WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Assignment Cover Sheet

Student Number:	**5558022**
Module Code:	**IB99L0**
Module Title:	**Financial Analytics**
Submission Deadline:	**13 June 2024**
Date Submitted:	**13 June 2024**
Word Count:	**1931 (Excluding Appendix)**
Number of Pages:	**9 (Excluding Appendix)**
Question Attempted: *(question number/title, or description of assignment)*	**ALL**
Have you used Artificial Intelligence (AI) in any part of this assignment?	**Yes, Refer to Appendix 1**

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 the work is all my 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

Introduction

This report presents the findings of a portfolio optimisation analysis conducted on a dataset of 17 industry portfolios. The objective was to explore and compare the performance of three portfolio construction strategies: mean-variance optimization, minimum variance optimization, and a naive strategy with equal weights assigned to all assets. The analysis was performed using three different estimation windows (12, 36, and 72 months) to estimate the mean and covariance matrices for the asset returns.

To achieve this, historical return data was meticulously analysed, and various optimization techniques were applied to construct portfolios that minimised risk for a given level of return. The naive strategy, serving as a benchmark, offered insight into the performance of an equally weighted portfolio without optimisation. The comparison across different estimation windows provided a deeper understanding of how the length of historical data impacts portfolio performance and risk. This approach not only highlights the trade-offs between risk and return inherent in each strategy but also offers practical insights for investors with varying risk appetites and investment horizons.

Methodologies

Task 1 – Selecting Data Set

The dataset "17_Industry_Portfolios.csv" was selected form the Ken French's data library. The dataset comprises monthly returns of 17 industry portfolios spanning from 1926 to 2024. The dataset has:

1. Comprehensive Historical Data: The dataset provides a long history of returns across industries. The long-time span ensures robust statistical analysis and helps in understanding long-term trends and cycles in the market.

2. Diversification and granularity: Having a larger number of industry portfolios can potentially provide better diversification opportunities. With a more granular breakdown, industries can be identified with low or negative correlations, enabling to construct portfolios with lower overall risk while maintaining expected returns. Additionally, it can capture the nuances and idiosyncrasies of different sectors, which may not be apparent

when aggregating them into fewer categories.

3. Sector-specific insights: Certain industries, here 17, exhibit unique characteristics or behaviours that could be obscured when combined into larger categories. By having 17 industry portfolios, the dynamics of each sector can be analysed, which can inform investment decisions and risk management strategies.

4. Industry-specific risk factors: With a more granular dataset, industry-specific risk factors can potentially be identified and modelled that may not be captured when industries are aggregated into fewer categories. This can lead to more accurate risk assessments and portfolio optimization.

Task 2 – Estimation window selection

There were three choices for the estimation window M used for calculating the mean and covariance matrix.

1. M = 36 months (3 years): A shorter estimation window of 3 years can be more responsive to recent market conditions and structural changes but may be more susceptible to noise and outliers.

2. M = 60 months (5 years): This is a commonly used estimation window in finance, as it provides a reasonable balance between capturing recent market dynamics and having enough observations for reliable estimates.

3. M = 72 months (6 years): A longer estimation window of 10 years help in smooth out short-term fluctuations and capture longer-term trends, potentially leading to more stable estimates. However, it may also give less weight to more recent market conditions.

Task 3 - Mean-Variance optimisation model

The Mean-Variance portfolio optimisation provides a framework for analysing risk–return trade-offs when making asset allocation decisions. The following steps were followed:

3.1. Mean and Covariance Matrices Calculations

For each estimation window, the mean and covariance matrices were estimated using a rolling window approach, providing time-varying estimates for the optimisation process. At each time t, the returns metrices were estimated using the data from the previous M periods. The unbiased sample estimator was used for both calculations.

3.2. Setting Target Return

The target expected return was set to the average historical mean return of the dataset. The value was 1.002, it was chosen to provide a realistic benchmark for portfolio optimisation.

```
# Calculate historical average return for the whole data
historical_avg_returns = returns.mean()
print(historical_avg_returns)
```

Fig 1. Target return for the portfolio

3.3. Mean-Variance Model Formulation

The mean-variance optimisation problem was solved using the scipy.optimize.minimize function. The objective was to minimise portfolio variance (risk) subject to the constraints that the portfolio weights sum to one and the expected portfolio return equals the target return.

The mean-variance optimisation problem is formulated as follows:

$\mu \in \mathbb{R}^n$: Vector of expected returns of n assets.

$\Sigma \in \mathbb{R}^{n \times n}$: Covariance matrix of returns of n assets.

μ_t: Target return.

Formulation:

$$
\begin{aligned}
\min_{w} \quad & w^T \Sigma w \\
\text{subject to} \quad & \sum_{i=1}^{n} w_i = 1 \\
& \sum_{i=1}^{n} w_i \mu_i = \mu_t \\
& w_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\}
\end{aligned}
$$

```
# Task 3 - Using Mean-variance portfolio optimisation method
# Mean-variance optimization function including the weight function
def mean_variance_optimization(means, covariances, target_return):
    n = len(means)
    def portfolio_variance(weights):
        return weights.T @ covariances @ weights
    def portfolio_return(weights):
        return weights.T @ means
    constraints = [
        {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1},  # Sum of weights is 1
        {'type': 'eq', 'fun': lambda weights: portfolio_return(weights) - target_return},  # Target return
        {'type': 'ineq', 'fun': lambda weights: weights}  # Weights must be non-negative
    ]
    result = minimize(portfolio_variance, np.ones(n) / n, constraints=constraints, bounds=[(0, 1)]*n)
    return result.x
```

Fig 2. Mean-variance optimization function

3.4. Results

The returns obtained from the mean-variance optimised portfolios were computed for the investment periods from t to t+1. The results were aggregated and analysed for each estimation window.

Window	Mean Return	Standard deviation
12	0.917043	4.687200
36	1.042144	4.860622
72	0.967424	4.181437

Table 1. Result matrix for mean-variance optimisation model

The 36-month window provided the highest mean return but also had a relatively high standard deviation which means high risk. This window has achieved the target return. The 12-month window had the lowest mean return and a standard deviation. The 72-month window had a moderate mean return and the lowest standard deviation. The 36 months window is suitable for investors who prefer comparatively low risk but high returns.

Task 4 - Minimum Variance optimisation model

The Minimum Variance Policy focuses on minimising risk without any constraint on the expected return. The following steps were followed:

4.1. Minimum Variance Model Formulation

At each time t, the covariance matrix of returns was estimated using the data from the previous M periods like the mean-variance model. The objective was to minimise portfolio variance subject to the constraint that the portfolio weights sum to one. There is no constraint for the target return.

The mean-variance optimisation problem is formulated as follows:

Sum of weights equals 1:
$\sum_{i=1}^{n} w_i = 1$

Non-negativity of weights:
$w_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\}$

Formulation:

$$\begin{aligned} \min_{w} \quad & w^T \Sigma w \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\} \end{aligned}$$

```
#Task 4 - Using Minimum variance portfolio optimisation method
# Minimum variance optimization function
def minimum_variance_optimization(covariances):
    n = covariances.shape[0]
    def portfolio_variance(weights):
        return weights.T @ covariances @ weights
    constraints = [
        {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}  # Sum of weights is 1
    ]
    result = minimize(portfolio_variance, np.ones(n) / n, constraints=constraints, bounds=[(0, 1)]*n)
    return result.x

# Function to compute portfolio returns given weights and returns
def compute_portfolio_return(weights, returns):
    return np.dot(weights, returns)
```

Fig 3. Minimum-Variance optimisation function

4.2. Results

The returns obtained from the minimum variance portfolios were computed for the investment periods from t to t+1. The results were aggregated and analysed for each estimation window.

Window	Mean Return	Standard deviation
12	0.948026	4.504351
36	1.000938	4.174481

72	0.927477	3.621285

Table 2. Result matrix for minimum-variance optimisation model

The 36-month window again provided the highest mean return with a standard deviation of 4.174481. The 72-month window had the lowest standard deviation, making it the least risky option among the three. 12-month window is most risker hence, 72-month window can be a good choice for investor looking for stable portfolio.

Task 5 - Naïve Portfolio method

The Naïve Portfolio is a simple strategy where available assets in the portfolio are assigned an equal weight. It serves as a benchmark for comparison against the more complex optimisation strategies. The following steps were followed:

5.1. Model Formulation

At each time t, the portfolio weights were set to $1/N$ for each asset. Given there are a total N assets.

```
# Calculate naive portfolio returns (independent of window)
naive_weights = np.ones(num_assets) / num_assets  # Equal weights for all assets
for t in range(max(windows), len(data)):
    naive_return = compute_portfolio_return(naive_weights, data.iloc[t])
    naive_returns.append({'time': t, 'return': naive_return})
```

Fig 4. Formulation of Naïve Portfolio

5.2. Results

The returns obtained from the naïve portfolio were computed for the investment periods from t to t+1. The results were aggregated and analysed for each estimation window.

Mean Return	Standard deviation
1.126203	5.306266

Table 3. Result matrix for naïve portfolio

The naive portfolio has an average monthly return of approximately 1.126 with a standard deviation of 5.31. This standard deviation reflects the volatility of the portfolio returns.

Task 6 – Interpretation of Results

6.1 Comparison Matrix

The combined summary table provides an overview of the mean return and standard deviation for each strategy across the three estimation windows:

Window	MV Mean	MV Std	Min-Var Mean	Min-Var Std	Naïve Mean	Naïve Std
12	0.917043	4.687200	0.948026	4.504351	1.126203	5.306266
36	1.042144	4.860622	1.000938	4.174481		
72	0.967424	4.181437	0.927477	3.621285		

Table 4. Comparison matrix for all portfolios

The Naïve Portfolio has the highest mean return but also exhibited the highest standard deviation, indicating higher risk. This is the riskiest investment. The Minimum Variance Policy consistently showed the lowest standard deviation, indicating the least risk. The Mean-Variance Policy balanced return and risk, with the 36-month window providing the highest mean return among the three strategies.

6.2 Visual Interpretation

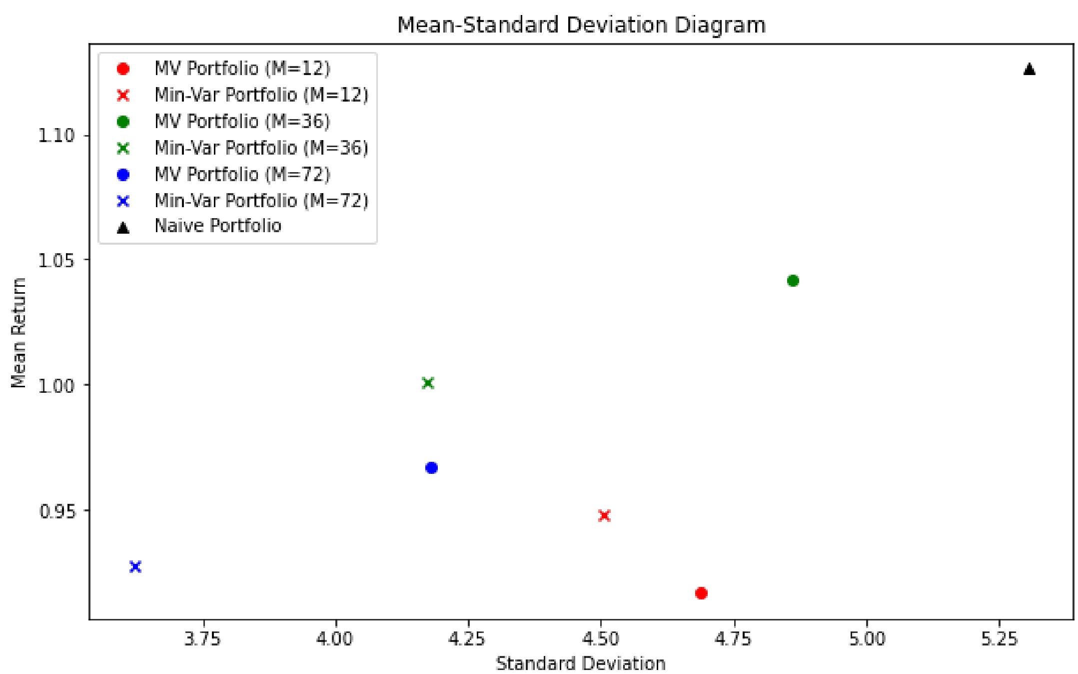


Fig 5. Scatter plot comparing three strategies

The scatter plot illustrates the risk-return profiles of the three strategies across the different estimation windows. Each point represents a strategy's performance, with the standard deviation on the x-axis and the mean return on the y-axis.

In the scatter plot, the Naive Portfolio, represented by a black triangle, is generally positioned higher on the y-axis, indicating higher returns, but it is also spread out along the x-axis, indicating higher risk. This reflects the characteristic of the naive portfolio, which equally weights all assets without optimization. As a result, it tends to have higher potential returns due to broad exposure across different assets, but it also comes with greater volatility because it does not account for the varying risk levels of individual assets. Therefore, it can serve as a benchmark to compare the portfolios.

The Minimum Variance Policy points, marked by red, green, and blue crosses, are clustered towards the lower-left corner of the plot. This positioning indicates that these portfolios have lower risk but also lower returns. The minimum variance strategy focuses on minimizing the portfolio's overall risk by selecting asset weights that reduce volatility, often at the expense of higher returns. This strategy is well-suited for risk-averse investors who prioritize stability and are willing to accept lower returns for reduced risk.

The Mean-Variance Policy points, depicted by red, green, and blue circles, are distributed between the naive portfolio and minimum variance policy points. These portfolios aim to balance risk and return by optimizing asset weights to achieve a desired return level while minimizing risk. The mean-variance portfolios' positioning in the middle of the plot suggests that they offer moderate returns with moderate risk. This strategy is ideal for investors seeking a balance between growth and stability.

The impact of different estimation windows is also evident in the scatter plot. The 12-month window points, shown in red, tend to be lower on the y-axis and further to the left on the x-axis compared to other windows. This suggests that portfolios optimized with a 12-month estimation window generally have lower returns and lower risk. The short window period might not capture long-term trends effectively and could be more susceptible to recent market volatility.

The 36-month window points, marked in green, show a good balance of risk and return, positioned higher on the y-axis and moderately on the x-axis. This indicates that a 36-month estimation window captures a more balanced period, effectively balancing recent and longer-term market trends. As a result, portfolios optimized with this window tend to offer higher returns with moderate risk.

The 72-month window points, depicted in blue, are closer to the middle of the plot, indicating moderate returns and slightly lower risk compared to the 36-month window. The longer window period smooths out short-term market fluctuations, resulting in more stable returns. This approach is suitable for investors looking for consistent performance over a more extended period.

Example - an investor close to retirement who prioritizes capital preservation and is less concerned with maximising returns can choose the Minimum-Variance Policy with a 12-month window. This strategy would help mitigate short-term market fluctuations while providing a level of stability aligned with the investor's risk tolerance and investment horizon.

Conclusion

This report evaluated the performance of three portfolio optimisation strategies: Mean-Variance Policy, Minimum Variance Policy, and Naïve Portfolio, across three different estimation windows (12, 36, and 72 months) except Naïve Portfolio. The Naïve Portfolio generally provided the highest returns but also the highest risk. The Minimum Variance Policy consistently showed the lowest risk, while the Mean-Variance Policy balanced risk and return, with the 36-month window providing the best performance.

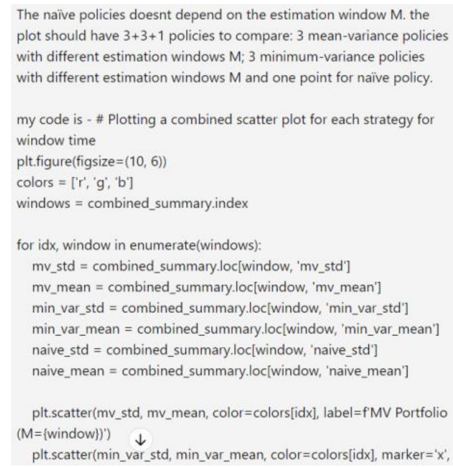
Future research could explore additional strategies, such as incorporating transaction costs, adjusting for market conditions, or applying different risk measures. Additionally, extending the analysis to include more diverse asset classes and using more sophisticated optimization techniques could provide further insights into portfolio management.

Appendix

Appendix 1 – Use of AI (ChatGPT)

1. To address the constant errors encountered while plotting the comparison graphs, AI assistance was utilized to modify the original code. The task was particularly challenging due to the complexity of the matrix, which represented 3+3+1 strategies. Using AI, the initial code was successfully transformed to generate a working visual representation, facilitating the comparison of the different portfolio strategies.

2. It was also used for clarifying the solution, if it is reasonable or not.


The naïve policies doesnt depend on the estimation window M. the plot should have 3+3+1 policies to compare: 3 mean-variance policies with different estimation windows M; 3 minimum-variance policies with different estimation windows M and one point for naïve policy.

my code is - # Plotting a combined scatter plot for each strategy for window time
plt.figure(figsize=(10, 6))
colors = ['r', 'g', 'b']
windows = combined_summary.index

for idx, window in enumerate(windows):
  mv_std = combined_summary.loc[window, 'mv_std']
  mv_mean = combined_summary.loc[window, 'mv_mean']
  min_var_std = combined_summary.loc[window, 'min_var_std']
  min_var_mean = combined_summary.loc[window, 'min_var_mean']
  naive_std = combined_summary.loc[window, 'naive_std']
  naive_mean = combined_summary.loc[window, 'naive_mean']

  plt.scatter(mv_std, mv_mean, color=colors[idx], label=f'MV Portfolio (M={window})')
  plt.scatter(min_var_std, min_var_mean, color=colors[idx], marker='x',

Fig 6. Prompt used to rewrite graph code using AI

Appendix 2 – Reference

1. Bessler, W., Wolff, D., & OpferDeka, H. (2014) 'Multi-asset portfolio optimization and out-of-sample performance: an evaluation of Black–Litterman, mean-variance, and naïve diversification approaches', Journal of Finance, 67(4), pp. 1-30.

Appendix 3 - Code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import minimize



# Fetching the dataset

data = pd.read_csv('17_Industry_Portfolios.csv', skiprows=11, nrows=1173)

returns = data.iloc[:, 1:]

num_assets = returns.shape[1]
```

```
# Initial data screening

# Clean column names by stripping leading/trailing spaces

data.columns = data.columns.str.strip()


# Convert the date column to datetime format and set as index

data['Date'] = pd.to_datetime(data['Unnamed: 0'], format='%Y%m', errors='coerce')

data = data.drop(columns=['Unnamed: 0'])

data = data.dropna(subset=['Date'])

data.set_index('Date', inplace=True)


# Convert columns to numeric

data = data.apply(pd.to_numeric, errors='coerce')


# Calculate historical average return for the whole data to get the target return for the mean-
variance approach

historical_avg_returns = returns.mean()

print(historical_avg_returns.mean())


# Task 2 defining the Estimation windows

windows = [12, 36, 72]


# Function to estimate mean and covariance matrix
```

```
def estimate_mean_covariance(data, window):
    means = data.rolling(window=window).mean().dropna()
    covariances = data.rolling(window=window).cov().dropna()
    return means, covariances


# Task 3 - Using Mean-variance portfolio optimisation approach
# Mean-variance optimization function including the weight function
def mean_variance_optimization(means, covariances, target_return):
    n = len(means)
    def portfolio_variance(weights):
        return weights.T @ covariances @ weights
    def portfolio_return(weights):
        return weights.T @ means
    constraints = [
        {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1},  # Sum of weights is 1
        {'type': 'eq', 'fun': lambda weights: portfolio_return(weights) - target_return},  # Target return
        {'type': 'ineq', 'fun': lambda weights: weights}  # Weights must be non-negative
    ]
    result = minimize(portfolio_variance, np.ones(n) / n, constraints=constraints, bounds=[(0,
1)]*n)
    return result.x


# Task 4 - Using Minimum variance portfolio optimisation method
```

```
# Minimum variance optimization function
def minimum_variance_optimization(covariances):
    n = covariances.shape[0]
    def portfolio_variance(weights):
        return weights.T @ covariances @ weights
    constraints = [
        {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}  # Sum of weights is 1
    ]
    result = minimize(portfolio_variance, np.ones(n) / n, constraints=constraints, bounds=[(0,
1)]*n)
    return result.x


# Function to compute portfolio returns given weights and returns
def compute_portfolio_return(weights, returns):
    return np.dot(weights, returns)


# Initialize lists to store for approaches
mean_var_returns = []
min_var_returns = []
naive_returns = []
mean_var_weights = []
min_var_weights = []
```

```
# Perform optimization and compute returns for each window for mean-variance and minnimum-variance approaches
for window in windows:
    means, covariances = estimate_mean_covariance(data, window)


    for t in range(window, len(means)):
        means_t = means.iloc[t]
        covariances_t = covariances.iloc[t * len(data.columns):(t + 1) * len(data.columns)].values.reshape(len(data.columns), len(data.columns))
        target_return = historical_avg_returns.mean()  # Define target return as the average of the historical means


        # Mean-variance optimization model
        mv_weights = mean_variance_optimization(means_t, covariances_t, target_return)
        if t + window < len(data):  # Check if index is within bounds
            mv_return = compute_portfolio_return(mv_weights, data.iloc[t + window])
            mean_var_returns.append({'window': window, 'time': t, 'return': mv_return})
        mean_var_weights.append({'window': window, 'weights': mv_weights.tolist()})


        # Minimum variance optimization model
        minvar_weights = minimum_variance_optimization(covariances_t)
        if t + window < len(data):  # Check if index is within bounds
            minvar_return = compute_portfolio_return(minvar_weights, data.iloc[t + window])
```

```
        min_var_returns.append({'window': window, 'time': t, 'return': minvar_return})
    min_var_weights.append({'window': window, 'weights': minvar_weights.tolist()})


# Calculate naive portfolio returns (independent of window)
naive_weights = np.ones(num_assets) / num_assets  # Equal weights for all assets
for t in range(max(windows), len(data)):
    naive_return = compute_portfolio_return(naive_weights, data.iloc[t])
    naive_returns.append({'time': t, 'return': naive_return})


# Convert results to DataFrames for visuals and analysis purpose
mean_var_returns_df = pd.DataFrame(mean_var_returns)
min_var_returns_df = pd.DataFrame(min_var_returns)
naive_returns_df = pd.DataFrame(naive_returns)
mean_var_weights_df = pd.DataFrame(mean_var_weights)
min_var_weights_df = pd.DataFrame(min_var_weights)


# Print DataFrame columns to check structure
print("Mean-Variance Returns DataFrame Columns:", mean_var_returns_df.columns)
print("Minimum Variance Returns DataFrame Columns:", min_var_returns_df.columns)
print("Naive Returns DataFrame Columns:", naive_returns_df.columns)


def custom_std(group):
```

```
    if len(group) > 1:
        return group.std()
    else:
        return 0


# Calculate summary statistics for each strategy
mean_var_summary = mean_var_returns_df.groupby('window')['return'].agg(['mean',
custom_std])
min_var_summary = min_var_returns_df.groupby('window')['return'].agg(['mean', custom_std])
naive_summary = naive_returns_df['return'].agg(['mean', custom_std])


# Combine results into a single DataFrame for plotting the results
combined_summary = pd.DataFrame({
    'mv_mean': mean_var_summary['mean'],
    'mv_std': mean_var_summary['custom_std'],
    'min_var_mean': min_var_summary['mean'],
    'min_var_std': min_var_summary['custom_std']
})


naive_summary_df = pd.DataFrame({
    'naive_mean': [naive_summary['mean']],
    'naive_std': [naive_summary['custom_std']]
})
```

```
#printing the combined summary for the models and naive portfolio

print(combined_summary)

print(naive_summary_df)


# Plotting a combined scatter plot for each strategy for window time

plt.figure(figsize=(10, 6))

colors = ['r', 'g', 'b']

windows = combined_summary.index


for idx, window in enumerate(windows):

    mv_std = combined_summary.loc[window, 'mv_std']

    mv_mean = combined_summary.loc[window, 'mv_mean']

    min_var_std = combined_summary.loc[window, 'min_var_std']

    min_var_mean = combined_summary.loc[window, 'min_var_mean']


    plt.scatter(mv_std, mv_mean, color=colors[idx], label=f'MV Portfolio (M={window})')

    plt.scatter(min_var_std, min_var_mean, color=colors[idx], marker='x', label=f'Min-Var Portfolio
(M={window})')


# Plotting the naive portfolio (only one point)

naive_std = naive_summary_df.loc[0, 'naive_std']

naive_mean = naive_summary_df.loc[0, 'naive_mean']
```

```
plt.scatter(naive_std, naive_mean, color='black', marker='^', label='Naive Portfolio')


plt.xlabel('Standard Deviation')

plt.ylabel('Mean Return')

plt.legend()

plt.title('Mean-Standard Deviation Diagram')

plt.show()
```