Portfolio Optimization Framework

Technical Documentation Report

Zaid Elkasemy

April 9, 2025

Contents

1 Introduction

This report provides a comprehensive overview of the Portfolio Optimization Framework, a Python-based system for financial portfolio optimization. The project implements modern portfolio theory concepts, primarily focused on maximizing the Sharpe ratio to find optimal asset allocations within a given investment universe.

1.1 Key Features

The framework provides the following key features:

- Portfolio optimization using the Sharpe ratio
- Interactive web interface for visualization and parameter input
- Asset allocation visualization with weights and performance metrics
- Historical performance analysis of optimized portfolios
- Fallback mechanisms when market data is unavailable

1.2 Target Audience

This framework is designed for:

- Financial analysts looking to implement portfolio optimization
- Traders seeking quantitative allocation strategies
- Students learning about modern portfolio theory
- Investors wanting to apply mathematical optimization to their portfolios

2 Theoretical Background

2.1 Modern Portfolio Theory

Modern Portfolio Theory (MPT) was introduced by Harry Markowitz in 1952. The central idea is that an investor's objective is to maximize expected return for a given level of risk, or equivalently, minimize risk for a given level of expected return.

2.2 The Sharpe Ratio

The Sharpe ratio, developed by William F. Sharpe, measures the excess return (or risk premium) per unit of risk in an investment asset or portfolio. It is defined as:

$$S = \frac{E[R_p] - R_f}{\sigma_p} \tag{1}$$

Where:

- $E[R_p]$ is the expected portfolio return
- R_f is the risk-free rate
- σ_p is the portfolio standard deviation (volatility)

The Sharpe ratio is a key metric used in this framework for portfolio optimization. A higher Sharpe ratio indicates better risk-adjusted returns.

3 System Architecture

3.1 Project Structure

The project follows a modular architecture with clear separation of concerns:

```
# Project Structure
.
|-- app.py                 # Main Streamlit application entry point
|-- requirements.txt       # Project dependencies
|-- README.md             # Project documentation
|-- src/                  # Core functionality
|    |-- settings.py      # Portfolio settings class
|    |-- optimization.py  # Portfolio optimization algorithms
|    |-- strategy_factory.py # Strategy creation factory
|    |-- strategies/      # Strategy implementations
|    |    |-- strategy.py  # Base strategy abstract class
|    |    |-- sharpe_ratio.py # Sharpe ratio implementation
|-- notebooks/           # Jupyter notebooks for analysis
```

3.2 Component Overview

The system is composed of the following key components:

- **Web Interface Layer**: Implemented using Streamlit, providing user input, visualization, and results display.
- **Domain Layer**: Contains the core business logic including strategy implementations and portfolio optimization.
- **Data Layer**: Handles data acquisition from Yahoo Finance API and data processing.

4 Implementation Details

4.1 Settings Module

The `PortfolioSettings` class in `settings.py` encapsulates the configuration parameters for a portfolio optimization run:

```python
# Key elements from settings.py
class PortfolioSettings:
    def __init__(self, capital: float, tickers: list[str], strategy_type: str):
        self.capital = capital
        self.tickers = tickers

        # Try to get risk-free rate from Yahoo Finance, use default if it fails
        try:
            tnx_data = yf.download("^IRX", period="1d")
            if not tnx_data.empty:
                self.risk_free_rate = tnx_data.iloc[-1]["Close"] / 100
            else:
                self.risk_free_rate = 0.045  # 4.5% as default
        except Exception as e:
            print(f"Could not retrieve risk-free rate: {e}")
            self.risk_free_rate = 0.045  # 4.5% as default

        self.strategy_type = strategy_type
```

4.2 Strategy Pattern Implementation

The framework uses the Strategy pattern to implement different portfolio optimization metrics. The base `Strategy` class in `strategy.py` defines the abstract interface:

```python
# Base Strategy class (abstract)
class Strategy:
    def __init__(self, settings: PortfolioSettings, start_date: str, end_date:
    str) -> None:
        self.settings = settings
        self.start_date = start_date
        self.end_date = end_date

    @abstractmethod
    def compute_metric(self, weights: List[float]):
        pass
```

The `SharpeRatio` class extends this base class and implements the Sharpe ratio calculation:

```python
# Key elements from sharpe_ratio.py
class SharpeRatio(Strategy):
    def compute_expected_returns(self, weights: List[float]):
        log_returns = self.data['Close'].pct_change().dropna()
        expected_returns = log_returns.mean()
        return sum([weights[i] * expected_returns[self.settings.tickers[i]]
                    for i in range(len(self.settings.tickers))])

    def compute_correlation_matrix(self, weights: List[float]):
        correlation_matrix = self.data['Close'].pct_change().dropna().corr()
        return weights @ correlation_matrix @ weights

    def compute_metric(self, weights: List[float]):
        expected_returns = self.compute_expected_returns(weights)
        volatility = self.compute_correlation_matrix(weights)
        sharpe_ratio = (expected_returns - self.settings.risk_free_rate) / np.
    sqrt(volatility)
        return sharpe_ratio
```

4.3 Optimization Process

The `OptimizationPortfolio` class in `optimization.py` handles the optimization process using scipy's optimization functions:

```python
# Key elements from optimization.py
class OptimizationPortfolio:
    def compute_negative_sharpe_ratio(self, weights):
        return -self.strategy.compute_metric(weights)

    def optimize(self):
        initial_weights = {stock: 1/len(self.settings.tickers)
                           for stock in self.settings.tickers}
        bounds = [(0, 1) for _ in range(len(self.settings.tickers))]
        constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
        result = minimize(self.compute_negative_sharpe_ratio,
                          list(initial_weights.values()),
                          method='SLSQP',
                          bounds=bounds,
                          constraints=constraints)
        self.weights = {stock: result.x[i]
                        for i, stock in enumerate(self.settings.tickers)}
        return self.weights
```

4.4 Web Interface

The web interface is implemented using Streamlit in `app.py`. It provides:

- Input controls for capital, stock selection, and date range
- Optimization execution
- Results visualization with tables and charts
- Performance metrics display

5 Error Handling and Resilience

The framework implements several error-handling mechanisms to ensure robustness:

- Fallback to default risk-free rate when Yahoo Finance API fails
- Generation of synthetic data when historical stock prices are unavailable
- Exception handling with informative error messages
- Validation of optimization inputs

6 Usage Guide

6.1 Installation

To set up the framework, follow these steps:

```bash
# Clone the repository
git clone <your-repository-url>
cd Finance

# Install dependencies
pip install -r requirements.txt
```

6.2 Running the Application

To launch the web interface:

```bash
streamlit run app.py
```

This will start a local web server, typically at http://localhost:8501.

6.3 Using the Interface

The interface workflow is as follows:

1. Set the initial capital amount in the sidebar
2. Enter or modify the list of stock tickers
3. Select the optimization strategy (currently Sharpe ratio)
4. Choose the historical date range for analysis
5. Click "Optimize Portfolio" to run the optimization

6. Review the results, including:
    - Optimal allocation weights
    - Pie chart visualization
    - Historical performance chart
    - Risk metrics (Annual Return, Volatility, Sharpe Ratio)

7 Extension Points

The framework is designed to be extensible in several dimensions:

7.1 Adding New Strategies

To implement a new portfolio optimization strategy:

1. Create a new class in the `strategies` directory that extends `Strategy`
2. Implement the `compute_metric` method
3. Add the new strategy type to the strategy factory
4. Update the UI to include the new strategy option

7.2 Enhanced Data Sources

The current implementation uses Yahoo Finance. To add alternative data sources:

1. Create adapter classes for new data providers
2. Implement a data source factory pattern
3. Update the UI to allow selection of data sources

8 Conclusion

The Portfolio Optimization Framework provides a robust implementation of modern portfolio theory concepts with a user-friendly interface. By implementing the Sharpe ratio optimization and providing visualization tools, it enables users to make data-driven investment decisions.

The modular architecture and resilient implementation make it suitable for both educational purposes and practical financial analysis. The framework can be extended with additional optimization strategies and data sources to meet more specialized needs.