TRANSFORMER-BASED PREDICTION MODEL FOR STOCK MARKET ANALYSIS

Padala Kalyani

2023005957

Addada Sai Keerthana

2023001219

School of Science

GITAM Deemed to University

Hyderabad Campus

M. Sc. Data Science 2025

TRANSFORMER-BASED PREDICTION MODEL FOR STOCK MARKET ANALYSIS

Name of Student: Padala Kalyani

ID No. of Student: 2023005957

Name of Student: Addada Sai Keerthana

ID No. of Student: 2023001219

Submitted in Partial fulfilment of the Requirements for the

Degree of Master of Science in Data Science

at GITAM Deemed to be University

Under Supervisor

Dr. Motahar Reza

School of Sciences

GITAM Deemed to be University

Rudraram, Hyderabad Campus

April 2025

Declaration and Approval

Declaration

I certify that this dissertation/Major Project is my own original work and has not been previously submitted or approved for the award of a degree at this or any other university. To the best of my knowledge, it does not include any material authored or published by others, except where appropriate citations and references are given.

Name of Student: Padala Kalyani	Name of Student: Addada Sai Keerthana
[Signature] [Date]	[Signature] [Date]

Approval

The dissertation of Padala Kalyani and Addada Sai Keerthana was reviewed and approved for examination by the following:

Dr. Motahar Reza,

Department of Mathematics

School of Sciences,

GITAM University, Hyderabad Campus

External Expert,

Principal,

School of Science,

GITAM University

Acknowledgements

I would like to express my sincere gratitude to my project supervisor, **Dr. Motahar Reza**, for his invaluable guidance, constant support, and encouragement throughout the course of my major project. His expertise and insights have played a crucial role in shaping this work, and I am truly grateful for his mentorship.

I extend my heartfelt thanks to **Dr. Motahar Reza**, Principal, School of Science, GITAM University, Hyderabad, for providing a stimulating academic environment and unwavering support. I am also deeply grateful to **Dr. Abhijit Patil**, Project Coordinator, for his assistance, constructive feedback, and motivation, which have been instrumental in the successful completion of this project.

I would also like to acknowledge **Prof. D. S. Rao**, Pro Vice-Chancellor, GITAM Hyderabad Campus, for his continuous support and for fostering an environment that encourages academic excellence and research.

I am grateful to my **professors and mentors**, whose teachings and guidance have been fundamental in shaping my knowledge and skills in Data Science. Their support has been invaluable in my academic journey.

Furthermore, I am profoundly thankful to my **parents and family** for their unconditional love, encouragement, and sacrifices, which have been my greatest source of strength. A special note of appreciation goes to my **friends** for their continuous support, insightful discussions, and encouragement, making this journey more enriching and memorable.

Finally, I express my gratitude to everyone who has contributed to my learning experience at the School of Science, GITAM University, Hyderabad. Their support has been instrumental in the successful completion of this project.

Padala Kalyani
2023005957
M.Sc. Data Science
GITAM Deemed University

Addada Sai Keerthana
2023001219
M.Sc. Data Science
GITAM Deemed University

Abstract

Stock market prediction is a key area of research because of its impact on investment strategies, financial planning and risk management. The complex patterns and volatility in stock price movements make forecasting a challenging task and require a progressive analytical approach. Traditional statistical models find it difficult to capture nonlinear dependencies in financial time series, requires the use of deep learning techniques.

This Project analyses 10 years of historical Nifty 50 index data and performs time series analysis to understand market behaviour. We explored variety of deep learning models, including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer architectures, to assess their effectiveness in predicting stock prices.

To enhance accessibility and usability, we have developed a mobile application that gives nifty50 close price predictions with market sentiment in a user-friendly manner. This project contributes to the growth field of financial forecasting with deep learning foundation, and provides insights into the applicability of various neural network architectures for stock market analysis.

Keywords: Financial Forecasting, Stock Market Prediction, Time Series Analysis, Nifty 50, Deep Learning, RNN, LSTM, Transformers

Table of Contents

List of Tables

List Of Figures

List of Abbreviations

1 **NIFTY 50** – National Stock Exchange Fifty

2 **RNN** – Recurrent Neural Network

3 **LSTM** – Long Short-term Memory

4 **ARIMA** – Autoregressive Integrated Moving Average

5 **SARIMA** – Seasonal Autoregressive Integrated Moving Average

6 **DNN** – Deep Neural networks

7 **EDA** – Exploratory Data Analysis

8 **ACF** – Autocorrelation Function

9 **PACF** – Partial Autocorrelation Function

1 Introduction

The stock market plays a critical role in the global economy, influencing business decisions, investor confidence, and economic policy. Predicting stock prices has long been a topic of critical importance due to its potential to guide informed investment strategies, reduce financial risk, and enhance portfolio management. Accurate stock price prediction enables investors and financial institutions to make timely decisions, manage risks more effectively, and allocate resources more efficiently. In a market environment driven by uncertainty, being able to forecast price movements even marginally better than chance can lead to significant financial gains and strategic advantages.

However, stock market forecasting remains one of the most challenging tasks in financial analysis. The behavior of financial markets is inherently complex, non-linear, and affected by multiple factors including macroeconomic indicators, company performance, political events, and public sentiment. Traditional statistical forecasting methods such as ARIMA, SARIMA and GARCH, while useful in certain contexts, are generally inadequate in capturing the intricate, long-term dependencies and irregular patterns present in stock price movements. These limitations have led to a growing interest in deep learning approaches that can process large volumes of historical data and learn complex patterns without extensive manual intervention.

To address these challenges, this project investigates the application of deep learning techniques to stock price forecasting, focusing on three widely used architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer-based models. These models are known for their ability to capture sequential patterns in time series data, with each offering unique benefits. RNNs are suited for modelling temporal dependencies, LSTMs effectively address long-range memory issues, and Transformers leverage self-attention mechanisms to handle long-term relationships and parallel processing efficiently. While these architectures have been individually explored in various studies, few works offer a consistent and direct comparison of their predictive capabilities using a standardized dataset and evaluation framework.

This project aims to address that gap by comparing the performance of RNN, LSTM, and Transformer models using 10 years of historical data from the Nifty 50 index. Instead of relying solely on pre-existing implementations, we undertook a comprehensive evaluation process involving the design, configuration, and fine-tuning of each model's architecture and hyperparameters to suit the characteristics of financial time series data. All models were trained and assessed under consistent conditions to ensure a fair and unbiased comparison. This methodical and hands-on approach provided both enhanced control over the experimental framework and valuable insights into the relative strengths and limitations of each architecture. While the models themselves are well-established in previous research, our structured and unified methodology offers a fresh perspective on their comparative effectiveness in the domain of stock price forecasting.

Additionally, sentiment analysis on news data related to the Nifty 50 index was conducted using a pretrained FinBERT model. While this sentiment analysis was not integrated into the predictive models, it was used to enhance the interpretability and user experience within a mobile application developed as part of the project. This application presents the predicted closing prices alongside sentiment insights, offering users a more contextual understanding of the market environment.

This project contributes to the field of financial forecasting by providing a detailed comparison of modern deep learning models in stock market prediction. The findings offer valuable insights that can help develop more reliable forecasting tools and guide analysts, developers, and investors in applying predictive deep learning to stock market analysis.

1.1 Background

Stock market prediction remains a crucial area of study due to its importance in financial risk management, investment planning, and strategic decision-making. Accurate forecasting of stock prices empowers investors, traders, and financial institutions to optimize asset allocation, mitigate risks, and prevent potential financial losses. However, the stock market is known for its volatility, complexity, and nonlinear behaviour driven by a range of external influences such as economic indicators, political developments, investor sentiment, and speculative activities which makes precise prediction a significant challenge.

Traditional statistical forecasting models like Autoregressive Integrated Moving Average (ARIMA), Seasonal ARIMA (SARIMA), and Exponential Smoothing have been commonly used for time series analysis. While these approaches are effective for modelling linear patterns and short-term dependencies, they often fall short when applied to financial time series data due to their limitations in handling complex and long-range temporal patterns.

With the advancement of machine learning and increased computational capabilities, deep learning techniques have become prominent alternatives for time series forecasting. Models such as Recurrent Neural Networks (RNNs) and their enhanced variant, Long Short-Term Memory (LSTM) networks, have shown strong performance in capturing sequential patterns and learning from large datasets. More recently, Transformer-based models originally designed for natural language processing have achieved remarkable success in time series forecasting. Their self-attention mechanisms and scalability allow them to capture both short-term variations and long-term trends, making them well-suited for modelling the intricate dynamics of stock prices.

This project investigates and compares the performance of RNN, LSTM, and Transformer models in forecasting the closing prices of the Nifty 50 index. Each model is evaluated under consistent experimental settings, with careful tuning of architectures and hyperparameters to highlight their strengths and limitations in stock price prediction. Additionally, sentiment analysis is performed using a pretrained FinBERT model to provide supplementary insights into market conditions. Although this sentiment data is not directly used in training the prediction models, it enriches the interpretability of results within a mobile application.

The insights from this study aim to support the advancement of predictive analytics in financial markets, enabling analysts, investors, and developers to harness the potential of deep learning models for more informed stock market forecasting and investment strategies.

1.2 Problem Statement

The stock market is known for its high volatility and sensitivity to a variety of factors, including economic data, political developments, and investor sentiment. Traditional forecasting techniques such as ARIMA and SARIMA are frequently used for stock price prediction; however, these statistical models struggle to effectively capture the complex, nonlinear, and ever-changing behaviour of financial markets. As a result, their predictions are often inaccurate or delayed, limiting their usefulness for real-time investment decisions.

To address these challenges, this project leverages advanced deep learning approaches including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer-based models which are designed to recognize and learn intricate temporal patterns from historical stock data. These architectures are applied to predict the next day's closing price of the Nifty 50 index with improved precision.

To make these forecasts easily accessible, a user-friendly mobile application has also been developed. This app provides real-time Nifty 50 predictions and integrates sentiment analysis to assist novice investors in making better-informed decisions through a clean and intuitive interface.

1.3 Research Objectives

This project focuses on creating an effective stock price prediction system using advanced deep learning methods. The primary goal is to predict the next-day closing price of the Nifty 50 index by analysing historical stock market data. It involves evaluating and comparing the performance of various deep learning models, including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer-based architectures, for time series forecasting tasks. A key objective is to determine which model best captures the nonlinear trends and long-term dependencies inherent in financial time series.

Another important aspect of the project is the development of a user-friendly mobile application that presents real-time stock predictions in a clear and accessible way. To further improve the accuracy and relevance of the predictions, the system incorporates sentiment analysis from market-related news. By combining robust predictive models with an intuitive interface, the project aims to support better decision-making for beginners for better investment decisions.

1.4 Research Questions

The purpose of this project is to answer the following questions

1. How do time series data analysis be performed for stock predictions?
2. How to predict next day closing prices of nifty50 using deep learning model?
3. When predicting the stock market, which factors influence the performance of the deep learning model?
4. What are the advantages and limitations of RNN, LSTM and transformer-based models for predicting stock prices?
5. How to design mobile application for effective user experience?

1.5 Justification

Predicting stock market trends is essential for shaping investment strategies, managing risk, and planning finances. However, the stock market's inherently volatile, dynamic, and nonlinear nature makes accurate prediction a complex challenge. Traditional time series models such as ARIMA and SARIMA can effectively capture linear patterns and seasonal trends, but they often struggle to adapt to abrupt market fluctuations or uncover deeper, nonlinear relationships in financial data. This limitation leads to reduced forecasting accuracy and limits their usefulness in real-world scenarios.

With the evolution of deep learning, advanced models like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer-based frameworks have shown strong potential in analysing sequential data and learning complex temporal patterns. While these models offer promising capabilities, there is still a need to systematically evaluate their performance in financial forecasting especially for stock indices like the Nifty 50 to identify the most suitable approach.

This project aims to fill that gap by conducting a comparative analysis of these deep learning models for next-day stock price prediction. Additionally, the project includes the development of a user-friendly mobile application that translates these technical forecasts into an accessible format. This ensures that even users without a technical background can interpret market trends and make informed investment decisions. By incorporating model with real-world usability, the project adds value both in academic research and practical financial applications.

1.6 Scope

This project centers on building a stock price prediction system designed to forecast the next-day closing value of the Nifty 50 index using deep learning techniques. The work involves performing time series analysis on historical market data and implementing sophisticated neural network models, including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer-based architectures. These models are assessed using key evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to determine which approach delivers the most reliable predictive performance.

In addition to model development and evaluation, the project also includes the creation of a mobile application that presents the predictions through an intuitive and easy-to-use interface.

This app not only displays daily forecasted closing prices for the Nifty 50 index but also offers simplified market insights to help users better interpret the predictions. Tailored for beginners, the application serves as both an educational tool and a decision-support system, making complex stock market data more understandable and actionable.

By integrating advanced deep learning models with a practical, user-oriented application, this project aims to close the gap between technical financial forecasting and everyday usability.

1.7 Limitations

While this project utilizes advanced deep learning models, it is important to acknowledge several limitations. The models have been trained on 10 years of historical data from the Nifty 50 index. Although this timeframe is generally adequate for identifying trends, it may not encompass rare events or very long-term market shifts. Furthermore, the models rely solely on past price data, which limits their ability to incorporate external influences such as macroeconomic developments, political disruptions, or company-specific news factors that often drive significant market movements.

Although deep learning architectures like RNNs, LSTMs, and Transformers are capable of recognizing complex temporal patterns, they may still face difficulties in handling abrupt market changes or unpredictable events that fall outside learned historical behaviour. The underlying assumption that past trends are indicative of future performance presents an inherent constraint.

Another limitation is the use of daily, publicly available data, without access to high-frequency or alternative datasets such as institutional trading signals or detailed sentiment analysis data sources that could potentially enhance prediction accuracy.

The mobile application developed as part of this project is limited in scope it provides only next-day closing price forecasts for the Nifty 50 and does not offer real-time updates or support for active trading strategies. It is intended primarily as an educational tool rather than a professional financial advisory platform.

2 Literature review

2.1 Study 1: Transformer-Based Models for Stock Prediction

Transformer-based models have gained significant attention in stock market prediction due to their ability to capture long-term dependencies and handle complex temporal data. Early studies like Wang et al. [1] showed that Transformer models, when applied to stock indices like the CSI 300 and S&P 500, outperformed traditional models such as CNNs and RNNs, highlighting their ability to learn intricate temporal patterns. This advantage was further demonstrated in work by Zhang et al. [2], who introduced TEANet, a model that combined both numerical stock data and textual sentiment information, utilizing attention mechanisms to improve predictive accuracy. Their findings indicated that incorporating real-time sentiment data enhanced the model's forecasting capabilities, showing that Transformers are effective in integrating diverse sources of information. Building on these advancements, Yañez et al. [3] incorporated frequency decomposition techniques like Empirical Mode Decomposition (EMD) with Transformers, allowing the model to focus on long-term trends while filtering out noise from high-volatility data. This hybrid approach further validated the Transformer's superiority over traditional models, such as LSTM, in forecasting volatile financial markets. Meanwhile, Gothai et al. [4] extended Transformer applications to global stock markets, demonstrating the model's versatility across different market conditions. Their results confirmed that Transformer models could capture both short-term fluctuations and long-term trends more effectively than other machine learning methods.

Further improvements to Transformer models were seen with the introduction of the Temporal Fusion Transformer (TFT) by Hu [5], which integrated both static and dynamic features for multi-horizon forecasting. This model was able to adapt to varying temporal trends, outperforming other models like LSTM and Support Vector Regression (SVR) in terms of predictive accuracy, particularly in volatile stock market conditions. This highlights the Transformer's ability to forecast not only short-term movements but also long-term price trends. Similarly, Sikarwar et al. [6] explored the application of Transformer models directly for stock price prediction and found that they significantly outperformed traditional RNN-based models, particularly when capturing long-term dependencies. Their research further reinforced the conclusion that Transformers, with their unique attention mechanisms, provide a more efficient method for modeling sequential data in the context of stock prices.

Taken together, these studies provide compelling evidence that Transformer models are highly effective for stock market prediction. Their ability to incorporate various data sources, capture long-term dependencies, and adapt to different market conditions makes them an excellent choice for financial forecasting. As research progresses, further improvements in Transformer architectures, such as hybrid models and multi-source data integration, are likely to enhance their predictive capabilities and expand their practical applications in the financial sector.

2.2 Study 2: LSTM and RNN-Based Models

Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN), have been widely adopted for stock market forecasting due to their ability to capture long-term dependencies in time-series data. A study by Moghar et al. [7] explored the use of LSTM for predicting stock market prices, finding that LSTM networks can effectively model the temporal patterns inherent in financial data. The researchers noted that increasing the number of training epochs improved model performance up to a certain point, after which overfitting became an issue, emphasizing the importance of carefully tuning training parameters. This finding was consistent with Bhandari et al. [8], who also utilized LSTM networks for predicting stock market indices. Their research demonstrated that LSTMs outperformed traditional statistical models, with the ability to capture the intricate dynamics of stock market trends, showcasing the effectiveness of LSTM in time-series forecasting tasks.

Similarly, Fathali et al. [9], evaluated various machine learning models, including LSTMs, for predicting the NIFTY 50 index in India. Their results showed that LSTM networks outperformed both RNNs and CNNs, making them especially effective for predicting time series data that require the modeling of long-term dependencies. This finding was also echoed by Jadoon et al. [10], who applied LSTM to forecast the Karachi Stock Exchange (KSE) 100 index. By integrating diverse economic and political indicators, the authors achieved an impressive 99% accuracy in predicting the stock index’s movements. This success highlights the LSTM model's ability to incorporate multiple factors influencing market behaviour, reinforcing its versatility in financial forecasting.

Further supporting these findings, Mehtab et al. [11] demonstrated the combination of machine learning and deep learning models, with LSTMs yielding superior performance for predicting the NIFTY 50 index. Their study showed that univariate LSTM models, which utilized a week's worth of prior data, were especially accurate in forecasting the next week's stock prices. This highlights the LSTM model’s efficiency in handling historical data to predict future trends. Moreover, Jafar et al. [12] incorporated backward elimination feature selection with LSTM models to predict NIFTY 50 prices, achieving a prediction accuracy of 95%. Their study emphasized the importance of feature selection in improving LSTM model performance, suggesting that eliminating irrelevant features can enhance forecasting precision.

These studies collectively underscore the effectiveness of LSTM and RNN models in stock market prediction, with LSTMs consistently outperforming traditional machine learning techniques. Their ability to capture long-term dependencies in sequential data, combined with innovations in feature selection and model optimization, makes them a powerful tool for time-series forecasting in financial markets.

2.3 Study 3: Hybrid Deep Learning Models

In recent years, hybrid deep learning models have gained considerable traction in stock price prediction due to their ability to leverage the complementary strengths of different architectures. Wang et al. [13] introduced the BiLSTM-MTRAN-TCN model, which combines Bidirectional LSTM, a Modified Transformer, and Temporal Convolutional Networks to effectively capture both past and future trends, handle long-range dependencies, and extract key temporal features. This model achieved a 24.3% reduction in RMSE, highlighting its robustness across diverse market scenarios. Li and Qian [14] proposed FDG-Trans, which addresses noise and subtle fluctuations in stock data by applying CEEMD for signal decomposition, followed by a Transformer encoder and sequential layers like GRU and LSTM for improved forecasting. This model demonstrated superior performance over other hybrids such as DeepLOB and DeepAtt, especially in high-frequency trading environments. Daiya and Lin [15] further advanced hybrid modeling with Trans-DiCE, a multimodal framework that fuses time-series and news sentiment data using DC-CNN and Transformer encoders, respectively. The model achieved a notable 77.74% accuracy and showed enhanced portfolio returns when integrated with Deep Reinforcement Learning. Collectively, these models exemplify the effectiveness of hybrid deep learning in addressing key challenges such as noise reduction, temporal dependency modeling, and multimodal data integration, leading to more accurate, adaptive, and generalizable financial forecasting systems.

2.4 Study 4: Sentiment Aware Forecasting Models

In the field of sentiment-aware forecasting models, integrating sentiment analysis with deep learning frameworks has significantly improved the accuracy of stock price predictions. Zhang et al. [16] proposed a Sentiment-Guided Conditional Generative Adversarial Network (CGAN), where an LSTM-based generator models temporal dependencies in stock prices, and an MLP-based discriminator distinguishes real from generated sequences. This model incorporates sentiment scores from Twitter using the VADER tool with enhanced sentiment lists, guiding the adversarial learning process. Compared to traditional models like Linear Regression and standalone LSTM, this approach showed marked improvements in Mean Absolute Percentage Error (MAPE). Similarly, Daradkeh [17] developed a Hybrid Data Analytics Framework combining CNN and BiLSTM to merge quantitative financial data with qualitative sentiment from news, demonstrating a notable boost in prediction accuracy for stocks such as ALDAR and ETISALAT from the Dubai Financial Market.

Ko and Chang [18] expanded on this by employing the BERT model for sentiment analysis of news articles, integrating the sentiment scores into an LSTM-based forecasting framework. Their method yielded a 12.05% improvement in RMSE over models lacking sentiment input, reinforcing the importance of blending technical indicators with sentiment analysis. Building on this approach, Gite et al. [19] used sentiment from financial news headlines with an LSTM model and enhanced their system with Explainable AI (XAI) techniques. This not only improved predictive accuracy but also offered greater model interpretability, an essential factor for investor trust. These studies collectively highlight the power of combining sentiment-aware components with deep learning to achieve more accurate and transparent stock price forecasts.

2.5 Study 5: Volatility Prediction and Comparative Studies

Forecasting stock market volatility and trends has attracted considerable research attention, with various models offering distinct advantages. Mahajan et al. [20] compared GARCH and RNN-based LSTM models for predicting NIFTY 50 volatility, finding that asymmetric GARCH variants like EGARCH and TARCH slightly outperformed LSTM in capturing actual volatility due to their ability to model asymmetry and leverage effects. In contrast, Bansal et al. [21] used machine learning models including Random Forest, SVM, and ANN for trend prediction, with Random Forest delivering the highest accuracy by leveraging key technical indicators like moving averages and RSI.

Building on this, Ma et al. [22] combined LSTM and CNN with portfolio optimization, showing that deep learning models not only improved return predictions but also enhanced portfolio performance through better asset allocation. Collectively, these studies highlight how traditional econometric models like GARCH are effective for volatility forecasting, while machine and deep learning models excel in trend prediction and investment optimization suggesting a hybrid approach could provide more comprehensive and accurate financial forecasting.

3 Methodology

3.1 Development Methodology

The development of this project followed a two-phased approach combining traditional time series methods and modern deep learning techniques. In first phase, we collected and preprocessed 10 years of NIFTY 50 historical data, applied exploratory analysis, and built baseline models like ARIMA, SARIMA, and RNNs to understand data patterns. In second phase, advanced models such as LSTM and Transformers were developed, with LSTM showing the best performance using a 60-day input sequence. After optimization and evaluation using MAE and RMSE, the best-performing LSTM model was deployed in a mobile app that provides real-time next-day price predictions and market insights to users.

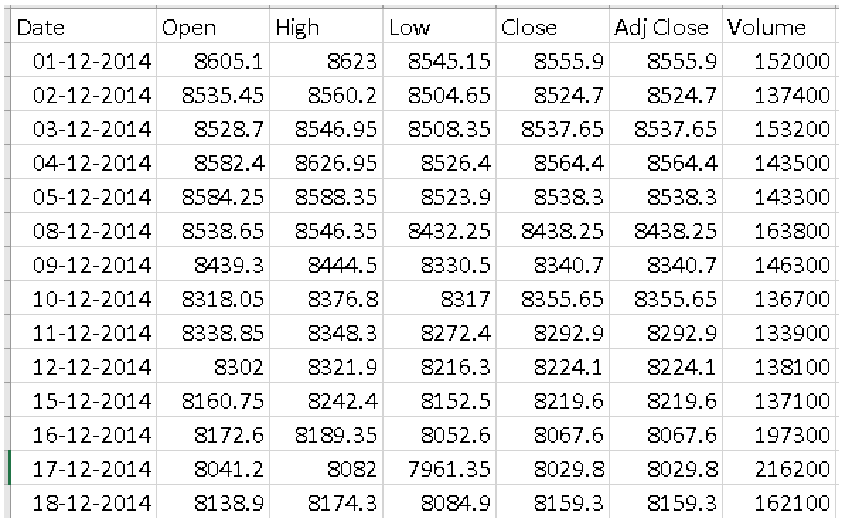
3.1.1 Phase 1: Data Preparation and Initial Modelling

a. Data Collection

We collected 10 years of historical stock market data for the NIFTY 50 index from December 1, 2014, to November 29, 2024. This dataset was Collected from yahoo Finance[23] using yfinance library.

The key attributes included are as follows:

- **Open:** The initial price of the stock index at the start of the trading day.
- **High:** The peak price achieved during the trading session.
- **Low:** The lowest price recorded throughout the trading day.
- **Close:** The final price at the conclusion of the trading session.
- **Adjusted Close:** The closing price modified to account for dividends and stock splits.
- **Volume:** The total number of shares exchanged on that particular day.



Date	Open	High	Low	Close	Adj Close	Volume
01-12-2014	8605.1	8623	8545.15	8555.9	8555.9	152000
02-12-2014	8535.45	8560.2	8504.65	8524.7	8524.7	137400
03-12-2014	8528.7	8546.95	8508.35	8537.65	8537.65	153200
04-12-2014	8582.4	8626.95	8526.4	8564.4	8564.4	143500
05-12-2014	8584.25	8588.35	8523.9	8538.3	8538.3	143300
08-12-2014	8538.65	8546.35	8432.25	8438.25	8438.25	163800
09-12-2014	8439.3	8444.5	8330.5	8340.7	8340.7	146300
10-12-2014	8318.05	8376.8	8317	8355.65	8355.65	136700
11-12-2014	8338.85	8348.3	8272.4	8292.9	8292.9	133900
12-12-2014	8302	8321.9	8216.3	8224.1	8224.1	138100
15-12-2014	8160.75	8242.4	8152.5	8219.6	8219.6	137100
16-12-2014	8172.6	8189.35	8052.6	8067.6	8067.6	197300
17-12-2014	8041.2	8082	7961.35	8029.8	8029.8	216200
18-12-2014	8138.9	8174.3	8084.9	8159.3	8159.3	162100

Figure 3.1.1.1: Sample Dataset

b. Data Pre-processing

Before implementing deep learning models, the raw data was pre-processed to ensure its quality, consistency, and reliability.

- **Addressing Missing Values:** Missing or null values within the dataset were detected and managed appropriately to ensure data integrity and avoid any interruptions during the modelling process.
- **Normalization:** To standardize numerical values and for better model convergence, Min-Max normalization was used to scale the stock prices to a range between 0 and 1. This transformation ensured that all features were aligned on a consistent scale, thereby enhancing training stability and avoiding bias towards larger numerical values.
- **Sliding Window Technique:** To convert the time series data into a supervised learning format, the sliding window technique was employed. In this method, a fixed-length window slides over the normalized historical data to generate input-output pairs. Each window captures a sequence of past prices (features), and the next immediate value serves as the prediction target. This approach enables the model to learn from both short-term fluctuations and long-term dependencies in the stock price.

  For example, given a window of length T, the model learns to map:

$$[x_{t-T}, x_{t-T+1}, \ldots\ldots\ldots\ldots\ldots x_{t-1}] \rightarrow x_t$$

Different sequence lengths were experimented with to determine the optimal setting for models such as LSTM, Transformer Encoder, and others. This method ensures that temporal relationships are preserved and learned effectively during model training.

c. Exploratory Data Analysis (EDA)

A comprehensive Exploratory Data Analysis (EDA) was performed to gain insights into market trends, identify outliers, and evaluate the relationships among various stock Features.

- **Stock Trend Visualization:** The data from the NIFTY 50 index was visualized to evaluate overall market trends, significant fluctuations, and moving averages over time.
- **Correlation Analysis:** A correlation matrix was created to uncover relationships among stock features, with a particular focus on price-related attributes and trading volume.
- **Box Plot Analysis:** Box plots were utilized to examine price distributions and identify potential outliers, thereby enhancing the understanding of stock price volatility across different time frames.

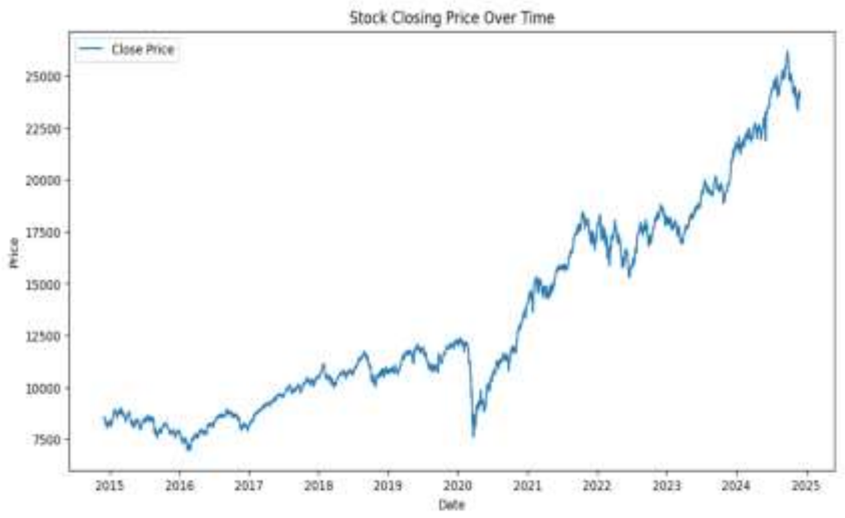


Figure 3.1.1.2: Close Price Trend Visualization

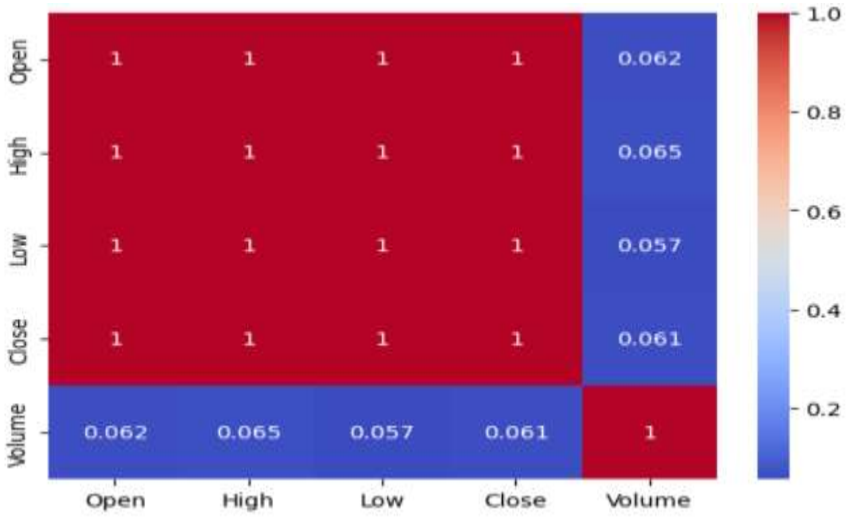


Figure 3.1.1.3: Correlation matrix

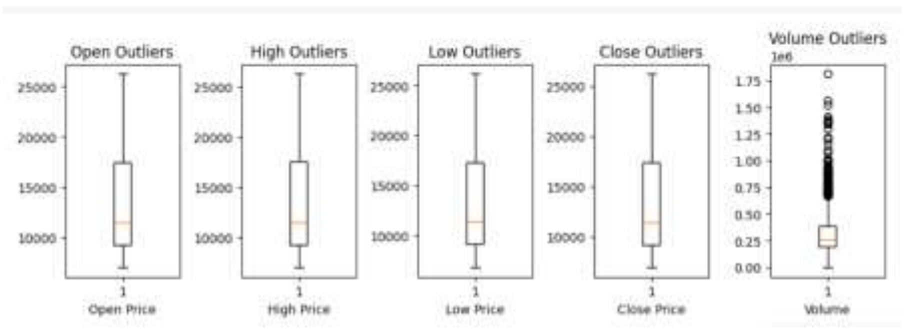


Figure 3.1.1.4: Boxplot

d. Time Series Analysis

Traditional time series methodologies were employed to examine stock price fluctuations, encompassing decomposition, stationarity assessments, and forecasting models.

- **Time Series Decomposition:** The stock price data was segmented into trend, seasonal, and residual components to evaluate both long-term trends and periodic variations.
- **Stationarity Assessment**: The Augmented Dickey-Fuller (ADF) test was utilized to evaluate stationarity, with differencing applied as needed.
- **Autocorrelation Examination:** ACF and PACF plots were analysed to identify lag dependencies and determine the optimal parameters for forecasting models.

- **ARIMA and SARIMA Models:** ARIMA was utilized for non-seasonal forecasting, while SARIMA was employed to capture seasonal patterns in stock prices, serving as benchmark models prior to the application of deep learning techniques.

  For ARIMA, the Augmented Dickey-Fuller (ADF) test confirmed non-stationarity, leading to the application of first-order differencing. The optimal configuration, ARIMA(2,1,1), was selected using ACF and PACF plots. The model captured overall trends but showed limitations in handling volatility and sudden price shifts, with a MAE of 169.2361 and RMSE of 244.3559.

  SARIMA also required differencing for stationarity and was configured as SARIMA(2,1,1)(1,1,1,12) to account for both non-seasonal and seasonal components. While it effectively captured long-term and periodic patterns, it similarly struggled with abrupt market changes, achieving a MAE of 169.1615 and RMSE of 244.1554.

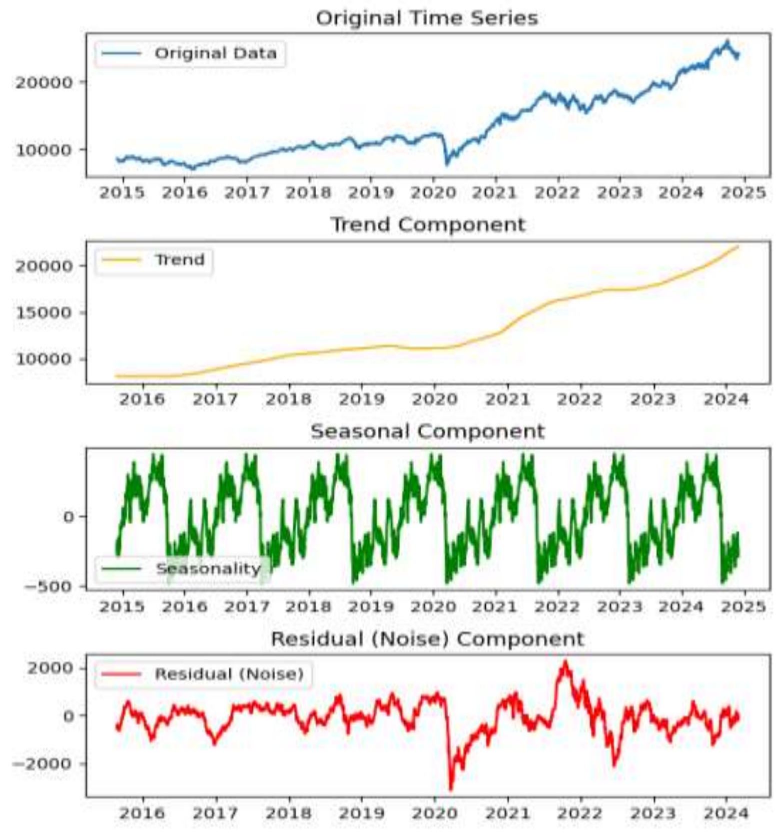


Figure 3.1.1.5: Time Series Decomposition

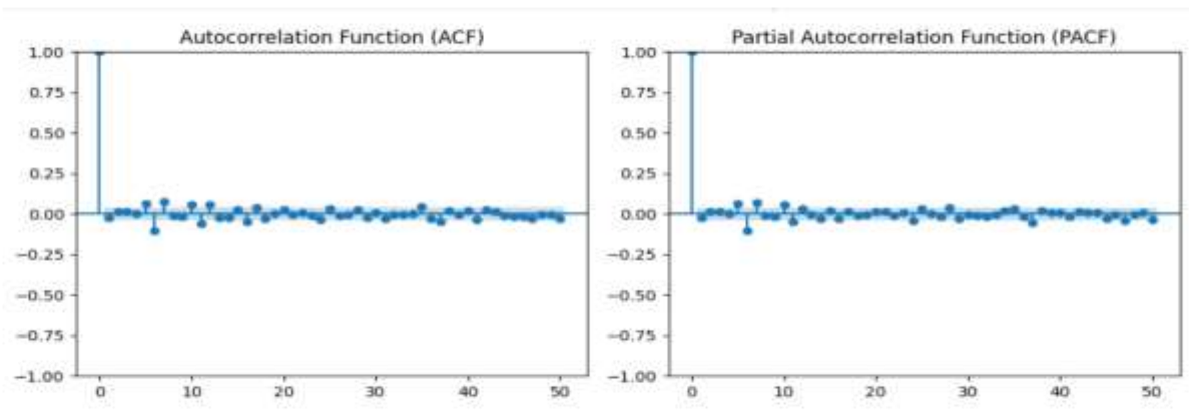


Figure 3.1.1.6**:** ACF and PACF Graphs

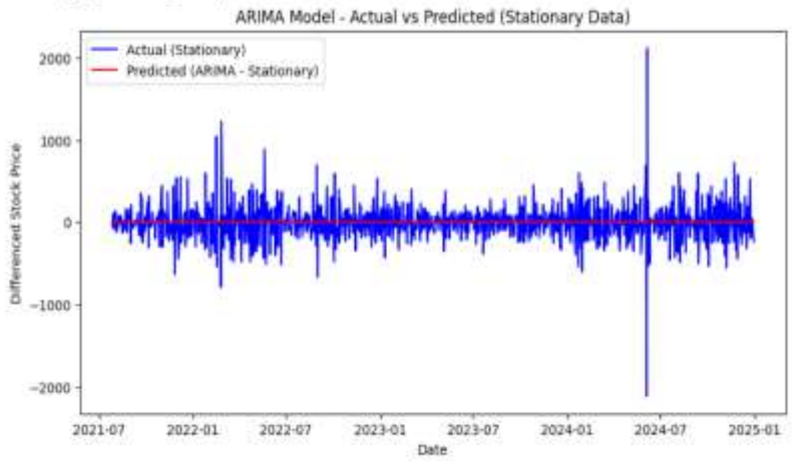


Figure 3.1.1.7: Results Graph of ARIMA

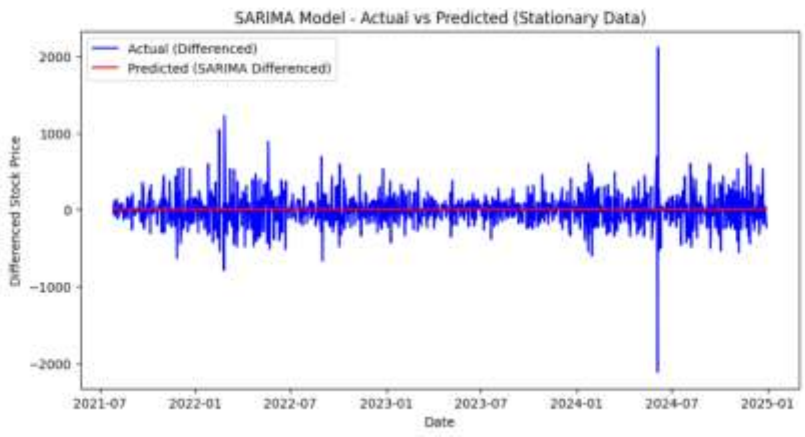


Figure 3.1.1.8: Results Graph of SARIMA

e. Recurrent Neural Network (RNN) Model

A Recurrent Neural Network (RNN) was implemented for Nifty50 close price prediction, by using a sliding window approach to utilize past stock close prices for predicting the next closing price. Although traditional time series models like ARIMA and SARIMA provided a strong statistical foundation, they struggled with capturing complex patterns and non-linear dependencies in stock market data. RNNs addressed this limitation by learning the data in a sequential pattern. However, Traditional RNNs suffered from vanishing gradient problem, which makes it difficult to retain long-term dependencies. To overcome this problem, we Shifted to LSTM and Transformer models.

The RNN model was evaluated using evaluation metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Although it captured short-term trends but struggled with long-term dependencies, mainly during market volatility.

To further improve performance of the model hyperparameter tuning was applied by adjusting dropout rate, learning rate, RNN units, batch size, and number of epochs. Despite these optimizations, the model faced vanishing gradient issues, leading to the development of LSTM and Transformer models for better accurate prediction.

3.1.2 Phase 2: Advanced Deep Learning Models & Mobile Application Development

Based on the insights gained from previous models, we implemented advanced deep learning models like Long Short-Term Memory (LSTM) networks and Transformer-based architectures, to enhance stock price prediction accuracy.

a. Long Short-Term Memory (LSTM) Model

To overcome the vanishing gradient problem faced by traditional RNNs, we implemented a LSTM mode, making it suitable for financial time series forecasting. As LSTM has the ability to retain historical information, selectively remember or forget data through gating mechanisms, and effectively capture short-term fluctuations and long-term trends making it a better choice for stock price prediction.

To further enhance the performance of LSTM model, hyperparameter tuning was performed using Bayesian Optimization and Grid Search. The parameters like number of LSTM units, dropout rate, learning rate, batch size, and epochs, were adjusted to optimize training stability and prevent overfitting. Evaluation metrics included Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to measure prediction accuracy. Finally, the optimized LSTM model demonstrated higher prediction accuracy by efficiently capturing market trends while minimizing errors.

b. Transformer (Encoder only) Model

To enhance stock price forecasting, a Transformer-based model utilizing only the encoder architecture was employed. This approach is well-suited for time series data, as it effectively captures long-term dependencies using self-attention mechanisms. By dynamically assessing the importance of past stock prices, the encoder enhances trend identification and forecasting accuracy. Its parallel processing capability allows for faster computation, while residual connections and layer normalization contribute to training stability and help mitigate vanishing gradients.

To further improve performance, the model underwent hyperparameter optimization, tuning parameters such as the number of attention heads, dimensions of the feedforward network, dropout rates, and optimizer settings (e.g., AdamW). The model's performance was evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), with additional validation through visual comparisons between actual and predicted values and residual error analysis.

c. Mobile Application Development

The best-performing deep learning model has been integrated into a mobile application to provide real-time stock price predictions for the NIFTY 50 index. This application not only accesses the most current stock data to forecast the next day’s closing price but also incorporates sentiment analysis on financial news using a pretrained finBERT model. This additional layer of sentiment intelligence enhances the prediction accuracy by capturing market sentiment from news headlines and descriptions. The backend, developed using Flask, handles API requests, data preprocessing, sentiment inference, and model predictions. Meanwhile, the frontend, built with Flutter, offers an intuitive and user-friendly interface for users to view forecasts, analyze trends, and gain insights. This seamless integration ensures a robust and efficient stock forecasting experience for investors and enthusiasts alike

4 System Design and Architecture

This chapter outlines the system design and architecture for the stock price prediction model, focusing on the deep learning techniques employed in this project, including Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Transformer models. The chapter provides a comprehensive analysis of the architecture and mathematical workings of each model.

4.1 Recurrent Neural Network (RNN) Architecture

Recurrent Neural Networks (RNNs) are a specialized type of neural network designed to process sequences of data by maintaining a hidden state that stores information from previous time steps. Unlike traditional feedforward neural networks, which process inputs independently, RNNs include feedback loops that allow them to retain information over time. This characteristic makes them well-suited for tasks involving time-dependent data, such as time series forecasting, natural language processing, and speech recognition.

At each step, an RNN takes in the current input and updates its hidden state based on both the new input and the previous state. This process allows RNNs to capture temporal dependencies within the data. However, basic RNNs may face challenges when dealing with long-term dependencies due to issues like vanishing gradients, which makes it harder for the model to remember information from earlier time steps. To mitigate this, more advanced variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) have been developed to better handle long-range dependencies.

The ability of RNNs to remember past inputs and make predictions based on historical context makes them particularly effective for applications where prior information is essential for accurate forecasting, such as predicting stock prices or analysing sequential data.

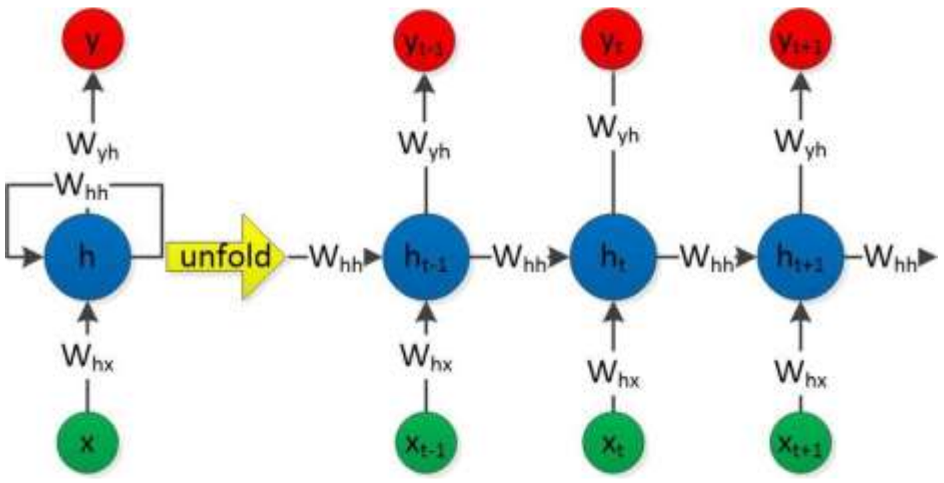


Figure 4.1.1: Architecture of RNN

Mathematical Function

At each time step t, the RNN receives an input x_t and updates its hidden state $h_t$using the information from the previous hidden state h_{t-1} recurrent update is mathematically expressed as:

$$h_t = \tanh(W_h h_{t-1}) + W_x x_t + b_h)$$

Where:

- h_{t-1} is the hidden state at time step t,
- x_t is the input at time step t,
- W_h and W_x are weight matrices that define how past information and current inputs contribute to the hidden state,
- b_h is the bias term,
- tanh represents an activation function, typically a non-linear function such as ReLU

The output at each time step is computed using:

$$y_t = w_y h_t + b_y$$

Where:

- y_t is the output at time step t,
- w_y is the weight matrix that maps the hidden state to the output,
- b_y is the bias term associated with the output layer.

4.2 Long Short-Term Memory (LSTM) Architecture

Long Short-Term Memory (LSTM) networks were introduced to address the limitations of traditional Recurrent Neural Networks (RNNs), especially the vanishing gradient problem, which limits the model's ability to capture long-range dependencies in sequential data. This issue is critical in time series tasks where recognizing patterns over extended periods is essential. LSTMs resolve this problem by using a specialized architecture with memory cells and gates that regulate the flow of information.

An LSTM unit consists of three key gates:

1. **Forget Gate**: This gate determines what information should be discarded from the cell’s memory, using a sigmoid function.
2. **Input Gate**: The input gate controls how much new information is added to the memory cell from the current time step, based on the significance of the data.
3. **Output Gate**: This gate decides what part of the memory cell's contents should be output to the hidden state, allowing the network to update the hidden state effectively.

These gates enable LSTMs to selectively store relevant information, forget irrelevant data, and update their memory over time, making them highly suited for tasks such as time series forecasting, stock price prediction, and natural language processing, where understanding long-term temporal dependencies is key.

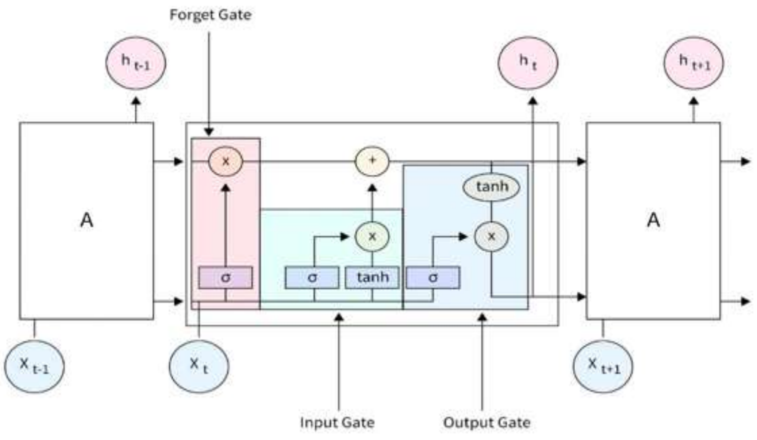


Figure 4.2.1: Architecture of LSTM

Mathematical Function

An LSTM unit consists of four key components: the forget gate**,** input gate**,** cell state update**,** and output gate**.**

Forget Gate

The forget gate determines how much of the previous cell state C_{t-1} should be retained or discarded. It takes the previous hidden state h_{t-1} and the current input x_t and applies a sigmoid activation function to compute a forget factor f_t .

$$f_t = \sigma\,(w_f\,[\,h_{t-1}x_t\,] + b_f\,)$$

Where:

- w_f and b_f are the forget gate's weight matrix and bias.
- σ is the sigmoid activation function, which outputs values between 0 and 1.
- f_t close to 1 means most of the previous information is retained, while values close to 0 mean the information is largely forgotten.

Input Gate and Candidate Cell State

The input gate decides what new information should be added to the cell state. It consists of two steps:

- An input gate determines which values should be updated:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

- **A** candidate cell state C_t is computed using the tanh activation function:

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

Where:

- W_i, W_c and b_i, b_c are weight matrices and biases for the input gate and candidate cell state.
- The $\tanh$ activation ensures that values range between -1 and 1, helping to regulate updates.

Cell State Update

The new cell state ct is computed as:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

This equation ensures that the LSTM can selectively retain past information using f_t and incorporate new relevant information using i_t.

Output Gate and Hidden State

Finally, the output gate determines what part of the cell state should be passed to the next hidden state:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

The hidden state h_t is then updated as:

$$h_t = o_t \cdot tanh(C_t)$$

Where:

- o_t regulates how much of the cell state contributes to the next hidden state.

4.3 Transformer (Encoder-only) Model Architecture

The Transformer Encoder is a deep learning model that processes input sequences in parallel, rather than sequentially as in traditional models like Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks. This architecture is built around the self-attention mechanism, which allows the model to simultaneously analyze the entire sequence and capture dependencies between distant elements.

In the Transformer Encoder, the input tokens are first transformed into three different vectors query, key, and value through linear transformations. These vectors are used to compute attention scores that measure the relevance of each token in the sequence to others. This mechanism enables the model to learn complex relationships within the data, regardless of the position of the elements.

The Transformer Encoder consists of several layers, each containing two main components:

1. **Multi-Head Self-Attention**: This mechanism enables the model to focus on multiple parts of the input sequence in parallel, capturing different types of relationships and dependencies.
2. **Feed-Forward Networks**: After the attention mechanism, the model uses fully connected layers to process the output. These layers are followed by layer normalization and residual connections to maintain stability and improve training performance.

The parallel processing capability of the Transformer Encoder allows it to handle long-range dependencies more efficiently and is particularly advantageous for time series prediction tasks, such as stock price forecasting. By using self-attention, the Transformer avoids the need for recurrent connections, significantly improving training speed and performance on tasks with complex temporal patterns.

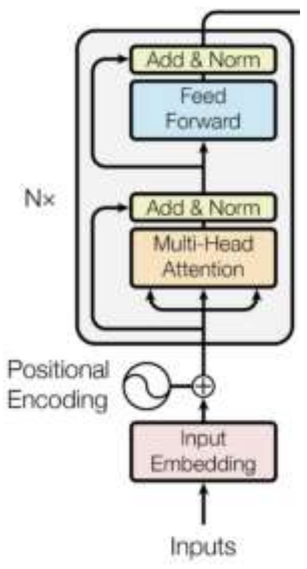


Figure 4.3.1: Architecture of Transformer (Encoder only)

Self-Attention Mechanism

The core of the Transformer model is the self-attention mechanism, which allows the model to dynamically weigh different parts of the input sequence when making predictions. The relationships between time steps in the input sequence are computed using Query (Q), Key (K), and Value (V) matrices, and attention scores are calculated as follows:

$$Attention\ (Q, K, V) = softmax\left(\frac{Qk^T}{\sqrt{d_k}}\right)V$$

Where:

- $Q = XW_Q, K = XW_K$ and $V = XW_V$ are the Query, Key, and Value matrices obtained by linearly projecting the input sequence X.
- d_k is the scaling factor to stabilize gradients.
- The Softmax function normalizes attention scores, ensuring that the model assigns higher importance to relevant time steps.

Unlike RNNs, which process inputs sequentially, self-attention allows the Transformer to directly compare every time step to every other time step in parallel, making it highly efficient for long-term trend analysis in stock price forecasting.

Multi-Head Attention Mechanism

Instead of using a single attention mechanism, Transformers employ multi-head attention to learn different representations of input dependencies:

$$MultiHead(Q, K, V) = Concat\ (heard_1, \dots\dots, head_h)W_o$$

Where:

- Each head is an independent self-attention layer with separate learnable parameters.
- The outputs of multiple heads are concatenated and passed through a final linear projection layer using W_o.
- Multi-head attention enables the model to capture multiple patterns simultaneously**,** improving performance on complex time series like stock price movements

Positional Encoding

Unlike RNNs and LSTMs, the Transformer model does not have an inherent sense of sequence order, as it processes the entire input in parallel. To incorporate information about the position of each time step, we use positional encodings, which are added to the input embeddings:

$$PE(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE(pos, 2i + 1) = cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Where:

- pos represents the position of a time step in the sequence.
- d is the embedding dimension**.**

The sine and cosine functions allow the model to learn relative positional information efficiently.

Feedforward Layers and Residual Connections

After self-attention, the Transformer model passes the output through a fully connected feedforward network with ReLU activations**:**

$$FFN(x) \; = \; ReLu\,(xW_1 \; + \; b_1)W_2 \; + \; b$$

To improve gradient flow and stabilize training, residual connections and layer normalization are applied:

$$LayerNorm(x \; + \; Sublayer(x))$$

where $Sublayer(x)$ represents either self-attention or the feedforward layer**.**

4.4 Evaluation Metrics

To assess the predictive performance of the proposed models—Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Transformer Encoder we utilized two standard regression evaluation metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)**.** These metrics provide a quantitative measure of the accuracy of the model's predicted stock prices compared to the actual values.

Mean Absolute Error (MAE)

Mean Absolute Error (MAE) calculates the average magnitude of errors between predicted and actual values without considering their direction. It provides a straightforward interpretation of how far the predictions deviate from the actual values on average.

$$MAE \; = \; \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2$$

Where:

- y_i is the actual stock price at time step i
- $\hat{y}_i$ is the predicted stock price at time step i
- n is the total number of predictions.

Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) measures the square root of sthe average squared differences between predicted and actual values. It penalizes larger errors more significantly, making it sensitive to outliers in the predictions.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:

- y_i and $\hat{y}_i$ are the actual and predicted values respectively,
- n denotes the number of observations in the evaluation dataset.

These metrics were applied during the testing phase to evaluate the models' generalization ability and overall forecasting accuracy. Lower values of MAE and RMSE indicate better model performance in capturing the underlying patterns in stock price movements.

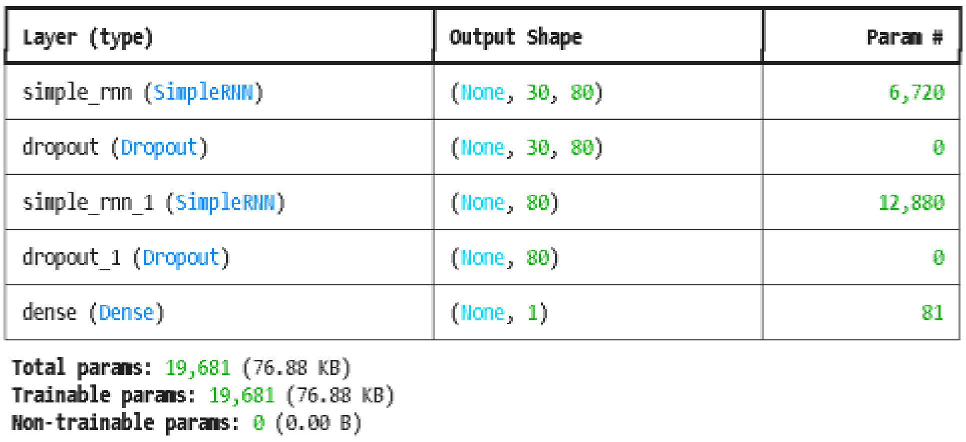
5 System Development, Testing and Validation

This chapter outlines the models developed for predicting stock prices. It covers the entire process of model creation, detailing the methodologies used for training, hyperparameter tuning, and the evaluation procedures. Additionally, the chapter highlights the metrics employed to assess model performance, focusing on error metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The validation results demonstrate the effectiveness of the models when applied to historical stock price data.

5.1 Recurrent Neural Network (RNN) Development and Validation

To integrate deep learning methodologies for predicting stock prices, we developed a Recurrent Neural Network (RNN) model. In contrast to conventional time series models, RNNs utilize sequential dependencies, rendering them effective for financial forecasting. Initially, the dataset underwent pre-processing, which included calculating the Simple Moving Average (SMA) and the Relative Strength Index (RSI) to introduce additional features that reflect short-term trends and momentum. Subsequently, the data was normalized through MinMax scaling to ensure numerical stability. A sequence length of 30 days was selected, indicating that the model was trained to forecast the stock price for the following day based on the preceding 30 days. The dataset was divided into 80% for training and 20% for testing, maintaining the chronological order without shuffling. Furthermore, a validation split of 20% was applied to the training data to effectively optimize the model's hyperparameters during the training process.

To enhance the performance of the RNN, hyperparameter tuning was performed using Keras Tuner, systematically examining the number of RNN units (ranging from 32 to 128), dropout rates (from 0.1 to 0.5), and learning rates (0.01, 0.001, and 0.0001). The optimal model configuration included 128 RNN units, a dropout rate of 0.1, and a learning rate of 0.001. The architecture comprised two stacked Simple RNN layers utilizing ReLU activation, integrated with dropout layers to mitigate the risk of overfitting. The model was trained for 50 epochs using the Adam optimizer and the Mean Squared Error (MSE) loss function, with validation conducted on a designated validation set. The training and validation loss curves indicated a consistent reduction in loss, signifying effective learning without signs of overfitting. When comparing the model's predictions to actual prices, it was evident that while the model successfully identified general trends, it faced challenges in responding to abrupt market changes. The final Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) served as quantitative indicators of the model's accuracy. Additionally, various RNN models also implemented using different sequence lengths and more technical indicators. The specifics of other RNN variants and their performance comparisons are elaborated upon in the Discussion of Results section. Although the RNN was capable of learning sequential patterns, it showed limitations in capturing long-term dependencies, leading to the decision to transition to Long Short-Term Memory (LSTM) networks for enhanced predictive capabilities.



Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 30, 80)	6,720
dropout (Dropout)	(None, 30, 80)	0
simple_rnn_1 (SimpleRNN)	(None, 80)	12,880
dropout_1 (Dropout)	(None, 80)	0
dense (Dense)	(None, 1)	81

```
Total params: 19,681 (76.88 KB)
Trainable params: 19,681 (76.88 KB)
Non-trainable params: 0 (0.00 B)
```

Figure 5.1.1: Model Summary of RNN

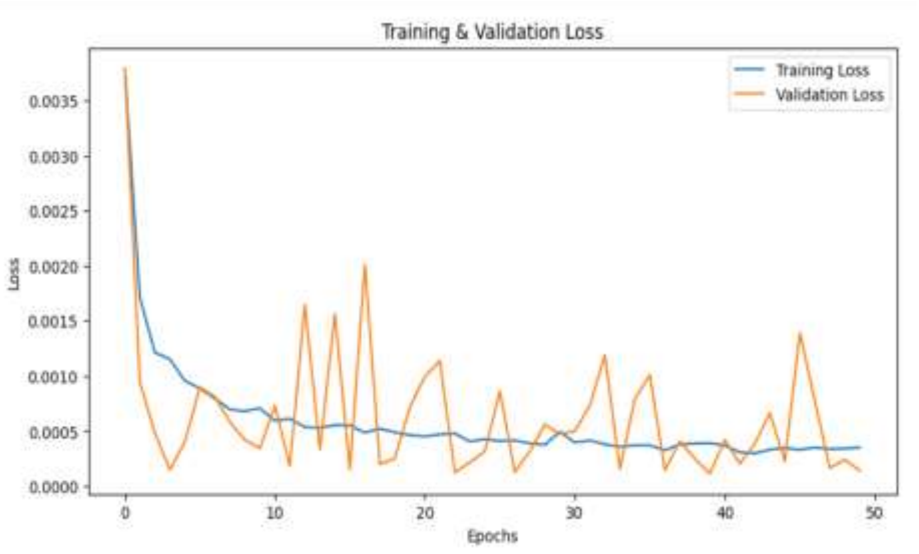


Figure 5.1.2: Loss Curve of RNN

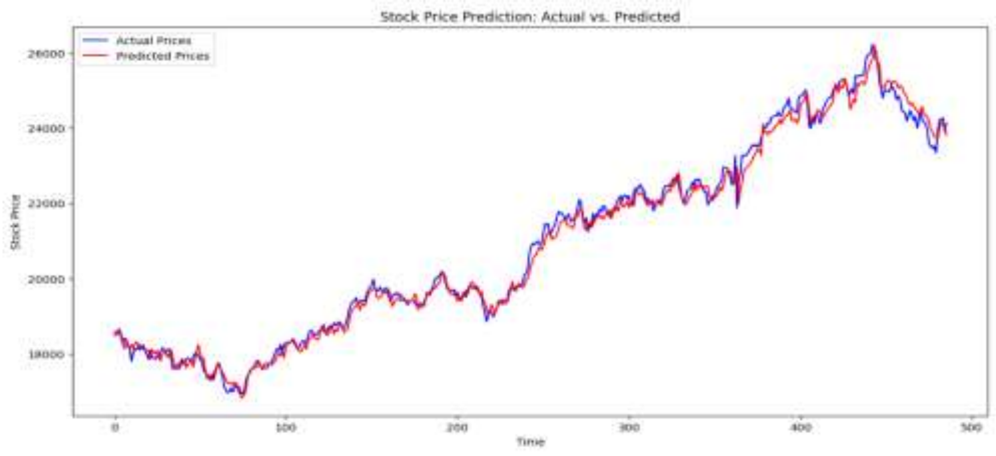


Figure 5.1.3: Results Graph of RNN

Evaluation Metrics

MAE: 180.4257, RMSE: 228.0501

5.2 Long Short-Term Memory (LSTM) Development and Validation

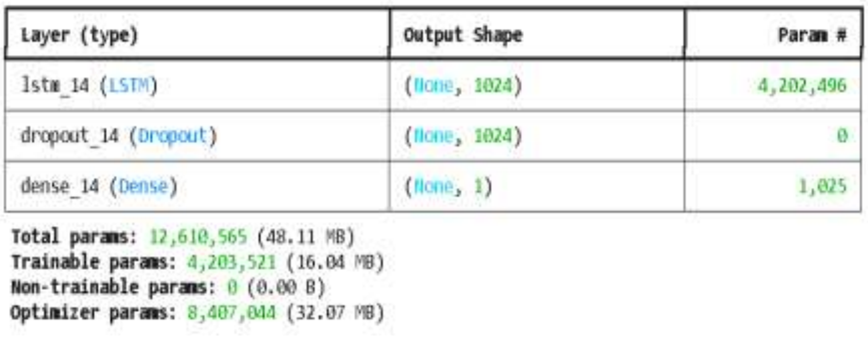
To address the problems of traditional Recurrent Neural Networks (RNNs) in capturing long-term dependencies, particularly due to the vanishing gradient problem, we developed a Long Short-Term Memory (LSTM) network aimed at enhancing the accuracy of stock price predictions. The unique gating mechanisms of LSTMs enable the retention of significant historical information, thereby alleviating the challenges associated with long-term dependency loss.

For effective learning, the dataset was pre-processed to manage missing values through forward filling and was normalized using MinMax scaling. The model was trained on sequence length of 60 days, allowing it to identify patterns from the previous 60 trading days to forecast the stock price for the following day. The dataset was divided into 80% for training and 20% for testing, with a consistent random state to ensure reproducibility. In contrast to RNNs, which typically necessitate multiple stacked layers for deeper feature extraction, a single LSTM layer comprising 256 units proved better for capturing long-term dependencies. Additionally, a lower dropout rate of 0.02 was employed to preserve more information while minimizing the risk of overfitting.

Hyperparameter tuning was essential for enhancing model performance. Rather than opting for a high learning rate, we used Adam optimizer with a reduced learning rate of 0.0005 was chosen to facilitate improved convergence. A batch size of 8 was employed to strike a balance between stability and training efficiency. Furthermore, Early Stopping with a patience parameter of 10 was utilized to mitigate overfitting and ensure the retention of the best model based on validation loss. The model underwent training for 100 epochs, enabling it to learn intricate sequential patterns without early termination.

The loss curves demonstrated a consistent decrease in both training and validation loss, indicating effective learning. To evaluate model performance, the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) were calculated. A comparison graph of actual versus predicted stock prices illustrated that the LSTM model successfully captured overall trends, although it displayed a slight delay in responding to rapid trend changes, a typical issue in stock market forecasting. Nevertheless, it significantly surpassed the baseline RNN model by more effectively maintaining long-term dependencies.

Multiple LSTM architectures were explored, and through a thorough evaluation process, the most effective configuration was identified. The specifics of other LSTM variants and their performance comparisons are elaborated upon in the Discussion of Results section.



Layer (type)	Output Shape	Param #
lstm_14 (LSTM)	(None, 1024)	4,202,496
dropout_14 (Dropout)	(None, 1024)	0
dense_14 (Dense)	(None, 1)	1,025

```
Total params: 12,610,565 (48.11 MB)
Trainable params: 4,203,521 (16.04 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 8,407,044 (32.07 MB)
```

Figure 5.2.1: Model Summary of LSTM

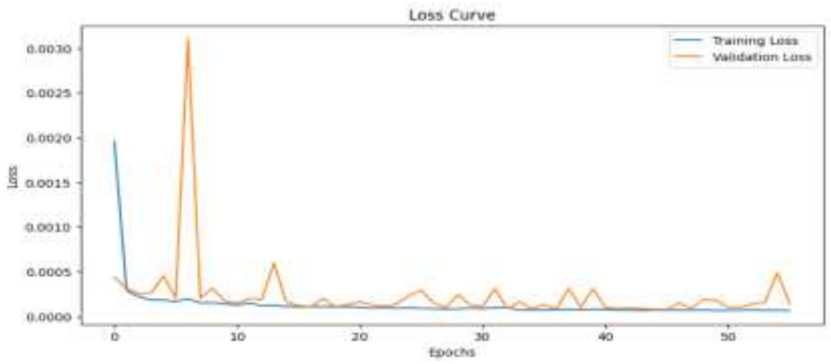


Figure 5.2.2: Loss Curve of LSTM

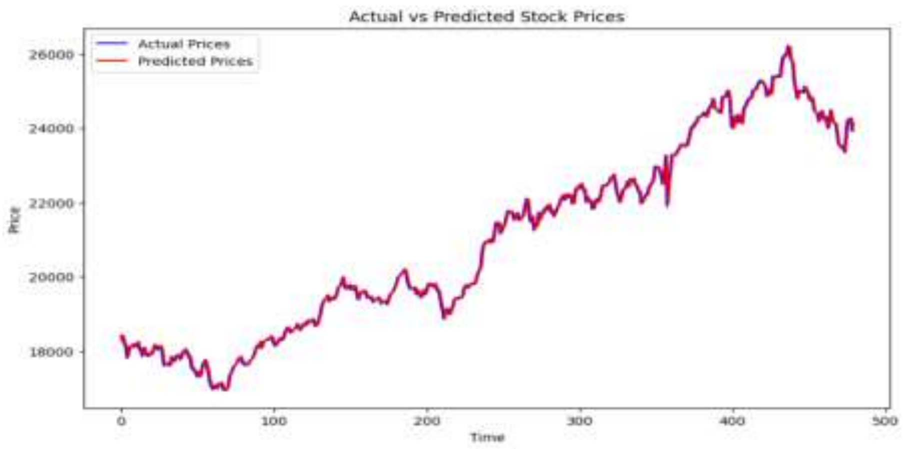


Figure 5.2.3: Results Graph of LSTM

Evaluation Metrics

MAE: 116.8200, RMSE: 167.6884

5.3 Transformer Encoder Development and Validation

A Transformer encoder-based model was utilized to forecast stock prices based on historical NIFTY50 data. In contrast to conventional time series models, Transformers utilize self-attention mechanisms that effectively capture both short-term variations and long-term trends, rendering them particularly suitable for financial predictions.

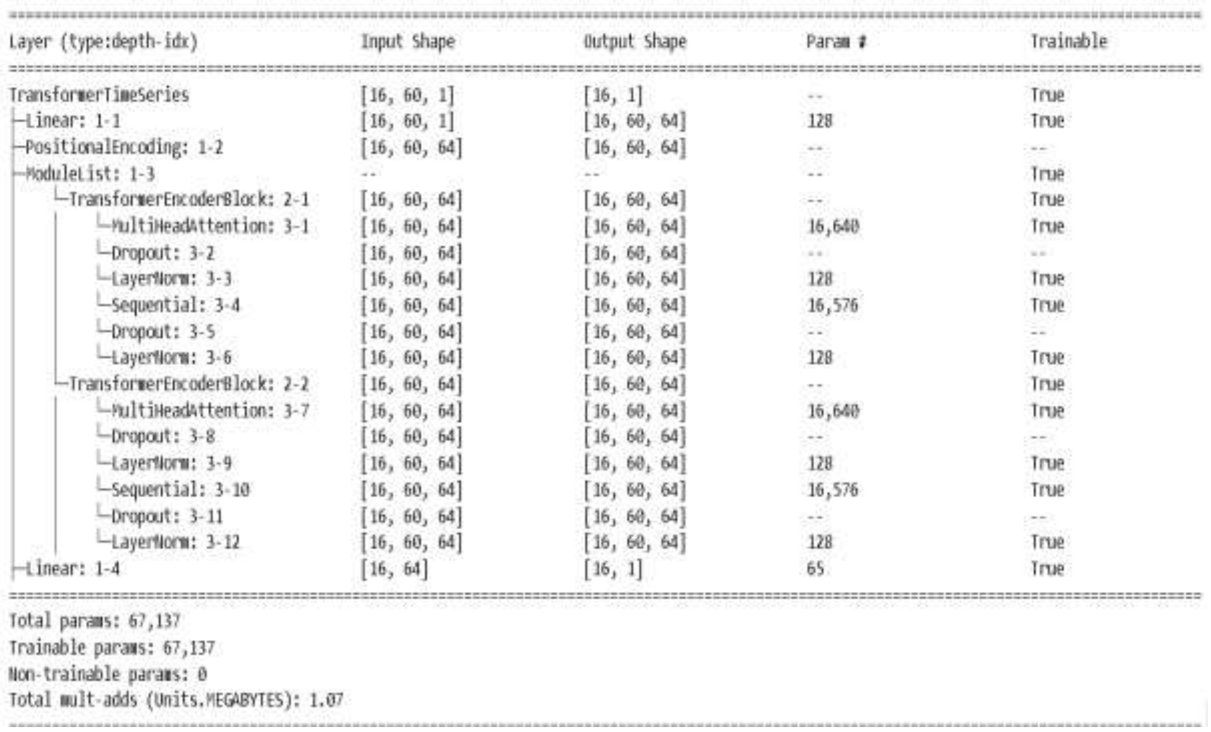
Initially, the dataset underwent preprocessing, focusing solely on the Close price, which was normalized through MinMax Scaling. Input sequences were generated with a length of 60 days, ensuring the model had sufficient historical context to predict the subsequent day's closing price. The data was divided into training (80%) and testing (20%) sets while preserving chronological order to maintain the time-dependent nature of the data.

The architecture of the Transformer encoder model includes several essential components. An embedding layer first transforms the input features into a higher-dimensional space (d_model=64) to improve representational learning. This is succeeded by positional encoding, which allows the model to retain the sequence of data points. The model's core comprises two Transformer Encoder layers, each featuring Multi-Head Self-Attention (num_heads=4) to capture intricate dependencies across time steps. Furthermore, a Feedforward Neural Network (hidden_dim=128) enhances the transformation of features, while Layer Normalization and Residual Connections contribute to stabilizing the learning process. Ultimately, a Dense output layer produces the predicted stock prices.

Training was conducted using the Adam optimizer with a learning rate of 0.001, and Mean Squared Error (MSE) loss was employed to minimize prediction inaccuracies. The model was trained for 100 epochs with a batch size of 16. The loss curves demonstrated successful convergence, with both training and validation losses consistently decreasing over time.

Upon assessment, the model demonstrated strong performance in the early time steps, accurately mirroring the actual stock prices. However, as time advanced, it encountered difficulties in maintaining precise alignment with the real prices. Although it effectively captured the overarching market trends, the model showed a tendency to lag behind during sudden price fluctuations and frequently strayed from actual values in volatile market scenarios. This indicates that while the model has grasped the fundamental patterns of the stock market, it struggles to adjust to rapid shifts in trends.

The plot comparing actual and predicted stock prices shows that, while the model does a good job of tracking the overall trend of the market, it struggles to keep up with sudden price movements. This leads to noticeable gaps in long-term predictions.



```
==========================================================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #                   Trainable
==========================================================================================================================
TransformerTimeSeries                    [16, 60, 1]               [16, 1]                   --                        True
├─Linear: 1-1                            [16, 60, 1]               [16, 60, 64]              128                       True
├─PositionalEncoding: 1-2                [16, 60, 64]              [16, 60, 64]              --                        --
├─ModuleList: 1-3                        --                        --                        --                        True
│    └─TransformerEncoderBlock: 2-1      [16, 60, 64]              [16, 60, 64]              --                        True
│    │    └─MultiHeadAttention: 3-1      [16, 60, 64]              [16, 60, 64]              16,640                    True
│    │    └─Dropout: 3-2                 [16, 60, 64]              [16, 60, 64]              --                        --
│    │    └─LayerNorm: 3-3               [16, 60, 64]              [16, 60, 64]              128                       True
│    │    └─Sequential: 3-4              [16, 60, 64]              [16, 60, 64]              16,576                    True
│    │    └─Dropout: 3-5                 [16, 60, 64]              [16, 60, 64]              --                        --
│    │    └─LayerNorm: 3-6               [16, 60, 64]              [16, 60, 64]              128                       True
│    └─TransformerEncoderBlock: 2-2      [16, 60, 64]              [16, 60, 64]              --                        True
│    │    └─MultiHeadAttention: 3-7      [16, 60, 64]              [16, 60, 64]              16,640                    True
│    │    └─Dropout: 3-8                 [16, 60, 64]              [16, 60, 64]              --                        --
│    │    └─LayerNorm: 3-9               [16, 60, 64]              [16, 60, 64]              128                       True
│    │    └─Sequential: 3-10             [16, 60, 64]              [16, 60, 64]              16,576                    True
│    │    └─Dropout: 3-11                [16, 60, 64]              [16, 60, 64]              --                        --
│    │    └─LayerNorm: 3-12              [16, 60, 64]              [16, 60, 64]              128                       True
├─Linear: 1-4                            [16, 64]                  [16, 1]                   65                        True
==========================================================================================================================
Total params: 67,137
Trainable params: 67,137
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.07
==========================================================================================================================
```

Figure 5.3.1: Model Summary of Transformer

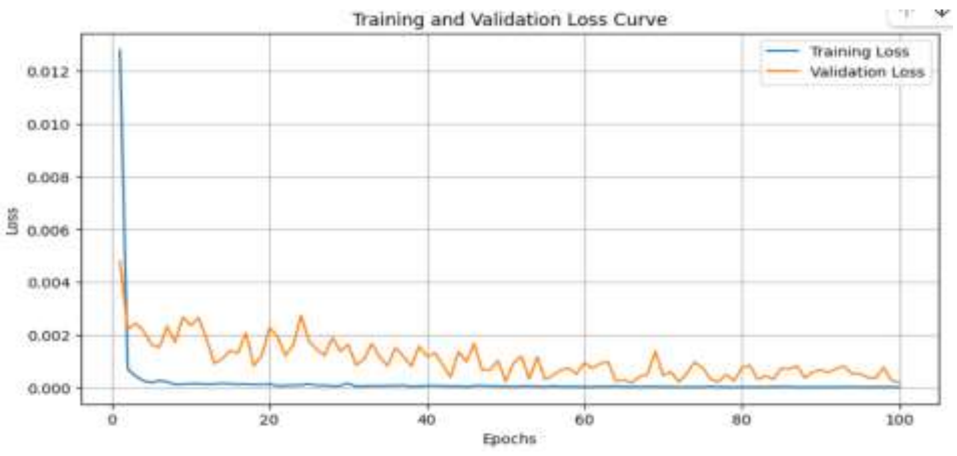


Figure 5.3.2: Loss Curve of Transformer

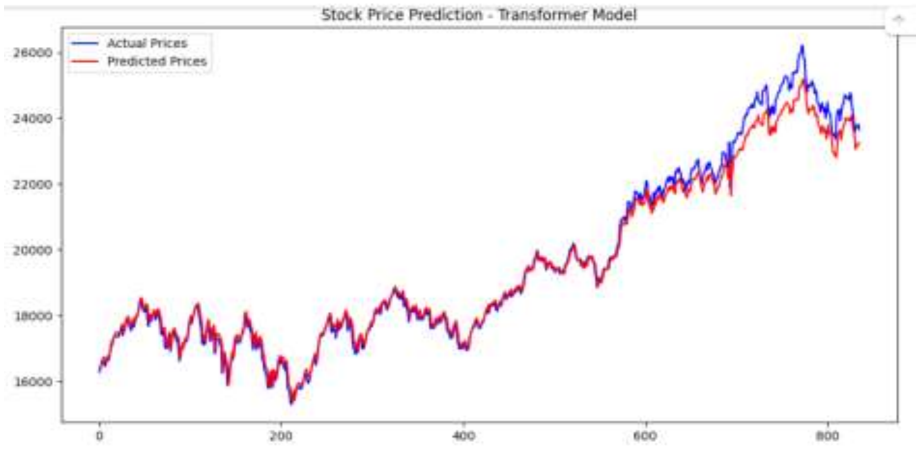


Figure 5.3.3: Result Graph of Transformer

Evaluation Metrics

MAE: 248.9035, RMSE: 348.1498

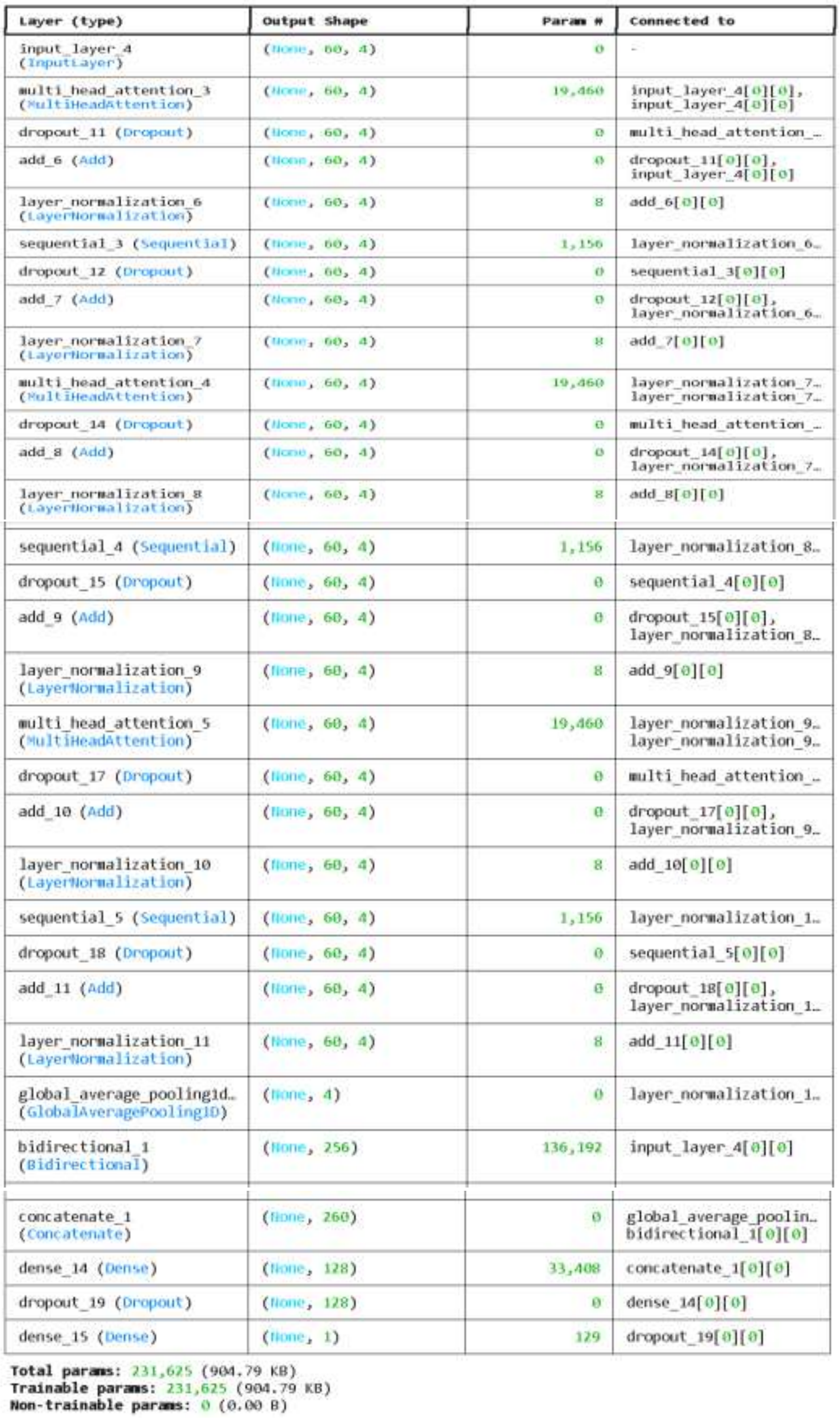
5.4 Hybrid Model Development and Validation (Combining Transformer and LSTM)

To enhance the accuracy of stock price predictions, a Hybrid Transformer-LSTM model was developed by combining the strengths of both architectures. Transformers are particularly effective at capturing long-term dependencies through self-attention mechanisms, making them well suited for identifying global patterns in time series data. In contrast, LSTMs are designed to learn sequential and short-term dependencies via memory cells. By integrating these two models, the hybrid architecture leverages the Transformer's ability to model global trends and the LSTM's strength in capturing local temporal relationships, leading to more robust predictions.

Feature engineering played a key role in improving the model’s predictive performance. Bollinger Bands (both upper and lower) were included to capture market volatility, while the Average True Range (ATR) was added to reflect daily price variation. These features, along with the closing price, were normalized using MinMaxScaler to scale the data between 0 and 1, improving model convergence. A sequence length of 60 days was used to provide sufficient historical context for predicting the next day’s stock price. The dataset was split sequentially, with 80% allocated for training and 20% for testing, preserving the temporal nature of the data.

The architecture of the hybrid model consists of three Transformer blocks, each comprising Multi-Head Self-Attention (with 8 heads and a head size of 128), followed by a feedforward neural network using ReLU activation. Residual connections and Layer Normalization were applied after each block to ensure training stability and mitigate vanishing gradients. To prevent overfitting, dropout layers with a rate of 0.3 were included. In parallel, a Bidirectional LSTM layer with 128 units was applied to the input sequence, allowing the model to learn patterns in both forward and backward directions. The outputs from the Transformer and LSTM paths were combined using a Concatenate layer, followed by a Dense-ReLU layer, a final Dropout, and a Dense output layer to produce the predicted closing price.

The model was compiled using the Huber loss function to reduce the impact of outliers, and the Adam optimizer with a learning rate of 0.0003 was used for efficient training. It was trained for 100 epochs with a batch size of 16. Evaluation on the test set showed strong predictive performance, with low values for Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE), indicating that the model closely follows actual stock price movements. The actual vs. predicted stock price plot further validated the model’s accuracy, demonstrating that the hybrid architecture captures both trend and variation effectively. Training and validation loss curves showed stable and consistent learning, with no significant signs of overfitting. While the model performed well, small deviations in highly volatile conditions suggest future improvements, such as incorporating additional indicators, using dynamic learning rate scheduling, or enhancing the model with attention visualization or ensemble techniques.



Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 60, 4)	0	-
multi_head_attention_3 (MultiHeadAttention)	(None, 60, 4)	19,460	input_layer_4[0][0], input_layer_4[0][0]
dropout_11 (Dropout)	(None, 60, 4)	0	multi_head_attention_…
add_6 (Add)	(None, 60, 4)	0	dropout_11[0][0], input_layer_4[0][0]
layer_normalization_6 (LayerNormalization)	(None, 60, 4)	8	add_6[0][0]
sequential_3 (Sequential)	(None, 60, 4)	1,156	layer_normalization_6…
dropout_12 (Dropout)	(None, 60, 4)	0	sequential_3[0][0]
add_7 (Add)	(None, 60, 4)	0	dropout_12[0][0], layer_normalization_6…
layer_normalization_7 (LayerNormalization)	(None, 60, 4)	8	add_7[0][0]
multi_head_attention_4 (MultiHeadAttention)	(None, 60, 4)	19,460	layer_normalization_7… layer_normalization_7…
dropout_14 (Dropout)	(None, 60, 4)	0	multi_head_attention_…
add_8 (Add)	(None, 60, 4)	0	dropout_14[0][0], layer_normalization_7…
layer_normalization_8 (LayerNormalization)	(None, 60, 4)	8	add_8[0][0]
sequential_4 (Sequential)	(None, 60, 4)	1,156	layer_normalization_8…
dropout_15 (Dropout)	(None, 60, 4)	0	sequential_4[0][0]
add_9 (Add)	(None, 60, 4)	0	dropout_15[0][0], layer_normalization_8…
layer_normalization_9 (LayerNormalization)	(None, 60, 4)	8	add_9[0][0]
multi_head_attention_5 (MultiHeadAttention)	(None, 60, 4)	19,460	layer_normalization_9… layer_normalization_9…
dropout_17 (Dropout)	(None, 60, 4)	0	multi_head_attention_…
add_10 (Add)	(None, 60, 4)	0	dropout_17[0][0], layer_normalization_9…
layer_normalization_10 (LayerNormalization)	(None, 60, 4)	8	add_10[0][0]
sequential_5 (Sequential)	(None, 60, 4)	1,156	layer_normalization_1…
dropout_18 (Dropout)	(None, 60, 4)	0	sequential_5[0][0]
add_11 (Add)	(None, 60, 4)	0	dropout_18[0][0], layer_normalization_1…
layer_normalization_11 (LayerNormalization)	(None, 60, 4)	8	add_11[0][0]
global_average_pooling1d… (GlobalAveragePooling1D)	(None, 4)	0	layer_normalization_1…
bidirectional_1 (Bidirectional)	(None, 256)	136,192	input_layer_4[0][0]
concatenate_1 (Concatenate)	(None, 260)	0	global_average_poolin… bidirectional_1[0][0]
dense_14 (Dense)	(None, 128)	33,408	concatenate_1[0][0]
dropout_19 (Dropout)	(None, 128)	0	dense_14[0][0]
dense_15 (Dense)	(None, 1)	129	dropout_19[0][0]

```
Total params: 231,625 (904.79 KB)
Trainable params: 231,625 (904.79 KB)
Non-trainable params: 0 (0.00 B)
```

Figure 5.4.1: Model Summary of Hybrid Model

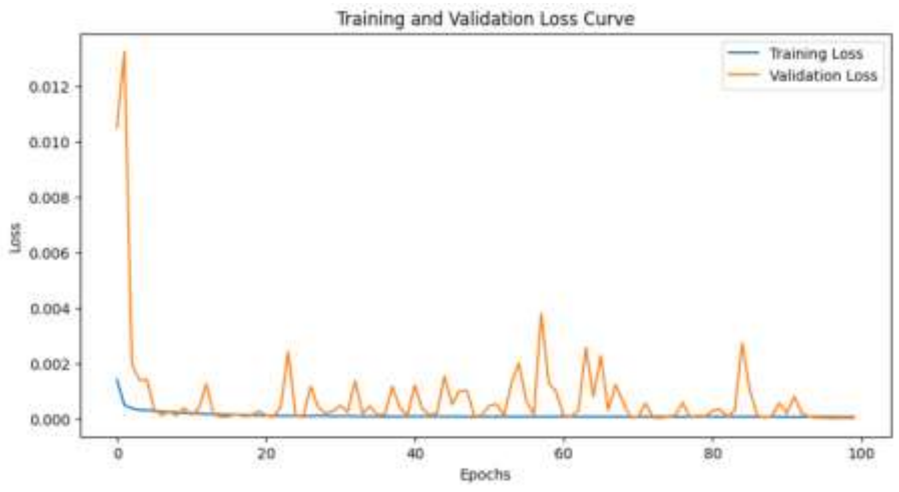


Figure 5.4.2: Loss curve of Hybrid Model

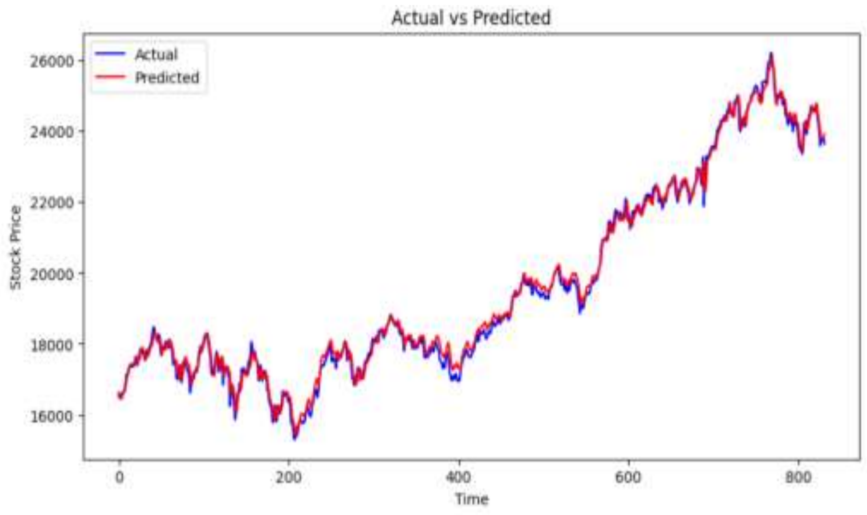


Figure 5.4.3: Results Graph of Hybrid Model

Evaluation Metrics

MAE: 159.4657, RMSE: 206.7501

5.5 Prediction Mechanism

The prediction mechanism uses the optimized LSTM model trained on the past 60 days of NIFTY 50 closing prices. Initially, the latest 60-day stock data up to date is loaded and normalized using Min-Max scaling by ensuring all values fall within the range 0 to 1. This normalized sequence is then reshaped into a 3D input format suitable for the LSTM model. The trained model processes this input and generates a prediction for the next day's closing price of the nifty50 index in the scaled form, which is then inverse transformed to retrieve the actual price. To ensure the prediction aligns with a valid trading day, the system checks for weekends and NSE holidays, adjusting the date accordingly. The final predicted closing price is displayed along with the predicted trading date, enabling timely and accurate stock market predictions for practical applications.

5.6 Mobile Application Implementation

To implement the most effective model in a practical setting, we created a user-friendly mobile application that incorporates the top-performing LSTM model with a sequence length of 60. This application automatically fetches the most recent 60 days of NIFTY 50 stock data from Yahoo Finance using the finance library on a daily basis and forecasts the closing price of the NIFTY 50 index for the following day. Beyond stock price predictions, the app functions as an educational resource for novices to understand the NIFTY 50 index and market dynamics. A key feature of the app is sentiment analysis of financial news headlines and descriptions**,** performed using a pretrained finBERT model, which captures the emotional tone of the market. Additionally, the application includes a Fear and Greed Index based on technical indicators and provides intelligent recommendations to buy, sell, or hold, derived from a combination of model predictions, sentiment evaluations, and technical analysis. By offering an engaging and straightforward method for stock analysis, this application helps beginners learn how machine learning models interpret market trends and generate forecasts, making it an excellent resource for those interested in stock trading and market prediction.

6 Discussion of Results

This module offers an examination of the outcomes derived from various models employed for predicting stock prices. A range of deep learning architectures was investigated, including RNN, LSTM, Transformer, and a Hybrid LSTM-Transformer model, each with distinct configurations aimed at enhancing performance. The evaluation of results is conducted using essential metrics such as MAE and RMSE, helped in a comparative analysis to identify the most efficient model. Furthermore, visual representations are included to understand the predictions of each model with the actual stock prices.

Table 6.1: Performance Comparison of RNN Variants

Model	**Sequence Length**	**Features used**	**MAE**	**RMSE**
RNN	30	Close, SMA_10, RSI_14	180.4257	228.0501
RNN	60	Close, SMA_10, RSI_14	1879.9052	1932.0890

Table 6.2: Performance Comparison of LSTM Variants

Model	**Sequence Length**	**Features Used**	**MAE**	**RMSE**
LSTM	10	Close	206.10	271.49
LSTM	15	Close	194.25	254.64
LSTM	20	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	147.32	194.75
LSTM	25	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	132.11	181.62
LSTM	30	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	166.53	217.82
LSTM	45	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	133.94	182.94
LSTM	50	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	177.36	228.22
LSTM	60	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	154.9166	202.7301
LSTM	60	Close	116.82	167.68
LSTM	75	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	144.19	190.05
LSTM	90	Close, SMA_30, RSI, MACD, Lag_1, Lag_2	150.65	203.38
Stacked LSTM	60	Close	248.81	299.92
Stacked LSTM	60	Close, SMA_30, SMA_50, EMA_20, EMA_50, ATR,	2903.39	3170.66

		RSI, MACD, Lag_1, Lag_2		
Bidirectional LSTM	60	Close, SMA_30, SMA_50, RSI, Bollinger_ high, Bollinger low	314.18	397.13

Table 6.3: Performance Comparison of Transformer (Encoder only) Variants

Model	Sequence Length	Features Used	MAE	RMSE
Transformer model	30	Close	319.05	470.46
Transformer model	60	Close	248.90	348.15
Transformer model	90	Close	360.52	532.64

Table 6.4: Performance Comparison of Hybrid (LSTM + Transformer) model Variants

Model	Sequence Length	Features Used	MAE	RMSE
Hybrid (LSTM + Transformer)	30	Close, Upper band, Lower band, ATR	163.87	205.35
Hybrid (LSTM + Transformer)	60	Close, Upper band, Lower band, ATR	159.47	206.75
Hybrid (LSTM+Transformer)	60	Close	225.10	291.86

Table 6.5: Final Comparison of Best Models Across All Architectures

Model	Sequence Length	Features Used	MAE	RMSE
Best RNN	30	Close, SMA_10, RSI_14	180.4257	228.0501
Best LSTM	60	Close	116.82	167.68
Best Transformer	60	Close	248.90	348.15
Best Hybrid (LSTM + Transformers)	60	Close, Upper band, Lower band, ATR	159.47	206.75

Comparison of Best-Performing Models: Actual vs. Predicted Graphs

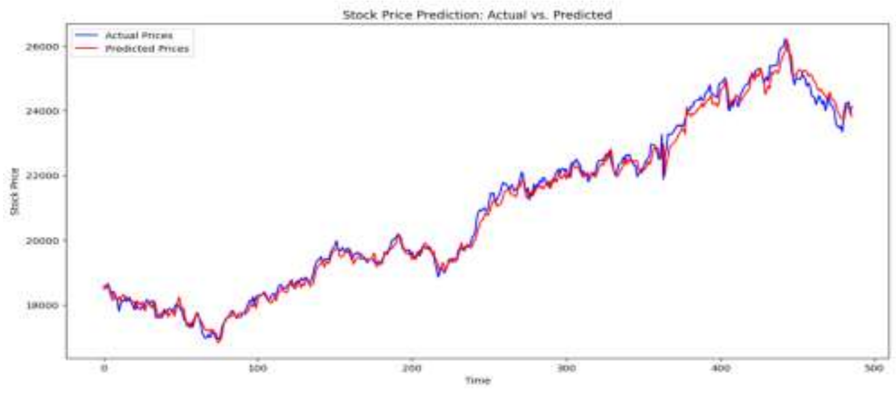


Figure 6.1: RNN

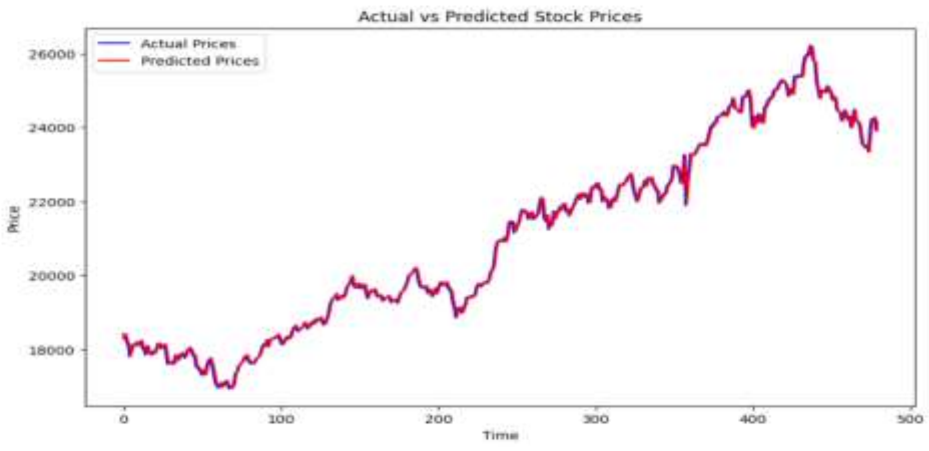


Figure 6.2: LSTM

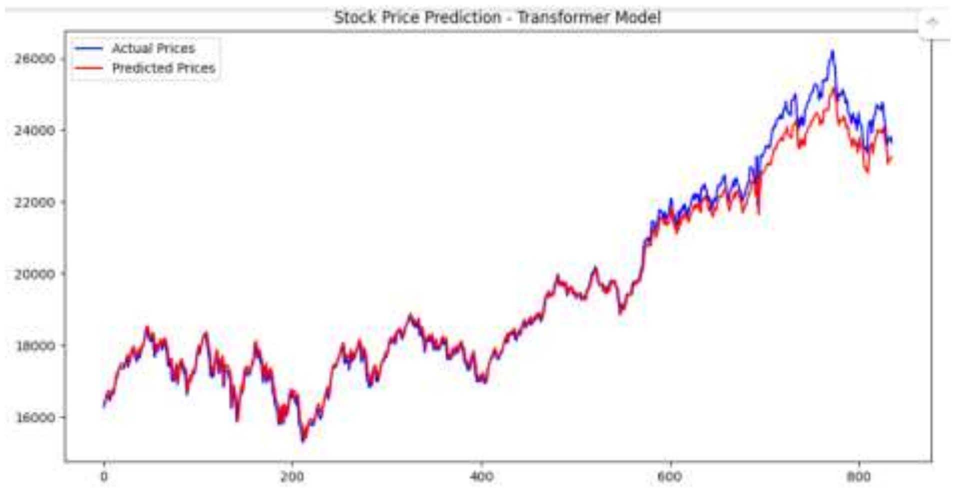


Figure 6.1: Transformer

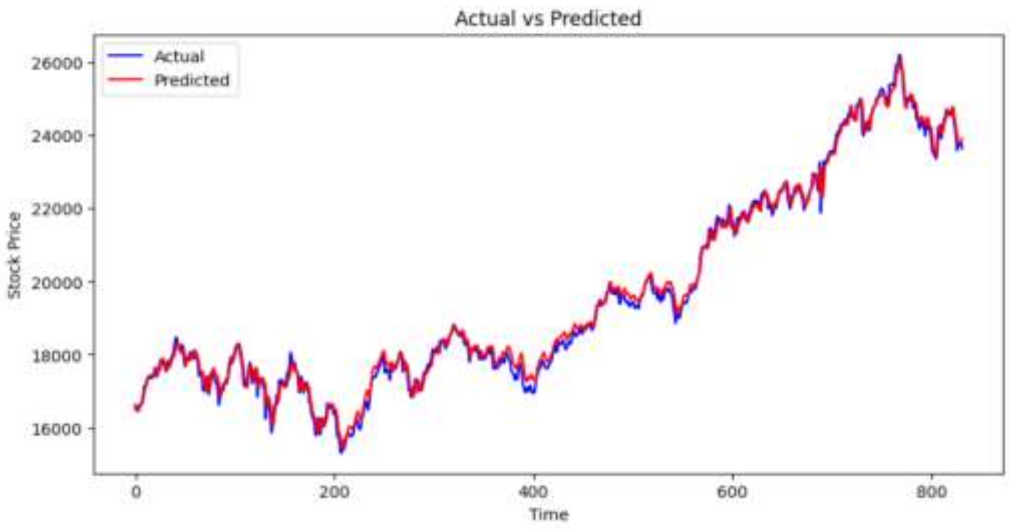


Figure 6.4: Hybrid model

The analysis presented in the comparison table and graphs clearly indicates that the LSTM model utilizing a sequence length of 60 delivered the highest performance among all models evaluated. It achieved the lowest Mean Absolute Error (MAE) of 116.82 and Root Mean Square Error (RMSE) of 167.68, signifying its superior accuracy in predictions. This model effectively captured stock price trends and demonstrated robust generalization across the dataset.

In contrast, the Transformer model faced challenges with trend alignment, resulting in a considerably higher error rate. Although the hybrid LSTM-Transformer model showed some enhancements compared to the standalone Transformer, it still fell short of surpassing the optimized LSTM model. Furthermore, while the RNN model outperformed the Transformers, it was less adept than the LSTM in managing long-term dependencies.

In summary, the LSTM model with a sequence length of 60, utilizing only the Close price as input features, emerged as the most dependable option for stock price prediction.

7 Conclusion, Recommendation and Future work

Conclusion

This project explored various deep learning models for predicting the closing price of the Nifty 50 index, including Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), Transformers, and hybrid LSTM-Transformer models. Among the models evaluated, the LSTM model utilizing a 60-day sequence length exhibited the highest level of accuracy and reliability, successfully capturing temporal dependencies and intricate patterns in stock price fluctuations. It significantly surpassed other deep learning models in terms of Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), establishing itself as the most effective model for short-term stock price predictions. Furthermore, the LSTM model demonstrated strong generalization on the test dataset, with predicted prices closely aligning with actual stock prices, establishing itself as the most effective model for short-term stock price predictions.

In addition, we assessed traditional statistical models such as ARIMA and SARIMA. Although these methods are commonly employed in time series forecasting, they struggled to address the nonlinear and highly volatile characteristics of financial markets, which deep learning models are more adept at handling.

Furthermore, we emphasized the practical implementation of our solution. A user-friendly mobile application was created, incorporating the top-performing LSTM model. This application not only retrieves real-time Nifty 50 data and forecasts the next day's closing price but also integrates sentiment analysis from financial news and a Fear & Greed Index based on technical indicators. These additional functionalities are designed to aid users especially beginners in making well-informed trading decisions.

Recommendations

The LSTM model utilizing a 60-day sequence length yielded the highest accuracy in forecasts, making it ideal for short-term market evaluations. The mobile application created in this project combines these predictions with an analysis of market sentiment, delivering valuable insights into stock trends for users. Nevertheless, it is important to treat these forecasts as a guideline

rather than the sole basis for decision-making, as external influences such as economic developments and geopolitical events can greatly affect stock prices.
Feature engineering was vital to the model's effectiveness, as not all technical indicators had a positive impact, underscoring the necessity for careful selection. Although Transformers have achieved success in various fields, they faced challenges in aligning trends within financial time series, while LSTMs exhibited superior predictive performance. Effective data preprocessing, which includes scaling and addressing missing values, was essential for ensuring accuracy. Furthermore, hyperparameter tuning such as refining learning rates, dropout rates, and sequence lengths was crucial for enhancing model performance. These findings lay the groundwork for advancing stock price forecasting through deep learning techniques.

Future Work

Future enhancements could involve the inclusion of macroeconomic indicators, interest rates, and global market trends to improve the accuracy of predictions. By integrating real-time sentiment analysis from news sources and social media trends, one can gain more profound insights into the market. Broadening the focus beyond the Nifty 50 to encompass individual stocks and sector indices would increase the model's relevance.

From a technical perspective, investigating hybrid models such as CNN-LSTM or advanced Transformers for time series forecasting could enhance performance. Implementing explain ability techniques would also render predictions more understandable.

For practical applications, the mobile application could be upgraded with real-time data, interactive dashboards, and customized alerts, thereby becoming a more valuable resource for investors. Additionally, optimizing computational efficiency would facilitate quicker and more scalable predictions.

8 References

[1] C. Wang, Y. Chen, S. Zhang, Q. Zhang, “Stock market index prediction using deep Transformer model,” Expert Systems with Applications, vol. 208, pp. 118128, Dec. 2022. [Online]. Available: https://dl.acm.org/doi/10.1016/j.eswa.2022.118128

[2] Q. Zhang, C. Qin, Y. Zhang, F. Bao, C. Zhang, P. Liu, “Transformer-based attention network for stock movement prediction,” Expert Systems with Applications, vol. 202, pp. 117239, Sep. 2022. [Online]. Available: https://dl.acm.org/doi/10.1016/j.eswa.2022.117239

[3] C. Yañez, W. Kristjanpoller, and M. C. Minutolo, "Stock market index prediction using transformer neural network models and frequency decomposition," Neural Computing and Applications,vol.36,pp.15777–15797,May2024.[Online].Available: https://link.springer.com/article/10.1007/s00521-024-09931-4

[4] E. Gothai, R. Thamilselvan, P. Natesan, and K. Vignesh, "Global Stock Market Prediction Using Transformer-Based Deep Learning Techniques," in 2023 International Conference on Intelligent Systems and Computer Vision (ISCV), IEEE, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10724893

[5] X. Hu, "Stock Price Prediction Based on Temporal Fusion Transformer," 2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI),Taiyuan,China,2021,pp.97–102,[Online].Available: https://ieeexplore.ieee.org/abstract/document/9731073

[6] S. S. Sikarwar, S. Jaswal, R. Sharma, G. Ganesan, R. Ganguli, and M. K. Mahawar, “Stock Price Prediction Using Transformers,” 2024 4th International Conference on Advancement in Electronics & Communication Engineering (AECE), Ghaziabad, India, Nov. 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10911285

[7] A. Moghar and M. Hamiche, “Stock Market Prediction Using LSTM Recurrent Neural Network,” Procedia Computer Science, vol. 170, pp. 1168–1173, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920304865

[8] H. N. Bhandari, B. Rimal, N. R. Pokhrel, R. Rimal, K. R. Dahal, and R. K. C. Khatri, “Predicting stock market index using LSTM,” Machine Learning with Applications, vol. 9, 100320,Sep.2022.[Online].Available:https://www.sciencedirect.com/science/article/pii/S2666827022000378

[9] Z. Fathali, Z. Kodia, and L. B. Said, “Stock Market Prediction of NIFTY 50 Index Applying Machine Learning Techniques,” Applied Artificial Intelligence, vol. 36, no. 1, 2022. [Online].Available:https://www.tandfonline.com/doi/full/10.1080/08839514.2022.2111134#d1e203

[10] A. K. Jadoon, T. Mahmood, A. Sarwar, M. F. Javaid, and M. Iqbal, "Prediction of Stock Market Movement Using Long Short-Term Memory (LSTM) Artificial Neural Network: Analysis of KSE 100 Index," Pakistan Journal of Life and Social Sciences, vol. 22, no. 1, pp. 107–119, 2024. [Online]. Available: https://pjlss.edu.pk/pdf_files/2024_1/107-119.pdf

[11] S. Mehtab, J. Sen, and A. Dutta, "Stock price prediction using machine learning and LSTM-based deep learning models," Department of Data Science and Artificial Intelligence, Praxis Business School, Kolkata, India, vol. 8, no. 2, pp. 45-63, Aug. 2021. [Online]. Available: https://arxiv.org/pdf/2009.10819

[12] S. H. Jafar, S. Akhtar, H. El-Chaarani, P. A. Khan, and R. Binsaddig, “Forecasting of NIFTY 50 Index Price by Using Backward Elimination with an LSTM Model,” J. Risk Financial Manag,vol. 16, no. 10, p. 423, Sep. 2023. [Online]. Available: https://www.mdpi.com/1911-8074/16/10/423

[13] S. Wang, “A Stock Price Prediction Method Based on BiLSTM and Improved Transformer,” IEEE Access, vol. 11, pp. 104211–104223, Jul. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10185006

[14] C. Li and G. Qian, "Stock Price Prediction Using a Frequency Decomposition Based GRU Transformer Neural Network," Applied Sciences, vol. 13, no. 1, p. 222, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/13/1/222

[15] D. Daiya and C. Lin, “Stock Movement Prediction and Portfolio Management via Multimodal Learning with Transformer,” in Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, Jun. 2021, pp. 8048–8052. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9414893

[16] Y. Zhang, J. Li, H. Wang, and S.-C. T. Choi, "Sentiment-guided adversarial learning for stock price prediction," Beijing Institute of Technology, Haidian District, Beijing, China; Kamakura Corporation, Honolulu, HI, United States; Department of Applied Mathematics, Illinois Institute of Technology, Chicago, IL, United States, Feb.2021. Published: May 26, 2021. [Online]. Available:

http://ndl.iitkgp.ac.in/re_document/frontiers/frontiers/f003a7b6f888062f1367e721b9a53842?e=16|stock%20prediction%20using%20sentiment%20analysis|||

[17] M. K. Daradkeh, "A Hybrid Data Analytics Framework with Sentiment Convergence and Multi-Feature Fusion for Stock Trend Prediction," Dubai Financial Market, Jan. 13, 2022. [Online].Available:http://ndl.iitkgp.ac.in/re_document/mdpi/scilit_mdpi/245fb4029ab5c458a48b07a3b0776213?e=1|stock%20prediction%20using%20sentiment%20analysis|||

[18] C.-R. Ko and H.-T. Chang, "LSTM-based Sentiment Analysis for Stock Price Forecast," Artificial Intelligence, Data Science, Databases, Emerging Technologies, Mar. 11, 2021. [Online].Available:http://ndl.iitkgp.ac.in/re_document/doaj/doaj/24f83f51eb7e40b1a74bb74c761b2184?e=5|stock%20prediction%20using%20sentiment%20analysis|||

[19] S. Gite, H. Khatavkar, K. Kotecha, S. Srivastava, P. Maheshwari, and N. Pandey, "Explainable Stock Prices Prediction from Financial News Articles Using Sentiment Analysis," Symbiosis International (Deemed University), Pune, Maharashtra, India, Jan. 2021. [Online]. Available: https://peerj.com/articles/cs-340/

[20] V. Mahajan, S. Thakan, and A. Malik, "Modeling and forecasting the volatility of NIFTY 50 using GARCH and RNN models," Economies, vol. 10, no. 5, p. 102, Apr. 2022. [Online]. Available: https://www.mdpi.com/2227-7099/10/5/102

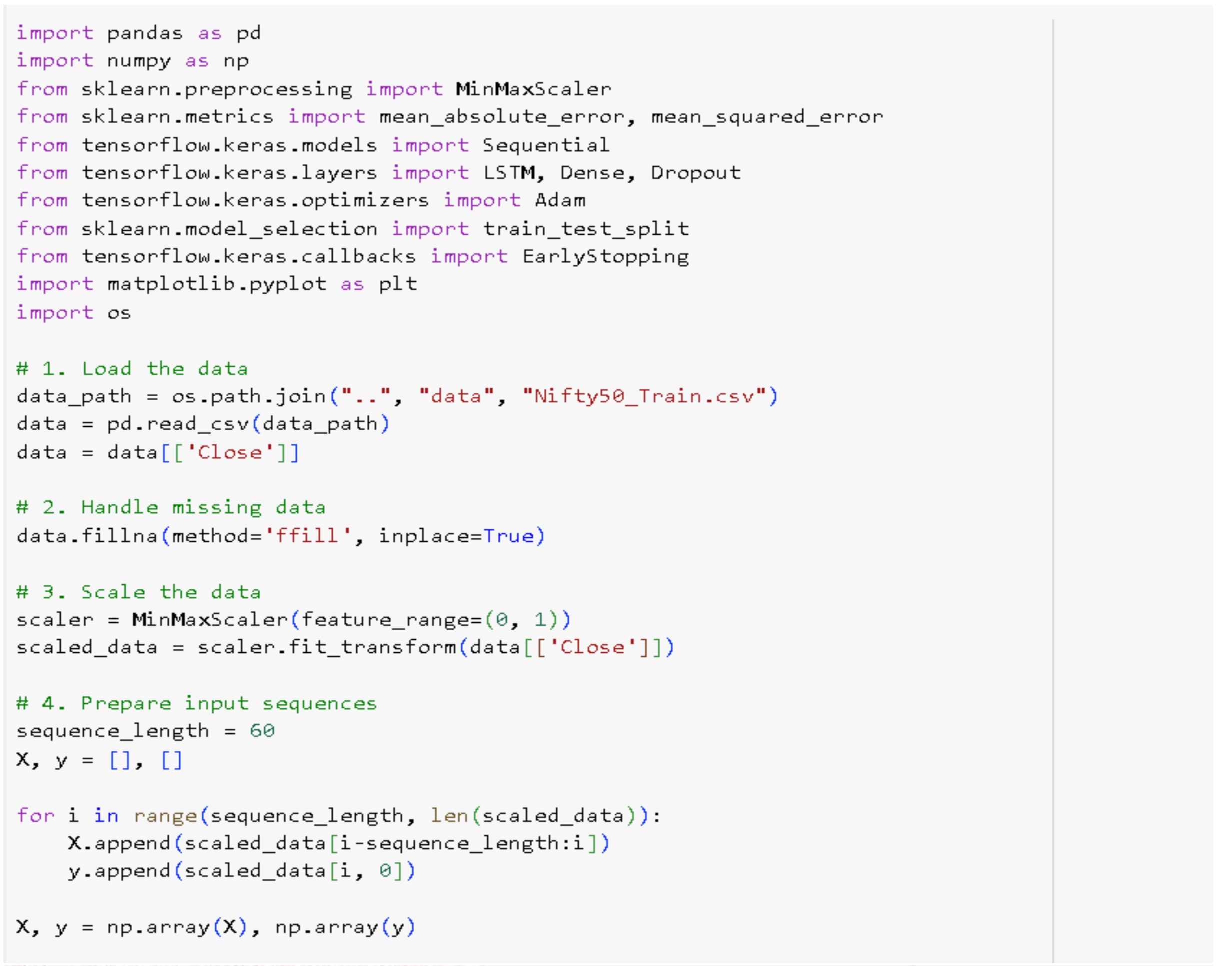
[21] M. Bansal, A. Goyal, and A. Choudhary, "Stock Market Prediction with High Accuracy using Machine Learning Techniques," *Procedia Computer Science*, vol. 215, pp. 247-265, 2022.[Online].Available:https://www.sciencedirect.com/science/article/pii/S1877050922020993

[22] Y. Ma, R. Han, and W. Wang, "Portfolio optimization with return prediction using deep learning and machine learning," Expert Syst. Appl., vol. 162, Aug. 2020, Art. no. 113973. Accessed:Aug.24,2024.[Online].Available:https://www.sciencedirect.com/science/article/pii/S0957417420307521

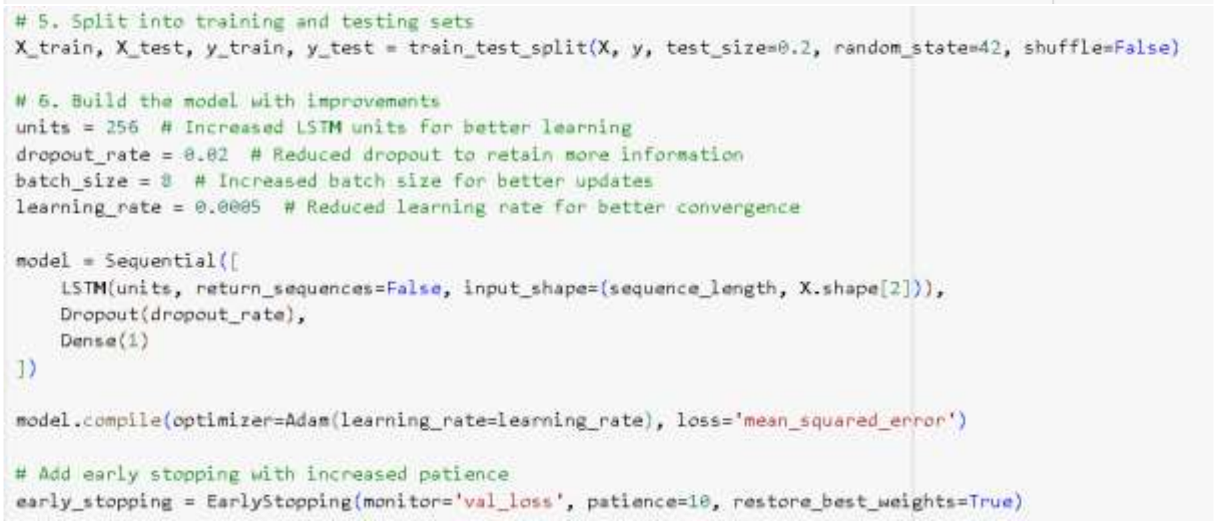
[23] Yahoo Finance – NIFTY 50 (^NSEI) Historical Data
URL:https://finance.yahoo.com/quote/%5ENSEI/history/

9 Appendix

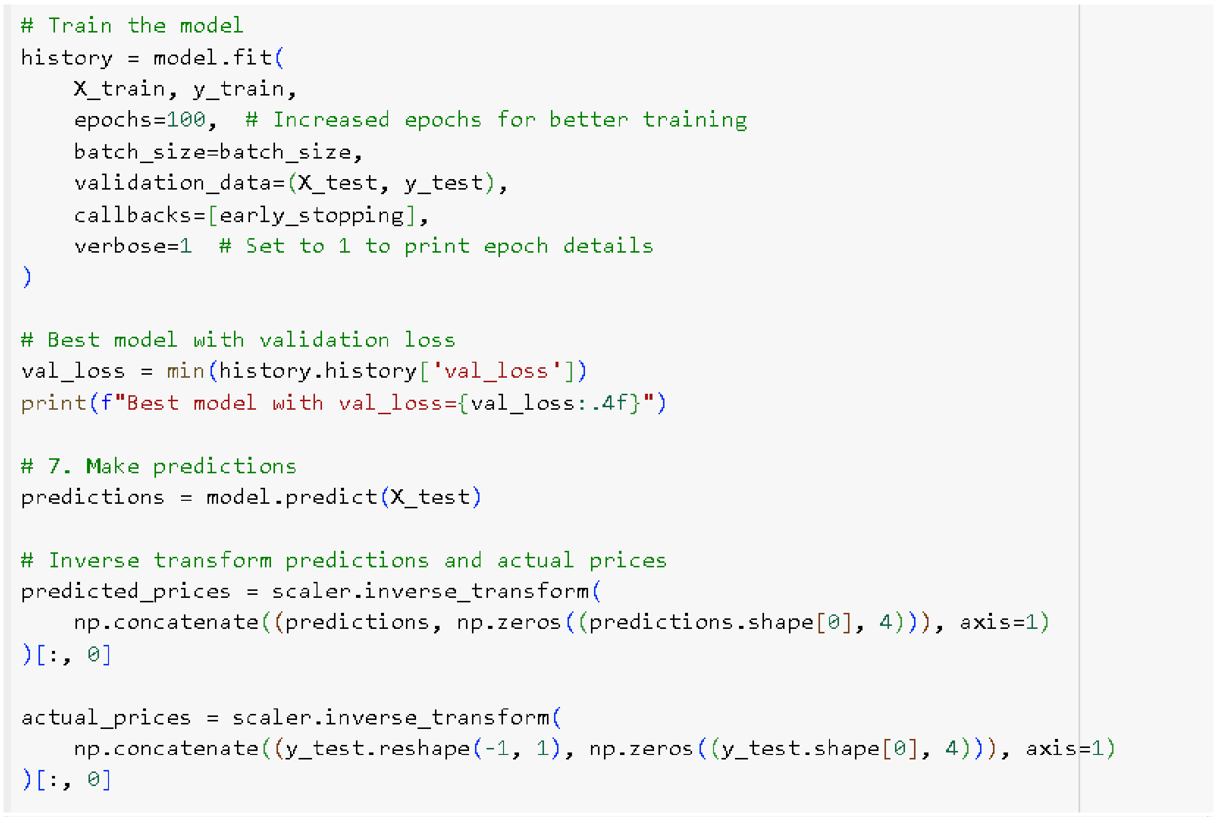
9.1 Appendix A: LSTM (Best) Model Code



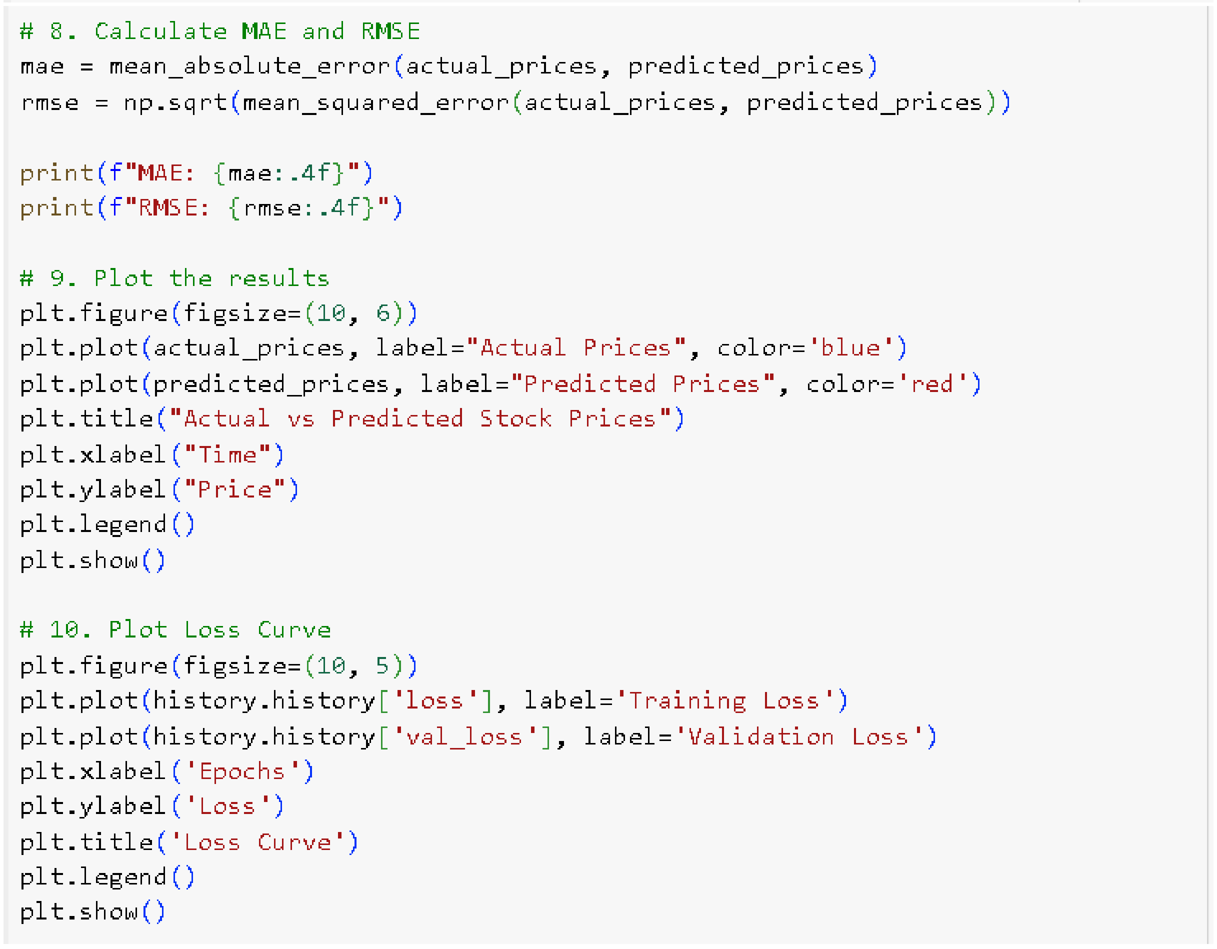
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import os

# 1. Load the data
data_path = os.path.join("..", "data", "Nifty50_Train.csv")
data = pd.read_csv(data_path)
data = data[['Close']]

# 2. Handle missing data
data.fillna(method='ffill', inplace=True)

# 3. Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']])

# 4. Prepare input sequences
sequence_length = 60
X, y = [], []

for i in range(sequence_length, len(scaled_data)):
    X.append(scaled_data[i-sequence_length:i])
    y.append(scaled_data[i, 0])

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



```
# 5. Split 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, shuffle=False)

# 6. Build the model with improvements
units = 256  # Increased LSTM units for better learning
dropout_rate = 0.02  # Reduced dropout to retain more information
batch_size = 8  # Increased batch size for better updates
learning_rate = 0.0005  # Reduced learning rate for better convergence

model = Sequential([
    LSTM(units, return_sequences=False, input_shape=(sequence_length, X.shape[2])),
    Dropout(dropout_rate),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mean_squared_error')

# Add early stopping with increased patience
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

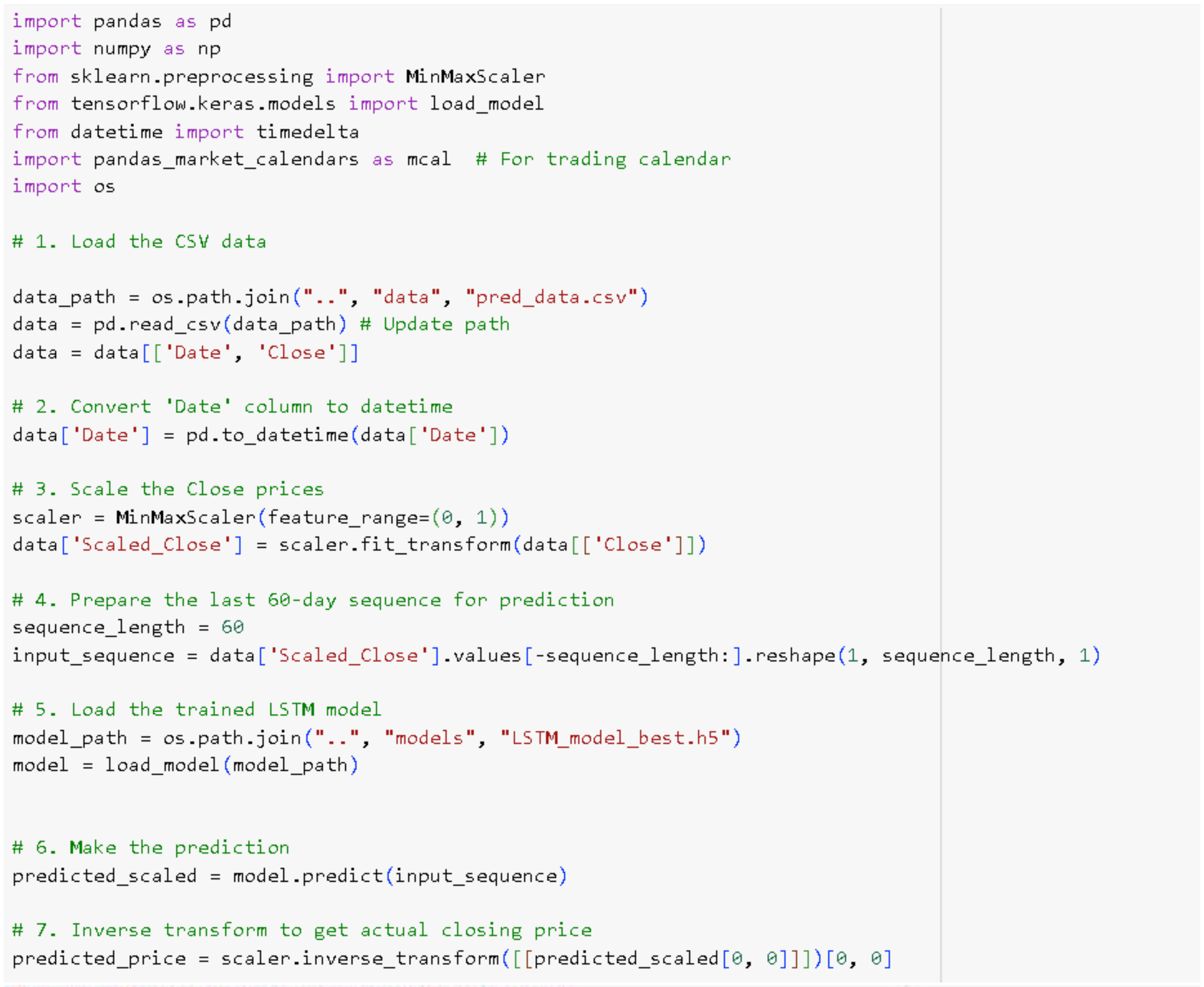


```python
# Train the model
history = model.fit(
    X_train, y_train,
    epochs=100,  # Increased epochs for better training
    batch_size=batch_size,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping],
    verbose=1  # Set to 1 to print epoch details
)

# Best model with validation loss
val_loss = min(history.history['val_loss'])
print(f"Best model with val_loss={val_loss:.4f}")

# 7. Make predictions
predictions = model.predict(X_test)

# Inverse transform predictions and actual prices
predicted_prices = scaler.inverse_transform(
    np.concatenate((predictions, np.zeros((predictions.shape[0], 4))), axis=1)
)[:, 0]

actual_prices = scaler.inverse_transform(
    np.concatenate((y_test.reshape(-1, 1), np.zeros((y_test.shape[0], 4))), axis=1)
)[:, 0]
```

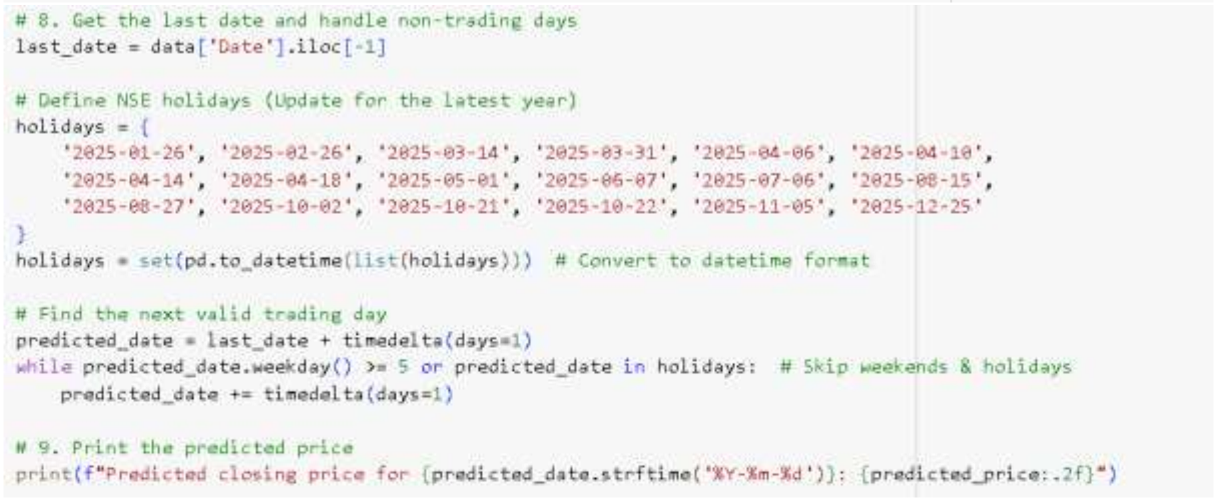


```python
# 8. Calculate MAE and RMSE
mae = mean_absolute_error(actual_prices, predicted_prices)
rmse = np.sqrt(mean_squared_error(actual_prices, predicted_prices))

print(f"MAE: {mae:.4f}")
print(f"RMSE: {rmse:.4f}")

# 9. Plot the results
plt.figure(figsize=(10, 6))
plt.plot(actual_prices, label="Actual Prices", color='blue')
plt.plot(predicted_prices, label="Predicted Prices", color='red')
plt.title("Actual vs Predicted Stock Prices")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.show()

# 10. Plot Loss Curve
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.legend()
plt.show()
```

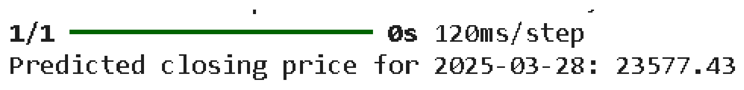
9.2 Appendix B: LSTM Prediction Code and Results



```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model
from datetime import timedelta
import pandas_market_calendars as mcal  # For trading calendar
import os

# 1. Load the CSV data

data_path = os.path.join("..", "data", "pred_data.csv")
data = pd.read_csv(data_path) # Update path
data = data[['Date', 'Close']]

# 2. Convert 'Date' column to datetime
data['Date'] = pd.to_datetime(data['Date'])

# 3. Scale the Close prices
scaler = MinMaxScaler(feature_range=(0, 1))
data['Scaled_Close'] = scaler.fit_transform(data[['Close']])

# 4. Prepare the last 60-day sequence for prediction
sequence_length = 60
input_sequence = data['Scaled_Close'].values[-sequence_length:].reshape(1, sequence_length, 1)

# 5. Load the trained LSTM model
model_path = os.path.join("..", "models", "LSTM_model_best.h5")
model = load_model(model_path)


# 6. Make the prediction
predicted_scaled = model.predict(input_sequence)

# 7. Inverse transform to get actual closing price
predicted_price = scaler.inverse_transform([[predicted_scaled[0, 0]]])[0, 0]
```



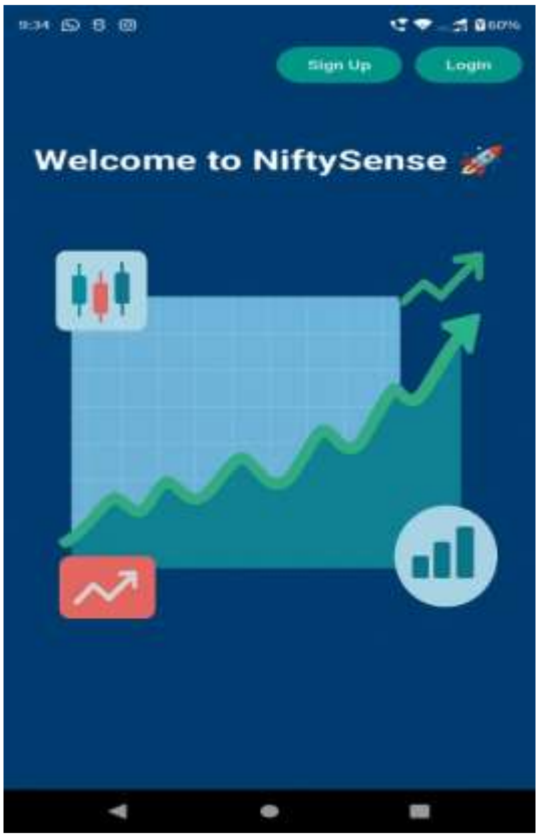
```
# 8. Get the last date and handle non-trading days
last_date = data['Date'].iloc[-1]

# Define NSE holidays (Update for the latest year)
holidays = {
    '2025-01-26', '2025-02-26', '2025-03-14', '2025-03-31', '2025-04-06', '2025-04-10',
    '2025-04-14', '2025-04-18', '2025-05-01', '2025-06-07', '2025-07-06', '2025-08-15',
    '2025-08-27', '2025-10-02', '2025-10-21', '2025-10-22', '2025-11-05', '2025-12-25'
}
holidays = set(pd.to_datetime(list(holidays)))  # Convert to datetime format

# Find the next valid trading day
predicted_date = last_date + timedelta(days=1)
while predicted_date.weekday() >= 5 or predicted_date in holidays:  # Skip weekends & holidays
    predicted_date += timedelta(days=1)

# 9. Print the predicted price
print(f"Predicted closing price for {predicted_date.strftime('%Y-%m-%d')}: {predicted_price:.2f}")
```

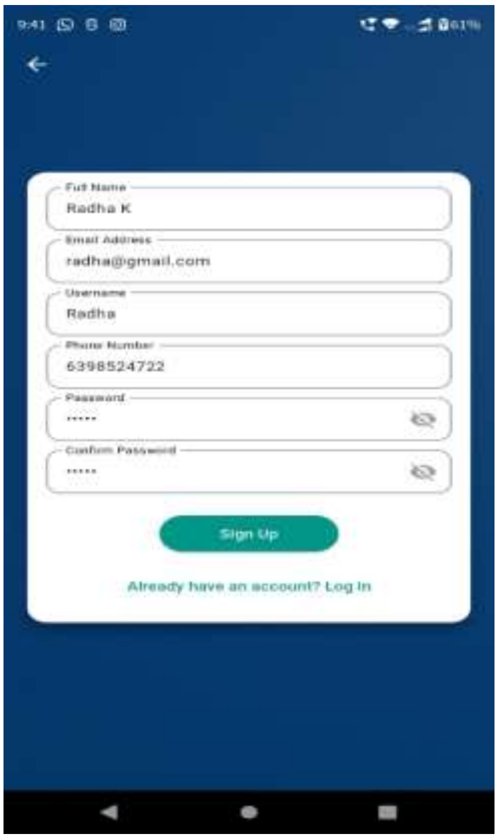


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
Predicted closing price for 2025-03-28: 23577.43
```

9.3 Appendix C: Mobile Application

Figure A.1: Screenshots of the Mobile Application Interface




Full Name: Radha K

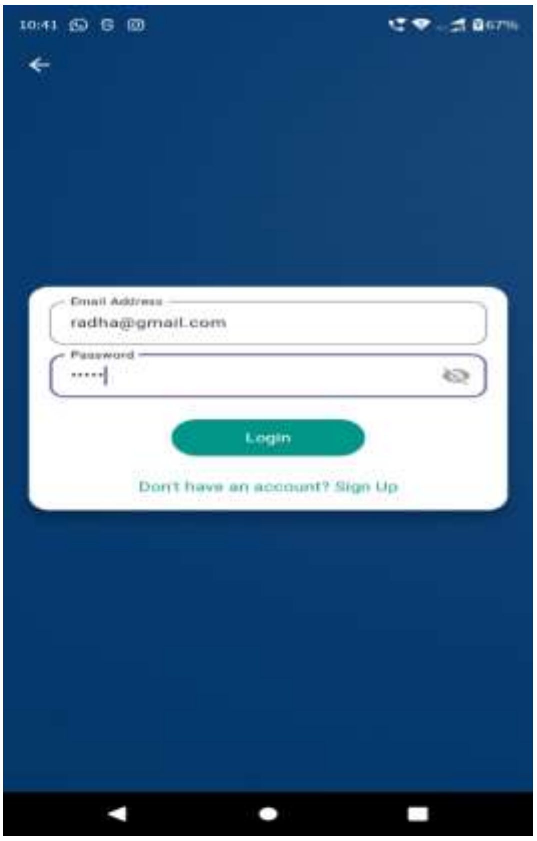
Email Address: radha@gmail.com

Username: Radha

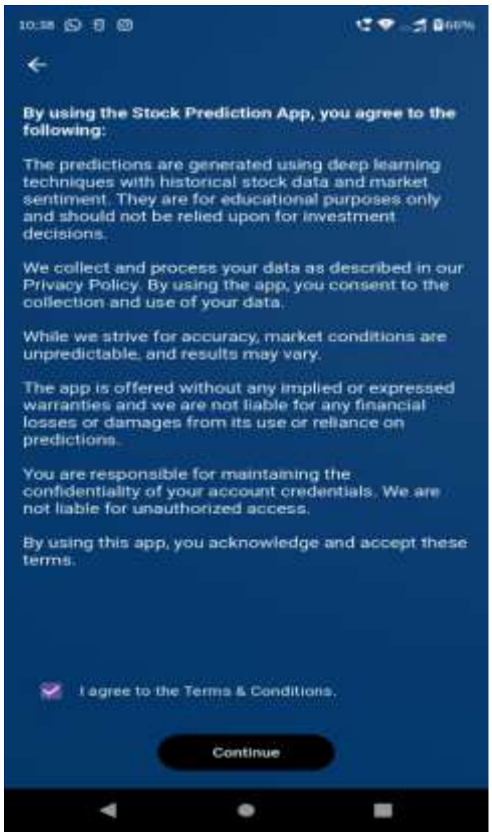
Phone Number: 6398524722

Password: •••••

Confirm Password: •••••


Email Address: radha@gmail.com

Password: •••••


By using the Stock Prediction App, you agree to the following:

The predictions are generated using deep learning techniques with historical stock data and market sentiment. They are for educational purposes only and should not be relied upon for investment decisions.

We collect and process your data as described in our Privacy Policy. By using the app, you consent to the collection and use of your data.

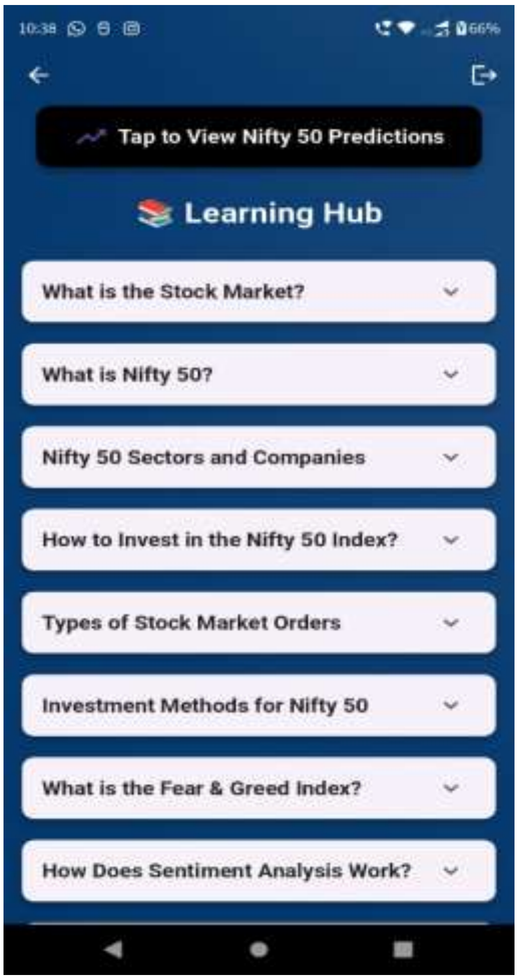
While we strive for accuracy, market conditions are unpredictable, and results may vary.

The app is offered without any implied or expressed warranties and we are not liable for any financial losses or damages from its use or reliance on predictions.

You are responsible for maintaining the confidentiality of your account credentials. We are not liable for unauthorized access.

By using this app, you acknowledge and accept these terms.

[✓] I agree to the Terms & Conditions.



Tap to View Nifty 50 Predictions

Learning Hub

What is the Stock Market?

What is Nifty 50?

Nifty 50 Sectors and Companies

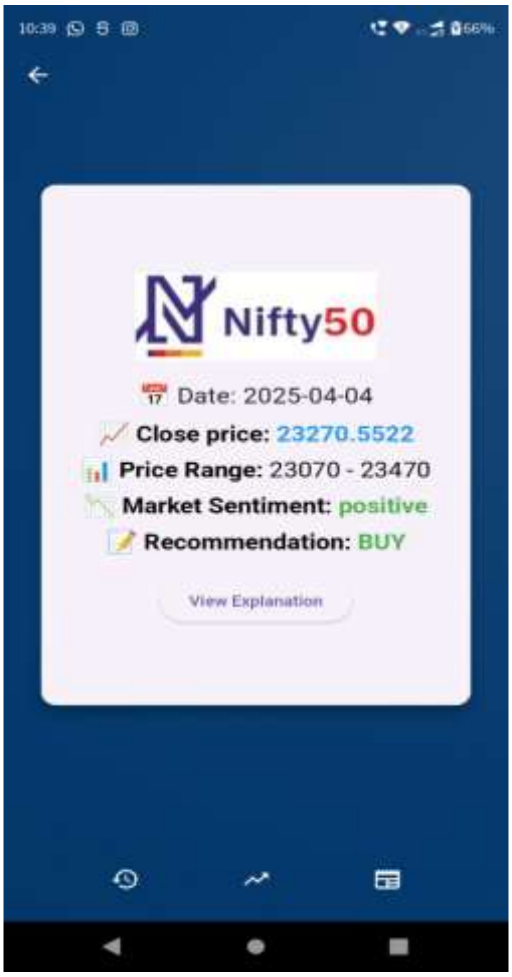
How to Invest in the Nifty 50 Index?

Types of Stock Market Orders

Investment Methods for Nifty 50

What is the Fear & Greed Index?

How Does Sentiment Analysis Work?



Nifty50

Date: 2025-04-04

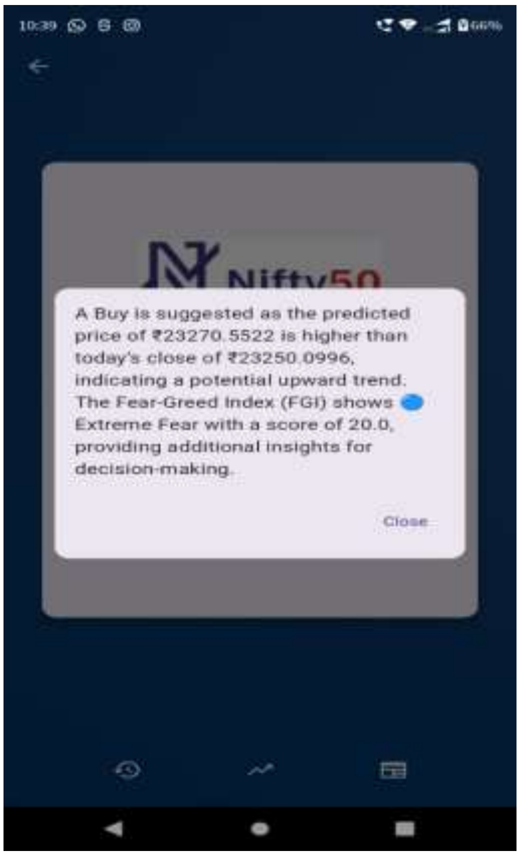
Close price: 23270.5522

Price Range: 23070 - 23470

Market Sentiment: positive

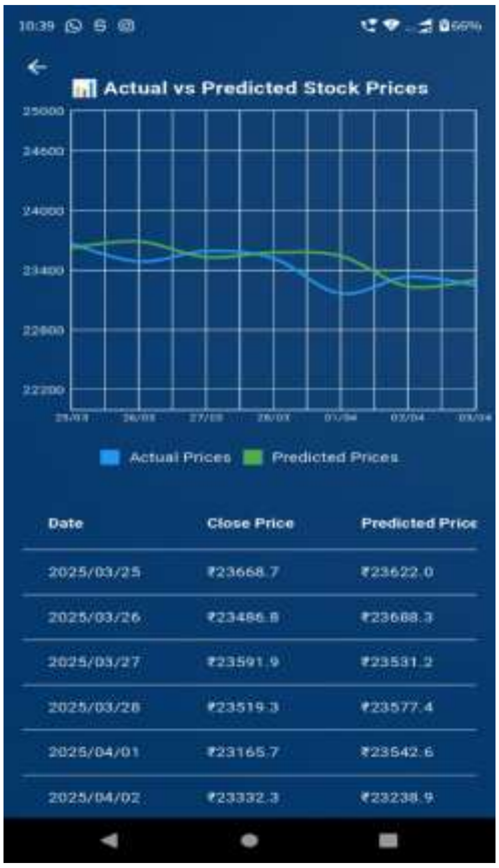
Recommendation: BUY

View Explanation

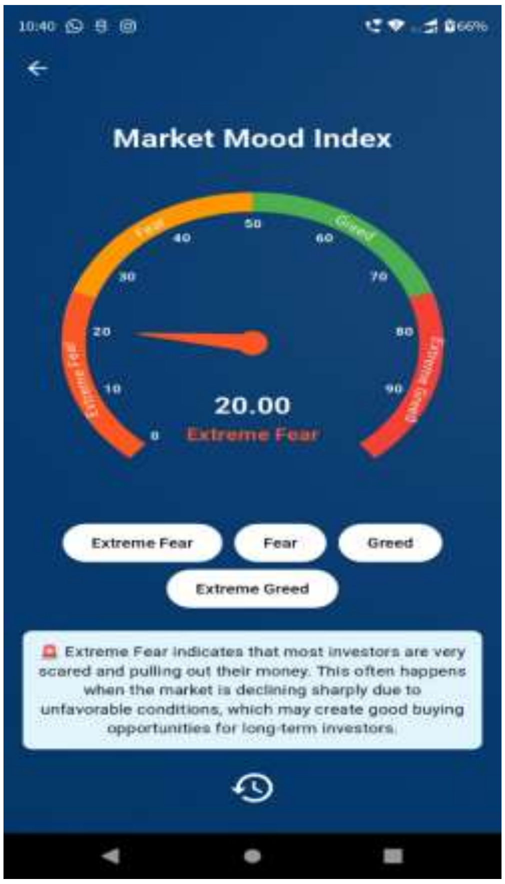


A Buy is suggested as the predicted price of ₹23270.5522 is higher than today's close of ₹23250.0996, indicating a potential upward trend. The Fear-Greed Index (FGI) shows 🔵 Extreme Fear with a score of 20.0, providing additional insights for decision-making.

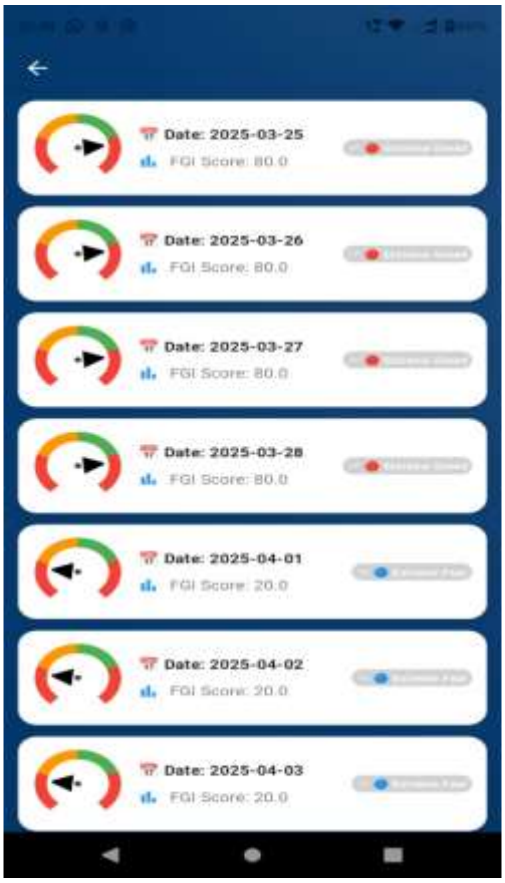
Close



Date	Close Price	Predicted Price
2025/03/25	₹23668.7	₹23622.0
2025/03/26	₹23486.8	₹23688.3
2025/03/27	₹23591.9	₹23531.2
2025/03/28	₹23519.3	₹23577.4
2025/04/01	₹23165.7	₹23542.6
2025/04/02	₹23332.3	₹23238.9



🚨 Extreme Fear indicates that most investors are very scared and pulling out their money. This often happens when the market is declining sharply due to unfavorable conditions, which may create good buying opportunities for long-term investors.



Date: 2025-03-25
FGI Score: 80.0

Date: 2025-03-26
FGI Score: 80.0

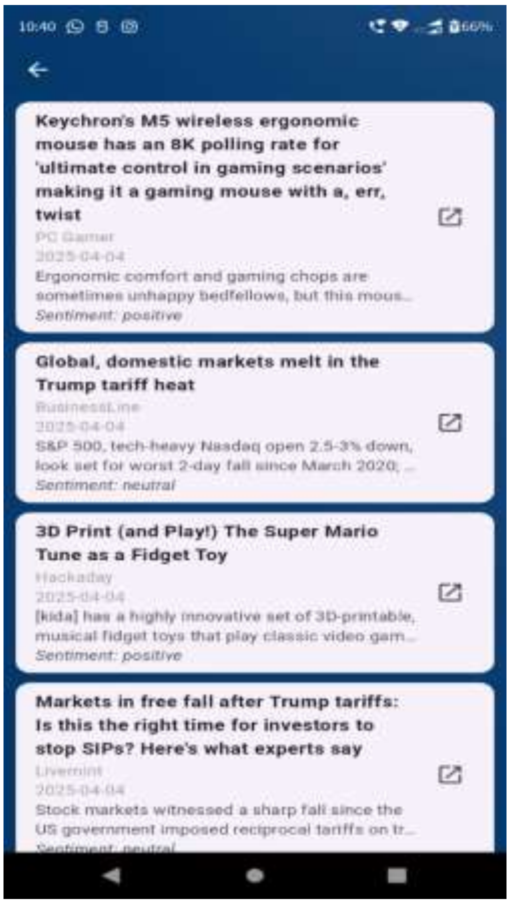
Date: 2025-03-27
FGI Score: 80.0

Date: 2025-03-28
FGI Score: 80.0

Date: 2025-04-01
FGI Score: 20.0

Date: 2025-04-02
FGI Score: 20.0

Date: 2025-04-03
FGI Score: 20.0



Keychron's M5 wireless ergonomic mouse has an 8K polling rate for 'ultimate control in gaming scenarios' making it a gaming mouse with a, err, twist
PC Gamer
2025-04-04
Ergonomic comfort and gaming chops are sometimes unhappy bedfellows, but this mous...
Sentiment: positive

Global, domestic markets melt in the Trump tariff heat
BusinessLine
2025-04-04
S&P 500, tech-heavy Nasdaq open 2.5-3% down, look set for worst 2-day fall since March 2020; ...
Sentiment: neutral

3D Print (and Play!) The Super Mario Tune as a Fidget Toy
Hackaday
2025-04-04
[kida] has a highly innovative set of 3D-printable, musical fidget toys that play classic video gam...
Sentiment: positive

Markets in free fall after Trump tariffs: Is this the right time for investors to stop SIPs? Here's what experts say
Livemint
2025-04-04
Stock markets witnessed a sharp fall since the US government imposed reciprocal tariffs on tr...
Sentiment: neutral

GITAM

DEEMED TO BE UNIVERSITY