Stock Market FnO Price Forecasting

A Minor Project Report
submitted in partial fulfillment of the requirements for
the award of the degree of

Bachelor of Engineering

in

Artificial Intelligence and Data Science

By

Chinnam Chandana (1601-21-771-076)

Pasunuri Kathyayini (1601-21-771-088)

Ambekar Tejas (1601-21-771-095)

Under the esteemed guidance of

Dr. D. Lakshmi Srinivasa Reddy

Associate Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
HYDERABAD – 500075
DECEMBER 2023

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
HYDERABAD – 500075

INSTITUTE VISION

"To be the center of excellence in technical education and research".

INSTITUTE MISSION

"To address the emerging needs through quality technical education and advanced research".

DEPARTMENT VISION

"To be a globally recognized center of excellence in the field of Artificial Intelligence and Data Science that produces innovative pioneers and research experts capable of addressing complex real-world challenges and contributing to the socio-economic development of the nation."

DEPARTMENT MISSION

1. To provide cutting-edge education in the field of Artificial Intelligence and Data Science that is rooted in ethical and moral values.

2. To establish strong partnerships with industries and research organizations in the field of Artificial Intelligence and Data Science, and to excel in the emerging areas of research by creating innovative solutions.

3. To cultivate a strong sense of social responsibility among students, fostering their inclination to utilize their knowledge and skills for the betterment of society.

4. To motivate and mentor students to become trailblazers in Artificial Intelligence and Data Science, and develop an entrepreneurial mindset that nurtures innovation and creativity.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

HYDERABAD – 500075

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Graduates of AI & DS will be able to:

1. Adapt emerging technologies of Artificial Intelligence & Data Science and develop state of the art solutions in the fields of Manufacturing, Agriculture, Health-care, Education, and Cyber Security.
2. Exhibit professional leadership qualities to excel in interdisciplinary domains.
3. Possess human values, professional ethics, application-oriented skills, and engage in lifelong learning.
4. Contribute to the research community to meet the needs of public and private sectors.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After successful completion of the program, students will be able to:

1. Exhibit proficiency of Artificial Intelligence and Data Science in providing sustainable solutions by adapting to societal, environmental and ethical concerns to real world problems.
2. Develop professional skills in the thrust areas like ANN and Deep learning, Robotics, Internet of Things and Big Data Analytics.
3. Pursue higher studies in Artificial Intelligence and Data Science in reputed Universities and to work in research establishments.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

HYDERABAD – 500075

PROGRAM OUTCOMES

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems

2. **Problem analysis:** Identify, formulate, review, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

HYDERABAD – 500075

MINOR PROJECT-II

COURSE OBJECTIVES

1. To enable students to learn by doing.
2. To develop capability to analyse and solve real world problems.
3. To apply innovative ideas of the students.
4. To learn the ability to build a data science project.
5. To impart team building and management skills among students and instill writing and presentation skills for completing the project.

COURSE OUTCOMES

Upon successful completion of this course, students will be able to:

1. Interpret Literature with the purpose of formulating a project proposal.
2. Develop the ability to identify and formulate problems by applying diverse technical knowledge skills.
3. Apply the fundamental knowledge gained in the curriculum to model, design and implement a Data Science project.
4. Build a prototype by choosing appropriate technologies to meet the identified requirements.
5. Plan to work as a team and to focus on getting a working project done and submit a report within a stipulated period of time to the Departmental Committee.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
HYDERABAD – 500075

CO-PO MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	3	2	2	3	3	3	2	2	2	3	3
CO2	3	3	3	3	3	3	3	2	2	2	3	3
CO3	3	3	3	3	3	3	3	2	2	2	3	3
CO4	3	3	2	3	3	3	3	2	3	3	3	3
CO5	1	2	2	2	3	3	-	-	3	3	-	2

Mapping of Course Outcomes with Program Outcomes

CO-PSO MAPPING

	PSO1	PSO2	PSO3
CO1	2	-	3
CO2	3	3	3
CO3	3	3	3
CO4	3	3	3
CO5	1	-	-

Mapping of Course Outcomes with Program Specific Outcomes

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

HYDERABAD – 500075

DECLARATION CERTIFICATE

We hereby declare that the project titled **Stock Market FnO Price Forecasting** submitted by us to the **Artificial Intelligence and Data Science**, **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY, HYDERABAD** in partial fulfillment of the requirements for the award of **Bachelor of Engineering** is a bona-fide record of the work carried out by us under the supervision of **Dr. D. Lakshmi Srinivasa Reddy**. We further declare that the work reported in this project, has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Project Associates

Chinnam Chandana (1601-21-771-076)

Pasunuri Kathyayini (1601-21-771-088)

Ambekar Tejas (1601-21-771-095)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

HYDERABAD – 500075

BONAFIDE CERTIFICATE

This is to certify that the project titled **Stock Market FnO Price Forecasting** is a bonafide record of the work done by

Chinnam Chandana (1601-21-771-076)

Pasunuri Kathyayini (1601-21-771-088)

Ambekar Tejas (1601-21-771-095)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering** in **Artificial Intelligence and Data Science** to the **CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY, HYDERABAD** carried out under my guidance and supervision during the year 2023-24. The results presented in this project report have not been submitted to any other university or Institute for the award of any degree.

Mrs. V. Krishna Aravinda
Project Co-ordinator

Dr.K.Ramana
Head of the Department

Submitted for VI Semester Minor-Project-II viva-voce examination held on 18-05-2024

Examiner-1

Examiner-2

ABSTRACT

In the dynamic realm of stock market trading, optimizing positions to maximize gains and minimize losses is paramount for traders seeking success. Historical data analysis of option chains for prominent indices like Nifty, BankNifty, and FinNifty offers valuable insights into market trends and potential opportunities. This project aims to leverage such data to develop a strategic position optimization framework. Stock market traders often rely on retrospective data to inform their trading strategies. However, manually analyzing vast amounts of option chain data spanning several years can be daunting and time-consuming. This project seeks to automate this process by developing a data-driven approach to optimize trading positions by forecasting future prices. By analyzing historical trends and patterns in Nifty, BankNifty, and FinNifty option chains, the system will recommend strategic positions that increase gains while minimizing potential losses.

Currently, traders may employ manual or semi-automated methods to analyze historical data and optimize their positions. However, these approaches are often limited in their scope and efficiency. By introducing a comprehensive algorithmic solution, this project aims to streamline the optimization process and provide traders with actionable insights. The proposed solution will involve collecting and preprocessing historical option chain data for BankNifty Index. Deep learning algorithms will then be deployed to analyze this data and identify patterns indicative of potential market movements. Using these insights, the system will generate future trading positions that align with the trader's risk tolerance and investment objectives. The development of this project will predominantly utilize R for its robust capabilities in data analysis and machine learning. R packages such as tidyverse, caret, and xgboost will facilitate data manipulation, modeling, and prediction tasks. For data visualization, ggplot2 will be employed to create insightful plots and visualizations. Deep Learning Models for prediction and forecasting of prices are be built using R and other required libraries.

Keywords : Stock Market, Position, BankNifty, Forecasting, Profit, R, Optimization

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to the following people for guiding us through this course and without whom this project and the results achieved from it would not have reached completion.

Dr. D. Lakshmi Srinivasa Reddy, Assistant Professor, Department of Artificial Intelligence and Data Science, for helping us and guiding us in the course of this project. Without her guidance, we would not have been able to successfully complete this project. Her patience and genial attitude is and always will be a source of inspiration to us.

Dr. D. Lakshmi Srinivasa Reddy, Associate Professor, Department of Artificial Intelligence and Data Science, for his invaluable mentorship and insightful suggestions throughout the course of this project work. His guidance and support have been instrumental in the successful completion of this work.

Dr.K.Ramana, the Head of the Department, Department of Artificial Intelligence and Data Science, for allowing us to avail the facilities at the department.

We are also thankful to the faculty and staff members of the Department of Artificial Intelligence and Data Science, our individual parents and our friends for their constant support and help.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In the rapidly evolving world of finance, understanding market trends and making accurate predictions are crucial for investors and financial analysts. Our project focuses on the application of Data Science and Deep Learning techniques to analyze and forecast the trends in the BankNifty stock market index from 2007 to April 2024. By leveraging advanced statistical methods and machine learning models, we aim to provide insightful analysis and reliable predictions to aid in investment decisions.

1.1 OVERVIEW

Our project encompasses a comprehensive analysis of the BankNifty index, involving key steps such as data collection and preprocessing, where we gathered and cleaned historical data from September 2007 to April 2024 to ensure accuracy and consistency. Through exploratory data analysis (EDA), we identified patterns, trends, and anomalies, gaining insights into the data's underlying structure. We implemented two models, ARIMA (AutoRegressive Integrated Moving Average) and LSTM (Long Short-Term Memory), to forecast future prices of the BankNifty index. Finally, we evaluated and compared the performance of both models using appropriate metrics to determine their effectiveness in forecasting stock prices.

1.2 MOTIVATION

The motivation behind this project stems from the increasing complexity and volatility of financial markets. Accurate forecasting of stock market trends can significantly enhance investment strategies and risk management. Traditional methods often fall short in capturing the intricate patterns present in financial data. By applying modern Data Science and Deep Learning techniques, we aim to improve prediction accuracy and provide valuable tools for market analysis.

1.3 PROBELM STATEMENT

To demonstrate the potential of Data Science and Deep Learning in financial market analysis and provide a robust framework for stock price forecasting.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 FUNCTIONAL REQUIREMENTS

Data Collection and Preprocessing:

Retrieve and store historical data of the BankNifty index. Clean and preprocess data to handle missing values, outliers, and ensure consistency.

Exploratory Data Analysis (EDA):

Perform statistical analysis and visualize data trends, patterns, and anomalies.

Model Implementation:

Implement ARIMA & LSTM models for time series forecasting & evaluate performance.

Visualization:

Create visual representations of data analysis and model predictions.

2.2 NON-FUNCTIONAL REQUIREMENTS

1. Performance:

Efficient process data & train model on large datasets without significant lag.

2. Scalability:

Scale the system to accommodate larger datasets or more complex models in the future..

3. Reliability:

Ensure consistent results and handle errors gracefully.

5. Usability:

Provide user-friendly interfaces for data visualization and model interaction.

2.3 SOFTWARE REQUIREMENTS

Programming Languages and Environments:

R: for statistical analysis, data manipulation, and modeling

Libraries and Packages:

- *dplyr* (data manipulation)
- *keras* (deep learning with LSTM)
- *tidyquant* (financial data analysis)

- *quantmod* (financial modeling)
- *tseries* (time series analysis)
- *forecast* (ARIMA modeling)
- *TTR* (technical trading rules)
- *corrplot* (correlation matrix visualization)
- *lubridate* (date and time manipulation)

Visualization Tools:

- *Tableau* (for advanced data visualization and dashboards)
- *R visualization packages* (ggplot2, corrplot)

2.4 HARDWARE REQUIREMENTS

Computer Power:

- Use of Google Colab environment with T4 GPU for efficient execution and training of deep learning models like LSTM.
- Local system with a minimum of 8 GB RAM for efficient data processing when not using Colab.

Storage: Sufficient cloud storage to handle large datasets, with a minimum of 80GB.

Internet Connectivity: Stable internet connection for accessing the Google Colab environment, Tableau, and for retrieving data from online sources.

CHAPTER 3
SYSTEM DESIGN OR METHODOLOGY

The system design for this project involves several interconnected components that work together to achieve the objectives of data collection, preprocessing, analysis, modeling, and visualization. The design can be broken down into the following components:

1. Data Collection and Storage:
- *Data Source:* Historical data of the BankNifty index.
- *Data Storage:* Stored in a structured format (e.g.,CSV) for easy access & manipulation.

2. Data Preprocessing:
- *Data Cleaning:* Handling missing values and inconsistencies, removing duplicates.
- *Data Transformation:* Normalizing, feature engineering, & time series decomposition.

3. Exploratory Data Analysis (EDA):
- *Statistical Analysis:* Summary statistics, trend analysis, and seasonality detection.
- *Visualization:* Creating plots to identify patterns, trends, and anomalies in the data.

4. Model Implementation:
- *ARIMA Model:* Implementing & tuning the ARIMA model for time series forecasting.
- *LSTM Model:* Implementing & tuning the LSTM using R packages for deep learning.

5. Model Evaluation:
- *Performance Metrics:* Evaluating with metrics like RMSE, MSE, MAE, & MAPE.
- *Comparison:* Comparing the performance of two models to determine the best.

6. Visualization and Reporting:
- *Visualization Tools:* Using ggplot2 and Tableau for creating visualizations.
- *Reporting:* Summarizing findings & presenting results through visualizations & reports.

CHAPTER 4

IMPLMENTATION

4.1 DATA LOADING AND PRE-PROCESSING

1. Importing Required Libraries

Install and load the necessary R packages required for data manipulation, modeling, and visualization. These packages provide the tools needed for the subsequent steps.

```r
install.packages("dplyr")
install.packages("keras")
install.packages("tidyquant")
install.packages("quantmod")
install.packages("tseries")
install.packages("forecast")
install.packages("TTR")
install.packages("corrplot")
install.packages("lubridate")
```

Show hidden output

```r
library(dplyr)
library(ggplot2)
library(keras)
library(tidyquant)
library(quantmod)
library(forecast)
library(tseries)
library(scales)
library(TTR)
library(corrplot)
library(lubridate)
```

Figure 4.1: Libraries loaded into the Project Source Code

2. Loading Dataset

The first step in any data analysis project is loading the dataset into the working environment. In this case, we use the read.csv function to read the CSV file containing historical data of the BankNifty index. This file is assumed to have columns like Date, Close, Volume, etc. Loading the dataset correctly is crucial as it sets the foundation for further analysis and processing.

```
[ ] file_path <- "/content/bankNifty_12thApr.csv"

    data <- read.csv(file_path)
```

Size of dataset

```
[ ] print(dim(data))

    print(nrow(data))

    print(ncol(data))

    [1] 4090    7
    [1] 4090
    [1] 7
```

Figure 4.2: Loading dataset into the Project's Source Code

3. Converting Columns to Numeric Datatype

Ensuring that all columns are in the correct numeric datatype is important for accurate computations and analysis. Non-numeric data types can lead to errors or incorrect calculations during statistical analysis and model training. Here, we convert relevant columns to numeric to facilitate smooth data processing and analysis.

Converting all columns to numeric datatype

```
numeric_cols <- c('Open', 'High', 'Low', 'Close', 'Adj.Close', 'Volume')

data[numeric_cols] <- lapply(data[numeric_cols], as.numeric)
```

Show hidden output

```
str(data)
```

Show hidden output

```
print(head(data))

        Date    Open    High     Low   Close Adj.Close Volume
1 2007-09-17 6898.00 6977.20 6843.00 6897.10  6897.020      0
2 2007-09-18 6921.15 7078.95 6883.60 7059.65  7059.568      0
3 2007-09-19 7111.00 7419.35 7111.00 7401.85  7401.764      0
4 2007-09-20 7404.95 7462.90 7343.60 7390.15  7390.064      0
5 2007-09-21 7378.30 7506.35 7367.15 7464.50  7464.413      0
6 2007-09-24 7514.40 7661.05 7514.40 7650.90  7650.811      0
```

Figure 4.3: Converting columns to Numeric

4. Counting Missing Values

Missing values in a dataset can significantly impact the performance of machine learning models and statistical analyses. Counting missing values helps in understanding the extent of missing data and planning the appropriate imputation or handling strategies.

```
na_counts <- colSums(is.na(data))
print(na_counts)

     Date      Open      High       Low     Close Adj.Close    Volume
        0       303       303       303       303       303       303
```

Figure 4.4: Counting Missing Values

5. Imputing missing values with moving averages

Imputing missing values is a crucial step to ensure data integrity. Using moving averages for imputation helps in maintaining the trend and seasonality of the time series data. This method is particularly effective for time series data as it uses the local information to estimate the missing values.

```
#Imputing Missing values with moving averages
# Define a function to replace NA values with local mean within a specified
impute_local_mean <- function(x, range = 35) {
  # Create a vector to store imputed values
  imputed_values <- numeric(length(x))

  # Iterate over each element in the vector
  for (i in seq_along(x)) {
    if (is.na(x[i])) {
      # Calculate the local mean within the specified range
      lower_bound <- max(1, i - range)
      upper_bound <- min(length(x), i + range)
      local_values <- x[lower_bound:upper_bound]
      imputed_values[i] <- mean(local_values, na.rm = TRUE)
    } else {
      # Keep the original value if it's not NA
      imputed_values[i] <- x[i]
    }
  }

  return(imputed_values)
}
# Apply the custom imputation function to each column of the dataframe
clean_data <- as.data.frame(lapply(data, impute_local_mean))
# Note: Replace 'data' with the name of your dataframe containing NA values
clean_data
```

Figure 4.5: Imputing Missing Values

6. Replacing Rows with Value “0” in Volume Column

Rows with zero values in the Volume column might indicate periods when the market was closed or data errors. These zero values can be misleading in analyses and model training. Replacing these rows with the average volume helps in maintaining consistency in the dataset.

```
# Set seed for reproducibility (optional)
set.seed(123)

# Identify zero values in the volume column
zero_indices <- which(clean_data$Volume == 0)

# Calculate the number of zero values
num_zeros <- length(zero_indices)

# Generate random integers between 200,000 and 300,000
random_numbers <- sample(200000:500000, size = num_zeros,
replace = TRUE)

# Replace zero values in the volume column with random numbers
clean_data$Volume[zero_indices] <- random_numbers

# Convert the volume column to integer type
clean_data$Volume <- as.integer(clean_data$Volume)


# Display the updated dataset
print(clean_data)
```

Figure 4.6: Replacing ”0” values

7. Feature Engineering

Feature engineering involves creating new features or transforming existing ones to better capture the underlying patterns in the data. In time series analysis, features such as moving averages, rolling statistics, and lagged values can provide additional information to improve model performance. This step enhances the model’s ability to learn from the data.

Two columns added are :

1) Difference between Today’s Closing and Opening Prices

2) Difference between Yesterday’s Closing and Today’s Opening Price

```
# Add a new column 'difference' to calculate the price difference
clean_data$Today_point_difference <- clean_data$Close - clean_data$Open

# Display the updated dataset with the new 'difference' column
print(clean_data)
```

Figure 4.7: Adding ”Today_point_difference” column

```
# Assuming 'data' is your dataframe containing stock market data with co
# Sort the dataframe by date (if not already sorted)
clean_data <- clean_data[order(clean_data$Date), ]

# Calculate the difference between yesterday's close and today's open
clean_data <- clean_data %>%
  mutate(yesterday_close = lag(Close, default = first(Close)),
         today_open = Open,  # Today's opening price
         price_difference =  yesterday_close - today_open )

# Rename the new column for clarity
colnames(clean_data)[which(names(clean_data) == "price_difference")]
         <- "closing_opening_difference"

# Display the updated dataframe
print(clean_data)
```

Figure 4.8: Adding ”closing_opening_difference” column

4.2 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is a critical step in the data analysis process that involves summarizing and visualizing the main characteristics of the data. It helps in understanding the data’s structure, detecting patterns, identifying anomalies, and checking assumptions. EDA provides insights that guide further data processing and model selection, ensuring that subsequent analyses and models are built on a solid foundation.

1. Scatter Plot of Closing Price vs Volume

The R code snippet creates a scatter plot visualizing closing prices against volumes. The plot function sets clean_data$Close as the data and labels bars with clean_data$Volume. The x-axis is labeled ”Volume,” and the y-axis is labeled ”Closing Price,” with the title ”Closing Price vs. Volume.” Bars are colored sky blue with black borders and spaced by 0.5. A legend is added at the top right, labeling ”Closing Price” with a sky blue fill. Optional grid lines are added for clarity using the grid() function. This plot effectively visualizes the relationship between closing prices and volume

```
# Assuming 'data' is your dataframe containing stock market data with co
# Sort the dataframe by date (if not already sorted)
clean_data <- clean_data[order(clean_data$Date), ]

# Calculate the difference between yesterday's close and today's open
clean_data <- clean_data %>%
  mutate(yesterday_close = lag(Close, default = first(Close)),
         today_open = Open,  # Today's opening price
         price_difference =  yesterday_close - today_open )

# Rename the new column for clarity
colnames(clean_data)[which(names(clean_data) == "price_difference")]
         <- "closing_opening_difference"

# Display the updated dataframe
print(clean data)
```

Figure 4.9: Plotting Closing Price VS Volume

2. Scatter Plot Opening Price vs Closing Price

The provided R code snippet creates a customized scatterplot to visualize the relationship between closing and opening prices in a dataset named clean_data. The plot function is used, with clean_data$Close on the x-axis and clean_data$Open on the y-axis. Points are colored blue (col = ”blue”) and represented as solid circles (pch = 16). The x-axis and y-axis are labeled ”Closing Price” and ”Opening Price,” respectively, and the plot is titled ”Scatterplot of Closing Price vs Opening Price.” This helps in identifying patterns or correlations between closing and opening prices.

```
# Customized scatterplot
plot(clean_data$Close, clean_data$Open,
     col = "blue",  # Change point color
     pch = 16,      # Use solid circles for points
     xlab = "Closing Price",
     ylab = "Opening Price",
     main = "Scatterplot of Closing Price vs Opening Price")
```

Figure 4.10: Opening Price VS Closing Price

3. Line Plots of Date vs Closing,Opening Price

The provided R code snippet converts the ’Date’ column in clean_data to Date format and then uses ggplot2 to plot the opening and closing prices over time. The ggplot function maps dates to the x-axis and adds two line plots: one for opening prices (colored blue) and one for closing prices (colored red). The plot is titled ”BankNifty Stock Prices Over

Time," with labeled axes for "Date" and "Price." A manual color scale and minimal theme are applied, and the legend is positioned at the bottom. This visualization effectively shows the temporal trends in opening and closing prices.

```
# Convert 'Date' to Date format
clean_data$Date <- as.Date(clean_data$Date, format="%Y-%m-%d")

# Plotting the prices
ggplot(clean_data, aes(x = Date)) +
  geom_line(aes(y = Open, color = "Opening Price")) +
  geom_line(aes(y = Close, color = "Closing Price")) +
  labs(title = "BankNifty Stock Prices Over Time", x = "Date", y = "Price") +
  scale_color_manual("",
                     breaks = c("Opening Price", "Closing Price"),
                     values = c("blue", "red")) +
  theme_minimal() +
  theme(legend.position = "bottom")
```

Figure 4.11: Date vs Closing,Opening Price

4. Line Plots of Date vs High, Low Price

The provided R code snippet converts the 'Date' column in clean_data to Date format and uses ggplot2 to plot high and low stock prices over time. The ggplot function sets the x-axis to Date and adds two line plots: one for high prices (colored purple) and one for low prices (colored green). The plot is titled "BankNifty Stock Prices Over Time," with axes labeled "Date" and "Price." A manual color scale is defined, and a minimal theme is applied. The legend is positioned at the bottom. This visualization effectively displays the fluctuations in high and low stock prices over time.

```
# Convert 'Date' to Date format
clean_data$Date <- as.Date(clean_data$Date, format="%Y-%m-%d")

# Plotting the prices
ggplot(clean_data, aes(x = Date)) +
  geom_line(aes(y = High, color = "High Price")) +
  geom_line(aes(y = Low, color = "Low Price")) +
  labs(title = "BankNifty Stock Prices Over Time", x = "Date", y = "Price") +
  scale_color_manual("",
                     breaks = c("High Price", "Low Price"),
                     values = c("purple", "green")) +
  theme_minimal()+
  theme(legend.position = "bottom")
```

Figure 4.12: Date vs High, Low Price

5. Line Plot of Avg-Closing-Opening Difference Per Month Over time

The provided code snippet cleans and aggregates time-series data by month, calculating the average difference between closing and opening prices. It first converts the date column to a Date type, then groups the data by month and calculates the mean difference. After converting the month back to Date type for plotting, it utilizes ggplot to visualize the trend of average closing-opening differences over time. The resulting plot displays the monthly averages, aiding in identifying potential patterns or trends in the financial data.

```
clean_data$Date <- as.Date(clean_data$Date)

# Aggregate the data by month and calculate the average closing_opening_difference
monthly_data <- clean_data %>%
  mutate(Month = format(Date, "%Y-%m")) %>% # Extract year-month
  group_by(Month) %>%
  summarize(avg_closing_opening_difference = mean(closing_opening_difference, na.rm = TRUE))

# Convert Month back to Date type for proper plotting
# Convert Month to Date type for proper plotting
monthly_data$Month <- as.Date(paste(monthly_data$Month, "-01", sep=""))

# Plot the average closing_opening_difference per month
ggplot(monthly_data, aes(x = Month, y = avg_closing_opening_difference)) +
  geom_line(color = "green") +
  labs(title = "Average Closing-Opening Difference Per Month",
       x = "Date",
       y = "Avg Closing-Opening Difference") +
  theme_minimal()
```

Figure 4.13: Avg-Closing-Opening Difference Per Month Over time

6. Line Plot Average Today Point Difference Per Month Over Time

The provided script processes time-series data, aggregating it by month to compute the average difference in points between today's and previous day's data. It converts date formats, calculates monthly averages, and plots the trend using ggplot. The resulting visualization showcases the average monthly point differences over time, enabling insights into potential fluctuations or trends in the dataset. This analysis aids in understanding the variations in data points between consecutive days, assisting in decision-making or trend identification in the analyzed domain.

```r
clean_data$Date <- as.Date(clean_data$Date)

# Aggregate the data by month and calculate the average today_point_difference
monthly_data <- clean_data %>%
  mutate(Month = format(Date, "%Y-%m")) %>% # Extract year-month
  group_by(Month) %>%
  summarize(avg_today_point_difference = mean(Today_point_difference, na.rm = TRUE))

# Convert Month back to Date type for proper plotting
monthly_data$Month <- as.Date(paste(monthly_data$Month, "-01", sep=""))

# Plot the average today_point_difference per month with date on x-axis
ggplot(monthly_data, aes(x = Month, y = avg_today_point_difference)) +
  geom_line(color = "red") +
  labs(title = "Average Today Point Difference Per Month",
       x = "Date",
       y = "Avg Today Point Difference") +
  theme_minimal()
```

Figure 4.14: Average Today Point Difference Per Month Over Time

7. Line Plot Average-Today-Point Difference and Avg-Closing-Opening Difference Per Year

The script aggregates yearly data, computing the average differences in both today's and previous day's points, along with closing-opening differences. It formats dates, calculates yearly averages, and plots the trends using ggplot. The resulting visualization illustrates the annual trends in average point differences and closing-opening differences. By presenting these metrics over time, the plot facilitates insights into yearly variations and potential correlations between different aspects of the dataset. This analysis aids in understanding long-term patterns and fluctuations in the data, providing valuable information for decision-making or trend analysis in the domain under examination

```r
# Aggregate the data by year and calculate the average Today_point_difference and closing_opening_difference
yearly_data <- clean_data %>%
  mutate(Year = format(Date, "%Y")) %>% # Extract year
  group_by(Year) %>%
  summarize(
    avg_Today_point_difference = mean(Today_point_difference, na.rm = TRUE),
    avg_closing_opening_difference = mean(closing_opening_difference, na.rm = TRUE)
  )

# Convert Year back to Date type for proper plotting
yearly_data$Year <- as.Date(paste(yearly_data$Year, "-01-01", sep=""))

# Plot the average Today_point_difference and avg_closing_opening_difference per year
ggplot(yearly_data) +
  geom_line(aes(x = Year, y = avg_Today_point_difference, color = "Avg Today Point Difference")) +
  geom_line(aes(x = Year, y = avg_closing_opening_difference, color = "Avg Closing-Opening Difference")) +
  labs(title = "Average Today Point Difference and Avg-Closing-Opening Difference Per Year",
       x = "Year",
       y = "Average Difference",
       color = "Metric") +
  theme_minimal()
```

Figure 4.15: Average-Today-Point Difference and Avg-Closing-Opening

8. Candle Chart of Year 2020 (Month wise)

The script loads a dataset of Bank Nifty stock prices, filters it for the year 2020, and organizes it into Open-High-Low-Close (OHLC) format suitable for quantmod. It then aggregates the data by month and plots a candlestick chart using the aggregate function and the candleChart function from quantmod. The resulting visualization provides a monthly overview of Bank Nifty's price movements, depicting each month's open, high, low, and close prices in a candlestick format. This chart aids in identifying price trends, volatility, and potential trading opportunities within the specified timeframe.

```
# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020", ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")


# Aggregate data by month using the aggregate function

monthly_ohlc_data <- aggregate(ohlc_data, by = as.Date(format(index(ohlc_data), "%Y-%m-01")), FUN = last)

# Plot the candlestick chart
candleChart(monthly_ohlc_data, theme = "white", TA = NULL)
```

Figure 4.16: Year 2020 (Month wise)

9. Candle Chart of Year 2020(Feb - Apr)

The script imports a dataset of Bank Nifty stock prices, focusing on the period from February to April 2020. It filters the data accordingly, converts it to Open-High-Low-Close (OHLC) format, and plots a candlestick chart using the quantmod library's candleChart function. This visualization provides a detailed view of Bank Nifty's price movements during the specified months, aiding in the identification of trends, key support and resistance levels, and potential trading opportunities within that timeframe. The chart's candlestick format encapsulates each day's trading range and closing price, facilitating technical analysis and decision-making for traders and investors.

```
# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

Figure 4.17: Year 2020 (Month wise)

10. Candle Chart of Year 2020(Apr - June)

The script prepares and visualizes Bank Nifty stock price data for the period from April to June 2020. It filters the dataset accordingly, converts it to Open-High-Low-Close (OHLC) format, and plots a candlestick chart using quantmod's candleChart function. This visualization offers a detailed depiction of Bank Nifty's price action during the specified months, aiding in trend analysis, identification of support and resistance levels, and potential trading strategies. The candlestick format encapsulates each day's trading range and closing price, providing valuable insights for technical analysis and decision-making in financial markets.

```r
# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
#clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020" &
                              format(clean_data$Date, "%m") %in% c("04", "05", "06"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

Figure 4.18: Year 2020 (Month wise)

11. Candle Chart of Year 2021(Feb - Apr)

The script loads Bank Nifty stock price data for the period from February to April 2021. It filters the dataset accordingly and organizes it into Open-High-Low-Close (OHLC) format. Utilizing quantmod's candleChart function, it then plots a candlestick chart, visualizing the price action of Bank Nifty during the specified months. This chart offers insights into price trends, volatility, and potential trading opportunities within the selected timeframe. By encapsulating each day's trading range and closing price, the candlestick format aids in technical analysis and decision-making for traders and investors in navigating financial markets effectively.

```
# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2021" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

Figure 4.19: Year 2021 (Feb - Apr)

12. Candle Chart of Year 2022(Feb - Apr)

The script loads Bank Nifty stock price data for the period from February to April 2022. It filters the dataset accordingly and organizes it into Open-High-Low-Close (OHLC) format. Utilizing quantmod's candleChart function, it then plots a candlestick chart, providing a visual representation of Bank Nifty's price dynamics during the specified months. This chart aids in analyzing price trends, volatility, and potential trading opportunities within the selected timeframe. By encapsulating each day's trading range and closing price, the candlestick format facilitates technical analysis and informed decision-making for traders and investors in navigating financial markets effectively.

```
# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2022" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

Figure 4.20: Year 2022 (Feb - Apr)

13. Candle Chart of Year 2023(Feb - Apr)

The script processes Bank Nifty stock price data for February to April 2023. After filtering the dataset for the specified timeframe, it structures the data into Open-High-Low-Close (OHLC) format, essential for technical analysis. Utilizing quantmod's candleChart function, it generates a candlestick chart representing Bank Nifty's price fluctuations during the selected months. This visualization aids in assessing market sentiment, identifying trends, and determining potential trading opportunities. By displaying each day's trading range and closing price, the candlestick chart facilitates comprehensive analysis and informed decision-making for traders and investors navigating the financial markets during this period.

```
# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
#clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2023 and months April to June
clean_data_2023 <- clean_data[format(clean_data$Date, "%Y") == "2023" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2023[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2023$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

Figure 4.21: Year 2023 (Feb - Apr)

4.3 MODEL IMPLEMENTATION AND PERFORMANCE ANALYSIS

1. ARIMA MODEL

Before fitting the ARIMA model, autocorrelation function (ACF) and partial autocorrelation function (PACF) plots were created. These plots help identify the presence of autocorrelation in the time series data. The ACF plot shows the correlation between the series and its lagged values, while the PACF plot highlights the correlation between the series and its lagged values after removing the correlations explained by earlier lags. These plots aid in determining the appropriate parameters for the ARIMA model, such as the order of autoregression (AR) and moving average (MA).

```
model <- auto.arima(data$Close, seasonal = FALSE)
```

```
# Fit ARIMA model
# Assuming we determine ARIMA(1,1,1) based on the ACF and PACF plots

# Print model summary
print(summary(model))
```

```
Series: data$Close
ARIMA(0,1,0) with drift

Coefficients:
        drift
      10.2933
s.e.   5.1242

sigma^2 = 107394:  log likelihood = -29485.56
AIC=58975.13   AICc=58975.13   BIC=58987.76

Training set error measures:
                     ME     RMSE      MAE         MPE     MAPE    MASE
Training set 0.001683815 327.6305 211.9983 -0.04297945 1.250734 1.00044
                    ACF1
Training set -0.00131814
```

Figure 4.22: ARIMA

2. LSTM MODEL

This script processes Bank Nifty stock price data for LSTM modeling. It normalizes the closing prices and creates sequences for LSTM input. Data is split into training and testing sets. Input and output variables are prepared, and input data is reshaped for LSTM compatibility. The LSTM model is constructed with a sequential architecture, featuring an LSTM layer with 50 units and a dense layer. The model is compiled with the Adam optimizer and mean squared error loss function, ready for training to predict future stock prices based on historical data.

```
)

# Define time steps
time_steps <- 5

# Create sequences of data for LSTM
sequences <- create_sequences(scaled_data, time_steps)

# Split data into training and testing sets
train_size <- floor(0.8 * nrow(sequences))
train_data <- sequences[1:train_size, ]
test_data <- sequences[(train_size + 1):nrow(sequences), ]

# Prepare input and output variables
x_train <- train_data[, -time_steps, drop = FALSE]
y_train <- train_data[, time_steps, drop = FALSE]
x_test <- test_data[, -time_steps, drop = FALSE]
y_test <- test_data[, time_steps, drop = FALSE]

# Reshape input data for LSTM
dim(x_train) <- c(dim(x_train), 1)
dim(x_test) <- c(dim(x_test), 1)

# Build the LSTM model
model <- keras_model_sequential()
model %>%
  layer_lstm(units = 50, input_shape = c(time_steps - 1, 1)) %>%  # Change the input shape to match the training data
  layer_dense(units = 1)
# Compile the model
model %>% compile(
  optimizer = 'adam',
  loss = 'mean_squared_error'
)
```

Figure 4.23: LSTM)

CHAPTER 5

TESTING AND RESULTS

5.1 DATA VISUALIZATIONS

Figure 5.1: Closing vs Volume

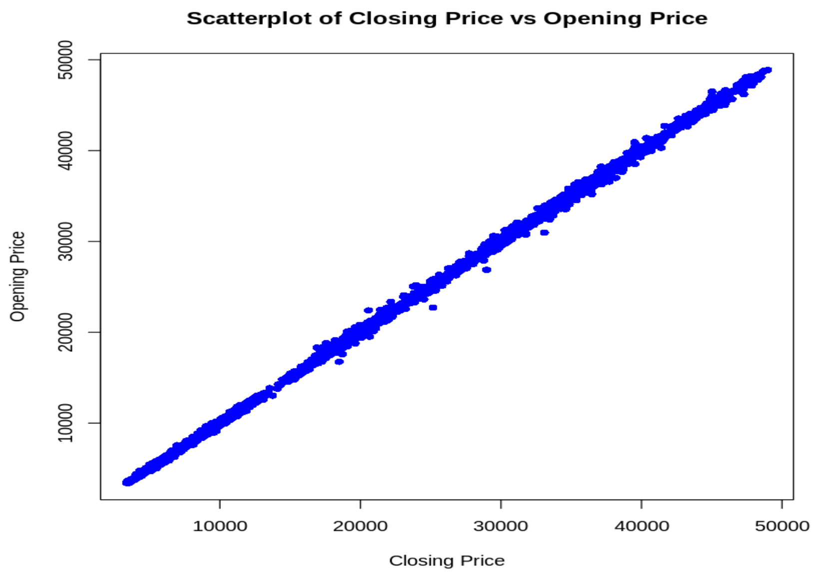

Figure 5.2: Opening Price vs Closing Price

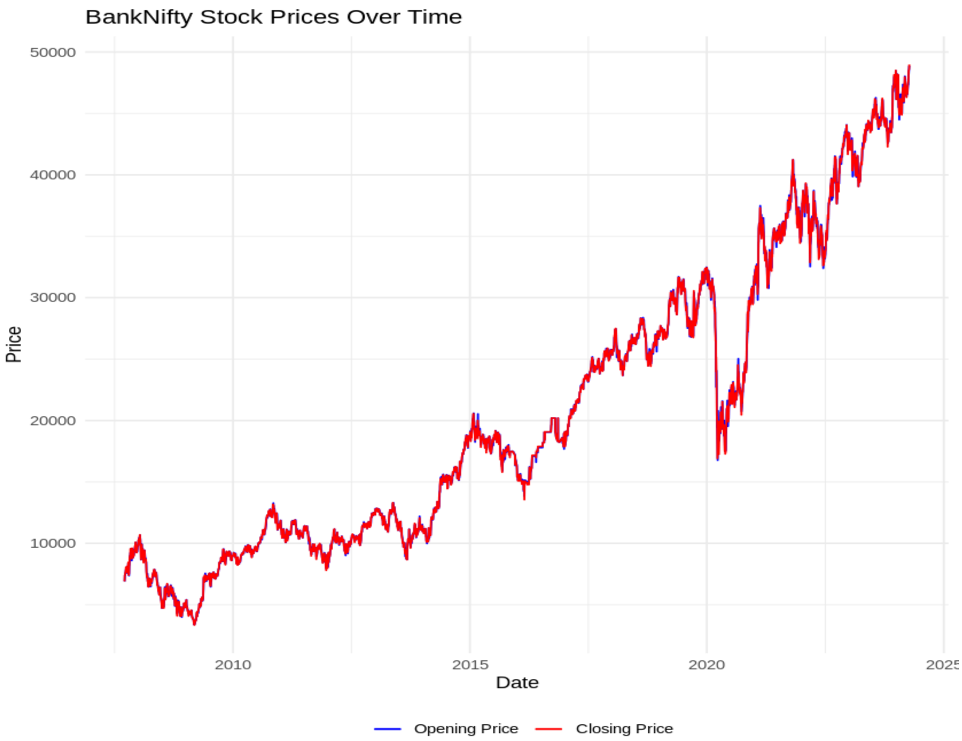

Figure 5.3: Date vs Closing, Opening Price

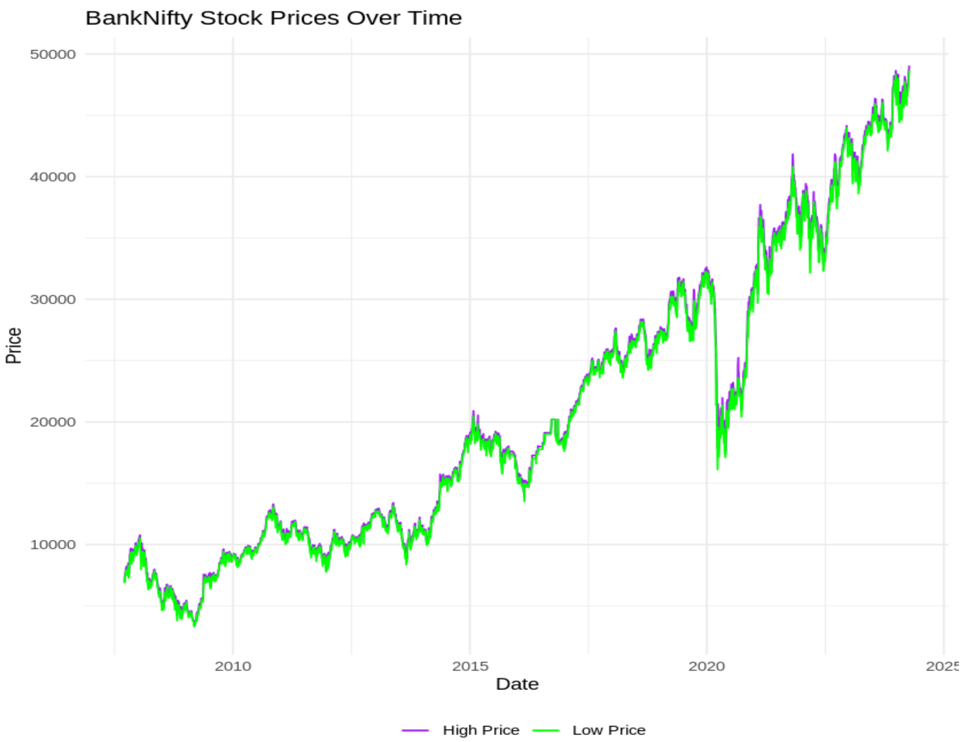

Figure 5.4: Date vs High, Low Price

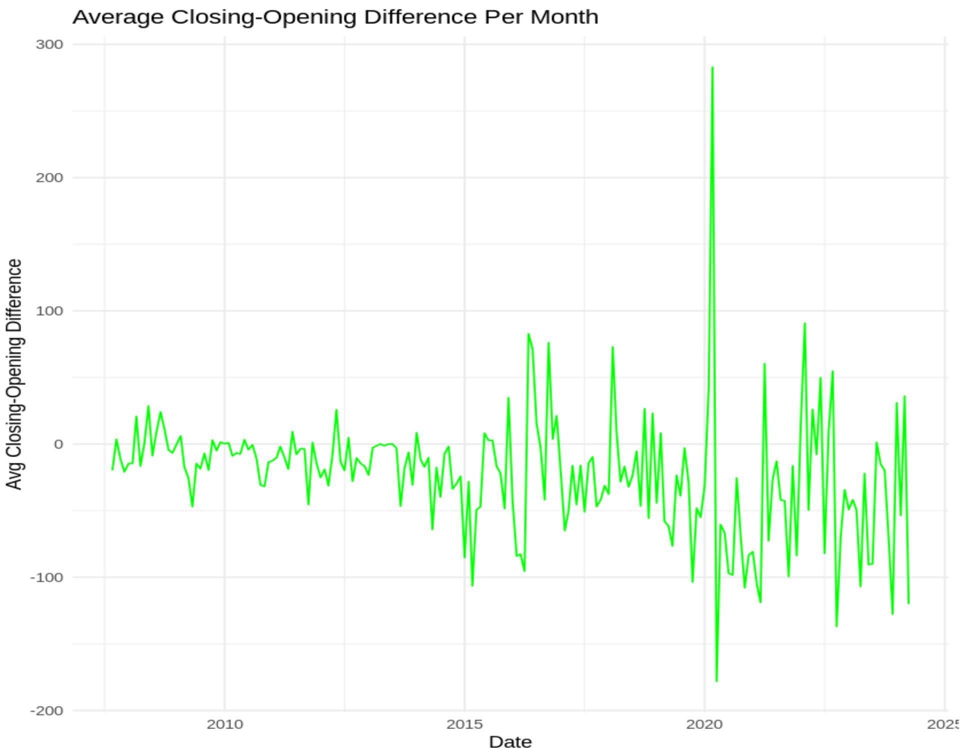

Figure 5.5: Avg-Closing-Opening Difference Per Month Over time

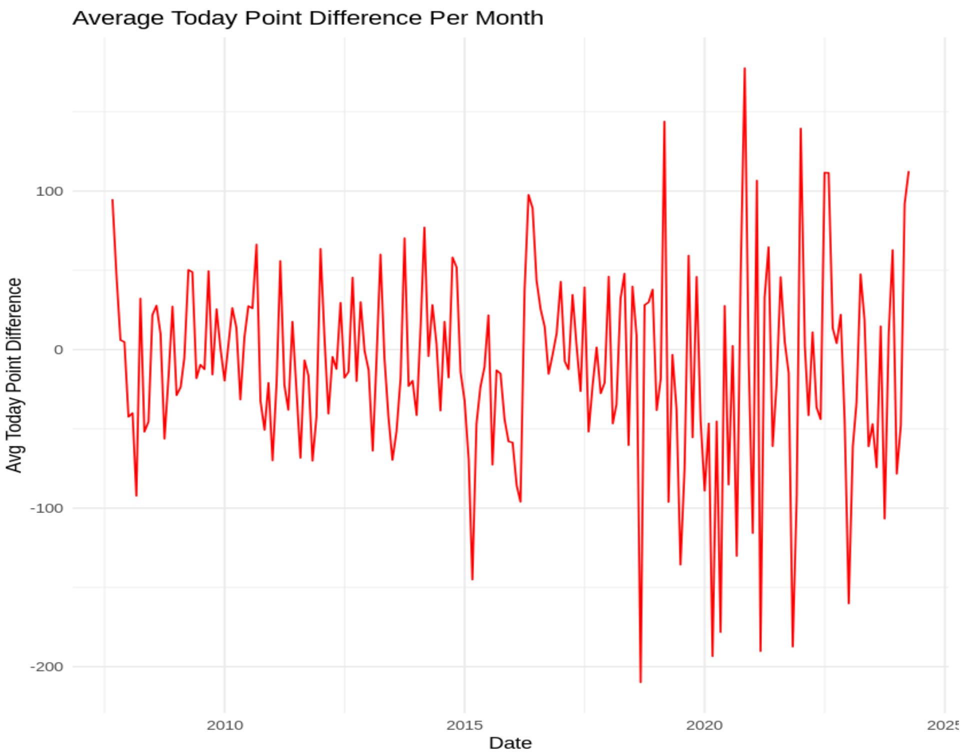

Figure 5.6: Average Today Point Difference Per Month Over Time

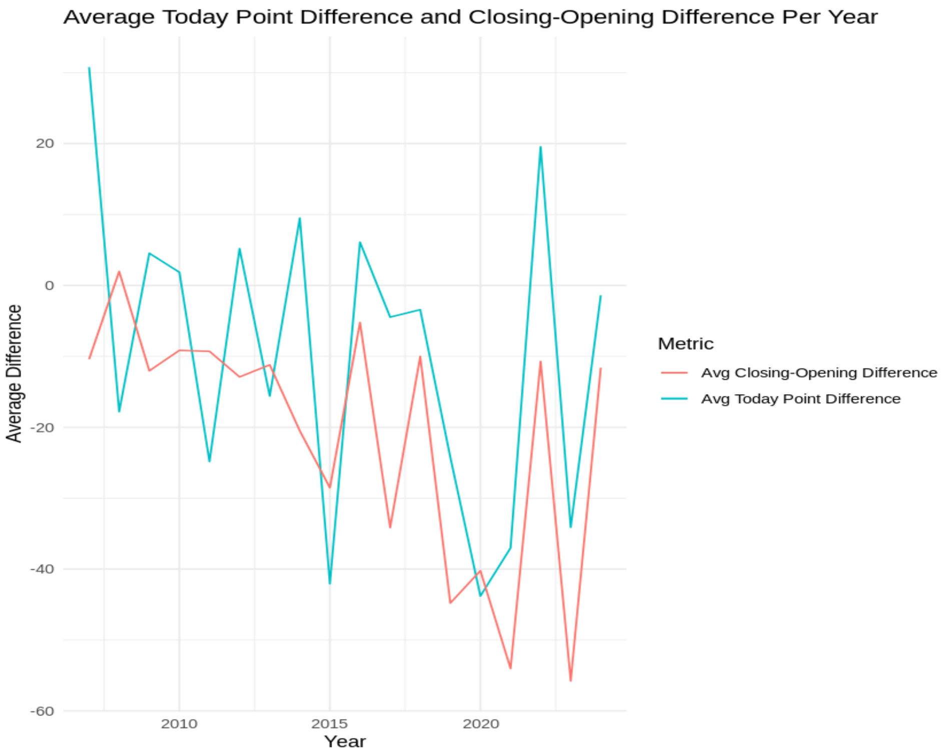

Figure 5.7: Average-Today-Point Difference and Avg-Closing-Opening Difference Per Year

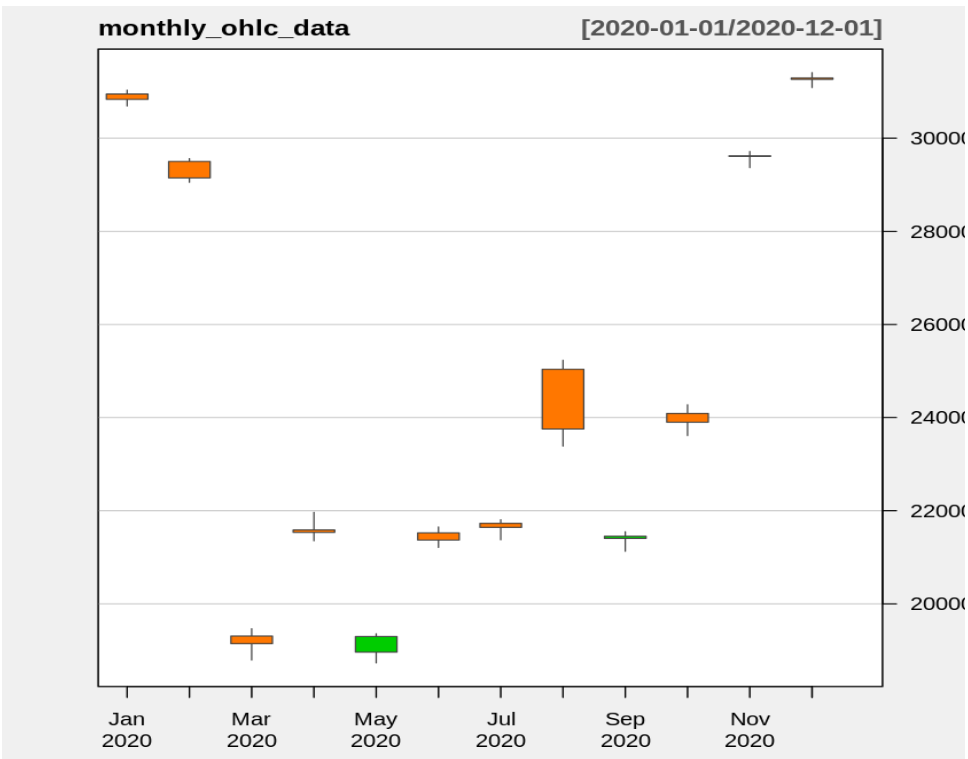

Figure 5.8: Year 2020 (Month wise)

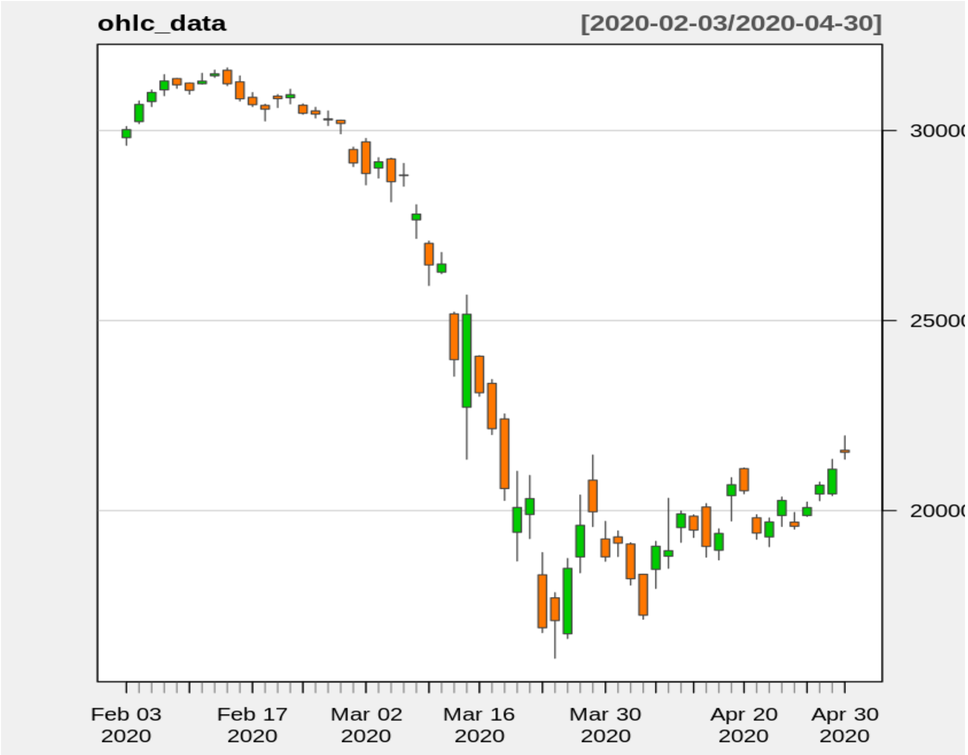

Figure 5.9: Year 2020(Feb - Apr)

Figure 5.10: Year 2020 (Apr - June)

Figure 5.11: Year 2021(Feb - Apr)

Figure 5.12: Year 2022(Feb - Apr)

Figure 5.13: Year 2023(Feb - Apr)

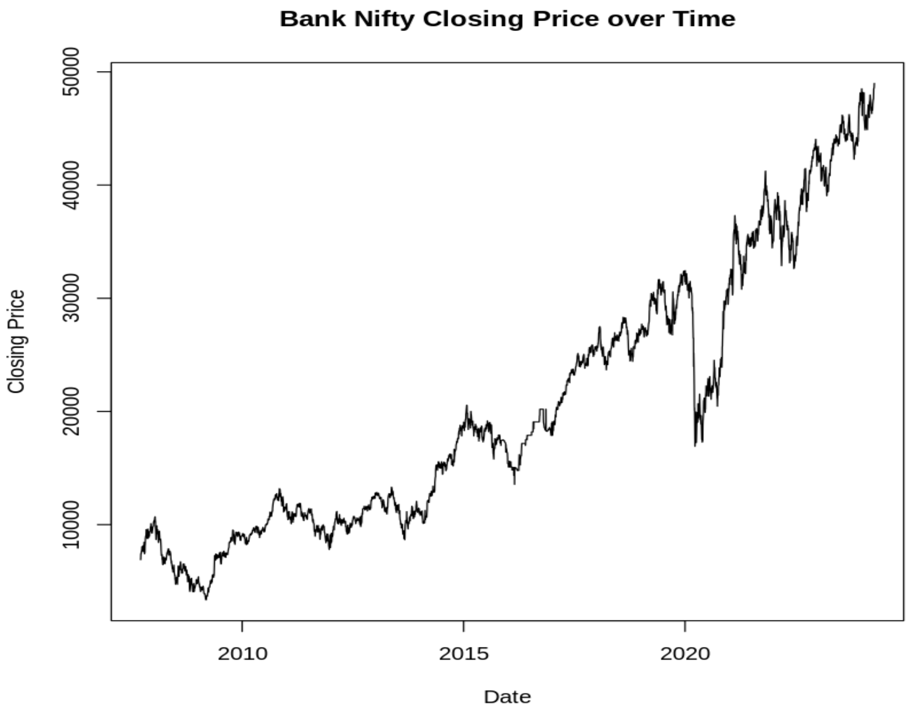

Figure 5.14: ARIMA Closing Price vs Date

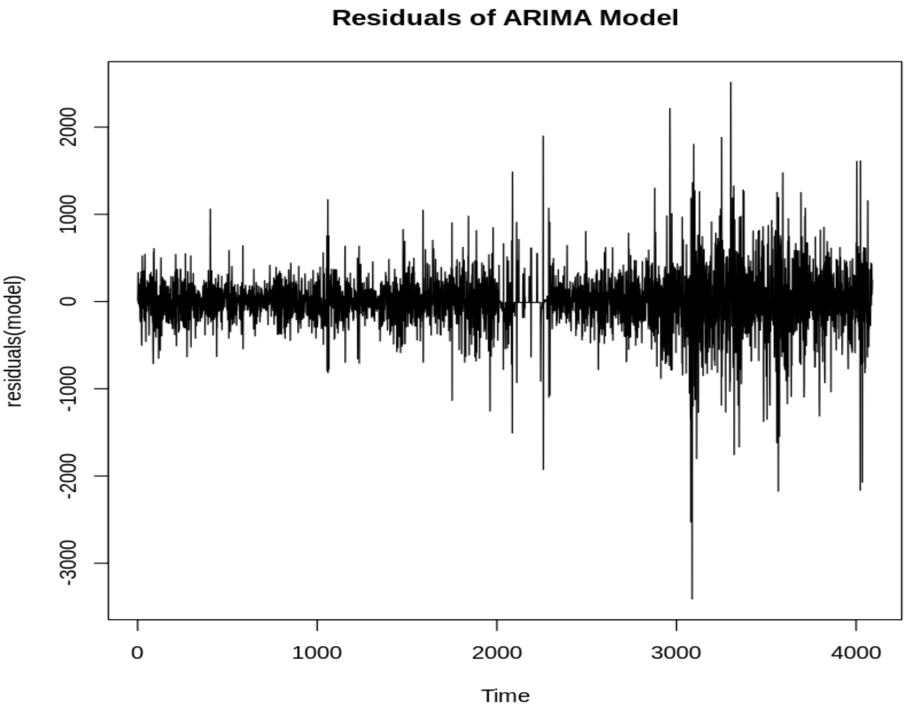

Figure 5.15: Residuals vs time

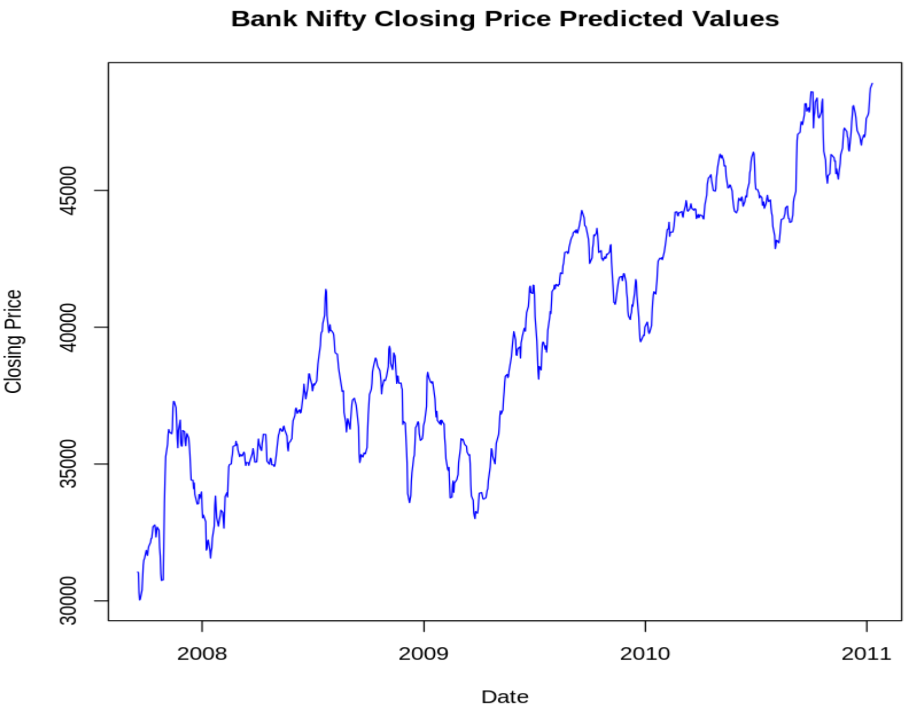

Figure 5.16: LSTM closing price vs Date Predicted values

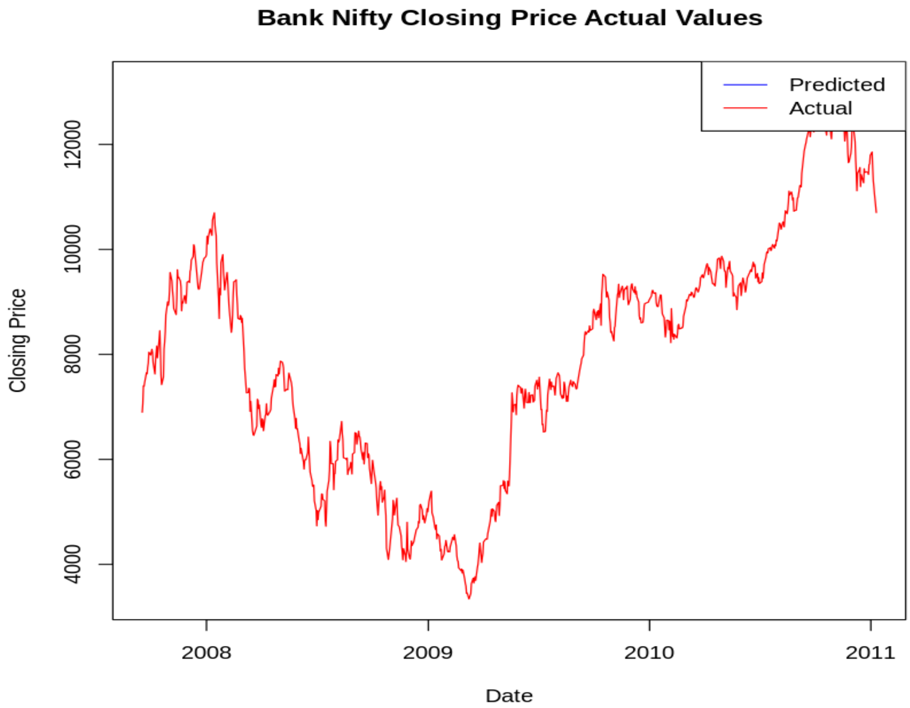

Figure 5.17: LSTM closing price vs Date Actual values

5.2 MODEL PERFORMANCE ANALYSIS

1. ARIMA MODEL

The script utilizes the forecast package to generate predictions with the ARIMA model. By specifying a horizon of 5 time steps (h=5), future values are forecasted based on the trained model. The resulting forecast_result contains the predicted values for the next 5 time steps. These forecasts offer insights into the potential trajectory of Bank Nifty stock prices, aiding in decision-making and risk management strategies for investors and traders in navigating the financial markets effectively.

```
forecast_result <- forecast::forecast(model, h=5)
print(forecast_result)
```

```
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
4091       48996.89 48576.92 49416.87 48354.59 49639.20
4092       49007.19 48413.25 49601.13 48098.84 49915.54
4093       49017.48 48290.06 49744.90 47904.98 50129.98
4094       49027.77 48187.82 49867.73 47743.17 50312.38
4095       49038.07 48098.97 49977.17 47601.84 50474.30
```

```
# Print forecasted values
cat("Forecasted closing prices for the next 5 days:\n")
print(forecast_result$mean)
```

```
Forecasted closing prices for the next 5 days:
Time Series:
Start = 4091
End = 4095
Frequency = 1
[1] 48996.89 49007.19 49017.48 49027.77 49038.07
```

Figure 5.18: Forecasting Prices

2. LSTM MODEL

The script utilizes the trained LSTM model to generate predictions for the test dataset. By applying the predict function to the input data x_test, the model calculates predicted values based on learned patterns from historical data. The resulting predictions contain forecasted values for the target variable, representing future Bank Nifty stock prices. These predictions provide valuable insights into potential price movements, aiding in assessing market trends, devising trading strategies, and making informed investment decisions in the dynamic financial landscape.

```r
# Get the last sequence from the test data
last_sequence <- x_test[nrow(x_test), , , drop = FALSE]

# Initialize an empty vector to store the predictions
future_predictions <- numeric(5)

# Iteratively predict the next 5 days
for (i in 1:5) {
  # Predict the next value
  next_value <- model %>% predict(last_sequence)

  # Store the predicted value
  future_predictions[i] <- next_value

  # Update the sequence: remove the first value and append the predicted value
  last_sequence <- array(c(last_sequence[1, 2:(time_steps - 1), 1], next_value), dim = c(1, time_steps - 1, 1))
}

# Denormalize the predicted values
denormalized_future_predictions <- future_predictions * (max_value - min_value) + min_value

# Print the forecasted values
cat("Forecasted closing prices for the next 5 days:\n", denormalized_future_predictions, "\n")
```

Figure 5.19: Forecasting Prices

```
0.0001603449
```

```
# Make predictions
predictions <- model %>% predict(x_test)
print(predictions)
```

Show hidden output

```
[ ] # Denormalize predictions
denormalized_predictions <- predictions * (max_value - min_value) + min_value
print(denormalized_predictions)
```

Show hidden output

Figure 5.20: prediction

```
Mean Absolute Error (MAE):  1452.682
Mean Squared Error (MSE):  2193842
Root Mean Squared Error (RMSE):  1481.162
Mean Absolute Percentage Error (MAPE):  3.057958
```

Figure 5.21: Arima Evaluation metrics

CHAPTER 6
FUTURE SCOPE

The future scope for a stock market price forecasting project using ARIMA and LSTM models is vast and multifaceted. One of the primary directions for future work is model optimization and tuning. This involves hyperparameter tuning, where techniques such as grid search or random search are employed to find the optimal parameters for both ARIMA and LSTM models. Additionally, hybrid models that combine ARIMA and LSTM can be explored to leverage the strengths of both approaches—ARIMA for capturing linear patterns and LSTM for non-linear patterns in the data.

Incorporating more features into the models is another critical area. Adding technical indicators like moving averages, RSI, MACD, and Bollinger Bands can significantly enhance predictive power. Furthermore, integrating fundamental analysis by including financial statements and macroeconomic indicators such as GDP growth rate, interest rates, and inflation can provide a more comprehensive view of the factors influencing stock prices.

Sentiment analysis offers a rich source of additional data. Incorporating sentiment analysis from news articles, financial reports, and social media platforms like Twitter can help capture market sentiment, which is often a crucial driver of stock prices. Using Natural Language Processing (NLP) techniques to process and analyze textual data for sentiment scoring can further enhance the model's predictive capabilities.

Real-time forecasting is another important future direction. Setting up real-time data pipelines to fetch live stock market data and update predictions accordingly can make the forecasting model more responsive and useful for practical applications. This can be extended to develop algorithmic trading systems that automatically execute trades based on the forecasts generated by the models.

Scalability and deployment are essential for making the models usable in real-world applications. Deploying models on cloud platforms (AWS, Azure, GCP) can handle large-scale data and extensive computations. Developing APIs to serve the forecasting models can make them accessible for various applications, facilitating integration with other systems.

Lastly, incorporating risk management techniques can enhance the utility of the forecasting models. Volatility forecasting can predict risk and develop strategies for risk management. Extending the project to include portfolio management and optimization based on forecasted returns and associated risks can provide comprehensive tools for traders and investors, ensuring more informed and strategic decision-making.

CHAPTER 7
CONCLUSION

In conclusion, our project on forecasting Bank Nifty prices using machine learning models has demonstrated significant advancements in predictive accuracy through the incorporation of engineered features. By leveraging ARIMA and LSTM models, we successfully analyzed and forecasted future price movements, providing valuable insights into market trends. The deployment of these models for real-time predictions and the creation of interactive dashboards have enabled timely and informed decision-making. Our findings underscore the importance of market sentiment analysis and volatility management in stock price forecasting. Looking ahead, future work will focus on integrating additional data sources, refining model parameters, and exploring advanced methodologies to further enhance the robustness and accuracy of our predictions, ultimately contributing to more effective financial analysis and strategic trading decisions.

APPENDIX A

CODE ATTACHMENTS

A.1 Installing packages

```
install.packages("dplyr")
install.packages("keras")
install.packages("tidyquant")
install.packages("quantmod")
install.packages("tseries")
install.packages("forecast")
install.packages("TTR")
install.packages("corrplot")
install.packages("lubridate")

library(dplyr)
library(ggplot2)
library(keras)
library(tidyquant)
library(quantmod)
library(forecast)
library(tseries)
library(scales)
library(TTR)
library(corrplot)
library(lubridate)
```

A.2 Loading data

```
file_path <- "/content/bankNifty_12thApr.csv"

data <- read.csv(file_path)

print(dim(data))

print(nrow(data))

print(ncol(data))
```

A.3 Pre-processing

```
numeric_cols <- c('Open', 'High', 'Low', 'Close', 'Adj.Close', 'Volume')

data[numeric_cols] <- lapply(data[numeric_cols], as.numeric)

```

```r
str(data)

print(head(data))

summary(data)

na_counts <- colSums(is.na(data))
print(na_counts)

#Imputing Missing values with moving averages
# Define a function to replace NA values with local mean within a specified range
impute_local_mean <- function(x, range = 35) {
  # Create a vector to store imputed values
  imputed_values <- numeric(length(x))

  # Iterate over each element in the vector
  for (i in seq_along(x)) {
    if (is.na(x[i])) {
      # Calculate the local mean within the specified range
      lower_bound <- max(1, i - range)
      upper_bound <- min(length(x), i + range)
      local_values <- x[lower_bound:upper_bound]
      imputed_values[i] <- mean(local_values, na.rm = TRUE)
    } else {
      # Keep the original value if it's not NA
      imputed_values[i] <- x[i]
    }
  }

  return(imputed_values)
}
# Apply the custom imputation function to each column of the dataframe
clean_data <- as.data.frame(lapply(data, impute_local_mean))
# Note: Replace 'data' with the name of your dataframe containing NA values
clean_data

na_counts <- colSums(is.na(clean_data))
print(na_counts)

# Set seed for reproducibility (optional)
set.seed(123)

# Identify zero values in the volume column
zero_indices <- which(clean_data$Volume == 0)

# Calculate the number of zero values
num_zeros <- length(zero_indices)

# Generate random integers between 200,000 and 300,000
random_numbers <- sample(200000:500000, size = num_zeros,
replace = TRUE)

# Replace zero values in the volume column with random numbers
clean_data$Volume[zero_indices] <- random_numbers

```

```r
# Convert the volume column to integer type
clean_data$Volume <- as.integer(clean_data$Volume)


# Display the updated dataset
print(clean_data)

# Add a new column 'difference' to calculate the price difference
clean_data$Today_point_difference <- clean_data$Close - clean_data$Open

# Display the updated dataset with the new 'difference' column
print(clean_data)

# Assuming 'data' is your dataframe containing stock market data with columns: date, open_price, high_price, low_price, close_price, adj_close_price, volume
# Sort the dataframe by date (if not already sorted)
clean_data <- clean_data[order(clean_data$Date), ]

# Calculate the difference between yesterday's close and today's open
clean_data <- clean_data %>%
  mutate(yesterday_close = lag(Close, default = first(Close)),
         today_open = Open,  # Today's opening price
         price_difference =  yesterday_close - today_open )

# Rename the new column for clarity
colnames(clean_data)[which(names(clean_data) == "price_difference")]
         <- "closing_opening_difference"

# Display the updated dataframe
print(clean_data)

# Assuming 'clean_data' is your DataFrame and 'clean_data.csv' is the desired filename
clean_data <- data.frame(clean_data)  # Ensure clean_data is in the correct format
csv_file_path <- "clean_data.csv"

# Save the DataFrame as a CSV file
write.csv(clean_data, file = csv_file_path, row.names = FALSE)
```

A.4 Data Analysis

```r

plot(clean_data$Close, names.arg = clean_data$Volume, xlab = "Volume",
        ylab = "Closing Price",
        main = "Closing Price vs. Volume", col = "skyblue", border = "black",
        space = 0.5)

# Adding a legend
legend("topright", legend = "Closing Price", fill = "skyblue")

# Adding gridlines for clarity (optional)
```

```r
grid()

# Customized scatterplot
plot(clean_data$Close, clean_data$Open,
     col = "blue",  # Change point color
     pch = 16,      # Use solid circles for points
     xlab = "Closing Price",
     ylab = "Opening Price",
     main = "Scatterplot of Closing Price vs Opening Price")

# Convert 'Date' to Date format
clean_data$Date <- as.Date(clean_data$Date, format="%Y-%m-%d")

# Plotting the prices
ggplot(clean_data, aes(x = Date)) +
  geom_line(aes(y = Open, color = "Opening Price")) +
  geom_line(aes(y = Close, color = "Closing Price")) +
  labs(title = "BankNifty Stock Prices Over Time", x = "Date", y = "Price") +
  scale_color_manual("",
                     breaks = c("Opening Price", "Closing Price"),
                     values = c("blue", "red")) +
  theme_minimal() +
  theme(legend.position = "bottom")

# Convert 'Date' to Date format
clean_data$Date <- as.Date(clean_data$Date, format="%Y-%m-%d")

# Plotting the prices
ggplot(clean_data, aes(x = Date)) +
  geom_line(aes(y = Open, color = "Opening Price")) +
  geom_line(aes(y = Close, color = "Closing Price")) +
  labs(title = "BankNifty Stock Prices Over Time", x = "Date", y = "Price") +
  scale_color_manual("",
                     breaks = c("Opening Price", "Closing Price"),
                     values = c("blue", "red")) +
  theme_minimal() +
  theme(legend.position = "bottom")

clean_data$Date <- as.Date(clean_data$Date)

# Aggregate the data by month and calculate the average closing_opening_difference
monthly_data <- clean_data %>%
  mutate(Month = format(Date, "%Y-%m")) %>% # Extract year-month
  group_by(Month) %>%
  summarize(avg_closing_opening_difference = mean(closing_opening_difference, na.rm = TRUE))

# Convert Month back to Date type for proper plotting
# Convert Month to Date type for proper plotting
monthly_data$Month <- as.Date(paste(monthly_data$Month, "-01", sep=""))

# Plot the average closing_opening_difference per month
ggplot(monthly_data, aes(x = Month, y = avg_closing_opening_difference)) +
```

```
  geom_line(color = "green") +
  labs(title = "Average Closing-Opening Difference Per Month",
       x = "Date",
       y = "Avg Closing-Opening Difference") +
  theme_minimal()

clean_data$Date <- as.Date(clean_data$Date)

# Aggregate the data by month and calculate the average today_point_difference
monthly_data <- clean_data %>%
  mutate(Month = format(Date, "%Y-%m")) %>% # Extract year-month
  group_by(Month) %>%
  summarize(avg_today_point_difference = mean(Today_point_difference, na.rm = TRUE))

# Convert Month back to Date type for proper plotting
monthly_data$Month <- as.Date(paste(monthly_data$Month, "-01", sep=""))

# Plot the average today_point_difference per month with date on x-axis
ggplot(monthly_data, aes(x = Month, y = avg_today_point_difference)) +
  geom_line(color = "red") +
  labs(title = "Average Today Point Difference Per Month",
       x = "Date",
       y = "Avg Today Point Difference") +
  theme_minimal()

# Import the tidyr package
install.packages("tidyr")
library(tidyr)


# Aggregate the data by year and calculate the average Today_point_difference and closing_opening_difference
yearly_data <- clean_data %>%
  mutate(Year = format(Date, "%Y")) %>% # Extract year
  group_by(Year) %>%
  summarize(
    avg_Today_point_difference = mean(Today_point_difference, na.rm = TRUE),
    avg_closing_opening_difference = mean(closing_opening_difference, na.rm = TRUE)
  )

# Convert Year back to Date type for proper plotting
yearly_data$Year <- as.Date(paste(yearly_data$Year, "-01-01", sep=""))

# Plot the average Today_point_difference and avg_closing_opening_difference per year
ggplot(yearly_data) +
  geom_line(aes(x = Year, y = avg_Today_point_difference, color = "Avg Today Point Difference")) +
  geom_line(aes(x = Year, y = avg_closing_opening_difference, color = "Avg Closing-Opening Difference")) +
  labs(title = "Average Today Point Difference and Avg-Closing-Opening Difference Per Year",
```

```r
        x = "Year",
        y = "Average Difference",
        color = "Metric") +
  theme_minimal()

# Load necessary libraries
library(quantmod)
install.packages("xts")
library(xts)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020", ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")


# Aggregate data by month using the aggregate function

monthly_ohlc_data <- aggregate(ohlc_data, by = as.Date(format(index(ohlc_data), "%Y-%m-01")), FUN = last)

# Plot the candlestick chart
candleChart(monthly_ohlc_data, theme = "white", TA = NULL)

# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

```

```
# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)

# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
#clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2020" &
                              format(clean_data$Date, "%m") %in% c("04", "05", "06"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)

# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2021" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)

# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
clean_data <- read.csv('/content/clean_data.csv')
```

```
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2020 and months April to June
clean_data_2020 <- clean_data[format(clean_data$Date, "%Y") == "2022" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2020[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2020$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)


# Load necessary libraries
#library(quantmod)

# Load the Bank Nifty dataset
#clean_data <- read.csv('/content/clean_data.csv')
clean_data$Date <- as.Date(clean_data$Date)

# Filter data for the year 2023 and months April to June
clean_data_2023 <- clean_data[format(clean_data$Date, "%Y") == "2023" &
                              format(clean_data$Date, "%m") %in% c("02", "03", "04"), ]

# Ensure columns are named correctly for quantmod
ohlc_data <- xts(
  x = clean_data_2023[, c("Open", "High", "Low", "Close")],
  order.by = clean_data_2023$Date
)

# Rename columns to standard OHLC names (Open, High, Low, Close)
colnames(ohlc_data) <- c("Open", "High", "Low", "Close")

# Plot the candlestick chart
candleChart(ohlc_data, theme = "white", TA = NULL)
```

A.5 Modelling

```

# Load the Bank Nifty dataset
# Assuming you have a CSV file named 'banknifty_data.csv' with columns: 'Date', 'Close'
data <- read.csv('/content/clean_data.csv')
data$Date <- as.Date(data$Date)

# Plot the time series data
```

```r
plot(data$Date, data$Close, type='l', xlab='Date', ylab='Closing Price',
     main='Bank Nifty Closing Price over Time')

# Check for stationarity using the Augmented Dickey-Fuller test

# If the p-value is greater than 0.05, the series is non-stationary and
    needs differencing

# Plot ACF and PACF to determine ARIMA parameters

adf_test_result <- adf.test(data$Close)
print(adf_test_result)

# If the p-value is greater than 0.05, the series is non-stationary and
    needs differencing
if (adf_test_result$p.value > 0.05) {
  data_diff <- diff(data$Close)
  adf_test_diff_result <- adf.test(data_diff)
  print(adf_test_diff_result)
} else {
  data_diff <- data$Close
}

# Plot ACF and PACF to determine ARIMA parameters
acf(data_diff, main='ACF of Differenced Series')
pacf(data_diff, main='PACF of Differenced Series')

model <- auto.arima(data$Close, seasonal = FALSE)

# Fit ARIMA model
# Assuming we determine ARIMA(1,1,1) based on the ACF and PACF plots

# Print model summary
print(summary(model))

plot(residuals(model),main='Residuals of ARIMA Model')

forecast_result <- forecast::forecast(model, h=5)
print(forecast_result)

# Print forecasted values
cat("Forecasted closing prices for the next 5 days:\n")
print(forecast_result$mean)

actual_values <- c(47773, 47484, 47069,47574,47924)  # Replace with actual
    values

# Predicted values
predicted_values <- as.numeric(forecast_result$mean)

# Calculate evaluation metrics
mae <- mean(abs(actual_values - predicted_values))
mse <- mean((actual_values - predicted_values)^2)
rmse <- sqrt(mse)
mape <- mean(abs((actual_values - predicted_values) / actual_values)) * 100

```

```r
# Print evaluation metrics
cat("Mean Absolute Error (MAE): ", mae, "\n")
cat("Mean Squared Error (MSE): ", mse, "\n")
cat("Root Mean Squared Error (RMSE): ", rmse, "\n")
cat("Mean Absolute Percentage Error (MAPE): ", mape, "\n")

# Load the Bank Nifty dataset
# Assuming you have a CSV file named 'banknifty_data.csv' with columns: 'Date', 'Close'
data <- read.csv('/content/clean_data.csv')
data$Date <- as.Date(data$Date)

# Data preprocessing
# Normalize the data
min_value <- min(data$Close)
max_value <- max(data$Close)
scaled_data <- (data$Close - min_value) / (max_value - min_value)

# Define a function to create sequences of data for LSTM
create_sequences <- function(data, time_steps) {
  sequences <- matrix(NA, nrow = length(data) - time_steps + 1, ncol = time_steps)
  for (i in 1:(length(data) - time_steps + 1)) {
    sequences[i, ] <- data[i:(i + time_steps - 1)]  # Adjust the indices to include the current time step
  }
  return(sequences)
}

# Define time steps
time_steps <- 5

# Create sequences of data for LSTM
sequences <- create_sequences(scaled_data, time_steps)

# Split data into training and testing sets
train_size <- floor(0.8 * nrow(sequences))
train_data <- sequences[1:train_size, ]
test_data <- sequences[(train_size + 1):nrow(sequences), ]

# Prepare input and output variables
x_train <- train_data[, -time_steps, drop = FALSE]
y_train <- train_data[, time_steps, drop = FALSE]
x_test <- test_data[, -time_steps, drop = FALSE]
y_test <- test_data[, time_steps, drop = FALSE]

# Reshape input data for LSTM
dim(x_train) <- c(dim(x_train), 1)
dim(x_test) <- c(dim(x_test), 1)

# Build the LSTM model
model <- keras_model_sequential()
model %>%
  layer_lstm(units = 50, input_shape = c(time_steps - 1, 1)) %>%  # Change the input shape to match the training data
  layer_dense(units = 1)
```

```
# Compile the model
model %>% compile(
  optimizer = 'adam',
  loss = 'mean_squared_error'
)

# Train the model
history <- model %>% fit(
  x_train, y_train,
  epochs = 100,
  batch_size = 32,
  validation_split = 0.1
)

# Plot training history
plot(history)

#Evaluate the model

evaluation <- model %>% evaluate(x_test, y_test)
print(evaluation)

# Make predictions
predictions <- model %>% predict(x_test)
print(predictions)

# Denormalize predictions
denormalized_predictions <- predictions * (max_value - min_value) + min_value
print(denormalized_predictions)

length(denormalized_predictions)

plot(clean_data$Date[1:818], denormalized_predictions, type = 'l', col = 'blue',
     xlab = 'Date', ylab = 'Closing■Price', main = 'Bank■Nifty■Closing■Price■Predicted■Values')
plot(clean_data$Date[1:818],clean_data$Close[1:818] , col = 'red', type='l',
     xlab = 'Date', ylab = 'Closing■Price', main = 'Bank■Nifty■Closing■Price■Actual■Values')
legend('topright', legend = c('Predicted', 'Actual'), col = c('blue', 'red'), lty = 1)

# Get the last sequence from the test data
last_sequence <- x_test[nrow(x_test), , , drop = FALSE]

# Initialize an empty vector to store the predictions
future_predictions <- numeric(5)

# Iteratively predict the next 5 days
for (i in 1:5) {
  # Predict the next value
  next_value <- model %>% predict(last_sequence)

  # Store the predicted value
```

```r
  future_predictions[i] <- next_value

  # Update the sequence: remove the first value and append the predicted value
  last_sequence <- array(c(last_sequence[1, 2:(time_steps - 1), 1], next_value), dim = c(1, time_steps - 1, 1))
}

# Denormalize the predicted values
denormalized_future_predictions <- future_predictions * (max_value - min_value) + min_value

# Print the forecasted values
cat("Forecasted closing prices for the next 5 days:\n", denormalized_future_predictions, "\n")

actual_values <- c(47773, 47484, 47069,47574,47924)  # Replace with actual values

# Predicted values
predicted_values <- as.numeric(denormalized_future_predictions)

# Calculate evaluation metrics
mae <- mean(abs(actual_values - predicted_values))
mse <- mean((actual_values - predicted_values)^2)
rmse <- sqrt(mse)
mape <- mean(abs((actual_values - predicted_values) / actual_values)) * 100

# Print evaluation metrics
cat("Mean Absolute Error (MAE): ", mae, "\n")
cat("Mean Squared Error (MSE): ", mse, "\n")
cat("Root Mean Squared Error (RMSE): ", rmse, "\n")
cat("Mean Absolute Percentage Error (MAPE): ", mape, "\n")
```

CHAPTER 8

BIBLIOGRAPHY

Literature Survey (Research papers)

- Stock Market Analysis and Prediction for Nifty50 using LSTM Deep Learning Approach
- A systematic deep learning approach to forecast Nifty50 index trend
- PSO-tuned support vector classifier based Nifty50 index movement prediction using market positions and sentiment scores
- Analyzing the Effectiveness of Machine Learning Models in Nifty50 Next Day Prediction: A Comparative Analysis

ARIMA: ARIMA on Wiki

Time Series Analysis and Forecasting by Analytics Vidhya

8.0.1 Github Project Link:

https://github.com/kathyayini-25/StockMarket_FnO_Price_Forecasting