Tribhuvan University

Faculty of Humanities and Social Science

Stock Prediction System

A PROJECT REPORT

Submitted to

Department of Computer Application

Pascal National College

In partial fulfillment of the requirements for the Bachelors in Computer Application

Submitted by

Keshab Poudel | 6-2-1226-10-2021

August 2025

Under Supervision of

Suresh Thapa

Tribhuvan University

Faculty of Humanities and Social Science

Pascal National College

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project, prepared under my supervision by KESHAB POUDEL entitled "**Stock Prediction System**" in partial fulfillment of the requirements for the degree of Bachelor of Computer Application, is recommended for the final evaluation.

_____

Suresh Thapa

SUPERVISOR

Faculty Member

Department of Computer Application

Pascal National College, Satdobato, Lalitpur

Tribhuvan University

Faculty of Humanities and Social Science

Pascal National College

LETTER OF APPROVAL

This is to certify that this project prepared by Keshab Poudel entitled "**Stock Prediction System**" in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

Ashok Kumar Pant Principal, Faculty Member Department of Computer Application Pascal National College Satdobato, Lalitpur	**Suresh Thapa** BCA Coordinator Department of Computer Application Pascal National College Satdobato, Lalitpur
(Internal Examiner)	(External Examiner)

ABSTRACT

This project presents the development of a Stock Prediction System aimed at forecasting short-term stock prices using the ARIMA (Autoregressive Integrated Moving Average) model. The system is designed to assist investors in making data-driven decisions by analyzing historical NEPSE stock data. The backend is built with Django and integrates a Python-based ARIMA forecasting module that handles data preprocessing, model training, and prediction generation. The frontend, developed using React.js, enables users to upload CSV data and visualize predictions through interactive graphs. The prediction logic and backend functionality have been fully implemented, and the frontend has also been successfully integrated and tested.

Keywords: Stock Forecasting, ARIMA, Time Series Analysis, NEPSE, Python

ACKNOWLEDGEMENT

I would like to sincerely thank everyone who contributed to the successful development of the **Stock Prediction System.**

To begin with, my heartfelt appreciation goes to our project supervisor, Suresh Thapa, and our **principal**, Ashok Kumar Pant, for their exceptional guidance and unwavering support. Their insights, encouragement, and expert advice played a vital role in steering the project in the right direction and motivating me to strive for excellence.

I am also grateful to the many individuals who offered their time and shared their knowledge, whether through thoughtful discussions, valuable feedback, or access to helpful research materials. Your input has been deeply appreciated and has significantly influenced the depth and quality of our work.

Lastly, I extend my thanks to modern AI tools, which provided assistance in refining ideas, organizing content, and enhancing the clarity of our documentation.

Sincerely,
Keshab Poudel

TABLE OF CONTENTS

LIST OF ABBREVIATIONS viii
LIST OF FIGURES ix
LIST OF TABLES x
CHAPTER 1: INTRODUCTION 1
1.1. Introduction 1
1.2. Problem Statement 1
1.3. Objectives 1
1.4. Scope and Limitations 1
1.4.1. Scope 1
1.4.2. Limitations 2
1.5. Development Methodology 2
1.6. Report Organization 3
CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW 4
2.1. Background Study 4
2.2. Literature Review 4
CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 6
3.1. System Analysis 6
3.1.1. Requirement Analysis 6
3.1.2. Feasibility Analysis 7
3.1.3. Data Modelling 8
3.1.4. Process Modelling 8
3.2. System Design 9
3.2.1. Architectural design 9
3.2.2. Database Schema design 10
3.2.3. Interface design (UI/UX) 11
3.2.4. Physical DFD 12
3.3. Algorithm Details 12
CHAPTER 4: IMPLEMENTATION AND TESTING 14
4.1. Implementation 14
4.1.1. Tools used 14
4.1.2. Implementation details of modules 15
4.2. Testing 16

4.2.1. Test cases for Unit testing 16
4.2.2. Test cases for System testing 17
Chapter 5: Conclusion 18
5.1. Conclusion 18
5.2. Lesson Learnt 18
5.3. Future Recommendations 18
REFERENCES 19
APPENDICES 20

LIST OF ABBREVIATIONS

API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
DFD	Data Flow Diagram
ER	Entity Relationship
JSON	JavaScript Object Notation
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
NEPSE	Nepal Stock Exchange
NRB	Nepal Rastra Bank
ORM	Object-Relational Mapping
REST	REpresentational State Transfer
RMSE	Root Mean Squared Error
UI	User Interface

LIST OF FIGURES

Figure 1.1: Iterative Waterfall Methodology 2
Figure 3.1: Use Case Diagram of Stock Prediction System 6
Figure 3.2: Gantt Chart 7
Figure 3.3: ER Diagram for Stock Prediction System 8
Figure 3.4: Context Level DFD of Stock Prediction System 8
Figure 3.5: Architecture Design of Stock Prediction System 9
Figure 3.6: Database Schema for of Stock Prediction System 10
Figure 3.7: Wireframe for dashboard 11
Figure 3.8: Wireframe for add stock page 11
Figure 3.9: Level 1 DFD of Stock Prediction System 12

LIST OF TABLES

Table 4.1: Test cases for Unit Testing 16
Table 4.2: Test cases for System Testing.....17

CHAPTER 1: INTRODUCTION

1.1. Introduction

One of the vital elements of a market economy is stock market. The reason behind this is mainly because of the foundation it lays for public listed companies to gain capital via investors, who invest to buy equity in the company. With the aid of refinements in the industries, stock market is expanding rapidly. The stock market is volatile in nature and the prediction of the same is not an easy task. Stock prices depend upon a variety of factors including economic, political, psychological, rational and other important aspects. Although, the stock trend is difficult to predict, investors seem to find new techniques in order to minimize the risk of investment and increase the probability of profiting from the investments.[1]

This project aims to develop a stock price prediction system using the ARIMA (Autoregressive Integrated Moving Average) model, a widely used statistical method for time-series forecasting. By analyzing historical stock data, the system will provide short-term price predictions, enabling investors to make informed decisions based on data-driven insights rather than speculation.

1.2. Problem Statement

Stock prices are highly volatile, making it difficult for investors to predict market trends. Many traders rely on intuition rather than analytical tools, increasing financial risks. This project aims to address this challenge by implementing ARIMA-based stock price forecasting, providing a reliable and systematic approach for predicting short-term price movements. By leveraging statistical modeling, the system will offer investors a structured approach to market analysis, helping them make informed decisions and navigate stock market fluctuations more effectively.

1.3. Objectives

- To develop an ARIMA model for stock price forecasting.
- To enhance investment decisions with data-driven predictions.

1.4. Scope and Limitations

1.4.1. Scope

- Predict stock prices using ARIMA model.

- Use historical NEPSE data for forecasting.
- Display results in graphs and tables.

1.4.2. Limitations

- Only supports time-series prediction using ARIMA
- Requires clean, preformatted CSV with date and close price; won't handle raw or noisy data well.
- Prediction accuracy declines with irregular or highly volatile data.
- Only supports short-term forecasting (typically up to 7 days).
- No live NEPSE data fetching—users must upload data manually.

1.5. Development Methodology

The Stock Prediction System uses the iterative waterfall methodology for development. This model combines the linear phases of the traditional waterfall model—requirements, design, implementation, testing, and maintenance—with the ability to revisit each phase in cycles. Feedback from each stage is used to refine the next iteration, allowing continuous improvement of the forecasting model and user interface. This approach is suitable for developing a data-driven system that requires ongoing adjustments based on evaluation metrics and test results.

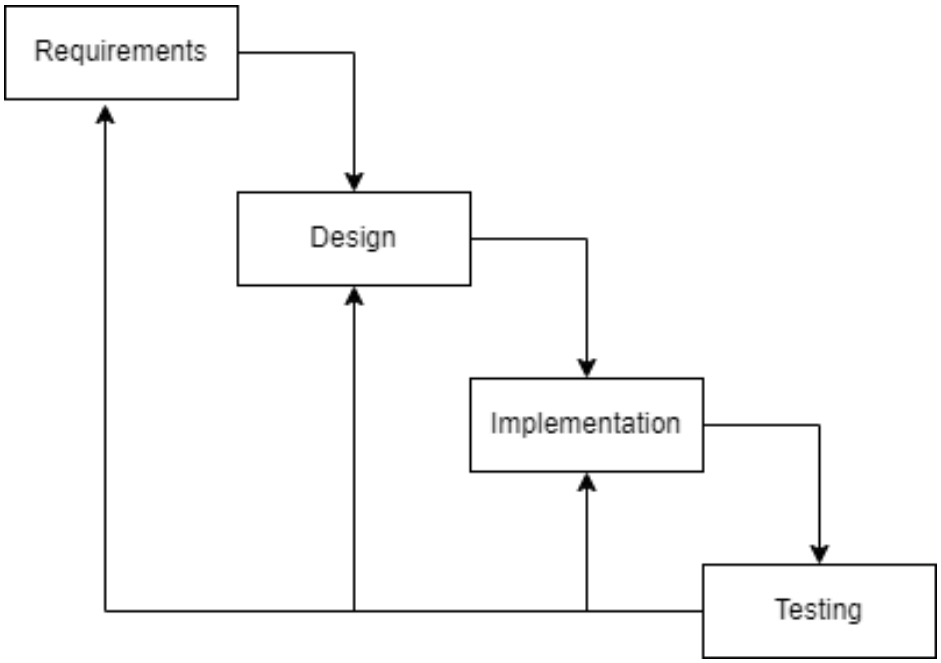


Figure 1.1: Iterative Waterfall Methodology

1.6. Report Organization

This report is structured into the following chapters:

Chapter 1: Introduction

Provides an overview of the project, defines the problem statement, outlines the objectives, scope, limitations, and briefly discusses the development methodology used.

Chapter 2: Background Study and Literature Review

Covers the foundational knowledge of stock market prediction and time series forecasting. Reviews research and models related to ARIMA and their applications in financial data analysis.

Chapter 3: System Analysis and Design

Describes the requirement analysis, feasibility study, and design of the system, including architecture, system flowcharts, and algorithm explanation. It focuses on the design considerations for implementing ARIMA.

Chapter 4: Implementation and Testing

Details the development of each module—data uploading, preprocessing, model training, and result visualization. It also includes testing methodologies, unit test cases, and system-level test results to validate system functionality.

Chapter 5: Conclusion

Summarizes the outcomes of the project.

References

Lists all academic, technical, and web-based resources cited during the project.

Appendices

Includes supplementary materials such as code snippets, screenshots.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1. Background Study

Initially it should be noted that due to the proprietary nature of many financial forecasting systems, detailed implementation specifics are often not publicly available. So, time series forecasting and prediction utilizing ARIMA model was studied for this project to understand how existing systems approach stock price prediction. Specifically, models used by platforms like Yahoo Finance and Google Finance, which often incorporate ARIMA or related time-series techniques, were analyzed to an extent. These systems often utilize historical stock data, including price and volume, to train and forecast future trends. A common approach involves pre-processing data to ensure stationarity, a critical requirement for ARIMA models, and then fitting the model to historical data to generate forecasts. The evaluation of these systems typically involves metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to assess the accuracy of predictions.

2.2. Literature Review

Past studies have consistently demonstrated the application of ARIMA models for stock price prediction. Foundational work [2] established the theoretical framework for ARIMA, emphasizing the importance of stationarity and autocorrelation in time series analysis. This methodology has been widely adopted in financial forecasting.

Research has shown that ARIMA models can effectively capture temporal dependencies in historical stock data. Numerous studies have explored the application of ARIMA, sometimes in conjunction with other techniques, to forecast stock prices. These studies typically utilize historical price and volume data obtained from sources like Yahoo Finance and Google Finance. A common practice involves preprocessing the data to ensure stationarity, a crucial requirement for ARIMA models.

Furthermore, studies like the one by the Nepal Rastra Bank (NRB) [3] have specifically explored the usefulness of ARIMA models for analyzing and forecasting the Nepalese stock market (NEPSE) index. The NRB study, using data from mid-2012 to the end of 2015, demonstrated that ARIMA models can effectively explain variations, trends, and fluctuations in the NEPSE index. This research confirms the applicability of time series techniques, particularly ARIMA, for modeling and forecasting daily stock index values.

While ARIMA models have limitations, including assumptions of linearity and stationarity, they remain a valuable tool for understanding and predicting stock market trends based on historical data.

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

3.1. System Analysis

3.1.1. Requirement Analysis

I. Functional Requirements:

- The system should be able to load and analyze historical data.
- The system should apply ARIMA for short-term stock prediction.
- The system should display predicted stock prices.

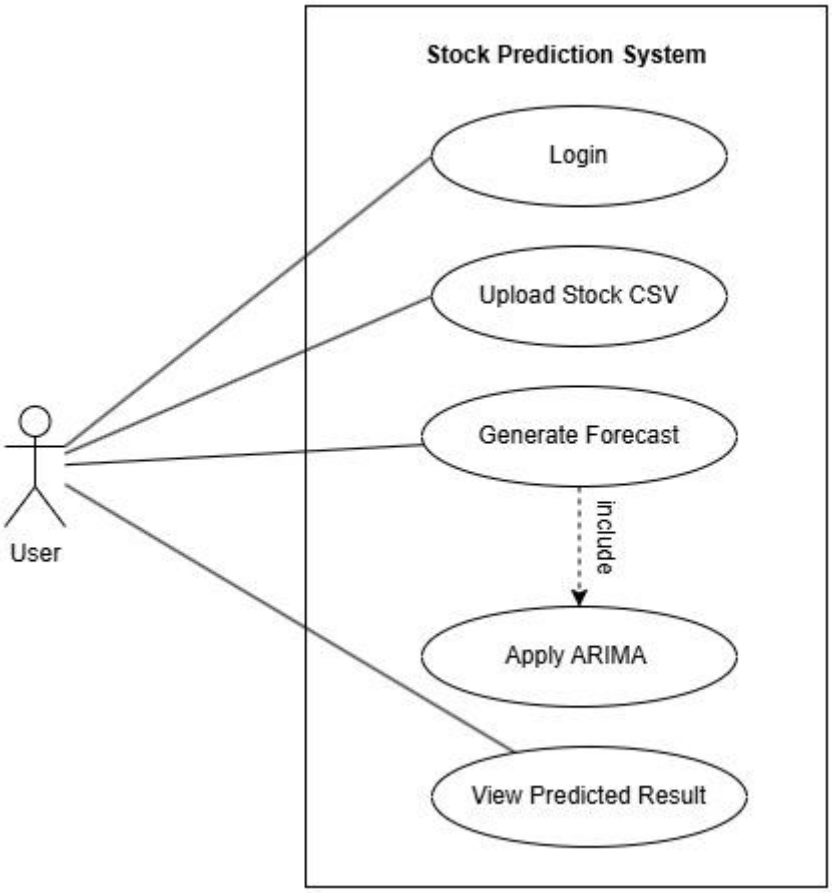


Figure 3.1: Use Case Diagram of Stock Prediction System

II. Non-Functional Requirements:

- The system should have simple and intuitive interface.
- The system should be able to provide quick and efficient predictions.
- The system should maintain accuracy in forecasting.

3.1.2. Feasibility Analysis

I. Technical

This project requires a single system where users interact with a stock prediction application. The system loads stock data and preprocesses the data, automatically selects ARIMA parameters, trains the model, and generates stock price forecasts. Users receive visualized results in the form of graphs and tables. The system is built using Python and its libraries, all of which are widely used open-source tools, ensuring no technical difficulties in development.

II. Operational

The system is designed to be simple, intuitive, and user-friendly. A user can select a stock, initiate a forecast, and view results without requiring prior technical knowledge of ARIMA models. The application will be accessible via a web interface, making it easy for users to interact with predictions in real time. No advanced configuration is needed, as the system will automatically optimize the forecasting model based on statistical tests and data characteristics.

III. Economic

This project is cost-effective since it leverages open-source tools. Since no proprietary software or paid tools are required, there are no additional costs involved in its development.

IV. Schedule

This project was scheduled as:

2025	Feb	March	April	May	June	July	Remarks
Requirement Gathering							Complete
System Design							Complete
Development Phase							Complete
Development Phase II							Complete
Testing							Complete
Documentation							Complete

Figure 3.2: Gantt Chart

3.1.3. Data Modelling

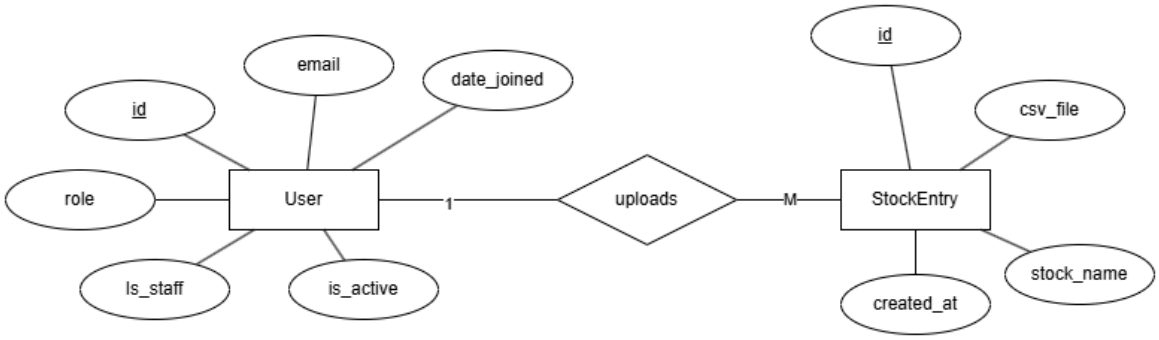


Figure 3.3: ER Diagram for Stock Prediction System

The Entity Relationship (ER) diagram illustrates the logical structure of the database for the Stock Prediction System. It consists of two main entities: User and StockEntry.

- The User entity stores information such as ID, email, role, date joined, and status fields like is_active and is_staff.
- The StockEntry entity includes fields such as ID, stock name, CSV file path, and creation timestamp.

The relationship between the entities is one-to-many, where one user can upload multiple stock entries. This is represented by the uploads relationship, with a cardinality of 1 to M (many).

Each attribute is connected to its respective entity, and the primary keys (id) are clearly identified. This data model ensures that uploaded stock files are always linked to a valid user and provides a structured way to organize and retrieve stock-related data.

3.1.4. Process Modelling

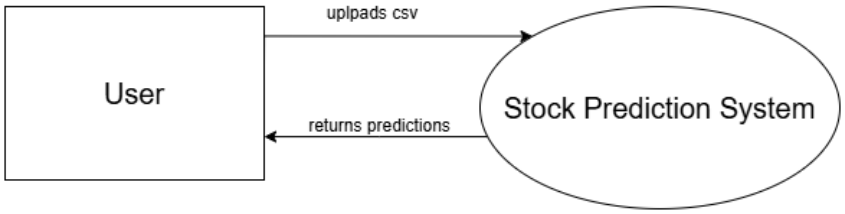


Figure 3.4: Context Level DFD of Stock Prediction System

The context-level DFD shows how the user interacts with the Stock Prediction System. The user uploads a CSV file, and the system processes it to return stock price predictions.

The diagram highlights two main data flows:

- Uploads CSV to the system

- Receives predictions from the system

This high-level model gives a clear overview of the system's main functionality.

3.2. System Design

3.2.1. Architectural design

The Stock Prediction System follows a client-server architecture that separates the frontend interface from the backend processing logic. The backend is developed using Django, which handles API endpoints, CSV data processing, ARIMA model execution, and database interactions. The frontend is built with React.js and communicates with the backend through RESTful APIs.

Users interact with the system through the web interface, where they can upload historical stock data in CSV format. This file is sent to the backend, where the system preprocesses the data, runs the ARIMA model, and returns the forecast results. The frontend then visualizes the results using graphs and tables. PostgreSQL is used to store user data and file information.

This layered architecture promotes modularity, scalability, and ease of maintenance. Each component is loosely coupled, enabling independent updates or replacements if needed in the future.

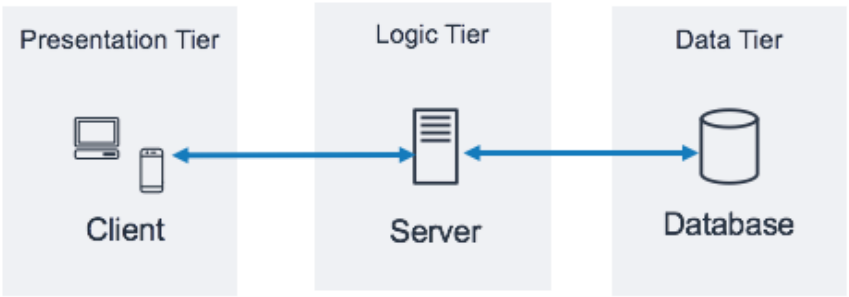


Figure 3.5: Architecture Design of Stock Prediction System

3.2.2. Database Schema design

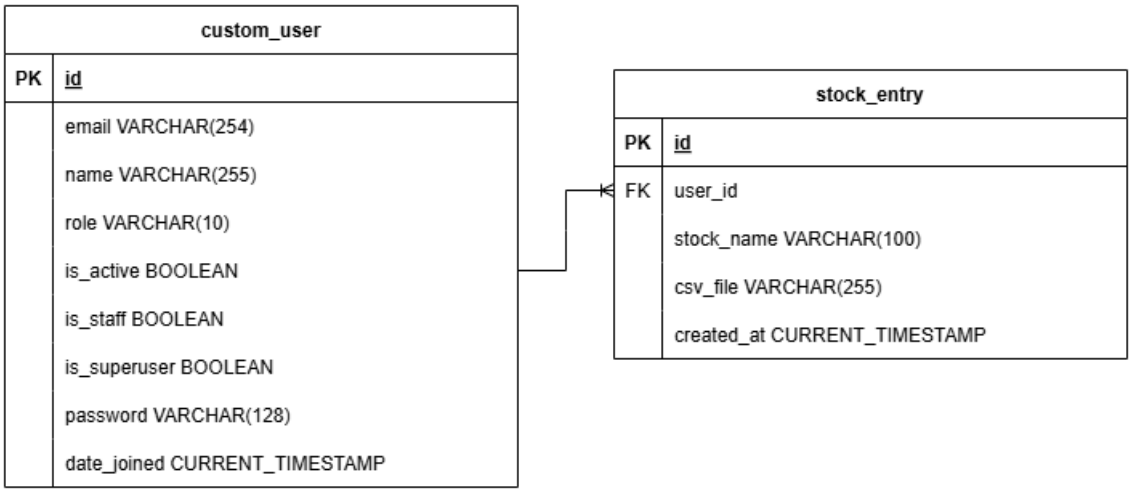


Figure 3.6: Database Schema for of Stock Prediction System

The database schema of the Stock Prediction System is designed using a relational structure to ensure data integrity, simplicity, and scalability. It consists of two main tables: custom_user and stock_entry.

The custom_user table stores user-related information. Each user has a unique ID and fields such as email, name, role (either staff or user), and authentication-related fields like password, active status, and date joined. The role field is used to differentiate between system users and administrative staff.

The stock_entry table records the uploaded stock data files for each user. Each record includes the stock name, the path to the uploaded CSV file, and the timestamp when it was uploaded. The user_id is a foreign key referencing the custom_user table, which ensures that every stock entry is linked to a valid user.

The schema follows a one-to-many relationship where one user can have multiple stock entries. This is achieved using a foreign key (user_id) in the stock_entry table that points to the id in the custom_user table.

This structure is optimized for modularity, simplifies user-specific data retrieval, and is fully compatible with Django's ORM system.

3.2.3. Interface design (UI/UX)

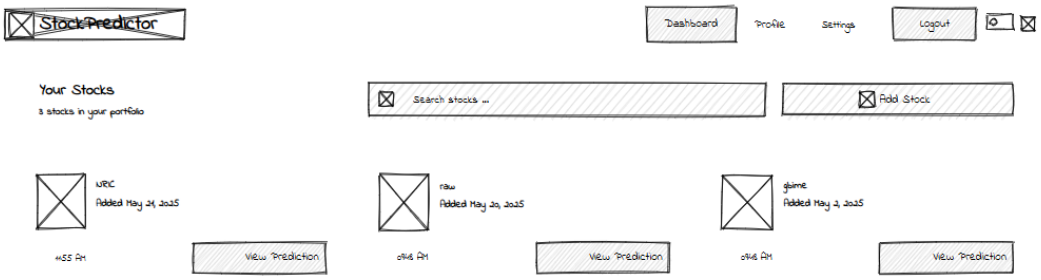


Figure 3.7: Wireframe for dashboard

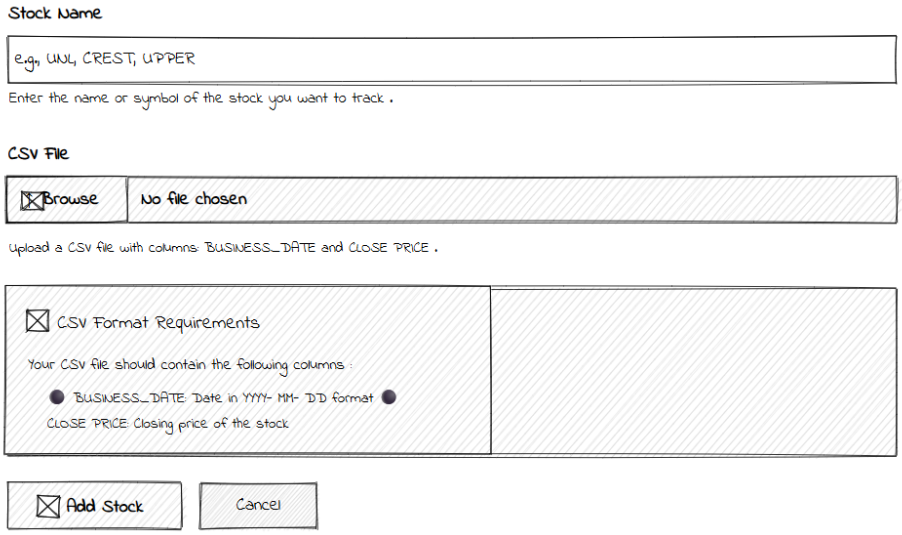


Figure 3.8: Wireframe for add stock page

The system has a simple and clean interface built for ease of use. The dashboard shows all uploaded stocks in a card layout with options to filter, search, and view predictions.

The Add Stock page includes a form to enter the stock name and upload a CSV file. It also shows format instructions to guide the user.

The prediction made that is also made shows relative metrics and graphs.

3.2.4. Physical DFD

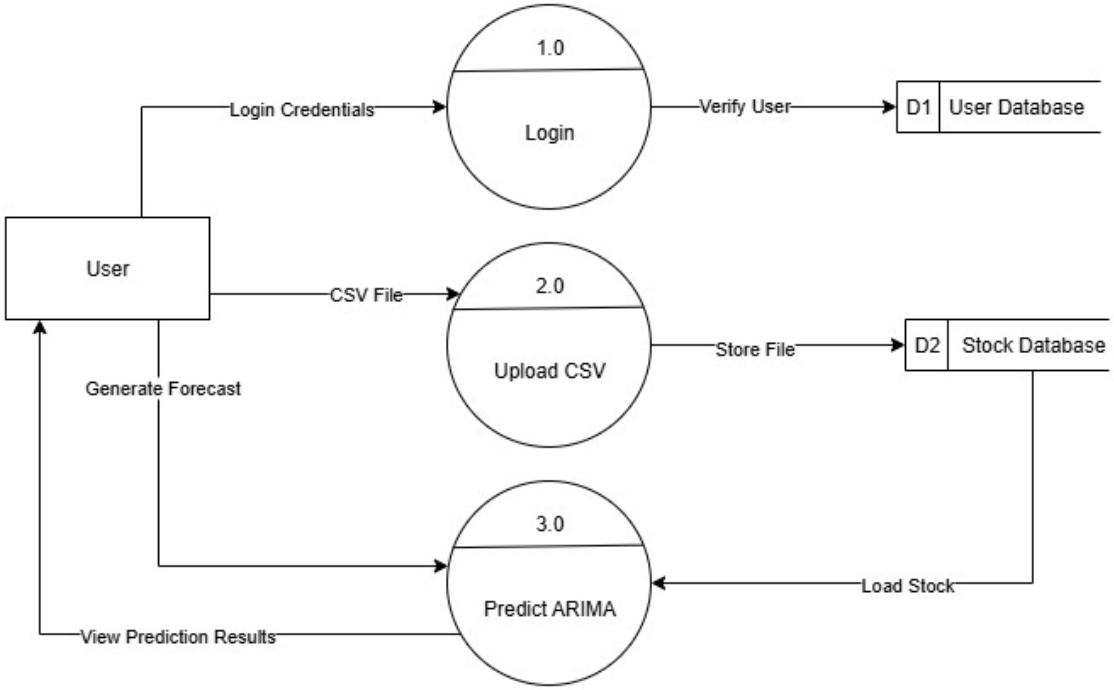


Figure 3.9: Level 1 DFD of Stock Prediction System

This Level 1 DFD represents a stock forecasting system using the ARIMA model. The process starts with the user logging in by submitting their credentials, which are verified using the User Database (D1). After successful login, the user uploads a CSV file containing stock data. This data is stored in the Stock Database (D2). When the user requests a forecast, the system loads the relevant stock data from the database, applies the ARIMA prediction model, and returns the prediction results to the user.

3.3. Algorithm Details

ARIMA is a widely used statistical model for time series forecasting. It works by capturing different aspects of a time-dependent structure: the autoregressive relationship (AR), the differencing to make the data stationary (I), and the moving average component (MA).

The ARIMA model is denoted as ARIMA (p, d, q), where:

- p is the number of lag observations included in the model (autoregression),
- d is the number of times the raw observations are differenced (integration),
- q is the size of the moving average window (moving average).

The model works by first transforming the original data to remove trends and make it stationary, then identifying and estimating the optimal values for p, d, and q, and finally using the fitted model to forecast future values based on past data and error terms.

Working Steps of auto_arima():

The auto_arima() function from the pmdarima library simplifies the ARIMA modeling process by automating the selection of the best p, d, and q parameters.

- Input Time Series: The function takes a univariate time series, typically stock closing prices.
- Stationarity Check: It applies statistical tests (such as the Augmented Dickey-Fuller test) to determine the optimal value of d (the order of differencing).
- Model Selection: It iterates through multiple combinations of p and q, fitting ARIMA (p, d, q) models for each. The best model is selected using information criteria such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).
- Model Fitting: The chosen ARIMA model is trained on the dataset.
- Forecast Generation: Once fitted, the model can forecast future values using methods like .predict().

CHAPTER 4: IMPLEMENTATION AND TESTING

4.1. Implementation

4.1.1. Tools used

- **Programming Languages:**
  - **Python:** Used for implementing the ARIMA prediction algorithm and backend logic.
  - **JavaScript (React.js):** Used for developing the interactive and responsive frontend interface.
- **Backend Framework:**
  - **Django:** Powers the backend server. It handles user interaction, API endpoints, CSV uploads, model execution, and serves processed prediction data to the frontend.
- **Frontend Framework:**
  - **React.js:** Used to build a dynamic single-page application where users can upload stock data and view prediction results with interactive graphs.
- **Python Libraries:**
  - **pandas** – For data loading and preprocessing.
  - **numpy** – For numerical operations.
  - **matplotlib** – For graph generation (optional server-side).
  - **pmdarima** – For training the ARIMA model and predicting future stock prices.
  - **sklearn.metrics** – For calculating performance metrics (MAE, RMSE, R^2).
- **Database Platform:**
  - **PostgreSQL:** Used as the main database to store user data, uploaded CSV file path. Integrated with Django ORM.
- **IDE (Integrated Development Environment):**
  - **Visual Studio Code:** Used for both backend and frontend development with plugins for Python and JavaScript.
- **CASE Tools:**
  - **Draw.io:** Used to design system diagrams including flowcharts, ER diagrams, and architecture.
  - **MS Word:** Used for creating and compiling the full project documentation.

- **Documentation Tool:**
    - **MS Word:** Used to write all report sections, insert diagrams and figures, and format the final submission.

4.1.2. Implementation details of modules

The Stock Prediction System is divided into several key modules, each responsible for a specific part of the application. The following describes the major components and their implementation logic:

- **CSV Upload Module (Django + React):**
    - Frontend allows users to upload CSV files containing stock data (BUSINESS_DATE, CLOSE_PRICE).
    - React handles the file input and sends the file to the backend via a POST request.
    - Django receives the file, validates format, and passes it to the prediction module.
- **Data Preprocessing Module (Python):**
    - Parses the uploaded CSV file using pandas.
    - Converts BUSINESS_DATE to datetime format.
    - Sorts and groups data by date.
    - Infers time-series frequency and fills missing values with forward-fill.
- **ARIMA Prediction Module (run_arima_on_csv_file()):**
    - Splits the data into training and validation sets.
    - Fits the ARIMA model using pmdarima.auto_arima() with parameter optimization.
    - Predicts the validation set and calculates evaluation metrics: MAE, RMSE, R^2.
    - Retrains the model on full data and generates a 7-day future forecast.
    - Returns structured JSON data containing train/test data, predictions, and metrics.
- **API Layer (Django Views):**
    - Handles endpoints which receives the uploaded file, invokes the prediction function, and returns the result to the frontend in JSON format.

- **Frontend Display Module (React.js):**
    - Sends CSV to the backend and receives the prediction data.
    - Displays the actual vs. predicted stock prices using graph libraries.
    - Shows forecasting results and evaluation metrics in a readable format.

Each module is separated by function and interacts via clear interfaces (REST API + JSON), making the system modular, scalable, and easy to maintain.

4.2. Testing

4.2.1. Test cases for Unit testing

Table 4.1: Test cases for Unit Testing

Test Case ID	Description	Input	Expected Output	Actual Result	Status
TC_U1	CSV upload validation	Valid CSV with BUSINESS_DATE, CLOSE_PRICE	File accepted and saved	File uploaded successfully	Pass
TC_U2	Date parsing	CSV with date in %Y-%m-%d format	Parsed as datetime objects	Dates parsed and sorted	Pass
TC_U3	ARIMA parameter tuning	Clean time series	Optimal (p,d,q) selected automatically	Selected: (2,1,1)	Pass
TC_U4	Forecast output length	Forecast 7 days	Array of 7 predicted values	7 predictions returned	Pass
TC_U5	Metric calculation	Actual and predicted values	MAE, RMSE, R^2	MAE: 9.3, RMSE: 12.7, R^2: 0.83	Pass

4.2.2. Test cases for System testing

Table 4.2: Test cases for System Testing

Test Case ID	Description	Input	Expected Output	Actual Result	Status
TC_S1	User login	Valid email & password	Redirect to dashboard	Dashboard loaded	Pass
TC_S2	CSV upload from frontend	Valid NEPSE stock CSV	Success message + file saved	File saved, user notified	Pass
TC_S3	Forecast generation	Click “Predict” after upload	Forecast graph with 7 points	Graph and metrics shown	Pass
TC_S4	Graph rendering	Valid prediction response	Display actual vs predicted line chart	Chart displayed with tooltips	Pass
TC_S5	Upload and forecast for new stock	New stock CSV file	New entry added, new forecast result	New card created with prediction	Pass
TC_S6	Multiple user file uploads	2 different users uploading CSVs	Each sees only their own uploads and results	User-based data isolated correctly	Pass
TC_S7	Backend fails during prediction	Corrupted CSV or backend down	Meaningful error message shown to user	"Internal server error" caught gracefully	Pass

CHAPTER 5: CONCLUSION

5.1. Conclusion

The Stock Prediction System has been fully developed and completed. All core components, including data uploading, ARIMA-based forecasting, backend processing, and the frontend interface, have been successfully implemented, integrated, and tested. The system enables users to upload historical stock data and receive short-term predictions in a clear and visual format.

The project has achieved its intended goals, such as implementing the forecasting logic, building a robust Django backend, and developing a user-friendly frontend using React. All modules now work together seamlessly to provide a smooth end-to-end experience. The system has been tested to ensure functional accuracy and reliability in real-world use cases.

5.2. Lesson Learnt

During the development of the Stock Prediction System, the following key lessons were learned:

- Clean and well-structured data is essential for accurate forecasting.
- ARIMA works well for linear and stationary time series but has limitations.
- Building a full-stack system improved understanding of API integration.
- Error handling and user feedback are crucial for system reliability.
- Iterative development allowed continuous improvement of features and UI.

5.3. Future Recommendations

To enhance the Stock Prediction System further, the following improvements are recommended:

- **Integrate Live NEPSE Data**: Automate data fetching from official NEPSE sources via APIs to remove manual CSV uploads.
- **Support More Models**: Add other forecasting models like SARIMA, LSTM, or Prophet to compare results and improve accuracy.
- **Enable Long-Term Forecasting**: Extend the prediction period beyond 7 days with improved algorithms and seasonal handling.
- **Add User Analytics**: Track user activity and prediction history for personalized recommendations.

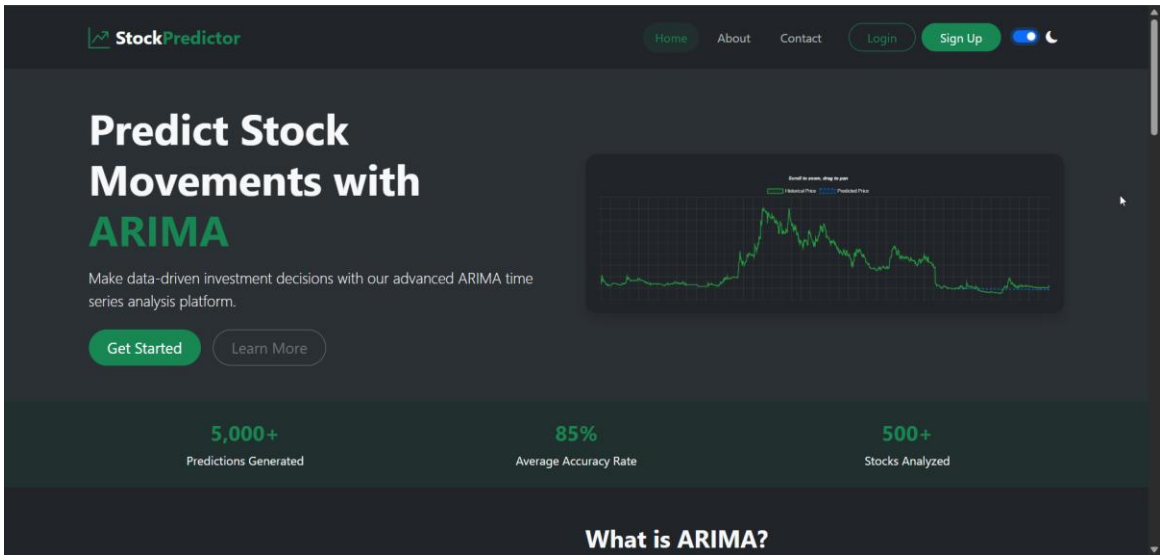
REFERENCES

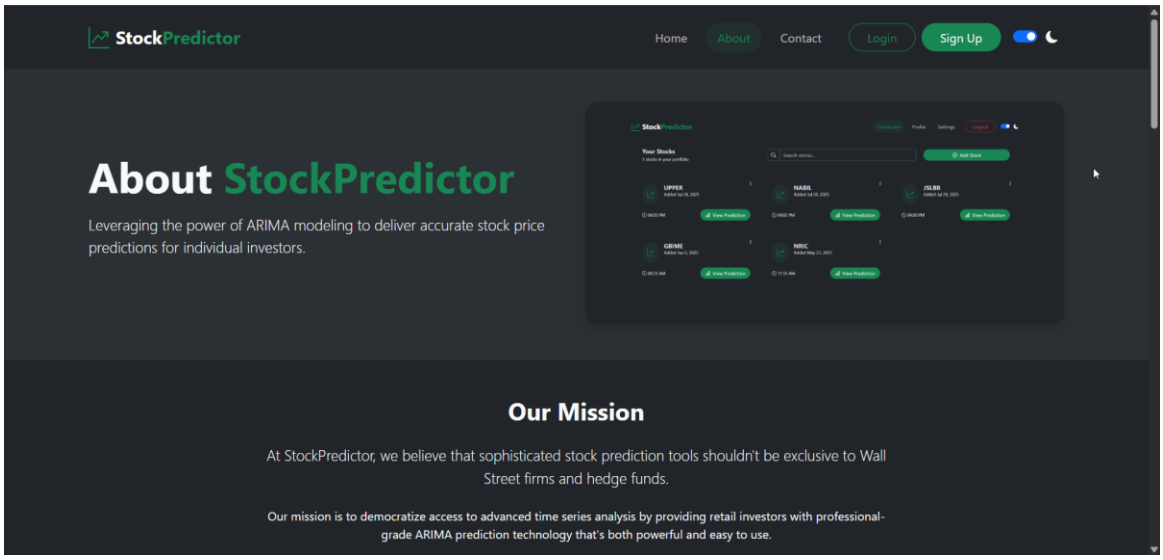
[1] N. M. Adhikari, “Forecasting stock index closing points using ARIMA-GARCH with a rolling data window,” *Int. J. Sci. Res. Arch.*, vol. 13, no. 1, pp. 3115–3125, 2024, doi: 10.30574/ijsra.2024.13.1.1982.

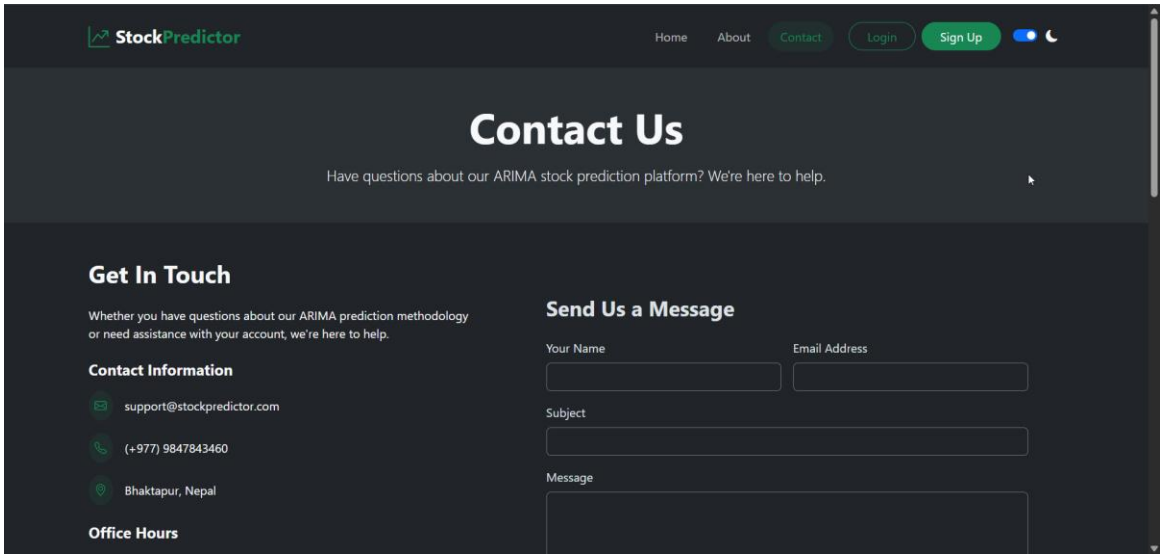
[2] G. T. Wilson, “Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015.,” *J. Time Ser. Anal.*, vol. 37, no. 5, pp. 709–711, 2016.

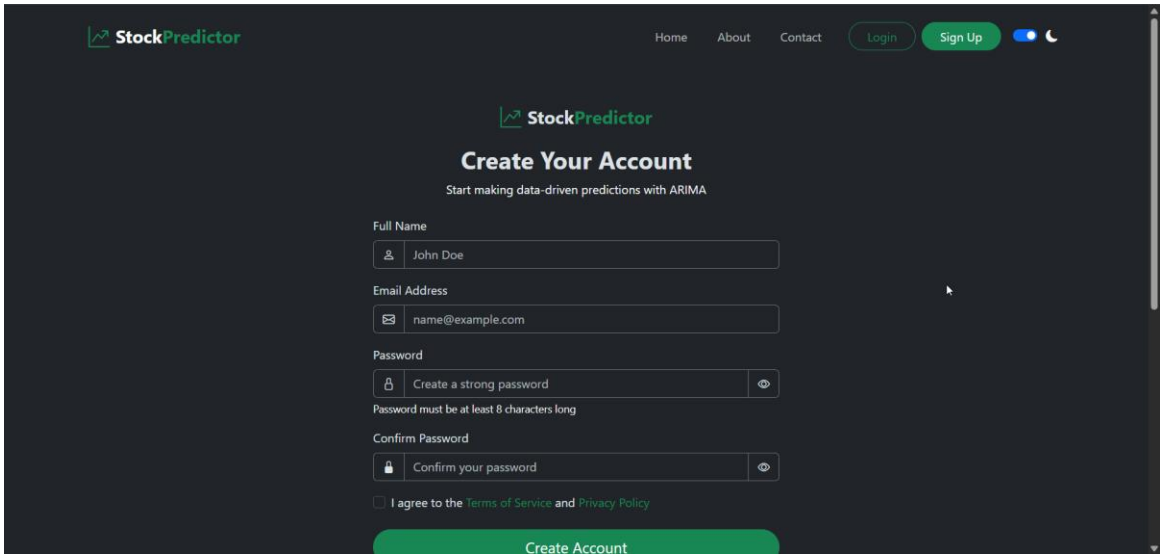
[3] H. N. Gaire, “Forecasting NEPSE Index: An ARIMA and GARCH Approach - the official site of the Central Bank of Nepal,” the official site of the Central Bank of Nepal. Accessed: Feb. 01, 2025. [Online]. Available: https://www.nrb.org.np/er-article/forecasting-nepse-index-an-arima-and-garch-approach/

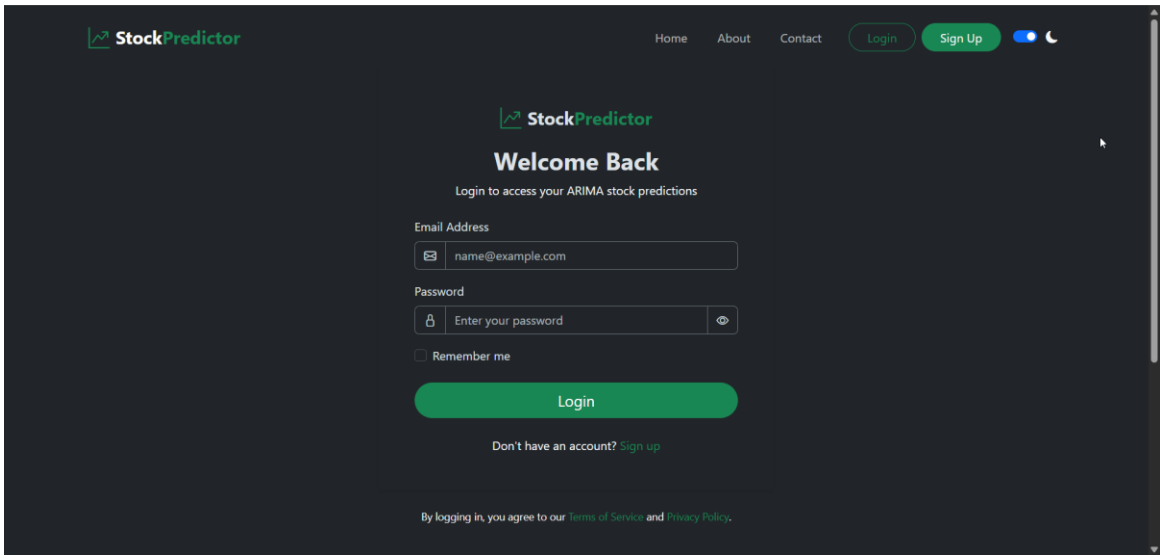
APPENDICES

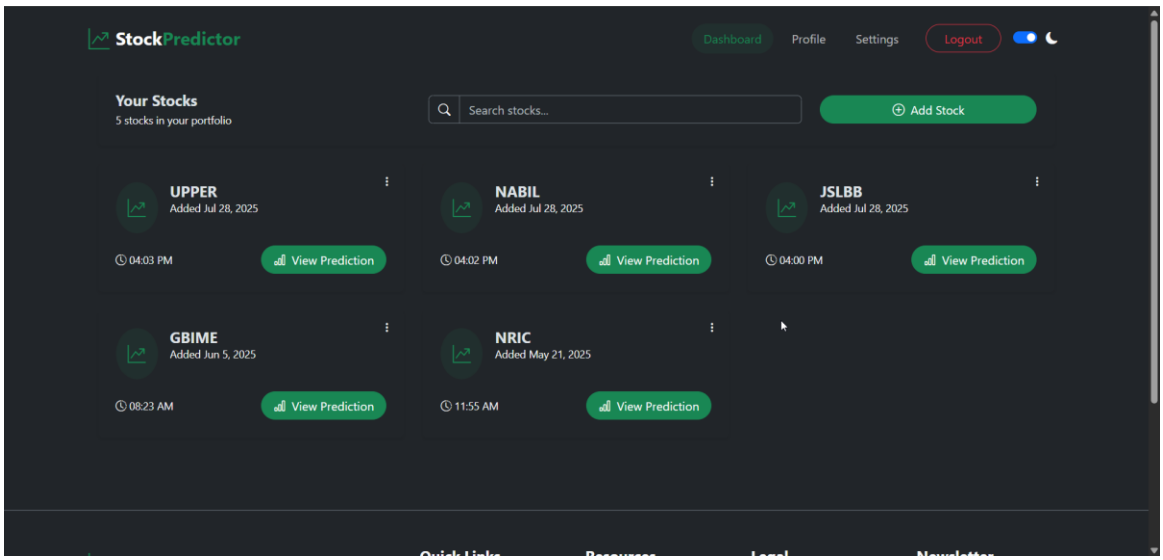

StockPredictor
Home
About
Contact
Login
Sign Up
Predict Stock Movements with ARIMA
Make data-driven investment decisions with our advanced ARIMA time series analysis platform.
Get Started
Learn More
5,000+
Predictions Generated
85%
Average Accuracy Rate
500+
Stocks Analyzed
What is ARIMA?

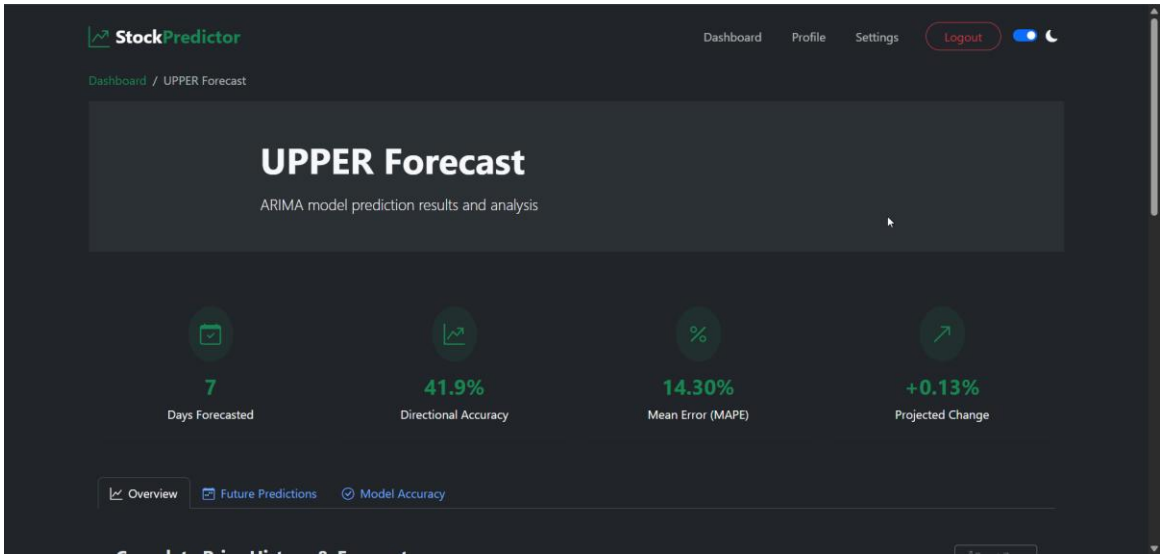

StockPredictor
Home
About
Contact
Login
Sign Up
About StockPredictor
Leveraging the power of ARIMA modeling to deliver accurate stock price predictions for individual investors.
Our Mission
At StockPredictor, we believe that sophisticated stock prediction tools shouldn't be exclusive to Wall Street firms and hedge funds.
Our mission is to democratize access to advanced time series analysis by providing retail investors with professional-grade ARIMA prediction technology that's both powerful and easy to use.


StockPredictor
Home
About
Contact
Login
Sign Up
Contact Us
Have questions about our ARIMA stock prediction platform? We're here to help.
Get In Touch
Whether you have questions about our ARIMA prediction methodology or need assistance with your account, we're here to help.
Contact Information
support@stockpredictor.com
(+977) 9847843460
Bhaktapur, Nepal
Office Hours
Send Us a Message
Your Name
Email Address
Subject
Message

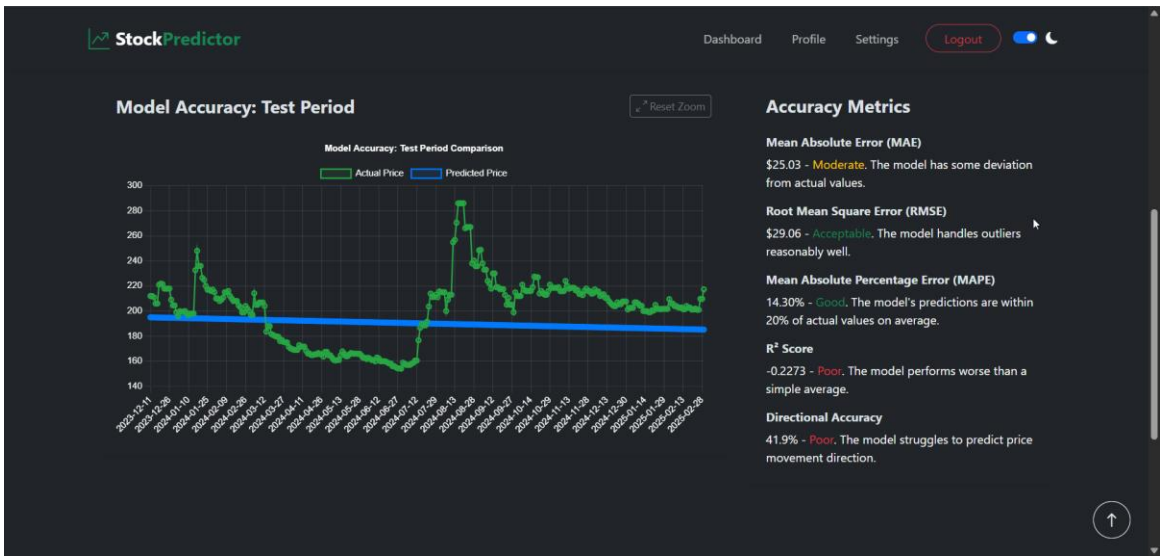

StockPredictor
Home
About
Contact
Login
Sign Up
StockPredictor
Create Your Account
Start making data-driven predictions with ARIMA
Full Name
John Doe
Email Address
name@example.com
Password
Create a strong password
Password must be at least 8 characters long
Confirm Password
Confirm your password
I agree to the Terms of Service and Privacy Policy
Create Account

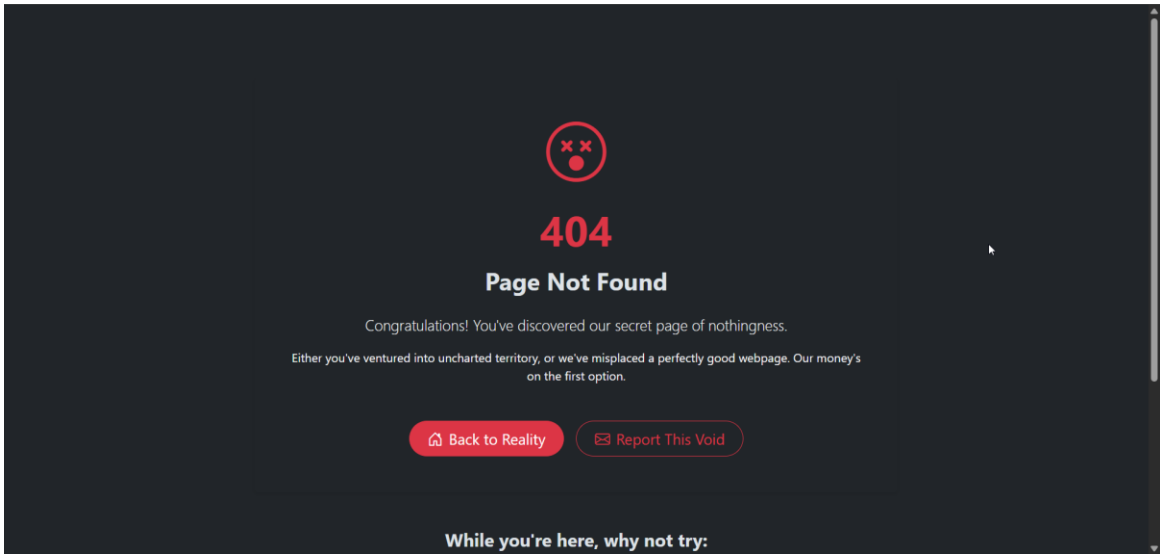

StockPredictor
Home
About
Contact
Login
Sign Up
StockPredictor
Welcome Back
Login to access your ARIMA stock predictions
Email Address
name@example.com
Password
Enter your password
Remember me
Login
Don't have an account? Sign up
By logging in, you agree to our Terms of Service and Privacy Policy.

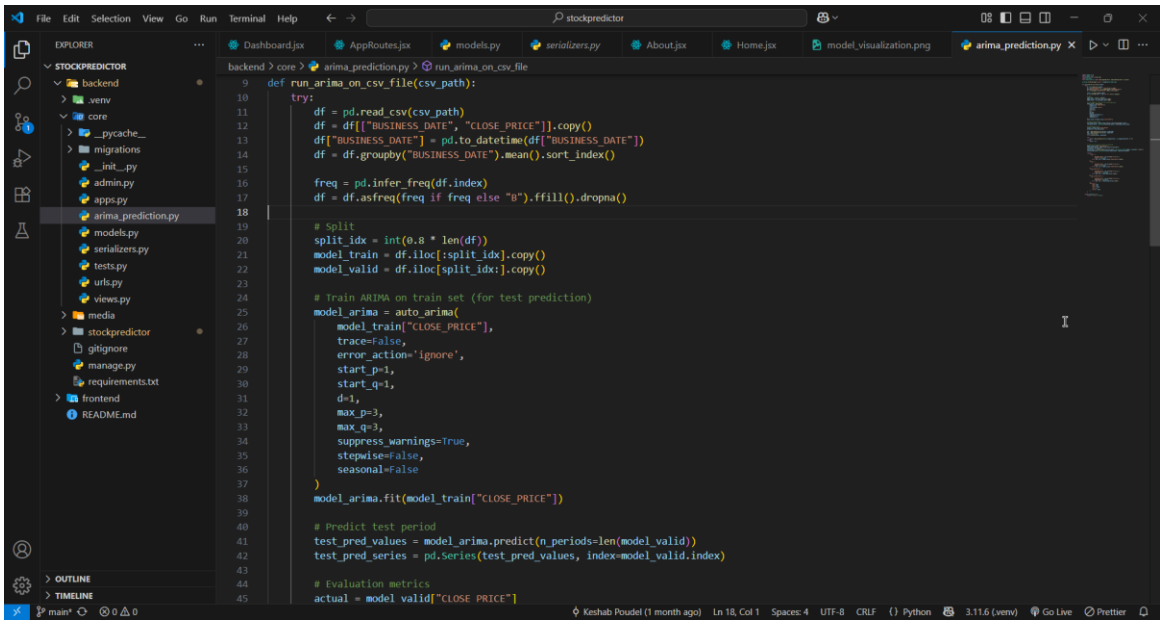

StockPredictor
Dashboard
Profile
Settings
Logout
Your Stocks
5 stocks in your portfolio
Search stocks...
Add Stock
UPPER
Added Jul 28, 2025
04:03 PM
View Prediction
NABIL
Added Jul 28, 2025
04:02 PM
View Prediction
JSLBB
Added Jul 28, 2025
04:00 PM
View Prediction
GBIME
Added Jun 5, 2025
08:23 AM
View Prediction
NRIC
Added May 21, 2025
11:55 AM
View Prediction

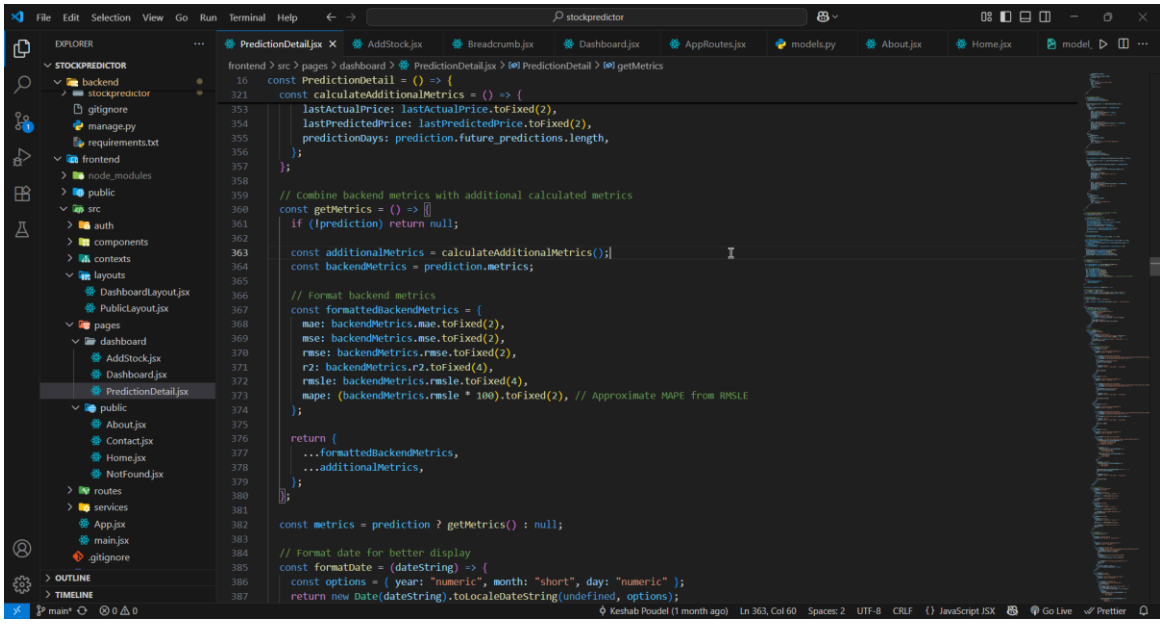

StockPredictor
Dashboard
Profile
Settings
Logout
Dashboard / UPPER Forecast
UPPER Forecast
ARIMA model prediction results and analysis
7
Days Forecasted
41.9%
Directional Accuracy
14.30%
Mean Error (MAPE)
+0.13%
Projected Change
Overview
Future Predictions
Model Accuracy


StockPredictor
Dashboard
Profile
Settings
Logout
Overview
Future Predictions
Model Accuracy
Complete Price History & Forecast
Reset Zoom
Scroll to zoom, drag to pan
Historical Price
Predicted Price


StockPredictor
Dashboard
Profile
Settings
Logout
Model Accuracy: Test Period
Reset Zoom
Model Accuracy: Test Period Comparison
Actual Price
Predicted Price
Accuracy Metrics
Mean Absolute Error (MAE)
$25.03 - Moderate. The model has some deviation from actual values.
Root Mean Square Error (RMSE)
$29.06 - Acceptable. The model handles outliers reasonably well.
Mean Absolute Percentage Error (MAPE)
14.30% - Good. The model's predictions are within 20% of actual values on average.
R² Score
-0.2273 - Poor. The model performs worse than a simple average.
Directional Accuracy
41.9% - Poor. The model struggles to predict price movement direction.


404
Page Not Found
Congratulations! You've discovered our secret page of nothingness.
Either you've ventured into uncharted territory, or we've misplaced a perfectly good webpage. Our money's on the first option.
Back to Reality
Report This Void
While you're here, why not try:


File Edit Selection View Go Run Terminal Help
stockpredictor
EXPLORER
STOCKPREDICTOR
backend
.venv
core
__pycache__
migrations
__init__.py
admin.py
apps.py
arima_prediction.py
models.py
serializers.py
tests.py
urls.py
views.py
media
stockpredictor
gitignore
manage.py
requirements.txt
frontend
README.md
Dashboard.jsx
AppRoutes.jsx
models.py
serializers.py
About.jsx
Home.jsx
model_visualization.png
arima_prediction.py
backend > core > arima_prediction.py > run_arima_on_csv_file
def run_arima_on_csv_file(csv_path):
try:
df = pd.read_csv(csv_path)
df = df[["BUSINESS_DATE", "CLOSE_PRICE"]].copy()
df["BUSINESS_DATE"] = pd.to_datetime(df["BUSINESS_DATE"])
df = df.groupby("BUSINESS_DATE").mean().sort_index()
freq = pd.infer_freq(df.index)
df = df.asfreq(freq if freq else "B").ffill().dropna()
# Split
split_idx = int(0.8 * len(df))
model_train = df.iloc[:split_idx].copy()
model_valid = df.iloc[split_idx:].copy()
# Train ARIMA on train set (for test prediction)
model_arima = auto_arima(
model_train["CLOSE_PRICE"],
trace=False,
error_action='ignore',
start_p=1,
start_q=1,
d=1,
max_p=3,
max_q=3,
suppress_warnings=True,
stepwise=False,
seasonal=False
)
model_arima.fit(model_train["CLOSE_PRICE"])
# Predict test period
test_pred_values = model_arima.predict(n_periods=len(model_valid))
test_pred_series = pd.Series(test_pred_values, index=model_valid.index)
# Evaluation metrics
actual = model_valid["CLOSE_PRICE"]
OUTLINE
TIMELINE
main*
Keshab Poudel (1 month ago) Ln 18, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.6 (venv) Go Live Prettier


File Edit Selection View Go Run Terminal Help
stockpredictor
EXPLORER
STOCKPREDICTOR
backend
stockpredictor
gitignore
manage.py
requirements.txt
frontend
node_modules
public
src
auth
components
contexts
layouts
DashboardLayout.jsx
PublicLayout.jsx
pages
dashboard
AddStock.jsx
Dashboard.jsx
PredictionDetail.jsx
public
About.jsx
Contact.jsx
Home.jsx
NotFound.jsx
routes
services
App.jsx
main.jsx
.gitignore
PredictionDetail.jsx
AddStock.jsx
Breadcrumb.jsx
Dashboard.jsx
AppRoutes.jsx
models.py
About.jsx
Home.jsx
frontend > src > pages > dashboard > PredictionDetail.jsx > PredictionDetail > getMetrics
const PredictionDetail = () => {
const calculateAdditionalMetrics = () => {
lastActualPrice: lastActualPrice.toFixed(2),
lastPredictedPrice: lastPredictedPrice.toFixed(2),
predictionDays: prediction.future_predictions.length,
};
};
// Combine backend metrics with additional calculated metrics
const getMetrics = () => {
if (!prediction) return null;
const additionalMetrics = calculateAdditionalMetrics();
const backendMetrics = prediction.metrics;
// Format backend metrics
const formattedBackendMetrics = {
mae: backendMetrics.mae.toFixed(2),
mse: backendMetrics.mse.toFixed(2),
rmse: backendMetrics.rmse.toFixed(2),
r2: backendMetrics.r2.toFixed(4),
rmsle: backendMetrics.rmsle.toFixed(4),
mape: (backendMetrics.rmsle * 100).toFixed(2), // Approximate MAPE from RMSLE
};
return {
...formattedBackendMetrics,
...additionalMetrics,
};
};
const metrics = prediction ? getMetrics() : null;
// Format date for better display
const formatDate = (dateString) => {
const options = { year: "numeric", month: "short", day: "numeric" };
return new Date(dateString).toLocaleDateString(undefined, options);
OUTLINE
TIMELINE
main*
Keshab Poudel (1 month ago) Ln 363, Col 60 Spaces: 2 UTF-8 CRLF JavaScript JSX Go Live Prettier