

# FINAL REPORT

Object Oriented Programming I – Rutgers MQF – Fall 2023

Atheesh Krishnan // Sagar Marathe // Anukansha Dugar

## **PORTFOLIO OPTIMIZATION FRAMEWORK (MAXIMIZING SHARPE RATIO / MINIMIZING VOLATILITY)**

---

### **Synopsis:**

This project involves developing a C++ application for portfolio optimization. It is centered around financial concepts such as the Efficient Frontier, aiming to either minimize the volatility or maximize the Sharpe ratio of a portfolio of stocks. The project begins with a clean dataset of closing prices for five stocks, stored in a local CSV file, and encompasses the calculation of various statistical measures essential for portfolio analysis and optimization.

The output of the project is supposed to be a set of weights for each of the stocks in percentage terms to be held by the investor in order to maximize/minimize the sharpe ratio/volatility respectively.

## **Description:**

- ❑ Data Input:
  - ❑ Reads stock price data from a CSV file.
  - ❑ The CSV file contains closing prices for 5 stocks over a given period with around 500 rows of pricing data.
  - ❑ The data has already been sanitized for processing in this program.
- ❑ Data Processing:
  - ❑ Converts the raw stock price data into a format suitable for analysis (using DataHandler).
- ❑ Statistical Analysis:
  - ❑ Calculates daily returns from the stock price data.
  - ❑ Computes mean daily returns for each stock.
  - ❑ Generates a covariance matrix for the returns, representing the interdependencies between stock returns.
- ❑ Portfolio Optimization:
  - ❑ Implements a portfolio optimization process (using PortfolioOptimizer).
  - ❑ Generates a large number of random portfolio weight combinations.
  - ❑ 1,000,000 portfolios were created to optimize for the best values.
  - ❑ For each portfolio combination:
    - ❑ Calculates the expected portfolio return.
    - ❑ Computes the portfolio volatility (standard deviation).
    - ❑ Determines the Sharpe ratio (a measure of risk-adjusted return).
- ❑ Strategy Implementation:
  - ❑ Employs different strategies to identify optimal portfolios (PortfolioStrategy):
    - ❑ MaxSharpePortfolio: Selects the portfolio with the maximum Sharpe ratio.
    - ❑ MinVolatilityPortfolio: Chooses the portfolio with the minimum volatility.
- ❑ Result Display:
  - ❑ Outputs the optimal portfolios based on the chosen strategies.
  - ❑ For each optimal portfolio:
    - ❑ Displays the weight of each stock both in decimal and percentage formats.
    - ❑ Shows the portfolio's overall Sharpe ratio and volatility for each max sharpe and min vol portfolios.

## **Challenges:**

- One major issue has been to include external libraries without errors. For example, our use case will heavily use the Eigen library in C++ to perform calculations to get the values for daily returns, volatility/standard deviation, covariance and correlation matrices.
- But we had trouble in properly including and using these libraries. Errors faced include: “Not able to read”. Usage: installed eigen using homebrew on Macintosh.
- Other miscellaneous errors faced during including custom built libraries were resolved after checking and updating paths in the VS Code config files like tasks.json, settings.json and launch.json.

## **Object Oriented Design:**

Utilizes Object-Oriented Programming principles for a modular and organized code structure. Includes separate classes for data handling (DataHandler), statistical computations (Statistics), portfolio optimization (PortfolioOptimizer), and result display (PortfolioDisplay). Inheritance is used to create a structured and extendable approach for different portfolio strategies.

### **Base Class: PortfolioStrategy**

- Role of Base Class:
  - The PortfolioStrategy class serves as a base class for different portfolio strategies. It acts as a blueprint defining the behavior that all strategy classes will follow.
- Abstract Methods:
  - The executeStrategy method is our implementation of the abstract method. This method defines a common interface for executing a strategy but leaves the implementation details to the derived classes, as explained below.

### **Derived Classes: MaxSharpePortfolio and MinVolatilityPortfolio**

- Inheriting from PortfolioStrategy:
  - These classes inherit from PortfolioStrategy. By doing so, they automatically have the properties and methods defined in PortfolioStrategy, although they can add or override these as needed.
  - They provide specific implementations of the abstract methods from the base class. For example, MaxSharpePortfolio implements executeStrategy to find the portfolio with the maximum Sharpe ratio, and correspondingly, the MinVolatilityPortfolio implements executeStrategy to find the portfolio with minimum volatility.
- Implementation of Abstract Methods:
  - Each derived class implements the executeStrategy method in a way that aligns with its particular strategy. This implementation is unique to each class and defines how the strategy is executed. Further details on implementation is in the code in the Appendix section.

## **External Libraries:**

### **Eigen**

- ❑ This is an external library used mainly for matrix computations and primarily since our input data set is tabular and can be conveniently manipulated with a matrix format.
- ❑ Calculating Daily Returns: Eigen is used to store and manipulate stock price data and calculate daily returns.
- ❑ Covariance Matrix Calculation: The library provides functions to compute the covariance matrix from the returns data, a key component in assessing the portfolio's risk.
- ❑ Sharpe Ratio and Volatility: Eigen's capabilities in handling vectorized operations are used to calculate metrics like Sharpe ratio and volatility efficiently.

## **Future Scope / Potential Enhancements:**

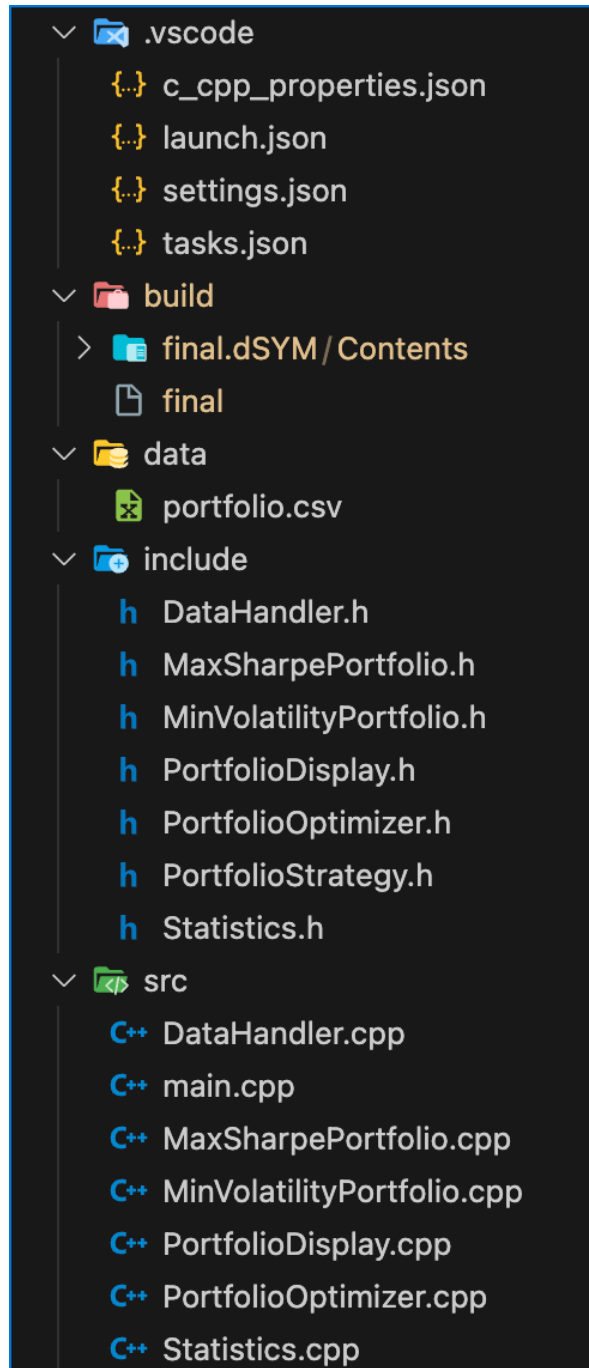
- ❑ Real-time Data Processing:
  - Implement functionality to process real-time stock market data based on live market feeds from providers thus allowing for dynamic portfolio adjustments based on current market conditions.
- ❑ Risk Management Features:
  - Develop risk management tools like Value at Risk (VaR) or Conditional Value at Risk (CVaR) to assess potential losses in adverse market conditions.
- ❑ User Interface Development:
  - Create a graphical interface for visualizing portfolio performance in the form of efficient frontiers, and comparing different optimization strategies.

## **Citations / References:**

- ❑ <https://eigen.tuxfamily.org/dox/>
- ❑ <https://mhittesdorf.wordpress.com/2013/04/27/introducing-quantlib-linear-optimization/>
- ❑ <https://docs.mosek.com/latest/cxxfusion/case-studies-portfolio.html>
- ❑ <https://mhittesdorf.wordpress.com/2013/06/20/introducing-quantlib-portfolio-optimization/>
- ❑ <https://www.lehnerinvestments.com/en/volatility-reduce-stock-portfolio-risk/>
- ❑ <https://moontowermeta.com/the-volatility-drain/>
- ❑ <https://www.geeksforgeeks.org/csv-file-management-using-c/>
- ❑ <https://stackoverflow.com/questions/16446665/c-read-from-csv-file>
- ❑ [https://developers.google.com/optimization/introduction/cpp#c\\_program](https://developers.google.com/optimization/introduction/cpp#c_program)
- ❑ <https://aleixpinardell.github.io/tudat/tutorials/prerequiredKnowledge/externalLibraries/eigen.html>


# APPENDIX

## DIRECTORY STRUCTURE:



## DATA:

The data consists of the closing prices of a set of five stocks in a CSV format, for around 500 rows. Our stock in this example are: Raymond, Cipla, Fortis, Jet Airways and Titan.

```
data >  portfolio.csv
 1  raymond_cp,cipla_cp,fortis_cp,jet_cp,titan_cp
 2  772.8,569.0,205.25,527.5,483.65
 3  785.0,565.6,207.9,534.15,488.3
 4  783.65,562.35,205.9,528.9,481.75
 5  746.95,560.1,196.85,522.7,471.65
 6  723.1,564.95,203.55,520.4,471.15
 7  715.95,563.1,194.8,475.65,481.4
 8  699.55,533.2,196.55,472.85,478.35
 9  662.25,519.65,194.35,464.8,469.2
10  678.3,504.0,202.55,468.95,463.9
11  721.15,488.9,202.75,483.4,473.45
12  695.5,505.35,196.1,474.6,470.65
13  695.45,510.5,198.15,459.65,469.8
14  694.2,516.35,195.15,477.9,474.9
15  693.9,515.55,198.35,484.5,470.7
16  702.65,530.05,192.95,484.85,471.75
17  751.25,534.8,194.9,497.15,551.85
18  728.45,534.65,191.05,497.2,527.15
19  729.05,540.2,194.05,509.1,536.2
20  733.5,550.15,192.95,519.8,529.1
21  732.1,551.1,193.55,527.95,525.45
22  717.7,553.35,191.45,534.8,515.8
23  730.85,551.55,189.45,530.3,520.55
24  721.6,539.9,187.0,532.85,521.9
25  729.9,549.45,197.65,534.3,517.65
```

## OUTPUT:

```
● atheeshkrishnan@Atheeshs-MacBook-Air build % ./final

Maximum Sharpe Ratio Portfolio:
Weights: 0.1187 0.0374 0.0149 0.0042 0.8248
Weights (Percentage): 11.87% 3.74% 1.49% 0.42% 82.48%
Portfolio Sharpe Ratio: 1.4370
Portfolio Volatility: 0.2751

Minimum Volatility Portfolio:
Weights: 0.1413 0.4300 0.1496 0.0405 0.2387
Weights (Percentage): 14.13% 43.00% 14.96% 4.05% 23.87%
Portfolio Sharpe Ratio: 0.5147
Portfolio Volatility: 0.1793
```

## Interpretation:

### ***Maximum Sharpe Ratio Portfolio:***

- ❑ **Weights:** The portfolio allocates 11.87%, 3.74%, 1.49%, 0.42%, and 82.48% to the five stocks, respectively.
- ❑ **Sharpe Ratio:** At 1.4370, this portfolio is considered to have a good risk-adjusted return, given that a Sharpe Ratio greater than 1 is typically seen as acceptable to investors.
- ❑ **Volatility:** The portfolio volatility is 0.2751, which represents the portfolio's standard deviation and indicates a moderate level of risk.

The maximum Sharpe Ratio portfolio focuses on maximizing returns for a given level of risk. The high weight of 82.48% in one stock suggests a strong conviction that this stock will significantly contribute to the portfolio's performance relative to its risk.

### ***Minimum Volatility Portfolio:***

- ❑ **Weights:** The allocations are 14.13%, 43.00%, 14.96%, 4.05%, and 23.87% for the respective stocks.
- ❑ **Sharpe Ratio:** With a Sharpe Ratio of 0.5147, this portfolio has a lower risk-adjusted return compared to the maximum Sharpe Ratio portfolio.
- ❑ **Volatility:** The volatility is 0.1793, which is significantly lower than that of the maximum Sharpe Ratio portfolio, indicating this portfolio is constructed to minimize risk.

The minimum volatility portfolio aims to reduce the potential for large swings in value and is weighted more evenly across the stocks, except for a 43% allocation to the second stock, suggesting a balance between diversification and a strategic overweight in this particular stock.



## CODE:

Main.cpp

```
#include <iostream>
#include <iomanip>
#include "DataHandler.h"
#include "Statistics.h"
#include "PortfolioOptimizer.h"
#include "MaxSharpePortfolio.h"
#include "MinVolatilityPortfolio.h"
#include "PortfolioDisplay.h"

using namespace std;

int main() {

    // Read stock prices from CSV
    std::string filePath = "/Users/atheeshkrishnan/AK/DEV/00P1/rutgersmqf-portfolio-optimization/data/portfolio.csv";
    int numberOfRows = 494;
    int numberOfColumns = 5;
    Eigen::MatrixXd portfolio = DataHandler::readCSV(filePath);

    // Calculate daily returns, mean returns, covariace matrix etc
    Eigen::MatrixXd returns = Statistics::calculateReturns(portfolio);
    Eigen::VectorXd meanReturns = Statistics::meanDailyReturns(returns);
    Eigen::MatrixXd covMatrix = Statistics::covarianceMatrix(returns);
    Eigen::VectorXd volatility = Statistics::volatility(returns);

    // Set the number of random portfolio weights to generate
    int numPortfolios = 1000000;
    std::vector<PortfolioResult> optimizedPortfolios =
PortfolioOptimizer::optimizePortfolio(
    meanReturns, covMatrix, numPortfolios, numberOfColumns);

    // Strategy Execution
    MaxSharpePortfolio maxSharpeStrategy;
    MinVolatilityPortfolio minVolStrategy;

    PortfolioResult maxSharpePortfolio =
maxSharpeStrategy.executeStrategy(optimizedPortfolios);
    PortfolioResult minVolatilityPortfolio =
minVolStrategy.executeStrategy(optimizedPortfolios);

    PortfolioDisplay::displayPortfolio("Maximum Sharpe Ratio Portfolio",
maxSharpePortfolio);
```

```

    PortfolioDisplay::displayPortfolio("Minimum Volatility Portfolio",
minVolatilityPortfolio);

    return 0;
}

```

## DataHandler.cpp

```

#include "DataHandler.h"
#include <fstream>
#include <sstream>

// Read CSV file and return data in Eigen::MatrixXd format
Eigen::MatrixXd DataHandler::readCSV(const std::string& file) {
    std::ifstream inFile(file);
    std::string line;

    // Skipping the header line which has stock names
    std::getline(inFile, line);

    // Initialize the matrix for 494 rows and 5 columns (hardcoded based on our data)
    Eigen::MatrixXd data(494, 5);
    int row = 0;

    while (std::getline(inFile, line)) {
        std::vector<std::string> tokens = split(line, ',');
        if (tokens.size() != 5) {
            // Handle error for rows with incorrect number of columns
            continue;
        }
        for (int col = 0; col < 5; col++) {
            try {
                data(row, col) = std::stod(tokens[col]);
            } // Handle error for invalid string-to-double conversion
            catch (const std::invalid_argument& e) {
            }
        }
        row++;
        // Ensure we don't exceed the expected number of rows
        if (row >= 494) break;
    }

    return data;
}

```

```
// Helper function to split a string using a delimiter
std::vector<std::string> DataHandler::split(const std::string& s, char delimiter) {
    std::vector<std::string> tokens;
    std::string token;
    std::istringstream tokenStream(s);

    while (std::getline(tokenStream, token, delimiter)) {
        tokens.push_back(token);
    }

    return tokens;
}
```

DataHandler.h

```
#include <Eigen/Dense>
#include <string>
#include <vector>

class DataHandler {
public:
    // Function to read CSV data into an Eigen matrix
    static Eigen::MatrixXd readCSV(const std::string& file);

private:
    // Helper function to split a string by a delimiter
    static std::vector<std::string> split(const std::string& s, char delimiter);
};
```

Statistics.cpp

```
#include "Statistics.h"
#include <cmath>

// Calculating daily returns
Eigen::MatrixXd Statistics::calculateReturns(const Eigen::MatrixXd& prices) {
    Eigen::MatrixXd returns = Eigen::MatrixXd(prices.rows() - 1, prices.cols());
    for (int i = 1; i < prices.rows(); ++i) {
        returns.row(i - 1) = (prices.row(i) - prices.row(i - 1)).cwiseQuotient(prices.row(i - 1));
    }
    return returns;
}

// Calculating Mean daily returns
Eigen::VectorXd Statistics::meanDailyReturns(const Eigen::MatrixXd& returns) {
```

```

        return returns.colwise().mean();
    }

    // Calculating Covariance of daily returns
    Eigen::MatrixXd Statistics::covarianceMatrix(const Eigen::MatrixXd& returns) {
        int n = returns.rows();
        Eigen::MatrixXd centered = returns.rowwise() - returns.colwise().mean();
        return (centered.adjoint() * centered) / double(n - 1);
    }

    // Calculating Volatility (Standard Deviation) of daily returns
    Eigen::VectorXd Statistics::volatility(const Eigen::MatrixXd& returns) {
        Eigen::MatrixXd centered = returns.rowwise() - returns.colwise().mean();
        return (centered.array().square().colwise().sum() / double(returns.rows() -
1)).sqrt();
    }
}

```

## Statistics.h

```

#include <Eigen/Dense>
#include <vector>

class Statistics {
public:
    static Eigen::MatrixXd calculateReturns(const Eigen::MatrixXd& prices);
    static Eigen::VectorXd meanDailyReturns(const Eigen::MatrixXd& returns);
    static Eigen::MatrixXd covarianceMatrix(const Eigen::MatrixXd& returns);
    static Eigen::VectorXd volatility(const Eigen::MatrixXd& returns);
};

```

## PortfolioOptimizer.cpp

```

#include "PortfolioOptimizer.h"

// Generates random weights for portfolio
std::vector<double> PortfolioOptimizer::generateRandomWeights(int numStocks) {
    std::vector<double> weights(numStocks);
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(0, 1);

    double sum = 0.0;
    for (int i = 0; i < numStocks; ++i) {
        weights[i] = dis(gen);
        sum += weights[i];
    }
}

```

```

    // Normalize the weights so they sum to 1
    for (double &weight : weights) {
        weight /= sum;
    }

    return weights;
}

// Calculates the return of the portfolio
double PortfolioOptimizer::calculatePortfolioReturn(const Eigen::VectorXd&
meanReturns,
                                                    const std::vector<double>&
weights) {
    double portfolioReturn = 0.0;
    for (size_t i = 0; i < weights.size(); ++i) {
        portfolioReturn += meanReturns[i] * weights[i];
    }
    return portfolioReturn * 252; // Assuming 252 trading days in a year
}

// Calculates the volatility of the portfolio
double PortfolioOptimizer::calculatePortfolioVolatility(const Eigen::MatrixXd&
covMatrix,
                                                    const std::vector<double>&
weights) {
    Eigen::VectorXd w = Eigen::Map<const Eigen::VectorXd>(weights.data(),
weights.size());
    double volatility = std::sqrt(w.transpose() * covMatrix * w);
    double annualized_vol = volatility * std::sqrt(252); // Annualizing the volatility
    return annualized_vol;
}

// Optimizes the portfolio over a number of random allocations
std::vector<PortfolioResult> PortfolioOptimizer::optimizePortfolio(const
Eigen::VectorXd& meanReturns,
                                                                    const
Eigen::MatrixXd& covMatrix,
                                                                    int numPortfolios,
int numStocks) {
    std::vector<PortfolioResult> results;
    for (int i = 0; i < numPortfolios; ++i) {
        std::vector<double> w = generateRandomWeights(numStocks);
        double portfolioReturn = calculatePortfolioReturn(meanReturns, w);
        double portfolioVol = calculatePortfolioVolatility(covMatrix, w);
        double sharpeRatio = portfolioReturn / portfolioVol;
        results.push_back({portfolioReturn, portfolioVol, sharpeRatio, w});
    }
}

```

```
    return results;
}
```

## PortfolioOptimizer.h

```
#ifndef PORTFOLIOOPTIMIZER_H
#define PORTFOLIOOPTIMIZER_H

#include <Eigen/Dense>
#include <vector>
#include <random>

// custom structure for resulting dataframe from which min vol and max sharpe weights
// will be extracted
struct PortfolioResult {
    double return_;
    double volatility;
    double sharpeRatio;
    std::vector<double> weights;
};

class PortfolioOptimizer {
public:
    static std::vector<PortfolioResult> optimizePortfolio(const Eigen::VectorXd&
meanReturns,
                                                         const Eigen::MatrixXd&
covMatrix,
                                                         int numPortfolios, int
numStocks);

private:
    static std::vector<double> generateRandomWeights(int numStocks);
    static double calculatePortfolioReturn(const Eigen::VectorXd& meanReturns,
                                         const std::vector<double>& weights);
    static double calculatePortfolioVolatility(const Eigen::MatrixXd& covMatrix,
                                             const std::vector<double>& weights);
};

#endif
```

## PortfolioStrategy.h

```
#ifndef PORTFOLIOSTRATEGY_H
#define PORTFOLIOSTRATEGY_H
```

```

#include <vector>
#include "PortfolioOptimizer.h"

class PortfolioStrategy {
public:
    PortfolioStrategy() = default;
    virtual ~PortfolioStrategy() = default;

    // Method to execute the strategy
    virtual PortfolioResult executeStrategy(const std::vector<PortfolioResult>&
portfolios) = 0;
};

#endif

```

### MaxSharpePortfolio.cpp

```

#include "MaxSharpePortfolio.h"
#include <limits>

PortfolioResult MaxSharpePortfolio::executeStrategy(const
std::vector<PortfolioResult>& portfolios) {
    double maxSharpe = std::numeric_limits<double>::lowest();
    PortfolioResult maxSharpePortfolio;

    for (const auto& portfolio : portfolios) {
        if (portfolio.sharpeRatio > maxSharpe) {
            maxSharpe = portfolio.sharpeRatio;
            maxSharpePortfolio = portfolio;
        }
    }
    return maxSharpePortfolio;
}

```

### MaxSharpePortfolio.h

```

#ifndef MAXSHARPEPORTFOLIO_H
#define MAXSHARPEPORTFOLIO_H

#include "PortfolioStrategy.h"

class MaxSharpePortfolio : public PortfolioStrategy {
public:
    MaxSharpePortfolio() = default;

```

```

        // Implementation specific to Max Sharpe Ratio strategy
        PortfolioResult executeStrategy(const std::vector<PortfolioResult>& portfolios)
override;
};

#endif

```

#### MinVolatilityPortfolio.cpp

```

#include "MinVolatilityPortfolio.h"
#include <limits>

PortfolioResult MinVolatilityPortfolio::executeStrategy(const
std::vector<PortfolioResult>& portfolios) {
    double minVolatility = std::numeric_limits<double>::max();
    PortfolioResult minVolatilityPortfolio;

    for (const auto& portfolio : portfolios) {
        if (portfolio.volatility < minVolatility) {
            minVolatility = portfolio.volatility;
            minVolatilityPortfolio = portfolio;
        }
    }

    return minVolatilityPortfolio;
}

```

#### MinVolatilityPortfolio.h

```

#ifndef MINVOLATILITYPORTFOLIO_H
#define MINVOLATILITYPORTFOLIO_H

#include "PortfolioStrategy.h"

class MinVolatilityPortfolio : public PortfolioStrategy {
public:
    // Implementation specific to Min volatility strategy
    PortfolioResult executeStrategy(const std::vector<PortfolioResult>& portfolios)
override;
};

#endif

```

#### PortfolioDisplay.cpp



```

#include "PortfolioDisplay.h"
#include <iomanip>

void PortfolioDisplay::displayPortfolio(const std::string& title, const
PortfolioResult& portfolio) {
    using namespace std;

    cout << "\n" << title << ":" << endl;
    cout << "Weights: ";
    for (const auto& weight : portfolio.weights) {
        cout << fixed << setprecision(4) << weight << " ";
    }
    cout << "\nWeights (Percentage): ";
    for (const auto& weight : portfolio.weights) {
        cout << fixed << setprecision(2) << weight * 100 << "% ";
    }
    cout << "\nPortfolio Sharpe Ratio: " << fixed << setprecision(4) <<
portfolio.sharpeRatio << endl;
    cout << "Portfolio Volatility: " << fixed << setprecision(4) <<
portfolio.volatility << endl;
}

```

#### PortfolioDisplay.h

```

#ifndef PORTFOLIODISPLAY_H
#define PORTFOLIODISPLAY_H

#include <iostream>
#include <vector>
#include "PortfolioOptimizer.h"

class PortfolioDisplay {
public:
    static void displayPortfolio(const std::string& title, const PortfolioResult&
portfolio);
};

#endif

```