Web-based Stock Price Projection using Deep Learning and Django

A Project Report submitted for the partial fulfilment of the requirement for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

SUBMITTED BY

PURAM NAGENDRA

Roll No: 22691F0096

Under the Guidance of

Dr. N. Naveen Kumar M.C.A, Ph. D

Associate Professor

DEPARTMENT OF COMPUTER APPLICATIONS

MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

MADANAPALLE

UGC AUTONOMOUS

(Approved by AICTE, New Delhi & affiliated to JNTUA, Ananthapuramu)

2023-2024

MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

MADANAPALLE

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu)

DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

This is to certify that the project work entitled "Web-based Stock Price Projection using Deep Learning and Django" is a bonafide work carried by PURAM NAGENDRA (22691F0096), submitted in the partial fulfilment of the requirements for the award of the degree of Master of Computer Applications in Madanapalle Institute of Technology & Science, Madanapalle, affiliated to Jawaharlal Nehru Technological University Anantapur, Ananthapuramu during the academic year 2023-2024.

PROJECT GUIDE

Dr. N. Naveen Kumar M.C.A, Ph. D

Associate Professor

HEAD OF THE DEPARTMENT

Dr. N. Naveen Kumar M.C.A, Ph. D

Associate Professor

Submitted for viva-voce examination held on _____

Internal Examiner

Date:

External Examiner

Date:

DECLARATION

I, Puram Nagendra (Roll No. 22691F0096) hereby declare that the project entitled "Web-based Stock Price Projection using Deep Learning and Django" is done by me, under the guidance of **Dr. N. Naveen Kumar M.C.A, Ph. D.,** submitted in partial fulfilment of the requirements for the award of degree for Master of Computer Applications at MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE Madanapalle, affiliated to Jawaharlal Nehru Technological University Ananthapur, Ananthapuramu during the academic year 2023-2024. This work has not been submitted by anybody towards the award of any degree.

Date: Signature of the Student

Place: Madanapalle

ACKNOWLEDGEMENT

First, I must thank the almighty who has granted me knowledge, wisdom, strength and courage to server in this world and to carry out my project work in a successful way.

Next, I have to thank my parents who encouraged me to undergo this MCA course and do this project in a proper way.

I express my sincere thanks to **Dr. N. Vijaya Bhaskar Chowdary Ph.D.,** Secretary & Correspondent, Madanapalle Institute of Technology & Science for his continuous encouragement towards practical education and constant support in all aspects which includes the provision of very good infrastructure facilities in the institute.

It is my duty to thank our Principal, **Dr. C. Yuvaraj M.E, Ph.D.,** Madanapalle Institute of Technology & Science, for his guidance and support at the time of my course and project.

I also wish to express my thanks to our vice principal, **Dr. P. Ramanathan Ph.D.**, for his continuous support throughout our MCA course.

It is my foremost duty to thank Head of the Department **Dr. N. Naveen Kumar M.C.A., Ph.D.,** who gave me constant support during the project time and continuous encouragement towards the completion of the project successfully.

I thank my faculty guide **Dr. N. Naveen Kumar M.C.A., Ph.D.,** for his continuous support by conducting periodical reviews and guidance until the completion of the project in a successful manner.

Puram Nagendra

(22691F0096)

ABSTRACT

The financial industry has seen a transformative impact from technological advancements, particularly in the domain of stock price forecasting. This major project presents a sophisticated system for web-based stock price projection utilizing deep learning methodologies, implemented within the Django web framework. The objective is to provide an accessible platform for investors and financial analysts to obtain accurate, real-time stock price predictions, thus facilitating more informed investment decisions.

The core of the system is built upon Gated Recurrent Unit (GRU) networks, renowned for their proficiency in handling time-series data. The GRU model is trained on extensive historical stock prices and relevant financial indicators to predict future stock prices with high accuracy. The model is designed to update continuously with new market data, ensuring its predictive capabilities remain robust and reliable.

Django, a high-level Python web framework, is utilized to develop the system's front-end. Its powerful features and scalability make it an ideal choice for integrating complex machine learning models into a web-based application. This integration ensures efficient data handling and real-time updates, providing a seamless user experience.

The web interface allows users to input specific stock symbols and receive projected price trends. It also includes advanced visualization tools that display historical and predicted stock performance, enhancing the interpretability of the data. The system's architecture is designed to be scalable, allowing for the easy addition of new stocks and financial indicators as needed.

This project demonstrates how deep learning and web development can work together to create practical tools for financial forecasting and investment planning. It highlights the potential of modern technology to improve financial analysis and decision-making.

CONTENTS

List of Tables

List of Figures

List of Abbreviations

CHAPTER-1
INTRODUCTION

1.1 About the Project

The project "Web-Based Stock Price Projection Using Deep Learning and Django" is an state-of-the-art initiative aimed at providing an open, user-friendly platform for predicting future stock prices. The stock market is known for its complication and volatility, influenced by a countless of economic, political, and psychological factors. Accurately forecasting stock prices is a significant challenge that has interested investors and researchers. This project leverages the power of deep learning, a subset of machine learning that outshines in identifying patterns in large datasets. Precisely, it utilizes GRU networks, which is designed to handle sequential data and is particularly effective for time series forecasting. By analyzing historical stock price data, this model can generate predictions about future price movements, offering valuable insights for investors.

To make these sophisticated predictions accessible to users, the project employs Django, a high-level Python web framework known for its robustness and scalability. Django facilitates the creation of a web-based interface where users can input their data, run predictive models, and visualize results in an intuitive manner. The platform aims to present predictions through interactive charts and graphs, ensuring that even users without a deep understanding of machine learning can interpret and act on the insights provided. Furthermore, the system is designed to be scalable, capable of handling large volumes of data and multiple users simultaneously, ensuring reliability and performance under varying load conditions.

The integration of deep learning and web technology in this project addresses several key objectives: providing accurate stock price predictions, ensuring ease of use through a web-based interface, and presenting data in a clear, visual format. This approach not only enhances the user experience but also allows access to advanced predictive tools, empowering individual investors and small trading firms with technology typically reserved for large financial institutions. By linking the gap between complex deep learning models and user-friendly web applications, this project stands to make a significant impact on the way investors analyze and predict market trends, ultimately contributing to more informed and strategic investment decisions.

1.2 Literature Survey

Stock price prediction has been a critical area of research in finance, driven by the potential financial benefits of accurately forecasting market trends. With the arrival of deep learning, models such as LSTM and GRU flavors of RNN have become important due to their ability of handling sequential data effectively.

A variety of studies have explored stock price prediction using various methods. The use of machine learning algorithms, highlighting the effectiveness of Random Forest Regression and also focusing on the use of LSTM [1][2]. A different approach was done using news sentiment analysis to improve prediction accuracy [3]. While another, combined technical, fundamental, and news-based methods to detect stock

price fluctuations [4]. These studies collectively demonstrate the potential of machine learning and data analysis techniques in enhancing stock price prediction.

Another classical machine learning model, linear regression is used stock price prediction, with varying degrees of success. Another research found that linear regression, when combined with sentiment analysis and historical stock data, respectively, can be effective in predicting stock prices [5][6]. However, another noted limitations, such as the model's inability to capture complex relationships and its performance in volatile markets [7][8]. Despite these limitations, linear regression remains a popular tool in stock price prediction, particularly when combined with other data sources and analysis techniques.

A Research observed, that LSTM outperformed other machine learning techniques, reporting an R2 value of 0.99 and achieving a small error rate [9][10]. Another paper also found promising results, reporting an average accuracy of 55.9% in predicting stock price movements and achieving competitive performance compared to traditional statistical models and other machine learning approaches. These findings collectively suggest that LSTM models are a promising tool for stock price prediction [11][12].

A range of studies have explored the use of GRU in stock price prediction, with promising results by introduction of a 3-way GRU architecture that outperformed other models [13]. A hybrid CNN-GRU model that also showed superior performance [14]. Further optimizing the RNN-GRU architecture, finding that a stacked structure and technical analysis data yielded the best results [15]. A completely different approach, integrating the GRU model with the HP filter to improve prediction accuracy [16]. These studies collectively highlight the potential of GRU in stock price prediction, with various architectural and methodological enhancements.

Stock price prediction is a challenging task due to the complex and confused nature of stock data [17]. Various models and techniques have been proposed to address this challenge, ARIMA and ANNs [18]. Also, Support Vector Machine, Random Forest, Linear Regression, Recursive Neural Network, and Long Short-Term Movement. However, the stock market's random walk nature and the gathering of variables involved make accurate prediction difficult [19]. The use of machine learning techniques and algorithms, such as those mentioned above, shows promise in addressing these challenges. Despite these efforts, the complexity of stock market forecasting persists, with the need for consideration of additional factors [20].

The integration of GRU models in stock price prediction, along with the use of Django for web development, presents a powerful combination. GRU models have shown promising results in predicting stock price movements due to their ability to handle sequential data effectively. When integrated into a Django-based web platform, users can access these predictive insights conveniently through a user-friendly interface, enhancing the platform's usability and value proposition.

CHAPTER-2

SYSTEM ANALYSIS

2.1. Existing System

Traditional Statistical Methods:

Time Series Analysis: Methods such as ARIMA which predict future values based on past values.

Linear Regression: Simple linear models to forecast stock prices based on historical data.

Machine Learning Approaches:

Support Vector Machines (SVM): Used for classification and regression challenges.

Random Forests: An ensemble learning method that improves prediction accuracy.

Deep Learning Methods:

Recurrent Neural Networks (RNN): Specifically designed for sequential data, capturing temporal dependencies.

Long Short-Term Memory Networks (LSTM): A type of RNN that can learn long-term dependencies.

Financial News Websites:

Yahoo Finance, Google Finance: Provide historical data, real-time stock prices, and basic forecasting tools.

2.2. Disadvantages of Existing System:

Accuracy: Traditional and some machine learning models often do not capture the complexity of stock market data.

Accessibility: Advanced tools and platforms can be too complex for non-expert users.

Integration: Lack of integrated platforms that combine advanced deep learning models with user-friendly web interfaces.

Real-time Processing: Many systems do not offer real-time data processing and predictions, which is crucial for stock trading.

2.3. Proposed System:

The proposed system for the project titled "Web-Based Stock Price Projection Using Deep Learning and Django" aims to create a robust, accurate, and user-friendly platform for predicting stock prices. This system will leverage the power of deep learning algorithms to analyze historical stock data and generate future price projections. The architecture of the system is divided into several key components: the frontend, backend, and machine learning model. The frontend has been developed using modern web technologies like HTML, CSS, and JavaScript frameworks, ensuring a responsive and intuitive user interface. This interface will facilitate user interactions, allowing them to input stock symbols, view predictions, and analyze historical data.

The backend will be driven by Django, a high-level Python web framework that encourages rapid development and clean, pragmatic design.

The heart of the system lies in its deep learning model. The project explores and implements advanced deep learning algorithms, specifically focusing on Gated Recurrent Unit (GRU) networks, which is well-

suited for time series forecasting tasks such as stock price prediction. The process will begin with data collection from reliable financial data source like Yahoo Finance. The collected data will undergo rigorous cleaning and preprocessing to handle missing values, normalize data, and create appropriate input sequences for the model.

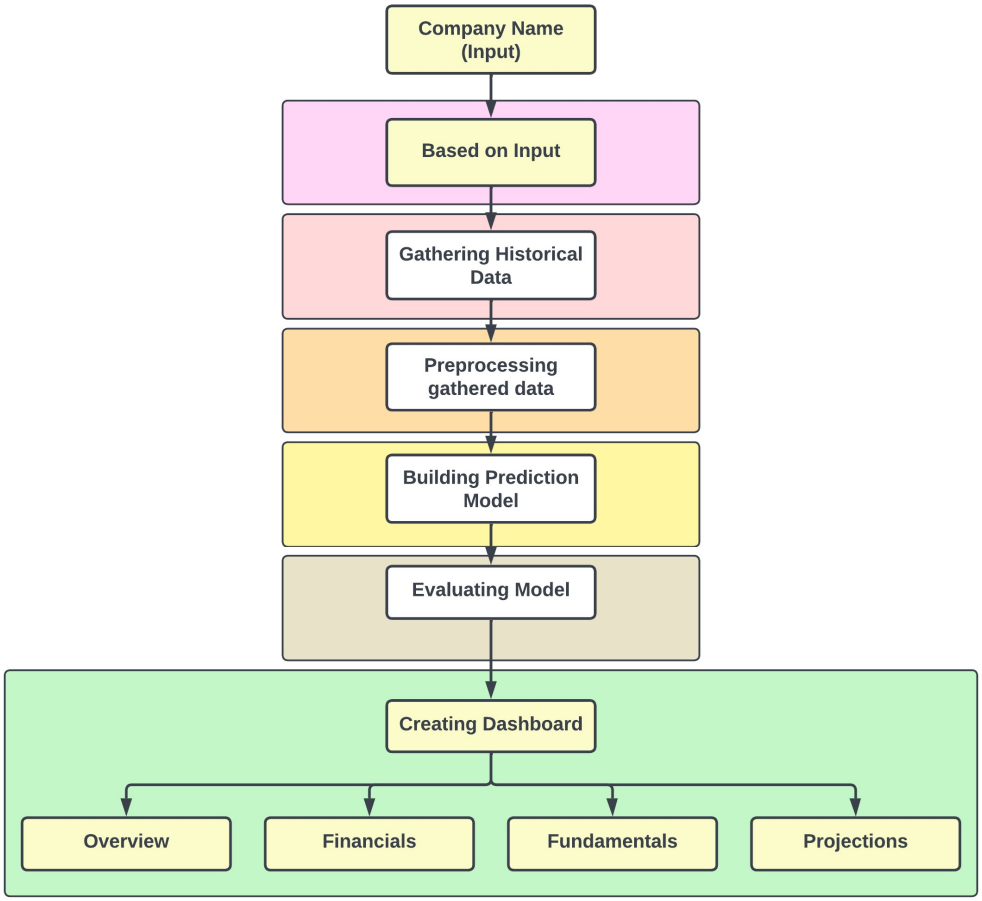


Fig 2.3.1 Workflow of Proposed System

Model development will involve building and training the deep learning models on historical stock data. Key evaluation metrics such as MAE and RMSE will be used to assess model performance. Hyperparameter tuning will be carried out to optimize the model's accuracy and predictive capabilities. The final model will be integrated into the Django backend, with mechanisms to periodically retrain and update the model with new data to maintain prediction accuracy.

The proposed system aims to provide users with a seamless experience in stock price prediction. Users will be able to register and log into the platform, access a dashboard displaying current stock prices, historical trends, and projected prices based on the latest model predictions. The intuitive design will allow users to easily input stock symbols, select prediction timeframes, and view results in various formats, including charts and tables.

In summary, this project will combine the strengths of deep learning and web development frameworks to build a comprehensive stock price projection system. By integrating GRU model with a robust Django backend and a user-friendly frontend, the system will offer accurate, real-time stock price predictions. This will empower investors, analysts, and enthusiasts with the insights needed to make informed financial decisions, ultimately bridging the gap between complex financial data analysis and everyday users.

2.4. Advantages of Proposed System

Advanced Predictive Models: By leveraging deep learning techniques, the system aims to offer more accurate stock price projections compared to traditional and simple machine learning methods.

Ease of Use: The web-based interface designed with Django will ensure that users can easily interact with the predictive models without needing extensive technical knowledge.

Real-time Predictions: Incorporate APIs and data streams to provide real-time stock price predictions, which are essential for making timely trading decisions.

Dashboard: For better and user-friendly visualization using plotly helps in easy inderstanding.

2.5. Hardware & Software Requirements

Hardware Requirements

- Processor: Intel Core i3 or above
- RAM: Minimum 8 GB
- Storage: SSD with at least 256 GB storage or 500GB Hard Disk
- Network: High-speed internet connection

Software Requirements

- Operating System: Windows 8/10/11
- Application Server: WSGI for serving the Django application
- Front-end: HTML, CSS, JavaScript
- Back-end: Python 3.8 or higher
- Web Framework: Django 3.2 or higher
- Machine Learning Libraries: TensorFlow 2.x
- Data Processing Libraries: Pandas, NumPy
- Visualization Libraries: Matplotlib, Plotly
- Version Control: Git

2.6. Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- OPERATIONAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client.

OPERATIONAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER-3

SYSTEM DESIGN

3.1. Input Design

Input design for the system is crucial for ensuring seamless interactions between users and the system. This involves developing specifications and procedures for preparing transaction data in a usable form for processing.

Things to be considered:

- What data should be given as input?
- How the input needs to be taken from user?
- How the data should be arranged or coded?
- Guiding user on how to select symbol based on dropdown menu.
- Creating user-friendly interface for better and easy understanding.
- Methods for preparing input validations and steps to follow when error occur.

3.2. Objectives

The primary objective of this system is to provide, analysis and predicting the performance of companies based on historical data. Below are the detailed objectives:

- **Data Accuracy and Integrity:**

  One of the foremost objectives is to ensure the accuracy and integrity of the data used in the analysis. This involves rigorous validation and cleaning processes to remove any inconsistencies or errors in the historical data collected. Accurate data forms the foundation of reliable analysis and predictions, which are crucial for making informed business decisions.

- **Comprehensive Data Collection:**

  The system aims to gather extensive historical data encompassing financial statements, market data, news articles, and other relevant information. By consolidating diverse data sources, the system provides a holistic view of the company's past performance and context, which is essential for thorough analysis.

- **Effective Data Preprocessing:**

  Another key objective is to preprocess the gathered data efficiently. This includes tasks like normalizing data, handling missing values, and creating derived metrics that are essential for accurate modeling.

- **Advanced Predictive Modeling:**

  The system seeks to utilize sophisticated algorithms, such as the Gated Recurrent Unit (GRU), to build robust predictive models. These models aim to identify patterns and trends in historical data to forecast future performance. By employing advanced machine learning techniques, the system can provide more accurate and reliable predictions, which are critical for strategic planning and risk management.

- **User-Friendly Dashboard Interface:**

  A major objective is to present the analysis results through an interactive and user-friendly dashboard. The dashboard aims to provide clear and intuitive visualizations of the data, enabling users to quickly grasp key insights and trends. It should facilitate easy navigation and interaction, allowing users to filter data, drill down into specifics, and compare different metrics and scenarios seamlessly.

- **Accessibility and Scalability:**

  Ensuring that the system is accessible to a wide range of users, regardless of their technical expertise, is a critical objective. The interface should be intuitive and straightforward, providing value to financial analysts, business managers, and other stakeholders. Additionally, the system should be scalable to handle large volumes of data and support multiple users simultaneously, accommodating the needs of growing organizations.

3.3. Output Design

Dashboard Overview

→ Pie Chart for Mutual Fund Holders

Display a visually appealing pie chart depicting the distribution of mutual fund holders by percentage. Each segment represents a different fund or category of holders, making it easy for users to grasp the overall composition.

→ Table for Financial Data and Fundamental Ratios

Present a comprehensive table that includes key financial data such as revenue, expenses, earnings per share (EPS), net income, debt-to-equity ratio, and other fundamental ratios like price-to-earnings (P/E) ratio, price-to-book (P/B) ratio, and return on equity (ROE).

→ Projection for Future Prices

3.4. Modules Description

Complete project development is divided into six modules:

- Company input and Validation
- Data Collection and Gathering
- Data Preprocessing
- Model Building
- Evaluation
- Dashboard Creation

1. Company Input and Validation

This initial module serves as the entry point for the entire analysis process. The user inputs the company name they wish to analyze. This step includes:

- Input: The system captures the company name input from the user.

- Validation: The input is checked for validity. This may involve ensuring that the company exists, confirming the correct spelling, and retrieving the correct company's profile if multiple companies have similar names.
- Initial Data Retrieval: Basic information about the company, such as its industry, size, and key identifiers, is retrieved to ensure the subsequent data gathering process is accurate and relevant.

2. **Data Collection and Gathering**

   Once the company is validated, this module focuses on collecting comprehensive historical data. This includes:

   - Historical Financial Data: Gathering data from financial statements like income statements, balance sheets, and cash flow statements over several years.
   - Market Data: Collecting stock prices, trading volumes, and market capitalization data.

3. **Data Preprocessing**

   Collected data often comes in various formats and might have inconsistencies. This module ensures the data is clean and ready for analysis:

   - Data Cleaning: Removing duplicates, handling missing values, correcting errors, and ensuring data consistency.
   - Data Transformation: Converting data into a uniform format. For example, normalizing numerical values, encoding categorical variables, and ensuring all time-series data are synchronized.
   - Feature Engineering: Creating new variables that might help in the analysis. This might include ratios, growth rates, or any other derived metrics.
   - Data Segmentation: Splitting data into training and testing sets if predictive modeling will be used, ensuring that the model can be validated effectively.

4. **Predictive Model Building**

   In this module, the GRU (Gated Recurrent Unit) algorithm, is employed to analyze the preprocessed data and build predictive models. A GRU unit usually has similar or better performance to a LSTM, but it does so with fewer parameters and operations.

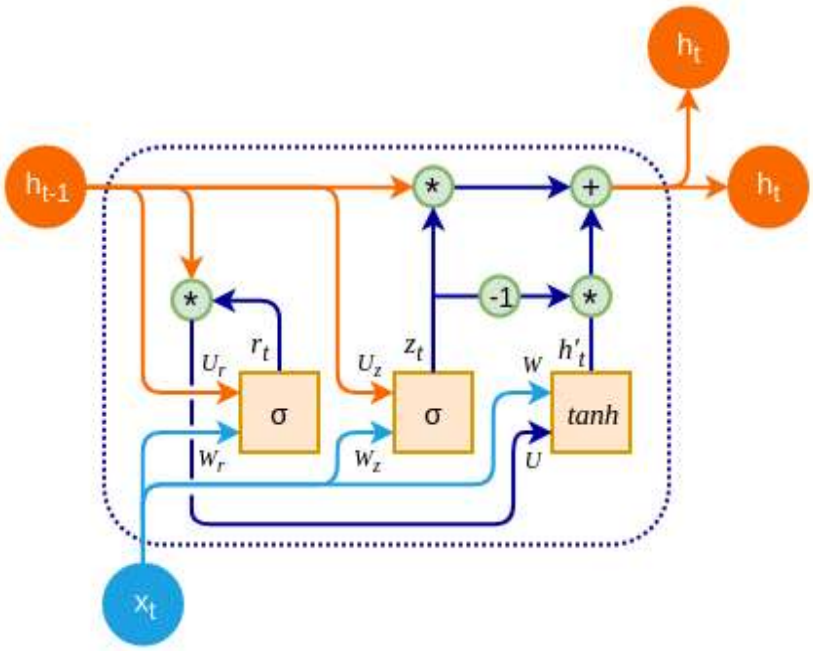


Fig 3.4.4.1 Architecture of GRU Cell

The GRU Cell has two gates:

- The update gate Z_t is like a mix of the input and forget gates in an LSTM. It figures out what old information to get rid of and what new information to add. It uses the network's current input Z_t and the previous hidden state h_{t-1} to make this decision. By combining these two gates, it ensures that old information is only discarded when new information is ready to take its place.

$$Z_t = \sigma(W_z x_t + U_z h_{t-1})$$

- A reset gate, r_t which uses the previous cell state h_{t-1} and the network input x_t to decide how much of the previous state to pass through.

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

The GRU Model Steps:

- Model Selection: Choosing the appropriate algorithm for the task. GRUs are particularly good for time-series data due to their ability to handle sequences and retain long-term dependencies.
- Model Training: Feeding the historical data into the model to learn patterns. This involves adjusting the model's parameters to minimize prediction errors.
- Model Tuning: Fine-tuning the model's hyperparameters to optimize performance. This may involve techniques like cross-validation.
- Model Validation: Evaluating the model's performance using a subset of the data not seen during training.

5. **Evaluation and Insights Generation**

   Once the model is built and validated, this module focuses on interpreting the results and generating actionable insights:

   - Model Evaluation: Continuous monitoring of the model's performance to ensure it remains accurate over time. This might involve updating the model with new data as it becomes available.
   - Insight Extraction: Translating model outputs into meaningful insights. This could include identifying trends, forecasting future performance, and highlighting potential risks or opportunities.

6. **Dashboard Creation**

   The final module involves presenting the analysis in a user-friendly manner through a comprehensive dashboard:

   - Dashboard Design: Creating a visually appealing and intuitive interface. This might involve using tools like Tableau, Power BI, or custom web-based dashboards.
   - Visualization: Implementing charts, graphs, and tables to display data. Common visualizations might include time-series plots, bar charts, pie charts, and heat maps.
   - Overview: Summarizes the most critical insights and metrics at a glance.
   - Financials: Provides detailed financial performance data, including revenue, expenses, profits, and key financial ratios.
   - Fundamentals: Displays fundamental analysis such as market position, management quality, competitive landscape, and industry trends.
   - Projections: Shows forecasted data and future trends based on the predictive model. Includes different scenarios and their potential impact.

3.5. UML Diagrams

3.5.1 Activity Diagram

The activity diagram illustrates the systematic workflow from initial input to final outputs. It begins with inputting company data and progresses through data gathering, preprocessing, model building, and evaluation. After creating a dashboard, the process divides into simultaneous tasks—creating an overview, analyzing financials, assessing fundamentals, and making projections—reflecting parallel activities. Once these tasks are completed, the diagram merges the results back into a unified flow, culminating in the comprehensive analysis of the company. This diagram effectively maps out the structured steps and concurrent activities involved in performing a detailed company analysis.

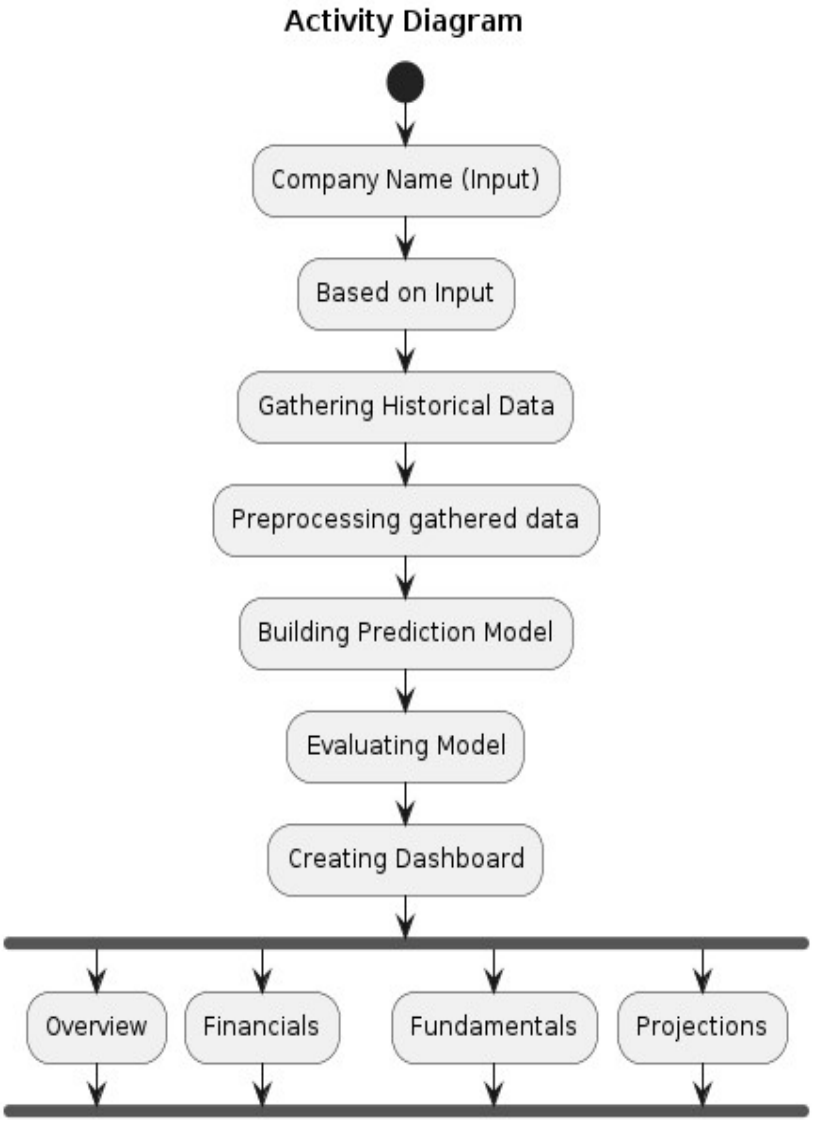


Fig 3.5.1.1 Activity Diagram

3.5.2 Use Case Diagram

The use case diagram outlines how users interact with the system to conduct comprehensive analysis tasks. Users initiate by inputting the company name, triggering the system to gather historical data, preprocess it, build a prediction model, evaluate its accuracy, and generate a dashboard. Primary actors, like users, interact with these system functionalities to achieve specific goals. Relationships like includes and extends depict dependencies and optional functionalities within the analysis process, ensuring a clear representation of how the system supports user-driven tasks and advanced analytical capabilities.

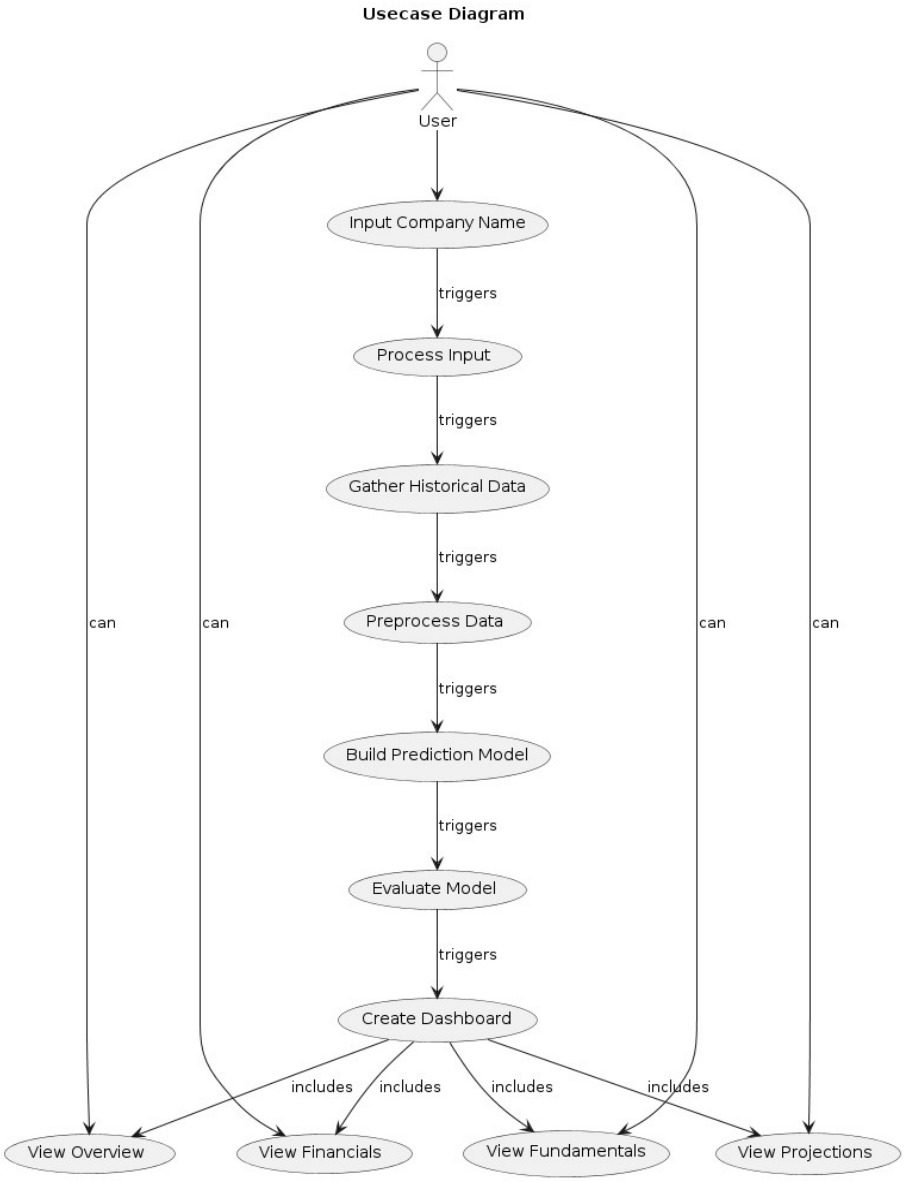


Fig 3.5.2.1 Usecase Diagram

3.5.3 Sequence Diagram

The sequence diagram illustrates a structured sequence of interactions between the User and system components. It begins with the User inputting the company name, followed by sequential actions including data gathering, preprocessing, model building, evaluation, and dashboard creation. Each component communicates through messages, depicting the flow of data and control. Parallel activities, such as creating the Overview, Financials, Fundamentals, and Projections in the dashboard, are synchronized within the diagram's timeline. This visualization effectively captures the dynamic exchange of information and tasks between participants, ensuring a clear understanding of the systematic process involved in conducting a comprehensive company analysis.

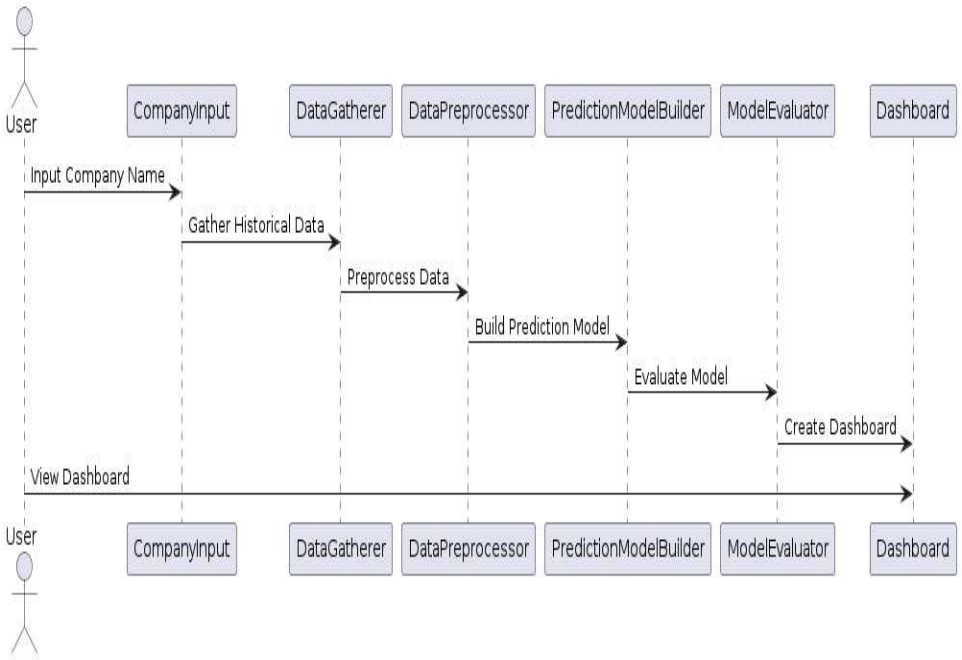


Fig 3.5.3.1 Sequence Diagram

3.5.4 Deployment Diagram

The deployment diagram for the system outlines the physical arrangement of software components and hardware nodes. User requests originate from the User's Device and are processed by the Web Server's Web Application, which interacts with the Database Server for data operations and the Prediction Server for predictive analytics. The Dashboard Server retrieves outputs from specialized modules (Overview, Financials, Fundamentals, Projections) hosted on dedicated servers. This diagram clarifies how components are deployed across servers, ensuring that data processing, analytics, and user interface functionalities are efficiently supported by appropriate hardware resources.

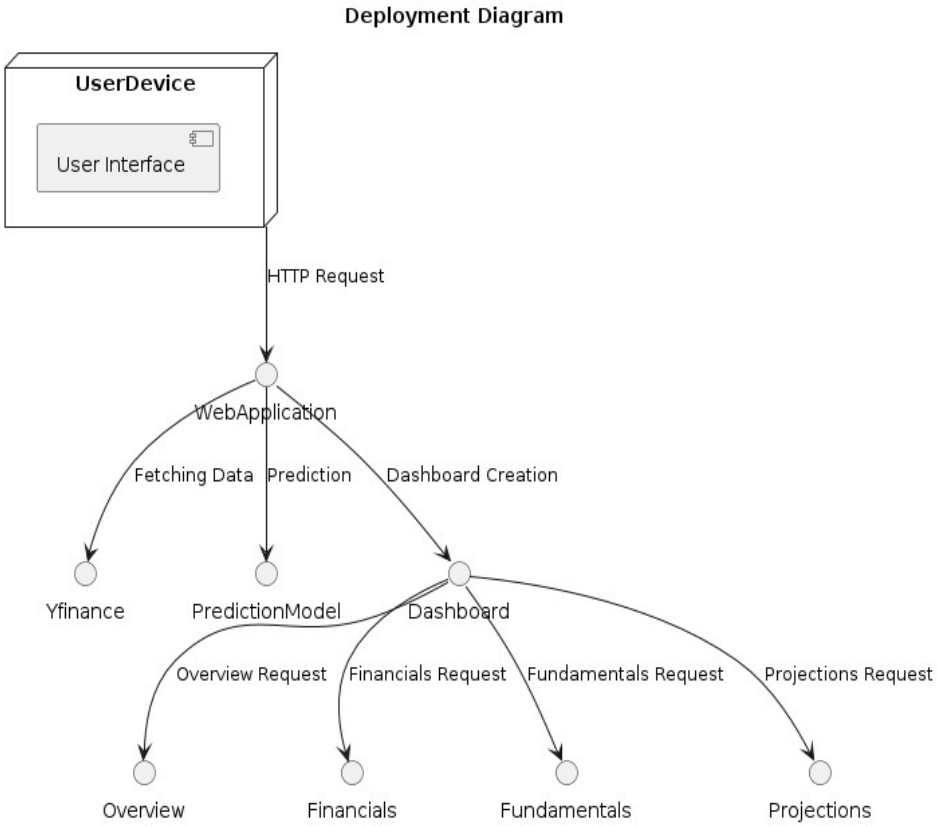


Fig 3.5.4.1 Deployment Diagram

3.5.5 Collaboration Diagram

The collaboration diagram for the system illustrates the flow of communication and interactions between different system components. It begins with user input and progresses through data gathering, preprocessing, model building, evaluation, and dashboard creation. Each component communicates effectively with others, exchanging necessary data and results to advance the analysis. The diagram emphasizes the coordinated effort among components to generate an integrated dashboard and subsequent analytical outputs such as overview, financials, fundamentals, and projections. This visual representation clarifies how each component collaborates within the system to achieve comprehensive company analysis, enhancing understanding of the process flow and information exchange.

3.5.6 Class Diagram

The class diagram for the system outlines its foundational components and relationships. The Company class represents the analyzed entity, linked to "DataGatherer" for historical data retrieval. "DataProcessor" prepares data for "PredictionModel", which builds and evaluates predictive models. The Dashboard class aggregates interfaces (Overview, Financials, Fundamentals, Projections) for presenting analytical insights. Associations show how entities collaborate, dependencies clarify operational sequences, and compositions highlight the dashboard's modular structure. This diagram provides a blueprint of the system's static architecture, detailing classes, their attributes, methods, and interactions crucial for performing comprehensive company analysis tasks.

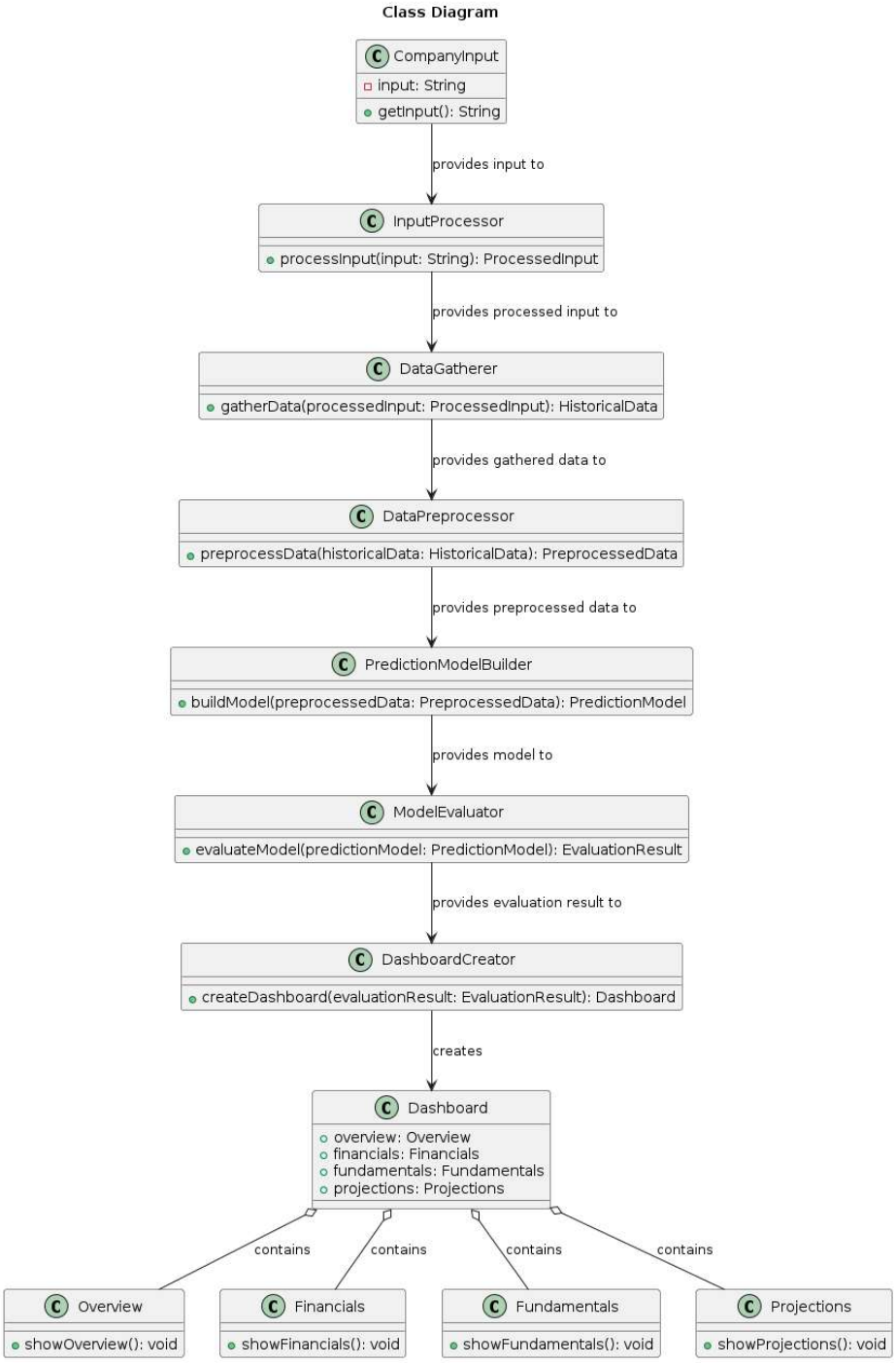


Fig 3.5.6.1 Class Diagram

3.5.7 Component Diagram

The component diagram for the company analysis system depicts its architectural components and their interactions. The User Interface initiates requests handled by the Web Application, which in turn interacts with the Database for data retrieval and storage. The Prediction Model component builds and evaluates models, while the Dashboard component aggregates and presents analysis results to users. The system is modularized with distinct components like Overview, Financials, Fundamentals, and Projections Modules, each focusing on specific analytical tasks. This diagram effectively illustrates how these components collaborate to facilitate comprehensive company analysis, enhancing decision-making through structured data processing and visualization.

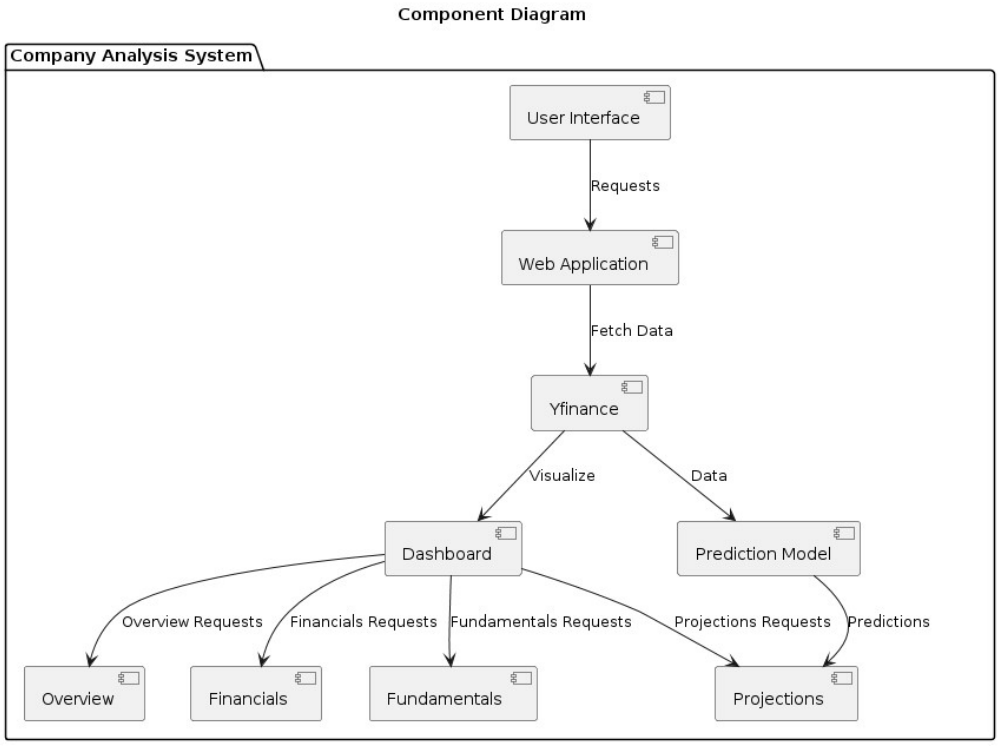


Fig 3.5.7.1 Component Diagram

CHAPTER-4

SYSTEM IMPLEMENTATION

4.1. Algorithms

The Gated Recurrent Unit (GRU) algorithm is a type of neural network architecture designed for processing sequential data, such as time-series data or natural language. It belongs to the family of recurrent neural networks (RNNs) and is renowned for its efficiency in capturing dependencies over long sequences while mitigating some of the issues like the vanishing gradient problem found in traditional RNNs.

At its core, the GRU architecture comprises a set of gating mechanisms that regulate the flow of information within the network. These mechanisms enable GRU to selectively update and access information over time, making it adept at modeling sequential dependencies. The key components of a GRU unit include:

1. Update Gate: This gate controls how much of the past information should be preserved and passed along to the current time step. It computes a vector that determines the relevance of the previous hidden state to the current state.

2. Reset Gate: The reset gate decides how much of the past information should be forgotten or reset. It calculates another vector to decide which parts of the previous hidden state should be ignored in computing the new state.

3. Current Memory Content: This is the combination of the update and reset gates' output, which controls the flow of information from past to future states. This content represents the current state of the unit, combining both the past and present information.

During training, the GRU algorithm learns the parameters that optimize the prediction or classification task at hand. This learning process involves adjusting the weights associated with each gate and state transition, guided by a loss function that quantifies the model's performance against the actual outputs.

Applications of GRU extend across various domains, including natural language processing (NLP), where it excels in tasks such as language modeling, machine translation, and sentiment analysis. In time-series analysis, GRU models are utilized for predicting future values based on historical data, such as forecasting stock prices, weather patterns, or patient health trends in healthcare.

In summary, the GRU algorithm represents a significant advancement in the realm of sequential data processing within neural networks. Its ability to selectively retain and update information over time, facilitated by gating mechanisms, makes it a versatile tool for tasks requiring memory and context preservation across long sequences. As research continues to refine and expand upon its capabilities, GRU remains a pivotal component in the development of sophisticated AI systems capable of understanding and predicting complex patterns in sequential data.

4.2. Language Selection

Introduction to Python

Python is a high-level programming language known for its simplicity, versatility, and readability. Created by Guido van Rossum and first released in 1991, Python has since become one of the most popular and widely used programming languages worldwide. It is renowned for its ease of learning and adoption, making it accessible to beginners while also powerful enough to handle complex applications in various domains.

Python's design philosophy emphasizes readability and clarity of code, making it easier to write and maintain compared to many other languages. It uses a clean and straightforward syntax, with significant whitespace and indentation playing a crucial role in structuring code blocks. This feature not only enhances readability but also encourages good coding practices among developers.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming styles. This versatility allows developers to choose the approach that best suits their project requirements and coding preferences. As an interpreted language, Python does not require compilation before execution, enabling rapid development and testing cycles.

What Can Be Achieved with Python

i. Web Development:

Python is widely used for web development, powering popular frameworks like Django and Flask. Django, a high-level web framework, simplifies the creation of complex web applications by providing built-in features for authentication, URL routing, and database management. Flask, on the other hand, is a lightweight framework ideal for smaller-scale applications and APIs.

ii. Data Science and Machine Learning:

Python has gained significant traction in the fields of data science and machine learning due to its rich ecosystem of libraries and tools. Libraries such as NumPy, Pandas, and SciPy provide powerful data manipulation and analysis capabilities. For machine learning and AI tasks, frameworks like TensorFlow and PyTorch enable the development of sophisticated models for tasks like image recognition, natural language processing, and predictive analytics.

iii. Scientific Computing:

Python is widely used in scientific computing and research environments. Libraries like Matplotlib and Seaborn facilitate data visualization, while SciPy provides functions for numerical integration, optimization, and signal processing. These tools make Python an excellent choice for researchers and scientists looking to analyze and visualize complex datasets.

iv. Desktop GUI Applications:

Python supports GUI (Graphical User Interface) development through frameworks such as PyQt and Tkinter. These frameworks allow developers to create cross-platform desktop applications with rich graphical interfaces, enhancing user interaction and experience.

v. Scripting and Automation:

Python's simplicity and ease of integration with other languages and systems make it ideal for scripting and automation tasks. It is commonly used for writing scripts to automate repetitive tasks, manage system configurations, or handle file operations.

vi. Game Development:

Python, coupled with libraries like Pygame, is used in game development for prototyping, scripting game logic, and creating interactive experiences. While not as performance-focused as some lower-level languages, Python's ease of use and rapid development capabilities make it suitable for certain types of games and simulations.

vii. Server-Side Development:

Python is utilized for server-side development, powering applications and APIs through frameworks like FastAPI and Django REST framework. These frameworks facilitate the creation of robust backend systems, handling HTTP requests, data processing, and interfacing with databases efficiently.

Introduction to Django

Django is a high-level web framework written in Python that encourages rapid development and clean, pragmatic design. It is open-source and free, maintained by the Django Software Foundation. Since its inception in 2005, Django has gained immense popularity for its simplicity, versatility, and robustness in building web applications of various scales and complexities.

At its core, Django follows the "Don't Repeat Yourself" (DRY) principle, aiming to reduce redundancy and promote efficiency in web development. It provides a set of tools and conventions that streamline the creation of web applications, allowing developers to focus more on writing code that implements business logic rather than boilerplate setup.

One of the key features of Django is its powerful Object-Relational Mapping (ORM) system. This system allows developers to define database models as Python classes. These classes map directly to database tables, facilitating database operations without needing to write SQL queries explicitly. Django supports multiple databases including PostgreSQL, MySQL, SQLite, and Oracle, offering flexibility depending on project requirements.

Another standout feature of Django is its built-in admin interface. With just a few lines of code, Django automatically generates a customizable admin interface for managing application data. This feature is particularly useful for content management systems (CMS), where administrators can add, edit, and delete records from the database through a user-friendly interface without requiring custom development.

Django’s architecture is based on a "batteries-included" philosophy, meaning it comes with a vast array of built-in modules and libraries that cover common web development tasks. These include authentication, URL routing, form handling, session management, caching, security features (such as protection against common web attacks like SQL injection and cross-site scripting), and internationalization (i18n) support.

These components are designed to work seamlessly together, ensuring compatibility and reducing the need for third-party libraries in many cases.

Django also promotes the concept of reusable applications. Developers can package and distribute their applications as reusable Django apps, which can be easily integrated into other projects. This modularity fosters code reusability, scalability, and collaboration within development teams or the broader Django community.

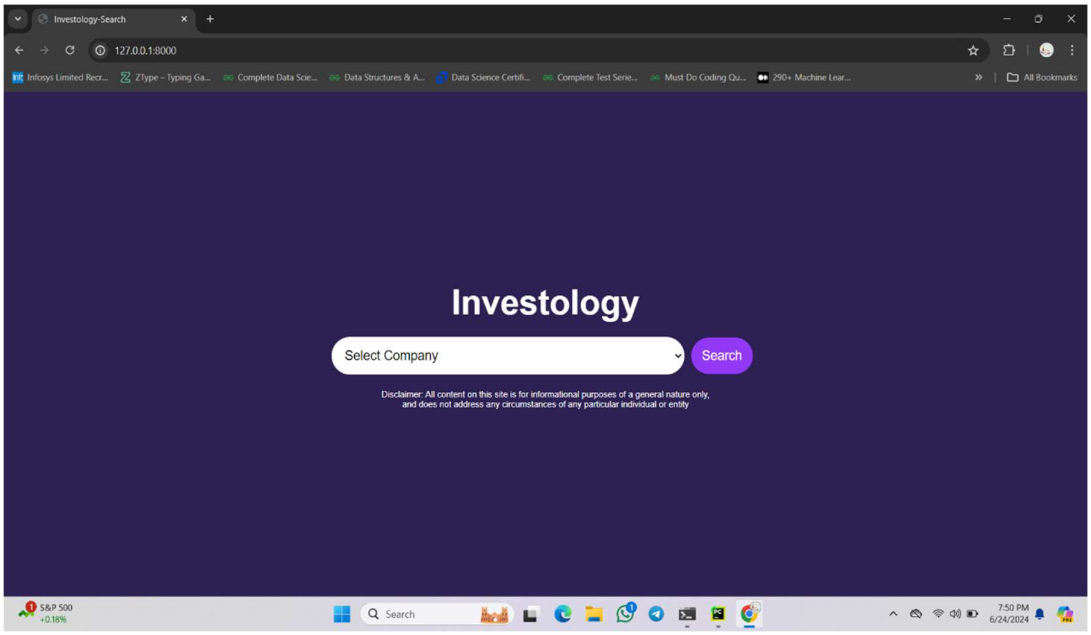
Furthermore, Django's templating engine provides a powerful way to generate dynamic HTML content. Templates allow developers to design user interfaces using HTML templates combined with Django template language (DTL), which includes template tags and filters for performing logic within templates. This separation of presentation and business logic enhances code maintainability and readability.

Django's ecosystem is enriched by a vibrant community that contributes plugins, extensions, and reusable components to extend its functionality further. These include third-party libraries for tasks such as RESTful API development (Django REST framework), asynchronous task execution (Celery), real-time web applications (Django Channels), and more.

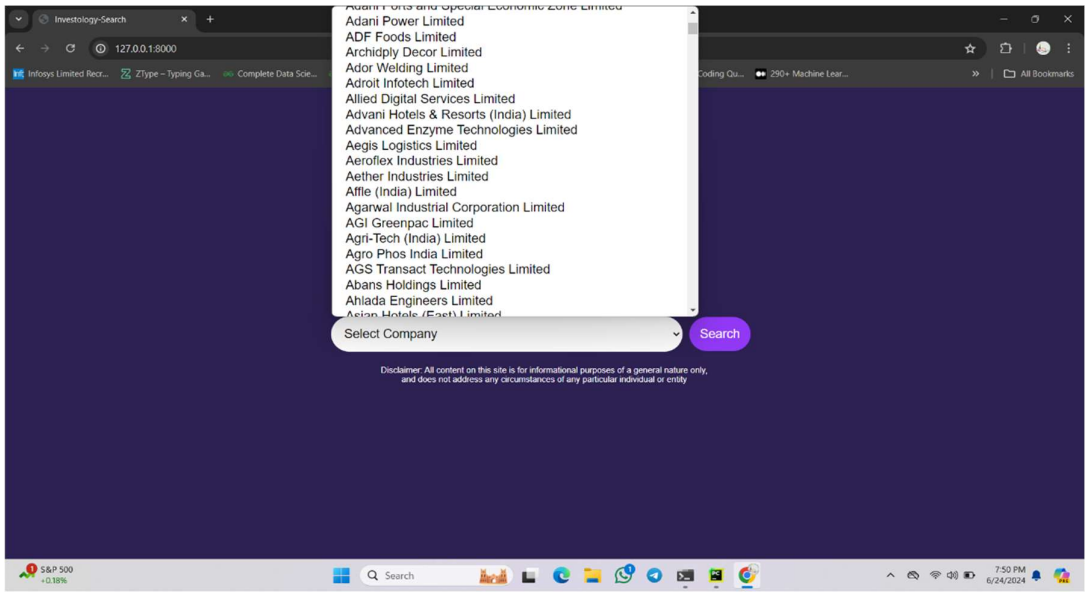
In summary, Django empowers developers to build robust web applications efficiently by providing a comprehensive toolkit, standardized conventions, and a supportive community. Its features like ORM for database interactions, built-in admin interface, modular architecture with reusable components, and powerful templating system make it suitable for a wide range of applications—from simple websites to complex enterprise-level systems. Whether you're starting a new project or maintaining an existing one, Django's flexibility and scalability make it a popular choice among developers aiming to deliver high-quality web applications with minimal effort and maximum productivity.

4.3. Screen Shots

Home Page



Home Page for Inputting Company Name

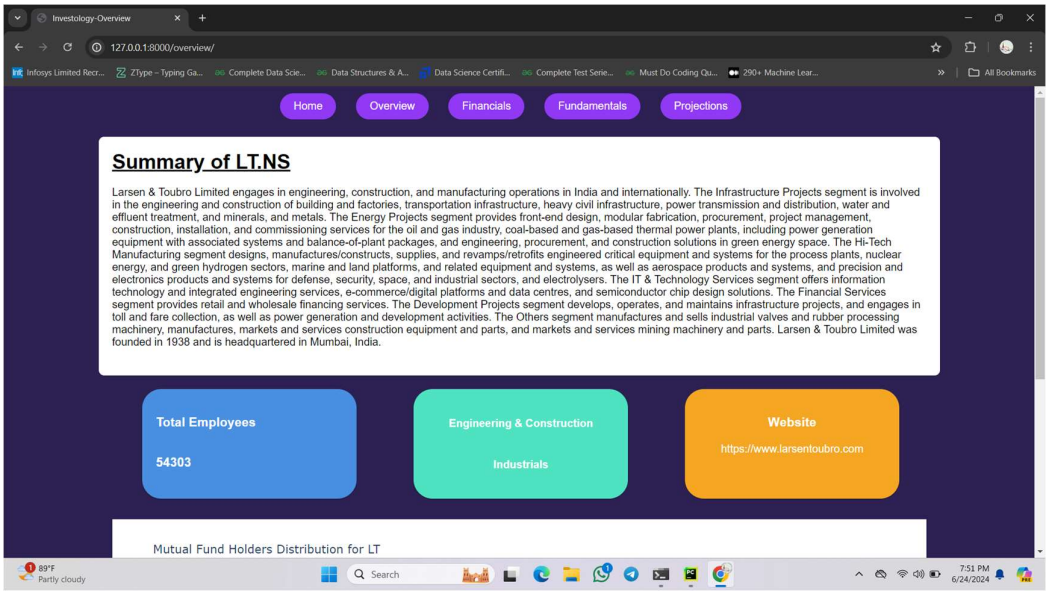


All Companies in dropdown are populated using a csv file.

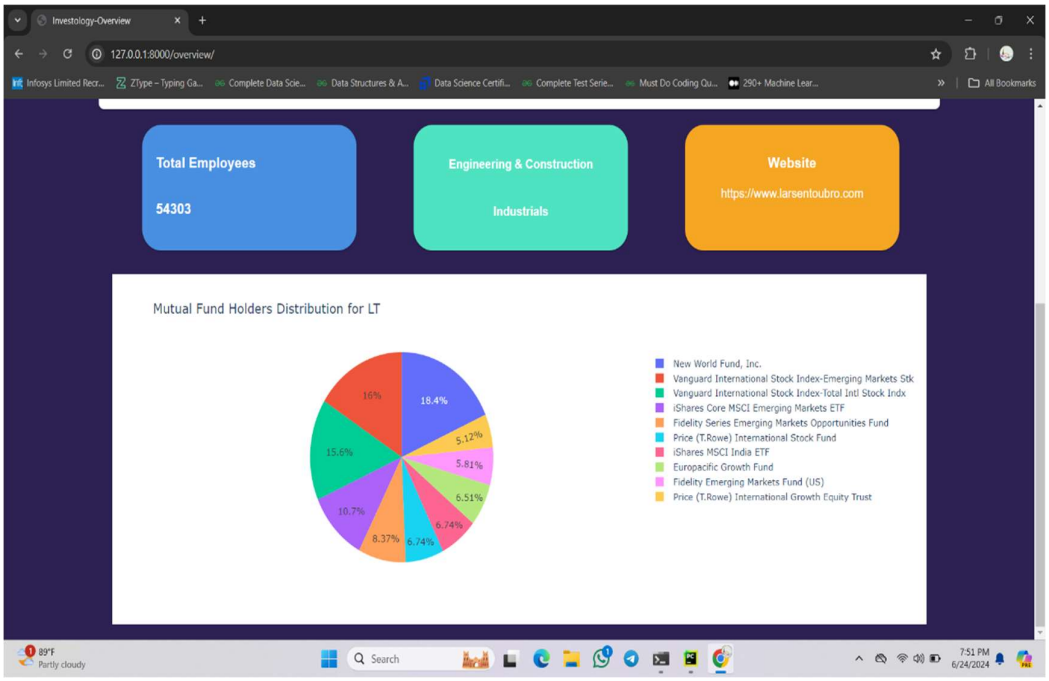
:

	SYMBOL	NAME OF COMPANY
0	20MICRONS	20 Microns Limited
1	21STCENMGM	21st Century Management Services Limited
2	360ONE	360 ONE WAM LIMITED
3	3IINFOLTD	3i Infotech Limited
4	3MINDIA	3M India Limited
5	3PLAND	3P Land Holdings Limited
6	5PAISA	5Paisa Capital Limited
7	63MOONS	63 moons technologies limited
8	A2ZINFRA	A2Z Infra Engineering Limited
9	AAATECH	AAA Technologies Limited

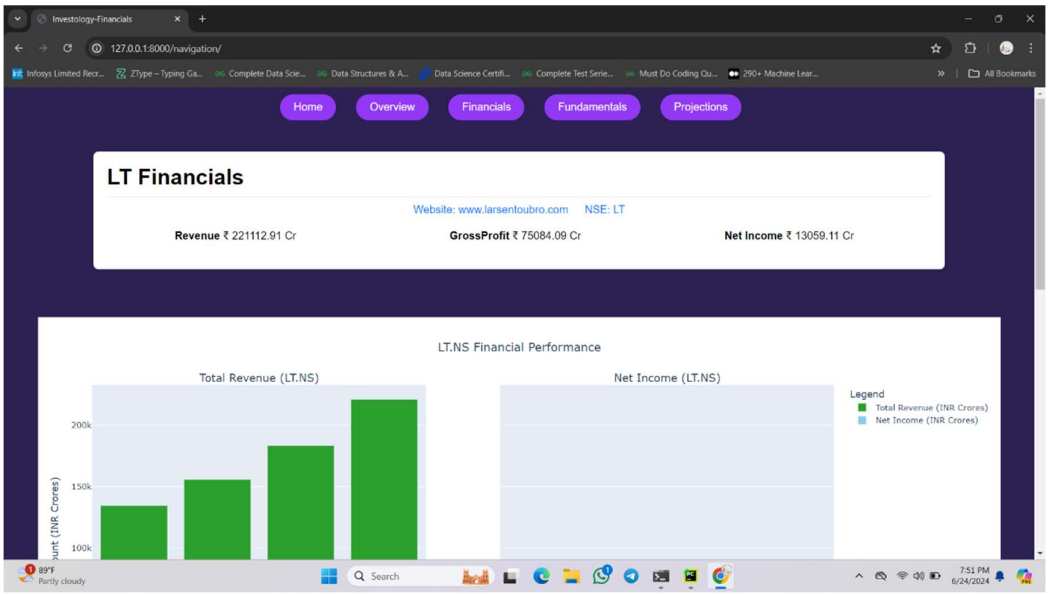
Overview Page after clicking 'Search' button



Visualizing Mutual Fund Holding



Financials Data of Selected Stock

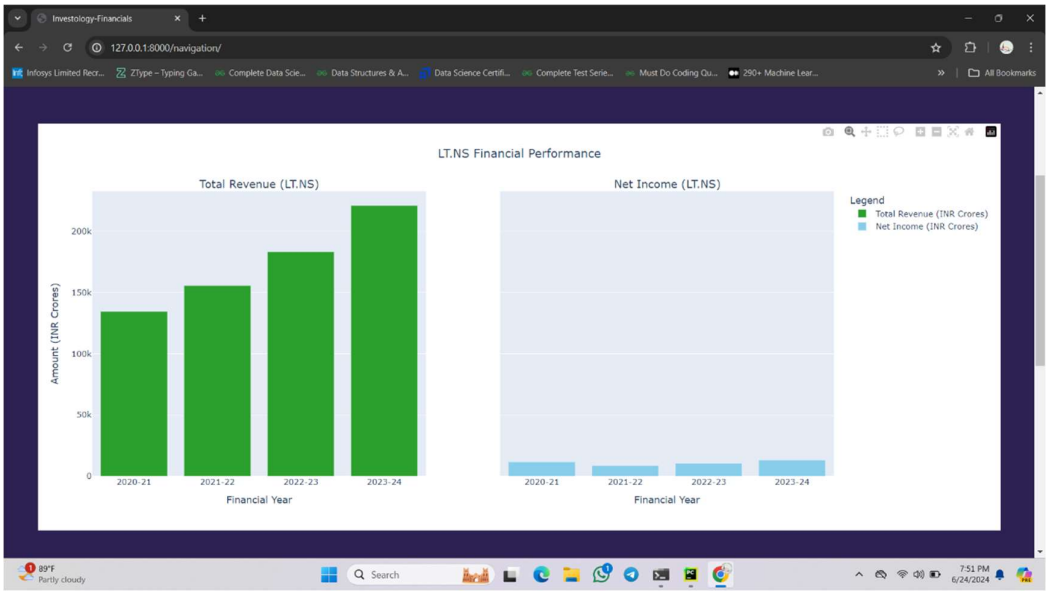


Financials Data is gathered from Yfinance.

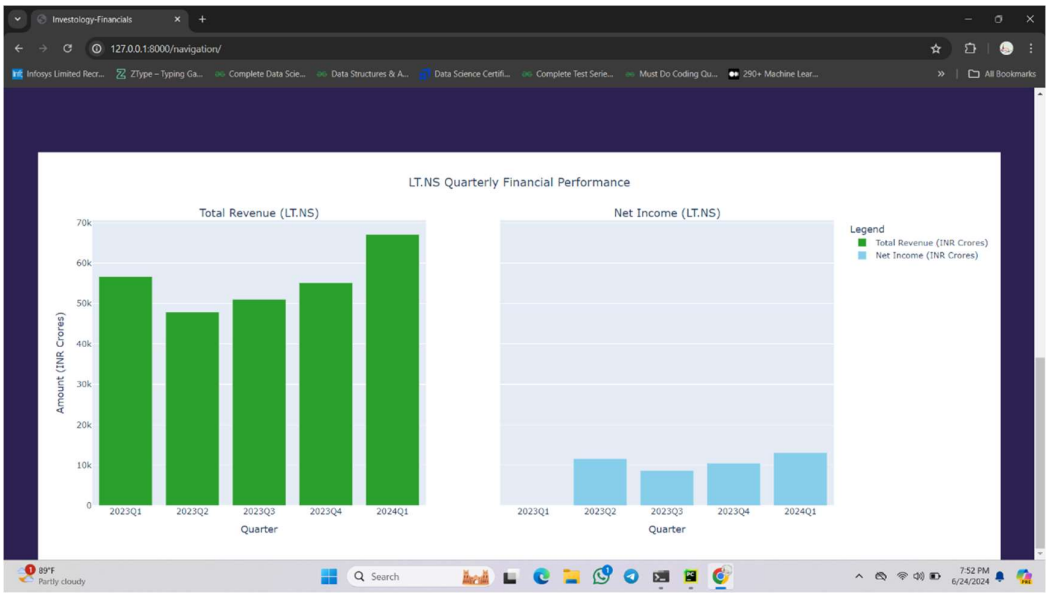
Out[8]:

	2024-03-31	2023-03-31	2022-03-31	2021-03-31
Tax Effect Of Unusual Items	-1924680000.0	277560000.0	238080000.0	-2821120000.0
Tax Rate For Calcs	0.258	0.257	0.256	0.256
Normalized EBITDA	685060000000.0	626000000000.0	569820000000.0	495640000000.0
Total Unusual Items	-7460000000.0	1080000000.0	930000000.0	-11020000000.0
Total Unusual Items Excluding Goodwill	-7460000000.0	1080000000.0	930000000.0	-11020000000.0
Net Income From Continuing Operation Net Minority Interest	459080000000.0	421470000000.0	383270000000.0	324300000000.0
Reconciled Depreciation	49850000000.0	50220000000.0	46040000000.0	40650000000.0
Reconciled Cost Of Revenue	1328710000000.0	1197590000000.0	1002670000000.0	868750000000.0
EBITDA	677600000000.0	627080000000.0	570750000000.0	484620000000.0
EBIT	627750000000.0	576860000000.0	524710000000.0	443970000000.0
Net Interest Income	30030000000.0	24690000000.0	18790000000.0	18670000000.0
Interest Expense	7780000000.0	7790000000.0	7840000000.0	6370000000.0
Interest Income	37810000000.0	32480000000.0	26630000000.0	25040000000.0
Normalized Income	464615320000.0	420667560000.0	382578080000.0	332498880000.0
Net Income From Continuing And Discontinued Operation	459080000000.0	421470000000.0	383270000000.0	324300000000.0
Total Expenses	1814680000000.0	1710810000000.0	1431660000000.0	1214950000000.0
Diluted Average Shares	3646851755.0	3659051373.0	3698832195.0	3740110733.0
Basic Average Shares	3646851755.0	3659051373.0	3698832195.0	3740110733.0
Diluted EPS	125.88	115.19	103.62	86.71
Basic EPS	125.88	115.19	103.62	86.71
Diluted NI Availto Com Stockholders	459080000000.0	421470000000.0	383270000000.0	324300000000.0

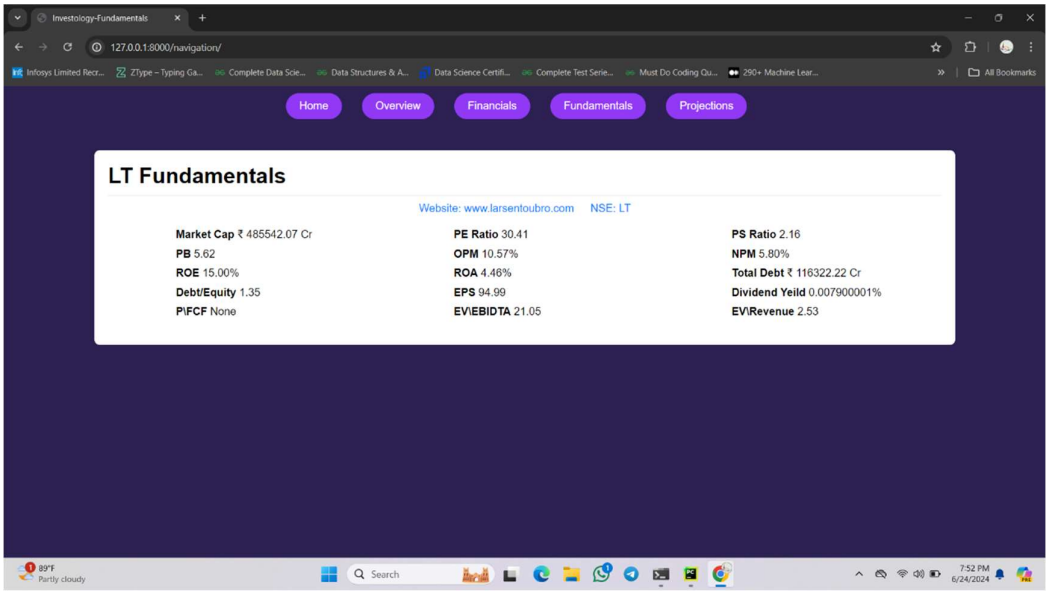
Financials Data Visualization for Yearly Revenue and Income



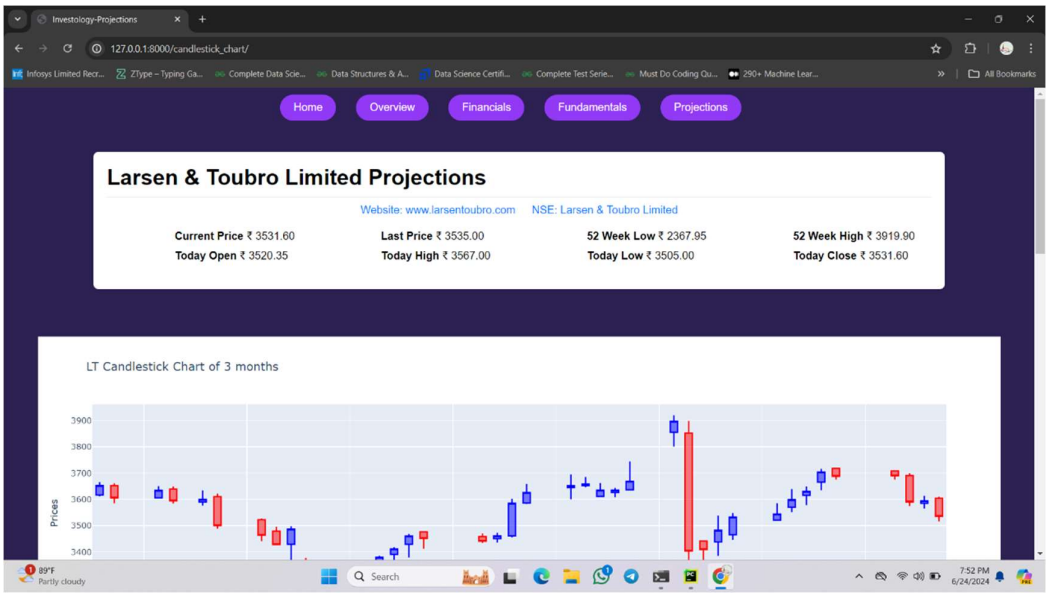
Financial Data Visualization for Quarterly Revenue and Income



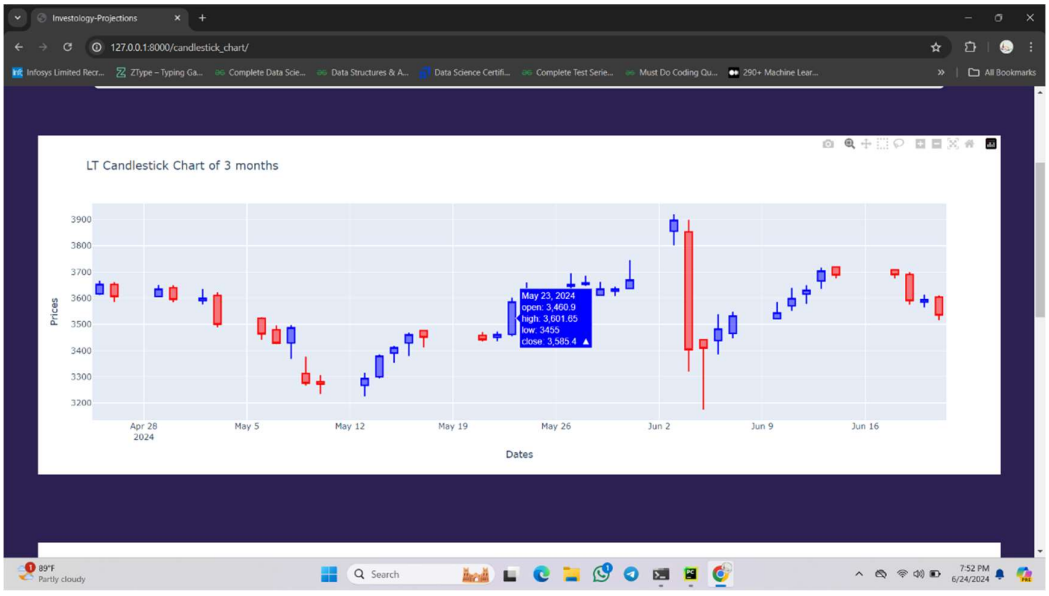
Stock Fundamental Data



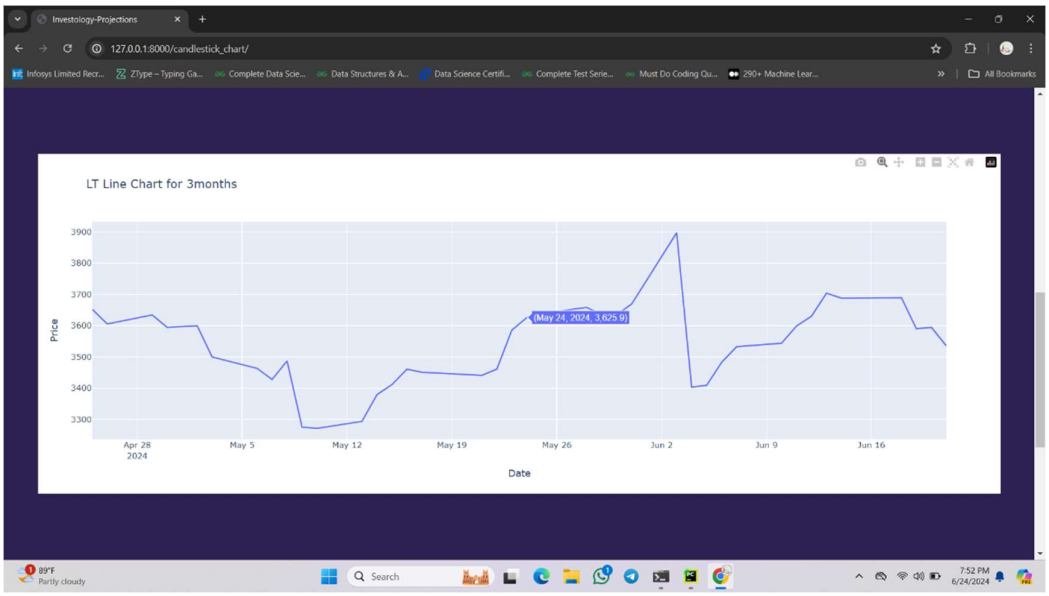
Stock Data Trends



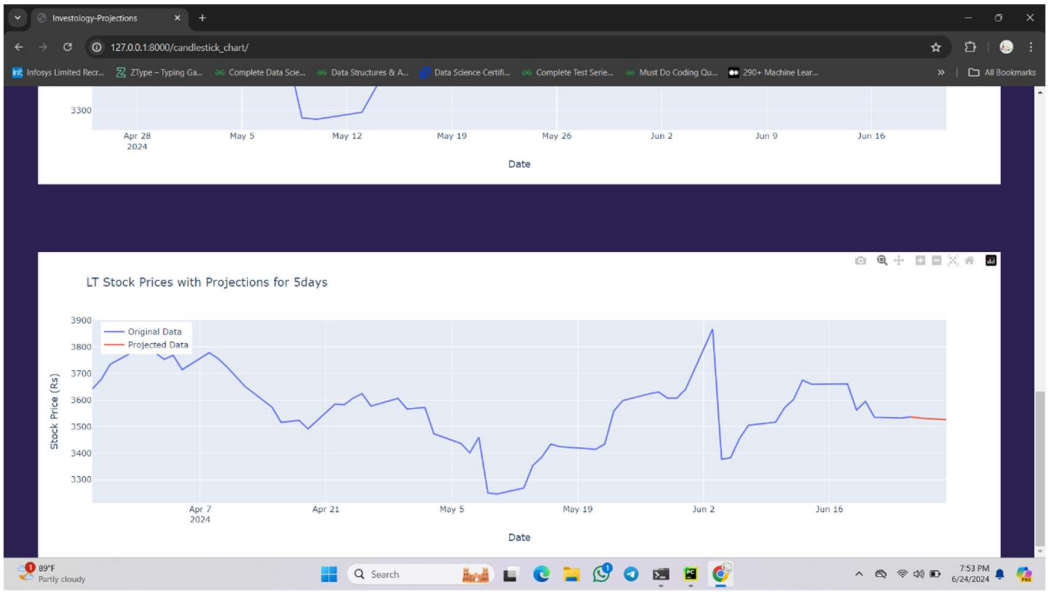
Candle Stick Pattern Chart



Line Chart for the Stock

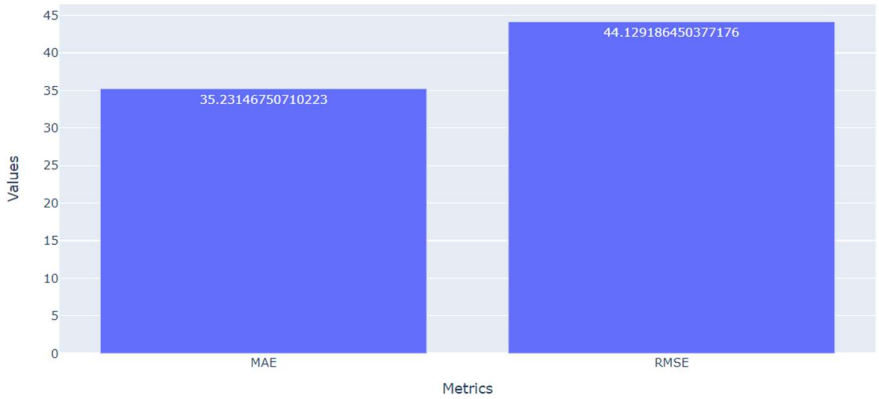


Projection Chart for next 5Days



Evaluation Metrics

```
MAE: 35.23146750710223
RMSE: 44.129186450377176
```



Epochs

```
Epoch 42/50
2/2 [==============================] - 0s 7ms/step - loss: 0.0339
Epoch 43/50
2/2 [==============================] - 0s 9ms/step - loss: 0.0331
Epoch 44/50
2/2 [==============================] - 0s 15ms/step - loss: 0.0326
Epoch 45/50
2/2 [==============================] - 0s 15ms/step - loss: 0.0327
Epoch 46/50
2/2 [==============================] - 0s 17ms/step - loss: 0.0327
Epoch 47/50
2/2 [==============================] - 0s 16ms/step - loss: 0.0327
Epoch 48/50
2/2 [==============================] - 0s 15ms/step - loss: 0.0327
Epoch 49/50
2/2 [==============================] - 0s 11ms/step - loss: 0.0326
Epoch 50/50
2/2 [==============================] - 0s 0s/step - loss: 0.0323
```

4.4. Source Code

searchPage.html

```
{% load static %}
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Investology-Search</title>
   <link rel="stylesheet" href="{% static 'css/searchPage.css' %}">
</head>
<body>
   <div class="container">
      <div class="title">Investology</div>
      <div class="search-box">
         {% if alert_message %}
         <script>
            alert("Redirecting to SearchPage. Please Enter Company Symbol!");
         </script>
         {% endif %}
         <form method="POST" action="{% url 'majorApp:overview' %}">
            {% csrf_token %}
            <div class="form-group">
               <select name="c_name" class="choose">
                  <option value="" selected>Select Company</option>
                  {% for option in options %}
                  <option value="{{ option.value }}">{{ option.text }}</option>
                  {% endfor %}
               </select>
               <button type="submit" name="search">Search</button>
            </div>
            <p class="disclaimer">Disclaimer: All content on this site is for informational purposes of a
general nature only,
               <br>and does not address any circumstances of any particular individual or entity</p>
         </form>
      </div>
   </div>
```

```
</body>
</html>
```

searchPage.css

```
body {
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #2C2152;
    color: white;
    font-family: Arial, sans-serif;
}
.container {
    text-align: center;
}
.title {
    font-size: 48px;
    font-weight: bold;
    margin: 20px 0;
}
.search-box {
    margin-top: 20px;
}
.search-box .form-group {
    display: flex;
    align-items: center;
    justify-content: center;
}
.search-box select, .search-box button {
    padding: 15px;
    font-size: 18px;
    border: none;
    border-radius: 50px;
    outline: none;
```

```
    margin-right: 10px;
}

.choose {
    width: 500px;
    padding: 15px 0px;
}
.search-box button {
    background-color: #9239F6;
    color: white;
    cursor: pointer;
}
```

Navigation.html

```
{% load static %}
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Investology-Navigation</title>
    <link rel="stylesheet" href="{% static 'css/navigation.css' %}">
</head>
<body>
    <div class="footer">
        <form method="post" action="{% url 'majorApp:navigation' %}">
            {% csrf_token %}
            <button type="submit" name="task" value="home">Home</button>
            <button type="submit" name="task" value="overview">Overview</button>
            <button type="submit" name="task" value="financials" >Financials</button>
            <button type="submit" name="task" value="fundamentals" >Fundamentals</button>
            <button type="submit" name="task" value="projections" >Projections</button>
        </form>
    </div>
        {%block content%}
        {% endblock %}
</body>
```

Navigation.css

```
body {
        margin: 0;
        padding: 0;
        display: flex;
        flex-direction: column;
        height: 100vh;
        background-color: #2C2152;
        color: white;
        font-family: Arial, sans-serif;
    }
    .header {
        display: flex;
        justify-content: center;
        align-items: center;
        background-color: #2C2152;
        padding: 20px 0;
        top: 0;
        width: 100%;
    }
    .header .search-container {
        display: flex;
        align-items: center;
    }
    .header input {
        width: 400px;
        padding: 15px;
        font-size: 18px;
        border: none;
        border-radius: 50px 0 0 50px;
        outline: none;
    }
    .header button {
        padding: 15px 20px;
        background-color: #9239F6;
        color: white;
```

```
        border: none;
        border-radius: 0 50px 50px 0;
        cursor: pointer;
        font-size: 18px;
    }
.header button:hover {
    background-color: white;
    color: #9239F6;
}
    .content {
        flex: 1;
        display: flex;
        justify-content: center;
        align-items: center;
    background-color: grey;
        padding-top: 100px; /* space for header */
        padding-bottom: 60px; /* space for footer */
    }
    .footer {
        display: flex;
        justify-content: space-evenly;
        align-items: center;
        background-color: #2C2152;
        padding: 10px 0;
        width: 100%;
    }
    .footer button {
        padding: 10px 20px;
        background-color: #9239F6;
        color: white;
        border: none;
        border-radius: 50px;
        cursor: pointer;
        font-size: 16px;
    margin-right: 25px;
}
```

Overview.html

```
{% load static %}
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>{% block title %}Investology-Overview{% endblock %}</title>
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.1/chart.min.css">
   <link rel="stylesheet" href="{% static 'css/overview.css' %}">
</head>
<body>
{% include 'majorApp/navigation.html' %}
{% block content %}
<div class="chart-container">
   <div class="chart-header">
      <p style="color: black;"><b style="font-size: 30px;"><u>Summary of
{{c_name}}</u></b><br><br>{{summary}}</p>
   </div>
</div>
<div class="card-container" style="left: 7.5; margin-left: 0px;">
   <div class="card" style="background-color: #4a90e2; color: white; align-items: normal">
         <h5>{{ first_officer_info.name }}</h5>
         <h5>{{ first_officer_info.title }}</h5>
   </div>
   <div class="card" style="background-color: #50e3c2; color: white;">
         <h5>{{ industry }}</h5>
         <h5>{{ sector }}</h5>
   </div>
   <div class="card" style="background-color: #f5a623; color: white;">
         <h5>Website</h5>
         <a>{{ website }}</a>
   </div>
</div>
<div class="pie_chart">
   <div class="chart-header">{{ pie_chart | safe }}</div>
</div>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.1/chart.min.js"></script>
{% endblock %}
</body>
```

Overview.css

```
body {
    margin: 0;
    font-family: Arial, sans-serif;
    background-color: #f0f2f5;
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 20px;
}
.card-container {
    display: flex;
    position: relative;
    justify-content: space-around;
    width: 100%;
    max-width: 1200px;
    margin-bottom: 20px;
    margin-left: -55px;
}
.card {
    background: white;
    border-radius: 28px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
    padding: 20px;
    width: 23%;
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-right: 10px;
    margin-bottom: 10px;
}
.card h2 {
```

```
    margin: 0;
    font-size: 24px;
}
.card h5 {
    margin-top: 10px;
    margin-bottom: 10px;
}
.card p {
    margin: 5px 0;
    color: grey;
}
.chart-container {
    width: 100%;
    max-width: 1200px;
    background: white;
    border-radius: 8px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
    padding: 20px;
    margin-bottom: 20px;
}
.chart-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
}
.chart-header p {
    margin: 0;
    background-color: white;
    color: black;
}
.chart-header .buttons {
    display: flex;
    gap: 10px;
}
.chart-header button {
```

```
    padding: 5px 10px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
.chart-header button.active {
    background-color: #007bff;
    color: white;
}
.chart-header button:not(.active) {
    background-color: #e0e0e0;
}
```

Financials.html

```
{% load static %}
{% load custom_filters %}
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Investology-Financials{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/financials.css' %}">
    <style>
        .plot-container {
            justify-content: center;
            margin: 50px 50px;
        }
    </style>
</head>
<body>
{% include 'majorApp/navigation.html' %}
{% block content %}
    <div class="container1">
        <div class="header1">
            <h1>{{ company_name }} Financials</h1>
        </div>
```

```
      <div class="links1">
        <a href="#">Website: {{website}}</a>
        <a href="#">NSE: {{ company_name }}</a>
      </div>
      <div class="content1">
        <div class="info1">
          <div class="row1">
            <div class="col1">
              <span class="label1">Revenue </span>
              <span class="value1">₹ {{ Revenue|human_readable_large_number }}</span>
            </div>
            <div class="col1">
              <span class="label1">GrossProfit</span>
              <span class="value1">₹ {{ Gross_Profit|human_readable_large_number}}</span>
            </div>
            <div class="col1">
              <span class="label1">Net Income</span>
              <span class="value1">₹ {{ Net_Income|human_readable_large_number}}</span>
            </div>
          </div>
        </div>
      </div>
    </div>
<div class="plot-container">
    <div>{{ year_revenue | safe }}</div>
</div>
<div class="plot-container">
    <div>{{ quat_plot | safe }}</div>
</div>
{% endblock %}
```

Financials.css

```
body {
    font-family: Arial, sans-serif;
```

```
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}
.container1 {
    width: 80%;
    margin: 20px auto;
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.header1 {
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid #ddd;
    padding-bottom: 10px;
}
.header1 h1 {
    margin: 0;
    color: black;
    align-items: center;
}
.links1 {
    margin: 10px 0;
    text-align: center;
}
.links1 a {
    margin: 0 10px;
    color: #007bff;
    text-decoration: none;
}
.content1 {
    display: flex;
    justify-content: space-between;
```

```
    padding-top: 10px;
    padding-bottom: 10px;
}
.info1 {
    width: 100%;
    margin-left: 100px;
    margin-right: -100px;
}
.row1 {
    display: flex;
    justify-content: space-between;
    margin-bottom: 10px;
    color: black;
}
.col1 {
    width: 45%;
}
.label1 {
    font-weight: bold;
}
.actions1 {
    text-align: center;
    margin-top: 20px;
}
.actions1 button {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 4px;
    cursor: pointer;
    margin: 0 10px;
}
.actions1 button:hover {
    background-color: #0056b3;
}
```

Fundamentals.html

```
{% load static %}
{% load custom_filters %}
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>{% block title %}Investology-Fundamentals{% endblock %}</title>
   <link rel="stylesheet" href="{% static 'css/fundamentals.css' %}">
</head>
<body>
{% include 'majorApp/navigation.html' %}
{% block content %}
   <div class="container1">
      <div class="header1">
         <h1>{{ company_name }} Fundamentals</h1>
      </div>
      <div class="links1">
         <a href="#">Website: {{website}}</a>
         <a href="#">NSE: {{company_name}}</a>
      </div>
      <div class="content1">
         <div class="info1">
            <div class="row1">
               <div class="col1">
                  <span class="label1">Market Cap</span>
                  <span class="value1">₹ {{ market_cap|human_readable_large_number }}</span>
               </div>
               <div class="col1">
                  <span class="label1">PE Ratio</span>
                  <span class="value1">{{ forward_pe|floatformat:2 }}</span>
               </div>
               <div class="col1">
                  <span class="label1">PS Ratio</span>
                  <span class="value1">{{ price_sales|floatformat:2 }}</span>
```

```
    </div>
</div>
<div class="row1">
    <div class="col1">
        <span class="label1">PB</span>
        <span class="value1">{{ price_book|floatformat:2 }}</span>
    </div>
    <div class="col1">
        <span class="label1">OPM</span>
        <span class="value1">{{ operating_margin|floatformat:2 }}%</span>
    </div>
    <div class="col1">
        <span class="label1">NPM</span>
        <span class="value1">{{ profit_margin|floatformat:2 }}%</span>
    </div>
</div>
<div class="row1">
    <div class="col1">
        <span class="label1">ROE</span>
        <span class="value1">{{ return_on_equity|floatformat:2 }}%</span>
    </div>
    <div class="col1">
        <span class="label1">ROA</span>
        <span class="value1">{{ return_on_assets|floatformat:2 }}%</span>
    </div>
    <div class="col1">
        <span class="label1">Total Debt</span>
        <span class="value1">₹ {{ total_debt|human_readable_large_number}}</span>
    </div>
</div>
<div class="row1">
    <div class="col1">
        <span class="label1">Debt/Equity</span>
        <span class="value1">{{ debtToEquity | floatformat:2 }}</span>
    </div>
    <div class="col1">
```

```
                <span class="label1">EPS</span>
                <span class="value1">{{EPS | floatformat:2}}</span>
            </div>
            <div class="col1">
                <span class="label1">Dividend Yeild</span>
                <span class="value1">{{ Dividend_Yield }}%</span>
            </div>
        </div>
        <div class="row1">
            <div class="col1">
                <span class="label1">P\FCF</span>
                <span class="value1">{{ P_FCF }}</span>
            </div>
            <div class="col1">
                <span class="label1">EV\EBIDTA</span>
                <span class="value1">{{ EV_EBITDA }}</span>
            </div>
            <div class="col1">
                <span class="label1">EV\Revenue</span>
                <span class="value1">{{ EV_Revenue }}</span>
            </div>
        </div>
    </div>
  </div>
{% endblock %}
```

Fundamentals.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}
.container1 {
    width: 80%;
```

```
    margin: 20px auto;
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.header1 {
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid #ddd;
    padding-bottom: 10px;
}
.header1 h1 {
    margin: 0;
    color: black;
    align-items: center;
}
.links1 {
    margin: 10px 0;
    text-align: center;
}
.links1 a {
    margin: 0 10px;
    color: #007bff;
    text-decoration: none;
}
.content1 {
    display: flex;
    justify-content: space-between;
    padding-top: 10px;
    padding-bottom: 10px;
}
.info1 {
    width: 100%;
    margin-left: 100px;
```

```
    margin-right: -100px;
}
.row1 {
    display: flex;
    justify-content: space-between;
    margin-bottom: 10px;
    color: black;
}
.col1 {
    width: 45%;
}
.label1 {
    font-weight: bold;
}
.actions1 {
    text-align: center;
    margin-top: 20px;
}
.actions1 button {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 4px;
    cursor: pointer;
    margin: 0 10px;
}
.actions1 button:hover {
    background-color: #0056b3;
}
```

Projections.html

```
{% load static %}
{% load custom_filters1 %}
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>{% block title %}Investology-Projections{% endblock %}</title>
   <link rel="stylesheet" href="{% static 'css/fundamentals.css' %}">
   <style>
      .plot-container {
         justify-content: center;
         margin: 50px 50px;
      }
   </style>
</head>
<body>
{% include 'majorApp/navigation.html' %}
{% block content %}
   <div class="container1">
      <div class="header1">
         <h1>{{ company_name }} Projections</h1>
      </div>
      <div class="links1">
         <a href="#">Website: {{website}}</a>
         <a href="#">NSE: {{company_name}}</a>
      </div>
      <div class="content1">
         <div class="info1">
            <div class="row1">
               <div class="col1">
                  <span class="label1">Current Price</span>
                  <span class="value1">₹ {{current_price|floatformat:2}}</span>
               </div>
               <div class="col1">
                  <span class="label1">Last Price</span>
                  <span class="value1">₹ {{yesterday_price|floatformat:2}}</span>
```

```
            </div>
            <div class="col1">
              <span class="label1">52 Week Low</span>
              <span class="value1">₹ {{52_week_low|floatformat:2}}</span>
            </div>
            <div class="col1">
              <span class="label1">52 Week High</span>
              <span class="value1">₹ {{52_week_high|floatformat:2}}</span>
            </div>
          </div>
          <div class="row1">
            <div class="col1">
              <span class="label1">Today Open</span>
              <span class="value1">₹ {{open|floatformat:2}}</span>
            </div>
            <div class="col1">
              <span class="label1">Today High</span>
              <span class="value1">₹ {{high|floatformat:2}}</span>
            </div>
            <div class="col1">
              <span class="label1">Today Low</span>
              <span class="value1">₹ {{low|floatformat:2}}</span>
            </div>
            <div class="col1">
              <span class="label1">Today Close</span>
              <span class="value1">₹ {{close|floatformat:2}}</span>
            </div>
          </div>
        </div>
      </div>
    </div>
<div class="plot-container">
    <div>{{ plotly_image1 | safe }}</div>
</div>
<div class="plot-container">
    <div>{{ plotly_image2 | safe }}</div>
```

```
</div>
<div class="plot-container">
  <div>{{ project | safe }}</div>
</div>
{% endblock %}
```

Projections.css

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
  background-color: #f0f2f5;
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 20px;
}
.card-container {
  display: flex;
position: relative;
  justify-content: space-around;
  width: 100%;
  max-width: 1200px;
  margin-bottom: 20px;
margin-left: -55px;
}
.card {
  background: white;
  border-radius: 28px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
  padding: 120px;
  width: 23%;
  display: flex;
  flex-direction: column;
  align-items: center;
margin-right: 10px;
margin-bottom: 50px;
```

```
}
.card h2 {
    margin: 0;
    font-size: 24px;
}
.card p {
    margin: 5px 0;
    color: grey;
}
.chart-container {
    width: 100%;
    max-width: 1200px;
    background: white;
    border-radius: 8px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
    padding: 20px;
margin-bottom: 20px;
}
.chart-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
}
.chart-header h2 {
    margin: 0;
}
.chart-header .buttons {
    display: flex;
    gap: 10px;
}
.chart-header button {
    padding: 5px 10px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
```

```
}
.chart-header button.active {
    background-color: #007bff;
    color: white;
}
.chart-header button:not(.active) {
    background-color: #e0e0e0;
}
```

majorProject.urls.py

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include(("majorApp.urls", "majorApp"), namespace="majorApp")),
]
```

majorApp.urls.py

```
from django.urls import path, include
from . import views
urlpatterns = [
    path('', views.searchPage, name="searchPage"),
    path('overview/', views.overview, name="overview"),
    path('navigation/', views.navigation, name="navigation"),
    path('candlestick_chart/', views.candlestick_chart, name='candlestick_chart'),
]
```

views.py

```
import io
from django.shortcuts import render, redirect
from django.http import HttpResponse
import yfinance as yf
import pandas as pd
import mplfinance as mpf
import matplotlib.pyplot as plt
from io import BytesIO
import plotly.graph_objects as go
import csv
from matplotlib.backends.backend_agg import FigureCanvasAgg
import logging
import plotly.express as px
from plotly.subplots import make_subplots
from .prediction import fetch_and_preprocess, create_sequences, build_model, predict_stock_prices


def searchPage(request):
    csv_file = 'majorApp/static/CSV/companies.csv'
    options = []
    with open(csv_file, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            options.append({'value': row['SYMBOL'], 'text': row['NAME OF COMPANY']})
    return render(request, 'majorApp/searchPage.html', {'options': options})
def overview(request):
    if request.method == 'POST':
        c_name = request.POST.get('c_name')
        cname = request.session['cname'] = c_name
        if len(c_name) == 0:
            alert_message = True
            return render(request, 'majorApp/searchPage.html', {'alert_message': alert_message})
        else:
            c_name += ".NS"
            name = yf.Ticker(c_name)
            # Mutualfund Shareholding
```

```
            mutual_fund_holders = name.mutualfund_holders
            fund_names = mutual_fund_holders['Holder']
            fund_percentages = mutual_fund_holders['pctHeld']
            if fund_names is not None and fund_percentages is not None:
                pie = go.Figure(data=[go.Pie(labels=fund_names, values=fund_percentages)])
                pie.update_layout(title=f'Mutual Fund Holders Distribution for {cname}', width=1200)
                pie_chart = pie.to_html(full_html=False)
            else:
                pie = go.Figure(data=[go.Pie(labels=['No Data'], values=[100])])
                pie.update_layout(title=f'No Data of Mutual Fund Holders Distribution for {cname}',
width=1200)
                pie_chart = pie.to_html(full_html=False)
            context = {'summary': get_summary(c_name).replace(';', ','),
                       'c_name': c_name,
                       'Total_Employees': name.info.get('fullTimeEmployees'),
                       'industry': name.info.get('industry'),
                       'sector': name.info.get('sector'),
                       'website': name.info.get('website'),
                       'pie_chart': pie_chart}
            return render(request, 'majorApp/overview.html', context)
    return render(request, 'majorApp/searchPage.html')

def navigation(request):
    if request.method == 'POST':
        cname = request.session.get('cname')
        action = request.POST.get('task')

        # Overview
        if action == 'overview':
            cname += ".NS"
            name = yf.Ticker(cname)

            # Mutualfund Shareholding
            mutual_fund_holders = name.mutualfund_holders
            fund_names = mutual_fund_holders['Holder']
            fund_percentages = mutual_fund_holders['pctHeld']
```

```
        print('Hello', fund_names)
        if fund_names is not None and fund_percentages is not None:
            pie = go.Figure(data=[go.Pie(labels=fund_names, values=fund_percentages)])
            pie.update_layout(title=f'Mutual Fund Holders Distribution for {cname}', width=1200)
            pie_chart = pie.to_html(full_html=False)
        else:
            pie = go.Figure(data=[go.Pie(labels=['No Data'], values=[100])])
            pie.update_layout(title=f'No Data of Mutual Fund Holders Distribution for {cname}',
width=1200)
            pie_chart = pie.to_html(full_html=False)

        context = {'summary': get_summary(cname).replace(';', ','),
                   'c_name': cname,
                   'Total_Employees': name.info.get('fullTimeEmployees'),
                   'industry': name.info.get('industry'),
                   'sector': name.info.get('sector'),
                   'website': name.info.get('website'),
                   'pie_chart': pie_chart}
        return render(request, 'majorApp/overview.html', context)

    # Financials
    elif action == 'financials':
        cname += ".NS"
        stock = yf.Ticker(cname)
        balance_sheet = stock.balance_sheet
        financials = stock.financials

        # Gross Profit
        if not financials.empty:
            total_revenue = financials.loc['Total Revenue'][0] if 'Total Revenue' in financials.index else
None
            cost_of_revenue = financials.loc['Cost Of Revenue'][
                0] if 'Cost Of Revenue' in financials.index else None

            if all([total_revenue, cost_of_revenue]):
                gross_profit = total_revenue - cost_of_revenue
```

```
    else:
        gross_profit = 'NA'
else:
    gross_profit = 'NA'

# EBITDA
if not financials.empty:
    operating_income = financials.loc['Operating Income'][
        0] if 'Operating Income' in financials.index else None
    depreciation_and_amortization = financials.loc['Depreciation & Amortization'][
        0] if 'Depreciation & Amortization' in financials.index else None

    if all([operating_income, depreciation_and_amortization]):
        ebitda = operating_income + depreciation_and_amortization
    else:
        ebitda = 'NA'
else:
    ebitda = 'NA'

# Plotting Total Revenue for Financial Years
income_stmt = stock.income_stmt
total_revenue = income_stmt.loc['Total Revenue']
df_total_rev = total_revenue.reset_index()
df_total_rev.columns = ['Date', 'Total Revenue']
df_total_rev['Date'] = pd.to_datetime(df_total_rev['Date'])
df_total_rev = df_total_rev.sort_values(by='Date').tail(4)
df_total_rev['Total Revenue (INR Crores)'] = df_total_rev['Total Revenue'] / 1e7
df_total_rev['Financial Year'] = (df_total_rev['Date'].dt.year - 1).astype(str) + '-' + df_total_rev[
                                                                                        'Date'].dt.year.astype(
    str).str[2:]
net_income = income_stmt.loc['Net Income']
df_net_inc = net_income.reset_index()
df_net_inc.columns = ['Date', 'Net Income']
df_net_inc['Date'] = pd.to_datetime(df_net_inc['Date'])
df_net_inc = df_net_inc.sort_values(by='Date').tail(4)
df_net_inc['Net Income (INR Crores)'] = df_net_inc['Net Income'] / 1e7
```

```
        df_net_inc['Financial Year'] = (df_net_inc['Date'].dt.year - 1).astype(str) + '-' + df_net_inc[
'Date'].dt.year.astype(str).str[2:]
        plots = make_subplots(rows=1, cols=2, shared_yaxes=True,
                              subplot_titles=[f'Total Revenue ({cname})', f'Net Income ({cname})'])
        plots.add_trace(go.Bar(x=df_total_rev['Financial Year'], y=df_total_rev['Total Revenue (INR
Crores)'],
                               marker_color='#2ca02c', name='Total Revenue (INR Crores)'), row=1, col=1)
        plots.add_trace(go.Bar(x=df_net_inc['Financial Year'], y=df_net_inc['Net Income (INR Crores)'],
                               marker_color='skyblue', name='Net Income (INR Crores)'), row=1, col=2)
        plots.update_layout(
            title=f'{cname} Financial Performance',
            height=600,
            legend_title='Legend',
            title_x=0.5,
            showlegend=True,
        )
        plots.update_xaxes(title_text='Financial Year', row=1, col=1)
        plots.update_xaxes(title_text='Financial Year', row=1, col=2)
        plots.update_yaxes(title_text='Amount (INR Crores)', row=1, col=1)
        year_revenue = plots.to_html(full_html = False)

        # Quaterly Revenue
        quat_income_stmt = stock.quarterly_income_stmt
        quat_total_revenue = quat_income_stmt.loc['Total Revenue']
        df_total_rev = quat_total_revenue.reset_index()
        df_total_rev = df_total_rev.head(5)
        df_total_rev.columns = ['Date', 'Total Revenue']
        df_total_rev['Date'] = pd.to_datetime(df_total_rev['Date'])
        df_total_rev = df_total_rev.sort_values(by='Date')
        df_total_rev['Total Revenue (INR Crores)'] = df_total_rev['Total Revenue'] / 1e7
        df_total_rev['Quarter'] = df_total_rev['Date'].dt.to_period('Q')

        quat_net_income = income_stmt.loc['Net Income']
        df_net_inc = quat_net_income.reset_index()
        df_net_inc = df_net_inc.head(5)
        df_net_inc.columns = ['Date', 'Net Income']
```

```
        df_net_inc['Date'] = pd.to_datetime(df_total_rev['Date'])
        df_net_inc = df_net_inc.sort_values(by='Date')
        df_net_inc['Net Income (INR Crores)'] = df_net_inc['Net Income'] / 1e7
        df_net_inc['Quarter'] = df_net_inc['Date'].dt.to_period('Q')
        plots1 = make_subplots(rows=1, cols=2, shared_yaxes=True,
                               subplot_titles=[f'Total Revenue ({cname})', f'Net Income ({cname})'])
        plots1.add_trace(go.Bar(x=df_total_rev['Quarter'].astype(str), y=df_total_rev['Total Revenue
(INR Crores)'],
                                marker_color='#2ca02c', name='Total Revenue (INR Crores)'), row=1, col=1)
        plots1.add_trace(go.Bar(x=df_net_inc['Quarter'].astype(str), y=df_net_inc['Net Income (INR
Crores)'],
                                marker_color='skyblue', name='Net Income (INR Crores)'), row=1, col=2)
        plots1.update_layout(
            title=f'{cname} Quarterly Financial Performance',
            height=600,
            legend_title='Legend',
            title_x=0.5,
            showlegend=True,
        )
        plots1.update_xaxes(title_text='Quarter', row=1, col=1)
        plots1.update_xaxes(title_text='Quarter', row=1, col=2)
        plots1.update_yaxes(title_text='Amount (INR Crores)', row=1, col=1)

        quat_plot = plots1.to_html(full_html=False)

        # Context to send
        context = {
            'company_name': cname[:-3],
            'website': stock.info.get('website')[8:],
            'Revenue': financials.loc['Total Revenue'][0],
            'Net_Income': financials.loc['Net Income'][0],
            'Gross_Profit': gross_profit,
            'year_revenue': year_revenue,
            'quat_plot': quat_plot
        }
        return render(request, 'majorApp/financials.html', context)
```

```
# Fundamentals
elif action == 'fundamentals':
    cname += ".NS"
    stock = yf.Ticker(cname)
    info = stock.info
    balance_sheet = stock.balance_sheet
    financials = stock.financials

    # earningPerShare
    if not financials.empty:
        net_income = financials.loc['Net Income'][0] if 'Net Income' in financials.index else None
        shares_outstanding = stock.info.get('sharesOutstanding')
        if all([net_income, shares_outstanding]):
            earningPerShare = net_income / shares_outstanding
        else:
            earningPerShare = 0

    # Debt To Equity
    if not financials.empty:
        total_debt = info.get('totalDebt')
        total_equity = balance_sheet.loc["Stockholders Equity"].iloc[0]
        if all([total_debt, total_equity]):
            debtyToEquity = total_debt / total_equity
        else:
            debtyToEquity = 0
    # Profit Margin
    if not financials.empty:
        net_income = financials.loc['Net Income'][0] if 'Net Income' in financials.index else None
        total_revenue = financials.loc['Total Revenue'][
            0] if 'Total Revenue' in financials.index else None

        if all([net_income, total_revenue]):
            profit_margin = (net_income / total_revenue) * 100
        else:
            profit_margin = 0
```

```
else:
    profit_margin = 0

# Operating Margin
if not financials.empty:
    operating_income = financials.loc['Operating Income'][
        0] if 'Operating Income' in financials.index else None
    total_revenue = financials.loc['Total Revenue'][
        0] if 'Total Revenue' in financials.index else None

    if all([operating_income, total_revenue]):
        operating_margin = (operating_income / total_revenue) * 100
    else:
        operating_margin = 0
else:
    operating_margin = 0

# Context Data
context = {
    'company_name': cname[:-3],
    'website': stock.info.get('website')[8:],
    'market_cap': info.get('marketCap'),
    'forward_pe': info.get('forwardPE'),
    'price_sales': info.get('priceToSalesTrailing12Months'),
    'price_book': info.get('priceToBook'),
    'profit_margin': info.get('profitMargins')*100,
    'operating_margin': info.get('operatingMargins')*100,
    'return_on_equity': info.get('returnOnEquity')*100,
    'return_on_assets': info.get('returnOnAssets')*100,
    'total_debt': info.get('totalDebt'),
    'debtToEquity': debtyToEquity,
    'EPS': earningPerShare,
    'Dividend_Yield': stock.info.get('dividendYield'),
    'P_FCF': stock.info.get('priceToFreeCashFlows'),
    'EV_EBITDA': stock.info.get('enterpriseToEbitda'),
    'EV_Revenue': stock.info.get('enterpriseToRevenue'),
```

```
            }
            return render(request, 'majorApp/fundamentals.html', context)

        # Projections
        elif action == 'projections':
            cname += ".NS"
            stock = yf.Ticker(cname)
            info = stock.info
            return redirect('majorApp:candlestick_chart')

        # Home
        elif action == 'home':
            return redirect('majorApp:searchPage')

        # Alert Message
        else:
            alert_message = True
            return render(request, 'majorApp/searchPage.html', {'alert_message': alert_message})
    return render(request, 'majorApp/navigation.html')


def get_summary(cname):
    name = yf.Ticker(cname)
    return name.info.get('longBusinessSummary')


def get_officers_info(cname):
    name = yf.Ticker(cname)
    officers = name.info.get('companyOfficers', [])
    officers_info = [{'name': officer.get('name'), 'title': officer.get('title')} for officer in officers]
    print(officers_info)
    first_officer_info = officers_info[0]
    return first_officer_info


# views.py
```

```
def candlestick_chart(request):
    # Fetch historical data for a specific stock, e.g., Tata Consultancy Services Limited (TCS)
    ticker = request.session.get('cname') + ".NS"
    stock = yf.Ticker(ticker)
    info = stock.info
    # Calculate the date 30 days ago from today
    end_date = pd.Timestamp.today().normalize()
    start_date = end_date - pd.Timedelta(days=60)
    # Fetch data for graph1
    data = yf.download(ticker, start=start_date, end=end_date)
    if data.empty:
        raise ValueError("Empty DataFrame returned by yfinance.")
    # Ensure the index is a DatetimeIndex
    data.index = pd.to_datetime(data.index)
    # Define the style with blue and red colors for the candles
    my_style = mpf.make_mpf_style(base_mpf_style='charles', rc={'figure.figsize': (10, 6)})
    # Custom market colors
    market_colors = mpf.make_marketcolors(up='blue', down='red', wick='i', edge='i', volume='in')

    # Apply the custom market colors to the style
    my_style.update(marketcolors=market_colors)
    # Create a Matplotlib figure
    fig, _ = mpf.plot(data, type='candle', style=my_style, ylabel='Price',
                      returnfig=True)
    # Render the figure as an image
    buffer = io.BytesIO()
    canvas = FigureCanvasAgg(fig)
    canvas.print_png(buffer)
    image_data = buffer.getvalue()
    # Extract company info
    stock = yf.Ticker(ticker)
    company_name = stock.info.get('longName', 'Unknown Company')
    website = stock.info.get('website', '')
    current_price = stock.history(period='1d')['Close'].iloc[-1]
    # Get yesterday's closing price
    yesterday_price = stock.history(period='2d')['Close'].iloc[0]
```

```
# Get 52-week high and low
high_52_weeks = stock.info['fiftyTwoWeekHigh']
low_52_weeks = stock.info['fiftyTwoWeekLow']
# Get open, high, low, and close prices for today
today_data = stock.history(period='1d')
open_price = today_data['Open'].iloc[0]
high_price = today_data['High'].iloc[0]
low_price = today_data['Low'].iloc[0]
close_price = today_data['Close'].iloc[0]
# Creating a Candlestick Plotly figure
fig1 = go.Figure()
fig1.add_trace(go.Candlestick(
    x=data.index,
    open=data['Open'],
    high=data['High'],
    low=data['Low'],
    close=data['Close'],
    increasing_line_color='blue',
    decreasing_line_color='red',
    name='Candlestick'
))
fig1.update_layout(
    title=f'{ticker[:-3]} Candlestick Chart of 3 months',
    xaxis_title='Dates',
    yaxis_title='Prices',
    xaxis_rangeslider_visible=False,
    height=500
)
plotly_image1 = fig1.to_html(full_html=False)
# Creating a line chart
fig2 = go.Figure()
line_trace = go.Scatter(
    x=data.index,
    y=data['Close'],
    mode='lines',
    name='Line Chart'
```

```
    )
    fig2.add_trace(line_trace)
    fig2.update_layout(
        title=f'{ticker[:-3]} Line Chart for 3months',
        xaxis_title='Date',
        yaxis_title='Price',
        xaxis_rangeslider_visible=False,
        height=500
    )
    plotly_image2 = fig2.to_html(full_html=False)
    # Prediction Model Building
    df = stock.history(period="3mo")
    df = df.reset_index()
    dates, close_prices, scaled_close_prices, scaler = fetch_and_preprocess(ticker)
    sequence_length = 5
    X = create_sequences(scaled_close_prices, sequence_length)
    split_ratio = 0.8
    split_index = int(len(X) * split_ratio)
    X_train, X_test = X[:split_index], X[split_index:]
    y = scaled_close_prices[sequence_length:]
    y_train, y_test = y[:split_index], y[split_index:]
    model = build_model(X_train, y_train, sequence_length)
    # loss = model.evaluate(X_test, y_test, verbose=0)
    predicted_prices = predict_stock_prices(model, scaled_close_prices, scaler, sequence_length)
    dates_predicted = pd.date_range(start=df['Date'].iloc[-1], periods=6)[1:]
    extended_dates = df['Date'].tolist() + [dates_predicted[0]]
    extended_prices = close_prices.tolist() + [predicted_prices.flatten()[0]]
    fig_pro = go.Figure()
    fig_pro.add_trace(go.Scatter(x=extended_dates, y=extended_prices, mode='lines', name='Original
Data'))
    fig_pro.add_trace(go.Scatter(x=dates_predicted, y=predicted_prices.flatten(), mode='lines',
name='Projected Data'))
    fig_pro.update_layout(title=f'{ticker[:-3]} Stock Prices with Projections for 5days',
                          xaxis_title='Date',
                          yaxis_title='Stock Price (Rs)',
                          legend=dict(yanchor="top", y=0.99, xanchor="left", x=0.01),
```

```
                    showlegend=True)
    project = fig_pro.to_html(full_html=False)
    # Context for template
    context = {
        'company_name': company_name,
        'website': website[8:],
        'current_price': current_price,
        'yesterday_price': yesterday_price,
        '52_week_high': high_52_weeks,
        '52_week_low': low_52_weeks,
        'open': open_price,
        'high': high_price,
        'low': low_price,
        'close': close_price,
        'image_data': image_data,
        'plotly_image1': plotly_image1,
        'plotly_image2': plotly_image2,
        'project': project
    }
    return render(request, 'majorApp/projections.html', context=context)
```

custom_filters.py

```
from django import template
register = template.Library()
@register.filter
def human_readable_large_number(value):
    try:
        value = float(value)
        if value >= 1_000_000_000_000:
            return f'{value / 1_000_000_000_000:.2f} T'
        elif value >= 1_000_000_000:
            return f'{value / 1_000_000_000:.2f} B'
        elif value >= 1_000_000:
            return f'{value / 1_000_000:.2f} M'
        elif value >= 1_000:
            return f'{value / 1_000:.2f} K'
```

```
        return f'{value:.2f}'
    except (ValueError, TypeError):
        return value
```

custom_filters1.py

```
# custom_filters.py
import base64
from django import template
register = template.Library()
@register.filter
def b64encode(value):
    return base64.b64encode(value).decode('utf-8')
```

Prediction.py

```
# prediction.py
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import yfinance as yf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense
# Fetch the data and preprocess it
def fetch_and_preprocess(stock_symbol):
    name = yf.Ticker(stock_symbol)
    df = name.history(period="3mo")
    df = df.reset_index()
    dates = df['Date']
    close_prices = df['Close']
    # Convert 'Close' prices to a numpy array
    close_prices_array = np.array(close_prices).reshape(-1, 1)
    # Normalize the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_close_prices = scaler.fit_transform(close_prices_array)
    return dates, close_prices, scaled_close_prices, scaler
# Function to create sequences for GRU
```

```
def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        seq = data[i: i + seq_length]
        sequences.append(seq)
    return np.array(sequences)
# Define the GRU model
def build_model(X_train, y_train, sequence_length):
    model = Sequential()
    model.add(GRU(units=50, activation='relu', input_shape=(sequence_length, 1)))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Train the model
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    return model
# Predict next 5 days' stock prices
def predict_stock_prices(model, scaled_close_prices, scaler, sequence_length):
    predicted_prices = []
    last_sequence = scaled_close_prices[-sequence_length:].reshape(1, sequence_length, 1)
    for _ in range(5):
        prediction = model.predict(last_sequence)
        predicted_prices.append(prediction[0, 0])
        last_sequence = np.append(last_sequence[:, 1:, :], prediction.reshape(1, 1, 1), axis=1)
    # Inverse transform the predicted prices
    predicted_prices = scaler.inverse_transform(np.array(predicted_prices).reshape(-1, 1))
    return predicted_prices.flatten()
```

CHAPTER-5

SYSTEM TESTING

5.1. Testing Description

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components on a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner.

- **Unit Testing**

  GRU Model Unit Tests

  The GRU model is central to stock price projection. Unit tests for the GRU model should verify its ability to handle various input scenarios and produce accurate predictions. These tests include checking input and output dimensions, ensuring the model processes data correctly.

  Django Views and URL Routing Unit Tests

  Django views handle user requests and return responses. Unit tests for Django views should ensure each view functions as expected under various conditions. This includes verifying that views return the correct HTTP status codes (e.g., 200 OK) and the right context data for rendering templates.

- **Integration Testing**

  Integration tests should focus on the interaction between the GRU model and Django views. These tests verify that the data flow from the user input, through the GRU model, to the user-facing response is correct. For example, a test can simulate a user submitting stock data via a web form, and then check that the GRU model processes this data and the correct prediction is rendered in the response.

- **Functional Testing**

  Functional testing ensures that the system behaves according to the specified requirements.

  → End-to-End Workflow: Test the entire user journey from inputting stock data to receiving projections. Verify that users can successfully input stock symbols or historical data, submit the form, and receive a projected stock price.

  → Model Integration: Ensure the GRU model correctly integrates with the Django application, processing input data and returning accurate predictions. Verify the consistency and accuracy of the model's predictions by comparing them with known outcomes.

  → User Interface: Test all UI components, including forms, buttons, and links, to ensure they function correctly. Verify that the projected stock prices are displayed correctly and that the interface is responsive and intuitive.

- **System Testing**

  System testing focuses on evaluating the complete and integrated application to ensure it meets the specified requirements. For a project titled "Web-Based Stock Price Projection Using GRU and

Django," system testing involves verifying the functionality, performance, security, and usability of the entire system.

→ Input Validation: Verify that all user inputs are properly validated to prevent injection attacks. Test for common vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

→ User Experience: Assess the ease of use and overall user experience. Gather feedback from users to identify any pain points or areas for improvement.

→ Regression Testing: Regression testing ensures that new changes or updates do not introduce new bugs or break existing functionality.

5.2.Test Cases

S. No	Input	Description	Test Case Result
1	Symbol Selection	User choosing company	Pass
2	User clicks search Button	Directs to overview page	Pass
3	Clicking overview Button	Displaying of details	Pass
4	Clicking Financials Button	Displaying Company Financials	Pass
5	Clicking Fundamentals Button	Displaying Company Fundamentals	Pass
6	Clicking Projections Button	Displaying Projections	Pass

CHAPTER-6

CONCLUSION & FUTURE ENHANCEMENT

Conclusion

The project "Web-Based Stock Price Projection Using GRU and Django" successfully integrates advanced machine learning techniques with a robust web application framework to deliver accurate stock price predictions. Through the implementation of GRU (Gated Recurrent Units) for time-series forecasting, the system provides sophisticated modeling capabilities, capturing complex patterns in historical stock data to forecast future prices.

The Django framework offers a powerful and flexible backend, ensuring that user interactions are handled efficiently and securely. By developing and thoroughly testing various components of the system, including the GRU model, Django views, and URL routing, the project achieves a high level of reliability and performance. Comprehensive system testing, encompassing functional, performance, security, usability, and regression tests, ensures that the application meets all specified requirements and delivers a seamless user experience.

The integration of these technologies results in a user-friendly web application that empowers users with valuable insights into stock price movements. Overall, the project demonstrates the effective application of machine learning in the financial domain, showcasing the potential of combining data science and web development to create impactful solutions.

Future Enhancement

Future enhancements for the project "Web-Based Stock Price Projection Using GRU and Django" can focus on leveraging Convolutional Neural Networks (CNNs) to recognize trends from candlestick charts and predict future trends with higher accuracy and granularity. CNNs excel in image recognition tasks, making them well-suited for analyzing visual patterns in candlestick charts, which are commonly used in technical analysis for stock price forecasting.

CNNs can be trained to extract meaningful features from candlestick chart images, capturing various patterns such as bullish/bearish trends, support/resistance levels, and chart formations like head and shoulders, triangles, or flags. By feeding historical candlestick chart data into the CNN model, it can learn to recognize these patterns and their significance in predicting future price movements.

References

1. Saripella, A.V. (2019). Stock Price Prediction. International Journal of Innovative Technology and Exploring Engineering.
2. Khairi, Teaba Wala Aldeen et al. “Stock Price Prediction using Technical, Fundamental and News based Approach.” 2019 2nd Scientific Conference of Computer Sciences (SCCS) (2019): 177-181.
3. Mohan, Saloni et al. “Stock Price Prediction Using News Sentiment Analysis.” 2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService) (2019): 205-208.
4. Deng, Shangkun et al. “Combining Technical Analysis with Sentiment Analysis for Stock Price Prediction.” 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (2011): 800-807.
5. Cakra, Yahya Eru and Bayu Distiawan Trisedya. “Stock price prediction using linear regression based on sentiment analysis.” 2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS) (2015): 147-154.
6. Scholar, Lavanya. M Research and Dr.P.Gnanasekaran. “Stock Exchange Price Prediction Using Linear Regression Model.” 2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA) (2023): 1054-1059.
7. Khan, Dr. Imtiyaz Gul et al. “Stock Price Prediction Using AI.” International Journal of Multidisciplinary Research and Growth Evaluation (2023): n. pag.
8. Kumar, Nikhil. “USER PERCEPTION TOWARDS STOCK PRICE PREDICTION USING LINEAR REGRESSION MODEL.” INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (2024): n. pag.
9. Jena, Swarnaprabha et al. “Prediction of Stock Price Using Machine Learning Techniques.” 2023 IEEE 2nd International Conference on Industrial Electronics: Developments & Applications (ICIDeA) (2023): 169-174.
10. Diqi, Mohammad. “StockTM: Accurate Stock Price Prediction Model Using LSTM.” International Journal of Informatics and Computation (2022): n. pag.
11. Nelson, David M. Q. et al. “Stock market's price movement prediction with LSTM neural networks.” 2017 International Joint Conference on Neural Networks (IJCNN) (2017): 1419-1426.
12. Gaurav, Akshat et al. “Long Short-Term Memory Network (LSTM) based Stock Price Prediction.” Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems (2023): n. pag.
13. Saud, Arjun Singh and Subarna Shakya. “3-WAY GATED RECURRENT UNIT NETWORK ARCHITECTURE FOR STOCK PRICE PREDICTION.” Indian Journal of Computer Science and Engineering (2021): n. pag.
14. Jaiswal, Rashi and Brijendra Kumar Singh. “A Hybrid Convolutional Recurrent (CNN-GRU) Model for Stock Price Prediction.” 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT) (2022): 299-304.
15. Caniago, Afif Ilham et al. “Recurrent Neural Network With Gate Recurrent Unit For Stock Price Prediction.” Telematika (2021): n. pag.
16. Xiong, Qiushi et al. “GRU Stock Price Forecasting Method Based on HP Filter.” 2021 International Conference on Aviation Safety and Information Technology (2021): n. pag.
17. Ansah, Kwabena et al. “Intelligent Models for Stock Price Prediction: A Comprehensive Review.” J. Inf. Technol. Res. 15 (2022): 1-17.
18. Mohammed, Walid Abass. “Challenges of Stock Prediction.” Advances in Business Information Systems and Analytics (2020): n. pag.
19. Shah, Dev et al. “Stock Market Analysis: A Review and Taxonomy of Prediction Techniques.” International Journal of Financial Studies (2019): n. pag.
20. Maddodi, Srivatsa and K. G. Nandha Kumar. “STOCK MARKET FORECASTING: A REVIEW OF LITERATURE.” (2021).