PROJECT REPORT

ON

SHARE MARKET PREDICTION

SUBMITTED

TO

ROURKELA INSTITUTE OF MANAGEMENT STUDIES

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

FOR

MASTER IN COMPUTER APPLICATION

SUBMITTED 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 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.

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 SYSTEM"** 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 SYSTEM"** has been and submitted by **SANJANA PRADHAN,** MCA 2023-2025, **Rourkela Institute of Management Studies, Rourkela,** has been examined by us.

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

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

I wish all success in her life.

Dean Academic

RIMS, ROURKELA

Prof. Bibhudendu Panda

Head of the Department, IT

Rourkela Institute of Management Studies

Rourkela

CERTIFICATE

This is to certify that **SANJANA PRADHAN** student of MCA, **Rourkela institute of Management Studies, Rourkela, Odisha** of session 2023-2024 has completed the project successfully.

I wish all success in her life.

Prof. Bibhudendu Panda(HOD)

DECLARATION

I, **SANJANA PRADHAN**, hereby declare that the project report entitled "**SHARE MARKET PREDICTION SYSTEM**" is of my work. The above work I submitted to "**Biju Patnaik University 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 **"SHARE MARKET PREDICTION 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
Diptirani Bindhani	2305260009	Frontend & User interaction
Sanjana Pradhan	2305260020	Sentiment Analysis & Research for all the work

TABLE OF CONTENTS

Chapter 1 – Introduction

Introduction	**13**
Problem Statement	**14**
Objective	**14**
Scope	**14**
Limitation	**14**

Chapter 2 – System Analysis

2.1 Literature Review	**15**
2.2 Requirement Collection and Analysis	**16**
2.2.1.i Use Case Diagram:	**17**
2.2.1.1 Data Modeling	**18**
2.2.1.1.i E-R Diagram	**18**
2.2.1.2 Process Modeling Data Flow Diagram(DFD)	**19-20**
2.2.2 User Requirements	**21**
2.2.3 System Requirements	**21**
2.2.4 Data Requirements 2.2.5 Non-Functional Requirements	**22** **22**
2.2.6 Software Requirement	**22-23**
2.3 Feasibility Study	**23**

2.3.1 Technical Feasibility:	**23**
2.3.2 Operational Feasibility	**23**
2.3.3 Schedule Feasibility	**23**

Chapter 3 – System Design

3.1 System design	**24**
3.1.1 User interface	**24**
3.1.2 System flow diagram	**24-25**
3.1.3 Class diagram	**26-27**
3.1.4 Sequence diagram	**28-29**

Chapter 4 – Implementation, Testing and Code explanation

4.1 Implementation	**31**
4.1.1 Algorithm Design	**31**
4.1.1.1 Linear regression	**31-34**
4.1.1.2 Random Forest	**34-36**
4.1.1.3 SVR	**36-38**
4.1.1.4 XGBoost	**38-40**
4.1.1.5 LSTM	**41-43**

4.1.1.6 ARIMA	**44-45**
4.1.1.7 Prophet	**45-48**
4.1.1.8 GARCH	**48-49**
4.1.2 Libraries	**50**
4.1.2.1 Streamlit	**50**
4.1.2.2 pandas	**50**
4.1.2.3. numpy	**51**
4.1.2.4. matplotlib	**51**
4.1.2.5. scikit-learn	**52**
4.1.2.6. xgboost	**52**
4.1.2.7. tensorflow	**53**
4.1.2.8. yfinance	**53**
4.1.2.9. newsapi-python	**54**
4.1.2.11. prophet	**54**
4.1.2.12. plotly	**-55**
4.1.2.13. arch	**55**
4.1.2.14. ta	**56**
4.1.2.15. nltk	**56**
4.1.2.16. textblob	**56**
4.2 Testing	**57**
4.2.1 Test Cases	**57-60**

4.2.1 Test Script	**60-62**
4.3 code	**63-121**
4.4 Code Explanation	**121-130**
4.5 Snapshot	**131-133**

CHAPTER 5 CONCLUSION & FUTURE ENHANCEMENTS

5.1 conclusion	**134**
5.2 Future Enhancement	**134**
5.2 Reference	**135**

CHAPTER 1 INTRODUCTION

1.1 Introduction

The stock market is an extremely unpredictable and fluid space affected by numerous external elements, including global economic shifts, political happenings, company-related news, and most importantly, investor sentiment shaped by real-time updates and social media. The dynamics of financial markets frequently exhibit non-linear and chaotic behavior, where even minor external disturbances can result in considerable price changes in a very short time frame.

Because of this inherent intricacy, accurately forecasting stock price movements has consistently posed a difficult challenge. Conventional models that depend exclusively on past price trends often struggle to account for the impact of abrupt news events or changes in market psychology. Furthermore, the accelerating pace of information dissemination, fueled by the internet and social media platforms, has underscored the necessity of integrating sentiment-driven signals into predictive models.

This initiative suggests creating a Multi-Algorithm Stock Predictor, a holistic solution that combines several machine learning techniques with sentiment analysis to more reliably predict stock price trends. The system utilizes the advantages of diverse predictive models like Random Forest, XGBoost, LSTM (Long Short-Term Memory networks), ARIMA, Prophet, and other regression methods to capture both linear and non-linear patterns found in stock market data.

To enhance the robustness of predictions, the platform employs an ensemble approach, aggregating the outputs of these models to reduce the biases or limitations inherent in individual algorithms. In addition to statistical learning approaches, the solution incorporates technical indicators such as the 20-day and 50-day Simple Moving Averages (SMA), which are commonly utilized by traders to determine short- and mid-term trends. These indicators assist in identifying bullish or bearish signals, guiding users toward potential entry and exit strategies for trades.

Beyond pricing data, the initiative also integrates real-time sentiment evaluation by collecting financial news articles and headlines via API-based data gathering processes. By utilizing Natural Language Processing (NLP) tools like TextBlob and NLTK, the system assesses whether the current market sentiment is positive, negative, or neutral. This added layer of sentiment enables the model to adapt its forecasts in response to external developments, such as earnings releases or geopolitical events, which often precede notable price movements.

The Streamlit-based dashboard provides a user-friendly platform that allows traders and investors to engage with real-time predictions, technical charts (like candlestick patterns and SMA overlays), sentiment scores, and risk evaluations. Users can tailor forecast durations, investigate model consensus, and review confidence scores, providing a more comprehensive perspective on market conditions before making trading choices.

This project holds considerable practical significance for both seasoned traders and individual investors, as it facilitates data-driven decision-making, lessening the dependence on intuition or speculation. Particularly during turbulent times when conventional strategies may falter, the combined insights from technical analysis, machine learning, and sentiment data furnish users with more actionable and timely intelligence.

In summary, this project connects quantitative analysis with real-time market sentiment, offering a contemporary method for stock market forecasting that adjusts to rapidly evolving financial environments.

1.2 Problem Statement

Forecasting stock prices has always been a challenging task since the inception of the New York Stock Exchange in 1817, with various methods for predicting stock movements being employed, both with and without technology. The market is often described as erratic, and analyses based on available data rarely lead to consistent profits. The structure of the market and its context limit investors from experiencing significant gains, as all relevant information is accessible to the public, making it nearly impossible for any single investor to secure optimal stock prices. Stock prices fluctuate based on daily market conditions, presenting the challenge of advising investors on the ideal moments to buy and sell shares. Numerous regression models and classifiers exist for prediction purposes, and there is a need to identify the most effective method that yields improved accuracy in forecasting stock prices and recognizing trends.

1.3 Objectives

The primary goals of the Stock Market Analysis and Prediction project are:

- To forecast the future value of company stocks
- To evaluate the current condition of the market
- To recognize elements influencing the stock market
- To simplify analysis for the general public
- To illustrate the stock market using interactive charts
- To apply machine learning techniques

1.4 Scope of the Project

The objectives of this project involve:

- Analyzing and forecasting the stock market while providing real-time market updates.
- Determining the sentiment of financial news.
- Serving as a resource for novice investors looking to invest in the stock market.

1.5 Limitations

The project's constraints consist of:

- The analysis relies solely on the closing price.
- The accuracy is limited to over 90%, meaning achieving 100% accuracy is not possible.

CHAPTER 2 SYSTEM ANALYSIS

2.1 Literature Review

The forecasting of stock prices has been extensively studied within financial markets, utilizing machine learning, deep learning, technical indicators, and sentiment analysis. Numerous studies have shown that multi-algorithm models can improve prediction precision and enhance decision-making processes.

Machine Learning in Stock Prediction

Machine learning methods have been widely used to analyze stock price trajectories. Traditional techniques like Support Vector Machines (SVM) (Cao & Tay, 2003), Random Forests (Patel et al., 2015), and Gradient Boosting methods such as XGBoost (Chen & Guestrin, 2016) have proved effective in identifying non-linear correlations within stock data. Research indicates that integrating multiple algorithms can counteract biases linked with individual models (Kim & Han, 2000).

Deep Learning for Market Forecasting

Deep learning frameworks, particularly Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs), have become increasingly favored for market predictions (Fischer & Krauss, 2018). Studies have shown that LSTMs can effectively represent temporal dependencies, making them particularly suitable for sequential stock price information (Hosseini et al., 2021). Moreover, hybrid models that combine LSTMs with attention mechanisms have demonstrated improved forecasting performance (Qin et al., 2017).

Sentiment Analysis in Financial Markets

Sentiment analysis plays a key role in stock price forecasting, given that investor sentiment frequently impacts market dynamics. Research has highlighted the influence of news headlines and social media sentiment on stock values (Bollen et al., 2011). Natural Language Processing (NLP) techniques such as Term Frequency-Inverse Document Frequency (TF-IDF), word embeddings, and transformer models like BERT have been employed to derive insights from financial news (Malo et al., 2014).

Technical Indicators and Statistical Methods

Technical indicators, including Moving Averages (MA), Relative Strength Index (RSI), and Bollinger Bands, have been traditionally utilized in stock evaluation (Achelis, 2000). Research suggests that integrating statistical approaches like ARIMA (Auto Regressive Integrated Moving Average) (Box & Jenkins, 1976) and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) (Bollerslev, 1986) with machine learning methods can lead to better predictive capabilities.

Multi-Algorithm Approach

Recent research highlights the advantages of multi-algorithm models in financial forecasting. Ensemble learning techniques, such as bagging and boosting, have been used to enhance generalization and minimize overfitting (Dietterich, 2000). Additionally, merging various models, including deep learning, technical indicators, and sentiment analysis, has been shown to produce robust and flexible prediction systems (Zhong & Enke, 2017).

2.2 Requirement Collection and Analysis

The process of gathering requirements is crucial for the successful management and development of any project. Having a precise understanding of what the project aims to deliver by the end of its timeline enables project managers and developers to be aware of the necessary steps to complete the task. In this project, we are collecting stock data from various companies through merolagani.com, which is utilized for analysis and forecasting of both current and future values. Our project primarily concentrates on predicting future values in which users (customers) can invest their funds. For this initiative, we considered two main requirement categories: functional requirements and non-functional requirements.

2.2.1 Functional Requirements

1. User Interface & Interaction:

- A web-based platform developed using Streamlit for visualization and user interaction.
- Inputs from users for selecting stocks and defining prediction parameters.
- Presentation of real-time stock information and historical data trends.

2. Data Collection & Processing:

- Automated collection of stock market data utilizing yfinance.
- Incorporation of news sentiment analysis for added insights.
- Preparation of historical stock data for training the models.

3. Machine Learning Models:

- Deployment of various machine learning algorithms (e.g., XGBoost, TensorFlow, Prophet, etc.).
- Generation of ensemble predictions from multiple models.
- Extraction of features and computation of technical indicators.

4. Prediction & Forecasting:

- Projections for short-term and long-term stock prices using Prophet and other models.
- Assessment of prediction confidence and risk evaluation.
- Creation of trading signals for buy, sell, or hold based on model consensus.

5. Visualization & Analysis:

- Interactive graphs depicting stock trends, moving averages, and technical indicators.
- Inclusion of confidence intervals and indicators for trend direction.
- Recognition of historical patterns to enhance decision-making.

6. Testing & Deployment:

- Conducting unit and integration tests for data processing and model precision.
- Launch of the application with live updates.
- Optimization for efficiency and scalability.

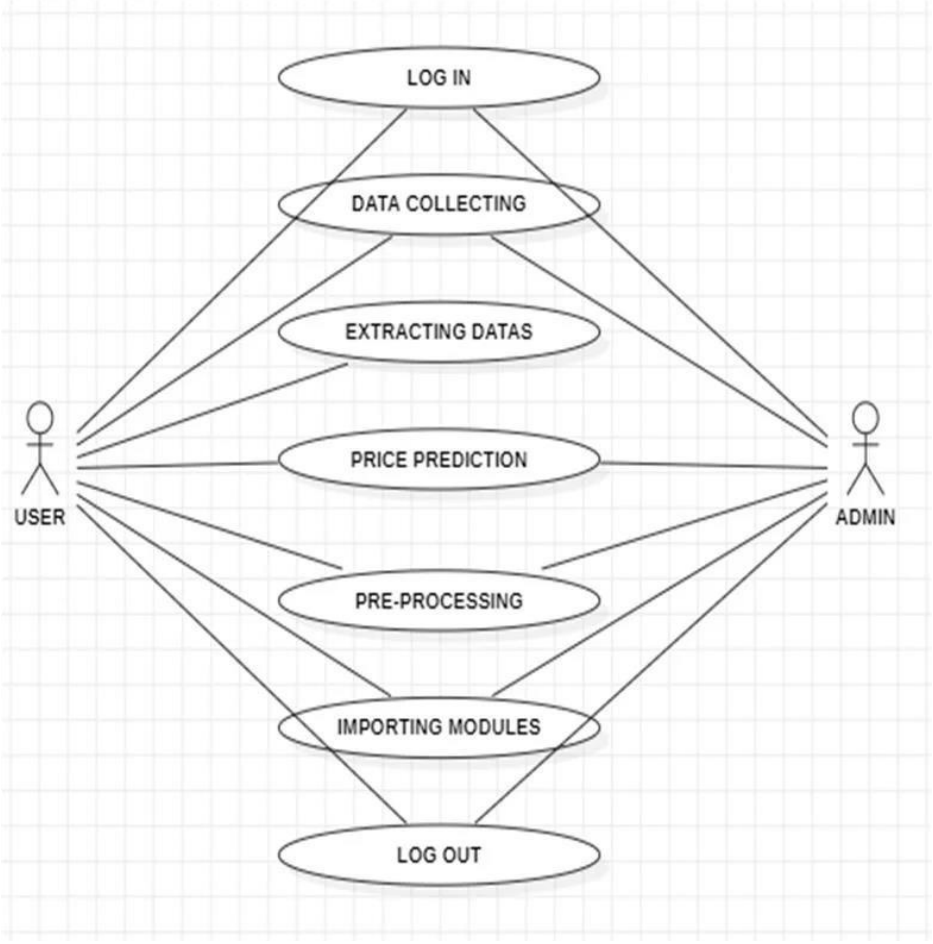
2.2.1.i Use Case Diagram:

Actor 1: User

Description: To gain full access to the system, users must register. Users log in using their username and password. If users are not logged into the system, they will be restricted from certain features. However, users without an account can still view market information. Authorized users can calculate forecasts for various companies, utilize feedback features, and receive updates on stock news.

Actor 2: Admin

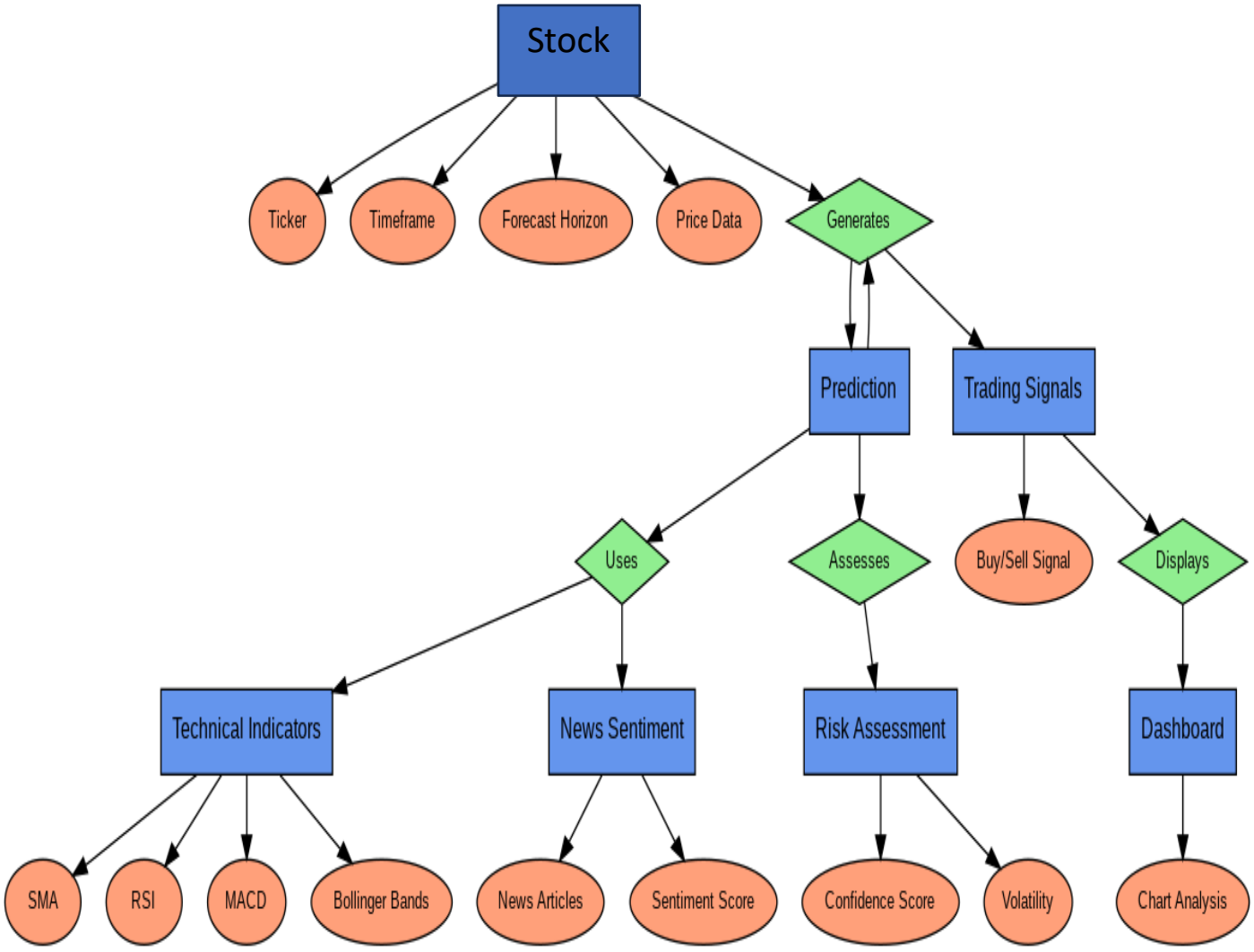
Description: Admins review new user registrations and are responsible for managing individuals within the system. The market information is updated by users in the system. All stock-related information is managed by the admin.



2.2.1.1 Data Modeling

2.2.1.1.i E-R Diagram

The E-R diagram demonstrates the connections between the entities. The system consists of four entities: admin, users, prediction, and company stock. The admin oversees both the company stock and the users. Additionally, the admin is responsible for generating the prediction value for the designated company stock. The admin possesses attributes such as id, username, and password. The company stock includes attributes like name, id, closing value, symbol, and date. Likewise, the prediction entity has attributes including id, date, predicted value, date, and actual value, while users have attributes of username, id, and password. The E-R diagram encompasses all entities within the system and illustrates the relationships among them, enhancing the overall comprehension of the system.



2.2.1.2 Process Modeling

Data Flow Diagram(DFD)

A Data Flow Diagram (DFD) visually demonstrates the movement of data within a system, depicting the involved processes, data storage, and external entities. It aids in grasping the logical progression of information, making it valuable for system analysis and design.

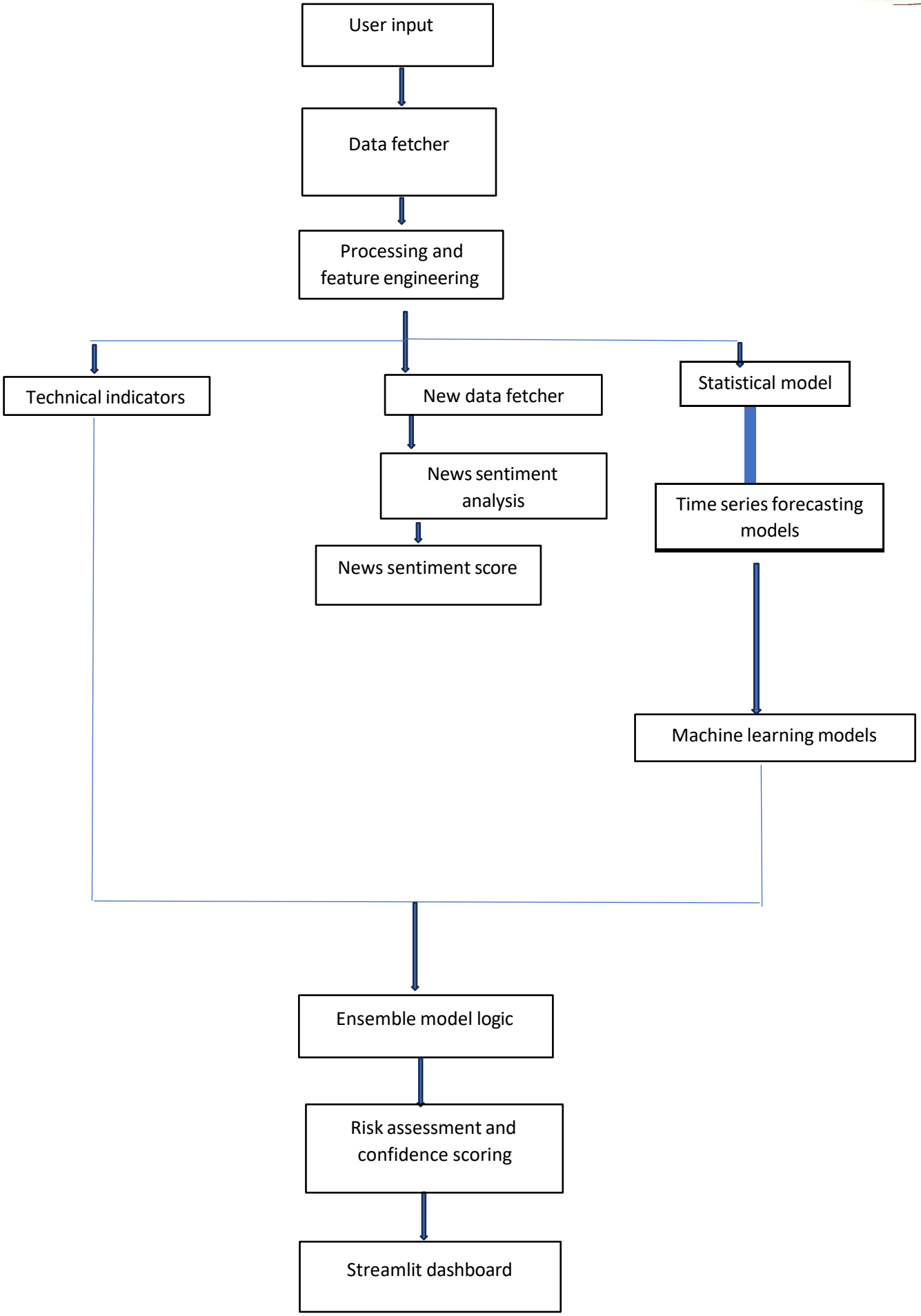
Main Elements of a DFD:

1. External Entities – Indicate the sources or destinations of data (e.g., users, systems).

2. Processes – Illustrate the changes applied to data within the system.

3. Data Stores – Represent locations where data is stored.

4. Data Flows – Show the transfer of data between entities, processes, and storage.


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 is expected to deliver precise and comprehensible predictions.
- The application should enable users to customize the duration of forecasts.
- Clear buy/sell/hold trading signals must be provided to users.

2.2.3 SYSTEM REQUIREMENTS

1. **Hardware Specifications**

- Processor: Intel Core i5 (8th Generation or newer) / AMD Ryzen 5 or superior
- RAM: At least 8GB (16GB is recommended for enhanced performance)
- Storage: Minimum of 10GB free space (SSD is advised for quicker computations)
- GPU (Optional): NVIDIA GPU with CUDA compatibility (for deep learning frameworks such as LSTM)

2. **Software Specifications**

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

3. **Necessary Python Libraries**

Install via: pip install -r requirements.txt

Data Processing: pandas, numpy

Machine Learning: scikit-learn, xgboost, tensorflow, statsmodels, prophet, arch

Technical Analysis: ta

Data Retrieval: yfinance, newsapi-python

Sentiment Evaluation: nltk, textblob

Visualization: matplotlib, plotly

Web Framework: streamlit

4. **Execution Command**

To run the application, use:

streamlit run stock_predictor.py

2.2.4 Data Requirements

The stock data collected from companies will include the date and closing value. This scraped data is saved in CSV format and then transferred to a database to train the prediction model. Additionally, the data is also stored in a MySQL database for system display. Before the application is used, the database should be refreshed with the latest market values and news. Charts and comparisons of the companies will be generated solely based on the most recent data. The predicted indications regarding market rises or falls will be recorded in the database prior to display.

2.2.5 Non-Functional Requirements

Reliability: The product's reliability will rely on the precision of the purchase date data, the volume of stock purchased, the high and low value range, as well as the opening and closing figures. Furthermore, the stock data employed in training will influence the software's reliability.

Security: Users will only have access to the website for entering stock prices through their login credentials and will not be able to view the computations occurring in the background.

Maintainability: Maintaining the product will necessitate updating the software with recent data so that recommendations remain current. The database must be refreshed with the latest values.

Portability: The website is entirely portable, and the recommendations are completely reliable since the data is updated dynamically.

Interoperability: The website exhibits high interoperability as it synchronizes all databases 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 using: pip install-r requirements.txt

- Data Management: pandas, numpy
- Machine Learning: scikit-learn, xgboost, tensorflow, statsmodels, prophet, arch
- Technical Analysis: ta
- Data Retrieval: yfinance, newsapi-python
- Sentiment Evaluation: nltk, textblob
- Data 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:

Execute using: streamlit run stock_predictor.py

2.3 Feasibility Study

A feasibility study evaluates the potential success of a project, considering factors such as economic viability, technical capability, legal considerations, and scheduling. Project managers utilize feasibility studies to assess the favorable or unfavorable results of a project prior to making any financial commitments. The various feasibility analyses are detailed below.

2.3.1 Technical Feasibility:

The project is technically viable as it utilizes popular Python libraries like pandas, scikit-learn, tensorflow, yfinance, and streamlit for its execution. The system requires moderate computational resources for both training and real-time predictions, which can be managed on a standard laptop or through cloud-based services. The integration of APIs for obtaining live stock data and conducting sentiment analysis is uncomplicated, leading to effortless deployment.

2.3.2 Operational Feasibility

The interactive interface of Streamlit enhances user accessibility, enabling traders with limited technical expertise to effectively use stock predictions. The system delivers immediate insights, consensus analysis among models, and assessments of risk, rendering it a significant resource for making informed decisions. Users are able to easily understand trading signals, identify trend patterns, and evaluate confidence scores.

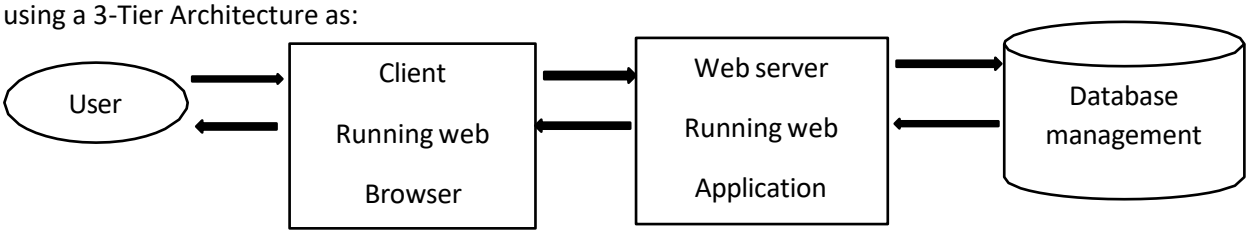
2.3.3 Schedule Feasibility

Schedule feasibility evaluates whether the project can be accomplished in a sensible timeframe, based on the resources available and the project's complexity. Below is a detailed timeline outlining the development phases for the stock market prediction system.

CHAPTER 3 SYSTEM DESIGN

3.1 System Design

System design refers to the comprehensive design of the system. The pre-defined parameters of system design are particularly beneficial for the micro process of system development, transforming the product from its blueprint stage to a functional application. This document outlines the complete design of the system. The system will be built using a 3-Tier Architecture as:



3.1.1 User Interface

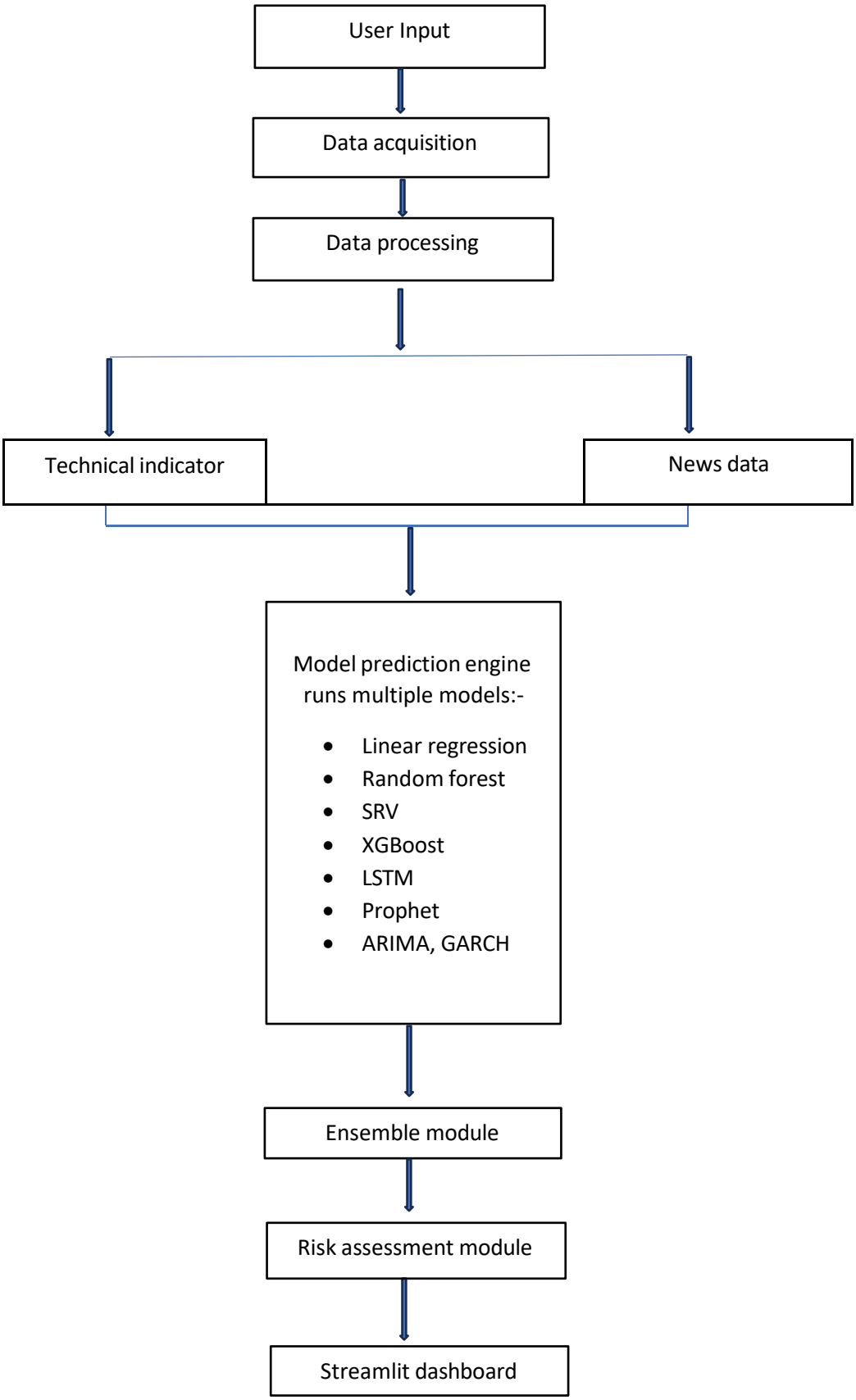
The objective of the system is to create an interactive and user-friendly interface. The design is straightforward, without any unclear areas, and is intuitive in nature.

3.1.2 System Flow Diagram

A system flowchart visually represents the workflow or processes of a system. It utilizes various shapes, symbols, and arrows, each representing a specific task or stage in the process. Arrows connecting these symbols denote the sequence of operations.

System flowcharts can depict a range of scenarios, including organizational business processes, IT software operations, manufacturing workflows, and even biological or chemical processes in scientific contexts.

By illustrating "what occurs when" in a system, these charts provide viewers with a comprehensive understanding of how different components of the system interact to achieve the intended outcome.


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

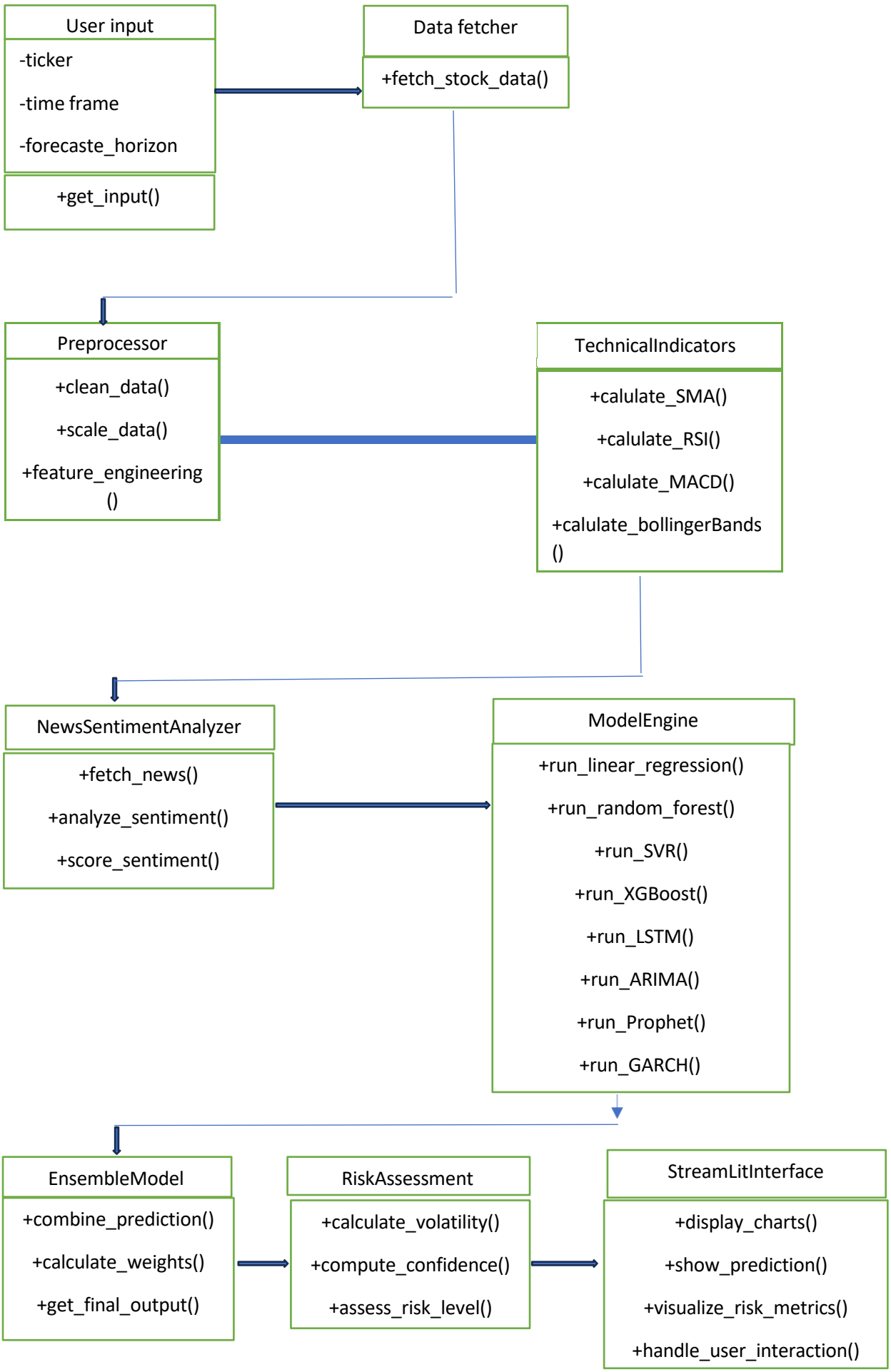
3.1.3 Class Diagram

Class diagrams utilize boxes that are divided into three sections to represent classes:

• The upper section contains the name of the class, while the middle section lists the class attributes.

• The bottom section shows the potential actions that can be performed.

Several tasks can be accomplished, such as adding, removing, and viewing stock. The stock prices for different companies fluctuate. A company has attributes including its email address, name, ID, and symbol. Various operations are carried out, such as adding companies and retrieving corporate information. A single corporation can have multiple sets of data, each containing attributes like id, close, obs_data, and date. Users are able to access company information and dates. A company can possess numerous news articles featuring attributes like author, date, image, title, and id that facilitate actions like browsing the news.


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

In a stock market project, a sequence diagram illustrates how different actors and system components interact over time. It is particularly useful for understanding processes such as user account management, order placement, and stock price checking. Below is a summary of the key components along with a brief illustration:

Key Components of a Sequence Diagram:

• Performers:

(Examples include "Trader," "System Administrator") represent external entities that interact with the system.

• Lifelines:

Vertical lines that depict the timeline of each actor or object.

• Messages:

Arrows that signify interactions between actors and objects. These can include:

- Synchronous messages: Indicate function calls where the sender awaits a response.
- Asynchronous messages: Indicate messages where the sender does not wait for a response.
- Return messages: Indicate the response to a synchronous message.

• Activation Boxes:

Rectangles on lifelines that denote when an object is actively processing a message.

Simplified Example: Executing a Stock Order:

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

1. Actors/Objects:

o Trader

o Trading Platform (Web/App)

o Order Management System

o Stock Exchange

2. Sequence of Interactions:

• Trader:

o Sends a "Place Buy Order" message to the "Trading Platform."

• Trading Platform:

o Receives the "Place Buy Order" message.

o Validates the order information.

o Sends an "Order Request" message to the "Order Management System."

• Order Management System:

o Receives the "Order Request" message.

o Verifies user funds and stock availability.

o Sends an "Order Submission" message to the "Stock Exchange."

• Stock Exchange:

o Receives the "Order Submission" message.

o Attempts to match the order.

o Sends back an "Order Confirmation/Failure" message to the "Order Management System."

• Order Management System:

o Receives the "Order Confirmation/Failure" message.

o Sends an "Order Status" message to the "Trading Platform."

• Trading Platform:

o Receives the "Order Status" message.

o Displays the order status to the "Trader."

Important Considerations for a Stock Market Project:

• **Real-time Data:**

o Sequence diagrams can demonstrate how the system obtains and updates real-time stock information.

• **Security:**

o They can illustrate authentication and authorization workflows.

• Transaction Processing:

o They are very useful in depicting the transaction flow for buying, selling, and canceling orders.

• **Error Handling:**

o They can show how the system addresses errors (e.g., inadequate funds, invalid stock symbols).

• **API Interactions:**

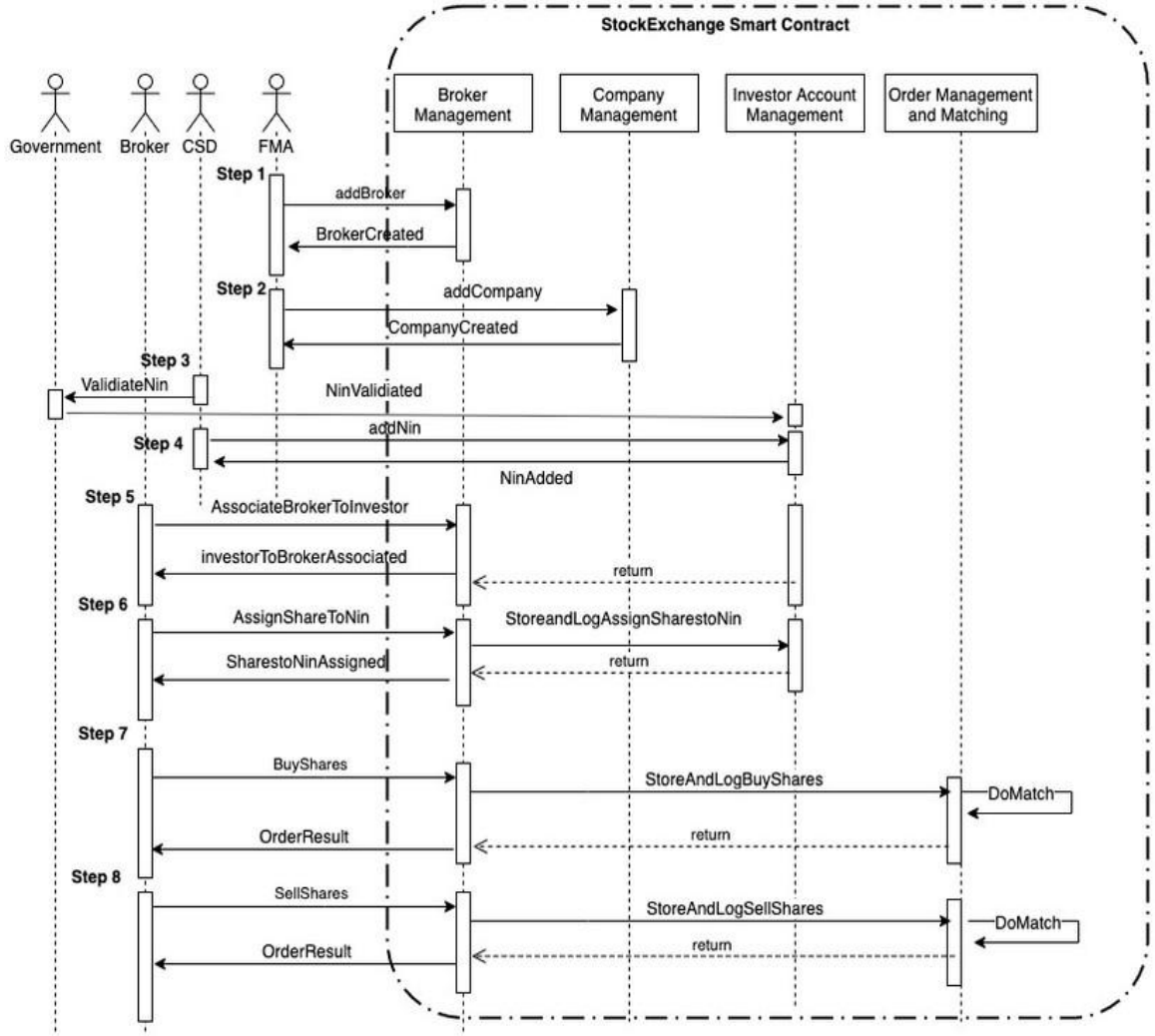
o If the system utilizes external APIs for market data or transaction processing, the sequence diagram will represent those API interactions.

Sequence diagrams serve as valuable instruments for:

• Clarifying system requirements.

• Designing resilient system architectures.

• Communicating intricate interactions between developers and stakeholders.



CHAPTER 4 IMPLEMENTATION

4.1 Implementation

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

4.1.1 Algorithm Design

The operational architecture of any project consists of algorithms that dictate the program's execution and generate results based on calculations. To ensure an efficient workflow and accurate processing outcomes, an algorithm must consider all available data variables for computation. Various online options exist for predictive analysis that utilize statistical data to generate associative results. Choosing from these many algorithms necessitates comprehensive research on the topic as well as a detailed evaluation of the system's predictions. Since there are numerous dependent variables in this scenario that are vital for the prediction, we have opted for the ARIMA algorithm.

Data Gathering

In the initial stage, several scraping scripts were developed to gather data from the sources referenced earlier in the project. The data consists of market information about companies.

4.1.1.1 Linear regression

A statistical method known as linear regression analyzes observed data to fit a linear equation, aiming to establish the relationship between a dependent variable and one or more independent variables. This technique is employed to project future stock prices based on past data in the realm of stock market forecasting. Here's a breakdown of its application:

Comprehending the Fundamentals:

• Dependent Variable: Typically, the stock price itself serves as the dependent variable in stock predictions.

• Independent Variables: These are the variables believed to influence stock prices. Some of the examples include:

· Historical stock prices, such as the previous day's closing price.

o Trading volume.

o Economic indicators, including inflation and interest rates.

o Company earnings reports.

o Market sentiment.

• Linear Equation: The goal is to identify which linear equation most accurately describes the relationship among these variables (y = mx + b, or a more complex version involving multiple independent variables).

How Linear Regression is Applied in Stock Forecasting:

1. **Data Gathering:**

- Assemble data on the chosen independent variables and historical stock price data.
- This information must be reliable and well-organized.

2. **Data Preparation:**

- Handle outliers and missing values to refine the data.
- To ensure that each variable contributes equally to the model, normalize or scale the data.
- Divide the data into training and testing sets. The training set is used to build the model, while the testing set is utilized to evaluate its performance.

3. **Model Development**:

- Utilize a linear regression approach to determine the best-fit line or hyperplane representing the relationship between independent and dependent variables.
- This process involves calculating the coefficients of the linear equation.

4. **Model Assessment**:

Assess the model's performance using metrics such as:

- Mean Squared Error (MSE): This quantifies the average squared difference between predicted and actual values.
- R-squared: This indicates how well the model corresponds with the data.
- A smaller MSE and a larger R-squared signify a better model.

5. **Generating Predictions**:

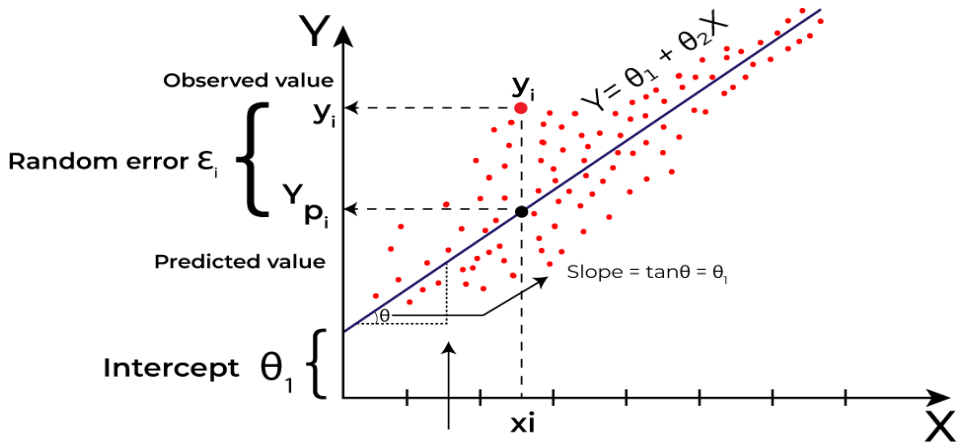
- Once the model has been trained and assessed, it can be utilized to forecast future stock prices.
- Enter the values of the independent variables into the linear equation to generate an anticipated stock price.

Limitations:

• Stock market fluctuations: The stock market is extremely volatile and affected by numerous unpredictable factors. Linear regression presumes a linear relationship, which may not always be applicable.

• Non-linear associations: Various elements impacting stock prices exhibit non-linear relationships. Linear regression is not particularly adept at handling such scenarios.

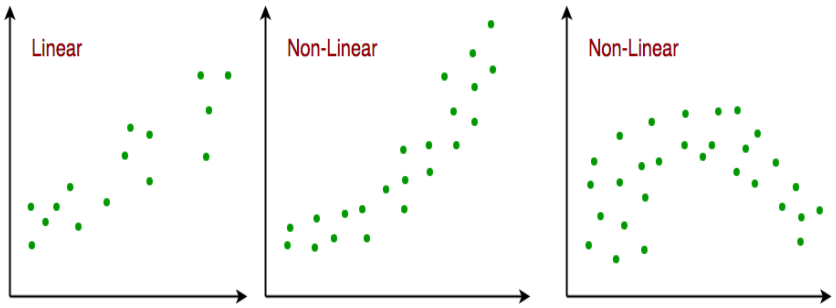
• Overfitting: If the model is overly complex, it may overfit the training data and underperform on new data.
• Economic occurrences: Unexpected economic events or news can lead to significant changes in stock prices



1. **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.

2. **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, shufle=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 constructs a large number of decision trees during the training process and produces the class based on the average prediction (for regression) or the mode of the classes (for classification) from the individual trees. While there isn't a single, clear-cut formula, understanding how it operates requires knowledge of the following key concepts:

Key Concepts:

• **Bootstrap Aggregating (Bagging):**

o Random Forest creates multiple subsets of the training dataset using the bagging technique.

o Because sampling with replacement is applied to form each subset, some data points may appear multiple times while others may be left out**.**

• **Random Feature Selection:**

o In Random Forest, a random subset of features is selected at each node of a decision tree instead of considering all features for the best split.

o This process decorrelates the trees and adds further randomness**.**

• **Decision Trees:**

o The fundamental learners within a Random Forest are decision trees.

o These trees generate predictions by recursively dividing the data based on the values of the features.

• **Ensemble Prediction**:

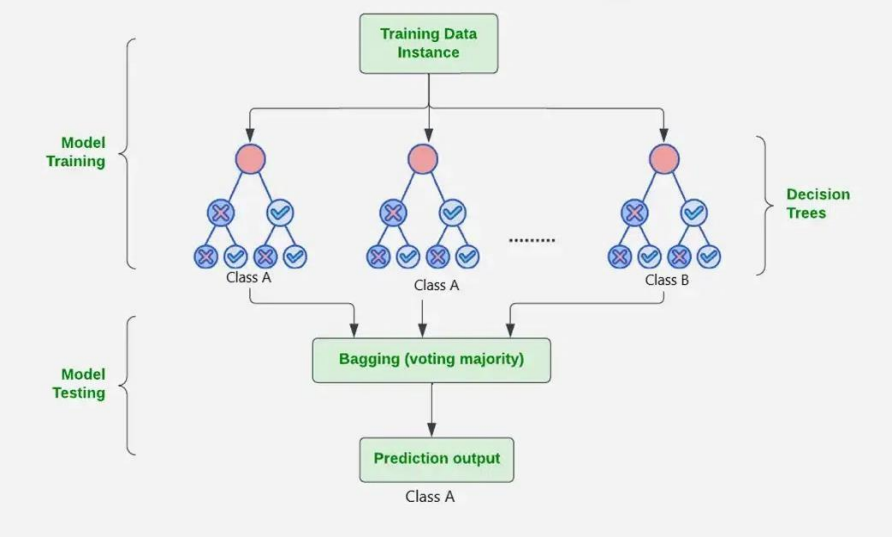
o For classification tasks, the ultimate prediction is based on a majority vote of the predictions made by the individual trees.

o For regression tasks, the final prediction is calculated as the average of the predictions from the individual trees**.**

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, shufle=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), an advanced regression method derived from Support Vector Machines (SVM), focuses on finding a function that best captures the relationship between input and output variables instead of categorizing data. Below is a concise overview of its core concepts and a simplified representation of its principles:

The central concept of SVR is to fit a line (or hyperplane in more dimensions) to the data in such a way that the maximum number of data points lies within a defined epsilon (error) margin surrounding the line.

• This "epsilon-tube" defines the permissible range of errors. The primary objective of the algorithm is to minimize errors occurring outside this tube. Kernel functions are commonly used to transform the data into higher-dimensional spaces to address non-linear relationships.

Key Concepts and Simplified Formulaic Representation:

1. Epsilon-Insensitive Loss Function:

- This component is fundamental to SVR as it determines the "penalty" associated with errors.
- The concept is that errors within a certain limit (epsilon, ε) are disregarded.
- In essence, if the predicted value is within ε of the true value, there is no associated loss.
- Linear penalties are applied to errors that fall outside this threshold.

2. Optimization Goal:

- SVR seeks to determine a function f(x) that minimizes:
- The smoothness of the function (by reducing the norm of the weight vector).
- The errors that lie beyond the epsilon-tube.

This can be framed as an optimization problem that includes:

Weight vector (w).

Bias (b).

Slack variables (ξ, ξ*).

Regularization parameter (C).

3. Kernel Functions:

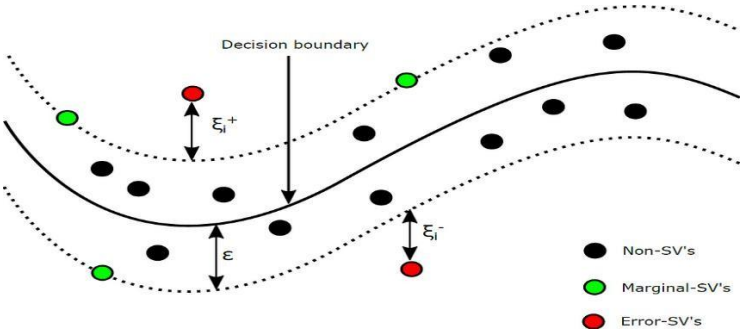
- To accommodate non-linear data, SVR incorporates kernel functions.
- These functions effectively map the data into higher-dimensional spaces where linear regression is feasible.
- Among the commonly utilized kernel functions are:
- Linear kernel.
- Polynomial kernel.
- Radial basis function (RBF) kernel.

The application of kernels is what enables SVR to capture intricate non-linear relationships.

1. 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).



2. **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, 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

One type of recurrent neural network (RNN) designed to address the vanishing gradient issue is the Long Short-Term Memory (LSTM) network, which enables the identification of long-term dependencies in sequential data. While its complexity cannot be embodied in a single, simple equation, we can break down the essential 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.

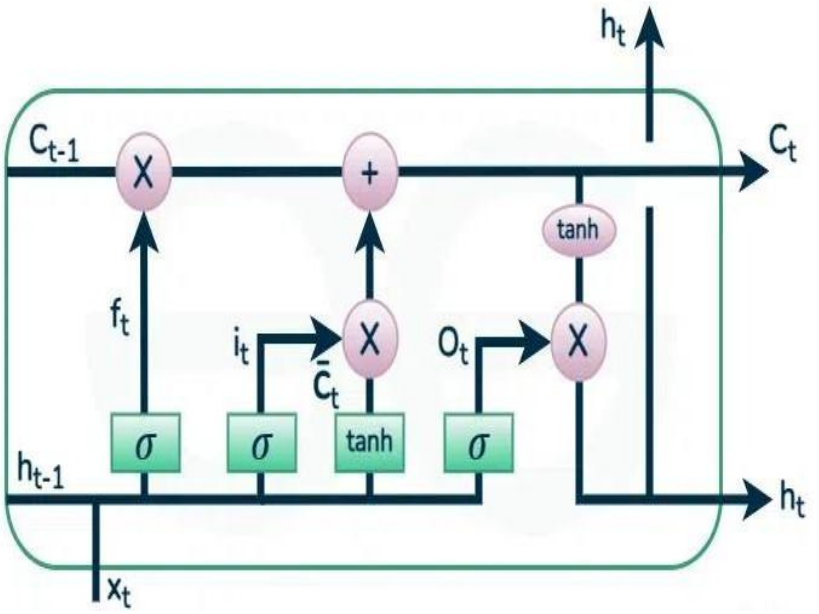
LSTM Equations:

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.

- $C\tilde{}t=\tanh(Wc\cdot[ht-1,xt]+bc)$
  - $C\tilde{}t$: 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:**
   - $Ct=ft\cdot Ct-1+it\cdot C\tilde{}t$
     - Ct: Updated cell state.
4. **Output Gate:**
   - $ot=\sigma(Wo\cdot[ht-1,xt]+bo)$
     - ot: Output gate activation.
     - Wo: Weight matrix for the output gate.
     - bo: Bias vector for the output gate.
5. **Hidden State Update:**
   - $ht=ot\cdot\tanh(Ct)$
     - ht: Updated hidden state.



Example:-

```
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

To forecast future values, the statistical time series analysis method known as ARIMA (AutoRegressive Integrated Moving Average) integrates moving averages, differencing, and autoregression. Below is a brief overview of its components and the overall equation:

ARIMA Components:

- AR (Autoregressive): Foresees future values based on past values of the time series.
- I (Integrated): Modifies the time series to remove seasonality and trends, ensuring it is stationary.
- MA (Moving Average): Estimates future values by using historical errors in forecasts.

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:

- $\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \ldots + \theta_q e_{t-q} + e_t$

Where:

- $\hat{y}_t$: Predicted value at time t.
- c: Constant term (intercept).
- $\phi_1, \phi_2, \ldots, \phi_p$: Autoregressive (AR) parameters.
- $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$: Past values of the time series.
- $\theta_1, \theta_2, \ldots, \theta_q$: Moving average (MA) parameters.
- $e_{t-1}, e_{t-2}, \ldots, e_{t-q}$: Past forecast errors.
- e_t: Current forecast error (white noise).

Breakdown of Components:

1. AR (p):
    - $\phi 1yt-1+\phi 2yt-2+...+\phi pyt-p$
    - 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: $yt'=yt-yt-1$.
    - 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):
    - $\theta 1et-1+\theta 2et-2+...+\theta qet-q$
    - 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

Meta (formerly Facebook) developed Prophet, a technique for forecasting time series data through an additive model that accommodates non-linear trends along with weekly and yearly seasonalities as well as holiday effects. It is most effective with time series that possess several historical seasons of data and pronounced seasonal influences.
Prophet Model Components:

Prophet decomposes a time series into three main components:

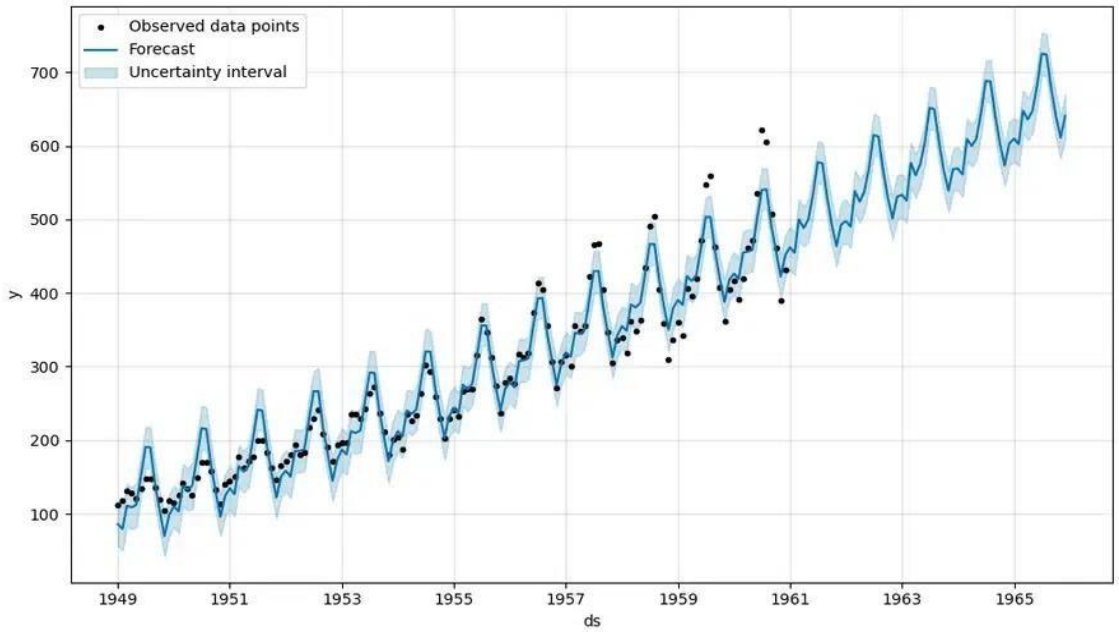
- Trend (g(t)):
    - Models non-periodic changes in the time series. Prophet offers two trend models:

- 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

By positing that the variance of the current error term depends on previous error terms and past variances, GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models effectively capture volatility clustering, a phenomenon in which high volatility periods are likely to be succeeded by additional high volatility periods, and the same applies for low volatility. GARCH models address the issue of heteroskedasticity, wherein the variance of errors in a time series fluctuates over time, and they are utilized to model and predict volatility in time series data, particularly within financial markets. 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.

Example:-

```
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 lightweight, open-source Python framework that simplifies the process of developing and deploying machine learning and data science web applications with minimal coding.

Why it was used:

This project's web interface is built on Streamlit, which allows for the rapid prototyping of interactive dashboards that enable users to analyze visual outputs (such as charts, sentiment analyses, and model predictions), modify forecasting horizons, switch technical indicators (like SMAs), and enter stock tickers. Streamlit supports real-time updates and interactive widgets (including checkboxes and sliders) that are 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 for handling labeled, tabular, and time-series data, including Series and DataFrames.

Why it was used:

Pandas is essential for gathering, organizing, and preprocessing stock market data in this project. It plays a crucial role in the following tasks:

• Aggregating daily OHLCV data into weekly or monthly periods.

• Calculating daily returns and moving averages.

• Merging sentiment scores, model outputs, and technical indicators into a comprehensive dataset for training and inference. Its robust time-series capabilities facilitate the alignment of news articles, technical indicators, and date-specific model predictions with stock data.

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 fundamental library for numerical computations in Python, enabling efficient operations on arrays and matrices.

Why it was utilized:

The advantages of NumPy within this system include:

• Facilitating mathematical tasks for model feature engineering, like calculating rolling statistics, log returns, and volatility.

• Allowing for vectorized calculations during data preprocessing, which accelerates the handling of large financial datasets.

• Incorporating TensorFlow and Scikit-learn to perform tensor and matrix calculations in machine learning models.

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 visualization library for Python that allows for static, animated, and interactive visual representations.

Why it was used:

It was employed for creating static charts like time-series plots that illustrate trend lines, moving averages, and historical price data.

• It provides visual tools for assessing models, including ACF/PACF graphs and ARIMA residual plots.

• It generates charts for risk assessment, such as volatility heatmaps. When constructing financial reports, it offers precise control over the appearance and layout.

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 that offers user-friendly and efficient tools for machine learning and predictive analytics.

Why it was used:

It is crucial for: • Training baseline models such as Random Forest, SVR, and Linear Regression with engineered features.

• Validating models through cross-validation techniques and tools like GridSearchCV.

• Scaling datasets (for instance, by using StandardScaler) to ensure that models like SVM and neural networks converge successfully. Moreover, Scikit-learn works well with NumPy and Pandas for managing 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 efficiency and high performance.

XGBoost is employed in this study to:

• Capture complex non-linear relationships between technical indicators of stocks and their expected price movements.

• Handle large datasets effectively with built-in regularization to reduce the risk of overfitting.

• Offer feature importance rankings to identify which sentiment features or technical indicators are most predictive.

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:
Google developed TensorFlow, a comprehensive open-source deep learning framework that includes both high-level and low-level APIs.

The application of LSTM (Long Short-Term Memory) neural networks, which excel at modeling sequential stock price data, is essential.

• Constructing deep learning models to uncover long-term patterns in time-series data associated with finance.

• Utilizing GPU acceleration to expedite the training on large datasets. The flexibility of TensorFlow also allows for integration with Keras, simplifying the process of building neural networks.

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:

Users can access Yahoo Finance's financial data through the yfinance Python package.

Why it was used:

• To gather current and historical stock price information (including open, high, low, close, and volume), serving as the data ingestion layer.

• To enable the automatic retrieval of data regarding stock splits and dividends.

• To provide daily and intraday intervals (ranging from 1 minute to 1 hour) for developing models with customizable 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:

Statsmodels is a Python package designed for statistical modeling and testing hypotheses.

Why it is used:

• Predicting stock prices through time-series models such as ARIMA and SARIMA;

• Checking for stationarity with statistical diagnostics like Augmented Dickey-Fuller (ADF) tests; and

• Evaluating traditional econometric models with regression diagnostics.

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:

Meta developed the Prophet library for time-series forecasting to detect changepoints and seasonal trends.

Why it was used:

• To predict stock prices over various customizable periods, ranging from 7 to 365 days, along with confidence intervals.

• To decompose stock data into weekly, yearly, and trend seasonal components.

• To adjust for known holidays or events influencing price changes and to effectively manage missing data.

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:

The interactive graphing library Plotly supports features like zooming, panning, and tooltips.

Why it was used:

By creating interactive candlestick charts, it enhances the user experience.

• It allows users to dynamically explore predictions within the Streamlit app.

• It enables the 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 specialized software designed for modeling volatility in financial time-series, focusing on GARCH-family models.

Why it was used:

• To simulate the clustering of stock return volatility.

• To forecast future market volatility, incorporating risk measures into price predictions.

• To apply volatility diagnostics for identifying residuals that display heteroskedasticity, or variable variance.

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)

Description: Ta is a Python library that provides over 100 implementations of technical indicators.

Purpose:

Employed in calculating trend indicators such as MACD, EMA, and SMA.

• Utilized for momentum indicators like the Stochastic Oscillator and RSI.

• Engaged for volatility indicators such as ATR and Bollinger Bands. These attributes are fed directly into machine learning models as predictive features.

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 suite designed for linguistic data examination and natural language processing.

Why it was used:

this project utilizes tokenization, removal of stop words, and basic text cleaning of news headlines.

• Prepares the text for sentiment analysis using TextBlob or other NLP tools in subsequent stages.

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

TextBlob is a simple NLP library that offers sentiment analysis and various NLP functionalities. It relies on NLTK and Pattern.

With TextBlob, users can:

• Analyze the subjectivity and polarity (positive or negative) of financial news headlines.

• By identifying a relationship between the sentiment of news articles and price fluctuations, you can incorporate a qualitative aspect into stock forecasting.

• Generate sentiment scores that can then serve 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 Retrieval

Test Case ID: TC_01

Test Case: Verify retrieval of historical stock data

Preconditions: yfinance installed and internet connectivity present

Input: Stock ticker symbol (e.g., AAPL)

Expected Output: Displayed historical stock data (OHLCV)

Priority: High

2. SMA Computation

Test Case ID: TC_02

Test Case: Validate the calculation of 20-day and 50-day SMA

Preconditions: Historical stock data must be available

Input: Closing price data

Expected Output: SMA values accurately integrated into the dataframe

Priority: High

3. RSI Computation

Test Case ID: TC_03

Test Case: Verify the calculation of the RSI indicator

Preconditions: Historical stock data must be available

Input: Closing price data

Expected Output: RSI values calculated and added to the dataframe

Priority: Medium

4. News Sentiment Evaluation

Test Case ID: TC_04

Test Case: Verify sentiment analysis using TextBlob

Preconditions: News API key and newsapi-python installed

Input: Stock ticker news headlines

Expected Output: Sentiment polarity score generated

Priority: High

5. ARIMA Forecasting

Test Case ID: TC_05

Test Case: Validate the output of the ARIMA model forecast

Preconditions: Dataframe containing Close prices available

Input: Historical closing prices

Expected Output: ARIMA model produces forecast values

Priority: High

6. RandomForest Predictions

Test Case ID: TC_06

Test Case: Confirm predictions made by the RandomForest model

Preconditions: Feature engineering must be completed (SMA, RSI)

Input: Features dataframe

Expected Output: RandomForest model provides predictions

Priority: High

7. XGBoost Predictions

Test Case ID: TC_07

Test Case: Verify correct predictions from the XGBoost model

Preconditions: Features dataframe must be available

Input: Features dataframe

Expected Output: Predicted values generated by XGBoost

Priority: High

8. LSTM Model Validation

Test Case ID: TC_08

Test Case: Ensure predictions from the LSTM model are accurate

Preconditions: Sequential data has been processed

Input: Sequences of price data

Expected Output: LSTM model returns predicted values

Priority: High

9. Prophet Forecasting Accuracy

Test Case ID: TC_09

Test Case: Validate the accuracy of the Prophet model's forecasts

Preconditions: Prophet library must be installed

Input: Closing price dataframe

Expected Output: Prophet returns future forecasts along with confidence intervals

Priority: High

10. GARCH Volatility Analysis

Test Case ID: TC_10

Test Case: Ensure the GARCH model is fitted properly

Preconditions: Log returns must be calculated

Input: Log returns

Expected Output: GARCH model summary is produced

Priority: Medium

11. Static Chart Generation

Test Case ID: TC_11

Test Case: Validate static plots created with matplotlib

Preconditions: Historical stock data must be available

Input: Closing prices, SMA, RSI

Expected Output: Static line chart displayed in Streamlit

Priority: Medium

12. Dynamic Plotly Chart

Test Case ID: TC_12

Test Case: Verify the candlestick chart created with Plotly

Preconditions: Plotly library must be installed

Input: Stock OHLC data

Expected Output: Interactive candlestick chart is shown

Priority: Medium

13. Streamlit Application Launch

Test Case ID: TC_13

Test Case: Ensure the Streamlit app loads correctly

Preconditions: App must be running with Streamlit

Input: Execute streamlit run command

Expected Output: UI components (charts, sliders, controls) are displayed properly

Priority: High

14. Combined Output Validation

Test Case ID: TC_14

Test Case: Verify the output of the ensemble results

Preconditions: All models must have executed 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(
        "# 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")
```

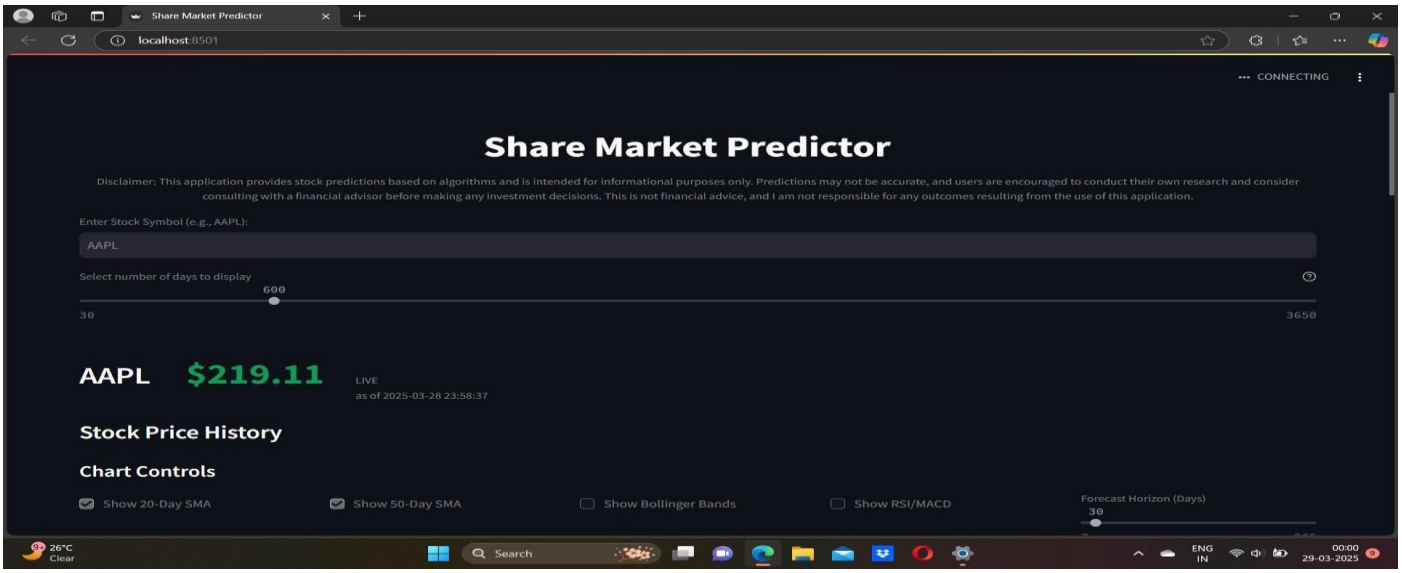
```
display_days = st.slider("Select number of days to display", min_value=30, max_value=3650, value=600)


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



This image illustrates a multi-algorithm stock prediction tool that utilizes Streamlit to project stock market trends.

• Stock Symbol Input: Users can enter a stock symbol, such as AAPL.

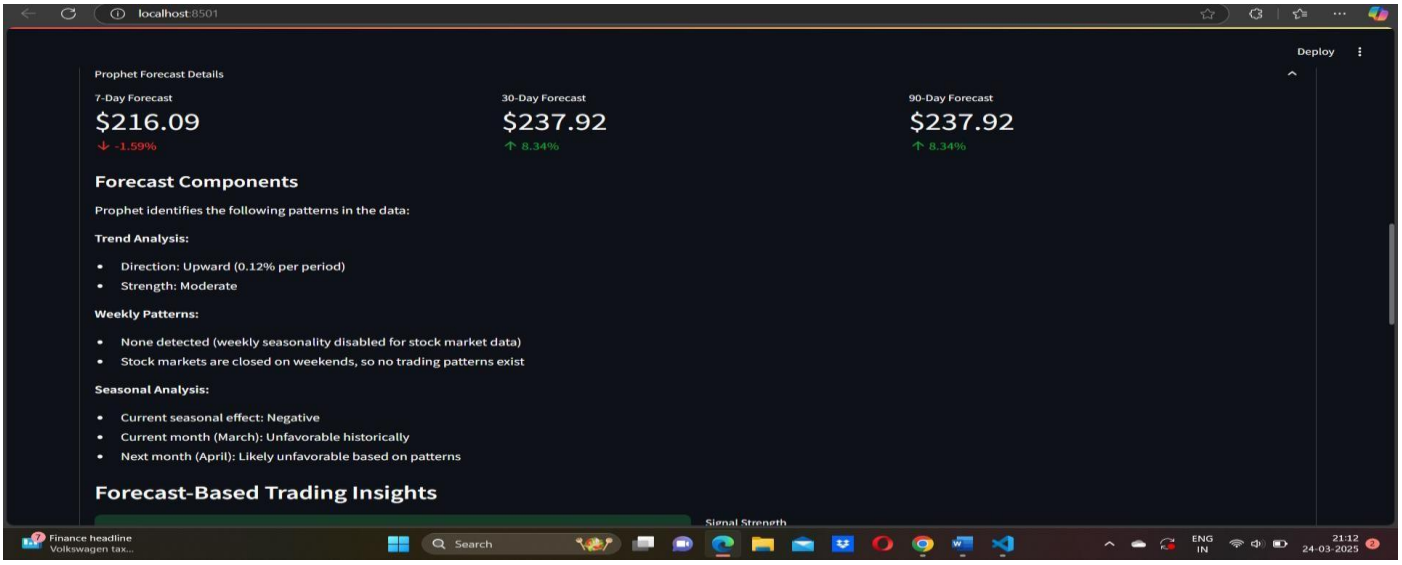
• Historical Data Range: A slider allows for the selection of a past range (30–3650 days); the present value stands at 1545 days.

• Current Stock Price: Displays the latest price of AAPL ($219.28) along with the timestamp.

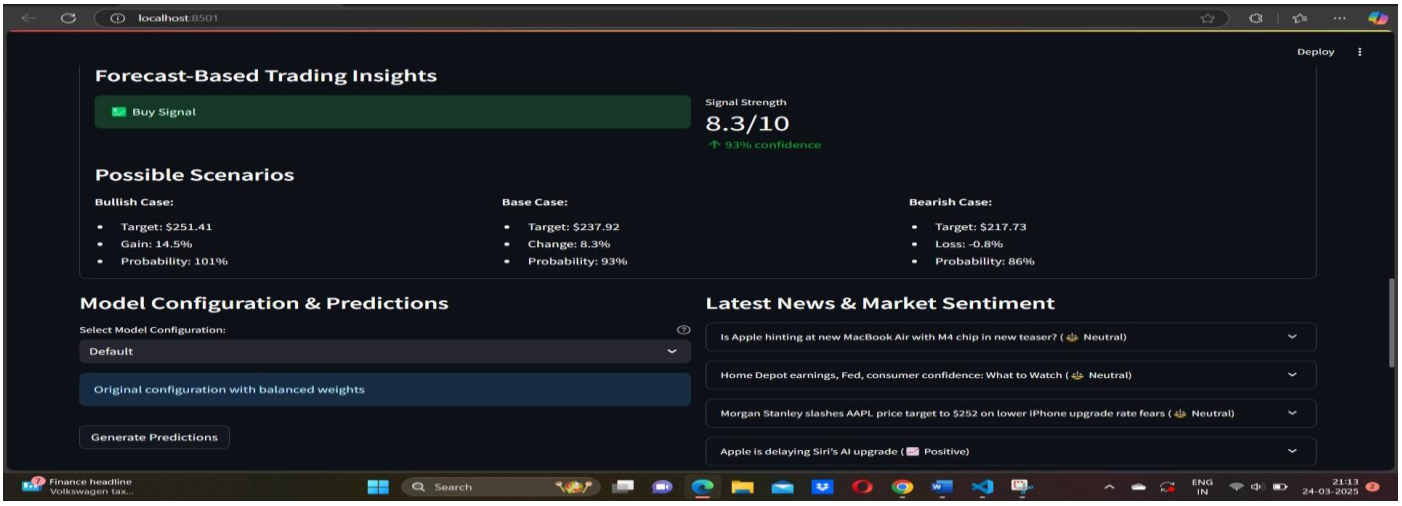
• Chart Options include the ability to show RSI/MACD, Bollinger Bands, 20-Day SMA, and 50-Day SMA.

• Forecasting Period: Currently set to 30 days, it projects stock movements over 7–365 days.

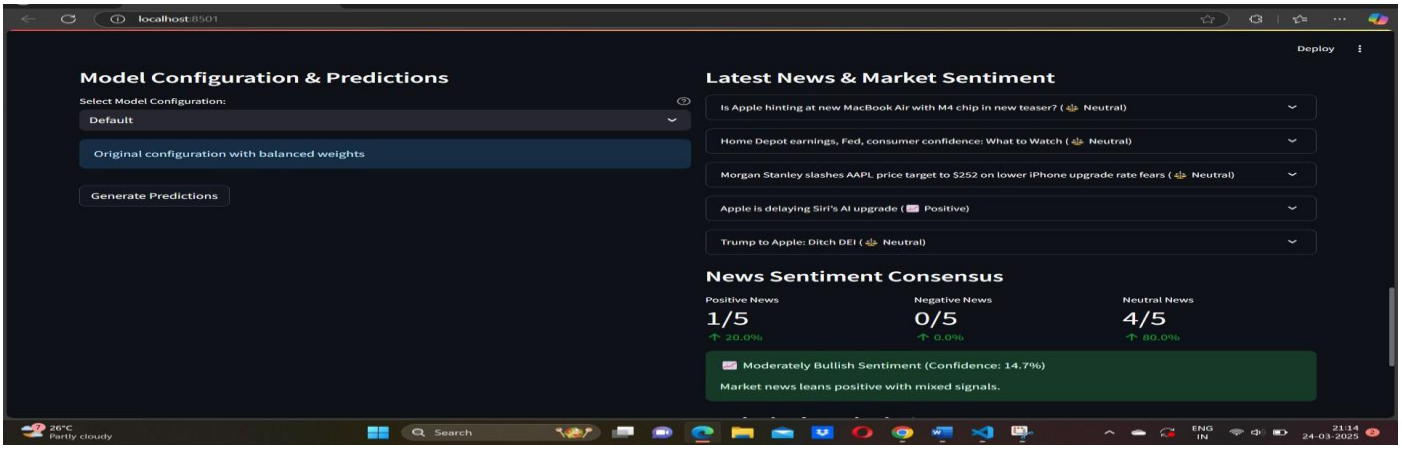
The chart illustrates anticipated AAPL stock prices based on technical indicators and machine learning. The blue line represents actual closing prices, while the yellow and green lines denote the 20-day and 50-day Simple Moving Averages (SMA), respectively, to analyze trends. Predicted future prices are depicted by the red dashed line, with uncertainty represented by the gray shaded area. Although the forecast horizon can be adjusted from 7 days to 365 days, it is currently set to 30 days. The prediction highlights the recent downward trend in the stock price, suggesting a potential decline. The integration of AI-driven forecasting and technical analysis aids in making market decisions.



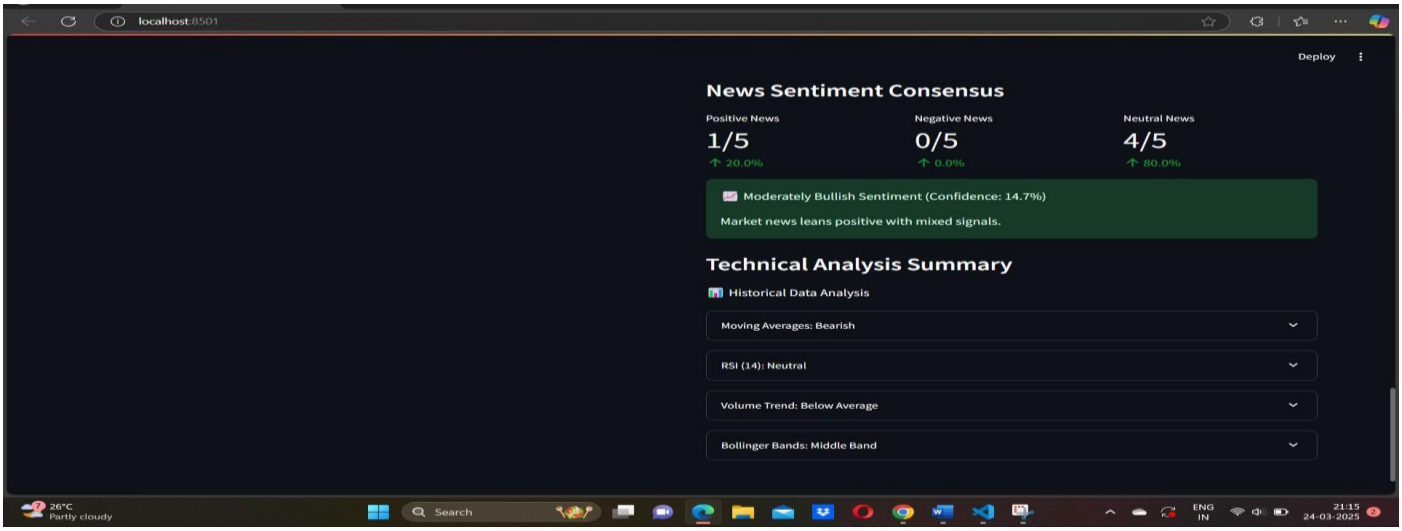
Utilizing the Prophet model, the Multi-Algorithm Stock Predictor's forecasting information provides insights into anticipated future stock price movements. The forecasts for both 30-day and 90-day periods predict a rise to $237.92 (+8.34%), indicating potential stability in the medium term, while the 7-day outlook suggests a slight decline to $216.09 (-1.59%). Trend analysis reveals a steady growth pattern, with a gradual increase of 0.12% in each period. There is no observed seasonality on a weekly basis, likely due to the stock market being closed on weekends. Historical data indicates that March and April tend to be unfavorable months for this stock, and seasonal analysis points to a negative influence during this time. These insights contribute to the assessment of potential investment strategies and market dynamics.



An appealing investment opportunity is indicated by the Multi-Algorithm Stock Predictor's Forecast-Based Trading Insights, which present a buy signal with strong confidence (8.3/10, 93%). In the base scenario, the stock is projected to climb to $237.92 (+8.3%) with a 93% probability; in the bearish case, it is estimated to drop to $217.73 (-0.8%) with an 86% probability; and in the bullish case, the stock is expected to hit $251.41 (+14.5%) with a likelihood of 101% (likely due to rounding effects). The predictor operates with balanced weights as its default setting but can be adjusted for various model configurations. Additionally, recent market sentiment analysis shows a predominantly neutral outlook, with the only positive news being Apple's AI upgrade. Other financial developments, such as Morgan Stanley's downgrade of AAPL and various economic reports, are similarly neutral.



The Multi-Algorithm Stock Predictor's Model Configuration & forecasts section shows that the "Default" model configuration is chosen, however users have the option to switch to the "Original configuration with balanced weights" in order to generate new forecasts. Five recent articles about the market are shown in the Latest News & Market Sentiment study; one is favorable (20%), four are neutral (80%), and there is no bad news (0%). Notably, the announcement that Apple is postponing Siri's AI update is well received, but other updates—like the teaser for Apple's M4 processor and Morgan Stanley's downgrade of AAPL—are met with neutrality. With a 14.7% confidence level, the News mood Consensus indicates a moderately favorable market mood, indicating a modest positive tilt but with conflicting signals. Given the neutral-heavy mood, traders may want to try using a different model to make new predictions.



The summary sections of the Technical Analysis and News Sentiment Consensus for the Multi-Algorithm Stock Predictor present a varied outlook for the market. With one article being positive (20%) and four categorized as neutral (80%), the sentiment in the news leans slightly bullish with a confidence level of 14.7%, suggesting a modestly optimistic view but with some uncertainty. On the other hand, the technical indicators demonstrate weakness as the moving averages indicate a bearish trend that might result in a downturn. Additionally, the trade volume has been below average, indicating weak market activity, while the RSI (14) remains neutral, showing no major buying or selling pressure. Furthermore, the stock price exhibits moderate volatility as it hovers near the middle Bollinger Band. Long-term investors may consider fundamental analysis alongside these technical signals, while traders should monitor price movements for clearer indicators given this mixed perspective.

CHAPTER 5 CONCLUSION & FUTURE ENHANCEMENTS

5.1 conclusion

To assist traders and investors in making informed trading choices, the stock market prediction initiative is a sophisticated and dependable system that combines the strengths of machine learning, deep learning, technical analysis, and sentiment analysis. The platform offers comprehensive predictions for stock prices, risk evaluations, and confidence scores by leveraging a range of algorithms, including Random Forest, XGBoost, LSTM, ARIMA, GARCH, Prophet, and ensemble techniques.

To ensure a seamless and engaging user experience, the system integrates various libraries such as Streamlit, scikit-learn, TensorFlow, XGBoost, yfinance, TextBlob, Prophet, and more. Beyond visualizing technical indicators, this tool provides real-time news sentiment analysis, model consensus assessment, and customizable prediction settings, allowing users to effectively respond to changing market dynamics.

Given that financial markets are inherently complex and unpredictable, it is important to acknowledge that no model can assure complete accuracy in stock price forecasting; however, this research lays a strong groundwork for creating data-driven decision-making instruments. It encourages users to practice disciplined risk management while blending technological insights with fundamental analysis.

This study also opens the door for several promising future advancements, such as automated trading execution, advanced deep learning models, social media sentiment exploration, real-time data streaming, and portfolio optimization. These enhancements will significantly increase the practical usability and predictive capacity of the system.

In conclusion, this project represents a substantial advancement towards developing an intelligent, user-centric financial forecasting platform that provides users with valuable insights while continuously adapting to reflect market shifts and technological advancements.

5.2 Future Enhancement

We can enhance our project by making key improvements that will increase accuracy and usability: Integrating Social Media Sentiment Analysis (Twitter, Reddit) for immediate market sentiment tracking, employing Deep Learning Models (GRU, Transformers) for improved time-series predictions, and implementing Automated Feature Engineering to incorporate macroeconomic indicators. A Market Regime Detection System can dynamically modify strategies, while Real-Time Data Feeds facilitate intraday forecasting. Incorporating Portfolio Optimization (MPT, Black-Litterman) broadens capabilities beyond individual stock forecasts. Risk Management Tools (VaR, Expected Shortfall) enhance trade evaluations, while Explainable AI (XAI) promotes transparency in models. Finally, a Customizable Backtesting Framework guarantees strategy validation using historical data.

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.