National College of Ireland

Configuration Manual

MSc Research Project
Fintech

Martha Abril Blanco Araujo
Student ID: 23401419

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Martha Abril Blanco Araujo
Student ID:	23401419
Programme	FinTech
Year:	2025
Module:	Practicum
Supervisor:	Noel Cosgrave
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	1,033
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Abril Blanco Araujo
Date:	01/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Martha Abril Blanco Araujo
23401419: 23401419

01/08/2025

1 Introduction

This document provides a comprehensive guide for replicating the comparative study of portfolio construction using AI tools versus traditional methods. The project evaluates three distinct approaches: vague prompts to ChatGPT-4, technical prompts with specific requirements, and Modern Portfolio Theory optimization.

Research Question: To what extent can Large Language Model (LLM)-based investment tools support financially inexperienced investors in constructing diversified and risk-balanced stock portfolios?

2 System Specifications and Environment Setup

2.1 Hardware

- **Device:** Dell Inspiron 15-3530
- **Processor:** Intel Core i7
- **RAM:** 16 GB

2.2 Software and Tools

- **Programming Language:** Python 3.11.13
- **Development Environment:** Google Colab Pro (Jupyter Notebook interface)
- **Main Libraries:** NumPy, pandas, yfinance, matplotlib, seaborn
- **AI System:** ChatGPT-4 (Large Language Model, OpenAI) — used via prompt-based interaction.

3 Package Installation and Dependencies

The project requires specific Python packages for data collection, portfolio optimization, and statistical analysis:

```
# Install required packages
!pip install yfinance pandas numpy matplotlib scipy cvxpy scikit-learn kagglehub --quiet

# Import essential libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import cvxpy as cp
from scipy.optimize import minimize
from datetime import datetime
import kagglehub
!pip install edhec-risk-kit --quiet
import kagglehub
import sys
```

Figure 1: Required Package Installation

As shown in Figure 1, the installation process includes core packages for financial data analysis (yfinance), numerical computation (numpy, scipy), optimization (cvxpy), and data manipulation (pandas). Version control ensures consistent results across different execution environments.

4 AI Portfolio Generation Setup

4.1 Methodology

The AI-based portfolio generation process employed two distinct prompting strategies to analyze the impact of prompt sophistication on investment recommendations. AI model responses were obtained through conversational interface in natural language format and subsequently processed into standardized tabular presentations for comparative analysis. All allocations and stock selections presented reflect the original AI recommendations without modification.

4.2 Configuration Parameters

Date: May 16, 2025
Investment Amount: €1,000 (simulated)
AI Model: OpenAI GPT-4 conversational interface (ChatGPT)
Model Configuration: Default parameters, no specific financial training modifications

4.3 Portfolio 1 Configuration

Prompt Strategy: Simple, vague request

Exact Prompt:

```
I have €1,000 to invest, give me a portfolio.
```

AI Model Response: The AI model generated the following portfolio allocation based on the provided prompt:

Ticker	Company	EUR	%
NVDA	NVIDIA	200	20.0
ON	ON Semiconductor	100	10.0
CRWV	CoreWeave	150	15.0
VNOM	Viper Energy	150	15.0
NBR	Nabors Industries	100	10.0
GH	Guardant Health	150	15.0
DG	Dollar General	150	15.0
Total		**1,000**	**100.0**

Table 1: AI-Generated Portfolio 1 Allocation

4.4 Portfolio 2 Configuration

Prompt Strategy: Technical, detailed specifications

Exact Prompt:

> "I'm looking to invest 1,000 EUR for the next 3 to 6 months specifically into individual common stocks, aiming for maximum capital appreciation with a moderately aggressive risk tolerance. I need a portfolio of highly liquid stocks that have strong short-term growth catalysts and a focused, strategic diversification to manage risk without diluting potential returns. Please provide the exact percentage allocation and ticker for each recommended stock."

AI Model Response: The AI model generated the following portfolio allocation based on the provided prompt:

Ticker	Company	EUR	%
ASML.AS	ASML Holding	250	25.0
RHM.DE	Rheinmetall AG	200	20.0
SAP.DE	SAP SE	150	15.0
GLE.PA	Société Générale	100	10.0
SIE.DE	Siemens AG	100	10.0
CRDO	Credo Tech	100	10.0
IAG	IAMGOLD Corp	100	10.0
Total		**1,000**	**100.0**

Table 2: AI-Generated Portfolio 2 Allocation

5 Portfolio 3; Modern Portfolio Theory Optimization

The MPT framework was implemented based on the established methodology from Yousef Saeedian's Kaggle notebook[1]. Adapting the core functions and optimization strategies

[1] `https://www.kaggle.com/code/yousefsaeedian/modern-portfolio-theory-mpt`

to the tickers and time selected. The implementation included Monte Carlo simulation to generate 4000 random portfolio combinations, from which the optimal portfolios were identified according to MPT criteria including minimum volatility, maximum Sharpe ratio, and target return optimization strategies.

Figure 2: Kaggle Dataset Integration and Framework Access

Figure 2 illustrates the integration of two key Kaggle resources: Yousef Saeedian's Modern Portfolio Theory implementation and Roger Tovalle's EDHEC Risk Kit. This combination provides both optimization algorithms and advanced risk calculation capabilities essential for Portfolio 3 construction.

The MPT-based Portfolio 3 construction employed distinct optimization strategies to identify the most practical allocation for implementation. The strategy approach balances mathematical optimality benchmarking the EDHEC Risk Management toolkit from Roger Tovalle's Kaggle implementation[2] which provides a baseline to compute the MPT and the Montecarlo simulation, the exctracted following package:

```
import edhec_risk_kit_ as erk
```

It was used specifically for:

- Efficient frontier simulation (4000 random portfolios)
- Portfolio selection from optimization candidates
- Risk-return calculations for MPT implementation

As demonstrated in Figure 3, the code optimization process includes minimum volatility portfolios, target return optimization, maximum Sharpe ratio calculations, and volatility-constrained optimization.

[2] `https://www.kaggle.com/code/rogertovalle/edhec-risk-kit`

```python
# MPT Portfolio 3 Implementation - Mathematical Framework
tickers = ['NVDA', 'PLTR', 'AMZN', 'VNOM', 'RHM.DE', 'SAP']
n_assets = len(tickers)
# Historical data collection (2018-2025 for parameter estimation)
stocks = pd.DataFrame()
start_date_hist = datetime(2018, 5, 15)
end_date_hist = datetime(2025, 7, 18)
for stock_name in tickers:
    data = yf.download(stock_name, start=start_date_hist, end=end_date_hist)
    stocks[stock_name] = data["Close"]
# Calculate daily returns (Equation 1)
daily_rets = stocks.pct_change().dropna()
# Correlation analysis (Equation 5): ρ12 = Cov(R1,R2) / [SD(R1) × SD(R2)]
correlation_matrix = daily_rets.corr()
# Portfolio variance parameters (Equation 6): σp² = w^T Σ w
mean_rets = daily_rets.mean()
cov_rets = daily_rets.cov()
```

```python
# Monte Carlo simulation (4000 portfolios for efficient frontier)
num_portfolios = 4000
periods_per_year = 252
risk_free_rate = 0.042
returns = np.zeros(num_portfolios)
volatilities = np.zeros(num_portfolios)
sharpe_ratios = np.zeros(num_portfolios)
weights_array = np.zeros((num_portfolios, n_assets))
for i in range(num_portfolios):
    weights = np.random.random(n_assets)
    weights /= np.sum(weights)
    # Calculate portfolio metrics using EDHEC Risk Kit
    ann_rets = erk.annualize_rets(daily_rets, periods_per_year)
    portfolio_ret = erk.portfolio_return(weights, ann_rets)
    # Portfolio volatility (Equation 2): σp = √(w^T Σ w)
    portfolio_vol = erk.portfolio_volatility(weights, cov_rets)
    portfolio_vol = erk.annualize_vol(portfolio_vol, periods_per_year)
    # Sharpe ratio (Equation 3): (Rp - Rf) / σp
    portfolio_spr = erk.sharpe_ratio(portfolio_ret, risk_free_rate, periods_per_year, v=portfolio_vol)
    returns[i] = portfolio_ret
    volatilities[i] = portfolio_vol
    sharpe_ratios[i] = portfolio_spr
    weights_array[i] = weights
```

```python
    # Four optimization strategies implementation:
    # Strategy 1: Minimum Volatility (Equation 12)
    weights_min_vol = erk.minimize_volatility(ann_rets, cov_rets)
    # Strategy 2: Target Return Optimization (Equation 13)
    target_return = 0.40
    weights_target_ret = erk.minimize_volatility(ann_rets, cov_rets, target_return)
    # Strategy 3: Maximum Sharpe Ratio (Equation 14)
    weights_max_sharpe = erk.maximize_sharpe_ratio(ann_rets, cov_rets, risk_free_rate, periods_per_year)
    # Strategy 4: Sharpe with Volatility Constraint (Equation 15)
    target_volatility = 0.28
    weights_sharpe_vol = erk.maximize_sharpe_ratio(ann_rets, cov_rets, risk_free_rate, periods_per_year, target_volatility)
    # Efficient frontier construction
    ef_frontier = erk.efficient_frontier(50, daily_rets, cov_rets, periods_per_year)
```

Figure 3: Code of MPT Optimization Strategies and Mathematical Framework

5.1 Portfolio 3 Final Selection

The MPT optimization process generated multiple portfolio candidates based on different optimization objectives (GMV, MSR, Target Return, and Target Volatility strategies). Following mathematical optimization and performance evaluation, the Target 40% Return portfolio was selected as the final Portfolio 3 allocation. The final portfolio selection involved human judgment to choose the most implementable solution from the mathematically optimal candidates, resulting in the following output:

Ticker	Company	EUR	%
NVDA	NVIDIA	258	25.8
PLTR	Palantir Technologies	154	15.4
AMZN	Amazon	193	19.3
VNOM	Viper Energy	147	14.7
RHM.DE	Rheinmetall AG	142	14.2
SAP	SAP SE	106	10.6
Total		**1,000**	**100.0**

Table 3: Human selected- Portfolio 3 Allocation

6 Historical Data Collection

Yahoo Finance API (*Yahoo Finance Historical Data* (2025)serves as the primary data source for historical price information across all portfolio constituents. The data collection process maintains consistency in time periods and data quality across all three portfolio approaches. As previously shown, to import the necessary historical data, in first instance it was necessary to:

```
import yfinance as yf
```

After it, the following figure 4 illustrates the data collection framework, highlighting the pertinent tickers, a consistent period range application for the May 16 - July 20, 2025 analysis period and closing prices.

```
# Portfolio compositions from AI recommendations
portfolio_1_tickers = ['NVDA', 'ON', 'CRWV', 'VNOM', 'NBR', 'GH', 'DG']
portfolio_2_tickers = ['ASML.AS', 'RHM.DE', 'SAP.DE', 'GLE.PA', 'SIE.DE', 'CRDO', 'IAG']
portfolio_3_tickers = ['NVDA', 'PLTR', 'AMZN', 'VNOM', 'RHM.DE', 'SAP']

# Analysis period (Across all portfolios)
start_date = '2025-05-16'
end_date = '2025-07-20'

# Data collection function implementing Equation (1): Rt = (Pt - Pt-1)/Pt-1 × 100
def collect_portfolio_data(tickers, start, end):
    data = yf.download(tickers, start=start, end=end, progress=False)
    returns = data['Close'].pct_change().dropna() * 100  # Daily Returns (Eq. 1)
    return returns

# Collect data for analysis period
returns_p1 = collect_portfolio_data(portfolio_1_tickers, start_date, end_date)
returns_p2 = collect_portfolio_data(portfolio_2_tickers, start_date, end_date)
returns_p3 = collect_portfolio_data(portfolio_3_tickers, start_date, end_date)
```

Figure 4: Historical Data Collection Implementation

6.1 Risk Analysis Implementation

Comprehensive risk analysis forms the foundation for portfolio comparison across all three approaches. The implementation includes traditional risk metrics alongside advanced measures for tail risk assessment.

```python
# Risk Analysis Implementation - Core Metrics Calculation
def calculate_comprehensive_risk_metrics(returns, weights):
    """
    Calculate risk metrics following established formulas
    """
    # Portfolio returns calculation
    portfolio_returns = (returns * weights).sum(axis=1)
    # Basic metrics
    daily_avg_return = portfolio_returns.mean()
    daily_volatility = portfolio_returns.std()  # Equation (2) daily version
    max_daily_gain = portfolio_returns.max()
    max_daily_loss = portfolio_returns.min()
    # Value at Risk (VaR) - 95% and 99% confidence levels
    var_95 = np.percentile(portfolio_returns, 5)
    var_99 = np.percentile(portfolio_returns, 1)
    # Conditional Value at Risk (CVaR) - Expected Shortfall
    cvar_95 = portfolio_returns[portfolio_returns <= var_95].mean()
    cvar_99 = portfolio_returns[portfolio_returns <= var_99].mean()
    # Correlation analysis (Equation 5)
    correlation_matrix = returns.corr()
    avg_correlation = correlation_matrix.values[np.triu_indices_from(correlation_matrix.values, k=1)].mean()
    return {
            'daily_return': daily_avg_return,
            'volatility': daily_volatility,
            'max_gain': max_daily_gain,
            'max_loss': max_daily_loss,
            'var_95': var_95,
            'var_99': var_99,
            'cvar_95': cvar_95,
            'cvar_99': cvar_99,
            'avg_correlation': avg_correlation
        }
# Portfolio weights for analysis
weights_p1 = np.array([0.20, 0.10, 0.15, 0.15, 0.10, 0.15, 0.15])  # Portfolio 1
weights_p2 = np.array([0.25, 0.20, 0.15, 0.10, 0.10, 0.10, 0.10])  # Portfolio 2
weights_p3 = np.array([0.1785, 0.0499, 0.1089, 0.0186, 0.4462, 0.1979])  # Portfolio 3
# Calculate risk metrics for all portfolios
risk_metrics_p1 = calculate_comprehensive_risk_metrics(returns_p1, weights_p1)
risk_metrics_p2 = calculate_comprehensive_risk_metrics(returns_p2, weights_p2)
risk_metrics_p3 = calculate_comprehensive_risk_metrics(returns_p3, weights_p3)
```

Figure 5: Code of the Comprehensive Risk Analysis Framework

Figure 5 shows the implementation of core risk metrics including daily volatility, Value at Risk (VaR), Conditional Value at Risk (CVaR), and maximum drawdown calculations. This framework provides consistent risk assessment methodology across all portfolio approaches.

6.2 Statistical Analysis and Market Benchmarking

Statistical analysis includes market beta calculation, CAPM-based expected returns, and Jensen's Alpha for risk-adjusted performance evaluation. S&P 500 serves as the market benchmark for all beta calculations.

```python
# Statistical Analysis - Beta and Jensen's Alpha Implementation
# Download S&P 500 for market benchmark
sp500 = yf.download("SPY", start="2025-05-16", end="2025-07-20", progress=False)
market_returns = sp500['Close'].pct_change().dropna() * 100
def calculate_beta_and_alpha(portfolio_returns, market_returns, risk_free_rate=0.042):
    """
    Calculate Beta and Jensen's Alpha following CAPM framework
    """
    # Align dates
    common_dates = portfolio_returns.index.intersection(market_returns.index)
    aligned_portfolio = portfolio_returns.loc[common_dates]
    aligned_market = market_returns.loc[common_dates]
    # Beta calculation: β = Cov(Portfolio, Market) / Var(Market)
    covariance = aligned_portfolio.cov(aligned_market)
    market_variance = aligned_market.var()
    beta = covariance / market_variance
    # CAPM Expected Return (Equation 4 components): E(R) = Rf + β(Rm - Rf)
    rf_daily = risk_free_rate / 252
    market_return = aligned_market.mean()
    expected_return = rf_daily + beta * (market_return - rf_daily)
    # Jensen's Alpha (Equation 4): α = Rp - [Rf + βp(Rm - Rf)]
    actual_return = aligned_portfolio.mean()
    jensen_alpha = actual_return - expected_return
    # Sharpe Ratio (Equation 3): (Rp - Rf) / σp
    excess_return = actual_return - rf_daily
    portfolio_volatility = aligned_portfolio.std()
    sharpe_ratio = excess_return / portfolio_volatility
        return {
            'beta': beta,
            'expected_return': expected_return,
            'actual_return': actual_return,
            'jensen_alpha': jensen_alpha,
            'sharpe_ratio': sharpe_ratio
        }
    # Calculate portfolio returns for statistical analysis
    portfolio_returns_p1 = (returns_p1 * weights_p1).sum(axis=1)
    portfolio_returns_p2 = (returns_p2 * weights_p2).sum(axis=1)
    portfolio_returns_p3 = (returns_p3 * weights_p3).sum(axis=1)
    # Statistical analysis for all portfolios
    stats_p1 = calculate_beta_and_alpha(portfolio_returns_p1, market_returns)
    stats_p2 = calculate_beta_and_alpha(portfolio_returns_p2, market_returns)
    stats_p3 = calculate_beta_and_alpha(portfolio_returns_p3, market_returns)
```

Figure 6: Statistical Analysis and Beta Calculation Framework

The statistical framework presented in Figure 6 enables systematic comparison of portfolio performance relative to market benchmarks. Beta calculations follow traditional CAPM methodology, while Jensen's Alpha provides insight into risk-adjusted outperformance across the three portfolio strategies.

7 Data Visualization

The Python code used to generate the visuals figures for portfolio analysis is shown in Figure 7.

`matplotlib` was used for plotting the portfolio allocation pie charts using the `.pie()` method, and also for setting up subplot layouts and titles.

```python
# Portfolio Allocation Pie Charts
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Portfolio allocations
portfolios_data = [
    (portfolio_1_tickers, weights_p1, "Portfolio 1 (Vague)"),
    (portfolio_2_tickers, weights_p2, "Portfolio 2 (Technical)"),
    (portfolio_3_tickers, weights_p3, "Portfolio 3 (MPT)")
]

for i, (tickers, weights, title) in enumerate(portfolios_data):
    axes[i].pie(weights, labels=tickers, autopct='%1.1f%%', startangle=90)
    axes[i].set_title(title)

plt.tight_layout()
plt.show()
```

Figure 7: Python Code for Generating Correlation Matrices and Allocation Pie Charts

Figure 7 displays the complete Python code used to construct the portfolio allocation visuals, allowing the construction of block pie charts that represent the asset weight distribution per portfolio. Both visualizations support the analysis of diversification and asset allocation differences between strategies.

8 Code artefact (Links)

Complete implementation code is available through Google Colab notebooks referenced in the Digital Resources section. Each figure in this document corresponds to executable code cells that generate the documented results.

The complete implementation is available through the following Google Colab notebooks:

- **Portfolio 1 Analysis:** Google Colab Link
- **Portfolio 2 Analysis:** Google Colab Link
- **Portfolio 3 MPT Implementation:** Google Colab Link
- **Portfolio 3 Analysis:** Google Colab Link

9 APPENDIX 1

9.1 Mathematical Framework and Python Implementation

This framework builds the financial formulas from Markowitz (1952) and Sharpe (1966) used for analyzing portfolio performance, risk, and optimization strategies it oulines them with the corresponding Python implementations

The following equations and code snippets detail the implementation of return metrics, risk measures, diversification analysis, and portfolio optimization under Modern Portfolio Theory.

9.1.1 Return Calculations

Daily Returns (Equation 1):

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \times 100$$

Python:

```
daily_rets = stocks.pct_change().dropna()
```

Portfolio Volatility (Equation 2):

$$\sigma_p = \sqrt{\mathbf{w}^T \Sigma \mathbf{w}}$$

Python:

```
port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_rets, weights)))
```

9.1.2 Risk-Adjusted Performance Metrics

Sharpe Ratio (Equation 3):

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

Jensen's Alpha (Equation 4):

$$\alpha = R_p - [R_f + \beta_p(R_m - R_f)]$$

Python:

```
sharpe_ratio = (port_return - risk_free_rate) / port_volatility
jensens_alpha = port_return - (risk_free_rate + beta * (market_return - risk_free_rat
```

9.1.3 Diversification and Correlation Analysis (MPT Framework)

Correlation Coefficient (Equation 5):

$$\rho_{12} = \frac{\text{Cov}(R_1, R_2)}{\sigma_1 \cdot \sigma_2}$$

Python:

```
correlation_matrix = daily_rets.corr()
```

Portfolio Variance (Equation 6):

$$\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}$$

Diversification Benefit (Equation 7):

$$\text{Benefit} = \frac{\sigma_{\text{weighted}} - \sigma_{\text{portfolio}}}{\sigma_{\text{weighted}}} \times 100\%$$

Diversification Score (Equation 8):

$$\text{Score} = (1 - \text{Average Correlation}) \times 100\%$$

Python (for 7–8):

```
weighted_vol = np.sum(weights * daily_rets.std())
div_benefit = (weighted_vol - port_volatility) / weighted_vol * 100
div_score = (1 - correlation_matrix.mean().mean()) * 100
```

9.1.4 Risk Management Metrics

Parametric VaR (Equation 9):

$$\text{VaR}_\alpha = \mu_p - z_\alpha \cdot \sigma_p \cdot \sqrt{t}$$

Historical VaR (Equation 10):

$$\text{VaR}_\alpha = \text{Percentile}(\text{Returns}, \alpha)$$

Beta (Equation 11):

$$\beta = \frac{\text{Cov}(R_p, R_m)}{\text{Var}(R_m)}$$

Python:

```
# Parametric VaR
z_score = norm.ppf(1 - alpha)
parametric_var = port_return - z_score * port_volatility * np.sqrt(t)

# Historical VaR
hist_var = np.percentile(portfolio_returns, 100 * (1 - alpha))

# Beta
beta = np.cov(portfolio_returns, market_returns)[0, 1] / np.var(market_returns)
```

9.1.5 Modern Portfolio Theory Implementation

Minimum Volatility Portfolio (Equation 12):

$$\min \frac{1}{2}\mathbf{w}^T\Sigma\mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^T\mathbf{1} = 1,\ w_i \geq 0\ \forall i$$

Target Return Optimization (Equation 13):

$$\min \frac{1}{2}\mathbf{w}^T\Sigma\mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^T\mu = R_0,\ \mathbf{w}^T\mathbf{1} = 1,\ w_i \geq 0\ \forall i$$

Maximum Sharpe Ratio (Equation 14):

$$\max \frac{\mathbf{w}^T\mu - r_f}{\sqrt{\mathbf{w}^T\Sigma\mathbf{w}}}$$

Sharpe Optimization with Volatility Constraint (Equation 15):

$$\max \frac{\mathbf{w}^T\mu - r_f}{\sqrt{\mathbf{w}^T\Sigma\mathbf{w}}} \quad \text{s.t.} \quad \sqrt{\mathbf{w}^T\Sigma\mathbf{w}} = \sigma_{\text{target}}$$

Python (Optimization via SciPy):

```
from scipy.optimize import minimize

# Objective function for min volatility
def portfolio_volatility(weights, cov_matrix):
```

```
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bounds = tuple((0, 1) for _ in range(len(stocks.columns)))
initial_guess = np.ones(len(stocks.columns)) / len(stocks.columns)

opt_result = minimize(portfolio_volatility, initial_guess, args=(cov_rets,),
                      method='SLSQP', bounds=bounds, constraints=constraints)
optimal_weights = opt_result.x
```

References

Markowitz, H. (1952). Portfolio selection, *The Journal of Finance* **7**(1): 77–91.

Sharpe, W. F. (1966). Mutual fund performance, *The Journal of Business* **39**(1): 119–138.

Yahoo Finance Historical Data (2025). Historical stock price data source.
URL: *https://finance.yahoo.com/*