Project Report: Real-Time Stock Portfolio Management System

1. Abstract

This project is a **Console-Based Application** designed to simulate stock market portfolio management. It leverages the **yfinance** library to fetch real-time stock data from the National Stock Exchange (NSE). The system allows users to perform virtual trading operations such as buying stocks, selling stocks, checking real-time prices, and tracking overall portfolio performance (Profit/Loss). The primary goal is to demonstrate the integration of external APIs with Python data structures to manage financial data dynamically.

---

2. Objectives

The primary objectives of this project are:

- **Real-Time Data Fetching:** To retrieve live market data using the Yahoo Finance API.
- **Portfolio Management:** To implement logic for adding (buying) and removing (selling) assets.
- **Financial Calculation:** To automatically calculate the **Weighted Average Buy Price** and Realized Profit/Loss.
- **User Interface:** To provide a simple, menu-driven interface for interaction.

---

3. System Requirements

Hardware

- **Processor:** Intel Core i3 or equivalent.
- **RAM:** 4GB (Minimum).
- **Internet Connection:** Active connection required (to fetch API data).

Software

- **Operating System:** Windows / Linux / macOS.
- **Language:** Python 3.x.
- **External Libraries:** `yfinance` (Install via `pip install yfinance`).

---

4. Methodology & Logic

The system uses a **Dictionary** data structure to act as a temporary database (`portfolio = {}`). The logic handles data persistence only during the runtime of the script.

4.1 Mathematical Formulas

The system uses specific formulas to manage costs and profits:

A. Buying (Weighted Average Price):

When a user buys more of a stock they already own, the system calculates a new average cost basis using the following formula:

$$\text{New Avg Price} = \frac{(\text{Current Qty} \times \text{Current Avg Price}) + (\text{Buy Qty} \times \text{Current Market Price})}{\text{Total New Qty}}$$

B. Selling (Profit Calculation):

When a user sells a stock, the profit is calculated based on the difference between the current market price and the average buy price:

$$\text{Profit} = \text{Sell Qty} \times (\text{Current Market Price} - \text{Avg Buy Price})$$

4.2 Data Flow

1. User selects an option from the Main Menu.
2. Input is validated (Symbol/Quantity).
3. yfinance fetches the live price via symbol + '.NS' (NSE Extension).
4. The Portfolio Dictionary is updated.
5. Results are printed to the console.

---

5. Module Description (Code Breakdown)

The code is divided into five distinct functional blocks:

Function Name	**Description**
get_price(symbol)	Connects to the yfinance Ticker object. Returns the currentPrice or 0 if the symbol is invalid or the connection fails.
check_price(symbol)	A utility function that calls get_price and simply prints the result for the user without executing a trade.
buy_stock(symbol, qty)	Handles the purchase logic. It checks if the stock exists in the portfolio. If yes, it recalculates the average price; if no, it initializes the stock entry.
sell_stock(symbol, qty)	Handles the sales logic. It validates if sufficient quantity exists. It calculates the profit realized on that specific transaction and reduces the held quantity.

show_portfolio()	Iterates through the portfolio dictionary. It fetches the *current* price for every holding to calculate the current total value vs. total invested amount.

6. Testing and Sample Output

Below is a simulation of how the program performs during execution.

Scenario 1: Buying Stocks

Input: Choice 2, Symbol: TATASTEEL, Qty: 10

Process: Fetches live price (e.g., 150.00).

Output: Bought 10 TATASTEEL at 150.00

Scenario 2: Buying More (Averaging)

Input: Choice 2, Symbol: TATASTEEL, Qty: 10

Process: Fetches live price (e.g., 160.00).

Calculation:

$$\frac{(10 \times 150) + (10 \times 160)}{20} = 155.00$$

Internal Update: Portfolio now holds 20 qty at avg 155.00.

Scenario 3: Viewing Portfolio

Input: Choice 4

Output:

TATASTEEL : Qty 20 Buy Avg 155.0 Current 160.0 Value 3200.0

Total Value: 3200.0 Total Profit: 100.0

7. Limitations & Future Scope

Limitations

1. **Data Persistence:** The current version stores data in a Python dictionary. All data is lost when the program exits.
2. **Error Handling:** Basic error handling exists for API calls, but network timeouts or API rate limits are not robustly handled.
3. **Market Hours:** Prices are only "live" during market hours; otherwise, they reflect the last closing price.

Future Scope

1. **Database Integration:** Implement SQL (SQLite or MySQL) to save portfolio data permanently.
2. **GUI Implementation:** Create a visual interface using **Tkinter** or **PyQt** for charts and graphs.
3. **Visual Analytics:** Integrate matplotlib to plot the historical price trend of the stocks in the portfolio.

8. Conclusion

The **Stock Portfolio Management System** successfully demonstrates the application of Python for financial data analysis. By using the yfinance library, the project bridges the gap between static coding and dynamic real-world data. The modular approach allows for easy debugging and future expansion into a full-fledged trading simulation platform.