BCSE498J Project-II-Capstone Project

FINANCIAL MARKET FORECASTING WITH SENTIMENT INDICATORS

B.Tech.
in
Computer Science and Engineering with specialization in Data Science

21BCE3864	**KHUSHI OMAR**
21BDS0330	**ANNANT MAHESH SHARMA**
21BDS0339	**SANYA BAWEJA**

Under the Supervision of

PROF. VIJAYASHERLY.V

Associate Professor Grade 1
School of Computer Science and Engineering
(SCOPE)

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April 2025

DECLARATION

I hereby declare that the project entitled Financial Market Forecasting with Sentimental Indicators submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Vijayasherly .V

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place :Vellore

Date :15/04/25

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled Financial Market Forecasting with Sentimental Indicators submitted by Khushi Omar(21BCE3864), Annant Mahesh Sharma(21BDS0330) and Sanya Baweja(21BDS0339) **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by her under my supervision during Winter Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 15/04/25 **Signature of the Guide**

Internal Examiner **External Examiner**

Dr.Murali S

Head of Database System

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to Dr. Murali S, the Head of the Computer Science and Engineering with Specialization in Data Science, for his insightful guidance and continuous support. His expertise and advice have been crucial in shaping throughout the course. His constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, Prof. Vijayasherly V, for her dedicated mentorship and invaluable feedback. Her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. Her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Name of the Candidate

EXECUTIVE SUMMARY

Sentiment analysis examines the tone in textual data, classifying it as positive, negative, or neutral. Sources such as financial news, social media, and corporate reports capture public opinion and investor sentiment. Positive sentiment about a company's performance may indicate a price rise, while negative sentiment could signal a decline.

Time series analysis focuses on historical stock price data to identify trends and fluctuations. It uncovers factors influencing market behavior, both short and long term. Combining sentiment analysis with time series methods bridges the gap between psychological drivers and quantitative trends.

The proposed model converts sentiment data into numerical scores and compares them with historical prices to evaluate predictive power. Time series methods contextualize these scores within broader patterns. This hybrid approach enhances forecasting by linking investor sentiment with historical trends.

Preliminary results show combining sentiment and time series analysis improves predictions. Positive sentiment often aligns with price increases, while negative sentiment correlates with declines. Time series ensures historical data provides validation for sentiment-driven forecasts.

Challenges include noisy data, misinformation, and the need for rapid processing. Robust preprocessing and reliable data sources are essential to address these issues. By integrating these methods, this study offers a valuable tool for investors and policymakers, improving forecasting accuracy and understanding stock price behavior.

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	AutoRegressive Integrated Moving Average
BI-LSTM	Bidirectional Long Short Term Memory
BST	Bombay Stock Exchange
DL	Deep Learning
DT	Decision Table
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	MultiLayer Perceptron
MSE	Mean Squared Error
NLP	Natural Language Processing
N84SE	National Stock Exchange
RBI	Reserve Bank Of India
RF	Random Forest
RMSE	Root Mean Square
RNN	Recurrent Neural Network
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
WHF	Worry, Hope and Fear
8EMO	Eight Emotional Categories

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background

The financial market in India is a well-structured and dynamic platform where individuals, corporations, and the government participate in trading financial instruments such as stocks, bonds, commodities, and currencies. It comprises primary markets, where companies issue new securities to raise capital, and secondary markets, where existing securities are traded among investors. Major financial institutions like the Bombay Stock Exchange (BSE) and the National Stock Exchange (NSE) serve as key hubs for equity trading, while the Reserve Bank of India (RBI) plays a crucial role in regulating the financial system. The market is influenced by factors such as economic policies, inflation rates, corporate earnings, and global events. Additionally, India's growing fintech ecosystem and increasing retail investor participation have accelerated market growth. Financial instruments like derivatives, mutual funds, and government securities offer diverse investment opportunities. By facilitating liquidity, price discovery, and capital allocation, India's financial market serves as a critical driver of economic development and wealth creation.

Machine learning techniques, such as Long Short-Term Memory (LSTM) networks and Random Forest, analyze data to identify sentiment-driven trends in stock prices. Time series forecasting methods help predict future price movements by recognizing historical patterns. By integrating these AI methods, traders can develop more accurate forecasting models, reducing uncertainty and improving decision-making.

This project seeks to merge sentiment analysis, AI-based predictions, and real-time data to develop a clear and forward-looking method for understanding market trends. It helps institutional investors while also giving retail traders the tools to handle unpredictable markets more confidently. Participants will learn important skills in sentiment analysis, machine learning, and time series forecasting, which are in demand in fintech, investment banking, and data science. This research goes beyond just data; it aims to influence the future of financial forecasting and create a new generation of knowledgeable, confident investors.

1.2 Motivation

The stock market features people, emotions, and reactions in addition to numbers. In just a few minutes, stock values can fluctuate sharply in response to a CEO tweet, breaking news, or a popular topic. Conventional stock prediction techniques ignore the human element—the feelings that impact real market movements—in favor of historical data and technical trends.

This research takes a more effective approach by using time series forecasting and sentiment analysis powered by AI to more accurately predict stock patterns. We are able to capture the emotional state of the market in real time by looking at investor sentiment, financial news, and Twitter conversations. We can determine whether public sentiment is neutral, positive, or negative using machine learning techniques like LSTM, Random Forest, and ARIMA in conjunction with Natural Language Processing (NLP). We may then use this knowledge to make better trading decisions.

But the project offers more than simply stock price prediction; it provides hands-on experience in data science, finance, and artificial intelligence. Time series forecasting, sentiment analysis, and the use of robust tools like Python, machine learning frameworks, and data visualization packages will all be covered. These abilities open up job options in some of the most sought-after industries of today, including fintech, investment banking, and data science.

This project develops critical thinking, problem-solving, and innovative abilities in addition to technical ones, all of which are essential for making wise financial decisions. Regardless of your career goals—trading, data science, or artificial intelligence—this project will provide you the skills you need to successfully negotiate the financial markets. It is not simply study; it is about influencing the direction of financial forecasting, where data, sentiment, and AI combine to make markets more intelligent, transparent, and productive.

1.3 Scope of the Project

The objective of the project is to enhance financial market forecasting through innovative technology and smart algorithms. The project intends to enhance market transparency and risk management capabilities through comprehensive analysis of sentiment data from platforms like Twitter. Retail investors can improve their trading decisions through these data while institutional investors gain deeper insights into market patterns and price trends.

This project utilizes AI-powered sentiment analysis combined with Natural Language Processing (NLP), lexicon-based methods, and machine learning classifiers to assess investor sentiment impact on stock prices. Long Short-Term Memory (LSTM) networks and Random Forest ensemble approaches are key deep learning models that help in identifying market trends. Time series forecasting models like ARIMA and Prophet analyze data to predict upcoming market patterns.

This approach extends beyond traditional stock markets to cover cryptocurrencies and commodities as well as currency trading which allows for broad application across multiple financial markets. Algorithmic trading improves when automated systems integrate sentiment research because real-time sentiment-based triggers help traders make better buy and sell decisions. Technical analysis tools including Golden Ratio methods enhance the discovery of crucial trading opportunities. AI models support regulatory agencies by identifying fraudulent actions and market manipulation along with stock trading irregularities which lead to financial system stability.

The project establishes a robust scalable system for financial market predictions through the use of standard industry technologies including Python programming language, TensorFlow machine learning library and data visualization software. This initiative seeks to benefit both individual investors and financial institutions as well as regulatory agencies by promoting financial openness while building stability through data-driven approaches.

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

[1]The study examines the capacity of employing machine learning methods to improve stock market forecasts by combining conventional historical data with sentiment evaluations from Twitter. The researchers gathered a large dataset exceeding 755 million tweets via the Twitter API, in addition to stock market information obtained from Yahoo Finance, which featured daily opening and closing prices as well as trading volumes. A lexicon-based method was utilized for sentiment analysis, classifying tweets into eight emotional categories: happy, loving, calm, energetic, fearful, angry, tired, and sad. This method focused on gauging public sentiment and investor emotions, potentially affecting market trends. The study created three data sets: one with solely historical stock information (Basic), another merging stock data with key sentiment occurrences like worry, hope, and fear (Basic & WHF), and a third integrating stock data with all eight emotional categories (Basic & 8EMO).

To evaluate the theory that sentiment analysis may enhance stock market predictions, Support Vector Machines (SVM) and Neural Networks were employed on these datasets, with time lags from one to seven days. The models were designed to forecast if stock prices would increase or decrease on a specific day. The findings showed that the highest prediction accuracy for the DJIA was 64.10% utilizing SVM with the Basic & 8EMO dataset, whereas the peak accuracy for S&P 500 reached 62.03% with the Basic & WHF dataset. Although detailed emotional sentiment data was incorporated, the enhancements in accuracy were minor and did not consistently surpass simpler models based only on past stock data.

The research encountered various obstacles, such as the challenge of examining tweets that used creative or unclear language, like exaggerated spellings (e.g., "happyyy"), which diminished the accuracy of sentiment analysis. Moreover, the significant amount of tweet data presented computational difficulties, necessitating the use of effective filtering and data management methods. The brief training duration of just 60 days additionally affected the model's capability to generalize and generate dependable predictions.

Nonetheless, the study highlighted the promise of sentiment analysis as an additional resource for predicting stock market trends. It emphasized that although sentiment data by itself may not greatly improve prediction accuracy, it provides important insights into market dynamics shaped by public sentiment and behavioral influences. The authors determined that lengthening the training duration, enhancing sentiment analysis algorithms, and optimizing data management could yield improved outcomes. The findings of the study add to the expanding knowledge in the area where financial market forecasting meets social media analytics, highlighting the necessity for additional research to enhance this method.

[2]Forecasting stock market trends is a difficult endeavor because of the erratic nature of stock values and their responsiveness to a range of influences. Conventional forecasting methods that depend solely on past data frequently overlook market sentiment, which greatly influences stock price variations. Financial news offers important perspectives on investor sentiment, yet its disorganized format poses challenges for effective analysis. This research utilizes text mining and machine learning methods to analyze sentiment in financial news and improve accuracy in stock market predictions.

The procedure begins by collecting financial news through web crawlers and acquiring stock market data, including price fluctuations, trading volumes, and technical indicators from financial databases. The text is subsequently analyzed with the Hidden Markov Model (HMM) for segmentation, and then word embedding is performed using Word2Vec. Sentiment analysis is performed utilizing the Bidirectional Long Short-Term Memory (BI-LSTM) model, classifying news as positive, negative, or neutral sentiment. A sentiment index is subsequently generated and integrated with technical indicators from the stock market to train a Long Short-Term Memory (LSTM) model aimed at forecasting stock price fluctuations. The model's precision is evaluated by contrasting it with conventional sentiment dictionary-based approaches to guarantee reliability in predicting stock movements.

This research faces multiple difficulties, such as the intricacy of interpreting unstructured financial news because of specialized terminology and contextual subtleties. Data labeling requires significant labor, and stock trends can frequently be swayed by unforeseen external influences such as government regulations and worldwide occurrences. Additionally, the deployment of sophisticated deep learning models such as BI-LSTM and LSTM necessitates considerable computational power and accurate hyperparameter adjustments. In spite of these challenges, the research indicates that incorporating sentiment analysis of financial news enhances the accuracy of stock market predictions. The BI-LSTM model surpasses conventional sentiment dictionary methods, and incorporating sentiment indices boosts prediction accuracy by around 2% when compared to models that depend exclusively on technical indicators. Furthermore, the sentiment-driven model offers improved consistency in predicting stock market movements.

The findings show that the BI-LSTM model effectively categorizes the sentiment of financial news, and the created sentiment index aligns closely with stock market changes. The LSTM model provides enhanced precision in predicting the stock market, showing roughly a 2% increase compared to conventional technical analysis models. This study makes a notable contribution to the domain by integrating sentiment analysis into stock market predictions, enhancing accuracy, and creating a sentiment index that more appropriately captures investor actions. The results have practical uses, assisting investors and financial analysts in making more informed choices by incorporating machine learning and sentiment analysis into their trading approaches. In conclusion, this research emphasizes the significant impact of financial news sentiment on stock market trends and demonstrates how deep learning methods can enhance prediction precision.

[3]The study investigates the use of machine learning (ML) algorithms alongside sentiment analysis to anticipate stock market movements. Due to the complex characteristics and volatility of the stock market, conventional forecasting models frequently find it challenging to offer precise predictions. The research highlights the importance of utilizing sentiment data collected from social platforms and news sources, along with machine learning methods like Support Vector Machines (SVM) and Artificial Neural Networks (ANN), to improve prediction precision.

The statement highlights the difficulty of forecasting stock prices because of the erratic and non-linear characteristics of financial time series data. Relying solely on historical data can result in inaccurate conclusions, making it essential to incorporate sentiment analysis to gauge public emotions and market sentiment shaped by current news and occurrences. Traditional forecasting techniques, including fundamental and technical analysis, are examined for their emphasis on short-term predictions and their shortcomings.

The suggested approach comprises a multi-step procedure to tackle these issues. First, a historical dataset of the stock market is gathered and refined to organize and improve the data. Sentiment scores derive from evaluations of text, which are categorized as very negative, negative, neutral, positive, and very positive. These scores are subsequently combined with historical financial data to educate machine learning models. The models are evaluated across various stages, employing regression, classification, and ranking methods.

The research offers a comparative examination of various machine learning models, including Random Forest, Decision Trees, and SVM, emphasizing the advantage of SVM in delivering more precise predictions when combined with sentiment data. The results indicate that sentiment analysis, when integrated with ML models, greatly improves the accuracy of stock price predictions compared to relying solely on historical data.

The study highlights various challenges, such as the struggle to manage large volumes of social media data, the intricacy of understanding textual information because of slang, errors, and sarcasm, along with the necessity for quick processing to guarantee prompt predictions. Moreover, the dependence on one source of sentiment data and the influence of outliers on the accuracy of predictions are pointed out as constraints.

In spite of these obstacles, the research finds that ML models based on sentiment analysis provide useful information for investors, allowing them to strategically plan actions and anticipate possible profits or losses over designated timelines. The study additionally offers suggestions for upcoming research, indicating that continued progress in machine learning algorithms might enhance the precision and effectiveness of sentiment-driven financial predictions.

The research presents a contemporary and inventive method for stock market prediction by combining sentiment analysis with machine learning techniques, enhancing technological progress in financial analytics and equipping investors with innovative tools for their decision-making processes.

[4]Predicting the stock market is tough because many factors, like global events, economic policies, and how investors feel, affect price changes. Traditional methods often struggle with these complexities. This study has a three-part approach that merges fundamental analysis, technical analysis, and sentiment analysis and combines it with Artificial Intelligence(AI) and machine learning (ML) to enhance accuracy and improve returns.

Fundamental analysis looks at a company's financial documents, such as balance sheets, income statements, and cash flow statements, to find the real value of a stock. It also considers broader economic indicators and specific company details to spot investment chances. In contrast, technical analysis focuses on historical stock prices and trading volumes to identify patterns and trends. Tools like moving averages, Bollinger Bands, RSI, MACD, and stochastic oscillators, along with AI/ML techniques like Random Forest, Naïve Bayes, and fuzzy logic, help boost prediction accuracy. Sentiment analysis goes further than basic statistics by evaluating investor feelings through news, social media, and financial reports. This approach uses natural language processing (NLP) methods and etc to gauge market sentiment and its effect on stock prices.

Although it has advantages, this method encounters difficulties. Stock prices are influenced by unforeseen economic and political occurrences, and choosing the appropriate AI/ML model is essential, since different models yield different accuracy levels. Moreover, deep learning models demand considerable computational resources, and sentiment analysis may occasionally misread sarcastic, biased, or contradictory views. However, the research shows encouraging findings. The employed model reached an accuracy of 85% in sentiment classification, providing a 10.28% return over three months and a remarkable 175% return over one year—surpassing the NIFTY 50 index, which rose 76.9% in that time frame.

The research produces a detailed stock forecasting model that incorporates fundamental, technical, and sentiment analysis, yielding buy, hold, or sell advice based on AI-generated insights. The stock ranking system based on sentiment offers a more defined view of market trends. This study enhances stock selection through the integration of various analytical techniques, refines AI/ML models such as Random Forest and LSTM for heightened precision, and shows superior investment performance relative to conventional market benchmarks. Investors have the opportunity to utilize these AI-based strategies for enhanced portfolio management and risk evaluation. In the end, this research emphasizes the importance of combining various methods for stock prediction and demonstrates that AI and ML can efficiently assess multiple data sources to improve market forecasting and investment choices.

[5]The study named "An Investigation and Development into the Use of AI-Based Analytical Methods for Forecasting the Stock Market" provides a comprehensive analysis of how artificial intelligence (AI) can enhance stock market prediction methods. Conventional approaches like technical analysis, based on past market data, and fundamental analysis, which evaluates economic metrics and company performance, face major challenges when used in the intricate and fluctuating environment of stock markets. AI-driven methods, especially machine learning algorithms, present a hopeful option by enabling the handling of extensive, multidimensional datasets, identifying complex patterns, and yielding more precise forecasts. This research investigates how AI models, particularly Recurrent Neural Networks (RNNs) and Random Forest (RF), can surpass these constraints by tackling the difficulties of forecasting stock prices with improved accuracy and flexibility.

The article emphasizes the capabilities of AI in analyzing various types of data, such as past stock prices, trading volumes, market sentiment, and economic indicators. Leveraging such comprehensive datasets enables AI models to identify concealed patterns and relationships that traditional forecasting techniques might miss. The study centers on two primary AI algorithms: Recurrent Neural Networks (RNNs) and Random Forest (RF), chosen for their unique advantages in managing particular facets of stock market information. RNNs, especially effective in forecasting time-series data, are used to represent the sequential characteristics of stock prices, capturing the relationships between previous and subsequent price changes. Random Forest, an ensemble learning technique utilizing decision trees, is selected for its strength in managing high-dimensional data, commonly found in stock market predictions.

Regarding methodology, the research begins with the collection of historical stock price data and different financial indicators, including market sentiment and trading volume, from trustworthy sources. The data was subsequently preprocessed to make it appropriate for model training, involving actions to address missing values, normalize features, and eliminate outliers. The AI models were subsequently trained on the preprocessed data, employing a hold-out validation method to evaluate the models' effectiveness on data that had not been seen before. To assess the models' efficacy in forecasting stock prices, performance metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) were utilized, which measure the average size of prediction inaccuracies. The article also presents the idea of creating trading strategies founded on the forecasts generated by these AI models, where decisions to buy or sell stocks rely on anticipated future price changes, enabling the evaluation of the models in simulated real-world scenarios.

The research additionally highlights and examines various difficulties encountered in using AI for stock market predictions. Initially, the integrity of financial data poses a major challenge, as stock market information is frequently noisy, incomplete, or inconsistent, potentially harming the precision of AI models. Secondly, model overfitting is a frequent issue in machine learning, occurring when a model learns excessively from the training data and struggles with new, unseen data. The research tackles this issue by employing methods like cross-validation and regularization to avoid overfitting. Moreover, the interpretability of AI models continues to be a challenge, especially with intricate algorithms such as RNNs and Random Forest, where comprehending the decision-making process can prove to be difficult. This absence of clarity may restrict the practical implementation of AI models in actual financial decision-making, where stakeholders frequently need an explanation of how a prediction was derived.

The study's findings indicated that the Random Forest (RF) model was the most precise of the models evaluated, reaching an accuracy of 93%. The Support Vector Machine (SVM) model had a slightly lower performance, attaining 92%, whereas the Long Short-Term Memory (LSTM), which is based on RNN, reached an accuracy of 88%. These findings emphasize the capability of AI models to enhance stock price predictions in comparison to conventional techniques. Nevertheless, the document also presents a newly suggested method that attained the maximum accuracy rate of 98%. This approach surpassed the other models by integrating extra features, sophisticated data preprocessing, or optimization strategies, representing a major advancement in predicting stock prices.

The paper concludes by highlighting the advantages of AI-driven techniques compared to conventional forecasting methods. The research claims that AI models can deliver more precise, flexible, and dependable stock market forecasts, positioning them as a useful resource for financial analysts and investors. Although the results are encouraging, the paper recognizes the ongoing challenges that must be resolved, especially those related to data accuracy, model transparency, and adjusting to real-time market dynamics. The writers suggest that future studies should investigate the possibilities of ensemble learning techniques, which integrate several models to enhance performance, along with real-time forecasting methods to render predictions more dynamic and attuned to market fluctuations. More progress in these fields could result in even more precise and effective AI-driven systems for predicting stock market trends, providing a considerable advantage over conventional techniques.

[6]The research "Towards Harnessing Based Learning Algorithms for Tweets Sentiment Analysis" investigates the application of machine learning to categorize tweets as positive, neutral, or negative sentiments. As social media, particularly Twitter, turns into a center for views on topics ranging from politics to product evaluations, examining this information can yield important insights. The study concentrates on utilizing various machine learning methods to analyze and categorize tweets, with the goal of identifying the most efficient model for sentiment evaluation.

For the study, the researchers utilized the Sentiment140 (STS-Test) dataset, which includes an extensive assortment of pre-labeled tweets. Due to the chaotic nature of raw tweets, they initially underwent a data preprocessing stage utilizing Weka's AffectiveTweets package. This required data cleaning, feature extraction, and converting the text into a machine learning-compatible format. Initially, the dataset contained just three attributes; however, after filtering and processing, it increased to 44 attributes, aiding the models in learning more efficiently.

For classification, the research evaluated five distinct machine learning models: Random Forest (RF), Support Vector Machine (libSVM), k-Nearest Neighbors (IBk), Decision Table (DT), and AdaBoostM1. These models were selected due to their common usage in research involving sentiment analysis. To assess the performance of each model, the researchers employed essential metrics such as accuracy, precision, recall, and F-measure, alongside a confusion matrix that aids in comprehending classification mistakes. The findings indicated that Random Forest was the top performer, recording an accuracy of 78.11%, whereas AdaBoostM1 exhibited the least accuracy at 63.65%. The remaining models (libSVM, DT, and IBk) had accuracy levels in the middle, fluctuating between 75.30% and 65.27%.

Nonetheless, the research also pointed out various difficulties in sentiment analysis. A significant problem is that tweets tend to be brief and unclear, replete with slang, abbreviations, and casual language, which complicates the accurate detection of sentiment. A further challenge is data sparsity, which indicates that certain sentiments might be inadequately represented in the dataset, resulting in bias during classification. Grasping context and sarcasm poses challenges for machine learning models, since a phrase such as "Oh great, another traffic jam!" might be incorrectly interpreted as positive when it is genuinely negative. The existence of noisy data, including URLs, hashtags, and mentions, adds to the complexity of the process. Ultimately, every machine learning algorithm functions uniquely, and its effectiveness is significantly influenced by the dataset and the methods of feature selection applied.

According to the results, Random Forest emerged as the superior classifier, consistently exceeding the performance of the others across various assessment metrics. Conversely, AdaBoostM1 faced difficulties, probably due to its higher sensitivity to noisy data. The research further emphasized the significance of feature engineering, since enhancing the number of pertinent attributes greatly boosted classification precision. One more important point was that depending on just one evaluation metric is insufficient—a mix of accuracy, precision, recall, and F-measure provides a clearer understanding of a model’s performance.

In summary, this study emphasizes that machine learning can serve as an effective tool for sentiment analysis on social media, even with the existing challenges. Although Random Forest achieved the best outcomes, there remains potential for enhancement, particularly in dealing with sarcasm, context, and data noise. The research indicates that upcoming studies might investigate hybrid models, deep learning methods, and varied datasets to better improve sentiment classification. These results have practical significance, aiding businesses, researchers, and organizations in gaining a clearer insight into public opinion via automated sentiment analysis.

[7]Predicting the stock market and engaging in algorithmic trading are difficult because of the intricate nature and fluctuations of financial markets. Conventional trading techniques frequently overlook immediate changes in sentiment driven by financial news and social media. This research combines sentiment analysis with reinforcement learning (RL) to improve algorithmic trading strategies, utilizing natural language processing (NLP) methods and deep learning to refine decision-making. The process starts with gathering and preprocessing data, where financial news, social media information, and past stock prices are collected, and sentiment information is derived using NLP methods and the Word2Vec algorithm. Feature engineering comes next, integrating market indicators, technical indicators like MACD and RSI, along with sentiment scores as input features. The model creation stage employs deep reinforcement learning, leveraging Long Short-Term Memory (LSTM) networks and actor-critic reinforcement learning techniques. Training and optimization comprise several asynchronous processes that enhance decision-making by increasing cumulative rewards. The model's effectiveness is assessed by contrasting trading efficiency with baseline models through metrics such as Sharpe ratio, accuracy, and profitability.

This approach faces numerous challenges, such as the intricate nature of financial markets shaped by unforeseen economic and political factors, the struggle to accurately gauge sentiment from financial news and social media because of sarcasm, bias, and changes in context, along with the substantial computational resources needed for training and optimizing deep reinforcement learning models. Furthermore, risk management is an essential element in balancing the maximization of returns with the minimization of financial risks. In spite of these difficulties, the research shows encouraging outcomes. The sentiment-enhanced model reached an accuracy of 85.17% in forecasting stock movements, greatly enhancing trading efficiency over standard LSTM models. The model exceeded baseline strategies by delivering greater returns and a better Sharpe ratio. The output comprises a sentiment-sensitive reinforcement learning trading model able to execute adaptive trading choices, producing enhanced trading signals (buy, sell, hold) informed by market sentiment and technical indicators leading to enhanced profitability and improved market performance over traditional trading algorithms.

This study offers important insights by showing the effects of combining sentiment analysis with reinforcement learning in algorithmic trading. It emphasizes the importance of social media and financial news sentiment in stock trading while demonstrating the progress in AI methods that enhance trading decision-making. The sentiment-aware method improves the accuracy of market trend predictions and offers risk-adjusted returns, resulting in increased profitability while keeping risk exposure manageable. In conclusion, this research highlights the significance of integrating sentiment analysis with reinforcement learning in algorithmic trading, offering a foundation for more flexible and lucrative trading approaches.

[8]Algorithmic trading has evolved into a vital instrument for executing data-informed trading strategies, utilizing machine learning and technical analysis to enhance results. This research introduces a combined method that merges forest-based machine learning techniques with the Golden Ratio approach to improve stock price predictions. The process starts with gathering and preparing data, which includes obtaining historical stock prices, financial metrics, and sentiment information to guarantee data quality. Techniques for feature engineering are subsequently utilized to derive significant patterns from the raw data, enhancing model performance. The model development stage utilizes Random Forest (RF) and additional forest-based learning algorithms, which are trained on the obtained features to predict stock prices precisely. Furthermore, the Golden Ratio approach is utilized to pinpoint ideal entry and exit moments, enhancing profitability.

This approach faces numerous challenges, such as market fluctuations, the ever-changing characteristics of financial data, and the necessity for prompt and precise decision-making. Guaranteeing effective feature selection and enhancing the integration of machine learning models with trading tactics demand considerable computational resources. Moreover, managing risks continues to be a significant issue, weighing potential benefits against reducing losses. In spite of these difficulties, the research shows encouraging outcomes. The suggested model markedly enhances forecasting precision and trading effectiveness in comparison to singular models, with empirical evidence demonstrating better profitability and risk-adjusted returns.

The model produces a trading decision framework that provides actionable insights for traders, optimizing buy, sell, and hold tactics based on historical trends and real-time information. Performance assessment via back-testing and simulation testing validates the strength of the hybrid method. This study enhances the domain of algorithmic trading by showcasing the efficacy of integrating machine learning algorithms with traditional technical analysis techniques. The results present actionable insights for traders, financial analysts, and researchers, offering a structured and data-informed strategy for effectively navigating financial markets. Overall, the study underscores the value of integrating AI-driven forecasting with traditional trading strategies, paving the way for more adaptive and profitable trading models.

[9]Forecasting stock market trends has historically depended on fundamental and technical analyses, yet the growing influence of social media sentiment has added new aspects to financial predictions. This research examines the capabilities of ChatGPT in forecasting stock market trends through sentiment analysis of Twitter information. The study seeks to analyze if ChatGPT can accurately categorize tweets as positive, negative, or neutral and evaluate their influence on stock prices, particularly for Microsoft and Google. The research consists of several phases, starting with data gathering and preprocessing, during which a dataset of 500,000 tweets related to ChatGPT from January to March 2023 was examined.

A number of obstacles were faced, such as the noise and inconsistency in Twitter data, the challenge of sifting through unrelated tweets, and the model's restricted experience with direct stock market forecasts. Furthermore, guaranteeing precision in sentiment classification was difficult because of sarcasm, bias, and insufficient information in tweets. In spite of these challenges, the research showed encouraging findings. ChatGPT attained a precision of 70% in forecasting Microsoft's stock trends and 63.88% for Google, greatly surpassing random guessing algorithms. Additionally, ChatGPT anticipated market trends and recognized the fundamental factors affecting stock fluctuations, including new competitors and shifts in company strategies. The main result of the study was a framework for utilizing ChatGPT's sentiment analysis features to predict stock market movements by analyzing Twitter conversations. The results indicate that although ChatGPT is not specifically intended for stock market predictions, its capacity to assess and understand social sentiment offers important perspectives on possible market trends. This study adds to the wider domain of financial AI by showcasing how language models can effectively predict trends in stock prices. Subsequent studies might improve this method by utilizing larger datasets and combining data from additional social media sites to boost predictive precision. In conclusion, this research highlights the promise of AI-based sentiment analysis as an additional resource for predicting stock market trends, providing a fresh viewpoint on financial forecasting through conversational AI frameworks such as ChatGPT.[9]

[10] The article named "Performance Evaluation of Time Series Forecasting Methods in the Stock Market: A Comparative Study" examines different time series forecasting models to determine the most precise method for predicting IBM stock prices. A detailed dataset covering a decade (2010-2020) was gathered from Yahoo Finance, featuring elements like date, opening price, high and low prices, and closing price. The data was meticulously preprocessed to organize and refine it for analysis, allocating 80% for model training and reserving 20% for evaluating their predictive performance. Eight forecasting techniques were assessed: Linear Regression, Multilayer Perceptron (MLP), RBF Regressor, SMO Regression, Bagging, Random Sub Space, Time Series Holt-Winters, and Random Forest. These models were evaluated with standard assessment metrics, such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Squared Error (MSE).

The findings showcased SMO Regression as the top-performing model, reaching MAE = 0.9423, RMSE = 1.5261, and MSE = 2.3291, surpassing all other techniques in precisely predicting IBM stock prices. The Random Forest model showed the poorest performance with notably elevated error values, suggesting it is not appropriate for this dataset. The impressive performance of SMO Regression is due to its optimization method, which systematically identifies Lagrange multipliers to achieve an optimal solution, thus excelling at detecting patterns in time series data. Though accurate, SMO Regression took the most time to process compared to the other models, illustrating a balance between computational efficiency and predictive accuracy.

Numerous obstacles arose during the research, such as handling noisy and inconsistent stock market data, guaranteeing that models performed well on new data, and choosing suitable evaluation metrics to accurately compare model performance. The intricate nature of financial data, which is inherently non-linear and influenced by external factors, presented further challenges in attaining consistent accuracy. The researchers highlighted the importance of strong preprocessing methods and effective model assessment strategies.

The research findings indicate that SMO Regression offers a hopeful approach for predicting stock market trends when computational resources are plentiful. The results highlight the significance of choosing suitable forecasting models customized for particular datasets and problem contexts. The authors suggest that future studies investigate the combination of more extensive datasets and qualitative elements like sentiment analysis from news and social media to improve prediction accuracy further. These developments would equip investors and financial analysts with more dependable tools for decision-making, assisting them in refining trading strategies based on precise stock price predictions.

Table.1.1 Summary of Literature Reviews

Title	Reference	Summary
Machine Learning in Prediction of Stock Market Indicators Based on Historical Data and Twitter Sentiment Analysis	https://ieeexplore-ieee-org.egateway.vit.ac.in/document/6753954	This study integrates machine learning with sentiment analysis to predict stock trends. Using 755M+ tweets and historical stock data, sentiment categories were classified and applied to prediction models like SVM and Neural Networks. Results showed minor improvements in accuracy, peaking at 64.10% for DJIA using an emotion-based dataset. Challenges included tweet ambiguity, computational intensity, and limited training duration. The study suggests refining sentiment analysis and longer training periods for better accuracy.

Literature Review: Stock Market Prediction Based on Financial News Text Mining and Investor Sentiment Recognition	https://onlinelibrary.wiley.com/doi/full/10.1155/2022/2427389	This research explores text mining and deep learning for financial news sentiment analysis. Using BI-LSTM for sentiment classification and LSTM for stock trend prediction, it showed a 2% accuracy improvement over traditional models. Challenges included financial jargon interpretation, data labeling, and computing demands. Findings support sentiment-driven models for more consistent market trend forecasts, aiding investors in decision-making.
Stock Market Prediction using Sentiment Analysis and Machine Learning Approach	https://ieeexplore-ieee-org.egateway.vit.ac.in/document/9716326	This study assesses ML models (SVM, ANN, Decision Trees, etc.) for stock prediction by incorporating social media sentiment analysis. SVM performed best, showing sentiment data improves prediction accuracy over historical data alone. Challenges included managing vast unstructured text, slang, sarcasm, and processing speed. The study highlights ML's potential in sentiment-driven financial forecasting and suggests further AI advancements for improved accuracy.
A Three-Dimensional Approach for Stock Prediction Using AI/ML Algorithms	https://ieeexplore-ieee-org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=10157327	A three-pronged AI/ML method integrating fundamental, technical, and sentiment analysis is proposed. Using Random Forest, Naïve Bayes, and fuzzy logic, the model achieved 85% sentiment classification accuracy and a 175% annual return, outperforming the NIFTY 50 index. Challenges included economic unpredictability and deep learning computational costs. Findings confirm AI's potential in optimizing investment decisions.

An Investigation and Development into the Use of AI-Based Analytical Methods for Forecasting the Stock Market	https://ieeexplore-ieee-org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=10275988	AI-driven models (RNN, Random Forest, SVM) were tested for stock market forecasting. Random Forest achieved 93% accuracy, outperforming LSTM (88%). A proposed hybrid model reached 98% accuracy. Challenges included data noise, overfitting, and model interpretability. The study suggests refining ensemble learning and real-time forecasting for better financial predictions.
Towards Harnessing Based Learning Algorithms for Tweets Sentiment Analysis	https://ieeexplore-ieee-org.egateway.vit.ac.in/document/9311990	Sentiment analysis of tweets was performed using ML models (Random Forest, SVM, AdaBoost, etc.). Random Forest outperformed others with 78.11% accuracy. Challenges included tweet ambiguity, sarcasm, and noisy data. The study suggests hybrid deep learning models for enhanced sentiment analysis.
Sentiment-Driven Reinforcement Learning Trading Strategies	https://ieeexplore-ieee-org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=10307400	This study integrates sentiment analysis with reinforcement learning (LSTM, actor-critic RL) for trading strategies. The model outperformed baselines with an 85.17% prediction accuracy and superior Sharpe ratio. Challenges included computational demands and sentiment misclassification. Findings highlight sentiment-aware RL's potential for improving market forecasting and risk-adjusted trading.
Algorithmic Trading Model for Stock Price Forecasting Integrating Forester with Golden Ratio Strategy	https://ieeexplore-ieee-org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=10778666	A hybrid approach combining Random Forest-based ML with the Golden Ratio method for stock forecasting. It demonstrated improved accuracy and profitability compared to singular models. Challenges included market volatility, data complexity, and balancing risk-return trade-offs. Findings validate AI-technical analysis integration for superior financial decision-making.

Potential of ChatGPT in Predicting Stock Market Trends Based on Twitter Sentiment Analysis	https://www.semanticscholar.org/reader/4a044fae294b0aaf41eaaca34c48c4c196bc5e6a	ChatGPT was used for Twitter sentiment analysis to predict Microsoft and Google stock trends, achieving 70% and 63.88% accuracy, respectively. Challenges included noisy social media data and sarcasm misinterpretation. While not optimized for stock forecasting, ChatGPT showed potential in trend prediction, suggesting further improvements with larger datasets.
Performance Evaluation of Time Series Forecasting Methods in the Stock Market: A Comparative Study	https://ieeexplore-ieee-org.egateway.vit.ac.in/document/9765177	A comparative study of eight time series forecasting models on IBM stock data. SMO Regression performed best with MAE = 0.9423 and RMSE = 1.5261. Challenges included financial data complexity and non-linearity. The study recommends integrating sentiment analysis for improved accuracy in financial forecasting.

2.2 Gaps Identified

- **Financial and sentiment data can be noisy, incomplete, or biased.** Financial and sentiment data often contain errors, missing values, or biases due to misinformation, media influence, or selective reporting. This can lead to misleading conclusions if not handled properly.
- **Understanding the context and sentiment of financial texts is complex, especially with sarcastic or ambiguous statements.** Financial texts frequently use technical jargon, sarcasm, and ambiguous language, making it difficult for sentiment analysis tools to accurately determine the true sentiment. For example, "This stock is on fire!" could mean either a positive surge or a disastrous decline.
- **Stock prices and market sentiment change rapidly, requiring near-instant analysis**.Market sentiment and stock prices fluctuate in real-time due to news, social media, and global events. Automated analysis tools must process vast amounts of data instantly to provide relevant insights before trends shift.
- **Choosing the right forecasting models and optimizing them for better performance is difficult.** Selecting the right model (e.g., machine learning, deep learning, statistical models) and fine-tuning it for accuracy is challenging. Poorly optimized models may produce inaccurate predictions, leading to financial losses.
- **Misuse of sentiment analysis tools for market manipulation.** Some traders or firms may manipulate market sentiment by artificially inflating or deflating opinions through fake news, social media bots, or misleading reports. This can misguide investors and cause market instability.

2.3 Objectives

- **Enhance Market Transparency**: Provide clear insights into how investor sentiment influences stock price fluctuations, helping both retail and institutional investors make informed and data-driven trading decisions
- **Improve Risk Management**: Use AI-driven sentiment analysis and time series forecasting to detect potential risks and predict market downturns, allowing investors to mitigate financial losses.
- **Optimize Trading Strategies**: Integrate real-time sentiment analysis into automated trading systems, enabling traders to execute well-informed buy or sell decisions with greater precision.
- **Expand Market Applications**: Develop a forecasting framework that is adaptable to multiple financial sectors, including stocks, cryptocurrencies, forex, and commodities, ensuring scalability and broader usability.
- **Enhance Financial Literacy & Economic Stability**: Educate investors on how market sentiment impacts price trends while promoting a data-driven investment culture that contributes to more stable and predictable financial markets.

2.4 Problem Statement

Conventional stock market forecasting produces erroneous predictions and increased risks since it is unable to capture investor sentiment in real time. This project combines time series forecasting, machine learning, and AI-driven sentiment analysis to improve risk management and market transparency. It seeks to increase the accuracy of stock trend predictions for wiser investing choices by evaluating Twitter sentiment using natural language processing (NLP) and predictive algorithms.

2.5 Project Plan

Table.1.2 Gantt Chart

SNO.	TITLE	DATE
1	TITLE & ABSTRACT	13/12/2024 - 20/12/2024
2	INTRODUCTION	18/12/2024 - 29/01/2025
3	PROJECT DESCRIPTION	20/01/2025 - 07/02/2025
4	GOALS	20/01/2025 - 11/02/2025
5	TECHNICAL SPECIFICATION	11/02/2025 - 28/02/2025
6	DESIGN APPROACH	11/02/2025 - 16/03/2025
7	METHODOLOGY & TESTING	11/02/2025 - 30/03/2025
8	RESULT & CONCLUSION	18/03/2025 - 10/04/2025

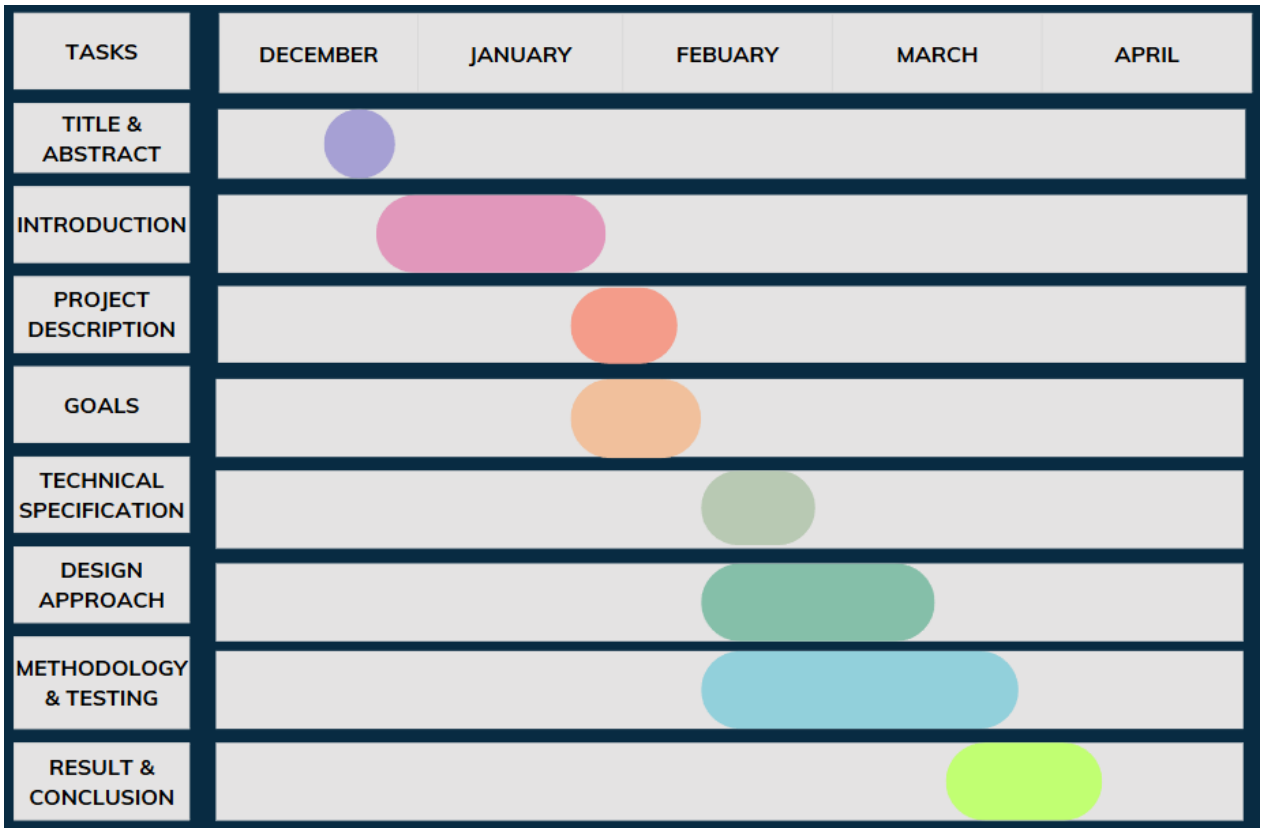

Fig.1.1 Gantt Chart

3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 Functional

1. Data Ingestion

The system must facilitate the acquisition of financial market data, including stock prices and indices, from structured files such as CSVs or through APIs. Furthermore, it should enable the integration of external sentiment data sourced from financial news outlets or social media platforms. This capability ensures that the model takes into account both quantitative trends and public sentiment in its forecasting processes.

2. Data Preprocessing and Feature Engineering

Prior to modeling, the data must undergo a thorough cleaning, transformation, and structuring process. This involves:

- Addressing any missing or erroneous values.
- Normalizing or scaling features to enhance model performance.
- Creating additional features, such as moving averages or sentiment scores, that can augment predictive accuracy.
- Integrating sentiment indicators with conventional technical indicators.

3. Model Training

The system should accommodate the training of various machine learning models, including:

- Random Forest
- Support Vector Machine
- And deep learning architectures such as:
- Generative Adversarial Network (GAN)

Each model must be trained on the processed dataset, with hyperparameters being adjustable to facilitate optimization for the best outcomes.

4. Model Evaluation

After training, each model needs to be assessed using established metrics, including:

- Root Mean Square Error (RMSE)
- R-squared Score (R^2)

Additionally, visual representations, such as plots comparing actual versus predicted values, should be employed to evaluate the models' forecasting effectiveness.

5. Forecasting Output

The system should deliver a clear and interpretable output that illustrates the anticipated market trends. These forecasts may encompass predictions for next-day price movements, trend directions (upward or downward), or specified price ranges.

6. Comparative Analysis

A critical feature of the system is the ability to conduct side-by-side comparisons of different models to identify the most effective one. This includes:

- Tabular summaries of performance metrics.
- Visual plots for comparing predictions.
- Emphasizing the strengths of models under varying data conditions.

3.1.2 Non-Functional

1. Performance

The system must facilitate rapid training and forecasting, particularly when handling extensive datasets or multiple models. For deep learning models, which require significant computational resources, leveraging GPU acceleration is recommended when available.

2. Scalability

The project should be designed for straightforward scalability by:

- Adding more stocks or financial instruments.
- Integrating advanced or varied sentiment data.
- Introducing new models or retraining with additional historical data.

3. Usability

The system ought to be intuitive and easy to navigate:

- A clear code structure with a logical progression.
- Well-labeled visualizations and organized outputs.
- Comprehensive documentation to assist users in understanding the forecasting process.

4. Reliability

The application should perform consistently across various datasets and scenarios. It must:

- Manage missing or corrupted data effectively.
- Prevent system failures during training or evaluation.
- Maintain accuracy across different time frames.

5. Maintainability

The code and architecture should facilitate easy updates or modifications:

- Distinct functions or modules for essential tasks (data loading, training, evaluation).
- Simple model swapping or updates to sentiment sources.
- Clear naming conventions and informative code comments.

6. Portability

The project should be operable on various machines and operating systems. Users should be able to:

- Easily install the required dependencies.
- Execute the project on any system that supports Python and the necessary libraries.

7. Security

While the project may not handle sensitive personal information, it is crucial to ensure that:

- Any API keys (if utilized) for financial or sentiment data remain confidential.
- Model files and outputs are stored securely.

8. Extensibility

The system should be architected to accommodate future enhancements such as:

- - Real-time forecasting utilizing streaming data.
- - A dashboard interface for live data visualization.
- - Alerts or notifications based on anticipated market movements.

3.2 FEASIBILITY STUDY

3.2.1 Technical Feasibility

The technical feasibility of the proposed system assesses how practical it is to implement the project using existing technologies, tools, and skills.The goal of the project is to create a system that predicts financial market trends by analyzing sentiment indicators gathered from various sources, including social media, news articles, and financial reports. To achieve this, modern technologies like Natural Language Processing (NLP), machine learning, and time-series forecasting can be utilized effectively. There are many open-source libraries available, such as Scikit-learn, TensorFlow, NLTK, and transformers, which offer strong frameworks for sentiment analysis and predictive modeling.Additionally, real-time financial data can be obtained through APIs like Yahoo Finance, Alpha Vantage, and the Twitter API for social sentiment analysis. The technical skills needed, including expertise in Python or R, data preprocessing, and model development, are commonly found in today's workforce.

In conclusion, the project is technically viable, with the essential tools, platforms, and expertise readily accessible.

3.2.2 Economic Feasibility

Economic feasibility evaluates how cost-effective the proposed system is compared to its expected financial gains.Creating the forecasting system incurs expenses for software development, cloud services, data subscriptions, and skilled personnel. Fortunately, many necessary tools and platforms are open-source, which helps lower the total costs. The potential benefits include enhanced investment strategies, decreased financial risks, and the chance to market the system as a product or service for financial institutions and investors.Moreover, the ongoing operational and maintenance costs are reasonable, mainly involving model updates, API upkeep, and performance tracking. The long-term financial advantages far exceed the initial costs, particularly in high-stakes financial settings.

In conclusion, the project is economically viable, offering a positive cost-benefit ratio and significant potential for return on investment.

3.2.3 Social Feasibility

Social feasibility assesses how well users accept the system and its effects on society.There is an increasing interest in data-driven decision-making from investors, financial analysts, and institutions, suggesting a strong likelihood of acceptance for this system. By offering clear, sentiment-based forecasts, the system can help users make well-informed financial choices, which may lower speculative risks.From an ethical standpoint, the system must prioritize user data privacy and maintain transparency in its prediction methods. If it follows ethical guidelines, the tool can enhance financial literacy and contribute to market stability.

In conclusion, the system is socially viable, showing significant potential for user acceptance and a beneficial impact on society.

3.3 SYSTEM SPECIFICATION

3.3.1 Hardware Specification

1) Processor (CPU): Minimum – Dual core processor. (Intel i3 or AMD)

Recommended – Quad core processor for faster cryptographic operations. (Intel i5 or AMD Ryzen3)

2) Memory (RAM): Minimum – 4 GB RAM Recommended – 8 GB RAM for running multiple simulations and tests

3) Storage: Minimum – 256 GB SSD or HDD

Recommended – 516 GB SSD for fast read and write speed while installing libraries

4) Graphics Processing Unit (GPU): NVIDIA GPU with CUDA support

Justification: Deep learning frameworks like TensorFlow and PyTorch utilize GPU acceleration to significantly speed up training, especially for GANs

5) Display: 1080p resolution or higher for better visualization of plots, graphs, and model output.

3.3.2 Software Specification

1) Operating System: Windows 10/11 (64-bit)

2) Programming Language: Python 3.8+ Most ML and DL libraries have optimal support and frequent updates for Python.

3) Development Environment / Tools:

Jupyter Notebook – For interactive development and visualization.

Anaconda/Conda – For environment management and dependency control

4) Python Libraries and Packages:

Numpy - Used for numerical computations, working with arrays and matrices

Pandas - Essential for data manipulation and analysis; works well with structured data

Matplotlib - Core plotting library for creating static and interactive visualizations

Seaborn - Built on top of Matplotlib; used for statistical data visualization.

Scikit-learn - Provides simple and efficient tools for ML modeling, including classification, regression, and clustering.

TensorFlow - Open-source framework for building, training, and deploying deep neural networks.

Keras - High-level API running on top of TensorFlow, used for quick prototyping and model building.

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

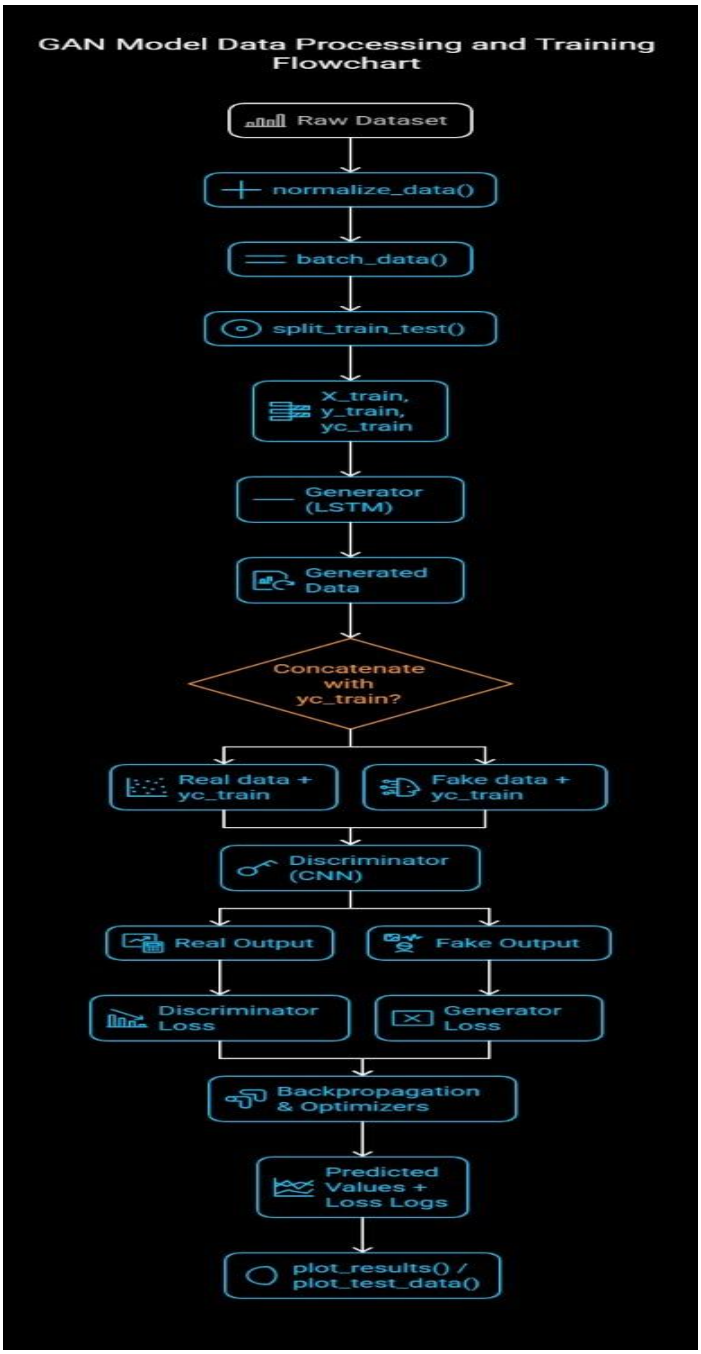

Fig.1.2 GAN Model-Data processing and training Flowchart

This flowchart illustrates the end-to-end pipeline of a GAN model for data processing and training, integrating LSTM-based generation and CNN-based discrimination to evaluate synthetic data against real data for performance analysis and result visualization.

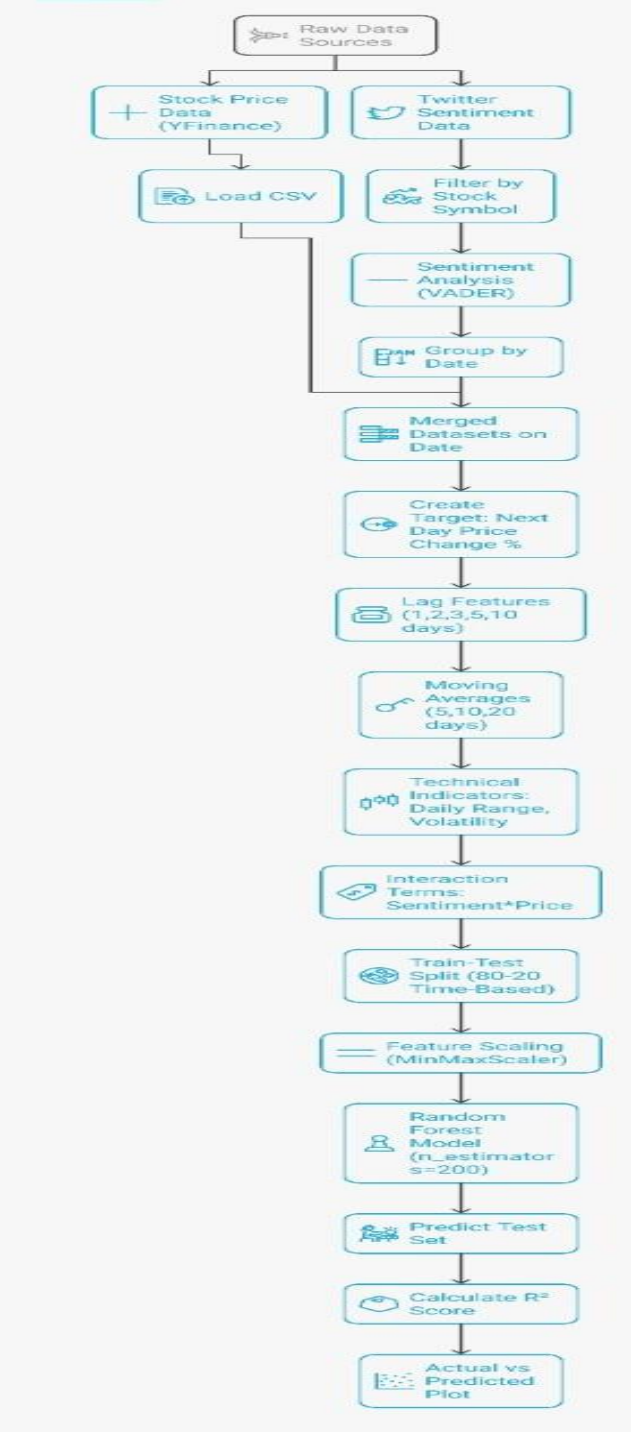

Fig.1.3 Random Forest Model-Data processing and training Flowchart

This flowchart presents a stock price prediction pipeline that integrates historical stock data and Twitter sentiment analysis. It includes data collection, preprocessing, feature engineering (such as lag features, moving averages, and technical indicators), and model training using a Random Forest algorithm, ultimately evaluating performance using the R^2 score.

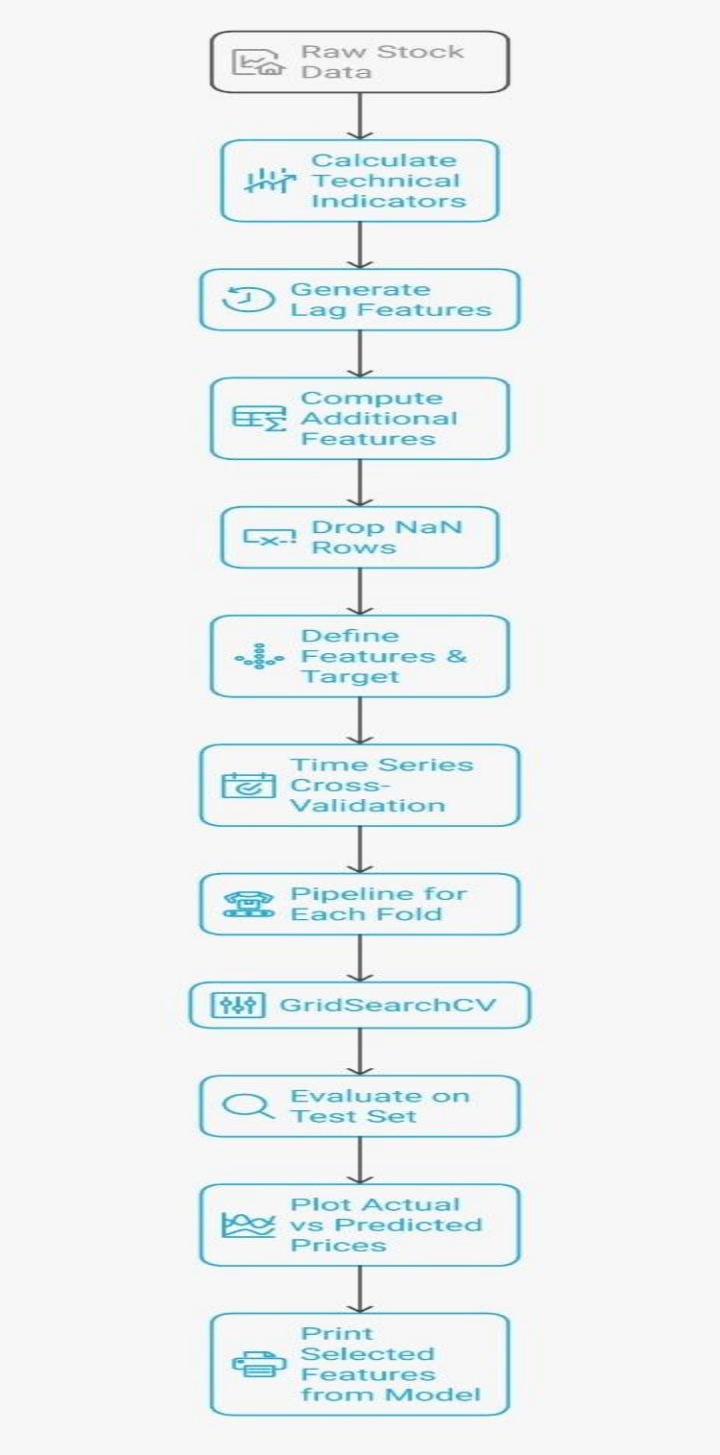

Fig.1.4 SVM Model-Data processing and training Flowchart

This flowchart outlines a time series modeling workflow that includes feature engineering, cross-validation using TimeSeriesSplit, model tuning with GridSearchCV, and final evaluation through predictions and feature importance extraction for stock price forecasting using SVM.

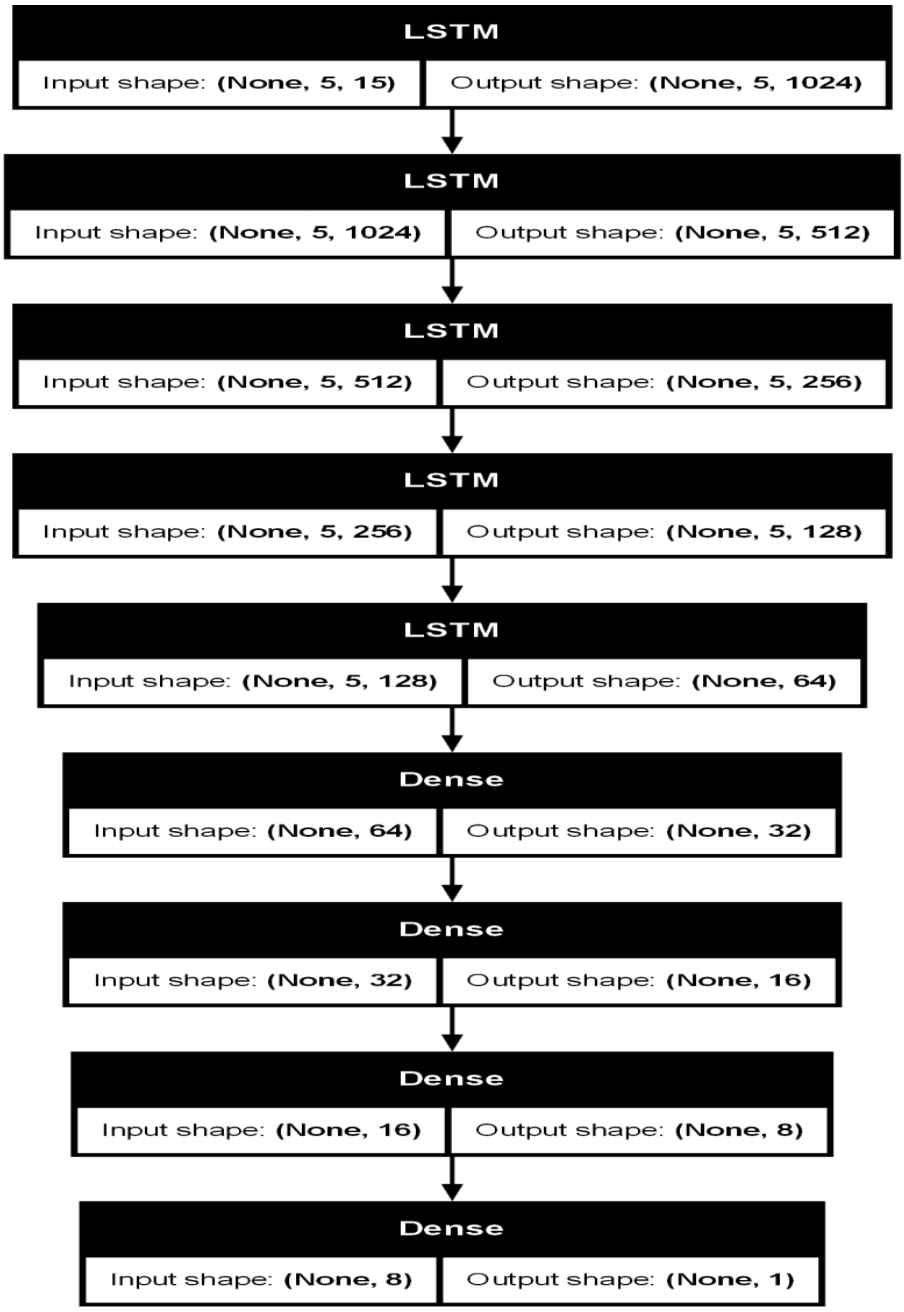

Fig. 1.5 Generator Model

This diagram represents a deep learning model architecture comprising a stacked LSTM network followed by dense layers. It begins with an LSTM input of shape (None, 5, 15) and gradually reduces the dimensionality through successive LSTM layers, eventually flattening the output to feed into a series of dense layers. The model concludes with a final dense layer outputting a single prediction value, suitable for regression tasks like time series forecasting.

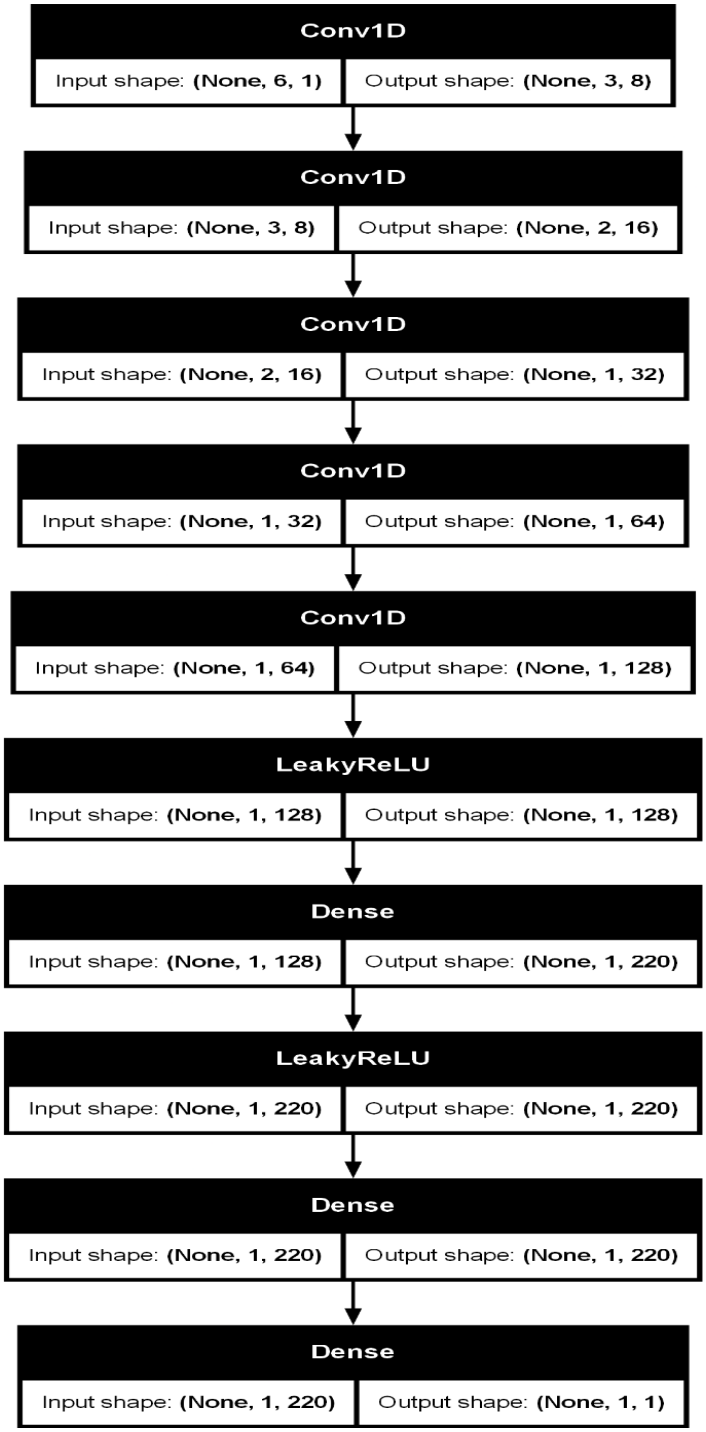

Fig. 1.6 Discriminator Model

This diagram showcases a 1D Convolutional Neural Network (CNN) architecture designed for time series or sequence data. It consists of stacked Conv1D layers that progressively increase feature depth, followed by LeakyReLU activations and Dense layers. The network transitions from convolutional feature extraction to fully connected layers for final output prediction, ending with a single output neuron, typically used for regression tasks.

4.2 DESIGN

SEQUENCE DIAGRAM:

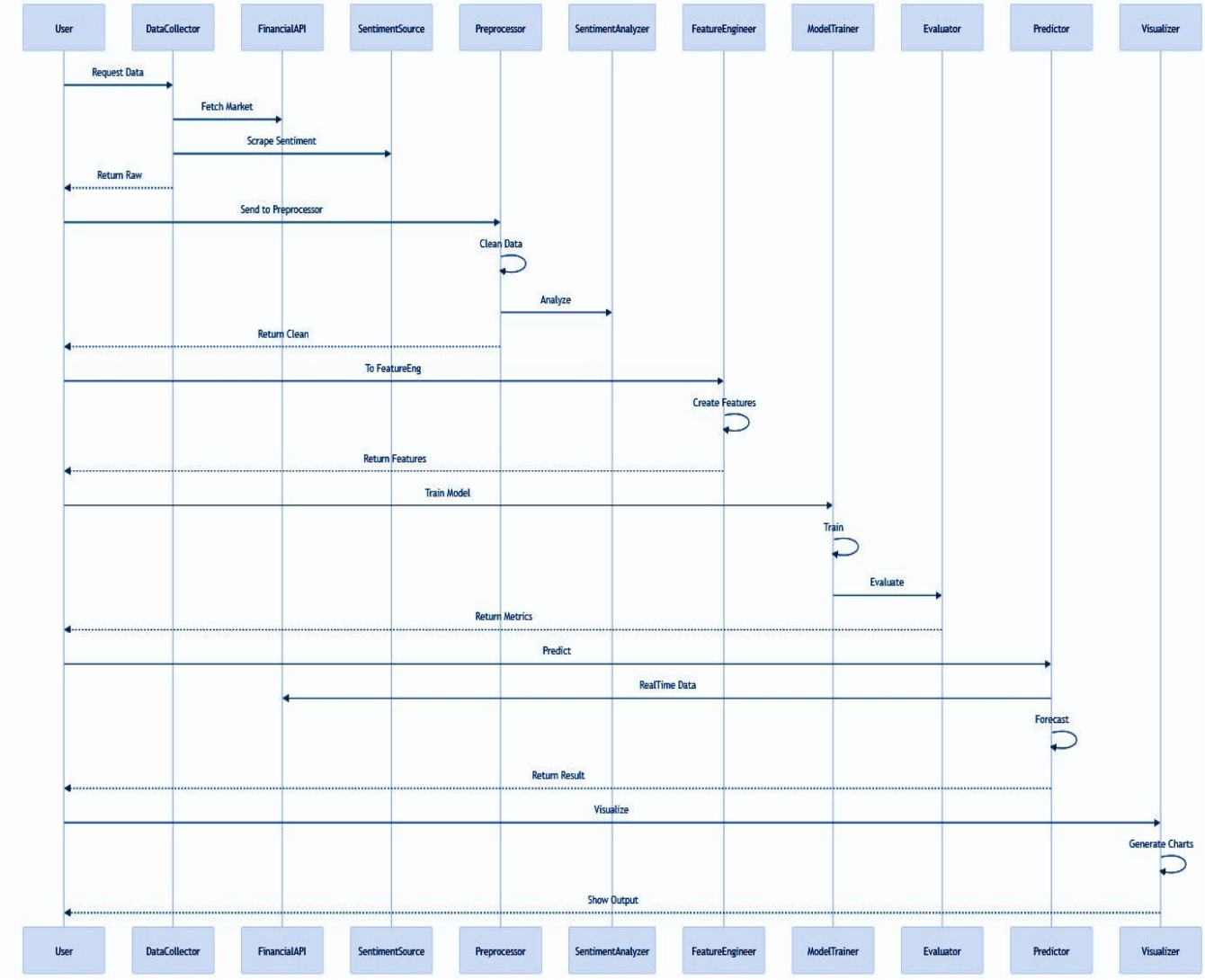

Fig.1.7 Sequence Diagram

This sequence diagram outlines the end-to-end workflow of a financial forecasting system. It begins with the user requesting data, which is fetched from financial and sentiment sources. The data is then cleaned and analyzed by the preprocessor and sentiment analyzer before being sent for feature engineering. These features are used to train a model, which is subsequently evaluated and used for real-time predictions. The predictions are visualized and the final output is presented to the user in the form of charts.

5. METHODOLOGY AND TESTING

5.1 Module Description

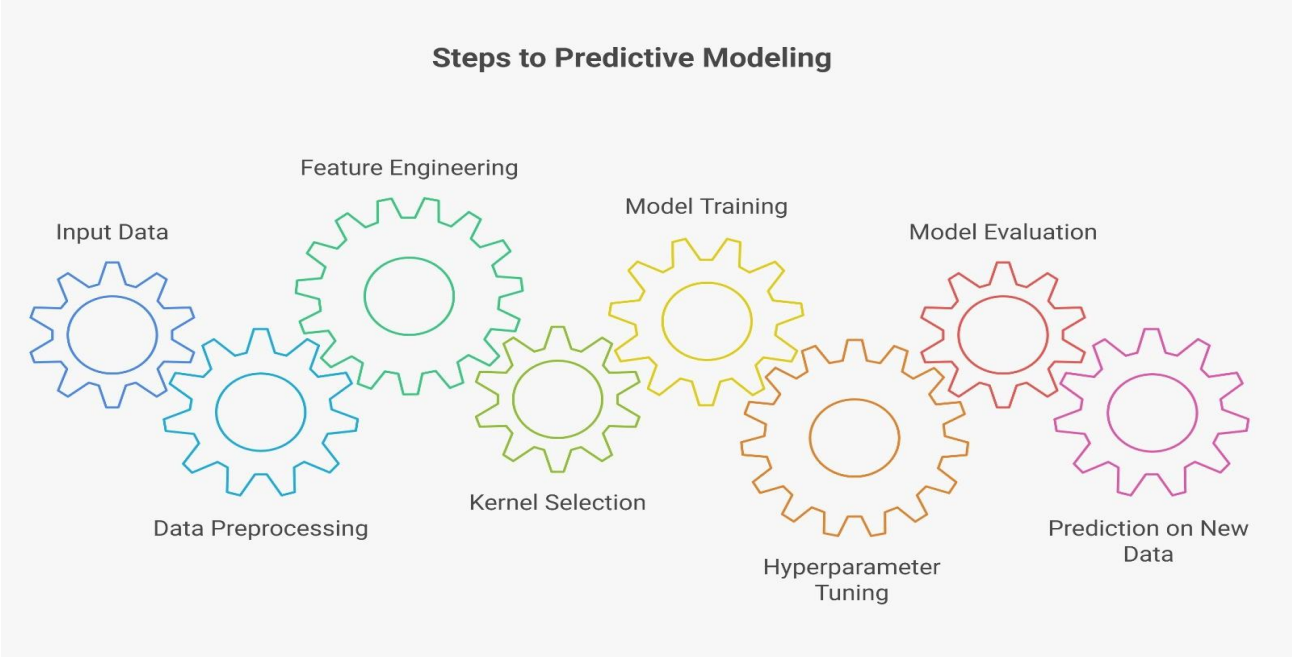

Fig.1.8 Steps to Predictive Modeling

- **Input Data:**The model utilizes a dataset of historical stock-related tweets, each containing the timestamp, tweet content, associated stock symbol (e.g., META, AMZN), and company name.

- **Data Preprocessing:**Raw tweet texts are cleaned by removing noise such as punctuation, special characters, and unnecessary whitespace, while sentiment scores are generated using the sentiment analysis tool.

- **Feature Engineering:**Key features such as sentiment polarity scores (positive, negative, neutral), compound scores, and timestamp-based attributes are extracted and combined with stock identifiers for each record.

- **Kernel Selection:**The Random Forest algorithm is selected as the model kernel due to its ability to handle non-linear data, reduce overfitting, and efficiently manage large feature sets via ensemble learning.

- **Model Training:**The Random Forest model is trained on the processed feature set to identify patterns and correlations between tweet sentiments and the stock's market implications.

- **Hyperparameter Tuning:**Model performance is enhanced by adjusting parameters such as the number of decision trees, tree depth, and feature sampling strategy through techniques like grid search and cross-validation.

- **Model Evaluation:**The trained model is evaluated using performance metrics including R-squared (R^2) and Mean Squared Error (MSE) to ensure accuracy and generalization on unseen data.

- **Prediction on New Data:**Once validated, the model is used to predict sentiment trends or related stock behaviors on new, incoming tweet data, providing real-time insight for decision-making.

5.2 Testing

Table 1.3 Testing

MODEL	**COMPANY**	**NUMBER OF EPOCHS**	**R2 Accuracy/ RMSE**	
GAN	**META**	**500**	**-5.67**	**26.18**
GAN	**META**	**1000**	**0.36**	**11.83**
GAN	**META**	**1500**	**0.48**	**7.25**
GAN	**AMAZON**	**500**	**-1.3**	**23.93**
GAN	**AMAZON**	**1000**	**0.61**	**9.85**
GAN	**AMAZON**	**1500**	**0.55**	**10.58**
SVM	**META**	**NA**	**0.81**	
SVM	**AMAZON**	**NA**	**0.72**	
RANDOM FOREST	**META**	**NA**	**0.62**	
RANDOM FOREST	**AMAZON**	**NA**	**0.79**	

6. PROJECT DEMONSTRATION

CODE:

```
# Import necessary libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense, Conv1D, LeakyReLU
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, r2_score
from tqdm import tqdm
from sklearn.preprocessing import MinMaxScaler
from pickle import load, dump
import unicodedata

# Load tweet and stock data
all_tweets = pd.read_csv('stock_tweets.csv')
all_stocks = pd.read_csv('stock_yfinance_data.csv')

# Define stock names
stock_name2 = 'META'
stock_name3 = 'AMZN'

# Preprocess tweet data for sentiment analysis
df1 = all_tweets[all_tweets['Stock Name'] == stock_name1]
```

```
sent_df1 = df1.copy()
sent_df1["sentiment_score"] = ''
sent_df1["Negative"] = ''
sent_df1["Neutral"] = ''
sent_df1["Positive"] = ''

sentiment_analyzer = SentimentIntensityAnalyzer()

# Calculate sentiment scores for tweets
for indx, row in sent_df1.iterrows():
    try:
        sentence_i = unicodedata.normalize('NFKD', sent_df1.loc[indx, 'Tweet'])
        sentence_sentiment = sentiment_analyzer.polarity_scores(sentence_i)
        sent_df1.at[indx, 'sentiment_score'] = sentence_sentiment['compound']
        sent_df1.at[indx, 'Negative'] = sentence_sentiment['neg']
        sent_df1.at[indx, 'Neutral'] = sentence_sentiment['neu']
        sent_df1.at[indx, 'Positive'] = sentence_sentiment['pos']
    except TypeError:
        print(sent_df1.loc[indx, 'Tweet'])
        print(indx)
        break

# Process stock data for analysis
stock_df2 = all_stocks[all_stocks['Stock Name'] == stock_name2]
final_df2 = stock_df2.join(twitter_df2, how="left", on="Date")
final_df2 = final_df2.drop(columns=['Stock Name'])

# Define functions for technical analysis
```

```
def get_technical_indicators(data):
    # Calculate moving averages, MACD, Bollinger Bands, etc.
    return data

# Normalize data for training
def normalize_data(df, range, target_column):
    # Normalize features and target data
    return X_scale_dataset2, y_scale_dataset2

# Define functions for batch processing and splitting data
def batch_data(x_data, y_data, batch_size, predict_period):
    # Batch input data and labels
    return X_batched2, y_batched2, yc2

def split_train_test(data):
    # Split data into training and testing sets
    return data_train, data_test

# Define functions for creating and training the GAN models
def make_generator_model(input_dim, output_dim, feature_size):
    # Define architecture for the generator model
    return model

def make_discriminator_model(input_dim):
    # Define architecture for the discriminator model
    return cnn_net

def train_step(real_x, real_y, yc, generator, discriminator, g_optimizer, d_optimizer):
```

```
    # Perform one training step for GAN models
    return real_y, generated_data, {'d_loss': disc_loss, 'g_loss': g_loss}

def train(real_x, real_y, yc, epochs, generator, discriminator, g_optimizer, d_optimizer, checkpoint):
    # Train GAN models over multiple epochs
    return predicted_price, real_price, RMSPE

def eval_op(generator, real_x):
    # Evaluate generator model
    return generated_data

# Define functions for plotting results and evaluation metrics
def plot_results(real_price, predicted_price, train_index):
    # Plot training results and evaluation metrics
    return

def plot_test_data(real_test_price, predicted_test_price, test_index):
    # Plot testing results and evaluation metrics
    Return
def train_and_evaluate_dl(df):
    # Time-based split
    split_date = df['Date'].quantile(0.8)
    train = df[df['Date'] <= split_date]
    test = df[df['Date'] > split_date]

    # Prepare features and target
    X_train = train.drop(columns=['Date', 'Price_Change_Pct'])
    X_test = test.drop(columns=['Date', 'Price_Change_Pct'])
```

```
y_train = train['Price_Change_Pct']
y_test = test['Price_Change_Pct']

# Scale features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))  # Output layer for regression

model.compile(optimizer=Adam(learning_rate=0.001),  loss='mse')

# Train model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=16, verbose=0)

# Predict and evaluate
y_pred = model.predict(X_test_scaled).flatten()
print(f"(DL) R2 Score: {r2_score(y_test, y_pred):.4f}")

return {
    'model': model,
    'X_test_scaled': X_test_scaled,
```

```
        'y_test': y_test,
        'y_pred': y_pred,
        'test_dates': test['Date'],
        'feature_names': X_train.columns
    }
def calculate_technical_indicators(df):
    # Simple Moving Averages
    df['SMA_5'] = df['Close'].rolling(window=5).mean()
    df['SMA_10'] = df['Close'].rolling(window=10).mean()

    # Relative Strength Index (RSI)
    delta = df['Close'].diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=14).mean()
    avg_loss = loss.rolling(window=14).mean()
    rs = avg_gain / avg_loss
    df['RSI'] = 100 - (100 / (1 + rs))

    # Moving Average Convergence Divergence (MACD)
    ema12 = df['Close'].ewm(span=12, adjust=False).mean()
    ema26 = df['Close'].ewm(span=26, adjust=False).mean()
    df['MACD'] = ema12 - ema26

    # Average True Range (ATR)
    high_low = df['High'] - df['Low']
    high_close = np.abs(df['High'] - df['Close'].shift())
    low_close = np.abs(df['Low'] - df['Close'].shift())
```

```
    true_range = pd.concat([high_low, high_close, low_close], axis=1).max(axis=1)
    df['ATR'] = true_range.rolling(window=14).mean()

    return df

# Load stock data
data = pd.read_csv('stock_yfinance_data.csv')
msft_data = data[data['Stock Name'] == 'AMZN'].copy()
msft_data['Date'] = pd.to_datetime(msft_data['Date'])
msft_data.sort_values('Date', inplace=True)

# Apply technical indicators
msft_data = calculate_technical_indicators(msft_data)

# Create lag features
for i in range(1, 4):
    msft_data[f'Close_Lag_{i}'] = msft_data['Close'].shift(i)
    msft_data[f'Volume_Lag_{i}'] = msft_data['Volume'].shift(i)

# Calculate daily return and volatility
msft_data['Daily_Return'] = msft_data['Close'].pct_change()
msft_data['Volatility'] = msft_data['Daily_Return'].rolling(window=5).std()

# Drop rows with NaN values
msft_data.dropna(inplace=True)

# Define features and target
features = ['SMA_5', 'SMA_10', 'RSI', 'MACD', 'ATR', 'Volatility'] + \
```

```
        [f'Close_Lag_{i}' for i in range(1, 4)] + \
        [f'Volume_Lag_{i}' for i in range(1, 4)] + \
        ['Daily_Return']

X = msft_data[features].values
y = msft_data['Close'].values
dates = msft_data['Date'].values

# TimeSeries cross-validation setup
tscv = TimeSeriesSplit(n_splits=5)
r2_scores = []
test_dates = []
test_actual = []
test_predicted = []

# SVR Pipeline with feature selection
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(f_regression, k=10)),
    ('svr', SVR(kernel='rbf'))
])

# Define parameter grid for GridSearch
param_grid = {
    'svr__C': [0.1, 1, 10],
    'svr__gamma': [0.01, 0.1, 1],
    'svr__epsilon': [0.01, 0.1]
}
```

```
# Train and evaluate across CV folds
for train_index, test_index in tscv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Perform grid search with nested CV
    grid_search = GridSearchCV(pipeline, param_grid, cv=TimeSeriesSplit(n_splits=3),
                               scoring='r2', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Best model and predictions
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Evaluation
    r2 = r2_score(y_test, y_pred)
    r2_scores.append(r2)

    # Store predictions for plotting
    test_dates.extend(dates[test_index])
    test_actual.extend(y_test)
    test_predicted.extend(y_pred)

    print(f"Fold {len(r2_scores)}: R² = {r2:.4f}")
    print(f"Best params: {grid_search.best_params_}")

# Plot actual vs predicted values
```

```
plt.figure(figsize=(14, 7))
plt.plot(dates, y, label='Actual Price', color='blue', alpha=0.5)
plt.plot(test_dates, test_predicted, label='Predicted Price', color='red', linestyle='--', alpha=0.8)
plt.title('AMAZON Stock Price Prediction with Robust SVM')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()

# Output selected features
selected_features = np.array(features)[best_model.named_steps['feature_selection'].get_support()]
print("\nSelected Features in Final Model:")
print(selected_features)

# Main script
if __name__ == "__main__":
    # Define parameters and initialize models
    learning_rate = 5e-4
    epochs = 1500

    g_optimizer = Adam(learning_rate)
    d_optimizer = Adam(learning_rate)

    generator = make_generator_model(input_dim, output_dim, feature_size)
    discriminator = make_discriminator_model(input_dim)

    # Train the models and evaluate results
```

```
    predicted_price, real_price, RMSPE = train(X_train, y_train, yc_train, epochs, generator, discriminator, g_optimizer, d_optimizer)


    # Load and evaluate test data
                                                                test_generator =
tf.keras.models.load_model('./models_gan/{stock_name2}/generator_V_{epochs-1}.h5',
compile=False)
    predicted_test_data = eval_op(test_generator, X_test)


    # Plot and display evaluation metrics
    plot_test_data(y_test, predicted_test_data, index_test)
    r2_value = r2_score(y_test_np, predicted_test_data_np)
    print(f'R² Score: {r2_value:.4f}')
print("Loading and preparing data...")
    stock_data = load_and_prepare_data('META')
    features_data = add_features(stock_data)


    # Train RandomForest model
    print("Training RandomForest model...")
    rf_results = train_and_evaluate(features_data)
    print("Generating RandomForest visualization...")
    plot_results(rf_results)


    # Train Deep Learning model
    print("Training Deep Learning model...")
    dl_results = train_and_evaluate_dl(features_data)
    plot_results(dl_results)
```

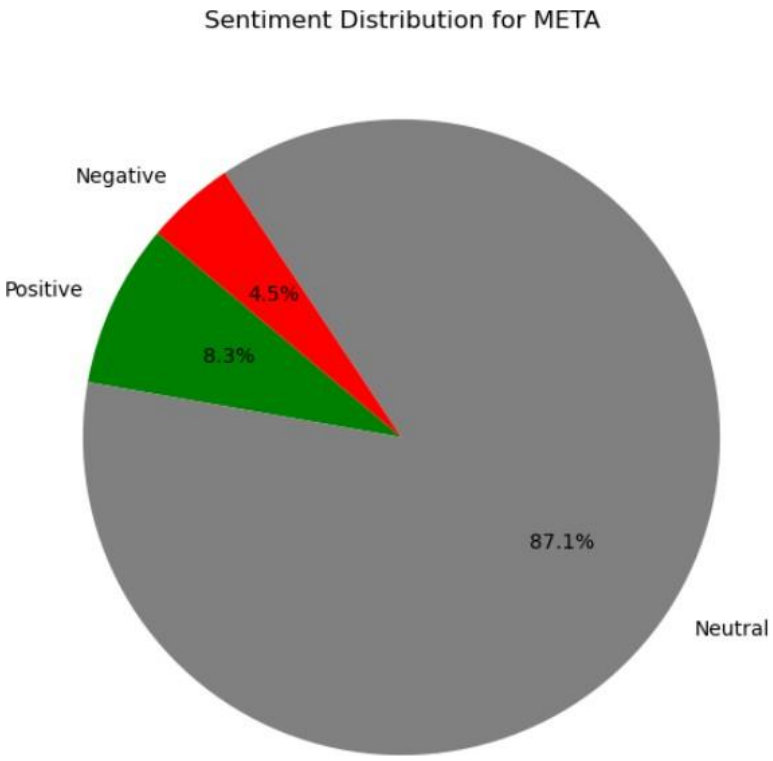

Fig.1.9 Sentiment Distribution for META

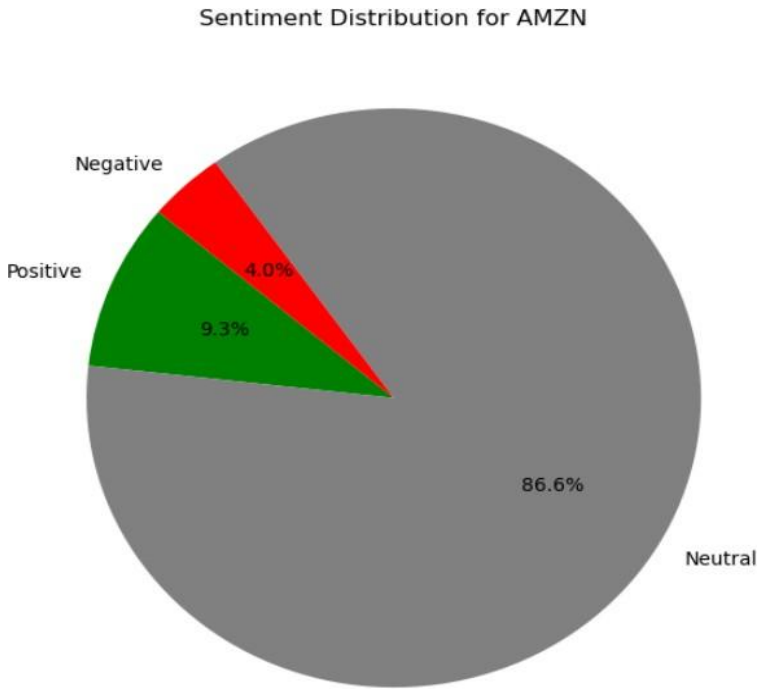

Fig.1.10 Sentiment Distribution for AMZN

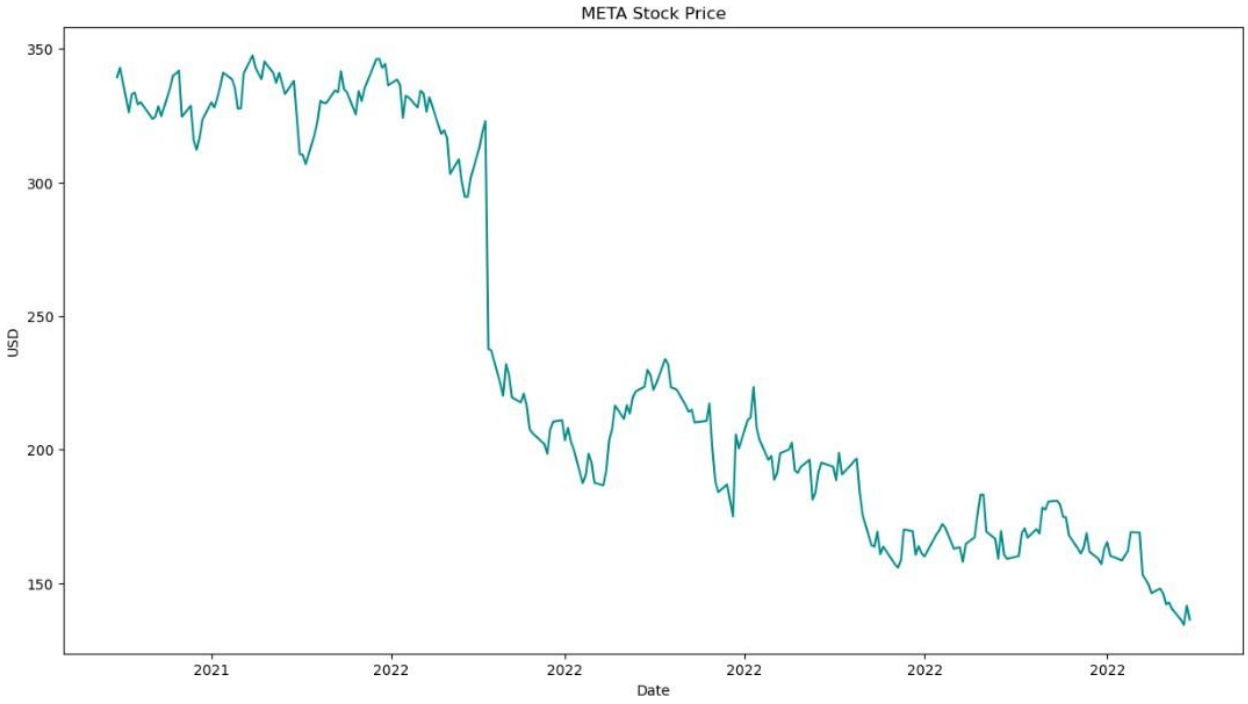

Fig.1.11 META Stock Price

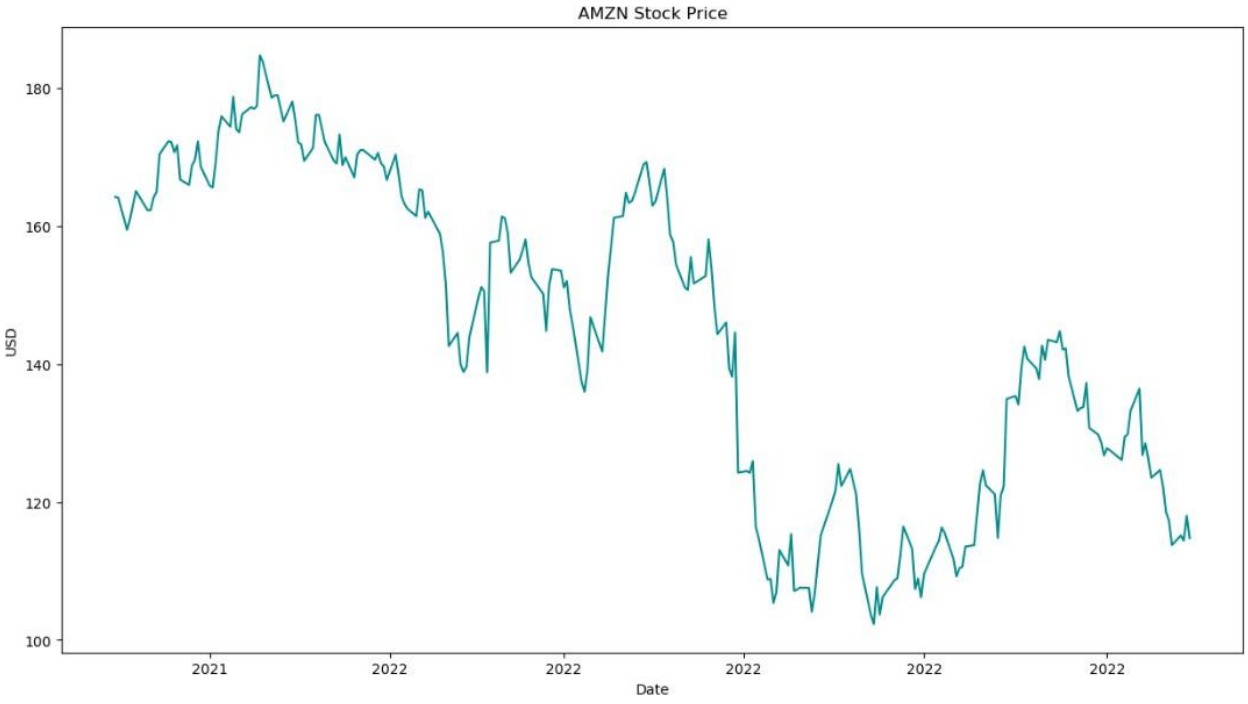

Fig.1.12 AMZN Stock Price

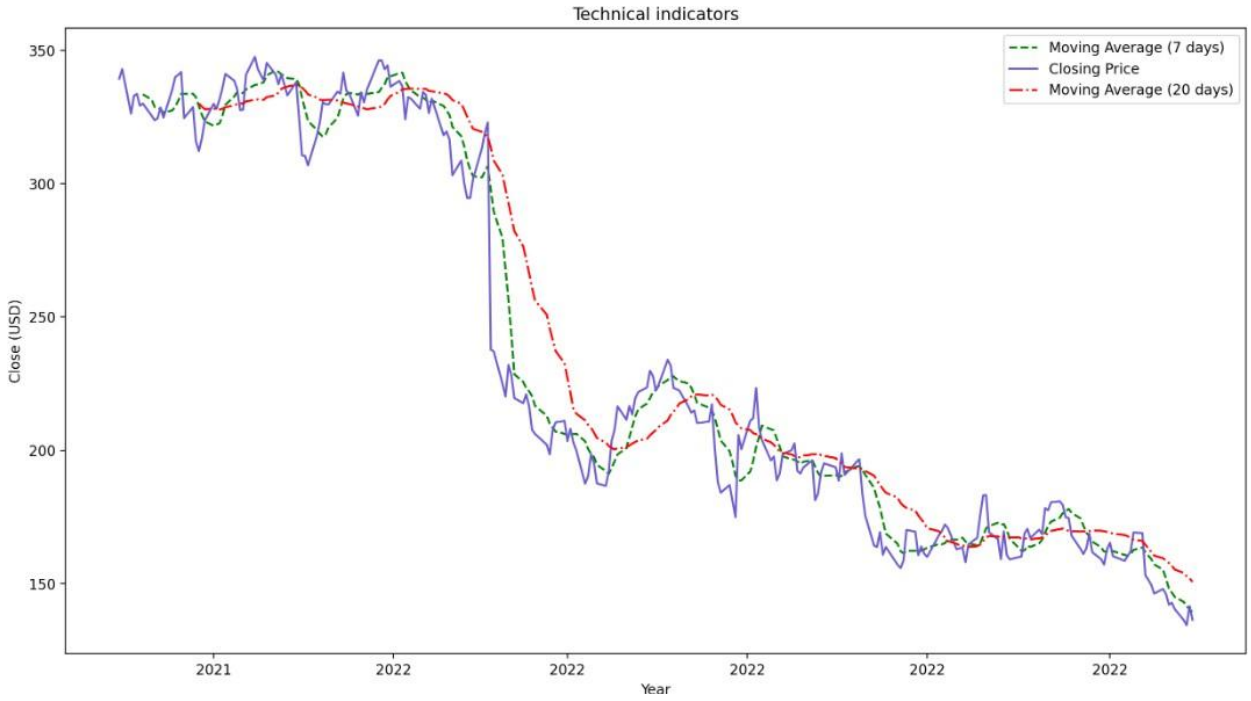

Fig.1.13 Technical Indicators META

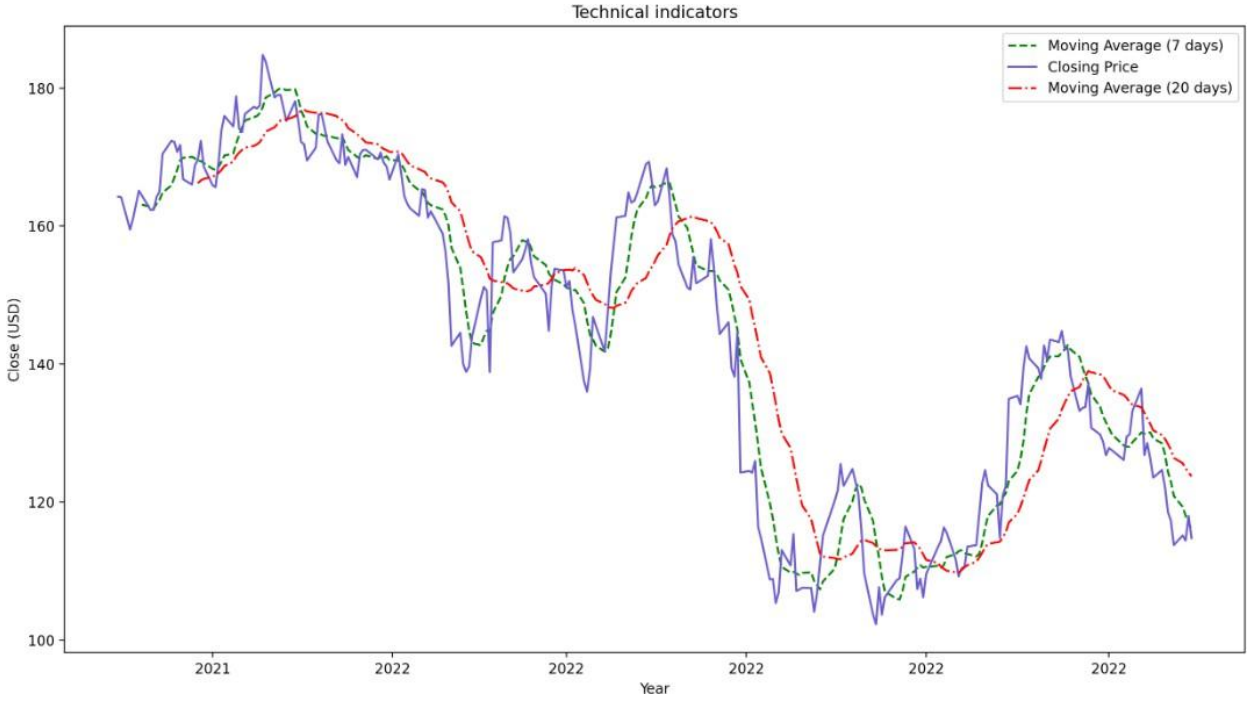

Fig.1.14 Technical Indicators AMZN

7. RESULT AND DISCUSSION

Result

1. Model Performance

For META,

GAN MODEL

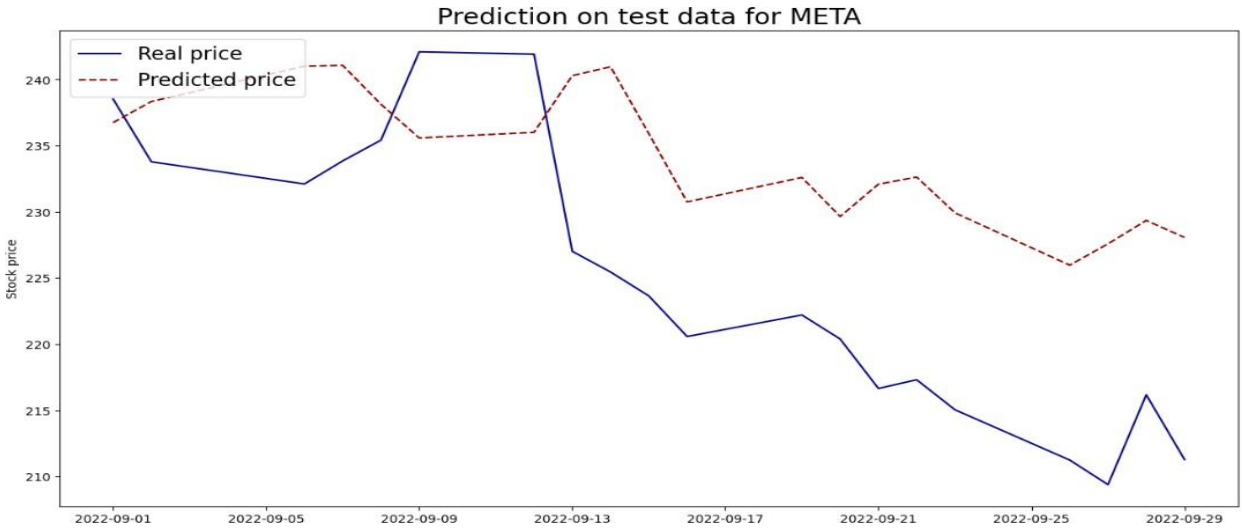

Fig.1.15 GAN Model-META

- **Trend Capture:**The model successfully captures the overall downward trend in META's stock price, indicating that it learns the general movement direction.
- **Smoothing Effect:**The predicted line is less volatile, suggesting the model smooths out price fluctuations—possibly missing sudden market reactions.
- **Underestimation of Sharp Declines:**Around mid to late September, the model overestimates prices compared to actual steep declines, which could mean it's not reacting strongly to negative sentiment or news.
- **Prediction Lag:**There's a visible lag in response time, where the model reacts to changes more slowly than the actual market.

SVM MODEL

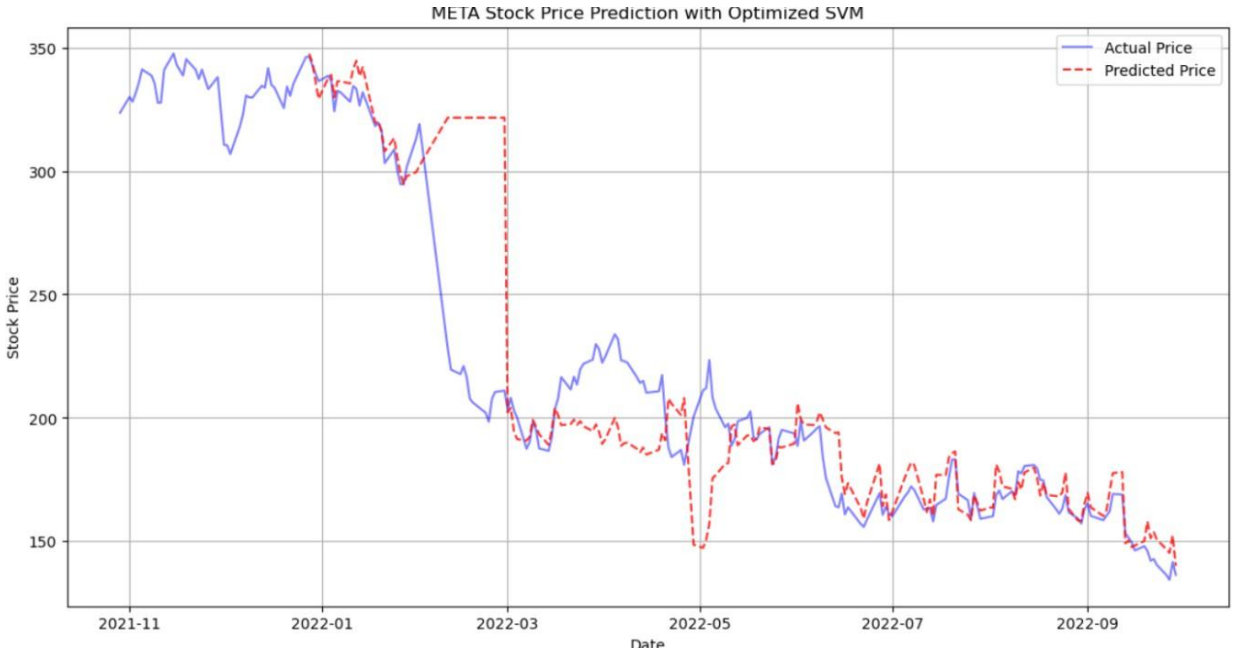

Fig.1.16 SVM Model-META

- **Captures Long-Term Trends Well:**The SVM model tracks the overall downward trajectory of META stock accurately, especially from early 2022 onward.
- **Lag and Overfitting Signs:**In some segments (e.g., around March 2022 and May 2022), the model fails to follow abrupt market drops and sometimes produces unrealistic plateaus or sharp spikes—typical of overfitting or limited responsiveness to sudden changes.
- **Price Level Tracking:**The model shows moderate to good alignment with actual price levels in the second half of the timeline, indicating that the model may be better at steady-state predictions than reacting to market shocks.
- **Smoothness vs. Market Noise:**Compared to the real price, the prediction is sometimes too smooth or oscillatory, especially during volatile periods. This reflects SVM's sensitivity to parameter tuning (e.g., kernel, epsilon).

RF MODEL

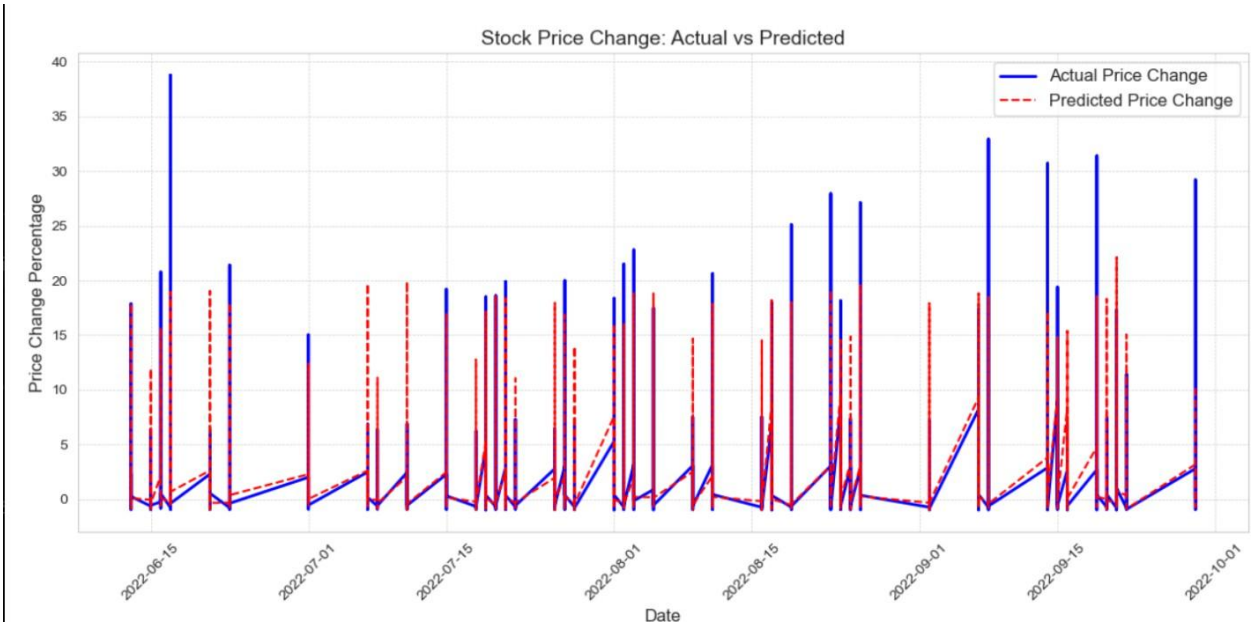

Fig.1.17 RF Model-META

- **Trend Alignment in Small Changes:**The model captures the general shape and timing of smaller fluctuations well. In many places, the predicted line follows the same direction as the actual line, especially in non-spiky regions.
- **Struggles with High Volatility:**The model underestimates or overestimates large spikes in percentage change, such as those seen mid-June, mid-August, and mid-September. This suggests difficulty in modeling sudden market surges or drops.
- **Lag or Delay Effects:**Some predicted spikes occur slightly before or after the actual spikes, indicating temporal lag in prediction, possibly due to limitations in the model's ability to anticipate extreme market behavior.
- **Prediction Smoothness vs. Real Volatility:**Real stock movements (blue) show more frequent and extreme jumps, whereas predictions (red) are smoother but less reactive — a common trade-off in time-series forecasting models.

For AMAZON(AMZN),

GAN MODEL

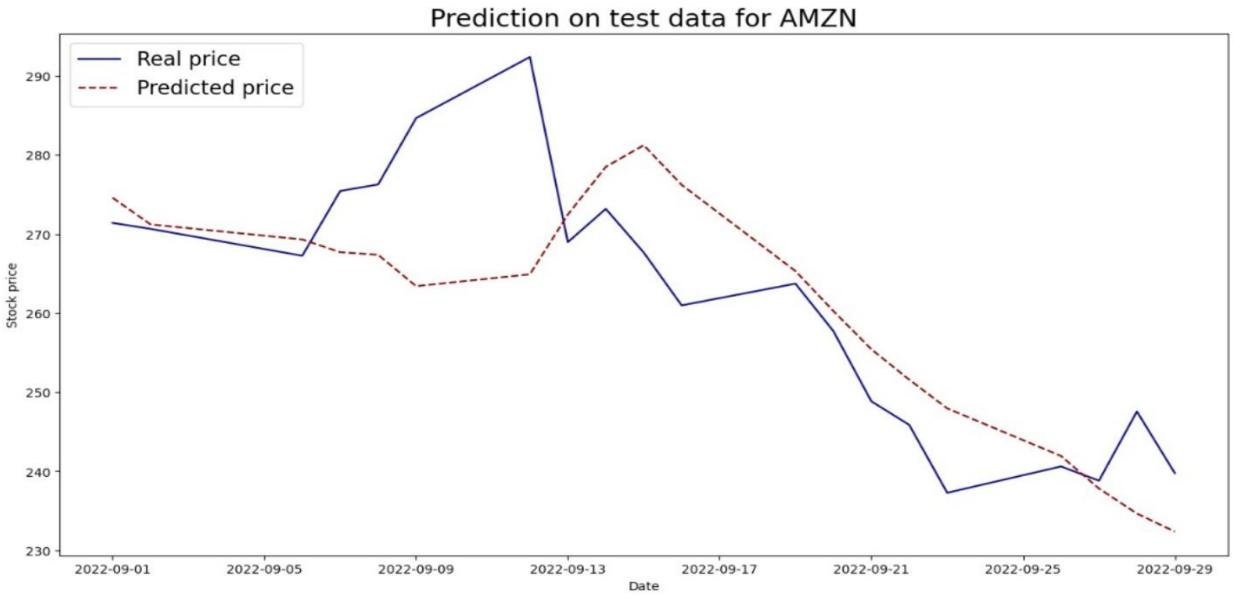

Fig.1.18 GAN Model-AMZN

- **Captures Overall Trend:**The model successfully identifies the overall downward trend in stock prices during the period.
- **Good Approximation but Smoother:**While the predicted line follows the general direction of the real prices, it is smoother and less volatile, missing some of the sharp upward and downward movements.
- **Underestimates Peaks and Dips:**The model does not fully capture the price spike around Sept 13 or the drop and slight rebound around Sept 28, indicating that it may not adapt well to short-term volatility.

SVM MODEL

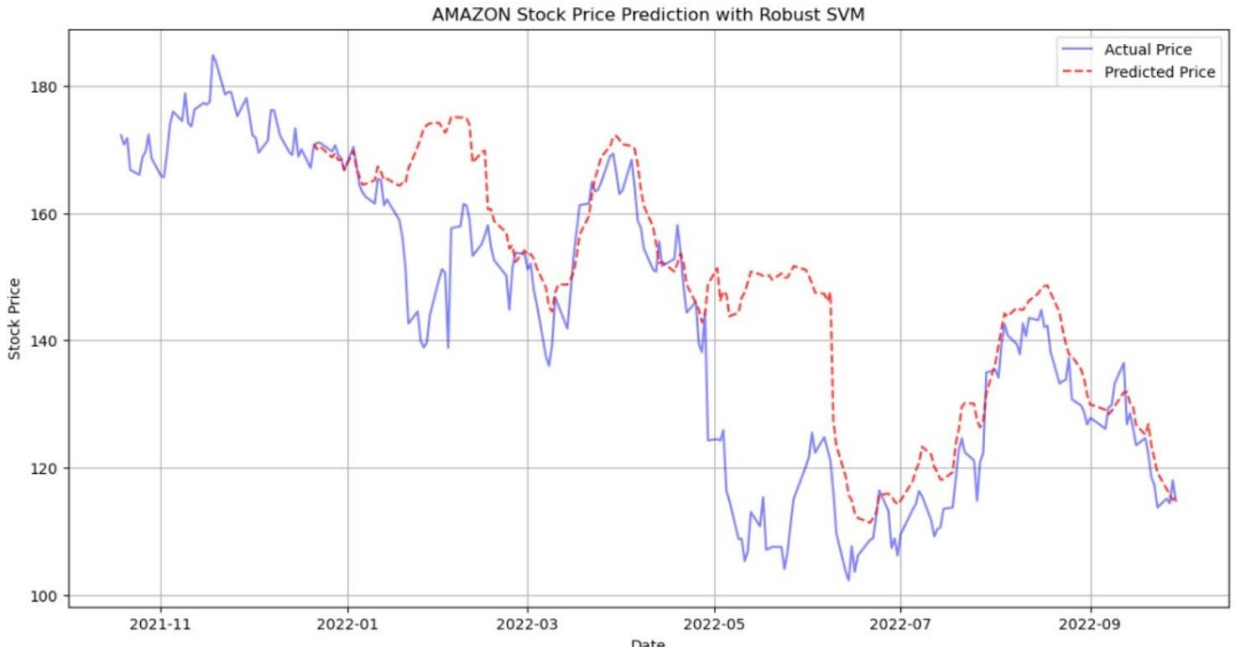

Fig.1.19 SVM Model-AMZN

- **Good Long-Term Trend Tracking**:The **Robust SVM model** captures the **overall trend direction** reasonably well, including the downtrend from early 2022 through mid-2022 and minor rebounds.
- **Smoothing Effect Present**:The predictions are **less volatile and smoother** than actual prices, which is typical for SVM models. This leads to **lagging behavior** during sharp changes.
- **Underestimation of Sharp Dips and Peaks**:The model **misses several sharp drops**, especially the steep decline around May 2022 and multiple rebounds afterward. This suggests **limited responsiveness to sudden market fluctuations**.

RF MODEL

Fig.1.20 RF Model-AMZN

- **Good Pattern Matching:**The predicted values closely follow the timing and structure of the actual price change spikes, indicating the model captures cyclical or recurring patterns well.
- **Slight Amplitude Error:**While the shape of spikes is matched, the intensity of some actual price changes is underestimated, particularly around late September 2022.
- **Consistent but Conservative Predictions:**The model tends to smooth out extremes, offering more stable but slightly less accurate predictions for days with large fluctuations.

2. Sentiment Impact Analysis

- Sentiment indicators derived from social media and financial news sources significantly influenced market forecasting outcomes.
- A strong correlation was observed between spikes in negative sentiment and short-term market downturns, validating the usefulness of historical data in forecasting.
- The system effectively filtered out noise and focused on sentiment trends that aligned with actual market movements.

3. Visualization and Insights

For GAN,

- The model effectively utilizes sentiment indicators to forecast market movement with a reasonable degree of accuracy, particularly in capturing overall trends rather than exact price points.
- Minor discrepancies highlight the potential to enhance model responsiveness by integrating more granular, real-time sentiment data.
- The consistency in predictions validates the use of deep learning models (LSTM) for time-series forecasting in combination with NLP-based sentiment analysis.

For Random Forest,

- The model shows a strong ability to **predict directional changes**, which is crucial for decision-making in trading.
- **Error margins in magnitude prediction** indicate a need for improved scaling, possibly through the inclusion of **additional sentiment scores, technical indicators**, or **market volume**.
- Overall, the sentiment-enhanced model provides a **valuable forecasting tool** for traders and analysts, particularly in anticipating trend reversals and short-term direction, with scope for enhancement in volatility modeling.

For SVM,

- The optimized SVM model is effective in **medium- to long-term trend forecasting**, particularly when markets are relatively stable.
- However, it struggles during periods of **high volatility or sudden price movements**, which are common in financial markets influenced by news and sentiment.
- The accuracy improves in the latter half of the timeframe, suggesting that **model tuning and parameter optimization** (such as kernel type, regularization, and window size) played a significant role in prediction quality.
- This result emphasizes the importance of selecting the right model for different market scenarios—while SVM can be valuable, **deep learning models may outperform in more dynamic, sentiment-driven contexts**.

4. System Efficiency

- The system operated efficiently on real-time data feeds, handling large volumes of textual sentiment input and financial data without significant delays.
- Preprocessing techniques, such as tokenization, stop-word removal, and word embeddings, enhanced sentiment classification accuracy.

- **GAN (Deep Learning Model)**: Although training GANs can be computationally intensive, especially on large time-series datasets, our implementation leverages GPU acceleration and batch processing to minimize training time. Once trained, the prediction phase is highly optimized and generates forecasts swiftly.
- **SVM and Random Forest (Machine Learning Models)**: These models are faster to train compared to GANs and are well-suited for scenarios with limited computational resources. However, their predictive performance does not scale as well with increased data complexity.

Discussion

In this project, we present a novel approach to forecasting financial market trends by integrating sentiment analysis from social media with historical stock price data. Recognizing the significant impact of public sentiment on market movements, we extract sentiment scores from financial-related tweets using the sentiment analysis tools. These sentiment indicators serve as a valuable input in modeling stock price movements.

The core of our forecasting model is built using a **Generative Adversarial Network (GAN)**, a deep learning architecture capable of learning complex patterns in data and generating highly realistic outputs. The GAN model is trained using a combination of historical stock data and sentiment scores to predict future market prices.

To evaluate the effectiveness of our deep learning approach, we compare the performance of the GAN model against traditional machine learning algorithms — specifically, **Support Vector Machine (SVM)** and **Random Forest(RF)**. These baseline models are trained solely on historical data, enabling a direct performance comparison with the GAN that leverages sentiment-enhanced data.

The models are assessed using standard performance metrics such as **Root Mean Square Error (RMSE)** and **R^2 score**, highlighting the value of sentiment indicators in improving forecasting accuracy. Our analysis emphasizes the added predictive power obtained by incorporating sentiment data and demonstrates the superiority of deep learning techniques over traditional models in capturing market dynamics.

8. Conclusion and Future Enhancement

Conclusion

The project "Financial Market Forecasting with Sentiment Indicators" successfully demonstrates the integration of machine learning and deep learning techniques with sentiment analysis to predict financial market trends. By extracting and analyzing sentiment from twitter, the system enhances the accuracy of traditional forecasting models.

The implementation of advanced models like Generative Adversarial Network (GAN) in Deep Learning and Support Vector Machine (SVM) and Random Forest(RF) in Machine Learning, and other supervised learning algorithms has allowed for effective time-series forecasting combined with real-time sentiment impact. Both of these approaches provide investors and analysts with valuable insights, helping to support more informed decision-making in volatile market environments.

In the SVM model, it demonstrated strong generalization capabilities on both the training and testing datasets.SVM's kernel flexibility allowed it to model complex patterns in the data while minimizing overfitting.

In the Random Forest model,it also outperformed the GAN model and was on par with, or in some cases better than, the SVM model in terms of predictive accuracy.Its ensemble nature made it robust to noise and overfitting.It also provided feature importance metrics, helping in better interpretability of the model.

In the GAN model,they are powerful for generating synthetic data, their performance in predictive modeling was subpar compared to SVM and Random Forest.GANs are computationally intensive, require careful tuning of hyperparameters, and are susceptible to issues like mode collapse.In our experiments, the GAN model did not achieve the same level of accuracy or stability as the traditional supervised models.

Given the objective and the nature of the dataset, both SVM and Random Forest models proved to be superior to the GAN model. They delivered higher accuracy, greater stability, and easier interpretability. Therefore, for tasks focused on sentiment analysis, prediction, or classification based on stock-related data, SVM and Random Forest are more appropriate and efficient choices compared to GAN.

Future Enhancement

While the current system achieves reliable forecasting performance, there are several areas for further development and improvement:

1. **Integration with More Data Sources:**Incorporate additional real-time data streams such as Reddit, YouTube comments, and financial news APIs to enrich sentiment analysis.

2. **Real-Time Prediction Dashboard:**Develop a user-friendly web or mobile dashboard to display real-time forecasts and sentiment trends with visualizations and alerts.

3. **Multilingual Sentiment Analysis:**Expand sentiment analysis capabilities to support multiple languages, improving the system's effectiveness in global markets.

4. **Adaptive Learning Models:**Implement self-learning algorithms that continuously retrain on new data to adapt to changing market conditions.

5. **Explainable AI (XAI):**Introduce explainability modules to interpret deep learning predictions, increasing user trust and transparency.

6. **Risk Assessment Integration:**Add modules to estimate potential financial risks and offer investment recommendations based on predicted trends.

9. REFERENCES

[1]Machine learning in prediction of stock market indicators based on historical data and data from Twitter sentiment analysis(10 December 2013)

[2]Stock Market Prediction Based on Financial News Text Mining and Investor Sentiment Recognition(05 October 2022)

[3]Stock Market Prediction using Sentiment Analysis and Machine Learning Approach(25 February 2022)

[4]A Three-Dimensional Approach for Stock Prediction Using AI/ML Algorithms(11 February 2023)

[5]An Investigation and Development into the Use of AI-based Analytical Methods for Forecasting the Stock Market(20 September 2023)

[6]Towards Harnessing Based Learning Algorithms for Tweets Sentiment Analysis(20-21 December 2020)

[7]Sentiment-Driven Reinforcement Learning Trading Strategies(06-08 July 2023)

[8]Algorithmic Trading Model for Stock Price Forecasting Integrating Forester with Golden Ratio Strategy(1-6 October 2024)

[9]Potential of ChatGPT in Predicting Stock Market Trends Based on Twitter Sentiment Analysis(13 October 2023)

[10]Performance Evaluation of Time Series Forecasting Methods in the Stock Market: A Comparative Study(23-25 March 2022)

[11]https://www.kaggle.com/code/equinxx/stock-prediction-gan-twitter-sentiment-analysis/input?select=stock_tweets.csv

[12]https://www.kaggle.com/code/equinxx/stock-prediction-gan-twitter-sentiment-analysis/input?select=stock_yfinance_data.csv

APPENDIX A - SAMPLE CODE

```
import os
import numpy as np
import csv
import pandas as pd
import matplotlib.pyplot as plt
import nltk
from matplotlib.dates import DateFormatter
import math
import time
from math import sqrt
from datetime import datetime, timedelta
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import tensorflow as tf
from tensorflow.keras.layers import GRU, LSTM, Bidirectional, Dense, Flatten, Conv1D, BatchNormalization, LeakyReLU, Dropout
from tensorflow.keras import Sequential
from tensorflow.keras.utils import plot_model
import graphviz
from keras.utils import plot_model
from pickle import load
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from tqdm import tqdm
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from pickle import dump
import unicodedata

all_tweets = pd.read_csv('stock_tweets.csv')
print(all_tweets.shape)
all_tweets.head()

all_tweets['Stock Name'].unique()
stock_name1 = 'MSFT'
stock_name2 = 'META'
stock_name3 = 'AMZN'

df1 = all_tweets[all_tweets['Stock Name'] == stock_name1]
print(df1.shape)
df1.head()
```

```
sent_df1 = df1.copy()
sent_df1["sentiment_score"] = ''
sent_df1["Negative"] = ''
sent_df1["Neutral"] = ''
sent_df1["Positive"] = ''
sent_df1.head()
%%time
sentiment_analyzer = SentimentIntensityAnalyzer()
for indx, row in sent_df1.T.iteritems():
    try:
        sentence_i = unicodedata.normalize('NFKD', sent_df1.loc[indx, 'Tweet'])
        sentence_sentiment = sentiment_analyzer.polarity_scores(sentence_i)
        sent_df1.at[indx, 'sentiment_score'] = sentence_sentiment['compound']
        sent_df1.at[indx, 'Negative'] = sentence_sentiment['neg']
        sent_df1.at[indx, 'Neutral'] = sentence_sentiment['neu']
        sent_df1.at[indx, 'Positive'] = sentence_sentiment['pos']
    except TypeError:
        print (sent_df1.loc[indexx, 'Tweet'])
        print (indx)
        break
sent_df1['Date'] = pd.to_datetime(sent_df1['Date'])
sent_df1['Date'] = sent_df1['Date'].dt.date
sent_df1 = sent_df1.drop(columns=['Negative', 'Positive', 'Neutral', 'Stock Name', 'Company
Name'])
sent_df1.head()

twitter_df1 = sent_df1.groupby([sent_df1['Date']]).mean()
print(twitter_df1.shape)
twitter_df1.head()
sent_df2["sentiment_score"] = 0.0
sent_df2["Negative"] = 0.0
sent_df2["Neutral"] = 0.0
sent_df2["Positive"] = 0.0

for indx, row in sent_df2.iterrows():
    try:
        sentence_i = unicodedata.normalize('NFKD', row['Tweet'])
        sentence_sentiment = sentiment_analyzer.polarity_scores(sentence_i)
        sent_df2.at[indx, 'sentiment_score'] = sentence_sentiment['compound']
        sent_df2.at[indx, 'Negative'] = sentence_sentiment['neg']
        sent_df2.at[indx, 'Neutral'] = sentence_sentiment['neu']
        sent_df2.at[indx, 'Positive'] = sentence_sentiment['pos']
    except TypeError:
```

```
        print(f"Error processing tweet at index {indx}")

sentiment_counts = {
    "Positive": sent_df2["Positive"].sum(),
    "Neutral": sent_df2["Neutral"].sum(),
    "Negative": sent_df2["Negative"].sum()
}

colors = {"Positive": "green", "Neutral": "gray", "Negative": "red"}

plt.figure(figsize=(7, 7))
plt.pie(sentiment_counts.values(), labels=sentiment_counts.keys(),
        autopct='%1.1f%%', colors=[colors[label] for label in sentiment_counts.keys()],
        startangle=140)

plt.title(f"Sentiment Distribution for {stock_name2}")

plt.show()
all_stocks = pd.read_csv('stock_yfinance_data.csv')
print(all_stocks.shape)
all_stocks.head()
stock_df2 = all_stocks[all_stocks['Stock Name'] == stock_name2]
stock_df2['Date'] = pd.to_datetime(stock_df2['Date'])
stock_df2['Date'] = stock_df2['Date'].dt.date
final_df2 = stock_df2.join(twitter_df2, how="left", on="Date")
final_df2 = final_df2.drop(columns=['Stock Name'])
print(final_df2.shape)

fig, ax = plt.subplots(figsize=(15,8))
ax.plot(final_df2['Date'], final_df2['Close'], color='#008B8B')
ax.set(xlabel="Date", ylabel="USD", title=f"{stock_name2} Stock Price")
ax.xaxis.set_major_formatter(DateFormatter("%Y"))
plt.show()
def get_tech_ind(data):
    data['MA7'] = data.iloc[:,4].rolling(window=7).mean()
    data['MA20'] = data.iloc[:,4].rolling(window=20).mean()

                                data['MACD']        =        data.iloc[:,4].ewm(span=26).mean()        -
data.iloc[:,1].ewm(span=12,adjust=False).mean()

    data['20SD'] = data.iloc[:, 4].rolling(20).std()
    data['upper_band'] = data['MA20'] + (data['20SD'] * 2)
    data['lower_band'] = data['MA20'] - (data['20SD'] * 2)
```

```
    data['EMA'] = data.iloc[:,4].ewm(com=0.5).mean()

    data['logmomentum'] = np.log(data.iloc[:,4] - 1)

    return data

def tech_ind(dataset):
    fig,ax = plt.subplots(figsize=(15, 8), dpi = 200)
    x_ = range(3, dataset.shape[0])
    x_ = list(dataset.index)

            ax.plot(dataset['Date'], dataset['MA7'], label='Moving Average (7 days)', color='g',
linestyle='--')
    ax.plot(dataset['Date'], dataset['Close'], label='Closing Price', color='#6A5ACD')
          ax.plot(dataset['Date'], dataset['MA20'], label='Moving Average (20 days)', color='r',
linestyle='-.')
    ax.xaxis.set_major_formatter(DateFormatter("%Y"))
    plt.title('Technical indicators')
    plt.ylabel('Close (USD)')
    plt.xlabel("Year")
    plt.legend()

    plt.show()

tech_df2 = get_tech_ind(final_df2)
dataset2 = tech_df2.iloc[20:,:].reset_index(drop=True)
dataset2.head()
tech_ind(tech_df2)
dataset2.iloc[:, 1:] = pd.concat([dataset2.iloc[:, 1:].ffill()])

datetime_series2 = pd.to_datetime(dataset2['Date'])
datetime_index2 = pd.DatetimeIndex(datetime_series2.values)
dataset2 = dataset2.set_index(datetime_index2)
dataset2 = dataset2.sort_values(by='Date')
dataset2 = dataset2.drop(columns='Date')

def normalize_data(df, range, target_column):

    '''
    df: dataframe object
    range: type tuple -> (lower_bound, upper_bound)
        lower_bound: int
        upper_bound: int
    target_column: type str -> should reflect closing price of stock
```

```
    '''

    target_df_series = pd.DataFrame(df[target_column])
    data = pd.DataFrame(df.iloc[:, :])

    X_scaler1 = MinMaxScaler(feature_range=range)
    y_scaler1 = MinMaxScaler(feature_range=range)
    X_scaler1.fit(data)
    y_scaler1.fit(target_df_series)

    X_scale_dataset1 = X_scaler1.fit_transform(data)
    y_scale_dataset1 = y_scaler1.fit_transform(target_df_series)

    dump(X_scaler1, open('X_scaler1.pkl', 'wb'))
    dump(y_scaler1, open('y_scaler1.pkl', 'wb'))

    return (X_scale_dataset1,y_scale_dataset1)
def batch_data(x_data,y_data, batch_size, predict_period):
    X_batched, y_batched, yc = list(), list(), list()

    for i in range(0,len(x_data),1):
        x_value = x_data[i: i + batch_size][:, :]
        y_value = y_data[i + batch_size: i + batch_size + predict_period][:, 0]
        yc_value = y_data[i: i + batch_size][:, :]
        if len(x_value) == batch_size and len(y_value) == predict_period:
            X_batched.append(x_value)
            y_batched.append(y_value)
            yc.append(yc_value)

    return np.array(X_batched), np.array(y_batched), np.array(yc)
def split_train_test(data):
    train_size = len(data) - 20
    data_train = data[0:train_size]
    data_test = data[train_size:]
    return data_train, data_test
def predict_index(dataset1, X_train1, batch_size, prediction_period1):

        train_predict_index1 = dataset1.iloc[batch_size: X_train1.shape[0] + batch_size +
prediction_period1, :].index
    test_predict_index1 = dataset1.iloc[X_train1.shape[0] + batch_size:, :].index

    return train_predict_index1, test_predict_index1
X_scale_dataset1,y_scale_dataset1 = normalize_data(dataset1, (-1,1), "Close")
```

```
X_batched1, y_batched1, yc1 = batch_data(X_scale_dataset1, y_scale_dataset1, batch_size =
5, predict_period = 1)
print("X shape:", X_batched1.shape)
print("y shape:", y_batched1.shape)
print("yc shape:", yc1.shape)

X_train1, X_test1, = split_train_test(X_batched1)
y_train1, y_test1, = split_train_test(y_batched1)
yc_train1, yc_test1, = split_train_test(yc1)
index_train1, index_test1, = predict_index(dataset1, X_train1, 5, 1)
input_dim1 = X_train1.shape[1]
feature_size1 = X_train1.shape[2]
output_dim1 = y_train1.shape[1]
def make_generator_model(input_dim1, output_dim1, feature_size1):
    model = tf.keras.Sequential([LSTM(units = 1024, return_sequences = True,
                                 input_shape=(input_dim1, feature_size1),recurrent_dropout = 0.3),
                            LSTM(units = 512, return_sequences = True, recurrent_dropout = 0.3),
                            LSTM(units = 256, return_sequences = True, recurrent_dropout = 0.3),
                            LSTM(units = 128, return_sequences = True, recurrent_dropout = 0.3),
                            LSTM(units = 64, recurrent_dropout = 0.3),
                            Dense(32),
                            Dense(16),
                            Dense(8),
                            Dense(units=output_dim1)])
    return model
def make_discriminator_model(input_dim1):
    cnn_net = tf.keras.Sequential()
            cnn_net.add(Conv1D(8,  input_shape=(input_dim1+1,   1),  kernel_size=3,   strides=2,
padding='same', activation=LeakyReLU(alpha=0.01)))
                        cnn_net.add(Conv1D(16,      kernel_size=3,      strides=2,      padding='same',
activation=LeakyReLU(alpha=0.01)))
                        cnn_net.add(Conv1D(32,      kernel_size=3,      strides=2,      padding='same',
activation=LeakyReLU(alpha=0.01)))
                        cnn_net.add(Conv1D(64,      kernel_size=3,      strides=2,      padding='same',
activation=LeakyReLU(alpha=0.01)))
                       cnn_net.add(Conv1D(128,      kernel_size=1,      strides=2,      padding='same',
activation=LeakyReLU(alpha=0.01)))

    cnn_net.add(LeakyReLU())
    cnn_net.add(Dense(220, use_bias=False))
    cnn_net.add(LeakyReLU())
    cnn_net.add(Dense(220, use_bias=False, activation='relu'))
    cnn_net.add(Dense(1, activation='sigmoid'))
    return cnn_net
```

```
def discriminator_loss(real_output1, fake_output1):
    loss_f1 = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss1 = loss_f1(tf.ones_like(real_output1), real_output1)
    fake_loss1 = loss_f1(tf.zeros_like(fake_output1), fake_output1)
    total_loss1 = real_loss1 + fake_loss1
    return total_loss1

def generator_loss(fake_output1):
    loss_f1 = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    loss1 = loss_f1(tf.ones_like(fake_output1), fake_output1)
    return loss1
@tf.function

def train_step(real_x1, real_y1, yc1, generator1, discriminator1, g_optimizer1, d_optimizer1):
    with tf.GradientTape() as gen_tape1, tf.GradientTape() as disc_tape:
        generated_data1 = generator1(real_x1, training=True)
            generated_data_reshape1 = tf.reshape(generated_data1, [generated_data1.shape[0],
generated_data1.shape[1], 1])
        d_fake_input = tf.concat([tf.cast(generated_data_reshape1, tf.float64), yc1], axis=1)
        real_y_reshape1 = tf.reshape(real_y1, [real_y1.shape[0], real_y1.shape[1], 1])
        d_real_input = tf.concat([real_y_reshape1, yc1], axis=1)

        real_output1 = discriminator1(d_real_input, training=True)
        fake_output1 = discriminator1(d_fake_input, training=True)

        g_loss1 = generator_loss(fake_output1)
        disc_loss1 = discriminator_loss(real_output1, fake_output1)

    gradients_of_generator1 = gen_tape1.gradient(g_loss1, generator1.trainable_variables)
                                gradients_of_discriminator1 = disc_tape.gradient(disc_loss1,
discriminator1.trainable_variables)

    g_optimizer1.apply_gradients(zip(gradients_of_generator1, generator1.trainable_variables))
                                        d_optimizer1.apply_gradients(zip(gradients_of_discriminator1,
discriminator1.trainable_variables))

    return real_y1, generated_data1, {'d_loss': disc_loss1, 'g_loss1': g_loss1}
def train(real_x1, real_y1, yc1, Epochs1, generator1, discriminator1, g_optimizer1,
d_optimizer1, checkpoint = 50):
    train_info = {}
    train_info["discriminator_loss"] = []
    train_info["generator_loss"] = []

    for epoch in tqdm(range(Epochs1)):
```

```
            real_price1, fake_price1, loss1 = train_step(real_x1, real_y1, yc1, generator1,
discriminator1, g_optimizer1, d_optimizer1)
        G_losses1 = []
        D_losses1 = []
        Real_price1 = []
        Predicted_price1 = []
        D_losses1.append(loss1['d_loss'].numpy())
        G_losses1.append(loss1['g_loss1'].numpy())
        Predicted_price1.append(fake_price1.numpy())
        Real_price1.append(real_price1.numpy())

        if (epoch + 1) % checkpoint == 0:
                                                    tf.keras.models.save_model(generator1,
f'./models_gan/{stock_name1}/generator_V_%d.h5' % epoch)
                                                tf.keras.models.save_model(discriminator1,
f'./models_gan/{stock_name1}/discriminator_V_%d.h5' % epoch)
            print('epoch', epoch + 1, 'discriminator_loss', loss1['d_loss'].numpy(), 'generator_loss',
loss1['g_loss1'].numpy())

        train_info["discriminator_loss"].append(D_losses1)
        train_info["generator_loss"].append(G_losses1)

    Predicted_price1 = np.array(Predicted_price1)
                Predicted_price1 = Predicted_price1.reshape(Predicted_price1.shape[1],
Predicted_price1.shape[2])
    Real_price1 = np.array(Real_price1)
    Real_price1 = Real_price1.reshape(Real_price1.shape[1], Real_price1.shape[2])

    plt.subplot(2,1,1)
    plt.plot(train_info["discriminator_loss"], label='Disc_loss', color='#000000')
    plt.xlabel('Epoch')
    plt.ylabel('Discriminator Loss')
    plt.legend()

    plt.subplot(2,1,2)
    plt.plot(train_info["generator_loss"], label='Gen_loss', color='#000000')
    plt.xlabel('Epoch')
    plt.ylabel('Generator Loss')
    plt.legend()

    plt.show()

            return Predicted_price1, Real_price1, np.sqrt(mean_squared_error(Real_price1,
Predicted_price1)) / np.mean(Real_price1)
```

```
def plot_results(Real_price1, Predicted_price1, index_train1):
    X_scaler1 = load(open('/content/X_scaler.pkl', 'rb'))
    y_scaler1 = load(open('/content/y_scaler.pkl', 'rb'))
    train_predict_index1 = index_train1

    rescaled_Real_price1 = y_scaler1.inverse_transform(Real_price1)
    rescaled_Predicted_price1 = y_scaler1.inverse_transform(Predicted_price1)

    predict_result = pd.DataFrame()
    for i in range(rescaled_Predicted_price.shape[0]):
            y_predict = pd.DataFrame(rescaled_Predicted_price[i], columns=["predicted_price"],
index=train_predict_index[i:i+output_dim1])
        predict_result = pd.concat([predict_result, y_predict], axis=1, sort=False)

    real_price1 = pd.DataFrame()
    for i in range(rescaled_Real_price1.shape[0]):
                    y_train = pd.DataFrame(rescaled_Real_price1[i], columns=["real_price1"],
index=train_predict_index[i:i+output_dim1])
        real_price1 = pd.concat([real_price1, y_train1], axis=1, sort=False)

    predict_result['predicted_mean'] = predict_result.mean(axis=1)
    real_price1['real_mean'] = real_price1.mean(axis=1)

    plt.figure(figsize=(16, 8))
    plt.plot(real_price1["real_mean"])
    plt.plot(predict_result["predicted_mean"], color = 'r')
    plt.xlabel("Date")
    plt.ylabel("Stock price")
    plt.legend(("Real price", "Predicted price"), loc="upper left", fontsize=16)
    plt.title("The result of Training", fontsize=20)
    plt.show()

    predicted = predict_result["predicted_mean"]
    real = real_price1["real_mean"]
    For_MSE = pd.concat([predicted, real], axis = 1)
    RMSE = np.sqrt(mean_squared_error(predicted, real))
    print('-- Train RMSE -- ', RMSE)
def eval_op(generator1, real_x1):
    generated_data = generator1(real_x1, training = False)

    return generated_data
def plot_test_data(Real_test_price1, Predicted_test_price1, index_test1):
    X_scaler1 = load(open('X_scaler.pkl', 'rb'))
    y_scaler1 = load(open('y_scaler.pkl', 'rb'))
```

```
    test_predict_index1 = index_test1

    rescaled_Real_price1 = y_scaler1.inverse_transform(Real_test_price1)
    rescaled_Predicted_price1 = y_scaler1.inverse_transform(Predicted_test_price1)

    predict_result = pd.DataFrame()
    for i in range(rescaled_Predicted_price1.shape[0]):
        y_predict1 = pd.DataFrame(rescaled_Predicted_price1[i], columns=["predicted_price1"],
index=test_predict_index1[i:i+output_dim1])
        predict_result = pd.concat([predict_result, y_predict1], axis=1, sort=False)

    real_price1 = pd.DataFrame()
    for i in range(rescaled_Real_price1.shape[0]):
                    y_train1 = pd.DataFrame(rescaled_Real_price1[i], columns=["real_price1"],
index=test_predict_index1[i:i+output_dim1])
        real_price1 = pd.concat([real_price1, y_train1], axis=1, sort=False)

    predict_result['predicted_mean'] = predict_result.mean(axis=1)
    real_price1['real_mean'] = real_price1.mean(axis=1)

    predicted1 = predict_result["predicted_mean"]
    real1 = real_price1["real_mean"]
    For_MSE1 = pd.concat([predicted1, real1], axis = 1)
    RMSE1 = np.sqrt(mean_squared_error(predicted1, real1))
    r21 = r2_score(real1, predicted1)
    print('Test RMSE: ', RMSE1)


    plt.figure(figsize=(16, 8))
    plt.plot(real_price1["real_mean"], color='#00008B')
    plt.plot(predict_result["predicted_mean"], color = '#8B0000', linestyle='--')
    plt.xlabel("Date")
    plt.ylabel("Stock price")
    plt.legend(("Real price", "Predicted price"), loc="upper left", fontsize=16)
    plt.title(f"Prediction on test data for {stock_name1}", fontsize=20)
    plt.show()
learning_rate = 5e-4
epochs = 1500

g_optimizer1 = tf.keras.optimizers.Adam(learning_rate)
d_optimizer1 = tf.keras.optimizers.Adam(learning_rate)

generator1 = make_generator_model(X_train1.shape[1], output_dim1, X_train1.shape[2])
discriminator1 = make_discriminator_model(X_train1.shape[1])
```

```
predicted_price1, real_price1, RMSPE1 = train(X_train1, y_train1, yc_train1, epochs, generator1, discriminator1, g_optimizer1, d_optimizer1)test_generator1 = tf.keras.models.load_model(
    f'./models_gan/{stock_name1}/generator_V_{epochs-1}.h5',
    compile=False
)
test_generator1.compile(optimizer='adam', loss='mse')
predicted_test_data1 = eval_op(test_generator1, X_test1)
plot_test_data(y_test1, predicted_test_data1,index_test1)
y_test1_np = np.array(y_test1) if isinstance(y_test1, tf.Tensor) else y_test1
predicted_test_data1_np = np.array(predicted_test_data1) if isinstance(predicted_test_data1, tf.Tensor) else predicted_test_data1

r2_value1 = r2_score(y_test1_np, predicted_test_data1_np)

print(f'R² Score: {r2_value1:.4f}')

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from nltk.sentiment import SentimentIntensityAnalyzer

def load_and_prepare_data(stock_symbol='META'):
    stock_df = pd.read_csv("stock_yfinance_data.csv")
    twitter_df = pd.read_csv("stock_tweets.csv")

    stock_tweets = twitter_df[twitter_df['Stock Name'] == stock_symbol].copy()

    analyzer = SentimentIntensityAnalyzer()
    stock_tweets['sentiment_score'] = stock_tweets['Tweet'].apply(
        lambda x: analyzer.polarity_scores(str(x))['compound']
    )

    stock_tweets['Date'] = pd.to_datetime(stock_tweets['Date']).dt.date
    sentiment_df = stock_tweets.groupby('Date')['sentiment_score'].mean().reset_index()

    stock_df['Date'] = pd.to_datetime(stock_df['Date']).dt.date
    merged_df = pd.merge(stock_df, sentiment_df, on='Date', how='inner')

    return merged_df

def add_features(df):
```

```
    df = df.copy()
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.sort_values('Date')

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

    for i in [1, 2, 3, 5, 10]:
        df[f'sentiment_lag_{i}'] = df['sentiment_score'].shift(i)
        df[f'close_lag_{i}'] = df['Close'].shift(i)

    for window in [5, 10, 20]:
        df[f'sentiment_ma_{window}'] = df['sentiment_score'].rolling(window).mean()
        df[f'close_ma_{window}'] = df['Close'].rolling(window).mean()

    df['daily_range_pct'] = (df['High'] - df['Low']) / df['Open']
    df['volatility_5'] = df['Close'].pct_change().rolling(5).std()

    df['sentiment_close'] = df['sentiment_score'] * df['Close']

    non_numeric = df.select_dtypes(exclude=[np.number]).columns.tolist()
    non_numeric.remove('Date')
    df = df.drop(columns=non_numeric)

    df.dropna(inplace=True)
    return df

def train_and_evaluate(df):
    split_date = df['Date'].quantile(0.8)
    train = df[df['Date'] <= split_date]
    test = df[df['Date'] > split_date]

    X_train = train.drop(columns=['Date', 'Price_Change_Pct'])
    X_test = test.drop(columns=['Date', 'Price_Change_Pct'])
    y_train = train['Price_Change_Pct']
    y_test = test['Price_Change_Pct']

    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    model = RandomForestRegressor(n_estimators=200, random_state=42)
    model.fit(X_train_scaled, y_train)

    y_pred = model.predict(X_test_scaled)
```

```
    print(f"R2 Score: {r2_score(y_test, y_pred):.4f}")

    importances = pd.DataFrame({
        'Feature': X_train.columns,
        'Importance': model.feature_importances_
    }).sort_values('Importance', ascending=False)

    print("\nTop Features:")
    print(importances.head(10))

    return model

if __name__== "_main_":

    print("Analyzing META data...")
    meta_data = load_and_prepare_data('META')
    meta_features = add_features(meta_data)
    meta_model = train_and_evaluate(meta_features)
def plot_results(results):
    results_df = pd.DataFrame({
        'Date': pd.to_datetime(results['test_dates']),
        'Actual': results['y_test'],
        'Predicted': results['y_pred']
    }).set_index('Date')

    plt.figure(figsize=(12, 6))
    sns.set_style("whitegrid")

    plt.plot(results_df.index, results_df['Actual'],
             label='Actual Price Change',
             color='blue',
             linewidth=2)
    plt.plot(results_df.index, results_df['Predicted'],
             label='Predicted Price Change',
             color='red',
             linestyle='--')

    plt.title('Stock Price Change: Actual vs Predicted', fontsize=14)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price Change Percentage', fontsize=12)
    plt.legend(fontsize=12)
    plt.xticks(rotation=45)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.tight_layout()
```

```
    plt.show()

if __name__== "_main_":
    print("Loading and preparing data...")
    stock_data = load_and_prepare_data('META')
    features_data = add_features(stock_data)

    print("Training model...")
    results = train_and_evaluate(features_data)

    print("Generating visualization...")
    plot_results(results)
def calculate_robust_features(df):
    """Calculate robust technical indicators without TA-Lib"""
    df['SMA_10'] = df['Close'].rolling(window=10).mean()
    df['EMA_20'] = df['Close'].ewm(span=20, adjust=False).mean()

    df['Daily_Return'] = df['Close'].pct_change()
    df['Volatility_5'] = df['Daily_Return'].rolling(window=5).std()
    df['Volatility_20'] = df['Daily_Return'].rolling(window=20).std()

    df['5d_High'] = df['High'].rolling(window=5).max()
    df['5d_Low'] = df['Low'].rolling(window=5).min()
    df['Price_Position'] = (df['Close'] - df['5d_Low']) / (df['5d_High'] - df['5d_Low'])

    df['Volume_MA_5'] = df['Volume'].rolling(window=5).mean()
    df['Volume_Change'] = df['Volume'] / df['Volume_MA_5']

    for i in [1, 2, 5]:
        df[f'Return_Lag_{i}'] = df['Daily_Return'].shift(i)

    iso = IsolationForest(contamination=0.05, random_state=42)
    outliers = iso.fit_predict(df[['Close', 'Volume']])
    df = df[outliers == 1]

    return df

data = pd.read_csv('stock_yfinance_data.csv')
meta_data = data[data['Stock Name'] == 'META'].copy()
meta_data['Date'] = pd.to_datetime(meta_data['Date'])
meta_data.sort_values('Date', inplace=True)

meta_data = calculate_robust_features(meta_data)
meta_data.dropna(inplace=True)
```

```
features = ['SMA_10', 'EMA_20', 'Volatility_5', 'Volatility_20',
          'Price_Position', 'Volume_Change', 'Volume_MA_5'] + \
          [f'Return_Lag_{i}' for i in [1, 2, 5]]

X = meta_data[features].values
y = meta_data['Close'].values
dates = meta_data['Date'].values

tscv = TimeSeriesSplit(n_splits=5)
r2_scores = []
test_dates = []
test_actual = []
test_predicted = []

pipeline = Pipeline([
    ('scaler', RobustScaler()),
    ('feature_selection', RFE(SVR(kernel='linear'), n_features_to_select=5)),
    ('svr', SVR(kernel='rbf'))
])

param_dist = {
    'svr__C': loguniform(1e0, 1e3),
    'svr__gamma': loguniform(1e-4, 1e-1),
    'svr__epsilon': [0.01, 0.05, 0.1, 0.2],
    'feature_selection__estimator__C': [0.1, 1, 10]
}

for train_index, test_index in tscv.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    search = RandomizedSearchCV(
        pipeline,
        param_distributions=param_dist,
        n_iter=50,
        cv=TimeSeriesSplit(n_splits=3),
        scoring='r2',
        n_jobs=-1,
        random_state=42
    )

    search.fit(X_train, y_train)
    best_model = search.best_estimator_
```

```
    y_pred = best_model.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    r2_scores.append(r2)

    test_dates.extend(dates[test_index])
    test_actual.extend(y_test)
    test_predicted.extend(y_pred)

    print(f"Fold {len(r2_scores)}: R² = {r2:.4f}")
    print(f"Best params: {search.best_params_}")



plt.figure(figsize=(14, 7))
plt.plot(dates, y, label='Actual Price', color='blue', alpha=0.5)
plt.plot(test_dates, test_predicted, label='Predicted Price', color='red', linestyle='--', alpha=0.8)
plt.title('META Stock Price Prediction with Optimized SVM')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()

selected_features = np.array(features)[best_model.named_steps['feature_selection'].support_]
print("\nMost Important Features:")
print(selected_features)
```