**Stock Price Prediction**

**A**

**Mini Project Report**

***Submitted to***



# Jawaharlal Nehru Technological University, Hyderabad

*In partial fulfilment of the requirements for the*

*award of the degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

By

## **BOMMARAJUHARITHA (22VE1A05K6)**

## **DUMMANI BHANUTEJA (22VE1A05L5)**

## **POOSA BHOOVIKA RANI (22VE1A05P8)**

## **SHEGANTI MEGHANA (22VE1A05R2)**

**Under the Guidance**

**of**

**Dr. Ch. Janakamma**

**ASSOCIATE PROFESSOR**



## SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

(Affiliated to JNTUH, Approved by A.I.C.T.E and Accredited by NAAC, New Delhi)

Beside Indu Aranya, Nagole, Hyderabad-500068, Ranga Reddy Dist.

**(2022-2026)**



# **SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# **CERTIFICATE**

This is to certify that the MiniProject Report on **“Stock Price Prediction”** submitted by **Bommaraju Haritha, Dummani Bhanu Teja, Poosa Bhoovika Rani, Sheganti Meghana** bearing Hall ticket numbers: **22VE1A05K6, 22VE1A05L5, 22VE1A05P8, 22VE1A05R2** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING** from Jawaharlal Nehru Technological University, Kukatpally, Hyderabad for the academic year 2024-2025 is a record of bonafide work carried out by them under our guidance and Supervision.

.

|  |  |
| --- | --- |
| **Project Coordinator** | **Head of the Department** |
| **Dr. M. Swapna** | **Dr. U. M. FERNANDES DIMLO** |
| **Associate Professor** | **Professor & Head** |
| **Internal Guide**  **Dr. Ch. Janakamma** | **External Examiner** |

## **Associate Professor**



# **SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

We **Bommaraju Haritha , Dummani Bhanu Teja, Poosa Bhoovika Rani , Sheganti Meghana** bearing Hall ticket number: **22VE1A05K6, 22VE1A05L5, 22VE1A05P8, 22VE1A05R2** hereby declare that the MiniProject titled **STOCK PRICE PREDICTION** done by us under the guidance of **Dr. Ch. Janakamma, Associate Professor** which is submitted in the partial ful fillment of the requirementfor the award of the B.Tech degree in **Computer Science and Engineering at Sreyas Institute of Engineering and Technology** for Jawaharlal Nehru Technological University, Hyderabad is our original work.

## **BOMMARAJUHARITHA (22VE1A05K6)**

## **DUMMANI BHANUTEJA (22VE1A05L5)**

## **POOSA BHOOVIKA RANI (22VE1A05P8)**

## **SHEGANTI MEGHANA (22VE1A05R2)**

**ACKNOWLEDGEMENT**

The successful completion of any task would be incomplete without mention of the people who made it possible through their guidance and encouragement crowns all the efforts with success.

We take this opportunity to acknowledge with thanks and deep sense of gratitude to **Dr. Ch. Janakamma, Associate Professor, Department of Computer Science and Engineering** for her constant encouragement and valuable guidance during the Project work.

A Special vote of Thanks to **Dr. U. M. Fernandes Dimlo, Head of the Department** and **Dr. M. Swapna, Associate Professor, Project Coordinator** who has been a source of Continuous motivation and support. They had taken time and effort to guide and correct us all through the span of this work.

We owe very much to the **Department Faculty, Principal** and the **Management** who made our term at Sreyas Institute of Engineering and Technology a stepping stone for our career. We treasure every moment we had spent in college.

Last but not the least, our heartiest gratitude to our parents and friends for their continuous encouragement and blessings. Without their support this work would not have been possible.

## **BOMMARAJU HARITHA (22VE1A05K6)**

## **DUMMANI BHANUTEJA (22VE1A05L5)**

## **POOSA BHOOVIKA RANI (22VE1A05P8)**

## **SHEGANTI MEGHANA (22VE1A05R2)**

## 

### ABSTRACT

The Stock Price Prediction project aims to create a user-friendly, efficient, and intelligent platform for predicting future stock prices and offering investment recommendations. The system is designed to streamline the decision-making process for users while providing robust machine learning tools to analyze historical stock data and forecast price trends. Key features include real-time data fetching using Yahoo Finance, moving average-based feature engineering, support for multiple prediction models (Random Forest, Linear Regression, and LSTM), model performance comparison, and recommendation generation (Buy, Hold, or Sell).

The system architecture leverages Python libraries and frameworks to ensure reliability, scalability, and responsiveness. Throughout the development process, emphasis is placed on intuitive interface design using Streamlit, structured data handling with pandas, and rigorous testing for consistent output. By combining ease of use with powerful analytics, the system aims to deliver a seamless and practical experience for investors and stock market enthusiasts.

**KEYWORDS:** *Stock Market, Price Forecasting, Random Forest, Linear Regression, LSTM, Streamlit, Machine Learning, Time Series, RMSE, R² Score*.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** |  | **TABLE OF CONTENTS** | **PAGE NO.** |
| **1** |  | **INTRODUCTION** | **1-8** |
| 1.1 | GENERAL | 1 |
| 1.2 | PROBLEM STATEMENT | 2 |
| 1.3 | EXISTING SYSTEM | 3 |
| 1.3.1 | DRAWBACKS | 4 |
| 1.4 | PROPOSED SYSTEM | 7 |
| 1.4.1 | ADVANTAGES | 7 |
| **2** |  | **LITERATURE SURVEY** | **9-11** |
| 2.1 | TECHNICAL PAPERS | 9 |
| **3** |  | **REQUIREMENTS** | **12-13** |
| 3.1 | GENERAL | 12 |
| 3.2 | HARDWARE REQUIREMENTS | 12 |
| 3.3 | SOFTWARE REQUIREMENTS | 13 |
| **4** |  | **SYSTEM DESIGN** | **14-24** |
| 4.1 | GENERAL | 14 |
| 4.2 | SYSTEM ARCHITECTURE | 15 |
| 4.3 | UML DISGN | 16 |
| 4.3.1 | USE-CASE DIAGRAM | 17 |
| 4.3.2 | CLASS DIAGRAM | 19 |
| 4.3.3 | ACTIVITY DIAGRAM | 22 |

|  |  |  |  |
| --- | --- | --- | --- |
| **5** |  | **TECHNOLOGY DESCRIPTION** | **25-28** |
| 5.1 | WHAT IS PYTHON? | 25 |
| 5.2 | ADVANTAGES OF PYTHON | 25 |
| 5.3 | WHAT IS STREAMLIT | 26 |
| 5.4 | ADVANTAGES OF STREAMLIT | 27 |
| 5.5 | PYTHON LIBRARIES USED | 28 |
| 5.6 | DISADVANTAGES OF PYTHON | 28 |
| **6** |  | **IMPLEMENTATION** | **29-37** |
| 6.1 | METHODOLOGY | 29 |
| 6.2 | SAMPLE CODE | 33 |
| **7** |  | **TESTING** | **38-42** |
| 7.1 | GENERAL | 38 |
| 7.2 | TYPES OF TESTING | 38 |
| 7.3 | TEST CASES | 42 |
| **8** |  | **RESULTS** | **43-44** |
| 8.1 | RESULTS SCREENSHOTS | 43 |
| **9** |  | **FUTURE SCOPE** | **45** |
| **10** |  | **CONCLUSION** | **46** |
| **11** |  | **REFERENCES** | **47** |

|  |  |  |
| --- | --- | --- |
| **FIG. NO/TAB.NO** | **LIST OF FIGURES AND TABLES** | **PAGE NO.** |
| 4.1 | Architecture Diagram | 17 |
| 4.2 | Use Case Diagram | 20 |
| 4.3 | Class Diagram | 23 |
| 4.4 | Activity Diagram | 26 |
| 7.1 | Types of Testing | 41 |
| 7.2 | Test cases table | 42 |

|  |  |  |
| --- | --- | --- |
| **SCREENSHOT. NO** | **LIST OF SCREENSHOTS** | **PAGE. NO** |
| 8.1 | Output-1 | 43 |
| 8.2 | Output-2 | 43 |
| 8.3 | Output-3 | 44 |

**LIST OF SYMBOLS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNO.** | **Name of Symbol** | **Notation** | **Description** |
| 1 | CLASS |  | Represents a collection of similar entities grouped together. |
| 2 | ASSOCIATION |  | Associations represent static relationships between classes. Roles represent the way the two classes see each other. |
| 3 | ACTOR |  | It aggregates several classes into a single class. |
| 4 | RELATION (uses) | *Uses* | Used for additional process communication. |
| 5 | RELATION (extents) |  | Extends relationship is used when one use case is similar to another use case. |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | COMMUNICATION |  | Communication between various use cases. |
| 7 | STATE |  | State of the process |
| 8 | INITIAL STATE |  | Initial state of the object |
| 9 | FINAL STATE |  | Final state of the object |
| 10 | CONTROL FLOW |  | Represents various control flow between the states. |
| 11 | DECISION BOX |  | Represents decision  making process from a    constraint |
| 12 | USE CASE |  | Interact ion between the system and external environment. |

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | COMPONENT |  | Represents physical modules which is a collection of components. |
| 14 | NODE |  | Represents physical modules which are a collection of components. |
| 15 | DATA PROCESS/ STATE |  | A circle in DFD represents a state or process which has been triggered due to some event or action. |
| 16 | EXTERNAL ENTITY |  | Represents external entities such as keyboard,  sensors, etc |
| 17 | TRANSITION |  | Represents communication that occurs between processes. |

|  |  |  |  |
| --- | --- | --- | --- |
| 18 | OBJECT LIFELINE |  | Represents the vertical dimensions that the object communications. |
| 19 | MESSAGE |  | Represents the message exchanged. |

**CHAPTER 1**

### INTRODUCTION

#### GENERAL

The **Stock Price Prediction** system is a comprehensive machine learning-based application meticulously designed to forecast future stock prices and provide users with actionable investment guidance. At its core, the system offers an intuitive and interactive interface that enables users to enter any valid stock ticker symbol and retrieve historical market data. The application processes this data using key financial indicators, including moving averages (MA7, MA20, and MA50), to detect trends and patterns in stock price behavior. One of the most notable features of the system is its ability to apply and compare multiple prediction models—**Random Forest**, **Linear Regression**, and **Long Short-Term Memory (LSTM)** networks—thereby offering insights into model accuracy and market predictability.

Users benefit from a real-time data interface powered by the Yahoo Finance API, allowing continuous access to up-to-date market information. The system complements its forecasting capabilities with intelligent recommendation logic that suggests whether a user should **Buy**, **Hold**, or **Sell** a particular stock based on the predicted change in price. Each model’s performance is clearly visualized using metrics such as **Root Mean Square Error (RMSE)** and **R² Score**, helping users understand how reliable the predictions are. The application also includes interactive graphs that compare actual stock prices against predicted values for enhanced decision-making.

Administrators or developers can monitor model training processes, view evaluation outputs, and modify the architecture for further experimentation or deployment. From a technical standpoint, the Stock Price Prediction system is built using Python and employs a combination of essential data science libraries such as **pandas**, **numpy**, **matplotlib**, **scikit-learn**, **TensorFlow**, and **Streamlit**. The user interface is browser-based and lightweight, allowing the system to be easily deployed locally or on cloud platforms such as **AWS**, **Heroku**, or **Google Cloud**.

To ensure high accuracy, performance, and user accessibility, the application emphasizes modular design, maintainable code, and secure data handling. With its scalable architecture and model flexibility, the Stock Price Prediction system is well-positioned to serve not only casual traders and students but also financial analysts and researchers looking to experiment with stock forecasting models. In essence, the project blends data science and software engineering to create a reliable, transparent, and scalable solution for navigating the complex world of financial markets.

#### 1.2 PROBLEM STATEMENT

In today’s highly dynamic and volatile stock market environment, investors and traders face significant challenges in making timely and informed decisions. Traditional methods of stock analysis, often reliant on manual research and intuition, are increasingly inadequate to cope with the vast amounts of data and rapid market fluctuations. Investors frequently encounter difficulties such as inconsistent price trends, lack of accurate forecasting tools, and the overwhelming complexity of analyzing historical and real-time data. These limitations contribute to missed opportunities, suboptimal investment decisions, and increased financial risk.

Furthermore, existing stock prediction systems often fail to integrate multiple analytical models or provide personalized recommendations based on individual risk profiles and market behavior. This fragmented approach hampers users from gaining comprehensive insights or optimizing their investment strategies effectively. In the current era of big data and machine learning, there is an urgent need for an intelligent and integrated solution that can accurately predict stock prices and recommend optimal actions to users.

The Stock Price Prediction and Recommendation System project aims to address these challenges by developing a sophisticated, Streamlit-based application that utilizes advanced machine learning algorithms such as Random Forest, Linear Regression, and LSTM to deliver reliable stock price forecasts. The system will offer an intuitive user interface that enables investors to visualize predicted price trends, analyze historical data, and receive tailored investment recommendations. Additionally, the platform will incorporate real-time data updates to ensure timely decision-making and maximize investment returns.

By leveraging modern technologies and predictive modeling techniques, this project seeks to enhance the accuracy of stock price forecasting, improve user confidence, and support data-driven investment decisions. Ultimately, the Stock Price Prediction and Recommendation System will provide a scalable, robust, and user-centric tool designed to empower investors in navigating the complexities of the stock market efficiently and effectively.

#### EXISTING SYSTEM

Several established systems currently dominate the stock market prediction and investment recommendation space, each offering various functionalities to assist investors and traders in making informed decisions. Here are some of the most prominent ones:

**1. Yahoo Finance**

Yahoo Finance is widely recognized for providing comprehensive market data, stock quotes, news, and basic analytical tools. It offers users real-time updates, historical data, and customizable watchlists. While Yahoo Finance serves as an accessible platform for casual investors to track market trends, its predictive capabilities are limited, and it primarily focuses on data presentation rather than advanced forecasting or personalized recommendations.

**2. TradingView**

TradingView is popular among traders for its powerful charting tools, social networking features, and scripting language that enables users to create custom indicators and strategies. It supports technical analysis and provides some predictive insights through community-shared algorithms. However, TradingView’s forecasts depend largely on user-generated content, and it lacks integrated machine learning models for automated price predictions or personalized investment guidance.

**3. MetaStock**

MetaStock offers a comprehensive suite of technical analysis tools and market forecasting software, utilizing a range of indicators and expert advisors to assist traders in decision-making. It supports backtesting and automated trading strategies but generally requires substantial user expertise to interpret data effectively. While MetaStock includes forecasting features, its focus is more on historical pattern recognition than on leveraging advanced AI techniques like deep learning.

**4. Kensho (S&P Global)**

Kensho, a part of S&P Global, utilizes AI and machine learning for predictive analytics in the financial sector. Its solutions include event-driven analytics and scenario-based forecasting to help institutional investors anticipate market movements. Despite its advanced capabilities, Kensho’s tools are primarily designed for large enterprises and are not easily accessible to retail investors seeking straightforward, real-time price prediction and recommendations.

**5.** **Zacks Investment Research**

Zacks offers stock research, analysis, and recommendation reports based on proprietary ranking systems and financial modeling. It provides daily updates and ratings to guide investment choices but does not incorporate real-time machine learning models for dynamic price prediction. Zacks is valued for its fundamental analysis but lacks integrated predictive analytics that combine multiple machine learning approaches.

* + 1. **DISADVANTAGES OF EXISTING SYSTEM**

**1. Complexity and Learning Curve**

- Many existing platforms have complex user interfaces and require significant technical knowledge to operate, which can be discouraging for novice investors and reduce overall accessibility.

**2. Integration Challenges**

- These systems often lack seamless integration with other financial tools or data sources, making it difficult for users to aggregate and analyze information holistically.

**3. Costly Subscriptions**

- Several stock analysis tools and forecasting platforms come with high subscription fees, which may be prohibitive for small investors or students seeking budget-friendly solutions.

**4. Limited Customization**

- Most existing platforms provide generic predictions and insights, offering little to no customization based on individual investment goals, risk appetite, or market behavior.

**5. Dependence on Technical Indicators**

-Many systems rely heavily on technical indicators without incorporating machine learning or contextual analysis, which can limit predictive accuracy and adaptability.

**6. Security Concerns**

-Many systems rely heavily on technical indicators without incorporating machine learning or contextual analysis, which can limit predictive accuracy and adaptability.

**7. Performance Limitations**

- Some platforms experience slow performance or fail to update in real-time during periods of high market activity, which can lead to missed opportunities and poor investment decisions.

**8. Inadequate Forecasting Models**

- Existing systems may use outdated or simplistic forecasting techniques that don’t account for market volatility, macroeconomic events, or advanced predictive analytics.

**9. Limited Innovation**

- A lack of AI integration and modern data science practices in some tools results in stagnation, making them ill-equipped to evolve with emerging technologies or user demands.

**10. User Experience Issues**

- Many platforms still use cluttered interfaces and complex navigation structures, making it hard for users to interpret data and access meaningful insights quickly.

**11. Scalability Constraints**

- Some tools are not scalable to accommodate large datasets, multiple user profiles, or expanding use cases such as portfolio optimization or sentiment analysis.

**12. Vendor Lock-In**

- Proprietary systems may restrict user flexibility by locking them into specific platforms, hindering migration to better or more advanced tools in the future.

**13. Data Silos**

- Information is often segmented across different parts of the platform or third-party applications, making it difficult to obtain a unified view for comprehensive decision-making.

**14. Maintenance Interruptions**

- Scheduled updates or system outages may disrupt access to critical market data during trading hours, potentially leading to financial loss.

**15. Limited Educational Support**

- Many platforms do not offer adequate tutorials, learning materials, or onboarding support, which negatively impacts new users attempting to understand market dynamics.

**16. Lack of Mobile Optimization**

- Several systems do not offer fully responsive mobile platforms, limiting access to vital forecasting tools and recommendations on-the-go.

**17. Complex Licensing Models**

- Licensing agreements and feature restrictions in freemium models can complicate access to essential features, forcing users to purchase expensive upgrades.

**18. Environmental Impact**

- Heavily resource-consuming platforms that rely on large-scale infrastructure without optimization may have a higher environmental footprint, conflicting with green computing principles.

**19. Delayed Updates and Upgrades**

- Slow adoption of new features or industry standards can leave platforms outdated and unable to meet evolving user expectations or market requirements.

**20. Competitive Disadvantage**

- Without intelligent prediction models and personalized recommendations, existing systems may lag behind innovative competitors offering AI-powered and user-focused solutions.

#### PROPOSED SYSTEM

The objective of the proposed system is to overcome the major limitations of existing stock prediction and recommendation platforms by enabling efficient analysis and management of stock data and user preferences, thereby enhancing the accuracy and relevance of investment decisions. By integrating advanced machine learning models and streamlined data processing, the system ensures better prediction performance and user satisfaction.

With improved computerization involved in handling historical stock data, real-time prices, and user-specific inputs, the proposed system minimizes errors and inconsistencies that are common in manual or semi-automated analysis methods. The implementation of well-defined prediction algorithms—such as Random Forest, Linear Regression, and LSTM—ensures that the system delivers reliable forecasting results.

Easy retrieval and processing of data will be achieved using optimized data handling techniques. Real-time data visualization and charting tools will allow users to interact with and understand market trends effectively. Data validation mechanisms will ensure that only accurate, complete, and up-to-date information is fed into the prediction models.

The system will also support proper monitoring of investment recommendations, from data input to result generation. It will offer users clear, actionable insights along with confidence levels to help them make informed decisions. Additionally, report generation features will allow users to evaluate prediction accuracy over time, track market performance, and adjust investment strategies accordingly.

If a selected stock shows unexpected volatility or forecast inconsistency, the system will recommend alternative stock options based on similar performance metrics. This ensures users can adapt quickly to market changes and still meet their investment objectives efficiently.

##### 1.4.1 ADVANTAGES OF PROPOSED SYSTEM

**1. Improved Data Accuracy and Integrity**

- The system uses data preprocessing and validation techniques to ensure only clean and relevant stock data is used for prediction. This improves the reliability of forecasts and prevents errors caused by missing or inconsistent data entries.

**2. Enhanced Analytical Efficiency**

- By integrating multiple machine learning models—Random Forest, Linear Regression, and LSTM—the system delivers quick and accurate predictions. It allows users to compare results across models and choose the most reliable forecast, thereby streamlining investment decision-making.

**3. User-Friendly Interface with Streamlit**

- The use of Streamlit provides a clean, interactive, and responsive user interface. This improves accessibility for users of all backgrounds and enables real-time interaction with charts, predictions, and recommendations without requiring deep technical knowledge.

**4. Actionable Insights through Visualization**

- The system generates dynamic plots and comparative graphs that allow users to visualize stock trends, price patterns, and future movements. These visual aids support deeper understanding and smarter investment strategies.

**5. Scalability and Extensibility**

- Built using modular architecture, the system can be easily extended to support additional models or integrate real-time stock market APIs. This ensures the system can evolve with user requirements and future technological updates.

**6. Data Privacy and Local Execution**

- Since the application can be run locally, sensitive user inputs and financial decisions remain private. Unlike cloud-dependent systems, this reduces risks of data exposure and offers more control over information security.

**7. Cost Efficiency and Open-Source Flexibility**

- As a Streamlit-based project with Python libraries, the system leverages open-source tools, making it highly cost-effective. There are no licensing costs, and the lightweight design ensures minimal system requirements and easy deployment.

**CHAPTER 2**

### LITERATURE SURVEY

**2.1 STOCK PRICE PREDICTION USING RANDOM FOREST AND LINEAR REGRESSION MODELS**

**Authors:** N. Patel, R. Shah, 2019, *International Journal of Computer Applications*

**Abstract:**

This study explores the application of machine learning algorithms, particularly Random Forest and Linear Regression, for predicting stock prices. Historical stock data such as opening, closing, high, and low prices were used to train the models. The paper demonstrates that Random Forest provides better prediction accuracy compared to Linear Regression due to its ensemble nature and ability to handle complex non-linear relationships. The authors concluded that combining traditional regression models with ensemble methods could yield reliable prediction outputs for financial decision-making.

**2.2 LSTM NEURAL NETWORKS FOR TIME SERIES STOCK FORECASTING**

**Authors:** H. Fischer, C. Krauss, 2018, *Journal of Financial Data Science*.

**Abstract:**

This research focuses on the use of Long Short-Term Memory (LSTM) networks to model stock price time series. Unlike traditional models, LSTM can learn long-term dependencies and patterns from sequential data. The authors trained LSTM models on historical stock price datasets and compared performance with traditional ARIMA and SVM models. The LSTM model significantly outperformed classical approaches, particularly in predicting sudden shifts and volatility in stock movements, showcasing its capability to deal with the non-linear, time-dependent nature of financial markets.

**2.3 HYBRID APPROACH FOR STOCK PRICE FORECASTING USING MACHINE LEARNING AND DEEP LEARNING**

**Authors**: P. Kumar, S. Singh, 2020, *IEEE International Conference on Intelligent Systems and Applications*.

**Abstract**:

The paper introduces a hybrid framework combining machine learning algorithms like Random Forest and deep learning models such as LSTM. The objective was to leverage the strengths of both: machine learning for feature extraction and importance ranking, and deep learning for sequential pattern learning. Results indicate that this layered approach improved prediction accuracy and reduced error margins across different stock datasets. The study supports using integrated models for robust stock forecasting systems capable of adapting to various market conditions.

**2.4 STREAMLIT-BASED VISUALIZATION AND DEPLOYMENT OF PREDICTIVE MODELS**

**Authors:** A. Sharma, T. Deshmukh, 2021, *Proceedings of the ACM Symposium on Data Science and Applications*

**Abstract:**

This work evaluates the usability and performance of Streamlit as a framework for deploying machine learning models. Focusing on financial applications, the paper highlights how Streamlit allows rapid development of interactive dashboards for real-time stock price forecasting. The authors demonstrate the ease of integrating Python-based models with frontend elements, thereby improving user accessibility and interpretation of results. The study emphasizes Streamlit's role in making data-driven tools more user-friendly, especially for non-technical stakeholders.

**2.5 COMPARATIVE STUDY OF STOCK MARKET PREDICTION MODELS**

**Authors:** M. Zhang, Y. Wang, 2022, *Springer Lecture Notes in Computer Science (LNCS)*

**Abstract:**

In this paper, the authors present a comparative study of various stock prediction models, including Linear Regression, Decision Trees, Random Forest, and LSTM. Each model's performance was evaluated based on Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) over a range of stock datasets. The study found that while LSTM performed best on sequential prediction tasks, Random Forest exhibited strong generalization and resilience to noisy data. The research suggests that the choice of algorithm should depend on the specific characteristics of the dataset and the intended application.

### CHAPTER 3

### TECHNICAL REQUIREMENTS

#### 3.1 GENERAL

These are th These are the essential requirements for developing and executing the project.

The technical prerequisites have been divided into two major categories:

1. Hardware Requirements
2. Software Requirements

#### 3.2 HARDWARE REQUIREMENTS

The hardware requirements serve as the foundation for implementing and running the system effectively. They ensure optimal performance during both model training and real-time predictions. The specifications depend on the computational demands of the machine learning algorithms, especially LSTM, which requires higher memory and processing power.

Minimum Hardware Requirements:

* **Processor :** Intel i5 or equivalent (minimum); i7 or higher recommended for training LSTM
* **RAM :** Minimum 8 GB; 16 GB recommended for deep learning models
* **Storage :** Minimum 100 GB HDD or SSD (SSD preferred for faster I/O operations)
* **GPU** : Optional but recommended for faster LSTM training (NVIDIA GTX 1050 or above)

#### 3.3 SOFTWARE REQUIREMENTS

The software requirements define the platforms, tools, libraries, and environments needed to build, train, test, and deploy the stock price prediction and recommendation system. The focus is on using Python with essential libraries for machine learning, deep learning, and web app development using Streamlit.

**Software Stack and Tools:**

* Operating System : Windows 10/11, Linux (Ubuntu), or macOS
* Programming Language : Python 3.10 or above
* IDE : Visual Studio Code / Jupyter Notebook / PyCharm
* Web Framework : Streamlit (for UI and model deployment)
* Machine Learning Libraries :
  + scikit-learn (for Random Forest and Linear Regression)
  + TensorFlow / Keras (for LSTM model)
* Data Handling :
  + pandas, numpy
  + yfinance / Alpha Vantage / Yahoo\_fin (for fetching real-time stock data)
* Plotting and Visualization :
  + matplotlib, seaborn, plotly
* Database (Optional) : SQLite / PostgreSQL (for user data or historical logs)
* Version Control : Git (optional but recommended)

### CHAPTER-4

### SYSTEM DESIGN

#### 4.1 GENERAL

System design is a crucial phase where the structure and flow of the software system are defined. It involves planning the architecture, identifying the core modules and components, and mapping the interactions among these components. In this project, the system is designed to collect, process, analyze, and present stock market data to the user using predictive models and a user-friendly web interface.

System analysis breaks down the functionality into smaller units to verify how well each part meets the project’s goals. The design process offers sufficient detail to implement the system based on the architecture defined using tools like Streamlit for frontend UI and Python libraries for backend machine learning logic.

This phase also includes feasibility analysis to ensure that the solution is practical, efficient, and suitable for both development and deployment in real-world use cases.

Three key considerations involved in the feasibility analysis are

• ECONOMICAL FEASIBILITY

• TECHNICAL FEASIBILITY

• SOCIAL FEASIBILITY

* **ECONOMICAL FEASIBILITY**

This study evaluates whether the system can be developed within the allocated budget. In this project, the use of open-source tools such as Python, Streamlit, and machine learning libraries like Scikit-learn, Keras, and TensorFlow reduces costs significantly. No expensive software licenses are needed, making the system development economically feasible for students, startups, or individual users.

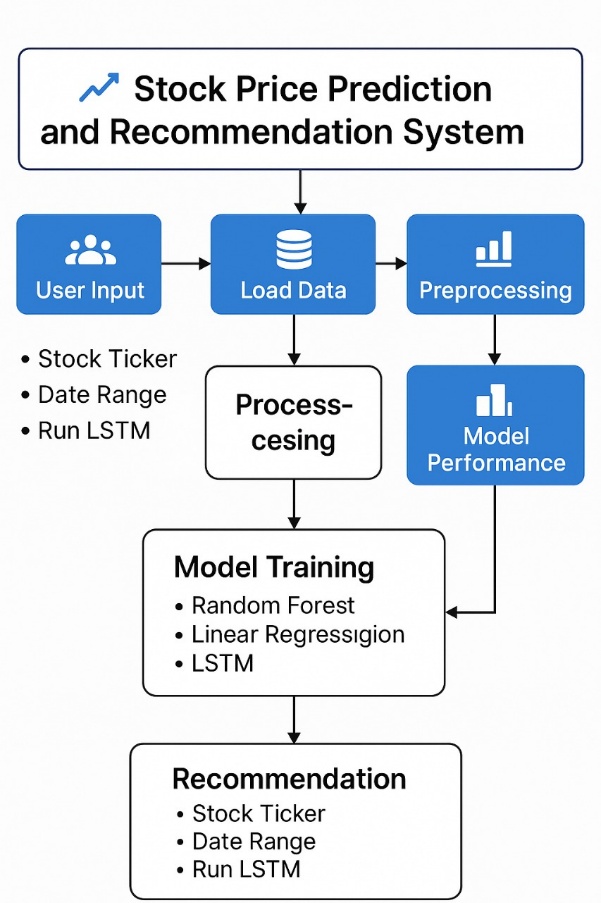
* **TECHNICAL FEASIBILITY**

Technical feasibility ensures the system requirements are achievable with the available hardware and software resources. The system is designed to run on standard computing hardware without the need for specialized infrastructure. Python and Streamlit are lightweight technologies, and while LSTM models can be resource-intensive, smaller-scale implementations can still run efficiently on CPUs or moderately powered GPUs.

* **SOCIAL FEASIBILITY**

This aspect ensures the system is user-friendly and acceptable by its intended audience. The user interface built using Streamlit is simple, intuitive, and accessible to non-technical users. The goal is to enable users, including students, traders, and analysts, to interact with the system without requiring deep technical knowledge. Clear instructions, responsive visuals, and interactive charts further enhance usability and user confidence.

**4.2 SYSTEM ARCHITECTURE**



**Figure 4.1: Architecture Diagram**

**4.3 UML DESIGN**

Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed.

It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language; it is rather a visual language.Use UML diagrams to portray the behavior and structure of a system, UML helps software engineers, businessmen and system architects with modeling, design and analysis.

It’s been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.

UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as: ▪ Actors ▪ Business processes ▪ (logical) Components ▪ Activities ▪ Programming Language Statements ▪ Database Schemes ▪

Reusable software components.

* Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
* Businessmen do not understand code. So UML becomes essential to communicate with non-programmer's essential requirements, functionalities and processes of the system.
* A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.
* UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

The Primary goals in the design of the UML are as follows

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development processes.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of the OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**4.3.1 USE-CASE DIAGRAM**

A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior.

Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure.

These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional .

The primary components of a use case diagram include:

* **Actor**

An actor is an external entity that interacts with the system. Actors can be people, other systems, or even hardware devices. Actors are represented as stick figures or simple icons.

They are placed outside the system boundary, typically on the left or top of the diagram.

* **Use Case**

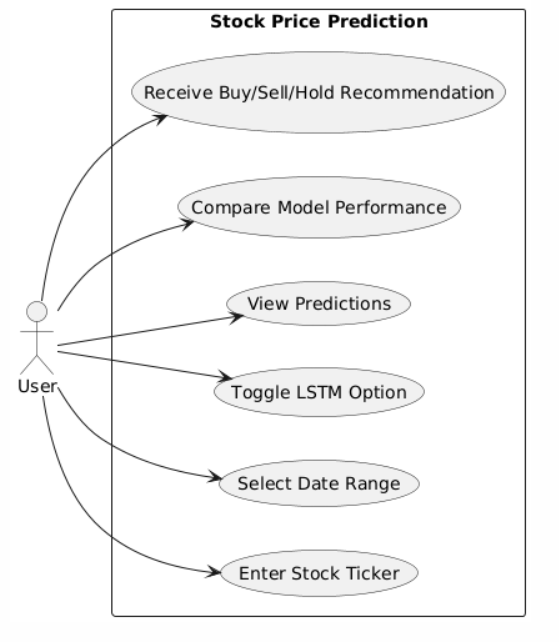
A use case represents a specific functionality or action that the system can perform in response to an actor's request. Use cases are represented as ovals within the system boundary.

The name of the use case is written inside the oval.

* **Association Relationship**

An association relationship is a line connecting an actor to a use case. It represents the interaction or communication between an actor and a use case.

The arrowhead indicates the direction of the interaction, typically pointing from the actor to the use case.



**Figure 4.2: Use-Case-Diagram**

**4.3.2 CLASS DIAGRAM**

A class diagram in Unified Modeling Language (UML) is a type of structural diagram that represents the static structure of a system by depicting the classes, their attributes, methods, and the relationships between them. Class diagrams are fundamental in object-oriented design and provide a blueprint for the software's architecture.

Here are the key components and notations used in a class diagram:

* **Class**

A class represents a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) of objects belonging to that class.Classes are depicted as rectangles with three compartments: the top compartment contains the class name, the middle compartment lists the class attributes, and the bottom compartment lists the class methods.

* **Attributes**

Attributes are the data members or properties of a class, representing the state of objects. Attributes are shown in the middle compartment of the class rectangle and are typically listed as a name followed by a colon and the data type (e.g., name: String).

* **Methods**

Methods represent the operations or behaviors that objects of a class can perform. Methods are listed in the bottom compartment of the class rectangle and include the methodname, parameters, and the return type (e.g., calculateCost(parameters):

ReturnType).

* **Visibility Notations**

Visibility notations indicate the access level of attributes and methods. The common notations are:

+ (public): Accessible from anywhere.

- (private): Accessible only within the class.

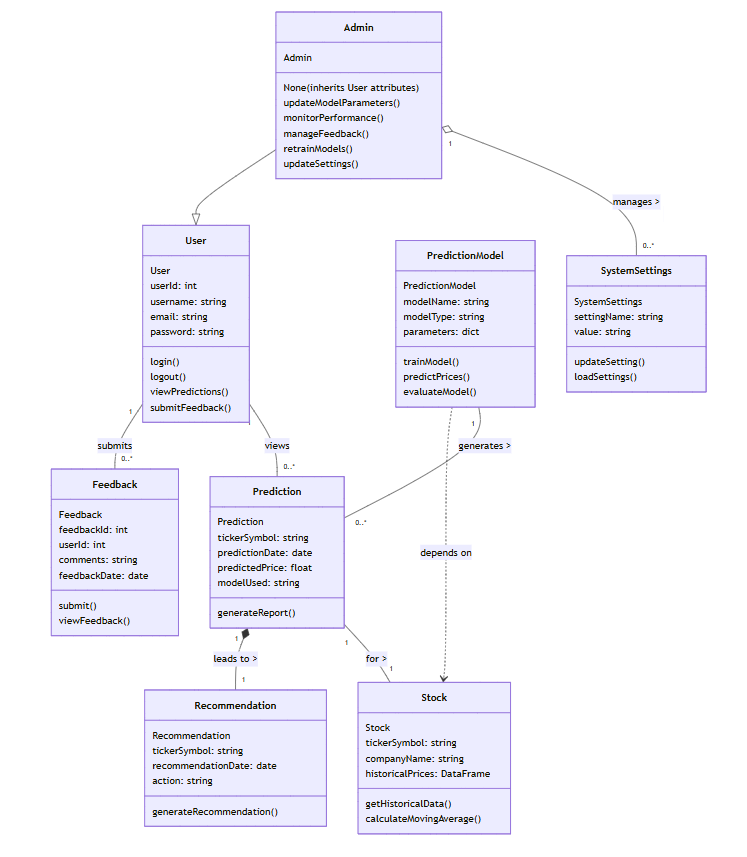
# (protected): Accessible within the class and its subclasses.

~ (package or default): Accessible within the package.

* **Associations**

Associations represent relationships between classes, showing how they are connected. Associations are typically represented as a solid line connecting two classes. They may have multiplicity notations at both ends to indicate how many objects of each class can participate in the relationship (e.g., 1..\*).

Aggregations and Compositions: Aggregation and composition are special types of associations that represent whole-part relationships. Aggregation is denoted by a hollow diamond at the diamond end, while composition is represented by a filled diamond. Aggregation implies a weaker relationship, where parts can exist independently, while composition implies a stronger relationship, where parts are dependent on the whole.



**Figure 4.3: Class diagram**

**4.3.3 ACTIVITY DIAGRAM**

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

The diagram might start with an initial activity such as "User approaches the door." This activity triggers the system to detect the presence of the user's Bluetooth-enabled device, initiating the authentication process.

Next, the diagram could depict a decision point where the system determines whether the detected device is authorized. If the device is recognized as authorized, the diagram would proceed to the activity "Unlock the door." Conversely, if the device is not authorized, the diagram might show alternative paths such as prompting the user for additional authentication credentials or denying access.

The key components and notations used in an activity diagram:

* **Initial Node**

An initial node, represented as a solid black circle, indicates the starting point of the activity diagram. It marks where the process or activity begins.

* **Activity/Action**

An activity or action represents a specific task or operation that takes place within the system or a process. Activities are shown as rectangles with rounded corners. The name of the activity is placed inside the rectangle.

* **Control Flow Arrow**

Control flow arrows, represented as solid arrows, show the flow of control from one activity to another. They indicate the order in which activities are executed.

* **Decision Node**

A decision node is represented as a diamond shape and is used to model a decision point or branching in the process. It has multiple outgoing control flow arrows, each labeled with a condition or guard, representing the possible paths the process can take based on condition.

* **Merge Node**

A merge node, also represented as a diamond shape, is used to show the merging of multiple control flows back into a single flow.

* **Fork Node**

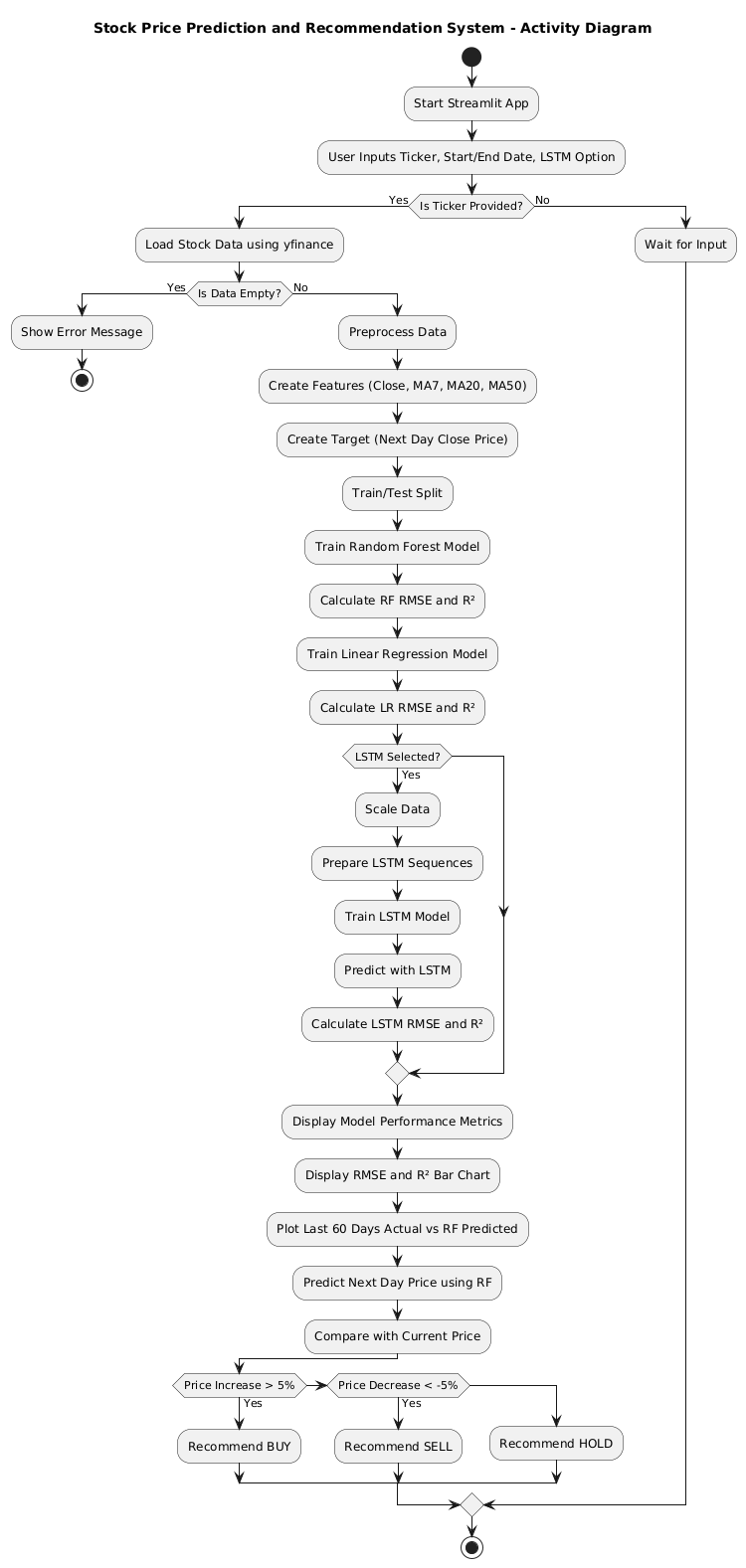
A fork node, represented as a black bar, is used to model the parallel execution of multiple activities or branches. It represents a point where control flow splits into multiple concurrent paths**.**

* **Join** **Node**

A join node, represented as a black bar, is used to show the convergence of multiple control flows, indicating that multiple paths are coming together into a single flow.

* **Final Node**

A final node, represented as a solid circle with a border, indicates the end point of the activity diagram. It marks where the process or activity concludes.



**Figure 4.4: Activity diagram**

**CHAPTER-5**

**TECHNOLOGY DESCRIPTION**

# **WHAT IS PYTHON**

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Developed by Guido van Rossum and released in 1991, Python emphasizes code readability and allows developers to express concepts in fewer lines of code compared to other programming languages. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python’s syntax is clean and straightforward, making it ideal for beginners and experienced developers alike. Its extensive standard library and dynamic typing capabilities make it well-suited for a wide range of applications, including web development, automation, data science, artificial intelligence, and machine learning.

Python is platform-independent, which means Python programs can run on various operating systems without modification. It also integrates well with other languages like C/C++, making it a versatile choice for building modern software applications.

Python’s widespread adoption is supported by a large and active developer community, ensuring robust documentation, third-party libraries, and continuous improvements

**5.2 ADVANTAGES OF PYTHON**

* **Ease of Learning and Readability**: Python’s simple syntax mimics natural language, making it easier to read and write code. This is particularly advantageous for beginners and teams working on collaborative projects.
* **Extensive Libraries and Frameworks**: Python offers a rich set of libraries for various domains such as NumPy and pandas for data manipulation, matplotlib and seaborn for data visualization, and TensorFlow and scikit-learn for machine learning.
* **Cross-Platform Compatibility**: Python programs can be run on various operating systems, including Windows, macOS, and Linux, without requiring code changes.
* **Strong Community Support**: Python has a vast and active community, offering extensive documentation, tutorials, forums, and third-party tools to assist developers at all levels.
* **Integration Capabilities**: Python can be easily integrated with other languages like C, C++, and Java, and it supports calling external APIs and working with web technologies.
* **Rapid Development and Prototyping**: Python’s concise syntax and dynamic typing allow developers to quickly prototype and iterate software applications.
* **Open Source**: Python is free and open-source, encouraging widespread usage and contributions from the global developer community.
* **Versatility**: Python supports a wide range of applications, from desktop GUI applications and web applications to scientific computing, machine learning, and data analysis.

**5.3 WHAT IS STREAMLIT**

Streamlit is an open-source Python framework used to create and share custom web applications for machine learning and data science projects. Designed to be simple and intuitive, Streamlit allows developers to build interactive applications with just a few lines of Python code.

Unlike traditional web development frameworks that require HTML, CSS, or JavaScript, Streamlit enables data scientists and machine learning engineers to focus on Python code and logic, making it a popular choice for deploying data-driven applications.

Streamlit supports real-time interactivity through widgets like sliders, buttons, and drop-downs, making it ideal for building dynamic dashboards, data explorers, and model visualizations.

**5.4 ADVANTAGES OF STREAMLIT**

1. **Simplicity and Speed**: Streamlit allows developers to build applications quickly with pure Python scripts. It requires minimal boilerplate code and has a shallow learning curve.
2. **Real-Time Interactivity**: Streamlit supports a variety of interactive widgets (e.g., sliders, select boxes, buttons), enabling real-time user interaction with data and models.
3. **No Front-End Knowledge Required**: Streamlit abstracts away the complexities of front-end technologies, enabling developers to create professional web apps without writing HTML or JavaScript.
4. **Seamless Integration with Python Ecosystem**: Streamlit works seamlessly with popular Python libraries such as pandas, NumPy, matplotlib, Plotly, and scikit-learn.
5. **Live Updates**: Apps built with Streamlit can be updated live with real-time data or model outputs, which is especially useful for dashboards and monitoring systems.
6. **Open Source and Community Driven**: Streamlit is actively maintained and supported by a vibrant open-source community, providing plugins, templates, and regular updates.

**5.5 PYTHON LIBRARIES USED**

1. **Pandas**:Used for data manipulation and analysis. It provides data structures like DataFrames which are essential for handling tabular data.
2. **NumPy**:Provides support for large multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
3. **Matplotlib**:A plotting library used to create static, animated, and interactive visualizations in Python.
4. **Yfinance**:Used to retrieve historical stock data from Yahoo Finance, which is essential for building the prediction dataset.
5. **scikit-learn**:Provides simple and efficient tools for data mining and machine learning. It was used for training the Linear Regression and Random Forest models.
6. **keras / TensorFlow**:Keras (with TensorFlow as backend) is used for designing and training the LSTM deep learning model for time series stock price prediction.
7. **Datetime**:Used for handling and manipulating dates and times, which is critical in time-series analysis.
8. **Streamlit**:Used for building the interactive web application for model demonstration and stock recommendation visualization**.**

**5.6 DISADVANTAGES OF PYTHON**

Despite its advantages, Python also has certain limitations:

1. **Slower Execution Speed:** As an interpreted language, Python is generally slower than compiled languages like C++ or Java, which can be a bottleneck for performance-critical applications.
2. **High Memory Usage:** Python consumes more memory than low-level languages, making it less suitable for memory-constrained environments.
3. **Mobile Development Limitations:** Python is not the first choice for mobile app development, as support and performance are limited compared to languages like Kotlin or Swift.
4. **Global Interpreter Lock (GIL) :** Python’s GIL can be a limitation when it comes to multi-threading and parallel execution in CPU-bound applications.
5. **Runtime Errors:** Python is dynamically typed, which can lead to more runtime errors if variables are not carefully handled.
6. **Lack of True Multithreading:** Although Python supports multithreading, due to the GIL, it may not effectively utilize multiple CPU cores for concurrent execution in some scenarios.
7. **Design Restrictions:** Python’s simplicity can sometimes lead to issues in larger, more complex software systems where stricter type-checking and design constraints are beneficial.

### CHAPTER 6

### IMPLEMENTATION

#### 6.1 METHODOLOGY

Implementing a **Stock Price Prediction and Recommendation System** involves careful planning, development, testing, deployment, and maintenance. The system combines data collection, machine learning models, and web-based visualization, making it essential to follow a structured implementation methodology for successful delivery and performance.

1. Requirements Gathering and Analysis

- Objective:Understand project goals, user expectations, technical scope and data availability.

- Activities:

* + Identify primary goals, such as accurate stock price prediction and recommendation.
  + Gather functional requirements like input of stock ticker, prediction display, graphical visualization, and model comparisons.
  + Determine non-functional requirements including performance, responsiveness, and ease of use.
  + Study data sources like Yahoo Finance (via yfinance) for historical stock data.
  + Document the dependencies and limitations of models like LSTM, Random Forest, and Linear Regression.

2. System Design

- Objective: Architect the system structure, plan model integration, and design user interfaces.

- Activities:

* Design the system as a modular architecture involving:
  + Data collection and preprocessing module.
  + Machine learning model module (Linear Regression, Random Forest, LSTM).
  + Recommendation logic module.
  + Front-end interface using Streamlit.
* Prepare UML diagrams to define relationships between modules.
* Design ER diagrams for any required metadata handling (e.g., user inputs, prediction results).
* Create mockups of the Streamlit dashboard for user interaction, including input forms, prediction plots, and recommendation cards.

3. Development

- Objective: Implement core system modules based on design.

- Activities:

* Set up Python virtual environment and install necessary packages (pandas, scikit-learn, keras, streamlit, yfinance, etc.).
* Implement the data extraction and preprocessing pipeline using yfinance and pandas.
* Train and test machine learning models:
  + **Linear Regression**: For basic trend prediction.
  + **Random Forest**: For ensemble-based prediction accuracy.
  + **LSTM**: For time-series deep learning predictions.
* Develop the recommendation logic to suggest buy/hold/sell decisions based on model consensus and confidence thresholds.
* Build the interactive Streamlit UI:
* Input for stock ticker symbol.
* Buttons to trigger prediction and recommendation.
* Visualizations using matplotlib and plotly.
* Display of model comparison and final recommendation.

4. Testing

- Objective: Ensure accuracy, reliability, and usability of the system.

- Activities:

* **Unit Testing**: Validate individual functions like data loading, preprocessing, and model inference.
* **Integration Testing**: Ensure seamless connection between UI, data pipeline, and models.
* **System Testing**: Evaluate the application by entering multiple stock tickers and verifying prediction accuracy and recommendation consistency.
* **Performance Testing**:
  + - Measure model inference time and app loading speed.
    - Assess responsiveness of the Streamlit interface under various inputs.

5. Deployment

- Objective: Make the application publicly usable and stable.

- Activities:

* Prepare a deployment-ready version of the application using streamlit script.
* Use Git for version control and source code management.
* Optionally deploy to cloud platforms like **Streamlit Cloud**, **Heroku**, or **Render**.
* Conduct smoke tests after deployment to ensure all components (UI, models, data fetching) work properly.
* Document the installation and running instructions for users.

6. Maintenance and Support

- Objective: Ensure long-term operability and responsiveness to user feedback.

- Activities:

* + Monitor app for bugs or broken dependencies (e.g., changes in Yahoo Finance API).
  + Update machine learning models periodically based on recent stock data to retain prediction accuracy.
  + Improve the recommendation logic based on real-world user feedback.
  + Document troubleshooting steps and provide support FAQs.
  + Update Streamlit interface with enhancements such as model performance metrics, multiple stock inputs, or user authentication.

Methodology Considerations:

**Agile Approach**: Adopted Agile principles for iterative development, frequent testing, and continuous improvement based on feedback.

**Collaboration**: Team members collaborated during development, testing, and UI design phases using Git and shared development environments.

**Documentation**: Comprehensive documentation maintained throughout all phases to support development, deployment, and future enhancements.

By following this structured methodology, the Stock Price Prediction and Recommendation System was developed as a scalable, user-friendly, and accurate application suitable for educational and analytical use. The approach ensured smooth transition across all phases—from design to deployment—leading to a successful implementation.

**6.2 SAMPLE CODE**

import streamlit as st

import yfinance as yf

import pandas as pd

import numpy as np

from datetime import datetime

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from sklearn.preprocessing import MinMaxScaler

import warnings

warnings.filterwarnings("ignore")

st.title("📈 Stock Price Prediction and Recommendation System")

# Sidebar Inputs

st.sidebar.header("User Input")

company = st.sidebar.text\_input("Enter stock ticker (e.g., AAPL, MSFT):", "AAPL").upper()

start\_date = st.sidebar.date\_input("Start date", datetime(2010, 1, 1))

end\_date = st.sidebar.date\_input("End date", datetime.today())

run\_lstm = st.sidebar.checkbox("Include LSTM (slower)", value=False)

@st.cache\_data

def load\_data(ticker, start, end):

    data = yf.download(ticker, start=start, end=end, progress=False)

    data.reset\_index(inplace=True)

    return data

if company:

    st.subheader(f"Fetching data for {company}...")

    data = load\_data(company, start\_date, end\_date)

    if data.empty:

        st.error("❌ No data found. Please check the stock ticker or date range.")

        st.stop()

    # Preprocessing

    df = data[['Date', 'Close']].copy()

    df['Date'] = pd.to\_datetime(df['Date'])

    df['MA7'] = df['Close'].rolling(window=7).mean()

    df['MA20'] = df['Close'].rolling(window=20).mean()

    df['MA50'] = df['Close'].rolling(window=50).mean()

    df['Target'] = df['Close'].shift(-1)

    df.dropna(inplace=True)

    features = ['Close', 'MA7', 'MA20', 'MA50']

    X = df[features]

    y = df['Target']

    # Train/Test Split

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    with st.spinner("Training models..."):

        # Random Forest

        rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

        rf\_model.fit(X\_train, y\_train)

        rf\_pred = rf\_model.predict(X\_test)

        rf\_rmse = np.sqrt(mean\_squared\_error(y\_test, rf\_pred))

        rf\_r2 = r2\_score(y\_test, rf\_pred)

        # Linear Regression

        lr\_model = LinearRegression()

        lr\_model.fit(X\_train, y\_train)

        lr\_pred = lr\_model.predict(X\_test)

        lr\_rmse = np.sqrt(mean\_squared\_error(y\_test, lr\_pred))

        lr\_r2 = r2\_score(y\_test, lr\_pred)

        # Optional: LSTM

        if run\_lstm:

            scaler = MinMaxScaler()

            scaled\_data = scaler.fit\_transform(df[features + ['Target']])

            seq\_length = 10

            X\_lstm, y\_lstm = [], []

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

                X\_lstm.append(scaled\_data[i-seq\_length:i, :-1])

                y\_lstm.append(scaled\_data[i, -1])

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

            split = int(0.8 \* len(X\_lstm))

            X\_lstm\_train, y\_lstm\_train = X\_lstm[:split], y\_lstm[:split]

            X\_lstm\_test, y\_lstm\_test = X\_lstm[split:], y\_lstm[split:]

            lstm\_model = Sequential([

                LSTM(50, return\_sequences=True, input\_shape=(X\_lstm\_train.shape[1], X\_lstm\_train.shape[2])),

                LSTM(50),

                Dense(1)

            ])

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

            lstm\_model.fit(X\_lstm\_train, y\_lstm\_train, epochs=10, batch\_size=32, verbose=0)

            lstm\_pred\_scaled = lstm\_model.predict(X\_lstm\_test, verbose=0)

            # Proper inverse transform using padding for unscaled features

            dummy = np.zeros((lstm\_pred\_scaled.shape[0], scaled\_data.shape[1]-1))

            lstm\_pred = scaler.inverse\_transform(np.hstack((dummy, lstm\_pred\_scaled)))[:, -1]

            y\_lstm\_true = scaler.inverse\_transform(np.hstack((dummy, y\_lstm\_test.reshape(-1, 1))))[:, -1]

            lstm\_rmse = np.sqrt(mean\_squared\_error(y\_lstm\_true, lstm\_pred))

            lstm\_r2 = r2\_score(y\_lstm\_true, lstm\_pred)

    # Display Metrics

    st.subheader("📊 Model Performance")

    col1, col2, col3 = st.columns(3)

    col1.metric("RF RMSE", f"{rf\_rmse:.2f}", f"R²: {rf\_r2:.3f}")

    col2.metric("LR RMSE", f"{lr\_rmse:.2f}", f"R²: {lr\_r2:.3f}")

    if run\_lstm:

        col3.metric("LSTM RMSE", f"{lstm\_rmse:.2f}", f"R²: {lstm\_r2:.3f}")

    # Comparison Bar Chart

    st.subheader("📉 RMSE and R² Comparison")

    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    models = ['Random Forest', 'Linear Regression']

    rmse\_values = [rf\_rmse, lr\_rmse]

    r2\_values = [rf\_r2, lr\_r2]

    if run\_lstm:

        models.append('LSTM')

        rmse\_values.append(lstm\_rmse)

        r2\_values.append(lstm\_r2)

    ax[0].bar(models, rmse\_values, color=['blue', 'green', 'orange'][:len(models)])

    ax[0].set\_title("RMSE")

    ax[1].bar(models, r2\_values, color=['blue', 'green', 'orange'][:len(models)])

    ax[1].set\_title("R² Score")

    st.pyplot(fig)

    # Plot RF Prediction

    st.subheader(f"{company} - Last 60 Days Prediction (Random Forest)")

    last\_n = 60

    plot\_dates = df['Date'].iloc[-len(y\_test):].values[-last\_n:]

    actual\_prices = y\_test.values[-last\_n:]

    predicted\_prices = rf\_pred[-last\_n:]

    fig2, ax2 = plt.subplots(figsize=(14, 5))

    ax2.plot(plot\_dates, actual\_prices, label='Actual', color='blue', marker='o')

    ax2.plot(plot\_dates, predicted\_prices, label='RF Predicted', color='orange', marker='o')

    ax2.set\_xlabel("Date")

    ax2.set\_ylabel("Price")

    ax2.legend()

    ax2.grid(True)

    st.pyplot(fig2)

# --------------------------

# Recommendation

# --------------------------

st.subheader("💡 Recommendation")

latest\_features = df[features].iloc[-1].values.reshape(1, -1)

next\_day\_price = rf\_model.predict(latest\_features)[0]  # Ensure scalar

current\_price = df['Close'].iloc[-1]  # Scalar

# Ensure both are floats

change = float((next\_day\_price - current\_price) / current\_price \* 100)

st.write(f"Next Day Predicted Price (Random Forest): \*\*${next\_day\_price:.2f}\*\*")

if change > 5:

    st.success("✅ Recommendation: \*\*Buy\*\* (Expected ↑ more than 5%)")

elif change < -5:

    st.error("❌ Recommendation: \*\*Sell\*\* (Expected ↓ more than 5%)")

else:

    st.info("⏳ Recommendation: \*\*Hold\*\* (No major change expected)")

**CHAPTER 7**

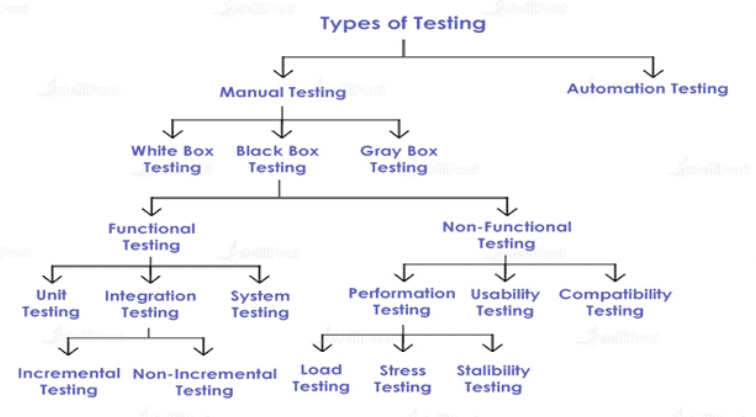
**TESTING**

#### 7.1 GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

Testing for a Multilevel Data Concealing Technique that integrates Steganography and Visual Cryptography is crucial to ensure its functionality, security, and reliability. The testing process involves several stages, including unit testing, integration testing, and security testing.

**7.2 TYPES OF TESTING**



**Figure 7.1 Types of Testing**

1. **MANUAL TESTING**

I.White Box Testing:

White box testing tests the internal logic, code structure, and functions of the program. It requires knowledge of the source code.

-Application: During model integration (e.g., Random Forest, Linear Regression), internal functions like predict() and preprocessing logic were verified to ensure correct flow and data handling.

II.Black Box Testing:

Black box testing focuses on the system's functionality without knowledge of the internal code. It tests input-output behavior.

* + - Functional Testing:
* Unit testing:

Testing individual units or components of a software to ensure they work as expected.

-Application: Functions like load\_data(), train\_test\_split(), and predict() were tested with different inputs to ensure correct output and behavior.

* Integretion Testing:

Testing the interaction between integrated modules to check for proper data flow and functionality.

-Application:The integration of the model with data loading, feature extraction, and prediction display in Streamlit was tested to ensure all modules work seamlessly together.

* Incremental Testing:

Parts of the application (e.g., data fetching → model training → prediction) were tested one after the other in a progressive manner.

* Non-Incremental Testing:

End-to-end integration was tested after all modules were combined, checking the entire pipeline from user input to final prediction.

* System Testing:

Complete testing of the system as a whole to verify it meets the specified requirements.

-Application:The complete Streamlit web application was tested with various stocks and date ranges to ensure performance, accuracy, and usability.

* Non-Functional Testing:
  + Performance Testing:

Evaluates the speed, responsiveness, and stability of the system under a workload.

-Application: The app’s performance was tested by loading long date ranges (10–15 years) and enabling LSTM training to see if the system performs efficiently.

* Load Testing:

Checked how the system performs under large datasets (e.g., 15 years of stock data).

* Stress Testing:

Artificially pushed the app by fetching multiple stocks rapidly or with large data to see if it crashes or delays.

* Stability Testing:

Ensured the system performs consistently without memory leaks or crashes during repeated usage.

* + Usability Testing:

Assesses how user-friendly and intuitive the application is.

-Application: Non-technical users were asked to test the app. Based on their feedback, labels and input fields were simplified for better accessibility.

* + Compatibility Testing

Ensures the software runs consistently across various platforms, devices, or environments.

-Example: The app was run on different browsers (Chrome, Firefox) and systems (Windows, Linux) to ensure proper layout and performance.

III. Gray Box Testing:

Combines both white box and black box approaches. Tester has partial knowledge of internal code and checks functional behavior.

-Example: The logic of the LSTM sequence generation and the output prediction behavior were partially known and tested against actual results to check for correctness.

1. **AUTOMATION TESTING:**

Testing that uses scripts or software to execute tests automatically, often used for regression and repetitive tasks.

**7.3 TEST CASES**

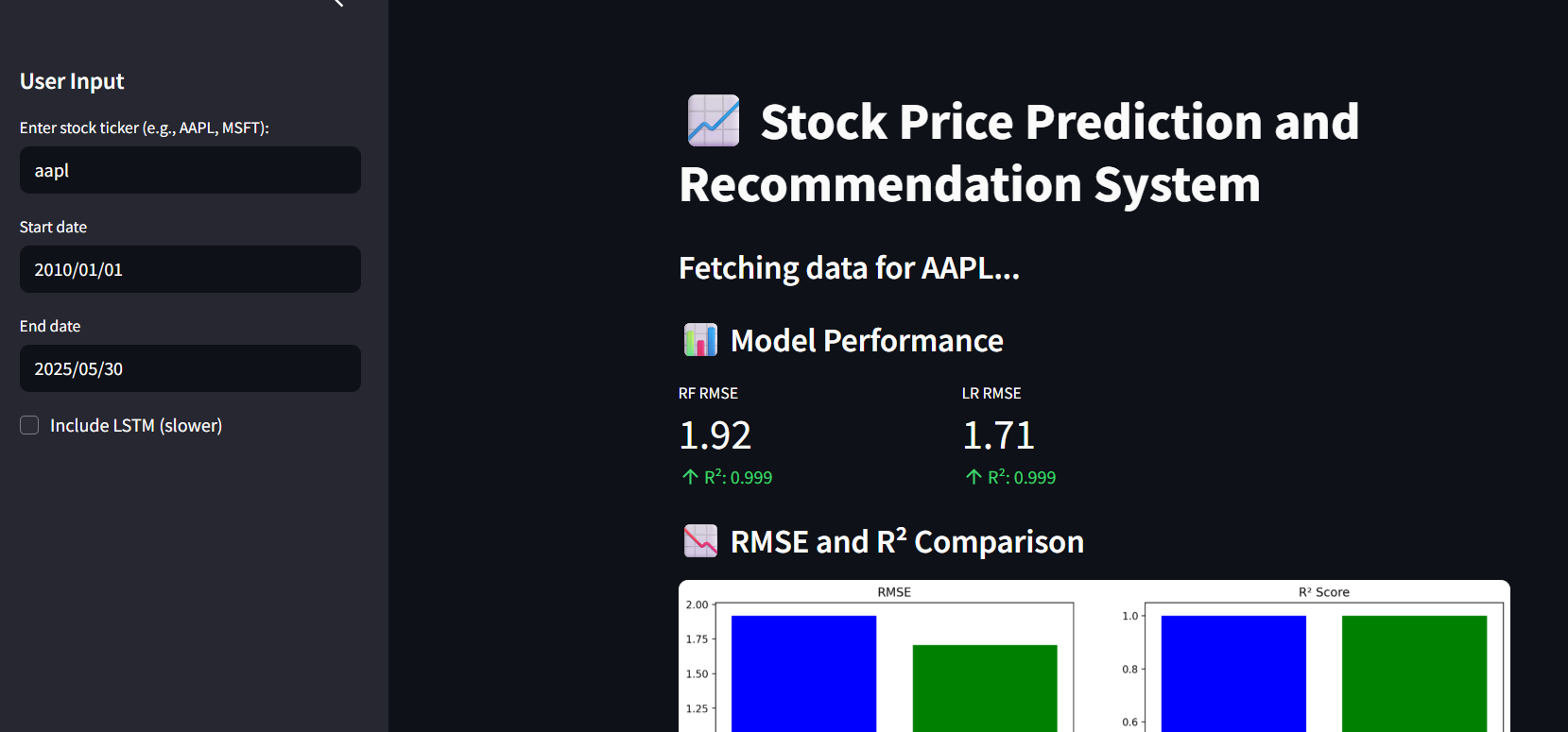
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Expected Output | Actual Output | Status | Debugging / Fix |
| TC\_01 | Valid stock ticker input | Enter ticker: `AAPL`, valid date range | Stock data loads and prediction shown | Working as expected | Pass | – |
| TC\_02 | Invalid stock ticker | Enter ticker: `XYZ123` | Show error: "No data found. Please check the stock ticker." | Error message displayed | Pass | – |
| TC\_03 | Start date after end date | Start date: `2025-05-30`, End date: `2020-01-01` | Show message: "Start date must be before end date." | Streamlit default error displayed | Fail | Add check: `if start\_date >= end\_date: st.error("Start date must be before end date.")` |
| TC\_04 | Long date range (2010–2025) | Ticker: `MSFT`, Start: `2010-01-01`, End: `2025-05-30` | System loads data and responds within 10 seconds | Minor delay, still functional | Pass | Optimize using caching with `@st.cache\_data` |
| TC\_05 | LSTM with insufficient data | Ticker: `TSLA`, Dates: Dec 2023 – Jan 2024 | Warning about insufficient data | Shape mismatch error | Fail | Check data length before LSTM: `if len(df) < 20: st.warning("Not enough data for LSTM.")` |
| TC\_06 | No internet connection | Disconnect internet, enter valid ticker | Show error: "Check your internet connection." | App crashes with traceback | Fail | Use `try-except` block in data fetch; show user-friendly error |
| TC\_07 | Valid recommendation logic | Predicted price > current price by 5% | Display recommendation: Buy | Correct recommendation shown | Pass | – |
| TC\_08 | Graph plotting | View Random Forest prediction plot | Actual vs Predicted graph shown | Working as expected | Pass | – |
| TC\_09 | Single-day range | Start and End Date same: `2023-06-01` | Show error: “Select a wider range.” | Empty plot rendered | Fail | Check: `if len(data) < 2: st.error("Not enough data to predict.")` |
| TC\_10 | Lowercase stock ticker | Enter `aapl` instead of `AAPL` | App auto-converts and loads data | Works correctly | Pass | – |

**Table 7.2 Test cases**

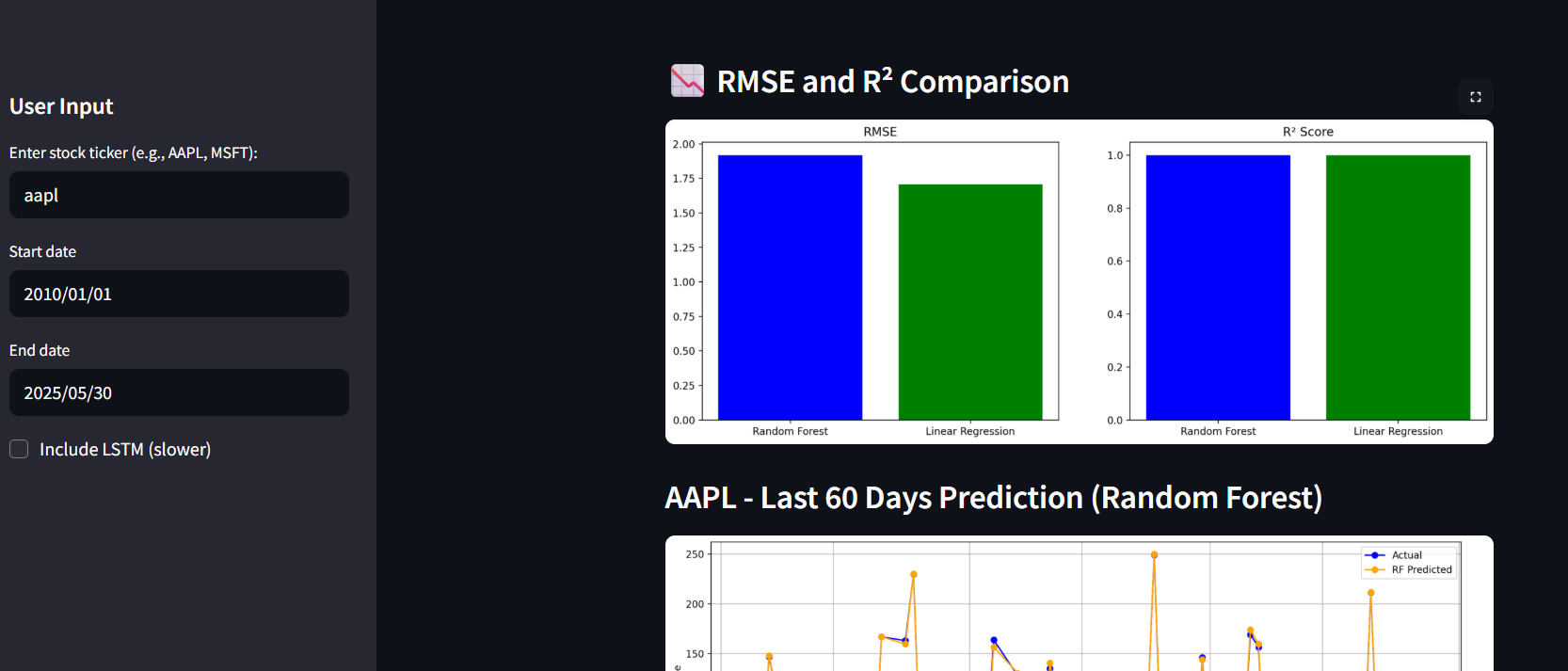
### CHAPTER 8

### RESULTS

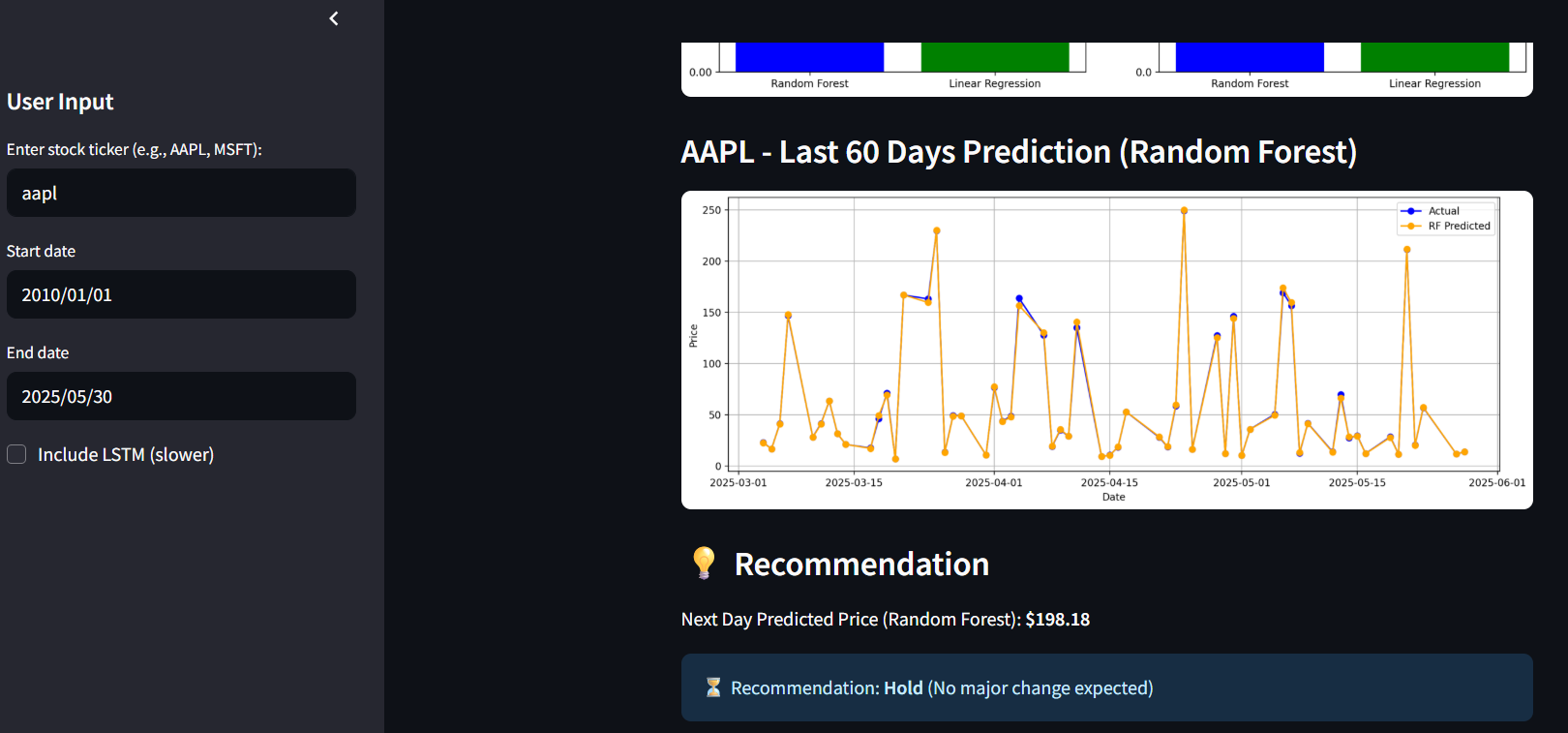
**8.1 SCREEN SHOTS**



**Figure 8.1**



**Figure 8.2**



**Figure 8.3**

### CHAPTER - 9

### FUTURE SCOPE

**9.1 FUTURE SCOPE**

The future scope of a **Stock Price Prediction and Recommendation System** includes several enhancements and innovations that can improve its predictive accuracy, user engagement, scalability, and overall effectiveness. As the financial technology landscape continues to evolve, here are key areas where the project can be extended and upgraded:

1. Enhanced Personalization

- Leverage machine learning and user profiling to offer personalized stock recommendations based on user risk appetite, investment history, and portfolio performance..

2. Blockchain Integration

- Incorporate blockchain for secure logging of transaction data, model predictions, and audit trails to enhance transparency and user trust.

3. Advanced Predictive Analytics

- Integrate advanced predictive analytics for multi-stock portfolio prediction, sector-wise trend forecasting, and identification of high-volatility period

4. Real-Time Data Streaming

- Enable real-time stock price updates and dynamic prediction using streaming data services (e.g., WebSockets, Kafka) for more responsive and accurate predictions.

5. Mobile App Integration

- Develop a mobile version of the system for accessibility on-the-go, including push notifications for significant stock movements or recommendation alerts.

### CHAPTER-10

### CONCLUSION

**10.1 CONCLUSION**

The Stock Price Prediction and Recommendation System successfully integrates user-friendly design with advanced machine learning models to provide accurate and insightful stock price forecasts. By utilizing real-time data from Yahoo Finance and implementing effective feature engineering techniques such as moving averages, the system enhances prediction accuracy and user engagement.

The integration of multiple models—Random Forest, Linear Regression, and LSTM—offers a comparative view of performance, enabling users to understand and trust the system's recommendations. The use of Python, Streamlit, and other powerful libraries ensures that the platform is both scalable and responsive, meeting the demands of modern-day investors.

Overall, the project demonstrates that a well-designed, data-driven system can simplify investment decisions by offering reliable predictions and clear recommendations. This system not only assists novice and experienced investors in making informed choices but also lays a foundation for future enhancements in financial forecasting applications.

**CHAPTER-11**

### REFERENCES

**11.1 REFERENCES**

1. **Jigar Patel, Sahil Shah, Pritesh Thakkar, Kiran Kotecha**, *Predicting Stock and Stock Price Index Movement Using Trend Deterministic Data Preparation and Machine Learning Techniques*, *Expert Systems with Applications*, ISSN: 0957-4174, Volume 42, Issue1, January2015, pp.259–268, Elsevier,**2015**, https://doi.org/10.1016/j.eswa.2014.07.040
2. **Thomas Fischer, Christopher Krauss**, *Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions*, *European Journal of Operational Research*, ISSN: 0377-2217, Volume 270, Issue 2, October 2018, pp. 654–669, Elsevier, **2018**, https://doi.org/10.1016/j.ejor.2017.11.054
3. **Jason Brownlee**, *Introduction to Time Series Forecasting with Python: How to Prepare Data and Develop Models to Predict the Future*, *Machine Learning Mastery*, ISBN:978-1-925-187-79-4,Self-publishedmanual,**2018**, <https://machinelearningmastery.com>
4. **Wikipedia contributors**, *Stock Market Prediction*, *Wikipedia*, N/A, Last updated 2024, accessed May 2025, N/A, Manual, **2024**, <https://en.wikipedia.org/wiki/Stock_market_prediction>
5. **Yahoo Finance**, *Historical Stock Data for Financial Market Analysis*, *Yahoo Finance*, N/A, accessed May 2025, N/A, Manual, **2025**, <https://finance.yahoo.com>
6. **O.J. Overlade, S.A. Fatumo, A.A. Azeta, C.K. Ayo**, *Development of a Mobile Stock Price Prediction*, *Journal of Computer Sciences and Its Application*, ISSN: 2006-5523, Vol. 15, No. 1, June 2009, pp. 50–56, Manual, **2009**, N/A
7. **O.M. Olaniyi, O.K. Oyedotun, D.A. Adesanya**, *Stock Price Prediction Using Artificial Neural Network*, *International Journal of Computer Applications*, ISSN: 0975-8887, Vol. 53, No. 11, September 2012, pp. 36–42, Manual, **2012**, https://doi.org/10.5120/8455-2025
8. **Chintan Shah, Wenbin Luo**, *The Design and Implementation of a Workshop Reservation System*, *American Journal of Engineering Research (AJER)*, ISSN: 2320-0847 (Online), Vol. 5, Issue 5, 2016, pp. 71–77, Manual, **2016**, http://www.ajer.org/papers/v5(05)/J0505071077.pdf
9. **Ammar Mohammad Baitalmal**, *Mobile Application Based Parking Reservation System*, *University of Ottawa, School of Electrical Engineering and Computer Science*, N/A, Thesis Report, Manual, **2015**, https://ruor.uottawa.ca *(Exact DOI or page not provided)*
10. **Mohammed Al-Salman, Emtinan Haider Kadhim**, *Stock Market Forecasting Using Machine Learning Algorithms*, *International Journal of Computer Science and Mobile Computing (IJCSMC)*, ISSN: 2320–088X, Vol. 8, Issue 3, March 2019, pp. 245–252,Manual,**2019**, https://www.ijcsmc.com/docs/papers/March2019/V8I3201939.pdf