Quantitative Analysis Tool
for
Stock Market

Report submitted in partial fulfillment of the requirement for the degree of

B.Tech
in
Information Technology

Under the supervision of

Dr Abhishek Swaroop

Dept of IT

Submitted by

Ashwin Gupta, 00720803121
Gaurav Verma, 01820803121
Vaibhav Agarwal, 02120803121
Ashit Kumar, 02520803121

Department of Information Technology
Bhagwan Parshuram Institute of Technology
PSP-4, Sec-17, Rohini, Delhi-89

Declaration

This is to certify that the report titled " ***Quantitative Analysis Tool for Stock Market*** ", is submitted by us in partial fulfillment of the requirements for the award of the degree of *B.Tech. in Information Technology* to BPIT, Rohini, Delhi, affiliated to GGSIP University, Delhi. The report is a result of our original work, and due acknowledgment has been made for any work done by others.

The work presented in this report has not been submitted elsewhere for the award of any degree or diploma. We hereby declare that the information provided is true to the best of our knowledge and belief.

Date: _____

Ashwin Gupta, 00720803121

Gaurav Verma, 01820803121

Vaibhav Agarwal, 02120803121

Ashit Kumar, 02520803121

Certificate by Supervisor

This is to certify that the report titled " ***Quantitative Analysis Tool for Stock Market*** " is submitted by **Gaurav Verma (01820803121), Ashwin Gupta (00720803121), Vaibhav Agarwal (02120803121), Ashit Kumar (02520803121)** in partial fulfillment of the requirement for the award of the degree of *B.Tech in Information Technology* to BPIT Rohini, affiliated to GGSIP University, Delhi.

This report is a record of the candidate's own work carried out under my supervision. The matter embodied in this report is original and has not been submitted for the award of any other degree.

Date: **Supervisor: Dr Abhishek Swaroop**

Signature: _____

Certificate by HOD

This is to certify that the report titled " ***Quantitative Analysis Tool for Stock Market*** " is submitted by **Gaurav Verma (01820803121), Ashwin Gupta (00720803121), Vaibhav Agarwal (02120803121), Ashit Kumar (02520803121)** under the guidance of **Dr Abhishek Swaroop** in partial fulfillment of the requirement for the award of the degree of *B.Tech in Information Technology* to BPIT Rohini, affiliated to GGSIP University, Delhi.

The matter embodied in this report is original and has been duly approved for submission.

Date: _____ **Prof. Abhishek Swaroop**

Signature: _____

Acknowledgement

We would like to express our sincere gratitude to all those who supported and guided us throughout the preparation of this report.

Firstly, we would like to thank our supervisor, **Prof. Abhishek Swaroop**, for his invaluable guidance, constructive criticism, and continuous encouragement throughout this project. His vast knowledge and expertise have greatly contributed to the successful completion of this work.

We would also like to thank all the faculty members of the **Department of Information Technology**, BPIT Rohini, for their support and the resources they provided.

Our sincere thanks to our classmates and friends who offered us moral support and valuable insights during the course of this project.

Lastly, we would like to express our heartfelt gratitude to our families for their patience, understanding, and unconditional support throughout the process of our project. Without their love and encouragement, this work would not have been possible.

Date: _____ **Prof. Abhishek Swaroop**

Signature: _____

Contents

1 Introduction

Our comprehensive quantitative analysis tool for the stock market integrates advanced data science methodologies and financial analysis techniques, presenting traders and analysts with a unified platform for data-driven decision-making. Built to harness the predictive power of data, this tool combines sentiment analysis, quantitative trading strategies, and machine learning algorithms to provide a multi-dimensional approach to market analysis. Each component has been meticulously designed to complement the others, offering users deeper insights into stock market trends and helping them make more informed trading decisions. The tool comprises three core modules:

- **Sentiment Analysis on Stock News**: Sentiment analysis is a powerful tool for assessing the mood of the market based on media and public sentiment. This module leverages natural language processing (NLP) to perform real-time analysis on news articles and other text data related to stocks. By parsing large volumes of financial news and categorizing sentiment (positive, negative, or neutral), the module provides an invaluable metric of market sentiment toward specific stocks or the broader market.

  This information is particularly useful for identifying sentiment-driven price movements, as news often acts as a catalyst for short-term volatility. For instance, positive news about a company can drive up its stock price, while negative sentiment can lead to a sell-off. By aggregating and quantifying sentiment across a wide array of news sources, this module enables users to gauge the market's psychological disposition, helping them make anticipatory trades that align with emerging sentiment trends.

- **Quantitative Trading Strategies**: At the heart of this component is the **Option Chain Strategy**, which focuses on analyzing options data to identify potential trading opportunities. Options chains provide insights into market expectations for a stock's future price movement, as options trading often reflects informed opinions from institutional investors and market makers. This strategy involves the application of quantitative models to determine optimal entry and exit points based on patterns in options pricing, implied volatility, and open interest.

  The option chain analysis is complemented by various quantitative techniques, such as statistical arbitrage and risk-reward optimization, that allow traders to capitalize on short-term market inefficiencies systematically. By backtesting these strategies on historical data, the module provides performance metrics that traders can use to refine and optimize their trading approach. This quantitative strategy framework offers a disciplined, rule-based approach to trading that reduces emotional biases and improves consistency in results.

- **Stock Price Prediction Using Machine Learning**: Predicting stock prices is one of the most challenging tasks in financial analysis due to the complex, non-linear nature of financial markets. This module harnesses the predictive capabilities of machine learning, specifically the **Long Short-Term Memory (LSTM) model**—a type of recurrent neural network (RNN) that excels at time series forecasting. LSTM networks are particularly suited to financial data as they can capture long-term dependencies in sequential data, enabling them to recognize patterns and trends over time.

  Using historical stock data, the LSTM model is trained to identify and learn from past price patterns, volume trends, and other indicators. This machine learning-driven approach goes beyond conventional technical analysis by dynamically adapting to new

data, which enhances its predictive accuracy. Developed using Keras, this model provides users with forward-looking insights, assisting them in making more confident trading decisions based on projected price movements. The model's architecture enables it to adjust predictions in real time as it ingests new data, making it a valuable tool for traders focused on timing their market entries and exits precisely.

Together, these three components create a robust, multi-faceted quantitative analysis tool that offers users a holistic view of the stock market landscape. By combining sentiment-driven insights, rigorous quantitative strategies, and predictive machine learning models, this platform empowers traders and investors to navigate financial markets with greater clarity, confidence, and precision. This all-in-one solution is tailored to meet the needs of modern investors seeking to leverage technology to achieve a competitive edge in an increasingly data-centric trading environment.

2 Motivation

2.1 The Dynamic Nature of Financial Markets

The global financial markets are notoriously volatile, characterized by sudden price swings, unpredictable trends, and the continuous interaction of various forces such as macroeconomic events, geopolitical developments, and investor sentiment. These factors make it difficult to predict stock prices accurately. For example, a political crisis or unexpected economic data release can cause rapid changes in stock prices, often in directions that contradict prevailing market trends.

Historically, financial markets were relatively less volatile, and traders could rely on traditional methods like fundamental analysis and technical analysis. However, over the last few decades, the financial landscape has become more complex. Increased globalization, the proliferation of high-frequency trading (HFT) algorithms, and the rise of institutional investors have all contributed to a market that is faster, more interconnected, and harder to predict.

The increasing complexity of the market environment makes it difficult for traders to rely solely on historical data or simplistic models to forecast future price movements. To successfully navigate this landscape, traders need access to powerful tools and strategies that can process vast amounts of real-time data, adjust to rapidly changing conditions, and identify profitable opportunities.

2.2 Traditional Stock Market Analysis: Limitations and Challenges

Traditional stock market analysis typically includes two main approaches: *technical analysis* and *fundamental analysis*. Both have been widely used for decades, but they come with limitations in the face of modern, high-speed markets.

2.2.1 Technical Analysis

Technical analysis involves analyzing past market data, primarily price and volume, to predict future price movements. Traders using technical analysis often rely on chart patterns, trends, and various technical indicators, such as moving averages and oscillators. While technical analysis can be useful in identifying short-term market trends, it assumes that historical price patterns are reliable indicators of future performance. However, this assumption is not always valid in rapidly changing markets, where new information or external events can disrupt historical patterns. Moreover, technical analysis does not account for underlying fundamental factors that may be driving price changes.

2.2.2 Fundamental Analysis

On the other hand, fundamental analysis focuses on evaluating a company's intrinsic value by examining factors like financial statements, management quality, competitive position, and overall industry conditions. While this approach is valuable for long-term investment decisions, it is often too slow to react to short-term market movements. In addition, fundamental analysis requires significant time and expertise to properly evaluate all relevant factors, and may overlook market sentiment and other factors that drive short-term volatility.

2.2.3 Challenges in Current Methods

Both of these methods have their drawbacks in the modern trading environment, especially in high-frequency and algorithmic trading. Traders today need to process real-time information and adapt quickly, which can be difficult when relying on traditional approaches. Furthermore, these methods do not fully utilize the wealth of data now available in the market, such as social media sentiment, macroeconomic indicators, and alternative data sources. There is an increasing need for more sophisticated approaches that integrate multiple data streams, can react to new information in real-time, and leverage the power of machine learning to discover hidden patterns in the market.

2.3 Rise of Technology in Financial Markets

The rise of technology has significantly transformed financial markets. The widespread use of algorithmic trading, artificial intelligence (AI), and machine learning has enabled traders to develop more sophisticated strategies and improve the accuracy of predictions. These technologies can process vast amounts of data in a fraction of the time it would take a human, allowing traders to identify opportunities and risks much more quickly.

Machine learning and AI have become indispensable tools in the world of finance. Predictive models based on these technologies can analyze historical market data, identify patterns, and forecast future stock prices with remarkable accuracy. In addition to price prediction, machine learning is increasingly used for risk management, portfolio optimization, and fraud detection. Machine learning algorithms, such as Long Short-Term Memory (LSTM) networks and decision trees, are being integrated into trading platforms to improve performance and decision-making capabilities.

2.3.1 Big Data and High-Performance Computing

In addition to machine learning, the use of big data and high-performance computing (HPC) has opened up new possibilities in financial markets. Traders now have access to an unprecedented amount of market data, including not only historical prices and volumes but also real-time information such as news sentiment, social media posts, and economic indicators. The ability to analyze these large datasets quickly and efficiently is crucial for successful trading, as even small delays in processing data can result in missed opportunities.

2.3.2 The Role of Real-Time Data and Automation

Moreover, the importance of real-time data and automation cannot be overstated. High-frequency trading (HFT) firms, for instance, use ultra-low-latency connections to execute trades in fractions of a second. For these firms, delays of even milliseconds can lead to significant losses. The ability to react quickly to changing conditions, process vast amounts of data, and adjust strategies in real time is essential in today's financial markets.

As a result, traders are increasingly adopting quantitative strategies, predictive models, and sentiment analysis tools to stay ahead of the competition. These tools allow for more accurate predictions, timely decision-making, and a better understanding of market dynamics.

2.4 Introduction to Advanced Techniques in Trading: Quantitative Analysis, Predictive Modeling, and Sentiment Analysis

With the challenges of traditional methods in mind, the financial industry is increasingly turning to advanced quantitative techniques. These include *quantitative analysis*, *predictive modeling*, and *sentiment analysis*.

2.4.1 Quantitative Analysis

Quantitative analysis involves using mathematical models and statistical techniques to analyze financial data and derive trading strategies. By applying these models, traders can identify patterns and trends that would be difficult to detect through traditional methods. This approach is especially useful for algorithmic trading, where models are applied to automate decisions based on pre-defined rules. Quantitative trading strategies can be more systematic, reducing human bias and emotion, which often play a significant role in traditional decision-making.

2.4.2 Predictive Modeling

Predictive modeling, particularly using machine learning algorithms such as LSTMs (Long Short-Term Memory) networks, allows for forecasting stock prices based on historical data. Unlike traditional methods, which may rely on static assumptions, machine learning models can adapt over time as new data becomes available. These models can capture complex relationships between variables and account for nonlinearities in the data. The ability to forecast future stock prices with high accuracy is one of the most important advancements in modern trading.

2.4.3 Sentiment Analysis

Sentiment analysis has become another powerful tool for traders. By analyzing news articles, social media posts, and other text data, sentiment analysis can provide insights into the market's mood and investor sentiment. For instance, positive sentiment surrounding a company or sector may indicate an upward trend in stock prices, while negative sentiment could signal a potential downturn. Sentiment analysis can help traders refine their strategies by providing additional data points and insights that are not available through traditional financial analysis.

2.5 The Need for a Comprehensive Trading Tool

While individual tools for stock prediction, sentiment analysis, and quantitative strategies exist, there is currently no unified platform that combines all of these features in an easy-to-use interface. This presents a significant opportunity to create a comprehensive tool that integrates multiple advanced techniques for stock market analysis.

2.5.1 Streamlining the Decision-Making Process

A unified platform would significantly streamline the decision-making process for traders, particularly those who are new to the market or lack the experience to fully leverage multiple disparate tools. By bringing together predictive models, sentiment analysis, and quantitative strategies, such a platform would offer a holistic approach to stock market analysis, enabling traders to make informed decisions faster and with greater confidence.

2.5.2 Empowering Traders with Real-Time Data

Moreover, by integrating real-time data and offering insights based on multiple perspectives, the platform would help traders better understand market trends and make more timely, accurate predictions. Whether they are assessing the impact of a major news event, predicting the direction of a stock's price, or identifying profitable trading opportunities, traders would benefit from having all of these capabilities in one place.

2.5.3 Improving Performance and Reducing Cognitive Load

Finally, by combining machine learning, sentiment analysis, and quantitative strategies in a single platform, the tool would reduce cognitive load, making it easier for traders to navigate complex markets. With sophisticated data analytics, users could optimize their strategies, analyze risk, and predict market behavior with a higher level of accuracy and efficiency.

3 Problem Statement

The stock market is a dynamic and complex ecosystem where traders and investors are continuously looking for opportunities to maximize returns while managing risks. As global financial markets become more interconnected, volatile, and influenced by real-time data, the ability to predict stock price movements with accuracy has become increasingly challenging. Furthermore, identifying profitable trading opportunities and mitigating risk in such a fast-paced and constantly evolving environment has become a major concern for traders at all levels.

In the modern world of finance, technology has radically transformed trading practices. In the past, traders and investors had relatively few tools at their disposal, typically relying on fundamental analysis and technical charting techniques to make decisions. However, with the advent of high-frequency trading, the proliferation of alternative data sources, and the rise of machine learning, the stock market today is vastly different from its past form. It presents unique challenges that traditional approaches to stock price prediction and risk management are ill-equipped to address.

This section explores the problem in detail, discusses the gaps that exist in current trading systems, and explains the motivation behind creating a comprehensive and data-driven platform.

3.1 Challenges in Stock Market Forecasting

The first major challenge that traders face in the stock market is forecasting price movements. Stock prices are influenced by a multitude of factors, including company performance, broader economic indicators, geopolitical events, market sentiment, and even social media trends. While traditional methods such as technical analysis and fundamental analysis have long been used to make predictions, these methods have limitations when applied to dynamic market conditions.

3.1.1 Volatility and Uncertainty

The stock market is inherently volatile, with prices frequently experiencing sharp swings. Traditional prediction models struggle to account for sudden market shifts driven by external factors such as unexpected news events or changes in investor sentiment. Even sophisticated statistical models may fall short in predicting extreme market movements, such as during financial crises, economic collapses, or political upheavals. Predictive models based on historical data often assume that the market behaves in a certain manner based on past trends, which may not always hold in times of extreme volatility.

3.1.2 Complexity of Factors Affecting Stock Prices

Another issue is the complexity of the factors that influence stock prices. Many of these factors are interrelated and dynamic, meaning that changes in one area can have cascading effects across the market. For example, a sudden change in interest rates can impact multiple sectors of the economy, such as real estate, financial institutions, and consumer spending, which in turn influences stock prices. Forecasting these interactions requires more than just historical price data. It demands a comprehensive understanding of economic, political, and social factors, as well as the ability to analyze vast amounts of real-time data in an efficient and actionable way.

3.1.3 Need for Advanced Prediction Models

Traditional models based on historical data, such as moving averages or autoregressive integrated moving average (ARIMA) models, often struggle to capture the full complexity of stock price movements. These models tend to overlook non-linear patterns and fail to account for the influence of unstructured data sources, such as news headlines, market sentiment, or tweets. Moreover, the increasing availability of big data presents both opportunities and challenges—while the data is abundant, the ability to process and analyze it effectively remains a significant hurdle.

3.2 Fragmentation of Tools and Platforms

Another significant challenge that traders face is the fragmentation of tools available for stock market analysis. Currently, traders have access to a variety of platforms and tools, each of which specializes in one aspect of trading—some focus on predicting stock prices, others on technical charting, and still others on sentiment analysis. However, these tools do not communicate with each other effectively, forcing traders to use multiple platforms simultaneously and manually integrate insights from various sources.

For example, to predict stock prices, a trader may need to use machine learning models or deep learning techniques such as Long Short-Term Memory (LSTM) networks. To analyze market sentiment, they might rely on news APIs or social media sentiment analysis tools. Similarly, to track options trading opportunities, a trader might use specific software designed to evaluate option chain data. The problem with this fragmented approach is that it increases the cognitive load on traders, making it harder for them to process large amounts of data and make informed decisions in real-time.

Moreover, the manual process of integrating insights from various sources is time-consuming and error-prone. Traders are forced to switch between platforms, copy data from one tool to another, and adjust their strategies accordingly. This not only wastes time but also introduces potential for inconsistency and mistakes. Additionally, many tools are not customizable, making it difficult for traders to tailor them to their specific needs. This fragmentation creates significant inefficiencies and missed opportunities, as traders are unable to access all the necessary information in one place.

3.3 Traditional Methods of Stock Price Prediction and Option Chain Analysis

Traditional methods of stock price prediction and option chain analysis have been widely used in the financial industry for decades, but they are increasingly being found to be inadequate in the face of rapidly changing market conditions. These methods rely on historical data, assumptions about market behavior, and basic statistical techniques. While these methods were useful in the past, they do not account for the complexity and volatility of modern financial markets.

3.3.1 Stock Price Prediction with Statistical Models

In traditional stock price prediction, analysts rely on statistical models such as ARIMA or simple moving averages. These models work well for predicting trends based on past price movements but fail to capture more complex relationships in the data. Additionally, these methods are often based on assumptions that may not hold true in volatile or dynamic markets.

For example, in times of economic turmoil, stock prices may behave unpredictably, making it difficult for simple statistical models to generate accurate predictions.

3.3.2 Option Chain Analysis for Trading Strategies

Similarly, option chain analysis has long been used by traders to evaluate the potential profitability of different option positions. By analyzing various strikes and expiration dates, traders can assess the best strategies for trading options. However, these traditional methods often fail to account for external factors that can influence option pricing, such as changes in volatility or news events that might significantly impact the underlying asset. Additionally, traditional methods of option analysis are often static and do not adapt quickly to changing market conditions.

3.4 The Rise of Machine Learning and Sentiment Analysis

The limitations of traditional stock market analysis methods have led to the rise of machine learning (ML), predictive modeling, and sentiment analysis as new approaches for forecasting stock prices and trading opportunities.

3.4.1 Machine Learning for Stock Price Prediction

Machine learning models, especially deep learning techniques such as LSTM networks, have demonstrated considerable success in forecasting stock prices by analyzing vast amounts of historical data. These models have the ability to capture non-linear relationships between features and make predictions based on complex patterns that traditional models may miss. LSTM networks, in particular, are well-suited for time series forecasting, as they can model dependencies in sequential data—such as daily stock prices—over time. Unlike traditional methods, ML-based models can adjust their predictions in real-time as new data becomes available, enabling them to react quickly to market changes.

3.4.2 Sentiment Analysis for Predicting Market Behavior

Another emerging field is sentiment analysis, which involves analyzing text data, such as news articles, social media posts, and financial reports, to assess market sentiment. By examining how the public is reacting to news events or corporate announcements, sentiment analysis tools can provide valuable insights into the future direction of a stock or sector. Positive sentiment around a company might indicate an upward trend in its stock price, while negative sentiment could suggest a decline. Sentiment analysis helps traders understand the emotional and psychological factors influencing market behavior, which can be particularly useful in times of uncertainty or volatility.

3.4.3 The Importance of Real-Time Data Integration

One of the key advantages of sentiment analysis and machine learning models is their ability to process real-time data. While traditional stock price prediction methods rely on historical data, sentiment analysis can immediately incorporate the latest news and social media trends to predict market movements. This real-time integration is crucial for traders who need to make decisions quickly, especially in fast-moving markets.

3.5 The Need for a Unified Platform

Given the challenges and limitations of current tools and strategies, there is a clear need for a unified platform that integrates quantitative trading strategies, stock price prediction, and sentiment analysis into a single tool. Such a platform would allow traders to access all the data they need in one place, make informed decisions based on multiple data sources, and automate many aspects of their trading strategies.

A unified platform would also reduce the cognitive load on traders by presenting all relevant information in an intuitive and interactive interface. By combining advanced machine learning models, real-time news sentiment analysis, and quantitative trading strategies, this platform would provide traders with a powerful toolkit for navigating the complexities of the stock market.

3.5.1 Key Features of the Platform

The proposed platform will incorporate several key features:

- **Option Chain Trading Strategies:** Implementing various option chain strategies, such as straddles, strangles, and spreads, that can help traders make more informed decisions based on the current market conditions.
- **Stock Price Prediction Using LSTM Models:** Using machine learning models, specifically LSTM networks, to predict stock prices by analyzing historical price data and identifying trends.
- **Sentiment Analysis Using News APIs:** Leveraging real-time news data and sentiment analysis to understand how market sentiment is affecting specific stocks or sectors.

4 Objectives

The objective of this project is to develop a stock market analysis tool that integrates quantitative analysis, predictive modeling, and sentiment analysis to assist traders in making informed decisions. The specific objectives of the project are as follows:

4.1 1. Implement Quantitative Trading Strategies Using Option Chains

4.1.1 Goal:

To develop a set of option chain trading strategies that help traders identify profitable trades based on option data.

4.1.2 Methodology and Approach:

Option chain trading strategies exploit pricing inefficiencies and predict market movements by analyzing option data such as strike prices, premiums, and expiration dates. This section will describe the implementation of various strategies that traders can use based on market conditions.

4.1.3 Key Strategies:

- **Straddle and Strangle:** These strategies are employed when significant market movements are expected. Both involve buying options at different strike prices to capitalize on large movements in either direction.
- **Covered Calls:** A conservative strategy involving holding a stock and selling a call option on it. This strategy generates income from the premium and is effective in stable markets.
- **Iron Condor:** Designed for low volatility, this strategy involves selling both a call and put option while buying further out-of-the-money options to limit risk.

4.1.4 User Interface Design:

The platform will feature a user interface allowing traders to:

- Input stock symbols and time frames.
- View real-time option chains and receive strategy suggestions.
- Customize strategies based on personal risk tolerance and market outlook.

4.1.5 Technical Details:

- Integration with APIs (e.g., Alpha Vantage) for real-time options data.
- Option pricing models (e.g., Black-Scholes) for determining option values.

4.1.6 Challenges:

- Collecting and processing large volumes of real-time data efficiently.
- Risk management and adapting strategies to changing market conditions.

4.2 2. Stock Price Prediction Using LSTM Models

4.2.1 Goal:

To develop a predictive model using Long Short-Term Memory (LSTM) networks to forecast stock prices.

4.2.2 Methodology and Approach:

LSTM networks, a type of recurrent neural network (RNN), are highly suited for time-series forecasting. This section focuses on building a model that predicts future stock prices based on historical market data and technical indicators.

4.2.3 Key Steps:

- **Data Collection:** Gather historical stock prices and technical indicators using APIs (e.g., Yahoo Finance, Alpha Vantage).
- **Preprocessing:** Clean and scale data for LSTM model training.
- **Model Architecture:** Design the LSTM model, optimize hyperparameters, and train it on historical data.
- **Prediction and Visualization:** Display predicted stock prices with confidence intervals alongside historical data.

4.2.4 User Interface:

- Input stock symbols and time periods for prediction.
- Graphical visualization of both historical and predicted price movements.
- Accuracy metrics such as Mean Squared Error (MSE) to evaluate predictions.

4.2.5 Technical Details:

- Use of TensorFlow or Keras for building LSTM models.
- Data processing with Python libraries such as `pandas` and `numpy`.
- Performance evaluation using MSE and RMSE metrics.

4.2.6 Challenges:

- Volatility in stock prices and the difficulty of accurate long-term predictions.
- Avoiding overfitting and ensuring the model generalizes well.

4.3 3. Real-Time Sentiment Analysis of News Articles

4.3.1 Goal:

To build a sentiment analysis module that processes news articles in real-time to gauge market sentiment and its impact on stock prices.

4.3.2 Methodology and Approach:

Sentiment analysis using Natural Language Processing (NLP) techniques will classify news articles and social media posts as positive, negative, or neutral. The sentiment analysis model will help traders understand public sentiment and anticipate stock price movements.

4.3.3 Key Steps:

- **Data Collection:** Use APIs like NewsAPI to collect real-time news articles related to specific stocks.
- **Sentiment Analysis:** Leverage pre-trained models (e.g., BERT) to analyze sentiment and predict market reactions.
- **Real-Time Alerts:** Notify users when a significant news event occurs that could impact the market.

4.3.4 User Interface:

- Display real-time news with sentiment labels.
- Dashboard showing the sentiment for a particular stock or sector.
- Notifications for high-impact news.

4.3.5 Technical Details:

- Use Python libraries like `spaCy` and `transformers` for sentiment analysis.
- Real-time data collection through APIs and WebSockets.

4.3.6 Challenges:

- Handling diverse and sometimes contradictory sentiments across multiple platforms.
- Scaling the system to handle a large number of real-time news articles.

5 System Analysis and Design

5.1 Software Requirement Specifications

5.1.1 Hardware Requirements

The system will have the following hardware requirements to ensure smooth performance and efficient processing power. These hardware specifications are essential for handling large

datasets, real-time stock data processing, machine learning tasks, and responsive web application performance.

- **Processor:** Intel i5 or equivalent (minimum), recommended Intel i7 or above for optimal performance. A high-frequency CPU is crucial for fast data analysis and running machine learning models.
- **RAM:** Minimum 8GB, recommended 16GB for handling large datasets and multiple applications running simultaneously. Adequate RAM is necessary for in-memory data processing and smooth operation of the web interface.
- **Storage:** 500GB HDD (minimum) or 250GB SSD for faster data access and retrieval. SSD is preferred for quicker read/write speeds, which are beneficial when dealing with large stock data files and logs.
- **Network:** High-speed internet connection (minimum 100 Mbps) for real-time data fetching from APIs and for trading execution. A reliable internet connection is essential for seamless communication with external services like stock price APIs and news sources.
- **Graphics:** Integrated graphics will suffice; however, a dedicated GPU is recommended if using machine learning for training models. The GPU will significantly speed up model training for predictions, especially for deep learning tasks.
- **Display:** Minimum 15-inch screen for ease of use; higher resolution is recommended for better visual representation. A larger display enables better handling of complex user interfaces and visualizations of stock data.

5.1.2 Software Requirements

The software requirements for developing and running the system are listed below. These tools and technologies provide a robust environment for building, testing, and deploying the stock market analysis tool.

- **Operating System:** Windows 10 or higher, macOS, or Linux. The system should be platform-agnostic to allow for flexible deployment across different environments.
- **Python:** Version 3.7 or higher for machine learning models, data processing, and web frameworks. Python is preferred due to its simplicity, vast library ecosystem, and support for machine learning frameworks.
- **Web Framework:** Django 3.0+ for backend development, React or Vue.js for frontend development. Django's robustness and ease of integration with databases make it ideal for managing the backend, while React/Vue.js provides a dynamic and interactive frontend experience.
- **Database:** PostgreSQL or MySQL for handling structured data storage. PostgreSQL is preferred due to its support for advanced data types and performance with complex queries.
- **Machine Learning Framework:** TensorFlow or PyTorch for building and deploying LSTM models. These frameworks are specifically suited for time-series forecasting and regression models for predicting stock prices.

- **API Integration:** Alpha Vantage, NewsAPI for fetching real-time stock and news data. APIs provide access to updated market information, which is crucial for real-time decision-making in stock trading.
- **Visualization Libraries:** Matplotlib, Plotly, or Seaborn for data visualization. These libraries allow the system to generate insightful charts, graphs, and dashboards that present data in an easy-to-understand format.
- **Version Control:** Git for source code management, with repositories hosted on GitHub or GitLab. Version control is essential for managing code changes, collaboration, and deployment.
- **Browser:** Google Chrome or Mozilla Firefox for accessing the web interface. Modern browsers ensure compatibility with JavaScript libraries and web technologies.

5.2 System Architecture

The system architecture follows a modular, scalable approach, ensuring that the system is flexible, easy to maintain, and extendable. It is designed to handle large amounts of stock data and provide fast, real-time analysis through a user-friendly interface.

5.2.1 Architectural Components

The architecture consists of the following main components:

- **Frontend:** The user interface built using React or Vue.js interacts with the backend and presents stock data, predictions, and visualizations to the user. It provides an intuitive interface for inputting stock symbols, viewing predictions, and analyzing real-time stock data.
- **Backend:** The backend, developed using Django, handles the system's core logic, including data processing, integration with APIs (such as Alpha Vantage and NewsAPI), and interacting with the machine learning models. The backend is responsible for processing requests from the frontend and sending responses.
- **Database:** A relational database (PostgreSQL or MySQL) is used to store historical stock data, user information, and logs of user actions. The database allows efficient querying and storage of large datasets.
- **Machine Learning Models:** The LSTM model, built using TensorFlow or PyTorch, is responsible for making stock price predictions based on historical data. The model is trained with past stock data and then deployed for making predictions on real-time data.
- **External APIs:** APIs like Alpha Vantage and NewsAPI provide real-time stock market data and news articles. These APIs allow the system to fetch live stock data, financial reports, and market sentiment information that are used for analysis and predictions.

5.2.2 System Flow

The system's flow is designed to provide real-time stock data, process it using machine learning models, and return predictions and insights to the user. The key steps in the system flow are:

1. **User Input:** The user enters a stock symbol to retrieve information for a specific stock.
2. **API Integration:** The system fetches real-time data for the entered stock using the Alpha Vantage API and NewsAPI.
3. **Data Processing:** The historical stock data is preprocessed and fed into the machine learning model for predictions.
4. **Model Prediction:** The trained LSTM model processes the data and generates predictions for the stock's future prices.
5. **Visualization:** The system visualizes the stock data, predictions, and sentiment analysis results on interactive charts.
6. **User Output:** The user receives the predictions, trends, and news for the stock symbol entered.

5.3 Use Case Diagram for qunatitative trading strategy

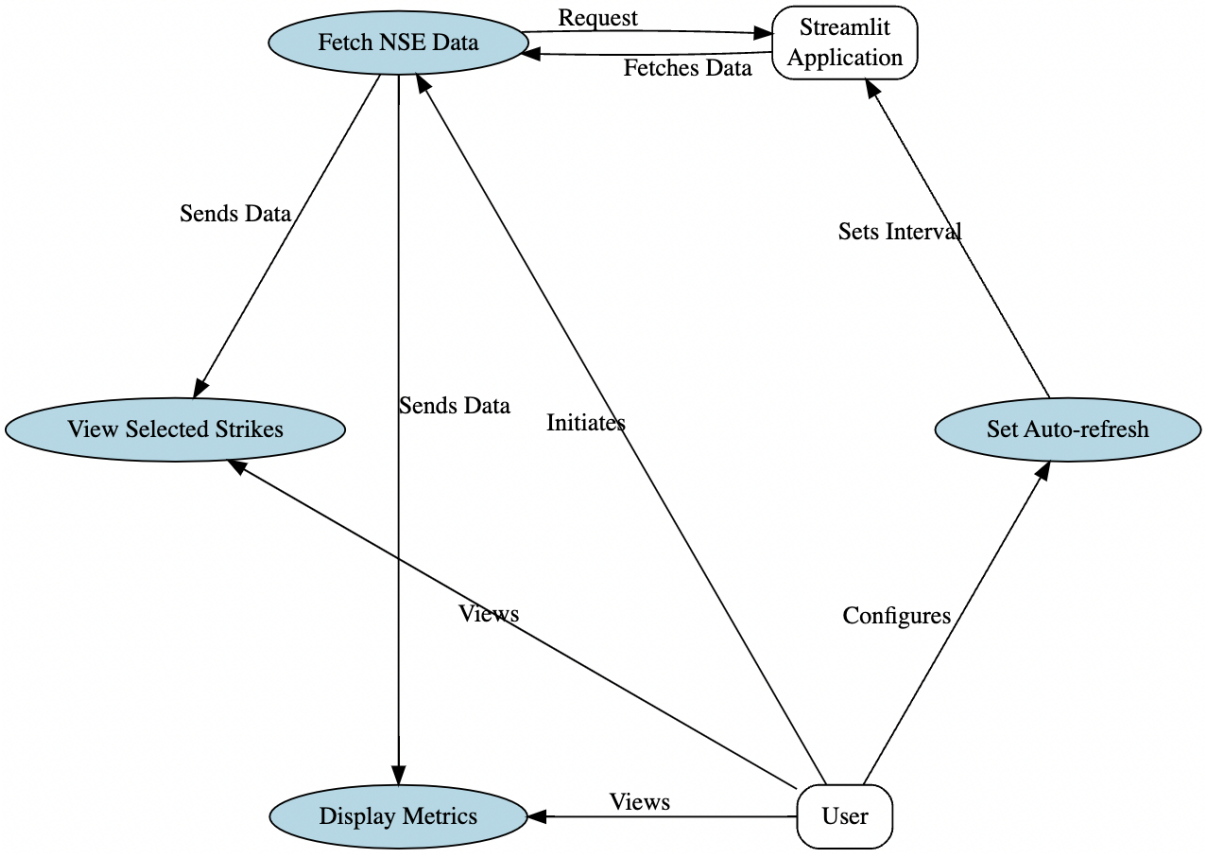

Figure 1: Use Case Diagram Option Chain Strategy

5.3.1 Actors

- **User**: Interacts with the Streamlit application to initiate actions and view data.
- **System (Streamlit Application)**: Handles requests from the user, fetches data from NSE, computes metrics, and displays information.

5.3.2 Use Cases

- **Fetch NSE Data**: User initiates fetching option chain data from NSE.
- **Display Metrics**: System computes and displays key metrics such as NIFTY Index Value, ATM Strike Price, and Put-Call Ratio.
- **View Selected Strikes**: User views detailed information on selected and nearby strike prices.
- **Set Auto-refresh**: User configures the auto-refresh interval for real-time updates.

5.3.3 Interactions

- **User** → **Fetch NSE Data**: User initiates fetching data.
- **Fetch NSE Data** → **System**: Fetches data from NSE API.
- **System** → **Display Metrics / View Selected Strikes**: Sends computed data to display.
- **User** → **Display Metrics / View Selected Strikes**: Views the computed metrics and strike information.
- **User** → **Set Auto-refresh**: Configures the auto-refresh interval.
- **Set Auto-refresh** → **System**: Sets the interval for auto-refresh.

5.4 Use Case Diagram for Stock Pridiction

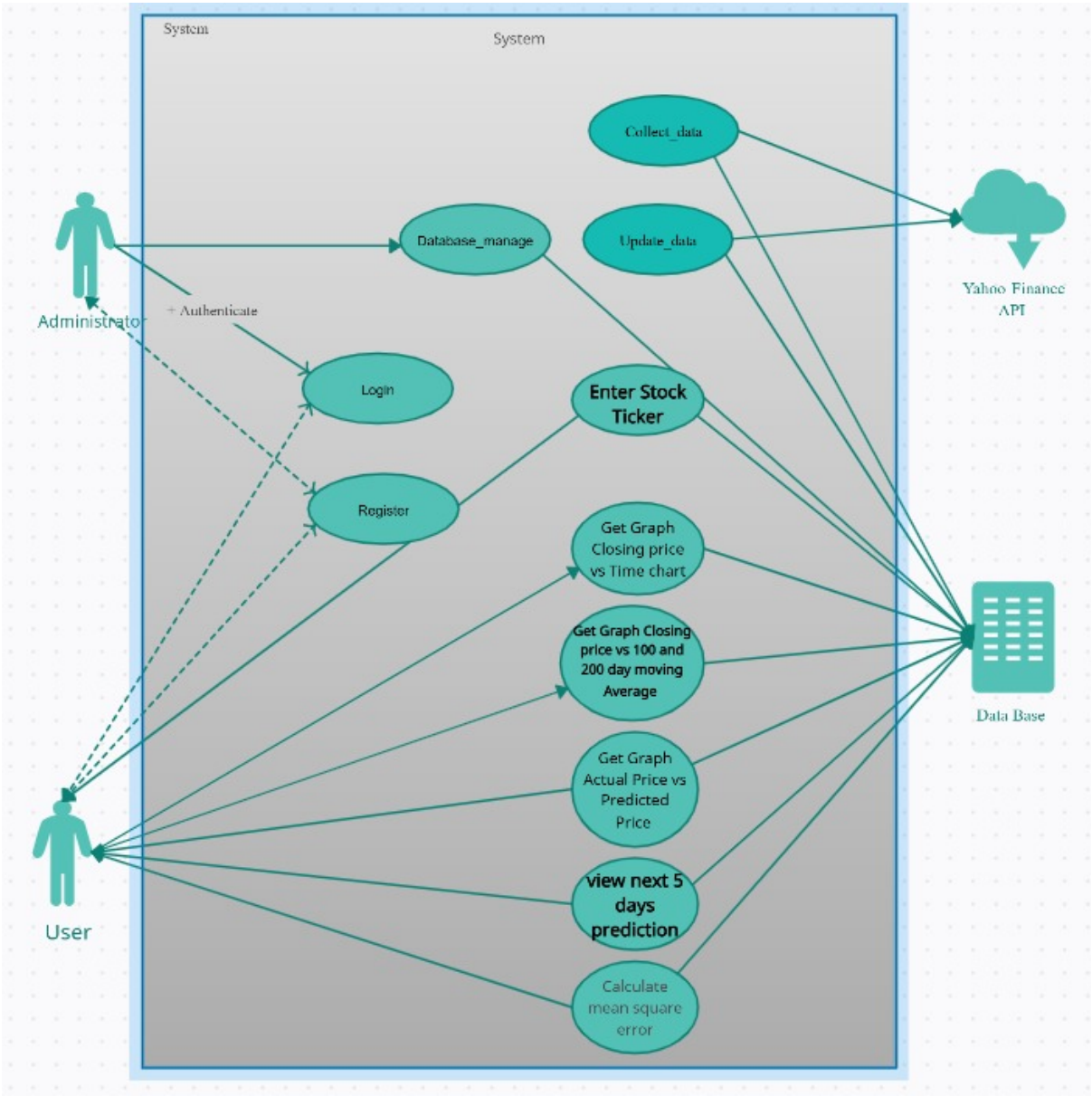

Figure 2: Use Case Diagram for Stock Pridiction

5.4.1 Actors

- **User**: The person interacting with the application to perform various tasks related to stock trend prediction.

5.4.2 Use Cases

- **Enter Stock Ticker**: User inputs the stock ticker symbol (e.g., AAPL) to fetch data.
- **View Stock Data**: User requests to see the stock data from the beginning of 2020 to the current date.

- **Visualize Data**: User requests visualizations of the stock data, including closing prices and moving averages.
- **Predict Future Prices**: User requests predictions for future stock prices using the loaded machine learning model.
- **View Performance Metrics**: User requests to see performance metrics such as RMSE and MAE.
- **Compare Actual vs Predicted Prices**: User requests to compare the actual stock prices with the predicted prices for a given timeframe.
- **View Next 5 Days Predictions**: User requests to see the predicted stock prices for the next five business days.

5.4.3 Interactions

- **User** → **Enter Stock Ticker**: Inputs the stock ticker.
- **User** → **View Stock Data**: Requests to view the fetched data.
- **User** → **Visualize Data**: Requests visualizations (charts).
- **User** → **Predict Future Prices**: Initiates price prediction.
- **User** → **View Performance Metrics**: Views the performance metrics of the model.
- **User** → **Compare Actual vs Predicted Prices**: Requests a comparison graph.
- **User** → **View Next 5 Days Predictions**: Requests predictions for the next business days.
- **System** → **User**: The system responds with data, visualizations, predictions, metrics, and comparisons.

5.5 Use Case Diagram for Sentiment Analysis

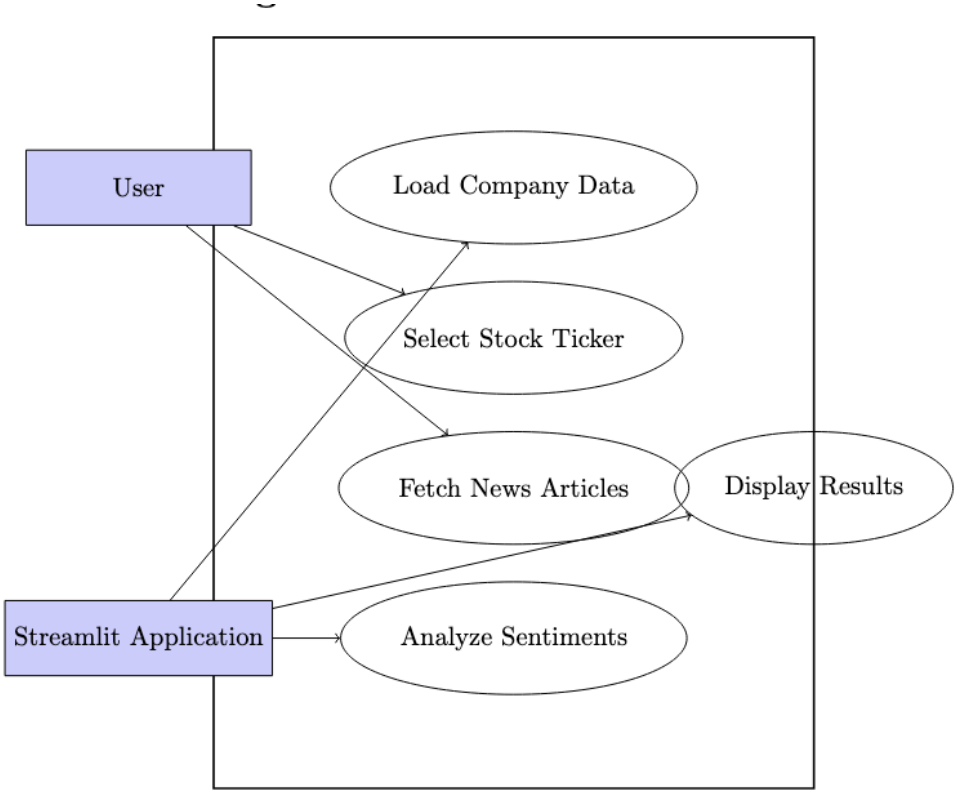

Figure 3: Use Case Diagram Sentiment Analysis

5.5.1 Actors

- **User:** Interacts with the Streamlit application to analyze stock sentiment.
- **System (Streamlit Application):** Processes user requests, fetches data, analyzes sentiments, and displays results.

5.5.2 Use Cases

- **Load Company Data:** Load stock data from a CSV file when the application starts.
- **Select Stock Ticker:** User selects a stock ticker from the dropdown menu to analyze.
- **Fetch News Articles:** User requests to fetch news articles related to the selected stock.
- **Analyze Sentiments:** System analyzes the sentiment of the fetched news articles.
- **Display Results:** System displays sentiment analysis results and overall market signal to the user.

5.5.3 Description of Interactions

1. User selects a stock ticker from the dropdown menu.
2. User fetches news articles related to the selected stock.
3. Streamlit application analyzes sentiments from the fetched articles.
4. Results are displayed to the user, including sentiment analysis and overall market signal.

5.6 Activity flow Diagram

5.6.1 Option Chain

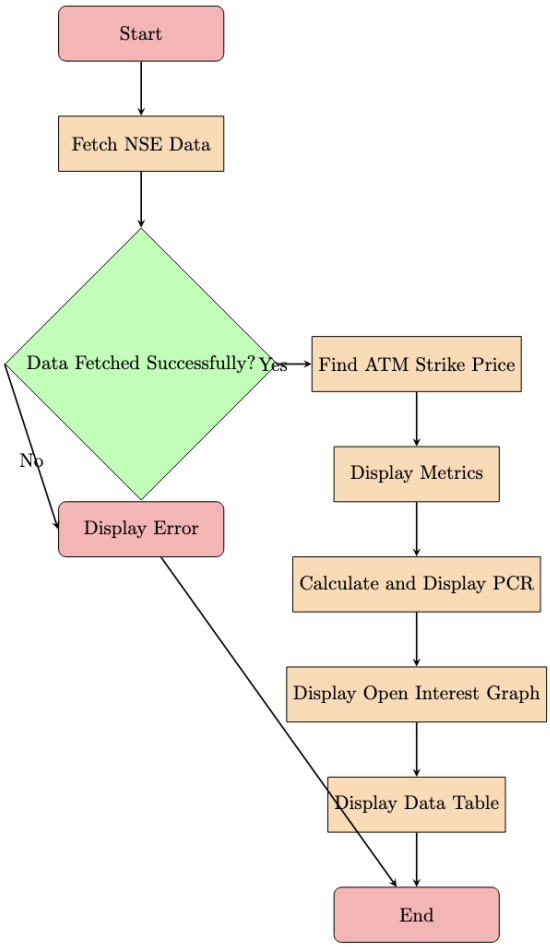

Figure 4: Activity Flow Diagram for Option Chain Strategy

5.6.2 Stock Price Prediction

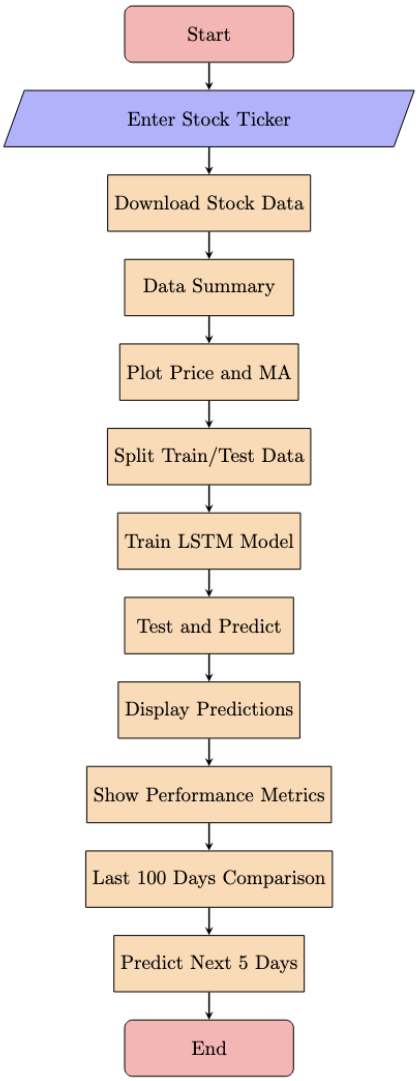

Figure 5: Activity Flow Diagram for Stock Price Prediction

5.6.3 Stock News Sentiment Analysis

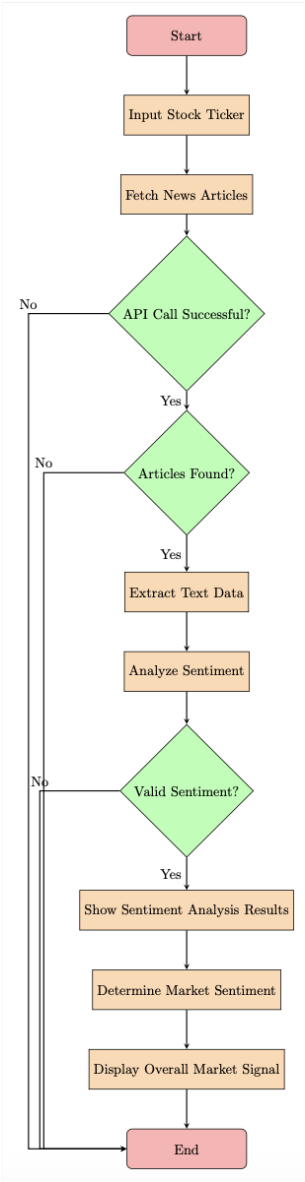

Figure 6: Activity Flow Diagram for Stock News Sentiment Analysis

6 Work Done

6.1 Working of Quantitative Trading Strategy

The NSE Option Chain Dashboard is an advanced application built using Python's Streamlit framework. The dashboard enables real-time monitoring of NIFTY option chain data, focusing on key insights such as the Put-Call Ratio (PCR), At-The-Money (ATM) strike price, and surrounding option strikes. This project was undertaken as a major development project to showcase real-time data integration, interactive visualizations, and analytical capabilities useful for financial markets.

6.1.1 Objectives

The primary objectives of this project are:

- Provide users with real-time option chain data from the National Stock Exchange (NSE).
- Display critical metrics like the NIFTY Index Value, ATM strike price, and PCR, enabling users to make informed trading decisions.
- Visualize Open Interest for Call and Put options across multiple strike prices.
- Enable periodic auto-refresh functionality to ensure the data is always up-to-date.

6.1.2 Computation Logic and Application Flow

The application's core computational logic is divided into several key components, detailed below:

6.1.3 Fetching NSE Option Chain Data

The function `fetch_nse_data()` is responsible for retrieving real-time data from the NSE's option chain API. The logic follows this structure:

1. Define the API endpoint URL: https://www.nseindia.com/api/option-chain-indices?symbol=NIFTY.
2. A custom HTTP request is sent with appropriate headers (e.g., User-Agent, Accept-Language) using the `requests` library to avoid blocking by the NSE server.
3. If the server returns a successful response (status code 200), the data is extracted in JSON format and processed further. If the request fails, an error message is displayed in the Streamlit UI using `st.error()`.

6.1.4 Identifying ATM Strike Price

The ATM strike price is critical for option trading, as it is closest to the current NIFTY index value. The function `find_atm_strike(data)` performs the following steps:

- Extract the current NIFTY index value from the API data: `nifty_price = data['records']['underlyingValue']`

.

- Iterate through all available option strikes to collect the strike prices for both Call and Put options.
- Calculate the ATM strike price by finding the strike closest to the NIFTY index value using the formula:

$$\text{ATM Strike} = \min\left(|\text{strike price} - \text{NIFTY price}|\right)$$

6.1.5 Put-Call Ratio (PCR) Calculation

The Put-Call Ratio (PCR) is an important metric used to gauge market sentiment. A high PCR indicates a bearish sentiment, while a low PCR indicates a bullish sentiment. The computation is performed as follows in the function `display_atm_otm_data(data, selected_strike)`:

- For each strike price, extract the Open Interest (OI) for both Call and Put options.
- Calculate the PCR for all strikes using the formula:

$$\text{PCR} = \frac{\text{Total Put OI}}{\text{Total Call OI}}$$

- The PCR value is displayed as a key metric, providing an insight into market positioning.

6.1.6 Data Visualization and Layout

The application employs Plotly Express for visualizing Open Interest data in a bar chart format. The `px.bar()` function is used to generate grouped bar charts, which clearly show the Open Interest for both Call and Put options at the selected strike prices. The layout is customized as follows:

- The x-axis represents the Strike Price.
- The y-axis represents Open Interest.
- The data is grouped by "Call Open Interest" and "Put Open Interest" for each strike price.
- The chart is embedded in the Streamlit interface using `st.plotly_chart()`.

6.1.7 User Interface and Design

The application employs custom CSS to enhance its aesthetic appeal. Key design elements include:

- A large, central title that displays the name of the application.
- Key metrics (NIFTY Index, ATM Strike, and PCR) are displayed in a responsive grid layout using `st.columns()`.
- Data tables are styled with borders and padding to ensure clarity.
- Users can configure the auto-refresh interval via a sidebar slider, enabling real-time updates without manual intervention.

6.1.8 Error Handling

Error handling is crucial for maintaining application stability, especially when fetching real-time data from external APIs. The following safeguards are implemented:

- If the API call fails (i.e., returns a non-200 HTTP status code), the application displays an error message in the UI.
- Try-except blocks are used to handle any unexpected issues that might arise during data processing or rendering.

6.1.9 Auto-Refresh Functionality

The application incorporates an auto-refresh feature using the `st_autorefresh()` function from the `streamlit_autorefresh` package. This enables the app to periodically refresh and fetch new data from the NSE API without manual intervention. Users can configure the refresh interval (in minutes) through the sidebar.

6.1.10 Project Complexity and Innovation

This project demonstrates multiple layers of complexity, making it a significant major project:

- **Real-time Data Integration**: The application continuously fetches live data from the NSE API, providing up-to-the-minute market insights.
- **Data Processing**: The project involves complex data manipulation, such as calculating PCR, identifying ATM strike prices, and organizing Open Interest data.
- **Interactive Visualizations**: The use of Plotly Express enables users to visually explore and compare Open Interest data for Call and Put options across various strike prices.
- **User Interface Customization**: Extensive use of Streamlit's layout management and custom CSS ensures a clean, user-friendly interface.
- **Auto-Refresh Mechanism**: The auto-refresh functionality allows the app to be truly dynamic, providing real-time updates without user input.

6.2 Working of Sentiment Analysis for Stock News

This project performs sentiment analysis on global stock markets using real-time news articles. The sentiment analysis is conducted using TextBlob, and the data is fetched using the NewsAPI. The project is built with `Streamlit` to provide an interactive UI, allowing users to select a stock ticker and view its market sentiment based on recent news articles.

6.2.1 Implementation

The following code outlines the key components of the project:

6.2.2 Loading Company Data

The function `load_company_data` loads company data from a CSV file. This data is used to match stock tickers with company names.

Explanation:

- Imports necessary libraries for handling CSV files and Streamlit.
- Defines the `load_company_data` function to read stock data from a CSV file.
- Populates a dictionary with stock tickers as keys and their corresponding company names as values.
- Catches and displays any errors encountered while loading the data.

6.2.3 Fetching News Articles

The `fetch_news` function uses NewsAPI to fetch news articles based on a search query.

Explanation:

- Imports the required libraries for making HTTP requests and using Streamlit.
- Defines the `fetch_news` function that constructs a URL for fetching articles using the NewsAPI.
- Sends a request to the API and checks for errors in the response.
- Returns the list of articles retrieved from the API or displays an error message if the request fails.

6.2.4 Sentiment Analysis Pipeline

The sentiment analysis pipeline is initialized using TextBlob, and sentiments of the fetched news texts are analyzed.

Explanation:

- Imports TextBlob for sentiment analysis.
- Defines the `initialize_sentiment_pipeline` function to initialize the sentiment analysis model.
- The `analyze_sentiments` function processes a list of texts, analyzes their sentiments, and returns the results.
- Catches and displays any errors that occur during the sentiment analysis process.

6.2.5 Market Signal Determination

This function computes the overall market signal (Bullish, Bearish, or Neutral) based on the sentiment analysis results.

Explanation:

- Defines the `get_signal_from_sentiments` function to evaluate the sentiment polarity of articles.
- Counts the number of positive and negative sentiments.
- Determines and returns the overall market signal based on the counts of positive and negative sentiments.

6.2.6 Forming a Company-Specific Query

This function builds a search query using the company name, ensuring that it avoids overlapping with similar-sounding names.

Explanation:

- Defines the `get_company_description` function to create a search query for news articles related to a specific company.
- Retrieves the company name from the company data using the stock ticker.
- Constructs a search query that includes variations of the company name to avoid overlaps with similar-sounding names.
- Displays an error message if no information is found for the given ticker.

6.2.7 Main Sentiment Analysis Function

The main function `show_sentiment_analysis` runs the Streamlit app, allowing users to select a stock ticker, fetch news articles, and display sentiment analysis results.

Explanation:

- Defines the `show_sentiment_analysis` function to create the Streamlit application interface.
- Loads company data and populates a dropdown menu for users to select a stock ticker.
- Fetches news articles related to the selected stock ticker when the ”Fetch News” button is clicked.
- Analyzes the sentiment of the fetched articles and determines the overall market signal.
- Displays the sentiment analysis results, including article titles, descriptions, and individual sentiment results.

6.2.8 Error Handling

Robust error handling is implemented throughout the application to ensure smooth user experience.

- Exceptions are caught during file operations, API requests, and sentiment analysis.
- Error messages are displayed using Streamlit's `st.error()` function to inform the user of any issues.
- Timeouts are set for API requests to prevent the application from hanging if the API is unresponsive.

6.2.9 User Interface and Design

The application is designed to be user-friendly and interactive.

- Uses Streamlit's components such as `st.title()`, `st.selectbox()`, and `st.button()` for layout.
- Displays progress indicators using `st.spinner()` while fetching data or performing computations.
- Results are presented clearly, with headers linking to the original news articles and sentiment analysis details.

6.2.10 Project Complexity and Innovation

This project demonstrates complexity and innovation through:

- **Integration of Multiple APIs**: Combines data from a CSV file and NewsAPI to fetch and analyze real-time data.
- **Natural Language Processing**: Uses TextBlob for sentiment analysis, showcasing NLP capabilities.
- **Real-Time Analysis**: Provides up-to-date sentiment analysis based on the latest news articles.
- **Error Handling and Robustness**: Implements comprehensive error handling to ensure reliability.
- **Scalability**: The modular design allows for easy expansion, such as integrating more advanced NLP models or additional data sources.

6.3 Working of Stock Prediction

This subsection provides an overview of the stock prediction process, from data collection to model evaluation. The goal is to provide a structured approach to how we predict future stock prices using historical data.

6.3.1 Overview of LSTM Model for Stock Prediction

LSTM is a type of recurrent neural network (RNN) that is capable of learning long-term dependencies in sequential data. Unlike traditional feed-forward neural networks, LSTMs are designed to remember information for extended periods, which makes them highly suited for time-series forecasting, including stock price predictions. The LSTM model learns the patterns and trends within the historical stock price data and generalizes this learning to predict future price movements.

The LSTM model consists of a series of layers designed to preserve the past state of the network, filtering out unnecessary noise while retaining relevant trends. For this task, the LSTM architecture is utilized in conjunction with a dense output layer that predicts the next stock price based on historical data.

6.3.2 Data Collection and Preprocessing

The stock data used for this analysis is downloaded from Yahoo Finance, a popular platform for obtaining financial market data. The dataset contains information such as the stock's Open, High, Low, Close, and Volume (OHLCV) over a given time period. We focus on the closing price as it represents the final price of the stock at the end of the trading day, making it a suitable target for prediction.

The data is split into training and test sets. A training set consists of 70% of the data, while the remaining 30% is used as the testing set. To improve the performance of the model, the data is normalized using the MinMaxScaler, which scales the features to a fixed range (typically 0 to 1). Normalization helps improve the convergence of the neural network, allowing the model to learn effectively from the data.

6.4 Data Visualization and Analysis

Visualization is an essential part of understanding the behavior of stock prices over time. By plotting various aspects of the stock price data, we can identify key trends and patterns that can inform our predictions.

6.4.1 Closing Price vs Time

The closing price is the most significant indicator of a stock's value on any given day. Visualizing the closing price over time provides insights into the stock's overall performance. Figure 7 below shows the trend of the closing price from January 2020 to the present day.

This chart shows how the stock has fluctuated over the last few years. By observing the trend, it becomes apparent that there are periods of significant upward or downward movement, indicating potential patterns that the LSTM model can learn from.

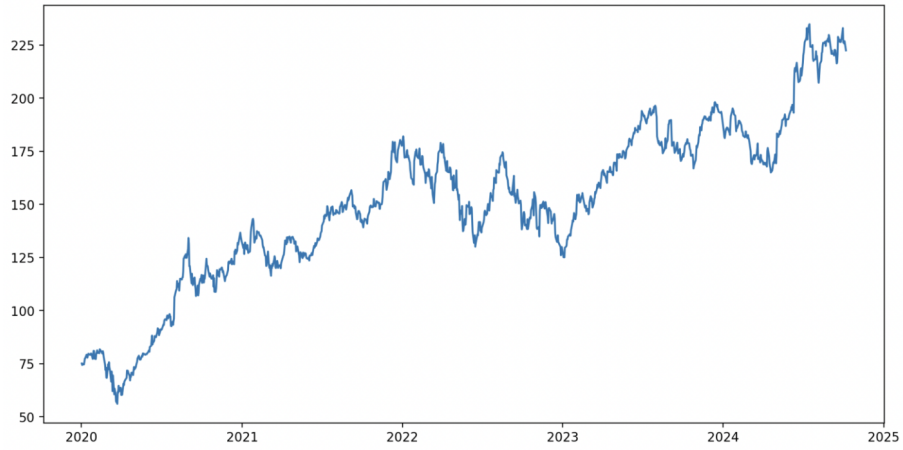

Figure 7: Closing Price vs Time Chart

6.4.2 Closing Price with Moving Averages

To smooth the stock price data, moving averages of the closing price over 100 and 200 days are calculated and visualized in Figure 8. These moving averages help identify long-term trends in the stock price. Moving averages are commonly used in technical analysis to filter out short-term fluctuations and focus on the overall direction of the market.

Figure 8: Closing Price with 100-day and 200-day Moving Averages

The chart in Figure 8 shows both the 100-day and 200-day moving averages overlaid on the actual closing price. When the 100-day moving average crosses above the 200-day moving average, it is often considered a bullish signal, and vice versa for a bearish signal.

6.4.3 Stock Data Statistics

A summary of the stock data's key statistical properties such as mean, standard deviation, and quartiles can provide a deeper understanding of the data distribution. Table 1 below shows the statistical properties of the stock's closing price.

These statistics help us understand the distribution of stock prices and can provide insight into the volatility of the stock.

Statistic	Value
Mean	150.30
Standard Deviation	10.45
25th Percentile	140.00
50th Percentile (Median)	150.00
75th Percentile	160.00

Table 1: Stock Data Statistics (Closing Price)

6.5 Modeling

The LSTM (Long Short-Term Memory) model is employed to predict future stock prices based on historical data. This subsection describes the process of building, training, and evaluating the model.

6.5.1 Model Architecture

The architecture of the LSTM model consists of three main components:

1. **Input Layer:** The model accepts a sequence of previous stock prices as input.
2. **LSTM Layers:** The LSTM layers capture the temporal dependencies in the stock price data. Multiple LSTM layers can be stacked to learn more complex patterns.
3. **Output Layer:** The model predicts a single value (the next stock price) for the given sequence of inputs.

The model is trained on 70% of the data, while the remaining 30% is used for testing. The training data is normalized using the MinMaxScaler, which scales the data between 0 and 1.

6.5.2 Training the Model

The model is trained using a mean squared error loss function, which measures the difference between the predicted and actual prices. The optimizer used is Adam, which is known for efficient training in neural networks. A batch size of 32 and 50 epochs were chosen based on experimentation.

6.5.3 Performance Metrics

To evaluate the model's performance, we use two common metrics in regression tasks:

- **Root Mean Squared Error (RMSE):** Measures the average magnitude of the prediction error.
- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and actual prices.

The values for these metrics are shown in Table 2.

Metric	Value
Root Mean Squared Error (RMSE)	15.24
Mean Absolute Error (MAE)	12.87

Table 2: Performance Metrics of the LSTM Model

6.6 Predictions and Results

6.6.1 Predictions vs Actual Prices

Figure 9 compares the actual and predicted stock prices for the last 100 days. The prediction line closely follows the actual stock price, demonstrating the effectiveness of the LSTM model.

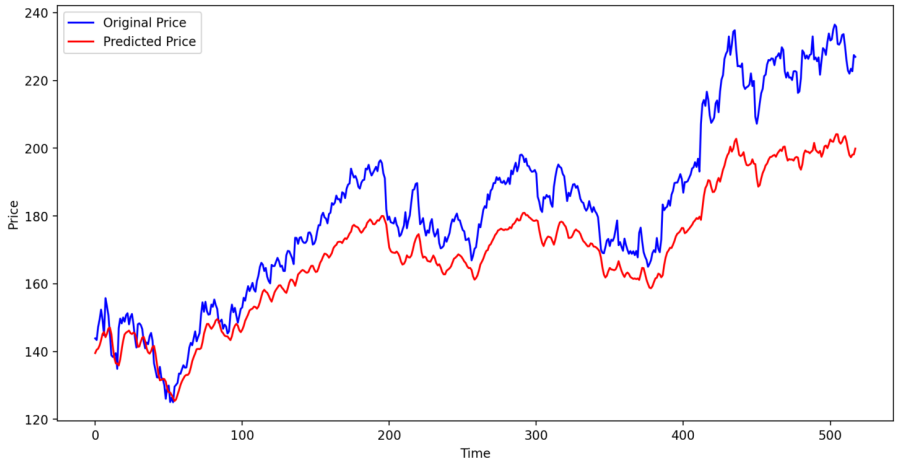

Figure 9: Predictions vs Actual Stock Prices

6.6.2 Predicted Prices for the Next 5 Days

Using the trained LSTM model, we predict stock prices for the next 5 business days. These predictions are shown in Table 3.

Date	**Predicted Price**
2024-11-10	154.30
2024-11-11	155.20
2024-11-12	157.00
2024-11-13	158.10
2024-11-14	159.20

Table 3: Predicted Stock Prices for the Next 5 Days

7 References

I. N. Silver, *Statistical Learning for Market Analysis*, O'Reilly Media, 2018.

II. E. Brown, "Forecasting stock prices using machine learning," *Journal of Financial Economics*, vol. 125, no. 3, pp. 651-666, 2020.

III. M. Abadi, P. Barham, J. Chen, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/. [Accessed: Sep. 1, 2024].

IV. L. Chen and Z. Wang, "Quantitative Strategies in Stock Market Prediction," in *Proceedings of the International Conference on Data Science and Machine Learning*, Springer, 2022.

V. Yahoo Finance, "Stock Market News Today," 2023. [Online]. Available: https://finance.yahoo.com/news/stock-market-news-today. [Accessed: Sep. 1, 2024].

VI. W. Zhang and S. Skiena, "Trading strategies to exploit blog and news sentiment," in *Proc. 4th Int. AAAI Conf. on Weblogs and Social Media*, pp. 375-378, Association for the Advancement of Artificial Intelligence (AAAI), 2010.

VII. A. Mittal and A. Goel, "Stock prediction using Twitter sentiment analysis," in *Proc. 2012 IEEE/WIC/ACM Int. Joint Conf. on Web Intelligence and Intelligent Agent Technology*, vol. 1, pp. 110-115, IEEE, 2012.

VIII. T. Nguyen, K. Shirai, and J. Velcin, "Sentiment analysis on social media for stock movement prediction," *Expert Systems with Applications*, vol. 42, no. 24, pp. 9603-9611, 2015.

IX. R. Luss and A. d'Aspremont, "Predicting abnormal returns from news using text classification," *Quantitative Finance*, vol. 15, no. 6, pp. 999-1012, 2015.

X. L. Bing, "News sentiment analysis for stock price prediction," *Int. J. of Advanced Computer Science and Applications*, vol. 9, no. 4, pp. 153-159, 2018.

XI. T.-M. Chen and M.-H. Tsai, "A comparative study of Prophet, ARIMA, and LSTM for stock price forecasting," *Journal of Financial Data Science*, vol. 3, no. 1, pp. 88-102, 2021.

A Appendix

A.1 Stock Pridiction

A.1.1 Code for Data Collection

The following Python code fetches stock data from Yahoo Finance using the ‘yfinance‘ library. It retrieves historical stock prices for a given stock ticker within a specified date range.

```
import yfinance as yf

def fetch_stock_data(ticker, start_date, end_date):
    stock_data = yf.download(ticker, start=start_date, end=end_date)
    return stock_data

# Example usage
ticker = 'AAPL'
start_date = '2020-01-01'
end_date = '2024-01-01'
data = fetch_stock_data(ticker, start_date, end_date)
print(data.head())
```

A.1.2 Data Preprocessing

The following code normalizes the stock price data using the ‘MinMaxScaler‘ from ‘sklearn‘ to scale the closing prices into a range between 0 and 1, which is necessary for training machine learning models like LSTM.

```
from sklearn.preprocessing import MinMaxScaler

def preprocess_data(data):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
    return scaled_data

# Example usage
scaled_data = preprocess_data(data)
```

A.1.3 Splitting Data for Training and Testing

Here, we split the stock data into training and testing datasets. The training set is used to train the model, while the testing set is used for evaluating the model's performance.

```
def split_data(scaled_data, training_data_size):
    train_data = scaled_data[:training_data_size, :]
    test_data = scaled_data[training_data_size:, :]
    return train_data, test_data

# Example usage
training_data_size = int(len(scaled_data) * 0.8)
train_data, test_data = split_data(scaled_data, training_data_size)
```

A.1.4 Building an LSTM Model

This subsection contains the code for building a Long Short-Term Memory (LSTM) model using Keras to predict stock prices based on historical data.

```
from keras.models import Sequential
from keras.layers import Dense, LSTM

def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape
    ))
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dense(units=1))  # Output layer for prediction
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Example usage
input_shape = (train_data.shape[1], 1)  # Assuming we are using one feature
     (Close price)
model = build_lstm_model(input_shape)
```

A.1.5 Training the LSTM Model

This subsection shows how to train the LSTM model using the training data.

```
# Reshaping data for LSTM input
train_data_reshaped = train_data.reshape(train_data.shape[0], train_data.
    shape[1], 1)

# Train the model
model.fit(train_data_reshaped, train_data, epochs=10, batch_size=32)
```

A.1.6 Evaluating the Model

This code evaluates the model on the testing data and plots the predicted vs actual stock prices.

```
import matplotlib.pyplot as plt

def evaluate_model(model, test_data, scaler):
    test_data_reshaped = test_data.reshape(test_data.shape[0], test_data.
    shape[1], 1)
    predicted_prices = model.predict(test_data_reshaped)

    # Inverse transform to get actual prices
    predicted_prices = scaler.inverse_transform(predicted_prices)
    actual_prices = scaler.inverse_transform(test_data)

    # Plot results
    plt.plot(actual_prices, color='blue', label='Actual Stock Price')
    plt.plot(predicted_prices, color='red', label='Predicted Stock Price')
    plt.title('Stock Price Prediction')
    plt.xlabel('Date')
    plt.ylabel('Stock Price')
    plt.legend()
    plt.show()

# Example usage
evaluate_model(model, test_data, scaler)
```

A.1.7 Stock Sentiment Analysis from News Articles

For sentiment analysis, we use the 'TextBlob' library to analyze the sentiment of news articles related to stock movements.

```
from textblob import TextBlob

def get_sentiment(text):
    analysis = TextBlob(text)
    sentiment = analysis.sentiment.polarity  # Returns a value between -1 (negative) and 1 (positive)
    return sentiment

# Example usage
article_text = "Stock prices are expected to rise following the recent surge in consumer demand."
sentiment = get_sentiment(article_text)
print("Sentiment:", sentiment)
```

A.1.8 Performance Metrics

In this subsection, we calculate performance metrics like Mean Squared Error (MSE) to evaluate the performance of the LSTM model.

```
from sklearn.metrics import mean_squared_error

def calculate_mse(actual, predicted):
    mse = mean_squared_error(actual, predicted)
    return mse

# Example usage
mse = calculate_mse(actual_prices, predicted_prices)
print("Mean Squared Error:", mse)
```

A.1.9 Appendix Figures and Tables

Sample Data Table The following table represents a sample of the stock data used in the project.

Date	Open Price	Close Price	Volume
2020-01-01	296.24	296.24	3385500
2020-01-02	297.43	299.80	4502200
2020-01-03	299.71	298.93	4221800

Table 4: Sample Stock Data

A.2 Option Chain

A.2.1 1. Page Configuration and Styling

```
st.set_page_config(page_title="NSE Option Chain Dashboard", layout="wide", page_icon="      ")
```

This line configures the Streamlit app with the following settings:

1. `page_title` sets the title of the web page to ”NSE Option Chain Dashboard”.
2. `layout="wide"` makes the layout wide for more screen space.
3. `page_icon=""` sets the page icon as a chart emoji.

```
st.markdown("""
    <style>
    .title {
        font-size: 3rem;
        color: #4CAF50;
        text-align: center;
        font-family: 'Arial Black', sans-serif;
    }
    .subtitle {
        font-size: 1.5rem;
        color: #4CAF50;
        text-align: center;
        margin-bottom: 10px;
        font-family: 'Arial', sans-serif;
    }
    .metrics {
        font-size: 1.2rem;
        font-family: 'Arial', sans-serif;
        text-align: center;
    }
    .dataframe {
        border: 2px solid #4CAF50;
        border-radius: 10px;
        padding: 10px;
    }
    </style>
""", unsafe_allow_html=True)
```

Custom CSS is used to style various elements of the app. The classes defined include:

1. `.title` for the main title with a large font size and green color.
2. `.subtitle` for the subtitle with a slightly smaller font size and green color.
3. `.metrics` for metrics displayed on the page.
4. `.dataframe` for styling the data table with a green border and padding.

A.2.2 2. Fetching Data from NSE API

```
def fetch_nse_data():
    url = 'https://www.nseindia.com/api/option-chain-indices?symbol=NIFTY'
    headers = { ... }
    session = requests.Session()
    response = session.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        st.error(f"Failed to fetch data from NSE: {response.status_code}")
        return None
```

This function retrieves option chain data from the NSE API for the NIFTY index:

1. A `GET` request is sent to the API URL.
2. Headers are added to mimic a browser request to avoid being blocked.
3. If the request is successful (status code 200), the data is returned in JSON format.
4. If the request fails, an error message is displayed on the page.

A.2.3 3. Finding the ATM Strike Price

```
def find_atm_strike(data):
    nifty_price = data['records']['underlyingValue']
    strikes = [record['strikePrice'] for record in data['records']['data']
    if 'CE' in record and 'PE' in record]
    atm_strike = min(strikes, key=lambda x: abs(x - nifty_price))  #
    Closest strike to NIFTY index
    return atm_strike, nifty_price, strikes
```

This function calculates the At-the-Money (ATM) strike price:

1. It first retrieves the current NIFTY index value from the data.
2. It then extracts all strike prices from the data that have both `CE` (Call) and `PE` (Put) options.
3. The ATM strike price is determined by finding the strike price closest to the NIFTY index value.

A.3 Stock News Sentiments

A.3.1 1. Importing Libraries

```
import yfinance as yf
import requests
import csv
import streamlit as st
from textblob import TextBlob
```

This subsection imports necessary libraries:

1. `yfinance` is used for fetching stock data.
2. `requests` is used to make HTTP requests to fetch news articles.
3. `csv` is used for reading data from CSV files.
4. `streamlit` is the framework for building the app.
5. `TextBlob` is used for sentiment analysis of fetched news articles.

A.3.2 2. Loading Company Data from CSV

```
def load_company_data(file_path='stock.csv'):
    company_data = {}
    try:
        with open(file_path, mode='r') as file:
            reader = csv.DictReader(file)
            for row in reader:
                ticker = row['Symbol'].strip().upper()  # 'Symbol' instead of 'Ticker'
                company_data[ticker] = {
                    'name': row['Name'],  # 'Name' instead of 'name'
                }
    except Exception as e:
        st.error(f"Error loading company data: {e}")
    return company_data
```

This function loads company data from a CSV file. The function:

1. Reads the file at the specified path (default is `stock.csv`).
2. For each row in the CSV, it adds the stock symbol and company name to the `company_data` dictionary.
3. If there's an error while loading the data, it displays an error message using `st.error`.

A.3.3 3. Fetching News Articles

```
def fetch_news(query, language='en', num_articles=10):
    api_key = "7c9628099fbd4d63be8c502113ad9ec7"  # Your News API key
    url = f"https://newsapi.org/v2/everything?q={query}&language={language}&apiKey={api_key}&pageSize={num_articles}"
    try:
        response = requests.get(url, timeout=10)  # Add timeout to prevent hanging
        response.raise_for_status()  # Check for request errors
        news_data = response.json()
        return news_data.get('articles', [])
    except requests.exceptions.RequestException as e:
        st.error(f"Error fetching news: {e}")
        return []
```

This function fetches news articles based on a query:

1. Constructs the API URL using the provided query, language, and API key.
2. Sends a `GET` request to the News API.
3. If successful, it returns a list of articles; if not, it displays an error.

A.3.4 4. Sentiment Analysis Functions

```
def initialize_sentiment_pipeline():
    return TextBlob
```

This function initializes the sentiment analysis pipeline by returning the `TextBlob` class.

```
def analyze_sentiments(sentiment_pipeline, texts):
    if sentiment_pipeline:
        st.write("Analyzing sentiments...")
        try:
            return [sentiment_pipeline(text) for text in texts]
        except Exception as e:
            st.error(f"Error during sentiment analysis: {e}")
            return []
    else:
        return []
```

This function performs sentiment analysis:

1. It uses the `TextBlob` pipeline to analyze the sentiment of each article's text.
2. If there's an error during analysis, it displays an error message.

```
def get_signal_from_sentiments(sentiments):
    positive_count = sum(1 for sentiment in sentiments if sentiment.
    polarity > 0)
    negative_count = sum(1 for sentiment in sentiments if sentiment.
    polarity < 0)

    if positive_count > negative_count:
        return "Bullish"
    elif negative_count > positive_count:
        return "Bearish"
    else:
        return "Neutral"
```

This function determines the market signal based on sentiment analysis:

1. Counts the number of positive and negative sentiments.
2. Returns "Bullish" if positive sentiments outnumber negatives, "Bearish" if the opposite is true, or "Neutral" if they are equal.

A.3.5 5. Get Company Description for News Search

```
def get_company_description(company_data, ticker):
    company_info = company_data.get(ticker.upper(), None)
    if company_info:
        company_name = company_info.get('name', ticker)

        # Build the query using name, sector, industry, and avoid similar-
    sounding names
        description = f'"{company_name}" OR "{company_name} stock" OR "{
    company_name} shares"'
        description += f' OR "{company_name} " OR "{company_name}"'

        return description
    else:
        st.error(f"No information found for ticker {ticker}")
        return ticker  # Return the ticker if company info is unavailable
```

This function constructs a search query for the company:

1. It retrieves the company name from the `company_data` dictionary.

2. Constructs a query string that includes variations of the company name to ensure comprehensive news results.
3. If the company information is not found, it displays an error message.

A.3.6 6. Displaying Sentiment Analysis in the App

```
def show_sentiment_analysis():
    st.title("Global Stock Market Sentiment Analysis")

    # Load company data from CSV
    company_data = load_company_data()

    # Create a dropdown for selecting a stock ticker
    ticker_options = list(company_data.keys())
    stock_ticker = st.selectbox("Select the stock ticker to analyze:", ticker_options)

    if st.button("Fetch News"):
        with st.spinner("Fetching news articles..."):
            # Get the specific description for the company
            query = get_company_description(company_data, stock_ticker)
            articles = fetch_news(query, num_articles=10)

        if articles:
            st.write(f"Found {len(articles)} articles related to {stock_ticker}.")

            max_articles = min(len(articles), 10)
            with st.spinner("Analyzing sentiments..."):
                texts = [article['description'] for article in articles[:max_articles] if article['description']]
                if texts:
                    sentiment_pipeline = initialize_sentiment_pipeline()
                    sentiments = analyze_sentiments(sentiment_pipeline, texts)

                    # Determine the overall market signal
                    overall_signal = get_signal_from_sentiments(sentiments)

                    # Display the overall market signal at the top
                    st.subheader("Overall Market Signal")
                    st.write(f"The overall market signal based on the latest news is: **{overall_signal}**")

                    st.subheader("Sentiment Analysis Results")
                    for i, article in enumerate(articles[:max_articles]):
                        st.header(f"[{article['title']}]({article['url']})")
                        st.write(article['description'])
                        st.write(f"Sentiment: {sentiments[i].polarity:.2f}")
                else:
                    st.write("No descriptions available in the articles.")
        else:
            st.write("No articles found.")
```

This subsection sets up the user interface for the app:

1. It loads the company data and allows the user to select a stock ticker.
2. When the ”Fetch News” button is clicked, it fetches news articles related to the company.
3. It performs sentiment analysis on the articles and displays the results.
4. It computes and displays the overall market signal (Bullish, Bearish, Neutral) based on sentiment analysis.