**FACULTY OF ENGINEERING, COMPUTING AND THE ENVIRONMENT**

**School of Computer Science and Mathematics**

**MSc DEGREE**

**IN**

**Data Science**

Name: Zakir Zahir

ID Number: K2016609

**Project Title: "Predicting Future Stock Prices: A Deep Learning Approach using Historical Stock Price Data"**

Date: 18th September 2024

Supervisor: Dr Jad Abbass

KU London Logo

**WARRANTY STATEMENT**

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data or conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

**Table of Contents**

[List of Figures 6](#_Toc177392917)

[List of Tables 8](#_Toc177392918)

[Glossary of terms 9](#_Toc177392919)

[Abstract 10](#_Toc177392920)

[Acknowledgements 11](#_Toc177392921)

[Chapter 1: Introduction 12](#_Toc177392922)

[1.1: Background 12](#_Toc177392923)

[1.2: Motivation 12](#_Toc177392924)

[1.3: Stakeholders 13](#_Toc177392925)

[1.4: Research Problem 13](#_Toc177392926)

[1.5: Aims and Objectives 13](#_Toc177392927)

[1.5.1: Aims 13](#_Toc177392928)

[1.5.2: Objectives 13](#_Toc177392929)

[1.6: Ethics 14](#_Toc177392930)

[1.7: Summary of Thesis Contributions 15](#_Toc177392931)

[1.8: Thesis Outline 15](#_Toc177392932)

[Chapter 2: Literature Review 16](#_Toc177392933)

[2.1 Introduction to Stock Price Prediction 16](#_Toc177392934)

[2.1.1 The Importance of Stock Price Prediction in Financial Markets 16](#_Toc177392935)

[2.1.2 Challenges in Stock Price Prediction 18](#_Toc177392936)

[2.2 Emergence of Deep Learning in Financial Forecasting 19](#_Toc177392937)

[2.2.1 Overview of Deep Learning 19](#_Toc177392938)

[2.3 RNN-Based Models in Stock Price Prediction 20](#_Toc177392939)

[2.3.1 Long Short-Term Memory (LSTM) Networks 20](#_Toc177392940)

[2.3.2 Gated Recurrent Units (GRU) 22](#_Toc177392941)

[2.4 Convolutional Neural Networks (CNN) in Stock Price Prediction 24](#_Toc177392942)

[2.4.1 Introduction to CNN 24](#_Toc177392943)

[2.4.2 CNN in Financial Data 24](#_Toc177392944)

[2.5 Comparative Studies of CNN, GRU and LSTM in Financial Forecasting 25](#_Toc177392945)

[2.5.1 Comparative Review 26](#_Toc177392946)

[2.5.2 Model Effectiveness and Real-World Applications 26](#_Toc177392947)

[2.6 Ethical Aspects in Financial Forecasting 27](#_Toc177392948)

[2.6.1 Ethical Implications of Predictive Models 27](#_Toc177392949)

[2.7 Summary of Literature and Gaps Identified 28](#_Toc177392950)

[2.7.1 Summary of Literature 28](#_Toc177392951)

[2.7.2 Relevance to Current Study 28](#_Toc177392952)

[Chapter 3: Methodology and Data Preparation 29](#_Toc177392953)

[3.1 Introduction 29](#_Toc177392954)

[3.2 Data Collection and Exploration 29](#_Toc177392955)

[3.2.1 Data Sources and Description 30](#_Toc177392956)

[3.2.2 Exploratory Data Analysis (EDA) 31](#_Toc177392957)

[3.2.3 Data Preprocessing 36](#_Toc177392958)

[Chapter 4: Model Design and Development 39](#_Toc177392959)

[4.1 Introduction 39](#_Toc177392960)

[4.2 Base Model Design 39](#_Toc177392961)

[4.2.1 LSTM Base Model Design 39](#_Toc177392962)

[4.2.2 CNN Base Model Design 41](#_Toc177392963)

[4.2.3 GRU Base Model Design 42](#_Toc177392964)

[4.3 Feature Engineering and Model Design 42](#_Toc177392965)

[4.3.1 Introduction to Feature Engineering 42](#_Toc177392966)

[4.3.2 Feature-Engineered LSTM Model Design 44](#_Toc177392967)

[4.3.3 Feature-Engineered CNN Model Design 45](#_Toc177392968)

[4.3.4 Feature-Engineered GRU Model Design 45](#_Toc177392969)

[Chapter 5: Model Training 46](#_Toc177392970)

[5.1 Introduction 46](#_Toc177392971)

[5.2 Training Base Models 46](#_Toc177392972)

[5.2.1 Training LSTM Base Model 47](#_Toc177392973)

[5.2.2 Training CNN Base Model 48](#_Toc177392974)

[5.2.3 Training GRU Base Model 48](#_Toc177392975)

[5.3 Training Feature Engineered Models 49](#_Toc177392976)

[5.3.1 Training Feature-Engineered LSTM Model 49](#_Toc177392977)

[5.3.2 Training Feature-Engineered CNN Model 50](#_Toc177392978)

[5.3.3 Training Feature-Engineered GRU Model 51](#_Toc177392979)

[Chapter 6: Performance Evaluation and Comparative Analysis 51](#_Toc177392980)

[6.1 Introduction 51](#_Toc177392981)

[6.2 Evaluation of Base Models 52](#_Toc177392982)

[6.2.1 Evaluation of LSTM Base Models 52](#_Toc177392983)

[6.2.2 Evaluation of CNN Base Models 53](#_Toc177392984)

[6.2.3 Evaluation of GRU Base Models 54](#_Toc177392985)

[6.3 Evaluation of Feature-Engineered Models 55](#_Toc177392986)

[6.3.1 Evaluation of Feature-Engineered LSTM Models 55](#_Toc177392987)

[6.3.2 Evaluation of Feature-Engineered CNN Models 56](#_Toc177392988)

[6.3.3 Evaluation of Feature-Engineered GRU Models 57](#_Toc177392989)

[6.4 Future Stock Price Predictions 58](#_Toc177392990)

[6.4.1 Future Stock Price Predictions of LSTM Base Models 58](#_Toc177392991)

[6.4.2 Future Stock Price Predictions of CNN Base Models 59](#_Toc177392992)

[6.4.3 Future Stock Price Predictions of GRU Base Models 61](#_Toc177392993)

[6.4.4 Future Stock Price Predictions of Feature-Engineered LSTM Models 62](#_Toc177392994)

[6.4.5 Future Stock Price Predictions of Feature-Engineered CNN Models 64](#_Toc177392995)

[6.4.6 Future Stock Price Predictions of Feature-Engineered GRU Models 66](#_Toc177392996)

[6.5 Comparative Analysis 68](#_Toc177392997)

[6.5.1 Base vs. Feature-Engineered Models 68](#_Toc177392998)

[6.5.2 Strengths and Weaknesses 69](#_Toc177392999)

[6.5.3 Practical Implications 70](#_Toc177393000)

[Chapter 7: Conclusion 70](#_Toc177393001)

[7.1 Summary of Findings 70](#_Toc177393002)

[7.2 Implications for Financial Forecasting 71](#_Toc177393003)

[7.3 Limitations of the Study 72](#_Toc177393004)

[7.4 Future Research 73](#_Toc177393005)

[References 75](#_Toc177393006)

[Appendices 79](#_Toc177393007)

[LSTM Base and Feature Engineered Model Stock Price Prediction Code (Apple) 79](#_Toc177393008)

[LSTM Base and Feature Engineered Model Stock Price Prediction Code (Microsoft) 100](#_Toc177393009)

[CNN Base and Feature Engineered Model Stock Price Prediction Code (Apple) 121](#_Toc177393010)

[CNN Base and Feature Engineered Model Stock Price Prediction Code (Microsoft) 142](#_Toc177393011)

[GRU Base and Feature Engineered Model Stock Price Prediction Code (Apple) 162](#_Toc177393012)

[GRU Base and Feature Engineered Model Stock Price Prediction Code (Microsoft) 183](#_Toc177393013)

# List of Figures

[Figure 1. Layers of a Deep Learning Network (Chauhan and Singh, 2018) 19](#_Toc177204444)

[Figure 2. Structure of a LSTM Network (Graves and Pedrycz, 2009) 20](#_Toc177204445)

[Figure 3. Model Flow of GRU-LSTM Model (Trivedi and Patel, 2022) 23](#_Toc177204446)

[Figure 4. Layout of CNN-LSTM Model (Li and Xie, 2022) 24](#_Toc177204447)

[Figure 5. Structure of the CNN Model (Chen, 2023) 27](#_Toc177204448)

[Figure 6. Data types 31](#_Toc177204449)

[Figure 7. Apple stock closing prices over time 31](#_Toc177204450)

[Figure 8. Microsoft stock closing prices over time 32](#_Toc177204451)

[Figure 9. Apple Stock Volatility Over Time 33](#_Toc177204452)

[Figure 10. Microsoft Stock Volatility Over Time 33](#_Toc177204453)

[Figure 11. Apple Stock Daily Returns 34](#_Toc177204454)

[Figure 12. Microsoft Stock Daily Returns 34](#_Toc177204455)

[Figure 13. Distribution of Apple Stock Closing Prices 35](#_Toc177204456)

[Figure 14. Distribution of Microsoft Stock Closing Prices 35](#_Toc177204457)

[Figure 15. Data Preprocessing of Base Model code 36](#_Toc177204458)

[Figure 16. Data Preprocessing of LSTM Feature-Engineered Model code 37](#_Toc177204459)

[Figure 17. Data Preprocessing of CNN and GRU Feature-Engineered Model code 38](#_Toc177204460)

[Figure 18. Architecture of the LSTM base model 39](#_Toc177204461)

[Figure 19. Layers of the Base LSTM Model 40](#_Toc177204462)

[Figure 20. Architecture and Layers of the Base CNN Model 41](#_Toc177204463)

[Figure 21. Architecture and Layers of the Base GRU Model 42](#_Toc177204464)

[Figure 22. Architecture and Layers of the Feature-Engineered LSTM Model 44](#_Toc177204465)

[Figure 23. Architecture and Layers of the Feature-Engineered CNN Model 45](#_Toc177204466)

[Figure 24. Architecture and Layers of the Feature-Engineered GRU Model 46](#_Toc177204467)

[Figure 25. Model Summary of LSTM Base Model 47](#_Toc177204468)

[Figure 26. Model Summary of CNN Base Model 48](#_Toc177204469)

[Figure 27. Model Summary of GRU Base Model 48](#_Toc177204470)

[Figure 28. Model Summary of Feature-Engineered LSTM Model 49](#_Toc177204471)

[Figure 29. Model Summary of Feature-Engineered CNN Model 50](#_Toc177204472)

[Figure 30. Model Summary of Feature-Engineered GRU Model 51](#_Toc177204473)

[Figure 31. Evaluation metrics of Base LSTM model (Apple Stock Price) 52](#_Toc177204474)

[Figure 32. Evaluation metrics of Base LSTM model (Microsoft Stock Price) 52](#_Toc177204475)

[Figure 33. Evaluation metrics of Base CNN model (Apple Stock Price) 53](#_Toc177204476)

[Figure 34. Evaluation metrics of Base CNN model (Microsoft Stock Price) 53](#_Toc177204477)

[Figure 35. Evaluation metrics of Base GRU model (Apple Stock Price) 54](#_Toc177204478)

[Figure 36. Evaluation metrics of Base GRU model (Microsoft Stock Price) 54](#_Toc177204479)

[Figure 37. Evaluation metrics of Feature-Engineered LSTM model (Apple Stock Price) 55](#_Toc177204480)

[Figure 38. Evaluation metrics of Feature-Engineered LSTM model (Microsoft Stock Price) 55](#_Toc177204481)

[Figure 39. Evaluation metrics of Feature-Engineered CNN model (Apple Stock Price) 56](#_Toc177204482)

[Figure 40. Evaluation metrics of Feature-Engineered CNN model (Microsoft Stock Price) 56](#_Toc177204483)

[Figure 41. Evaluation metrics of Feature-Engineered GRU model (Apple Stock Price) 57](#_Toc177204484)

[Figure 42. Evaluation metrics of Feature-Engineered GRU model (Microsoft Stock Price) 57](#_Toc177204485)

[Figure 43. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Base Model) 58](#_Toc177204486)

[Figure 44. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Base Model) 59](#_Toc177204487)

[Figure 45. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Base Model) 60](#_Toc177204488)

[Figure 46. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Base Model) 61](#_Toc177204489)

[Figure 47. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Base Model) 61](#_Toc177204490)

[Figure 48. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Base Model) 62](#_Toc177204491)

[Figure 49. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Feature-Engineered Model) 63](#_Toc177204492)

[Figure 50. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Feature-Engineered Model) 64](#_Toc177204493)

[Figure 51. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Feature-Engineered Model) 65](#_Toc177204494)

[Figure 52. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Feature-Engineered Model) 66](#_Toc177204495)

[Figure 53. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Feature-Engineered Model) 67](#_Toc177204496)

[Figure 54. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Feature-Engineered Model) 68](#_Toc177204497)

# List of Tables

[Table 1. Objectives with their descriptions 14](#_Toc177204498)

[Table 2. Descriptions of the dataset columns 30](#_Toc177204499)

[Table 3. Description of technical indicators used in the feature-engineered models 43](#_Toc177204500)

# Glossary of terms

|  |  |
| --- | --- |
| ARIMA | Autoregressive Integrated Moving Average |
| CNN | Convolutional Neural Networks |
| EDA | Exploratory Data Analysis |
| EMA | Exponential Moving Average |
| EMH | Efficient Market Hypothesis |
| GRU | Gated Recurrent Units |
| LSTM | Long Short-Term Memory |
| MACD | Moving Average Convergence/Divergence |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| MSE | Mean Squared Error |
| NLP | Natural Language Processing |
| RMSE | Root Mean Squared Error |
| RNN | Recurrent Neural Networks |
| RSI | Relative Strength Index |
| SVM | Support Vector Machines |

# Abstract

This dissertation explores the use of deep learning techniques, specifically Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN) and Gated Recurrent Units (GRU) to predict stock prices using historical data. Stock price prediction is a complex task influenced by various factors such as market volatility. Traditional approaches, while useful, often struggle with the non-linear and dynamic nature of financial markets. The rise of deep learning models offers new possibilities for addressing these challenges due to their ability to capture temporal dependencies and patterns in large datasets.

This project develops and evaluates both base and feature-engineered versions of the LSTM, CNN and GRU models using technical indicators such as Moving Averages, RSI and MACD to improve the model’s predictive power. Using historical stock price data from Apple and Microsoft, the models are trained and tested on both time series data and feature-engineered data. Comparative analysis between the base and feature-engineered models highlights the impact of additional market indicators on prediction accuracy.

The evaluation metrics employed include Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) providing a robust comparison of model performance. The results demonstrate that feature-engineered models generally outperform their base counterparts capturing more complex market trends and reducing prediction error. The dissertation concludes by discussing the strengths, weaknesses and practical implications of these models for real-world financial forecasting alongside suggestions for future research to refine and improve stock price prediction methods using deep learning.

# Acknowledgements

I would like to extend my sincere thanks to my supervisor, Dr Jad Abbass, for his invaluable guidance and support throughout this project. His insights and feedback have been crucial in shaping the direction of my research.

Finally, I am forever grateful to my parents for their unwavering support. Their belief in me has been a constant source of motivation and I dedicate this achievement to them.

# Chapter 1: Introduction

## 1.1: Background

In recent years, the financial markets have witnessed unprecedented growth and volatility, fueled by rapid technological advancements and global economic shifts. (Boubaker et al., 2023). The ability to predict stock prices has become an area of intense research interest as accurate predictions can lead to significant financial gains. However, stock price prediction remains one of the most challenging tasks due to the complex, non-linear and dynamic nature of financial markets.

Traditional approaches to stock price forecasting have relied on statistical models such as ARIMA while useful often struggle to capture the intricacies of financial data (Vuong et al., 2024). Deep learning has introduced new possibilities with models such as LSTM networks and GRU showing promising results in time series forecasting tasks (Yang, 2021). These models which are part of the broader RNN family are particularly well-suited to capturing temporal dependencies making them ideal models to use for stock price prediction.

This dissertation explores the application of deep learning models, specifically CNN, GRU and LSTM networks to predict the stock prices of Apple and Microsoft. The study aims to compare the performance of these models and evaluate their effectiveness in capturing market trends and analyse their potential for real-world financial applications. By leveraging historical stock price data this research seeks to contribute to the growing body of knowledge on the use of advanced deep learning techniques in financial forecasting providing insights that could enhance decision-making processes for investors and financial analysts.

The subsequent chapters will detail the methodology employed in this study including data preprocessing techniques, model selection and training and evaluation metrics. The results will be presented followed by a discussion on the implications of the findings, the limitations of the current approach and potential directions for future research.

## 1.2: Motivation

The motivation for this research stems from the need to explore and evaluate the effectiveness of these advanced deep learning models in predicting stock prices. By comparing the performance of CNN, GRU and LSTM networks, this project aims to contribute to the ongoing efforts to improve financial forecasting methods. Additionally, this research is motivated by the practical implications of accurate stock price predictions. Improved predictive models could enhance investment strategies, reduce financial risks and contribute to more efficient market operations.

Moreover, the rapid advancements in computational power and the availability of vast datasets have made it feasible to apply advanced deep learning models to financial data. This progress presents an exciting opportunity to improve the accuracy of stock price predictions which could lead to better investment strategies and ultimately better financial outcomes for both individuals and institutions.

Furthermore, as financial markets continue to grow in complexity, there is a pressing need for models that can adapt to and predict these changes in real time. The exploration of CNN, GRU and LSTM models in this context is a potential pathway to developing tools that can have real-world applications in finance. The possibility of contributing to this field, offering new insights and potentially improving investment outcomes serves as the primary motivation for this dissertation.

## 1.3: Stakeholders

The outcomes of this research on stock price prediction are of significant interest to a diverse range of stakeholders each with a unique perspective on the financial markets. Identifying and understanding the needs and expectations of these stakeholders is crucial for ensuring the practical relevance and impact of the research.

Individual investors are constantly seeking reliable methods to improve their investment decisions. Institutional investors such as hedge funds and mutual funds manage large portfolios and require robust predictive models to optimize their investment strategies. Financial analysts and market researchers play a key role in advising both individual and institutional investors.

## 1.4: Research Problem

This research seeks to understand whether these advanced deep learning models can provide more accurate and reliable predictions in the context of stock price forecasting. Additionally, the research aims to identify the conditions under which these models perform best and to what extent they can overcome the limitations these models have. The challenge lies not only in applying these models to stock price data but also in fine-tuning them to capture the unique characteristics of financial markets such as sudden spikes or drops, seasonal trends, and the impact of features.

## 1.5: Aims and Objectives

### 1.5.1: Aims

The primary aim of this project is to explore and evaluate the effectiveness of advanced deep learning models in predicting stock prices, with a particular focus on CNN, GRU and LSTM networks. By comparing these models, the project seeks to determine the relative accuracy, robustness and practical applicability of each approach in the context of financial time series forecasting.

### 1.5.2: Objectives

|  |  |
| --- | --- |
| **Objectives** | **Description** |
| 1. Data Collection, EDA and Preprocessing | Gather historical stock price data of Apple and Microsoft from Yahoo! Finance. Conduct EDA to identify correlations, seasonality and other patterns within the collected stock price data. Clean and preprocess the data, including handling missing values, normalizing the data and splitting it into training, validation and test sets. |
| 2. Model Development | Design and develop CNN, GRU and LSTM models for stock price prediction ensuring that each model is properly configured with the appropriate architecture including layers, activation functions and optimization techniques. |
| 3. Model Training | Train the CNN, GRU and LSTM models using the prepared datasets. Validating the models on a separate validation set to monitor performance. |
| 4. Performance Evaluation | Evaluate the performance of each model using a range of metrics including MAE and RMSE values. Conduct a comparative analysis to identify which model provides the most accurate and reliable predictions. |
| 5. Documentation and Reporting | Document the entire process from data collection and preprocessing to model development, training and evaluation.  Compile the findings into a comprehensive dissertation that conducts an in-depth literature review, discusses the implications of the results, the strengths and weaknesses of each model and suggestions for future research. |

Table 1. Objectives with their descriptions

## 1.6: Ethics

In conducting this research, adherence to ethical guidelines is crucial to ensure the integrity of the project and the responsible use of data and methodologies.

The project relies on historical stock price data that is publicly available and does not contain any personal or confidential information. Nevertheless, the data has been handled with care ensuring that any datasets are used in accordance with the terms of use provided by the data sources.

The potential impact of predictive models is acknowledged on financial decision-making. Therefore, the limitations of the models are clearly communicated and the results are presented with appropriate caution to avoid over-reliance on the predictions.

The project does not provide financial advice and it emphasizes that the models developed are for academic purposes. Users of these models are encouraged to exercise their own judgment and consider multiple factors when making investment decisions.

By addressing these ethical considerations, the project ensures that it not only contributes valuable knowledge to the field of stock price prediction but also upholds the highest standards of research integrity and social responsibility.

## 1.7: Summary of Thesis Contributions

This dissertation makes several key contributions to the field of stock price prediction using deep learning models. The primary contributions are as follows:

* The project presents the design and implementation of three base models—CNN, GRU and LSTM networks for stock price prediction. Additionally, it explores enhanced versions of these models with feature engineering to improve predictive accuracy.
* A comprehensive comparative analysis is conducted between the three deep learning models providing insights into their relative performance in predicting stock prices. This analysis identifies the strengths and limitations of each model in capturing complex patterns in financial data.
* The project incorporates technical indicators such as RSI, MACD and Bollinger Bands into the feature-engineered models demonstrating how these indicators can enhance the predictive power of deep learning models in financial forecasting.
* The models are evaluated using metrics including MAE and RSME providing a robust assessment of their accuracy and reliability in stock price prediction.

## 1.8: Thesis Outline

Chapter 1, which is the introduction, sets the base for this dissertation by providing a comprehensive overview of the project. It contains the background and highlights the potential of deep learning models such as CNN, GRU and LSTM in addressing these challenges. The chapter also identifies the motivation behind the project and the primary stakeholders involved. The aims and objectives are laid out and the chapter concludes with a discussion on ethical considerations.

Chapter 2 consists of the Literature review which provides an overview of deep learning with a focus on RNN-based models such as LSTM and GRU as well as CNN and explores how these models have been applied to stock price prediction. The chapter also discusses the role of ethical implications in financial forecasting.

Chapter 3 outlines the approach used to develop stock price prediction models by collecting historical data for Apple and Microsoft from Yahoo! Finance using the yfinance library covering the period from 2017 to 2024. It includes an EDA to identify trends, volatility and correlations through various visualizations. The final section discusses data preprocessing, including normalization, sequence generation and data splitting which ensures the dataset is prepared for effective model training in the subsequent chapters.

Chapter 4 outlines the design and development of six models which is the base and feature-engineered versions of LSTM, CNN, and GRU. The chapter begins by explaining the rationale for selecting these models highlighting their suitability for predicting stock prices. It covers the architecture and implementation details of the base models. The chapter then introduces the feature-engineered models which include technical indicators such as moving averages and RSI.

Chapter 5 outlines the training process for both the base and feature-engineered LSTM, CNN and GRU models. The chapter also includes detailed model summaries that list the parameters involved in each model and their corresponding training processes. By providing both base and feature-engineered models, Chapter 5 sets the stage for the subsequent evaluation of model performance.

Chapter 6 provides an evaluation of the performance of the base and feature-engineered models used for predicting stock prices. It begins with a detailed analysis of the evaluation metrics for the base models of LSTM, CNN and GRU. This is followed by a similar evaluation for the feature-engineered versions of these models. The chapter also includes sections that present future stock price predictions generated by each model highlighting their effectiveness over the next four days. The comparisons between the models are further explored along with discussions on their respective strengths, weaknesses and practical implications in real-world financial forecasting.

Chapter 7 provides a comprehensive summary of the findings from the project. It recaps the key insights gained from the comparative analysis of the base and feature-engineered models highlighting their strengths, weaknesses and practical implications. The chapter also explores the broader significance of the findings for financial forecasting. Additionally, the chapter acknowledges the limitations of the study such as data constraints and model complexity and suggests areas for future research to build on the models developed in this project.

# Chapter 2: Literature Review

## 2.1 Introduction to Stock Price Prediction

### 2.1.1 The Importance of Stock Price Prediction in Financial Markets

Overview of Why Stock Price Prediction is a Critical Area of Study

Stock price prediction has long been a topic of critical importance within the financial industry. Accurate forecasting of stock prices offers the potential for substantial financial rewards providing investors with the insights needed to make informed decisions (Bustos and Pomares-Quimbaya, 2020). The ability to predict future price movements allows for strategic investments and risk management. Therefore, both individual and institutional investors have an interest in developing and employing reliable methods for predicting stock prices.

The significance of stock price prediction lies in its potential to influence a wide array of financial decisions and strategies. For individual investors, the ability to forecast stock prices accurately can mean the difference between substantial gains and significant losses. Moreover, accurate stock price predictions contribute to the overall efficiency of financial markets by facilitating better allocation of resources and ensuring that capital is directed towards the most promising investments (Fama, 1970).

In the context of global financial markets which have become increasingly complex, the need for effective predictive models has only intensified. Events in one part of the world can have almost sudden effects on markets elsewhere creating an environment where price movements can be significant (Venditti and Veronese, 2020). This volatility highlights the need for advanced predictive tools that can capture the rapid changes in market conditions. As a result, stock price prediction has emerged as a focal point for financial analysts and traders driving innovation in both traditional and modern forecasting methodologies.

Historical Context and Evolution of Methods Used in Stock Price Forecasting

In the mid-20th century, the EMH suggested that stock prices fully reflect all available information implying that it is impossible to consistently achieve returns above the market average through prediction (Fama, 1970). Despite this, researchers continued to explore methods that could exploit inefficiencies in the market.

The advent of computing technology in the late 20th and early 21st centuries brought about a revolution in stock price forecasting. Machine learning models such as decision trees and SVM began to be used offering more flexible approaches that could capture non-linear patterns in data (Zhang et al., 1998). However, these models still faced limitations in dealing with time series data where the temporal order of observations is crucial.

The emergence of deep learning techniques marked a significant milestone in the evolution of stock price prediction. Models such as GRU and LSTM networks which are capable of learning long-term dependencies in sequential data have shown great promise in capturing the complex temporal dynamics of financial markets (Hochreiter and Schmidhuber, 1997).

Today, the integration of deep learning with other advanced techniques such as reinforcement learning for dynamic portfolio management represents the cutting edge of stock price prediction (Fischer and Krauss, 2018). These methods continue to push the boundaries of what is possible in financial forecasting offering new opportunities for investors and contributing to more stable and efficient markets.

### 2.1.2 Challenges in Stock Price Prediction

Despite the clear importance of stock price prediction, achieving accurate forecasts is challenging. The primary difficulty lies in the complex, non-linear and dynamic nature of financial markets. Stock prices are influenced by a multitude of factors including macroeconomic indicators, geopolitical events and corporate performance (Ahmed, 2020). This creates a highly volatile environment where predicting future prices with high accuracy remains a significant challenge.

The unpredictability of external factors such as sudden economic downturns and political unrest adds another layer of difficulty to stock price forecasting (Talbi et al., 2021). These events can lead to abrupt price changes that are difficult to predict using traditional models. As such, the quest for more accurate and reliable prediction methods has led researchers to explore new avenues particularly in deep learning which offer the potential to overcome some of these traditional limitations (Chaajer et al., 2021).

Another significant challenge in stock price prediction arises from the inherent noise and randomness in financial markets. This noise can be attributed to a variety of sources such as market manipulation and the impact of high-frequency trading. The rapid execution of trades by algorithms can create short-term price fluctuations that are difficult to model or predict using traditional statistical methods. This high level of noise further complicates the task of identifying genuine signals that indicate long-term market trends (Hendershott et al., 2011).

Furthermore, the adaptive and evolving nature of financial markets presents a challenge for static prediction models. Markets are influenced by a continuous flow of new information including technological advancements and regulatory changes. This dynamic environment requires models that can adapt and learn from new data in real-time. Traditional models, which often assume that market conditions remain constant, may not be capable of capturing these evolving patterns leading to inaccuracies in their predictions (Lo, 2004).

In summary, while stock price prediction holds great importance in financial markets it also presents considerable challenges. The limitations of traditional statistical models have paved the way for more sophisticated techniques including those based on deep learning which are better equipped to handle the complex and dynamic nature of financial data. The following sections of this literature review will explore these methods in greater detail highlighting their advantages and ongoing challenges in the field of stock price prediction.

## 2.2 Emergence of Deep Learning in Financial Forecasting

### 2.2.1 Overview of Deep Learning

Introduction to Deep Learning and Its Relevance to Time Series Forecasting

Deep learning, a subset of machine learning has revolutionized various fields including natural language processing, computer vision and more recently financial forecasting (Sarker, 2021). Unlike traditional machine learning algorithms, deep learning models consist of multiple layers that allow them to learn patterns from large datasets (Shrestha and Mahmood, 2019).

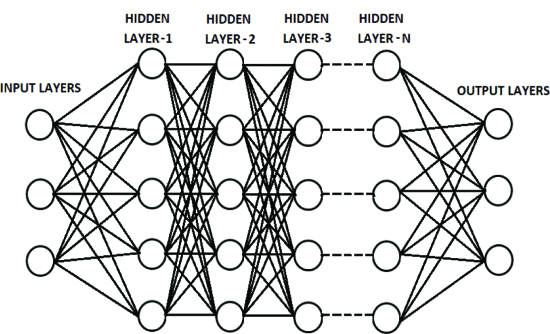


Figure 1. Layers of a Deep Learning Network (Chauhan and Singh, 2018)

This learning process enables deep learning models to capture complex, non-linear relationships in data making them particularly well-suited for time series forecasting where patterns can be complex.

Time series forecasting (including stock price prediction) involves predicting future data points based on historical observations. The temporal dependencies and often non-stationary nature of financial time series make this task challenging. Statistical models such as ARIMA rely heavily on assumptions of linearity and stationarity which would not be feasible to use in financial data (Dong et al., 2020). Deep learning models, on the other hand can automatically learn these temporal dependencies without requiring extensive assumptions about the data's underlying distribution (Ahmed et al., 2023). This flexibility and power to model complex patterns have led to the growing use of deep learning techniques in financial forecasting.

Key Developments in Deep Learning Architectures

A key development is the emergence of attention mechanisms which allow models to focus on the most relevant parts of the input data when making predictions. This is particularly useful in financial forecasting where certain events or periods may have a more significant impact on future prices than others. The integration of attention mechanisms into deep learning architectures has led to state-of-the-art performance in various time series forecasting tasks (Vaswani et al., 2017).

More recently, hybrid models that combine different deep learning architectures have gained popularity in financial forecasting. For example, models that integrate CNNs with GRUs or LSTMs can leverage the strengths of each architecture improving the model's ability to capture both local patterns and long-term dependencies in financial data. These hybrid approaches have been shown to outperform traditional models particularly in complex and volatile market environments (Wen et al., 2019).

Conclusion

In conclusion, the emergence of deep learning in financial forecasting represents a significant advancement in the ability to model complex, non-linear relationships in time series data. The continued development of deep learning architectures particularly those designed to handle the unique challenges of financial time series is likely to play a crucial role in improving the accuracy and reliability of stock price predictions.

## 2.3 RNN-Based Models in Stock Price Prediction

### 2.3.1 Long Short-Term Memory (LSTM) Networks

LSTM networks are a type of RNN specifically designed to address the limitations of standard RNNs. By using memory cells and gates (input, output and forget gates), LSTMs can capture long-term dependencies and avoid the vanishing gradient problem (Hochreiter and Schmidhuber, 1997).

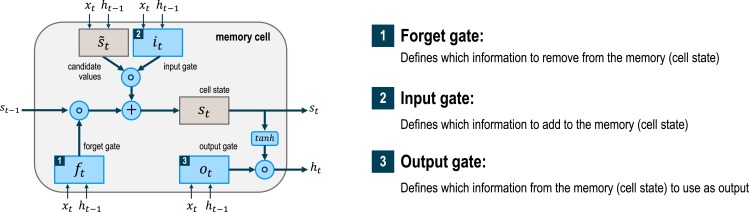


Figure 2. Structure of a LSTM Network (Graves and Pedrycz, 2009)

This makes LSTMs highly effective for tasks that require understanding of long-term temporal dependencies such as stock price prediction.

Performance and Effectiveness

LSTM networks have consistently shown the ability to capture long-term dependencies within financial data. For example, a paper by Fischer and Krauss (2018) applied LSTM networks to predict stock prices of the S&P 500. The LSTM model achieved an RMSE of 0.0206. They found that LSTM models outperformed traditional models such as logistic regression by a significant margin demonstrating the effectiveness of LSTM in providing more accurate predictions over longer periods.

In another paper, Bao et al. (2017) used LSTM networks for stock price prediction on a dataset which consisted of Chinese stock prices and compared the results with ARIMA and standard RNN models. In their comparison of LSTM, ARIMA and RNN models for stock price prediction, the LSTM model had the lowest RMSE demonstrating its effectiveness in handling complex patterns in financial data. The study highlighted that LSTM models were particularly effective in scenarios involving high volatility and complex market dynamics.

A paper by Chen et al. (2020) investigated the use of LSTM networks to predict stock prices under different market conditions including bull and bear markets. The paper found that LSTM networks adapted well to changes in market conditions maintaining robust performance across different scenarios. This robustness was attributed to the model's ability to learn and adapt from long-term dependencies. This adaptability is crucial in financial markets where conditions can change rapidly.

Limitations

One major drawback noted in the literature is the high computational cost associated with training LSTM networks. This was highlighted in a paper by Siami-Namini et al. (2018) where the LSTM model required significantly more time and computational resources compared to simpler models such as ARIMA. This limitation can be a barrier in real-time applications where quick predictions are necessary.

LSTM models are prone to overfitting particularly when trained on limited data or when the model architecture is overly complex. A paper by Nelson et al. (2017) found that while LSTM networks performed well on training data, their performance on unseen test data was less impressive indicating overfitting. The study suggested the use of regularization techniques such as dropout to mitigate this issue.

The "black box" nature of LSTM networks is another significant limitation. Financial professionals often require clear explanations for model predictions but LSTM networks provide little in the way of interpretability. This was discussed in a paper by Kim and Kang (2019), who emphasized the need for methods that could provide more transparent insights into how predictions are made.

Conclusion

LSTM networks offer significant advantages in stock price prediction due to their ability to handle temporal dependencies and adapt to market volatility. However, their application is hindered by computational demands, risk of overfitting and lack of interpretability. Despite these challenges, LSTM networks remain a powerful tool in financial forecasting particularly when paired with appropriate regularization techniques and computational resources.

### 2.3.2 Gated Recurrent Units (GRU)

GRUs are a simplified variant of LSTMs that combine the forget and input gates into a single update gate. This simplification reduces the computational complexity of the model while retaining the ability to capture long-term dependencies in time series data (Dey and Salem, 2017). GRUs are often used as an alternative to LSTMs when a faster model is desired without sacrificing much performance.

Performance and Effectiveness

GRU networks have demonstrated considerable potential in financial forecasting, particularly in stock price prediction. For example, Chen et al. (2023) proposed an improved GRU-based model that integrates data from other stocks within four industries to enhance feature extraction and mitigate overfitting. The study found that their model achieved a RMSE scores of 0.263, 0.306, 0.242, and 0.205 across the four industries which marked a significant improvement over traditional RNN models.

In another paper, Shrestha et al. (2023) explored the combination of GRU with XGBoost to predict stock prices. The GRU-XGBoost hybrid model demonstrated a significant improvement in prediction accuracy, reducing the RMSE by 0.572 compared to the standalone GRU model. This hybrid approach was particularly effective in capturing long-term trends highlighting the potential for combining GRUs with other machine learning techniques.

Wan (2023) conducted a comparative analysis of GRU and LSTM models in predicting stock prices for companies such as Tesla, Ferrari and Walmart. The study found that the GRU model had a slight edge over the LSTM model particularly in predicting Ferrari's stock prices where it achieved an RMSE of 0.027 compared to LSTM's 0.031. This result suggests that GRU might offer advantages in certain market segments or conditions.

Trivedi and Patel (2022) developed a hybrid GRU-LSTM model for predicting HDFC Bank's stock prices.

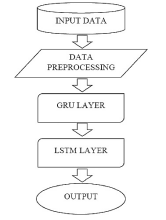


Figure 3. Model Flow of GRU-LSTM Model (Trivedi and Patel, 2022)

Their model outperformed both standalone GRU and LSTM models achieving an RMSE of 0.0524 and a MSE of 0.0027 compared to higher errors in the individual models. This demonstrates the effectiveness of hybrid models in enhancing predictive accuracy.

Limitations

Despite the promising results, GRU models are not without their limitations. One major drawback is their sensitivity to the quality and preprocessing of input data. Wu and Huang (2023) noted that GRU models, when trained on noisy high-frequency trading data, could show volatility in their predictions. This finding underscores the importance of robust data preprocessing techniques to filter out noise and anomalies.

Another limitation is the potential for overfitting particularly in highly volatile market conditions. Pang (2024) observed that GRU models might struggle to maintain accuracy when the market is unstable due to their reliance on historical data. This challenge is compounded by the "black box" nature of GRU networks which similar to other deep learning models offer little interpretability. This lack of transparency can be problematic in financial contexts where understanding the reasoning behind predictions is crucial.

Conclusion

GRU networks have proven to be a powerful tool for stock price prediction, offering a balance of accuracy and computational efficiency. Their performance is competitive with and sometimes superior to LSTM models especially when integrated into hybrid frameworks or combined with other machine learning techniques. However, challenges such as data sensitivity, overfitting and interpretability remain significant hurdles. Addressing these issues through improved data preprocessing and regularization techniques could further enhance the utility of GRU models in financial forecasting.

## 2.4 Convolutional Neural Networks (CNN) in Stock Price Prediction

### 2.4.1 Introduction to CNN

Originally developed for image processing tasks, CNNs have also been adapted for time series forecasting. CNNs can capture local patterns in the data by applying convolutional filters across the time series, detecting short-term trends and fluctuations (Lai et al., 2018). When combined with RNNs, CNNs can model both short-term patterns and long-term dependencies offering a powerful approach to financial time series forecasting.

### 2.4.2 CNN in Financial Data

Performance and Effectiveness

Li and Xie (2022) conducted a study on CNN feature extraction for stock price prediction. Their research demonstrated that a CNN-LSTM hybrid model outperformed a standard LSTM model when predicting prices of stocks with significant market capitalization.

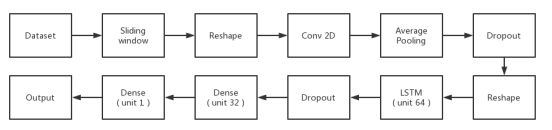


Figure 4. Layout of CNN-LSTM Model (Li and Xie, 2022)

The CNN-LSTM model achieved a RMSE of 0.149 which was a significant improvement over the RMSE of 0.434 obtained by the standalone LSTM model. This result underscores the CNN's effectiveness in capturing short-term dependencies and localized patterns in financial data which were further refined by the LSTM layers for temporal dependencies.

A paper by Wu et al. (2021) proposed a graph-based CNN-LSTM model with leading indicators for stock price prediction. Their model integrated CNN for feature extraction and LSTM for sequence learning, showing superior performance in terms of prediction accuracy compared to traditional methods. The integration of CNN for feature extraction and LSTM for sequence learning combined with leading indicators significantly enhanced the model's ability to predict market movements.

In another paper, Kanwal et al. (2023) introduced a model that integrates multiple CNN layers with LSTM for stock price prediction. This model outperformed other hybrid and standalone models across various performance metrics, including RMSE, MAE, and MAPE. The deep CNN layers facilitated the extraction of complex features from financial time series data leading to more accurate and reliable predictions.

Mehtab and Sen (2020) examined the use of CNN and LSTM-based deep learning models for stock price prediction. They found that a univariate encoder-decoder convolutional LSTM model provided the most accurate forecast achieving a RMSE of 0.0369. On the other hand, the univariate CNN model which utilized one week of historical data was the fastest in terms of execution speed and a performs greater with a RMSE of 0.0349. This study highlights the trade-off between accuracy and computational efficiency when using CNNs for financial forecasting.

Limitations

Despite their effectiveness, CNN models also face several limitations when applied to financial data. Zheng (2023) highlighted the challenges in integrating CNNs with sentiment analysis for stock price forecasting. While the CNN-BiLSTM model with attention mechanisms showed high accuracy, the complexity of combining multiple layers can increase the model's computational demands and exposure to overfitting.

Another limitation, noted by Zhang (2023) is the model's sensitivity to the amount of historical data used. In their study, CNN models provided better accuracy with short-term data but struggled with long-term trend predictions especially when compared to LSTM models which are inherently better at capturing temporal dependencies.

Conclusion

CNNs have demonstrated significant promise in the field of stock price prediction, particularly when integrated with other models such as LSTM or GRU. These hybrid models leverage CNN's ability to extract complex features from time series data leading to improved accuracy in financial forecasting. However, challenges remain in terms of computational complexity and model sensitivity to data. Future research should focus on optimizing CNN architectures for financial data and exploring more efficient ways to integrate external data sources to improve model robustness and interpretability.

## 2.5 Comparative Studies of CNN, GRU and LSTM in Financial Forecasting

In recent years, the application of deep learning models in financial forecasting has gained significant attention, particularly with the use of CNN, GRU and LSTM networks. Each of these models has its own strengths and limitations and comparative studies have been conducted to evaluate their effectiveness in predicting stock prices.

### 2.5.1 Comparative Review

A paper by Rajhans et al. (2024) conducted a study comparing the performance of CNN, GRU and LSTM models in predicting stock prices using financial news sentiment. The study found that LSTM and GRU outperformed CNN in terms of accuracy as measured by MSE. However, CNN showed potential in handling specific patterns related to sentiment data suggesting that its performance could be enhanced when used in hybrid models.

In another paper, Paygude et al. (2023) undertook a parameter-based comparative study of deep learning algorithms including CNN, GRU and LSTM for stock price prediction. Their study found that LSTM provided the highest prediction accuracy with an RMSE of 0.029 and an MAE of 0.021. GRU followed closely with an RMSE of 0.032 and MAE of 0.024 while CNN achieved an RMSE of 0.035 and MAE of 0.026. The study highlighted that while LSTM is the most accurate GRU offers a better trade-off between accuracy and computational efficiency and CNN excels in feature extraction making it suitable for hybrid models.

### 2.5.2 Model Effectiveness and Real-World Applications

In real-world applications, the effectiveness of CNN, GRU and LSTM models in financial forecasting is demonstrated through various studies each showcasing the strengths of these models in different contexts.

Sulistio et al. (2023) explored the use of a hybrid CNN-LSTM-GRU model for predicting energy sector stock prices. The hybrid model outperformed other methods achieving an RMSE of 0.076 in predicting stock closing prices. This study underscores the potential of hybrid models that combine the strengths of different deep learning architectures particularly in complex sectors such as energy where both short-term and long-term trends need to be captured.

Chen (2023) proposed a CNN-GRU-attention model for stock price prediction which outperformed other models in performance metrics such as RMSE and MAE.

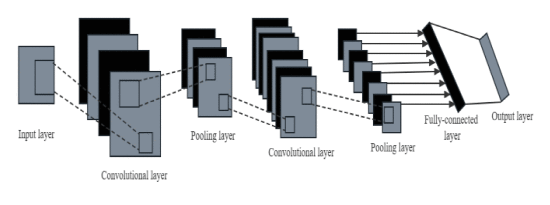


Figure 5. Structure of the CNN Model (Chen, 2023)

The integration of attention mechanisms with CNN and GRU significantly improved the model's ability to focus on the most relevant features in the data thereby enhancing prediction accuracy. This study demonstrates the effectiveness of combining different neural network architectures to capture both spatial and temporal dependencies in financial data.

Conclusion

These studies collectively show that while LSTM models often provide the highest accuracy for long-term predictions, GRU models offer a good balance between accuracy and computational efficiency and CNN models are highly effective in feature extraction particularly when used in hybrid models. The choice of model should be tailored to the specific needs of the financial forecasting task with hybrid models offering a promising approach for improving accuracy and robustness in complex financial environments.

## 2.6 Ethical Aspects in Financial Forecasting

The use of predictive models in financial forecasting particularly in stock price prediction raises significant ethical concerns. These concerns center on the impact of these models on market behavior, the potential for misuse and the broader implications for fairness, transparency and accountability in financial markets.

### 2.6.1 Ethical Implications of Predictive Models

One of the primary ethical considerations is the potential for predictive models to influence market behavior. According to Shukla et al. (2024), the integration of machine learning models with manual forecasting techniques, while enhancing accuracy, also raises concerns about the amplification of biases inherent in the data. This can lead to feedback loops where predictions based on biased data perpetuate existing market inequalities.

The ethical implications of data privacy and transparency are also critical in the use of predictive models in finance. Tuboalabo et al. (2024) emphasize the importance of strict data privacy measures and transparent practices in predictive modeling. Without such measures, there is a risk of violating individual privacy rights and eroding trust within the financial industry. The study highlights that ensuring compliance with data protection regulations is essential for maintaining the integrity of financial forecasting practices.

Another significant ethical concern is the consistency and fairness of predictive models. Yang et al. (2023) argue that ensuring the consistency of these models is crucial for building user trust and avoiding performance degradation. Ethical considerations must include measures to prevent alterations that could undermine the reliability of financial predictions.

Lastly, the effectiveness of including uncertainty considerations into neural network-based financial forecasting models is discussed by Maeda et al. (2019). The study highlights that addressing uncertainty in predictions is critical for improving decision-making processes in investments. Ethical implications include the need for transparency in how these uncertainties are communicated to stakeholders ensuring that predictions are used responsibly

Conclusion

In conclusion, the ethical use of predictive models in financial forecasting involves careful consideration of data privacy, transparency, model interpretability and fairness. Ensuring that these models are developed and deployed with these ethical principles in mind is essential for maintaining the integrity of financial markets. Robust regulatory frameworks and ongoing ethical reviews are necessary to address the challenges posed by these advanced technologies in finance.

## 2.7 Summary of Literature and Gaps Identified

### 2.7.1 Summary of Literature

The reviewed literature underscores the rapid advancements and growing reliance on deep learning models in financial forecasting particularly in stock price prediction. The effectiveness of models such as CNN, GRU and LSTM has been consistently demonstrated across various studies. LSTM models, known for their ability to capture long-term dependencies, have shown greater performance in scenarios where modelling temporal sequences is crucial. GRU models, while similar to LSTM, offer a more computationally efficient alternative making them attractive in environments where resource constraints are a concern. CNN models, traditionally used for image processing have been adapted for financial data to effectively capture localized patterns often leading to improved short-term predictions.

However, despite these advancements several gaps in the literature have been identified. One significant gap is the lack of comprehensive studies that compare these models across different market conditions. Most existing research focuses on specific markets or stocks limiting the use of the findings.

Another gap is the insufficient exploration of the interpretability of these deep learning models. As these models are increasingly used for high-stakes financial decisions there is a growing need for methods that can explain their predictions in a way that is understandable to financial professionals.

### 2.7.2 Relevance to Current Study

This dissertation seeks to address the identified gaps by conducting a comprehensive comparative analysis of CNN, GRU and LSTM models in the context of stock price prediction. By evaluating these models in diverse scenarios, this study aims to provide more generalizable insights into their relative effectiveness and the circumstances under which each model excels.

In addressing the gap related to model interpretability, the dissertation will explore techniques for enhancing the transparency of predictions made by these deep learning models. By incorporating these methods, the aims are to make these models more accessible and trustworthy as well as adhering to ethical guidelines.

The significance of this project lies in its potential to enhance the accuracy and reliability of stock price prediction models, thereby contributing to more informed decision-making in financial markets. By addressing key gaps in the literature, this dissertation will provide valuable insights that can help shape the future development and application of deep learning models in financial forecasting.

# Chapter 3: Methodology and Data Preparation

## 3.1 Introduction

In this chapter, the methodology and data preparation processes used in this dissertation which are crucial to the development of accurate and reliable stock price prediction models will be explained. The chapter begins with a comprehensive overview of the methodology adopted highlighting the rationale behind the selection of both base models and feature-engineered models for predicting stock prices.

The decision to utilize base models specifically CNN, GRU and LSTM stems from their proven capabilities in handling time series data particularly in capturing temporal dependencies and non-linear patterns inherent in financial datasets. These models serve as the foundational approaches for understanding the stock price dynamics of Apple and Microsoft.

However, to enhance predictive accuracy, the project also incorporates feature-engineered models. These models integrate additional technical indicators such as RSI, MACD and Bollinger Bands which are commonly used in financial analysis. The inclusion of these indicators aims to provide the models with richer information potentially leading to more precise predictions.

## 3.2 Data Collection and Exploration

The section is structured to first introduce the data collection process, followed by EDA and finally the data preprocessing steps that ensure the models are trained on relevant and normalized data. This approach lays a solid foundation for the subsequent chapters which details the model design, training and evaluation processes.

### 3.2.1 Data Sources and Description

For this project, historical stock price data for Apple (AAPL) and Microsoft (MSFT) were sourced using the ‘yfinance’ Python library, a widely recognized tool for accessing financial data directly from Yahoo! Finance. yfinance retrieved stocks of Apple and Microsoft from the NASDAQ stock exchange where both companies are listed and traded. The datasets include daily stock prices over a period spanning from January 1st 2017 to January 1st 2024 providing a robust time series for model training and evaluation. The total rows of both datasets are 1760 rows.

The data collected contains several key financial metrics crucial for stock price prediction including:

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| Date | The specific day the trading activity occurred. |
| Open | The price at which a stock started trading during the regular trading hours. |
| High | The highest price reached by a stock during the trading day. |
| Low | The lowest price recorded during the trading day. |
| Close | The price of the stock at the end of the trading day. |
| Adjusted Close | The closing price of the stock after adjustments for all applicable splits and dividend distributions. |
| Volume | The number of shares traded during the day. |

Table 2. Descriptions of the dataset columns

These metrics were gathered for each trading day over the seven-year period resulting in a comprehensive dataset that reflects various market conditions such as periods of high volatility.

The choice of Yahoo! Finance as the data source was driven by its reliability, extensive historical data coverage and ease of integration with Python-based data analysis workflows. The yfinance library was particularly useful in automating the data retrieval process ensuring that the data was both up-to-date and accurately reflected the market conditions during the specified period.

This data forms the foundation for developing and testing the predictive models discussed in subsequent sections allowing for a thorough analysis of the effectiveness of various deep learning approaches in forecasting stock prices.

### 3.2.2 Exploratory Data Analysis (EDA)

EDA plays a critical role in understanding the underlying patterns and characteristics of the dataset used in this study. In the context of stock price prediction EDA helps in identifying trends, seasonality and correlations within the data which are essential for building accurate predictive models. This section details the EDA process conducted on the historical stock price data of Apple and Microsoft covering key insights such as price movements and volume fluctuations.

Data Types

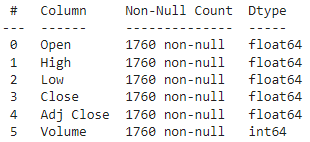


Figure 6. Data types

The data types of these columns are critical for understanding the nature of the data and for performing appropriate statistical analyses and modelling. Open, High, Low, Close and Adjusted Close are all of the data type float64 indicating that they contain continuous numerical values with decimal points. The Volume column has the data type int64 representing the number of shares traded during the day. As an integer type it signifies that the volume is always a whole number which is appropriate given that shares cannot be traded in fractional amounts.

Stock Closing Prices Over Time

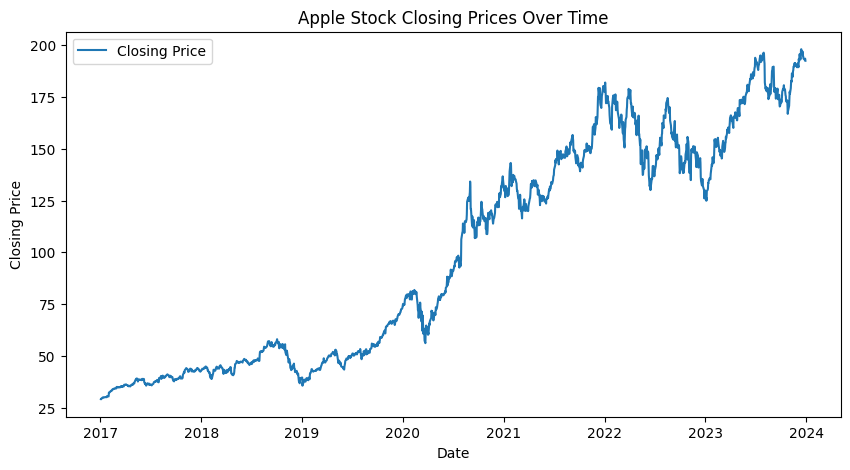


Figure 7. Apple stock closing prices over time

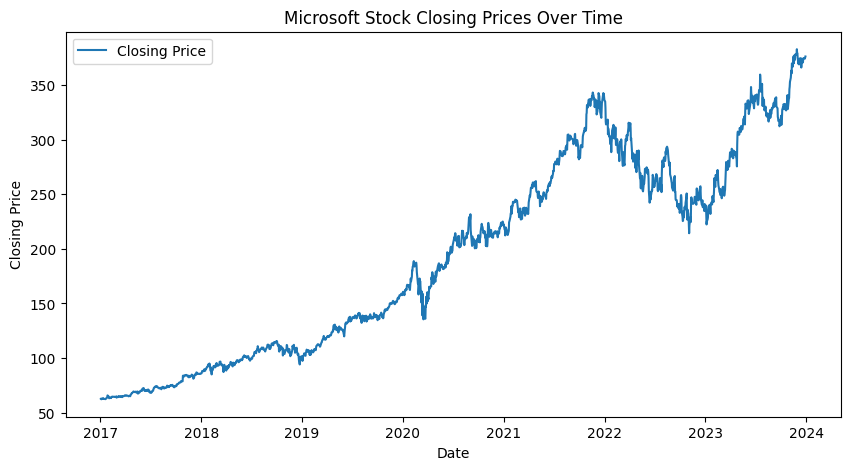


Figure 8. Microsoft stock closing prices over time

Figures 7 and 8 above display the closing stock prices of Apple and Microsoft over the period from January 2017 to January 2024. Both graphs provide a clear visual representation of the trends in the stock prices of these two major technology companies over time.

Both companies display similar growth patterns reflecting broader market trends and the success of the tech sector over the past seven years. The fluctuations in stock prices for both Apple and Microsoft are indicative of market responses to various economic conditions and global events such as the COVID-19 pandemic but the long-term upward trend underscores their dominant positions in the market.

These visualizations are essential in understanding the historical performance of these stocks and set the stage for predictive modeling in this research. The upward trend in both company’s stock prices suggests a growth phase making the prediction task more challenging due to the inherent volatility but also potentially rewarding given the stakes involved in accurately forecasting these prices.

Stock Volatility Over Time

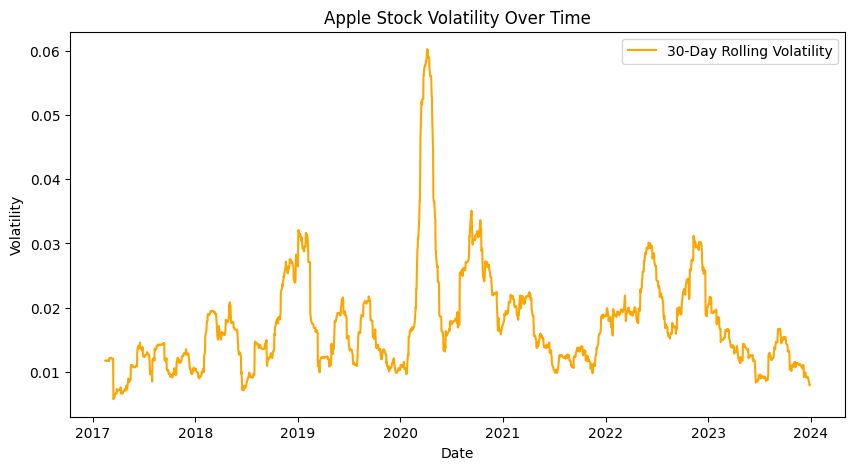


Figure 9. Apple Stock Volatility Over Time

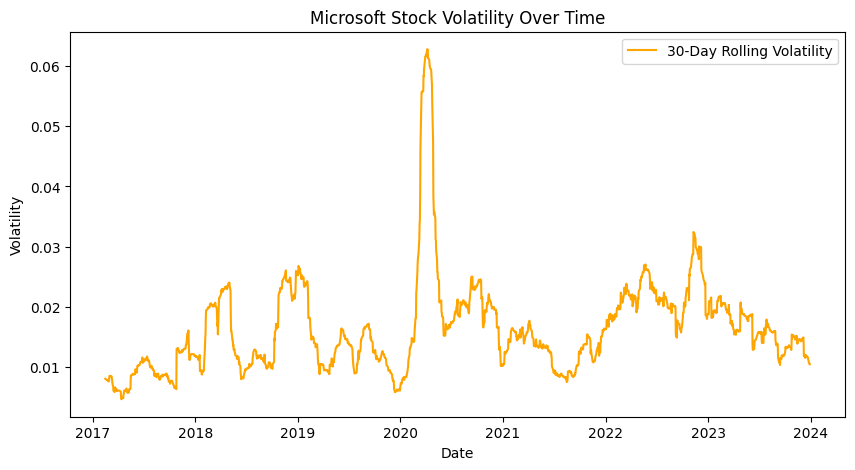


Figure 10. Microsoft Stock Volatility Over Time

Figures 9 and 10 depict the 30-day rolling volatility of Apple and Microsoft stocks from 2017 to 2024. Volatility measures the degree of variation in a stock's price over time and is often used as an indicator of risk in the financial markets.

To summarize, the volatility analysis of Apple and Microsoft stocks highlights the significant impact of the COVID-19 pandemic on the financial markets, causing a sharp increase in volatility during 2020. The subsequent years show a gradual normalization although both stocks continue to experience periodic fluctuations reflecting ongoing market dynamics. Understanding these volatility patterns is crucial as it provides insights into the potential risks and uncertainties associated with these stocks over time.

Daily Return of Stocks

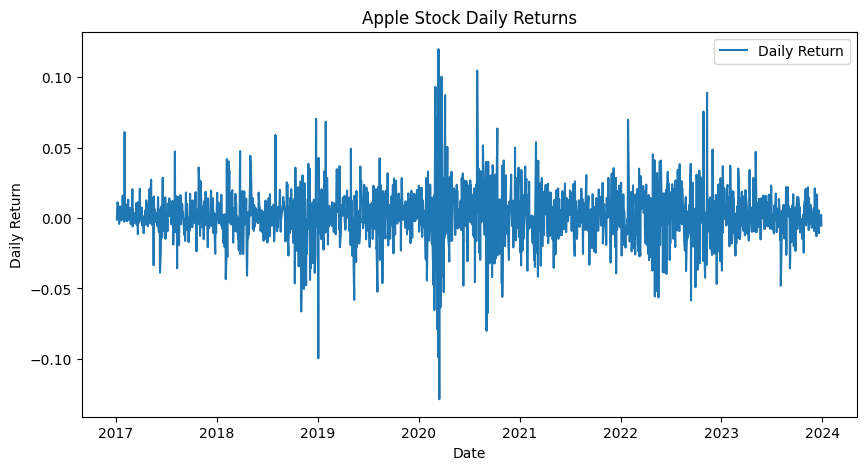


Figure 11. Apple Stock Daily Returns

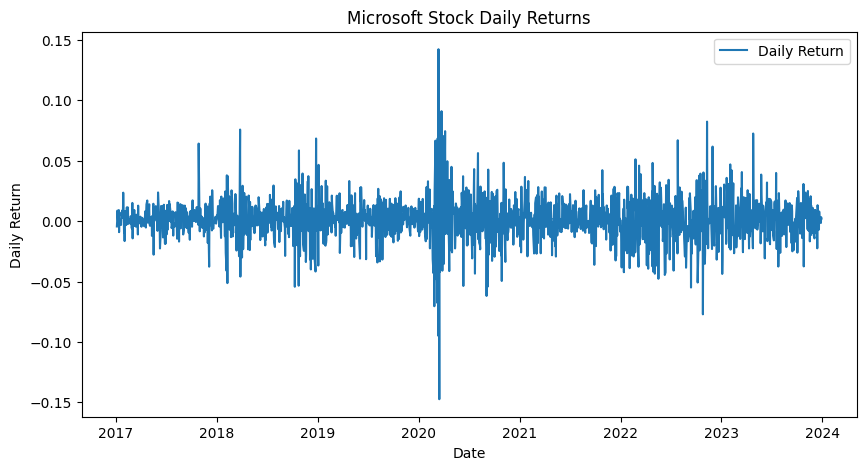


Figure 12. Microsoft Stock Daily Returns

Figures 11 and 12 represents the daily returns of Apple and Microsoft from 2017 to 2024 which showcases the volatility in the stock's performance on a day-to-day basis. The daily returns are calculated as the percentage change in the stock price from one day to the next.

There is a significant increase in daily return volatility around the year 2020 likely reflecting the market's reaction to the COVID-19 pandemic. During this period, daily returns are highly variable indicating both large positive and negative movements in the stock price. Following the heightened volatility of 2020, the daily returns gradually stabilize although fluctuations remain evident. This stabilization suggests that while the market has calmed somewhat, investors are still cautious and the stock remains sensitive to external factors.

The magnitude of daily returns for Microsoft seems slightly more volatile compared to Apple particularly during the sharp market reactions of 2020. This might indicate that Microsoft's stock was more sensitive to market dynamics during this period.

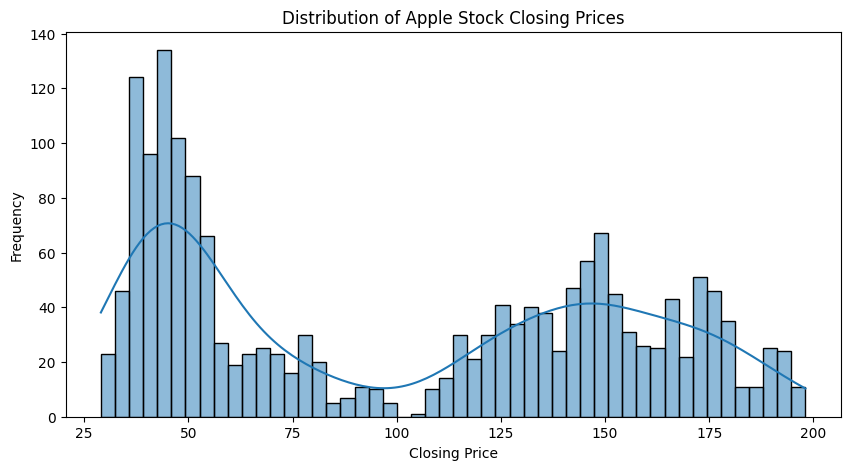


Figure 13. Distribution of Apple Stock Closing Prices

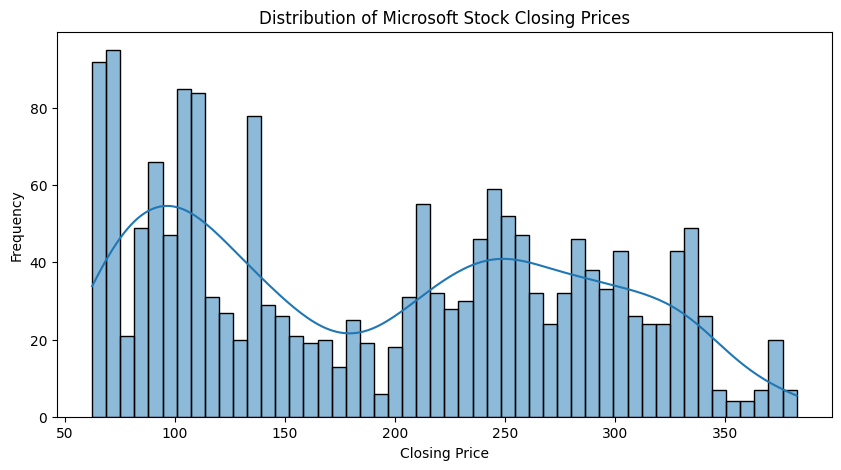


Figure 14. Distribution of Microsoft Stock Closing Prices

Figure 14 and 15 illustrate the distribution of closing prices for Apple and Microsoft stocks over the period from 2017 to 2024. These distributions offer insights into the range and frequency of stock prices during this period highlighting the performance trends of these companies in the market.

The histogram for Apple's closing prices shows a bimodal distribution. The first peak occurs in the price range of $25 to $75 which corresponds to the earlier years within the data period when Apple's stock price was relatively lower. The second peak is seen between $125 and $175 representing more recent years as Apple's stock price increased significantly. The distribution's shape suggests a clear upward trend in Apple's stock price over time with a noticeable increase in the frequency of higher closing prices in recent years.

The distribution of Microsoft's closing prices also shows a bimodal pattern with the first peak between $50 and $100 representing the lower price range during the earlier part. The second peak occurs between $200 and $250 indicating a significant increase in Microsoft's stock value over the years. Additionally, there is a gradual decline in frequency as the closing prices increase particularly above $250 suggesting that Microsoft's stock price has been consistently rising with fewer occurrences of lower prices as the company has grown.

### 3.2.3 Data Preprocessing

Data preprocessing is a crucial step in the development of any predictive model particularly in the context of stock price prediction. This process involves transforming raw data into a structured format suitable for input into deep learning models. The quality of the data fed into the model significantly impacts the accuracy and reliability of the predictions. In this section, we outline the steps taken to prepare the Apple and Microsoft stock price data for modeling.

Data Preprocessing of Base Models

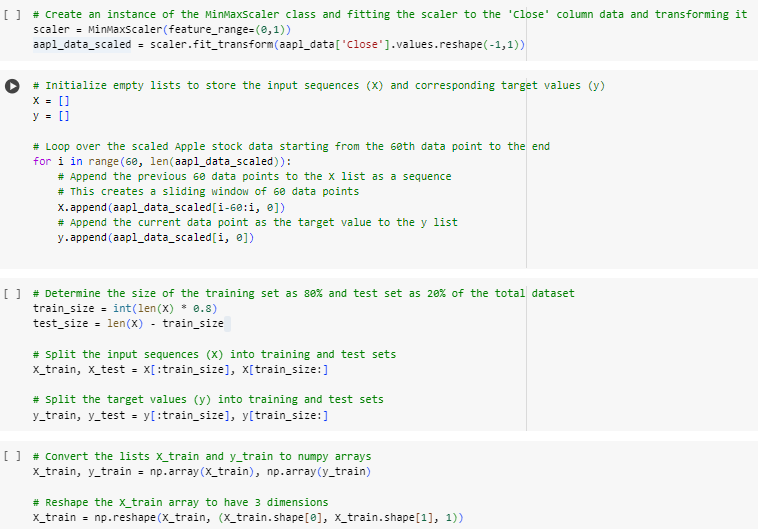


Figure 15. Data Preprocessing of Base Model code

The preprocessing begins with the normalization of the data. Since stock prices can have vastly different scales, it is crucial to bring them into a uniform range to improve the model’s learning process. For this purpose, the MinMaxScaler class from the sklearn.preprocessing module is used to scale the 'Close' price values of Apple stock between 0 and 1. This step is particularly important for deep learning models as it helps in speeding up the convergence during training and avoids issues related to scale differences.

Following normalization, we prepare the data for model training by creating sequences of past stock prices to predict future prices. Specifically, a sliding window approach is employed where each sequence consists of 60 consecutive data points which is the previous 60 days of closing prices. This sequence serves as the input (X) to the model while the closing price of the next day serves as the target output (y). This method captures the temporal dependencies in stock price movements which is essential for time series forecasting.

The data is then split into training and testing sets with 80% of the data allocated for training and the remaining 20% reserved for testing. This split allows the model to learn from the majority of the data while ensuring that its performance can be evaluated on unseen data. The input sequences are reshaped to the required format for deep learning models which expects a 3-dimensional array as input.

This preparation of the data aims to improve the model's ability to capture patterns in the stock price movements and make accurate predictions.

Data Preprocessing of LSTM Feature-Engineered Model

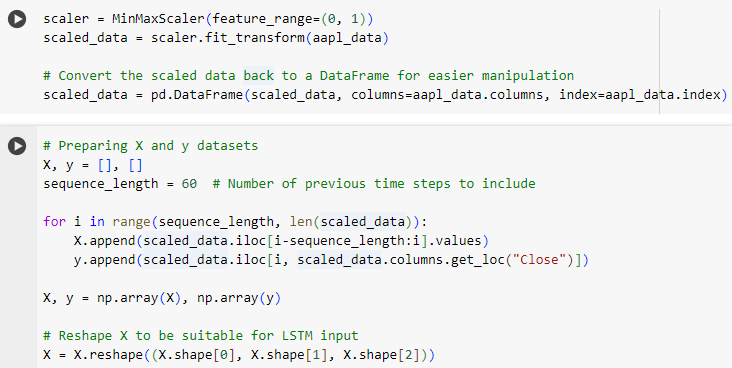


Figure 16. Data Preprocessing of LSTM Feature-Engineered Model code

The first step in preprocessing the data for the LSTM feature-engineered model is scaling all the features into a range between 0 and 1 using the MinMaxScaler. This scaling process ensures that the model treats all features with equal importance during training preventing features with larger numerical ranges from dominating the learning process.

Once the data is scaled, it is used to generate sequences of 60 previous time steps as input (X) and the corresponding target values (y). This approach allows the model to learn from historical patterns in the data. For each point in time, the model looks back over the past 60-time steps to predict the next value in the series. This process effectively converts the data into a format that LSTM networks can use to understand temporal dependencies.

Finally, the input data is reshaped to have three dimensions. This reshaping is critical because LSTM networks require input data in a 3D format which allows them to process each time step in the sequence individually while maintaining the context of the entire sequence.

Data Preprocessing of CNN and GRU Feature-Engineered Model

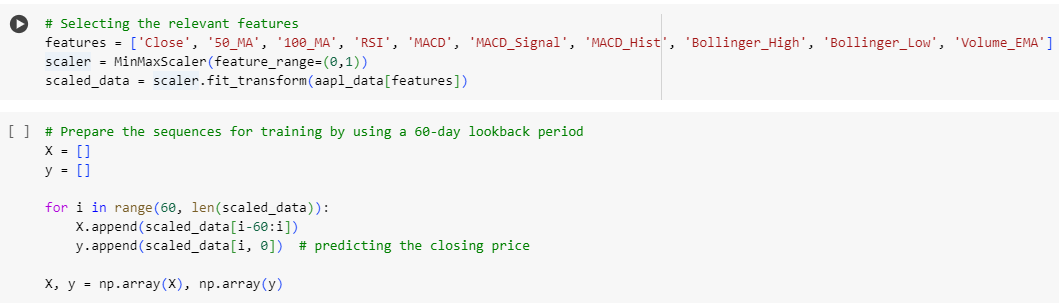


Figure 17. Data Preprocessing of CNN and GRU Feature-Engineered Model code

For the CNN and GRU feature-engineered models, the first step involves selecting the most relevant features from the dataset which includes technical indicators that will be introduced in Chapter 4 of this project. These features are crucial as they provide a comprehensive view of the stock's price movements, trends and market momentum which are essential for accurate predictions.

Once the relevant features are selected, the data is scaled using the MinMaxScaler to bring all values within the range of 0 to 1. This scaling is important to ensure that all features contribute equally during model training preventing any feature with a larger numerical range from disproportionately influencing the model. The scaled data is then used to prepare sequences for training.

To prepare the input data (X) and target values (y) for the CNN and GRU models, a 60-day lookback period is used. This means that the model is trained on the past 60 days of data to predict the closing price on the 61st day. The CNN and GRU models require input data in a format where each input sequence contains multiple features over the 60-day period allowing the model to learn patterns and relationships among the different technical indicators over time. Finally, the prepared sequences are converted into numpy arrays to facilitate efficient computation during model training.

# Chapter 4: Model Design and Development

## 4.1 Introduction

This chapter explores into the design and development of six distinct models used for predicting stock prices. The rationale behind selecting these models stems from the need to explore both standard and enhanced approaches in stock price forecasting. CNN, GRU and LSTM are chosen for their established effectiveness in handling time series data. These models form the foundation of the predictive framework.

In addition to the base models, feature-engineered versions of CNN, GRU and LSTM are also developed. These enhanced models incorporate additional technical indicators and features aiming to improve prediction accuracy by providing the models with more contextual information. This chapter outlines the design of these models highlighting the methodologies employed to integrate feature engineering and the architectural choices that underpin their development.

## 4.2 Base Model Design

### 4.2.1 LSTM Base Model Design

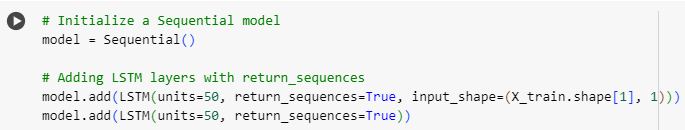


Figure 18. Architecture of the LSTM base model

The design of the LSTM base model is rooted in its ability to capture and utilize temporal dependencies within time series data which is critical for accurately predicting stock prices. The architecture begins with the initialization of a Sequential model using Keras making it suitable for the construction of deep learning models.

The model comprises two LSTM layers each with 50 units. These layers are designed to handle the sequential nature of the stock price data. The return\_sequences=True parameter is applied to both LSTM layers ensuring that each LSTM layer returns the full sequence of outputs. This configuration is crucial for allowing subsequent layers to have access to the entire sequence of the data.

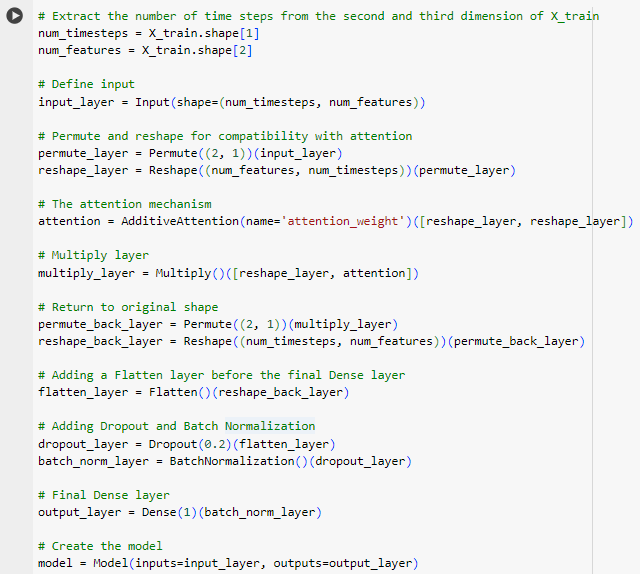


Figure 19. Layers of the Base LSTM Model

In addition to the LSTM layers, an attention mechanism is introduced to enhance the model's focus on specific parts of the sequence that are more critical for the prediction task. This is implemented using an AdditiveAttention layer which helps the model weigh different time steps differently allowing it to concentrate on the most relevant information in the sequence.

The architecture also includes a Flatten layer to reduce the dimensionality of the output before feeding it into the Dense layer which serves as the output layer. To improve the model's generalization capabilities and prevent overfitting, a Dropout layer with a 0.2 dropout rate is added followed by a BatchNormalization layer. The output layer, a Dense layer with a single unit, is configured to produce a single scalar value representing the predicted stock price.

The model is compiled using the Adam optimizer known for its efficiency in handling large-scale data and its capability of adaptive learning rate optimization. The loss function chosen is MSE which is a standard choice for regression problems such as stock price prediction.

### 4.2.2 CNN Base Model Design

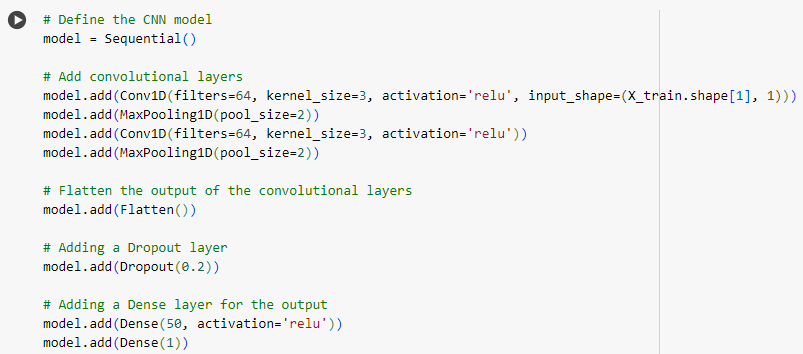


Figure 20. Architecture and Layers of the Base CNN Model

The CNN model was chosen for this stock price prediction task due to its ability to automatically learn spatial hierarchies of features from input data. CNNs are particularly effective in capturing patterns within data such as trends in stock prices over time.

The model begins with two convolutional layers each with 64 filters and a kernel size of 3. These layers are responsible for detecting patterns such as trends and fluctuations in the stock prices. The relu activation function is applied to introduce non-linearity into the model, enabling it to learn more complex patterns.

After each convolutional layer, a MaxPooling layer is included with a pool size of 2. MaxPooling reduces the dimensionality of the data by down-sampling which helps in reducing computational complexity and preventing overfitting by retaining only the most important features.The output of the final MaxPooling layer is flattened transforming the 2D matrix into a 1D vector which can then be fed into the fully connected layers. This step is crucial for transitioning from feature extraction to classification or regression tasks.

A Dropout layer is added with a rate of 0.2 to prevent overfitting by randomly setting 20% of the input units to 0 at each update during training. This encourages the model to generalize better by not relying too heavily on specific neurons.

Finally, the model includes two Dense layers. The first Dense layer consists of 50 units with a relu activation function further processing the features extracted by the convolutional layers. The output Dense layer has a single unit which is used to predict the stock price.

### 4.2.3 GRU Base Model Design

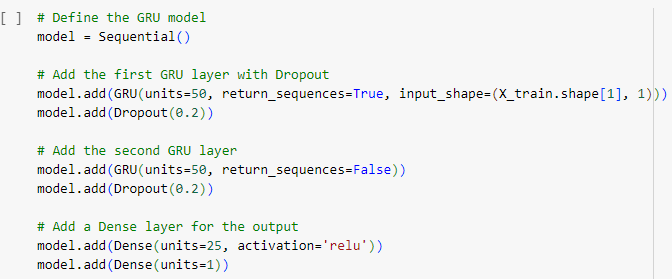


Figure 21. Architecture and Layers of the Base GRU Model

The GRU model was selected for stock price prediction due to its efficiency and capability in handling sequential data similar to the LSTM but with a simpler architecture. GRU is particularly advantageous in scenarios where the dataset is large and training time needs to be minimized while still capturing long-term dependencies in the data.

The model starts with two GRU layers. The first GRU layer consists of 50 units and is configured to return sequences which means that it will output the full sequence of hidden states for each input sequence rather than just the last hidden state. This setup allows the model to preserve information across time steps which is crucial for predicting time series data such as stock prices. The second GRU layer also with 50 units does not return sequences meaning it outputs only the last hidden state. This reduces the data dimensionality and prepares it for the subsequent Dense layers.

To prevent overfitting, Dropout layers are added after each GRU layer with a dropout rate of 0.2. Dropout randomly sets 20% of the input units to zero at each update during training time which helps in regularizing the model.

The model then includes a Dense layer with 25 units and a relu activation function. This layer serves to further process the output from the GRU layers extracting important features before making the final prediction. The final output layer is a Dense layer with a single unit which outputs the predicted stock price. This layer does not use any activation function as it is a regression task aiming to predict a continuous value.

## 4.3 Feature Engineering and Model Design

### 4.3.1 Introduction to Feature Engineering

Feature engineering plays a crucial role in enhancing the predictive power of deep learning models by creating new input features from existing data. In the context of stock price prediction, feature engineering involves the generation of technical indicators that summarize important aspects of market behavior.

For the feature-engineered models developed in this project, several technical indicators were created to help the models better understand market dynamics. These indicators are well-established in financial analysis and are widely used by traders and analysts to inform investment decisions.

The key features introduced through feature engineering include:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| 50-day Moving Average | The 50-day moving average is a commonly used indicator that smoothens price data by calculating the average closing price over the last 50 days. It helps in identifying the underlying trend by filtering out short-term fluctuations. |
| 100-day Moving Average | Similar to the 50-day moving average, the 100-day moving average covers a longer period and provides a broader perspective on the trend. This indicator is particularly useful for understanding long-term trends and potential support or resistance levels. |
| RSI | The RSI measures the speed and change of price movements. It ranges from 0 to 100 and is typically used to identify overbought or oversold conditions in a stock. An RSI above 70 indicates that a stock may be overbought while an RSI below 30 suggests it may be oversold. |
| Volume EMA | The Volume EMA is applied to the trading volume providing a smoothed view of volume trends (Chen, 2019). It helps in understanding whether the market activity is increasing or decreasing over time which can be a precursor to price movements. |
| MACD | MACD is a trend-following momentum indicator that shows the relationship between two moving averages of a stock’s price. The MACD line subtracts the 26-day EMA from the 12-day EMA. The MACD signal line is the 9-day EMA of the MACD line and the MACD histogram represents the difference between the MACD line and the signal line 19 (Khandelwal, 2024). These components help in identifying potential buy or sell signals. |
| Bollinger Bands | Bollinger Bands consist of three lines: a simple moving average (middle band) and an upper and lower band that are typically two standard deviations away from the middle band (Stokes, 2024). These bands expand based on market volatility helping to identify potential overbought or oversold conditions. |

Table 3. Description of technical indicators used in the feature-engineered models

A Python package for technical analysis known as “ta” was utilized to efficiently compute the technical indicators from Table 3 required for feature engineering. This library provides a comprehensive suite of tools that calculate the technical indicators. This enabled the development of robust feature-engineered models that could capture the underlying patterns in the stock market data.

### 4.3.2 Feature-Engineered LSTM Model Design

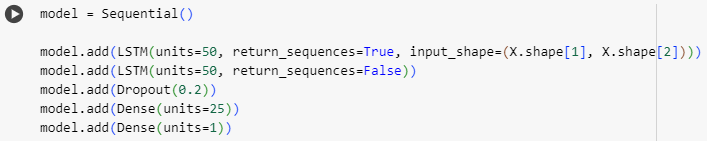


Figure 22. Architecture and Layers of the Feature-Engineered LSTM Model

The feature-engineered LSTM model is designed to incorporate a richer set of technical indicators as input features allowing the model to better understand and predict stock price movements. The architecture follows the same foundational structure as the base LSTM model but with additional features such as Moving Averages, RSI, MACD, Bollinger Bands and Volume EMA.

The model architecture begins with two LSTM layers. The first LSTM layer contains 50 units and is configured to return sequences to ensure the model captures the temporal dependencies between consecutive data points. A second LSTM layer also with 50 units processes the outputs from the first LSTM layer without returning sequences effectively condensing the learned information. A Dropout layer with a dropout rate of 20% is added to prevent overfitting during the training process. The model then includes a Dense layer with 25 units to introduce non-linearity followed by another Dense layer with a single unit which outputs the predicted stock price.

The technical indicators serve as additional features in the input data allowing the LSTM model to learn from a more comprehensive set of market conditions. This feature-engineered model is expected to perform better by capturing not only past stock prices but also broader market trends represented by the engineered features.

### 4.3.3 Feature-Engineered CNN Model Design

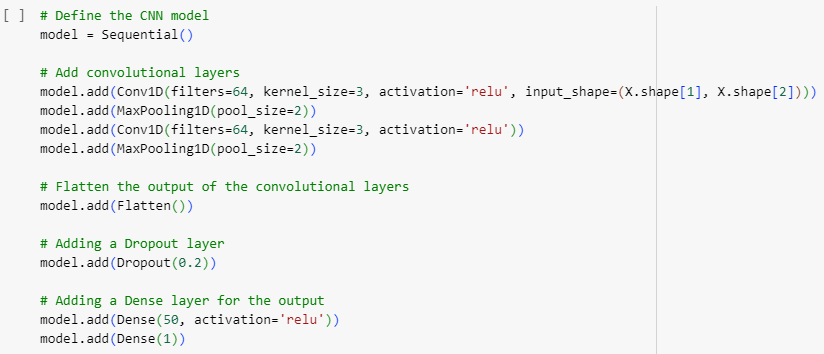


Figure 23. Architecture and Layers of the Feature-Engineered CNN Model

The feature-engineered CNN model is designed to utilize technical indicators such as Moving Averages, RSI, MACD, Bollinger Bands, and Volume EMA to predict stock prices. By using a convolutional architecture, the model can efficiently capture patterns and trends in the time series data as CNNs are adept at detecting local dependencies and patterns within sequential data.

The CNN model begins with a 1-dimensional convolutional layer which applies 64 filters of size 3 to extract features from the input data. This layer is followed by a max-pooling layer with a pool size of 2 which down samples the feature maps and reduces the computational complexity. Another 1D convolutional layer with 64 filters is applied followed by a second max-pooling layer to further extract high-level features from the time series data.

After the convolutional and pooling layers, the output is flattened into a 1D vector which is then passed through a dropout layer with a dropout rate of 20% to prevent overfitting. A fully connected layer with 50 units and relu activation is added to introduce non-linearity followed by another Dense layer with a single unit that outputs the predicted stock price. This architecture leverages the convolutional layers to detect patterns in the feature-engineered data providing a robust framework for stock price prediction.

### 4.3.4 Feature-Engineered GRU Model Design

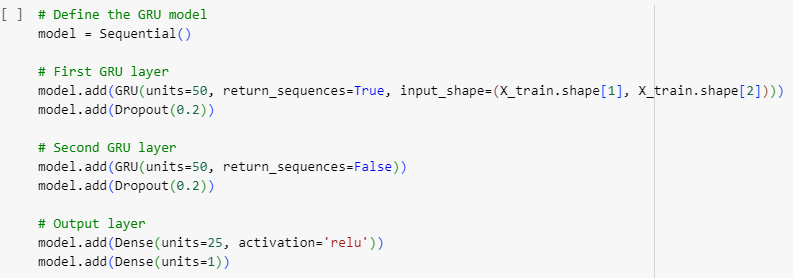


Figure 24. Architecture and Layers of the Feature-Engineered GRU Model

The feature-engineered GRU model is designed to capture the temporal dependencies in stock price data while utilizing a range of technical indicators to enhance prediction accuracy. GRU is particularly suited for time-series data as it addresses the vanishing gradient problem and allows the model to learn long-term dependencies.

This GRU model is built using two GRU layers. The first GRU layer consists of 50 units and uses the return\_sequences parameter set to True to ensure that the entire sequence is passed to the next layer. This layer is followed by a dropout layer with a rate of 0.2 which helps in preventing overfitting by randomly ignoring some units during training. The second GRU layer also contains 50 units but has return\_sequences set to False to output the final prediction for the time step. Another dropout layer is included after the second GRU layer.

After the recurrent layers, a Dense layer with 25 units and relu activation is added to introduce non-linearity followed by the final Dense layer which outputs the predicted stock price. This model architecture enables the GRU to capture the temporal patterns in stock price data effectively while leveraging the additional information provided by the technical indicators.

# Chapter 5: Model Training

## 5.1 Introduction

This chapter provides an in-depth overview of the model training process for both base and feature-engineered LSTM, CNN, and GRU models. It outlines the consistent approach used to train each model ensuring that the training methodologies are comparable across architectures. The LSTM, CNN and GRU models are presented with details about their trainable parameters and the layers involved in the prediction process.

## 5.2 Training Base Models

The base models (LSTM, CNN and GRU) were all trained using a consistent approach to ensure a fair comparison. Each model was trained with 80% of the data allocated for training and 20% for validation. The training process involved 100 epochs with a batch size of 25 using the MSE loss function to measure performance which is commonly used for regression tasks. The Adam optimizer was employed across all models to adjust weights and optimize the training process. This uniform setup allowed for effective performance monitoring and model comparison.

### 5.2.1 Training LSTM Base Model

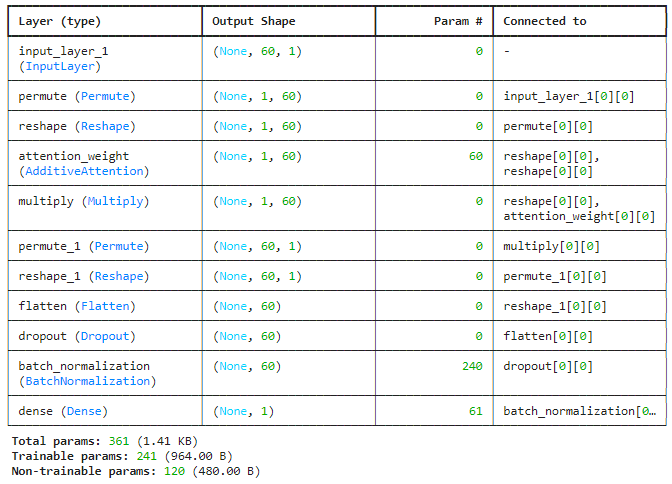


Figure 25. Model Summary of LSTM Base Model

The LSTM base model has 241 trainable parameters. These include 60 attention weights, 120 parameters from the batch normalization layer and 61 weights and biases from the dense layer. The model has 120 non-trainable parameters which are the shift factors from the batch normalization layer that are adjusted during training but not learned as weights.

In total, the model has 361 parameters: 241 of them are learned by the model during training to predict the stock prices, and 120 non-trainable parameters assist in the batch normalization process to ensure that the learning process is stable and efficient. The relatively small number of parameters reflects the simplicity and efficiency of the model, making it well-suited for time-series stock price prediction while avoiding overfitting.

### 5.2.2 Training CNN Base Model

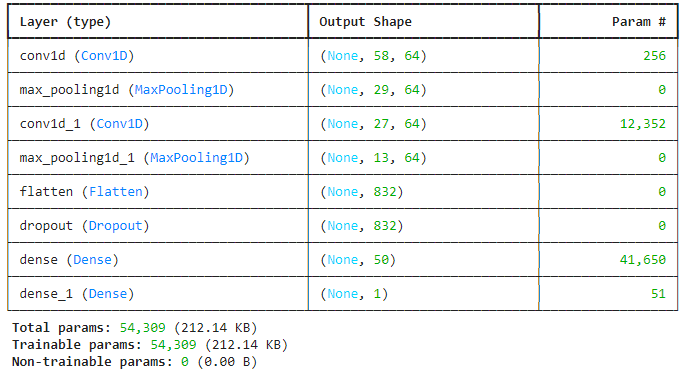


Figure 26. Model Summary of CNN Base Model

The CNN base model has 54,309 trainable parameters. These parameters include the convolutional filter weights, biases and the weights in the dense layers, all of which are updated during training to optimize stock price prediction accuracy. The model has 0 non-trainable parameters, meaning all parameters are adjusted during training.

In total, the model has 54,309 parameters all of which are trainable. The convolutional layers are responsible for extracting temporal features from the stock price data while the fully connected dense layers predict the final stock price. The model is efficient with a moderate number of parameters making it suitable for capturing relevant features in stock price prediction tasks.

### 5.2.3 Training GRU Base Model

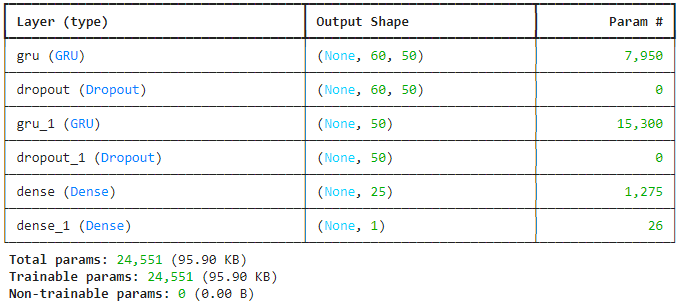


Figure 27. Model Summary of GRU Base Model

The GRU base model has 24,551 trainable parameters. These include all the weights and biases in the GRU and dense layers which are adjusted during the training process to minimize the error between predicted and actual stock prices. The model has 0 non-trainable parameters, meaning that all parameters are updated during training to optimize the model's predictions.

The GRU base model has a total of 24,551 parameters, all of which are trainable. The model's architecture consists of two GRU layers that capture temporal dependencies followed by dense layers that map the learned features to stock price predictions. This relatively lightweight architecture allows for efficient training while still being capable of learning meaningful patterns from historical stock price data.

## 5.3 Training Feature Engineered Models

The feature-engineered versions of the LSTM, CNN and GRU models followed a similar training process. Similar to the base models, 80% of the data was used for training and 20% for validation. These models were trained for 100 epochs with batch sizes of 25 for the CNN and GRU models and 32 for the LSTM Model. This setup ensured that the model’s performance was closely monitored on unseen data during training helping to prevent overfitting. The MSE loss function and Adam optimizer were again used across all feature-engineered models ensuring a consistent approach to performance evaluation and optimization. The inclusion of technical indicators aimed to improve the ability of the models to capture market trends and make more accurate predictions.

### 5.3.1 Training Feature-Engineered LSTM Model

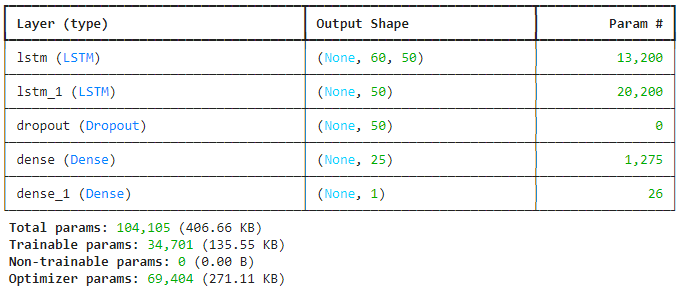


Figure 28. Model Summary of Feature-Engineered LSTM Model

The Feature-Engineered LSTM Model has 34,701 trainable parameters which include all the weights and biases in the LSTM and dense layers. These parameters are adjusted during training to minimize the difference between the predicted and actual stock prices. The model has 0 non-trainable parameters meaning that all parameters are updated during the training process. The optimizer (Adam) has 69,404 parameters which help adjust the learning rate and other optimization aspects to ensure stable training.

The feature-engineered LSTM model has a total of 104,105 parameters of which 34,701 are trainable. The model uses two LSTM layers to capture the temporal relationships between stock prices and technical indicators followed by dense layers to transform the learned features into a stock price prediction. The inclusion of technical indicators as input features helps the model capture more comprehensive market patterns enhancing its ability to predict future stock prices accurately.

### 5.3.2 Training Feature-Engineered CNN Model

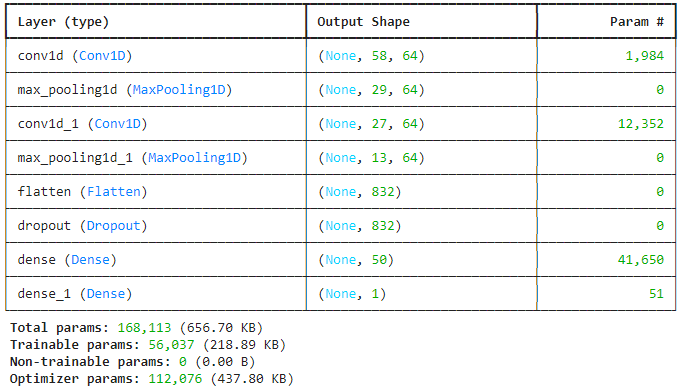


Figure 29. Model Summary of Feature-Engineered CNN Model

The feature-engineered CNN model has 56,037 trainable parameters which include the weights and biases in the convolutional and dense layers. These parameters are optimized during the training process. The model has 0 non-trainable parameters meaning that all parameters are updated during training. The optimizer (Adam) has 112,076 parameters which help adjust the learning rate and other optimization aspects to ensure stable training.

The feature-engineered CNN model has a total of 168,113 parameters of which 56,037 are trainable. The model uses convolutional layers to capture spatial and temporal patterns from the input data followed by dense layers to transform the learned features into stock price predictions. The use of max-pooling and flattening helps to reduce the dimensionality of the data allowing the model to focus on the most relevant features.

### 5.3.3 Training Feature-Engineered GRU Model

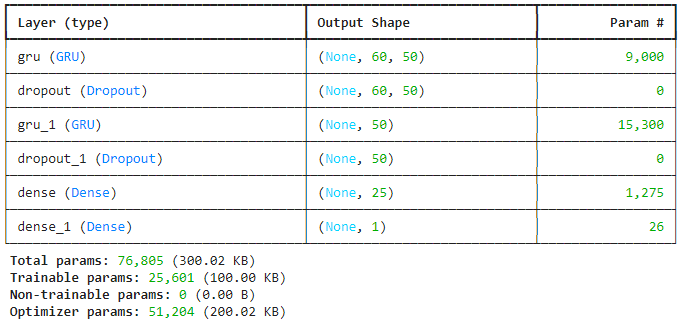


Figure 30. Model Summary of Feature-Engineered GRU Model

The feature-engineered GRU model has 25,601 trainable parameters which include the weights and biases in the GRU and dense layers. These parameters are optimized during training to minimize the difference in the stock prices. The model has 0 non-trainable parameters meaning that all parameters are updated during training. The optimizer (Adam) has 51,204 parameters which control how the model's weights are updated during training to ensure convergence.

The feature-engineered GRU model has a total of 76,805 parameters of which 25,601 are trainable. The GRU layers capture temporal dependencies in the input data while the dense layers transform these learned features into stock price predictions. The inclusion of technical indicators along with historical stock prices provides additional information improving the model's accuracy in predicting future stock prices.

# Chapter 6: Performance Evaluation and Comparative Analysis

## 6.1 Introduction

This chapter focuses on the evaluation of the base and feature-engineered models developed in this study to predict stock prices for Apple and Microsoft. The chapter compares the effectiveness of LSTM, CNN and GRU in capturing stock price trends and producing accurate predictions. The evaluation process uses a range of metrics, such as MAE and RMSE to quantify the accuracy of each model in forecasting stock prices.

In this chapter, a comprehensive comparative analysis between the models is conducted aiming to identify strengths and weaknesses as well as practical implications for financial forecasting. The evaluation sheds light on which models perform best in different market conditions providing insights into the trade-offs between complexity and accuracy.

## 6.2 Evaluation of Base Models

### 6.2.1 Evaluation of LSTM Base Models

Apple Stock

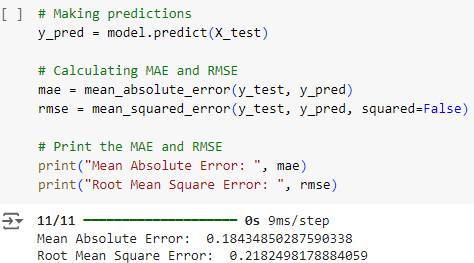


Figure 31. Evaluation metrics of Base LSTM model (Apple Stock Price)

The MAE and RMSE were calculated to provide further insights into the prediction accuracy. The MAE of the model was 0.184 and the RMSE was 0.218 which highlight the average deviation and the overall error magnitude in predicting stock prices.

Microsoft Stock

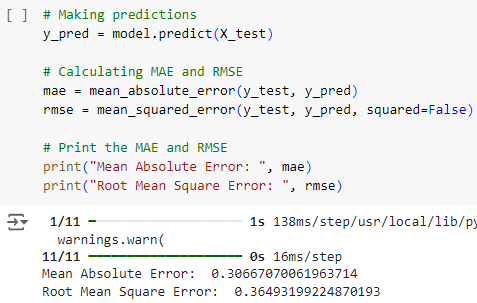


Figure 32. Evaluation metrics of Base LSTM model (Microsoft Stock Price)

The MAE was calculated to be 0.307 which is the average deviation. The RMSE was 0.365 which accounts for the squared differences between predicted and actual values highlighting areas where the model’s predictions showed larger deviations from the real stock prices.

### 6.2.2 Evaluation of CNN Base Models

Apple Stock

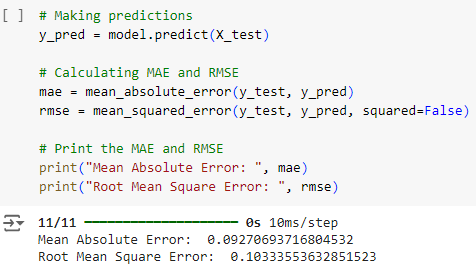


Figure 33. Evaluation metrics of Base CNN model (Apple Stock Price)

The MAE and RMSE were calculated to provide further insights into the prediction accuracy. The MAE of the model was 0.093 and the RMSE was 0.103 which highlight the average deviation and overall error magnitude in predicting Apple stock prices.

Microsoft Stock

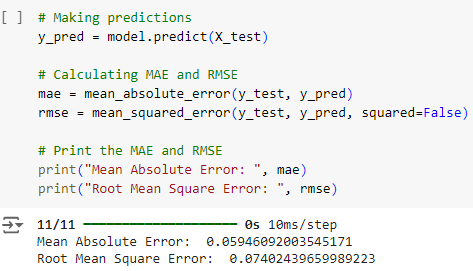


Figure 34. Evaluation metrics of Base CNN model (Microsoft Stock Price)

The MAE which captures the average difference between the predicted and actual stock prices was 0.059. The RMSE which provides an overall measure of error magnitude was 0.074. These metrics indicate that the CNN base model demonstrates strong accuracy in predicting Microsoft stock price with relatively low error values.

### 6.2.3 Evaluation of GRU Base Models

Apple Stock

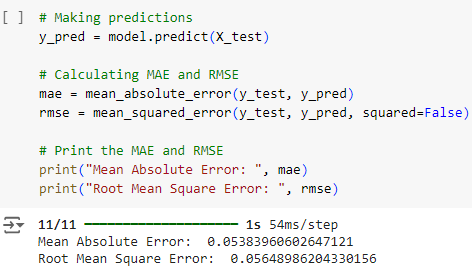


Figure 35. Evaluation metrics of Base GRU model (Apple Stock Price)

The MAE which measures the average absolute difference between the predicted and actual prices was 0.0538. The RMSE which gives a measure of the overall error magnitude was 0.0565. These results highlight the GRU model’s ability to predict Apple stock prices with a high degree of accuracy characterized by low error margins.

Microsoft Stock

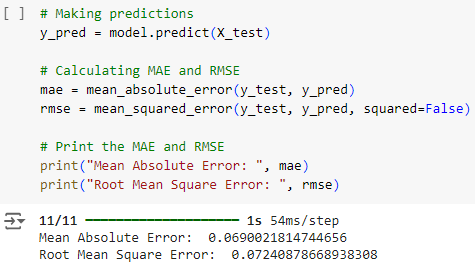


Figure 36. Evaluation metrics of Base GRU model (Microsoft Stock Price)

The MAE was 0.0690 representing the average difference between the predicted and actual stock prices for Microsoft. The RMSE was 0.0724 measuring the overall magnitude of prediction errors. This value highlights the model's effectiveness in predicting stock prices with relatively low deviation from the actual values.

## 6.3 Evaluation of Feature-Engineered Models

### 6.3.1 Evaluation of Feature-Engineered LSTM Models

Apple Stock

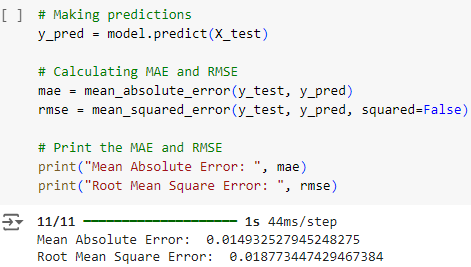


Figure 37. Evaluation metrics of Feature-Engineered LSTM model (Apple Stock Price)

The MAE was 0.0149 representing the average absolute difference between the predicted and actual Apple stock prices. This low value highlights the model’s ability to generate highly accurate predictions. The RMSE was 0.0188 measuring the overall error magnitude in the predictions. This further supports the model's precision in forecasting stock prices.

Microsoft Stock

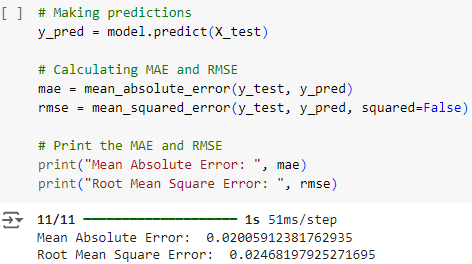


Figure 38. Evaluation metrics of Feature-Engineered LSTM model (Microsoft Stock Price)

The MAE was 0.0201 representing the average absolute difference between the predicted and actual stock prices. This value demonstrates the model's ability to produce accurate predictions with minimal error. The RMSE was 0.0247 highlighting the overall error magnitude in the predictions and further supporting the model's accuracy.

### 6.3.2 Evaluation of Feature-Engineered CNN Models

Apple Stock

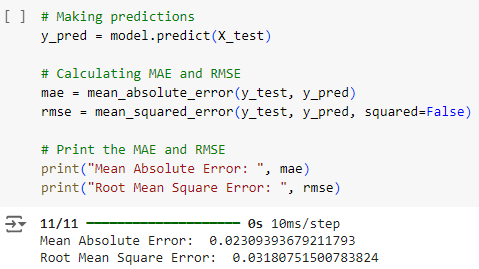


Figure 39. Evaluation metrics of Feature-Engineered CNN model (Apple Stock Price)

The MAE was 0.0231 representing the average absolute difference between predicted and actual stock prices. This low value suggests that the model produces accurate predictions with minimal deviations. The RMSE was 0.0318 indicating the overall error magnitude in predictions further validating the model's ability to predict Apple stock prices accurately.

Microsoft Stock

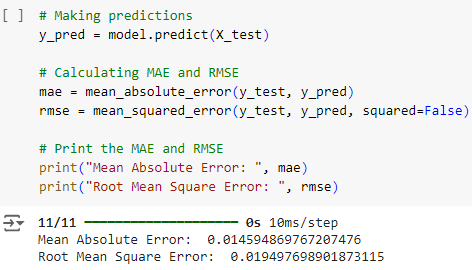


Figure 40. Evaluation metrics of Feature-Engineered CNN model (Microsoft Stock Price)

The MAE was 0.0146 signifying the average absolute difference between the predicted and actual stock prices highlighting the model's precision. The RMSE was 0.0195 providing a measure of the overall error magnitude further reflecting the model's strong ability to predict Microsoft stock prices with high accuracy.

### 6.3.3 Evaluation of Feature-Engineered GRU Models

Apple Stock

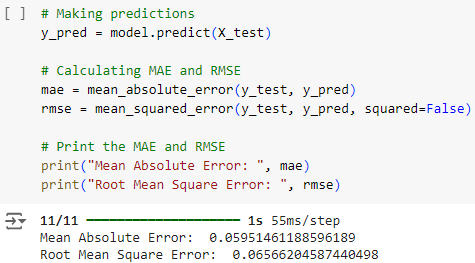


Figure 41. Evaluation metrics of Feature-Engineered GRU model (Apple Stock Price)

The model achieved an MAE of 0.0595 indicating the average absolute difference between the predicted and actual stock prices showing the model’s precision in stock price forecasting. The RMSE was 0.0657 offering a measure of the overall error magnitude highlighting the model’s strong performance in predicting Apple stock prices with a high level of accuracy.

Microsoft Stock

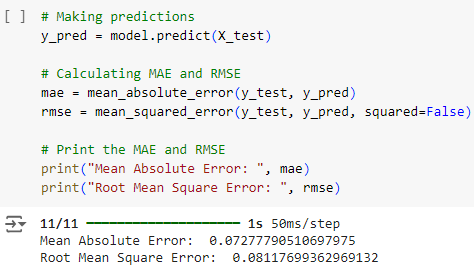


Figure 42. Evaluation metrics of Feature-Engineered GRU model (Microsoft Stock Price)

The model achieved an MAE of 0.0728 representing the average absolute difference between the predicted and actual Microsoft stock prices showcasing the model’s precision in its predictions. The RMSE was 0.0812 indicating the overall magnitude of error in predictions demonstrating that the model was able to predict Microsoft stock prices effectively.

## 6.4 Future Stock Price Predictions

In addition to evaluating the base and feature-engineered models on historical data, the models were used to predict stock prices for the next four days for both Apple and Microsoft. This forecasting was done by feeding the last available data points from the test set into the trained models and generating the future price values.

The prediction process involved utilizing the most recent data from the test dataset and the models output the expected closing prices for the subsequent trading days. These predictions aim to demonstrate the real-world application of the trained models in forecasting near-term price movements.

### 6.4.1 Future Stock Price Predictions of LSTM Base Models

Apple Stock

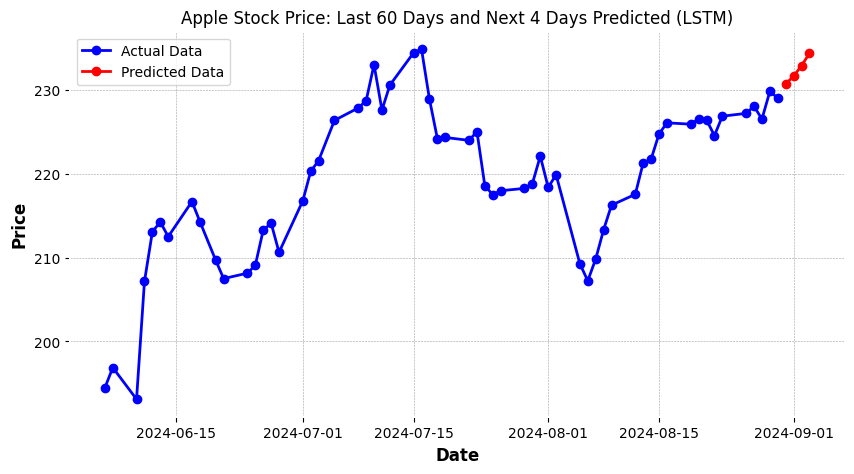


Figure 43. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Base Model)

Figure 43 illustrates the Apple stock price for the last 60 days along with the predicted prices for the next 4 days generated using the LSTM base model. The blue line represents the actual stock prices observed during the historical period while the red dotted line indicates the LSTM model's predicted prices for the upcoming four days.

From the historical data, we can observe fluctuations in the Apple stock price over the last two months with a general upward trend emerging in the later period. The stock price peaked around mid-July and then experienced a drop followed by a recovery phase throughout August. The LSTM model captures this recovery and projects a continued rise in the stock price over the next four days.

The graph clearly distinguishes the predicted values from the actual data allowing for easy comparison between the past performance and the model's future predictions. The steady upward trend in the predicted stock prices aligns with the momentum observed in the latter part of the historical data suggesting that the LSTM model has successfully captured the market's general direction and anticipates its continuation.

Microsoft Stock

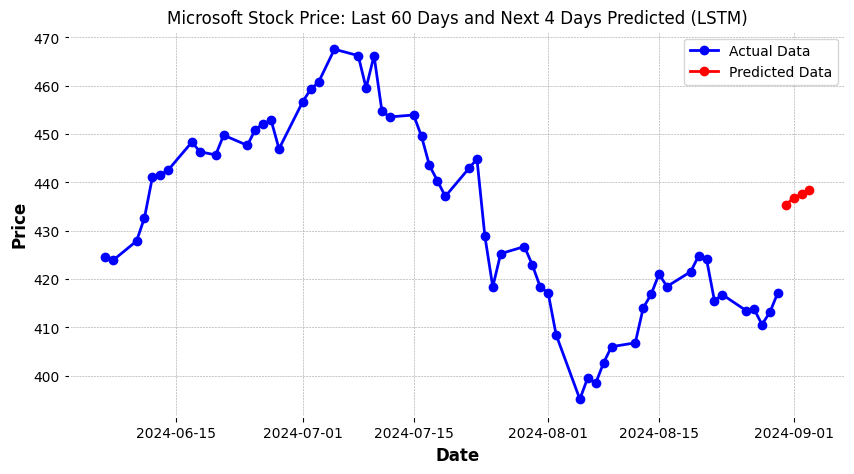


Figure 44. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Base Model)

Figure 44 represents the Microsoft stock price prediction for the next 4 days using the LSTM base model alongside the actual stock prices from the last 60 days.

From the graph, the stock price appears to be in a recovery phase after a decline and the model predicts a gradual rise in Microsoft stock prices over the forecasted period. The next four-day predicted prices are plotted as red points indicating a steady upward trend based on the model's learning from past patterns.

This visualization emphasizes the LSTM model's capability to capture trends from the previous 60 days and extrapolate them into the near future providing a visual representation of the model's predictive performance.

### 6.4.2 Future Stock Price Predictions of CNN Base Models

Apple Stock

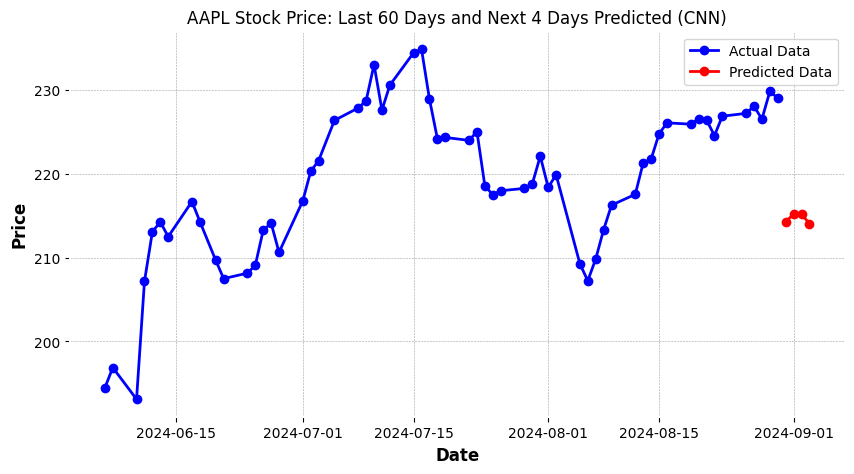


Figure 45. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Base Model)

Figure 45 illustrates the Apple stock price prediction for the next 4 days using the CNN base model with the actual stock prices from the previous 60 days included for context.

The actual data from the last 60 days displays a clear upward trend with some fluctuations peaking around mid-July. However, the CNN model predicts a slight decline in stock prices for the next 4 days as indicated by the red line descending after August 31st.

This graph demonstrates the CNN model’s predictions based on its ability to capture patterns from the historical data, yet unlike the LSTM model's prediction, it forecasts a downturn suggesting that the model anticipates short-term downward momentum for the Apple stock prices.

Microsoft Stock

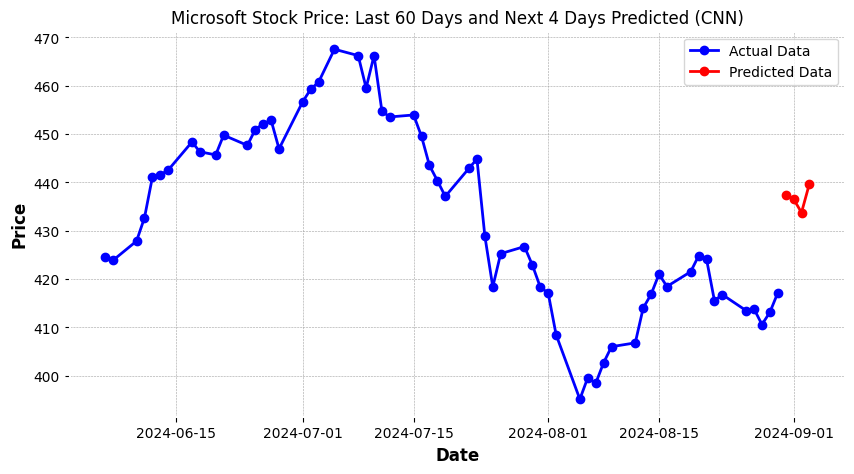


Figure 46. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Base Model)

Figure 46 represents the Microsoft stock price prediction for the next 4 days using the CNN base model alongside the actual stock prices from the previous 60 days for context.

The historical data shows a declining trend from mid-July followed by a partial recovery towards the end of August. The CNN model's prediction for the next 4 days reflects a continued upward movement as indicated by the red line climbing between August 31st and September 3rd. The forecast suggests that the model anticipates a minor increase in Microsoft stock prices in the immediate future contrasting with the more volatile fluctuations seen in recent months.

This graph demonstrates how the CNN model captures the recent upward trend and expects this momentum to persist though it is predicting relatively smaller gains compared to the larger movements seen in prior months.

### 6.4.3 Future Stock Price Predictions of GRU Base Models

Apple Stock

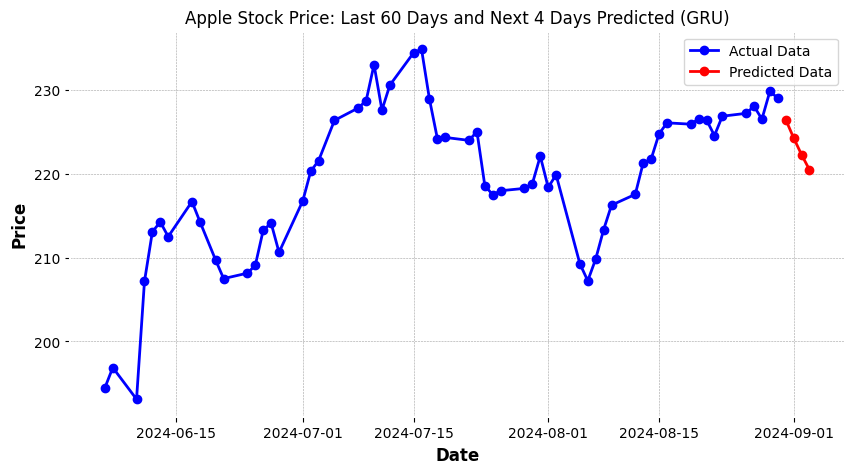


Figure 47. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Base Model)

Figure 47 represents the Apple stock price prediction for the next 4 days using the GRU base model combined with the actual stock prices from the previous 60 days for reference.

The historical data shows an overall upward trend from mid-June to mid-July followed by some volatility in August. The GRU model predicts a slight downward movement in the stock price for the next 4 days as shown by the red line dropping from the end of August into early September. This suggests that the model anticipates a short-term decline in Apple's stock price following the recent fluctuations.

This prediction highlights the GRU model’s ability to capture recent volatility in the stock price after the upward trend observed in mid-August. The model predicts a pullback indicating that Apple’s stock might face a period of short-term depreciation according to this model's forecast.

Microsoft Stock

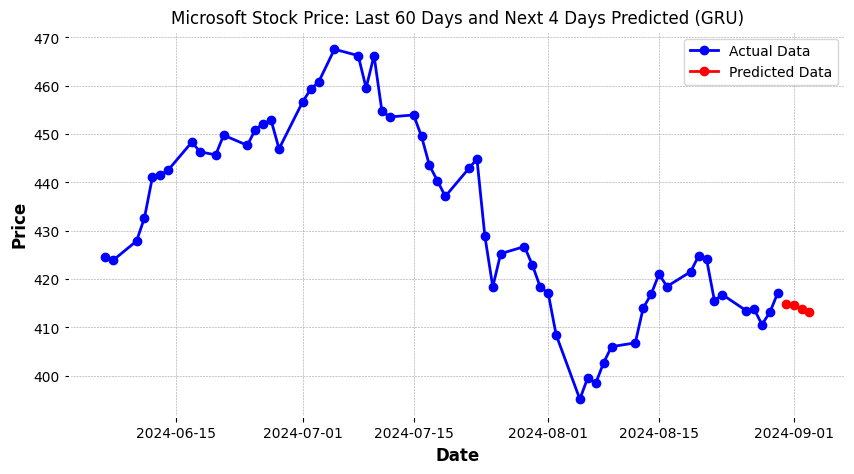


Figure 48. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Base Model)

Figure 48 represents the Microsoft stock price prediction for the next 4 days using the GRU base model alongside the actual stock prices from the previous 60 days.

The historical data shows a steady rise in Microsoft stock prices from mid-June to mid-July followed by a noticeable decline that persisted throughout the remainder of July and early August. The GRU model predicts a slight decrease in Microsoft stock prices over the next 4 days as indicated by the red points that dip below the current price level. This suggests that the model anticipates a short-term downward trend possibly following the recent fluctuations observed in the price.

The forecast indicates that the GRU model expects Microsoft’s stock price to continue its relatively flat trajectory but with a slight decrease over the upcoming days.

### 6.4.4 Future Stock Price Predictions of Feature-Engineered LSTM Models

Apple Stock

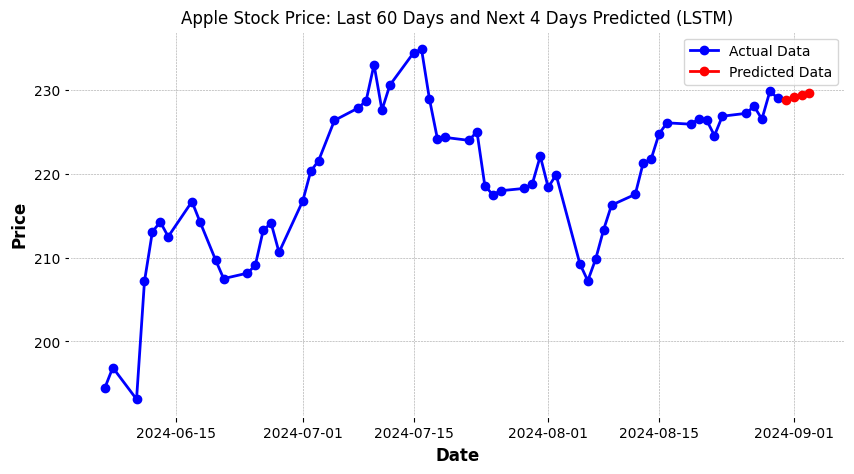


Figure 49. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Feature-Engineered Model)

Figure 49 showcases the Apple stock price prediction for the next 4 days using the LSTM Feature-Engineered model alongside the actual stock prices from the past 60 days.

The historical data reflects a period of volatility in mid-June 2024 followed by a notable increase in stock prices throughout July peaking mid-month. The prices subsequently experienced some fluctuations in late July and August. The LSTM Feature-Engineered model predicts a slight upward movement in the stock price over the next 4 days with the red points showing a consistent rise toward early September 2024. This indicates that the model expects Apple's stock price to continue increasing slightly in the short-term maintaining the overall positive momentum observed in the recent data.

This prediction reflects the model's ability to integrate technical indicators along with historical stock prices capturing the trends in the market for a more accurate short-term forecast.

Microsoft Stock

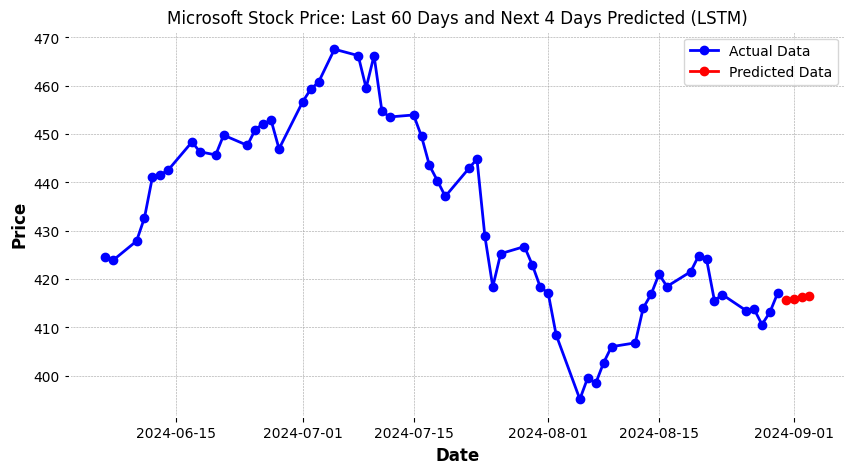


Figure 50. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (LSTM Feature-Engineered Model)

Figure 50 displays the Microsoft stock price prediction for the next 4 days using the LSTM Feature-Engineered model alongside the actual stock prices from the past 60 days.

The historical data reveals a downward trend that started around mid-July 2024 with a significant decline in early August. After reaching a low point, the stock price began to recover in mid-August showing a slight increase by the end of the month. The LSTM Feature-Engineered model predicts that the stock price will continue to rise slightly over the next 4 days. The red points show a steady upward trajectory in early September 2024 indicating the model's expectation of a minor positive trend in Microsoft stock prices.

This forecast suggests that the model is capturing the recent recovery in Microsoft's stock and anticipates a continuation of this upward momentum in the short term.

### 6.4.5 Future Stock Price Predictions of Feature-Engineered CNN Models

Apple Stock

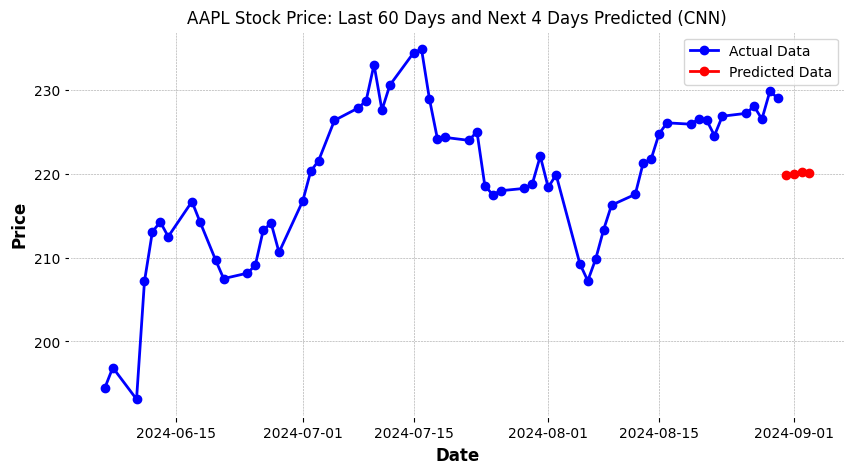


Figure 51. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Feature-Engineered Model)

Figure 51 presents the Apple stock price prediction for the next 4 days using the CNN Feature-Engineered model alongside the actual stock prices from the past 60 days.

The historical data shows a strong upward trend in June and July 2024 followed by some fluctuation including a dip in early August and a recovery by late August. The model predicts that the stock price will drop and then remain relatively stable for the next few days as indicated by the red points clustering together around the $220 mark.

The model expects minimal movement in Apple's stock price in the short term. This suggests that the model may have identified a period of consolidation or stabilization forecasting that the stock price will maintain its current levels rather than follow the recent upward momentum seen in August.

Microsoft Stock

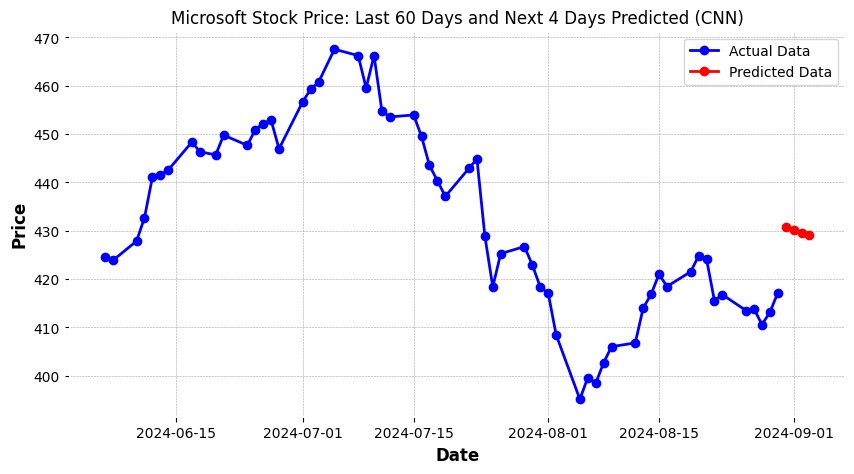


Figure 52. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (CNN Feature-Engineered Model)

Figure 52 illustrates the Microsoft stock price prediction for the next 4 days using the CNN Feature-Engineered model along with the actual stock prices from the past 60 days.

From mid-June 2024 to mid-July, the stock shows a steady increase peaking around mid-July before entering a declining trend that extends through early August. The price then begins to recover slightly toward the end of August. The model predicts that the stock price will rise slightly over the next 4 days as indicated by the red dots.

These predicted prices are slightly higher than the most recent data points suggesting that the CNN Feature-Engineered model anticipates a continuation of the recent upward movement but at a moderate pace. This prediction reflects a cautious optimism in the stock’s short-term outlook.

### 6.4.6 Future Stock Price Predictions of Feature-Engineered GRU Models

Apple Stock

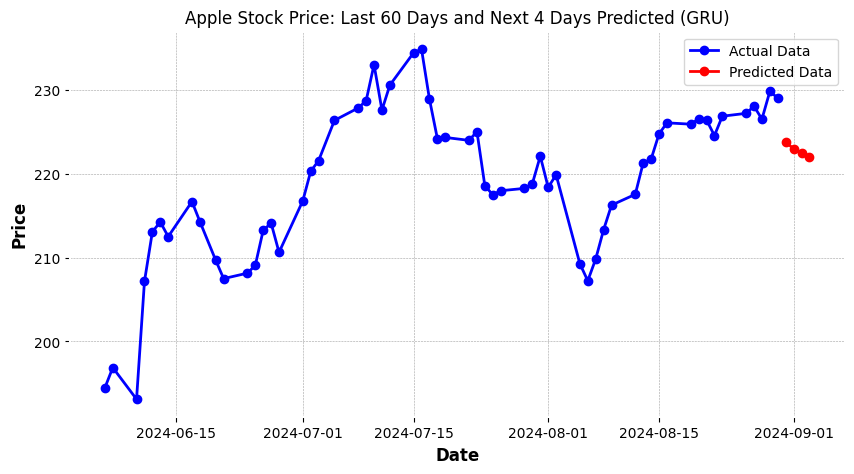


Figure 53. Apple Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Feature-Engineered Model)

Figure 53 shows the Apple stock price prediction for the next 4 days using the GRU Feature-Engineered model alongside the actual stock prices from the previous 60 days.

Over the observed period, the Apple stock price demonstrated significant volatility with a noticeable rise through mid-July 2024 followed by a sharp dip in early August. The price then recovers somewhat ending August on an upward trend. However, the GRU Feature-Engineered model predicts a slight decline in the stock price over the next 4 days as indicated by the descending red points. This suggests that the model expects a short-term pullback despite the recent upward momentum.

This prediction reflects the model’s interpretation of historical price patterns and additional engineered features suggesting a cautious outlook for the immediate future of Apple's stock price.

Microsoft Stock

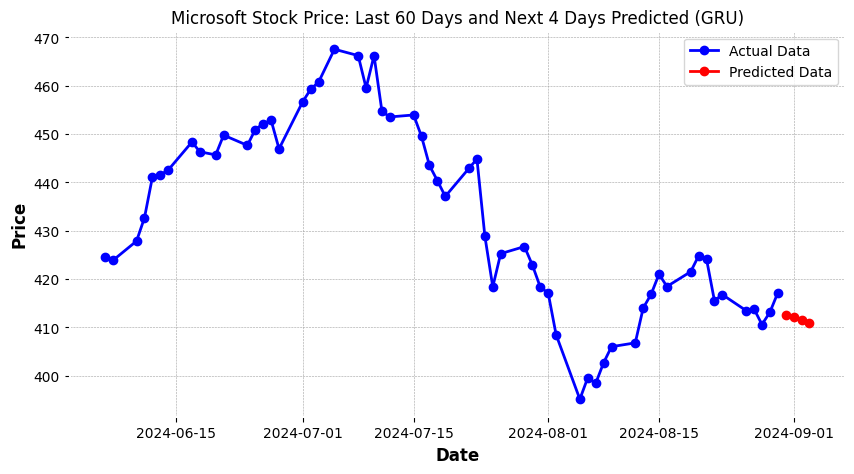


Figure 54. Microsoft Stock Price Prediction - Last 60 Days and Next 4 Days Forecasted (GRU Feature-Engineered Model)

Figure 54 shows the Microsoft stock price prediction for the next 4 days using the GRU Feature-Engineered model compared with the actual stock prices from the previous 60 days.

The graph shows a notable downward trend in Microsoft stock throughout July 2024 followed by some recovery in mid-August. However, the GRU Feature-Engineered model predicts a slight decline in stock prices over the next 4 days as indicated by the downward trend of the red dots. This indicates that the model expects the stock to experience a brief pullback aligning with the slight drop seen at the end of August.

The GRU Feature-Engineered model which takes into account historical stock patterns and engineered features suggests that the stock price may continue to experience some short-term weakness after the recent recovery.

## 6.5 Comparative Analysis

### 6.5.1 Base vs. Feature-Engineered Models

The base models of LSTM, CNN and GRU were trained using only historical stock price data. These models demonstrated robust performance in capturing the temporal dependencies and patterns inherent in the stock price movements. However, the prediction accuracy was somewhat limited particularly when dealing with volatile periods in the stock market. This limitation was observed in both the Apple and Microsoft stock price predictions where the base models displayed higher MAE and RMSE values than the Feature-Engineered models.

In contrast, the feature-engineered models which incorporated additional technical indicators significantly improved the predictive power of all three model types. The inclusion of these indicators allowed the models to capture short-term fluctuations and long-term trends resulting in more accurate stock price predictions. As demonstrated in the evaluation metrics, the feature-engineered models consistently outperformed their base counterparts showing lower MAE and RMSE rates.

For instance, the LSTM feature-engineered model was particularly effective in reducing error during volatile market periods outperforming the base LSTM model in both Apple and Microsoft stock price predictions. Similarly, the feature-engineered CNN and GRU models displayed greater performance suggesting that the technical indicators provided additional context that improved the model’s understanding of the market.

In summary, while the base models provided a solid foundation for stock price prediction, the feature-engineered models demonstrated superior performance by incorporating additional market indicators. This comparison highlights the importance of feature engineering in improving the accuracy and robustness of deep learning models in financial forecasting.

### 6.5.2 Strengths and Weaknesses

Strengths and weaknesses of Base Models

The base models demonstrated strong capability in capturing the temporal dependencies of stock price movements using only the historical data. The recurrent layers in LSTM and GRU models effectively managed sequential dependencies while the CNN model's convolutional layers were able to extract key features from the time-series data.

One of the key strengths of the base models is their simplicity. By relying solely on the stock price data, these models require less feature engineering and are more straightforward to implement. This makes them useful in scenarios where limited domain knowledge is available or when the primary goal is to scale model performance quickly.

However, the base models struggled to accurately predict stock price movements in volatile periods and their ability to generalize was limited when market conditions shifted drastically. Without additional market context, the models missed short-term fluctuations that are better captured by more complex inputs such as technical indicators. This limitation was reflected in the higher RMSE and MAE scores than the feature-engineered models.

Strengths and weaknesses of Feature-Engineered Models

The feature-engineered models greatly enhanced the model’s ability to predict stock prices particularly during periods of high volatility. The technical indicators provided a richer context for the models allowing them to learn more patterns in stock price movements.

By combining these additional features with the strengths of the LSTM, CNN and GRU architectures, the feature-engineered models showed superior predictive accuracy and robustness. This was reflected in lower RMSE and MAE scores highlighting their ability to capture more subtle market dynamics and make more precise predictions.

However, the feature-engineered models come with added complexity. Incorporating technical indicators requires domain knowledge and additional computational resources for feature engineering. This increases the overall training time and can make the models more difficult to fine-tune. Additionally, there is a risk of overfitting when too many features are added as the model may become too dependent on the training data without generalizing well to new unseen data.

### 6.5.3 Practical Implications

In practice, the best model for stock price prediction will depend on the specific goals of the user. If the goal is to create a fast, general-purpose stock price predictor with low computational overhead, the base models (particularly CNN) are suitable candidates. These models can offer decent accuracy for long-term trends without requiring significant computational resources or advanced feature engineering.

On the other hand, for traders who require high accuracy in short-term predictions and the ability to navigate volatile market conditions, the feature-engineered models provide the necessary depth and sophistication. The inclusion of technical indicators significantly boosts performance in these contexts making them invaluable for trading systems where precision is key.

In conclusion, while base models are more accessible and faster to implement, feature-engineered models offer superior accuracy and robustness for real-time trading and complex market conditions. The decision on which model to use should be driven by the specific needs of the application whether it is simplicity and speed or accuracy.

# Chapter 7: Conclusion

## 7.1 Summary of Findings

This project explored the effectiveness of deep learning models in predicting future stock prices using historical stock data from Apple and Microsoft. The primary focus of the study was to compare the performance of base models with feature-engineered versions of the same models examining whether incorporating technical indicators improved predictive accuracy.

The base models demonstrated the ability to capture temporal dependencies in stock price data with each model showing varying levels of success in predicting stock price trends. The LSTM base model, known for its ability to handle long-term dependencies performed well. The GRU base model offered a balance between accuracy and computational efficiency while the CNN model was effective at capturing localized patterns.

The introduction of feature engineering significantly improved the predictive power of all models. Feature-engineered models consistently outperformed their base counterparts in terms of MAE and RMSE. For instance, the feature-engineered LSTM model demonstrated improved accuracy particularly for Apple's stock where the technical indicators helped the model better capture short-term fluctuations and broader market trends.

Across the models, the feature-engineered GRU emerged as one of the top performers providing a good balance between computational efficiency and predictive accuracy. The GRU's ability to handle noisy data combined with the added context from technical indicators made it particularly suitable for volatile market conditions.

In conclusion, the findings of this research support the use of feature-engineered deep learning models for stock price prediction. The results indicate that while base models are capable of accurate predictions incorporating technical indicators further increases model performance making them more suitable for real-world applications in financial forecasting.

## 7.2 Implications for Financial Forecasting

The findings from this project have significant implications for the field of financial forecasting. The performance comparison between base models and feature-engineered models highlights several key insights that can be applied to real-world financial forecasting.

First, the project demonstrates that feature engineering can substantially improve the predictive accuracy of deep learning models. This suggests that financial institutions and traders who rely on stock price predictions could benefit from including technical indicators into their predictive models. The ability of feature-engineered models to capture both short-term fluctuations and long-term trends makes them well-suited for a variety of trading strategies such as long-term investment forecasting.

The improved performance of the feature-engineered GRU and LSTM models underscores the importance of considering temporal relationships in stock price data. These models, when combined with feature engineering, were able to capture complex patterns more effectively than the base models. This demonstrates that advanced deep learning techniques combined with domain-specific knowledge such as technical indicators can improve decision-making in stock markets.

Furthermore, the consistent outperformance of feature-engineered models over base models suggests that data-driven approaches to stock price forecasting should not rely solely on historical price data but should include additional market signals. This approach aligns with modern financial forecasting practices where a combination of quantitative analysis, technical indicators and deep learning is becoming the norm.

In summary, this project shows that integrating feature engineering with deep learning models can lead to more accurate and reliable stock price predictions. These insights are valuable for traders looking to improve the precision of their forecasts and improve their decision-making processes in volatile markets.

## 7.3 Limitations of the Study

While this study demonstrates the potential of deep learning models in stock price prediction several limitations must be acknowledged particularly in the context of feature-engineered models.

First, the data used for training and evaluation was limited to historical stock prices from Apple and Microsoft between 2017 and 2024. This relatively small and specific dataset may not fully capture the complexities of broader stock markets or other financial instruments. Additionally, stock prices are influenced by numerous factors such as economic conditions and corporate decisions which were not accounted for in the feature engineering process. The feature-engineered models focused primarily on technical indicators and did not incorporate fundamental data such as earnings reports and interest rates which could improve predictive performance.

Second, the models were tested on only two stocks (Apple and Microsoft), both of which are large technology companies. This limits the findings to smaller companies which may show different volatility patterns. Feature-engineered models might perform differently when applied to more volatile stocks and additional testing across a broader range of stocks is necessary to validate the results.

Another limitation is the assumption of time-series data being stationary meaning that the models assume market conditions remain relatively stable over time. However, real-world financial markets are subject to changes such as sudden market crashes which can drastically affect stock prices. The models in this study may not fully capture these shifts limiting their effectiveness in unpredictable market environments.

Additionally, while feature engineering improved the accuracy of the models it introduced complexity to the modeling process. Choosing the right technical indicators as well as their parameters required domain knowledge and trial and error which may not be practical or scalable in larger-scale implementations. Furthermore, the added complexity of feature engineering can sometimes lead to overfitting especially when applied to limited datasets.

In summary, while this study provides valuable insights into stock price prediction using deep learning and feature engineering, the results are subject to the limitations of the dataset and model assumptions. These factors should be considered when interpreting the findings and applying the models to real-world financial forecasting tasks.

## 7.4 Future Research

The findings from this dissertation provide a solid foundation for future research in stock price prediction using deep learning models particularly in the context of feature engineering. However, there are several areas where future studies could expand and improve upon the base and feature-engineered models explored in this work.

First, future research should consider incorporating a wider variety of stocks. This project focused on Apple and Microsoft but applying these models to a more diverse dataset of stocks from different sectors could provide a broader understanding of the model’s effectiveness across different market conditions. This would also test the robustness of the feature-engineering techniques as certain indicators may be more relevant to particular sectors or asset classes.

Incorporating additional data sources, such as macroeconomic indicators (interest rates and GDP growth) could improve model performance. While the feature-engineered models in this study relied heavily on technical indicators, future research could investigate how combining technical and fundamental data impacts predictive accuracy.

Another promising area for future research is the use of more advanced deep learning architectures. While this study focused on LSTM, CNN and GRU models, future studies could explore hybrid architectures such as combining LSTM with transformer models which have recently gained popularity in time-series forecasting. These architectures could potentially capture complex dependencies in stock price data and improve predictive performance.

The project also revealed some challenges with overfitting when using feature-engineered models. Future research could investigate more robust techniques to prevent overfitting, such as regularization methods or ensemble models. These techniques could improve the applicability of the models especially when applied to unseen data.

Moreover, the models in this project assumed relatively stable market conditions. Future research could address this limitation by developing models that adapt to changing markets. Reinforcement learning algorithms could be used to dynamically adjust model parameters based on the current market environment potentially improving the model’s ability to handle periods of high volatility.

Finally, future studies could explore the practical application of these models in real-world trading systems. By integrating the models with live market data, traders could test their effectiveness in real-time trading environments assessing their ability to generate profitable trading strategies.

In conclusion, while this project has contributed to the understanding of deep learning models for stock price prediction, there remains significant scope for future research to refine these models, expand their applicability and address the challenges identified. By incorporating more diverse data sources, testing advanced architectures and exploring real-world applications, future studies can build upon the insights gained in this work and continue advancing the field of financial forecasting.

# References

1. Ahmed S.F., Bin, S., Hassan, M., Rozbu, M.R., Ishtiak, T., Rafa, N., Mofijur, A and Gandomi, A.H. (2023). Deep learning modelling techniques: current progress, applications, advantages, and challenges. *Artificial Intelligence Review*, 56. doi: <https://doi.org/10.1007/s10462-023-10466-8> .
2. Ahmed, W.M.A. (2020). Stock market reactions to domestic sentiment: Panel CS-ARDL evidence. *Research in International Business and Finance*, 54, p.101240. doi: <https://doi.org/10.1016/j.ribaf.2020.101240>.
3. Bao, W., Yue, J. and Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE*, 12(7). doi: <https://doi.org/10.1371/journal.pone.0180944>.
4. Boubaker, S., Karim, S., Naeem, M.A. and Sharma, G.D. (2023). Financial markets, energy shocks, and extreme volatility spillovers. *Energy Economics*, 126, pp.107031. doi: <https://doi.org/10.1016/j.eneco.2023.107031>
5. Bustos, O. and Pomares-Quimbaya, A. (2020). Stock market movement forecast: A Systematic review. *Expert Systems with Applications,* 156, p.113464. doi: <https://doi.org/10.1016/j.eswa.2020.113464>.
6. Chaajer, P., Shah, M. and Kshirsagar, A. (2021). The applications of artificial neural networks, support vector machines, and long-short term memory for stock market prediction. *Decision Analytics Journal*, 2, p.100015. doi: <https://doi.org/10.1016/j.dajour.2021.100015>.
7. Chauhan, N.K. and Singh, K. (2018). A Review on Conventional Machine Learning vs Deep Learning. *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*. doi: <https://doi.org/10.1109/gucon.2018.8675097>.
8. Chen, C. (2023). Stock Price Prediction Based on the Fusion of CNN-GRU Combined Neural Network and Attention Mechanism. doi: <https://doi.org/10.1109/icet58434.2023.10211379>.
9. Chen, C., Xue, L. and Xing, W. (2023). Research on Improved GRU-Based Stock Price Prediction Method. *Applied Sciences*, 13(15), p.8813. doi: <https://doi.org/10.3390/app13158813>.
10. Chen, J. (2019). *Exponential Moving Average - EMA*. Available at: <https://www.investopedia.com/terms/e/ema.asp>.
11. Chen, K., Zhou, Y. and Dai, F. (2015). A LSTM-based method for stock returns prediction: A case study of China stock market. *2015 IEEE International Conference on Big Data)*. doi: <https://doi.org/10.1109/bigdata.2015.7364089>.
12. Dey, R. and Salem, F.M. (2017). Gate-variants of Gated Recurrent Unit (GRU) neural networks. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. doi: <https://doi.org/10.1109/mwscas.2017.8053243>.
13. Dong, H., Guo, X., Reichgelt, H. and Hu, R. (2020). Predictive power of ARIMA models in forecasting equity returns: a sliding window method. *Journal of Asset Management*, 21(6), pp.549–566. doi: <https://doi.org/10.1057/s41260-020-00184-z>.
14. Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), pp.654–669. doi: <https://doi.org/10.1016/j.ejor.2017.11.054>.
15. Graves, D. and Pedrycz, W. (2009). Fuzzy prediction architecture using recurrent neural networks. *Neurocomputing*, 72(7-9), pp.1668–1678. doi: <https://doi.org/10.1016/j.neucom.2008.07.009>.
16. Fama, E.F. (1970). Efficient Capital Markets: a Review of Theory and Empirical Work. *The Journal of Finance*, 25(2), pp.383–417. doi: <https://doi.org/10.2307/2325486>.
17. Henderschott, T., Jones, C.M. and Menkveld, A.J. (2011). Does Algorithmic Trading Improve Liquidity? *The Journal of Finance*, 66(1), pp.1–33. doi: <https://doi.org/10.1111/j.1540-6261.2010.01624.x>.
18. Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780. doi: <https://doi.org/10.1162/neco.1997.9.8.1735>.
19. Kanwal, A., Chandrasekaran, S. and Akhunzada, A. (2023). Stock Price Prediction using CuDNNLSTM and multiple CNN layers. doi: <https://doi.org/10.1109/aibthings58340.2023.10291016>.
20. Khandelwal, D. (2024). *What Is MACD? - Moving Average Convergence/Divergence*. Available at: <https://www.investing.com/academy/analysis/macd-definition-uses/>.
21. Kim, S. and Kang, M. (2019). Financial series prediction using Attention LSTM. *arXiv (Cornell University)*. doi: <https://doi.org/10.48550/arxiv.1902.10877>.
22. Lai, G., Chang, W.C., Yang, Y. and Liu, H. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. doi: <https://doi.org/10.1145/3209978.3210006>.
23. Li, Y. and Xie, Z. (2022). A Study on CNN Feature Extraction for Stock Price Prediction. *BCP Business & Management*, 33, pp.326–330. doi: <https://doi.org/10.54691/bcpbm.v33i.2771>.
24. Lo, A.W. (2004). *The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective*. [online] papers.ssrn.com. Available at: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=602222>.
25. Maeda, I., Matsushima, H., Sakaji, H., Izumi, K., deGraw, D., Kato, A. and Kitano, M. (2019). Effectiveness of Uncertainty Consideration in Neural-Network-Based Financial Forecasting. doi: <https://doi.org/10.1109/iiai-aai.2019.00139>.
26. Mehtab, S. and Sen, J. (2020) Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models. *2020 International Conference on Decision Aid Sciences and Application (DASA)*. pp. 447-453. doi: [10.1109/DASA51403.2020.9317207](https://doi.org/10.1109/DASA51403.2020.9317207)
27. Nelson, D.M.Q., Pereira, A.C.M. and de Oliveira, R.A. (2017). Stock market’s price movement prediction with LSTM neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)*. doi: <https://doi.org/10.1109/ijcnn.2017.7966019>.
28. Pang, T. (2024). APPL stock price prediction based on LSTM and GRU. *Applied and Computational Engineering*, 47(1), pp.200–206. doi: <https://doi.org/10.54254/2755-2721/47/20241343>.
29. Paygude, P., Singh, A., Tripathi, E., Priya, S., Gayakwad, M., Chavan, P., Chaudhary, S., Joshi, R. and Kotecha, K. (2023). A Parameter Based Comparative Study of Deep Learning Algorithms for Stock Price Prediction. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11(7), pp.138–146. doi: <https://doi.org/10.17762/ijritcc.v11i7s.6985>.
30. Rajhans, A., Das, T., Kumar, A. and Sinha, B.B. (2024). Exploring the Impact of Financial News Sentiment on Stock Price Forecasting: A Comparative Deep Learning Approach. doi: <https://doi.org/10.1109/incccs60947.2024.10593315>.
31. Sarker, I.H. (2021). Deep Learning: a Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*, 2(6). doi: <https://doi.org/10.1007/s42979-021-00815-1>.
32. Siami-Namini, S., Tavakoli, N. and Siami Namin, A. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. doi: <https://doi.org/10.1109/icmla.2018.00227>.
33. Shrestha, A., Kumari, R., Ajay Kumar Jena, Ajaya Kumar Parida, Nirupama Parida and Raj Kumar Parida (2023). Stock Market Price Prediction Using GRU and XGB. doi: <https://doi.org/10.1109/ocit59427.2023.10430888>.
34. Shrestha, A. and Mahmood, A. (2019). Review of Deep Learning Algorithms and Architectures. *IEEE Access*, 7, pp.53040–53065. doi: <https://doi.org/10.1109/access.2019.2912200>.
35. Shukla, G.P., Balwani, N., and Kumar, S. (2023). Predicting stock prices with LSTM: A hybrid machine learning model for financial forecasting. *Journal of Information and Optimization Sciences*, 44(3), pp.575–584. doi: <https://doi.org/10.47974/jios-1416>.
36. Stokes, T. (2024). *Bollinger Bands Explained for Beginners - How This Tool Works*. Available at: <https://stokestrades.com/bollinger-bands/>
37. Sulistio, B., Hendric, L., Gaol, F.L., and Soewito, B. (2023). Energy Sector Stock Price Prediction Using The CNN, GRU & LSTM Hybrid Algorithm. doi: <https://doi.org/10.1109/iccosite57641.2023.10127847>.
38. Talbi, D., Chaibi, H. and Maoueti, A. (2021). Political uncertainty, financial crises, and stock market volatility: Evidence from MENA region. *Journal of Public Affairs*. doi: <https://doi.org/10.1002/pa.2783>.
39. Trivedi, D.V. and Patel, S. (2022). An Analysis of GRU-LSTM Hybrid Deep Learning Models for Stock Price Prediction. *International Journal of Scientific Research in Science, Engineering and Technology*, pp.47–51. doi: <https://doi.org/10.32628/ijsrset229264>.
40. Tuboalabo, A., Buinwi, J.A., Buinwi, U., Okatta, C.G. and Johnson, E. (2024). Leveraging business analytics for competitive advantage: Predictive models and data-driven decision making. *International Journal of Management & Entrepreneurship Research*, 6(6), pp.1997–2014. doi: <https://doi.org/10.51594/ijmer.v6i6.1239>.
41. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I. (2017). *Attention Is All You Need*. doi: <https://doi.org/10.48550/arXiv.1706.03762>.
42. Venditti, F. and Veronese, G.F. (2020). Global Financial Markets and Oil Price Shocks in Real Time. *SSRN Electronic Journal*. doi: <https://doi.org/10.2139/ssrn.3701652>.
43. Vuong, P.H., Phu, L.H., Hong, T., Duy, L.N., Bao, P.T. and Trinh, T.D. (2024). A bibliometric literature review of stock price forecasting: From statistical model to deep learning approach. *Science progress*, 107(1). doi: <https://doi.org/10.1177/00368504241236557>.
44. Wan, X. (2023). Stock Price Prediction of High-tech Industry Based on LSTM and GRU: Tesla, Ferrari, and Walmart. *Highlights in Science Engineering and Technology*, 76, pp.585–590. doi: <https://doi.org/10.54097/dy17ph79>.
45. Wen, R., Torkkola, K., Narayanaswamy, B. and Madeka, D. (2017). A Multi-Horizon Quantile Recurrent Forecaster. doi: <https://doi.org/10.48550/arxiv.1711.11053>.
46. Wu, J.M-T., Li, Z., Herencsar, N., Vo, B. and Lin, J.C-W. (2021). A graph-based CNN-LSTM stock price prediction algorithm with leading indicators. *Multimedia Systems*. doi: <https://doi.org/10.1007/s00530-021-00758-w>.
47. Wu, Z. and Huang, M. (2023). Stock Price Prediction During the COVID-19 Pandemic Based on the GRU Algorithm. *Advances in Economics Management and Political Sciences*, 47(1), pp.94–101. doi: <https://doi.org/10.54254/2754-1169/47/20230370>.
48. Yang, L., Ma, Y. and Zhang, Y. (2023). *Measuring Consistency in Text-based Financial Forecasting Models*. doi: <https://doi.org/10.48550/arXiv.2305.08524>.
49. Yang, S. (2021). A novel study on deep learning framework to predict and analyze the financial time series information. *Future Generation Computer Systems*. doi: <https://doi.org/10.1016/j.future.2021.07.017>.
50. Zhang, G., Patuwo, E., B. and Y. Hu, M. (1998). Forecasting with artificial neural networks: *International Journal of Forecasting*, 14(1), pp.35–62. doi: <https://doi.org/10.1016/s0169-2070(97)00044-7>.
51. Zheng, X. (2023). Stock Price Prediction Based on CNN-BiLSTM Utilizing Sentiment Analysis and a Two-layer Attention Mechanism. *Advances in Economics Management and Political Sciences*, 47(1), pp.40–49. doi: <https://doi.org/10.54254/2754-1169/47/20230369>.

# Appendices

## LSTM Base and Feature Engineered Model Stock Price Prediction Code (Apple)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization

from keras.optimizers import Adam

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Apple Stock data

aapl\_data = yf.download('AAPL', start='2017-01-01', end='2024-01-01')

# Display the dataframe

aapl\_data

# Save the data to a CSV file

aapl\_data.to\_csv('applestockprice.csv')

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

aapl\_data.isnull().sum()

# Filling missing values

aapl\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

aapl\_data.info()

# Display descriptive statistics

aapl\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Close'], label='Closing Price')

plt.title('Apple Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(aapl\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Apple Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Apple Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = aapl\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Apple Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

aapl\_data['Daily Return'] = aapl\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = aapl\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Apple Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Daily Return'], label='Daily Return')

plt.title('Apple Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

aapl\_data['Month'] = aapl\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=aapl\_data, estimator=np.mean, errorbar=None)

plt.title('Apple Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

aapl\_data\_scaled = scaler.fit\_transform(aapl\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Apple stock data starting from the 60th data point to the end

for i in range(60, len(aapl\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(aapl\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(aapl\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base LSTM Model\*\*

## \*\*Building the LSTM Model\*\*

# Initialize a Sequential model

model = Sequential()

# Adding LSTM layers with return\_sequences

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(units=50, return\_sequences=True))

# Extract the number of time steps from the second and third dimension of X\_train

num\_timesteps = X\_train.shape[1]

num\_features = X\_train.shape[2]

# Define input

input\_layer = Input(shape=(num\_timesteps, num\_features))

# Permute and reshape for compatibility with attention

permute\_layer = Permute((2, 1))(input\_layer)

reshape\_layer = Reshape((num\_features, num\_timesteps))(permute\_layer)

# The attention mechanism

attention = AdditiveAttention(name='attention\_weight')([reshape\_layer, reshape\_layer])

# Multiply layer

multiply\_layer = Multiply()([reshape\_layer, attention])

# Return to original shape

permute\_back\_layer = Permute((2, 1))(multiply\_layer)

reshape\_back\_layer = Reshape((num\_timesteps, num\_features))(permute\_back\_layer)

# Adding a Flatten layer before the final Dense layer

flatten\_layer = Flatten()(reshape\_back\_layer)

# Adding Dropout and Batch Normalization

dropout\_layer = Dropout(0.2)(flatten\_layer)

batch\_norm\_layer = BatchNormalization()(dropout\_layer)

# Final Dense layer

output\_layer = Dense(1)(batch\_norm\_layer)

# Create the model

model = Model(inputs=input\_layer, outputs=output\_layer)

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the LSTM Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of LSTM Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (LSTM)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (LSTM)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*LSTM Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

aapl\_data['50\_MA'] = aapl\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

aapl\_data['100\_MA'] = aapl\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

aapl\_data['RSI'] = ta.momentum.RSIIndicator(close=aapl\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=aapl\_data['Close'])

aapl\_data['MACD'] = macd.macd()

aapl\_data['MACD\_Signal'] = macd.macd\_signal()

aapl\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=aapl\_data['Close'], window=20, window\_dev=2)

aapl\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

aapl\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

aapl\_data['Volume\_EMA'] = aapl\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

aapl\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(aapl\_data)

# Convert the scaled data back to a DataFrame for easier manipulation

scaled\_data = pd.DataFrame(scaled\_data, columns=aapl\_data.columns, index=aapl\_data.index)

# Preparing X and y datasets

X, y = [], []

sequence\_length = 60 # Number of previous time steps to include

for i in range(sequence\_length, len(scaled\_data)):

X.append(scaled\_data.iloc[i-sequence\_length:i].values)

y.append(scaled\_data.iloc[i, scaled\_data.columns.get\_loc("Close")])

X, y = np.array(X), np.array(y)

# Reshape X to be suitable for LSTM input

X = X.reshape((X.shape[0], X.shape[1], X.shape[2]))

## \*\*Building the LSTM Model\*\*

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X.shape[1], X.shape[2])))

model.add(LSTM(units=50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=25))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the LSTM Model\*\*

# Split data into training and testing sets

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test))

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of LSTM Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

num\_features = X\_train.shape[2]

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], num\_features))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# The last 60 days to predict the next day in order to reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate 15 features across 15 dimensions.

num\_features = 15

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 15

# Prepare the initial input batch

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Append the new prediction while keeping the feature dimension intact

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (LSTM)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 15

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (LSTM)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

## LSTM Base and Feature Engineered Model Stock Price Prediction Code (Microsoft)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization

from keras.optimizers import Adam

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Microsoft Stock data

msft\_data = yf.download('MSFT', start='2017-01-01', end='2024-01-01')

# Display the dataframe

msft\_data

# Save the data to a CSV file

msft\_data.to\_csv('microsoftstockprice.csv')

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

msft\_data.isnull().sum()

# Filling missing values

msft\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

msft\_data.info()

# Display descriptive statistics

msft\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Close'], label='Closing Price')

plt.title('Microsoft Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(msft\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Microsoft Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Microsoft Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = msft\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Microsoft Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

msft\_data['Daily Return'] = msft\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = msft\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Microsoft Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Daily Return'], label='Daily Return')

plt.title('Microsoft Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

msft\_data['Month'] = msft\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=msft\_data, estimator=np.mean, errorbar=None)

plt.title('Microsoft Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

msft\_data\_scaled = scaler.fit\_transform(msft\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Microsoft stock data starting from the 60th data point to the end

for i in range(60, len(msft\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(msft\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(msft\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base LSTM Model\*\*

## \*\*Building the LSTM Model\*\*

# Initialize a Sequential model

model = Sequential()

# Adding LSTM layers with return\_sequences

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(units=50, return\_sequences=True))

# Extract the number of time steps from the second and third dimension of X\_train

num\_timesteps = X\_train.shape[1]

num\_features = X\_train.shape[2]

# Define input

input\_layer = Input(shape=(num\_timesteps, num\_features))

# Permute and reshape for compatibility with attention

permute\_layer = Permute((2, 1))(input\_layer)

reshape\_layer = Reshape((num\_features, num\_timesteps))(permute\_layer)

# The attention mechanism

attention = AdditiveAttention(name='attention\_weight')([reshape\_layer, reshape\_layer])

# Multiply layer

multiply\_layer = Multiply()([reshape\_layer, attention])

# Return to original shape

permute\_back\_layer = Permute((2, 1))(multiply\_layer)

reshape\_back\_layer = Reshape((num\_timesteps, num\_features))(permute\_back\_layer)

# Adding a Flatten layer before the final Dense layer

flatten\_layer = Flatten()(reshape\_back\_layer)

# Adding Dropout and Batch Normalization

dropout\_layer = Dropout(0.2)(flatten\_layer)

batch\_norm\_layer = BatchNormalization()(dropout\_layer)

# Final Dense layer

output\_layer = Dense(1)(batch\_norm\_layer)

# Create the model

model = Model(inputs=input\_layer, outputs=output\_layer)

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the LSTM Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of LSTM Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Microsoft Stock Price with Predicted Next 4 Days (LSTM)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (LSTM)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*LSTM Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

msft\_data['50\_MA'] = msft\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

msft\_data['100\_MA'] = msft\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

msft\_data['RSI'] = ta.momentum.RSIIndicator(close=msft\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=msft\_data['Close'])

msft\_data['MACD'] = macd.macd()

msft\_data['MACD\_Signal'] = macd.macd\_signal()

msft\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=msft\_data['Close'], window=20, window\_dev=2)

msft\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

msft\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

msft\_data['Volume\_EMA'] = msft\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

msft\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(msft\_data)

# Convert the scaled data back to a DataFrame for easier manipulation

scaled\_data = pd.DataFrame(scaled\_data, columns=msft\_data.columns, index=msft\_data.index)

# Preparing X and y datasets

X, y = [], []

sequence\_length = 60 # Number of previous time steps to include

for i in range(sequence\_length, len(scaled\_data)):

X.append(scaled\_data.iloc[i-sequence\_length:i].values)

y.append(scaled\_data.iloc[i, scaled\_data.columns.get\_loc("Close")])

X, y = np.array(X), np.array(y)

# Reshape X to be suitable for LSTM input

X = X.reshape((X.shape[0], X.shape[1], X.shape[2]))

## \*\*Building the LSTM Model\*\*

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X.shape[1], X.shape[2])))

model.add(LSTM(units=50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=25))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the LSTM Model\*\*

# Split data into training and testing sets

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test))

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of LSTM Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

num\_features = X\_train.shape[2] # Number of features used during training

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], num\_features))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# The last 60 days to predict the next day in order to reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate 15 feature across 15 15 dimensions.

num\_features = 15

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 15

# Prepare the initial input batch

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Append the new prediction while keeping the feature dimension intact

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("MSFT Stock Price with Predicted Next 4 Days (LSTM)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 15

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (LSTM)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

## CNN Base and Feature Engineered Model Stock Price Prediction Code (Apple)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization, Conv1D, MaxPooling1D

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Apple Stock data

aapl\_data = yf.download('AAPL', start='2017-01-01', end='2024-01-01')

# Display the dataframe

aapl\_data

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

aapl\_data.isnull().sum()

# Filling missing values

aapl\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

aapl\_data.info()

# Display descriptive statistics

aapl\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Close'], label='Closing Price')

plt.title('Apple Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(aapl\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Apple Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Apple Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = aapl\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Apple Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

aapl\_data['Daily Return'] = aapl\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = aapl\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Apple Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Daily Return'], label='Daily Return')

plt.title('Apple Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

aapl\_data['Month'] = aapl\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=aapl\_data, estimator=np.mean, errorbar=None)

plt.title('Apple Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

aapl\_data\_scaled = scaler.fit\_transform(aapl\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Apple stock data starting from the 60th data point to the end

for i in range(60, len(aapl\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(aapl\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(aapl\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base CNN Model\*\*

## \*\*Building the CNN Model\*\*

# Define the CNN model

model = Sequential()

# Add convolutional layers

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

# Flatten the output of the convolutional layers

model.add(Flatten())

# Adding a Dropout layer

model.add(Dropout(0.2))

# Adding a Dense layer for the output

model.add(Dense(50, activation='relu'))

model.add(Dense(1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the CNN Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of CNN Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (CNN)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (CNN)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*CNN Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

aapl\_data['50\_MA'] = aapl\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

aapl\_data['100\_MA'] = aapl\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

aapl\_data['RSI'] = ta.momentum.RSIIndicator(close=aapl\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=aapl\_data['Close'])

aapl\_data['MACD'] = macd.macd()

aapl\_data['MACD\_Signal'] = macd.macd\_signal()

aapl\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=aapl\_data['Close'], window=20, window\_dev=2)

aapl\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

aapl\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

aapl\_data['Volume\_EMA'] = aapl\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

aapl\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

# Selecting the relevant features

features = ['Close', '50\_MA', '100\_MA', 'RSI', 'MACD', 'MACD\_Signal', 'MACD\_Hist', 'Bollinger\_High', 'Bollinger\_Low', 'Volume\_EMA']

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(aapl\_data[features])

# Prepare the sequences for training by using a 60-day lookback period

X = []

y = []

for i in range(60, len(scaled\_data)):

X.append(scaled\_data[i-60:i])

y.append(scaled\_data[i, 0]) # predicting the closing price

X, y = np.array(X), np.array(y)

## \*\*Building the CNN Model\*\*

# Define the CNN model

model = Sequential()

# Add convolutional layers

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(X.shape[1], X.shape[2])))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

# Flatten the output of the convolutional layers

model.add(Flatten())

# Adding a Dropout layer

model.add(Dropout(0.2))

# Adding a Dense layer for the output

model.add(Dense(50, activation='relu'))

model.add(Dense(1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the CNN Model\*\*

# Splitting the dataset 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)

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

num\_features = X\_train.shape[2]

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], num\_features))

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of CNN Model\*\*

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select the 'Close' price and convert to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scale the data

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# Use the last 60 days to predict the next day

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate across 10 dimensions to match the training

num\_features = 10

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Make predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 10

# Prepare the initial input batch with the correct number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Predict the next 4 days iteratively

predicted\_prices = []

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Append the new prediction while keeping the feature dimension intact

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (CNN)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 10

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (CNN)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

## CNN Base and Feature Engineered Model Stock Price Prediction Code (Microsoft)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization, Conv1D, MaxPooling1D

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Microsoft Stock data

msft\_data = yf.download('MSFT', start='2017-01-01', end='2024-01-01')

# Display the dataframe

msft\_data

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

msft\_data.isnull().sum()

# Filling missing values

msft\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

msft\_data.info()

# Display descriptive statistics

msft\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Close'], label='Closing Price')

plt.title('Microsoft Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(msft\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Microsoft Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Microsoft Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = msft\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Microsoft Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

msft\_data['Daily Return'] = msft\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = msft\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Microsoft Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Daily Return'], label='Daily Return')

plt.title('Microsoft Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

msft\_data['Month'] = msft\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=msft\_data, estimator=np.mean, errorbar=None)

plt.title('Microsoft Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

msft\_data\_scaled = scaler.fit\_transform(msft\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Microsoft stock data starting from the 60th data point to the end

for i in range(60, len(msft\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(msft\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(msft\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base CNN Model\*\*

## \*\*Building the CNN Model\*\*

# Define the CNN model

model = Sequential()

# Add convolutional layers

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

# Flatten the output of the convolutional layers

model.add(Flatten())

# Adding a Dropout layer

model.add(Dropout(0.2))

# Adding a Dense layer for the output

model.add(Dense(50, activation='relu'))

model.add(Dense(1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the CNN Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of CNN Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Microsoft Stock Price with Predicted Next 4 Days (CNN)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (CNN)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*CNN Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

msft\_data['50\_MA'] = msft\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

msft\_data['100\_MA'] = msft\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

msft\_data['RSI'] = ta.momentum.RSIIndicator(close=msft\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=msft\_data['Close'])

msft\_data['MACD'] = macd.macd()

msft\_data['MACD\_Signal'] = macd.macd\_signal()

msft\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=msft\_data['Close'], window=20, window\_dev=2)

msft\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

msft\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

msft\_data['Volume\_EMA'] = msft\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

msft\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

# Selecting the relevant features

features = ['Close', '50\_MA', '100\_MA', 'RSI', 'MACD', 'MACD\_Signal', 'MACD\_Hist', 'Bollinger\_High', 'Bollinger\_Low', 'Volume\_EMA']

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(msft\_data[features])

# Prepare the sequences for training (e.g., using a 60-day lookback period)

X = []

y = []

for i in range(60, len(scaled\_data)):

X.append(scaled\_data[i-60:i])

y.append(scaled\_data[i, 0]) # predicting the closing price

X, y = np.array(X), np.array(y)

## \*\*Building the CNN Model\*\*

# Define the CNN model

model = Sequential()

# Add convolutional layers

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(X.shape[1], X.shape[2])))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

# Flatten the output of the convolutional layers

model.add(Flatten())

# Adding a Dropout layer

model.add(Dropout(0.2))

# Adding a Dense layer for the output

model.add(Dense(50, activation='relu'))

model.add(Dense(1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the CNN Model\*\*

# Splitting the dataset 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)

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

num\_features = X\_train.shape[2]

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], num\_features))

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of CNN Model\*\*

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select the 'Close' price and convert to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scale the data

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# Use the last 60 days to predict the next day

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate across 10 dimensions to match the training

num\_features = 10

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Make predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 10

# Prepare the initial input batch with the correct number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Predict the next 4 days iteratively

predicted\_prices = []

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Append the new prediction while keeping the feature dimension intact

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Microsoft Stock Price with Predicted Next 4 Days (CNN)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 10

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (CNN)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

## GRU Base and Feature Engineered Model Stock Price Prediction Code (Apple)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization, Conv1D, MaxPooling1D, GRU

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Apple Stock data

aapl\_data = yf.download('AAPL', start='2017-01-01', end='2024-01-01')

# Display the dataframe

aapl\_data

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

aapl\_data.isnull().sum()

# Filling missing values

aapl\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

aapl\_data.info()

# Display descriptive statistics

aapl\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Close'], label='Closing Price')

plt.title('Apple Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(aapl\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Apple Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Apple Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = aapl\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Apple Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

aapl\_data['Daily Return'] = aapl\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = aapl\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Apple Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(aapl\_data['Daily Return'], label='Daily Return')

plt.title('Apple Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

aapl\_data['Month'] = aapl\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=aapl\_data, estimator=np.mean, errorbar=None)

plt.title('Apple Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

aapl\_data\_scaled = scaler.fit\_transform(aapl\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Apple stock data starting from the 60th data point to the end

for i in range(60, len(aapl\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(aapl\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(aapl\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base GRU Model\*\*

## \*\*Building the GRU Model\*\*

# Define the GRU model

model = Sequential()

# Add the first GRU layer with Dropout

model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

# Add the second GRU layer

model.add(GRU(units=50, return\_sequences=False))

model.add(Dropout(0.2))

# Add a Dense layer for the output

model.add(Dense(units=25, activation='relu'))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the GRU Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of GRU Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (GRU)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (GRU)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*GRU Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

aapl\_data['50\_MA'] = aapl\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

aapl\_data['100\_MA'] = aapl\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

aapl\_data['RSI'] = ta.momentum.RSIIndicator(close=aapl\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=aapl\_data['Close'])

aapl\_data['MACD'] = macd.macd()

aapl\_data['MACD\_Signal'] = macd.macd\_signal()

aapl\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=aapl\_data['Close'], window=20, window\_dev=2)

aapl\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

aapl\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

aapl\_data['Volume\_EMA'] = aapl\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

aapl\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

# Selecting the features to use for training

features = ['Close', '50\_MA', '100\_MA', 'RSI', 'MACD', 'Bollinger\_High', 'Bollinger\_Low', 'Volume\_EMA']

# Scale the selected features

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(aapl\_data[features])

# Scale the selected features

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(aapl\_data[features])

# Define X and y for the model, using a sliding window approach

X = []

y = []

for i in range(60, len(scaled\_data)):

X.append(scaled\_data[i-60:i]) # Using the last 60 time steps

y.append(scaled\_data[i, 0]) # Predicting the 'Close' price

X, y = np.array(X), np.array(y)

# Splitting into train and test sets

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

## \*\*Building the GRU Model\*\*

# Define the GRU model

model = Sequential()

# First GRU layer

model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dropout(0.2))

# Second GRU layer

model.add(GRU(units=50, return\_sequences=False))

model.add(Dropout(0.2))

# Output layer

model.add(Dense(units=25, activation='relu'))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the GRU Model\*\*

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of GRU Model\*\*

# Ensure X\_test has the correct shape

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], X\_train.shape[2]))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select the 'Close' price and convert to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scale the data

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# Use the last 60 days to predict the next day

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate across the correct number of dimensions to match the training

num\_features = 8

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Make predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of AAPL stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 8

timesteps = 60

# Prepare the initial input batch with the correct number of features and timesteps

# Replicating the closing price across the number of features

current\_batch = np.repeat(scaled\_data[-timesteps:].reshape(1, timesteps, 1), num\_features, axis=-1)

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Since next\_prediction is a scalar, expand it across the number of features

next\_prediction\_expanded = np.repeat(next\_prediction.reshape(1, 1), num\_features, axis=-1).reshape(1, 1, num\_features)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_expanded, axis=1)

# Inverse transform the first feature of the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction.reshape(-1, 1))[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Apple Stock Price with Predicted Next 4 Days (GRU)")

plt.show()

# Fetch the last 3 months of Apple stock data

data = yf.download('AAPL', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 8

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Apple Stock Price: Last 60 Days and Next 4 Days Predicted (GRU)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

## GRU Base and Feature Engineered Model Stock Price Prediction Code (Microsoft)

# \*\*Import Libraries\*\*

!pip install mplfinance -qqq

!pip install ta

import tensorflow as tf

import keras

import yfinance as yf

import mplfinance as mpf

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mpl\_dates

import ta

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, AdditiveAttention, Permute, Reshape, Multiply, Input, Flatten, BatchNormalization, Conv1D, MaxPooling1D, GRU

from tensorflow.keras.models import Model

# \*\*Loading the data\*\*

# Fetch Microsoft Stock data

msft\_data = yf.download('MSFT', start='2017-01-01', end='2024-01-01')

# Display the dataframe

msft\_data

# \*\*Data Cleaning and EDA\*\*

# Checking for missing values

msft\_data.isnull().sum()

# Filling missing values

msft\_data.fillna(method='ffill', inplace=True)

# Display basic information about the dataframe

msft\_data.info()

# Display descriptive statistics

msft\_data.describe()

# Plot the closing prices over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Close'], label='Closing Price')

plt.title('Microsoft Stock Closing Prices Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.legend()

plt.show()

# Plot the distribution of closing prices

plt.figure(figsize=(10, 5))

sns.histplot(msft\_data['Close'], bins=50, kde=True)

plt.title('Distribution of Microsoft Stock Closing Prices')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

# Plot the volume traded over time

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Volume'], label='Volume Traded', color='orange')

plt.title('Microsoft Stock Volume Traded Over Time')

plt.xlabel('Date')

plt.ylabel('Volume Traded')

plt.legend()

plt.show()

# Calculate monthly average closing prices

monthly\_avg\_close = msft\_data['Close'].resample('M').mean()

# Plot monthly average closing prices

plt.figure(figsize=(10, 5))

plt.plot(monthly\_avg\_close, label='Monthly Average Closing Price')

plt.title('Monthly Average Closing Prices of Microsoft Stock')

plt.xlabel('Date')

plt.ylabel('Average Closing Price')

plt.legend()

plt.show()

# Calculate daily returns

msft\_data['Daily Return'] = msft\_data['Close'].pct\_change()

# Calculate rolling standard deviation (volatility)

volatility = msft\_data['Daily Return'].rolling(window=30).std()

# Plot the volatility

plt.figure(figsize=(10, 5))

plt.plot(volatility, label='30-Day Rolling Volatility', color='orange')

plt.title('Microsoft Stock Volatility Over Time')

plt.xlabel('Date')

plt.ylabel('Volatility')

plt.legend()

plt.show()

# Plot daily returns

plt.figure(figsize=(10, 5))

plt.plot(msft\_data['Daily Return'], label='Daily Return')

plt.title('Microsoft Stock Daily Returns')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.legend()

plt.show()

# Extract the Month

msft\_data['Month'] = msft\_data.index.month

# Plot average closing price by month

plt.figure(figsize=(10, 5))

sns.barplot(x='Month', y='Close', data=msft\_data, estimator=np.mean, errorbar=None)

plt.title('Microsoft Average Closing Price by Month')

plt.xlabel('Month')

plt.ylabel('Average Closing Price')

plt.show()

# \*\*Data Preprocessing\*\*

# Create an instance of the MinMaxScaler class and fitting the scaler to the 'Close' column data and transforming it

scaler = MinMaxScaler(feature\_range=(0,1))

msft\_data\_scaled = scaler.fit\_transform(msft\_data['Close'].values.reshape(-1,1))

# Initialize empty lists to store the input sequences (X) and corresponding target values (y)

X = []

y = []

# Loop over the scaled Microsoft stock data starting from the 60th data point to the end

for i in range(60, len(msft\_data\_scaled)):

# Append the previous 60 data points to the X list as a sequence

# This creates a sliding window of 60 data points

X.append(msft\_data\_scaled[i-60:i, 0])

# Append the current data point as the target value to the y list

y.append(msft\_data\_scaled[i, 0])

# Determine the size of the training set as 80% and test set as 20% of the total dataset

train\_size = int(len(X) \* 0.8)

test\_size = len(X) - train\_size

# Split the input sequences (X) into training and test sets

X\_train, X\_test = X[:train\_size], X[train\_size:]

# Split the target values (y) into training and test sets

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Convert the lists X\_train and y\_train to numpy arrays

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshape the X\_train array to have 3 dimensions

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# \*\*Base GRU Model\*\*

## \*\*Building the GRU Model\*\*

# Define the GRU model

model = Sequential()

# Add the first GRU layer with Dropout

model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

# Add the second GRU layer

model.add(GRU(units=50, return\_sequences=False))

model.add(Dropout(0.2))

# Add a Dense layer for the output

model.add(Dense(units=25, activation='relu'))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Compile the model

# MSE is commonly used for regression problems where we want to minimize the difference between predicted and actual values

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Print a summary of the model architecture

model.summary()

## \*\*Training the GRU Model\*\*

# Train the model using the training data

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

## \*\*Evaluating the Performance of GRU Model\*\*

# Convert X\_test and y\_test to Numpy arrays

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Reshape X\_test to match the input shape used for X\_train

# This ensures that the test data has the same dimensionality as the training data

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetching the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Selecting the 'Close' price and converting to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scaling the data

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1,1))

# Since we need the last 60 days to predict the next day, we reshape the data accordingly

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Reshaping the data for the model (adding batch dimension)

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], 1))

# Making predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price)

print("Predicted Stock Price for the next day: ", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Most recent 60 days

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Reshape the prediction to fit the batch dimension

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

# Append the prediction to the batch used for predicting

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Inverse transform the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Microsoft Stock Price with Predicted Next 4 Days (GRU)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

current\_batch = scaled\_data[-60:].reshape(1, 60, 1) # Last 60 days

# Predicting 4 days

for i in range(4):

next\_prediction = model.predict(current\_batch)

next\_prediction\_reshaped = next\_prediction.reshape(1, 1, 1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Adding predictions to the DataFrame

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Combining both actual and predicted data

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

combined\_data = combined\_data[-64:] # Last 60 days of actual data and 4 days of predictions

# Plotting the actual data

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

# Plotting the predicted data

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (GRU)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()

# \*\*GRU Model with Feature Engineering\*\*

## \*\*Calculating Technical Indicators\*\*

# 50-day Moving Average

msft\_data['50\_MA'] = msft\_data['Close'].rolling(window=50).mean()

# 100-day Moving Average

msft\_data['100\_MA'] = msft\_data['Close'].rolling(window=100).mean()

# Relative Strength Index (RSI)

msft\_data['RSI'] = ta.momentum.RSIIndicator(close=msft\_data['Close'], window=14).rsi()

# MACD

macd = ta.trend.MACD(close=msft\_data['Close'])

msft\_data['MACD'] = macd.macd()

msft\_data['MACD\_Signal'] = macd.macd\_signal()

msft\_data['MACD\_Hist'] = macd.macd\_diff()

# Bollinger Bands

bollinger = ta.volatility.BollingerBands(close=msft\_data['Close'], window=20, window\_dev=2)

msft\_data['Bollinger\_High'] = bollinger.bollinger\_hband()

msft\_data['Bollinger\_Low'] = bollinger.bollinger\_lband()

# Volume Exponential Moving Average (Volume\_EMA)

msft\_data['Volume\_EMA'] = msft\_data['Volume'].ewm(span=21).mean()

# Drop any NaN values created during feature engineering

msft\_data.dropna(inplace=True)

## \*\*Data Preprocessing\*\*

# Selecting the features to use for training

features = ['Close', '50\_MA', '100\_MA', 'RSI', 'MACD', 'Bollinger\_High', 'Bollinger\_Low', 'Volume\_EMA']

# Scale the selected features

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(msft\_data[features])

# Scale the selected features

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(msft\_data[features])

# Define X and y for the model, using a sliding window approach

X = []

y = []

for i in range(60, len(scaled\_data)):

X.append(scaled\_data[i-60:i]) # Using the last 60 time steps

y.append(scaled\_data[i, 0]) # Predicting the 'Close' price

X, y = np.array(X), np.array(y)

# Splitting into train and test sets

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

## \*\*Building the GRU Model\*\*

# Define the GRU model

model = Sequential()

# First GRU layer

model.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dropout(0.2))

# Second GRU layer

model.add(GRU(units=50, return\_sequences=False))

model.add(Dropout(0.2))

# Output layer

model.add(Dense(units=25, activation='relu'))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

## \*\*Training the GRU Model\*\*

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=25, validation\_split=0.2)

# Print a summary of the model architecture

model.summary()

## \*\*Evaluating the Performance of GRU Model\*\*

# Ensure X\_test has the correct shape

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], X\_train.shape[2]))

# Evaluate the model on the test data

test\_loss = model.evaluate(X\_test, y\_test)

# Print the test loss to see the performance of the model on the test data

print("Test Loss: ", test\_loss)

# Making predictions

y\_pred = model.predict(X\_test)

# Calculating MAE and RMSE

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

# Print the MAE and RMSE

print("Mean Absolute Error: ", mae)

print("Root Mean Square Error: ", rmse)

## \*\*Predicting the next 4 days\*\*

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select the 'Close' price and convert to numpy array

closing\_prices = data['Close'].values

# Ensure we have at least 60 days of data

if len(closing\_prices) < 60:

raise ValueError("Not enough data to make a prediction")

# Scale the data

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices.reshape(-1, 1))

# Use the last 60 days to predict the next day

X\_latest = np.array([scaled\_data[-60:].reshape(60)])

# Replicate across the correct number of dimensions to match the training

num\_features = 8

X\_latest = np.repeat(X\_latest[..., np.newaxis], num\_features, axis=-1)

# Ensure X\_latest has the correct shape and data type

X\_latest = np.reshape(X\_latest, (X\_latest.shape[0], X\_latest.shape[1], num\_features))

X\_latest = X\_latest.astype('float32')

# Make predictions for the next day

predicted\_stock\_price = model.predict(X\_latest)

predicted\_stock\_price = scaler.inverse\_transform(predicted\_stock\_price[:, 0].reshape(-1, 1))

print("Predicted Stock Price for the next day:", predicted\_stock\_price)

# Fetch the latest 60 days of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

# Predict the next 4 days iteratively

predicted\_prices = []

num\_features = 8

timesteps = 60

# Prepare the initial input batch with the correct number of features and timesteps

# Replicating the closing price across the number of features

current\_batch = np.repeat(scaled\_data[-timesteps:].reshape(1, timesteps, 1), num\_features, axis=-1)

for i in range(4): # Predicting 4 days

# Get the prediction (next day)

next\_prediction = model.predict(current\_batch)

# Since next\_prediction is a scalar, expand it across the number of features

next\_prediction\_expanded = np.repeat(next\_prediction.reshape(1, 1), num\_features, axis=-1).reshape(1, 1, num\_features)

# Update the current batch by removing the oldest data point and adding the new prediction

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_expanded, axis=1)

# Inverse transform the first feature of the prediction to the original price scale

predicted\_prices.append(scaler.inverse\_transform(next\_prediction.reshape(-1, 1))[0, 0])

print("Predicted Stock Prices for the next 4 days: ", predicted\_prices)

# Creating a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# list of predicted prices for the next 4 days

predictions\_df = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

# Plotting the actual data with mplfinance

mpf.plot(data, type='candle', style='charles', volume=True)

# Overlaying the predicted data

plt.figure(figsize=(10,5))

plt.plot(predictions\_df.index, predictions\_df['Close'], linestyle='dashed', marker='o', color='red')

plt.title("Microsoft Stock Price with Predicted Next 4 Days (GRU)")

plt.show()

# Fetch the last 3 months of Microsoft stock data

data = yf.download('MSFT', period='3mo', interval='1d')

# Select 'Close' price and scale it

closing\_prices = data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(closing\_prices)

num\_features = 8

# Repeat the single feature across the expected number of features

current\_batch = np.repeat(scaled\_data[-60:].reshape(1, 60, 1), num\_features, axis=-1)

# Now predict the next 4 days iteratively

predicted\_prices = []

for i in range(4):

next\_prediction = model.predict(current\_batch)

predicted\_prices.append(scaler.inverse\_transform(next\_prediction)[0, 0])

# Update current\_batch by adding the new prediction and removing the oldest data point

next\_prediction\_reshaped = np.repeat(next\_prediction.reshape(1, 1, 1), num\_features, axis=-1)

current\_batch = np.append(current\_batch[:, 1:, :], next\_prediction\_reshaped, axis=1)

# Create a list of dates for the predictions

last\_date = data.index[-1]

next\_day = last\_date + pd.Timedelta(days=1)

prediction\_dates = pd.date\_range(start=next\_day, periods=4)

# Combine predictions with the actual data

predicted\_data = pd.DataFrame(index=prediction\_dates, data=predicted\_prices, columns=['Close'])

combined\_data = pd.concat([data['Close'], predicted\_data['Close']])

# Plot the results

plt.figure(figsize=(10,5))

plt.plot(data.index[-60:], data['Close'][-60:], linestyle='-', marker='o', color='blue', label='Actual Data')

plt.plot(prediction\_dates, predicted\_prices, linestyle='-', marker='o', color='red', label='Predicted Data')

plt.title("Microsoft Stock Price: Last 60 Days and Next 4 Days Predicted (GRU)")

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()