# MUTUAL FUNDS PORTFOLIO AND PERFORMANCE ANALYSIS USING ML ALGORITHM

*In the partial fulfillment of the requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**
***In***
**INFORMATION TECHNOLOGY**

SUBMITTED BY

**RUTTALA DIVYA VANI**
**Regd. No: 321107311062**

*Under the esteemed guidance of*

**Dr. K. RAJA KUMAR**

Associate Professor


**DEPARTMENT OF INFROMATION TECHNOLOGY AND COMPUTER APPLICATIONS**

**AU COLLEGE OF ENGINEERING (A)**

**ANDHRA UNIVERSITY**

**2021 – 2025**

# AU COLLEGE OF ENGINEERING(A)

# ANDHRA UNIVERSITY

# VISAKHAPATNAM

## CERTIFICATE

This is to certify that the project report entitled, "MUTUAL FUNDS PORTFOLIO AND PERFORMANCE ANALYSIS USING ML ALGORITHM " is the Bonafide work carried out by **RUTTALA DIVYAVANI** (**321107311062),** a student of **B-Tech** in **AU COLLEGE OF ENGINEERING(A) , ANDHRA UNIVERSITY, VISAKHAPATNAM,** during the year 2021-2025, in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY**.

**Dr. K. RAJA KUMAR**

**Project Guide**

**Prof. KUNJAM NAGESWARA RAO**

**Head of the Department**

# DECLARATION

I declare that the project report entitled "**MUTUAL FUNDS PORTFOLIO AND PERFORMANCE ANALYSIS USING ML ALGORITHM "** has been done by me in partial fulfillment of requirement for the award of degree of "**Bachelor of Technology**", during the academic year 2021-2025 under the guidance of "**Dr. K . RAJA KUMAR**", department of Information Technology and Computer Applications, AU College of Engineering(A), Andhra University, Visakhapatnam. I, here by declare that this project work has not been submitted to any other universities/institutions for the award of any degree.

PLACE : VISAKHAPATNAM

DATE :

RUTTALA DIVYAVANI

(Regd. No :321107311062)

# ACKNOWLEDGEMENT

I Would like to thank all the people who helped us in successful completion of our project

"**MUTUAL FUNDS PORTFOLIO AND PERFORMANCE ANALYSIS USING ML ALGORITHM "**

I would like to thank **Dr. K. RAJA KUMAR**, Associate Professor, Department of Computer Science & System Engineering, Andhra University College of Engineering(A), for his encouragement and valuable guidelines in bringing shape to the dissertation.

I would like to show my greatest appreciation To **Prof. Kunjam Nageswara Rao**, Department of Information Technology & Computer Applications, Andhra University College of Engineering(A),enough for his tremendous support and help without his encouragement and guidance this project Would not have materialized.

I wish to express thanks to **Prof. G. Sasibhushana Rao,** Principal, Andhra University College of Engineering (A), for helping me in completing the project work on time.

We express our sincere gratitude to **Prof. G.P.RAJASEKHAR** vice chancellor of Andhra University, for his keen interest and for providing necessary facilitates and support for this project study.

I would like to thank teaching staff and non-teaching staff members of the department of information technology and computer applications, Andhra University College of Engineering (A), Visakhapatnam, for their constant support in successful completion of my study.
Finally, I express my indebtedness to my beloved parents and friends without whose blessings and encouragement I would not have completed my work fruitfully.

# ABSTRACT

Mutual Fund Portfolio Management plays a vital role in helping investors make informed decisions, maximize returns, and minimize risks. With the rising interest in financial investments, there is a growing need for intelligent systems that assist users in managing mutual fund portfolios efficiently. This system aims to provide a centralized platform for investors to track, analyze and manage their mutual fund investments with ease. The proposed web-based application uses a combination of real-time data visualization and machine learning to offer insights into fund performance.

The system allows users to maintain a transaction history that includes date, NAV (Net Asset Value), investment amount, number of units, and total units held. It supports real-time NAV updates and interactive graphs to visualize fund performance trends. To further aid investment decisions, the system integrates a NAV prediction module that uses historical data and machine learning models to forecast future NAV trends. The objective of the proposed system is to help individual investors make data-driven decisions by tracking their portfolio and anticipating future fund values. The system is built using Python for backend logic, MySQL for database management, and HTML, CSS, and JavaScript for the frontend interface.

This project involves preprocessing historical NAV data and training machine learning models for accurate NAV prediction. The NAV prediction model helps reduce investment risks by giving users a foresight of expected fund behaviour. The complete system enhances user experience through clear visual analytics and predictive tools, making it a reliable solution for personal financial planning.

**TABLE OF CONTENTS** **PAGE NO**

LIST OF FIGURES

# CHAPTER-1
# INTRODUCTION

# CHAPTER-1

# INTRODUCTION

## 1.1 OVERVIEW

In recent years, financial awareness and investment practices have become integral to the way individuals build their wealth. With an increasing number of investors looking for ways to create long-term wealth while managing risks, mutual funds have emerged as a preferred investment vehicle. Mutual funds are managed by professional **Asset Management Companies (AMCs)**, which pool funds from various investors to invest in a diversified portfolio of assets such as **stocks**, **bonds**, **government securities**, and **money market instruments**.

**Types of Mutual Funds**

- **By Asset Class**
  - **Equity Funds**: Invest in stocks, offer high returns, higher risk.
  - **Debt Funds**: Invest in fixed income like bonds and debentures.
  - **Hybrid Funds**: Combine equity and debt instruments for balanced performance.
- **By Structure**
  - **Open-Ended**: Can be bought/sold anytime based on current NAV.
  - **Close-Ended**: Locked-in for a fixed period, tradable on exchanges.
  - **Interval Funds**: Transactions allowed only at specific intervals.
- **By Investment Objective**
  - **Growth Funds**: Reinvest gains for long-term capital appreciation.
  - **Income Funds**: Focus on generating regular income.
  - **Tax-Saving Funds (ELSS)**: Eligible for tax deductions under Section 80C.

Each fund type has a unique risk-return profile and suits different investor needs.

A key term that all investors must understand when it comes to mutual funds is **Net Asset Value (NAV)**. NAV represents the per-unit price of a mutual fund and serves as an essential metric for tracking its performance. NAV is calculated at the end of each trading day using the

formula:

$$\text{NAV} = \frac{\text{Total Assets} - \text{Total Liabilities}}{\text{Total Outstanding Units}}$$

Where:

- **Total Assets** includes the total market value of all investments held by the fund.
- **Total Liabilities** accounts for any fund-related liabilities, such as operating costs, management fees, etc.
- **Total Outstanding Units** refers to the number of units in circulation, which investors hold. NAV represents the price at which investors can buy or sell units of the mutual fund.

Another fundamental concept in mutual fund investing is the **Systematic Investment Plan (SIP)**. **SIP** enables investors to invest a fixed amount at regular intervals, often monthly, into a mutual fund of their choice.

The **number of units purchased** with each SIP installment is calculated as:

$$\text{Units Purchased} = \frac{\text{Amount Invested}}{\text{NAV of the Fund}}$$

The **redemption amount** can be calculated by

$$\text{Redemption Amount} = \text{Units Redeemed} \times \text{Current NAV}$$

One of the core functionalities of **MFTool** is its ability to automate and simplify the tracking of investments. The tool can be integrated with **real-time NAV data feeds**, allowing investors to monitor the performance of their mutual fund investments throughout the day. This integration ensures that investors always have access to the most up-to-date information, which is critical for making informed decisions. As the tool tracks the performance of multiple funds simultaneously, it provides a comprehensive view of an investor's entire portfolio.

This computational approach is designed to develop a robust and efficient **Mutual Fund Portfolio Management System** that empowers users to manage their investments and predict future NAV (Net Asset Value) values accurately. The system integrates real-time mutual fund data collection, portfolio tracking, and **predictive analytics** through the **AMIR (AutoRegressive Moving Average Integrated with Ridge Regression) model**. The objective

is to allow users to track their investments, receive real-time predictions, and optimize their decision-making process for mutual fund investments.

The system comprises several stages, including data acquisition, feature engineering, model development, and evaluation. Each stage plays a pivotal role in ensuring the accuracy and efficiency of the forecasting system.

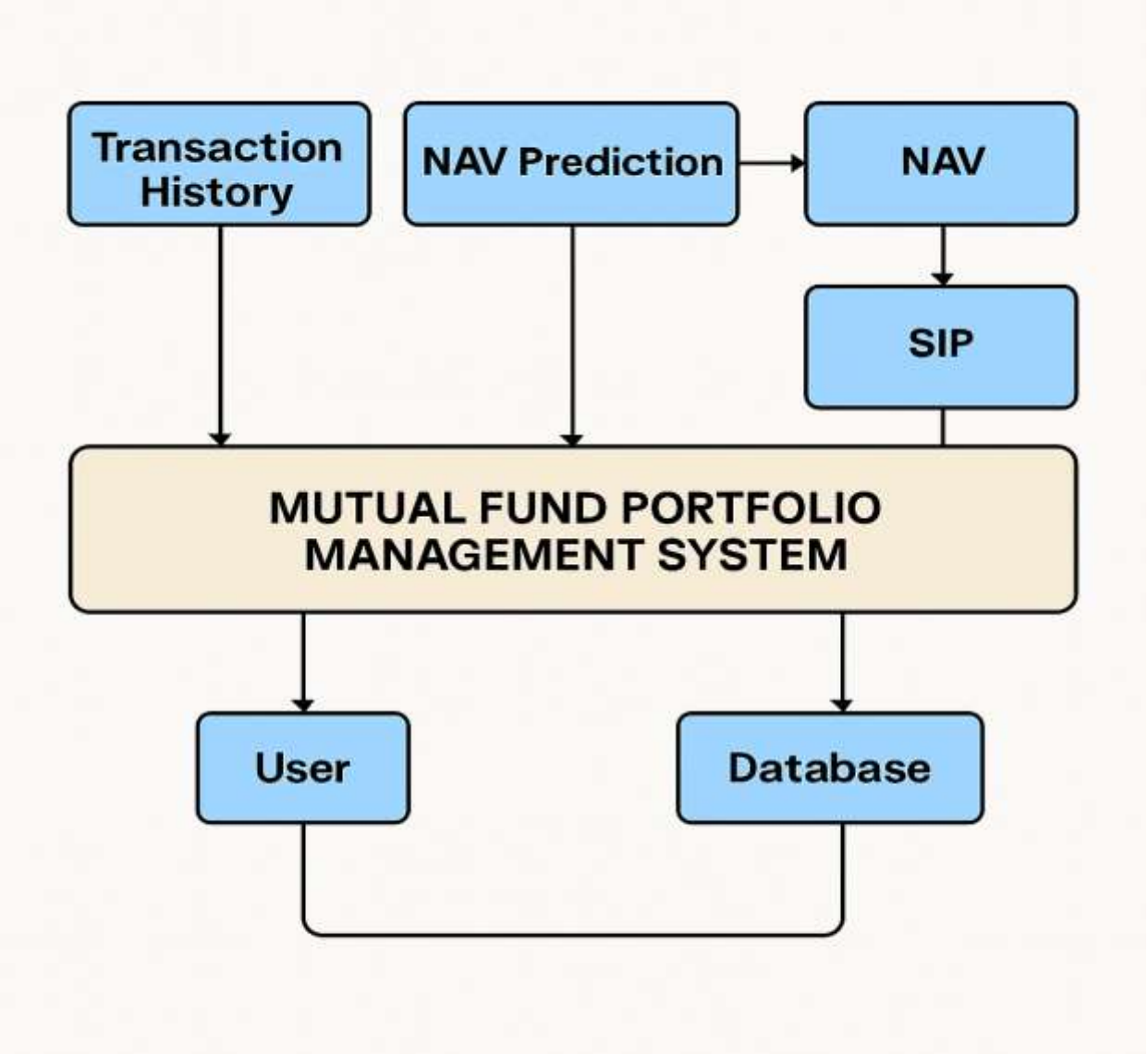



Fig.1.1 Mutual Fund portfolio design

The **Mutual Fund Portfolio Management System** integrates several technologies to deliver a seamless user experience. **Python** is used for backend processing, enabling the system to handle complex calculations, data processing, and machine learning models. **MySQL** is used for managing the database, where users' transactions and portfolio data are securely stored. The frontend of the system is developed using **HTML**, **CSS**, and **JavaScript**, ensuring that the interface is both user-friendly and interactive. These technologies come together to provide an efficient, intuitive platform where investors can track their investments, monitor real-time updates, and make data-driven decisions.

In conclusion, the goal of a Mutual Fund Portfolio Management System is to provide investors with a unified platform to manage and track their mutual fund investments. By combining features like transaction tracking, real-time updates, visual performance insights, and NAV predictions, the system helps investors make more informed decisions. With the integration of machine learning and predictive models, the system goes beyond just monitoring investments, enabling proactive decision-making. The ease of use, coupled with robust data analysis tools, makes such a system an essential tool for both beginner and experienced investors looking to optimize their mutual fund portfolios and achieve their long-term financial goals with confidence

## 1.2 EXISTING SYSTEM

In the current mutual fund management systems, users can only view their investments, buy or sell funds, and check the latest NAVs. These platforms mostly display past performance using charts and graphs but do not provide any prediction or future value estimation. Users have to rely on manual tracking if they have invested through different platforms or offline methods. There is no way to add custom transactions or track unit-wise buy/sell history with clear visuals. Although some tools use basic machine learning algorithms like Decision Tree, Random Forest, Extra Trees, and XGBoost for general fund classification, they are not used for predicting future NAVs or personalized portfolio insights.

## Disadvantages of existing system

**1. No NAV Prediction Capability:** Most existing systems do not offer the ability to predict future NAVs of mutual funds. This restricts users from planning ahead or making informed decisions based on market trends or expected fund performance.

**2. No Custom Transaction Tracking:**

Users who invest through multiple platforms or offline modes often face difficulties in tracking their entire portfolio in one place. The existing systems do not allow manual entry or viewing of offline transactions, leading to an incomplete investment view.

**3. Lack of Visual Buy/Sell Analysis:**

Existing platforms do not display unit-wise buy and sell history with visual indicators. This makes it hard for users to understand their past actions, evaluate returns per transaction, or get insights on their investment behavior over time.

## 1.3 PROBLEM STATEMENT

In today's dynamic financial markets, individual investors face significant challenges in effectively managing their mutual fund portfolios. A lack of personalized tools for tracking investments, accessing real-time insights, and leveraging predictive analytics often results in suboptimal decision-making. Existing platforms fail to empower investors with the actionable intelligence needed to navigate market complexities and optimize their strategies.

To address this gap, this project aims to develop an Intelligent Mutual Fund Portfolio Management System that prioritizes user-centric features and advanced predictive capabilities. The system will allow investors to seamlessly manage transactions .such as adding, removing, buying, or selling funds while maintaining a comprehensive transaction history and delivering real-time portfolio statistics.

This innovative solution bridges the gap between traditional portfolio management tools and intelligent financial forecasting, empowering investors with a modern platform to effectively manage and grow their mutual fund investments.

### 1.3.1 SOLUTION

This project aims to solve the problems faced by individual investors in managing their mutual fund investments. Most existing tools don’t provide real-time updates or smart features, making it hard for investors to make informed decisions. To solve this, we are building an Intelligent Mutual Fund Portfolio Management System that helps users track their investments easily and get useful predictions.The system allows users to add, remove, buy, or sell mutual funds. Every transaction is recorded with details like amount, NAV, units, and date. Users can view their full transaction history, which helps in understanding past investments and current holdings. The system also shows the total value of the portfolio using the latest NAVs.

One of the best features is that the system uses machine learning to predict future NAV values of funds. This helps users plan when to buy or sell a fund. The predictions are shown in simple graphs for better understanding. The system also gives alerts when the NAV reaches a certain level, helping users take action quickly.

The platform is built using Python, MySQL, HTML, CSS, and JavaScript. It is designed to be easy to use, secure, and expandable. Overall, this project gives investors a smart and simple way to manage their mutual fund investments and make better financial decisions

## 1.4 PROPOSED SYSTEM

The proposed system is a web-based Mutual Fund Portfolio Management platform integrated with intelligent prediction capabilities. It is designed to assist investors in efficiently managing their mutual fund holdings while gaining actionable insights into future performance trends.
This system enables users to perform key portfolio operations such as adding or removing funds, executing buy and sell transactions, and tracking real-time NAV updates. It maintains a comprehensive transaction history and dynamically calculates the total investment, units held, and current value of the portfolio.

A major enhancement is the inclusion of a machine learning-based **future NAV prediction module** powered by the **ARIMA model** (AutoRegressive Moving Average Integrated with Ridge Regression). By analyzing historical NAV data, the system forecasts the potential future NAVs of mutual funds, allowing users to make informed and proactive investment decisions.
The system architecture includes:

- **Frontend**: Built using HTML, CSS, and JavaScript for an interactive user interface.
- **Backend**: Developed in Python for data processing and ML model integration.
- **Database**: MySQL for managing user data, fund transactions, and NAV records.
- **Visualization**: Graphical representation of fund performance trends and prediction results.

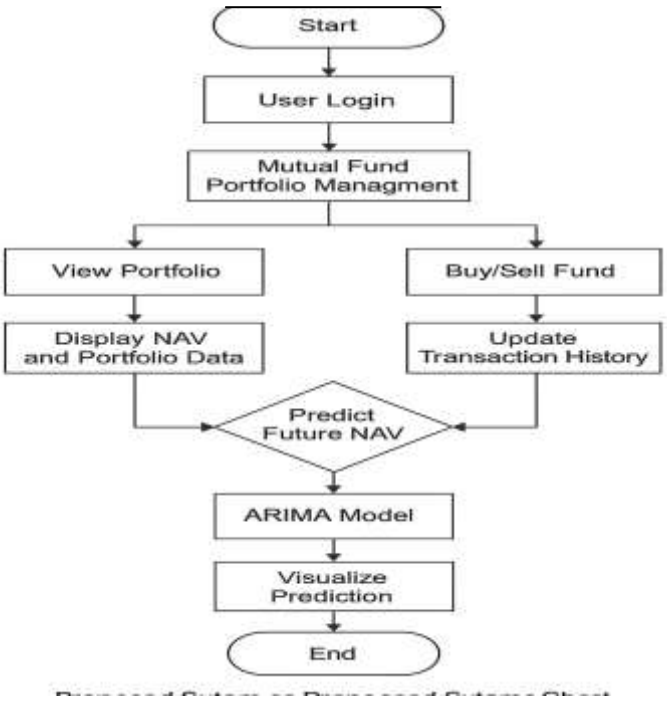



**Fig 1.2 Proposed system**

**Objectives of the Proposed System**

- Provide a **user-friendly interface** for managing mutual fund investments.
- Allow users to **track real-time NAVs** and portfolio valuation.
- Enable **Buy/Sell** operations with proper transaction history tracking.
- Use the **ARIMA model** to predict **future NAV trends**.
- Visualize historical and predicted data through **interactive graphs**.
- Maintain secure and organized data storage using **MySQL**.
- Support users in making **data-driven investment decisions**.
- Promote better **financial planning and awareness** through smart forecasting.

# CHAPTER-2
# LITERATURE SURVEY

# CHAPTER-2

# LITERATURE SURVEY

**K. R. Rajeswari and Dr. S. S. R. Abidi** explain that with the growing adoption of mutual fund investments among retail investors, there is a critical need for systems that provide analytical and comparative insights. Their proposed model enables users to analyze historical NAV data, compare mutual fund schemes, and understand performance metrics like the Sharpe ratio and standard deviation. However, they acknowledge a major limitation—the absence of a predictive component that could guide investors on future NAV trends, recommending that machine learning techniques be incorporated for a more intelligent decision-support framework.

**P. Bhargavi and Dr. R. Vasanthi** focus on the performance of machine learning models in financial time-series forecasting. Their study compares ARIMA, SVR, and LSTM for predicting NAV values, showing that hybrid models outperform traditional statistical or standalone ML models. The authors highlight the effectiveness of ARIMA in capturing trend and seasonality, while Ridge regression adds regularization benefits to reduce overfitting. They advocate for the ARIMA model (ARIMA + Ridge Regression) as a promising approach, especially for noisy and non-stationary financial datasets like mutual fund NAVs.

**A. Sharma and R. Jain** present a mutual fund prediction system using Linear Regression and Random Forest algorithms. They use historical AMFI NAV data to predict fund growth and assist investors in choosing high-performing schemes. Although their system performs well on recent data, it lacks the ability to handle time-series dependencies, resulting in lower long-term forecasting accuracy. They suggest incorporating models like ARIMA or Prophet to improve time-based predictions, and they acknowledge the value of blending statistical and ML methods to enhance reliability.

**S. A. Parameshwar and G. U. Kiran** designed a mutual fund simulator that enables investors to manually manage a virtual portfolio and track returns. While the system includes useful features like fund categorization, asset allocation visualization, and risk profiling, it is limited to descriptive analytics. The authors recommend integrating forecasting capabilities to help users make decisions based on future fund performance, transforming the platform into a proactive advisory system rather than a static tracker.

**M. Shyam Sundar and R. Venkat** explore and compare the accuracy of ARIMA, Prophet, and Ridge Regression for NAV forecasting. ARIMA models excel at identifying trends in stationary data but suffer from overfitting when used alone. Ridge Regression, on the other hand, handles multicollinearity and stabilizes predictions in high-dimensional datasets. Their findings support a hybrid strategy like ARIMA, which benefits from the time series capabilities of ARIMA and the regularization strengths of Ridge Regression. They emphasize the importance of using ensemble models to manage the complexity of financial data.

**S. Vishwakarma and R. Banerjee** built a full-stack mutual fund tracking application using Python and MySQL. The application allows users to add and remove investments, monitor unit values, and track portfolio performance using interactive dashboards. Although the UI/UX and database components are well-developed, the system lacks predictive modeling features. The authors conclude that integrating machine learning for NAV forecasting and real-time AMFI API feeds would substantially enhance the platform's utility and allow for smarter investment planning.

**P. Rajasekar and K. Meenakshi Sundaram** examine the use of neural networks for mutual fund NAV forecasting. Their model uses a multilayer perceptron (MLP) trained on multiple fund types (equity, hybrid, debt) and historical economic indicators. The study shows that deep learning models can capture non-linear relationships and improve NAV forecasting accuracy. However, they caution that these models require large datasets and proper tuning, making hybrid approaches like ARIMA more practical in data-constrained environments.

**D. Chakraborty and A. Talukdar** present a comparative study on traditional forecasting models like Exponential Smoothing and ARIMA versus machine learning models like Gradient Boosting and XGBoost. They find that while boosting algorithms perform well in short-term predictions, ARIMA-based models show better consistency over longer time periods, especially when combined with a regularized regressor like Ridge. Their work reinforces the importance of hybrid models for balancing prediction stability with responsiveness to recent trends.

**N. Gupta and R. Agarwal** analyze investor behavior and propose a recommendation engine that suggests mutual funds based on past NAV performance and user risk appetite. Though their focus is on personalization and user engagement, they recognize that without forecasting,

recommendations are backward-looking. Their future scope highlights the importance of predictive analytics in transforming advisory systems from reactive to proactive.

**S. Rathi and P. Tiwari** develop a mutual fund portfolio manager that integrates real-time NAV updates using AMFI APIs and visualizations through Matplotlib and Dash. Their system enables tracking of fund performance, expense ratio, and asset allocation. However, they report that user retention improves significantly when systems include forecasting, automatic alerts, and goal-based planning. They advocate for embedding machine learning models to predict NAVs and generate insights that can drive investor confidence.

# CHAPTER-3
# SYSTEM ANALYSIS AND DEIGN

# CHAPTER-3
# SYSTEM ANALYSIS AND DESIGN

## 3.1 INTRODUCTION

System Analysis is the foundational phase in the System Development Life Cycle (SDLC), where the functional and technical requirements of the system are studied and documented. In the context of the Mutual Fund Portfolio Management and NAV Prediction System, system analysis involves a comprehensive examination of the current methods used by investors and institutions to track mutual fund investments, analyze fund performance, and estimate future Net Asset Value (NAV). This phase aims to understand the existing manual or semi-automated processes, identify inefficiencies, and determine how a machine learning-powered solution can add value.

During analysis, inputs were gathered through research papers, market tools (like MFTool and AMFI), and user interactions to understand portfolio tracking behaviors. Various logical models, such as data flow diagrams and use-case scenarios, were employed to define how information flows through the proposed system. Special attention was given to user roles, data integration (like live AMFI feeds), and prediction models (like the ARIMA model). The analyst's role in this phase is critical, requiring technical experience, system design knowledge, and domain familiarity with financial data handling.

**Feasibility Study:**

A key outcome of this analysis phase is establishing whether the proposed system is feasible across several dimensions technical, economic, and operational.

**Technical Feasibility:**

The proposed Mutual Fund Portfolio Management and NAV Prediction System is technically feasible. The system leverages standard web technologies such as Python (Flask/D), HTML, CSS, JavaScript, MySQL for backend storage, and libraries like Pandas, NumPy, and Scikit-learn for machine learning. The hardware requirements are minimal, and the software stack is open-source, making it well-aligned with the technical capabilities of most development and

hosting environments. Since the ARIMA model combines time series forecasting with Ridge Regression, it is computationally efficient and can be integrated into the system for near real-time predictions without requiring advanced GPUs.

**Economic Feasibility:**

From an economic standpoint, the system is highly feasible. The tools and libraries required for development (e.g., Python, Scikit-learn, MySQL) are free and open-source. Hosting can be done on low-cost cloud platforms or local servers. No proprietary software licenses are necessary. Additionally, the development does not require large financial outlays, and maintenance costs are minimal. The system adds economic value by helping users make informed investment decisions and track their portfolio efficiently, potentially increasing their returns and confidence.

**Operational Feasibility:**

The operational feasibility is strongly positive. Users (both beginner and experienced investors) will benefit from an intuitive interface that allows them to add or remove mutual funds, view real-time NAV updates, and visualize past trends and future predictions. Since the system includes graphical representations, unit tracking, and buy/sell transaction history, it simplifies mutual fund management significantly. Moreover, the system is designed to integrate live NAV feeds and provide daily insights, ensuring user engagement and satisfaction. No major user resistance is expected due to its user-friendly interface and clear utility.

## 3.2 SYSTEM REQUIREMENTS

An SRS (System Requirements Specification) outlines the expectations for the software system to be developed. It serves as a mutual agreement between the client and the developer, and it is a critical part of the software development lifecycle (SDLC). This document includes the functional and non-functional requirements that define what the system must do, as well as the hardware and software specifications.

### 3.2.1 Functional Requirements

The **functional requirements** describe the specific behaviors, services, and operations that system must provide.

1. **Data Preprocessing:**
    - The system will preprocess mutual fund data to ensure it is in a format suitable for machine learning algorithms, including removing missing values, handling outliers, and normalizing numerical features.
2. **Machine Learning Model Selection:**
    - The system will select an appropriate machine learning model, such as the ARIMA model (ARIMA + Ridge Regression), for predicting future NAV (Net Asset Value) based on historical data.
3. **Model Training:**
    - The system will use the preprocessed data to train the selected machine learning model. It will ensure that the training data is split into training and validation sets for better model accuracy.
4. **Model Evaluation:**
    - The system will evaluate the performance of the machine learning model using performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared values.
5. **Real-Time Portfolio Updates:**
    - The system will automatically update the portfolio with the latest NAV values by integrating real-time data from AMFI (Association of Mutual Funds in India).
6. **Transaction Management:**
    - The system will allow users to buy and sell funds, updating their portfolio accordingly with real-time NAV adjustments, and track transaction data like date, amount, units, and total units.
7. **Visualization:**
    - The system will provide visual graphs of the fund's historical performance and future predictions, including trends and risk assessments.

8. **User Authentication:**
    - The system will authenticate users through a secure login process and provide role-based access control (admin, user, guest).
9. **Portfolio Management:**
    - The system will allow users to add/remove mutual funds, view their portfolio's details, and perform transactions (buy/sell).
10. **Notifications:**
    - The system will generate notifications or alerts to inform users about significant changes in fund performance or transaction updates.

### 3.2.2 Non-functional Requirements

Non-functional requirements define the system's quality attributes, such as performance, reliability, and scalability.

1. **Performance:**The system should operate efficiently with minimal latency, ensuring that portfolio updates and predictions occur without noticeable delay.

2. **Reliability:**The system must be highly reliable with minimal downtime, ensuring continuous availability for users to manage their portfolios.

3. **Scalability:**The system should be able to handle an increasing number of users, mutual funds, and transactions without degrading performance.

4. **Security:**The system should implement secure authentication, data encryption, and other security measures to protect user information and transaction history.

5. **Usability:**The system should have an intuitive and user-friendly interface, making it easy for users to add funds, track their portfolios, and access NAV updates.

6. **Compliance:**The system must adhere to relevant data protection laws and regulations, such as GDPR (General Data Protection Regulation) or other financial regulations applicable to the users' location.

7. **Maintainability:**The system should be easy to maintain and update, with the flexibility to incorporate new features or handle changes in regulations.

8. **Availability:**The system should be highly available, ideally with redundancy mechanisms in place to ensure uninterrupted access to the platform, even during hardware or software failures.

9. **Scalability:**The system should be capable of expanding its capacity to handle a growing number of users, funds, and transactions as needed.

10. **Accuracy:**The system should ensure that NAV predictions are accurate and minimize false positives and false negatives in portfolio performance evaluations.

### 3.2.3 Hardware Requirements

- RAM: 4GB and Higher
- Processor: Intel i3 and above
- Hard Disk: 500GB: Minimum

### 3.2.4 Software Requirements

**Operating System:** Windows or Linux

**Python IDE:** Python 3.6.x or above

**Required IDEs:** PyCharm IDE, Jupyter Notebook

**Libraries:**

- Pandas, NumPy for data manipulation
- Flask for web framework
- Matplotlib for data visualization

**Database:** MySQL or PostgreSQL for storing portfolio data

**Libraries for Real-Time Data:** Requests (for API integration with AMFI)

## 3.3 SYSTEM ARCHITECTURE

This architecture represents how the Intelligent Mutual Fund Portfolio and Prediction System is structured and how the components interact:

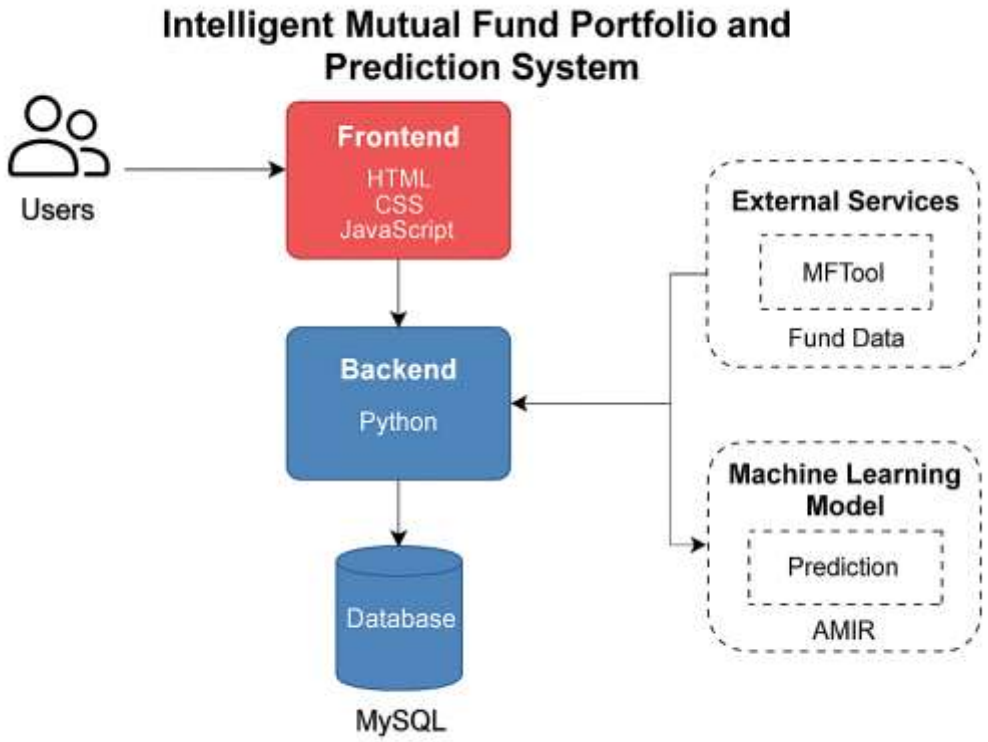



**Fig 3.1 system architecture**

**WORK FLOW:**

1. Users: Users interact with the system through a web interface. They perform actions like logging in, buying/selling funds, and checking their portfolio.

2. Frontend (HTML, CSS, JavaScript): This is the part of the system that users see. It's built using HTML, CSS, and JavaScript. It allows users to input data, view transactions, and see real-time statistics and graphs.

3.Backend (Python): This handles the business logic. It receives user requests from the frontend, processes them, interacts with the database, fetches data from external services, and responds back to the frontend.

4. Database (MySQL): The MySQL database stores all the important data, such as:

- User details
- Fund names and NAVs
- Buy/sell transactions
- Total units and amounts
- Prediction results

5. External Services (MFTool / Fund Data): The backend fetches latest fund data (like current NAVs) using tools like MFTool or APIs. This ensures real-time values are shown in the portfolio.

6.Machine Learning Model (AMIR): This module predicts future NAVs based on historical data. The backend passes data to this model and displays predicted results (like NAV trends) in the frontend.

## 3.4 DATA FLOW DIAGRAM

This data flow diagram illustrates how a mutual funds portfolio system manages investment data, analyzes performance, and leverages historical insights to predict future trends. By integrating real-time portfolio data with predictive analytics, the system supports smarter investment decisions, enhances risk management, and aims to maximize returns through informed forecasting

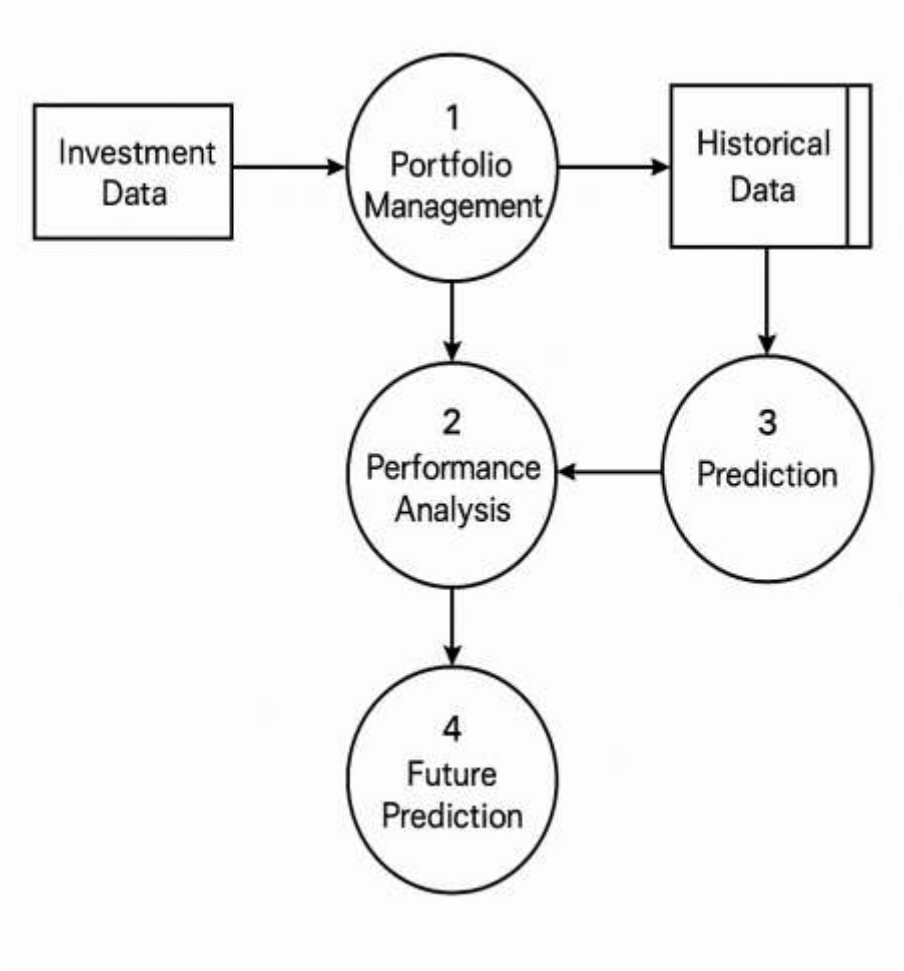



**Fig.3.2 Data Flow Diagram**

**1. Portfolio Management:**

This module gathers and organizes **investment data** such as fund types, asset allocation, transaction history, and current holdings. It helps maintain and structure the mutual fund portfolio for an investor.

**2. Performance Analysis:**

This stage uses the data from the portfolio to assess how each fund is performing. It evaluates returns, risk levels, and diversification to understand strengths and weaknesses in the investment strategy.

**3. Prediction:**

Here, **historical data** is used along with machine learning or statistical models to identify trends and patterns. It analyzes past behaviors to generate insights that feed into forecasting.

**4. Future Prediction:**

Based on performance analysis and prediction models, this module projects potential future outcomes. It estimates returns, suggests optimal allocation, and provides risk assessments to guide better decision-making.

## 3.5 ENTITY RELATIONSHIP(ER)-DIAGRAM

The ER diagram shows how the Mutual Funds Portfolio system organizes its core data. A portfolio manages various mutual funds, each linked to investment data. This structured data is then used for making informed predictions. The relationships between entities help define the logic needed to assess current investments and forecast future outcomes, supporting intelligent and strategic financial decisions.

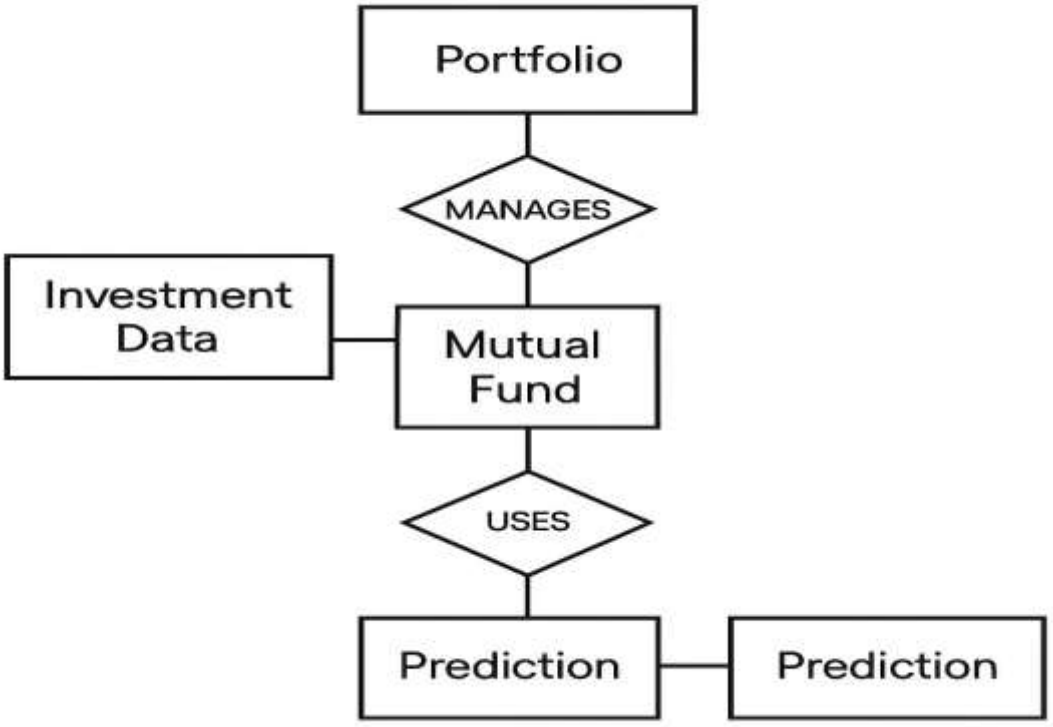



**Fig.3.3 ER Diagram**

## 3.6 DESIGN APPROACHES

Software design is a critical part of the software development process, sitting at the core of software engineering. It begins once the system requirements have been specified and analyzed. The design phase is a pivotal step as it serves as a blueprint for the development process, guiding the code implementation and testing phases. The goal is to produce a model or representation of the software system that will later be developed. A well-executed design ensures that the system meets customer expectations, has high quality, and is scalable and maintainable.

Design is the stage where quality is established. By translating customer requirements into a structured design, developers can ensure that the software is robust, stable, and capable

of handling future changes or expansions. The design phase is critical because it directly affects the quality of the software, impacting later phases like testing and maintenance.

The output of this phase is the **design document**, which serves as a detailed guide for the implementation, testing, and maintenance of the system.

## 3.6.1 OBJECT-ORIENTED APPROACH (OOA)

Object-Oriented Approach (OOA) is a software design technique where we represent real-world entities as objects. Each object contains data (attributes) and functions (methods).
In this project, we are using OOA to build a system that can:

- Manage mutual fund investments (buy/sell)
- Track user portfolio and transaction history
- Predict future NAV (Net Asset Value) using machine learning

**Object Oriented Concepts**

**1. Objects**

- Real-world entities like User, Fund, Transaction, Portfolio.
- Each object has **attributes** (data) and **methods** (operations).

**2. Attributes (Properties)**

- These define the state of the object. For example:
    - User → name, user_id
    - Fund → fund_id, nav_history
    - Transaction → date, amount

**3. Methods (Behaviors)**

- These define what operations can be performed on an object. For example:
    - **User** can view_portfolio(), make_transaction().
    - **Fund** can get_nav(), update_nav_history().
    - **Transaction** can record_buy(), record_sell().

**4. Relationships**

- Relationships define how objects are connected to each other. Examples:
    - A **User** has one **Portfolio**.
    - A **Portfolio** contains multiple **Transactions** and multiple **Funds**.
    - **NAVPredictor** uses **Fund** to get NAV history for prediction.

## 3.7 UML DESIGN OVERVIEW

A graphical tool used to describe and analyze the moment of data through a system manual orautomated including the process, stores of data, and delays in the system. Data Flow Diagrams are thecentral tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physicalcomponents associated with the system. The DFD is also known as a data flow graph or a bubble chart. DFDs are the model of the proposed system. Later during design activity this is taken as the basis for drawingthe system's structure charts.

In the context of designing a **Mutual Fund Portfolio and Future NAV Prediction System**, UML provides a structured approach to model the portfolio tracking, fund transactions (buy/sell), real-time NAV integration, and future NAV prediction using the ARIMA model (ARIMA + Ridge Regression). This modeling technique allows clear visualization of user interactions, data flow, backend logic, and system integration.

### 3.7.1 Class Diagram

Class diagrams are the backbone of almost every object-oriented method including UML. Theydescribe the static structure of a system.

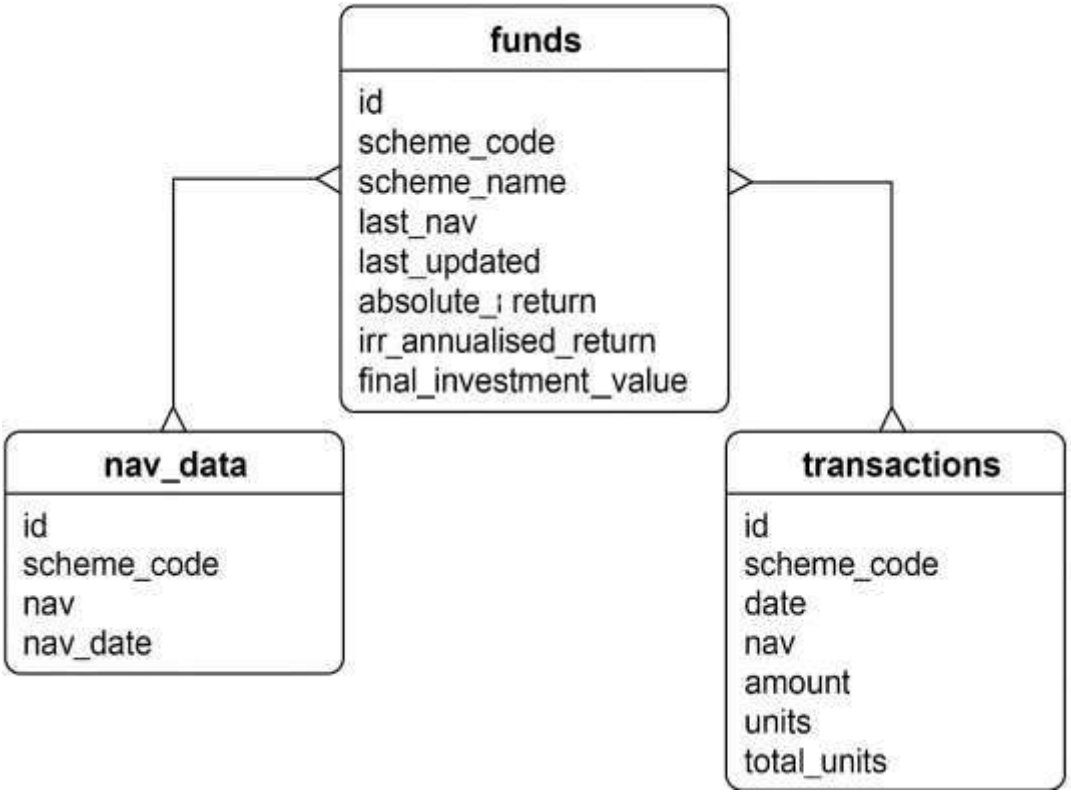



**Fig 3.4 Class Diagram of mutual funds portfolio**

**Classes and Their Attributes/Methods:**

**1. Class: Fund**

The Fund class is responsible for representing each mutual fund scheme in the system. It contains attributes such as the scheme_code, scheme_name, last_nav (latest Net Asset Value), and timestamps indicating when the data was last updated. It also stores computed investment metrics like the absolute return, internal rate of return (IRR), and final investment value. These attributes help track the performance of a particular fund. The methods defined in this class allow the system to update NAV values, calculate returns, and generate investment summaries for display to the user. This class plays a critical role in summarizing the overall value and profitability of each mutual fund in the user's portfolio.

**Methods:**

- update_nav(new_nav, date)
- calculate_absolute_return()
- calculate_irr()
- get_investment_summary()

**2. Class: NAVData**

The NAVData class manages the historical Net Asset Values of all mutual fund schemes. This information is essential for analyzing fund performance trends over time and feeding data to the prediction model. The class includes methods to fetch the latest NAV of a fund, retrieve NAVs on specific dates, and return the full NAV history of a scheme. This allows the system to offer both real-time updates and historical analysis. By structuring NAV data separately, the system ensures that forecasting, graphs, and reports are accurate and based on organized data.

**Methods:**

- get_latest_nav(scheme_code)
- get_nav_by_date(scheme_code, date)
- get_nav_history(scheme_code)

**3. Class: Transaction**

The Transaction class keeps track of every buy or sell action made by the user. Each transaction is recorded with details such as the fund involved (identified by scheme_code), the date, the NAV at that time, the amount transacted, and the units bought or sold. It also calculates and updates the total units held after each transaction. The class includes methods to handle unit calculations during purchases and sales, and to summarize transaction history for each scheme. This class is central to maintaining a complete and accurate investment history, enabling the

system to provide current holdings and their value at any point in time.

**Methods:**

- buy_units(scheme_code, amount)
- sell_units(scheme_code, amount)
- calculate_units(nav, amount)
- get_transaction_summary(scheme_code)
- get_total_units(scheme_code)

### 3.7.2 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processesor objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

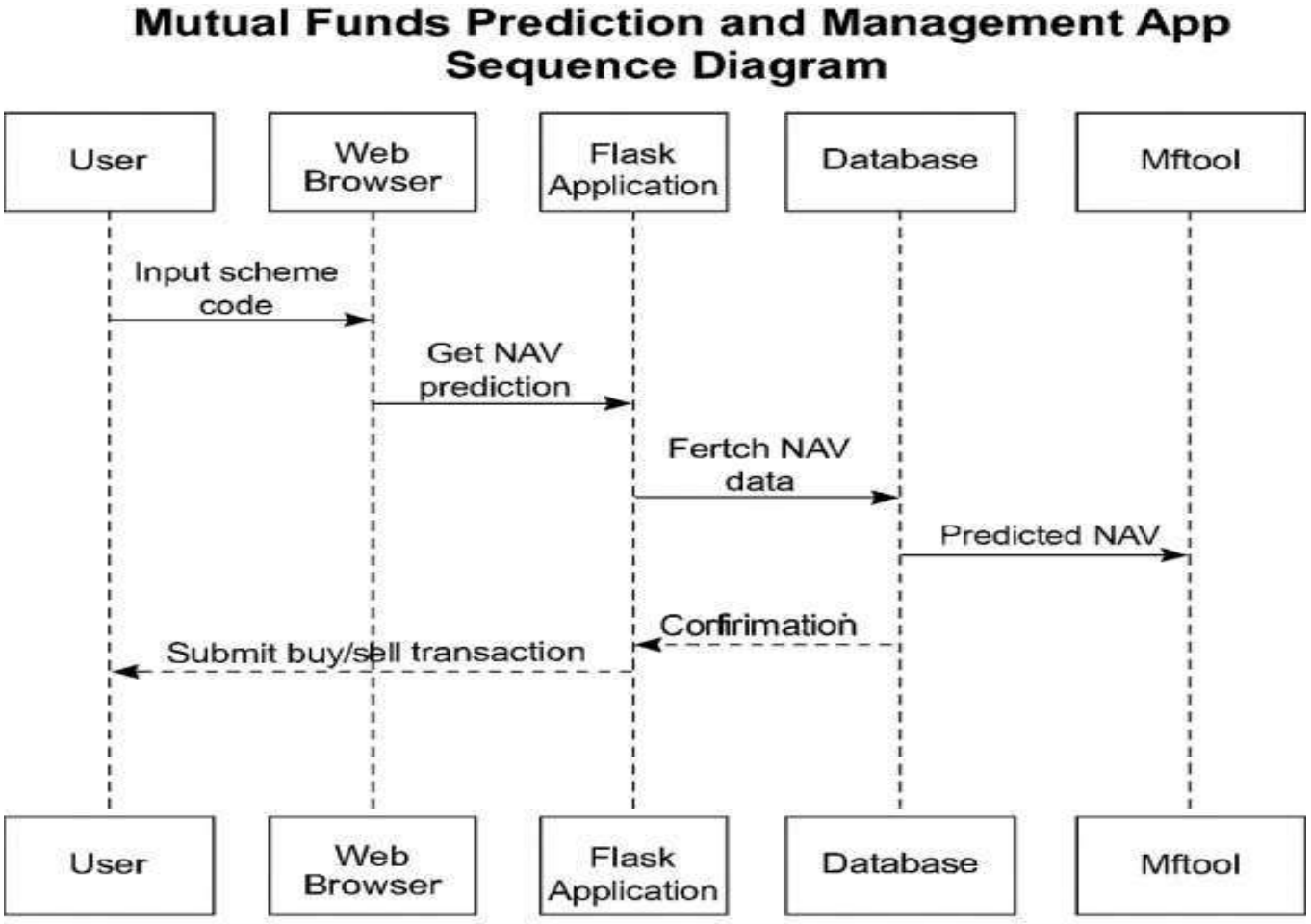



**Fig 3.5. Sequence Diagram**

Explanation of the Sequence Diagram:

**Actors and Objects**:

1. User: User
    - Initiates actions like selecting a fund, viewing NAV, and predicting returns.
2. Frontend (Web UI)
    - Interface through which the user interacts (HTML/CSS/JS).
    - Sends requests and displays results.
3. Backend (Flask API)
    - Handles business logic, database access, and model execution.
4. Database (MySQL)
    - Stores data in funds, nav_data, and transactions tables.
5. ARIMA Model
    - Machine learning model used to forecast future NAV values.

**Sequence Flow Description**

1. User Requests Fund Details
    - The user selects a mutual fund from the frontend UI.
    - The frontend sends a request to the backend.
2. Backend Fetches Data
    - Backend queries the funds table to fetch scheme details.
    - It may also retrieve NAV history from the nav_data table.
3. User Triggers Prediction
    - The user clicks on a "Predict NAV" or similar button.
    - A request is sent to the backend to start prediction.
4. Backend Loads Data
    - Backend pulls historical NAV data for the selected scheme from the nav_data table.
5. ARIMA Model Trained & Forecasted
    - The ARIMA model is fitted on the NAV history.
    - Forecasts the future NAV for a specified period.

6. Prediction Sent Back
   - Predicted NAV values are returned to the frontend.
   - Backend might also update the funds table with the last prediction.
7. User Views Result
   - Frontend receives the data and displays a graph/table of predicted values.
8. User Adds a Transaction (Optional Flow)
   - User may choose to invest or record a transaction.
   - A POST request is sent to the backend, which updates the transactions table accordingly.

### 3.7.3 Use Case Diagram

Draw use cases using ovals. Label with ovals with verbs that represent the system'sfunctions. Actors are the users of a system. When one system is the actor of another system, label the actorsystem with the actor stereotype.

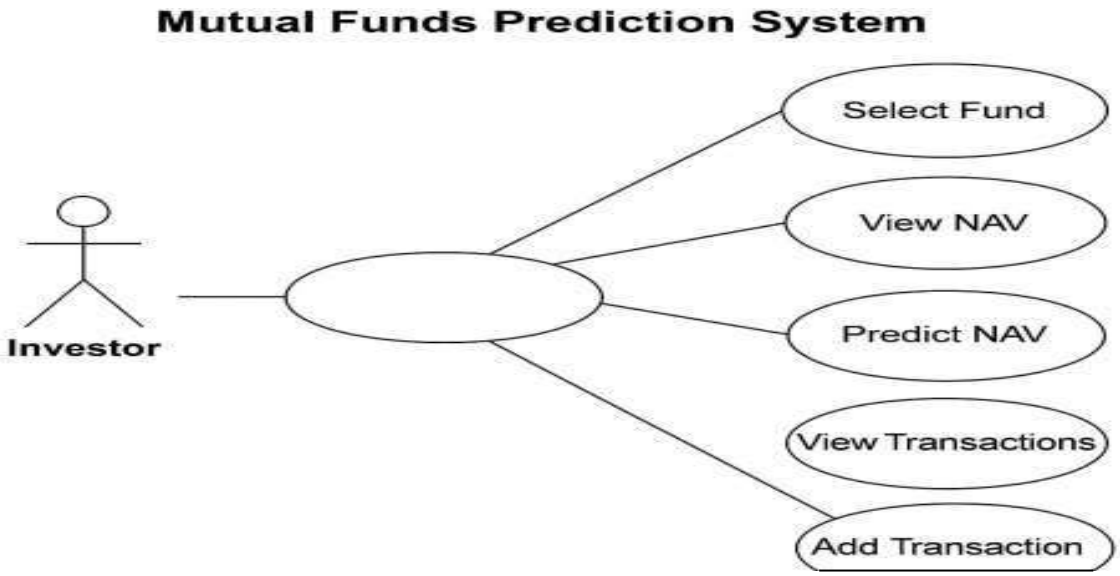



**Fig 3.6 Use Case Diagram**

**Actor:**

User – Can be an investor or analyst using the app.

**Use Cases :**

1. View Mutual Fund List
   User can browse/search funds from the database.
2. View NAV History
   User can see past NAV values as graphs or tables.
3. Predict Future NAV
   User selects a fund to forecast future NAV using the ARIMA model.
4. Record New Transaction
   User enters investment details; system calculates units and saves the transaction.
5. View Investment Returns
   User checks profits or losses (absolute return, IRR) based on past transactions.
6. Update Fund Data (Admin only)
   System or admin updates latest fund info and NAVs from external sources.

### 3.7.4 Activity Diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actionswith support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of componentsin a system. An activity diagram shows the overall flow of control

**Activity Flow:**

**1.Start:** The user opens the application and begins.

**2.User Login :** The user logs into their account.

**3.display Mutual Fund List :** The system shows a list of all available mutual fund schemes.

**4. User Selects a Fund :** The user chooses one fund from the list to work with.

**5. Now the user has two options:**

**Option 1: View and Record Transaction**

The user views the fund's historical NAV data.

Then, they enter the amount they want to invest and the date.

The system calculates how many units the user will get based on the NAV of that date.

The transaction is stored.

The system calculates and shows investment returns like absolute return and IRR.

**Option 2: Predict Future NAV**

The system fetches the fund's past NAV data.

It runs the ARIMA model to predict future NAV values.

The predictions are shown to the user.

The user can also view their investment returns based on these predictions.

**6.End :**

The user finishes and exits the application.

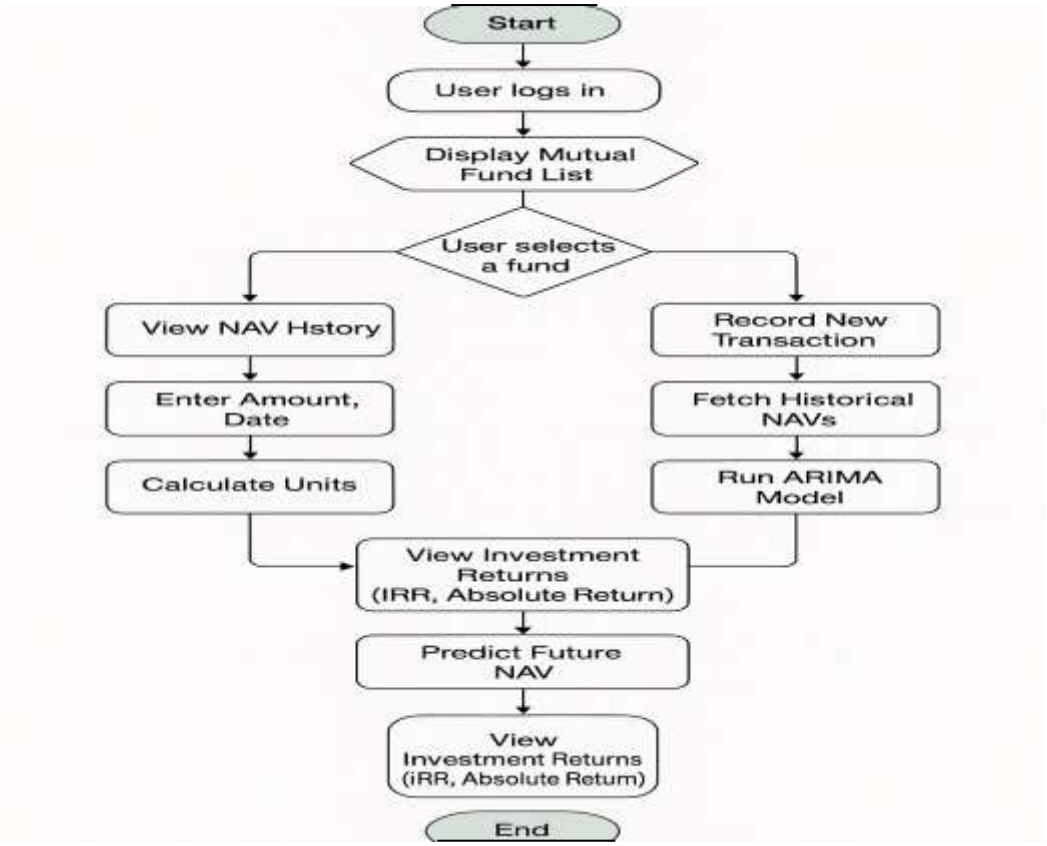



**Fig 3.7 Activity Diagram**

## 3.8 METHODOLOGY AND ALGORITHM

**DATASET:**

The dataset used in this project is primarily focused on mutual fund performance, with data regarding fund NAVs (Net Asset Values), transactions (buy/sell), and other financial metrics. The data includes historical performance data for various mutual funds, including their NAV trends, transaction histories, and details about the funds' underlying assets. This dataset is critical for predicting future NAV trends and assisting users in managing their mutual fund portfolios.

### 3.8.1 Machine Learning Design – ARIMA Algorithm

The **AMIR model** is a hybrid model combining **ARMA (AutoRegressive Moving Average)** and **Ridge Regression**, particularly designed for time series forecasting tasks. The model is adept at predicting future values of financial time series, such as mutual fund NAVs, by capturing complex temporal patterns while preventing overfitting through regularization.

**AutoRegressive Moving Average (ARMA)**

The ARMA model is one of the foundational techniques in time series forecasting. It is used when data shows evidence of correlation between observations over time (autocorrelation). The ARMA model consists of two main components:

**1.AutoRegressive (AR) Part**:

- The AR part uses the past observations to predict the current value.
- The model assumes that the current value of the time series is a linear combination of its previous values (lags).
- The AR part of the model is represented as:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t$$

where:

- Yt is the value of the time series at time t
- ϕi are the coefficients (weights) for each lag term,
- εt is the residual error at time t.

**2. Moving Average (MA) Part**:

- The MA part models the relationship between the observation and a series of lagged forecast errors.
- This component helps to smooth out short-term fluctuations in the time series data, capturing any noise.
- The MA model can be represented as:

$$Y_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

where:

- θi are the coefficients for the lagged error terms,
- εt is the white noise error at time t

By combining both AR and MA components, the ARMA model captures both the trend and the noise in the time series data.

**Ridge Regression**

While ARMA captures the temporal dependencies, **Ridge Regression** is applied to avoid overfitting by adding a **penalty term** (L2 regularization) to the loss function. Ridge Regression adjusts the weights in a way that shrinks them, allowing for better generalization and improved model stability.

The ridge regression loss function can be expressed as:

$$\text{Loss} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \lambda \sum_{j=1}^{k} w_j^2$$

where:

- Yi are the actual NAV values,
- Y^i are the predicted NAV values,
- wj are the coefficients (model parameters),
- λ is the **regularization parameter**, controlling the strength of the penalty.

In this system, Ridge Regression refines the predictions made by the ARMA model, improving overall prediction accuracy and reducing overfitting.

## 3.8.2 Mutual Fund Portfolio Process

**Real-Time Data Collection**

The system collects real-time **NAV data** using the **MFTool** library, which fetches mutual fund data from **AMFI (Association of Mutual Funds in India)**. The following steps are involved:

1. **Requesting Data**: Using a unique **scheme code**, the system sends a request to fetch the NAV data for various mutual funds.
2. **Storing Data**: The data is stored in a **MySQL database** with the following information:

- **Scheme Name** (e.g., HDFC Mutual Fund),
- **NAV** (Net Asset Value),
- **Date** (timestamp for when the NAV was recorded)

**Transaction Handling**

When users make **Buy** or **Sell** transactions, the system records:

- **Investment Amount**: The amount the user wishes to invest.
- **NAV at Transaction Time**: The NAV at the moment of the transaction.
- **Units Purchased/Sold**: Calculated by dividing the investment amount by the NAV.
- **Total Units**: Updated after each transaction (accumulated units for the user).
- **Transaction Type**: Whether the transaction is a **Buy** or **Sell**.

For **visualization**:

- **Buy Transactions** are displayed in **green** with a **'+' sign**.
- **Sell Transactions** are shown in **red** with a **'–' sign**.

### 3.8.3 Forecasting NAV Using AMIR

**Data Preprocessing**

Before applying the AMIR model, the historical NAV data is **preprocessed** to ensure accuracy and improve model performance:

1. **Stationarity Check**: The time series data is checked for stationarity using **Augmented Dickey-Fuller (ADF) test**. If the data is non-stationary, it is differenced to make it stationary.
2. **Normalization**: The data is normalized (scaled) to ensure that all features have a similar scale, which helps improve model performance.

**Model Training**

Once the data is ready, the **AMIR model** is trained on the historical NAV values. The model performs the following tasks:

**1. Fit the ARMA Model**: The ARMA model captures the temporal patterns in the NAV data. The order of the AR and MA components is determined using **grid search** based on model performance (usually through **AIC/BIC criteria**).

**2. Apply Ridge Regression**: The residuals and predictions from the ARMA model are passed through **Ridge Regression** to further refine the predictions and reduce overfitting.

**3. Forecast Future NAV**: Using the trained model, the system predicts the NAV for future time periods.

### 3.8.4 Data Types Involved

The system deals with several types of data:

**1.Numeric Data**: If a feature represents a characteristic measured in numbers, it is called a numeric feature.

- **NAV** values (continuous numerical data),
- **Investment Amount**,
- **Units Purchased/Sold**.

**2.Categorical Data**: A categorical feature is an attribute that can take on one of a limited, and usually fixed, number of possible values on the basis of some qualitative property.A categorical feature is also called a nominal feature.

- **Transaction Type**: Indicates whether the user is buying or selling.

**3.Time Series Data**:

- Historical **NAV values** over time (temporal data).

### 3.8.5 Model Evaluation Metrics

The AMIR model's performance is evaluated using the following metrics:

**1. Mean Absolute Error (MAE)**: Measures the average of the absolute errors between predicted and actual NAV values. It is less sensitive to large errors compared to RMS

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

**2. Root Mean Squared Error (RMSE)**: Measures the square root of the average squared differences between predicted and actual NAV values, providing a measure of how well the model captures the trend.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

**3.$R^2$ Score**: A statistical measure indicating how well the predictions match the actual data. An $R^2$ score closer to **1** indicates a better fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

The evaluation results are displayed on a **dashboard**, providing the user with insights into the accuracy and reliability of the model's predictions.

**Application of the ARIMA Model for NAV Prediction**: The ARIMA model is implemented in the following steps:

- **Step 1**: Data preprocessing includes handling missing data, scaling numerical features, and generating time-series features like lagged NAV values and moving averages.
- **Step 2**: The ARIMA model is first applied to forecast future NAV based on past NAV data.
- **Step 3**: Ridge Regression is then used to refine these predictions, accounting for other influencing variables such as economic indicators, sector performance, etc.
- **Step 4**: After model training, the system can predict future NAV values, which are critical for users when making buy/sell decisions.
- **Step 5**: The predicted NAV values are displayed on the front-end in a user-friendly format, with graphs showing past and predicted trends.

**Features of the Application**:

1. **Portfolio Management**:
    - Add/remove funds from the portfolio.
    - View current portfolio status with total investments, NAVs, and performance.
    - Buy/Sell funds with transaction details (e.g., units, amounts).
2. **NAV Prediction**:
    - Display predicted future NAV for selected funds using the ARIMA model.
    - Show graphical representations of the fund's past and predicted performance.
3. **Transaction Data Display**:
    - Display transaction history with relevant details: date, NAV, amount, units

bought/sold, and total units after each transaction.
    - Distinguish between Buy and Sell transactions by color-coding the units (green for Buy, red for Sell).
4. **Real-time Updates**:
    - NAV updates in real-time, ensuring users have the latest information for decision-making.
5. **Graphs and Analytics**:
    - Users can view performance graphs of their portfolios and individual funds, including predicted trends based on the ARIMA model.

**Technologies Used**:

- **Backend**: Python (Flask/Django), MySQL for database management
- **Frontend**: HTML, CSS, JavaScript (React or Angular for dynamic elements)
- **Modeling & Machine Learning**: ARIMA, Ridge Regression (scikit-learn), ARIMA Model (combination of ARIMA and Ridge)
- **Visualization**: Matplotlib/Plotly for interactive graphs

**Challenges**:

- **Data Quality**: Ensuring the dataset is complete and accurate, especially when dealing with real-time updates and historical data.
- **Model Performance**: Balancing model accuracy with computational efficiency, especially when handling large datasets.
- **User Experience**: Designing an intuitive interface that clearly displays predictions and transaction details without overwhelming the user.

# CHAPTER-4
# IMPLEMENTATION

# CHAPTER-4
# IMPLEMENTATION

## 4.1 INTRODUCTION TO PYTHON

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

### 4.1.1 Python concepts

If you're not interested in the haws and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it's a great one to start programming with.

- Open source general-purpose language.
- Object Oriented, Procedural, Functional
- Easy to interface with C/Object/Java/Fortran
- Easy to interface with
- Great interactive environment

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner- level programmers and supports the development of a wide range .

### 4.1.2 Libraries

A python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes python programming simpler and convenient for the programmer. Python libraries play a very vital role in fields of Machine learning, Data Science, Data Visualization etc. The libraries that are used in this project are pandas, matplotlib, seaborn.

- **Pandas:** Pandas provide us with many Series and Data Frames. It allows you to easily organize, explore, represent, and manipulate data. Smart alignment and indexing featured in Pandas offer you perfect organization and data labeling. Pandas have some special features that allow you to handle missing data or values with proper measures.
- **Matplotlib:** This library is responsible for the plotting of numerical data. It is utilized in data analysis for this reason. An open-source library plots superior quality figures, for example, pie outlines, scatterplots, boxplots, and diagrams, in addition to other things.
- **Seaborn:** Seaborn is an amazing visualization library for statistical graphical plotting in Python. It provides beautiful default styles and color palettes to make statistical plots are more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.
- **Scikit-Learn**: It is a famous Python library to work with complex data. Scikit learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy.

## 4.2 SAMPLE CODE

**requirements.txt**

```
flask
mysql-connector-python
pandas
mftool
matplotlib
scikit-
learn
numpy
```

**nav_prediction.py**

```
import mysql.connector

DB_CONFIG = {
    'host': 'localhost',
    'user': 'root',
    'password': '123456789',
    'database': 'mutual_funds'
}
def db_connection():
    try:
        connection = mysql.connector.connect(DB_CONFIG)
        if connection.is_connected():
            print("Database connected successfully!")
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

def get_dict_cursor(connection):
    """Returns a cursor that fetches results as dictionaries."""
    return connection.cursor(dictionary=True)
```

## App.py

```
from flask import Flask, render_template, request, jsonify
import pandas as pd
import matplotlib.pyplot as plt
from mftool import Mftool
from decimal import Decimal
from statsmodels.tsa.arima.model import ARIMA
import os
from config import db_connection, get_dict_cursor
import yfinance as yf
from datetime import datetime, timedelta

app = Flask(__name__)
mf = Mftool()
INDIAN_HOLIDAYS = {
    '01-01', '26-01', '15-08', '02-10', '25-12', '29-03', '14-04', '01-05'
}

if not os.path.exists("static/images"):
    os.makedirs("static/images")

def recommend_action(forecast):
    last_nav = forecast.iloc[-2]
    next_nav = forecast.iloc[-1]

    if next_nav > last_nav:
        return "BUY ✅ - Expected Growth", "The NAV is predicted to increase, indicating potential growth. It might be a good opportunity to invest."
    elif next_nav < last_nav:
        return "SELL ❌ - Expected Drop", "The NAV is predicted to decrease, suggesting a decline. You might consider selling or holding off investments."
    else:
        return "HOLD ⚖️ - No Significant Change", "There are no significant fluctuations in NAV. Holding the investment may be a stable choice."

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/get_nav', methods=['POST'])
def get_nav():
    scheme_code = request.form['scheme_code']
    nav_data = mf.get_scheme_historical_nav(scheme_code)

    try:
        fund_details = mf.calculate_returns(
            code=scheme_code, balanced_units=1, monthly_sip=1, investment_in_months=1
        )
    except ZeroDivisionError:
        return render_template('index.html', error="Calculation Error: Initial investment cannot be zero.")
```

```
if nav_data and 'data' in nav_data and fund_details:
    df = pd.DataFrame(nav_data['data'])
    df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
    df['nav'] = df['nav'].astype(float)
    df = df.sort_values('date')

    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute(
        """
        INSERT INTO funds (scheme_code, scheme_name, last_nav, last_updated,
                           absolute_return, irr_annualised_return, final_investment_value)
        VALUES (%s, %s, %s, NOW(), %s, %s, %s)
        ON DUPLICATE KEY UPDATE
            last_nav = VALUES(last_nav),
            last_updated = NOW(),
            scheme_name = VALUES(scheme_name),
            absolute_return = VALUES(absolute_return),
            irr_annualised_return = VALUES(irr_annualised_return),
            final_investment_value = VALUES(final_investment_value)
        """,
        (
            scheme_code,
            fund_details.get('scheme_name', 'Unknown'),
            df['nav'].iloc[-1],
            fund_details.get('absolute_return', 0),
            fund_details.get('IRR_annualised_return', 0),
            fund_details.get('final_investment_value', 0)
        )
    )
    conn.commit()
    cursor.close()
    conn.close()

    model = ARIMA(df['nav'], order=(5, 1, 0))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=30)

    future_dates = pd.date_range(start=df['date'].iloc[-1], periods=31, freq='D')[1:]

    action, description = recommend_action(forecast)

    plt.figure(figsize=(10, 6))
    plt.plot(df['date'], df['nav'], label='Historical NAV', color='blue')
    plt.plot(future_dates, forecast, label='Predicted NAV', linestyle='dashed', color='red')
    plt.xlabel('Date')
    plt.ylabel('NAV Price')
    plt.title(f'Predicted NAV for Scheme Code {scheme_code}')
    plt.xticks(rotation=45)
    plt.legend()
    plt.grid(True)
```

```
        img_path = f'static/images/nav_{scheme_code}.png'
        plt.savefig(img_path, bbox_inches='tight')
        plt.close()

        return render_template(
            'index.html',
            img_path=img_path,
            scheme_code=scheme_code,
            action=action,
            description=description,
            fund_details=fund_details
        )

    return render_template('index.html', error="Invalid Scheme Code or No Data Found")

@app.route('/add_transaction', methods=['POST'])
def add_transaction():
    try:
        data = request.json
        scheme_code = data.get('scheme_code')
        amount = Decimal(str(data.get('amount', 0)))
        input_date = data.get('date')

        if amount <= 0:
            return jsonify({"error": "Amount must be greater than zero."}), 400

        # Convert input date
        try:
            transaction_date = datetime.strptime(input_date, '%Y-%m-%d')
        except ValueError:
            return jsonify({"error": "Invalid date format. Use YYYY-MM-DD."}), 400

        print(f"Original Transaction Date: {transaction_date.strftime('%Y-%m-%d')}")

        nav_data = mf.get_scheme_historical_nav(scheme_code)

        if not nav_data or 'data' not in nav_data:
            return jsonify({"error": "Invalid Scheme Code or No Data Found"}), 400

        def get_nav_for_date(date):
            return next(
                (Decimal(str(item['nav'])) for item in nav_data['data']
                 if item['date'] == date.strftime('%d-%m-%Y')),
                None
            )

        # Step 1: Try fetching NAV for the original date
        latest_nav = get_nav_for_date(transaction_date)

        # Step 2: If unavailable, check for weekend or holiday
        if (transaction_date.weekday() in [5, 6] or  # Weekend
```

```
    transaction_date.strftime('%d-%m') in INDIAN_HOLIDAYS or
    not latest_nav):

    print(f"{transaction_date.strftime('%Y-%m-%d')} is a holiday/weekend or NAV unavailable.")

    # Step 3: Try the same date and month from the previous year
    previous_year_date = transaction_date.replace(year=transaction_date.year - 1)
    latest_nav = get_nav_for_date(previous_year_date)

    if not latest_nav:
        # Step 4: If still not available, roll back day-by-day
        while not latest_nav:
            print(f"No NAV for {transaction_date.strftime('%Y-%m-%d')}. Rolling back...")
            transaction_date -= timedelta(days=1)
            latest_nav = get_nav_for_date(transaction_date)

if not latest_nav:
    return jsonify({"error": "No valid NAV available after adjustments."}), 400

units = round(amount / latest_nav, 4)
print(f"Final Transaction Date: {transaction_date.strftime('%Y-%m-%d')}")
print(f"NAV: {latest_nav}, Units: {units}")

# Database connection
conn = db_connection()
cursor = conn.cursor()

# Fetch latest total_units for accumulation
cursor.execute(
    "SELECT total_units FROM transactions WHERE scheme_code = %s ORDER BY id DESC
    LIMIT 1",
    (scheme_code,)
)
result = cursor.fetchone()
previous_total_units = Decimal(str(result[0])) if result and result[0] else Decimal(0)
total_units = round(previous_total_units + units, 4)

# Insert transaction
cursor.execute(
    "INSERT INTO transactions (scheme_code, date, nav, amount, units, total_units) "
    "VALUES (%s, %s, %s, %s, %s, %s)",
    (scheme_code, transaction_date.strftime('%Y-%m-%d'), latest_nav, amount, units, total_units)
)

conn.commit()
cursor.close()
conn.close()

print(f"✅ Transaction Inserted: {scheme_code} | Total Units: {total_units}")

return jsonify({
    "message": "Transaction added successfully.",
```

```
            "scheme_code": scheme_code,
            "transaction_date": transaction_date.strftime('%Y-%m-%d'),
            "latest_nav": float(latest_nav),
            "amount": float(amount),
            "units": float(units),
            "total_units": float(total_units)
        })

    except ValueError:
        return jsonify({"error": "Invalid data format. Please check the inputs."}), 400

    except Exception as e:
        print(f"Error Occurred: {e}")
        return jsonify({"error": f"An unexpected error occurred: {str(e)}"}), 500

def get_latest_valid_nav(scheme_code, input_date):
    date = datetime.strptime(input_date, '%Y-%m-%d')
    while True:
        date_key = date.strftime('%d-%m')
        # Skip weekends and Indian holidays
        if date.weekday() not in (5, 6) and date_key not in INDIAN_HOLIDAYS:
            nav_data = mf.get_scheme_historical_nav(scheme_code)
            if nav_data and 'data' in nav_data:
                for entry in nav_data['data']:
                    entry_date = datetime.strptime(entry['date'], '%d-%m-%Y')
                    if entry_date <= date:
                        return Decimal(str(entry['nav']))
        date -= timedelta(days=1)  # Move to the previous day if invalid

@app.route('/sell_transaction', methods=['POST'])
def sell_transaction():
    try:
        data = request.json
        scheme_code = data.get('scheme_code')
        sell_amount = Decimal(str(data.get('amount', 0)))
        input_date = data.get('date')

        if sell_amount <= 0:
            return jsonify({"error": "Sell amount must be greater than zero."}), 400

        conn = db_connection()
        cursor = conn.cursor()

        # Fetch latest total units for the scheme
        cursor.execute(
            "SELECT total_units FROM transactions WHERE scheme_code = %s ORDER BY id
             DESC LIMIT 1",
            (scheme_code,)
        )
        result = cursor.fetchone()

        if not result or result[0] is None:
```

```
            return jsonify({"error": "No units available for this scheme code."}), 400

        previous_total_units = Decimal(str(result[0]))

        # Get the latest valid NAV based on the input date
        latest_nav = get_latest_valid_nav(scheme_code, input_date)

        # Calculate units to sell
        units_to_sell = round(sell_amount / latest_nav, 4)
        if units_to_sell > previous_total_units:
            return jsonify({"error": "Insufficient units to sell."}), 400

        total_units = round(previous_total_units - units_to_sell, 4)

        # Insert sale transaction
        cursor.execute(
            "INSERT INTO transactions (scheme_code, date, nav, amount, units, total_units) "
            "VALUES (%s, %s, %s, %s, %s, %s)",
            (scheme_code, input_date, latest_nav, -sell_amount, -units_to_sell, total_units)
        )

        conn.commit()
        cursor.close()
        conn.close()

        return jsonify({
            "message": "Transaction (Sale) recorded successfully.",
            "scheme_code": scheme_code,
            "date": input_date,
            "latest_nav": float(latest_nav),
            "sell_amount": float(sell_amount),
            "units_sold": float(units_to_sell),
            "remaining_total_units": float(total_units)
        })

    except Exception as e:
        return jsonify({"error": f"An unexpected error occurred: {str(e)}"}), 500

@app.route('/get_average_nav/<scheme_code>', methods=['GET'])
def get_average_nav(scheme_code):
    try:
        conn = db_connection()
        cursor = conn.cursor()
        cursor.execute(
            "SELECT SUM(amount), MAX(total_units) FROM transactions WHERE scheme_code = %s",
            (scheme_code,)
        )
        result = cursor.fetchone()
        cursor.close()
        conn.close()

        total_invested = Decimal(str(result[0])) if result[0] else Decimal(0)
```

```
        total_units = Decimal(str(result[1])) if result[1] else Decimal(0)

        average_nav = round(total_invested / total_units, 4) if total_units > 0 else Decimal(0)

        return jsonify({"scheme_code": scheme_code, "average_nav": float(average_nav)})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/get_transactions/<scheme_code>')
def get_transactions(scheme_code):
    conn = db_connection()
    cursor = conn.cursor(dictionary=True)

    cursor.execute(
        "SELECT date, nav, amount, units, total_units FROM transactions WHERE
          scheme_code = %s ORDER BY date DESC",
        (scheme_code,)
    )
    transactions = cursor.fetchall()

    cursor.close()
    conn.close()

    return jsonify(transactions)



@app.route('/get_all_funds')
def get_all_funds():
    conn = db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM funds")
    funds = cursor.fetchall()
    cursor.close()
    conn.close()
    return jsonify(funds)

@app.route('/remove_fund', methods=['POST'])
def remove_fund():
    scheme_code = request.json.get('scheme_code')

    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM funds WHERE scheme_code = %s", (scheme_code,))
    conn.commit()
    cursor.close()
    conn.close()

    return jsonify({"message": "Fund removed successfully"})
```

```
@app.route('/get_nifty_value')
def get_nifty_value():
    nifty = yf.Ticker("^NSEBANK")
    nifty_data = nifty.history(period="1d")

    if nifty_data.empty:
        return jsonify({"error": "NIFTY 50 data unavailable"}), 500

    latest_price = nifty_data["Close"].iloc[-1]
    return jsonify({"nifty_value": latest_price})

if __name__ == '__main__':
    app.run(debug=True)
```

# CHAPTER-5

# TESTING

# CHAPTER-5

# TESTING

## 5.1 INTRODUCTION

The goal of testing is to acquire errors. Testing is that the technique of trying to get each possible error or weakness in an extremely work product. It provides the way to observe the practicality of parts, sub-assemblies, assemblies and or a finished product it is the technique of effort code with the concentrating of guaranteeing that the software meets its requirements and user hopes and does not fail in an undesirable manner. There are numerous sorts of check. Every check sort reports a designated testing demand.

**Testing objectives:**

The key objective of testing is to determine a mass of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with resolved of discover an error.

- A successful test is one that determines an as however undiscovered error.
- A good test case is one that has possibility of discover an error, if it exists.
- The test is insufficient to detect possibly present errors.
- The software more or less approves to the quality and unswerving standards

**Testing:**

During During the **testing phase**, the implementation is evaluated against the initial requirements to ensure the system performs as expected and fulfills user needs.

The testing includes:

- **Unit Testing**: Performed on individual modules such as NAV calculation, prediction logic, database interactions, and UI components to verify each part functions correctly.
- **System/Acceptance Testing**: Conducted on the entire application to ensure that modules work together seamlessly and the system meets functional and non-functional requirements.

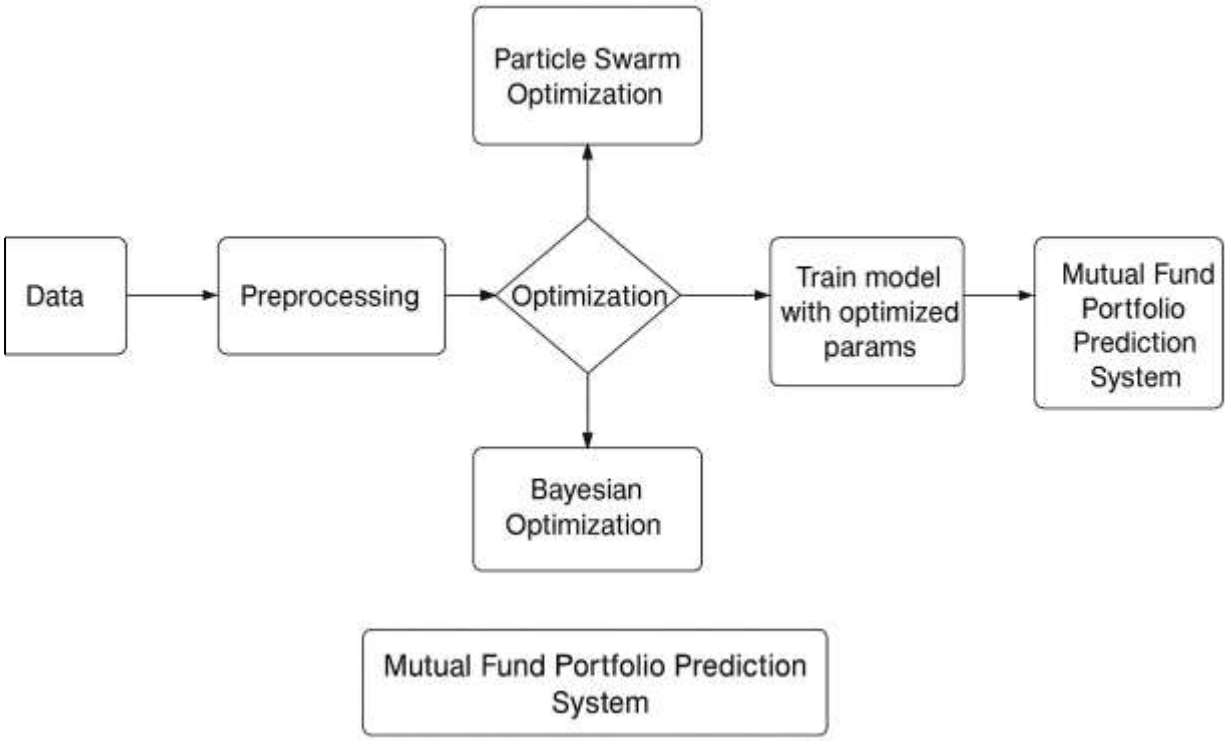



**Fig 5.1 mutual funds portfolio design**

## System Components and Flow:

1. **Dataset**
    - **Description**: The system uses historical Net Asset Value (NAV) data of mutual funds. Data may be sourced from trusted APIs or tools like **MFTool**, which provide daily NAVs and fund metadata.
2. **Preprocessing**
    - **Description**: The collected NAV data undergoes preprocessing to:
        - Handle missing or inconsistent values
        - Normalize NAV scales
        - Structure the data for time series modeling and machine learning processes
3. **Feature Extraction**
    - **Description**: Important features relevant to NAV trends are extracted. These may include:
        - Previous NAV values (lag features)
        - Fund-specific indicators
        - Moving averages or volatility measures

4. **Prediction Model**
    - **Description**: The system utilizes the **ARIMA model (ARIMA + Ridge Regression)** to predict future NAV values.
        - **ARIMA** captures time-series patterns.
        - **Ridge Regression** improves prediction accuracy by handling multicollinearity in features.
    - These models are trained and validated using historical data to provide reliable future forecasts.
5. **Portfolio Management**
    - **Description**: Users can:
        - Add or remove mutual funds from their portfolio
        - Buy/Sell units (tracked with date, NAV, amount, units, and total units)
        - View a clear log of transactions, with color-coded entries:
            - **Buy** transactions show units with a + sign
            - **Sell** transactions show units in **red** with a - sign
6. **NAV Display and Updates**
    - **Description**: The current NAVs are updated regularly using real-time data. The user interface shows:
        - Fund-wise NAVs
        - Total investment value
        - Historical NAV graphs
7. **Prediction Visualization**
    - **Description**: The system provides graphical views of:
        - Historical NAV trends
        - Predicted NAVs for the upcoming days
    - These help users in decision-making regarding their investments.

## 5.2 Types Of Tests

### 5.2.1 Unit testing:

A unit is the smallest piece of source code that can be tested. It is also known as a module which consists of numerous lines of code that are processed by a single programmer. The key purpose of performing unit testing is to expose that a particular unit doesn't satisfy the specified

functional requirements and also to show that the structural implementation is not like to the projected structure designed.

| TesAt Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| TC5 | ARIMA model NAV prediction logic | 1. Pass historical NAV data to model → 2. Run model → 3. Check forecast | ARIMA predicts NAV based on time-series patterns | Pass/Fail |
| TC6 | Buy units calculation | 1. Enter amount and NAV → 2. Execute buy function | Correct units are calculated and returned | Pass/Fail |
| TC7 | Sell units deduction logic | 1. Enter sell amount → 2. System subtracts correct units | Units are reduced appropriately | Pass/Fail |
| TC8 | Store transactions in DB | 1. Complete buy/sell → 2. Check backend DB | Transaction data is saved accurately in MySQL | Pass/Fail |
| TC11 | Invalid input handling | 1. Enter invalid data in form → 2. Try submission | System throws validation error and blocks submission | Pass/Fail |

### 5.2.2 Integration testing:

Tests are intended to test incorporated programming segments to figure out whether they really keep running as one system. Testing is occasion driven and is more worried with the fundamental result of screens or fields. Reconciliation tests exhibit that in spite of the fact that the parts were separately fulfillment, as appeared by effectively unit testing, the blend of segments is right and comprised. Integration testing is specifically aimed at revealing the problems that rise from the mixture of components.

| Test Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| ITC1 | Add fund and check real-time NAV update | 1. Add a new fund → 2. Wait for NAV refresh | Fund is added and NAV updates in real-time | Pass/Fail |
| ITC2 | Buy fund and check portfolio update | 1. Select fund → 2. Buy with amount → 3. View portfolio | Portfolio reflects increased units, “+” shown | Pass/Fail |
| ITC3 | Sell fund and validate unit reduction | 1. Select fund → 2. Sell with details → 3. View portfolio | Units are reduced, “-” shown in red | Pass/Fail |
| ITC4 | Trigger NAV prediction and show in UI | 1. Select fund → 2. Trigger prediction → 3. Observe UI | Predicted NAV appears in chart/text | Pass/Fail |
| ITC5 | Add, Buy, and Predict flow | 1. Add fund → 2. Buy units → 3. Predict NAV | All components integrate and work seamlessly | Pass/Fail |

| | | | | |
|---|---|---|---|---|
| ITC6 | Remove fund and clear transaction data | 1. Select and remove fund → 2. View transaction history | Fund is removed, related data cleared | Pass/Fail |
| **Test Case ID** | **Test Case Description** | **Test Steps** | **Expected Outcome** | **Status** |
| ITC1 | Add fund and check real-time NAV update | 1. Add a new fund → 2. Wait for NAV refresh | Fund is added and NAV updates in real-time | Pass/Fail |
| ITC2 | Buy fund and check portfolio update | 1. Select fund → 2. Buy with amount → 3. View portfolio | Portfolio reflects increased units, "+" shown | Pass/Fail |
| ITC3 | Sell fund and validate unit reduction | 1. Select fund → 2. Sell with details → 3. View portfolio | Units are reduced, "-" shown in red | Pass/Fail |

### 5.2.3 Functional testing:

Functional tests give efficient challenges that capacities tried are accessible as determined by the business and specialized necessities, framework documentation, and client manuals. Association and arrangement of practical tests is centered around prerequisites, key capacities, or unique experiments. Likewise, efficient scope relating to recognize Business procedure streams; information fields, predefined procedures, and progressive procedures must be considered for testing. Before utilitarian testing is finished, extra tests are distinguished and the powerful estimation of current tests is resolved.

| Test Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| FTC1 | Add new mutual fund | 1. Go to portfolio → 2. Click "Add Fund" → 3. Fill and submit | Fund appears in portfolio list | Pass/Fail |
| FTC2 | Remove mutual fund | 1. Select fund → 2. Click "Remove" → 3. Confirm | Fund disappears from list | Pass/Fail |
| FTC3 | View fund details | 1. Open portfolio section | Holdings, NAV, and history shown | Pass/Fail |
| FTC4 | Execute Buy transaction | 1. Select fund → 2. Choose “Buy” → 3. Enter & submit | Units added; transaction saved with “+” | Pass/Fail |
| FTC5 | Execute Sell transaction | 1. Select fund → 2. Choose “Sell” → 3. Enter & submit | Units deducted; “-” in red shown | Pass/Fail |

### 5.2.4 System Testing:

System testing guarantees that the entire coordinated programming framework meets prerequisites. It tests a design to guarantee known and unsurprising results. A sample of framework testing is the arrangement situated framework combination test. Framework testing depends on procedure portrayals and streams, stressing pre-driven procedure connections and mix focuses.

| Test Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| TC1 | Add a new mutual fund to the portfolio | 1. Navigate to portfolio → 2. Click "Add Fund" → 3. Fill details → 4. Submit | Fund is added successfully, and portfolio is updated | Pass/Fail |
| TC2 | Remove a mutual fund from the portfolio | 1. Go to portfolio → 2. Select fund → 3. Click "Remove" → 4. Confirm removal | Fund is removed, and portfolio updates accordingly | Pass/Fail |
| TC3 | View portfolio details | 1. Open portfolio section → 2. Check if all fund data is visible | Holdings, units, NAV, and history are shown correctly | Pass/Fail |
| TC4 | Real-time NAV update | 1. Add a fund → 2. Wait or trigger NAV update → 3. Observe NAV change | NAV is updated correctly with current data | Pass/Fail |
| TC5 | Predict future NAV using ARIMA model | 1. Select fund → 2. Trigger prediction → 3. View forecasted NAV | Predicted NAV is displayed based on historical trends | Pass/Fail |
| TC6 | Buy transaction functionality | 1. Go to transaction section → 2. | Transaction recorded; units | Pass/Fail |

| | | | | |
|---|---|---|---|---|
| | | Select “Buy” → 3. Enter data → 4. Submit | shown with “+” symbol | |
| TC7 | Sell transaction functionality | 1. Navigate to transaction → 2. Choose “Sell” → 3. Enter details → 4. Submit | Transaction recorded; units shown in red with “-” symbol | Pass/Fail |
| TC9 | Portfolio performance graph | 1. Open portfolio → 2. View graph → 3. Analyze performance over time | Graph displays accurate performance history | Pass/Fail |
| TC10 | UI responsiveness across devices | 1. Use platform on mobile/tablet/PC → 2. Perform common actions | Interface adapts and works smoothly on all devices | Pass/Fail |
| TC13 | Real-time notification on updates | 1. Update NAV of a fund → 2. Wait for system message | User receives timely notification for NAV change | Pass/Fail |

### 5.2.5 White Box Testing:

It is a testing in which the product analyzer has information of the internal workings, structure and dialect of the product, or if nothing else its motivation. It is reason. It is utilized to test ranges that can't be gotten a handle on from a discovery level.

| Test Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| TC5 | Internal working of ARIMA prediction model | 1. Trace data flow → 2. Step through prediction logic | ARIMA correctly forecasts NAV with visible intermediate steps | Pass/Fail |
| TC8 | MySQL transaction saving (code-level) | 1. Analyze SQL insertion logic during transaction | Data gets inserted into correct table/columns | Pass/Fail |
| TC12 | Handling large transaction dataset | 1. Add 100+ transactions → 2. Monitor performance → 3. Check system response | System handles large data smoothly with no delay | Pass/Fail |

### 5.2.6 Black Box Testing:

It is the testing the product with no information of within workings, structure or dialect of the part being tried. Discovery tests, as most different sorts of tests, must be composed from a complete source report, for example, prerequisite or necessities archive, for example, determination or necessities record. It is a trying in which the product under test is dealt with, as a discovery. you can't "see" into it. The test gives inputs and reacts to yields without considering how the product functions.

| Test Case ID | Test Case Description | Test Steps | Expected Outcome | Status |
|---|---|---|---|---|
| TC1 | Add mutual fund (UI-level validation) | 1. Use interface to add fund with valid data | Fund is added successfully | Pass/Fail |
| TC2 | Remove fund based on visible options | 1. Use UI options to remove selected fund | Fund disappears from view and database | Pass/Fail |

| | | | | |
|---|---|---|---|---|
| TC3 | Check portfolio information on screen | 1. Load portfolio screen | System displays all fund and transaction details | Pass/Fail |
| TC4 | Observe NAV update from user view | 1. Wait or manually trigger update | NAV changes reflect accurately | Pass/Fail |
| TC6 | Buy via UI (no internal logic visible) | 1. Fill buy form and submit | Units and transaction show correctly | Pass/Fail |
| TC7 | Sell via UI (output only observed) | 1. Enter sell form details and submit | Red-colored negative units added to history | Pass/Fail |
| TC11 | Invalid values on form | 1. Submit letters in amount field | System blocks form and shows error | Pass/Fail |
| TC13 | Notifications visibility | 1. Modify portfolio/NAV and observe | Notification pops up on screen | Pass/Fail |

## Test objectives

- To ensure that the Dataset recognition is done properly.
- To enhance each label category should have enough data without any mishandled values.

## Features To Be Tested

- The Dataset preprocessing and feature selection should be done in order.
- The feature sampling and ensemble preparation will be handled with only processed dataset.

# CHAPTER-6
# RESULT AND ANALYSIS

# CHAPTER-6
# RESULT AND ANALYSIS

## 6.1. NAV PREDICTION GRAPH – HISTORICAL AND FORECASTED

This is the interface where users enter or select a mutual fund scheme code. Upon confirmation, the system fetches NAV data and applies forecasting (e.g., ARIMA model) to predict future trends.

## 6.2. ENTERING SCHEME CODE

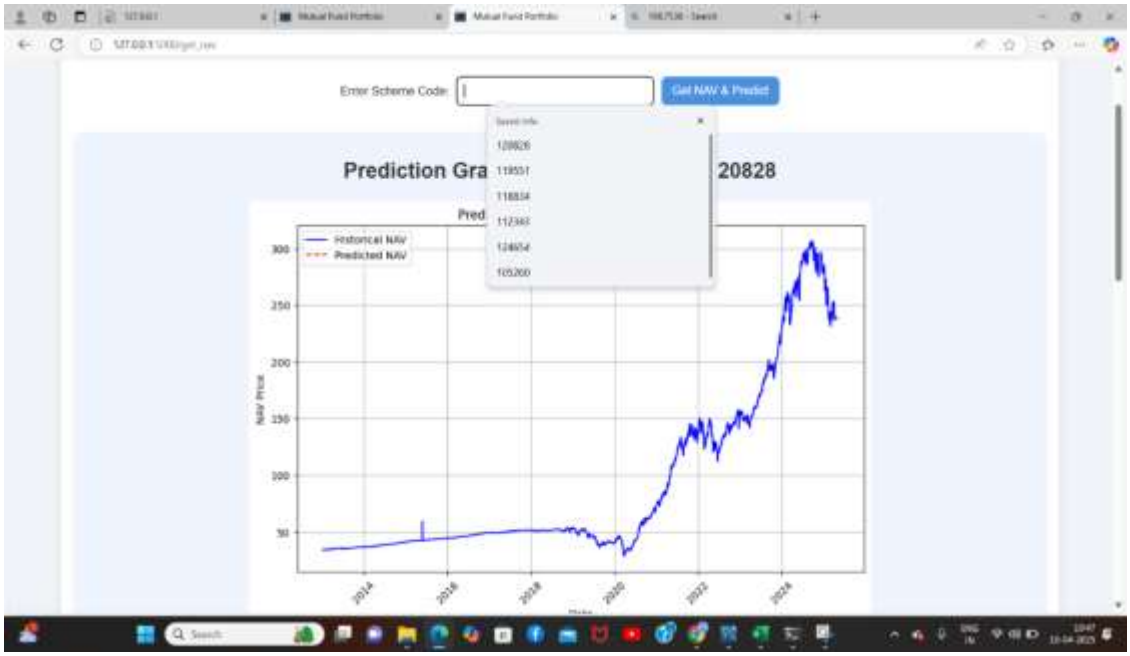



**Fig 6.1 Entering Scheme Code**

The generated graph displays historical NAV values as a solid blue line, while predicted values appear in a dashed red line. This visualization enables users to estimate potential fund performance and plan their investments accordingly.

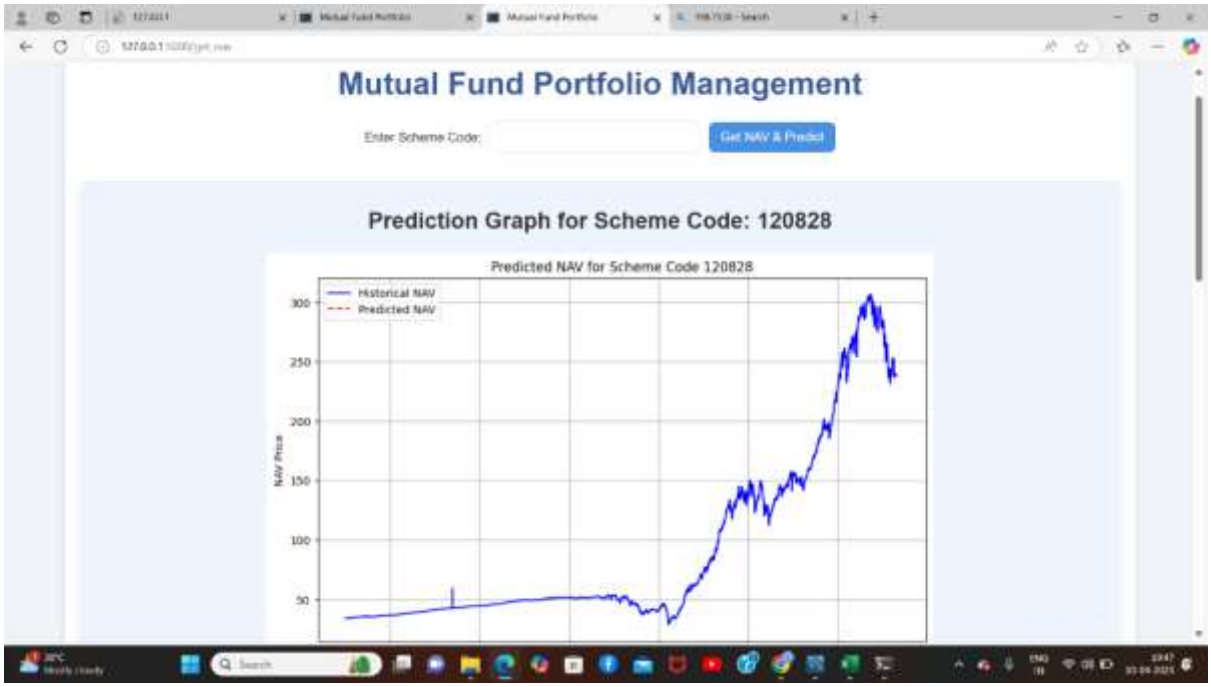



**Fig 6.2 entering code**

## 6.3 INVESTMENT RECOMMENDATION

The **Investment Recommendation** feature suggests mutual funds to the user based on their investment goals, risk tolerance, and past investment patterns. The system analyzes factors like the user's investment history, fund performance, and market trends to recommend funds that align with their preferences. This helps users make informed decisions and choose funds that best suit their financial goals and risk profile.

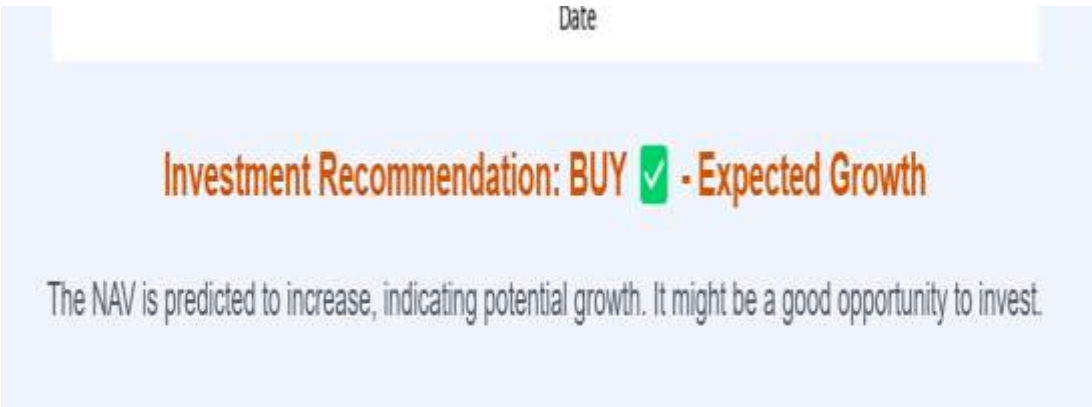



**Fig 6.3. Output of recommendation**

## 6.4 ALL FUND PORTFOLIO


**Fig.6.4 Transaction History**

This is the central view of the mutual fund portfolio web application. It presents a table listing all mutual fund investments made by the user. Each row shows:

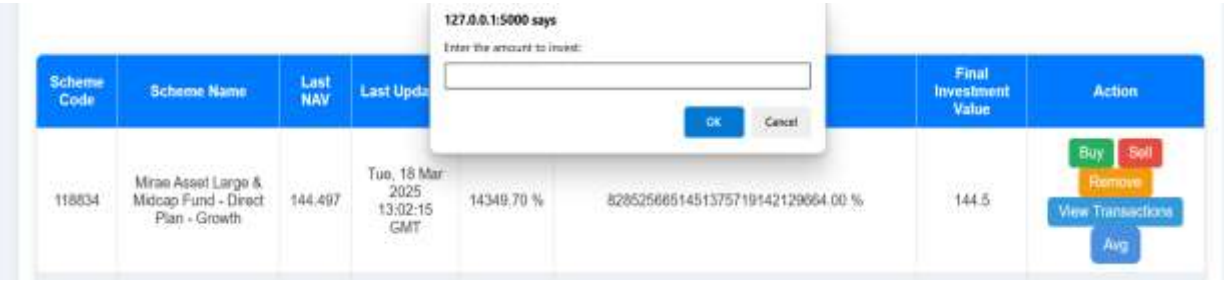
- **Scheme Code**: The fund's unique identifier.
- **Scheme Name**: Full name of the mutual fund.
- **Last NAV and Last Updated:** The latest Net Asset Value and the date/time of the update.
- **Absolute and Annualized Returns:** Reflect performance metrics.
- **Final Investment Value:** The current worth of the investment.
- **Actions**: Buttons to Buy, Sell, Remove, View Transactions, or view Average NAV for the fund.

## 6.5 BUYING A FUND

The "Add Fund" feature in the Mutual Fund Portfolio Website allows users to add new mutual fund investments to their portfolio. This process begins on the frontend with a simple HTML form where users can enter essential details such as the fund name, investment amount, NAV

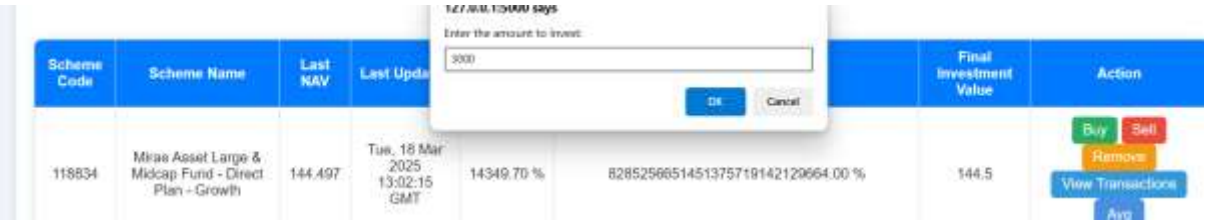
(Net Asset Value), and the date of investment. Once the user submits this form, the backend—developed in Python using Flask—captures the submitted data and calculates the number of units by dividing the investment amount by the NAV. The calculated units, along with the other entered details, are then inserted into the transactions table .The transactions are displayed with complete transparency, showing the date, NAV, investment amount, and units bought, with buys highlighted using a "+" sign and a standard color. This feature helps users manage and track their mutual fund investments in an organized and efficient way.

### 6.5.1 Enter Amount To Invest of A Fund

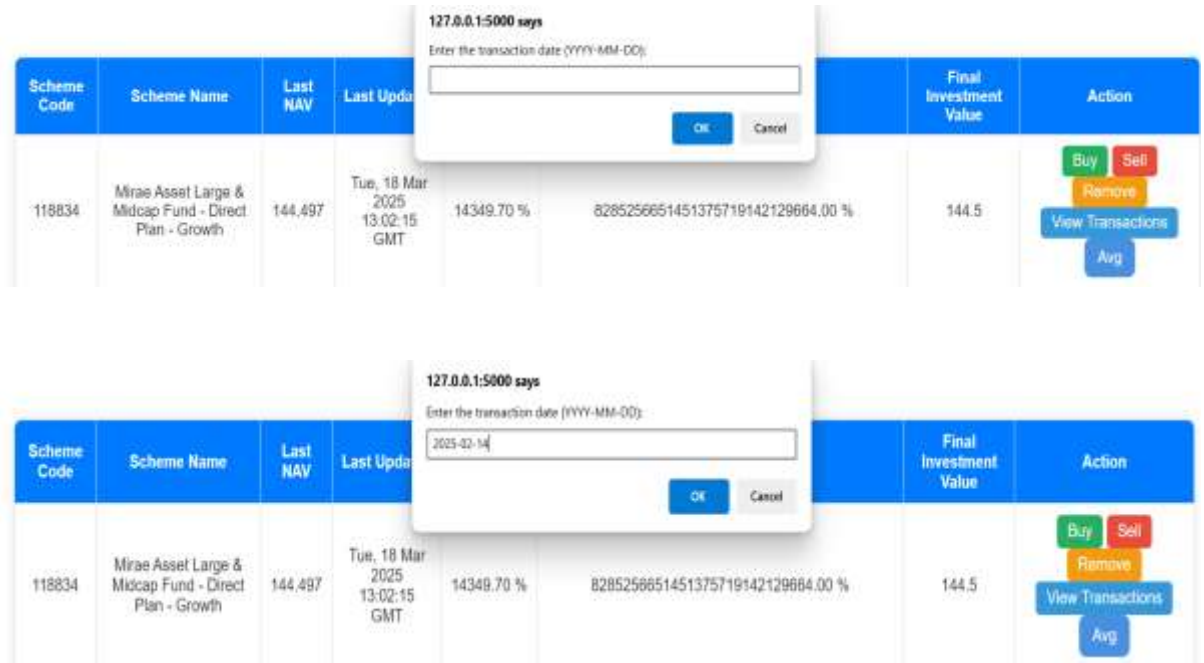



- **Fig.6.5 Entering Amount**
-

In this step, the user types the amount they want to invest in a mutual fund. This amount will be used to calculate how many units the user will get based on the current NAV (Net Asset Value). The system will divide the amount by the NAV to find out the number of units. This helps keep the investment record clear and accurate in the user's portfolio.



### 6.5.2 Buying Mutual Fund – Select Transaction Date

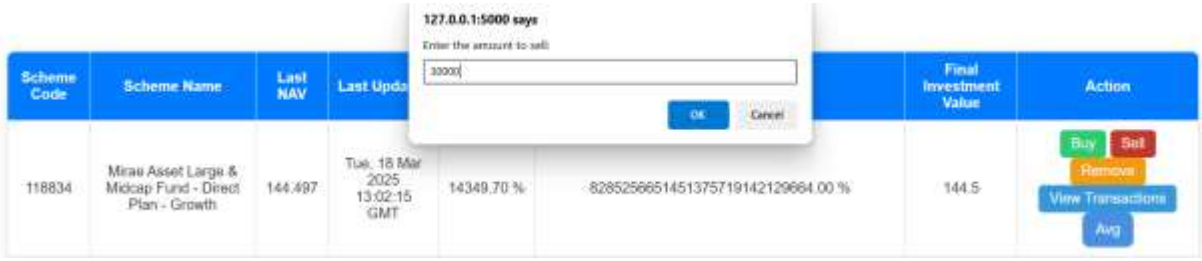



While buying a mutual fund, the user can select any past date to add old or missed investment history. If the selected date is a holiday, like a public holiday or a weekend, the system will automatically take the NAV of the **previous working day**. For example, if the user selects a **Sunday or Saturday**, the system picks the NAV from **Friday**. This ensures the NAV used is valid and accurate based on the actual trading days.

## 6.6 SELLING MUTUAL FUND

In this step, the user enters the amount they want to sell from their mutual fund investment. The system will use the current NAV (Net Asset Value) on the selected date to calculate how many units should be sold. The formula used is: **units = amount ÷ NAV**. These units will then be subtracted from the total units in the portfolio. If the entered amount is more than the available units' worth, the system will show an error to prevent over-selling. This step helps users easily sell part of their investment based on how much money they want to withdraw

### 6.6.1 Enter Amount To Sell of A Fund




In this step, the user enters the amount they want to take out from the fund. The system uses the NAV to calculate how many units to sell. These units are then removed from the total in the portfolio. If the amount is more than what the user has, an error will be shown.

### 6.6.2 Enter Transaction Date

When selling a mutual fund, the user can choose any past date to record the transaction. If the selected date is a holiday or weekend (like Saturday or Sunday), the system will automatically take the NAV from the previous working day (Friday). This ensures that the NAV used for the sale is valid and reflects the actual trading day.

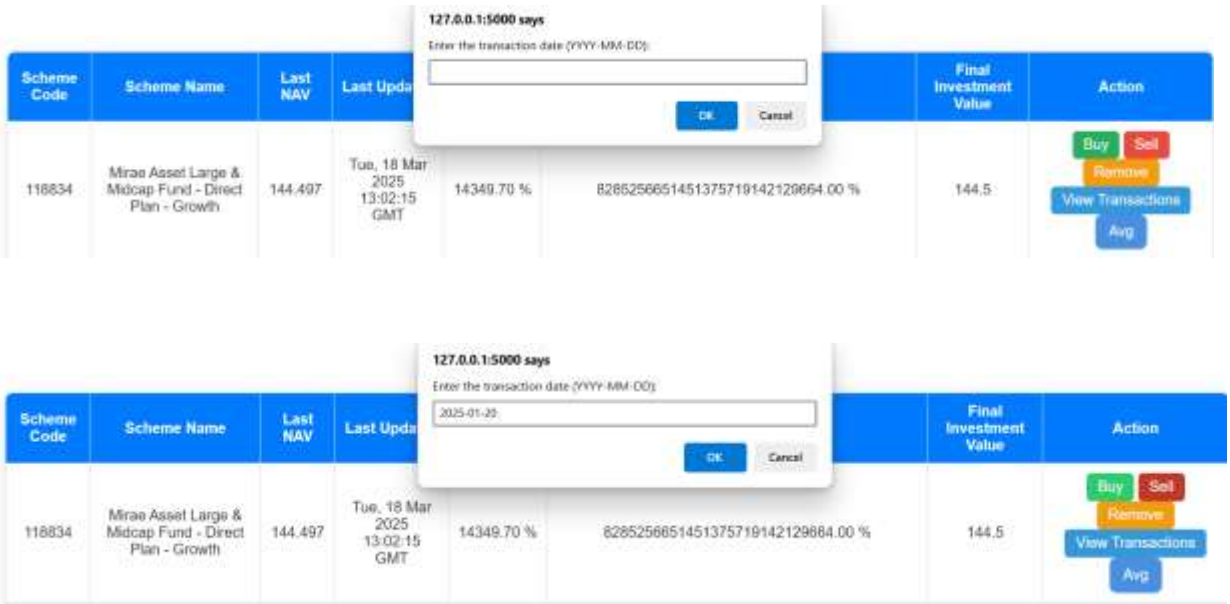
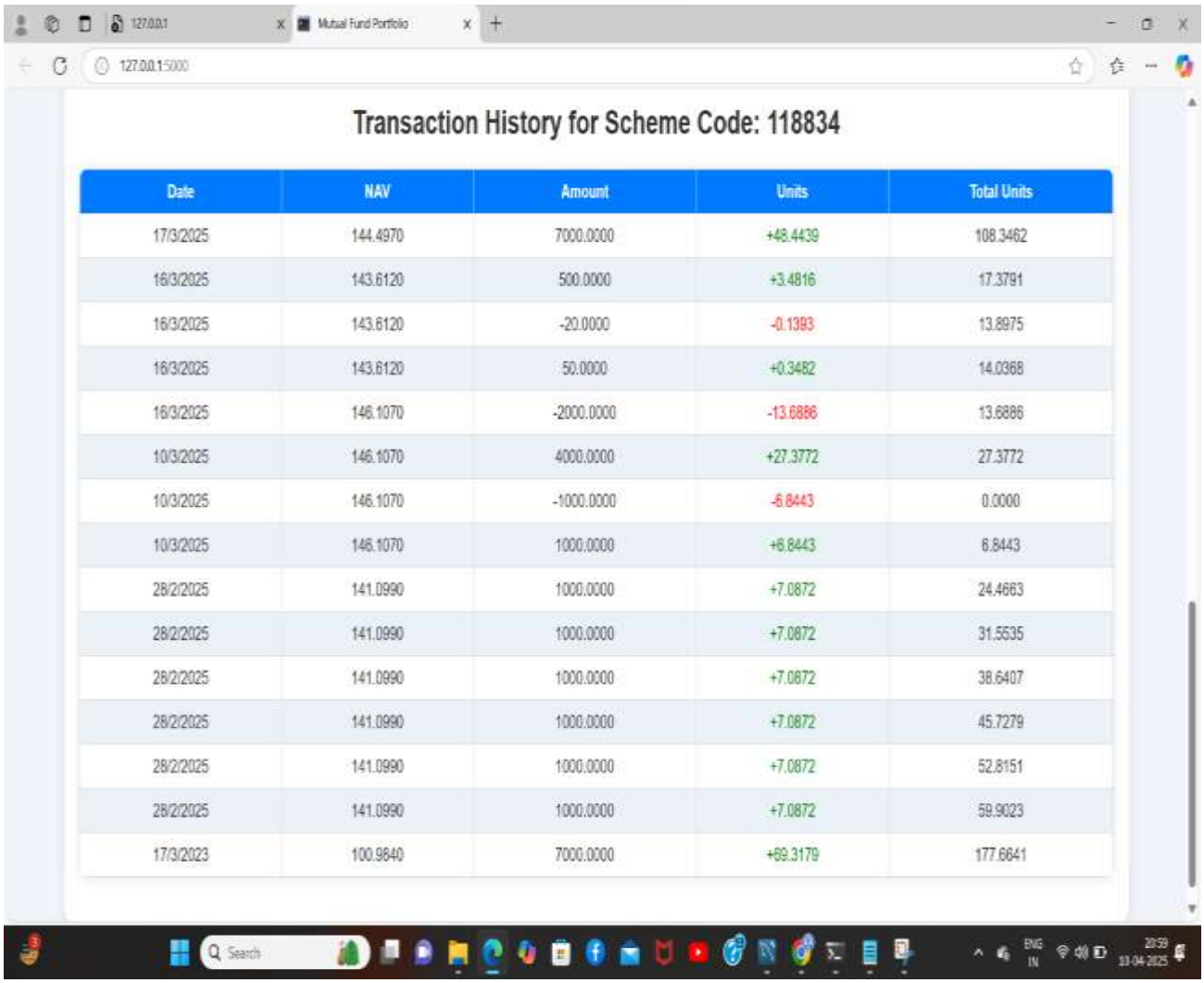



**Fig.6.6 Entering Data**

## 6.7 VIEWING TRANSACTION HISTORY

When the user clicks on a specific mutual fund in their portfolio, they can view the complete transaction history for that fund. This includes all past buys, sells, and any related details such

as the transaction date, NAV at the time, amount invested, and the number of units bought or sold. This feature allows users to track their investment journey and see how each fund has performed over time.




## 6.8 REMOVING FUND

To remove a mutual fund from the portfolio, the user can select the fund they wish to remove. Once confirmed, the system will delete the fund from the portfolio, including its associated transaction history. This ensures that the portfolio stays up-to-date, reflecting only the funds the user currently holds. If the user has sold all units of a fund, it can be safely removed without affecting any remaining holdings.


**Fig.6.7 Remove Fund**

## 6.9 SCHEME WISE AVERAGE INVESTMENT

The **Scheme-Wise Average Investment** shows the average amount invested in each mutual fund scheme across all transactions. It calculates the total amount invested in a particular fund and divides it by the number of transactions made for that fund. This helps the user understand their average investment in each fund, giving a clear picture of how much money has been allocated to each scheme over time.

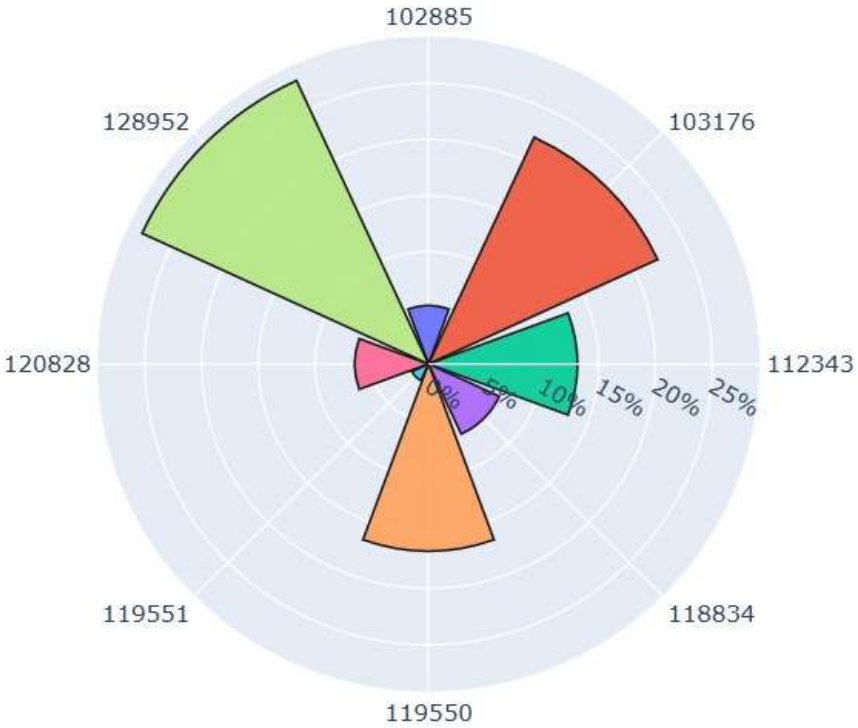



**Fig.6.8 Average Invested Amount**

**Comparison:**

- **Optimization Techniques**: In the context of your mutual fund portfolio and NAV prediction, you applied optimization techniques like **PSO** and **Bayesian Optimization** to tune machine learning models used for predicting the future NAV.
- **Prediction Accuracy**: Both optimization techniques have produced accurate predictions for future NAVs, with the majority of predictions closely matching the actual NAV data. The models consistently predicted NAV values that are very close to the observed values.

**Observation:**

Both **PSO** and **Bayesian Optimization** effectively fine-tune the machine learning models used for predicting NAV in the mutual fund portfolio. Despite small variations in performance, both optimization techniques show high accuracy and robustness. Neither method stands out significantly over the other, making both equally suitable for optimizing your model. These results ensure that your mutual fund prediction system can reliably predict future NAV values, with minimal error and robust performance across different optimization approaches.

# CHAPTER -7
# CONCLUSION AND FUTURE SCOPE

# CHAPTER -7
# CONCLUSION AND FUTURE SCOPE

## CONCLUSION

The **Mutual Fund Portfolio Management and NAV Prediction Project** offers an innovative solution for managing and optimizing mutual fund investments using machine learning techniques. By employing the **ARIMA model (ARIMA + Ridge Regression)**, the project successfully predicts future NAV values of mutual funds, providing users with valuable insights into the performance of their investments. This helps users make informed decisions based on accurate predictions of NAV movements.

The system enables users to manage their mutual fund portfolios efficiently, with features like adding, removing, and tracking investments. The incorporation of real-time NAV updates and visualization tools ensures users stay informed about the performance of their holdings. The integration of MySQL for secure transaction and portfolio data storage ensures that the application handles user data efficiently and reliably.

Overall, the project provides a comprehensive, data-driven approach to mutual fund portfolio management and prediction, helping users maximize returns and minimize risks.

## FUTURE SCOPE

The future scope of the **Mutual Fund Portfolio Management and NAV Prediction Project** includes several potential improvements. One of the key areas for enhancement is the use of more advanced prediction models like **LSTM** or **XGBoost**, which could improve the accuracy of NAV predictions by analyzing more complex data patterns. Additionally, integrating **risk management tools** such as portfolio optimization and **risk calculators** like **Value at Risk (VaR)** could help users manage their investment risks more effectively. Incorporating **real-time market data** would allow users to receive up-to-the-minute NAV updates and track their portfolios with greater accuracy. Developing a **mobile app** would also enhance user accessibility, enabling them to manage their investments and receive notifications on the go. Finally, adding features for **tax calculations and compliance** would provide users with the ability to track tax liabilities and ensure they are following local financial regulations.

# CHAPTER-8
# REFERENCES

# CHAPTER-8
# REFERENCES

[1]. AMIRIA Model. "AutoRegressive Moving Average Integrated with Ridge Regression, used for mutual fund NAV prediction in the project."

[2]. MFTool. "A Python-based library for mutual fund data extraction and analysis from AMFI."

[3]. Matplotlib Library. "Used for basic plotting and visualization of mutual fund performance."

[4]. AMFI (Association of Mutual Funds in India). "Source for mutual fund NAVs, fund categories, and general regulations."

[5]. Pandas Documentation. "Used for reading, analyzing, and manipulating mutual fund data from CSV/Excel files."

[6]. Plotly for Python. "Used for advanced data visualization including interactive charts and graphs."

[7]. Seaborn Documentation. "Used for statistical data visualization alongside Matplotlib."

[8]. MoneyControl - Mutual Funds. "Referenced for top holdings, fund performance data, and portfolio details."

[9]. Python Official Documentation. "General reference for programming logic and implementation in the application."

[10]. Yahoo Finance (via yfinance API). "Can be used for live stock data integration if expanded in future work."

[11]. Scikit-learn Documentation. "Used for implementing machine learning models such as Ridge Regression in the AMIRIA model."

[12]. NumPy Documentation. "Provides support for numerical operations and efficient array handling during data preprocessing."