**Investment System: Project Proposal & Detailed Plan**

1. Project Overview

Goal:

Develop a comprehensive investment system that screens stocks, optimizes portfolio construction, applies commonly used models for price prediction, implements risk management strategies, and allows exploration through an interactive dashboard.

Use Case:

For Portfolio: Demonstrates real-world data science skills in data collection, statistical modelling, machine learning, and visualization.

For Personal Use: Provides actionable stock recommendations based on data-driven insights.

Scope:

✅ Stock Screener (Fundamental & Technical Analysis)

✅ Position Sizing & Portfolio Construction

✅ Price Prediction (Machine Learning & Time-Series Analysis)

✅ Stop Loss, Take Profit & Risk Management

✅ Portfolio Optimization & Final Deployment

✅ Scoring System for Stock Ranking

✅ Dashboard for Explorability

Success Criteria

1. Functionality & Core Features

✅ Stock Screener: The system accurately screens stocks using both fundamental and technical analysis.

✅ Position Sizing & Portfolio Construction: Correct implementation of position sizing (e.g., Kelly Criterion) and portfolio allocation (e.g., mean-variance optimization).

✅ Price Prediction Models: Implementation of ARIMA and LSTM-based price prediction with reasonable accuracy.

✅ Risk Management: Effective stop loss and take profit strategies to mitigate downside risk.

✅ Portfolio Optimization: Backtested portfolio strategies that show superior risk-adjusted returns compared to a benchmark (e.g., S&P 500).

✅ Scoring System: Stocks are correctly ranked based on weighted scoring criteria (Fundamentals, Technical Indicators, Risk Metrics).

✅ Interactive Dashboard: A functional Streamlit/Dash dashboard that allows users to explore and filter stocks.

2. Data Accuracy & Integrity

✅ Data retrieval modules (fundamentals, price data, risk metrics) function correctly and update without duplication.

✅ No missing or inconsistent data in SQL database.

✅ Normalization and ranking of stock scores are correctly applied.

3. Performance Metrics

✅ Backtesting Accuracy: The system’s backtested strategies outperform simple passive investing or at least show robustness.

✅ Predictive Model Performance: Price prediction models achieve acceptable RMSE or MAPE scores on test data.

✅ Risk-Adjusted Returns: Portfolio performance metrics (Sharpe, Sortino ratios) show improvement over traditional strategies.

4. Scalability & Efficiency

✅ The system efficiently handles large datasets without excessive runtime.

✅ New stocks and missing data can be fetched and stored dynamically without reprocessing everything.

5. User Experience & Deployment

✅ The dashboard is intuitive and visually informative.

✅ Deployment on GitHub with complete documentation (README, setup instructions).

✅ A master script (main.py) successfully integrates all components without errors.

6. Professional Showcase & Impact

✅ The project serves as a strong portfolio piece, demonstrating data science, machine learning, and software development skills.

✅ Employers and hiring managers can easily run/test the system based on the provided instructions.

2. Folder-Based Modular Project Structure

investment\_system/

│── database\_and\_data\_storage/ # Fetching stock data

│ ├── database\_setup.py

│── data\_collection/ # Define schema for fundamentals, price data, technical indicators, risk metrics, etc.

│ ├── fetch\_fundamentals.py

│ ├── fetch\_price\_data.py

│ ├── fetch\_risk\_metrics.py

│── data\_processing/ # Cleaning and feature engineering

│ ├── compute\_ratios.py

│ ├── calculate\_alpha\_sortino.py

│── scoring\_system/ # Ranking and normalizing stock scores

│ ├── normalize\_and\_rank.py

│── position\_sizing/ # Determining trade size and allocation

│ ├── kelly\_criterion.py

│── prediction/ # Machine learning models for forecasting

│ ├── arima\_forecast.py

│ ├── lstm\_forecast.py

│── backtesting/ # Testing strategies on historical data

│ ├── backtest\_strategy.py

│── risk\_management/ # Stop Loss, Take Profit strategies

│ ├── stop\_loss\_take\_profit.py

│── web\_app/ # Visualizing stock data

│ ├── app.py

│ ├── layout.py

│ ├── dashboard\_functions.py

│ ├── data\_loader.py

│── database/ # SQL Database storage

│ ├── database\_setup.py

│ ├── insert\_data.py

│── utils/ # Helper functions

│ ├── config.py # Store API keys, constants

│ ├── logger.py # Logging setup

│── main.py # Master script to integrate everything

│── requirements.txt # Python package dependencies

│── README.md # Project documentation

3. Every time you run main.py, the system should:

* Check when each stock was last updated.
* Fetch only the missing data from the last recorded date to today.
* Insert only the new data into the database (avoid duplicates).
* Delete any stock data that is older than the defined time window (e.g., 5 years).
* Could possibly make this a script in the database folder called in main.py

**Parallel Processing for Faster Execution**

Use Dask or Ray for parallel data processing, especially when handling large stock datasets.

**Backtesting Enhancements**

Walk-Forward Optimization: Instead of relying on static train-test splits, use walk-forward testing for robust model validation.

Out-of-Sample Performance Metrics: Ensure backtesting results aren't overfit by evaluating on unseen data.

3. Data Collection Module

Objective:

Retrieve fundamental, technical, and risk-related data for the S&P 500 stocks.

Data Sources & APIs:

|  |  |  |
| --- | --- | --- |
| **Data** | **Source** | **Retrieval Method** |
| Stock Prices (OHLC) | Yahoo Finance | yfinance API |
| Financial Ratios | Alpha Vantage | requests API |
| Market Risk Factors | Kenneth French Data Library | CSV Downloads |
| Technical Indicators (MACD, RSI) | Yahoo Finance | pandas Computation |

<https://www.quantconnect.com/>

<https://www.quantlib.org/>

Implementation:

1. fetch\_fundamentals.py → Retrieves financial metrics (EV/EBITDA, P/E, PEG, CROCI).
2. fetch\_price\_data.py → Gets historical price data for technical analysis.
3. fetch\_risk\_metrics.py → Downloads Fama-French Alpha and Sortino Ratio.

Storage:

Data is stored in an SQL database/Time series Database for efficient retrieval.

Processed and cleaned data saved as CSV files for backup.

4. Data Processing & Feature Engineering

Objective:

Calculate important investment metrics and prepare data for analysis.

Implementation:

compute\_ratios.py → Computes financial ratios like CROCI, PEG, and revenue CAGR.

calculate\_alpha\_sortino.py → Calculates Alpha using the Fama-French 3-Factor model and Sortino Ratio.

Store processed data in the SQL database.

5. Scoring System for Stock Ranking

Objective:

Rank stocks based on a weighted, research-backed scoring model.

Scoring Methodology:

Category

Metrics

Weight (%)

Fundamentals

PEG Ratio, CROCI, Revenue CAGR

50%

Technical Indicators

MACD Crossover, Price % Change

30%

Risk-Adjusted Metrics

Alpha (Fama-French), Sortino Ratio

20%

Implementation:

Normalize all metrics using MinMaxScaler.

Apply weightings and calculate final score.

Rank stocks based on score in normalize\_and\_rank.py.

6. Position Sizing & Portfolio Construction

Objective:

Allocate capital efficiently based on risk and potential return using Mean-Variance Optimization.

Implementation:

Markovitz\_mean\_variance.py → Uses mean-variance optimisation for determining portfolio weights.

Simulate portfolio performance across different allocations.

Store results in portfolio\_allocations table.

Could also use a black-litterman model or Hierarchial Risk Parity – **compare them all**

<https://pyportfolioopt.readthedocs.io/en/latest/>

7. Machine Learning Price Prediction

Objective:

Forecast future stock prices using time-series modeling.

Implementation:

ARIMA Forecasting (arima\_forecast.py) → For short-term trends.

LSTM Neural Networks (lstm\_forecast.py) → For deep learning price prediction.

Could use other models

8. Risk Management: Stop Loss & Take Profit

Objective:

Implement automated risk strategies to maximize gains & minimize losses.

Implementation:

ATR-Based Stop Loss → Adjusts dynamically based on volatility.

Percentage-Based Stop Loss → Fixed % drop from entry price.

Trailing Stop Loss → Locks in gains by adjusting dynamically.

Value at risk and conditional value at risk to estimate worst cases

Stored in stop\_loss\_take\_profit.py.

9. Portfolio Optimization & Deployment

Objective:

Combine stock screener, ML price predictions, and risk management into a portfolio management tool.

Deploy a user-friendly dashboard to explore insights.

Implementation:

Backtest the entire strategy in backtest\_strategy.py.

Could do a monte-carlo simulation to simulate performance and include this in the dashboard

Optimize parameters to improve performance.

Deploy an interactive dashboard (stock\_dashboard.py).

10. web app: Interactive Stock Exploration

**Objective:**

Enable users to filter stocks, view rankings, analyse performance, and track risk through an interactive web-based dashboard.

**Implementation:**

* **Framework:** Use **Streamlit** (simpler) or **Dash** (more flexible) for an interactive dashboard.
* **Integration with Other Modules:**
  + **Stock Screener:** Allow users to filter stocks based on **fundamental & technical metrics** from normalize\_and\_rank.py.
  + **Portfolio Construction:** Display optimal portfolio allocations from markovitz\_mean\_variance.py.
  + **Price Predictions:** Show **ARIMA & LSTM forecasts** from arima\_forecast.py and lstm\_forecast.py in time-series charts.
  + **Risk Management:** Allow users to explore **stop loss/take profit strategies** from stop\_loss\_take\_profit.py.
  + **Backtesting & Performance Metrics:** Show historical performance and Monte Carlo simulations from backtest\_strategy.py.

**Key Features:**

✅ **Stock Ranking & Filtering:** User-friendly controls for selecting stocks based on scoring criteria.  
✅ **Historical Performance Graphs:** Interactive plots of past stock performance.  
✅ **Machine Learning Forecasts:** Graphs of predicted stock prices using ML models.  
✅ **Portfolio Performance Metrics:** Sharpe Ratio, Sortino Ratio, Alpha, and other risk-adjusted returns.  
✅ **Risk Management Exploration:** Visualize stop-loss strategies on selected stocks.

**Deployment:**

1. **Local Testing:** Run with streamlit run stock\_dashboard.py or equivalent Dash setup.
2. **Hosting:** Deploy on **Streamlit Cloud, Render, or Heroku** for public access.
3. **GitHub Integration:** Include a launch link in the **README.md** for easy access.

11. GitHub Deployment & Documentation

Objective:

Ensure the project is clean, well-structured, and easy to showcase.

Use logging (logger.py) to track errors and system performance

Create a README with:

**📌 README Structure Example:**

1. **Project Title & Summary**
   * A one-liner about what your project does.
   * Example: *"An interactive investment system that screens stocks, predicts prices, and optimizes portfolios using data science techniques."*
2. **Table of Contents (if it's a long README)**
3. **Project Motivation & Goals**
   * Why did you create this project?
   * What does it aim to achieve?
4. **Features**
   * Stock screener (metrics used).
   * Portfolio optimization methods (MPT, Black-Litterman).
   * Price prediction models (ARIMA, LSTMs).
   * Risk management strategies.
   * Dashboard interactivity.
5. **Installation & Setup**
   * Dependencies (pip install -r requirements.txt).
   * How to run the project locally.
   * API keys setup if necessary.
6. **Usage Guide (Screenshots if possible!)**
   * How to interact with each component.
   * Example queries / workflows.
7. **Technical Details & Implementation**
   * Model selection, data sources, architecture.
   * Algorithm choices and justifications.
8. **Results & Performance**
   * Accuracy, backtesting results, portfolio performance.
9. **Future Improvements & Next Steps**
   * What could be added/improved?
10. **How to Contribute (Optional)**

* If open-sourcing, include contribution guidelines.

1. **License & Acknowledgments**

* If you use external resources, give credit.

1. Jupyter Notebook Walkthrough for Hiring Managers

**Objective**

Provide an interactive step-by-step guide that demonstrates key features of the investment system. This allows hiring managers and recruiters to test stock screening, portfolio optimization, and machine learning models without needing to set up the entire project.

**Implementation**

* A **Jupyter Notebook (investment\_demo.ipynb)** will be included in the repository.
* The notebook will contain **preloaded data** (so users don’t need API keys to fetch stock data).
* Each section will explain key features with **code, charts, and markdown comments**.

**Structure of the Notebook**

| **Section** | **Description** |
| --- | --- |
| **1. Introduction** | Overview of the system, how it works, and the dataset being used. |
| **2. Stock Screener Demo** | Run fundamental and technical stock screening using preloaded data. |
| **3. Portfolio Optimization** | Optimize portfolio allocation using mean-variance optimization and visualize risk-return profiles. |
| **4. Price Prediction** | Predict stock prices using ARIMA and LSTM, displaying results graphically. |
| **5. Risk Management Strategies** | Demonstrate stop-loss, take-profit, and risk-adjusted metrics like Sharpe/Sortino ratios. |
| **6. Summary & Next Steps** | Recap findings and suggest how the system could be extended further. |

**Minimum Viable Product**

The MVP should demonstrate the core functionalities of the investment system without implementing all advanced features. It will:

* Retrieve stock data (fundamentals and price history)
* Implement a basic stock screener based on key ratios (EV/EBITDA, P/E, revenue CAGR, etc.)
* Construct a simple portfolio (equal-weighted or basic Markowitz mean-variance)
* Implement basic backtesting (historical performance simulation)
* Deploy a basic interactive dashboard (for stock filtering and visualization)

**Plan**

Phase 1: Data Collection & Storage

* Set up a data pipeline to retrieve stock data from Yahoo Finance, Alpha Vantage, or an alternative source.
* Store data in a structured format (SQL database or CSV files for initial implementation).

Phase 2: Basic Stock Screener

* Compute fundamental ratios: P/E, PEG, revenue CAGR, CROCI.
* Rank stocks based on a weighted scoring system.

Phase 3: Portfolio Construction

* Implement equal-weighted allocation (simpler than optimization initially).
* Compute basic risk metrics (e.g., Sharpe ratio).

Phase 4: Backtesting & Performance Evaluation

* Simulate past portfolio performance.
* Compare against a simple benchmark (e.g., S&P 500).

Phase 5: Basic Dashboard Deployment

* Use Streamlit/Dash to allow users to filter stocks.
* Display historical price trends, ratios, and performance metrics.