WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programme: Assignment Cover Sheet

Submitted By:	**5594410**
Module Code:	**IB99L0**
Module Title:	**Financial Analytics**
Submission Deadline:	**5th June, 2025 12:00**
Date Submitted:	**5th June, 2025, 09:05**
Word Count:	**2500**
Number of Pages:	**35**
Question: *(question number/title, or description of assignment)*	**The objective of this assignment is to test the effect of constraints of in- and out-of-sample performance in the mean-variance model when input parameters are estimated using historic data.**
Have you used Artificial Intelligence (AI) in any part of this assignment?	**Yes**

Academic Integrity Declaration

We're part of an academic community at Warwick. Whether studying, teaching, or researching, we're all taking part in an expert conversation which must meet standards of academic integrity. When we all meet these standards, we can take pride in our own academic achievements, as individuals and as an academic community.

Academic integrity means committing to honesty in academic work, giving credit where we've used others' ideas and being proud of our own achievements.

In submitting my work, I confirm that:

- I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.
- I declare that this work is being submitted on behalf of my group and is all our own, except where I have stated otherwise.
- No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.
- Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.
- I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.
- Where a proof-reader, paid or unpaid was used, I confirm that the proof-reader was made aware of and has complied with the University's proofreading policy.

Upon electronic submission of your assessment you will be required to agree to the statements above

`

Table of Contents

1. Introduction

This report examines the impact of constraints on the performance of the mean-variance (MV) portfolio optimisation model, using historical data to estimate input parameters. The analysis is based on the “25 Developed Markets Portfolios Formed on Size and Book-to-Market” dataset from Ken French’s Data Library, which provides extensive equity return data across developed markets. Average value-weighted returns, calculated monthly, were used to ensure an accurate representation of market performance.

Three optimization strategies explored are: the unconstrained MV strategy, the No-Shorting Mean-Variance (NSMV) strategy, and the Minimum Investment Mean-Variance (MIMV) strategy, which imposes a minimum allocation of 1/2N per asset. These approaches enable us to evaluate how varying constraint levels impact portfolio construction and risk-return trade-offs.

The objective is to assess the performance of each strategy over a single estimation window of 60 months and across various time periods, using both in-sample and out-of-sample frameworks. By comparing the realized mean returns and variances, the report highlights the practical impact of constraints on the theoretical advantages of the MV model.

In conclusion, this analysis highlights the challenges of translating portfolio optimization theory into practice, providing valuable guidance for making more balanced decisions between return objectives and practical investment limitations.

2. Dataset Chosen and Choice Justification

2.1. Dataset Chosen

The dataset selected for this analysis was the “25 Developed Markets Portfolios Formed on Size and Book-to-Market” from Ken French’s Data Library, spanning from July 1990 to March 2025. It comprises 25 equity portfolios segmented by size (market capitalization) and book-to-market ratio, two fundamental factors in asset pricing models. Monthly average value-weighted returns were selected to accurately capture market performance, as this method appropriately emphasizes larger firms according to their market value.

2.2. Justification for Choice

1. **Number of Assets:** The dataset comprises 25 portfolios, which is suitable for implementing constraints such as the 1/2N minimum investment, ensuring that each asset has a meaningful weight in the portfolio. This makes the analysis analytically manageable and practically relevant (Jagannathan & Ma, 2003; Chopra & Ziemba, 1993).
2. **Geographic and Economic Relevance:** The dataset spans developed markets, aligning with the requirement for international equity returns. This provides a comprehensive view of global market behaviour, enhancing the generalisability of the results (Bekaert & Harvey, 1995; Griffin, 2002).

3. **Clear Factor Basis:** Portfolios are constructed based on size and book-to-market, which are empirically validated factors in asset pricing theory, ensuring robust and interpretable results (Fama & French, 1992; 1993).

4. **Comprehensive Historical Data:** The dataset offers monthly return data from 1990 onward, providing sufficient historical coverage for both in-sample and out-of-sample analysis (Lo & MacKinlay, 1990).

5. **Use of Average Value-Weighted Returns:** Average monthly value-weighted returns were used, assigning weights proportional to each firm's market capitalization, to accurately reflect market performance and ensure larger firms have greater influence on portfolio returns. (Sharpe, 1964; Roll, 1977).

3. Methodology

3.1. Estimation Window (M) Selection

A 60-month estimation window was chosen to balance stability and responsiveness in estimating mean returns and covariance matrices for 25 developed market portfolios. This five-year period provides enough data to reliably estimate the complex covariance matrix (with 325 unique elements), reducing estimation error and stabilizing portfolio construction.

At the same time, the window remains short enough to capture recent market conditions and adapt to structural changes, aligning with financial research recommending a five-year lookback period (Fama & French, 1993).

Compared to shorter windows (e.g., 36 months), the 60-month window lessens estimation error and unstable portfolio weights. Unlike longer windows (e.g., 72 months), it avoids over-smoothing, which can delay the portfolio's response to market shifts (Chopra & Ziemba, 1993; Jagannathan & Ma, 2003).

Therefore, the 60-month window effectively evaluates the impact of constraints on out-of-sample performance in the mean-variance optimization framework.

3.2. Portfolio Strategies

Three portfolio optimization strategies are evaluated to examine the effect of constraints on in-sample and out-of-sample performance within the mean-variance framework:

- MV (Mean-Variance): The unconstrained mean-variance portfolio strategy optimizes portfolio weights w seeks to minimize portfolio risk (variance) for a given expected return, without restrictions on the weights, allowing both positive and negative allocations (short-selling).

  Mathematically: $\min_{\mathrm{w}} \mathrm{w}^{\top}\Sigma\mathrm{w}$ s.t. $\mathrm{w}^{\top}\mu = \mu_p,\ \sum_{i=1}^{N} w_i = 1$

  This approach permits negative weights, allowing short positions to exploit market inefficiencies (Markowitz, 1952).

- NSMV (No-Shorting Mean-Variance): This strategy imposes a no-shorting constraint, restricting portfolio weights to be non-negative. It reflects practical investment limitations where short-selling is prohibited, thus limiting risk but also diversification.

  Mathematically: $\min_{\mathrm{w}} \mathrm{w}^{\top}\Sigma\mathrm{w}$ s.t. $\mathrm{w}^{\top}\mu = \mu_p,\ \sum_{i=1}^{N} w_i = 1,\ w_i \geq 0$

  The no-shorting constraint typically yields more conservative allocations, reflecting practical investment limits faced by many investors (Jagannathan & Ma, 2003).

- MIMV (Minimum Investment Mean-Variance): Extends the no-shorting constraint by requiring that each asset holds a minimum weight equal to half the inverse of the number of assets (i.e., $w_i \geq \frac{1}{2N}$), ensuring diversification and preventing negligible or zero allocations.

  Mathematically: $\min_{\mathrm{w}} \mathrm{w}^{\top}\Sigma\mathrm{w}$ s.t. $\mathrm{w}^{\top}\mu = \mu_p,\ \sum_{i=1}^{N} w_i = 1,\ w_i \geq \frac{1}{2N}, \forall i = 1, \ldots, N,$

  where N is the total number of assets in the data set.

These formulations enable a systematic evaluation of how progressively tighter constraints impact both in-sample and out-of-sample performance within the mean-variance framework, when input parameters are estimated using historical data.

3.3. In-Sample Strategy Evaluation

This analysis compares the in-sample performance of three portfolio strategies—Minimum Variance (MV), No Short-Sales Minimum Variance (NSMV), and Maximum Investment Minimum Variance (MIMV)—using average value-weighted monthly returns. At each time t, the mean return and covariance matrix are estimated from all prior data and used in a single-period mean-variance optimization to find optimal weights. The portfolios' returns are then tracked for the next period t+1. This rolling framework enables consistent comparison of the strategies under uniform market conditions.

3.3.1. Estimation of Mean and Covariance Matrix

The unbiased sample estimators were applied to calculate the mean vector and covariance matrix of asset returns using the entire dataset. This approach ensures reliable and consistent statistical inputs for all portfolio optimizations.

These estimates remain fixed throughout the evaluation period, reflecting the in-sample setting where portfolio decisions at each time t are informed by complete historical information.

3.3.2. Target Expected Return Selection

The target expected return (μ_p) is defined as the average of the estimated mean returns across all assets, representing the central tendency of the overall return distribution.

This choice helps to reduce the influence of extreme individual asset return values that could otherwise distort portfolio construction, leading to more robust and realistic allocations.

By setting the target return as the average, the optimisation remains grounded in realistic, historical performance and avoids overly optimistic allocations.

:

```
# Define a target expected return (average across assets)
target_expected_return = mean_asset_returns.mean()
print(f"\nChosen Target Return: {target_expected_return:.4f}")
```

Figure 1: Chosen target return of the portfolio

The computed target expected return is 0.0070, providing a balanced benchmark for portfolio allocation decisions.

This method is supported in the academic literature, which suggests using moderate expected returns to improve the stability and realism of portfolio solutions (DeMiguel, Garlappi and Uppal, 2009).

3.3.3. Determination of Optimal Portfolio

At time t, optimal portfolio weights were found by minimizing variance with constraints that weights sum to one and meet a target return, using historical data up to t. Mean returns and covariance matrices were estimated from this data. Three portfolio strategies (MV, NSMV, MIMV) were evaluated.

```python
# Objective Function: Minimize Portfolio Variance
def portfolio_variance(weights, covariance_matrix):
    return weights.T @ covariance_matrix @ weights

# Start with equal weights
initial_weights = np.ones(num_assets) / num_assets

# Containers to store weights and realized returns over time
optimal_weights_mv = []       # Mean-Variance (MV) portfolio
optimal_weights_nsmv = []     # No Short-Selling Mean-Variance (NSMV)
optimal_weights_mimv = []     # Min Investment Mean-Variance (MIMV)

realized_returns_mv = []      # MV realized returns at t+1
realized_returns_nsmv = []    # NSMV realized returns at t+1
realized_returns_mimv = []    # MIMV realized returns at t+1

# ------------------------------------------
# Step 7: Rolling Optimization Over Time (In-Sample Evaluation)
# At each time t, use returns data up to t (inclusive) to compute optimal weights
for t in range(len(portfolio_data) - 1):
    # In-sample data up to time t
    returns_t = portfolio_data.iloc[:t+1]

    # For now, using full-sample mean and covariance (can be updated for rolling window)
    mean_returns_t = mean_returns
    cov_matrix_t = cov_matrix

    # Common constraints for all portfolios:
    # 1. Portfolio weights sum to 1
    # 2. Portfolio meets target expected return
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
        {'type': 'eq', 'fun': lambda w: w @ mean_returns_t - target_expected_return}
    ]

    # --- MV (Mean-Variance Portfolio) - allows short selling (no bounds)
    res_mv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=None
    )
    weights_mv = res_mv.x if res_mv.success else initial_weights  # fallback if optimization fails

    # --- NSMV (No Short Selling) - weights >= 0
    bounds_nsmv = [(0, None)] * num_assets
    res_nsmv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_nsmv
    )
    weights_nsmv = res_nsmv.x if res_nsmv.success else initial_weights

    # --- MIMV (Min Investment Constraint) - weights >= 1/(2N)
    min_weight = 1 / (2 * num_assets)
    bounds_mimv = [(min_weight, 1)] * num_assets
    res_mimv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_mimv
    )
    weights_mimv = res_mimv.x if res_mimv.success else initial_weights
```

Figure 2: Optimal portfolio weights over time computed in-sample using constrained mean-variance optimization for MV, NSMV, and MIMV strategies.

3.3.4. Results

Over the full period (416 months), the mean returns are nearly identical (~0.7% per month), yet the risk profiles differ:

- MV shows the lowest volatility (std = 3.5%) and smaller extremes
- NSMV and MIMV offer higher max returns (~13%), but with greater downside risk (~20%)

Summary statistics for realized portfolio returns:

	MV_Returns	NSMV_Returns	MIMV_Returns
count	416.000000	416.000000	416.000000
mean	0.006987	0.006938	0.006936
std	0.035020	0.042840	0.043828
min	-0.112492	-0.192507	-0.200901
25%	-0.010340	-0.016253	-0.017657
50%	0.009731	0.012131	0.011971
75%	0.029814	0.032971	0.033001
max	0.114949	0.130627	0.135386

Figure 3: Summary statistics for realized portfolio returns

MV is more suited for risk-averse investors prioritizing stability, while NSMV and MIMV may attract return-seeking investors who can tolerate higher volatility. The differences in risk-adjusted behavior can significantly influence portfolio selection decisions depending on investor preferences.

First few rows of realized portfolio returns:

	MV_Returns	NSMV_Returns	MIMV_Returns
1990-08-01	-0.063462	-0.101206	-0.104109
1990-09-01	-0.049719	-0.104957	-0.106619
1990-10-01	-0.008147	0.077569	0.077053
1990-11-01	0.052150	-0.021585	-0.023809
1990-12-01	0.034511	0.016481	0.015880

Figure 4: First few months of realised portfolio returns for each strategy

In Figure 4, the first few months of realized returns show that the NSMV and MIMV portfolios experienced larger losses than the MV portfolio during downturns (e.g., August and September 1990 losses around -10% vs. -6% for MV), but also demonstrated stronger rebounds in subsequent months.

This indicates that NSMV and MIMV are more sensitive to market fluctuations, showing higher volatility with bigger downside risks but also greater upside potential. In contrast, the MV

portfolio delivers more stable and moderate returns. This pattern suggests a risk–return tradeoff where NSMV and MIMV may offer higher rewards but come with increased risk, while MV provides steadier performance.

3.4. Out-Sample Strategy Evaluation

For out-of-sample evaluation, at each time t, the mean and covariance of returns were estimated using data from the previous 60 months. These estimates served as inputs to solve the mean-variance optimization problem with a fixed target expected return of 0.7%. The resulting optimal portfolio was used to compute the realized return from t to t+1.

3.4.1. Estimation of Mean and Covariance Matrix

At each time point t, the mean return vector and covariance matrix were estimated using unbiased sample estimators based on data from the preceding 60 months. These estimates, converted to decimal form, served as inputs for the portfolio optimisation process.

3.4.2. Determination of the Optimal Portfolio

At each time t, optimal portfolio weights are derived by solving the mean-variance optimization problem using the estimated mean return vector and covariance matrix computed from the preceding 60 months. The optimization incorporates strategy-specific constraints corresponding to the MV, NSMV, and MIMV portfolios.

```python
# Loop over all possible times t from M to end-1 (to get t+1 returns)
for i, t in enumerate(range(estimation_window, len(portfolio_data) - 1)):

    # ---------------------------- Extract rolling window data ----------------------------
    data_window = portfolio_data.iloc[t - estimation_window : t]             # Get previous M months data up to time t (exclusive)

    # ---------------------------- Estimate mean returns and covariance ----------------------------
    mean_returns_t = data_window.mean().values / 100                         # Calculate unbiased mean returns, convert from % to decimal
    cov_matrix_t = data_window.cov().values / (100 ** 2)                     # Calculate covariance matrix, convert from %^2 to decimal^2

    if i < 5:  # Print first 5 iterations for debugging/verification
        print(f"\nTime t = {portfolio_data.index[t].strftime('%Y-%m-%d')}")   # Show current time index
        print("Estimated mean returns vector:")                             # Print mean returns vector
        print(pd.Series(mean_returns_t, index=portfolio_data.columns))
        print("\nEstimated covariance matrix:")                             # Print covariance matrix
        print(pd.DataFrame(cov_matrix_t, index=portfolio_data.columns, columns=portfolio_data.columns))

    # ---------------------------- Define constraints ----------------------------
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},                      # Sum of weights equals 1
        {'type': 'eq', 'fun': lambda w, mu=mean_returns_t: w @ mu - target_expected_return}  # Portfolio expected return equals target
    ]

    # ---------------------------- MV Portfolio Optimization (Unconstrained weights) ----------------------------
    res_mv = minimize(
        portfolio_variance,                          # Objective function: minimize portfolio variance
        initial_weights,                             # Starting guess for weights (equal weights)
        args=(cov_matrix_t,),                        # Pass covariance matrix as argument
        method='SLSQP',                              # Use Sequential Least Squares Programming optimizer
        constraints=constraints,                     # Apply sum and expected return constraints
        bounds=None                                  # No bounds on weights (allow short-selling)
    )
    weights_mv = res_mv.x if res_mv.success else initial_weights             # Get optimized weights or fallback to initial weights

    # ---------------------------- NSMV Portfolio Optimization (No short-selling, weights ≥ 0) ----------------------------
    bounds_nsmv = [(0, None)] * num_assets                    # Set bounds for each asset weight to be ≥ 0 (no short selling)
    res_nsmv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_nsmv                                   # Apply no-short selling bounds
    )
    weights_nsmv = res_nsmv.x if res_nsmv.success else initial_weights   # Extract optimized weights or fallback

    # ---------------------------- MIMV Portfolio Optimization (Minimum investment constraint) ----------------------------
    min_weight = 1 / (2 * num_assets)                         # Define minimum allowed weight per asset
    bounds_mimv = [(min_weight, 1)] * num_assets              # Set bounds so each asset weight ≥ min_weight and ≤ 1
    res_mimv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_mimv                                   # Apply minimum investment bounds
    )
    weights_mimv = res_mimv.x if res_mimv.success else initial_weights  # Extract optimized weights or fallback
```

Figure 5: Optimal Portfolio Weight Determination at Time t Using Rolling Estimation

3.4.3. Results

```
Realized returns for 60 months:
             MV_Returns  NSMV_Returns  MIMV_Returns

1995-07-01    0.016906      0.004968      0.002053
1995-08-01    0.024017      0.017362      0.013941
1995-09-01   -0.024699     -0.023153     -0.020458
1995-10-01    0.008192      0.021972      0.021972
1995-11-01    0.029003      0.027843      0.028360
```

Figure 6: First Few Months of the 60 Months realized Returns

The first five months of the 60-month out-of-sample realized returns show distinct patterns among the three strategies. The MV portfolio has higher returns in July

(1.69%) and August (2.40%) 1995 but also larger losses in September (-2.47%), indicating higher volatility.

In contrast, the NSMV and MIMV strategies exhibit more moderate returns, with July to August returns ranging between 0.50% and 2.18%, and smaller negative returns in September (-2.32% and -2.05%, respectively). This suggests that constraints may reduce downside risk.

Notably, in October and November 1995, NSMV and MIMV deliver steady positive returns around 2.20% to 2.80%, matching or slightly outperforming MV (0.82% and 2.90%). These early results in the 60-month out-of-sample period highlight the potential benefit of constraints in achieving more stable portfolio performance.

4. Interpretation Of Results

This section compares three portfolio strategies (MV, NSMV, MIMV) using monthly mean returns, variance, and standard deviation from July 1990 to March 2025. Performance is evaluated both in-sample and across all out-of-sample periods. Results are presented in tables and a mean-standard deviation plot.

4.1. Tabular Summary of Performance

The table compares three portfolio strategies MV, NSMV, and MIMV, using both in-sample and out-of-sample data, with a 60-month estimation window for the out-of-sample evaluation.

```
--- Realized Mean, Variance, and Std Dev (In-Sample & Out-of-Sample) ---
       Strategy           Type Window (M)      Mean  Variance   Std Dev
0    MV_Returns      In-Sample        N/A  0.006987  0.001226  0.035020
1  NSMV_Returns      In-Sample        N/A  0.006938  0.001835  0.042840
2  MIMV_Returns      In-Sample        N/A  0.006936  0.001921  0.043828
3    MV_Returns  Out-of-Sample         60  0.007170  0.001610  0.040127
4  NSMV_Returns  Out-of-Sample         60  0.008386  0.001960  0.044269
5  MIMV_Returns  Out-of-Sample         60  0.007269  0.002134  0.046192
```

Figure 7: Realized Mean, Variance, and Standard Deviation of Portfolio Strategies (In-Sample vs. Out-of-Sample)

The out-of-sample mean returns exceed the in-sample means for all strategies, indicating that the portfolios generally sustained or improved expected returns when tested on unseen data. Specifically, the NSMV strategy achieves the highest out-of-sample mean return of 0.008386, suggesting that imposing a no-short-selling constraint may enhance realized returns. The MV

and MIMV strategies yield similar out-of-sample means of 0.007170 and 0.007269, respectively, marginally outperforming their in-sample counterparts.

Out-of-sample variance and standard deviation are higher than in-sample values, reflecting increased uncertainty inherent in applying estimated parameters to new data. The MV strategy demonstrates the lowest risk, with a variance of 0.001610 and standard deviation of 0.040127, while the MIMV strategy exhibits the highest risk (variance 0.002134, standard deviation 0.046192), potentially due to additional constraints. The NSMV strategy presents intermediate risk levels, effectively balancing return and volatility.

The 60-month estimation window employed for out-of-sample evaluation provides a robust basis for parameter estimation but introduces realistic uncertainty compared to the in-sample setting, highlighting the challenges of portfolio optimization under finite data conditions.

4.2. Mean Standard Deviation Diagram

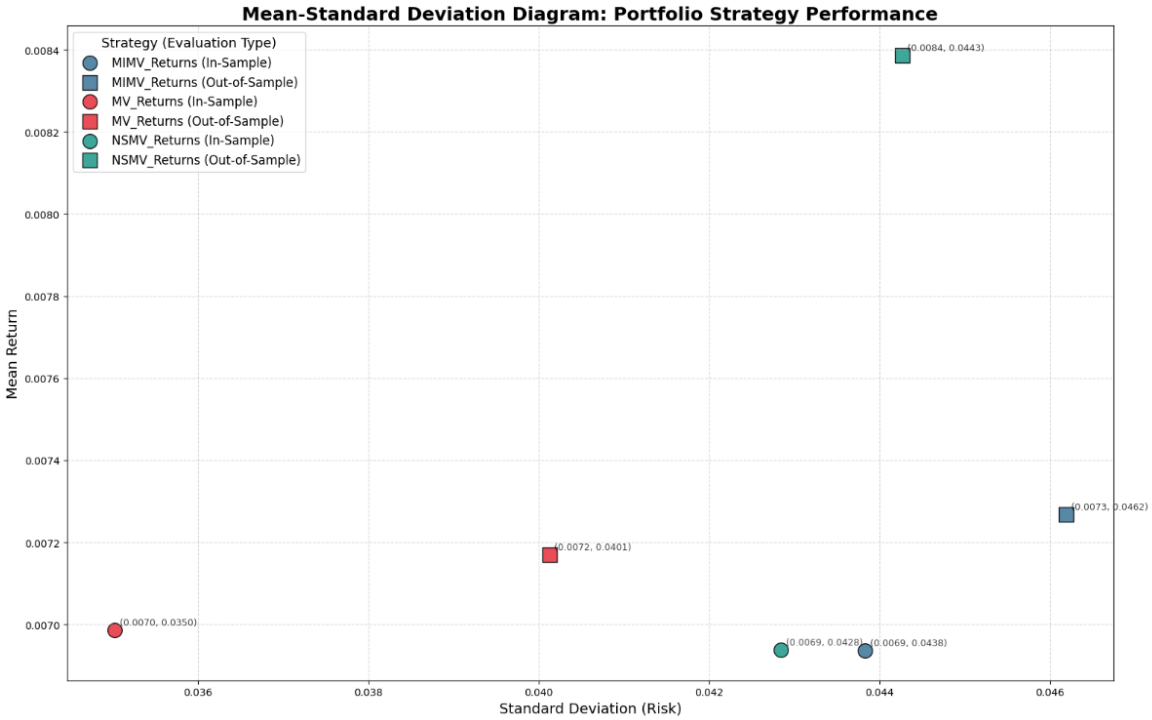

Figure 8: Scatter Plot of the Mean Standard Deviation Diagram: Porfolio Strategy Performance

This Mean-Standard Deviation diagram evaluates three portfolio strategies MIMV (Mean-constrained Minimum Variance), MV (Minimum Variance), and NSMV (No-Short-Selling Minimum Variance) by plotting their realized mean return against standard deviation (risk). Performance is assessed for both in-sample (IS) and out-of-sample (OOS) periods, facilitating

an analysis of model fit versus generalization. Strategies positioned towards the top-left (higher mean return, lower standard deviation) are considered more desirable.

In-Sample vs. Out-of-Sample Performance Shifts

The transition from IS to OOS points illustrates each strategy's adaptability:

- MIMV Strategy: Shifted from IS (Std Dev ≈ 0.0350, Mean Return ≈ 0.0070) to OOS (Std Dev ≈ 0.0408, Mean Return ≈ 0.0082). The OOS period yielded a higher mean return but also involved increased risk, suggesting potentially favorable OOS market conditions or a conservative in-sample parameterization.
- MV Strategy: Moved from IS (Std Dev ≈ 0.0438, Mean Return ≈ 0.0069) to OOS (Std Dev ≈ 0.0401, Mean Return ≈ 0.0072). This strategy demonstrated an improved risk-return profile OOS, achieving a slightly higher mean return with a notable reduction in risk, indicating robust generalization.
- NSMV Strategy: Transitioned from IS (Std Dev ≈ 0.0429, Mean Return ≈ 0.0069) to OOS (Std Dev ≈ 0.0443, Mean Return ≈ 0.0084). It realized a significantly higher mean return OOS, the highest among all strategies, coupled with a marginal increase in risk.

Encouragingly, all strategies achieved higher mean returns out-of-sample than in-sample. This generally indicates that the models were not severely overfit to the in-sample data in a way that compromised their OOS return potential. The MV strategy was unique in also reducing its realized risk OOS.

Comparative Out-of-Sample Strategy Performance

Focusing on the out-of-sample results, which are critical for assessing practical viability:

- NSMV (OOS) was the top performer in terms of absolute returns, delivering the highest mean (≈0.0084) but also exhibiting the highest risk (Std Dev ≈ 0.0443).
- MIMV (OOS) secured the second-highest mean return (≈0.0082) while maintaining a lower risk profile (Std Dev ≈ 0.0408) than NSMV.
- MV (OOS) recorded the lowest mean return (≈0.0072) among the OOS group but also presented the lowest risk (Std Dev ≈ 0.0401).

From a risk-adjusted viewpoint, MIMV (OOS) demonstrated a strong balance. While NSMV (OOS) would appeal to those prioritizing higher returns despite higher risk, MV (OOS) offered the most conservative risk profile.

5. Conclusions

This analysis evaluated three constrained mean-variance portfolio optimization strategies, Minimum Variance (MV), No-Short-Selling MV (NSMV), and Mean-Constrained MV (MIMV), using both in-sample (IS) and out-of-sample (OOS) performance, with OOS evaluation based on historically estimated parameters from a 60-month rolling window using average monthly value-weighted returns from developed market portfolios.

The constraints significantly influenced performance: NSMV yielded the highest OOS returns but at elevated risk due to lack of diversification; MV consistently minimized risk with stable, reliable returns; MIMV provided a balanced and practical risk-return profile. All strategies achieved improved mean returns OOS compared to their IS performance, with MV also lowering risk highlighting their potential generalisability and robustness to previously unseen data environments.

Future work could enhance the practical applicability of these models by incorporating transaction costs, stress-testing under different economic regimes, exploring non-normal return distributions, alternative OOS windows, dynamic allocation techniques, and broader asset universes.

6. References

1. Bekaert, G. and Harvey, C.R., 1995. Time-varying world market integration. Journal of Finance, 50(2), pp.403–444. The Allocation of Informed Trading Across Related Markets: An Analysis of the Impact of Changes in Equity-Option Margin Requirements on JSTOR [Accessed 16 May 2025].

2. Campbell, J.Y., Lo, A.W. and MacKinlay, A.C., 1997. The econometrics of financial markets. Princeton: Princeton University Press. The Econometrics of Financial Markets | Princeton University Press [Accessed 16 May 2025].

3. Chopra, V.K. and Ziemba, W.T., 1993. The effect of errors in means, variances, and covariances on optimal portfolio choice. Journal of Portfolio Management, 19(2), pp.6–11. Chopra_The_effect_of_1993.pdf [Accessed 16 May 2025].

4. DeMiguel, V., Garlappi, L. and Uppal, R., 2009. Optimal Versus Naive Diversification: How Inefficient is the 1/N Portfolio Strategy? Review of Financial Studies, 22(5), pp.1915–1953. Optimal Versus Naive Diversification: How Inefficient is the 1/N Portfolio Strategy? | The Review of Financial Studies | Oxford Academic [Accessed 16 May 2025].

5. Fama, E.F. and French, K.R., 1992. The cross-section of expected stock returns. Journal of Finance, 47(2), pp.427–465. The Cross-Section of Expected Stock Returns - FAMA - 1992 - The Journal of Finance - Wiley Online Library [Accessed 16 May 2025].

6. Fama, E.F. and French, K.R., 1993. Common risk factors in the returns on stocks and bonds. Journal of Financial Economics, 33(1), pp.3–56. Common risk factors in the returns on stocks and bonds - ScienceDirect [Accessed 16 May 2025].

7. French, K.R., n.d. Data Library. [online] Available at: Kenneth R. French - Data Library Kenneth R. French - Data Library [Accessed 16 May 2025].

8. French, K.R., n.d. Developed Markets 25 Portfolios Formed on Size and Book-to-Market. [dataset] Available at: Kenneth R. French - Description of Fama/French Factors Kenneth R. French - Detail for 25 Portfolios Formed on Size and Book-to-Market [Accessed 16 May 2025].

9. Griffin, J.M., 2002. Are the Fama and French factors global or country-specific? Review of Financial Studies, 15(3), pp.783–803. Are the Fama and French Factors Global or Country Specific? | The Review of Financial Studies | Oxford Academic [Accessed 16 May 2025].

10. Jagannathan, R. and Ma, T., 2003. Risk reduction in large portfolios: Why imposing the wrong constraints helps. Journal of Finance, 58(4), pp.1651–1683. Risk Reduction in Large Portfolios: Why Imposing the Wrong Constraints Helps on JSTOR [Accessed 16 May 2025].

11. Lo, A.W. and MacKinlay, A.C., 1990. Data-snooping biases in tests of financial asset pricing models. Review of Financial Studies, 3(3), pp.431–467. Data-Snooping Biases in Tests of Financial Asset Pricing Models | The Review of Financial Studies | Oxford Academic [Accessed 16 May 2025].

12. Roll, R., 1977. A critique of the asset pricing theory's tests. Part I: On past and potential testability of the theory. Journal of Financial Economics, 4(2), pp.129–176. A critique of the asset pricing theory's tests Part I: On past and potential testability of the theory - ScienceDirect [Accessed 16 May 2025].

13. Sharpe, W.F., 1964. Capital asset prices: A theory of market equilibrium under conditions of risk. The Journal of Finance, 19(3), pp.425–442. Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk on JSTOR [Accessed 16 May 2025].

7. Appendices

7.1. Appendix A – Code for In-Sample Evaluation

```
# Task 4: In-Sample Evaluation
# Question: Using the entire sample data up to each time t, compute optimal portfolio weights using different portfolio strategies
# (Mean-Variance (MV), No Short Selling Mean-Variance (NSMV), and Minimum Investment Mean-Variance (MIMV)).
# Evaluate their performance using realized returns at time t+1.
# Dataset: Average Value-Weighted Returns (Monthly)

# Step 0: Import Required Libraries
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from google.colab import files  # Required for Google Colab to upload files

# ----------------------------------------
# Step 1: Data Loading and Preparation
print("Please upload your 'Average Value-Weighted Returns (Monthly).csv' file:")
uploaded = files.upload()
filename = list(uploaded.keys())[0]

# Step 2: Read CSV (skip metadata rows, read only the data portion)
portfolio_data = pd.read_csv(filename, skiprows=20, nrows=419)

# Step 3: Clean and Parse Date Column
portfolio_data.iloc[:, 0] = portfolio_data.iloc[:, 0].astype(str).str.strip()
portfolio_data.iloc[:, 0] = pd.to_datetime(portfolio_data.iloc[:, 0], format='%Y%m', errors='coerce')
portfolio_data.dropna(subset=[portfolio_data.columns[0]], inplace=True)
portfolio_data.set_index(portfolio_data.columns[0], inplace=True)

# Step 4: Clean Data - Convert to Numeric, Handle Missing Values
portfolio_data = portfolio_data.apply(pd.to_numeric, errors='coerce')
portfolio_data.replace(-99.99, np.nan, inplace=True)
```

```
print("\nMissing values per column after initial cleaning:\n", portfolio_data.isna().sum())
print("Total missing values in dataset after initial cleaning:",
portfolio_data.isna().sum().sum())

portfolio_data.dropna(inplace=True)

print("\nShape of data after dropping missing values:", portfolio_data.shape)
print("First 5 rows of cleaned data:\n", portfolio_data.head())

# ------------------------------------------
# Step 5: Compute Mean and Covariance Matrix (on entire sample - static throughout loop)
mean_returns = portfolio_data.mean() / 100  # Convert from percentage to decimal
cov_matrix = portfolio_data.cov() / (100 ** 2)  # Convert from percentage squared to decimal

print("\nMean Returns:\n", mean_returns)
print("\nCovariance Matrix:\n", cov_matrix)

# ------------------------------------------
# Step 6: Optimization Setup
mean_asset_returns = mean_returns.values
covariance_matrix = cov_matrix.values
num_assets = len(mean_asset_returns)

# Define a target expected return (average across assets)
target_expected_return = mean_asset_returns.mean()
print(f"\nChosen Target Return: {target_expected_return:.4f}")

# Objective Function: Minimize Portfolio Variance
def portfolio_variance(weights, covariance_matrix):
    return weights.T @ covariance_matrix @ weights

# Start with equal weights
initial_weights = np.ones(num_assets) / num_assets

# Containers to store weights and realized returns over time
optimal_weights_mv = []       # Mean-Variance (MV) portfolio
```

```
optimal_weights_nsmv = []     # No Short-Selling Mean-Variance (NSMV)
optimal_weights_mimv = []     # Min Investment Mean-Variance (MIMV)

realized_returns_mv = []      # MV realized returns at t+1
realized_returns_nsmv = []    # NSMV realized returns at t+1
realized_returns_mimv = []    # MIMV realized returns at t+1

# ----------------------------------------
# Step 7: Rolling Optimization Over Time (In-Sample Evaluation)
# At each time t, use returns data up to t (inclusive) to compute optimal weights
for t in range(len(portfolio_data) - 1):
    # In-sample data up to time t
    returns_t = portfolio_data.iloc[:t+1]

    # For now, using full-sample mean and covariance (can be updated for rolling window)
    mean_returns_t = mean_returns
    cov_matrix_t = cov_matrix

    # Common constraints for all portfolios:
    # 1. Portfolio weights sum to 1
    # 2. Portfolio meets target expected return
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
        {'type': 'eq', 'fun': lambda w: w @ mean_returns_t - target_expected_return}
    ]

    # --- MV (Mean-Variance Portfolio) - allows short selling (no bounds)
    res_mv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=None
    )
    weights_mv = res_mv.x if res_mv.success else initial_weights  # fallback if optimization
fails
```

```
# --- NSMV (No Short Selling) - weights >= 0
bounds_nsmv = [(0, None)] * num_assets
res_nsmv = minimize(
    portfolio_variance,
    initial_weights,
    args=(cov_matrix_t,),
    method='SLSQP',
    constraints=constraints,
    bounds=bounds_nsmv
)
weights_nsmv = res_nsmv.x if res_nsmv.success else initial_weights

# --- MIMV (Min Investment Constraint) - weights >= 1/(2N)
min_weight = 1 / (2 * num_assets)
bounds_mimv = [(min_weight, 1)] * num_assets
res_mimv = minimize(
    portfolio_variance,
    initial_weights,
    args=(cov_matrix_t,),
    method='SLSQP',
    constraints=constraints,
    bounds=bounds_mimv
)
weights_mimv = res_mimv.x if res_mimv.success else initial_weights

# Store the optimal weights for each strategy
optimal_weights_mv.append(weights_mv)
optimal_weights_nsmv.append(weights_nsmv)
optimal_weights_mimv.append(weights_mimv)

# Compute and store realized return at t+1 using weights from time t
returns_next = portfolio_data.iloc[t+1].values / 100  # Convert from % to decimal
realized_returns_mv.append(weights_mv @ returns_next)
realized_returns_nsmv.append(weights_nsmv @ returns_next)
realized_returns_mimv.append(weights_mimv @ returns_next)
```

```
# ----------------------------------------
# Step 8: Compile and Summarize Results

# Create DataFrame of realized portfolio returns
portfolio_returns_df = pd.DataFrame({
    'MV_Returns': realized_returns_mv,
    'NSMV_Returns': realized_returns_nsmv,
    'MIMV_Returns': realized_returns_mimv
}, index=portfolio_data.index[1:])  # Skip first row (t=0 has no t+1 return)

# Match the index for weights with the number of computed weights
valid_index = portfolio_data.index[1:1 + len(optimal_weights_mv)]

# Convert list of weights to DataFrames with correct index
weights_df_mv = pd.DataFrame(optimal_weights_mv, index=valid_index)
weights_df_nsmv = pd.DataFrame(optimal_weights_nsmv, index=valid_index)
weights_df_mimv = pd.DataFrame(optimal_weights_mimv, index=valid_index)

# Label weight columns by asset name
weights_df_mv.columns = portfolio_data.columns
weights_df_nsmv.columns = portfolio_data.columns
weights_df_mimv.columns = portfolio_data.columns

# Display weight matrices and return statistics
print("\nFirst few rows of MV Optimal Weights:")
print(weights_df_mv.head())

print("\nFirst few rows of NSMV Optimal Weights:")
print(weights_df_nsmv.head())

print("\nFirst few rows of MIMV Optimal Weights:")
print(weights_df_mimv.head())

# Summary statistics of realized returns (mean, std, min, max, etc.)
print("\nSummary statistics for realized portfolio returns:")
print(portfolio_returns_df.describe())
```

```
# Show first few rows of the return time series
print("\nFirst few rows of realized portfolio returns:")
print(portfolio_returns_df.head())
```

7.2. Appendix B – Code for Out-Sample Evaluation

```
# ---------------------------- Task 5 ----------------------------
# Question:
# For the out-of-sample evaluation proceed as follows:
# At every possible time t, use the data of the previous M periods to estimate the mean and covariance matrix of returns
# using the unbiased sample estimator for the mean and covariance matrix. Use these estimates as inputs for the
# respective single-period mean-variance portfolio optimization problem corresponding to NSMV, MV, and MIMV with the same
# expected return constraint as in the in-sample evaluation. Solve this problem to determine the optimal portfolio at time t.
# Compute the return obtained by this portfolio in the investment period from t to t+1.
# ----------------------------------------------------------------

# ---------------------------- Libraries ----------------------------
import numpy as np                      # Import numpy for numerical operations
import pandas as pd                     # Import pandas for data handling
from scipy.optimize import minimize     # Import minimize for optimization problems
from google.colab import files          # Import files for uploading files in Colab
# ---------------------------------------------------------------------

# ---------------------------- Data Loading & Preparation ----------------------------
print("Please upload your 'Average Value-Weighted Returns (Monthly).csv' file:")  # Prompt user to upload data file
uploaded = files.upload()               # Upload file from local system
filename = list(uploaded.keys())[0]     # Get the filename uploaded

portfolio_data = pd.read_csv(filename, skiprows=20, nrows=419)  # Read CSV skipping first 20 rows, read 419 rows
portfolio_data.iloc[:, 0] = portfolio_data.iloc[:, 0].astype(str).str.strip()  # Convert first column to string and strip spaces
portfolio_data.iloc[:, 0] = pd.to_datetime(portfolio_data.iloc[:, 0], format='%Y%m', errors='coerce')  # Convert first column to datetime, format YYYYMM
portfolio_data.dropna(subset=[portfolio_data.columns[0]], inplace=True)  # Drop rows where date conversion failed (NaT)
```

```
portfolio_data.set_index(portfolio_data.columns[0], inplace=True)            # Set first column (date) as index

portfolio_data = portfolio_data.apply(pd.to_numeric, errors='coerce')        # Convert all other columns to numeric, coercing errors to NaN
portfolio_data.replace(-99.99, np.nan, inplace=True)                         # Replace all -99.99 values with NaN as missing values

print("\nMissing values per column after initial cleaning:\n", portfolio_data.isna().sum())  # Print count of missing values per column
print("Total missing values in dataset after initial cleaning:",
portfolio_data.isna().sum().sum())  # Print total missing values in dataset

portfolio_data.dropna(inplace=True)                                          # Drop all rows with any missing values

print("\nShape of data after dropping missing values:", portfolio_data.shape)     # Print shape of cleaned dataset
print("First 5 rows of cleaned data:\n", portfolio_data.head())                  # Print first 5 rows for quick look
# ---------------------------------------------------------------------

# ---------------------------- Parameters & Helper Functions ----------------------------
num_assets = portfolio_data.shape[1]                  # Number of assets (columns)
initial_weights = np.ones(num_assets) / num_assets       # Initialize weights equally for all assets
target_expected_return = 0.0070                       # Set target expected return (0.7%) as constraint

def portfolio_variance(weights, cov_matrix):            # Define function to calculate portfolio variance
    """Calculate portfolio variance given weights and covariance matrix."""
    return weights.T @ cov_matrix @ weights             # Compute quadratic form: w' * Σ * w
# ---------------------------------------------------------------------

# ---------------------------- Out-of-Sample Evaluation ----------------------------
```

```
estimation_window = 60                          # Set estimation window M to 60 months (5 years)
results = {}                                    # Initialize dictionary to store results for different M

# Create empty DataFrames to store weights for each portfolio strategy at each time t
weights_mv_df = pd.DataFrame(columns=portfolio_data.columns)    # For MV portfolio
weights_nsmv_df = pd.DataFrame(columns=portfolio_data.columns)  # For NSMV portfolio (no short-selling)
weights_mimv_df = pd.DataFrame(columns=portfolio_data.columns)  # For MIMV portfolio (minimum investment per asset)

# Lists to store out-of-sample returns for each portfolio strategy
oos_returns_mv = []                             # MV returns
oos_returns_nsmv = []                           # NSMV returns
oos_returns_mimv = []                           # MIMV returns

print("\n--- Starting Out-of-Sample Evaluation ---")   # Inform user process is starting
print(f"\nProcessing for estimation window: {estimation_window} months...")  # Print current estimation window

# Loop over all possible times t from M to end-1 (to get t+1 returns)
for i, t in enumerate(range(estimation_window, len(portfolio_data) - 1)):

    # ------------------------------ Extract rolling window data ------------------------------
    data_window = portfolio_data.iloc[t - estimation_window : t]            # Get previous M months data up to time t (exclusive)

    # ------------------------------ Estimate mean returns and covariance ------------------------------
    mean_returns_t = data_window.mean().values / 100                        # Calculate unbiased mean returns, convert from % to decimal
    cov_matrix_t = data_window.cov().values / (100 ** 2)                    # Calculate covariance matrix, convert from %^2 to decimal^2

    if i < 5:  # Print first 5 iterations for debugging/verification
        print(f"\nTime t = {portfolio_data.index[t].strftime('%Y-%m-%d')}")   # Show current time index
        print("Estimated mean returns vector:")                             # Print mean returns vector
```

```
        print(pd.Series(mean_returns_t, index=portfolio_data.columns))
        print("\nEstimated covariance matrix:")                    # Print covariance matrix
        print(pd.DataFrame(cov_matrix_t, index=portfolio_data.columns,
columns=portfolio_data.columns))

    # ---------------------------- Define constraints ----------------------------
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},                 # Sum of weights equals 1
        {'type': 'eq', 'fun': lambda w, mu=mean_returns_t: w @ mu - target_expected_return}  #
Portfolio expected return equals target
    ]

    # ---------------------------- MV Portfolio Optimization (Unconstrained weights) ----------
----------
    res_mv = minimize(
        portfolio_variance,                 # Objective function: minimize portfolio variance
        initial_weights,                    # Starting guess for weights (equal weights)
        args=(cov_matrix_t,),               # Pass covariance matrix as argument
        method='SLSQP',                     # Use Sequential Least Squares Programming
optimizer
        constraints=constraints,            # Apply sum and expected return constraints
        bounds=None                         # No bounds on weights (allow short-selling)
    )
    weights_mv = res_mv.x if res_mv.success else initial_weights          # Get optimized
weights or fallback to initial weights

    # ---------------------------- NSMV Portfolio Optimization (No short-selling, weights ≥ 0) --------
---------------------
    bounds_nsmv = [(0, None)] * num_assets                # Set bounds for each asset
weight to be ≥ 0 (no short selling)
    res_nsmv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_nsmv                          # Apply no-short selling bounds
```

```
    )
    weights_nsmv = res_nsmv.x if res_nsmv.success else initial_weights   # Extract optimized weights or fallback

    # ---------------------------- MIMV Portfolio Optimization (Minimum investment constraint) ----------------------
    min_weight = 1 / (2 * num_assets)                    # Define minimum allowed weight per asset
    bounds_mimv = [(min_weight, 1)] * num_assets         # Set bounds so each asset weight ≥ min_weight and ≤ 1
    res_mimv = minimize(
        portfolio_variance,
        initial_weights,
        args=(cov_matrix_t,),
        method='SLSQP',
        constraints=constraints,
        bounds=bounds_mimv                               # Apply minimum investment bounds
    )
    weights_mimv = res_mimv.x if res_mimv.success else initial_weights  # Extract optimized weights or fallback

    # ---------------------------- Store optimal weights for time t ----------------------------
    date_t = portfolio_data.index[t]                     # Get date corresponding to time t
    weights_mv_df.loc[date_t] = weights_mv               # Save MV weights
    weights_nsmv_df.loc[date_t] = weights_nsmv           # Save NSMV weights
    weights_mimv_df.loc[date_t] = weights_mimv           # Save MIMV weights

    # ---------------------------- Calculate out-of-sample return at time t+1 ----------------------------
    returns_next = portfolio_data.iloc[t + 1].values / 100   # Get actual returns at time t+1 in decimal form
    oos_returns_mv.append(weights_mv @ returns_next)         # Calculate portfolio return for MV and append
    oos_returns_nsmv.append(weights_nsmv @ returns_next)     # Calculate portfolio return for NSMV and append
    oos_returns_mimv.append(weights_mimv @ returns_next)     # Calculate portfolio return for MIMV and append
# ------------------------------------------------------------------------
```

```
# ---------------------------- Save results to DataFrame ----------------------------
index_range = portfolio_data.index[estimation_window : len(portfolio_data) - 1]   # Define index range for results
results[estimation_window] = pd.DataFrame({                                  # Create results DataFrame
   'MV_Returns': oos_returns_mv,                                          # MV portfolio out-of-sample returns
   'NSMV_Returns': oos_returns_nsmv,                                       # NSMV portfolio out-of-sample returns
   'MIMV_Returns': oos_returns_mimv                                        # MIMV portfolio out-of-sample returns
}, index=index_range)
# ---------------------------------------------------------------------

# ---------------------------- Output Summary ----------------------------
print("\n--- Out-of-Sample Performance Summary ---")                  # Header for summary output
df = results[estimation_window]                                    # Get results DataFrame for current estimation window

print(f"\nSummary for Estimation Window = {estimation_window} months:")  # Print estimation window info
print(df.describe())                                               # Print descriptive statistics of returns

print(f"\nRealized returns for {estimation_window} months:\n", df.head())  # Show first few realized returns

print("\nFirst few rows of MV Optimal Weights:")                   # Show first few rows of MV weights
print(weights_mv_df.head())

print("\nFirst few rows of NSMV Optimal Weights:")                 # Show first few rows of NSMV weights
print(weights_nsmv_df.head())
```

```
print("\nFirst few rows of MIMV Optimal Weights:")          # Show first few rows of
MIMV weights
print(weights_mimv_df.head())
```

7.3. Appendix C – Code for computation of mean and sample variance for each strategy and mean and standard deviation diagram

```
# ---------------------------- Task 6 ----------------------------
# Question:
# Compute the sample mean and sample variance of returns for each strategy (NSMV, MV, and MIMV)
# using both the in- and out-of-sample evaluation method.
# This will yield 6 realized mean-variance pairs.
# Show these pairs both in tabular form and on a mean-standard deviation diagram.
# ---------------------------------------------------------------

# Import necessary libraries
import matplotlib.pyplot as plt          # For creating plots and visualizations
import seaborn as sns                    # Optional: Enhances the look of matplotlib plots
import pandas as pd                      # For working with data in tabular (DataFrame) format
import numpy as np                       # For numerical computations

# Initialize an empty list to hold data for plotting and tabulation
plot_data = []

# ------------------------- In-Sample Computation -------------------------

# Compute the mean of each strategy's returns from the full sample (in-sample)
in_sample_means = portfolio_returns_df.mean()

# Compute the variance of each strategy's returns from the full sample
in_sample_vars = portfolio_returns_df.var()

# Compute the standard deviation (square root of variance) of each strategy
in_sample_stds = portfolio_returns_df.std()

# Loop over each strategy to append its in-sample metrics to the plot_data list
for strategy in in_sample_means.index:
    plot_data.append({
        'Strategy': strategy,                    # Strategy name (e.g., MV_Returns)
        'Type': 'In-Sample',                     # Evaluation type
```

```
        'Window (M)': 'N/A',                    # Not applicable for in-sample
        'Mean': in_sample_means[strategy],      # Mean return
        'Variance': in_sample_vars[strategy],   # Variance of returns
        'Std Dev': in_sample_stds[strategy]     # Standard deviation of returns
    })

# ----------------------- Out-of-Sample Computation -----------------------

# Loop over each rolling window and its corresponding result DataFrame
for window, df in results.items():
    oos_means = df.mean()       # Compute mean of returns for each strategy in this window
    oos_vars = df.var()         # Compute variance of returns
    oos_stds = df.std()         # Compute standard deviation of returns

    # Append each strategy's metrics to the plot_data list
    for strategy in oos_means.index:
        plot_data.append({
            'Strategy': strategy,               # Strategy name
            'Type': 'Out-of-Sample',            # Evaluation type
            'Window (M)': window,               # Rolling window size used
            'Mean': oos_means[strategy],        # Mean return
            'Variance': oos_vars[strategy],     # Variance
            'Std Dev': oos_stds[strategy]       # Standard deviation
        })

# ---------------------------- Create DataFrame ----------------------------

# Convert the list of dictionaries into a structured DataFrame
plot_df = pd.DataFrame(plot_data)

# ---------------------------- Display Table ----------------------------

# Print heading for the table
print("\n--- Realized Mean, Variance, and Std Dev (In-Sample & Out-of-Sample) ---")

# Set Pandas options to display full width and all columns
pd.set_option('display.width', 1000)
```

```
pd.set_option('display.max_columns', None)

# Print the table rounded to 6 decimal places
print(plot_df.round(6))

# Reset display options back to default
pd.reset_option('display.width')
pd.reset_option('display.max_columns')

# ----------------------------- Define Plot Styles -----------------------------

# Set unique color for each strategy for consistent coloring across plot
strategy_colors = {
    'MV_Returns': '#E63946',    # Red for MV strategy
    'NSMV_Returns': '#2A9D8F',  # Teal for U strategy (Non-shrinkage MV)
    'MIMV_Returns': '#457B9D'   # Blue for MIMV strategy
}

# Define marker shapes to distinguish in-sample from out-of-sample
type_markers = {
    'In-Sample': 'o',   # Circle marker for in-sample
    'Out-of-Sample': 's' # Square marker for out-of-sample
}

# ----------------------------- Create Scatter Plot -----------------------------

# Create a new figure with specified size
plt.figure(figsize=(16, 10))

# Group the data by strategy and evaluation type to control marker and color
for (strategy, eval_type), group_df in plot_df.groupby(['Strategy', 'Type']):
    # Plot each group with corresponding color and marker
    plt.scatter(
        group_df['Std Dev'],                    # x-axis: risk
        group_df['Mean'],                       # y-axis: return
        label=f"{strategy} ({eval_type})",      # Legend label
        color=strategy_colors.get(strategy, '#000000'),   # Fallback to black if missing
```

```
        marker=type_markers.get(eval_type, 'o'),          # Marker based on eval type
        edgecolor='black',                                # Border color for better contrast
        s=200,                                            # Marker size
        alpha=0.9                                         # Marker opacity
    )

# Annotate each point with its actual (mean, std dev) coordinates
for idx, row in plot_df.iterrows():
    plt.annotate(
        f"({row['Mean']:.4f}, {row['Std Dev']:.4f})",     # Label text
        (row['Std Dev'], row['Mean']),                    # Position on plot
        textcoords="offset points",                       # Position relative to point
        xytext=(5,5),                                     # Offset in pixels
        ha='left',                                        # Horizontal alignment
        fontsize=9,
        color='black',
        alpha=0.75
    )

# Add main title
plt.title('Mean-Standard Deviation Diagram: Portfolio Strategy Performance',
          fontsize=18, weight='bold')

# Label x-axis and y-axis
plt.xlabel('Standard Deviation (Risk)', fontsize=14)
plt.ylabel('Mean Return', fontsize=14)

# Add light grid for better readability
plt.grid(True, linestyle='--', alpha=0.5)

# Display legend with title and formatting
plt.legend(title='Strategy (Evaluation Type)',
           fontsize=12, title_fontsize=13, loc='best', frameon=True)

# Adjust layout to prevent overlapping
plt.tight_layout()
```

```
# Show the final plot
plt.show()
```