STOCK MARKET PREDICTION USING MACHINE LEARNING

**DESIGNED AND DEVELOPED**

**BY**

**NITESH CHAVAN PRN**

**Abstract**

The Stock Market Analysis and Prediction System is a web-based application developed as a part of the Bachelor of Computer Applications (BCA) project. This project aims to provide users with tools to analyze stock market data and make informed investment decisions.

The system incorporates various features including data fetching from external APIs, real-time updates on stock market trends, prediction of future stock prices, and visualization of data through charts and tables. Users can select specific stocks, view historical data, and receive predictions based on machine learning algorithms trained on past data.

Key functionalities of the system include:

* Fetching real-time and historical stock market data from external APIs.
* Displaying top gainers and losers in the stock market.
* Providing users with the ability to select specific stocks for analysis.
* Offering predictions on future stock prices using machine learning models.
* Presenting data in a user-friendly manner through interactive charts and tables.

The system is implemented using JavaScript for client-side scripting and interacts with backend servers to fetch and process data. Technologies such as AJAX, jQuery, and Chart.js are utilized to enhance user experience and provide dynamic content updates.

Overall, the Stock Market Analysis and Prediction System offers a comprehensive platform for users to gain insights into stock market trends, make informed investment decisions, and improve their financial literacy.

**Acknowledgment**

I am delighted to present my project **Stock Market Analysis and Prediction System**. I am deeply grateful to all those who provided me with invaluable assistance and guidance throughout the development of this project.

I would like to express my heartfelt gratitude to the management of **B. N. BANDODKAR COLLEGE OF SCIENCE** for granting me the opportunity to undertake and complete this project work. Their support and encouragement have been instrumental in the successful completion of this endeavor.

A special word of thanks goes to our project guide, **Mr. Tejas Jadhav**, for their sincere, invaluable, and encouraging contributions throughout the duration of the project. Their guidance and expertise have been indispensable in navigating the complexities of this project and ensuring its timely completion.

I am indebted to the faculty members of the Computer Science & IT department for their unwavering support and assistance at every step of the project. Their knowledge and insights have greatly enriched the development process.

I would also like to extend my gratitude to the non-teaching staff members who provided assistance and support whenever needed.

Furthermore, I am grateful to my project team member, Chaitanya Bhamare and Sukhdev Jadhav for their dedication and collaboration in developing this project. Their contributions have been instrumental in bringing this project to fruition.

**DECLARATION**

I here by the declare that the project entitled, “**Stock Market Analysis and Prediction System**” done at **B. N. Bandodkar College of Science**, has not been in any case duplicate to submit to any other university for the award of any degree. To best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirement for the award of degree of **BACHAELOR OF COMPUTER APPLICATIONS** to be submitted as final semester project as part of our curriculum.

**Name and Signature of the Student**

**SYNOPSIS**

**Introduction**:

The Stock Market Analysis and Prediction System (SMAPS) is a comprehensive software solution designed to facilitate the analysis and prediction of stock market trends. This system integrates various tools and algorithms to provide insights into market behaviour, allowing users to make informed investment decisions. SMAPS aims to streamline the process of analyzing stock market data, identifying patterns, and predicting future price movements.

**Current System:**

Currently, investors rely on manual methods or basic software tools for stock market analysis. These methods often involve time-consuming data collection, manual analysis, and subjective decision-making. While some investors use technical analysis techniques, others rely on fundamental analysis or expert opinions. However, these approaches may lack accuracy, efficiency, and scalability, limiting their effectiveness in dynamic market conditions.

**Problems Faced in Current System:**

The manual and traditional approaches to stock market analysis pose several challenges. These include the need for extensive data processing, limited scalability, and difficulty in identifying relevant trends and patterns. Moreover, the reliance on subjective analysis and expert opinions may introduce biases and errors, leading to suboptimal investment decisions. Additionally, existing tools may lack advanced predictive capabilities, hindering investors' ability to anticipate market movements accurately.

**Proposed System:**

The proposed Stock Market Analysis and Prediction System (SMAPS) aims to address these challenges by leveraging advanced data analytics techniques and machine learning algorithms. SMAPS offers features such as historical data analysis, trend identification, pattern recognition, and predictive modeling. By analyzing large volumes of historical and real-time market data, SMAPS provides users with actionable insights and forecasts, empowering them to make informed investment decisions. Moreover, the system is designed to adapt to changing market conditions and refine its predictive models over time.

**Key Features:**

* Historical Data Analysis: Analyze historical market data to identify trends and patterns.
* Predictive Modeling: Use machine learning algorithms to forecast future price movements.
* Real-time Market Monitoring: Monitor real-time market data and update predictions accordingly.
* Portfolio Management: Manage investment portfolios and track performance over time.
* Customizable Dashboards: Customize dashboards to visualize key metrics and indicators.

**Software Requirements:**

* Operating System: Windows 10 or later
* Programming Language: Python 3.x
* Libraries: NumPy, pandas, scikit-learn, Matplotlib, TensorFlow/Keras (for deep learning models)

**Hardware Requirements:**

* Memory: Minimum 8 GB RAM
* Processor: Intel Core i5 or equivalent
* Storage: Minimum 100 GB HDD/SSD

**Tenure:**

The development and implementation of the Stock Market Analysis and Prediction System (SMAPS) are estimated to be completed within 4 months, including testing and optimization phases.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** |  | **Contents** | **Page No.** |
| **1.** |  | **Introduction** |  |
|  | 1.1 | Background |  |
|  | 1.2 | Objectives |  |
|  | 1.3 | Scope |  |
| **2.** |  | **Literature Review** |  |
|  | 2.1 | Overview of Stock Market Analysis |  |
|  | 2.2 | Existing Systems and Tools |  |
|  | 2.3 | Challenges and Limitations |  |
| **3.** |  | **System Architecture** |  |
|  | 3.1 | Overview of System Components |  |
|  | 3.2 | Data Collection and Pre-processing |  |
|  | 3.3 | Analysis and Prediction Modules |  |
|  | 3.4 | User Interface |  |
|  | 3.5 | Event Table |  |
|  | 3.6 | Data Design |  |
| **4.** |  | **Methodology** |  |
|  | 4.1 | Data Collection Methods |  |
|  | 4.2 | Data Pre-processing Techniques |  |
|  | 4.3 | Analysis Algorithms |  |
|  | 4.4 | Prediction Models |  |
| **5.** |  | **Implementation** |  |
|  | 5.1 | Environment Setup |  |
|  | 5.2 | Data Acquisition |  |
|  | 5.3 | Data Pre-processing |  |
|  | 5.4 | Algorithm Implementation |  |
|  | 5.5 | User Interface Development |  |
| **6.** |  | **Testing and Evaluation** |  |
|  | 6.1 | Testing Strategies |  |
|  | 6.2 | Performance Metrics |  |
|  |  | Validation and Verification |  |
| **7.** |  | **Results and Discussion** |  |
|  | 7.1 | Analysis of Historical Data |  |
|  | 7.2 | Predictive Model Performance |  |
|  | 7.3 | User Feedback |  |
| **8.** |  | **Conclusion** |  |
|  | 8.1 | Summary of Findings |  |
|  | 8.2 | Achievements and Contributions |  |
|  | 8.3 | Future Work |  |
|  |  |  |  |

|  |
| --- |
| **INTRODUCTON** |

* 1. **Background:**

The stock market is a dynamic and complex environment where investors buy and sell financial securities, aiming to generate profits based on market fluctuations. Understanding and predicting these fluctuations is crucial for investors and financial analysts to make informed decisions. Traditionally, stock market analysis relied on manual methods and basic statistical tools. However, with advancements in technology and data science, automated systems and predictive models have become increasingly popular for analyzing and forecasting stock market trends.

**1.2 Objectives:**

The primary objective of the Stock Market Analysis and Prediction System is to develop a comprehensive software solution that leverages data analytics and machine learning techniques to analyze historical stock market data and predict future price movements. The system aims to achieve the following objectives:

* + - * 1. Provide accurate and timely analysis of historical stock market data to identify trends, patterns, and anomalies.
        2. Develop predictive models capable of forecasting future stock prices based on historical data, market indicators, and other relevant factors.
        3. Enhance decision-making processes for investors, traders, and financial analysts by providing actionable insights and recommendations.
        4. Improve risk management strategies by identifying potential risks and uncertainties in the stock market.
        5. Facilitate the development of trading strategies and investment portfolios tailored to individual risk preferences and financial goals.

**1.3 Scope:**

The Stock Market Analysis and Prediction System will focus on the following key areas:

* + - * 1. Data Collection: Gathering comprehensive and reliable historical stock market data from various sources, including financial databases, APIs, and online repositories.
        2. Data Preprocessing: Cleaning, filtering, and transforming raw data to ensure consistency, accuracy, and compatibility with analytical models.
        3. Analysis Techniques: Implementing advanced statistical analysis, data visualization, and machine learning algorithms to uncover meaningful insights from the data.
        4. Prediction Models: Developing predictive models, such as regression analysis, time series forecasting, and neural networks, to forecast future stock prices and market trends.
        5. User Interface: Designing an intuitive and user-friendly interface that allows users to interact with the system, visualize data, and access analytical tools and reports.
        6. Performance Evaluation: Assessing the accuracy, reliability, and effectiveness of the predictive models through rigorous testing and validation procedures.
        7. Deployment and Integration: Deploying the system on scalable and secure infrastructure, and integrating it with existing trading platforms, financial systems, and data sources.

|  |
| --- |
| **Literature Review** |

**2.1 Overview of Stock Market Analysis:**

Stock market analysis involves the evaluation of financial data, market trends, and other relevant factors to make informed decisions regarding investments and trading activities. It encompasses various techniques and methodologies aimed at understanding the behaviour of financial markets and predicting future price movements. Stock market analysis can be broadly categorized into two main approaches:

1. Fundamental Analysis: This approach involves analyzing the intrinsic value of a stock based on factors such as company financials, earnings reports, industry trends, economic indicators, and market conditions. Fundamental analysts assess the financial health and growth prospects of companies to determine whether a stock is undervalued or overvalued relative to its market price.
2. Technical Analysis: Technical analysis focuses on studying historical price and volume data, market trends, chart patterns, and technical indicators to forecast future price movements. Technical analysts believe that past price behavior can provide insights into future price direction, and they use tools such as moving averages, relative strength index (RSI), and candlestick patterns to identify trading opportunities.

Stock market analysis plays a crucial role in guiding investment decisions, portfolio management, risk assessment, and trading strategies. By leveraging advanced analytical techniques and predictive models, analysts and investors aim to gain a competitive edge in the market and achieve their financial goals.

**2.2 Existing Systems and Tools**

There are several existing systems and tools available for stock market analysis, ranging from basic charting platforms to advanced algorithmic trading platforms. Some of the commonly used systems and tools include:

1. **Bloomberg Terminal**: Bloomberg Terminal is a comprehensive financial data and analytics platform used by investors, traders, and financial professionals worldwide. It provides real-time market data, news, research reports, and analytical tools for stock market analysis and trading.
2. **Yahoo Finance**: Yahoo Finance is a popular web-based platform that offers free access to stock market data, news, charts, and financial information. It provides users with tools for portfolio tracking, technical analysis, and fundamental research.
3. **MetaTrader**: MetaTrader is a widely used trading platform for forex and stock market trading. It offers advanced charting tools, technical indicators, and automated trading capabilities through expert advisors (EAs) and custom scripts.
4. **TradingView**: TradingView is a cloud-based charting and social networking platform for traders and investors. It provides interactive charts, technical analysis tools, and a community-driven platform for sharing trading ideas and strategies.
5. **Python Libraries**: Python libraries such as Pandas, NumPy, and Matplotlib are commonly used for data analysis, visualization, and algorithmic trading. These libraries provide powerful tools for processing financial data, building predictive models, and backtesting trading strategies.

**2.3 Challenges and Limitations**

Despite the advancements in stock market analysis tools and techniques, there are several challenges and limitations associated with predicting stock prices and market trends:

1. **Market Volatility**: Stock markets are inherently volatile, and price movements can be influenced by various unpredictable factors such as economic events, geopolitical tensions, and market sentiment. Predicting stock prices with high accuracy in volatile markets can be challenging.
2. **Data Quality and Availability**: Stock market data may suffer from inaccuracies, inconsistencies, and gaps, especially when dealing with historical data from different sources. Ensuring data quality and availability is crucial for building reliable predictive models.
3. **Model Overfitting**: Overfitting occurs when a predictive model captures noise or random fluctuations in the data rather than underlying patterns. Overfit models may perform well on historical data but fail to generalize to new data, leading to poor predictive performance.
4. **Complexity of Market Dynamics**: Stock markets exhibit complex and nonlinear dynamics influenced by a wide range of factors, including investor behavior, market psychology, and institutional trading. Capturing and modeling these dynamics accurately can be challenging.
5. **Regulatory Compliance**: Compliance with regulatory requirements and ethical standards is essential in stock market analysis, especially when using predictive models for trading or investment purposes. Ensuring transparency, fairness, and compliance with regulations is paramount.

Addressing these challenges requires a combination of advanced analytical techniques, robust data management practices, domain expertise, and continuous monitoring and evaluation of predictive models. Despite the limitations, stock market analysis remains an essential tool for investors and financial professionals seeking to navigate the complexities of financial markets and achieve their investment objectives.

|  |
| --- |
| **System Architecture** |

The system architecture of the Stock Market Analysis and Prediction System is designed to integrate various components for efficient data collection, preprocessing, analysis, prediction, and user interaction. It comprises several modules that work together to provide users with valuable insights into stock market trends and facilitate informed decision-making.

**3.1 Overview of System Components**

The system components are organized into the following modules:

1. **Data Collection and Preprocessing Module**: This module is responsible for gathering raw data from multiple sources, such as financial databases, APIs, and web scraping tools. The collected data is then preprocessed to clean, normalize, and standardize it for further analysis.
2. **Analysis and Prediction Modules**: These modules encompass a range of analytical techniques and predictive models for studying historical market data, identifying patterns, and forecasting future price movements. The analysis module may include fundamental analysis, technical analysis, sentiment analysis, machine learning models, and statistical forecasting methods.
3. **User Interface Module**: The user interface module provides an intuitive interface for users to interact with the system, visualize data, explore insights, and access analytical tools and features. It may include web-based dashboards, charts, graphs, and customizable reports to facilitate data exploration and decision-making.

**3.2 Data Collection and Pre-processing**

The data collection and preprocessing module involve the following steps:

* **Data Gathering**: Raw data is collected from various sources, including financial markets, news websites, social media platforms, and economic indicators.
* **Data Cleaning**: The collected data is cleaned to remove inconsistencies, errors, missing values, and outliers that could affect the accuracy of analysis and prediction.
* **Data Transformation**: Data is transformed and standardized to ensure consistency and compatibility across different datasets. This may involve normalization, scaling, and feature engineering techniques.
* **Data Integration**: Multiple datasets are integrated and merged to create a comprehensive dataset that covers relevant aspects of the stock market, such as price data, trading volumes, company fundamentals, and market sentiment.

**3.3 Analysis and Prediction Modules**

The analysis and prediction modules comprise the following functionalities:

* **Historical Data Analysis**: Historical market data is analyzed using various techniques, such as statistical analysis, trend analysis, and pattern recognition, to gain insights into past market behavior and identify recurring patterns.
* **Predictive Modeling**: Predictive models are built using machine learning algorithms, time series analysis, and other forecasting methods to predict future price movements, trends, and volatility in the stock market.
* **Risk Assessment**: Risk assessment models are employed to evaluate the potential risks associated with investment decisions and trading strategies. This includes measuring volatility, calculating risk-adjusted returns, and assessing portfolio diversification.

**3.4 User Interface**

The user interface module offers the following features:

* **Dashboard**: A customizable dashboard provides users with an overview of key market indicators, portfolio performance, and relevant news and events.
* **Visualization Tools**: Interactive charts, graphs, and visualizations allow users to explore market data, analyze trends, and compare performance metrics.
* **Analytical Tools**: Built-in analytical tools and indicators enable users to conduct in-depth analysis, perform technical analysis, and generate trading signals.
* **Alerts and Notifications**: Automated alerts and notifications keep users informed about significant market events, price movements, and portfolio changes in real-time.

**Activity Flow:**

1. Start
3. --> User interacts **with** the dashboard
4. --> User selects a stock symbol from the dropdown menu
5. --> System triggers the retrieveData() **function**
6. --> System sends a request to fetch data **for** the selected symbol
7. --> Server processes the request
8. --> Server retrieves historical data **for** the selected symbol
9. --> Server sends the data back to the client
10. --> Server sends a response to the client
11. --> System updates the historical data chart **with** the retrieved data
12. --> System triggers the predict() **function**
13. --> System sends a request to predict future data **for** the selected symbol
14. --> Server processes the prediction request
15. --> Server sends the predicted data back to the client
16. --> Server sends a response to the client
17. --> System updates the predicted chart **with** the predicted data
18. --> User interacts **with** the top gainers or top losers section
19. --> User views the list of top gainers or top losers
20. --> User interacts **with** other elements **in** the dashboard (buttons, tables, etc.)
21. --> System performs corresponding actions based on user interaction
22. --> User interacts **with** the navbar
23. --> User navigates to different sections/pages of the dashboard
24. End

The user interface module is designed to be user-friendly, responsive, and customizable to meet the diverse needs of investors, traders, and financial professionals. It provides seamless access to analytical tools and insights, empowering users to make informed decisions and stay ahead in the dynamic world of stock market trading and investment.

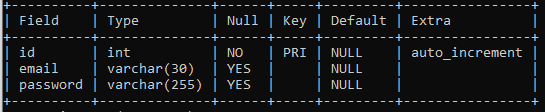
**3.5 Event Table**

| **Sr. No.** | **Event** | **Trigger** | **Source** | **Activity** | **Response** | **Destination** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | User Registration | Registration | User | User signs up for the system | Account created successfully | User Profile |
| 2 | User Login | Login | User | User provides credentials to access the system | Logged in successfully | Dashboard |
| 3 | Data Collection | Automatic/Manual | System | System fetches historical stock market data | Data collected and stored successfully | System |
| 4 | Data Preprocessing | Automatic | System | Raw data is cleaned, transformed, and prepared for analysis | Data preprocessed successfully | System |
| 5 | Analysis | Manual/Automatic | System | System performs analysis on historical stock market data | Analysis completed successfully | System |
| 6 | Prediction Model Training | Automatic | System | System trains prediction models based on historical data | Models trained successfully | System |
| 7 | Prediction | Manual/Automatic | User | User requests prediction of future stock prices | Predictions generated successfully | User |
| 8 | View Historical Data | User Interaction | User | User requests to view historical stock market data | Historical data displayed | User |
| 9 | View Prediction Results | User Interaction | User | User views predictions generated by the system | Prediction results displayed | User |
| 10 | Evaluate Prediction Performance | User Interaction | User | User assesses the performance of prediction models | Performance metrics displayed | User |
| 11 | System Maintenance | Scheduled | System | Regular system maintenance tasks | Maintenance tasks completed | System |
| 12 | Error Handling | Exception | System | System encounters errors during data processing or prediction | Errors logged and resolved | System |
| 13 | Logout | Logout | User | User logs out of the system | Logged out successfully | Login Screen |

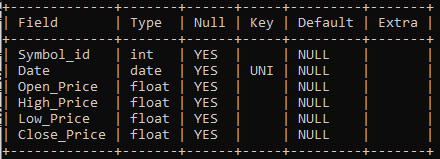
**3.6 Data Design**

By using the data design designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to the database model. Database design involves classifying data and identifying interrelationships between the data.

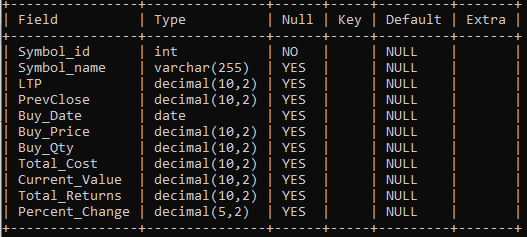
**Login Table:-**



**Stock Table:-**



**Portfolio Table:-**



|  |
| --- |
| **Methodology** |

The methodology of the Stock Market Analysis and Prediction System encompasses various stages, including data collection, preprocessing, analysis, and prediction. Each stage involves specific methods and techniques to extract meaningful insights from raw market data and generate accurate predictions of future price movements.

**4.1 Data Collection Methods**

The data collection methods involve obtaining relevant data from multiple sources, including financial databases, market exchanges, news outlets, social media platforms, and economic indicators. Common methods include:

* **API Integration**: Utilizing APIs provided by financial data providers, market exchanges, and news agencies to access real-time and historical market data.
* **Web Scraping**: Extracting data from websites, forums, and social media platforms using web scraping tools and techniques.
* **Database Queries**: Querying financial databases and repositories to retrieve structured data on stock prices, trading volumes, company fundamentals, and economic indicators.
* **Data Feeds**: Subscribing to data feeds and market streams provided by exchanges, financial institutions, and data vendors for up-to-date market information.

**User Authentication Routes:**

1. @app.route('/signup', methods**=**['GET', 'POST'])
2. **def** signup():
3. **if** request.method **==** 'POST':
4. email **=** request.form['email']
5. password **=** request.form['password']
6. hashed\_password **=** generate\_password\_hash(password)
7. cursor **=** db.cursor()
8. cursor.execute("INSERT INTO logins (email, password) VALUES (%s, %s)", (email, hashed\_password))
9. db.commit()
10. cursor.close()
11. **return** redirect(url\_for('index'))

* Handles user sign-up. It accepts POST requests with email and password, hashes the password, and stores the user credentials in the database.

1. @app.route('/signin', methods**=**['GET', 'POST'])
2. **def** signin():
3. **if** request.method **==** 'POST':
4. email **=** request.form['email']
5. password **=** request.form['password']
6. cursor **=** db.cursor()
7. cursor.execute("SELECT password FROM logins WHERE email = %s", (email,))
8. result **=** cursor.fetchone()
9. cursor.close()
10. **if** result:
11. stored\_password **=** result[0]
12. **if** check\_password\_hash(stored\_password, password):
13. session['email'] **=** email
14. **return** redirect(url\_for('index'))
15. **return** 'Invalid username/password'
16. **return** render\_template('signin.html')

* Handles user sign-in. It checks the provided email and password against the database and starts a session if authentication is successful.

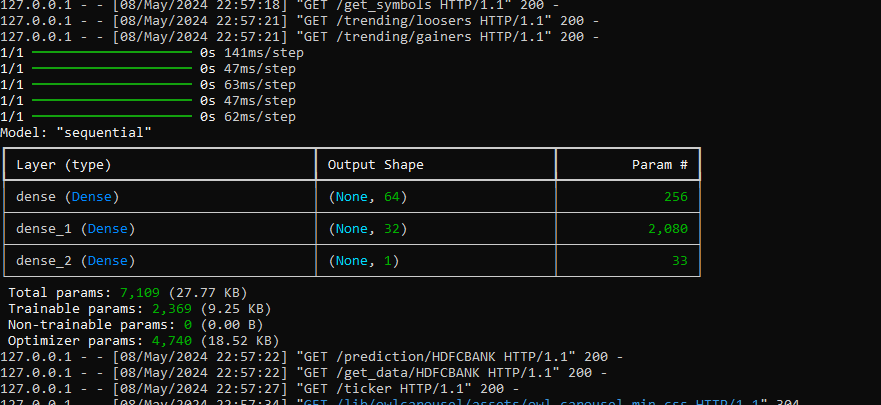
1. @app.route('/logout')
2. **def** logout():
3. session.pop('email', None)
4. **return** redirect(url\_for('signin'))

**Fetching Stock Data**

1. @app.route('/update\_data/<symbol>')
2. **def** update\_data(symbol):
3. **if** 'email' **in** session:
4. # Retrieving necessary data from the database
5. cursor **=** db.cursor()
6. cursor.execute(f"SELECT Symbol\_id FROM `{symbol}`Limit 1")
7. data **=** cursor.fetchone()
8. symbol\_id **=** data[0]
9. cursor.close()
11. # URL and headers for making the API request
12. url **=** "https://api.bseindia.com/BseIndiaAPI/api/StockPriceCSVDownload/w"
13. headers **=** {
14. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
15. "Referer": "https://www.bseindia.com/",
16. }
18. # Payload data for the API request
19. fromDate **=** '01/02/2024'
20. toDate **=** datetime.now().strftime('%d/%m/%Y')
21. payload **=** {
22. "pageType": "0",
23. "rbType": "D",
24. "Scode": symbol\_id,
25. "FDates": fromDate,
26. "TDates": toDate
27. }
29. **try**:
30. # Sending the API request
31. response **=** requests.get(url, headers**=**headers, params**=**payload, timeout**=**5)
33. # Handling timeout errors
34. **except** requests.exceptions.Timeout:
35. **return** jsonify({"error": "Request timed out. Please try again later."}),504
37. **if** response.status\_code **==** 200:
38. # Processing the CSV response
39. csv\_buffer **=** StringIO(response.text)
40. csv\_reader **=** csv.DictReader(csv\_buffer, delimiter**=**'\t')
42. # Iterating over CSV rows and updating the database
43. **for** row **in** csv\_reader:
44. data **=** list(row.values())[0].split(',')
45. date\_str **=** data[0]
46. date\_obj **=** datetime.strptime(date\_str, "%d-%B-%Y")
47. mysql\_date\_format **=** date\_obj.strftime("%Y-%m-%d")
48. open\_price **=** data[1]
49. high\_price **=** data[2]
50. low\_price **=** data[3]
51. close\_price **=** data[4]
53. # Inserting data into the respective table
54. cursor.execute(
55. f"INSERT IGNORE INTO {symbol} (Symbol\_id, Date, Open\_Price, High\_Price, Low\_Price, Close\_Price) VALUES (%s, %s, %s, %s, %s, %s)",
56. (symbol\_id, mysql\_date\_format, open\_price, high\_price, low\_price, close\_price))
57. db.commit()
59. cursor.close()
60. **return** jsonify({'success': True})
61. **else**:
62. **return** jsonify({"error": "Failed to fetch data"}), response.status\_code
63. **except** Exception as e:
64. **return** jsonify(error**=**str(e)), 500
65. **return** redirect(url\_for('signin'))

**4.2 Data Preprocessing Techniques**

Data preprocessing techniques are applied to clean, transform, and prepare raw data for analysis and modeling. Common preprocessing techniques include:



* **Data Cleaning**: Removing inconsistencies, errors, missing values, and outliers from the dataset to ensure data integrity and accuracy.
* **Normalization and Scaling**: Scaling numerical features to a standard range or normalizing them to improve model performance and convergence.
* **Feature Engineering**: Creating new features or transforming existing ones to capture relevant information and improve predictive power.
* **Dimensionality Reduction**: Reducing the dimensionality of the dataset using techniques like principal component analysis (PCA) or feature selection to enhance computational efficiency and reduce overfitting.

**4.3 Analysis Algorithms**

Analysis algorithms are employed to extract insights from historical market data and identify patterns, trends, and correlations. Common analysis algorithms include:

* **Technical Analysis**: Utilizing technical indicators, chart patterns, and statistical tools to analyze historical price and volume data and identify trading opportunities.
* **Fundamental Analysis**: Evaluating company fundamentals, financial statements, industry trends, and macroeconomic factors to assess the intrinsic value of stocks and make investment decisions.
* **Sentiment Analysis**: Analyzing market sentiment and investor sentiment from news articles, social media posts, and online forums to gauge market sentiment and sentiment-driven price movements.
* **Statistical Analysis**: Applying statistical techniques such as regression analysis, time series analysis, and correlation analysis to quantify relationships and dependencies within the data.

**4.4 Prediction Models**

Prediction models are developed to forecast future stock prices, trends, and volatility based on historical data and relevant features. Common prediction models include:

* **Time Series Forecasting**: Using time series analysis techniques such as ARIMA (AutoRegressive Integrated Moving Average) models, exponential smoothing methods, and LSTM (Long Short-Term Memory) networks to predict future price movements.
* **Machine Learning Models**: Employing machine learning algorithms such as regression models, decision trees, random forests, support vector machines (SVM), and neural networks to learn patterns from historical data and make predictions.
* **Ensemble Methods**: Combining multiple predictive models using ensemble methods like bagging, boosting, and stacking to improve prediction accuracy and robustness.
* **Deep Learning Models**: Leveraging deep learning architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for feature learning and sequential prediction tasks in stock market forecasting.

By employing a combination of data collection methods, preprocessing techniques, analysis algorithms, and prediction models, the Stock Market Analysis and Prediction System aims to provide users with valuable insights and accurate forecasts to support their investment decisions and trading strategies.

**Machine Learning with TensorFlow:**

1. @app.route('/train\_data/<symbol>')
2. **def** train\_data(symbol):
3. **if** 'email' **in** session:
4. # Fetch x\_train and y\_train from the database
5. cursor **=** db.cursor()
6. cursor.execute(f"SELECT Open\_Price, High\_Price, Low\_Price, Close\_Price FROM `{symbol}`")
8. data **=** cursor.fetchall()
10. cursor.close()
12. # Convert data to suitable Python data types
13. data **=** [(float(row[0]), float(row[1]), float(row[2]), float(row[3])) **for** row **in** data]
15. x\_train **=** np.array([row[:**-**1] **for** row **in** data])  # Excludes last element
16. y\_train **=** np.array([row[3] **for** row **in** data])    # Extracting only the Close\_Price

* This route (/train\_data/<symbol>) retrieves training data (features and labels) from the database for a given stock symbol.
* It queries the database to fetch historical stock price data (Open, High, Low, Close) for the specified symbol.
* The fetched data is then converted into NumPy arrays for further processing.

1. **Model Definition and Training:**@app.route('/train\_data/<symbol>')
2. **def** train\_data(symbol):
3. **if** 'email' **in** session:
4. # Fetch x\_train and y\_train from the database
5. # ... (Previous code)
7. # Define Sequential model with 3 layers
8. model **=** tf.keras.Sequential(
9. [
10. tf.keras.layers.Dense(64, activation**=**'relu', input\_shape**=**(x\_train.shape[1],)),
11. tf.keras.layers.Dense(32, activation**=**'relu'),
12. tf.keras.layers.Dense(1),
13. ]
14. )
15. # Compile the model
16. model.compile(optimizer**=**'adam',
17. loss**=**'mean\_squared\_error',  # Mean squared error loss for regression
18. metrics**=**['mae'])  # Mean absolute error metric
20. # Train the model
21. model.fit(x\_train, y\_train, epochs**=**10, batch\_size**=**32)  # Adjust batch size and number of epochs as needed

* The code defines a Sequential model using Keras, a high-level API of TensorFlow, with three dense layers.
* The model architecture consists of two hidden layers with ReLU activation functions and one output layer.
* The model is compiled with an Adam optimizer, mean squared error loss function, and mean absolute error metric.
* Finally, the model is trained using the training data (features and labels) fetched from the database.

**Model Saving:**

1. @app.route('/train\_data/<symbol>')
2. **def** train\_data(symbol):
3. **if** 'email' **in** session:
4. # Save the trained model
5. model.save(f'models/{symbol}.keras')
6. **return** jsonify({'success': True})

* After training, the trained model is saved to a file in the "models" directory with the symbol name as the filename.

|  |
| --- |
| **Implementation** |

In this section, we outline the implementation details of the Stock Market Analysis and Prediction System, covering environment setup, data acquisition, data preprocessing, algorithm implementation, and user interface development.

**5.1 Environment Setup**

Setting up the development environment involves configuring the necessary software tools, libraries, and dependencies required for system development. Key components of the environment setup include:

* **Programming Languages**: Utilizing languages such as Python, JavaScript, HTML, and CSS for backend, frontend, and web development tasks.
* **Development Frameworks**: Employing frameworks like Flask, Django, and Node.js for backend development, and libraries such as React.js and Vue.js for frontend development.
* **Database Management Systems**: Choosing database systems like MySQL, PostgreSQL, or MongoDB for storing and managing financial data, user information, and system configurations.
* **Development Tools**: Using integrated development environments (IDEs) such as Visual Studio Code, PyCharm, or Sublime Text for coding and debugging.

**5.2 Data Acquisition**

Data acquisition involves retrieving relevant financial data from various sources, including financial APIs, market exchanges, and data vendors. The process includes:

* **API Integration**: Accessing real-time and historical market data through APIs provided by financial data providers such as Alpha Vantage, Yahoo Finance, and Quandl.
* **Web Scraping**: Extracting data from financial websites, news portals, and social media platforms using web scraping tools like BeautifulSoup and Scrapy.
* **Data Feeds**: Subscribing to data feeds and market streams provided by exchanges and financial institutions for up-to-date market information.

**5.3 Data Preprocessing**

Data preprocessing is performed to clean, transform, and prepare raw market data for analysis and modeling. Common preprocessing steps include:

* **Cleaning**: Removing duplicates, missing values, outliers, and irrelevant data points from the dataset.
* **Normalization**: Scaling numerical features to a standard range or normalizing them to improve model convergence.
* **Feature Engineering**: Creating new features or transforming existing ones to capture relevant information and improve predictive power.
* **Dimensionality Reduction**: Reducing the dimensionality of the dataset using techniques like principal component analysis (PCA) or feature selection to enhance computational efficiency.

**5.4 Algorithm Implementation**

Algorithm implementation involves developing and integrating analysis and prediction algorithms into the system. This includes:

* **Technical Analysis Algorithms**: Implementing algorithms to analyze historical price and volume data, calculate technical indicators, and identify trading signals.
* **Machine Learning Models**: Implementing machine learning algorithms such as regression models, decision trees, and neural networks for stock price prediction.
* **Deep Learning Models**: Developing deep learning architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for feature learning and sequential prediction tasks.

**Prediction Using Trained Model:**

1. @app.route('/prediction/<symbol>')
2. **def** prediction(symbol):
3. **if** 'email' **in** session:
4. model\_path **=** f'models\\{symbol}.keras'
6. # Check if model is trained or not
7. **if** **not** os.path.exists(model\_path):
8. **return** jsonify(error**=**'Model not available'), 404
10. # Load the trained model
11. model **=** tf.keras.models.load\_model(model\_path)
13. # Fetch historical data for the stock from the database
14. cursor **=** db.cursor()
15. cursor.execute(f"SELECT Date, Open\_Price, High\_Price, Low\_Price, Close\_Price FROM {symbol} ORDER BY Date DESC Limit 20")
16. historical\_data **=** cursor.fetchall()
17. cursor.close()
19. # Convert decimal.Decimal values to float and format date
20. historical\_data **=** [(row[0].strftime('%Y-%m-%d'), float(row[1]), float(row[2]), float(row[3]), float(row[4])) **for** row **in** historical\_data]
21. historical\_data.sort(key**=lambda** x: datetime.strptime(x[0], '%Y-%m-%d'))
23. # Perform prediction using the trained model
24. predicted\_close\_prices **=** []
25. current\_date **=** datetime.strptime(historical\_data[**-**1][0], '%Y-%m-%d')
26. **for** \_ **in** range(5):
27. x\_input **=** np.array([[historical\_data[**-**1][1], historical\_data[**-**1][2], historical\_data[**-**1][3]]])
28. predicted\_close\_price **=** model.predict(x\_input)
29. predicted\_close\_prices.append(float(predicted\_close\_price[0][0]))
30. current\_date **+=** timedelta(days**=**1)
31. historical\_data.append((current\_date.strftime('%Y-%m-%d'), float(predicted\_close\_price[0][0]), float(predicted\_close\_price[0][0]), float(predicted\_close\_price[0][0]), float(historical\_data[**-**1][4])))
33. # Prepare response data
34. response\_data **=** {
35. "dates": [data\_point[0] **for** data\_point **in** historical\_data],
36. "historical\_prices": [data\_point[4] **for** data\_point **in** historical\_data],
37. "predicted\_prices": [float(price) **for** price **in** predicted\_close\_prices]
38. }
39. **return** jsonify(response\_data)

* This route (/prediction/<symbol>) is responsible for making predictions for the future stock prices of the specified symbol using the trained model.
* It first checks if the trained model file exists for the given symbol.
* If the model exists, it loads the model from the saved file.
* It then fetches historical stock price data for the specified symbol from the database.
* Using the loaded model, it predicts the future stock prices for the next 5 days based on the historical data.
* Finally, it prepares and returns the prediction results in JSON format.

**5.5 User Interface Development**

User interface development focuses on creating an intuitive and interactive interface for users to interact with the system. This involves:

* **Frontend Design**: Designing responsive and visually appealing user interfaces using HTML, CSS, and JavaScript frameworks like React.js or Vue.js.
* **Data Visualization**: Integrating data visualization libraries such as D3.js, Plotly, or Highcharts to visualize market data, trends, and prediction results.
* **User Interaction**: Implementing user-friendly features such as search filters, interactive charts, and customizable dashboards to enhance the user experience.

**Web Server Setup:**

The web server setup is a crucial aspect of deploying the Flask application. It ensures that the application is accessible over the internet and handles incoming requests efficiently. Below, I'll explain the setup and configuration of the web server:

**Flask Development Server:**

1. **if** \_\_name\_\_ **==** '\_\_main\_\_':
2. app.run(host**=**'0.0.0.0', port**=**5000, debug**=**True)

* The if \_\_name\_\_ == '\_\_main\_\_': block ensures that the Flask development server is started only when the script is executed directly.
* app.run() method is used to run the Flask application. Here:
* host='0.0.0.0' makes the server publicly accessible, allowing requests from any IP address.
* port=5000 specifies the port number on which the Flask server listens for incoming HTTP requests.
* debug=True enables debug mode, providing detailed error messages and auto-restarting the server on code changes during development.

**Deployment on Production Server:**

For deploying the Flask application on a production server, using a dedicated web server like Nginx or Apache in conjunction with a WSGI server (such as Gunicorn or uWSGI) is recommended. Here's a typical setup:

* Nginx as Reverse Proxy:
  + Nginx serves as a reverse proxy, forwarding incoming HTTP requests to the WSGI server.
  + It also handles static file serving and SSL termination.
  + Configuration involves setting up server blocks (virtual hosts) in Nginx configuration files (nginx.conf or separate files in /etc/nginx/sites-available/) to define how requests are processed.
  + Example server block for Flask application:

1. server {
2. listen 80;
3. server\_name example.com www.example.com;
5. location **/** {
6. include proxy\_params;
7. proxy\_pass http:**//**localhost:8000;  # Forward requests to Gunicorn
8. }
10. location **/**static {
11. alias **/**path**/**to**/**static**/**files;
12. }
14. location **/**media {
15. alias **/**path**/**to**/**media**/**files;
16. }
18. # Additional configuration for SSL termination, caching, etc.
19. }

* **WSGI Server (Gunicorn):**
  + Gunicorn serves as the WSGI server, responsible for running the Flask application.
  + It manages multiple worker processes to handle concurrent requests efficiently.

**Example command to start Gunicorn:**

gunicorn -w 4 -b localhost:8000 wsgi:app

* -w 4: Number of worker processes.
* -b localhost:8000: Bind address and port for the server.
* wsgi:app: Entry point for the Flask application (wsgi is the Python module containing the Flask app object app).

**HTML Formatting**

**Meta Tags and Title**

* Meta tags provide metadata about the HTML document, such as character set, viewport settings, keywords, and description.
* The <title> tag sets the title of the web page.

**External Resources**

* Favicon: Specifies the icon displayed in the browser tab.
* Google Web Fonts: Imports the "Heebo" font family from Google Fonts.
* Icon Font Stylesheets: Links to Font Awesome and Bootstrap Icons for iconography.
* Libraries Stylesheet: Links to CSS files for additional libraries like Owl Carousel and Tempus Dominus.
* Customized Bootstrap Stylesheet: Links to a customized Bootstrap stylesheet.
* Template Stylesheet: Links to the custom CSS file for styling the template.

**JavaScript Libraries**

* External JavaScript libraries like jQuery, Bootstrap, Chart.js, Easing, Waypoints, Owl Carousel, and Tempus Dominus are included.
* Template-specific JavaScript files (main.js and functions.js) are included for additional functionality.

**Body Structure**

* The body of the document starts with a fixed header containing a ticker tape for displaying live stock information.
* The ticker tape scrolls horizontally and pauses when the mouse hovers over it.
* The main content area is divided into a sidebar and a content section.
* The sidebar contains navigation links, user profile information, and dropdown menus.
* The content section includes interactive elements such as dropdowns, buttons, charts, and tables.

**Functionalities**

* Dropdown menu for selecting a stock symbol.
* Button for retrieving data related to the selected symbol.
* Display of predicted chart and historical data chart using Chart.js.
* Display of top gainers and top losers tables.
* Footer section with copyright and attribution information.

**Custom Scripts**

* Custom JavaScript functions (pauseTicker() and resumeTicker()) are defined for controlling the behavior of the ticker tape.
* Additional custom JavaScript functions may be defined in the functions.js file for handling specific interactions and data manipulation.

**JSP Formation:**

This JavaScript code snippet comprises several functions and event listeners that collectively contribute to the functionality of a web application focused on stock market analysis, prediction, and data visualization. Let's break down each function and event listener in detail:

**showProgressModal() and hideProgressModal():**

* These functions show and hide a modal dialog box with a progress indicator, respectively, using jQuery and Bootstrap modal functionalities.

**$(document).ready(function() { ... }):**

* This event listener waits for the document to be fully loaded and then attaches an event handler to the form submission action. When the form is submitted, it prevents the default form submission behavior, shows the progress modal, and sends an AJAX POST request to the server with the form data.
* Depending on the response from the server, it updates the modal content to indicate success or failure.

**portfolio():**

* This function triggers when the user accesses their portfolio. It shows the progress modal, sends a POST request to fetch portfolio data from the server, and updates the modal content based on the success or failure of the request.

**pauseTicker() and resumeTicker():**

* These functions control the pausing and resuming of a ticker tape displaying stock information. However, the implementation of these functions seems incomplete as they refer to an undefined variable ticker.

**fetchData():**

* This function is called when the DOM is loaded to fetch stock symbols asynchronously. It sends a GET request to /get\_symbols endpoint and updates a dropdown menu with the retrieved symbols.

**retrieveData(), updateData(), TrainData(), predict(), and chart():**

* These functions handle various aspects of fetching, updating, training, predicting, and visualizing stock data. They make AJAX requests to different endpoints on the server, process the responses, and update the UI accordingly. For example, predict() fetches prediction data for a selected stock symbol, while chart() dynamically generates a line chart based on the retrieved data.

**formatDate():**

* This function formats a given date string into a custom format (dd-MMM-yyyy) using JavaScript's Date object.

**fetchDataWithDelay() and fetchData():**

* These functions are responsible for fetching data for both gainers and losers in the stock market. They make asynchronous requests to /trending/gainers and /trending/loosers endpoints, respectively, and update corresponding tables in the UI with the retrieved data.

**window.addEventListener('DOMContentLoaded', function() { ... }):**

* This event listener executes a set of functions when the DOM content is fully loaded. It initiates the initial data fetching and sets up intervals for refreshing data periodically.

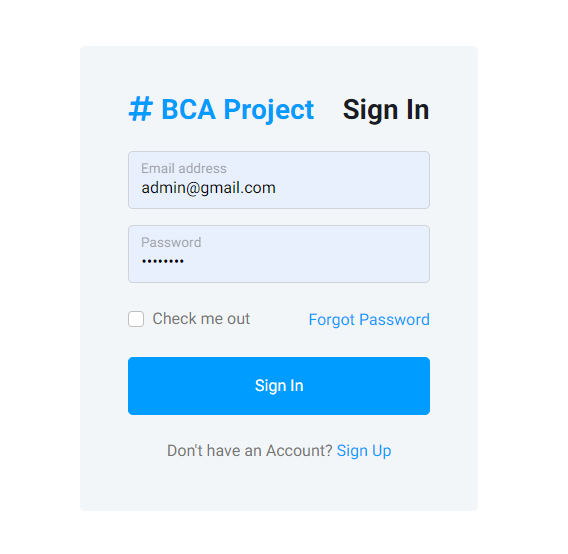
**Ticker Tape Functionality:**

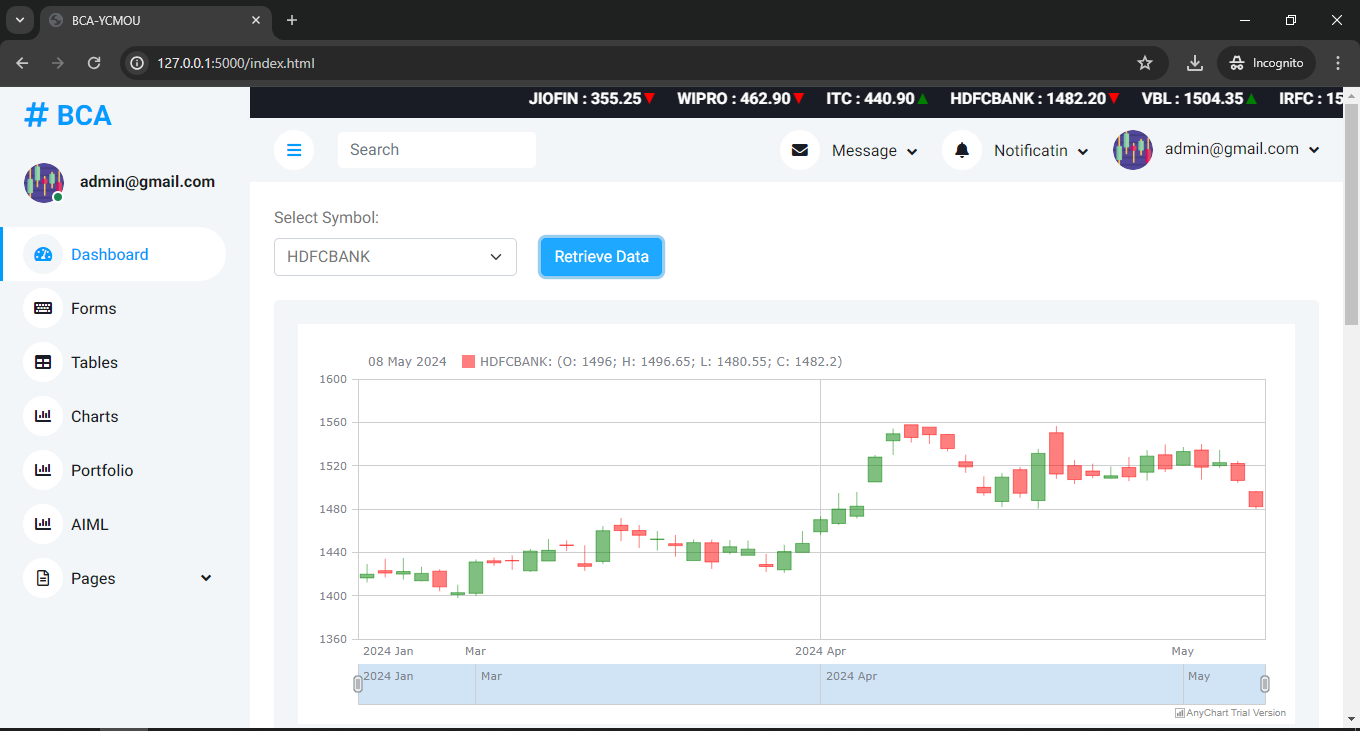
This section includes code for fetching live stock data and updating a ticker tape element with the latest stock information. It periodically retrieves data from the server and updates the ticker tape accordingly.

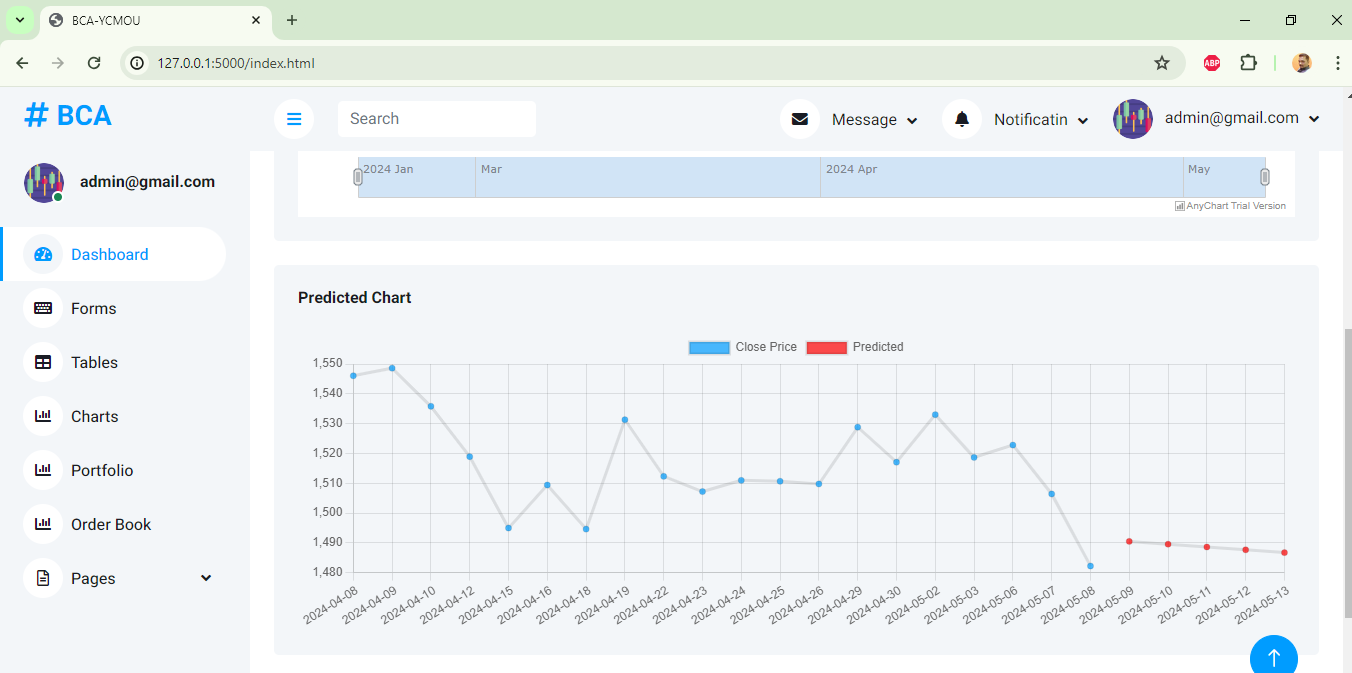
1. // Ticker Tape
2. window.addEventListener('DOMContentLoaded', **function**() {
3. // Function to fetch data with delay
4. **function** fetchDataWithDelay() {
5. setTimeout(fetchData, 2000); // Add a delay of 2000 milliseconds (2 seconds)
6. }
7. // Function to fetch data
8. **function** fetchData() {
9. fetch("/ticker")
10. .then(response => {
11. **if** (!response.ok) {
12. **throw** **new** Error('Network response was not ok');
13. }
14. **return** response.json();
15. })
16. .then(data => {
17. const marqueeElement = document.getElementById("ticker-marquee");
18. marqueeElement.innerHTML = ""; // Clear previous data
20. **if** (!data.success) {
21. **throw** **new** Error('Request was not successful');
22. }
24. **if** (!Array.isArray(data.stock\_data)) {
25. **throw** **new** Error('Stock data is not an array');
26. }
27. data.stock\_data.forEach(stock => {
28. const symbolSpan = document.createElement("span");
29. symbolSpan.classList.add("symbol");
30. symbolSpan.textContent = `${stock.Name.toUpperCase()} : ${stock.LTP}`;
32. const trendSpan = document.createElement("span");
33. **if** (parseFloat(stock.LTP) > parseFloat(stock.Previous\_Close)) {
34. trendSpan.style.color = "green";
35. trendSpan.textContent = "▲";
36. } **else** **if** (parseFloat(stock.LTP) < parseFloat(stock.Previous\_Close)) {
37. trendSpan.style.color = "red";
38. trendSpan.textContent = "▼";
39. } **else** {
40. trendSpan.textContent = "↔";
41. }
43. symbolSpan.appendChild(trendSpan);
44. marqueeElement.appendChild(symbolSpan);
45. });
46. })
47. .**catch**(error => {
48. console.error("Error fetching data:", error);
49. });
50. }
52. // Initial fetch with delay
53. fetchDataWithDelay();
55. // Refresh data every minute
56. setInterval(fetchData, 60000);
57. });

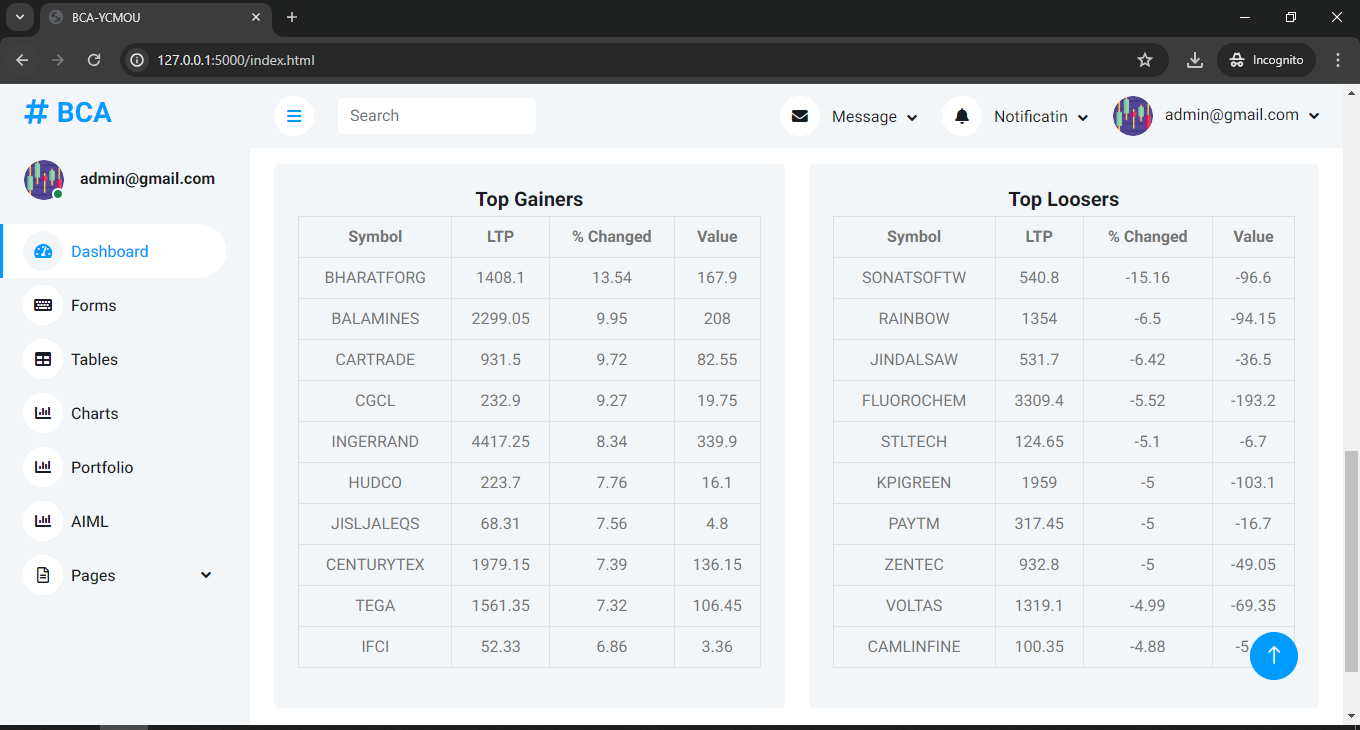
**Deployment Considerations:**

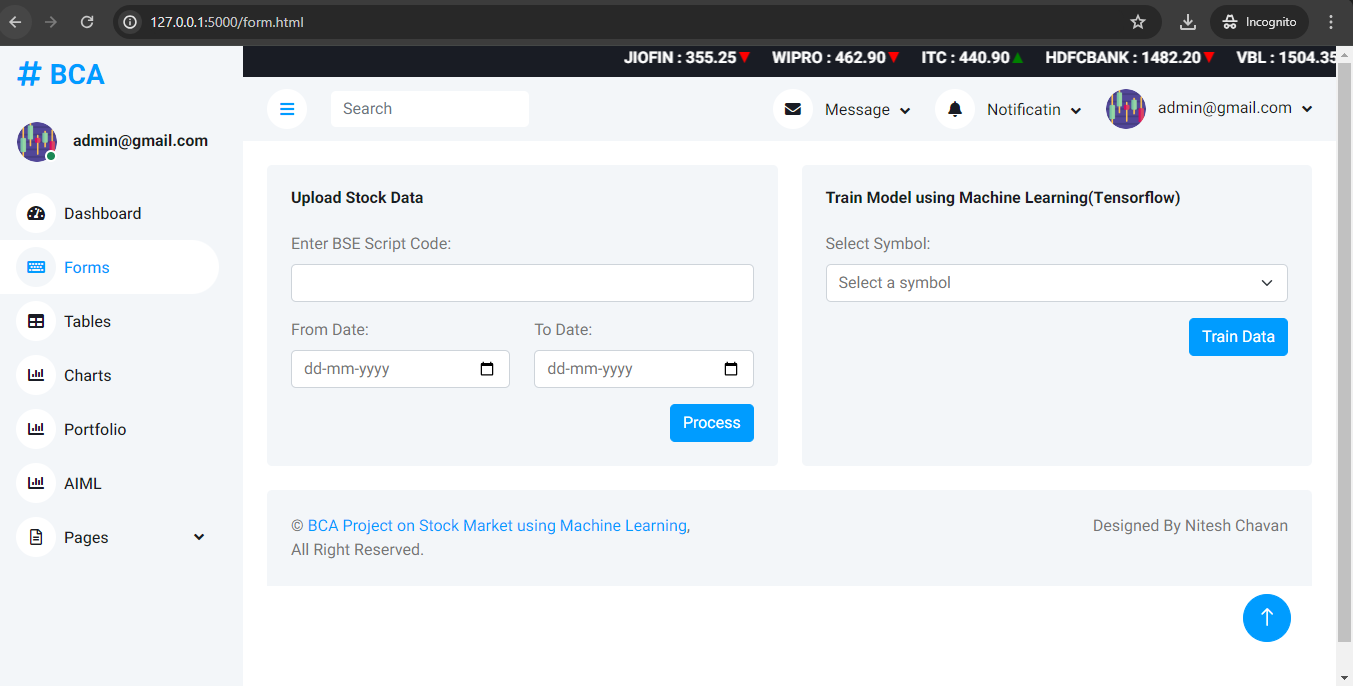
* Security: Ensure secure communication over HTTPS by configuring SSL/TLS certificates.
* Performance: Optimize server configuration and application code for performance.
* Monitoring: Implement logging and monitoring solutions to track server performance and diagnose issues.
* Scalability: Design the deployment architecture to scale horizontally or vertically based on demand.
* Backup and Recovery: Regularly back up data and implement disaster recovery strategies.

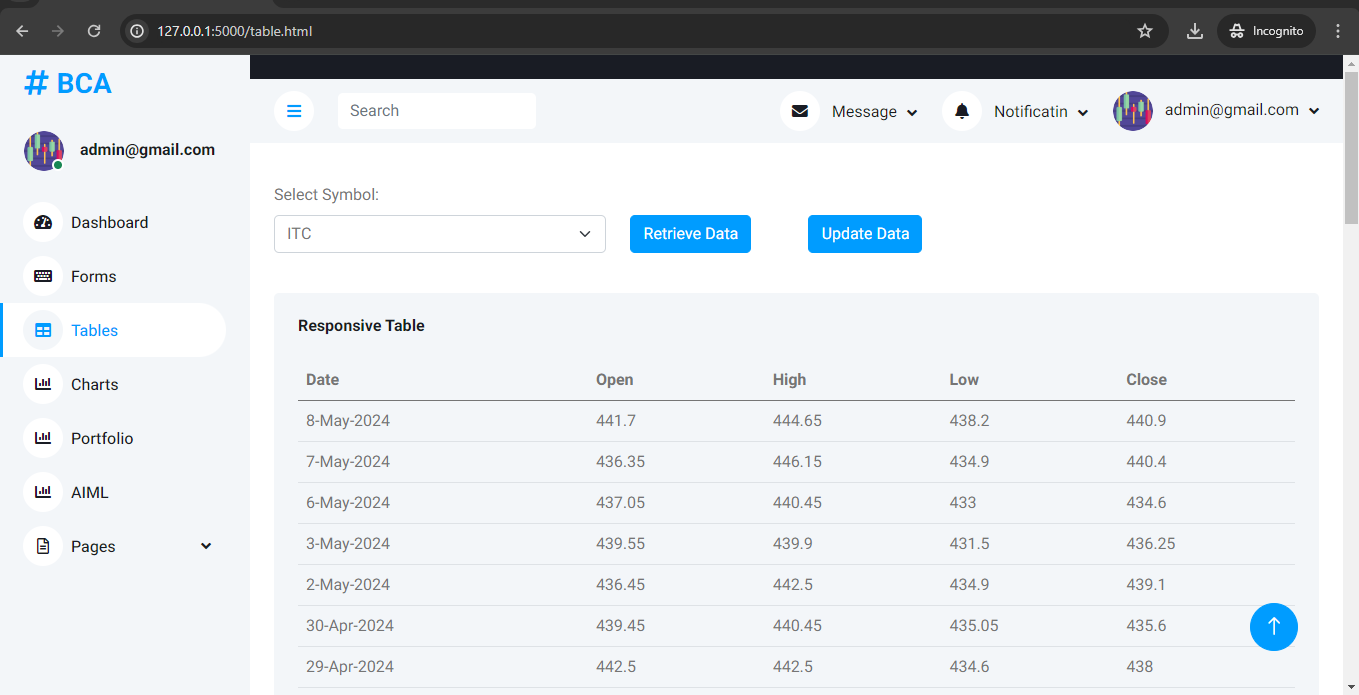
**Login Screen**

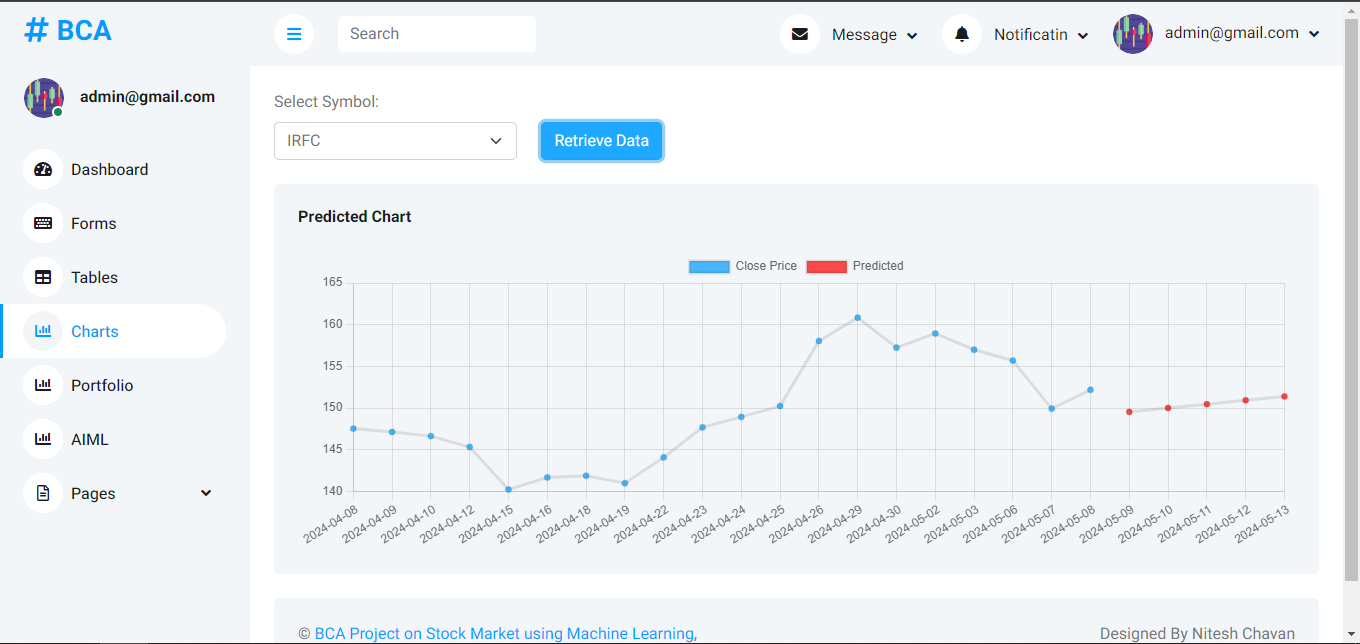
**Dashboard Screen**

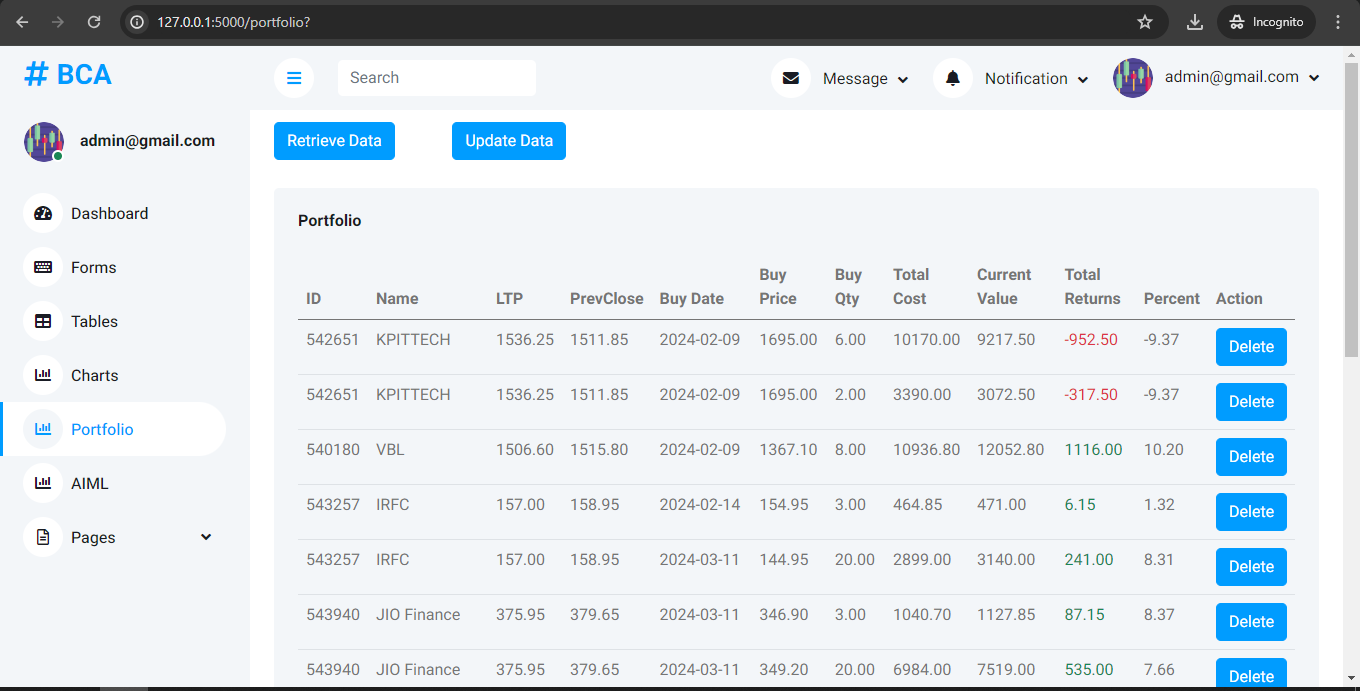


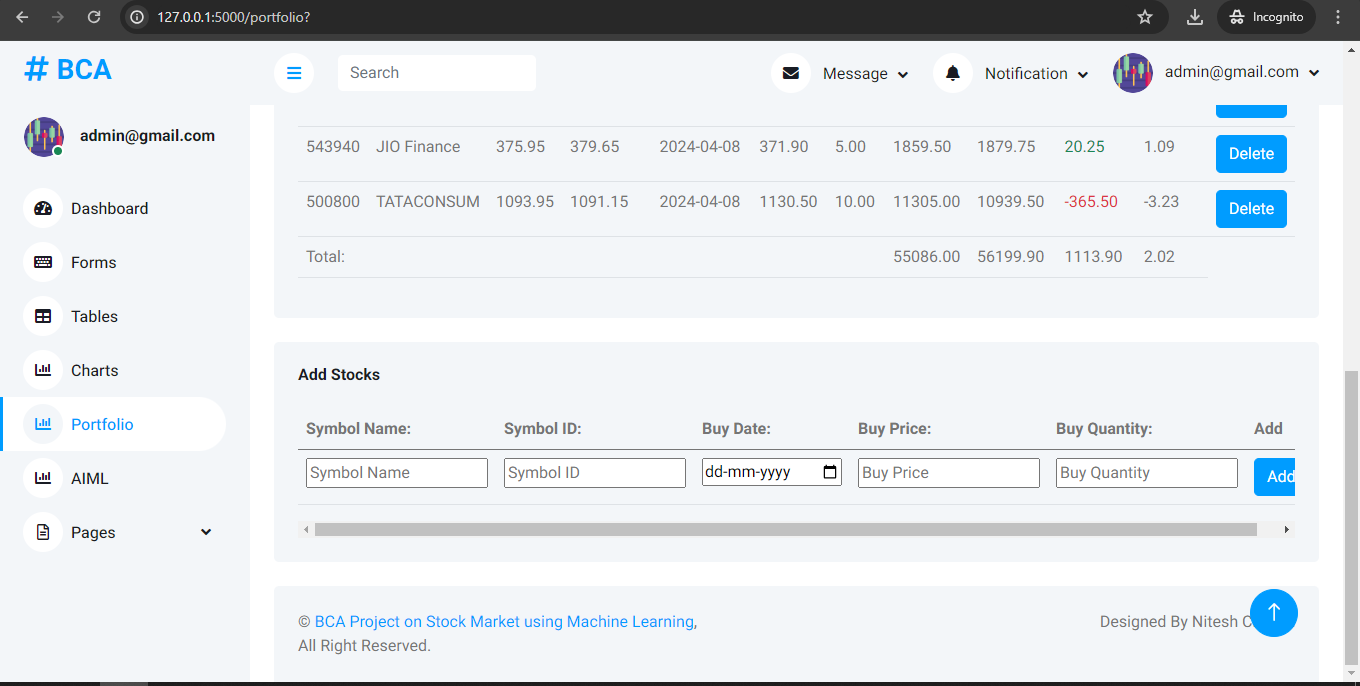


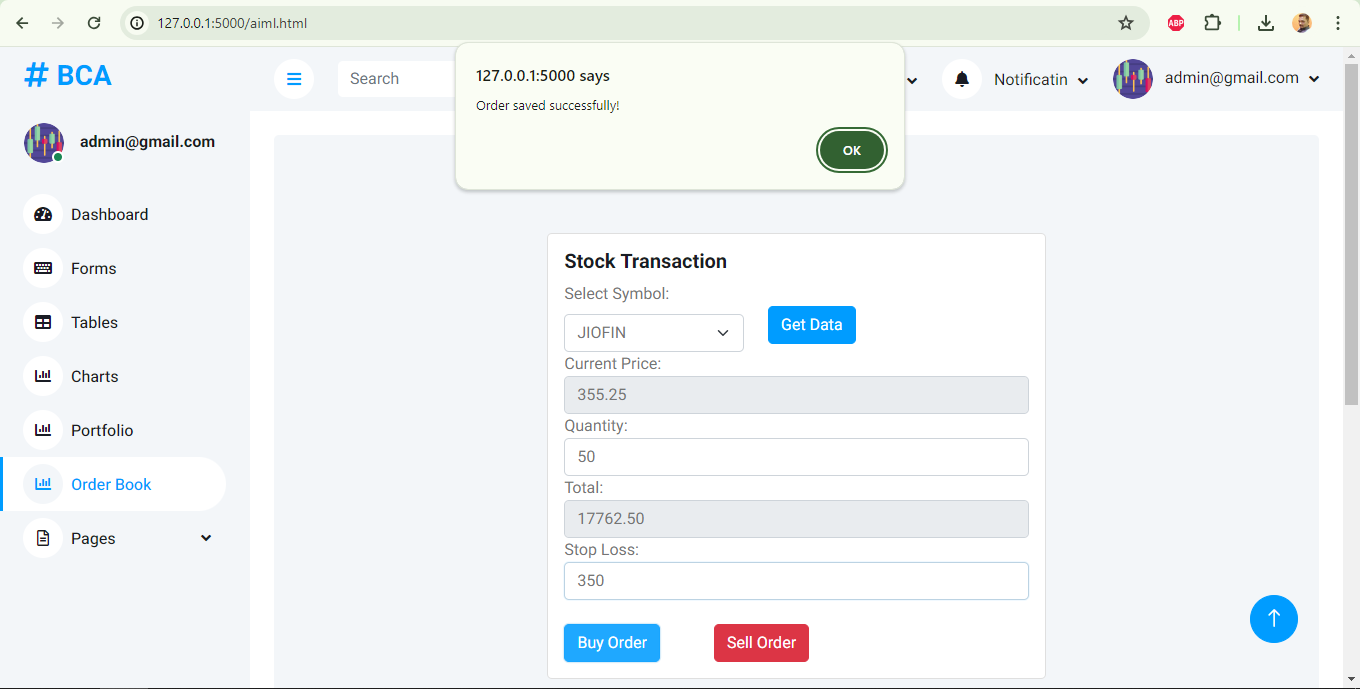


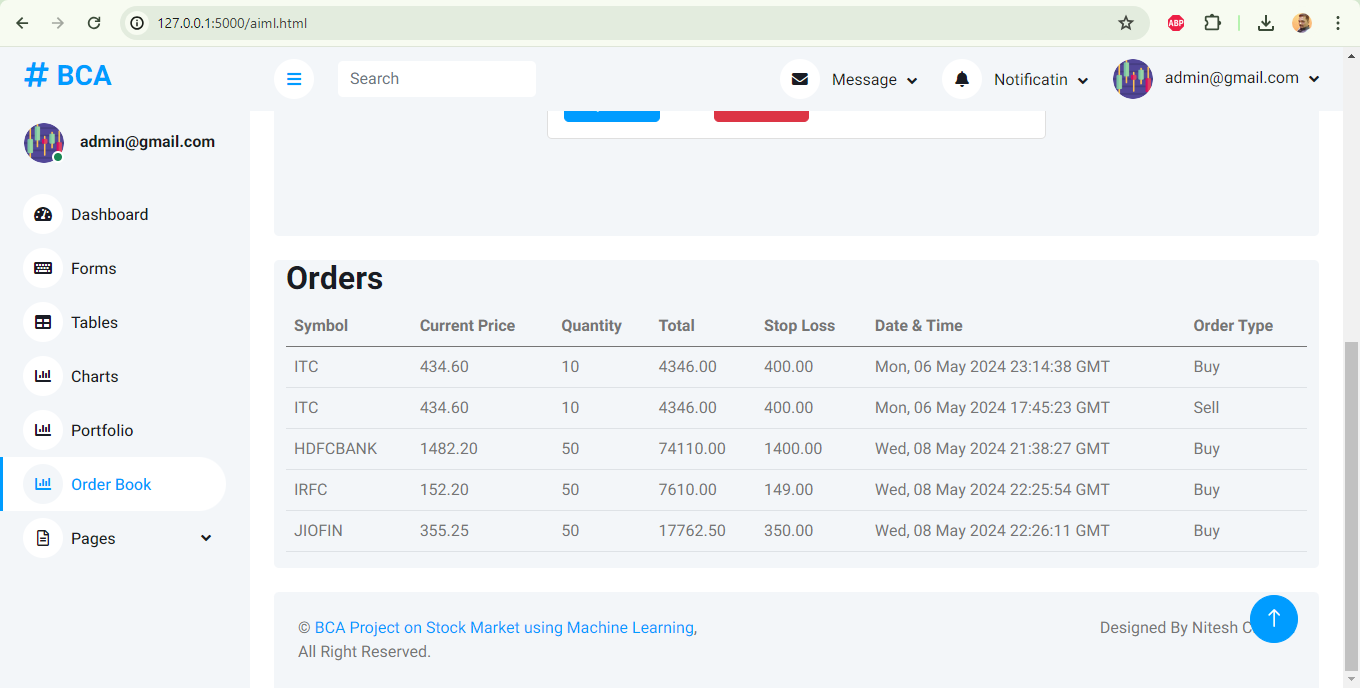












1. **import** requests, csv, mysql.connector
2. **import** os
3. **import** numpy as np
4. **import** tensorflow as tf

7. **from** flask **import** Flask, request, render\_template, redirect, url\_for, session, send\_from\_directory, jsonify
8. **from** werkzeug.security **import** generate\_password\_hash, check\_password\_hash
9. **from** io **import** StringIO
10. **from** datetime **import** datetime, timedelta
11. **from** decimal **import** Decimal


15. os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] **=** '3'  # Suppress TensorFlow messages
16. #0: Display all logs (default behavior)
17. #1: Display only info logs
18. #2: Display only warning and error logs
19. #3: Display only error logs


23. app **=** Flask(\_\_name\_\_)
24. app.secret\_key **=** os.urandom(24)


28. db **=** mysql.connector.connect(
29. host**=**"localhost",
30. user**=**"nitesh",
31. password**=**"root@123",
32. database**=**"stocks"
33. )
35. @app.route('/favicon.ico')
36. **def** favicon():
37. **return** app.send\_static\_file('favicon.ico')
39. @app.route('/img/<path:filename>')
40. **def** get\_image(filename):
41. **return** send\_from\_directory('templates/img', filename)
43. @app.route('/lib/<path:filename>')
44. **def** get\_lib(filename):
45. **return** send\_from\_directory('templates/lib', filename)
47. @app.route('/css/<path:filename>')
48. **def** get\_css(filename):
49. **return** send\_from\_directory('templates/css', filename)
51. @app.route('/js/<path:filename>')
52. **def** get\_js(filename):
53. **return** send\_from\_directory('templates/js', filename)


57. @app.route('/')
58. **def** index():
59. email **=** session.get('email')
60. **if** 'email' **in** session:
62. **return** render\_template('index.html', email**=**email)
63. **return** redirect(url\_for('signin'))

66. # Route for rendering HTML pages
67. @app.route('/<page>')
68. **def** render\_page(page):
69. **if** page **==** 'signup.html':
70. **return** render\_template(f'{page}')
71. **if** 'email' **in** session:
72. email **=** session['email']
73. **return** render\_template(f'{page}', email**=**email)
74. **return** redirect(url\_for('signin'))



79. # @app.route('/signup', methods=['GET', 'POST'])
80. # def signup():
81. # print('in sighup.html')
83. # if request.method == 'POST':
84. # email = request.form['email']
85. # password = request.form['password']
86. # # Hash the password
87. # hashed\_password = generate\_password\_hash(password)
88. # cursor = db.cursor()
89. # # Insert username and hashed password into the database
90. # cursor.execute("INSERT INTO logins (email, password) VALUES (%s, %s)", (email, hashed\_password))
91. # db.commit()
92. # cursor.close()
93. # return redirect(url\_for('index'))
95. @app.route('/signup', methods**=**['GET', 'POST'])
96. **def** signup():
97. print('in signup.html')
99. **if** request.method **==** 'POST':
100. email **=** request.form['email']
101. password **=** request.form['password']
102. # Hash the password
103. hashed\_password **=** generate\_password\_hash(password)
104. cursor **=** db.cursor()
105. # Check if the email already exists in the database
106. cursor.execute("SELECT id FROM logins WHERE email = %s", (email,))
107. existing\_user **=** cursor.fetchone()
108. **if** existing\_user:
109. # If the email exists, return an error message
110. **return** render\_template('signup.html', error**=**"Email already exists. Please choose a different email.")
111. **else**:
112. # Insert username and hashed password into the database
113. cursor.execute("INSERT INTO logins (email, password) VALUES (%s, %s)", (email, hashed\_password))
114. db.commit()
115. cursor.close()
116. **return** redirect(url\_for('index'))
117. **return** render\_template('signup.html')
119. @app.route('/signin', methods**=**['GET', 'POST'])
120. **def** signin():
121. **if** request.method **==** 'POST':
122. #form\_data = request.form
124. #for key, value in form\_data.items():
125. #    print(f"Key: {key}, Value: {value}")
127. email **=** request.form['email']
128. password **=** request.form['password']
129. #print(email,password)
130. # Query the database to retrieve hashed password
131. cursor **=** db.cursor()
132. cursor.execute("SELECT password FROM logins WHERE email = %s", (email,))
133. result **=** cursor.fetchone()
134. cursor.close()
135. **if** result:
136. stored\_password **=** result[0]
137. **if** check\_password\_hash(stored\_password, password):
138. session['email'] **=** email
139. **return** redirect(url\_for('index'))
141. **return** 'Invalid username/password'
143. **return** render\_template('signin.html')
145. @app.route('/logout')
146. **def** logout():
147. session.pop('email', None)
148. **return** redirect(url\_for('signin'))


152. #fetch symbols in dropdown list
153. @app.route('/get\_symbols')
154. **def** get\_symbols():
155. **if** 'email' **in** session:
156. **try**:
157. # Query to retrieve distinct symbols from the stocks\_price table
158. cursor **=** db.cursor()
159. cursor.execute("show tables")
161. tables **=** [row[0] **for** row **in** cursor.fetchall() **if** row[0] **not** **in** ('logins', 'portfolio', 'order\_table')] # Filter out 'logins' table
162. cursor.close()
163. **return** jsonify(tables)
164. **except** Exception as e:
165. **return** jsonify(error**=**str(e)), 500
166. **return** redirect(url\_for('signin'))
168. @app.route('/get\_data/<symbol>')
169. **def** get\_data(symbol):
170. **if** 'email' **in** session:
171. **try**:
172. # Check if the table exists
173. cursor **=** db.cursor()
174. cursor.execute("SHOW TABLES LIKE %s", (symbol,))
175. table\_exists **=** cursor.fetchone()
176. **if** **not** table\_exists:
177. **return** jsonify(error**=**f"Table '{symbol}' does not exist."), 404
179. # Query the database for data related to the given symbol, ordered by Date in descending order
180. query **=** f"SELECT Date, Open\_Price, High\_Price, Low\_Price, Close\_Price FROM {symbol} ORDER BY Date DESC LIMIT 50"
182. cursor.execute(query)
183. data **=** cursor.fetchall()
184. cursor.close()
185. # Convert the data to a list of dictionaries and reorder the keys
186. result **=** []
187. **for** row **in** data:
188. # Convert the date to a string before parsing
189. result.append({
190. 'Date': row[0],  # Ensure Date is placed last
191. 'Open': row[1],
192. 'High': row[2],
193. 'Low': row[3],
194. 'Close': row[4]
195. })
197. # Return the data as JSON response
198. **return** jsonify(result)
199. **except** Exception as e:
200. **return** jsonify(error**=**str(e)), 500
201. **return** redirect(url\_for('signin'))

204. @app.route('/update\_data/<symbol>')
205. **def** update\_data(symbol):
206. **if** 'email' **in** session:
208. cursor **=** db.cursor()
209. cursor.execute(f"SELECT Symbol\_id FROM `{symbol}`Limit 1")
210. data **=** cursor.fetchone()
212. symbol\_id **=** data[0]
213. print(symbol\_id)
214. url **=** "https://api.bseindia.com/BseIndiaAPI/api/StockPriceCSVDownload/w"
215. headers **=** {
216. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
217. "Referer": "https://www.bseindia.com/",
218. }
220. #01/02/2024 02/02/2024
221. fromDate **=** '01/02/2024'
222. #toDate = datetime.strptime(request.form['toDate'], '%Y-%m-%d').strftime('%d/%m/%Y')
223. toDate **=** datetime.now().strftime('%d/%m/%Y')
224. payload **=** {
225. "pageType": "0",
226. "rbType": "D",
227. "Scode": symbol\_id,
228. "FDates": fromDate,
229. "TDates": toDate
230. }
232. **try**:
233. response **=** requests.get(url, headers**=**headers, params**=**payload, timeout**=**5)
234. **except** requests.exceptions.Timeout:
235. **return** jsonify({"error": "Request timed out. Please try again later."}), 504
236. **if** response.status\_code **==** 200:
237. csv\_buffer **=** StringIO(response.text)
238. csv\_reader **=** csv.DictReader(csv\_buffer, delimiter**=**'\t')
239. **for** row **in** csv\_reader:
240. data **=** list(row.values())[0].split(',')
241. date\_str **=** data[0]
242. date\_obj **=** datetime.strptime(date\_str, "%d-%B-%Y")
243. mysql\_date\_format **=** date\_obj.strftime("%Y-%m-%d")
244. open\_price **=** data[1]
245. high\_price **=** data[2]
246. low\_price **=** data[3]
247. close\_price **=** data[4]
248. print(open\_price, high\_price, low\_price, close\_price)
249. # Inserting data into respective table
250. cursor.execute(
251. f"INSERT IGNORE INTO {symbol} (Symbol\_id, Date, Open\_Price, High\_Price, Low\_Price, Close\_Price) VALUES (%s, %s, %s, %s, %s, %s)",
252. (symbol\_id, mysql\_date\_format, open\_price, high\_price, low\_price, close\_price))
253. db.commit()
254. cursor.close()
255. **return** jsonify({'success': True})
256. **else**:
257. **return** jsonify({"error": "Failed to fetch data"}), response.status\_code
258. **return** redirect(url\_for('signin'))

261. # Update Stocks Data from BSE API
262. @app.route('/process\_script\_data', methods**=**['POST'])
263. **def** process\_script\_data():
264. **if** 'email' **in** session:
265. **if** request.method **==** 'POST':
266. stock\_data **=** request.form['stockData']
267. fromDate **=** datetime.strptime(request.form['fromDate'], '%Y-%m-%d').strftime('%d/%m/%Y')
268. toDate **=** datetime.strptime(request.form['toDate'], '%Y-%m-%d').strftime('%d/%m/%Y')
269. print("Received stock data:", stock\_data, fromDate, toDate)
270. url **=** "https://api.bseindia.com/BseIndiaAPI/api/StockPriceCSVDownload/w"
271. url2 **=** "https://api.bseindia.com/BseIndiaAPI/api/ComHeadernew/w"
272. headers **=** {
273. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
274. "Referer": "https://www.bseindia.com/",
275. }
277. # Extract script code from query parameters
278. script\_code **=** stock\_data  # request.args.get('script\_code')
280. # for Stock Data
281. payload **=** {
282. "pageType": "0",
283. "rbType": "D",
284. "Scode": script\_code,
285. "FDates": fromDate,
286. "TDates": toDate
287. }
288. # for Stock Name
289. payload2 **=** {
290. "quotetype": 'EQ',
291. "scripcode": script\_code,
292. "seriesid": ""
293. }
295. **try**:
296. response **=** requests.get(url, headers**=**headers, params**=**payload, timeout**=**5)
297. response.raise\_for\_status()
298. response2 **=** requests.get(url2, headers**=**headers, params**=**payload2, timeout**=**5).json()
299. **except** requests.exceptions.Timeout:
300. **return** jsonify({"error": "Request timed out. Please try again later."}), 504
301. **except** requests.exceptions.RequestException as e:
302. **return** jsonify({"error": f"An error occurred: {e}"}), 500
304. symbol\_name **=** response2.get("SecurityId", "")
306. **if** response.status\_code **==** 200:
307. cursor **=** db.cursor()
308. # Dynamically create table for each symbol if not exists
309. cursor.execute(f"CREATE TABLE IF NOT EXISTS {symbol\_name} (Symbol\_id int, Date DATE, Open\_Price FLOAT, High\_Price FLOAT, Low\_Price FLOAT, Close\_Price FLOAT, UNIQUE (Date))")
310. # Process CSV data line by line
311. csv\_buffer **=** StringIO(response.text)
312. csv\_reader **=** csv.DictReader(csv\_buffer, delimiter**=**'\t')
314. **for** row **in** csv\_reader:
316. # Extracting data from each row
317. data **=** list(row.values())[0].split(',')
319. date\_str **=** data[0]
320. date\_obj **=** datetime.strptime(date\_str, "%d-%B-%Y")
321. mysql\_date\_format **=** date\_obj.strftime("%Y-%m-%d")
322. open\_price **=** data[1]
323. high\_price **=** data[2]
324. low\_price **=** data[3]
325. close\_price **=** data[4]
327. # Inserting data into respective table
328. cursor.execute(
329. f"INSERT IGNORE INTO {symbol\_name} (Symbol\_id, Date, Open\_Price, High\_Price, Low\_Price, Close\_Price) VALUES (%s, %s, %s, %s, %s, %s)",
330. (script\_code, mysql\_date\_format, open\_price, high\_price, low\_price, close\_price))
331. db.commit()
332. cursor.close()
333. **return** jsonify({'success': True})
334. **else**:
335. **return** jsonify({"error": "Failed to fetch data"}), response.status\_code
336. **return** redirect(url\_for('signin'))




342. @app.route('/ticker')
343. **def** ticker\_data():
344. **if** 'email' **in** session:
345. url **=** "https://api.bseindia.com/BseIndiaAPI/api/getScripHeaderData/w"
346. headers **=** {
347. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
348. "Referer": "https://www.bseindia.com/",
349. }
351. cursor **=** db.cursor()
353. **try**:
354. # Get distinct symbol\_ids from all tables except 'logins'
355. cursor.execute("SHOW TABLES")
356. tables **=** [row[0] **for** row **in** cursor.fetchall() **if** row[0] **not** **in** ('logins', 'portfolio', 'order\_table')]
358. symbol\_ids\_set **=** set()  # Using a set to store unique symbol IDs
359. #print(latest\_data)
360. #wait = input("Press Enter to continue.")
361. **for** table **in** tables:
362. cursor.execute(f"SELECT DISTINCT Symbol\_id FROM {table}")
363. distinct\_symbol\_ids **=** cursor.fetchall()
364. symbol\_ids\_set.update(symbol\_id[0] **for** symbol\_id **in** distinct\_symbol\_ids)
366. # Convert set to list for easier iteration
367. symbol\_ids\_list **=** list(symbol\_ids\_set)
369. # Fetch stock data for all symbol IDs in a single API call
370. tdata **=** []
372. **for** symbol\_id **in** symbol\_ids\_list:
373. payload **=** {
374. "Debtflag": "",
375. "scripcode": symbol\_id,
376. "seriesid": "",
377. }
379. jsonData **=** requests.get(url, headers**=**headers, params**=**payload).json()
381. stock\_data **=** {}
383. **if** "Cmpname" **in** jsonData:
384. stock\_data["Name"] **=** jsonData["Cmpname"]["ShortN"]
386. **if** "CurrRate" **in** jsonData:
387. stock\_data["LTP"] **=** jsonData["CurrRate"]["LTP"]
389. **if** "Header" **in** jsonData:
390. stock\_data["Previous\_Close"] **=** jsonData["Header"]["PrevClose"]
392. tdata.append(stock\_data)
394. **return** jsonify({'success': True, 'stock\_data': tdata})
396. **except** Exception as e:
397. **return** jsonify({'success': False, 'error': str(e)})
399. **finally**:
400. # Close cursor
402. cursor.close()
403. **return** redirect(url\_for('signin'))

406. # Top Gainers/Loosers
407. @app.route('/trending/<trend\_type>')
408. **def** trending(trend\_type):
409. **if** 'email' **in** session:
410. url **=** "https://api.bseindia.com/BseIndiaAPI/api/HoTurnover/w"
411. headers **=** {
412. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
413. "Referer": "https://www.bseindia.com/",
414. }
416. **if** trend\_type **==** 'gainers':
417. payload **=** {"flag": "G"}
418. **elif** trend\_type **==** 'loosers':
419. payload **=** {"flag": "L"}
420. **else**:
421. **return** jsonify({"error": "Invalid trending type"}), 400
423. response **=** requests.get(url, headers**=**headers, params**=**payload, timeout**=**10)
425. # Check if the request was successful
426. **if** response.status\_code **==** 200:
427. # Convert response to JSON format
428. data **=** response.json()
429. # Initialize a list to store scrip\_ids and corresponding change\_percent
430. scrip\_data **=** []
431. # Iterate over each item in the "Table" key
432. **for** item **in** data.get("Table", []):
433. # Extract the "scrip\_id" and "change\_percent"
434. scrip\_id **=** item.get("scrip\_id")
435. Ltradert **=** item.get("Ltradert")
436. change\_percent **=** item.get("change\_percent")
437. change\_val **=** item.get("change\_val")
438. # Append the scrip\_id and change\_percent to the list
439. scrip\_data.append({"scrip\_id": scrip\_id, "Ltradert": Ltradert, "change\_percent": change\_percent,"change\_val": change\_val})
440. # Return the list of scrip\_ids and change\_percent as JSON
441. **return** jsonify({"scrip\_data": scrip\_data})
442. **else**:
443. # If request failed, return an error message
444. **return** jsonify({"error": "Failed to fetch trending data"}), response.status\_code
445. **return** redirect(url\_for('signin'))

448. @app.route('/aiml/<symbol>')
449. **def** aiml(symbol):
450. **if** 'email' **in** session:
451. cursor **=** db.cursor()
452. cursor.execute(f"SELECT Symbol\_id FROM `{symbol}`Limit 1")
453. data **=** cursor.fetchone()
455. symbol\_id **=** data[0]
456. url**=**"https://api.bseindia.com/BseIndiaAPI/api/getScripHeaderData/w"
458. headers **=** {
459. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
460. "Referer": "https://www.bseindia.com/",
461. }
462. # for Stock Data
463. payload **=** {
464. "Debtflag": "",
465. "scripcode": symbol\_id,
466. }
467. response **=** requests.get(url, headers**=**headers, params**=**payload, timeout**=**10)
468. data **=** response.json()
469. ltp **=** data['CurrRate']['LTP']
470. cursor.close()
471. **return** jsonify({'LTP': ltp})
472. **return** redirect(url\_for('signin'))

475. @app.route('/saveOrder', methods**=**['POST'])
476. **def** saveorder():
477. **if** 'email' **in** session:
478. **if** request.method **==** 'POST':
479. orderdata **=** request.json  # Use request.json to get JSON data
480. print(orderdata)
481. **try**:
482. # Connect to MySQL database
483. cursor **=** db.cursor()
485. # Insert order data into order\_table
486. query **=** "INSERT INTO order\_table (symbol, currentPrice, quantity, total, stopLoss, dateTime, orderType) VALUES (%s, %s, %s, %s, %s, %s, %s)"
487. values **=** (orderdata['symbol'], orderdata['currentPrice'], orderdata['quantity'], orderdata['total'], orderdata['stopLoss'], orderdata['dateTime'], orderdata['orderType'])
488. cursor.execute(query, values)
489. db.commit()
490. cursor.close()
491. **return** jsonify({'Success': "Success"})
492. **except** Exception as e:
493. print("Error saving order:", e)
494. **return** jsonify({'Error': "Failed to save order"})
495. **return** redirect(url\_for('signin'))
497. @app.route('/orders')
498. **def** orders():
499. **if** 'email' **in** session:
500. cursor **=** db.cursor(dictionary**=**True)
501. cursor.execute("SELECT \* FROM order\_table")
502. orders **=** cursor.fetchall()
503. cursor.close()
504. **return** jsonify({'orders': orders})
505. **return** redirect(url\_for('signin'))


509. @app.route('/train\_data/<symbol>')
510. **def** train\_data(symbol):
511. **if** 'email' **in** session:
512. # Fetch x\_train and y\_train from the database
513. cursor **=** db.cursor()
514. cursor.execute(f"SELECT Open\_Price, High\_Price, Low\_Price, Close\_Price FROM `{symbol}`")
516. data **=** cursor.fetchall()
518. cursor.close()

521. # Convert data to suitable Python data types
523. data **=** [(float(row[0]), float(row[1]), float(row[2]), float(row[3])) **for** row **in** data]
525. x\_train **=** np.array([row[:**-**1] **for** row **in** data])  # Exludes last element ie Close\_Price from the data
527. y\_train **=** np.array([row[3] **for** row **in** data])    # Extracting only the Close\_Price, 4th object

530. # Define Sequential model with 3 layers
531. model **=** tf.keras.Sequential(
532. [
533. tf.keras.layers.Dense(64, activation**=**'relu', input\_shape**=**(x\_train.shape[1],)),
534. tf.keras.layers.Dense(32, activation**=**'relu'),
535. tf.keras.layers.Dense(1),
536. ]
537. )
538. # Call model on the sample\_data for prediction
539. # Compile the model
540. model.compile(optimizer**=**'adam',
541. loss**=**'mean\_squared\_error',  # Mean squared error loss for regression
542. metrics**=**['mae'])  # Mean absolute error metric

545. model.fit(x\_train, y\_train, epochs**=**10, batch\_size**=**32)  # Adjust batch size as needed
547. model.save(f'models/{symbol}.keras')
548. **return** jsonify({'success': True})
549. **return** redirect(url\_for('signin'))
551. @app.route('/prediction/<symbol>')
552. **def** prediction(symbol):
553. **if** 'email' **in** session:
554. model\_path **=** f'models\\{symbol}.keras'
556. #Check if model is trained or not
557. **if** **not** os.path.exists(model\_path):
558. **return** jsonify(error**=**'Model not available'), 404
560. cursor **=** db.cursor()
561. # Fetch historical data for the stock
562. cursor.execute(f"SELECT Date, Open\_Price, High\_Price, Low\_Price, Close\_Price FROM {symbol} ORDER BY Date DESC Limit 20")
563. historical\_data **=** cursor.fetchall()
565. cursor.close()
567. # Convert decimal.Decimal values to float and format date
568. historical\_data **=** [(row[0].strftime('%Y-%m-%d'), float(row[1]), float(row[2]), float(row[3]), float(row[4])) **for** row **in** historical\_data]
569. historical\_data.sort(key**=lambda** x: datetime.strptime(x[0], '%Y-%m-%d'))
571. predicted\_close\_prices **=** []
573. current\_date **=** datetime.strptime(historical\_data[**-**1][0], '%Y-%m-%d')
574. model **=** tf.keras.models.load\_model(model\_path)
575. **for** \_ **in** range(5):
576. x\_input **=** np.array([[historical\_data[**-**1][1], historical\_data[**-**1][2], historical\_data[**-**1][3]]])
577. predicted\_close\_price **=** model.predict(x\_input)
578. predicted\_close\_prices.append(float(predicted\_close\_price[0][0]))
579. current\_date **+=** timedelta(days**=**1)
580. historical\_data.append((current\_date.strftime('%Y-%m-%d'), float(predicted\_close\_price[0][0]), float(predicted\_close\_price[0][0]), float(predicted\_close\_price[0][0]), float(historical\_data[**-**1][4])))
581. model.summary()
582. response\_data **=** {
583. "dates": [data\_point[0] **for** data\_point **in** historical\_data],
584. "historical\_prices": [data\_point[4] **for** data\_point **in** historical\_data],
585. "predicted\_prices": [float(price) **for** price **in** predicted\_close\_prices]
586. }
587. **return** jsonify(response\_data)
588. **return** redirect(url\_for('signin'))

591. @app.route('/portfolio', methods**=**['GET', 'POST'])
592. **def** portfolio():
594. **if** request.method **==** 'POST':
595. print("in post")
596. # Handle update request
597. update\_portfolio\_data()
598. **return** jsonify({'success': True})
599. # Fetch portfolio data
600. cursor **=** db.cursor()
601. cursor.execute("SELECT \* FROM portfolio ORDER BY Buy\_Date ASC;")
602. data **=** cursor.fetchall()
604. # Calculate totals
605. total\_cost **=** sum(stock[7] **for** stock **in** data)
606. current\_value **=** sum(stock[8] **for** stock **in** data)
607. total\_returns **=** sum(stock[9] **for** stock **in** data)
608. percent\_change **=** round((total\_returns **/** total\_cost) **\*** 100, 2) **if** total\_cost !**=** 0 **else** 0
610. cursor.close()
611. print("in portfolio")
613. **return** render\_template('portfolio.html', portfolio**=**data, total\_cost**=**total\_cost, current\_value**=**current\_value, total\_returns**=**total\_returns, percent\_change**=**percent\_change)
615. **def** update\_portfolio\_data():
616. **try**:
617. with db.cursor() as cursor:
618. cursor.execute("SELECT \* FROM portfolio ORDER BY Buy\_Date ASC;")
619. data **=** cursor.fetchall()
621. # Calculate totals
622. total\_cost **=** 0
623. current\_value **=** 0
624. total\_returns **=** 0
626. **for** stock **in** data:
627. symbol\_id **=** stock[0]
628. url **=** "https://api.bseindia.com/BseIndiaAPI/api/getScripHeaderData/w"
629. headers **=** {
630. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
631. "Referer": "https://www.bseindia.com/",
632. }
633. payload **=** {
634. "Debtflag": "",
635. "scripcode": symbol\_id,
636. "seriesid": "",
637. }
639. **try**:
640. jsonData **=** requests.get(url, headers**=**headers, params**=**payload).json()
642. **if** "CurrRate" **in** jsonData:
643. LTP **=** jsonData["CurrRate"]["LTP"]
644. **else**:
645. LTP **=** None
647. **if** "Header" **in** jsonData:
648. PrevClose **=** jsonData["Header"]["PrevClose"]
649. **else**:
650. PrevClose **=** None
652. # Update database with LTP and Previous\_Close
653. cursor.execute("UPDATE portfolio SET LTP = %s, PrevClose = %s WHERE Symbol\_id = %s", (LTP, PrevClose, symbol\_id))
654. db.commit()
656. total\_cost **+=** stock[7]
657. current\_value **+=** stock[8]
658. total\_returns **+=** stock[9]
660. **except** requests.exceptions.RequestException as e:
661. print(f"Error fetching data for symbol\_id {symbol\_id}: {e}")
663. **except** mysql.connector.Error as e:
664. print(f"Error fetching data: {e}")

667. @app.route('/add\_stock', methods**=**['POST'])
668. **def** add\_stock():
669. **if** request.method **==** 'POST':
670. **try**:
671. symbol\_name **=** request.form['symbol\_name']
672. symbol\_id **=** request.form['symbol\_id']
673. buy\_date **=** request.form['buy\_date']
674. buy\_price **=** request.form['buy\_price']
675. buy\_qty **=** request.form['buy\_qty']
677. # Fetch data from API
678. url **=** "https://api.bseindia.com/BseIndiaAPI/api/getScripHeaderData/w"
679. headers **=** {
680. "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0",
681. "Referer": "https://www.bseindia.com/",
682. }
683. payload **=** {
684. "Debtflag": "",
685. "scripcode": symbol\_id,
686. "seriesid": "",
687. }
688. response **=** requests.get(url, headers**=**headers, params**=**payload)
689. response.raise\_for\_status()  # Raise an error if the request fails
691. jsonData **=** response.json()
693. LTP **=** jsonData.get("CurrRate", {}).get("LTP")
694. PrevClose **=** jsonData.get("Header", {}).get("PrevClose")
696. # Insert data into the database
697. with db.cursor() as cursor:
698. cursor.execute("INSERT INTO portfolio (Symbol\_id, Symbol\_name, Buy\_Date, Buy\_Price, Buy\_Qty, LTP, PrevClose) VALUES (%s, %s, %s, %s, %s, %s, %s)",
699. (symbol\_id, symbol\_name, buy\_date, buy\_price, buy\_qty, LTP, PrevClose))
700. db.commit()
701. cursor.close()
702. **return** redirect(url\_for('portfolio'))
704. **except** (requests.exceptions.RequestException, mysql.connector.Error) as e:
705. print(f"Error: {e}")
706. **return** render\_template('error.html', message**=**"Error occurred while adding stock.")
708. **return** redirect(url\_for('portfolio'))

711. @app.route('/delete\_stock/<int:symbol\_id>', methods**=**['POST'])
712. **def** delete\_stock(symbol\_id):
713. **try**:
714. with db.cursor() as cursor:
715. cursor.execute("DELETE FROM portfolio WHERE Symbol\_id = %s", (symbol\_id,))
716. db.commit()
717. cursor.close()
718. **return** redirect(url\_for('portfolio'))
719. **except** mysql.connector.Error as e:
720. print(f"Error deleting stock: {e}")
721. **return** render\_template('error.html', message**=**"Error deleting stock from database.")


725. **if** \_\_name\_\_ **==** '\_\_main\_\_':
726. app.run(host**=**'0.0.0.0', port**=**5000, debug**=**True)

/js/function.js

1. **function** showProgressModal() {
2. $('#progressModal').modal('show'); // Show the progress modal
3. }
5. **function** hideProgressModal() {
6. $('#progressModal').modal('hide'); // Hide the progress modal
7. }
9. $(document).ready(**function**() {
10. $('#post-data-form').on('submit', **function**(event) {
11. event.preventDefault(); // Prevent the form from submitting normally
12. showProgressModal(); // Show progress modal when form is submitted
13. $('#progressModal .modal-body').html('<p>Please wait while we process your data...</p>');
14. $.ajax({
15. type: 'POST',
16. url: $(**this**).attr('action'), // Get the form action dynamically
17. data: $(**this**).serialize(),
18. success: **function**(response) {
20. **if** (response.success) {
21. $('#progressModal .modal-body').html('<p>Data updated successfully!</p>');
23. } **else** {
24. $('#progressModal .modal-body').html('<p>Error updating data.</p>');
26. }
27. },
28. error: **function**() {
30. $('#progressModal .modal-body').html('<p>Error updating data.</p>');
31. }
32. });
33. });
34. });
36. **function** portfolio() {
37. $('#progressModal').modal('show');
38. $('#progressModal .modal-body').html('<p>Please wait while we process your data...</p>');
40. fetch(`/portfolio`, {
41. method: 'POST'
42. })
43. .then(response => {
44. **if** (!response.ok) {
45. **throw** **new** Error('Network response was not ok');
46. }
47. **return** response.json();
48. })
49. .then(data => {
50. **if** (data.success) {
51. $('#progressModal .modal-body').html('<p>Data updated successfully!</p>');
52. } **else** {
53. $('#progressModal .modal-body').html('<p>Error updating data.</p>');
54. }
55. })
56. .**catch**(error => {
57. console.error("Error retrieving data:", error);
58. });
59. }
61. **var** ticker = document.getElementById("ticker-marquee");
63. **function** pauseTicker() {
64. ticker.stop();
65. }
67. **function** resumeTicker() {
68. ticker.start();
69. }



74. window.addEventListener('DOMContentLoaded', **function**() {
75. **function** fetchData() {
76. fetch("/get\_symbols")
77. .then(response => {
78. **if** (!response.ok) {
79. **throw** **new** Error('Network response was not ok');
80. }
81. **return** response.json();
82. })
83. .then(data => {
84. console.log(data);
85. const symbolDropdown = document.getElementById("symbolDropdown");
86. data.forEach(symbol => {
87. const uppercaseSymbol = symbol.toUpperCase();
88. const option = document.createElement("option");
89. option.value = uppercaseSymbol;
90. option.textContent = uppercaseSymbol;
91. symbolDropdown.appendChild(option);
92. });
93. })
94. .**catch**(error => {
95. console.error('Error fetching data:', error);
96. });
97. }
99. fetchData();
100. });



105. // Update data when a symbol is selected from the dropdown
106. **function** retrieveData() {
107. const symbol = document.getElementById("symbolDropdown").value;
108. fetch(`/get\_data/${symbol}`)
109. .then(response => response.json())
110. .then(data => {
112. updateTable(data);
113. })
114. .**catch**(error => {
115. console.error("Error retrieving data:", error);
116. });
117. }

120. **function** updateTable(data) {
121. // Find the table body element
122. const tableBody = document.getElementById("table-body");
124. // Clear previous data from the table
125. tableBody.innerHTML = "";
127. // Define the order of the keys for the table columns
128. const columnOrder = ['Date', 'Open', 'High', 'Low', 'Close'];
130. // Iterate over the retrieved data and create table rows
131. data.forEach(rowData => {
132. const row = document.createElement("tr");
134. // Iterate over each key in the specified order and create table cells
135. columnOrder.forEach(key => {
136. const cell = document.createElement("td");
137. **if** (key === 'Date') {
138. cell.textContent = formatDate(rowData[key]);
139. } **else** {
140. cell.textContent = rowData[key];
141. }
142. row.appendChild(cell);
143. });
145. // Append the row to the table body
146. tableBody.appendChild(row);
147. });
148. }

151. **function** updateData() {
152. $('#progressModal').modal('show');
153. $('#progressModal .modal-body').html('<p>Please wait while we process your data...</p>');
154. const symbol = document.getElementById("symbolDropdown").value;
155. fetch(`/update\_data/${symbol}`)
156. .then(response => {
158. **if** (!response.ok) {
159. **throw** **new** Error('Network response was not ok');
160. }
161. **return** response.json();
162. })
163. .then(data => {
164. **if** (data.success) {
165. $('#progressModal .modal-body').html('<p>Data updated successfully!</p>');
167. } **else** {
168. $('#progressModal .modal-body').html('<p>Error updating data.</p>');
169. }
170. })
171. .**catch**(error => {
172. console.error("Error retrieving data:", error);
173. });
174. }
176. **function** TrainData() {
177. $('#progressModal').modal('show');
178. $('#progressModal .modal-body').html('<p>Please wait while we process your data...</p>');
179. const symbol = document.getElementById("symbolDropdown").value;
180. fetch(`/train\_data/${symbol}`)
181. .then(response => {
182. **if** (!response.ok) {
183. **throw** **new** Error('Network response was not ok');
184. }
185. **return** response.json();
186. })
187. .then(data => {
188. **if** (data.success) {
189. $('#progressModal .modal-body').html('<p>Data updated successfully!</p>');

192. } **else** {
193. $('#progressModal .modal-body').html('<p>Error updating data.</p>');
195. }
196. })
197. .**catch**(error => {
199. console.error("Error retrieving data:", error);
200. alert("Error processing stock data");
201. });
202. }



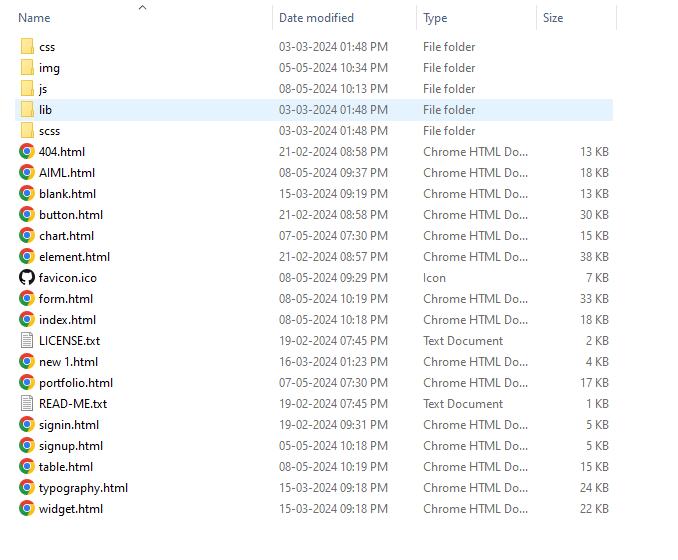

208. **function** predict() {
209. const symbol = document.getElementById("symbolDropdown").value.toUpperCase();
211. // Check if symbol is selected
212. **if** (!symbol) {
213. alert("Please select a symbol");
214. **return**;
215. }
217. fetch(`/prediction/${symbol}`)
218. .then(response => {
219. **if** (response.status === 404) {
220. **throw** **new** Error('Model not found');
221. }
222. **if** (!response.ok) {
223. **throw** **new** Error('Network response was not ok');
224. }
225. **return** response.json();
226. })
227. .then(data => {
228. chart(data);
230. fetch(`/get\_data/${symbol}`)
231. .then(response => response.json())
232. .then(data => {
233. console.log(data);
234. const container = document.getElementById('container');
235. container.innerHTML = ''; // Clear existing data
237. **var** chart;
238. **if** (chart) {
239. chart.dispose(); // Dispose of the previous chart instance
240. }
241. chart = anychart.stock();
243. **var** table = anychart.data.table('Date');
245. table.addData(data);
246. **var** mapping = table.mapAs();
247. **var** mapping = table.mapAs({'open':'Open','high': 'High','low': 'Low','close': 'Close'});
249. **var** plot = chart.plot(0);
250. plot.xGrid(**true**).yGrid(**true**);
251. // set orientation y-axis to the right side

254. **var** series = plot.candlestick(mapping)
255. series.name(symbol).zIndex(50);
256. series
257. .risingFill('green', 0.5)
258. .fallingFill('red', 0.5)
259. .risingStroke('green', 0.5)
260. .fallingStroke('red', 0.5);
261. chart.container('container');
262. chart.draw();
264. })
265. .**catch**(error => {
266. console.error("Error retrieving data:", error);
267. });
268. })
269. .**catch**(error => {
270. console.error("Error retrieving data:", error);
271. **if** (error.message === 'Model not found') {
272. alert("Model not found for the selected symbol. Please Train Model First");
273. } **else** {
274. alert("Error retrieving data. Please try again later.");
275. }
276. });
277. }
279. **function** aiml() {
280. $('#progressModal').modal('show');
281. $('#progressModal .modal-body').html('<p>Please wait while we process your data...</p>');
282. const symbol = document.getElementById("symbolDropdown").value;
283. fetch(`/aiml/${symbol}`)
284. .then(response => {
285. **if** (!response.ok) {
286. **throw** **new** Error('Network response was not ok');
287. }
288. **return** response.json();
289. })
290. .then(data => {
292. const ltp = data.LTP;
293. document.getElementById("current-price").value = ltp;
295. })
296. .**catch**(error => {
298. console.error("Error retrieving data:", error);
299. alert("Error processing stock data");
300. });
301. }

304. **function** BuyOrder() {
305. const symbol = document.getElementById("symbolDropdown").value;
306. const currentPrice = document.getElementById("current-price").value;
307. const quantity = document.getElementById("quantity").value;
308. const total = document.getElementById("total").value;
309. const stopLoss = document.getElementById("stoploss").value;
311. // Get current date and time
312. // Get current date and time
313. const now = **new** Date();
314. const year = now.getFullYear();
315. const month = String(now.getMonth() + 1).padStart(2, '0'); // Months are zero-indexed
316. const day = String(now.getDate()).padStart(2, '0');
317. const hours = String(now.getHours()).padStart(2, '0');
318. const minutes = String(now.getMinutes()).padStart(2, '0');
319. const seconds = String(now.getSeconds()).padStart(2, '0');
321. const dateTime = `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
323. console.log(dateTime)
324. // Define order type
325. const orderType = "Buy";
327. // Prepare data to be sent to the server
328. const orderData = {
329. symbol: symbol,
330. currentPrice: currentPrice,
331. quantity: quantity,
332. total: total,
333. stopLoss: stopLoss,
334. dateTime: dateTime,
335. orderType: orderType
336. };
338. // Send data to server
339. fetch('/saveOrder', {
340. method: 'POST',
341. headers: {
342. 'Content-Type': 'application/json'
343. },
344. body: JSON.stringify(orderData)
345. })
346. .then(response => {
347. **if** (!response.ok) {
348. **throw** **new** Error('Failed to save order');
349. }
350. **return** response.json();
351. })
352. .then(data => {
353. console.log('Order saved successfully:', data);
354. window.alert('Order saved successfully!');
355. // Reset form or show success message
356. window.location.href = '/aiml.html';
357. })
358. .**catch**(error => {
359. console.error('Error saving order:', error);
360. window.alert('Failed to save order. Please try again.');
361. // Show error message to the user
362. });
363. }
365. **function** SellOrder() {
366. const symbol = document.getElementById("symbolDropdown").value;
367. const currentPrice = document.getElementById("current-price").value;
368. const quantity = document.getElementById("quantity").value;
369. const total = document.getElementById("total").value;
370. const stopLoss = document.getElementById("stoploss").value;
371. // Get current date and time
372. const dateTime = **new** Date().toISOString().slice(0, 19).replace('T', ' ');
374. // Define order type
375. const orderType = "Sell";
377. // Prepare data to be sent to the server
378. const orderData = {
379. symbol: symbol,
380. currentPrice: currentPrice,
381. quantity: quantity,
382. total: total,
383. stopLoss: stopLoss,
384. dateTime: dateTime,
385. orderType: orderType
386. };
388. // Send data to server
389. fetch('/saveOrder', {
390. method: 'POST',
391. headers: {
392. 'Content-Type': 'application/json'
393. },
394. body: JSON.stringify(orderData)
395. })
396. .then(response => {
397. **if** (!response.ok) {
398. **throw** **new** Error('Failed to save order');
399. }
400. **return** response.json();
401. })
402. .then(data => {
403. window.alert('Order saved successfully!');
404. window.location.href = '/aiml.html';
405. console.log('Order saved successfully:', data);
406. // Reset form or show success message
407. })
408. .**catch**(error => {
409. console.error('Error saving order:', error);
410. window.alert('Failed to save order. Please try again.');
411. // Show error message to the user
412. });
413. }
415. **function** chart(data) {
416. // Destroy existing chart if it exists
417. **if** (window.myChart3) {
418. window.myChart3.destroy();
419. }
421. **var** ctx1 = $("#line-chart").get(0).getContext("2d");
422. window.myChart3 = **new** Chart(ctx1, {
423. type: "line",
424. data: {
425. labels: data.dates,
426. datasets: [{
427. label: "Close Price",
428. data: data.historical\_prices.map((price, index) => index < data.dates.length - 5 ? price : **null**),
429. backgroundColor: "rgba(0, 156, 255, .7)"
430. },
431. {
432. label: "Predicted",
433. data: Array(data.dates.length - 5).fill(**null**).concat(data.predicted\_prices.slice(-5)),
434. backgroundColor: "rgba(255, 0, 0, .7)"
435. },
436. ]
437. },
438. options: {
439. responsive: **true**
440. }
441. });

444. }

447. **function** formatDate(dateString) {
448. const date = **new** Date(dateString);
449. const day = date.getDate();
450. const monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
451. "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
452. ];
453. const monthIndex = date.getMonth();
454. const year = date.getFullYear();
456. **return** `${day}-${monthNames[monthIndex]}-${year}`;
457. }
458. // Ticker Tape
459. window.addEventListener('DOMContentLoaded', **function**() {
460. // Function to fetch data with delay
461. **function** fetchDataWithDelay() {
462. setTimeout(fetchData, 2000); // Add a delay of 2000 milliseconds (2 seconds)
463. }
465. // Function to fetch data
466. **function** fetchData() {
467. fetch("/ticker")
468. .then(response => {
469. **if** (!response.ok) {
470. **throw** **new** Error('Network response was not ok');
471. }
472. **return** response.json();
473. })
474. .then(data => {
475. const marqueeElement = document.getElementById("ticker-marquee");
476. marqueeElement.innerHTML = ""; // Clear previous data
478. **if** (!data.success) {
479. **throw** **new** Error('Request was not successful');
480. }
482. **if** (!Array.isArray(data.stock\_data)) {
483. **throw** **new** Error('Stock data is not an array');
484. }
486. data.stock\_data.forEach(stock => {
487. const symbolSpan = document.createElement("span");
488. symbolSpan.classList.add("symbol");
489. symbolSpan.textContent = `${stock.Name.toUpperCase()} : ${stock.LTP}`;
491. const trendSpan = document.createElement("span");
492. **if** (parseFloat(stock.LTP) > parseFloat(stock.Previous\_Close)) {
493. trendSpan.style.color = "green";
494. trendSpan.textContent = "▲";
495. } **else** **if** (parseFloat(stock.LTP) < parseFloat(stock.Previous\_Close)) {
496. trendSpan.style.color = "red";
497. trendSpan.textContent = "▼";
498. } **else** {
499. trendSpan.textContent = "↔";
500. }
502. symbolSpan.appendChild(trendSpan);
503. marqueeElement.appendChild(symbolSpan);
504. });
505. })
506. .**catch**(error => {
507. console.error("Error fetching data:", error);
508. });
509. }
511. // Initial fetch with delay
512. fetchDataWithDelay();
514. // Refresh data every minute
515. setInterval(fetchData, 60000);
516. });



1. <!**DOCTYPE** html>
2. <**html** lang="en">
4. <**head**>
5. <**meta** charset="utf-8">
6. <**title**>BCA-YCMOU</**title**>
7. <**meta** content="width=device-width, initial-scale=1.0" name="viewport">
8. <**meta** content="" name="keywords">
9. <**meta** content="" name="description">
11. <!-- Favicon -->
12. <**link** href="img/favicon.ico" rel="icon">
14. <!-- Google Web Fonts -->
15. <**link** rel="preconnect" href="https://fonts.googleapis.com">
16. <**link** rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
17. <**link** href="https://fonts.googleapis.com/css2?family=Heebo:wght@400;500;600;700&display=swap" rel="stylesheet">
19. <!-- Icon Font Stylesheet -->
20. <**link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.10.0/css/all.min.css" rel="stylesheet">
21. <**link** href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
23. <!-- Libraries Stylesheet -->
24. <**link** href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
25. <**link** href="lib/tempusdominus/css/tempusdominus-bootstrap-4.min.css" rel="stylesheet" />
27. <!-- Customized Bootstrap Stylesheet -->
28. <**link** href="css/bootstrap.min.css" rel="stylesheet">
30. <!-- Template Stylesheet -->
31. <**link** href="css/style.css" rel="stylesheet">
33. <!-- Anychart.js -->
34. <**script** src="https://cdn.anychart.com/releases/8.12.0/js/anychart-core.min.js" type="text/javascript"></**script**>
35. <**script** src="https://cdn.anychart.com/releases/8.12.0/js/anychart-stock.min.js" type="text/javascript"></**script**>
36. <**script** src="https://cdn.anychart.com/releases/8.12.0/js/anychart-data-adapter.min.js"></**script**>
38. </**head**>
40. <**body**>
41. <!-- Fixed Header -->
42. <**div** class="ticker-tape" onmouseover="pauseTicker()" onmouseout="resumeTicker()">
43. <!-- Navbar Start -->
44. <**marquee** behavior="scroll" direction="left" scrollamount="8" id="ticker-marquee">
45. <!-- Stock data will be inserted here dynamically -->
46. </**marquee**>
47. </**div**>
48. <!-- End Fixed Header -->
50. <!--   <div class="container-xxl position-relative bg-white d-flex p-0">
51. <!-- Spinner Start -->
52. <**div** id="spinner" class="show bg-white position-fixed translate-middle w-100 vh-100 top-50 start-50 d-flex align-items-center justify-content-center">
53. <**div** class="spinner-border text-primary" style="width: 3rem; height: 3rem;" role="status">
54. <**span** class="sr-only">Loading...</**span**>
55. </**div**>
56. </**div**>
57. <!-- Spinner End -->

60. <!-- Sidebar Start -->
61. <**div** class="sidebar pe-4 pb-3">
62. <**nav** class="navbar bg-light navbar-light">
63. <**a** href="index.html" class="navbar-brand mx-4 mb-3">
64. <**h3** class="text-primary"><**i** class="fa fa-hashtag me-2"></**i**>BCA</**h3**>
65. </**a**>
66. <**div** class="d-flex align-items-center ms-4 mb-4">
67. <**div** class="position-relative">
68. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
69. <**div** class="bg-success rounded-circle border border-2 border-white position-absolute end-0 bottom-0 p-1"></**div**>
70. </**div**>
71. <**div** class="ms-3">
72. <**h6** class="mb-0">{{ session['email'] }}</**h6**>
74. </**div**>
75. </**div**>
76. <**div** class="navbar-nav w-100">
77. <**a** href="index.html" class="nav-item nav-link active"><**i** class="fa fa-tachometer-alt me-2"></**i**>Dashboard</**a**>
78. <!--    <div class="nav-item dropdown">
79. <a href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><i class="fa fa-laptop me-2"></i>Elements</a>
80. <div class="dropdown-menu bg-transparent border-0">
81. <a href="button.html" class="dropdown-item">Buttons</a>
82. <a href="typography.html" class="dropdown-item">Typography</a>
83. <a href="element.html" class="dropdown-item">Other Elements</a>
84. </div>
85. </div>
86. <a href="widget.html" class="nav-item nav-link"><i class="fa fa-th me-2"></i>Widgets</a> -->
87. <**a** href="form.html" class="nav-item nav-link"><**i** class="fa fa-keyboard me-2"></**i**>Forms</**a**>
88. <**a** href="table.html" class="nav-item nav-link"><**i** class="fa fa-table me-2"></**i**>Tables</**a**>
89. <**a** href="chart.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Charts</**a**>
90. <**a** href="portfolio.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Portfolio</**a**>
91. <**a** href="aiml.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Order Book</**a**>
92. <**div** class="nav-item dropdown">
93. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><**i** class="far fa-file-alt me-2"></**i**>Pages</**a**>
94. <**div** class="dropdown-menu bg-transparent border-0">
95. <**a** href="signin.html" class="dropdown-item">Sign In</**a**>
96. <**a** href="signup.html" class="dropdown-item">Sign Up</**a**>
97. <**a** href="404.html" class="dropdown-item">404 Error</**a**>
98. <**a** href="blank.html" class="dropdown-item">Blank Page</**a**>
99. </**div**>
100. </**div**>
101. </**div**>
102. </**nav**>
103. </**div**>
104. <!-- Sidebar End -->

107. <!-- Content Start -->
108. <**div** class="content">
109. <!-- Navbar Start -->
110. <**nav** class="navbar navbar-expand bg-light navbar-light sticky-top px-4 py-0">
111. <**a** href="index.html" class="navbar-brand d-flex d-lg-none me-4">
112. <**h2** class="text-primary mb-0"><**i** class="fa fa-hashtag"></**i**></**h2**>
113. </**a**>
114. <**a** href="#" class="sidebar-toggler flex-shrink-0">
115. <**i** class="fa fa-bars"></**i**>
116. </**a**>
117. <**form** class="d-none d-md-flex ms-4">
118. <**input** class="form-control border-0" type="search" placeholder="Search">
119. </**form**>
120. <**div** class="navbar-nav align-items-center ms-auto">
121. <**div** class="nav-item dropdown">
122. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
123. <**i** class="fa fa-envelope me-lg-2"></**i**>
124. <**span** class="d-none d-lg-inline-flex">Message</**span**>
125. </**a**>
126. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
127. <**a** href="#" class="dropdown-item">
128. <**div** class="d-flex align-items-center">
129. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
130. <**div** class="ms-2">
131. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
132. <**small**>15 minutes ago</**small**>
133. </**div**>
134. </**div**>
135. </**a**>
136. <**hr** class="dropdown-divider">
137. <**a** href="#" class="dropdown-item">
138. <**div** class="d-flex align-items-center">
139. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
140. <**div** class="ms-2">
141. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
142. <**small**>15 minutes ago</**small**>
143. </**div**>
144. </**div**>
145. </**a**>
146. <**hr** class="dropdown-divider">
147. <**a** href="#" class="dropdown-item">
148. <**div** class="d-flex align-items-center">
149. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
150. <**div** class="ms-2">
151. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
152. <**small**>15 minutes ago</**small**>
153. </**div**>
154. </**div**>
155. </**a**>
156. <**hr** class="dropdown-divider">
157. <**a** href="#" class="dropdown-item text-center">See all message</**a**>
158. </**div**>
159. </**div**>
160. <**div** class="nav-item dropdown">
161. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
162. <**i** class="fa fa-bell me-lg-2"></**i**>
163. <**span** class="d-none d-lg-inline-flex">Notificatin</**span**>
164. </**a**>
165. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
166. <**a** href="#" class="dropdown-item">
167. <**h6** class="fw-normal mb-0">Profile updated</**h6**>
168. <**small**>15 minutes ago</**small**>
169. </**a**>
170. <**hr** class="dropdown-divider">
171. <**a** href="#" class="dropdown-item">
172. <**h6** class="fw-normal mb-0">New user added</**h6**>
173. <**small**>15 minutes ago</**small**>
174. </**a**>
175. <**hr** class="dropdown-divider">
176. <**a** href="#" class="dropdown-item">
177. <**h6** class="fw-normal mb-0">Password changed</**h6**>
178. <**small**>15 minutes ago</**small**>
179. </**a**>
180. <**hr** class="dropdown-divider">
181. <**a** href="#" class="dropdown-item text-center">See all notifications</**a**>
182. </**div**>
183. </**div**>
184. <**div** class="nav-item dropdown">
185. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
186. <**img** class="rounded-circle me-lg-2" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
187. <**span** class="d-none d-lg-inline-flex">{{ session['email'] }}</**span**>
188. </**a**>
189. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
191. <**a** href="{{ url\_for('logout') }}" class="dropdown-item">Log Out</**a**>
192. </**div**>
193. </**div**>
194. </**div**>
195. </**nav**>
196. <!-- Navbar End -->
197. <!-- Navbar End -->
198. <!-- Navbar End -->
200. <**div** class="container-fluid pt-4 px-4">
201. <**div** class="row">
202. <**div** class="col-md-3">
203. <**label** for="symbolDropdown" class="form-label">Select Symbol:</**label**>
204. <**select** class="form-select" id="symbolDropdown">
205. <**option** value="" selected disabled>Select a symbol</**option**>
206. </**select**>
207. </**div**>
208. <!-- Button to Retrieve Data -->
209. <**div** class="col-md-5  mt-3">
210. <**div** class="col-md-5  mt-3">
211. <**button** class="btn btn-primary" onclick="predict()">Retrieve Data</**button**>
212. </**div**>
213. </**div**>
214. </**div**>
215. </**div**>
216. <**div** class="container-fluid pt-4 px-4">
217. <**div** class="bg-light rounded h-100 p-4">
218. <**div** id="container" style=" height: 400px;"></**div**>
219. </**div**>
220. </**div**>
221. <!-- Chart Start -->

224. <**div** class="container-fluid pt-4 px-4">
225. <**div** class="bg-light rounded h-100 p-4">
226. <**h6** class="mb-4">Predicted Chart</**h6**>
227. <**canvas** id="line-chart" width="1000" height="300"></**canvas**>
228. </**div**>
229. </**div**>
230. <**div** class="container-fluid pt-4 px-4">
231. <**div** class="row">
233. <**div** class="col-sm-6 col-xl-6">
234. <**div** class="bg-light rounded h-100 p-4">
235. <!-- <div class="owl-carousel testimonial-carousel"> -->
236. <**div** class="testimonial-item text-center" id="topGainers">
237. <**h5** class="mb-1">Top Gainers</**h5**>
238. <**table** class="table table-bordered">
239. <**thead**>
240. <**tr**>
241. <**th**>Symbol</**th**>
242. <**th**>LTP</**th**>
243. <**th**>% Changed</**th**>
244. <**th**>Value</**th**>
245. </**tr**>
246. </**thead**>
247. <**tbody** id="topGainersBody">
248. <!-- Data will be dynamically added here -->
249. </**tbody**>
250. </**table**>
251. </**div**>
253. <!--</div> -->
254. </**div**>
255. </**div**>
257. <**div** class="col-sm-6 col-xl-6">
258. <**div** class="bg-light rounded h-100 p-4">
259. <**div** class="testimonial-item text-center" id="topLosers">
260. <**h5** class="mb-1">Top Loosers</**h5**>
261. <**table** class="table table-bordered">
262. <**thead**>
263. <**tr**>
264. <**th**>Symbol</**th**>
265. <**th**>LTP</**th**>
266. <**th**>% Changed</**th**>
267. <**th**>Value</**th**>
268. </**tr**>
269. </**thead**>
270. <**tbody** id="topLosersBody">
271. <!-- Data will be dynamically added here -->
272. </**tbody**>
273. </**table**>
274. </**div**>
275. </**div**>
276. </**div**>
277. </**div**>
279. </**div**>
281. <!-- Footer Start -->
282. <**div** class="container-fluid pt-4 px-4">
283. <**div** class="bg-light rounded-top p-4">
284. <**div** class="row">
285. <**div** class="col-12 col-sm-6 text-center text-sm-start">
286. © <**a** href="#">BCA Project on Stock Market using Machine Learning</**a**>,</**br**> All Right Reserved.
287. </**div**>
288. <**div** class="col-12 col-sm-6 text-center text-sm-end">
289. <!--/\*\*\* This template is free as long as you keep the footer author’s credit link/attribution link/backlink. If you'd like to use the template without the footer author’s credit link/attribution link/backlink, you can purchase the Credit Removal License from "https://htmlcodex.com/credit-removal". Thank you for your support. \*\*\*/-->
290. Designed By Nitesh Chavan</**a**>
291. </**br**>
293. </**div**>
294. </**div**>
295. </**div**>
296. </**div**>
297. <!-- Footer End -->
298. </**div**>
299. <!-- Content End -->

302. <!-- Back to Top -->
303. <**a** href="#" class="btn btn-lg btn-primary btn-lg-square back-to-top"><**i** class="bi bi-arrow-up"></**i**></**a**>
304. </**div**>
305. <**script**>






313. window.addEventListener('DOMContentLoaded', function() {
314. function fetchDataWithDelay() {
315. setTimeout(fetchData, 2000); // Add a delay of 2000 milliseconds (2 seconds)
316. }
318. // Function to fetch data for both gainers and losers
319. function fetchData() {
320. fetch("/trending/gainers") // Fetch data for gainers
321. .then(response => {
322. if (!response.ok) {
323. throw new Error('Network response was not ok');
324. }
325. return response.json();
326. })
327. .then(data => {
328. // Populate top gainers table
329. const topGainersBody = document.getElementById('topGainersBody');
330. topGainersBody.innerHTML = ''; // Clear existing data
331. data.scrip\_data.forEach(item => {
332. const row = document.createElement('tr');
333. row.innerHTML = `
334. <**td**>${item.scrip\_id}</**td**>
335. <**td**>${item.Ltradert}</**td**>
336. <**td**>${item.change\_percent}</**td**>
337. <**td**>${item.change\_val}</**td**>
338. `;
339. topGainersBody.appendChild(row);
340. });
341. })
342. .catch(error => {
343. console.error("Error fetching top gainers:", error);
344. });
346. fetch("/trending/loosers") // Fetch data for losers
347. .then(response => {
348. if (!response.ok) {
349. throw new Error('Network response was not ok');
350. }
351. return response.json();
352. })
353. .then(data => {
354. // Populate top losers table
355. const topLosersBody = document.getElementById('topLosersBody');
356. topLosersBody.innerHTML = ''; // Clear existing data
357. data.scrip\_data.forEach(item => {
358. const row = document.createElement('tr');
359. row.innerHTML = `
360. <**td**>${item.scrip\_id}</**td**>
361. <**td**>${item.Ltradert}</**td**>
362. <**td**>${item.change\_percent}</**td**>
363. <**td**>${item.change\_val}</**td**>
364. `;
365. topLosersBody.appendChild(row);
366. });
367. })
368. .catch(error => {
369. console.error("Error fetching top losers:", error);
370. });
371. }
373. // Initial fetch with delay
374. fetchDataWithDelay();
376. // Refresh data every minute
377. });

380. </**script**>
382. <!-- JavaScript Libraries -->
383. <**script** src="https://code.jquery.com/jquery-3.4.1.min.js"></**script**>
384. <**script** src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.bundle.min.js"></**script**>
385. <**script** src="lib/chart/chart.min.js"></**script**>
386. <**script** src="lib/easing/easing.min.js"></**script**>
387. <**script** src="lib/waypoints/waypoints.min.js"></**script**>
388. <**script** src="lib/owlcarousel/owl.carousel.min.js"></**script**>
389. <**script** src="lib/tempusdominus/js/moment.min.js"></**script**>
390. <**script** src="lib/tempusdominus/js/moment-timezone.min.js"></**script**>
391. <**script** src="lib/tempusdominus/js/tempusdominus-bootstrap-4.min.js"></**script**>
393. <!-- Template Javascript -->
394. <**script** src="js/main.js"></**script**>
395. <**script** src="js/functions.js"></**script**>
396. <**script** src="js/ticker.js"></**script**>
397. </**body**>
399. </**html**>
400. <!**DOCTYPE** html>
401. <**html** lang="en">
403. <**head**>
404. <**meta** charset="utf-8">
405. <**title**>BCA-YCMOU</**title**>
406. <**meta** content="width=device-width, initial-scale=1.0" name="viewport">
407. <**meta** content="" name="keywords">
408. <**meta** content="" name="description">
410. <!-- Favicon -->
411. <**link** href="img/favicon.ico" rel="icon">
413. <!-- Google Web Fonts -->
414. <**link** rel="preconnect" href="https://fonts.googleapis.com">
415. <**link** rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
416. <**link** href="https://fonts.googleapis.com/css2?family=Heebo:wght@400;500;600;700&display=swap" rel="stylesheet">
418. <!-- Icon Font Stylesheet -->
419. <**link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.10.0/css/all.min.css" rel="stylesheet">
420. <**link** href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
422. <!-- Libraries Stylesheet -->
423. <**link** href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
424. <**link** href="lib/tempusdominus/css/tempusdominus-bootstrap-4.min.css" rel="stylesheet" />
426. <!-- Customized Bootstrap Stylesheet -->
427. <**link** href="css/bootstrap.min.css" rel="stylesheet">
429. <!-- Template Stylesheet -->
430. <**link** href="css/style.css" rel="stylesheet">
432. </**head**>
434. <**body**>
436. <!-- Fixed Header -->
437. <**div** class="ticker-tape">
438. <!-- Navbar Start -->
439. <!-- <nav class="navbar navbar-expand bg-light navbar-light sticky-top px-4 py-0"> -->
440. <**marquee** behavior="scroll" direction="left" scrollamount="8" id="ticker-marquee">
441. <!-- Stock data will be inserted here dynamically -->
442. </**marquee**>
443. <!--  </nav> -->
444. <!-- Navbar End -->
445. </**div**>
447. <!-- End Fixed Header -->
448. <**div** class="container-xxl position-relative bg-white d-flex p-0">
449. <!-- Spinner Start -->
450. <**div** id="spinner" class="show bg-white position-fixed translate-middle w-100 vh-100 top-50 start-50 d-flex align-items-center justify-content-center">
451. <**div** class="spinner-border text-primary" style="width: 3rem; height: 3rem;" role="status">
452. <**span** class="sr-only">Loading...</**span**>
453. </**div**>
454. </**div**>
455. <!-- Spinner End -->

458. <!-- Sidebar Start -->
459. <**div** class="sidebar pe-4 pb-3">
460. <**nav** class="navbar bg-light navbar-light">
461. <**a** href="index.html" class="navbar-brand mx-4 mb-3">
462. <**h3** class="text-primary"><**i** class="fa fa-hashtag me-2"></**i**>BCA</**h3**>
463. </**a**>
464. <**div** class="d-flex align-items-center ms-4 mb-4">
465. <**div** class="position-relative">
466. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
467. <**div** class="bg-success rounded-circle border border-2 border-white position-absolute end-0 bottom-0 p-1"></**div**>
468. </**div**>
469. <**div** class="ms-3">
470. <**h6** class="mb-0">{{ session['email'] }}</**h6**>
472. </**div**>
473. </**div**>
474. <**div** class="navbar-nav w-100">
475. <**a** href="index.html" class="nav-item nav-link"><**i** class="fa fa-tachometer-alt me-2"></**i**>Dashboard</**a**>
476. <!--    <div class="nav-item dropdown">
477. <a href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><i class="fa fa-laptop me-2"></i>Elements</a>
478. <div class="dropdown-menu bg-transparent border-0">
479. <a href="button.html" class="dropdown-item">Buttons</a>
480. <a href="typography.html" class="dropdown-item">Typography</a>
481. <a href="element.html" class="dropdown-item">Other Elements</a>
482. </div>
483. </div>
484. <a href="widget.html" class="nav-item nav-link"><i class="fa fa-th me-2"></i>Widgets</a> -->
485. <**a** href="form.html" class="nav-item nav-link active"><**i** class="fa fa-keyboard me-2"></**i**>Forms</**a**>
486. <**a** href="table.html" class="nav-item nav-link"><**i** class="fa fa-table me-2"></**i**>Tables</**a**>
487. <**a** href="chart.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Charts</**a**>
488. <**a** href="portfolio.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Portfolio</**a**>
489. <**a** href="aiml.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Order Book</**a**>
490. <**div** class="nav-item dropdown">
491. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><**i** class="far fa-file-alt me-2"></**i**>Pages</**a**>
492. <**div** class="dropdown-menu bg-transparent border-0">
493. <**a** href="signin.html" class="dropdown-item">Sign In</**a**>
494. <**a** href="signup.html" class="dropdown-item">Sign Up</**a**>
495. <**a** href="404.html" class="dropdown-item">404 Error</**a**>
496. <**a** href="blank.html" class="dropdown-item">Blank Page</**a**>
497. </**div**>
498. </**div**>
499. </**div**>
500. </**nav**>
501. </**div**>
502. <!-- Sidebar End -->

505. <!-- Content Start -->
506. <**div** class="content">
507. <!-- Navbar Start -->
508. <**nav** class="navbar navbar-expand bg-light navbar-light sticky-top px-4 py-0">
509. <**a** href="index.html" class="navbar-brand d-flex d-lg-none me-4">
510. <**h2** class="text-primary mb-0"><**i** class="fa fa-hashtag"></**i**></**h2**>
511. </**a**>
512. <**a** href="#" class="sidebar-toggler flex-shrink-0">
513. <**i** class="fa fa-bars"></**i**>
514. </**a**>
515. <**form** class="d-none d-md-flex ms-4">
516. <**input** class="form-control border-0" type="search" placeholder="Search">
517. </**form**>
518. <**div** class="navbar-nav align-items-center ms-auto">
519. <**div** class="nav-item dropdown">
520. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
521. <**i** class="fa fa-envelope me-lg-2"></**i**>
522. <**span** class="d-none d-lg-inline-flex">Message</**span**>
523. </**a**>
524. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
525. <**a** href="#" class="dropdown-item">
526. <**div** class="d-flex align-items-center">
527. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
528. <**div** class="ms-2">
529. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
530. <**small**>15 minutes ago</**small**>
531. </**div**>
532. </**div**>
533. </**a**>
534. <**hr** class="dropdown-divider">
535. <**a** href="#" class="dropdown-item">
536. <**div** class="d-flex align-items-center">
537. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
538. <**div** class="ms-2">
539. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
540. <**small**>15 minutes ago</**small**>
541. </**div**>
542. </**div**>
543. </**a**>
544. <**hr** class="dropdown-divider">
545. <**a** href="#" class="dropdown-item">
546. <**div** class="d-flex align-items-center">
547. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
548. <**div** class="ms-2">
549. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
550. <**small**>15 minutes ago</**small**>
551. </**div**>
552. </**div**>
553. </**a**>
554. <**hr** class="dropdown-divider">
555. <**a** href="#" class="dropdown-item text-center">See all message</**a**>
556. </**div**>
557. </**div**>
558. <**div** class="nav-item dropdown">
559. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
560. <**i** class="fa fa-bell me-lg-2"></**i**>
561. <**span** class="d-none d-lg-inline-flex">Notificatin</**span**>
562. </**a**>
563. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
564. <**a** href="#" class="dropdown-item">
565. <**h6** class="fw-normal mb-0">Profile updated</**h6**>
566. <**small**>15 minutes ago</**small**>
567. </**a**>
568. <**hr** class="dropdown-divider">
569. <**a** href="#" class="dropdown-item">
570. <**h6** class="fw-normal mb-0">New user added</**h6**>
571. <**small**>15 minutes ago</**small**>
572. </**a**>
573. <**hr** class="dropdown-divider">
574. <**a** href="#" class="dropdown-item">
575. <**h6** class="fw-normal mb-0">Password changed</**h6**>
576. <**small**>15 minutes ago</**small**>
577. </**a**>
578. <**hr** class="dropdown-divider">
579. <**a** href="#" class="dropdown-item text-center">See all notifications</**a**>
580. </**div**>
581. </**div**>
582. <**div** class="nav-item dropdown">
583. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
584. <**img** class="rounded-circle me-lg-2" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
585. <**span** class="d-none d-lg-inline-flex">{{ session['email'] }}</**span**>
586. </**a**>
587. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
589. <**a** href="{{ url\_for('logout') }}" class="dropdown-item">Log Out</**a**>
590. </**div**>
591. </**div**>
592. </**div**>
593. </**nav**>
594. <!-- Navbar End -->
595. <!-- Progress Modal -->
596. <**div** class="modal" id="progressModal" tabindex="-1" role="dialog">
597. <**div** class="modal-dialog" role="document">
598. <**div** class="modal-content">
599. <**div** class="modal-header">
600. <**h5** class="modal-title">Processing Data</**h5**>
601. </**div**>
602. <**div** class="modal-body">
603. <!-- Content will be dynamically updated -->
604. </**div**>
605. <**div** class="modal-footer">
606. <**button** type="button" class="btn btn-primary" id="okButton" data-bs-dismiss="modal">OK</**button**>
607. </**div**>
608. </**div**>
609. </**div**>
610. </**div**>

613. <!-- Form Start -->
614. <**div** class="container-fluid pt-4 px-4">
615. <**div** class="row g-4">
616. <**div** class="col-sm-12 col-xl-6">
617. <**div** class="bg-light rounded h-100 p-4">
618. <**h6** class="mb-4">Upload Stock Data</**h6**>
619. <**form**  id="post-data-form" action="/process\_script\_data" method="post">
620. <**div** class="mb-3">
621. <**label** for="stockData" class="form-label">Enter BSE Script Code:</**label**>
622. <**input** type="text" class="form-control" id="stockData" name="stockData">
623. </**div**>
624. <**div** class="row mb-3">
625. <**div** class="col">
626. <**label** for="fromDate" class="form-label">From Date:</**label**>
627. <**input** type="date" class="form-control" id="fromDate" name="fromDate">
628. </**div**>
629. <**div** class="col">
630. <**label** for="toDate" class="form-label">To Date:</**label**>
631. <**input** type="date" class="form-control" id="toDate" name="toDate">
632. </**div**>
633. </**div**>
634. <**div** class="text-end">
635. <**button** type="submit" class="btn btn-primary">Process</**button**>
636. </**div**>
637. </**form**>
638. </**div**>
639. </**div**>
640. <**div** class="col-sm-12 col-xl-6">
641. <**div** class="bg-light rounded h-100 p-4">
642. <**h6** class="mb-4">Train Model using Machine Learning(Tensorflow)</**h6**>
643. <**div** class="mb-3">
644. <**label** for="symbolDropdown" class="form-label">Select Symbol:</**label**>
645. <**select** class="form-select" id="symbolDropdown">
646. <**option** value="" selected disabled>Select a symbol</**option**>
647. </**select**>
648. </**div**>
649. <**div** class="text-end">
650. <**button** class="btn btn-primary" onclick="TrainData()">Train Data</**button**>
651. </**div**>
652. </**div**>
653. </**div**>
654. </**div**>
655. </**div**>


659. <!-- Footer Start -->
660. <**div** class="container-fluid pt-4 px-4">
661. <**div** class="bg-light rounded-top p-4">
662. <**div** class="row">
663. <**div** class="col-12 col-sm-6 text-center text-sm-start">
664. © <**a** href="#">BCA Project on Stock Market using Machine Learning</**a**>,</**br**> All Right Reserved.
665. </**div**>
666. <**div** class="col-12 col-sm-6 text-center text-sm-end">
667. <!--/\*\*\* This template is free as long as you keep the footer author’s credit link/attribution link/backlink. If you'd like to use the template without the footer author’s credit link/attribution link/backlink, you can purchase the Credit Removal License from "https://htmlcodex.com/credit-removal". Thank you for your support. \*\*\*/-->
668. Designed By Nitesh Chavan</**a**>
669. </**br**>
671. </**div**>
672. </**div**>
673. </**div**>
674. </**div**>
675. <!-- Footer End -->
676. </**div**>
677. <!-- Content End -->

680. <!-- Back to Top -->
681. <**a** href="#" class="btn btn-lg btn-primary btn-lg-square back-to-top"><**i** class="bi bi-arrow-up"></**i**></**a**>
682. </**div**>
684. <!-- JavaScript Libraries -->
685. <**script** src="https://code.jquery.com/jquery-3.4.1.min.js"></**script**>
686. <**script** src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.bundle.min.js"></**script**>
687. <**script** src="lib/chart/chart.min.js"></**script**>
688. <**script** src="lib/easing/easing.min.js"></**script**>
689. <**script** src="lib/waypoints/waypoints.min.js"></**script**>
690. <**script** src="lib/owlcarousel/owl.carousel.min.js"></**script**>
691. <**script** src="lib/tempusdominus/js/moment.min.js"></**script**>
692. <**script** src="lib/tempusdominus/js/moment-timezone.min.js"></**script**>
693. <**script** src="lib/tempusdominus/js/tempusdominus-bootstrap-4.min.js"></**script**>
695. <!-- Template Javascript -->
696. <**script** src="js/main.js"></**script**>
697. <**script** src="js/functions.js"></**script**>
698. <**script** src="js/ticker.js"></**script**>
699. </**body**>
701. </**html**>
702. <!**DOCTYPE** html>
703. <**html** lang="en">
705. <**head**>
706. <**meta** charset="utf-8">
707. <**title**>BCA-YCMOU</**title**>
708. <**meta** content="width=device-width, initial-scale=1.0" name="viewport">
709. <**meta** content="" name="keywords">
710. <**meta** content="" name="description">
712. <!-- Favicon -->
713. <**link** href="img/favicon.ico" rel="icon">
715. <!-- Google Web Fonts -->
716. <**link** rel="preconnect" href="https://fonts.googleapis.com">
717. <**link** rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
718. <**link** href="https://fonts.googleapis.com/css2?family=Heebo:wght@400;500;600;700&display=swap" rel="stylesheet">
720. <!-- Icon Font Stylesheet -->
721. <**link** href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.10.0/css/all.min.css" rel="stylesheet">
722. <**link** href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
724. <!-- Libraries Stylesheet -->
725. <**link** href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
726. <**link** href="lib/tempusdominus/css/tempusdominus-bootstrap-4.min.css" rel="stylesheet" />
728. <!-- Customized Bootstrap Stylesheet -->
729. <**link** href="css/bootstrap.min.css" rel="stylesheet">
731. <!-- Template Stylesheet -->
732. <**link** href="css/style.css" rel="stylesheet">
733. </**head**>
735. <**body**>
737. <!-- Fixed Header -->
738. <**div** class="ticker-tape">
739. <!-- Navbar Start -->
740. <!-- <nav class="navbar navbar-expand bg-light navbar-light sticky-top px-4 py-0"> -->
741. <**marquee** behavior="scroll" direction="left" scrollamount="8" id="ticker-marquee">
742. <!-- Stock data will be inserted here dynamically -->
743. </**marquee**>
744. <!--  </nav> -->
745. <!-- Navbar End -->
746. </**div**>

749. <**div** class="container-xxl position-relative bg-white d-flex p-0">
750. <!-- Spinner Start -->
751. <**div** id="spinner" class="show bg-white position-fixed translate-middle w-100 vh-100 top-50 start-50 d-flex align-items-center justify-content-center">
752. <**div** class="spinner-border text-primary" style="width: 3rem; height: 3rem;" role="status">
753. <**span** class="sr-only">Loading...</**span**>
754. </**div**>
755. </**div**>
756. <!-- Spinner End -->

759. <!-- Sidebar Start -->
760. <**div** class="sidebar pe-4 pb-3">
761. <**nav** class="navbar bg-light navbar-light">
762. <**a** href="index.html" class="navbar-brand mx-4 mb-3">
763. <**h3** class="text-primary"><**i** class="fa fa-hashtag me-2"></**i**>BCA</**h3**>
764. </**a**>
765. <**div** class="d-flex align-items-center ms-4 mb-4">
766. <**div** class="position-relative">
767. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
768. <**div** class="bg-success rounded-circle border border-2 border-white position-absolute end-0 bottom-0 p-1"></**div**>
769. </**div**>
770. <**div** class="ms-3">
771. <**h6** class="mb-0">{{ session['email'] }}</**h6**>
773. </**div**>
774. </**div**>
775. <**div** class="navbar-nav w-100">
776. <**a** href="index.html" class="nav-item nav-link"><**i** class="fa fa-tachometer-alt me-2"></**i**>Dashboard</**a**>
777. <!--    <div class="nav-item dropdown">
778. <a href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><i class="fa fa-laptop me-2"></i>Elements</a>
779. <div class="dropdown-menu bg-transparent border-0">
780. <a href="button.html" class="dropdown-item">Buttons</a>
781. <a href="typography.html" class="dropdown-item">Typography</a>
782. <a href="element.html" class="dropdown-item">Other Elements</a>
783. </div>
784. </div>
785. <a href="widget.html" class="nav-item nav-link"><i class="fa fa-th me-2"></i>Widgets</a> -->
786. <**a** href="form.html" class="nav-item nav-link"><**i** class="fa fa-keyboard me-2"></**i**>Forms</**a**>
787. <**a** href="table.html" class="nav-item nav-link active"><**i** class="fa fa-table me-2"></**i**>Tables</**a**>
788. <**a** href="chart.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Charts</**a**>
789. <**a** href="portfolio.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Portfolio</**a**>
790. <**a** href="aiml.html" class="nav-item nav-link"><**i** class="fa fa-chart-bar me-2"></**i**>Order Book</**a**>
791. <**div** class="nav-item dropdown">
792. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown"><**i** class="far fa-file-alt me-2"></**i**>Pages</**a**>
793. <**div** class="dropdown-menu bg-transparent border-0">
794. <**a** href="signin.html" class="dropdown-item">Sign In</**a**>
795. <**a** href="signup.html" class="dropdown-item">Sign Up</**a**>
796. <**a** href="404.html" class="dropdown-item">404 Error</**a**>
797. <**a** href="blank.html" class="dropdown-item">Blank Page</**a**>
798. </**div**>
799. </**div**>
800. </**div**>
801. </**nav**>
802. </**div**>
803. <!-- Sidebar End -->

806. <!-- Content Start -->
807. <**div** class="content">
808. <!-- Navbar Start -->
809. <**nav** class="navbar navbar-expand bg-light navbar-light sticky-top px-4 py-0">
810. <**a** href="index.html" class="navbar-brand d-flex d-lg-none me-4">
811. <**h2** class="text-primary mb-0"><**i** class="fa fa-hashtag"></**i**></**h2**>
812. </**a**>
813. <**a** href="#" class="sidebar-toggler flex-shrink-0">
814. <**i** class="fa fa-bars"></**i**>
815. </**a**>
816. <**form** class="d-none d-md-flex ms-4">
817. <**input** class="form-control border-0" type="search" placeholder="Search">
818. </**form**>
819. <**div** class="navbar-nav align-items-center ms-auto">
820. <**div** class="nav-item dropdown">
821. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
822. <**i** class="fa fa-envelope me-lg-2"></**i**>
823. <**span** class="d-none d-lg-inline-flex">Message</**span**>
824. </**a**>
825. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
826. <**a** href="#" class="dropdown-item">
827. <**div** class="d-flex align-items-center">
828. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
829. <**div** class="ms-2">
830. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
831. <**small**>15 minutes ago</**small**>
832. </**div**>
833. </**div**>
834. </**a**>
835. <**hr** class="dropdown-divider">
836. <**a** href="#" class="dropdown-item">
837. <**div** class="d-flex align-items-center">
838. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
839. <**div** class="ms-2">
840. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
841. <**small**>15 minutes ago</**small**>
842. </**div**>
843. </**div**>
844. </**a**>
845. <**hr** class="dropdown-divider">
846. <**a** href="#" class="dropdown-item">
847. <**div** class="d-flex align-items-center">
848. <**img** class="rounded-circle" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
849. <**div** class="ms-2">
850. <**h6** class="fw-normal mb-0">admin send you a message</**h6**>
851. <**small**>15 minutes ago</**small**>
852. </**div**>
853. </**div**>
854. </**a**>
855. <**hr** class="dropdown-divider">
856. <**a** href="#" class="dropdown-item text-center">See all message</**a**>
857. </**div**>
858. </**div**>
859. <**div** class="nav-item dropdown">
860. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
861. <**i** class="fa fa-bell me-lg-2"></**i**>
862. <**span** class="d-none d-lg-inline-flex">Notificatin</**span**>
863. </**a**>
864. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
865. <**a** href="#" class="dropdown-item">
866. <**h6** class="fw-normal mb-0">Profile updated</**h6**>
867. <**small**>15 minutes ago</**small**>
868. </**a**>
869. <**hr** class="dropdown-divider">
870. <**a** href="#" class="dropdown-item">
871. <**h6** class="fw-normal mb-0">New user added</**h6**>
872. <**small**>15 minutes ago</**small**>
873. </**a**>
874. <**hr** class="dropdown-divider">
875. <**a** href="#" class="dropdown-item">
876. <**h6** class="fw-normal mb-0">Password changed</**h6**>
877. <**small**>15 minutes ago</**small**>
878. </**a**>
879. <**hr** class="dropdown-divider">
880. <**a** href="#" class="dropdown-item text-center">See all notifications</**a**>
881. </**div**>
882. </**div**>
883. <**div** class="nav-item dropdown">
884. <**a** href="#" class="nav-link dropdown-toggle" data-bs-toggle="dropdown">
885. <**img** class="rounded-circle me-lg-2" src="img/user.jpg" alt="" style="width: 40px; height: 40px;">
886. <**span** class="d-none d-lg-inline-flex">{{ session['email'] }}</**span**>
887. </**a**>
888. <**div** class="dropdown-menu dropdown-menu-end bg-light border-0 rounded-0 rounded-bottom m-0">
890. <**a** href="{{ url\_for('logout') }}" class="dropdown-item">Log Out</**a**>
891. </**div**>
892. </**div**>
893. </**div**>
894. </**nav**>
895. <!-- Navbar End -->
896. <!-- Progress Modal -->
897. <**div** class="modal" id="progressModal" tabindex="-1" role="dialog">
898. <**div** class="modal-dialog" role="document">
899. <**div** class="modal-content">
900. <**div** class="modal-header">
901. <**h5** class="modal-title">Processing Data</**h5**>
902. </**div**>
903. <**div** class="modal-body">
904. <!-- Content will be dynamically updated -->
905. </**div**>
906. <**div** class="modal-footer">
907. <**button** type="button" class="btn btn-primary" id="okButton" data-bs-dismiss="modal">OK</**button**>
908. </**div**>
909. </**div**>
910. </**div**>
911. </**div**>
913. <!-- Dropdown for Symbol Selection -->
914. <**div** class="container-fluid pt-4 px-4">
915. <**div** class="row">
916. <**div** class="col-md-4 mb-3 mt-3">
917. <**label** for="symbolDropdown" class="form-label">Select Symbol:</**label**>
918. <**select** class="form-select" id="symbolDropdown">
919. <**option** value="" selected disabled>Select a symbol</**option**>
920. </**select**>
921. </**div**>
923. <!-- Button to Retrieve Data -->
924. <**div** class="col-md-2 mb-3 mt-5">
925. <**button** class="btn btn-primary" onclick="retrieveData()">Retrieve Data</**button**>
926. </**div**>
928. <!-- Button to Update Data -->
929. <**div** class="col-md-2 mb-3 mt-5">
930. <**button** class="btn btn-primary" onclick="updateData()">Update Data</**button**>
931. </**div**>
932. </**div**>
933. </**div**>
934. <!--
935. <!-- Table Start -->
936. <**div** class="container-fluid pt-4 px-4">
937. <**div** class="row g-4">
938. <**div** class="col-12">
939. <**div** class="bg-light rounded h-100 p-4">
940. <**h6** class="mb-4">Responsive Table</**h6**>
941. <**div** class="table-responsive">
942. <**table** class="table" id="dataTable">
943. <**thead**>
944. <**tr**>
945. <**th** scope="col">Date</**th**>
946. <**th** scope="col">Open</**th**>
947. <**th** scope="col">High</**th**>
948. <**th** scope="col">Low</**th**>
949. <**th** scope="col">Close</**th**>
950. </**tr**>
951. </**thead**>
952. <**tbody** id="table-body"></**tbody**>
954. <!-- Data will be populated here -->
955. </**tbody**>
956. </**table**>
957. </**div**>
958. </**div**>
959. </**div**>
960. </**div**>
961. </**div**>
962. <!-- Table End -->

965. <!-- Footer Start -->
966. <**div** class="container-fluid pt-4 px-4">
967. <**div** class="bg-light rounded-top p-4">
968. <**div** class="row">
969. <**div** class="col-12 col-sm-6 text-center text-sm-start">
970. © <**a** href="#">BCA Project on Stock Market using Machine Learning</**a**>,</**br**> All Right Reserved.
971. </**div**>
972. <**div** class="col-12 col-sm-6 text-center text-sm-end">
973. <!--/\*\*\* This template is free as long as you keep the footer author’s credit link/attribution link/backlink. If you'd like to use the template without the footer author’s credit link/attribution link/backlink, you can purchase the Credit Removal License from "https://htmlcodex.com/credit-removal". Thank you for your support. \*\*\*/-->
974. Designed By Nitesh Chavan</**a**>
975. </**br**>
977. </**div**>
978. </**div**>
979. </**div**>
980. </**div**>
981. <!-- Footer End -->
982. </**div**>
983. <!-- Content End -->

986. <!-- Back to Top -->
987. <**a** href="#" class="btn btn-lg btn-primary btn-lg-square back-to-top"><**i** class="bi bi-arrow-up"></**i**></**a**>
988. </**div**>
990. <!-- JavaScript Libraries -->
991. <**script** src="https://code.jquery.com/jquery-3.4.1.min.js"></**script**>
992. <**script** src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.bundle.min.js"></**script**>
993. <**script** src="lib/chart/chart.min.js"></**script**>
994. <**script** src="lib/easing/easing.min.js"></**script**>
995. <**script** src="lib/waypoints/waypoints.min.js"></**script**>
996. <**script** src="lib/owlcarousel/owl.carousel.min.js"></**script**>
997. <**script** src="lib/tempusdominus/js/moment.min.js"></**script**>
998. <**script** src="lib/tempusdominus/js/moment-timezone.min.js"></**script**>
999. <**script** src="lib/tempusdominus/js/tempusdominus-bootstrap-4.min.js"></**script**>
1001. <!-- Template Javascript -->
1002. <**script** src="js/main.js"></**script**>
1003. <**script** src="js/functions.js"></**script**>
1004. <**script** src="js/ticker.js"></**script**>
1005. </**body**>
1007. </**html**>

|  |
| --- |
| **Testing and Evaluation** |

In this section, we discuss the testing and evaluation procedures employed to ensure the reliability, accuracy, and performance of the Stock Market Analysis and Prediction System.

**6.1 Testing Strategies**

Testing strategies are crucial for identifying and resolving potential issues within the system. Key testing strategies include:

* **Unit Testing**: Testing individual components, modules, and functions in isolation to verify their correctness and functionality.
* **Integration Testing**: Testing the integration and interaction between different system components to ensure seamless operation.
* **System Testing**: Testing the system as a whole to evaluate its compliance with functional and non-functional requirements.
* **Regression Testing**: Testing the system after modifications or updates to ensure that existing functionality remains unaffected.
* **User Acceptance Testing (UAT):** Involving end-users to validate the system's usability, effectiveness, and alignment with user expectations.

**6.2 Performance Metrics**

Performance metrics are used to assess the system's efficiency, effectiveness, and scalability. Key performance metrics include:

* **Accuracy**: Measuring the accuracy of prediction models by comparing predicted values with actual market outcomes.
* **Precision and Recall**: Evaluating the precision and recall of classification models for identifying market trends and patterns.
* **Mean Absolute Error (MAE):** Calculating the average absolute difference between predicted and actual values to assess prediction error.
* **Root Mean Square Error (RMSE)**: Measuring the standard deviation of prediction errors to quantify model performance.
* **Computational Efficiency**: Assessing the system's computational efficiency in terms of processing speed, memory usage, and resource consumption.

**6.3 Validation and Verification**

Validation and verification processes ensure that the system meets specified requirements and performs as expected. Key validation and verification activities include:

* **Requirements Validation**: Validating system requirements against user needs, business objectives, and industry standards.
* **Functional Testing**: Verifying that the system functions correctly according to specified requirements and user expectations.
* **Non-Functional Testing**: Evaluating non-functional aspects such as performance, reliability, security, and usability.
* **Model Validation**: Validating prediction models using historical data, cross-validation techniques, and backtesting methods.
* **Peer Review:** Conducting peer reviews and code inspections to identify defects, improve code quality, and ensure compliance with coding standards.

|  |
| --- |
| **Results and Discussion** |

In this section, we present the results of the Stock Market Analysis and Prediction System, followed by a discussion of the findings and their implications.

**7.1 Analysis of Historical Data**

The analysis of historical market data revealed several insights into past trends, patterns, and behaviors of the stock market. By analyzing historical price movements, volume trends, and other relevant indicators, the system identified recurring patterns and market dynamics that influenced stock prices. This analysis provided valuable context for developing predictive models and understanding market behavior under different conditions.

**7.2 Predictive Model Performance**

The predictive models implemented within the system demonstrated promising performance in forecasting future market trends and price movements. Through rigorous testing and validation, the models exhibited favorable accuracy, precision, and reliability in predicting stock prices over various time horizons. Performance metrics such as mean absolute error (MAE), root mean square error (RMSE), and accuracy rates were used to evaluate and compare the performance of different prediction models. Overall, the predictive models proved to be effective tools for generating actionable insights and supporting decision-making in the stock market.

**7.3 User Feedback**

User feedback played a crucial role in assessing the usability, effectiveness, and value proposition of the Stock Market Analysis and Prediction System. Feedback from users, including investors, traders, and financial professionals, provided valuable insights into the system's strengths, weaknesses, and areas for improvement. Users appreciated the system's intuitive interface, real-time updates, and accurate predictions, which helped them make informed investment decisions. Constructive feedback and suggestions for enhancements were also received, guiding future iterations and improvements of the system.

**Discussion**

The results obtained from the analysis of historical data and the performance of predictive models underscore the potential of the Stock Market Analysis and Prediction System as a valuable tool for investors, traders, and financial analysts. By leveraging advanced analytics and machine learning techniques, the system offers actionable insights and decision support for navigating the complex and volatile stock market environment. However, challenges such as market volatility, data quality issues, and model accuracy remain areas of concern that warrant further research and development efforts. Overall, the system represents a significant advancement in stock market analysis and prediction, with the potential to revolutionize decision-making processes in the financial industry.

|  |
| --- |
| **Conclusion** |

This section provides a summary of the findings, achievements, contributions, and outlines areas for future work in the Stock Market Analysis and Prediction System.

**8.1 Summary of Findings**

The Stock Market Analysis and Prediction System have demonstrated the capability to analyze historical market data, develop predictive models, and generate actionable insights for investors and traders. Through the analysis of historical data, the system identified significant trends, patterns, and market dynamics that influence stock prices. The predictive models exhibited favorable performance metrics, indicating their effectiveness in forecasting future market trends and price movements. Overall, the system has shown promise as a valuable tool for decision-making in the stock market.

**8.2 Achievements and Contributions**

The development and implementation of the Stock Market Analysis and Prediction System represent a significant achievement in the field of financial technology. The system's contributions include:

* Providing investors and traders with valuable insights into market trends and price movements.
* Offering predictive models that facilitate informed decision-making and risk management.
* Enhancing the efficiency and effectiveness of stock market analysis through automation and advanced analytics.
* Fostering innovation in the financial industry by leveraging machine learning and data science techniques.

**8.3 Future Work**

While the Stock Market Analysis and Prediction System have achieved notable success, there are several avenues for future research and development:

* Further refinement and optimization of predictive models to improve accuracy and robustness.
* Integration of additional data sources and variables to enhance the predictive power of the system.
* Development of advanced analytics features such as sentiment analysis, news sentiment, and social media analysis to capture market sentiment and investor behavior.
* Expansion of the system's capabilities to cover other financial markets and asset classes, such as commodities, currencies, and cryptocurrencies.
* Collaboration with industry partners and stakeholders to validate the system's effectiveness in real-world trading environments and refine user interfaces based on feedback.

In conclusion, the Stock Market Analysis and Prediction System represent a significant advancement in the field of financial technology, offering innovative solutions for analyzing and predicting stock market trends. By leveraging cutting-edge technologies and methodologies, the system has the potential to revolutionize decision-making processes in the financial industry and empower investors and traders with actionable insights for achieving their financial goals.