Training Set**
Mean Error (ME)	0.00
Mean Squared Error (MSE)	5.25
Mean Absolute Error (MAE)	0.34
R-Squared	1.00
Root Mean Squared Error (RMSE)	2.29

The model returned an R^2 score of 1.00 for the training and test sets, indicating that the XGBoost model explains all of the variance in the data perfectly as presented by Table 3. This is a rather unusual outcome. It normally indicates that the model might be overfitting in the training data. However, going to the test set metrics, it is noticed that while the ME is 0.00—thus, there is no bias in predictions—the MSE and MAE are both higher than the training set values, reflecting some degree of error in predictions. It has an RMSE of 2.46 on the test set, indicating that its average prediction error is reasonable but nowhere near as minimal as the training set error. The best parameters for the XGBoost model were a learning rate of 0.1, a maximum depth of 10, and 100 estimators. Although it gives an R^2 score of 1.0 on the training set, this model performed well on the test set, which proves that it is strong against nonlinearities.

4.4 OVERALL ANALYSIS

1. The LSTM model did much better than the RNN model in terms of capturing long-term dependencies and temporal patterns in stock price data.
2. With a perfect R^2 score on both train and test sets, XGBoost truly performs very strong; however, test set metrics suggest some sort of error, which indicates that while it is handling nonlinear relationships pretty nice, there could still be some limitations to the model.
3. The LSTM model with a time step of 1 day stays the most accurate and reliable for this task, returning the best values of performance metrics in general. The RNN and especially the XGBoost models can provide very valuable alternatives, where the latter seems very robust against complex feature interactions and nonlinear relations.

Figure 6 Comparison of Model Error Matrices



CHAPTER 5 CONCLUSION

This study has been done upon the historical stock price data of companies listed in S&P 500 by many approaches to build models for stock price predictions. The main traction models belong to the category of LSTM, RNN, and XGBoost, each of which has its strengths and limitations. Quite good results have been presented with respect to catching the long-term dependency as well as the temporal structure present in the data on stock prices. Trained with a 1-day time step, it obtained the greatest result for the performance metrics among the models tried: a very high R^2 of 0.9883, very low MAE, and MSE values. This is a sign that the LSTM has learned about the sequential nature of the data of stock prices very well, resulting in providing accurate and reliable predictions. The RNN model was effective, but not as good as the LSTM model. It returned better error metrics than LSTM but had a very low R^2 value of 0.6415. Hence, suggest that while RNNs may get some temporal patterns, long-term dependency resolution will, in all probability, not be achieved properly. The prediction of the XGBoost model into optimized nonlinearities among features and their complex interactions was registered with an R^2 of 1.00 on the training data. To such a perfect score within the training set, the possibility of overfitting increased, which was evidenced by the escalation of error metrics within the test set. At any rate, the robustness of XGBoost to capture nonlinear relationships gives strong baseline alternatives in the stock price forecast.

Overall, the accuracy of this dataset showed that the LSTM model is pretty decently good and reliable, indicating its effectiveness in predicting time series. RNN and the XGBoost models presented pretty well but showed some limitations.

5.1 FUTURE WORK

The LSTM model, though fairly well-performing, still leaves some space to improve its accuracy using more complex network architectures in the model, which might be stacked LSTMs or bidirectional LSTMs, to capture even more complex temporal patterns together. Further fine-tuning of the hyperparameters and applying advanced regularization techniques might add a little bit more of a boost. The inclusion of more technical indicators, macroeconomic factors, or alternate data sources, such as news sentiment, could enrich the feature set, making this model a lot more comprehensive and, hence, a better model in the end. The same is true for adding fundamental data more dynamically or adding external datasets to enrich the analysis. Research into alternate learning techniques such as Transformer models, or possibly even hybrid models, might provide newer opportunities to improve on the current work. Exploring such advanced techniques might result in newer methods for better enhancement in the prediction of stock prices more effectively.

CHAPTER 6 REFERENCES

Fenghua, W.E.N., Jihong, X.I.A.O., Zhifang, H.E. and Xu, G.O.N.G., 2014. Stock price prediction based on SSA and SVM. Procedia Computer Science, 31, pp.625-631.

Gao, P., Zhang, R. and Yang, X., 2020. The application of stock index price prediction with neural network. Mathematical and Computational Applications, 25(3), p.53.

Mohan, S., Mullapudi, S., Sammeta, S., Vijayvergia, P. and Anastasiu, D.C., 2019, April. Stock price prediction using news sentiment analysis. In 2019 IEEE fifth international conference on big data computing service and Applications (BigDataService) (pp. 205-208). IEEE.

Nabipour, M., Nayyeri, P., Jabani, H., Mosavi, A. and Salwana, E., 2020. Deep learning for stock market prediction. Entropy, 22(8), p.840.

Oukhouya, H., Kadiri, H., El Himdi, K. and Guerbaz, R., 2024. Forecasting International Stock Market Trends: XGBoost, LSTM, LSTM-XGBoost, and Backtesting XGBoost Models. Statistics, Optimization & Information Computing, 12(1), pp.200-209.

Pawar, K., Jalem, R.S. and Tiwari, V., 2019. Stock market price prediction using LSTM RNN. In Emerging Trends in Expert Applications and Security: Proceedings of ICETEAS 2018 (pp. 493-503). Springer Singapore.

Rezaei, H., Faaljou, H. and Mansourfar, G., 2021. Stock price prediction using deep learning and frequency decomposition. Expert Systems with Applications, 169, p.114332.

Shi, Z., Hu, Y., Mo, G. and Wu, J., 2022. Attention-based CNN-LSTM and XGBoost hybrid model for stock prediction. arXiv preprint arXiv:2204.02623.

Vijh, M., Chandola, D., Tikkiwal, V.A. and Kumar, A., 2020. Stock closing price prediction using machine learning techniques. Procedia computer science, 167, pp.599-606.

Zheng, A. and Jin, J., 2017. Using ai to make predictions on stock market. cs229. stanford. edu.

APPENDIX

```
# -*- coding: utf-8 -*-
"""Khubaib_(SE_price_prediction).ipynb

Automatically generated by Colab.

Original file is located at
https://colab.research.google.com/drive/1lOGhfV9kjmQ2JIVHion0ssOSaY4tjrJJ
## Mount to Google Drive
"""
from google.colab import drive
drive.mount('/content/drive')

"""## Import Packages"""

import os
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, LSTM
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import ParameterGrid
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

"""## Load datasets"""

# Load datasets
prices = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/prices.csv')
prices_split_adjusted = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/prices-split-adjusted.csv')
securities = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/securities.csv')
fundamentals = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/fundamentals.csv')

# Merge prices and prices_split_adjusted on 'symbol' and 'date'
merged_prices = pd.merge(prices, prices_split_adjusted, on=['symbol', 'date'], suffixes=('', '_adjusted'))

# Merge with securities on 'symbol' and 'Ticker symbol'
merged_data = pd.merge(merged_prices, securities, left_on='symbol', right_on='Ticker symbol')

# Merge with fundamentals on 'symbol' and 'Ticker symbol'
merged_data = pd.merge(merged_prices, fundamentals, left_on='symbol', right_on='Ticker Symbol')

"""# EDA
```

```
### 1. Display first few rows of each dataset
"""

prices.head()

prices_split_adjusted.head()

securities.head()

fundamentals.head()

"""### 2. Summary of the datasets"""

print("Price Dataset")
print(prices.info())

print("Prices_split_adjusted Dataset")
print(prices_split_adjusted.info())

print("Securities Dataset")
print(securities.info())

print("Fundamentals Dataset")
print(fundamentals.info())

"""### 3. Check for Missing Values"""
```

```
print("Price Dataset")
print(prices.isnull().sum())

print("Prices_split_adjusted Dataset")
print(prices_split_adjusted.isnull().sum())

print("Securities Dataset")
print(securities.isnull().sum())

# Print the dataset name
print("Fundamentals Dataset")

# Print column-wise null value counts
for column in fundamentals.columns:
print(f"{column}: {fundamentals[column].isnull().sum()} records")

"""### 4. Descriptive Statistics"""

print("Prices Dataset")
prices.describe()

print("Prices_split_adjusted Dataset")
prices_split_adjusted.describe()

print("Securities Dataset")
securities.describe()
```

```
print("Fundamentals Dataset")
fundamentals.describe()

"""###  Distribution of sectors"""


# Distribution of sectors
plt.figure(figsize=(10, 6))
securities['GICS Sector'].value_counts().plot(kind='bar')
plt.title('Distribution of Sectors')
plt.xlabel('GICS Sector')
plt.ylabel('Number of Companies')
plt.xticks(rotation=80)
plt.show()


sample_symbols = prices['symbol'].unique()[:10]
sample_prices = prices[prices['symbol'].isin(sample_symbols)]


plt.figure(figsize=(14, 7))
for symbol in sample_symbols:
plt.plot(sample_prices[sample_prices['symbol'] == symbol]['date'],
sample_prices[sample_prices['symbol'] == symbol]['close'], label=symbol)
plt.legend()
plt.title('Stock Price Trends for Sample Companies')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.xticks([])
plt.xlabel('Date')
plt.show()

merged_data.info()

"""### Correlation matrix"""

# Exclude 'symbol' and 'date' columns from correlation calculation
cols_to_exclude = ['symbol', 'date', 'Ticker Symbol', "For Year", "Period Ending", "Unnamed: 0"]
cols_to_include = [col for col in merged_data.columns if col not in cols_to_exclude]
corr_matrix = merged_data[cols_to_include].corr()

# Plotting the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap of Merged Dataset')
plt.show()

# Assuming you have already calculated the correlation matrix corr_matrix
# List of key variables
key_variables = ['open','open_adjusted','close', 'close_adjusted', 'high','high_adjusted',
'low','low_adjusted', 'volume', "volume_adjusted", 'Earnings Per Share']

# Find correlations of key variables with all other features
correlations_with_key_vars = corr_matrix[key_variables].drop(key_variables)

# Plotting the heatmap
plt.figure(figsize=(20, 20))
```

```
sns.heatmap(correlations_with_key_vars, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)

plt.title('Correlations of Key Variables with Other Features')

plt.show()


correlations_with_key_vars


# Assuming you have the correlation matrix provided earlier

# Here's a subset of relevant columns based on the correlation matrix you've shown

relevant_columns = ['open','open_adjusted','close', 'close_adjusted', 'high','high_adjusted',
'low','low_adjusted', 'volume', "volume_adjusted", 'Earnings Per Share']


# Subset the correlation matrix

relevant_corr_matrix = corr_matrix.loc[relevant_columns, relevant_columns]


# Plotting the heatmap for relevant correlations

plt.figure(figsize=(10, 8))

sns.heatmap(relevant_corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap of Key Variables for Stock Price Prediction')

plt.show()


"""## Pre-processing


## Feature Selection
"""


# Selecting initial important features

initial_features = [
'date', 'symbol', 'open', 'close','low', 'high', 'volume',
'Earnings Before Interest and Tax', 'Net Income', 'Operating Income', 'Gross Profit',
'Profit Margin', 'Gross Margin', 'Operating Margin', 'Pre-Tax Margin', 'After Tax ROE',
'Earnings Per Share', 'Current Ratio', 'Quick Ratio', 'Cash Ratio', 'Long-Term Debt',
'Short-Term Debt / Current Portion of Long-Term Debt', 'Total Liabilities', 'Total Equity',
'Net Cash Flow', 'Net Cash Flow-Operating', 'Net Cash Flows-Financing',
'Net Cash Flows-Investing', 'Estimated Shares Outstanding', 'Sale and Purchase of Stock'
]

# Drop rows with missing values in the selected features
df_sf = merged_data[initial_features].dropna()

# Convert 'date' column to datetime if not already done
df_sf['date'] = pd.to_datetime(df_sf['date'])

# Check the updated dataframe
df_sf

"""## Extract New Features"""

!pip install ta

import ta
from ta.volatility import AverageTrueRange, BollingerBands
from ta.trend import MACD, ADXIndicator
from ta.momentum import RSIIndicator, StochasticOscillator, ROCIndicator
from ta.trend import EMAIndicator,  CCIIndicator
from ta.volume import OnBalanceVolumeIndicator, VolumePriceTrendIndicator, MFIIndicator
from ta.volatility import KeltnerChannel


# Sort the data by date for each symbol
df_sf = df_sf.sort_values(by=['symbol', 'date'])


# Initialize new columns for technical indicators
df_sf['RSI'] = df_sf.groupby('symbol')['close'].transform(lambda x: RSIIndicator(x, window=14).rsi())
df_sf['MV20'] = df_sf.groupby('symbol')['close'].transform(lambda x: x.rolling(window=20).mean())
df_sf['MV50'] = df_sf.groupby('symbol')['close'].transform(lambda x: x.rolling(window=50).mean())
df_sf['MV200'] = df_sf.groupby('symbol')['close'].transform(lambda x: x.rolling(window=200).mean())


macd = df_sf.groupby('symbol')['close'].apply(lambda x: MACD(x).macd())
df_sf['MACD'] = macd.reset_index(level=0, drop=True)


adx = df_sf.groupby('symbol').apply(lambda x: ADXIndicator(x['high'], x['low'], x['close']).adx())
df_sf['ADX'] = adx.reset_index(level=0, drop=True)


# Manually calculate the Accumulation/Distribution (AD) line
def calculate_ad(high, low, close, volume):
clv = ((close - low) - (high - close)) / (high - low)
clv = clv.fillna(0)  # Fill NaN values
ad = (clv * volume).cumsum()
return ad
```

```
df_sf['AD'] = df_sf.groupby('symbol').apply(lambda x: calculate_ad(x['high'], x['low'], x['close'], x['volume'])).reset_index(level=0, drop=True)


# Bollinger Bands

bollinger = df_sf.groupby('symbol')['close'].apply(lambda x: BollingerBands(close=x).bollinger_mavg())

df_sf['Bollinger_MAVG'] = bollinger.reset_index(level=0, drop=True)

df_sf['Bollinger_High'] = df_sf.groupby('symbol')['close'].transform(lambda x: BollingerBands(close=x).bollinger_hband())

df_sf['Bollinger_Low'] = df_sf.groupby('symbol')['close'].transform(lambda x: BollingerBands(close=x).bollinger_lband())


# Stochastic Oscillator

stochastic = df_sf.groupby('symbol').apply(lambda x: StochasticOscillator(x['high'], x['low'], x['close']).stoch())

df_sf['Stochastic'] = stochastic.reset_index(level=0, drop=True)


# Rate Of Change (ROC)

roc = df_sf.groupby('symbol')['close'].apply(lambda x: ROCIndicator(x).roc())

df_sf['ROC'] = roc.reset_index(level=0, drop=True)


# Drop rows with NaN values generated by technical indicators

df_sf = df_sf.dropna(subset=['RSI', 'MV20', 'MV50', 'MV200', 'MACD', 'ADX', 'AD'])


# Compute Stochastic Oscillator

df_sf['stoch'] = StochasticOscillator(high=df_sf['high'], low=df_sf['low'], close=df_sf['close'], window=14).stoch()


# Compute ROC (Rate of Change)
```

```
df_sf['roc'] = ROCIndicator(close=df_sf['close'], window=14).roc()

# Compute Exponential Moving Average (EMA)

df_sf['ema_20'] = EMAIndicator(close=df_sf['close'], window=20).ema_indicator()

df_sf['ema_50'] = EMAIndicator(close=df_sf['close'], window=50).ema_indicator()

# Compute On-Balance Volume (OBV)

df_sf['obv'] = OnBalanceVolumeIndicator(close=df_sf['close'],
volume=df_sf['volume']).on_balance_volume()

# Compute Volume Price Trend (VPT)

df_sf['vpt'] = VolumePriceTrendIndicator(close=df_sf['close'],
volume=df_sf['volume']).volume_price_trend()

# Compute Commodity Channel Index (CCI)

df_sf['cci'] = CCIIndicator(high=df_sf['high'], low=df_sf['low'], close=df_sf['close'],
window=20).cci()

# Compute Money Flow Index (MFI)

df_sf['mfi'] = MFIIndicator(high=df_sf['high'], low=df_sf['low'], close=df_sf['close'],
volume=df_sf['volume'], window=14).money_flow_index()

# Compute Keltner Channel

indicator_kc = KeltnerChannel(high=df_sf['high'], low=df_sf['low'], close=df_sf['close'],
window=20)

df_sf['kc_middle'] = indicator_kc.keltner_channel_mband()

df_sf['kc_upper'] = indicator_kc.keltner_channel_hband()
```

```
df_sf['kc_lower'] = indicator_kc.keltner_channel_lband()

df_sf.columns

"""# Model Traing and Testing"""

# # Save the scaled DataFrame to a CSV file
# df_sf.to_csv("/content/drive/MyDrive/NYSE_Dataset/df_sf.csv", index=False)

df_sf = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/df_sf.csv')

# List of ten symbols to filter
symbols_to_keep = ['AAPL', 'MSFT','GOOGL', 'AMZN', 'FB', 'TSLA', 'NVDA', 'NFLX', 'JPM',
'CSCO',
'WMT', 'XOM', 'PG', 'AMD', 'IBM']

# Assuming df_sf is your DataFrame
stock_df = df_sf[df_sf['symbol'].isin(symbols_to_keep)]

# Select the features for the RNN
selected_features = [
'open', 'close', 'low', 'high', 'volume',
'Earnings Before Interest and Tax', 'Net Income', 'Operating Income',
'Gross Profit', 'Profit Margin', 'Gross Margin', 'Operating Margin',
'Pre-Tax Margin', 'After Tax ROE', 'Earnings Per Share',
'Current Ratio', 'Quick Ratio', 'Cash Ratio', 'Long-Term Debt',
'Short-Term Debt / Current Portion of Long-Term Debt',
'Total Liabilities', 'Total Equity', 'Net Cash Flow',
'Net Cash Flow-Operating', 'Net Cash Flows-Financing',
'Net Cash Flows-Investing', 'Estimated Shares Outstanding',
'Sale and Purchase of Stock', 'RSI', 'MV20', 'MV50', 'MV200', 'MACD',
'ADX', 'AD', 'Bollinger_MAVG', 'Bollinger_High', 'Bollinger_Low',
'Stochastic', 'ROC', 'stoch', 'roc', 'ema_20', 'ema_50', 'obv', 'vpt',
'cci', 'mfi', 'kc_middle', 'kc_upper', 'kc_lower']

# Scale the features
scaler = MinMaxScaler()
stock_df[selected_features] = scaler.fit_transform(stock_df[selected_features])

# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Fit and transform the 'symbol' column
stock_df['symbol_encoded'] = label_encoder.fit_transform(stock_df['symbol'])

# Drop the original 'symbol' column if no longer needed
stock_df = stock_df.drop(columns=['symbol'])

# Define X (features) and y (target)
X = stock_df.drop(['date', 'close'], axis=1)
y = stock_df['close']

# Convert to numpy arrays
X = np.array(X)
y = np.array(y)
```

```
"""## Make Time steps for RNN and Lstm model"""

time_steps_list = [1, 2, 5, 10]
base_dir = '/content/drive/MyDrive/NYSE_Dataset'  # Base directory to save data

# Create the base directory if it doesn't exist
if not os.path.exists(base_dir):
os.makedirs(base_dir)

for time_steps in time_steps_list:
X_lstm = []
y_lstm = []

for i in range(time_steps, len(X)):
X_lstm.append(X[i-time_steps:i])
y_lstm.append(y[i])

X_lstm, y_lstm = np.array(X_lstm), np.array(y_lstm)

# Split the data into training and testing sets
split = int(0.8 * len(X_lstm))
X_train, X_test = X_lstm[:split], X_lstm[split:]
y_train, y_test = y_lstm[:split], y_lstm[split:]

print(f"Time steps: {time_steps}")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print("\n")

# Create a directory for this time step
dir_path = os.path.join(base_dir, f'time_steps_{time_steps}')
if not os.path.exists(dir_path):
os.makedirs(dir_path)

# Save the datasets
np.save(os.path.join(dir_path, 'X_train.npy'), X_train)
np.save(os.path.join(dir_path, 'X_test.npy'), X_test)
np.save(os.path.join(dir_path, 'y_train.npy'), y_train)
np.save(os.path.join(dir_path, 'y_test.npy'), y_test)

"""## Define Lstm Model"""

# Function to load data for a given time step
def load_data(time_steps):
dir_path = f'/content/drive/MyDrive/NYSE_Dataset/time_steps_{time_steps}'
X_train = np.load(os.path.join(dir_path, 'X_train.npy'))
X_test = np.load(os.path.join(dir_path, 'X_test.npy'))
y_train = np.load(os.path.join(dir_path, 'y_train.npy'))
y_test = np.load(os.path.join(dir_path, 'y_test.npy'))
return X_train, X_test, y_train, y_test
```

```
# Function to build the LSTM model
def build_model(units, learning_rate):
model = Sequential()
model.add(LSTM(units=units, activation='tanh', return_sequences=False,
input_shape=(time_steps, num_features)))
model.add(Dropout(0.2))
model.add(Dense(units=1))
optimizer = Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='mean_squared_error')
return model

"""## Model Training"""

# List of time steps
time_steps_list = [1, 2, 5, 10]

# Define the parameter grid
param_grid = {
'units': [200, 300, 500],
'learning_rate': [1e-4, 5e-4]
}

# Iterate through each time step, perform grid search, and save the best model
for time_steps in time_steps_list:
# Load the data
X_train, X_test, y_train, y_test = load_data(time_steps)

# Define the number of features (assumed to be same for all time steps)
```

```
num_features = X_train.shape[2]

best_val_loss = float('inf')
best_model = None

# Iterate through the parameter grid
for params in ParameterGrid(param_grid):
units = params['units']
learning_rate = params['learning_rate']

# Build the model
model = build_model(units, learning_rate)

# EarlyStopping callback
early_stopping = EarlyStopping(
monitor='val_loss',
patience=2,
restore_best_weights=True
)

# Train the model
history = model.fit(
X_train, y_train,
epochs=300,
batch_size=32,
validation_data=(X_test, y_test),
callbacks=[early_stopping],
verbose=1
)

# Get the validation loss of the best epoch
val_loss = min(history.history['val_loss'])

# Update the best model if the current one is better
if val_loss < best_val_loss:
best_val_loss = val_loss
best_model = model

# Save the best model
best_model.save(f'/content/drive/MyDrive/NYSE_Dataset/best_model_time_steps_{time_steps}.
h5')

print(f"Best model for time_steps={time_steps} saved with val_loss={best_val_loss}.")

# Load the data
df_sf = pd.read_csv('/content/drive/MyDrive/NYSE_Dataset/df_sf.csv')

"""## Evaluate the five model to select the best one among them"""

# Function to calculate rRMSE
def root_relative_mean_squared_error(y_true, y_pred):
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
rrmse = rmse / np.mean(y_true)
return rrmse
```

```
# Function to load data for a given time step
def load_data(time_steps):
dir_path = f'/content/drive/MyDrive/NYSE_Dataset/time_steps_{time_steps}'
X_train = np.load(os.path.join(dir_path, 'X_train.npy'))
X_test = np.load(os.path.join(dir_path, 'X_test.npy'))
y_train = np.load(os.path.join(dir_path, 'y_train.npy'))
y_test = np.load(os.path.join(dir_path, 'y_test.npy'))
return X_train, X_test, y_train, y_test

# List of time steps
time_steps_list = [1, 2, 5, 10]

# Iterate through each time step and visualize actual vs predicted prices
for time_steps in time_steps_list:
# Load the data
X_train, X_test, y_train, y_test = load_data(time_steps)

# Load the best model
model_path = f'/content/drive/MyDrive/NYSE_Dataset/best_model_time_steps_{time_steps}.h5'
best_model = load_model(model_path)

# Make predictions
y_pred = best_model.predict(X_test)

# Calculate metrics
me = np.mean(y_pred - y_test)
mae = mean_absolute_error(y_test, y_pred)
```

```
mse = mean_squared_error(y_test, y_pred)
rrmse = root_relative_mean_squared_error(y_test, y_pred)

# Print metrics
print(f"Metrics for time_steps={time_steps}:")
print(f"ME: {me:.2f}%")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"rRMSE: {rrmse:.4f}\n")

"""# LSTM Best Model
## The best model is time steps 1
"""

model_path = f'/content/drive/MyDrive/NYSE_Dataset/best_model_time_steps_{1}.h5'
lstm_best_model = load_model(model_path)

X_train, X_test, y_train, y_test = load_data(1)

y_pred = lstm_best_model.predict(X_test)

# Calculate metrics
lstm_me = np.mean(y_pred - y_test)
lstm_mae = mean_absolute_error(y_test, y_pred)
lstm_mse = mean_squared_error(y_test, y_pred)
lstm_rrmse = root_relative_mean_squared_error(y_test, y_pred)
lstm_r_squared = r2_score(y_test, y_pred)
```

```
# Print metrics
print(f"Metrics for time_steps={time_steps}:")
print(f"ME: {lstm_me:.2f}%")
print(f"MAE: {lstm_mae:.4f}")
print(f"MSE: {lstm_mse:.4f}")
print(f"rRMSE: {lstm_rrmse:.4f}\n")
print(f"R_squared: {lstm_r_squared:.4f}\n")

"""# LSTM best model Summary"""

lstm_best_model.summary()

print("Optimizer:", lstm_best_model.optimizer)
print("Loss Function:", lstm_best_model.loss)
print("Units:", lstm_best_model.get_layer(index=0).units)
print("Learning Rate:", lstm_best_model.optimizer.learning_rate.numpy())

"""## Visualization of Actual vs Predict close price for Lstm"""

# Visualize actual vs predicted prices
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='Actual Price')
plt.plot(y_pred, label='Predicted Price')
plt.title(f'Actual vs Predicted Prices time_steps={1}')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()

plt.show()


"""## Visualization of a specific Stocks"""


# List of symbols to filter

symbols_to_keep = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'FB', 'TSLA', 'NVDA', 'NFLX', 'JPM',
'CSCO',

'WMT', 'XOM', 'PG', 'AMD', 'IBM']


# Filter the DataFrame

stock_df = df_sf[df_sf['symbol'].isin(symbols_to_keep)]


# Select the features for the RNN

selected_features = [

'open', 'close', 'low', 'high', 'volume',

'Earnings Before Interest and Tax', 'Net Income', 'Operating Income',

'Gross Profit', 'Profit Margin', 'Gross Margin', 'Operating Margin',

'Pre-Tax Margin', 'After Tax ROE', 'Earnings Per Share',

'Current Ratio', 'Quick Ratio', 'Cash Ratio', 'Long-Term Debt',

'Short-Term Debt / Current Portion of Long-Term Debt',

'Total Liabilities', 'Total Equity', 'Net Cash Flow',

'Net Cash Flow-Operating', 'Net Cash Flows-Financing',

'Net Cash Flows-Investing', 'Estimated Shares Outstanding',

'Sale and Purchase of Stock', 'RSI', 'MV20', 'MV50', 'MV200', 'MACD',

'ADX', 'AD', 'Bollinger_MAVG', 'Bollinger_High', 'Bollinger_Low',

'Stochastic', 'ROC', 'stoch', 'roc', 'ema_20', 'ema_50', 'obv', 'vpt',

'cci', 'mfi', 'kc_middle', 'kc_upper', 'kc_lower'
]

# Scale the features
scaler = MinMaxScaler()
stock_df[selected_features] = scaler.fit_transform(stock_df[selected_features])

# Encode the 'symbol' column
label_encoder = LabelEncoder()
stock_df['symbol_encoded'] = label_encoder.fit_transform(stock_df['symbol'])

# Drop the original 'symbol' column if no longer needed
stock_df = stock_df.drop(columns=['symbol'])

# Define a function to prepare the data for a specific stock and time steps
def prepare_data_for_stock(stock_df, stock_symbol, time_steps):
stock_data = stock_df[stock_df['symbol_encoded'] ==
label_encoder.transform([stock_symbol])[0]]
X = stock_data.drop(['date', 'close'], axis=1).values
y = stock_data['close'].values

X_time_steps = []
y_time_steps = []

for i in range(len(X) - time_steps):
X_time_steps.append(X[i:i + time_steps])
y_time_steps.append(y[i + time_steps])

return np.array(X_time_steps), np.array(y_time_steps)
```

```
# Predict prices for two stocks using Time Steps = 1
time_steps = 1


# Load the best model for Time Steps = 1
model_path = f'/content/drive/MyDrive/NYSE_Dataset/best_model_time_steps_{time_steps}.h5'
best_model = load_model(model_path)

predictions = {}


X_stock, y_stock = prepare_data_for_stock(stock_df, 'AAPL', time_steps)

y_pred = best_model.predict(X_stock)

# Store the predictions
predictions['AAPL'] = (y_stock, y_pred)

# Visualize actual vs predicted prices
plt.figure(figsize=(14, 7))
plt.plot(y_stock, label='Actual Price')
plt.plot(y_pred, label='Predicted Price')
plt.title(f'Actual vs Predicted Prices for Apple Inc with time_steps={time_steps}')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()

predictions = {}


X_stock, y_stock = prepare_data_for_stock(stock_df, 'AMZN', time_steps)

y_pred = best_model.predict(X_stock)

# Store the predictions
predictions['AMZN'] = (y_stock, y_pred)

# Visualize actual vs predicted prices
plt.figure(figsize=(14, 7))
plt.plot(y_stock, label='Actual Price')
plt.plot(y_pred, label='Predicted Price')
plt.title(f'Actual vs Predicted Prices for Amazon.com Inc with time_steps={time_steps}')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()

"""# RNN Model"""

# Function to load data for a given time step
def load_data(time_steps):
dir_path = f'/content/drive/MyDrive/NYSE_Dataset/time_steps_{time_steps}'
```

```
X_train = np.load(os.path.join(dir_path, 'X_train.npy'))
X_test = np.load(os.path.join(dir_path, 'X_test.npy'))
y_train = np.load(os.path.join(dir_path, 'y_train.npy'))
y_test = np.load(os.path.join(dir_path, 'y_test.npy'))
return X_train, X_test, y_train, y_test

def build_rnn_model(input_shape, neurons, learning_rate, optimizer_name):
model = Sequential([
SimpleRNN(neurons, activation='tanh', input_shape=input_shape, return_sequences=True),
Dropout(0.2),
SimpleRNN(neurons, activation='tanh'),
Dropout(0.2),
Dense(1)
])

if optimizer_name == 'adam':
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.999)
elif optimizer_name == 'adamax':
optimizer = tf.keras.optimizers.Adamax(learning_rate=learning_rate)
else:
raise ValueError("Unsupported optimizer")

model.compile(optimizer=optimizer, loss='mean_squared_error')
return model

"""## Define parameter for grid search and model training"""
```

```
rnn_param_grid = {
'neurons': [100, 200, 500],
'learning_rate': [0.0001, 0.00005],
'optimizer': ['adam', 'adamax']
}

# List of time steps
time_steps_list = [1, 2, 5, 10]

# Base directory to save models
model_base_dir = '/content/drive/MyDrive/NYSE_Dataset/rnn_models'
if not os.path.exists(model_base_dir):
os.makedirs(model_base_dir)

# Initialize early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Store results for each time step
all_results = {}

for time_steps in time_steps_list:
print(f"\nProcessing time steps: {time_steps}")

# Load the data
X_train, X_test, y_train, y_test = load_data(time_steps)

# Prepare to store results for the current time step
```

```
results = []

# Perform grid search
for params in ParameterGrid(rnn_param_grid):
print(f"Training with parameters: {params}")

neurons = params['neurons']
learning_rate = params['learning_rate']
optimizer_name = params['optimizer']

# Build the model
model = build_rnn_model(input_shape=(X_train.shape[1], X_train.shape[2]),
neurons=neurons,
learning_rate=learning_rate,
optimizer_name=optimizer_name)

# Train the model
history = model.fit(X_train, y_train,
epochs=20,
batch_size=32,
validation_split=0.2,
callbacks=[early_stopping],
verbose=1)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
```

```
results.append({
'params': params,
'history': history,
'loss': loss,
'model': model
})
print(f"Test Loss: {loss}\n")

# Find the best result for the current time step
best_result = min(results, key=lambda x: x['loss'])
all_results[time_steps] = best_result

# Save the best model for the current time step
best_model_path = os.path.join(model_base_dir, f'best_model_time_steps_{time_steps}.h5')
best_result['model'].save(best_model_path)
print(f"Saved best model for time steps {time_steps} at {best_model_path}")

print(f"Best parameters for time steps {time_steps}: {best_result['params']}")
print(f"Best loss for time steps {time_steps}: {best_result['loss']}")

# Select the best time step based on the lowest loss
best_time_steps = min(all_results, key=lambda k: all_results[k]['loss'])
best_params = all_results[best_time_steps]['params']
print(f"\nBest time steps: {best_time_steps}")
print(f"Best parameters: {best_params}")

# Define the best time step and parameters
```

```
rnn_best_time_steps = 1

rnn_best_model_path =
f'/content/drive/MyDrive/NYSE_Dataset/rnn_models/best_model_time_steps_{rnn_best_time_st
eps}.h5'


# Load the data for the best time step

X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn = load_data(rnn_best_time_steps)


# Load the best model

rnn_best_model = load_model(rnn_best_model_path)


# Evaluate the best model

rnn_loss = rnn_best_model.evaluate(X_test, y_test)

print(f"Final Best Model (Time steps {rnn_best_time_steps}): Test Loss = {rnn_loss}")


"""## Visualization of Actual vs Predicted Values for RNN"""


# Make predictions on the test data

rnn_y_pred = rnn_best_model.predict(X_test_rnn)


# Plot the actual vs predicted values

plt.figure(figsize=(14, 7))


# Plot the entire test data

plt.plot(np.arange(len(y_test_rnn)), y_test_rnn, color='blue', label='Actual')

plt.plot(np.arange(len(rnn_y_pred)), rnn_y_pred, color='red', linestyle='dashed',
label='Predicted')


plt.title(f'Actual vs Predicted Values (Time steps {rnn_best_time_steps})')
plt.xlabel('Sample Index')
plt.ylabel('Value')
plt.legend()
plt.show()

# Calculate Mean Error (ME)
rnn_me = np.mean(rnn_y_pred - y_test_rnn)

# Calculate Mean Squared Error (MSE)
rnn_mse = mean_squared_error(y_test_rnn, rnn_y_pred)

# Calculate Root Mean Squared Error (RMSE)
rnn_rmse = np.sqrt(rnn_mse)

# Calculate R-squared (R²)
rnn_r2 = r2_score(y_test_rnn, rnn_y_pred)

rnn_mae = mean_absolute_error(y_test_rnn, rnn_y_pred)

# Print the metrics
print(f"Mean Error (ME): {rnn_me}")
print(f"Mean Squared Error (MSE): {rnn_mse}")
print(f"Mean Absolute Error (MAE): {rnn_mae}")
print(f"Root Mean Squared Error (RMSE): {rnn_rmse}")
print(f"R-squared (R²): {rnn_r2}")

"""# XGBoost Model
```

```
## Pre-processing for XGBoost
"""

# Drop rows with any missing values
df_sf_clean = df_sf.dropna()

# Remove the 'date' column and label encode 'symbol'
svm_df = df_sf_clean.drop(columns=['date'])
label_encoder = LabelEncoder()
svm_df['symbol'] = label_encoder.fit_transform(svm_df['symbol'])



X = svm_df.drop('close', axis=1)
y = svm_df['close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# MinMaxScaler the features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

"""## Grid Search the model"""

# Define the parameter grid for XGBoost
```

```
param_grid = {
'n_estimators': [50, 100],
'learning_rate': [0.01, 0.1],
'max_depth': [5, 10],
}

# Initialize the XGBRegressor model
xgb_model = XGBRegressor(objective='reg:squarederror')

# Setup GridSearchCV with cross-validation and verbosity
grid_search = GridSearchCV(
estimator=xgb_model,
param_grid=param_grid,
cv=5,
scoring='neg_mean_squared_error',
n_jobs=-1,
verbose=2
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Score (Negative Mean Squared Error): {grid_search.best_score_:.2f}")

# import joblib
# # To load the model later
# best_model = joblib.load('/content/drive/MyDrive/NYSE_Dataset/xg_boost_best_model.pkl')

# Evaluate the best model on the test set
best_model = grid_search.best_estimator_



# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate Mean Error (ME)
xg_mean_error = np.mean(y_pred - y_test)

# Calculate Mean Squared Error (MSE)
xg_mse = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
xg_rmse = np.sqrt(xg_mse)

# Calculate Mean Absolute Error (MAE)
xg_mae = mean_absolute_error(y_test, y_pred)

# Calculate R-squared (R²)
xg_r_squared = r2_score(y_test, y_pred)
```

```
# Print the metrics
print(f"Test Set Mean Error (ME): {xg_mean_error:.2f}")
print(f"Test Set Mean Squared Error (MSE): {xg_mse:.2f}")
print(f"Test Set Mean Absolute Error (MAE): {xg_mae:.2f}")
print(f"Test Set Root Mean Squared Error (RMSE): {xg_rmse:.2f}")
print(f"Test Set R-squared (R²): {xg_r_squared:.2f}")

# import joblib

# # Save the best model to a file
# joblib.dump(best_model, '/content/drive/MyDrive/NYSE_Dataset/xg_boost_best_model.pkl')

"""# Comparision of the Three Best Models"""

import matplotlib.pyplot as plt
import numpy as np

# Data for the models
models = ['XGBoost', 'RNN', 'LSTM']
metrics = {
'ME': [xg_mean_error, rnn_me, lstm_me],
 'MSE': [xg_mse, rnn_mse, lstm_mse],
 'MAE': [xg_mae, rnn_mae, lstm_mae],
'RMSE': [xg_rmse, rnn_rmse, lstm_rrmse]
}

# Number of models and metrics
```

```
num_models = len(models)
num_metrics = len(metrics)

# Set up the bar plot
fig, ax = plt.subplots(figsize=(12, 6))
bar_width = 0.2
index = np.arange(num_models)

# Create bars for each metric
for i, (metric, values) in enumerate(metrics.items()):
bars = ax.bar(index + i * bar_width, values, bar_width, label=metric)

# Add labels, title, and legend
ax.set_xlabel('Models')
ax.set_ylabel('Error Values')
ax.set_title('Comparison of Error Metrics Across Models')
ax.set_xticks(index + bar_width * (num_metrics / 2 - 0.5))
ax.set_xticklabels(models)
ax.legend()

# Display the plot
plt.tight_layout()
plt.show()

"""# After Compairsion LSTM is the best model among them"""
```