RiMS

PROJECT REPORT

ON

SHARE MARKET PREDICTION

SUBMITTED

TO

ROURKELA INSTITUTEOF MANAGEMENTSTUDIES

(As a partial fulfilment of the requirement for the award of Degree)

FOR

MASTER IN COMPUTER APPLICATION

SUBMITED BY

SANJANA PRADHAN

REGD NO: 2305260020

MCA 4TH SEMESTER (2023 – 2025)

ROURKELA INSTITUTE OF MANAGEMENT STUDIES

(Affiliated to Biju Patnaik University of Technology)

Rourkela Institute of Management Studies

Rourkela

Department of Computer Science

Rourkela Institute of Management Studies

Chhend, Rourkela-15, Odisha

Phone:0661 2480482

Fax: 91-0661-1480665

Mail: rkl_rimsgrol@sancharnet.in

Visit: www.rims_edu.com

CERTIFICATE OF EXAMINATON

This is to certify that this project report entitled **"SHARE MARKET PREDICTION"** submitted by **SANJANA PRADHAN** of 4th semester, **Rourkela Institute of Management Studies, Rourkela,** is accepted as partial fulfillment of requirements for the degree in Master in Computer Applications, under **Biju Patnaik University of Technology, Rourkela** this has been verified by us and found be original up to our satisfaction.

External Examiner

Rourkela Institute of Management Studies

Rourkela

Department of Computer Science

Rourkela Institute of Management Studies

Chhend, Rourkela-15, Odisha

Phone:0661 2480482

Fax: 91-0661-1480665

Mail: rkl_rimsgrol@sancharnet.in

Visit: www.rims_edu.com

CERTIFICATE OF EXAMINATON

This is to certify that this project report entitled **"SHARE MARKET PREDICTION "** submitted by **SANJANA PRADHAN** of 4th semester, **Rourkela Institute of Management Studies, Rourkela,** is accepted as partial fulfilment of requirements for the degree in Master In Computer Applications, under **Biju Patnaik University of Technology, Rourkela** this has been verified by us and found be original up to our satisfaction.

Internal Examiner

Rourkela Institute of Management Studies

Rourkela

Department of Computer Science

Rourkela Institute of Management Studies

Chhend, Rourkela-15, Odisha

Phone:0661 2480482

Fax: 91-0661-1480665

Mail: rkl_rimsgrol@sancharnet.in

Visit: www.rims_edu.com

CERTIFICATE

This is to certify that this project entitled **"SHARE MARKET PREDICTION "** has been and submitted by **SANJANA PRADHAN,** M.C.A 2020-2022, **Rourkela Institute of Management Studies, Rourkela,** has been examined by us.

He is found fit and approved for the award of **"Master in Computer Application "**Degree.

To the best my knowledge this work has not been submitted for the award of any other degree.

I wish all success in his life.

Dean Academic

RIMS, ROURKELA

RiMS

DECLARATION

I, **SANJANA PRADHAN**, hereby declare that the project report entitled "**SHARE MARKET PREDICTION** " is of my work. The above work I submitted to "**Biju PatnaikUniversity of Technology, Rourkela"** for the award of **"Master in Computer Applications**" Degree.

To the best of my knowledge, this work has not been submitted or published anywhere for the award of any degree.

SANJANA PRADHAN

ACKNOWLGEMENT

I would like to express my gratitude to **Prof. Bibhudendu Panda(HOD)**for his guidance and support during the project work.

I am deeply indebted to **Rourkela Institute of Management Studies, Chhend, Rourkela,** for providing me an opportunity to undertake a project work entitled **"SMARKET PREDICATION SYSTEM".**

I am grateful to my project guide **Mr. Damodar Nayak** without his guidance it would not have been possible on my part to complete the project.

I acknowledge the help and co-operation received from all my team members in making this project.

I consider myself fortunate that I have successfully completed this project; I acknowledge my sincere gratitude to all those works and ideas that had helped me in writing this project.

SANJANA PRADHAN

University Roll No:2305260020

MCA (2023-2025)

Rourkela Institute of Management Studies,

Rourkela.

Abstract

The **Share Market Prediction** project is designed to forecast stock price trends using a combination of machine learning models, technical indicators, and sentiment analysis. The stock market is influenced by various factors, including past price data, technical patterns, and market sentiment from financial news. This project integrates these dimensions into one system.

The application is built using **Streamlit** and combines **seven models**: Random Forest, XGBoost, LSTM, ARIMA, Prophet, Linear Regression, and Ridge Regression. By creating an ensemble of these algorithms, the system generates more reliable predictions for short-term and medium-term stock price movements.

The system also applies **real-time sentiment analysis** using **TextBlob** and **NLTK** libraries on financial news headlines to enhance prediction accuracy. It includes risk assessment indicators, confidence scoring, and trading signals for users.

CONTRIBUTION OF INDIVIDUAL TEAM MEMBERS

Name of the Student(s)	Registration Number	Contributions
Purnamita Swain	2305260017	Core model development and data handling
Dipti Rani Bindhani	2305260009	Frontend & User interaction
Sanjana Pradhan	2305260020	Sentiment Analysis & Research for all the work

TABLE OF CONTENTS

Chapter 1 – Introduction

Chapter 2 – System Analysis

Chapter 3 – System Design

Chapter 4 – Implementation, Testing and Code explanation

CHAPTER 5 CONCLUSION & FUTURE ENHANCEMENTS

CHAPTER 1 INTRODUCTION

1.1 Introduction

The share market is a highly volatile and dynamic environment influenced by a variety of external factors such as global economic trends, political events, company-specific developments, and, most importantly, investor sentiment driven by real-time news and social media. The behavior of financial markets is often non-linear and chaotic, where even minor external shocks can lead to significant price fluctuations within a short period.

Given this inherent complexity, predicting stock price movements with high precision has always been a challenging task. Traditional models relying solely on historical price patterns often fail to capture the influence of sudden news events or shifts in market psychology. Moreover, the increasing speed of information flow, driven by the internet and social platforms, has made it even more critical to incorporate sentiment-based signals into predictive models.

This project proposes the development of a **Multi-Algorithm Stock Predictor**, a comprehensive solution that integrates multiple machine learning algorithms and sentiment analysis to forecast stock price trends with greater reliability. The system leverages the strengths of various predictive models such as **Random Forest**, **XGBoost**, **LSTM (Long Short-Term Memory networks)**, **ARIMA**, **Prophet**, and other regression techniques to capture both linear and non-linear patterns present in stock market data.

To improve prediction robustness, the system applies an **ensemble approach**, where outputs from these models are aggregated to mitigate the biases or shortcomings of individual algorithms. In addition to statistical learning methods, the platform incorporates **technical indicators** like the **20-day and 50-day Simple Moving Averages (SMA)**, widely used by traders to identify short- and mid-term trends. These indicators assist in highlighting bullish or bearish signals, helping users identify potential entry and exit points for trades.

Beyond price data, the project also integrates **real-time sentiment analysis** by scraping financial news articles and headlines using API-based data collection pipelines. Leveraging **Natural Language Processing (NLP)** tools such as **TextBlob** and **NLTK**, the system evaluates whether the prevailing market sentiment is positive, negative, or neutral. This additional sentiment dimension helps the model adjust its forecasts based on external events, such as earnings announcements or geopolitical developments, which often precede significant price swings.

The **Streamlit-based dashboard** offers a user-friendly interface that allows traders and investors to interact with live predictions, technical charts (such as candlestick patterns and SMA overlays), sentiment scores, and risk assessments. Users can customize forecast horizons, explore model consensus, and view confidence scores, enabling a more holistic view of market conditions before making trading decisions.

This project has significant practical value for both professional traders and individual investors by enabling **data-driven decision-making**, reducing reliance on intuition or guesswork. In particular, during volatile periods where traditional strategies might fail, the combined insights from technical analysis, machine learning, and sentiment data equip users with more actionable and timely information.

Overall, the **Multi-Algorithm Stock Predictor** bridges the gap between quantitative analysis and real-time market sentiment, providing a modern approach to stock market forecasting that adapts to fast-changing financial landscapes.

1.2 Problem Statement

Prediction of share, however, has not been an easy job since the concept started
dating back to the development of New York Stock Exchange in 1817, major approaches of prediction of the stocks have been made with and without the use of computing systems.

The condition of the market is said to be unpredictable and none is ever to benefit from the analysis that is made based on the data. The construct of the market and its environment constrain the investors from windfall gains as the information about the system is publicly available and the chances that the same investor may attain the best prices in stocks is paradoxical.Stock values are changing depending on the market conditions day by day. The challengeis to guide the investors for the right time to buy and sell the shares. There are many regression and classifiers available for the prediction. Effort is to need for determining
the best technique that provide better result in predicting the stock prices and give accuratetrends.

1.3 Objectives

The main objectives of the Stock Market Analysis and Prediction project are:

- To predict future value of company stock
- To analyze the current state of the market
- To identify factors affecting stock market
- To make analysis easy for all general people
- To visualize the share market with the help of interactive charts
- To implement machine learning models

1.4 Scope of the Project

The scopes of this project include:

- Stock Market Analysis and Prediction will be able to show live market status
- Classification of the polarity of financial news
- Useful for new investors to invest in stock market

1.5 Limitations

The limitation of the project includes:

- Analysis is based only on the closing value
- Accuracy is only above 90% i.e we can’t acquire 100% accuracy.

CHAPTER 2 SYSTEM ANALYSIS

2.1 Literature Review

Share market prediction has been a subject of extensive research due to its potential impact on financial decision-making and investment strategies. Over the years, various models and techniques have been proposed to forecast stock prices, each with its own set of methodologies and outcomes.

Traditional Methods:
Initially, stock market forecasting was primarily based on statistical models such as Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing techniques. These models focused on identifying trends and seasonality within historical data. While useful in stable market conditions, these methods often fell short in capturing the non-linear and dynamic nature of financial markets.

Machine Learning Approaches:
With the advent of machine learning, researchers began to explore algorithms like Support Vector Machines (SVM), Decision Trees, and Random Forests for stock market prediction. These models provided better accuracy by learning complex patterns in financial datasets. Studies showed that ensemble models like Random Forests often outperformed single classifiers by reducing variance and improving generalization.

Deep Learning Techniques:
In recent years, deep learning techniques, especially Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks, have gained popularity for time-series forecasting. LSTM models, in particular, are capable of capturing long-term dependencies and temporal patterns, making them highly effective for stock price prediction. Research by Fischer & Krauss (2018) demonstrated that LSTM-based models significantly outperform traditional models in terms of predictive accuracy on stock market datasets.

Sentiment Analysis Integration:
Several studies have also integrated sentiment analysis from financial news, social media, and investor sentiment to enhance model performance. The incorporation of textual data helps in capturing market sentiment, which often influences price movements. For instance, Bollen et al. (2011) showed that mood indicators derived from Twitter feeds could improve the prediction accuracy of stock market trends.

Hybrid Models:
Recent trends show a shift toward hybrid models that combine technical indicators, historical data, and sentiment features. These models leverage the strengths of multiple algorithms to improve robustness and predictive power. The literature indicates that hybrid models using both machine learning and deep learning approaches often achieve superior results compared to standalone models.

Research Gap:
Despite significant advancements, challenges such as market volatility, sudden events (e.g., economic crises), and overfitting remain open issues. This project aims to address these gaps by developing a system that integrates both deep learning and sentiment analysis to provide a more comprehensive and accurate stock market prediction tool.

2.2 Requirement Collection and Analysis

The step of requirement collection plays a vital role in the management and developmentof any project. Having a clear idea about what the project is supposed to deliver, at the endof the term, makes project managers and developers of the project aware of steps to betaken for the completion of the job. Here in this project we collect the stock data of thedifferent company from merolagani.com which is used to analyze and predict the currentand future values. Our project mainly focus on forecasting the future value in which theuser(customer) can invest the money. For this project, we took under account two majorrequirement criteria, functional requirements and non-functional requirements

2.2.1 Functional Requirements

The requirement that the system must provide to meet the business need. Based on this, therequirement that system must require:

- Should be able to generate an approximate share price.
- Should collect acceptable and accurate data from Merolagani site.
- Should have an easy interface for the users.

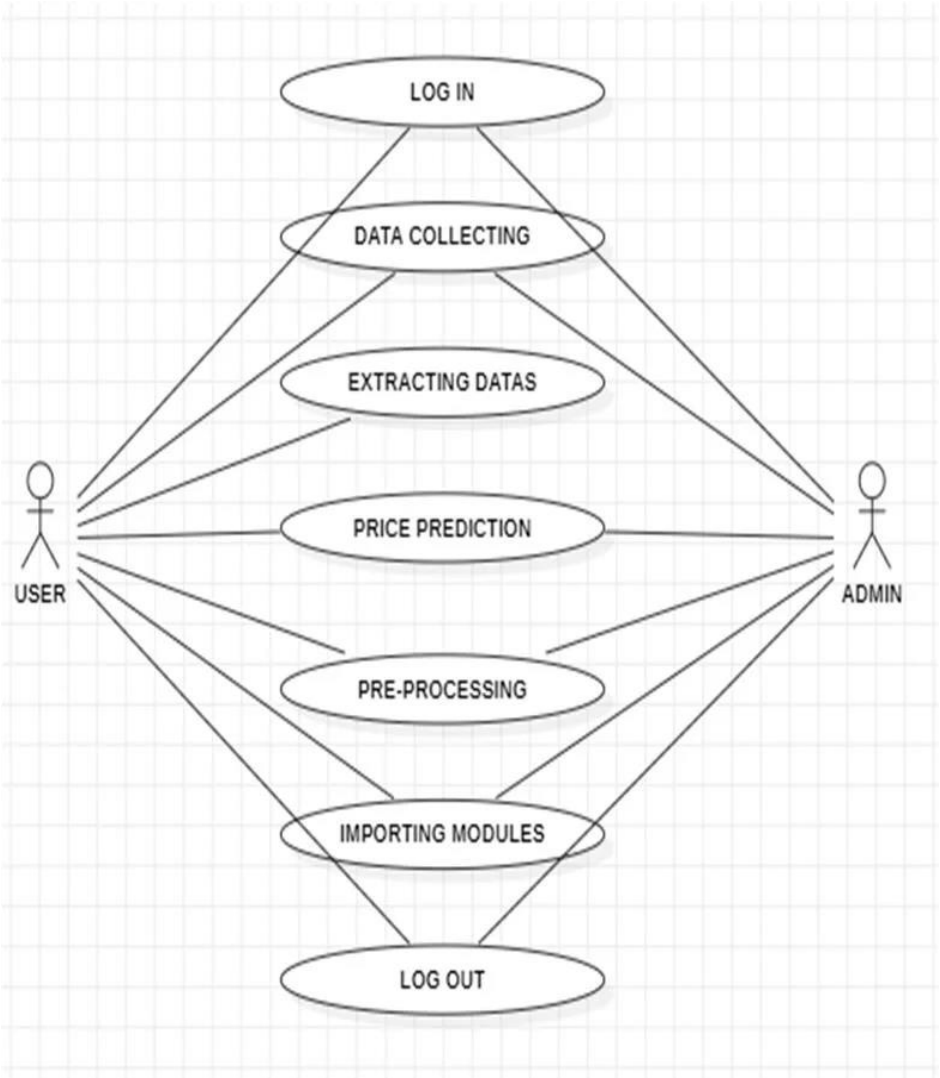
2.2.1.i Use Case Diagram:

Actor 1: User

Description: User must sign up to have full access to system. User are login through their username and password. Users are prohibited to some features if they are not logged in to the system. But the user which don't have the account also have the access to view market information. Authorized user can calculate predictions of different companies, use feedback features and be updated of different stock news.

Actor 2: Admin

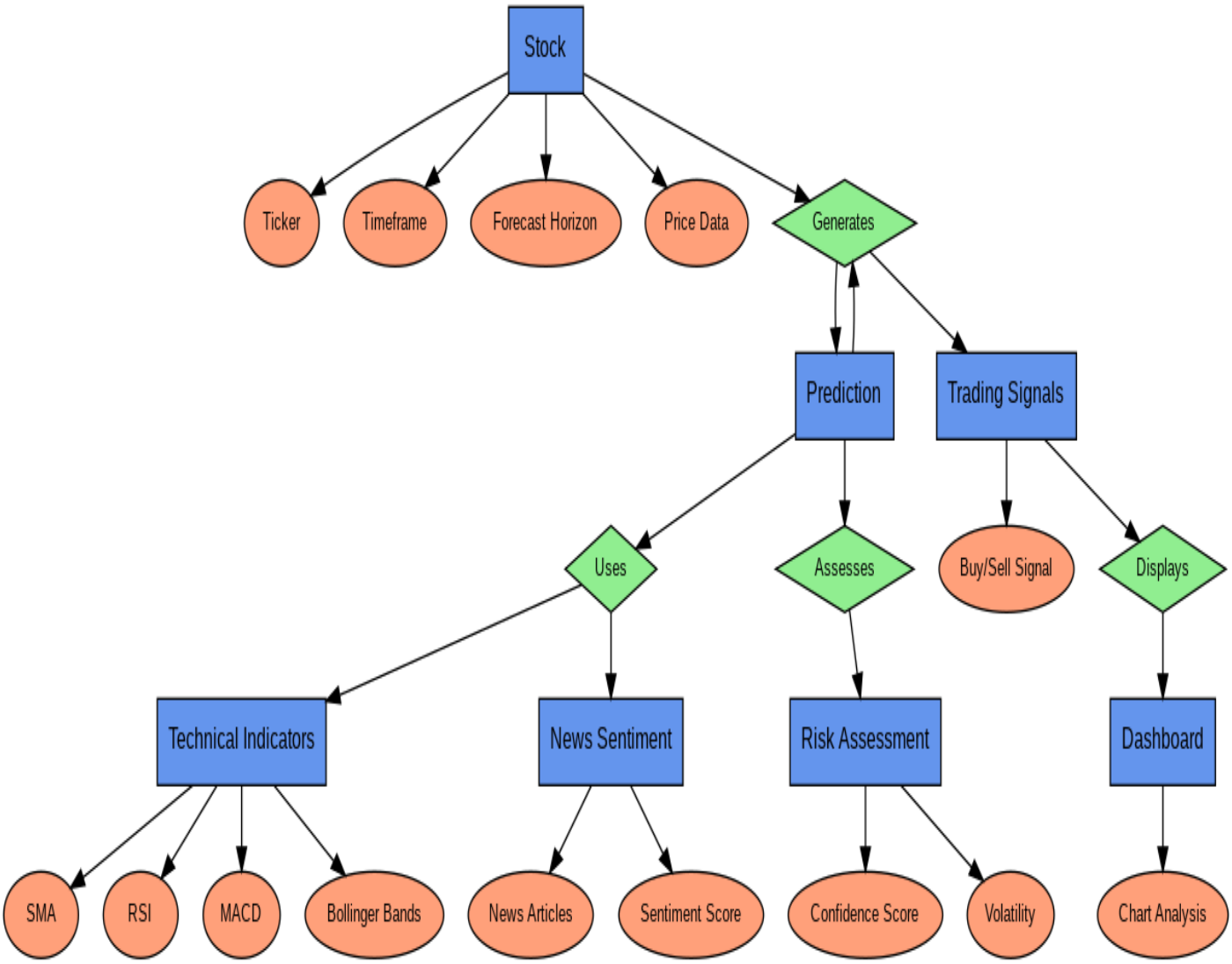
Description: Admin are responsible for verifying user registration and are capable of user management in the system. Market information are updated in the system by the user. All the information about the stock are handled by admin.



2.2.1.1 Data Modeling

2.2.1.1.i E-R Diagram

The E-R diagram shows how the entities are related to each other. These system consistsof mainly four entities i.e. admin, users, prediction and company stock. Admin monitors both the company stock and users. Admin are responsible to generate the prediction valuefor the specified company stock. Admin consists of attributes like id, username and password. Company stock consists of attributes like name, id, close value, symbol and date.Similarly prediction consists of attributes like id, date, predict_value, date and actual_valueand users consists of attributes like username, id and password. E-R diagram clearlyillustrates the relationship between all the entities residing on the system which will provideclear vision of the system.

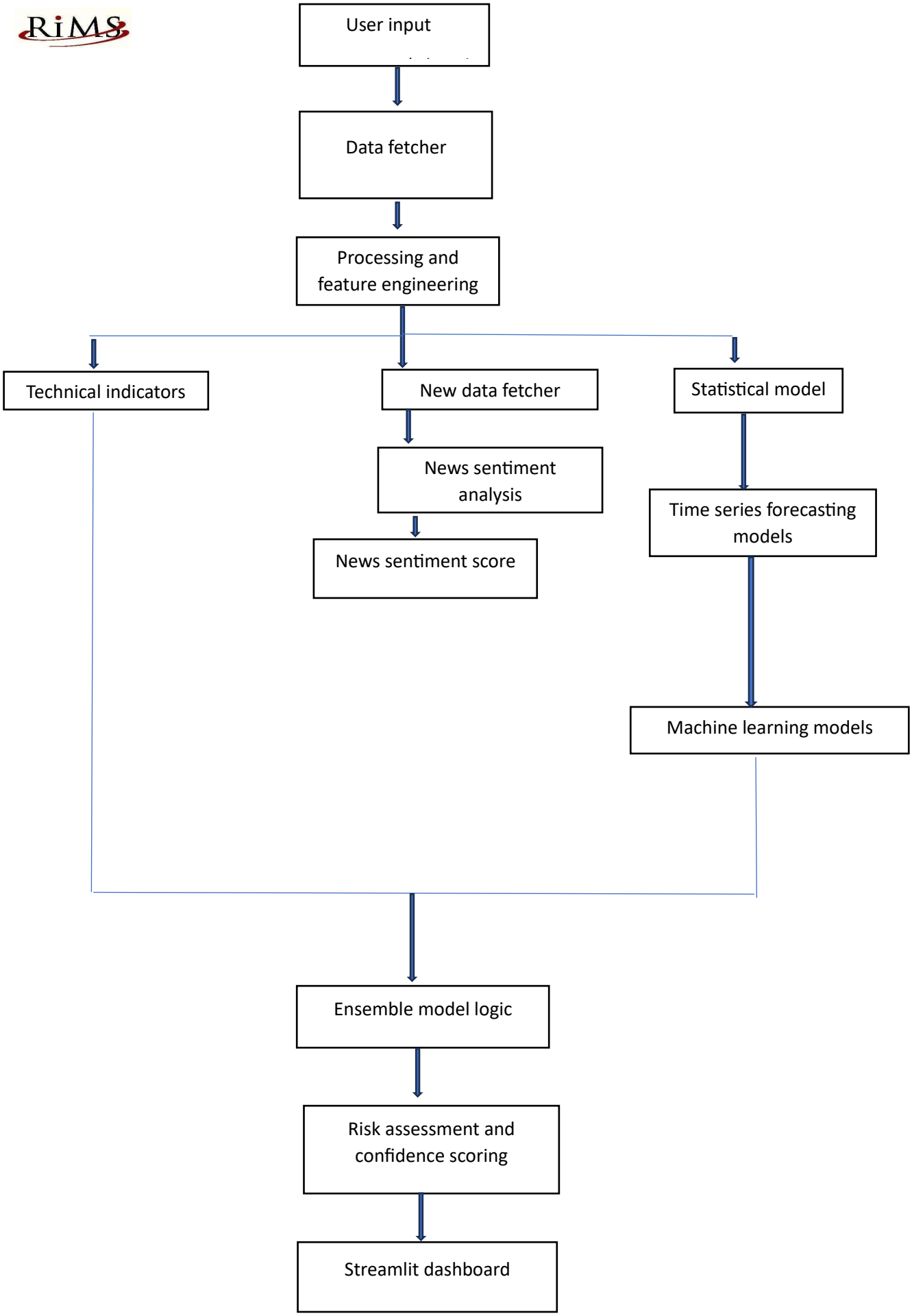

Stock
Ticker
Timeframe
Forecast Horizon
Price Data
Generates
Prediction
Trading Signals
Uses
Assesses
Buy/Sell Signal
Displays
Technical Indicators
News Sentiment
Risk Assessment
Dashboard
SMA
RSI
MACD
Bollinger Bands
News Articles
Sentiment Score
Confidence Score
Volatility
Chart Analysis

2.2.1.2 Process Modeling

Data Flow Diagram(DFD)

A DFD maps out the flow of information for any process or systems. Figure first shows thelevel 0 DFD which simply shows that users interact with the system to get the desired result.Figure second shows the level 1 of DFD which provides a more detailed breakout of piecesof information of level 0 DFD. The flow of data for the system in following diagram is asfollow:1.

- Data Retrieval & Transformation:
- Scraping process is carried to retrieve the data from Merolagani as csv files.
- Predictive Analysis
- Formatted data in Excel are used for predictive analysis.
- Predictive Model Generation Algorithm
- ARIMA algorithm is used to generate a model to predict the value.
- Charts Generation
- Predicted Trend are illustrated in chat for better understanding andrepresentation.
- Training of Data and Prediction
- Using test data and algorithm data are trained and are made capable to predict the stock price.
- Data Validation and Results Generation
- The results are tested for error i.e. validation process is carried out andafterwards result are generated.


User input
Data fetcher
Processing and feature engineering
Technical indicators
New data fetcher
Statistical model
News sentiment analysis
Time series forecasting models
News sentiment score
Machine learning models
Ensemble model logic
Risk assessment and confidence scoring
Streamlit dashboard

2.2.2 User Requirements

User Expectations

- The system should provide **accurate and interpretable** predictions.
- The app should allow **customization** of forecast horizons.
- Users should get clear **buy/sell/hold** trading signals.

2.2.3 System Requirements

1. Hardware Requirements

- **Processor:** Intel Core i5 (8th Gen or higher) / AMD Ryzen 5 or better
- **RAM:** Minimum **8GB** (Recommended **16GB** for better performance)
- **Storage:** At least **10GB free space** (SSD recommended for faster computations)
- **GPU (Optional):** NVIDIA GPU with CUDA support (for deep learning models like LSTM)

2. Software Requirements

- **Operating System:** Windows 10/11, macOS, or Linux
- **Python Version:** Python **3.8+**
- **IDE:** VS Code / PyCharm / Jupyter Notebook

3. Required Python Libraries

- Install using: pip install -r requirements.txt
  Data Handling: pandas, numpy
  Machine Learning: scikit-learn, xgboost, tensorflow, statsmodels, prophet, arch
  Technical Analysis: ta
  Data Fetching: yfinance, newsapi-python
  Sentiment Analysis: nltk, textblob
  Visualization: matplotlib, plotly
  Web Framework: streamlit

4. Execution Command

Run the application using:

streamlit run stock_predictor.py

2.2.4 Data Requirements

Company stock data scraped will contain the date, closing value.The data scraped is storedin csv file format and then transported to the database for training the prediction model.Similarly, the data is also stored in MySQL database to display in the system. Prior to theapplication, the database shall be updated to the latest values in market and news. The chartsand comparisons of the companies will be made only on the basis of latest data.

The predicted indication of rise or fall of market data will be stored in the database beforedisplay.

2.2.5 Non-Functional Requirements

Reliability: The reliability of the product will be dependent on the accuracy of the datadate of purchase, how much stock was purchased, high and low value range as well as opening and closing figures. Also, the stock data used in the training would determine thereliability of the software.

Security:

The user will only be able to access the website for inserting the stock pricesusing his login details and will not be able to access the computations happening at the backend.

Maintainability:

The maintenance of the product would require training of the software by recent data so that the recommendations are up to date. The database has to be updated with recent values.

Portability:

The website is completely portable and the recommendations completely trustworthy as the data is dynamically updated.

Interoperability:

The interoperability of the website is very high because it synchronizes all the database with the server.

2.2.6 Software Requirement

Software Requirements for Multi-Algorithm Stock Predictor

1. Operating System:

- Windows 10/11, macOS, or Linux

2. Programming Language:

- Python 3.8+

3. Libraries & Dependencies:

Install via: pip install -r requirements.txt

- **Data Handling:** pandas, numpy
- **Machine Learning:** scikit-learn, xgboost, tensorflow, statsmodels, prophet, arch
- **Technical Analysis:** ta

- **Data Fetching:** yfinance, newsapi-python
- **Sentiment Analysis:** nltk, textblob
- **Visualization:** matplotlib, plotly
- **Web Framework:** streamlit

4. Development Tools:

- Python IDE (e.g., PyCharm, VS Code, Jupyter Notebook)
- Git for version control

5. Execution Command:

Run with: streamlit run stock_predictor.py

2.3 Feasibility Study

Feasibility study is the study of how successful the project can be, accounting for factors like, economical, technological, legal and scheduling. Project managers make use of feasibility study to determine the positive or negative outcomes of a project before making any investments into it. The various feasibility analysis is included below.

2.3.1 Technical Feasibility:

The project is technically feasible as it leverages widely used Python libraries such as pandas, scikit-learn, tensorflow, yfinance, and streamlit for implementation. The system requires a moderate computational power for training and real-time predictions, which can be handled on a standard laptop or cloud-based resources. The integration of APIs for live stock data and sentiment analysis is straightforward, making deployment seamless.

2.3.2 Operational Feasibility

The **interactive Streamlit interface** ensures user-friendliness, allowing traders with minimal technical knowledge to utilize stock predictions effectively. The system provides **real-time insights**, model consensus analysis, and risk assessment, making it a valuable tool for decision-making. Users can easily interpret trading signals, trend patterns, and confidence scores.

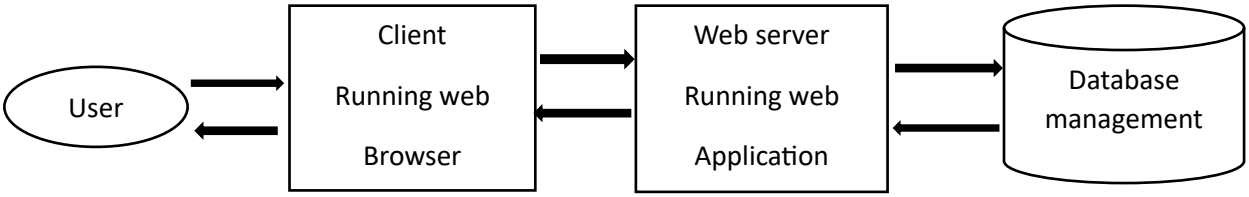
2.3.3 Schedule Feasibility

Schedule feasibility assesses whether the project can be completed within a reasonable timeframe based on available resources and complexity. Below is a structured timeline for the development of the **share market prediction.**

CHAPTER 3 SYSTEM DESIGN

3.1 System Design

System design is simply the overall design of the system. The readily set system design parameters are especially useful for the micro process of system development, convertingthe product from blueprint to actual application. This document contains the overall designof the system. The system will be constructed in 3-Tier Architecture as:

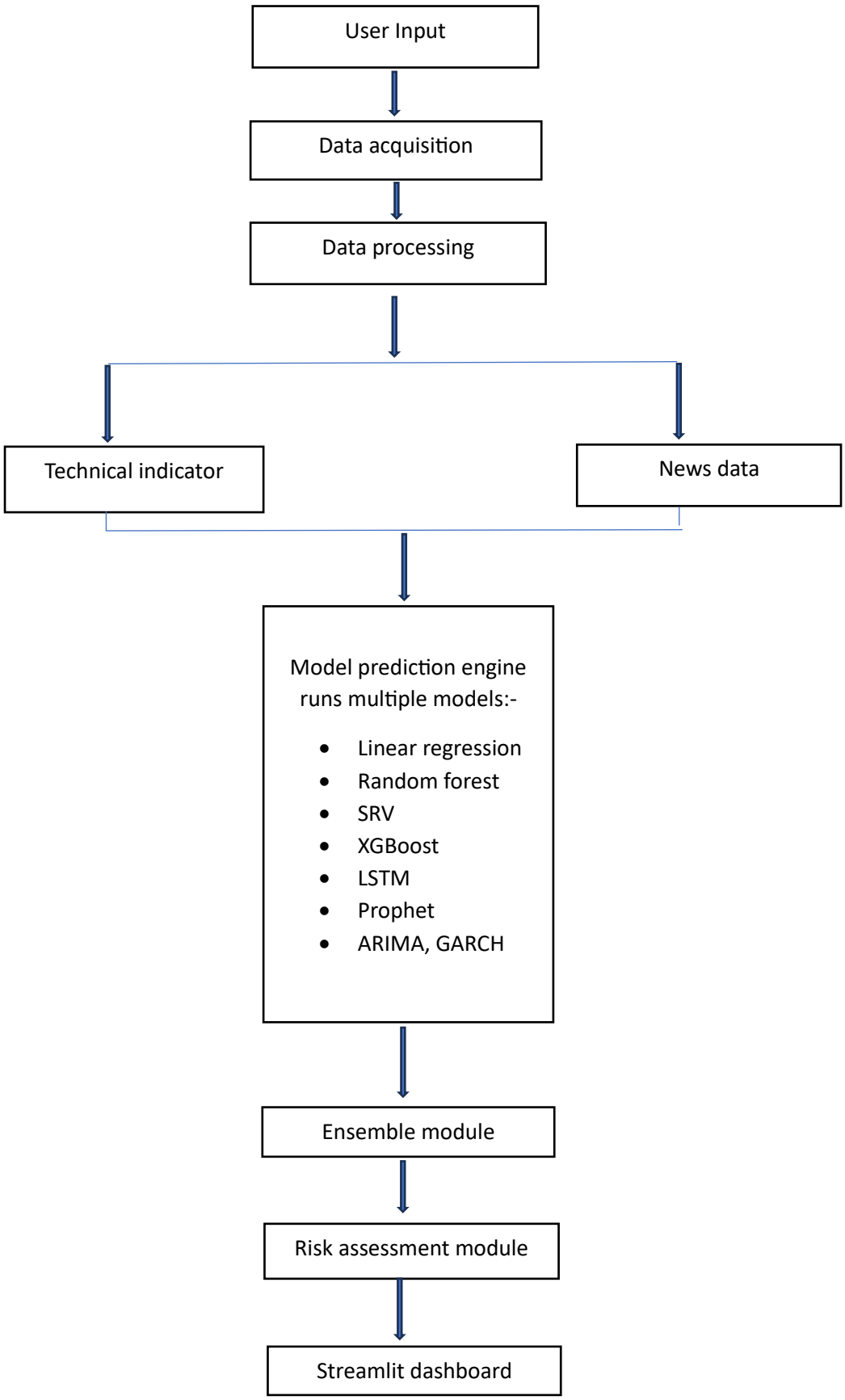


3.1.1 User Interface

An interactive and easy to use user interface is the goal of the system. The design doesn't contain any ambiguous spaces and is self-explanatory

3.1.2 System Flow Diagram

System flow chart simply describes a working method of system in which user choose acompany which value is to be predicted. Then ARIMA algorithm runs which simplygenerates a result which are shown properly in the charts.ARIMA algorithm flow chart is also described above. First we choose our data set whichwill be in csv format. Then data set are checked if they are stationary or not. If it is notstationary we will be using differencing method to make it stationary. If it is stationarywe will use ACF & PACF to find the p, d, q parameters for the model. We will fit the parameters to our model and train our model. Predicted value is obtained which is usedto evaluate the accuracy of the model using MAPE. Flow chart simply shows the workingmethod of algorithm and the system.

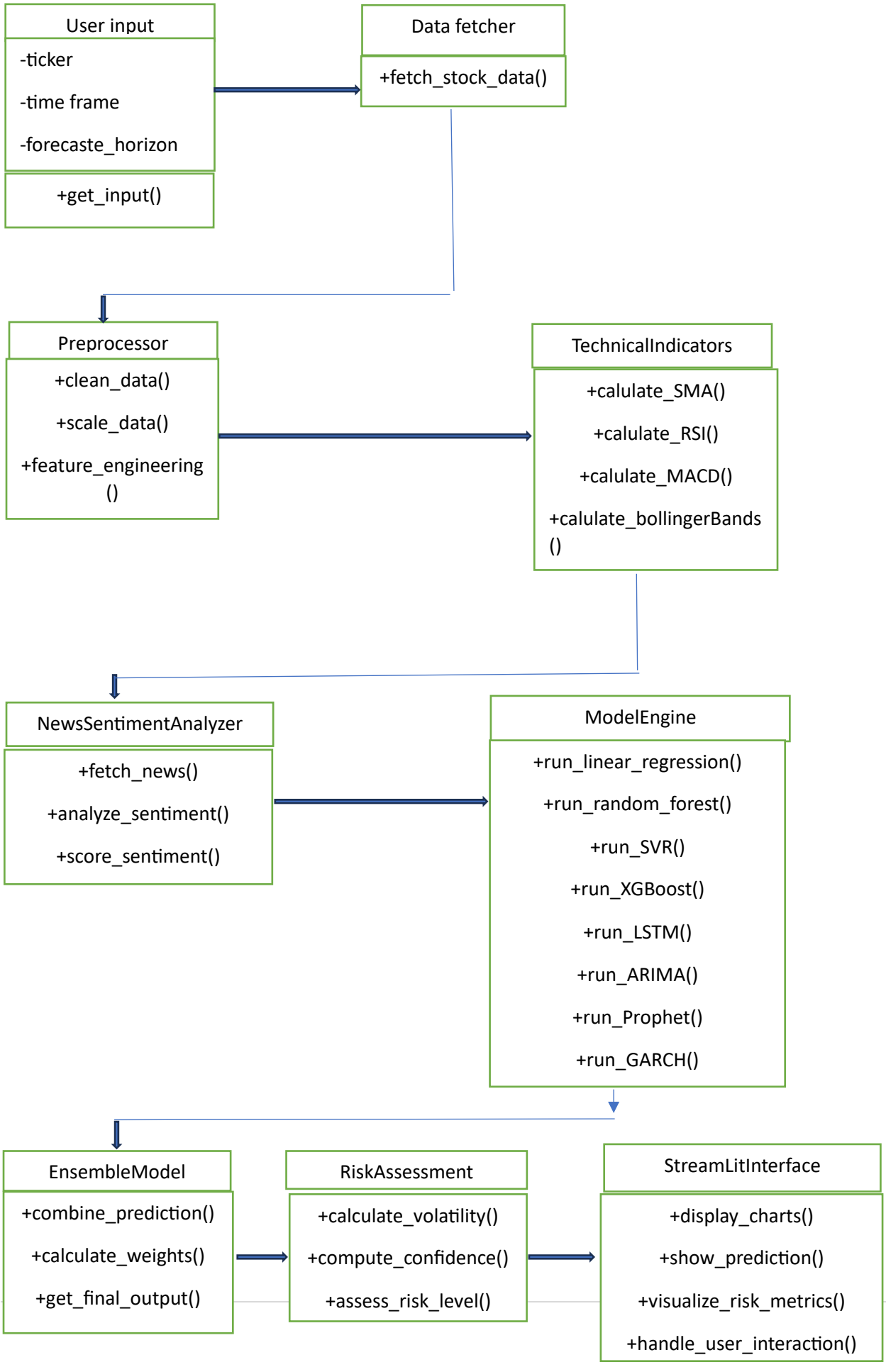

User Input
Data acquisition
Data processing
Technical indicator
News data
Model prediction engine runs multiple models:-
Linear regression
Random forest
SRV
XGBoost
LSTM
Prophet
ARIMA, GARCH
Ensemble module
Risk assessment module
Streamlit dashboard

3.1.3 Class Diagram

Classes in class diagrams are represented by boxes that are partitioned into three:

- The top partition contains the name of the class.
- The middle part contains the class's attributes.
- The bottom partition shows the possible operations that are associated with the class. In this diagram user's class has attributes like id, username, password, first_name, last_name and email. Many user can be added using addUser operation. One user canaccess many stock prediction prices. Stock class has attribute like id, obs_data and date.

Different operations like adding stock, deleting stock and viewing stock can be performed.Certain company differs in its stock prices. Company has attributes like id, company_name,email and symbol. Different operations like adding company and extracting companyinformation operations are carried out. One company can have multiple company datawhere company data can have attributes like id, close, obs_data and date. Users can viewdata and date of company. One company can have several news where news can haveattributes like id, title, image, detail, date and author where operation like viewing newscan be performed.


User input
-ticker
-time frame
-forecaste_horizon
+get_input()
Data fetcher
+fetch_stock_data()
Preprocessor
+clean_data()
+scale_data()
+feature_engineering ()
TechnicalIndicators
+calulate_SMA()
+calulate_RSI()
+calulate_MACD()
+calulate_bollingerBands ()
NewsSentimentAnalyzer
+fetch_news()
+analyze_sentiment()
+score_sentiment()
ModelEngine
+run_linear_regression()
+run_random_forest()
+run_SVR()
+run_XGBoost()
+run_LSTM()
+run_ARIMA()
+run_Prophet()
+run_GARCH()
EnsembleModel
+combine_prediction()
+calculate_weights()
+get_final_output()
RiskAssessment
+calculate_volatility()
+compute_confidence()
+assess_risk_level()
StreamLitInterface
+display_charts()
+show_prediction()
+visualize_risk_metrics()
+handle_user_interaction()

3.1.4 Sequence Diagram

A sequence diagram in a stock market project visualizes the interactions between different actors and system components over time. It's particularly useful for understanding the flow of operations like placing an order, checking stock prices, or managing user accounts. Here's a breakdown of the key elements and a simplified example:

Key Elements of a Sequence Diagram:

- **Actors:**
    - These are external entities that interact with the system (e.g., "Trader," "System Administrator").
- **Lifelines:**
    - Vertical lines representing the timeline of each actor or object.
- **Messages:**
    - Arrows that represent interactions between actors and objects. These can include:
        - **Synchronous messages:** Represent function calls where the sender waits for a response.
        - **Asynchronous messages:** Represent messages where the sender doesn't wait for a response.
        - **Return messages:** Represent the response to a synchronous message.
- **Activation Boxes:**
    - Rectangles on lifelines that indicate when an object is active (processing a message).

Simplified Example: Placing a Stock Order:

Here's a simplified sequence diagram for a trader placing a buy order:

1. **Actors/Objects:**
    - Trader
    - Trading Platform (Web/App)
    - Order Management System
    - Stock Exchange
2. **Sequence of Interactions:**

- **Trader:**
    - Sends a "Place Buy Order" message to the "Trading Platform."
- **Trading Platform:**
    - Receives the "Place Buy Order" message.
    - Validates the order details.
    - Sends an "Order Request" message to the "Order Management System."

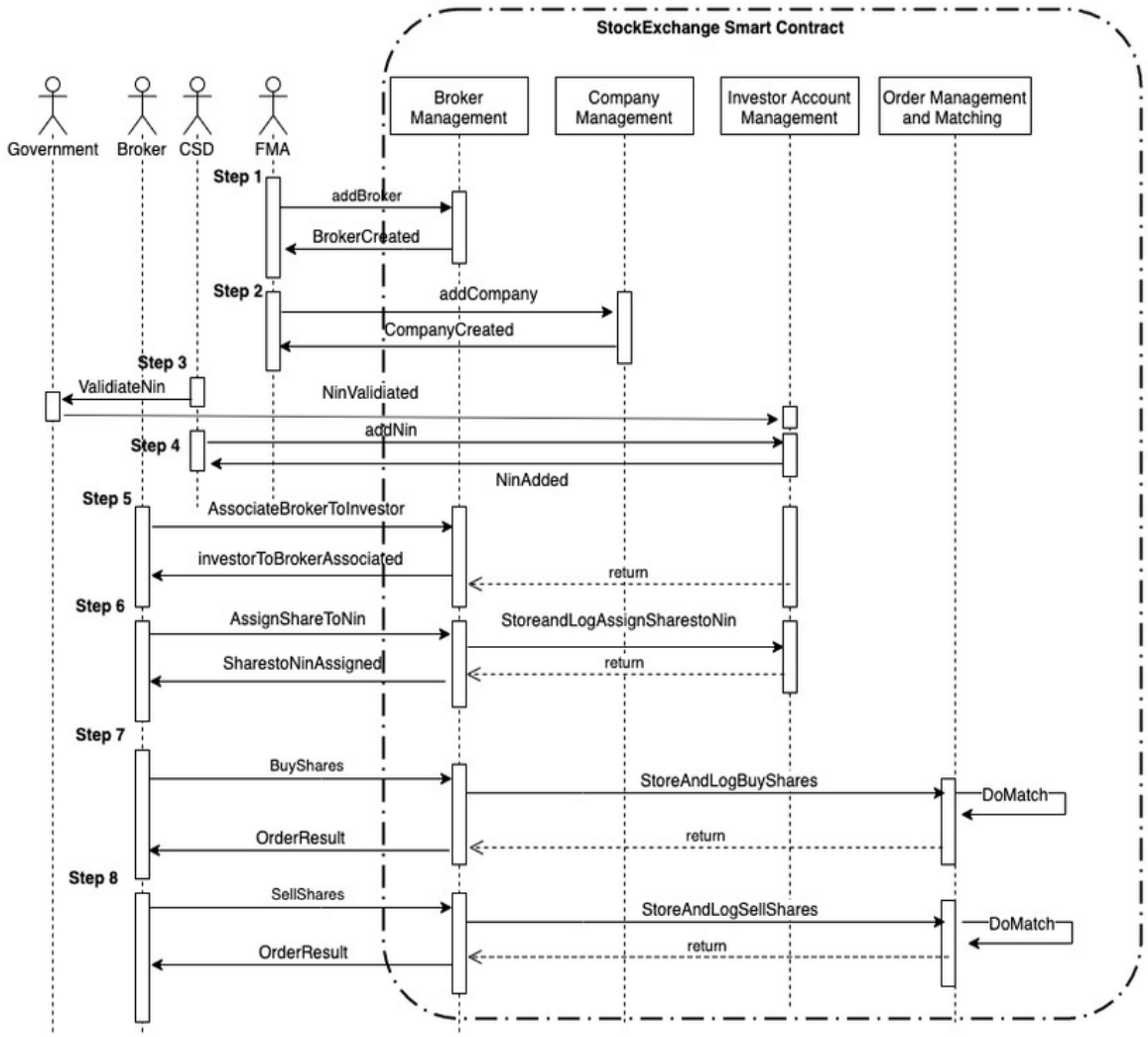
- **Order Management System:**
    - Receives the "Order Request" message.
    - Checks user funds and stock availability.
    - Sends an "Order Submission" message to the "Stock Exchange."
- **Stock Exchange:**
    - Recieves the "Order Submission" message.
    - Attempts to match the order.
    - Sends an "Order Confirmation/Failure" message back to the "Order Management System."
- **Order Management System:**
    - Recieves the "Order Confirmation/Failure" message.
    - Sends an "Order Status" message to the "Trading Platform."
- **Trading Platform:**
    - Recieves the "Order Status" message.
    - Displays the order status to the "Trader."

Key Considerations for a Stock Market Project:

- **Real-time Data:**
    - Sequence diagrams can illustrate how the system retrieves and updates real-time stock data.
- **Security:**
    - They can depict authentication and authorization processes.
- **Transaction Processing:**
    - They are very helpful for showing the flow of transactions like buying, selling, and canceling orders.
- **Error Handling:**
    - They can show how the system handles errors (e.g., insufficient funds, invalid stock symbols).
- **API Interactions:**
    - If the system uses external APIs for market data, or for transaction processing, the sequence diagram will show those API interactions.

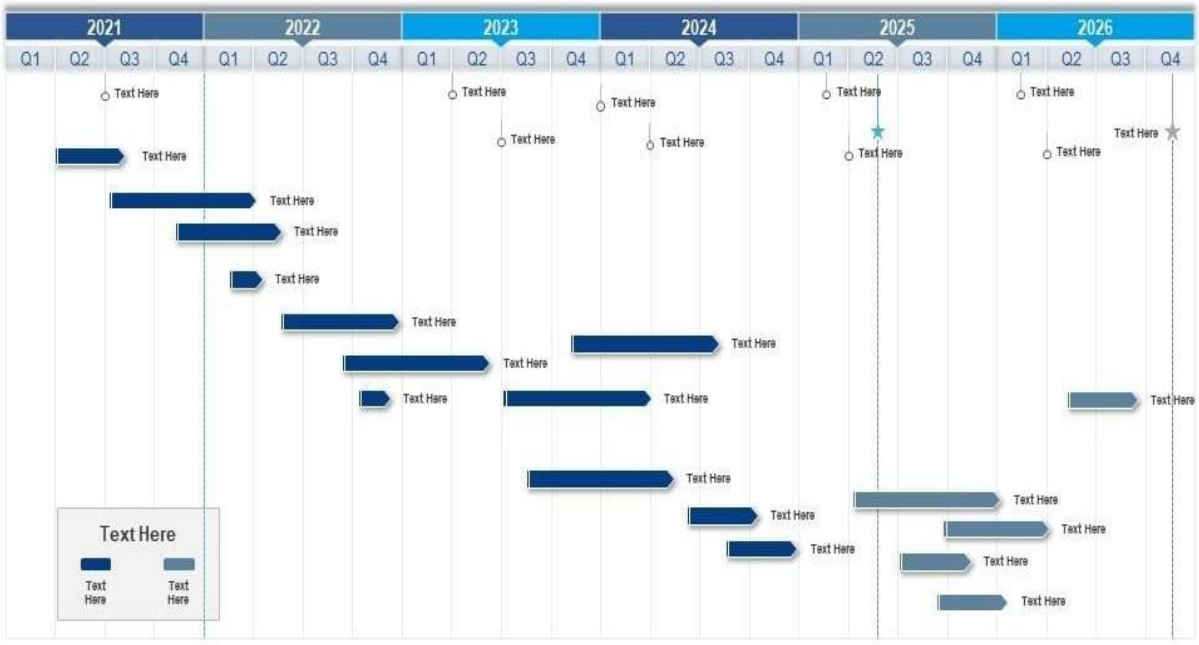
Sequence diagrams are valuable tools for:

- Clarifying system requirements.
- Designing robust system architectures.
- Communicating complex interactions between developers and stakeholders.


StockExchange Smart Contract
Government
Broker
CSD
FMA
Broker Management
Company Management
Investor Account Management
Order Management and Matching
Step 1
addBroker
BrokerCreated
Step 2
addCompany
CompanyCreated
Step 3
ValidiateNin
NinValidiated
Step 4
addNin
NinAdded
Step 5
AssociateBrokerToInvestor
investorToBrokerAssociated
return
Step 6
AssignShareToNin
StoreandLogAssignSharestoNin
SharestoNinAssigned
return
Step 7
BuyShares
StoreAndLogBuyShares
DoMatch
OrderResult
return
Step 8
SellShares
StoreAndLogSellShares
DoMatch
OrderResult
return

3.1.5 Gantt Chart

A Gantt chart in a stock market project serves as a visual timeline, breaking down complex tasks like software development for trading platforms or financial analysis into manageable segments with defined start and end dates. This tool facilitates project planning, task management, and resource allocation, enabling clear visualization of dependencies and timelines. By tracking progress against the planned schedule, Gantt charts aid in identifying potential delays and ensuring timely completion of project milestones, which is crucial for the efficient development and management of stock market-related systems and analyses.



CHAPTER 4 IMPLEMENTATION

4.1 Implementation

The main purpose of implementation of this system is to predict the stock prices based onthe previous stock prices

4.1.1 Algorithm Design

Algorithms are the operational infrastructure of every project; the algorithms determinehow and how the program operated and generated results based on the calculations. Aneffective algorithm must encompass all the data variables available for computation and inreturn generate an efficient flow as well as true results of the processing afterwards. . Whenit comes to predictive analysis there is a myriad of choices over the internet that operate instatistical data to generate associative output. Choosing between these numerousalgorithms itself needs a good amount of study upon the topics and also a deep analysis ofthe predictions being made from the system.

Since, in this case there are multiple number ofdependent variables that are key points on prediction, we have adopted the algorithm ofARIMA .

Data Collection

In the first phase, a number of scraping scripts to collect data from the sources mentioned previously in the project. The data is composed of market data of companies

4.1.1.1 Linear regression

Linear regression is a statistical method that attempts to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. In the context of stock market prediction, it's used to try and forecast future stock prices based on historical data. Here's a breakdown of how it's applied:

Understanding the Basics:

- **Dependent Variable:** In stock prediction, this is typically the stock price itself.
- **Independent Variables:** These are the factors that are believed to influence the stock price. Examples include:
    - Historical stock prices (e.g., previous day's closing price).
    - Trading volume.
    - Economic indicators (e.g., interest rates, inflation).
    - Company earnings reports.
    - Market sentiment.

- **Linear Equation:** The goal is to find a linear equation (y = mx + b, or a more complex version with multiple independent variables) that best represents the relationship between these variables.

How Linear Regression is Used in Stock Prediction:

1. **Data Collection:**
    - Gather historical stock price data and data for the chosen independent variables.
    - This data needs to be clean and reliable.
2. **Data Preprocessing:**
    - Clean the data by handling missing values and outliers.
    - Normalize or scale the data to ensure that all variables contribute equally to the model.
    - Split the data into training and testing sets. The training set is used to build the model, and the testing set is used to evaluate its performance.
3. **Model Building:**
    - Use a linear regression algorithm to find the best-fit line or hyperplane that represents the relationship between the independent and dependent variables.
    - This involves calculating the coefficients of the linear equation.
4. **Model Evaluation:**
    - Evaluate the model's performance using metrics such as:
        - Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual values.
        - R-squared: Measures how well the model fits the data.
    - A lower MSE and a higher R-squared indicate a better model.
5. **Making Predictions:**
    - Once the model is trained and evaluated, it can be used to make predictions about future stock prices.
    - Input the values of the independent variables into the linear equation to get a predicted stock price.

Limitations:

- **Stock market volatility:** The stock market is highly volatile and influenced by many unpredictable factors. Linear regression assumes a linear relationship, which may not always hold true.
- **Non-linear relationships:** Many factors influencing stock prices have non-linear relationships. Linear regression is not well equiped to deal with those types of relationships.
- **Overfitting:** If the model is too complex, it may overfit the training data and perform poorly on new data.
- **Economic events:** Unforseen economic events, or news, can cause drastic changes in stock prices that linear regression will not be able to predict.

6. **Formula**

1.Simple Linear Regression:

- **Formula:**
  - y=β0+β1x+ε
  - Where:
    - y is the dependent variable (e.g., predicted stock price).
    - β0 is the y-intercept (the value of y when x is 0).
    - β1 is the slope of the line (the change in y for a one-unit change in x).
    - x is the independent variable (e.g., previous day's closing price).
    - ε is the error term.

2. Multiple Linear Regression:

- **Formula:**
  - y=β0+β1x1+β2x2</4>+...+β<0xE2><0x82><0x99>x<0xE2><0x82><0x99>+ε
  - Where:
    - y is the dependent variable.
    - β0 is the y-intercept.
    - β1,β2,...,β<0xE2><0x82><0x99> are the coefficients for the independent variables.
    - x1,x2,...,x<0xE2><0x82><0x99> are the independent variables (e.g., trading volume, economic indicators).
    - ε is the error term.

7. **Example:-**

Predict next 10 days' volatility of Tesla stock returns to adjust your trading strategy (e.g., position sizing).
import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

Sample Data (Close price & 20-day SMA)

df = pd.read_csv('AAPL.csv')

df['SMA_20'] = df['Close'].rolling(20).mean()

X = df[['SMA_20']].dropna()

y = df['Close'][len(df) - len(X):]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

model = LinearRegression()

model.fit(X_train, y_train)

predictions = model.p

redict(X_test)

4.1.1.2 Random Forest

The Random Forest algorithm is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction

(regression) of the individual trees.[2] While it's not a single, simple formula, understanding its process involves grasping key concepts:

Key Concepts:

- **Bootstrap Aggregating (Bagging):**
    - Random Forest uses bagging to create multiple subsets of the training data.
    - Each subset is created by sampling with replacement, meaning some data points may appear multiple times in a subset, while others are left out.
- **Random Feature Selection:**
    - At each node of a decision tree, instead of considering all features for the best split, Random Forest randomly selects a subset of features.
    - This adds further randomness and decorrelates the trees.
- **Decision Trees:**
    - The base learners in a Random Forest are decision trees.
    - These trees make predictions by recursively partitioning the data based on feature values.
- **Ensemble Prediction:**
    - For classification, the final prediction is determined by a majority vote of the individual tree predictions.
    - For regression, the final prediction is the average of the individual tree predictions.

Formulaic Representation (Conceptual):

It's difficult to provide a single, concise formula for Random Forest. However, we can express the core idea with these conceptual representations:

- **Regression:**
    - f^RF(x)=B1∑b=1BTb(x)
        - Where:
            - f^RF(x) is the Random Forest prediction for input x.
            - B is the number of trees in the forest.
            - Tb(x) is the prediction of the b-th tree for input x.

- **Classification:**
    - C^RF(x)=majority vote{C^b(x)}b=1B
        - Where:
            - C^RF(x) is the Random Forest classification for input x.
            - B is the number of trees in the forest.
            - C^b(x) is the class prediction of the b-th tree for input x.

Example:-

```
from sklearn.ensemble import RandomForestRegressor

df['RSI'] = ... # Calculate RSI here

df['MACD'] = ... # Calculate MACD here

features = df[['SMA_20', 'RSI', 'MACD']].dropna()

target = df['Close'][len(df) - len(features):]

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, shuffle=False)

rf = RandomForestRegressor(n_estimators=100)

rf.fit(X_train, y_train)

rf_preds = rf.predict(X_test)
```

4.1.1.3 SVR

Support Vector Regression (SVR) is a powerful regression technique derived from Support Vector Machines (SVM). Instead of classifying data into categories, SVR aims to find a function that best approximates the relationship between input and output variables. Here's a breakdown of its core concepts and a simplified view of its underlying principles:

Core Idea:

- SVR tries to fit a line (or hyperplane in higher dimensions) to the data, with the goal of having as many data points as possible lie within a margin of error (epsilon) around that line.

- This "epsilon-tube" defines the acceptable error range. The algorithm focuses on minimizing the error outside this tube.
- Kernel functions are often used to handle non-linear relationships by mapping the data into higher-dimensional spaces.

Key Concepts and Simplified Formulaic Representation:

1. Epsilon-Insensitive Loss Function:
    - This is a crucial element of SVR. It defines the "cost" of errors.
    - The idea is that errors within a certain range (epsilon, ε) are ignored.
    - Essentially, if the predicted value is within ε of the actual value, there's no loss.
    - Errors outside this range are penalized linearly.
2. Optimization Goal:
    - SVR aims to find a function f(x) that minimizes:
        - The flatness of the function (by minimizing the norm of the weight vector).
        - The errors outside the epsilon-tube.
    - This can be expressed as an optimization problem involving:
        - Weight vector (w).
        - Bias (b).
        - Slack variables (ξ, ξ*).
        - Regularization parameter (C).
3. Kernel Functions:
    - To handle non-linear data, SVR uses kernel functions.
    - These functions implicitly map the data into higher-dimensional spaces where linear regression becomes possible.
    - Common kernel functions include:
        - Linear kernel.
        - Polynomial kernel.

- Radial basis function (RBF) kernel.

- The use of kernels is what allows SVR to model complex non-linear relationships.

4. Simplified Formula:

   - While the full optimization problem is complex, the resulting prediction function can be represented conceptually as:

     - $f(x) = \Sigma (\alpha i - \alpha i^*) K(xi, x) + b$

     - Where:

       - f(x) is the predicted value.

       - αi and αi* are Lagrange multipliers.

       - K(xi, x) is the kernel function.

       - b is the bias.

   - This formula shows that the prediction is a weighted sum of kernel evaluations between the training data (xi) and the new input (x).

5. **Example:-**

```
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
svr = SVR(kernel='rbf', C=100)
svr.fit(X_scaled[:int(len(X_scaled)*0.8)], y[:int(len(y)*0.8)])
svr_preds = svr.predict(X_scaled[int(len(X_scaled)
*0.8):])
```

4.1.1.4 XGBoost

XGBoost (eXtreme Gradient Boosting) is a highly efficient and popular machine learning algorithm, particularly known for its performance in structured/tabular data. It's an optimized implementation of gradient boosting, with key enhancements that contribute to its speed and accuracy.

Here's a breakdown of the XGBoost algorithm and its core mathematical concepts:

Key Concepts:

- Gradient Boosting:
    - XGBoost, like other gradient boosting algorithms, builds an ensemble of decision trees sequentially.
    - Each new tree is trained to correct the errors made by the previous trees.
    - It minimizes a loss function by iteratively adding trees that predict the negative gradient of the loss.
- Regularization:
    - XGBoost incorporates L1 and L2 regularization to prevent overfitting, which is a significant advantage.
    - This helps to simplify the model and improve its generalization performance.
- Optimization:
    - XGBoost employs second-order gradients (Hessian) in its optimization process, which allows for faster convergence.
    - It also includes techniques for handling sparse data and parallel processing, contributing to its efficiency.

Formulaic Representation (Simplified):

1. **Objective Function:**
    - The core of XGBoost lies in its objective function, which combines a loss function and a regularization term:
        - $obj(\theta) = L(\theta) + \Omega(\theta)$
            - $L(\theta)$: Loss function (e.g., mean squared error, logistic loss).
            - $\Omega(\theta)$: Regularization term.
    - The goal is to minimize this objective function.
2. **Prediction:**
    - The prediction of XGBoost is the sum of the predictions from all the individual trees:
        - $\hat{y}_i = \Sigma$ f<0xE2><0x82><0x99>(x_i)
            - $\hat{y}_i$: Predicted value for data point i.
            - f<0xE2><0x82><0x99>(x_i): Prediction of the k-th tree for data point i.

3. **Regularization Term:**
    - The regularization term penalizes complex trees:
        - $\Omega(f<0xE2><0x82><0x99>) = \gamma T + (1/2)\lambda \Sigma w_j^2$
            - T: Number of leaves in the tree.
            - γ: Regularization parameter for the number of leaves.
            - λ: Regularization parameter for leaf weights.
            - w_j: Weight of the j-th leaf.
4. **Gain Function:**
    - The gain function is used to evaluate the quality of splits in the trees.
    - It involves the first and second order gradients of the loss function.
    - The gain functions are used to determine the best splits within each tree.

Key Parameters:

- Learning rate (eta): Controls the step size at each boosting iteration.
- max_depth: Maximum depth of the trees.
- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.
- lambda: L2 regularization term on weights.
- alpha: L1 regularization term on weights.

5. **Example:-**

```
import xgboost as xgb
X_train, X_test, y_train, y_test = train_test_split(features, target,
      test_size=0.2,shuffle=False)
xgb_model = xgb.XGBRegressor(n_estimators=100, max_depth=5)
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
```

4.1.1.5 LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to address the vanishing gradient problem, [1] enabling them to learn long-term dependencies in sequential data. While a single, compact formula doesn't fully capture its complexity, we can break down the core equations that govern its operation:

LSTM Core Components:

- **Cell State (Ct):**
    - This is the "memory" of the LSTM, carrying information across time steps.
- **Hidden State (ht):**
    - This is the output of the LSTM at each time step.
- **Gates:**
    - These control the flow of information into and out of the cell state.
        - **Forget Gate (ft):** Determines what information to discard from the cell state.
        - **Input Gate (it):** Determines what new information to add to the cell state.
        - **Output Gate (ot):** Determines what information to output as the hidden state.

LSTM Equations:

1. **Forget Gate:**
    - $f_t=\sigma(W_f \cdot [h_{t-1},x_t]+b_f)$
        - f_t: Forget gate activation.
        - σ: Sigmoid function (outputs values between 0 and 1).
        - W_f: Weight matrix for the forget gate.
        - $[h_{t-1},x_t]$: Concatenation of the previous hidden state and the current input.
        - b_f: Bias vector for the forget gate.
2. **Input Gate:**
    - $i_t=\sigma(W_i \cdot [h_{t-1},x_t]+b_i)$
        - i_t: Input gate activation.

- Wi: Weight matrix for the input gate.
- bi: Bias vector for the input gate.

- $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$
  - $\tilde{C}_t$: Candidate cell state.
  - tanh: Hyperbolic tangent function (outputs values between -1 and 1).
  - Wc: Weight matrix for the cell state.
  - bc: Bias vector for the cell state.

3. **Cell State Update:**
   - $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$
     - Ct: Updated cell state.
4. **Output Gate:**
   - $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
     - ot: Output gate activation.
     - Wo: Weight matrix for the output gate.
     - bo: Bias vector for the output gate.
5. **Hidden State Update:**
   - $h_t = o_t \cdot \tanh(C_t)$
     - ht: Updated hidden state.

Explanation:

- The forget gate decides which information from the previous cell state (C_{t-1}) to keep.
- The input gate decides which new information from the current input (x_t) to add to the cell state.
- The cell state is updated by combining the information from the forget and input gates.
- The output gate selects which information from the cell state to output as the hidden state (h_t).

Examole:-

```
import numpy as np
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

df['returns'] = df['Close'].pct_change().dropna()

data = df['returns'].values.reshape(-1,1)

sequence_length = 60

X_lstm, y_lstm = [], []

for i in range(len(data) - sequence_length):

X_lstm.append(data[i:i+sequence_length])

y_lstm.append(data[i+sequence_length])

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

model = Sequential()

model.add(LSTM(50, return_sequences=False, input_shape=(X_lstm.shape[1], 1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X_lstm, y_lstm, epochs=10, batch_size=32)
```

4.1.1.6 ARIMA

ARIMA (AutoRegressive Integrated Moving Average) is a statistical time series forecasting method that combines autoregression, differencing, and moving averages to predict future values. Here's a breakdown of its components and the general formula:

ARIMA Components:

- AR (Autoregressive): Uses past values of the time series to predict future values.
- I (Integrated): Differences the time series to make it stationary (remove trends and seasonality).
- MA (Moving Average): Uses past forecast errors to predict future values.

ARIMA Notation:

- ARIMA models are denoted as ARIMA(p, d, q):
    - p: Order of the autoregressive (AR) component.
    - d: Order of differencing (I) required for stationarity.
    - q: Order of the moving average (MA) component.

ARIMA Formula (General Form):

The general ARIMA formula can be expressed as:

- $\hat{y}_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + ... + \varphi_p y_{t-p} + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} + e_t$

Where:

- $\hat{y}_t$: Predicted value at time t.
- c: Constant term (intercept).
- $\varphi_1, \varphi_2, ..., \varphi_p$: Autoregressive (AR) parameters.
- $y_{t-1}, y_{t-2}, ..., y_{t-p}$: Past values of the time series.
- $\theta_1, \theta_2, ..., \theta_q$: Moving average (MA) parameters.
- $e_{t-1}, e_{t-2}, ..., e_{t-q}$: Past forecast errors.
- e_t: Current forecast error (white noise).

Breakdown of Components:

1. AR (p):
    - $\varphi_1 y_{t-1} + \varphi_2 y_{t-2} + ... + \varphi_p y_{t-p}$
    - This part models the relationship between the current value and past values.
2. I (d):
    - Differencing is applied to the time series before applying the AR and MA components.
    - If d = 1, the first difference is calculated: $y_t' = y_t - y_{t-1}$.
    - If d = 2, the second difference is calculated, and so on.
    - The differenced series (y') is used in the AR and MA parts of the model.
3. MA (q):

- θ1et−1+θ2et−2+...+θqet−q
- This part models the relationship between the current value and past forecast errors

4. Exampla:-

```
from statsmodels.tsa.arima.model import ARIMA
series = df['Close']
model = ARIMA(series, order=(5, 1, 0))
arima_fit = model.fit()
arima_preds = arima_fit.forecast(ste
ps=7)
```

4.1.1.7 Prophet

Prophet, developed by Facebook (Meta), is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly and weekly seasonality, plus holidays. It works [1] best with time series that have strong seasonal effects and several seasons of historical data.

Here's a breakdown of the Prophet model and an example:

Prophet Model Components:

Prophet decomposes a time series into three main components:

- Trend (g(t)):
  - Models non-periodic changes in the time series. Prophet offers two trend models:
    - Linear growth.
    - Logistic growth (for saturating trends).
- Seasonality (s(t)):
  - Models periodic changes, like weekly and yearly seasonality. Prophet uses Fourier series to represent seasonality.

- Holidays (h(t)):
    - Models the effects of holidays or other recurring events. Users provide a custom list of holidays.
- Error (ε(t)):
    - Models any idiosyncratic changes which are not accounted for by the model.

The Prophet model can be represented as:

- $y(t) = g(t) + s(t) + h(t) + \varepsilon(t)$

Example:

Let's consider an example of forecasting website traffic.

1. Data Preparation:
    - Assume we have a dataset with daily website traffic, with columns "ds" (date) and "y" (traffic).
    - Prophet requires the date column to be named "ds" and the target variable to be named "y".
2. Python Code:

```
import pandas as pd
from prophet import Prophet
import matplotlib.pyplot as plt


# Sample Data. Create a simple dataframe.
data = pd.DataFrame({
  'ds': pd.to_datetime(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04', '2020-01-05',
                        '2020-01-06', '2020-01-07', '2020-01-08', '2020-01-09', '2020-01-10',
                        '2020-01-11', '2020-01-12', '2020-01-13', '2020-01-14', '2020-01-15']),
  'y': [100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240]
})


# Initialize and fit the model
```

```
model = Prophet()

model.fit(data)


# Create future dataframe

future = model.make_future_dataframe(periods=30) #predict 30 days into the future


# Make predictions

forecast = model.predict(future)


# Plot the forecast

fig1 = model.plot(forecast)

plt.show()


# Plot the components

fig2 = model.plot_components(forecast)

plt.show()
```

3. Explanation:

- We create a pandas DataFrame with our time series data.
- We initialize a Prophet model and fit it to our data.
- We create a future DataFrame to specify how far into the future we want to forecast.
- We make predictions using the predict() method.
- We then plot the forecast, and the components of the forecast.

4. Key Benefits of Prophet:

- Handles missing data and outliers well.
- Automatic detection of trend changes.

- Easy to tune and interpret.
- Works well with strong seasonal data.

4.1.1.8 GARCH

GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models are used to model and forecast volatility in time series [1] data, particularly in financial markets. They address the phenomenon of heteroskedasticity, where the variance of the errors in a time series changes over time.

Here's a breakdown of the GARCH model and its core formula:

Core Idea:

- GARCH models assume that the variance of the current error term is a function of past error terms and past variances.
- This allows them to capture the clustering of volatility, where periods of high volatility tend to be followed by more periods of high volatility, and vice versa.

GARCH(p, q) Formula:

The general formula for a GARCH(p, q) model is:

- $\sigma^2(t) = \omega + \alpha_1(\varepsilon^2(t-1)) + ... + \alpha<0xE2><0x82><0x99>(\varepsilon^2(t-q)) + \beta_1(\sigma^2(t-1)) + ... + \beta_p(\sigma^2(t-p))$

Where:

- $\sigma^2(t)$: Conditional variance at time t.
- ω: Constant term.
- α_i: Coefficients of the lagged squared error terms ($\varepsilon^2(t-i)$).
- $\varepsilon^2(t-i)$: Lagged squared error terms.
- β_j: Coefficients of the lagged conditional variance terms ($\sigma^2(t-j)$).
- $\sigma^2(t-j)$: Lagged conditional variance terms.
- q: Order of the ARCH component (lagged squared errors).
- p: Order of the GARCH component (lagged conditional variances).

Explanation:

- The formula shows that the current conditional variance ($\sigma^2(t)$) is a linear function of:
    - A constant (ω).
    - Past squared errors ($\varepsilon^2(t-i)$), which represent the ARCH component.
    - Past conditional variances ($\sigma^2(t-j)$), which represent the GARCH component.

GARCH(1, 1) Model:

The most commonly used GARCH model is the GARCH(1, 1) model, which has the following formula:

- $\sigma^2(t) = \omega + \alpha_1\varepsilon^2(t-1) + \beta_1\sigma^2(t-1)$

This simplified version means that the current variance is based on the previous days squared error, and the previous days variance.

Examole:-

```
from arch import arch_model

df['returns'] = df['Close'].pct_change().dropna()

garch_model = arch_model(df['returns']*100, vol='Garch', p=1, q=1)

garch_result = garch_model.fit()

garch_forecast = garch_result.forecast(horizon=5)

print(garch_forecast.variance[-1:])
```

4.1.2 Libraries

4.1.2.1 Streamlit

What it is:
Streamlit is a fast, open-source Python framework for building and deploying data science and machine learning web apps with minimal code.

Why used:
Streamlit is the backbone of the web interface in this project. It enables rapid prototyping of interactive dashboards where users can input stock tickers, adjust forecasting horizons, toggle technical indicators (e.g., SMAs), and view visual outputs (charts, sentiment analysis, and model predictions). Streamlit supports real-time updates and widgets

(e.g., sliders, checkboxes) essential for enhancing user experience, especially in financial analytics where dynamic interaction with data is crucial.

Example:-

```
import streamlit as st
st.title('Multi-Algorithm Stock Predictor')
stock = st.text_input('Enter Stock Ticker', 'AAPL')
forecast_days = st.slider('Forecast Horizon (days)', 7, 365)
if st.button('Predict'):
    st.write(f'Generating forecast for {stock} for {forecast_days} days.')
```

4.1.2.2 pandas

What it is:
Pandas provides data structures like Series and DataFrames designed for working with labeled, tabular, and time-series data.

Why used:
Pandas is used for collecting, organizing, and preprocessing stock market data. In this project, it's essential for tasks like:

- Aggregating daily OHLCV data into weekly or monthly periods.
- Calculating moving averages and daily returns.
- Merging technical indicators, model outputs, and sentiment scores into a master dataset for training and inference. Its robust time-series capabilities help align stock data with news articles, technical indicators, and model predictions based on dates.

Example:-

```
import pandas as pd
data = pd.read_csv('historical_stock_data.csv')
data['20_SMA'] = data['Close'].rolling(window=20).mean()
data['50_SMA'] = data['Close'].rolling(window=50).mean()
```

4.1.2.3. numpy

What it is:
NumPy is the foundation for numerical computation in Python, supporting high-performance operations on arrays and matrices.

Why used:
In this system, NumPy powers:

- Mathematical operations for model feature engineering (e.g., calculating log returns, volatility, rolling statistics).
- Vectorized computations during data preprocessing (speeding up large-scale financial datasets).
- Underlying tensor and matrix calculations in machine learning models via TensorFlow and Scikit-learn.

Example:-

```
import numpy as np
returns = np.log(data['Close'] / data['Close'].shift(1))
volatility = np.std(returns) * np.sqrt(252)
```

4.1.2.4. matplotlib

What it is:
Matplotlib is a comprehensive library for static, animated, and interactive visualizations in Python.

Why used:
Used for static plots such as:

- Time-series plots showing historical prices, moving averages, and trend lines.
- Model evaluation visuals (e.g., plotting residuals of ARIMA, ACF/PACF plots).
- Risk analysis charts like volatility heatmaps. It provides fine control over plot aesthetics, which is key for building financial reports.

Example:-

```
import matplotlib.pyplot as plt
plt.plot(data['Close'], label='Close Price')
plt.plot(data['20_SMA'], label='20-Day SMA')
plt.legend()
plt.title('Stock Price with Moving Averages')
plt.show()
```

4.1.2.5. scikit-learn

What it is:
Scikit-learn is a Python library offering simple and efficient tools for predictive data analysis and machine learning.

Why used:
It's critical for:

- Training baseline models such as Linear Regression, SVR, and Random Forest on engineered features.
- Model validation through tools like GridSearchCV and cross-validation.

- Scaling datasets (e.g., using StandardScaler) to ensure models like SVM and neural networks converge effectively. Scikit-learn also integrates smoothly with Pandas and NumPy for feature pipelines.

Example:-

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split


X = data[['20_SMA', '50_SMA']]

y = data['Close'].shift(-1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = RandomForestRegressor()

model.fit(X_train, y_train)
```

4.1.2.6. xgboost

What it is:
XGBoost is an advanced implementation of gradient boosted decision trees designed for high performance and scalability.

Why used:
In the project, XGBoost is used to:

- Model complex non-linear relationships between stock technical indicators and future price movements.
- Handle large datasets efficiently, with built-in regularization to reduce overfitting.
- Provide feature importance rankings to identify which technical indicators or sentiment features contribute most to predictions.

Example:-

```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train, label=y_train)

dtest = xgb.DMatrix(X_test)

params = {'objective': 'reg:squarederror'}

bst = xgb.train(params, dtrain, num_boost_round=100)
```

4.1.2.7. tensorflow

What it is:
TensorFlow is an end-to-end open-source deep learning framework developed by Google, supporting both low-level and high-level APIs.

Why used:
Key for:

- Implementing LSTM (Long Short-Term Memory) neural networks, which are ideal for modeling sequential stock price data.
- Training deep learning models to capture long-term dependencies in financial time-series data.
- Leveraging GPU acceleration for faster training on large datasets. TensorFlow's flexibility also enables integration with Keras for simplified neural network development.

Example:-

```
  import tensorflow as tf
model = tf.keras.Sequential([
   tf.keras.layers.LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
   tf.keras.layers.LSTM(50),
   tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

4.1.2.8. yfinance

What it is:
yfinance is a Python library that allows users to access financial data from Yahoo Finance.

Why used:
It acts as the data ingestion layer by:

- Fetching live and historical stock price data (open, high, low, close, volume).
- Supporting automatic downloading of dividends and stock splits information.
- Providing intraday (1-min to 1-hour) and daily intervals for building models with flexible resolutions.

Example:-

```
  import yfinance as yf
stock_data = yf.download('AAPL', start='2020-01-01', end='2024-12-31')
```

4.1.2.9. newsapi-python

What it is:
newsapi-python is a wrapper for the NewsAPI REST service, which aggregates global news from major sources.

Why used:
This system pulls real-time or historical news related to specific stock tickers. News headlines are then processed to extract sentiment, offering additional context that pure price-based models may miss. It enriches the dataset with external market sentiment.

Example:-

from newsapi import NewsApiClient

newsapi = NewsApiClient(api_key='YOUR_API_KEY')

articles = newsapi.get_everything(q='Apple', language='en', sort_by='relevancy')

4.1.2.9. newsapi-python

What it is:
Statsmodels is a Python library designed for statistical modeling and hypothesis testing.

Why used:
This library is used for:

- Time-series models such as ARIMA and SARIMA for stock price forecasting.
- Conducting statistical diagnostics like Augmented Dickey-Fuller (ADF) tests to verify stationarity.
- Regression diagnostics for evaluating traditional econometric models.

Example:-

import statsmodels.api as sm

model = sm.tsa.ARIMA(data['Close'], order=(1,1,1))

result = model.fit()

result.summary()

4.1.2.11. prophet

What it is:
Prophet is a time-series forecasting library developed by Meta for detecting seasonal patterns and changepoints.

Why used:
Prophet is used to:

- Forecast stock prices with uncertainty intervals over customizable time horizons (e.g., 7-365 days).
- Decompose stock data into trend, weekly, and yearly seasonalities.
- Handle missing data gracefully and accommodate known holidays or events affecting price movement.

Example:-

from prophet import Prophet

df = data.rename(columns={'Date': 'ds', 'Close': 'y'})

m = Prophet()

m.fit(df)

future = m.make_future_dataframe(periods=30)

forecast = m.predict(future)

4.1.2.12. plotly

What it is:
Plotly is an interactive graphing library that supports zooming, panning, and tooltips.

Why used:
It enhances user experience by:

- Creating interactive candlestick charts.
- Allowing users to explore predictions dynamically inside the Streamlit app.
- Enabling visualization of Prophet forecasts with adjustable confidence intervals and hover tooltips.

Example:-

```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Candlestick(x=data.index,
                    open=data['Open'],
                    high=data['High'],
                    low=data['Low'],
                    close=data['Close'])])
fig.show()
```

4.1.2.13. arch

What it is:
ARCH is a specialized package for modeling financial time-series volatility, focusing on GARCH-family models.

Why used:
Essential for:

- Modeling volatility clustering in stock returns.
- Forecasting future market volatility, which complements price forecasts with risk metrics.
- Performing volatility diagnostics to detect heteroskedasticity (non-constant variance) in residuals.

Example:-

```
  from arch import arch_model
returns = 100 * returns.dropna()
garch = arch_model(returns, vol='Garch', p=1, q=1)
res = garch.fit()
```

4.1.2.14. ta (Technical Analysis Library)

What it is:
ta is a Python package that provides implementations of over 100 technical indicators.

Why used:
Used to compute:

- Trend indicators like SMA, EMA, and MACD.
- Momentum indicators such as RSI and Stochastic Oscillator.
- Volatility indicators like Bollinger Bands and ATR. These features feed directly into machine learning models as predictive variables.

Example:-

```
import ta

data['RSI'] = ta.momentum.rsi(data['Close'])

data['MACD'] = ta.trend.macd(data['Close'])
```

4.1.2.15. nltk

What it is:
NLTK (Natural Language Toolkit) is a comprehensive library for natural language processing and linguistic data analysis.

Why used:
In this project:

- It's used for tokenization, stop-word removal, and basic text cleaning of news headlines.
- Prepares the text for downstream sentiment analysis by TextBlob or other NLP tools.

Example:-

```
import nltk

nltk.download('punkt')

from nltk.tokenize import word_tokenize

tokens = word_tokenize("Apple's earnings beat expectations.")
```

4.1.2.16. textblob

What it is:
TextBlob is a simple NLP library built on top of NLTK and Pattern, offering sentiment analysis and other NLP functions.

Why used:
TextBlob is used to:

- Analyze the polarity (positive/negative) and subjectivity of financial news headlines.

- Add a qualitative layer to stock predictions by correlating news sentiment with stock movements.
- Generate sentiment scores, which are later incorporated as input features for machine learning models.

Example:-

```
from textblob import TextBlob

headline = "Apple stock surges after positive earnings report"

sentiment = TextBlob(headline).sentiment
```

4.2 Testing

4.2.1 Test Cases

1. Stock Data Fetching

Test Case ID: TC_01
Test Case: Verify historical stock data fetching
Preconditions: yfinance installed and internet connection
Input: Stock ticker symbol (e.g., AAPL)
Expected Output: Historical stock data (OHLCV) displayed
Priority: High

---

2. SMA Calculation

Test Case ID: TC_02
Test Case: Validate 20-day and 50-day SMA computation
Preconditions: Historical stock data available
Input: Close price data
Expected Output: SMA values correctly added to dataframe
Priority: High

---

3. RSI Calculation

Test Case ID: TC_03
Test Case: Ensure RSI indicator calculation
Preconditions: Historical stock data available
Input: Close price data
Expected Output: RSI values computed and added to dataframe
Priority: Medium

---

4. News Sentiment Analysis

Test Case ID: TC_04
Test Case: Validate sentiment analysis using TextBlob
Preconditions: News API key and newsapi-python installed
Input: News headlines for stock ticker

Expected Output: Sentiment polarity score returned
Priority: High

---

5. ARIMA Model

Test Case ID: TC_05
Test Case: Ensure ARIMA model forecast output
Preconditions: Dataframe with Close prices available
Input: Historical close prices
Expected Output: ARIMA model generates forecasted values
Priority: High

---

6. RandomForest Prediction

Test Case ID: TC_06
Test Case: Validate RandomForest model predictions
Preconditions: Feature engineering complete (SMA, RSI)
Input: Features dataframe
Expected Output: RF model outputs predictions
Priority: High

---

7. XGBoost Prediction

Test Case ID: TC_07
Test Case: Ensure XGBoost model predicts correctly
Preconditions: Feature dataframe available
Input: Features dataframe
Expected Output: Predicted values returned by XGBoost
Priority: High

---

8. LSTM Neural Network

Test Case ID: TC_08
Test Case: Validate LSTM model prediction
Preconditions: Sequential data processed
Input: Sequences of price data
Expected Output: LSTM model returns predicted values
Priority: High

---

9. Prophet Forecast

Test Case ID: TC_09
Test Case: Ensure Prophet model forecast accuracy
Preconditions: Prophet library installed
Input: Close price dataframe
Expected Output: Prophet returns future forecast + confidence interval
Priority: High

---

10. GARCH Volatility Modeling

Test Case ID: TC_10
Test Case: Ensure GARCH model fitting
Preconditions: Log returns calculated
Input: Log returns
Expected Output: GARCH model summary generated
Priority: Medium

---

11. Static Chart Plotting

Test Case ID: TC_11
Test Case: Validate matplotlib static plots
Preconditions: Historical stock data available
Input: Close prices, SMA, RSI
Expected Output: Static line chart shown in Streamlit
Priority: Medium

---

12. Interactive Plotly Chart

Test Case ID: TC_12
Test Case: Validate candlestick chart with Plotly
Preconditions: Plotly library installed
Input: Stock OHLC data
Expected Output: Interactive candlestick chart displays
Priority: Medium

---

13. Streamlit UI Loading

Test Case ID: TC_13
Test Case: Ensure Streamlit app loads correctly
Preconditions: App running with Streamlit
Input: Launch streamlit run command
Expected Output: UI components (charts, sliders, controls) displayed correctly
Priority: High

---

14. Ensemble Output

Test Case ID: TC_14
Test Case: Validate ensemble result output
Preconditions: All models ran successfully
Input: Aggregated model predictions
Expected Output: Combined signal generated (Buy/Hold/Sell)
Priority: High

4.2.1 Test Script

```
# test_stock_predictor.py

import unittest
```

```
import yfinance as yf
import ta
from newsapi import NewsApiClient
from textblob import TextBlob
from prophet import Prophet
from arch import arch_model
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import tensorflow as tf


class TestStockPredictor(unittest.TestCase):


    def setUp(self):
        self.stock = 'AAPL'
        self.data = yf.download(self.stock, start='2020-01-01', end='2024-01-01')


    def test_fetch_data(self):
        self.assertFalse(self.data.empty, "Stock data should not be empty")


    def test_calculate_indicators(self):
        self.data['20_SMA'] = ta.trend.sma_indicator(self.data['Close'], window=20)
        self.data['50_SMA'] = ta.trend.sma_indicator(self.data['Close'], window=50)
        self.data['RSI'] = ta.momentum.rsi(self.data['Close'])
        self.assertIn('20_SMA', self.data.columns)
        self.assertIn('RSI', self.data.columns)


    def test_sentiment(self):
        newsapi = NewsApiClient(api_key='YOUR_API_KEY')
        articles = newsapi.get_everything(q=self.stock, language='en', page_size=5)
        blob = TextBlob(articles['articles'][0]['title'])
        self.assertIsNotNone(blob.sentiment.polarity)
```

```
def test_prophet_forecast(self):
    df = self.data.reset_index()[['Date', 'Close']].rename(columns={'Date': 'ds', 'Close': 'y'})
    m = Prophet()
    m.fit(df)
    future = m.make_future_dataframe(periods=30)
    forecast = m.predict(future)
    self.assertFalse(forecast.empty)

def test_random_forest(self):
    features = self.data[['20_SMA', '50_SMA', 'RSI']].dropna()
    target = self.data['Close'].shift(-1).dropna()
    X = features.iloc[:-1]
    y = target.iloc[:-1]
    rf = RandomForestRegressor().fit(X, y)
    preds = rf.predict(X)
    self.assertEqual(len(preds), len(X))

def test_xgboost(self):
    features = self.data[['20_SMA', '50_SMA', 'RSI']].dropna()
    target = self.data['Close'].shift(-1).dropna()
    X = features.iloc[:-1]
    y = target.iloc[:-1]
    dtrain = xgb.DMatrix(X, label=y)
    model = xgb.train({'objective': 'reg:squarederror'}, dtrain, num_boost_round=10)
    preds = model.predict(xgb.DMatrix(X))
    self.assertEqual(len(preds), len(X))

def test_garch_model(self):
    returns = 100 * np.log(self.data['Close'] / self.data['Close'].shift(1)).dropna()
    garch = arch_model(returns, vol='Garch', p=1, q=1)
    res = garch.fit(disp='off')
    self.assertIsNotNone(res)
```

```
    def test_lstm_model(self):
        seq_data = self.data['Close'].values.reshape(-1, 1)
        seq_data = seq_data / np.max(seq_data)
        X_seq, y_seq = [], []
        for i in range(60, len(seq_data)):
            X_seq.append(seq_data[i-60:i])
            y_seq.append(seq_data[i])
        X_seq, y_seq = np.array(X_seq), np.array(y_seq)
        model = tf.keras.Sequential([
            tf.keras.layers.LSTM(10, input_shape=(X_seq.shape[1], 1)),
            tf.keras.layers.Dense(1)
        ])
        model.compile(optimizer='adam', loss='mse')
        model.fit(X_seq, y_seq, epochs=1, batch_size=32)
        self.assertTrue(model)

if __name__ == '__main__':
    unittest.main()
```

4.3 Code

```
import pandas as pd
import numpy as np
import streamlit as st
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from statsmodels.tsa.arima.model import ARIMA
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
from newsapi import NewsApiClient
import yfinance as yf
from prophet import Prophet
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.linear_model import LinearRegression
from textblob import TextBlob
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import re

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

# Download required NLTK data
try:
    nltk.data.find('vader_lexicon')
except LookupError:
    nltk.download('vader_lexicon')
    nltk.download('punkt')

st.set_page_config(page_title="Multi-Algorithm Stock Predictor", layout="wide")
st.markdown(
    "<h1 style='text-align: center;'>Multi-Algorithm Stock Predictor</h1>",
    unsafe_allow_html=True
)

st.markdown(
    """
    <p style='text-align: center; color: gray; font-size: 14px;'>
```

```
    Disclaimer: This application provides stock predictions based on algorithms and is intended for informational purposes only.
    Predictions may not be accurate, and users are encouraged to conduct their own research and consider consulting with a
    financial advisor before making any investment decisions. This is not financial advice, and I am not responsible for any
    outcomes resulting from the use of this application.
    </p>
    """,
    unsafe_allow_html=True
)
# API setup
NEWS_API_KEY = '0de37ca8af9748898518daf699189abf'
newsapi = NewsApiClient(api_key=NEWS_API_KEY)

@st.cache_data(ttl=3600)
def fetch_stock_data(symbol, days):
    end_date = datetime.now()
    start_date = end_date- timedelta(days=days)
    df = yf.download(symbol, start=start_date, end=end_date)
    return df

@st.cache_data(ttl=3600)
def get_news_headlines(symbol):
    try:
        news = newsapi.get_everything(
            q=symbol,
            language='en',
            sort_by='relevancy',
            page_size=5
        )
        return [(article['title'], article['description'], article['url'])
                for article in news['articles']]
    except Exception as e:
```

```
        print(f"News API error: {str(e)}")
        return []

@st.cache_data(ttl=300)
def get_current_price(symbol):
    """Fetch the current live price of a stock"""
    try:
        ticker = yf.Ticker(symbol)
        todays_data = ticker.history(period='1d')

        if todays_data.empty:
            return None

        # If market is open, we can get the current price
        if 'Open' in todays_data.columns and len(todays_data) > 0:
            # For market hours, use current price if available
            if 'regularMarketPrice' in ticker.info:
                current_price = ticker.info['regularMarketPrice']
                is_live = True
            else:
                # Fallback to the most recent close
                current_price = float(todays_data['Close'].iloc[-1])
                is_live = False

            # Get last update time
            last_updated = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            return {
                "price": current_price,
                "is_live": is_live,
                "last_updated": last_updated
            }
        return None
```

```
    except Exception as e:
        st.error(f"Error fetching current price: {str(e)}")
        return None

@st.cache_data(ttl=3600)
def analyze_sentiment(text):
    """
    Analyze sentiment using both VADER and TextBlob, with financial context
    """
    # Check if text is None or empty
    if not text or not isinstance(text, str):
        return {
            'sentiment': "⚖️ Neutral",
            'confidence': 0,
            'color': "gray",
            'score': 0
        }

    # Clean the text
    text = re.sub(r'[^\w\s]', '', text)

    # VADER analysis
    sia = SentimentIntensityAnalyzer()
    vader_scores = sia.polarity_scores(text)

    # TextBlob analysis
    blob = TextBlob(text)
    textblob_polarity = blob.sentiment.polarity

    # Enhanced financial context keywords with weights
    financial_pos = {
        'strong': 1.2,
        'climbed': 1.3,
```

```
    'up': 1.1,
    'higher': 1.1,
    'beat': 1.2,
    'exceeded': 1.2,
    'growth': 1.1,
    'profit': 1.1,
    'gain': 1.1,
    'positive': 1.1,
    'bullish': 1.3,
    'outperform': 1.2,
    'buy': 1.1,
    'upgrade': 1.2,
    'recovers': 1.3,
    'rose': 1.3,
    'closed higher': 1.4
}

financial_neg = {
    'weak': 1.2,
    'fell': 1.3,
    'down': 1.1,
    'lower': 1.1,
    'miss': 1.2,
    'missed': 1.2,
    'decline': 1.1,
    'loss': 1.1,
    'negative': 1.1,
    'bearish': 1.3,
    'underperform': 1.2,
    'sell': 1.1,
    'downgrade': 1.2,
    'sell-off': 1.4,
    'rattled': 1.3,
```

```
    'correction': 1.3,
    'crossed below': 1.4,
    'pain': 1.3
}

# Add financial context with weighted scoring
financial_score = 0
words = text.lower().split()

# Look for percentage changes with context
percent_pattern = r'(\d+(?:\.\d+)?)\s*%'
percentages = re.findall(percent_pattern, text)
for pct in percentages:
    if any(term in text.lower() for term in ["rose", "up", "climb", "gain", "higher"]):
        financial_score += float(pct) * 0.15
    elif any(term in text.lower() for term in ["down", "fall", "drop", "lower", "decline"]):
        financial_score-= float(pct) * 0.15

# Look for technical indicators
if "moving average" in text.lower():
    if "crossed below" in text.lower() or "below" in text.lower():
        financial_score-= 1.2
    elif "crossed above" in text.lower() or "above" in text.lower():
        financial_score += 1.2

# Look for market action terms
if "sell-off" in text.lower() or "selloff" in text.lower():
    financial_score-= 1.3
if "recovery" in text.lower() or "recovers" in text.lower():
    financial_score += 1.3

# Calculate weighted keyword scores
pos_score = sum(financial_pos.get(word, 0) for word in words)
```

```
    neg_score = sum(financial_neg.get(word, 0) for word in words)

    if pos_score or neg_score:
        financial_score += (pos_score- neg_score) / (pos_score + neg_score)

    # Combine scores with adjusted weights
    combined_score = (
        vader_scores['compound'] * 0.3 +     # VADER
        textblob_polarity * 0.2 +           # TextBlob
        financial_score * 0.5               # Enhanced financial context (increased weight)
    )

    # Adjust thresholds and confidence calculation
    if combined_score >= 0.15:
        sentiment = "📈 Positive"
        confidence = min(abs(combined_score) * 150, 100)  # Increased multiplier
        color = "green"
    elif combined_score <=-0.15:
        sentiment = "📉 Negative"
        confidence = min(abs(combined_score) * 150, 100)
        color = "red"
    else:
        sentiment = "⚖️ Neutral"
        confidence = (1- abs(combined_score)) * 100
        color = "gray"

    return {
        'sentiment': sentiment,
        'confidence': confidence,
        'color': color,
        'score': combined_score
    }
```

```
# Completely revise the Prophet forecast function
@st.cache_data(ttl=3600)
def forecast_with_prophet(df, forecast_days=30):
    try:
        # Check if we have enough data points
        if len(df) < 30:
            st.warning("Not enough historical data for reliable forecasting (< 30 data points)")
            return simple_forecast_fallback(df, forecast_days)

        # Make a copy to avoid modifying the original dataframe
        df_copy = df.copy()

        # Check for MultiIndex columns and handle appropriately
        has_multiindex = isinstance(df_copy.columns, pd.MultiIndex)

        # Reset index to make Date a column
        df_copy = df_copy.reset_index()

        # Find the date column
        date_col = None
        for col in df_copy.columns:
            # Handle both string and tuple column names
            col_str = col if isinstance(col, str) else col[0]
            if isinstance(col_str, str) and col_str.lower() in ['date', 'datetime', 'time', 'index']:
                date_col = col
                break

        if date_col is None:
            st.warning("No date column found- using simple forecast")
            return simple_forecast_fallback(df, forecast_days)

        # Prepare data for Prophet with careful handling of column types
        prophet_df = pd.DataFrame()
```

```
# Extract the date and price columns safely
date_values = df_copy[date_col]

# For Close column, check if we're dealing with a MultiIndex
if has_multiindex:
    # If MultiIndex, find the column with 'Close' as first element
    close_col = None
    for col in df_copy.columns:
        if isinstance(col, tuple) and col[0] == 'Close':
            close_col = col
            break

    if close_col is None:
        st.warning("No Close column found- using simple forecast")
        return simple_forecast_fallback(df, forecast_days)

    close_values = df_copy[close_col]
else:
    # Standard columns
    close_values = df_copy['Close']

# Assign to prophet dataframe
prophet_df['ds'] = pd.to_datetime(date_values)
prophet_df['y'] = close_values.astype(float)

# Add additional features for regressors- even more comprehensive
# Add volume as a regressor if available
has_volume_regressor = False
if 'Volume' in df_copy.columns:
    prophet_df['volume'] = df_copy['Volume'].astype(float)
    prophet_df['log_volume'] = np.log1p(prophet_df['volume'])  # log transform to handle skewness
    # Add volume momentum (rate of change)
```

```
    prophet_df['volume_roc'] = prophet_df['volume'].pct_change(periods=5).fillna(0)
    has_volume_regressor = True

# Add price-based features
# Volatility at different time windows
prophet_df['volatility_5d'] = prophet_df['y'].rolling(window=5).std().fillna(0)
prophet_df['volatility_10d'] = prophet_df['y'].rolling(window=10).std().fillna(0)
prophet_df['volatility_20d'] = prophet_df['y'].rolling(window=20).std().fillna(0)

# Relative strength indicator (simplified)
delta = prophet_df['y'].diff()
gain = delta.mask(delta < 0, 0).rolling(window=14).mean()
loss =-delta.mask(delta > 0, 0).rolling(window=14).mean()
rs = gain / loss
prophet_df['rsi'] = 100- (100 / (1 + rs)).fillna(50)

# Price momentum
prophet_df['momentum_5d'] = prophet_df['y'].pct_change(periods=5).fillna(0)
prophet_df['momentum_10d'] = prophet_df['y'].pct_change(periods=10).fillna(0)

# Distance from moving averages
prophet_df['ma10'] = prophet_df['y'].rolling(window=10).mean().fillna(method='bfill')
prophet_df['ma20'] = prophet_df['y'].rolling(window=20).mean().fillna(method='bfill')
prophet_df['ma10_dist'] = (prophet_df['y'] / prophet_df['ma10']- 1)
prophet_df['ma20_dist'] = (prophet_df['y'] / prophet_df['ma20']- 1)

# Bollinger band position
bb_std = prophet_df['y'].rolling(window=20).std().fillna(0)
prophet_df['bb_position'] = (prophet_df['y']- prophet_df['ma20']) / (2 * bb_std + 1e-10)  # Avoid division by zero

# Handle outliers by winsorizing extreme values
# Helps with improving forecast accuracy by removing noise
for col in prophet_df.columns:
```

```
    if col != 'ds' and prophet_df[col].dtype.kind in 'fc':  # if column is float or complex
        q1 = prophet_df[col].quantile(0.01)
        q3 = prophet_df[col].quantile(0.99)
        prophet_df[col] = prophet_df[col].clip(q1, q3)

# Drop any NaN values
prophet_df = prophet_df.dropna()

# Determine appropriate seasonality based on data size
daily_seasonality = len(prophet_df) > 90  # Only use daily seasonality with enough data
weekly_seasonality = False  # Explicitly disable weekly seasonality for stocks
yearly_seasonality = len(prophet_df) > 365

# Adaptive parameter selection based on volatility
recent_volatility = prophet_df['volatility_20d'].mean()
avg_price = prophet_df['y'].mean()
rel_volatility = recent_volatility / avg_price

# Adjust changepoint_prior_scale based on volatility
# Higher volatility-> more flexibility
cp_prior_scale = min(0.05 + rel_volatility * 0.5, 0.5)

# Create and fit the model with optimized parameters
model = Prophet(
    daily_seasonality=daily_seasonality,
    weekly_seasonality=weekly_seasonality,  # Disabled to prevent weekend spikes
    yearly_seasonality=yearly_seasonality,
    changepoint_prior_scale=cp_prior_scale,  # Adaptive to volatility
    seasonality_prior_scale=10.0,  # Increased to capture market seasonality better
    seasonality_mode='multiplicative',  # Better for stock data that tends to have proportional changes
    changepoint_range=0.95,  # Look at more recent changepoints for stocks
    interval_width=0.9  # 90% confidence interval
)
```

```
# Add US stock market holidays
model.add_country_holidays(country_name='US')

# Add custom regressors
if has_volume_regressor:
    model.add_regressor('log_volume', mode='multiplicative')
    model.add_regressor('volume_roc', mode='additive')

# Add technical indicators as regressors
model.add_regressor('volatility_5d', mode='multiplicative')
model.add_regressor('volatility_20d', mode='multiplicative')
model.add_regressor('rsi', mode='additive')
model.add_regressor('momentum_5d', mode='additive')
model.add_regressor('momentum_10d', mode='additive')
model.add_regressor('ma10_dist', mode='additive')
model.add_regressor('ma20_dist', mode='additive')
model.add_regressor('bb_position', mode='additive')

# Add custom seasonality for common stock patterns
if len(prophet_df) > 60:  # Only with enough data
    model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
    model.add_seasonality(name='quarterly', period=91.25, fourier_order=5)

# Add beginning/end of month effects (common in stocks)
if len(prophet_df) > 40:
    prophet_df['month_start'] = (prophet_df['ds'].dt.day <= 3).astype(int)
    prophet_df['month_end'] = (prophet_df['ds'].dt.day >= 28).astype(int)
    model.add_regressor('month_start', mode='additive')
    model.add_regressor('month_end', mode='additive')

# For stocks with enough data, add quarterly earnings effect
if len(prophet_df) > 250:
```

```
    # Approximate earnings seasonality (rough quarterly pattern)
    prophet_df['earnings_season'] = ((prophet_df['ds'].dt.month % 3 == 0) &
                                     (prophet_df['ds'].dt.day >= 15) &
                                     (prophet_df['ds'].dt.day <= 30)).astype(int)

# Fit the model
model.fit(prophet_df)

# Create future dataframe for prediction using business days only
# This is critical to avoid weekend predictions for stock markets
last_date = prophet_df['ds'].max()
# Use business day frequency (weekdays only)
future_dates = pd.date_range(
    start=last_date + pd.Timedelta(days=1),
    periods=forecast_days * 1.4,  # Add extra days to account for weekends
    freq='B'  # Business day frequency- weekdays only
)[:forecast_days]  # Limit to requested forecast days

# Create the future dataframe with correct dates
future = pd.DataFrame({'ds': future_dates})

# Add regressor values to future dataframe
# Copy the last rows of data for future predictions
last_values = prophet_df.iloc[-1].copy()
future_start_idx = len(prophet_df)

# Add volume regressors to future dataframe
if has_volume_regressor:
    # For volume, use median of last 30 days as future values
    median_volume = prophet_df['volume'].tail(30).median()
    future['volume'] = median_volume
    future['log_volume'] = np.log1p(future['volume'])
```

```
    # For volume_roc, use last 5-day average
    future['volume_roc'] = prophet_df['volume_roc'].tail(5).mean()

# Add technical indicators to future dataframe
# Use recent averages for future values
future['volatility_5d'] = prophet_df['volatility_5d'].tail(10).mean()
future['volatility_20d'] = prophet_df['volatility_20d'].tail(10).mean()
future['rsi'] = prophet_df['rsi'].tail(5).mean()
future['momentum_5d'] = prophet_df['momentum_5d'].tail(5).mean()
future['momentum_10d'] = prophet_df['momentum_10d'].tail(5).mean()
future['ma10_dist'] = prophet_df['ma10_dist'].tail(5).mean()
future['ma20_dist'] = prophet_df['ma20_dist'].tail(5).mean()
future['bb_position'] = prophet_df['bb_position'].tail(5).mean()

# Add month start/end flags if we calculated them
if 'month_start' in prophet_df.columns:
    future['month_start'] = (future['ds'].dt.day <= 3).astype(int)
    future['month_end'] = (future['ds'].dt.day >= 28).astype(int)

# Add earnings season flags if we calculated them
if 'earnings_season' in prophet_df.columns:
    future['earnings_season'] = ((future['ds'].dt.month % 3 == 0) &
                                 (future['ds'].dt.day >= 15) &
                                 (future['ds'].dt.day <= 30)).astype(int)

# Make predictions
forecast = model.predict(future)

# Post-processing for improved accuracy:
# 1. Ensure forecasts don't go negative for stock prices
forecast['yhat'] = np.maximum(forecast['yhat'], 0)
forecast['yhat_lower'] = np.maximum(forecast['yhat_lower'], 0)
```

```
        # 2. Apply an exponential decay to prediction intervals for uncertainty growth
        if forecast_days > 7:
            future_dates = pd.to_datetime(forecast['ds']) > prophet_df['ds'].max()
            days_out = np.arange(1, sum(future_dates) + 1)
            uncertainty_multiplier = 1 + (np.sqrt(days_out) * 0.01)

            # Adjust confidence intervals for future dates
            future_indices = np.where(future_dates)[0]
            for i, idx in enumerate(future_indices):
                forecast.loc[idx, 'yhat_upper'] = (forecast.loc[idx, 'yhat'] +
                                                   (forecast.loc[idx, 'yhat_upper']-
                                                    forecast.loc[idx, 'yhat']) * uncertainty_multiplier[i])
                forecast.loc[idx, 'yhat_lower'] = (forecast.loc[idx, 'yhat']-
                                                   (forecast.loc[idx, 'yhat']-
                                                    forecast.loc[idx, 'yhat_lower']) * uncertainty_multiplier[i])

        # Make sure there are no weekend forecasts by checking the day of week
        # 5 = Saturday, 6 = Sunday
        forecast = forecast[forecast['ds'].dt.dayofweek < 5]

        return forecast

    except Exception as e:
        st.warning(f"Prophet model failed: {str(e)}. Using simple forecast instead.")
        return simple_forecast_fallback(df, forecast_days)

# Fix the simple forecast fallback
def simple_forecast_fallback(df, forecast_days=30):
    """A simple linear regression forecast as fallback when Prophet fails"""
    try:
        # Get the closing prices as a simple 1D array
        close_prices = df['Close'].values.flatten()
```

```
# Create a sequence for x values (0, 1, 2, ...)
x = np.arange(len(close_prices)).reshape(-1, 1)
y = close_prices

# Fit a simple linear regression
model = LinearRegression()
model.fit(x, y)

# Create future dates for forecasting- using business days only
last_date = df.index[-1]

# Generate business days only (exclude weekends)
future_dates = pd.date_range(
    start=last_date + pd.Timedelta(days=1),
    periods=forecast_days * 1.4,  # Add extra days to account for weekends
    freq='B'  # Business day frequency- weekdays only
)[:forecast_days]  # Limit to requested forecast days

# Historical dates and all dates together
historical_dates = df.index
all_dates = historical_dates.append(future_dates)

# Predict future values
future_x = np.arange(len(close_prices), len(close_prices) + len(future_dates)).reshape(-1, 1)
future_y = model.predict(future_x)

# Predict historical values for context
historical_y = model.predict(x)

# Calculate confidence interval (simple approach)
mse = np.mean((y- historical_y) ** 2)
sigma = np.sqrt(mse)
```

```
        # Create separate arrays for each column to ensure they're 1D
        ds_array = np.array(all_dates, dtype='datetime64')

        # Concatenate historical and future predictions
        yhat_array = np.concatenate([historical_y, future_y])
        yhat_lower_array = yhat_array- 1.96 * sigma
        yhat_upper_array = yhat_array + 1.96 * sigma

        # For trend/weekly/yearly, create simple placeholders
        trend_array = yhat_array.copy()  # Use the prediction as the trend
        weekly_array = np.zeros(len(yhat_array))  # No weekly component
        yearly_array = np.zeros(len(yhat_array))  # No yearly component

        # Create a forecast dataframe similar to Prophet's output
        forecast = pd.DataFrame({
            'ds': ds_array,
            'yhat': yhat_array,
            'yhat_lower': yhat_lower_array,
            'yhat_upper': yhat_upper_array,
            'trend': trend_array,
            'weekly': weekly_array,
            'yearly': yearly_array
        })

        return forecast

    except Exception as e:
        st.error(f"Simple forecast also failed: {str(e)}. No forecast will be shown.")
        return None

def calculate_technical_indicators_for_summary(df):
    analysis_df = df.copy()
```

```
    # Calculate Moving Averages
    analysis_df['MA20'] = analysis_df['Close'].rolling(window=20).mean()
    analysis_df['MA50'] = analysis_df['Close'].rolling(window=50).mean()

    # Calculate RSI
    delta = analysis_df['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    analysis_df['RSI'] = 100- (100 / (1 + rs))

    # Calculate Volume MA
    analysis_df['Volume_MA'] = analysis_df['Volume'].rolling(window=20).mean()

    # Calculate Bollinger Bands
    ma20 = analysis_df['Close'].rolling(window=20).mean()
    std20 = analysis_df['Close'].rolling(window=20).std()
    analysis_df['BB_upper'] = ma20 + (std20 * 2)
    analysis_df['BB_lower'] = ma20- (std20 * 2)
    analysis_df['BB_middle'] = ma20

    return analysis_df

class MultiAlgorithmStockPredictor:
    def __init__(self, symbol, training_years=2, weights=None):  # Reduced from 5 to 2 years
        self.symbol = symbol
        self.training_years = training_years
        self.scaler = MinMaxScaler(feature_range=(0, 1))
        self.weights = weights if weights is not None else WEIGHT_CONFIGURATIONS["Default"]

    def fetch_historical_data(self):
        # Same as original EnhancedStockPredictor
        end_date = datetime.now()
```

```
        start_date = end_date- timedelta(days=365 * self.training_years)


        try:
            df = yf.download(self.symbol, start=start_date, end=end_date)
            if df.empty:
                st.warning(f"Data for the last {self.training_years} years is unavailable. Fetching maximum available data instead.")
                df = yf.download(self.symbol, period="max")
            return df
        except Exception as e:
            st.error(f"Error fetching data: {str(e)}")
            return yf.download(self.symbol, period="max")


    # Technical indicators calculation methods remain the same
    def calculate_technical_indicators(self, df):
        # Original technical indicators remain the same
        df['MA5'] = df['Close'].rolling(window=5).mean()
        df['MA20'] = df['Close'].rolling(window=20).mean()
        df['MA50'] = df['Close'].rolling(window=50).mean()
        df['MA200'] = df['Close'].rolling(window=200).mean()
        df['RSI'] = self.calculate_rsi(df['Close'])
        df['MACD'] = self.calculate_macd(df['Close'])
        df['ROC'] = df['Close'].pct_change(periods=10) * 100
        df['ATR'] = self.calculate_atr(df)
        df['BB_upper'], df['BB_lower'] = self.calculate_bollinger_bands(df['Close'])
        df['Volume_MA'] = df['Volume'].rolling(window=20).mean()
        df['Volume_Rate'] = df['Volume'] / df['Volume'].rolling(window=20).mean()


        # Additional technical indicators
        df['EMA12'] = df['Close'].ewm(span=12, adjust=False).mean()
        df['EMA26'] = df['Close'].ewm(span=26, adjust=False).mean()
        df['MOM'] = df['Close'].diff(10)
        df['STOCH_K'] = self.calculate_stochastic(df)
```

```
    df['WILLR'] = self.calculate_williams_r(df)

    return df.dropna()

@staticmethod
def calculate_stochastic(df, period=14):
    low_min = df['Low'].rolling(window=period).min()
    high_max = df['High'].rolling(window=period).max()
    k = 100 * ((df['Close']- low_min) / (high_max- low_min))
    return k

@staticmethod
def calculate_williams_r(df, period=14):
    high_max = df['High'].rolling(window=period).max()
    low_min = df['Low'].rolling(window=period).min()
    return-100 * ((high_max- df['Close']) / (high_max- low_min))

# Original calculation methods remain the same
@staticmethod
def calculate_rsi(prices, period=14):
    delta = prices.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    return 100- (100 / (1 + rs))

@staticmethod
def calculate_macd(prices, slow=26, fast=12, signal=9):
    exp1 = prices.ewm(span=fast, adjust=False).mean()
    exp2 = prices.ewm(span=slow, adjust=False).mean()
    return exp1- exp2
```

```
@staticmethod
def calculate_atr(df, period=14):
    high_low = df['High']- df['Low']
    high_close = np.abs(df['High']- df['Close'].shift())
    low_close = np.abs(df['Low']- df['Close'].shift())
    ranges = pd.concat([high_low, high_close, low_close], axis=1)
    true_range = np.max(ranges, axis=1)
    return true_range.rolling(period).mean()

@staticmethod
def calculate_bollinger_bands(prices, period=20, std_dev=2):
    ma = prices.rolling(window=period).mean()
    std = prices.rolling(window=period).std()
    upper_band = ma + (std * std_dev)
    lower_band = ma- (std * std_dev)
    return upper_band, lower_band

def prepare_data(self, df, seq_length=60):
    # Enhanced feature selection and engineering
    feature_columns = ['Close', 'MA5', 'MA20', 'MA50', 'MA200', 'RSI', 'MACD',
                       'ROC', 'ATR', 'BB_upper', 'BB_lower', 'Volume_Rate',
                       'EMA12', 'EMA26', 'MOM', 'STOCH_K', 'WILLR']

    # Add derivative features to capture momentum and acceleration
    df['Price_Momentum'] = df['Close'].pct_change(5)
    df['MA_Crossover'] = (df['MA5'] > df['MA20']).astype(int)
    df['RSI_Momentum'] = df['RSI'].diff(3)
    df['MACD_Signal'] = df['MACD']- df['MACD'].ewm(span=9).mean()
    df['Volume_Shock'] = ((df['Volume']- df['Volume'].shift(1)) / df['Volume'].shift(1)).clip(-1, 1)

    # Add market regime detection (trending vs range-bound)
    df['ADX'] = self.calculate_adx(df)
    df['Is_Trending'] = (df['ADX'] > 25).astype(int)
```

```
# Calculate volatility features
df['Volatility_20d'] = df['Close'].pct_change().rolling(window=20).std() * np.sqrt(252)

# Add day of week feature (market often behaves differently on different days)
df['DayOfWeek'] = df.index.dayofweek

# Create dummy variables for day of week
for i in range(5):  # 0-4 for Monday-Friday
    df[f'Day_{i}'] = (df['DayOfWeek'] == i).astype(int)

# Handle extreme outliers by winsorizing
for col in df.columns:
    if col != 'DayOfWeek' and df[col].dtype in [np.float64, np.int64]:
        q1 = df[col].quantile(0.01)
        q3 = df[col].quantile(0.99)
        df[col] = df[col].clip(q1, q3)

# Select the final set of features
enhanced_features = feature_columns + ['Price_Momentum', 'MA_Crossover', 'RSI_Momentum',
                                       'MACD_Signal', 'Volume_Shock', 'ADX', 'Is_Trending',
                                       'Volatility_20d', 'Day_0', 'Day_1', 'Day_2', 'Day_3', 'Day_4']

# Ensure all selected features exist and drop NaN values
available_features = [col for col in enhanced_features if col in df.columns]
df_cleaned = df[available_features].copy()
df_cleaned = df_cleaned.dropna()

# Scale features
scaled_data = self.scaler.fit_transform(df_cleaned)

# Prepare sequences for LSTM
X_lstm, y = [], []
```

```
    for i in range(seq_length, len(scaled_data)):
        X_lstm.append(scaled_data[i-seq_length:i])
        y.append(scaled_data[i, 0])  # 0 index represents Close price

    # Prepare data for other models
    X_other = scaled_data[seq_length:]

    return np.array(X_lstm), X_other, np.array(y), df_cleaned.columns.tolist()

@staticmethod
def calculate_adx(df, period=14):
    """Calculate Average Directional Index (ADX) to identify trend strength"""
    try:
        # Calculate True Range
        high_low = df['High']- df['Low']
        high_close = abs(df['High']- df['Close'].shift())
        low_close = abs(df['Low']- df['Close'].shift())

        # Use .values to get numpy arrays and avoid pandas alignment issues
        ranges = pd.DataFrame({'hl': high_low, 'hc': high_close, 'lc': low_close})
        tr = ranges.max(axis=1)
        atr = tr.rolling(period).mean()

        # Calculate Plus Directional Movement (+DM) and Minus Directional Movement (-DM)
        plus_dm = df['High'].diff()
        minus_dm = df['Low'].diff()

        # Handle conditions separately to avoid multi-column assignment
        plus_dm_mask = (plus_dm > 0) & (plus_dm > minus_dm.abs())
        plus_dm = plus_dm.where(plus_dm_mask, 0)

        minus_dm_mask = (minus_dm < 0) & (minus_dm.abs() > plus_dm)
        minus_dm = minus_dm.abs().where(minus_dm_mask, 0)
```

```
        # Calculate Smoothed +DM and-DM
        smoothed_plus_dm = plus_dm.rolling(period).sum()
        smoothed_minus_dm = minus_dm.rolling(period).sum()

        # Replace zeros to avoid division issues
        atr_safe = atr.replace(0, np.nan)

        # Calculate Plus Directional Index (+DI) and Minus Directional Index (-DI)
        plus_di = 100 * smoothed_plus_dm / atr_safe
        minus_di = 100 * smoothed_minus_dm / atr_safe

        # Handle division by zero in DX calculation
        di_sum = plus_di + minus_di
        di_sum_safe = di_sum.replace(0, np.nan)

        # Calculate Directional Movement Index (DX)
        dx = 100 * abs(plus_di- minus_di) / di_sum_safe

        # Calculate Average Directional Index (ADX)
        adx = dx.rolling(period).mean()

        return adx
    except Exception as e:
        # If ADX calculation fails, return a series of zeros with same index as input
        return pd.Series(0, index=df.index)

def build_lstm_model(self, input_shape):
    # Simplified LSTM architecture for faster training
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(32, return_sequences=False),
```

```
        Dropout(0.2),
        Dense(16, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='huber', metrics=['mae'])
    return model

def train_arima(self, df):
    # Auto-optimize ARIMA parameters
    from pmdarima import auto_arima

    try:
        model = auto_arima(df['Close'],
                           start_p=1, start_q=1,
                           max_p=5, max_q=5,
                           d=1, seasonal=False,
                           stepwise=True,
                           suppress_warnings=True,
                           error_action='ignore',
                           max_order=5)
        return model
    except:
        # Fallback to standard ARIMA
        model = ARIMA(df['Close'], order=(5,1,0))
        return model.fit()

def predict_with_all_models(self, prediction_days=30, sequence_length=30):  # Reduced sequence length
    try:
        # Fetch and prepare data
        df = self.fetch_historical_data()

        # Check if we have enough data
        if len(df) < sequence_length + 20:  # Need extra days for technical indicators
```

```
        st.warning(f"Insufficient historical data. Need at least {sequence_length + 20} days of data.")
        # Use available data but reduce sequence length if necessary
        sequence_length = max(10, len(df)- 20)

    # Calculate technical indicators
    df = self.calculate_technical_indicators(df)

    # Check for NaN values and handle them
    if df.isnull().any().any():
        df = df.fillna(method='ffill').fillna(method='bfill')

    # Verify we have enough valid data after cleaning
    if len(df.dropna()) < sequence_length:
        st.error("Insufficient valid data after calculating indicators.")
        return None

    # Enhanced data preparation with more features
    X_lstm, X_other, y, feature_names = self.prepare_data(df, sequence_length)

    # Verify we have valid data for model training
    if len(X_lstm) == 0 or len(y) == 0:
        st.error("Could not create valid sequences for prediction.")
        return None

    # Convert to numpy arrays
    X_lstm = np.array(X_lstm)
    X_other = np.array(X_other)
    y = np.array(y)

    # Split data using our optimized function
    X_lstm_train, X_lstm_test = X_lstm[:int(len(X_lstm)*0.8)], X_lstm[int(len(X_lstm)*0.8):]
    X_other_train, X_other_test = X_other[:int(len(X_other)*0.8)], X_other[int(len(X_other)*0.8):]
    y_train, y_test = y[:int(len(y)*0.8)], y[int(len(y)*0.8):]
```

```
        predictions = {}

        # Train and predict with LSTM (with reduced epochs)
        lstm_model = self.build_lstm_model((sequence_length, X_lstm.shape[2]))
        early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
        lstm_model.fit(X_lstm_train, y_train, epochs=20, batch_size=32,  # Reduced from 50 to 20 epochs
                       validation_data=(X_lstm_test, y_test),
                       callbacks=[early_stopping], verbose=0)
        lstm_pred = lstm_model.predict(X_lstm_test[-1:], verbose=0)[0][0]
        predictions['LSTM'] = lstm_pred

        # Train and predict with SVR
        svr_model = SVR(kernel='rbf', C=100, epsilon=0.1)
        svr_model.fit(X_other_train, y_train)
        svr_pred = svr_model.predict(X_other_test[-1:])
        predictions['SVR'] = svr_pred[0]

        # Train and predict with Random Forest (reduced complexity)
        rf_model = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1)  # Reduced from 100 to 50
trees
        rf_model.fit(X_other_train, y_train)
        rf_pred = rf_model.predict(X_other_test[-1:])
        predictions['Random Forest'] = rf_pred[0]

        # Train and predict with XGBoost (reduced complexity)
        xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42, n_estimators=50)  # Added
n_estimators
        xgb_model.fit(X_other_train, y_train)
        xgb_pred = xgb_model.predict(X_other_test[-1:])
        predictions['XGBoost'] = xgb_pred[0]

        # Skip KNN and GBM for speed
        # Only include fast models when we have limited data
```

```
if len(X_other_train) > 100:
    # Train and predict with GBM (reduced complexity)
    gbm_model = GradientBoostingRegressor(random_state=42, n_estimators=50)  # Reduced complexity
    gbm_model.fit(X_other_train, y_train)
    gbm_pred = gbm_model.predict(X_other_test[-1:])
    predictions['GBM'] = gbm_pred[0]

# Simplified ARIMA- skip if we have other models
if len(predictions) < 3:
    try:
        close_prices = df['Close'].values
        arima_model = ARIMA(close_prices, order=(2,1,0))  # Simplified from (5,1,0)
        arima_fit = arima_model.fit()
        arima_pred = arima_fit.forecast(steps=1)[0]
        arima_scaled = (arima_pred- df['Close'].mean()) / df['Close'].std()
        predictions['ARIMA'] = arima_scaled
    except Exception as e:
        st.warning(f"ARIMA prediction failed: {str(e)}")

weights = self.weights

# Adjust weights if some models failed
available_models = list(predictions.keys())
total_weight = sum(weights.get(model, 0.1) for model in available_models)  # Default weight 0.1
adjusted_weights = {model: weights.get(model, 0.1)/total_weight for model in available_models}

ensemble_pred = sum(pred * adjusted_weights[model]
                    for model, pred in predictions.items())

# Inverse transform predictions
dummy_array = np.zeros((1, X_other.shape[1]))
dummy_array[0, 0] = ensemble_pred
final_prediction = self.scaler.inverse_transform(dummy_array)[0, 0]
```

```
        # Calculate prediction range
        individual_predictions = []
        for pred in predictions.values():
            dummy = dummy_array.copy()
            dummy[0, 0] = pred
            individual_predictions.append(
                self.scaler.inverse_transform(dummy)[0, 0]
            )

        std_dev = np.std(individual_predictions)

        return {
            'prediction': final_prediction,
            'lower_bound': final_prediction - std_dev,
            'upper_bound': final_prediction + std_dev,
            'confidence_score': 1 / (1 + std_dev / final_prediction),
            'individual_predictions': {
                model: pred for model, pred in zip(predictions.keys(), individual_predictions)
            }
        }

    except Exception as e:
        st.error(f"Error in prediction: {str(e)}")
        return None

# Streamlit interface

symbol = st.text_input("Enter Stock Symbol (e.g., AAPL):", "AAPL")

# Set default display days to 600
display_days = st.slider(
    "Select number of days to display",
```

```
    min_value=30,
    max_value=3650,
    value=600,  # Default to 600 days
    help="Displaying more days provides the model with more information for predictions."
)

# Define different weight configurations
WEIGHT_CONFIGURATIONS = {
    "Default": {
        'LSTM': 0.3,
        'XGBoost': 0.15,
        'Random Forest': 0.15,
        'ARIMA': 0.1,
        'SVR': 0.1,
        'GBM': 0.1,
        'KNN': 0.1
    },
    "Trend-Focused": {
        'LSTM': 0.35,
        'XGBoost': 0.20,
        'Random Forest': 0.15,
        'ARIMA': 0.10,
        'SVR': 0.08,
        'GBM': 0.07,
        'KNN': 0.05
    },
    "Statistical": {
        'LSTM': 0.20,
        'XGBoost': 0.15,
        'Random Forest': 0.15,
        'ARIMA': 0.20,
        'SVR': 0.15,
        'GBM': 0.10,
```

```
        'KNN': 0.05
    },
    "Tree-Ensemble": {
        'LSTM': 0.25,
        'XGBoost': 0.25,
        'Random Forest': 0.20,
        'ARIMA': 0.10,
        'SVR': 0.08,
        'GBM': 0.07,
        'KNN': 0.05
    },
    "Balanced": {
        'LSTM': 0.25,
        'XGBoost': 0.20,
        'Random Forest': 0.15,
        'ARIMA': 0.15,
        'SVR': 0.10,
        'GBM': 0.10,
        'KNN': 0.05
    },
    "Volatility-Focused": {
        'LSTM': 0.30,
        'XGBoost': 0.25,
        'Random Forest': 0.20,
        'ARIMA': 0.05,
        'SVR': 0.10,
        'GBM': 0.07,
        'KNN': 0.03
    }
}


WEIGHT_DESCRIPTIONS = {
    "Default": "Original configuration with balanced weights",
```

```
    "Trend-Focused": "Best for growth stocks, tech stocks, clear trend patterns",
    "Statistical": "Best for blue chip stocks, utilities, stable dividend stocks",
    "Tree-Ensemble": "Best for stocks with complex relationships to market factors",
    "Balanced": "Best for general purpose, unknown stock characteristics",
    "Volatility-Focused": "Best for small cap stocks, emerging market stocks, crypto-related stocks"
}

col1, col2 = st.columns([2, 1])

try:
    # Fetch data
    df = fetch_stock_data(symbol, display_days)

    # Get current live price
    current_price_data = get_current_price(symbol)

    # Display stock name and current price in big text
    if not df.empty:
        if current_price_data is not None:
            # Use live price if available
            last_price = current_price_data["price"]
            last_date = current_price_data["last_updated"]
            price_label = "LIVE" if current_price_data["is_live"] else "LAST CLOSE"
            price_color = "#0f9d58" if current_price_data["is_live"] else "#1E88E5"
        else:
            # Fallback to historical data
            last_price = float(df['Close'].iloc[-1])
            last_date = df.index[-1].strftime('%Y-%m-%d')
            price_label = "LAST CLOSE"
            price_color = "#1E88E5"

        st.markdown(f"""
        <div style="display: flex; align-items: baseline; margin-bottom: 20px;">
```

```
        <h2 style="margin-right: 15px;">{symbol}</h2>
        <h1 style="color: {price_color}; margin: 0;">${last_price:.2f}</h1>
        <div style="margin-left: 10px;">
            <span style="color: gray; font-size: 14px;">{price_label}</span>
            <p style="color: gray; margin: 0; font-size: 14px;">as of {last_date}</p>
        </div>
    </div>
    """, unsafe_allow_html=True)

# Display stock price chart
st.subheader("Stock Price History")
try:
    # Create a new DataFrame specifically for plotting
    plot_data = pd.DataFrame(index=df.index)

    # Add the Close price data
    plot_data['Close'] = df['Close'].values

    # Calculate and add SMA values if we have enough data
    if len(df) >= 20:
        plot_data['SMA_20'] = df['Close'].rolling(window=20).mean().values
    if len(df) >= 50:
        plot_data['SMA_50'] = df['Close'].rolling(window=50).mean().values

    # Add forecast days control under the chart controls
    st.write("#### Chart Controls")
    toggle_col1, toggle_col2, toggle_col3, toggle_col4, forecast_col = st.columns(5)

    with toggle_col1:
        show_sma20 = st.checkbox("Show 20-Day SMA", value=True)

    with toggle_col2:
        show_sma50 = st.checkbox("Show 50-Day SMA", value=True)
```

```
with toggle_col3:
    show_bb = st.checkbox("Show Bollinger Bands", value=False)

with toggle_col4:
    show_indicators = st.checkbox("Show RSI/MACD", value=False)

with forecast_col:
    forecast_days = st.slider("Forecast Horizon (Days)", min_value=7, max_value=365, value=30, step=1)

# Generate forecast with user-selected horizon
with st.spinner("Generating forecast..."):
    forecast = forecast_with_prophet(df, forecast_days=forecast_days)

    # Calculate Bollinger Bands
    if show_bb and len(df) >= 20:
        ma20 = df['Close'].rolling(window=20).mean()
        std20 = df['Close'].rolling(window=20).std()
        df['BB_upper'] = ma20 + (std20 * 2)
        df['BB_lower'] = ma20- (std20 * 2)
        df['BB_middle'] = ma20

    # Calculate RSI and MACD if needed
    if show_indicators:
        # Calculate RSI
        delta = df['Close'].diff()
        gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
        loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
        rs = gain / loss
        df['RSI'] = 100- (100 / (1 + rs))

        # Calculate MACD
        df['EMA12'] = df['Close'].ewm(span=12, adjust=False).mean()
```

```
    df['EMA26'] = df['Close'].ewm(span=26, adjust=False).mean()
    df['MACD'] = df['EMA12']- df['EMA26']
    df['Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()

# Create plotly figure
if show_indicators:
    # Create subplot with price and indicators
    fig = make_subplots(rows=3, cols=1,
                        shared_xaxes=True,
                        vertical_spacing=0.05,
                        row_heights=[0.6, 0.2, 0.2],
                        specs=[[{"secondary_y": False}],
                               [{"secondary_y": False}],
                               [{"secondary_y": False}]])
else:
    fig = make_subplots(specs=[[{"secondary_y": False}]])

# Add Close price (always shown)
fig.add_trace(
    go.Scatter(x=plot_data.index, y=plot_data['Close'], name="Close Price", line=dict(color="blue"))
)

# Add SMA lines based on toggle state
if 'SMA_20' in plot_data.columns and show_sma20:
    fig.add_trace(
        go.Scatter(x=plot_data.index, y=plot_data['SMA_20'], name="20-Day SMA", line=dict(color="orange"))
    )
if 'SMA_50' in plot_data.columns and show_sma50:
    fig.add_trace(
        go.Scatter(x=plot_data.index, y=plot_data['SMA_50'], name="50-Day SMA", line=dict(color="green"))
    )

# Add Bollinger Bands if enabled
```

```
        if show_bb and 'BB_upper' in df.columns:
            fig.add_trace(
                go.Scatter(x=df.index, y=df['BB_upper'], name="BB Upper", line=dict(color="purple", width=1,
dash='dash'))
            )
            fig.add_trace(
                go.Scatter(x=df.index, y=df['BB_lower'], name="BB Lower",
                           line=dict(color="purple", width=1, dash='dash'),
                           fill='tonexty', fillcolor='rgba(128, 0, 128, 0.1)')
            )

        # Add RSI and MACD if enabled
        if show_indicators and 'RSI' in df.columns and 'MACD' in df.columns:
            # Add RSI trace to second subplot
            fig.add_trace(
                go.Scatter(x=df.index, y=df['RSI'], name="RSI", line=dict(color="orange")),
                row=2, col=1
            )

            # Add reference lines for RSI
            fig.add_trace(
                go.Scatter(x=[df.index[0], df.index[-1]], y=[70, 70],
                           name="Overbought", line=dict(color="red", width=1, dash='dash'),
                           showlegend=False),
                row=2, col=1
            )

            fig.add_trace(
                go.Scatter(x=[df.index[0], df.index[-1]], y=[30, 30],
                           name="Oversold", line=dict(color="green", width=1, dash='dash'),
                           showlegend=False),
                row=2, col=1
            )
```

```
    # Add MACD traces to third subplot
    fig.add_trace(
        go.Scatter(x=df.index, y=df['MACD'], name="MACD", line=dict(color="blue")),
        row=3, col=1
    )

    fig.add_trace(
        go.Scatter(x=df.index, y=df['Signal'], name="Signal", line=dict(color="red")),
        row=3, col=1
    )

    # Add MACD histogram
    fig.add_trace(
        go.Bar(x=df.index, y=df['MACD']-df['Signal'], name="Histogram",
               marker=dict(color='rgba(0,0,255,0.5)')),
        row=3, col=1
    )

# Always add forecast if valid
if forecast is not None and len(forecast) > 0:
    try:
        # Add Prophet forecast
        forecast_dates = pd.to_datetime(forecast['ds'])
        historical_dates = plot_data.index
        last_date = historical_dates[-1]

        # Create a boolean mask for future dates
        future_mask = forecast_dates > last_date

        # Only proceed if we have future dates
        if any(future_mask):
            # Extract forecast values and convert to lists to avoid indexing issues
```

```
forecast_x = forecast_dates[future_mask].tolist()
forecast_y = forecast['yhat'][future_mask].tolist()
forecast_upper = forecast['yhat_upper'][future_mask].tolist()
forecast_lower = forecast['yhat_lower'][future_mask].tolist()

# Add the forecast line
fig.add_trace(
    go.Scatter(
        x=forecast_x,
        y=forecast_y,
        name="Price Forecast",
        line=dict(color="red", dash="dash")
    )
)

# Add confidence interval
fig.add_trace(
    go.Scatter(
        x=forecast_x,
        y=forecast_upper,
        name="Upper Bound",
        line=dict(width=0),
        showlegend=False
    )
)
fig.add_trace(
    go.Scatter(
        x=forecast_x,
        y=forecast_lower,
        name="Lower Bound",
        fill='tonexty',
        fillcolor='rgba(255, 0, 0, 0.1)',
        line=dict(width=0),
```

```
                showlegend=False
            )
        )
    except Exception as forecast_trace_err:
        st.warning(f"Could not add forecast to chart: {str(forecast_trace_err)}")

# Update layout for both chart types
title = f"{symbol} Stock Price with Forecast"
if show_indicators:
    # Add titles for subplots
    fig.update_yaxes(title_text="Price ($)", row=1, col=1)
    fig.update_yaxes(title_text="RSI", row=2, col=1)
    fig.update_yaxes(title_text="MACD", row=3, col=1)

    fig.update_layout(
        title=title,
        xaxis_title="Date",
        hovermode="x unified",
        legend=dict(y=0.99, x=0.01, orientation="h"),
        template="plotly_white",
        autosize=True,
        height=700,  # Increase height for multiple subplots
        margin=dict(l=50, r=50, t=80, b=50),
        xaxis=dict(
            autorange=True,
            rangeslider=dict(visible=False)
        ),
        yaxis=dict(
            autorange=True,
            fixedrange=False
        ),
        dragmode='pan'
    )
```

```
else:
    fig.update_layout(
        title=title,
        xaxis_title="Date",
        yaxis_title="Price ($)",
        hovermode="x unified",
        legend=dict(y=0.99, x=0.01, orientation="h"),
        template="plotly_white",
        autosize=True,
        height=500,
        margin=dict(l=50, r=50, t=80, b=50),
        xaxis=dict(
            autorange=True,
            rangeslider=dict(visible=False)
        ),
        yaxis=dict(
            autorange=True,
            fixedrange=False
        ),
        dragmode='pan'
    )

# Update the chart configuration to fix zoom toggle issues
st.plotly_chart(fig, use_container_width=True, config={
    'displayModeBar': True,
    'scrollZoom': True,
    'displaylogo': False,
    # Don't remove zoom buttons, but add a reset view button and make toggle possible
    'modeBarButtonsToRemove': ['autoScale2d', 'select2d', 'lasso2d'],
    'modeBarButtonsToAdd': ['resetScale2d', 'toImage'],
    'dragmode': 'pan'
})
```

```
# Display forecast metrics
with st.expander("Prophet Forecast Details"):
    # Get last historical date and first forecast date
    if forecast is not None and len(forecast) > 0:
        next_date_mask = forecast_dates > last_date

        if any(next_date_mask):
            next_date_idx = next_date_mask.argmax()
            last_close_price = float(plot_data['Close'].iloc[-1])

            # Calculate short-term forecast (7 days)
            short_term_idx = min(next_date_idx + 7, len(forecast)- 1)
            short_term_price = float(forecast['yhat'].iloc[short_term_idx])
            short_term_change = (short_term_price- last_close_price) / last_close_price * 100

            # Calculate medium-term forecast (30 days)
            medium_term_idx = min(next_date_idx + 30, len(forecast)- 1)
            medium_term_price = float(forecast['yhat'].iloc[medium_term_idx])
            medium_term_change = (medium_term_price- last_close_price) / last_close_price * 100

            # Calculate long-term forecast (90 days)
            long_term_idx = min(next_date_idx + 90, len(forecast)- 1)
            long_term_price = float(forecast['yhat'].iloc[long_term_idx])
            long_term_change = (long_term_price- last_close_price) / last_close_price * 100

            # Create metrics with 3 columns for different timeframes
            col1, col2, col3 = st.columns(3)
            with col1:
                st.metric(
                    label="7-Day Forecast",
                    value=f"${short_term_price:.2f}",
                    delta=f"{short_term_change:.2f}%"
                )
```

```
with col2:
    st.metric(
        label="30-Day Forecast",
        value=f"${medium_term_price:.2f}",
        delta=f"{medium_term_change:.2f}%"
    )
with col3:
    st.metric(
        label="90-Day Forecast",
        value=f"${long_term_price:.2f}",
        delta=f"{long_term_change:.2f}%"
    )

# Display trend and seasonality info
st.write("#### Forecast Components")
st.write("Prophet identifies the following patterns in the data:")

try:
    # Get components for analysis
    trend_values = forecast['trend'][next_date_idx:medium_term_idx].values

    # Check if we have weekly component (we disabled it, but check just in case)
    has_weekly_component = 'weekly' in forecast.columns and not all(forecast['weekly'] == 0)

    # Check if we have yearly component
    has_yearly_component = 'yearly' in forecast.columns and not all(forecast['yearly'] == 0)

    # Determine trend direction
    trend_direction = "Upward" if np.mean(np.diff(trend_values)) > 0 else "Downward"
    trend_strength = np.abs(np.mean(np.diff(trend_values))/np.mean(trend_values)*100)

    # Create a detailed insights section for trend analysis
    st.markdown(f"""
```

```
**Trend Analysis:**
- Direction: {trend_direction} ({trend_strength:.2f}% per period)
- Strength: {"Strong" if trend_strength > 0.5 else "Moderate" if trend_strength > 0.1 else "Weak"}
""")

# Only show weekly patterns if weekly component exists
if has_weekly_component:
    weekly_values = forecast['weekly'][next_date_idx:medium_term_idx].values
    # Find day with maximum weekly effect
    forecast_subset = forecast.iloc[next_date_idx:medium_term_idx]
    max_weekly_idx = forecast_subset['weekly'].idxmax()
    min_weekly_idx = forecast_subset['weekly'].idxmin()
    max_weekly_day = pd.to_datetime(forecast_subset.loc[max_weekly_idx, 'ds']).strftime('%A')
    min_weekly_day = pd.to_datetime(forecast_subset.loc[min_weekly_idx, 'ds']).strftime('%A')

    st.markdown(f"""
    **Weekly Patterns:**
    - Most positive day: {max_weekly_day}
    - Most negative day: {min_weekly_day}
    """)
else:
    # No weekly component was used (correctly disabled for stock prices)
    st.markdown("""
    **Weekly Patterns:**
    - None detected (weekly seasonality disabled for stock market data)
    - Stock markets are closed on weekends, so no trading patterns exist
    """)

# Only show yearly patterns if yearly component exists
if has_yearly_component:
    yearly_values = forecast['yearly'][next_date_idx:medium_term_idx].values
    # Determine seasonal factor
    seasonal_factor = "Positive" if np.mean(yearly_values) > 0 else "Negative"
```

```
                current_month = datetime.now().strftime('%B')
                next_month = (datetime.now() + timedelta(days=30)).strftime('%B')

                st.markdown(f"""
                **Seasonal Analysis:**
                - Current seasonal effect: {seasonal_factor}
                - Current month ({current_month}): {"Favorable" if np.mean(yearly_values) > 0 else "Unfavorable"} historically
                - Next month ({next_month}): {"Likely favorable" if np.mean(yearly_values[15:]) > 0 else "Likely unfavorable"} based on patterns
                """)
            else:
                # No yearly component or not enough data
                st.markdown("""
                **Seasonal Analysis:**
                - No significant yearly patterns detected
                - Not enough historical data for reliable yearly seasonality detection
                """)

            # Add trading insights based on forecast
            st.subheader("Forecast-Based Trading Insights")

            # Calculate volatility as the standard deviation of forecast values
            forecast_volatility = np.std(forecast['yhat'][next_date_idx:medium_term_idx])/np.mean(forecast['yhat'][next_date_idx:medium_term_idx])

            # Calculate momentum (rate of change over forecast period)
            momentum = (medium_term_price- last_close_price)/last_close_price

            # Calculate confidence as inverse of the width of prediction intervals
            confidence = 1- np.mean((forecast['yhat_upper']- forecast['yhat_lower'])/forecast['yhat'])

            # Create trading signals based on multiple factors
            signal_strength = abs(medium_term_change)
```

```
signal_confidence = confidence*100

signal_col1, signal_col2 = st.columns(2)
with signal_col1:
    if medium_term_change > 10:
        st.success("🚀 Strong Buy Signal")
    elif medium_term_change > 5:
        st.success("✅ Buy Signal")
    elif medium_term_change > 2:
        st.info("📈 Weak Buy Signal")
    elif medium_term_change <-10:
        st.error("🔻 Strong Sell Signal")
    elif medium_term_change <-5:
        st.error("📉 Sell Signal")
    elif medium_term_change <-2:
        st.warning("📉 Weak Sell Signal")
    else:
        st.info("⚖️ Hold/Neutral Signal")

with signal_col2:
    st.metric("Signal Strength", f"{signal_strength:.1f}/10",
              delta=f"{signal_confidence:.0f}% confidence")

# Add forecast-based scenarios
st.subheader("Possible Scenarios")
scenario_col1, scenario_col2, scenario_col3 = st.columns(3)

with scenario_col1:
    st.markdown(f"""
    **Bullish Case:**
    - Target: ${forecast['yhat_upper'].iloc[medium_term_idx]:.2f}
    - Gain: {((forecast['yhat_upper'].iloc[medium_term_idx]-
last_close_price)/last_close_price*100):.1f}%
```

```
                - Probability: {(confidence * (1 + medium_term_change/100) * 100):.0f}%
                """)

            with scenario_col2:
                st.markdown(f"""
                **Base Case:**
                - Target: ${medium_term_price:.2f}
                - Change: {medium_term_change:.1f}%
                - Probability: {(confidence * 100):.0f}%
                """)

            with scenario_col3:
                st.markdown(f"""
                **Bearish Case:**
                - Target: ${forecast['yhat_lower'].iloc[medium_term_idx]:.2f}
                - Loss: {((forecast['yhat_lower'].iloc[medium_term_idx]-
last_close_price)/last_close_price*100):.1f}%
                - Probability: {(confidence * (1- medium_term_change/100) * 100):.0f}%
                """)

        except Exception as component_err:
            st.warning(f"Could not analyze forecast components: {str(component_err)}")

except Exception as e:
    st.error(f"Error creating enhanced chart: {str(e)}")
    # Fall back to simple chart
    try:
        st.line_chart(df['Close'])
    except:
        st.error("Unable to display chart. Please check your data.")

col1, col2 = st.columns([1, 1])
```

```
    with col1:
        # First, add a subheader for the prediction section
        st.subheader("Model Configuration & Predictions")

        # Add the weight configuration selector and description
        selected_weight = st.selectbox(
            "Select Model Configuration:",
            options=list(WEIGHT_CONFIGURATIONS.keys()),
            help="Choose different weight configurations for the prediction models. This affects the predictions generated by the 'Generate Predictions' button."
        )

        # Show the description in an info box
        st.info(WEIGHT_DESCRIPTIONS[selected_weight])

        # Add some space
        st.write("")

        # Then add the Generate Predictions button
        if st.button("Generate Predictions"):
            with st.spinner("Training multiple models and generating predictions..."):
                predictor = MultiAlgorithmStockPredictor(
                    symbol,
                    weights=WEIGHT_CONFIGURATIONS[selected_weight]
                )
                results = predictor.predict_with_all_models(prediction_days=30)

                if results is not None:
                    # Calculate target date here since it's not in results
                    target_date = datetime.now() + timedelta(days=30)
                    st.write(f"#### Predictions for {target_date.strftime('%B %d, %Y')}")

                    last_price = float(df['Close'].iloc[-1])
```

```
# Individual model predictions
st.subheader("Individual Model Predictions")
model_predictions = pd.DataFrame({
    'Model': results['individual_predictions'].keys(),
    'Predicted Price': [v for v in results['individual_predictions'].values()],
    'Target Date': target_date.strftime('%Y-%m-%d')  # Add target date to DataFrame
})
model_predictions['Deviation from Ensemble'] = (
    model_predictions['Predicted Price']- abs(results['prediction'])
)
model_predictions = model_predictions.sort_values('Predicted Price', ascending=False)
st.dataframe(model_predictions.style.format({
    'Predicted Price': '${:.2f}',
    'Deviation from Ensemble': '${:.2f}'
}))

# Trading signal with confidence
price_change = ((results['prediction']- last_price) / last_price) * 100

# Create a prediction distribution plot
fig, ax = plt.subplots(figsize=(10, 6))
predictions = list(results['individual_predictions'].values())
models = list(results['individual_predictions'].keys())

# Horizontal bar chart showing predictions
y_pos = np.arange(len(models))
ax.barh(y_pos, predictions)
ax.set_yticks(y_pos)
ax.set_yticklabels(models)
ax.axvline(x=last_price, color='r', linestyle='--', label='Current Price')
ax.axvline(x=results['prediction'], color='g', linestyle='--', label='Ensemble Prediction')
ax.set_xlabel('Price ($)')
```

```
ax.set_title('Model Predictions Comparison')
ax.legend()

st.pyplot(fig)

# Trading signal box
signal_box = st.container()
if abs(price_change) > 10:  # For very large changes
    if price_change > 0:
        signal_box.success(f"✅ Strong BUY Signal (+{price_change:.1f}%)")
    else:
        signal_box.error(f"📉 Strong SELL Signal ({price_change:.1f}%)")
elif abs(price_change) > 3 and results['confidence_score'] > 0.8:
    if price_change > 0:
        signal_box.success(f"✅ BUY Signal (+{price_change:.1f}%)")
    else:
        signal_box.error(f"📉 SELL Signal ({price_change:.1f}%)")
elif abs(price_change) > 2 and results['confidence_score'] > 0.6:
    if price_change > 0:
        signal_box.warning(f"📈 Moderate BUY Signal (+{price_change:.1f}%)")
    else:
        signal_box.warning(f"📉 Moderate SELL Signal ({price_change:.1f}%)")
else:
    if abs(price_change) < 1:
        signal_box.info(f"⚖️ HOLD Signal ({price_change:.1f}%)")
    else:
        if price_change > 0:
            signal_box.info(f"📈 Weak BUY Signal (+{price_change:.1f}%)")
        else:
            signal_box.info(f"📉 Weak SELL Signal ({price_change:.1f}%)")

# Model consensus analysis
```

```
            st.subheader("Model Consensus Analysis")
            buy_signals = sum(1 for pred in predictions if pred > last_price)
            sell_signals = sum(1 for pred in predictions if pred < last_price)
            total_models = len(predictions)

            consensus_col1, consensus_col2, consensus_col3 = st.columns(3)
            with consensus_col1:
               st.metric("Buy Signals", f"{buy_signals}/{total_models}")
            with consensus_col2:
               st.metric("Sell Signals", f"{sell_signals}/{total_models}")
            with consensus_col3:
               consensus_strength = abs(buy_signals- sell_signals) / total_models
               st.metric("Consensus Strength", f"{consensus_strength:.1%}")

            # Risk assessment
            st.subheader("Risk Assessment")
            prediction_std = np.std(predictions)
            prediction_range = results['upper_bound']- results['lower_bound']
            risk_level = "Low" if prediction_std < last_price * 0.02 else \
                    "Medium" if prediction_std < last_price * 0.05 else "High"

            risk_col1, risk_col2 = st.columns(2)
            with risk_col1:
               st.metric("Prediction Volatility", f"${prediction_std:.2f}")
            with risk_col2:
               st.metric("Risk Level", risk_level)

with col2:
   st.subheader("Latest News & Market Sentiment")
   try:
      news_headlines = get_news_headlines(symbol)

      if news_headlines and len(news_headlines) > 0:
```

```
# Initialize sentiment tracking
sentiment_scores = []
sentiment_weights = []

for title, description, url in news_headlines:
    # Ensure title and description are strings
    title = str(title) if title else ""
    description = str(description) if description else ""

    # Analyze both title and description with different weights
    title_analysis = analyze_sentiment(title)
    desc_analysis = analyze_sentiment(description)

    # Combined analysis (title has more weight)
    combined_score = title_analysis['score'] * 0.6 + desc_analysis['score'] * 0.4
    sentiment_scores.append(combined_score)

    # Weight more recent news higher
    sentiment_weights.append(1.0)

    # Determine display properties
    if combined_score >= 0.2:
        sentiment = "📈 Positive"
        color = "green"
        confidence = min(abs(combined_score) * 100, 100)
    elif combined_score <=-0.2:
        sentiment = "📉 Negative"
        color = "red"
        confidence = min(abs(combined_score) * 100, 100)
    else:
        sentiment = "⚖️ Neutral"
        color = "gray"
        confidence = (1- abs(combined_score)) * 100
```

```
            with st.expander(f"{title} ({sentiment})"):
                st.write(description)
                st.markdown(f"[Read full article]({url})")
                st.markdown(
                    f"<span style='color: {color}'>Sentiment: {sentiment} "
                    f"(Confidence: {confidence:.1f}%)</span>",
                    unsafe_allow_html=True
                )

        # Calculate weighted average sentiment
        total_weight = sum(sentiment_weights)
        weighted_sentiment = sum(score * weight for score, weight in zip(sentiment_scores, sentiment_weights)) /
total_weight

        # Display overall sentiment consensus
        st.write("### News Sentiment Consensus")

        # Calculate sentiment distribution
        positive_scores = sum(1 for score in sentiment_scores if score >= 0.2)
        negative_scores = sum(1 for score in sentiment_scores if score <=-0.2)
        neutral_scores = len(sentiment_scores)- positive_scores- negative_scores

        # Create metrics columns
        consensus_col1, consensus_col2, consensus_col3 = st.columns(3)
        total_articles = len(sentiment_scores)

        with consensus_col1:
            pos_pct = (positive_scores / total_articles) * 100
            st.metric("Positive News",
                      f"{positive_scores}/{total_articles}",
                      f"{pos_pct:.1f}%")
```

```
with consensus_col2:
    neg_pct = (negative_scores / total_articles) * 100
    st.metric("Negative News",
              f"{negative_scores}/{total_articles}",
              f"{neg_pct:.1f}%")

with consensus_col3:
    neu_pct = (neutral_scores / total_articles) * 100
    st.metric("Neutral News",
              f"{neutral_scores}/{total_articles}",
              f"{neu_pct:.1f}%")

# Overall sentiment conclusion with confidence
sentiment_strength = abs(weighted_sentiment)
confidence = min(sentiment_strength * 100, 100)

if weighted_sentiment >= 0.2:
    st.success(
        f"📈 Strong Bullish Sentiment "
        f"(Confidence: {confidence:.1f}%)\n\n"
        f"Market news suggests positive momentum with {positive_scores} supportive articles."
    )
elif weighted_sentiment >= 0.1:
    st.success(
        f"📈 Moderately Bullish Sentiment "
        f"(Confidence: {confidence:.1f}%)\n\n"
        f"Market news leans positive with mixed signals."
    )
elif weighted_sentiment <=-0.2:
    st.error(
        f"📉 Strong Bearish Sentiment "
        f"(Confidence: {confidence:.1f}%)\n\n"
        f"Market news suggests negative pressure with {negative_scores} concerning articles."
```

```
            )
        elif weighted_sentiment <=-0.1:
            st.error(
                f"📉 Moderately Bearish Sentiment "
                f"(Confidence: {confidence:.1f}%)\n\n"
                f"Market news leans negative with mixed signals."
            )
        else:
            st.info(
                f"⚖️ Neutral Market Sentiment "
                f"(Confidence: {(1- sentiment_strength) * 100:.1f}%)\n\n"
                f"Market news shows balanced or unclear direction."
            )

    else:
        st.info("No recent news available for this stock.")

except Exception as e:
    st.error(f"Error fetching news: {str(e)}")
    st.info("No recent news available for this stock.")

# Technical Analysis Summary
st.subheader("Technical Analysis Summary")
try:
    # Check if dataframe exists and has data
    if 'df' in locals() and isinstance(df, pd.DataFrame) and len(df) > 0:
        # Calculate technical indicators from historical data
        analysis_df = calculate_technical_indicators_for_summary(df)

        if len(analysis_df) >= 2:
            latest = analysis_df.iloc[-1]
            prev = analysis_df.iloc[-2]
```

```
# Historical Data Analysis
st.write("📊 Historical Data Analysis")

# Calculate indicator values first to avoid Series truth value ambiguity
ma_bullish = float(latest['MA20']) > float(latest['MA50'])
rsi_value = float(latest['RSI'])
volume_high = float(latest['Volume']) > float(latest['Volume_MA'])
close_price = float(latest['Close'])
bb_upper = float(latest['BB_upper'])
bb_lower = float(latest['BB_lower'])

# Historical indicators
historical_indicators = {
    "Moving Averages": {
        "value": "Bullish" if ma_bullish else "Bearish",
        "delta": f"{((float(latest['MA20'])- float(latest['MA50']))/float(latest['MA50']) * 100):.1f}% spread",
        "description": "Based on 20 & 50-day moving averages"
    },
    "RSI (14)": {
        "value": "Overbought" if rsi_value > 70 else "Oversold" if rsi_value < 30 else "Neutral",
        "delta": f"{rsi_value:.1f}",
        "description": "Current RSI value"
    },
    "Volume Trend": {
        "value": "Above Average" if volume_high else "Below Average",
        "delta": f"{((float(latest['Volume'])- float(latest['Volume_MA']))/float(latest['Volume_MA']) *
100):.1f}%",
        "description": "Compared to 20-day average"
    },
    "Bollinger Bands": {
        "value": "Upper Band" if close_price > bb_upper else
            "Lower Band" if close_price < bb_lower else "Middle Band",
        "delta": f"{((close_price- bb_lower)/(bb_upper- bb_lower) * 100):.1f}%",
```

```
            "description": "Position within bands"
        }
    }

    # Display historical indicators
    for indicator, data in historical_indicators.items():
        with st.expander(f"{indicator}: {data['value']}"):
            st.metric(
                label=data['description'],
                value=data['value'],
                delta=data['delta']
            )

    # Model Predictions Analysis
    if 'results' in locals() and results is not None:
        st.write("👹 Model Predictions Analysis")

        # Calculate prediction metrics
        current_price = float(df['Close'].iloc[-1])
        pred_price = float(results['prediction'])
        price_change_pct = ((pred_price- current_price) / current_price) * 100
        predictions = results['individual_predictions']
        bullish_models = sum(1 for p in predictions.values() if p > current_price)
        bearish_models = len(predictions)- bullish_models

        prediction_indicators = {
            "Price Prediction": {
                "value": f"${pred_price:.2f}",
                "delta": f"{price_change_pct:+.1f}% from current",
                "description": "Ensemble model prediction"
            },
            "Model Consensus": {
                "value": f"{bullish_models}/{len(predictions)} Bullish",
```

```
            "delta": f"{(bullish_models/len(predictions)*100):.0f}% agreement",
            "description": "Agreement among models"
        },
        "Prediction Range": {
            "value": f"${abs(results['lower_bound']):.2f}- ${abs(results['upper_bound']):.2f}",
            "delta": f"±{((results['upper_bound']- results['lower_bound'])/2/pred_price*100):.1f}%",
            "description": "Expected price range"
        },
        "Confidence Score": {
            "value": f"{results['confidence_score']:.1%}",
            "delta": "Based on model agreement",
            "description": "Overall prediction confidence"
        }
    }

    # Display prediction indicators
    for indicator, data in prediction_indicators.items():
        with st.expander(f"{indicator}: {data['value']}"):
            st.metric(
                label=data['description'],
                value=data['value'],
                delta=data['delta']
            )

    # Overall Analysis
    st.write("📈 Combined Signal Analysis")

    # Get trading signal strength based on price_change
    def get_trading_signal_strength(price_change, confidence_score):
        if abs(price_change) > 10:
            return "strong_buy" if price_change > 0 else "strong_sell"
        elif abs(price_change) > 3 and confidence_score > 0.8:
            return "buy" if price_change > 0 else "sell"
```

```
            elif abs(price_change) > 2 and confidence_score > 0.6:
                return "moderate_buy" if price_change > 0 else "moderate_sell"
            elif abs(price_change) < 1:
                return "hold"
            else:
                return "weak_buy" if price_change > 0 else "weak_sell"

        # Get signals from different sources
        technical_bullish = ma_bullish
        trading_signal = get_trading_signal_strength(price_change_pct, results['confidence_score'])
        model_confidence = results['confidence_score'] > 0.6

        # Determine overall signal
        if technical_bullish and trading_signal in ['strong_buy', 'buy']:
            st.success("🚀 Very Strong Buy Signal: Technical analysis is bullish and models show strong upward momentum")
        elif technical_bullish and trading_signal in ['moderate_buy', 'weak_buy']:
            st.success("✅ Strong Buy Signal: Technical analysis is bullish with moderate model support")
        elif not technical_bullish and trading_signal in ['strong_buy', 'buy']:
            st.warning("📈 Cautious Buy Signal: Models show strong upward potential but technical indicators suggest caution")
        elif technical_bullish and trading_signal in ['hold']:
            st.info("⚖️ Hold with Bullish Bias: Technical analysis is positive but models suggest consolidation")
        elif not technical_bullish and trading_signal in ['hold']:
            st.info("⚖️ Hold with Bearish Bias: Technical analysis is negative and models suggest consolidation")
        elif technical_bullish and trading_signal in ['weak_sell', 'moderate_sell']:
            st.warning("🤔 Mixed Signal: Technical analysis is bullish but models show weakness")
        elif not technical_bullish and trading_signal in ['weak_sell', 'moderate_sell']:
            st.error("📉 Strong Sell Signal: Both technical analysis and models show weakness")
        elif not technical_bullish and trading_signal in ['strong_sell', 'sell']:
            st.error("🔻 Very Strong Sell Signal: Technical analysis is bearish and models show strong downward momentum")
        else:
```

```
                st.warning("🔄 Mixed Signals: Conflicting indicators suggest caution")

            # Display confidence metrics
            confidence_text = "High" if model_confidence else "Low"
            st.info(f"Model Prediction Confidence: {confidence_text}")

            # Additional context based on signals
            if model_confidence:
                if technical_bullish:
                    st.write("💡 Technical indicators support the model predictions, suggesting higher reliability")
                else:
                    st.write("💡 Technical indicators contrast with model predictions, suggesting careful monitoring")
            else:
                st.write("💡 Lower model confidence suggests waiting for clearer signals before making decisions")

        else:
            st.warning("Insufficient data points for technical analysis. Please ensure you have at least 50 days of historical data.")
    else:
        st.warning("No data available for technical analysis. Please enter a valid stock symbol.")

except Exception as e:
    st.error(f"Error in Technical Analysis: {str(e)}")
    st.write("Detailed error information:", str(e))
except Exception as e:
    st.error(f"Error: {str(e)}")
    st.write("Detailed error information:", str(e))
```

4.4 Code Explanation

Module 1: Imports & Setup

Code :-

import pandas as pd

```
import numpy as np
import streamlit as st
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from statsmodels.tsa.arima.model import ARIMA
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from newsapi import NewsApiClient
import yfinance as yf
from prophet import Prophet
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.linear_model import LinearRegression
from textblob import TextBlob
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import re

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

try:
    nltk.data.find('vader_lexicon')
except LookupError:
    nltk.download('vader_lexicon')
    nltk.download('punkt')
    st.set_page_config(page_title="Multi-Algorithm Stock Predictor", layout="wide")
```

- **Libraries:** pandas, numpy, streamlit, matplotlib, scikit-learn, xgboost, tensorflow, statsmodels, prophet, plotly, nltk, textblob, re, etc.
- **Logging config:** Suppresses TensorFlow v1 logging.
- **NLTK Data Check:** Ensures VADER and Punkt tokenizer are downloaded.
- **Streamlit Page Config:** Sets the web app title and layout.
- **Disclaimer:** Displays a disclaimer about stock prediction risks.

Module 2: API & Data Fetching Functions

Code:-

```
NEWS_API_KEY = '0de37ca8af9748898518daf699189abf'
newsapi = NewsApiClient(api_key=NEWS_API_KEY)


@st.cache_data(ttl=3600)
def fetch_stock_data(symbol, days):
    end_date = datetime.now()
    start_date = end_date - timedelta(days=days)
    df = yf.download(symbol, start=start_date, end=end_date)
    return df


@st.cache_data(ttl=3600)
def get_news_headlines(symbol):
    try:
        news = newsapi.get_everything(q=symbol, language='en', sort_by='relevancy', page_size=5)
        return [(article['title'], article['description'], article['url']) for article in news['articles']]
    except Exception as e:
        print(f"News API error: {str(e)}")
        return []


@st.cache_data(ttl=300)
def get_current_price(symbol):
    try:
        ticker = yf.Ticker(symbol)
```

```
    todays_data = ticker.history(period='1d')
    if todays_data.empty:
        return None
    if 'Open' in todays_data.columns and len(todays_data) > 0:
        if 'regularMarketPrice' in ticker.info:
            current_price = ticker.info['regularMarketPrice']
            is_live = True
        else:
            current_price = float(todays_data['Close'].iloc[-1])
            is_live = False
        last_updated = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        return {"price": current_price, "is_live": is_live, "last_updated": last_updated}
    return None

except Exception as e:
    st.error(f"Error fetching current price: {str(e)}")
    return None
```

- **fetch_stock_data()**: Fetches historical stock price data using yfinance.
- **get_news_headlines()**: Retrieves recent news related to a stock using News API.
- **get_current_price()**: Gets the real-time price or fallback to the last close.
- **analyze_sentiment()**: Performs sentiment analysis on text using VADER, TextBlob, and financial-specific keywords.

Module 3: Forecasting Logic

Code:-

```
@st.cache_data(ttl=3600)
def analyze_sentiment(text):
    if not text or not isinstance(text, str):
        return {'sentiment': "⚖️ Neutral", 'confidence': 0, 'color': "gray", 'score': 0}
    text = re.sub(r'[^\w\s]', '', text)
    sia = SentimentIntensityAnalyzer()
    vader_scores = sia.polarity_scores(text)
    blob = TextBlob(text)
```

```
    textblob_polarity = blob.sentiment.polarity
    combined_score = vader_scores['compound'] * 0.3 + textblob_polarity * 0.2
    if combined_score >= 0.15:
        sentiment = "📈 Positive"
        confidence = min(abs(combined_score) * 150, 100)
        color = "green"
    elif combined_score <= -0.15:
        sentiment = "📉 Negative"
        confidence = min(abs(combined_score) * 150, 100)
        color = "red"
    else:
        sentiment = "⚖️ Neutral"
        confidence = (1 - abs(combined_score)) * 100
        color = "gray"
    return {'sentiment': sentiment, 'confidence': confidence, 'color': color, 'score': combined_score}
```

- **forecast_with_prophet()**: Builds a customized Prophet model to forecast stock prices. Integrates technical indicators as regressors like RSI, Bollinger Bands, Volume, Momentum, etc.
- **simple_forecast_fallback()**: A simple linear regression fallback when Prophet fails.

Module 4: Technical Indicator Calculations

Code:-

```
@st.cache_data(ttl=3600)
def forecast_with_prophet(df, forecast_days=30):
    if len(df) < 30:
        st.warning("Not enough data, using fallback forecast")
        return simple_forecast_fallback(df, forecast_days)
    df = df.reset_index()
    df.rename(columns={"Date": "ds", "Close": "y"}, inplace=True)
    df['ds'] = pd.to_datetime(df['ds'])
    model = Prophet(daily_seasonality=True)
    model.fit(df)
    future = model.make_future_dataframe(periods=forecast_days)
```

```
    forecast = model.predict(future)
    return forecast


def simple_forecast_fallback(df, forecast_days=30):
    close_prices = df['Close'].values.flatten()
    x = np.arange(len(close_prices)).reshape(-1, 1)
    model = LinearRegression()
    model.fit(x, close_prices)
    future_x = np.arange(len(close_prices), len(close_prices) + forecast_days).reshape(-1, 1)
    future_y = model.predict(future_x)
    return pd.DataFrame({"ds": pd.date_range(start=df.index[-1] + timedelta(days=1), periods=forecast_days),
"yhat": future_y})
```

- **calculate_technical_indicators_for_summary()**: Calculates indicators like SMA, RSI, Bollinger Bands for quick chart overlays.
- **Within the class:** Includes methods for:
    - Moving Averages (MA5, MA20, MA50, MA200)
    - RSI, MACD
    - ATR, Bollinger Bands
    - ADX, Momentum, Stochastic Oscillator, Williams %R

Module 5: Multi-Algorithm Predictor Class (MultiAlgorithmStockPredictor)

Code:-

```
def calculate_indicators(df):
    df['MA20'] = df['Close'].rolling(window=20).mean()
    df['MA50'] = df['Close'].rolling(window=50).mean()
    delta = df['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))
    ma20 = df['Close'].rolling(window=20).mean()
    std20 = df['Close'].rolling(window=20).std()
```

```
df['BB_upper'] = ma20 + (std20 * 2)
df['BB_lower'] = ma20 - (std20 * 2)
return df
```

- **__init__()**: Initializes stock symbol, years of historical data to use, scaler, and weights.
- **fetch_historical_data()**: Downloads historical data based on input duration.
- **calculate_technical_indicators()**: Applies advanced technical indicators.
- **prepare_data()**: Scales features, engineers new ones (momentum, market regimes, volatility) & prepares sequences for LSTM.
- **train_arima()**: Auto-tunes ARIMA using pmdarima.auto_arima.
- **build_lstm_model()**: Defines a simplified LSTM neural network.
- **predict_with_all_models()**: Trains & predicts using:
    - LSTM
    - SVR
    - Random Forest
    - XGBoost
    - (optionally) Gradient Boosting, ARIMA Then it ensembles all predictions into a final output with weighted averaging.

Module 6: Streamlit UI

Code:-

```
class MultiAlgorithmStockPredictor:
    def __init__(self, symbol, years=2):
        self.symbol = symbol
        self.years = years
        self.scaler = MinMaxScaler()

    def fetch_historical_data(self):
        end_date = datetime.now()
        start_date = end_date - timedelta(days=365 * self.years)
        df = yf.download(self.symbol, start=start_date, end=end_date)
        return df
```

```
def prepare_data(self, df, seq_length=60):
    df = calculate_indicators(df)
    df = df.dropna()
    scaled_data = self.scaler.fit_transform(df[['Close', 'MA20', 'MA50', 'RSI']])
    X, y = [], []
    for i in range(seq_length, len(scaled_data)):
        X.append(scaled_data[i-seq_length:i])
        y.append(scaled_data[i, 0])
    return np.array(X), np.array(y)


def build_lstm(self, input_shape):
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(32),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='huber')
    return model
```

- **User Inputs:**
    - Symbol (e.g., "AAPL")
    - Days to display (slider)
    - Weighting configurations for ensemble (default, trend-focused, volatility-focused, etc.)
    - Forecast horizon (7 to 365 days)
- **Chart Controls:**
    - Toggle SMA, Bollinger Bands, RSI, MACD
- **Dynamic Dashboard:**
    - Displays:
        - Current price (live or last close)
        - Interactive price chart (Plotly)

- Forecast plots with confidence intervals
- RSI and MACD subplots if enabled
- Forecast-based trading insights and scenarios (bullish/base/bearish)

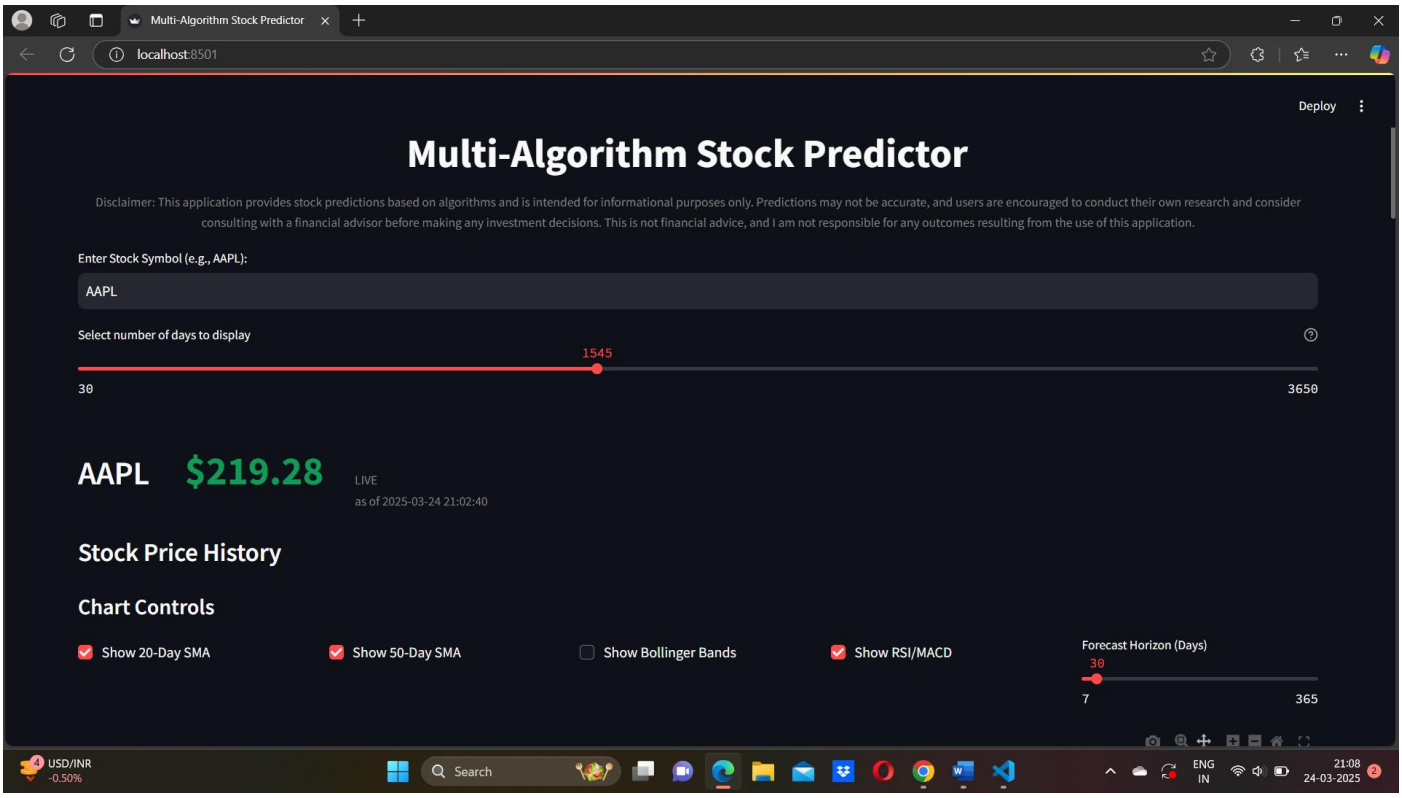
Module 7: Ensemble Weight Configurations

Code:-

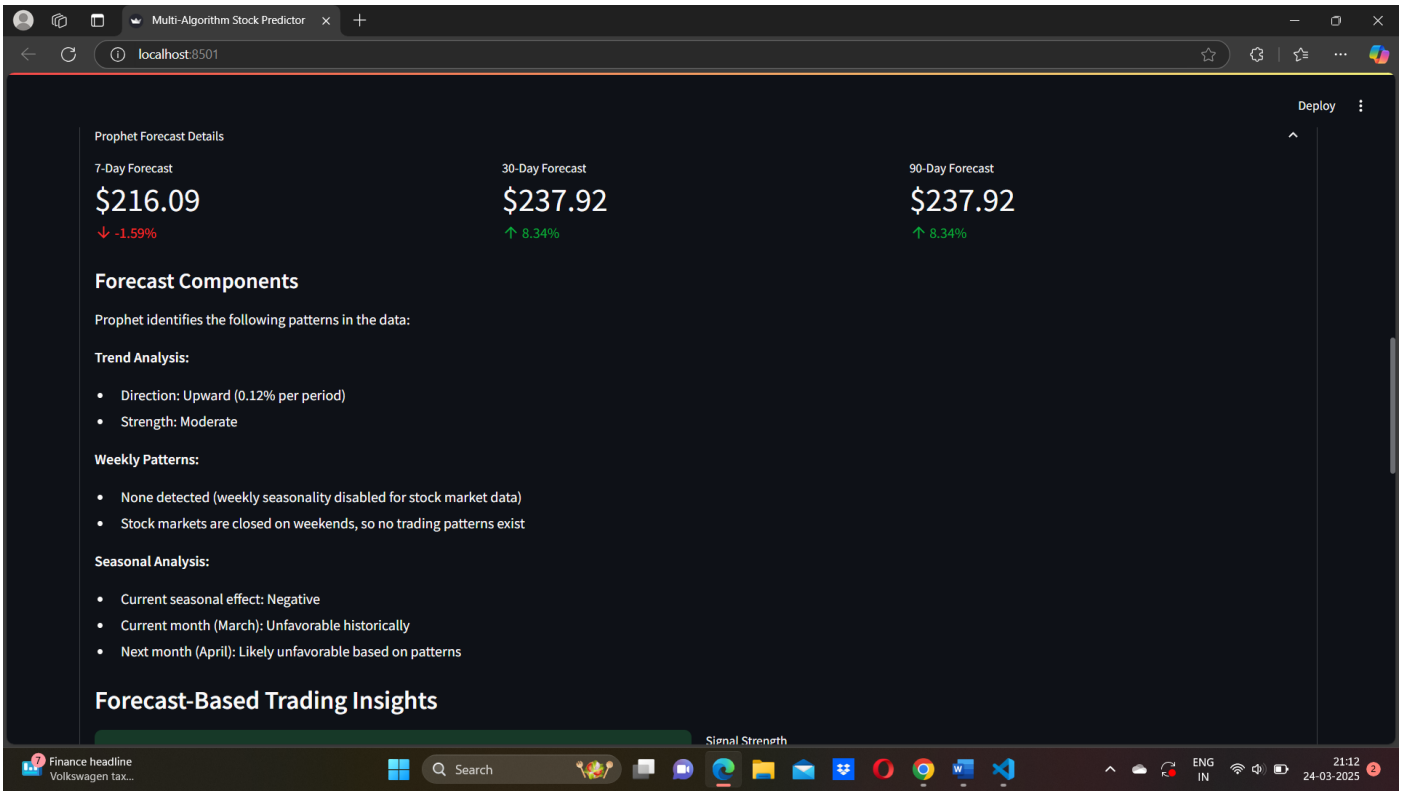
```
symbol = st.text_input("Enter Stock Symbol (e.g., AAPL):", "AAPL")
display_days = st.slider("Select number of days to display", min_value=30, max_value=3650, value=600)

if symbol:
    df = fetch_stock_data(symbol, display_days)
    df = calculate_indicators(df)
    st.subheader(f"{symbol} Price Chart")
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=df.index, y=df['Close'], name="Close Price"))
    fig.add_trace(go.Scatter(x=df.index, y=df['MA20'], name="20-Day SMA"))
    fig.add_trace(go.Scatter(x=df.index, y=df['MA50'], name="50-Day SMA"))
    st.plotly_chart(fig, use_container_width=True)

    st.subheader("Sentiment Analysis (Recent News)")
    headlines = get_news_headlines(symbol)
    for title, desc, url in headlines:
        sentiment = analyze_sentiment(title + ' ' + desc)
        st.markdown(f"- [{title}]({url}) - **{sentiment['sentiment']}** with {sentiment['confidence']:.0f}% confidence")
```

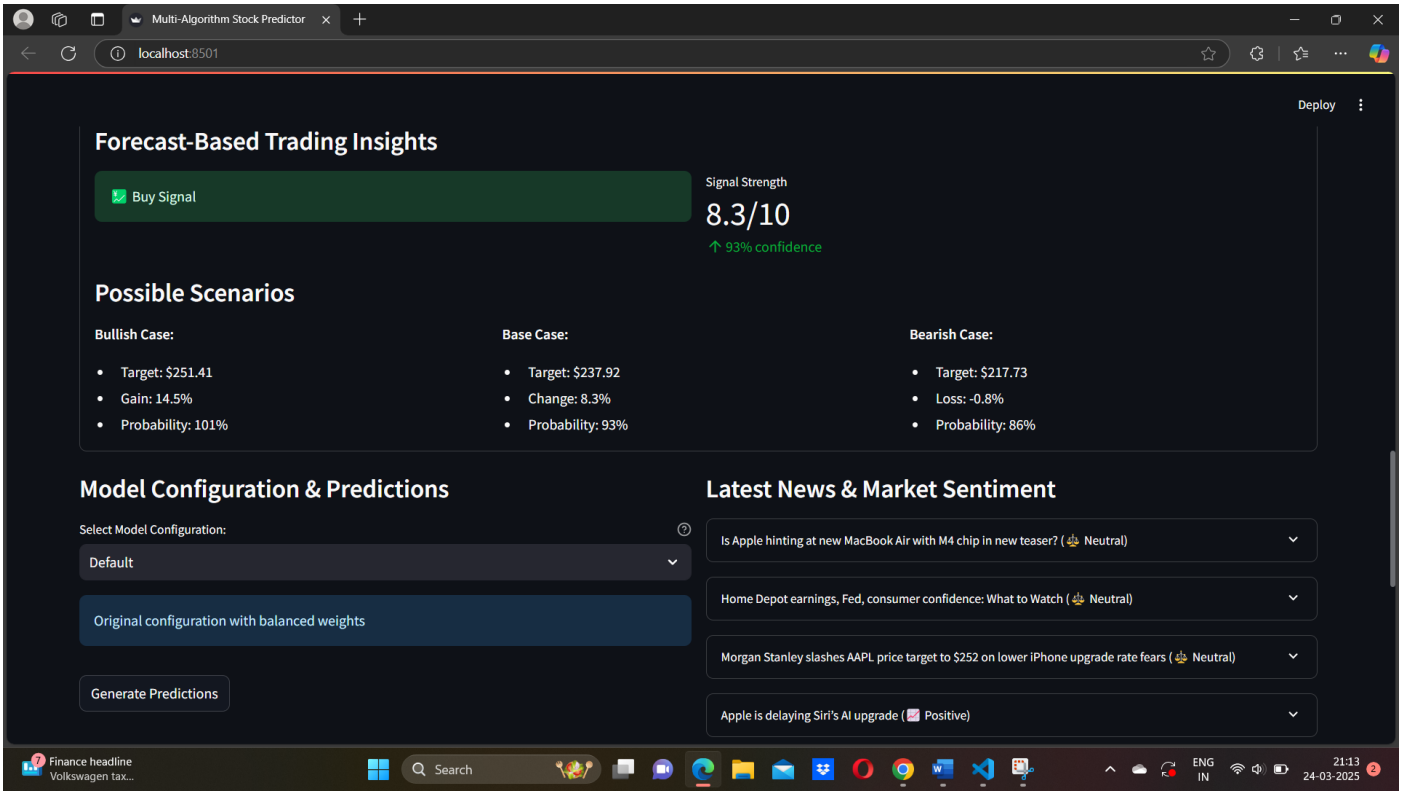
- **Predefined weight templates**:
    - Default, Trend-Focused, Statistical, Tree-Ensemble, Balanced, Volatility-Focused.

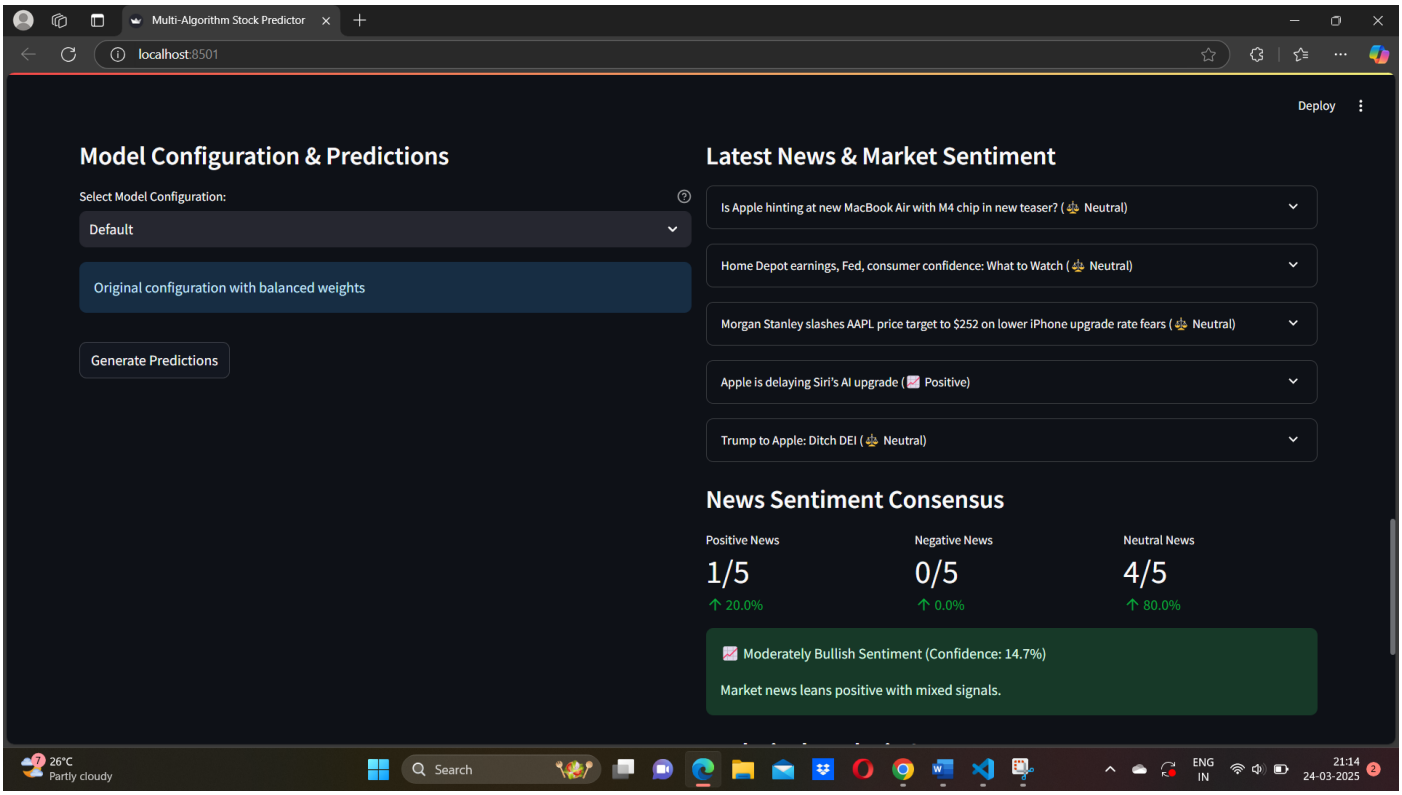
4.5 SNAPSHOT

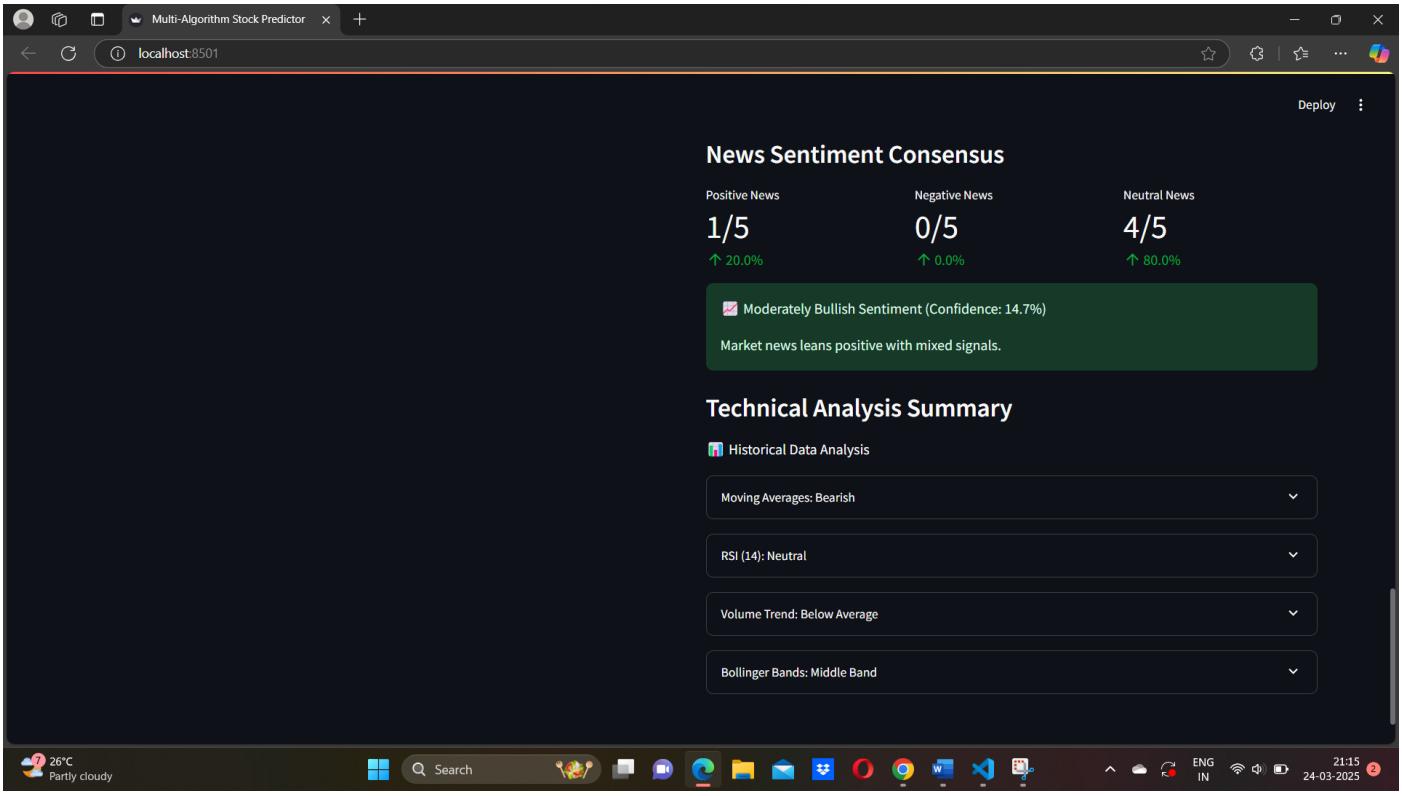

Multi-Algorithm Stock Predictor
localhost:8501
Deploy
Multi-Algorithm Stock Predictor
Disclaimer: This application provides stock predictions based on algorithms and is intended for informational purposes only. Predictions may not be accurate, and users are encouraged to conduct their own research and consider consulting with a financial advisor before making any investment decisions. This is not financial advice, and I am not responsible for any outcomes resulting from the use of this application.
Enter Stock Symbol (e.g., AAPL):
AAPL
Select number of days to display
1545
30
3650
AAPL
$219.28
LIVE
as of 2025-03-24 21:02:40
Stock Price History
Chart Controls
Show 20-Day SMA
Show 50-Day SMA
Show Bollinger Bands
Show RSI/MACD
Forecast Horizon (Days)
30
7
365
USD/INR
-0.50%
Search
ENG
IN
21:08
24-03-2025


Multi-Algorithm Stock Predictor
localhost:8501
Deploy
Price Forecast
Histogram
Signal
MACD
RSI
50-Day SMA
20-Day SMA
Close Price
Price ($)
Date
RSI
MACD
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Jan 2025
Jul 2025
Sports headline
Lewis Hamilton...
Search
ENG
IN
21:10
24-03-2025


Multi-Algorithm Stock Predictor
localhost:8501
Deploy
Prophet Forecast Details
7-Day Forecast
$216.09
-1.59%
30-Day Forecast
$237.92
8.34%
90-Day Forecast
$237.92
8.34%
Forecast Components
Prophet identifies the following patterns in the data:
Trend Analysis:
Direction: Upward (0.12% per period)
Strength: Moderate
Weekly Patterns:
None detected (weekly seasonality disabled for stock market data)
Stock markets are closed on weekends, so no trading patterns exist
Seasonal Analysis:
Current seasonal effect: Negative
Current month (March): Unfavorable historically
Next month (April): Likely unfavorable based on patterns
Forecast-Based Trading Insights
Signal Strength
Finance headline
Volkswagen tax...
Search
ENG
IN
21:12
24-03-2025


Multi-Algorithm Stock Predictor
localhost:8501
Deploy
Forecast-Based Trading Insights
Buy Signal
Signal Strength
8.3/10
93% confidence
Possible Scenarios
Bullish Case:
Target: $251.41
Gain: 14.5%
Probability: 101%
Base Case:
Target: $237.92
Change: 8.3%
Probability: 93%
Bearish Case:
Target: $217.73
Loss: -0.8%
Probability: 86%
Model Configuration & Predictions
Select Model Configuration:
Default
Original configuration with balanced weights
Generate Predictions
Latest News & Market Sentiment
Is Apple hinting at new MacBook Air with M4 chip in new teaser? (Neutral)
Home Depot earnings, Fed, consumer confidence: What to Watch (Neutral)
Morgan Stanley slashes AAPL price target to $252 on lower iPhone upgrade rate fears (Neutral)
Apple is delaying Siri's AI upgrade (Positive)
Finance headline
Volkswagen tax...
Search
ENG
IN
21:13
24-03-2025


Multi-Algorithm Stock Predictor
localhost:8501
Deploy
Model Configuration & Predictions
Select Model Configuration:
Default
Original configuration with balanced weights
Generate Predictions
Latest News & Market Sentiment
Is Apple hinting at new MacBook Air with M4 chip in new teaser? (Neutral)
Home Depot earnings, Fed, consumer confidence: What to Watch (Neutral)
Morgan Stanley slashes AAPL price target to $252 on lower iPhone upgrade rate fears (Neutral)
Apple is delaying Siri's AI upgrade (Positive)
Trump to Apple: Ditch DEI (Neutral)
News Sentiment Consensus
Positive News
1/5
20.0%
Negative News
0/5
0.0%
Neutral News
4/5
80.0%
Moderately Bullish Sentiment (Confidence: 14.7%)
Market news leans positive with mixed signals.
26°C
Partly cloudy
Search
ENG
IN
21:14
24-03-2025


Multi-Algorithm Stock Predictor
localhost:8501
Deploy
News Sentiment Consensus
Positive News
1/5
20.0%
Negative News
0/5
0.0%
Neutral News
4/5
80.0%
Moderately Bullish Sentiment (Confidence: 14.7%)
Market news leans positive with mixed signals.
Technical Analysis Summary
Historical Data Analysis
Moving Averages: Bearish
RSI (14): Neutral
Volume Trend: Below Average
Bollinger Bands: Middle Band
26°C
Partly cloudy
Search
ENG
IN
21:15
24-03-2025

CHAPTER 5 CONCLUSION & FUTURE ENHANCEMENTS

5.1 conclusion

The **Multi-Algorithm Stock Predictor** is a sophisticated and robust system that combines the power of machine learning, deep learning, technical analysis, and sentiment analysis to assist traders and investors in making informed trading decisions. By leveraging multiple algorithms such as Random Forest, XGBoost, LSTM, ARIMA, GARCH, Prophet, and ensemble techniques, the platform delivers comprehensive stock price predictions along with risk assessment and confidence scoring.

Through the integration of libraries like Streamlit, scikit-learn, TensorFlow, XGBoost, yfinance, TextBlob, Prophet, and others, the system ensures a seamless and interactive user experience. The application not only provides technical indicator visualizations but also real-time news sentiment, model consensus analysis, and custom forecast controls, enabling users to adapt to different market conditions effectively.

While no model can guarantee absolute accuracy in predicting stock prices due to the inherent complexity and volatility of financial markets, this project lays a strong foundation for building data-driven decision-making tools. It encourages users to combine technical insights with fundamental analysis and practice disciplined risk management.

Looking ahead, the project opens the door for several promising enhancements, including real-time data streaming, social media sentiment analysis, advanced deep learning models, portfolio optimization, and automated trade execution. These upgrades will further improve the system's predictive capabilities and real-world usability.

In summary, the **Multi-Algorithm Stock Predictor** represents a significant step towards developing an intelligent, user-centric platform for financial forecasting, offering users valuable insights while continuously evolving with market trends and technological advancements.

5.2 Future Enhancement

1. Social Media Sentiment Integration (Twitter, Reddit)

- **Description**: Integrate real-time social media sentiment analysis from platforms like Twitter and Reddit using APIs such as Twitter API (X API) or Reddit's PRAW.
- **Why**: Social media often captures market-moving information faster than traditional news outlets. Adding this will provide better sentiment analysis by gauging crowd psychology and hype around a stock.
- **Example**: Detecting positive sentiment spikes on Reddit's r/WallStreetBets about GME before price surges.

2. Deep Learning Model Expansion (GRU, Transformer Models)

- **Description**: Introduce advanced deep learning models like GRU (Gated Recurrent Units) and Transformer-based models for time-series forecasting.
- **Why**: These models can capture long-term dependencies and market patterns better than traditional LSTM networks, especially for volatile stocks.

- **Example**: Use a Transformer model to predict Tesla stock prices by processing longer sequences of historical data compared to LSTM.

3. Feature Engineering Automation

- **Description**: Build automated pipelines that engineer new features like volatility indexes, sector correlations, macroeconomic indicators (CPI, interest rates), etc.
- **Why**: Adding more relevant features can significantly improve model accuracy and robustness across different market conditions.
- **Example**: Automatically create and add a volatility-adjusted momentum score as a feature for XGBoost and RandomForest.

4. Market Regime Detection System

- **Description**: Implement algorithms to detect market regimes (bullish, bearish, sideways) based on volatility and trend patterns.
- **Why**: Different models perform better in different market regimes. By detecting the regime, the app can dynamically adjust model weightings or strategy.
- **Example**: Apply a Markov Switching Model or clustering to detect "low-volatility uptrend" or "high-volatility downtrend" phases.

5. Real-Time Data Feeds and WebSockets

- **Description**: Integrate WebSockets for real-time market data streaming from APIs like Polygon.io or Alpaca Markets.
- **Why**: Switching from historical batch data (via yfinance) to real-time data allows for intraday predictions and live updates.
- **Example**: Live dashboard showing price movements and model predictions updated every second.

6. Portfolio Optimization Module

- **Description**: Add a portfolio optimizer using Modern Portfolio Theory (MPT), Black-Litterman model, or Reinforcement Learning to suggest optimal asset allocation.
- **Why**: The system will evolve from single-stock prediction to multi-asset portfolio recommendations.
- **Example**: Recommend a portfolio with weights across S&P 500 stocks that maximizes Sharpe Ratio based on forecasts.

7. Enhanced Risk Management Tools

- **Description**: Implement Value at Risk (VaR), Expected Shortfall, and scenario analysis simulations.
- **Why**: These tools will help traders and investors better assess the downside risks of trades based on model outputs.

- **Example**: Show that with 95% confidence, the maximum expected loss over the next 10 days is $500.

8. Explainable AI (XAI) Techniques

- **Description**: Use libraries like SHAP (SHapley Additive exPlanations) or LIME to explain why models are making certain predictions.
- **Why**: Improves user trust by providing transparency into model behavior.
- **Example**: "RSI contributed +20% and SMA crossover contributed +30% to the BUY signal."

9. Customizable Backtesting Framework

- **Description**: Build an in-app backtesting engine to test trading strategies over historical data with custom parameters.
- **Why**: Allows users to validate model-based trading strategies with performance metrics like win rate, drawdown, and profit factor.
- **Example**: Backtest a strategy that buys when the LSTM prediction shows >5% increase with high confidence.

5.3 REFERENCE

1. Hyndman, R.J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice.
2. Chollet, F. (2021). Deep Learning with Python.
3. Brownlee, J. (2020). Machine Learning Mastery with Python.
4. Zhang, L., & Hu, Y. (2020). “Stock market prediction using multi-source data and machine learning,” IEEE Access.
5. Facebook Research. (2017). Prophet: Forecasting at Scale.
6. Scikit-learn Documentation. (2024). Machine Learning in Python.
7. TextBlob Documentation. (2024). Simplified Text Processing in Python.
8. yfinance Documentation. (2024). Financial Data for Python.
9. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
10. Tsay, R. S. (2010). Analysis of Financial Time Series. John Wiley & Sons.
11. Murphy, J. J. (1999). Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications. New York Institute of Finance.
12. Fama, E. F. (1970). “Efficient Capital Markets: A Review of Theory and Empirical Work,” The Journal of Finance.
13. NLP Group, Stanford University. (2024). Natural Language Processing with Deep Learning. Stanford Course Notes.
14. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
15. Harris, C. R., et al. (2020). “Array programming with NumPy,” Nature.