Comparing Parametric and Non-Parametric Monte-Carlo and Machine Learning Approaches to Development of Risk-Adjusted Portfolio Construction

Table of contents

Section 1: Introduction and Summary

This notebook intends to analyze, compare, and provide insights on the performance of two fundamentally different approaches to **portfolio risk modelling and risk-adjusted portfolio construction**:

- One **machine-learning–based risk modelling framework**, and
- Two **Monte Carlo simulation–based risk modelling frameworks - Parametric and Non-Parametric**.

The comparison is conducted across the following dimensions:

1. Risk-adjusted performance comparison:
Measures how much excess return each approach generates - after accounting for its own risk profile. For this dimension, the 3 approaches will be compared using the following Out-of-Sample (OOS) parameters:

- Rolling realized volatility (12Months)
- Rolling model-implied volatility (under fixed weights w^T)
- Regime-controlled Sharpe and Volatility
- Regime-controlled Sharpe versus CVaR Trade-off

2. Tail risk and drawdown analysis:
Measures the effectiveness of each portfolio construction approach in capturing and controlling extreme downside risk. For this dimension, the 3 approaches will be compared using the following OOS parameters:

- Tail loss frequency
- Expected shortfall duration
- Maximum drawdown

3. Portfolio stability analysis:
Measures the stability and consistency of portfolio decisions generated by each portfolio construction approach over time. For this dimension, the 3 approaches will be compared using the Monthly Turnover parameter.

Based on the findings above, this notebook aims to determine **Model Applicability**—that is, which framework is better suited for **risk estimation, portfolio optimization support, and stress-aware decision making**.

1. Dataset Description

The analysis uses **daily closing price data** spanning approximately **2001–2026**, covering multiple market regimes including the Global Financial Crisis (2008), the COVID-19 shock (2020), and the post-pandemic monetary tightening cycle.

The asset universe is intentionally restricted to three broad instruments to emphasize **diversification, correlation structure, and regime behavior**, rather than security-selection alpha:

1. **NIFTY 50 Index** – proxy for broad Indian equity market exposure (sourced from Zerodha using Kite API)
2. **NIFTY Bank Index** – proxy for cyclical, leverage-sensitive financial equities (sourced from Zerodha using Kite API)
3. **Gold** – proxy for defensive and inflation-hedging behavior (sourced from Yahoo Finance - COMEX Gold Futures - CMEGroup - GC=F)

Daily prices are transformed into **logarithmic returns**, which form the foundational input for all subsequent modelling stages. Log returns ensure time-additivity, scale consistency, and comparability across assets with heterogeneous price levels.

2. Analytical Objective

Unlike return-forecasting exercises, this notebook focuses explicitly on **risk estimation and portfolio behavior**. The primary objective of this exercise is **not** to attempt at predicting future prices or returns directly. Instead, this project evaluates how different risk-modelling philosophies interpret uncertainty and how that uncertainty affects portfolio outcomes.

3. Modelling Approach

Two distinct and complementary modelling frameworks are implemented.

Monte Carlo Risk Modelling (Gaussian and Non-Gaussian)

The Monte Carlo framework represents a **distribution-based, probabilistic approach** to risk estimation. Using historical return data, the model:

- Estimates expected returns and covariance structure
- Simulates a large number of potential future return paths
- Aggregates simulated outcomes to infer portfolio-level risk measures

This approach explicitly models **distributional uncertainty** and is particularly well-suited for understanding tail risk and scenario dispersion.

Machine Learning Risk Modelling

The machine learning framework treats **risk as a conditional, learnable function of historical return dynamics**. Rather than assuming a fixed parametric distribution, the model infers patterns such as:

- Volatility clustering
- Non-linear dependencies
- Regime-dependent behavior

Engineered features encode memory, dispersion, and drawdown effects observed in financial time series. Model outputs are used to generate **forward-looking volatility estimates**, which feed directly into portfolio construction.

4. Portfolio Construction and Evaluation

Portfolio construction is governed by a **single, fixed decision rule** applied consistently across both modelling approaches.

- **Risk-adjusted return metric:** Sharpe ratio
- **Optimization objective:** Maximize Sharpe ratio
- **Constraints:**
  - Long-only
  - Fully invested

By comparing **pre- and post-rebalancing portfolios under each risk model and portfolio construction rule**, the notebook highlights how **model choice alone can materially alter portfolio behavior—even when using the same underlying assets**. Evaluation focuses on the 8 OOS parameters defined above.

5. Implementation Methodology

The empirical analysis follows a structured, walk-forward research design - to ensure methodological rigor and avoid look-ahead bias.

Step 1: Data Collection and Cleansing

Three independent price series are sourced:

- NIFTY 50 prices
- NIFTY Bank prices
- Gold price proxy (Future Rates instead of Spot Rates)

All datasets are:

- Cleaned for missing observations

- Aligned to a common date range
- Prepared to ensure consistency across assets

Step 2: Computation of Daily Log Returns

For each asset:

- Daily log returns are computed
- This return series constitutes the **only shared input across all models**

All downstream modelling—Monte Carlo and ML—is built exclusively on this return representation.

Step 3: Feature Engineering (Asset-Specific)

For each asset independently, the following features are engineered:

- Rolling volatility estimates (20-day, 60-day)
- Lagged return terms
- Rolling mean returns
- Drawdown-based measures

These features are **used exclusively by the machine learning framework** and are not inputs to the Monte Carlo simulation.

Step 4: Portfolio Decision Rule Definition

Prior to any modelling, the portfolio construction rule is defined **once and fixed throughout the study**.

- **Objective:** Maximize Sharpe ratio
- **Constraints:**
  - Long-only (only positive positions held in assets - no short positions considered)
  - Fully invested (portfolio weight = 100%)

This ensures that differences in outcomes arise solely from **risk estimation**, not from changing optimization logic.

Step 5: Walk-Forward Portfolio Construction

This step constitutes the core of the project and is repeated at each monthly rebalancing date.

Step 5A — Monte Carlo Risk Estimation

At rebalancing date T:

- Use only historical returns available up to T (e.g., trailing 3 years)
- Estimate:
  - Mean returns
  - Covariance matrix (volatility and correlations)
- Run Monte Carlo simulations to generate future return paths
- Compute risk inputs for portfolio optimization

- Pass these inputs into the Sharpe optimizer
- Obtain portfolio weights

Step 5B — Machine Learning Risk Estimation

At the same rebalancing date T:

- Train ML models (per asset) using only historical features
- Predict next-period volatility for each asset
- Combine predicted volatilities with rolling correlation estimates
- Construct a forecasted covariance matrix
- Pass these inputs into the same Sharpe optimizer
- Obtain portfolio weights

Step 6: Out-of-Sample Holding Period

- Both portfolios are held for the subsequent month
- Realized portfolio returns are recorded

Step 7: Iteration Through Time

Steps 5 and 6 are repeated across the full sample period, generating:

- A Monte Carlo–based portfolio return series
- A Machine Learning–based portfolio return series

Step 8: Comparative Evaluation

The two portfolios are compared across the 8 OOS parameters to draw conclusions on each.

Summarized Findings:

Across the full out-of-sample diagnostic stack—volatility recognition (realized and implied), regime-conditioned Sharpe and Sharpe–CVaR trade-offs, tail-loss behavior (TLF/ESD), drawdown containment (MDD), and turnover stability—the **Bootstrap Monte-Carlo** approach best matches the project's objective of risk-adjusted portfolio construction: it is empirically consistent with the non-normal return structure identified upfront, achieves the lowest Tail Loss Frequency (3.6%), preserves (and marginally improves) Sharpe without worsening CVaR versus Gaussian MC, and maintains moderate turnover that remains implementable relative to XGBoost. Gaussian MC remains a valuable stable baseline but is structurally constrained by distributional misspecification when normality is rejected. XGBoost, while offering the greatest modelling flexibility and the clearest regime sensitivity (sharp implied/realized risk transitions) and slightly better maximum drawdown, pays for this with lower Sharpe, deeper CVaR in stress, higher tail-loss recurrence, and materially higher turnover—making it better positioned as a risk regime identification and signalling framework (or an overlay) than as the primary engine for stable risk-adjusted portfolio construction in this specific setup.

Additionally, a central methodological principle of this study is that model selection is driven by data characteristics rather than implementation convenience. Prior to any portfolio simulations or machine-learning training, the underlying daily log-return series was subjected to a battery of statistical diagnostics, including skewness and kurtosis analysis, normality tests, and checks for volatility clustering. The objective was to explicitly assess whether a parametric Gaussian risk model was statistically admissible for the assets and regimes under consideration.

The analysis clearly indicated persistent non-normality, fat tails, and asymmetry across assets and regimes. As a result, a Gaussian Monte-Carlo framework could only be justified as a benchmarking reference rather than as a primary risk engine. This upfront statistical assessment naturally motivated the selection of Bootstrap (non-parametric) Monte-Carlo as a more appropriate distribution-aware approach. By resampling directly from empirical return distributions, the bootstrap framework preserves observed tail behavior and skewness without imposing restrictive parametric assumptions. Importantly, this methodological decision was validated ex post by the out-of-sample results, demonstrating that correct statistical foundations materially influence downstream portfolio outcomes.

Disclaimer

This notebook is intended solely for **methodological comparison and analytical insight**. The portfolios constructed are illustrative and do not constitute investment advice. All modelling frameworks rely on historical information and implicitly assume that past statistical properties are informative of future risk. In practice, financial markets are subject to **structural breaks, policy interventions, and exogenous shocks** that may invalidate these assumptions. Accordingly, results should be interpreted as **indicative rather than definitive**, and any real-world deployment would require additional validation, stress testing, and governance controls.

Section 2: Data Load, Statistical Tests and Exploratory Data Analysis

Since the raw data has already been pre-processed using a separate ETL pipeline (please refer to GitHub repo: https://github.com/thenaivecoder89/statistical_and_machine_learning_models/tree/main/data_sourcing filename: ZERODHA_YFINANCE_ETL_PROJECT_3.py for the full ETL codebase), this section will focus more on loading the pre-processed data and visualizing the same - along with performing the following statistical tests to gather insights for approach identification:

1. Test of Kurtosis - to measure tail thickness - if leptokurtic/ platykurtic: select Non-Gaussian Monte-Carlo Approach - No change to XGBoost implementation but engineered features to be modified accordingly.
2. Test of Skewness - to check assymetry in the distribution - if left/ right skewed: select Non-Gaussian Monte-Carlo Approach - No change to XGBoost implementation but engineered features to be modified accordingly.
3. Normality Rejection Tests (Jarque–Bera (moment-based) and Anderson-Darling (tail-sensitive)) - to conclusively reject Gaussian Monte-Carlo Approach.
4. Volatility Clustering - ACF and Ljung-Box on squared returns - to confirm presence of heteroskedacity and subsequently support development of features for XGBoost model.

While the above tests provide strong empirical guidance on the appropriateness of Gaussian versus Non-Gaussian Monte-Carlo assumptions, both approaches are implemented in this study to enable a thorough comparative analysis. In addition, this section reports Pearson correlation coefficients across all three assets to quantify cross-asset dependence and justify the use of multivariate risk modeling in the Monte-Carlo simulations.

2.1. Data Load and Environment Initialization

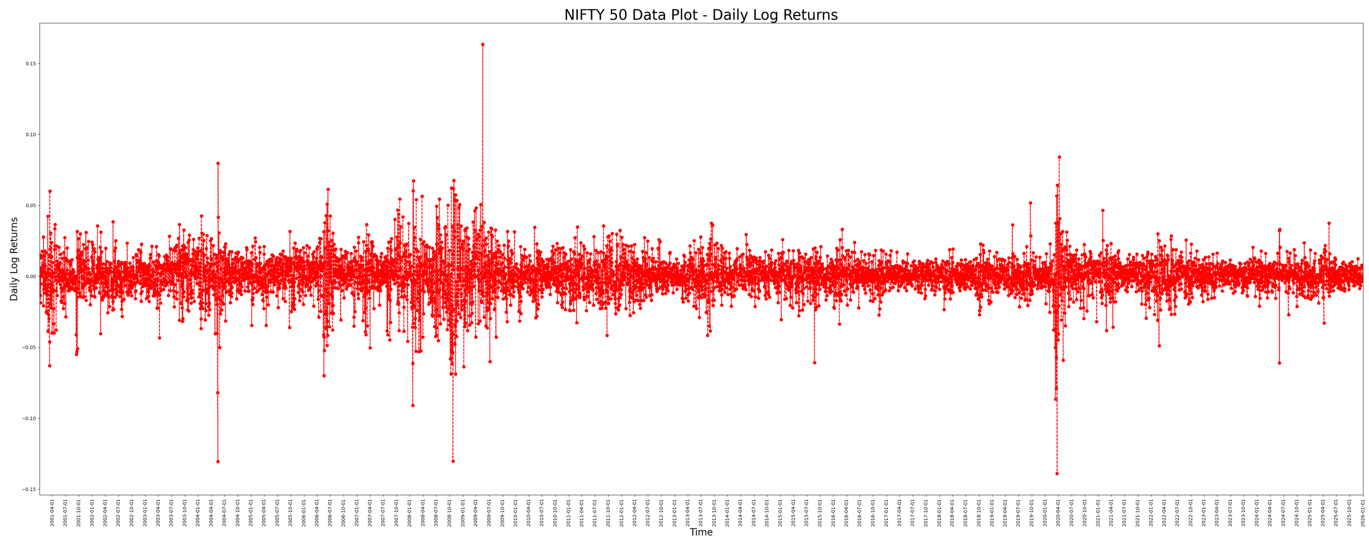
2.2. Exploratory Data Analysis (EDA) and Statistical Tests

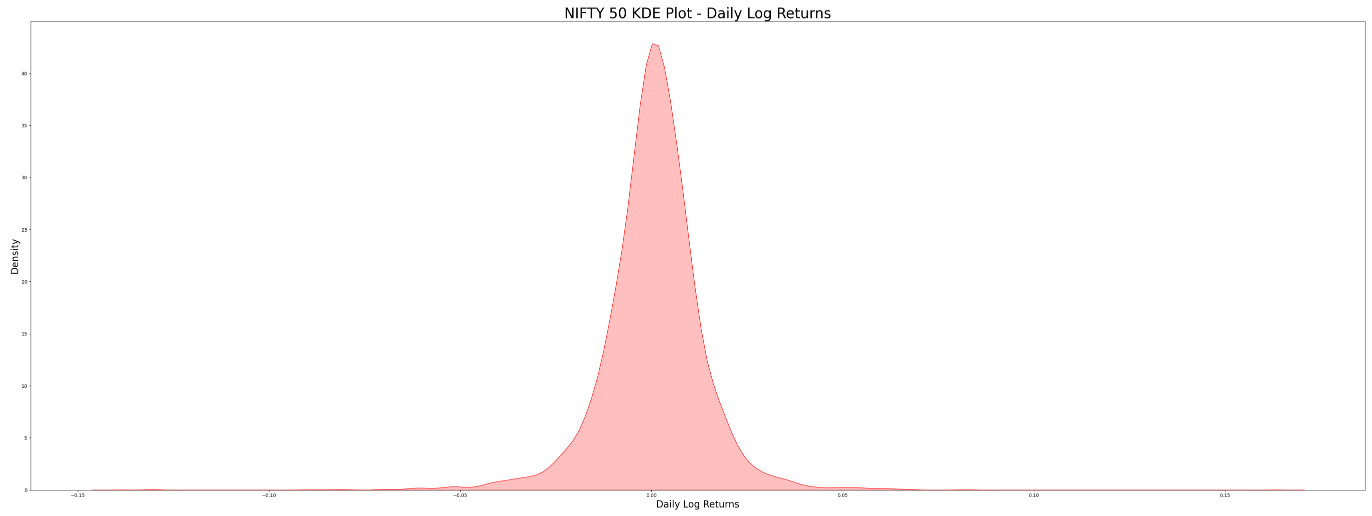
This section focuses on conducting EDA and statistical analysis on each of the 3 datasets: - NIFTY 50 - NIFTY Bank - Gold Futures (considered as a proxy for Gold spot)

To generate critical insights on the project way-forward across the following areas: - Monte-Carlo: Gaussian v/s Non-Gaussian (accounting for kurtosis, skewness and volatility clustering) - XGBoost: Feature engineering focus - Risk Metrics: Emphasis on Value at Risk (VaR) or Conditional Value at Risk (CVaR)

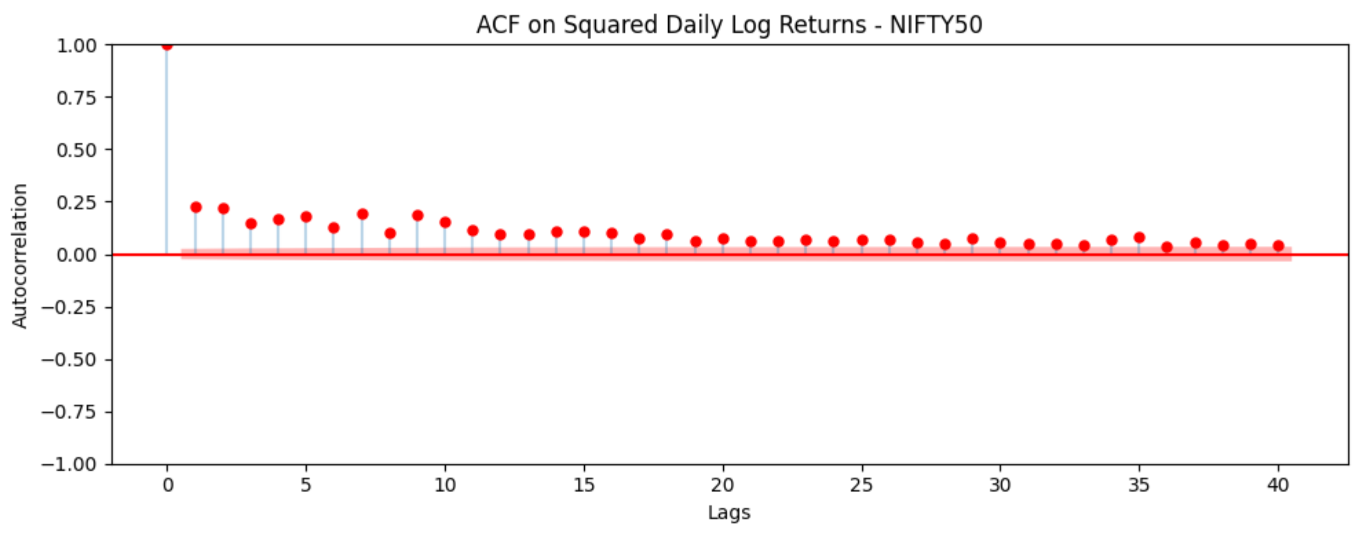
2.2.1. EDA and Tests on NIFTY50 data

```
SUMMARY STATISTICS
NIFTY50 dataset - Top 5 and Bottom 5 rows:
   SNo.        date  closing_value  daily_closing_pct_change  \
0     1  2001-01-02         1271.8                  0.013952
1     2  2001-01-03         1291.2                  0.015254
2     3  2001-01-04         1307.6                  0.012701
3     4  2001-01-05         1327.2                  0.014989
4     5  2001-01-08         1309.2                 -0.013562

   daily_log_closing_value  instrument_token trading_symbol
0                 0.013856            256265       NIFTY 50
1                 0.015139            256265       NIFTY 50
2                 0.012621            256265       NIFTY 50
3                 0.014878            256265       NIFTY 50
4                -0.013655            256265       NIFTY 50
      SNo.        date  closing_value  daily_closing_pct_change  \
6213  6214  2025-12-26       26042.30                 -0.003818
6214  6215  2025-12-29       25942.10                 -0.003848
6215  6216  2025-12-30       25938.85                 -0.000125
6216  6217  2025-12-31       26129.60                  0.007354
6217  6218  2026-01-01       26146.55                  0.000649

      daily_log_closing_value  instrument_token trading_symbol
6213                -0.003825            256265       NIFTY 50
6214                -0.003855            256265       NIFTY 50
6215                -0.000125            256265       NIFTY 50
6216                 0.007327            256265       NIFTY 50
6217                 0.000648            256265       NIFTY 50
NIFTY50 summary statistics:
              SNo.  closing_value  daily_closing_pct_change  \
count  6218.000000    6218.000000               6218.000000
mean   3109.500000    8496.479468                  0.000578
std    1795.126319    6706.618571                  0.013340
min       1.000000     854.200000                 -0.129805
25%    1555.250000    3526.550000                 -0.005418
50%    3109.500000    6025.675000                  0.000850
75%    4663.750000   11297.212500                  0.007149
max    6218.000000   26216.050000                  0.177438

       daily_log_closing_value  instrument_token
count              6218.000000      6.218000e+03
mean                  0.000488      2.562650e+05
std                   0.013368      9.104009e-12
min                  -0.139038      2.562650e+05
25%                  -0.005432      2.562650e+05
50%                   0.000850      2.562650e+05
75%                   0.007123      2.562650e+05
max                   0.163341      2.562650e+05
========================================================
EDA
```





```
============================================================
STATISTICAL TESTS
1. Kurtosis Test

Kurtosis = 12.40.
Outcome:
    Distribution is Leptokurtic (fat tails) - meaning there is a higher probability of
     ↪  extreme outcomes and volatility clustering is likely present. Thus, our project
     ↪  will adopt the below approach:
    1. Monte-Carlo: During implementation, we will incorporate fat-tailed innovations
     ↪  (e.g., Student-t) to preserve tail behavior.
    2. XGBoost: No winsorization of extremes. Outliers shall be retained as-is as they
     ↪  represent genuine market behavior rather than pure noise.
    3. Risk Metrics: Conditional Value at Risk (CVaR) - over VaR - will be the preferred
     ↪  metric of choice to ensure sensitivity to tail risk.

Kurtosis sanity check:
Pass

2. Skewness Test

Skewness = -0.50.
Outcome:
    Distribution is negatively skewed (left-skewed) - indicating a heavier left tail and
     ↪  a higher frequency of large negative returns relative to positive extremes.
    This highlights pronounced downside asymmetry and elevated crash risk. Accordingly,
     ↪  the project will adopt the following approach:
    1. Monte-Carlo: Consider asymmetric or skew-aware innovations (e.g., skew-t), or
     ↪  focus explicitly on left-tail behavior in simulations.
    2. XGBoost: Preserve negative return extremes without winsorization, and prioritize
     ↪  downside-sensitive features (e.g., downside volatility, drawdown-related
     ↪  measures).
    3. Risk Metrics: Emphasize left-tail CVaR as the primary risk metric to ensure
     ↪  sensitivity to downside tail risk.

Skewness sanity check:
Pass

3. Normality Rejection Tests (Jarque-Bera and Anderson-Darling)
3.1. Jarque-Bera Test: H0: Returns are normally distributed
Jarque-Bera Test Results: JB-Statistic = 39994.87 and P-Value = 0.0
Reject H0: Returns are not normally distributed. Accordingly, the project will adopt the
 ↪  following approach:
1. Monte-Carlo: Gaussian Monte-Carlo is explicitly rejected.
2. XGBoost: Extreme observations will be retained without winsorization and feature
 ↪  engineering will explicitly capture nonlinear and regime-dependent behavior,
 ↪  including rolling volatility and downside-sensitive features.
3. Risk Metrics: CVaR will be emphasized over VaR to ensure sensitivity to tail risk.

3.2. Anderson-Darling Test: H0: Returns are normally distributed
Anderson-Darling Test Results: AD-Statistic = 96.40 and AD-Critical Value @5%
 ↪  Significance = 0.786
Reject H0: Returns are not normally distributed. Project approach remains same as
 ↪  outlined in JB above.

4. Volatility Clustering Tests (ACF and Ljung-Box)
4.1. ACF on squared log returns
```



```
4.2. Ljung-Box on squared log returns
LB Test:
        lb_stat  lb_pvalue
10  1919.593515        0.0
20  2471.621175        0.0
40  2913.596673        0.0
```

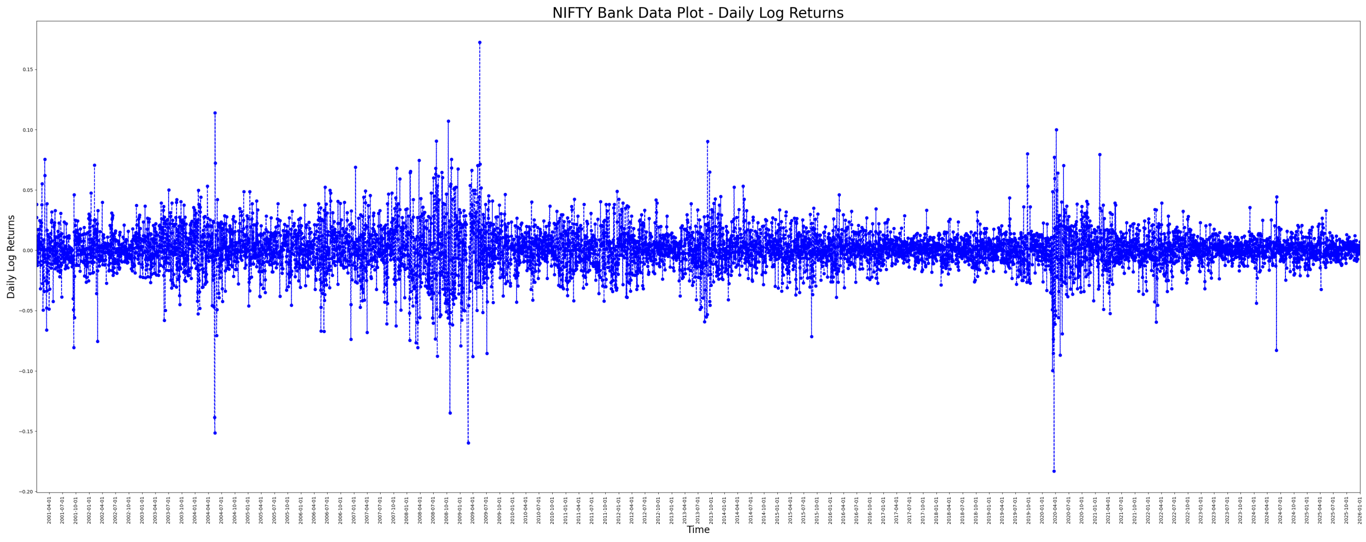
Conclusion - NIFTY50

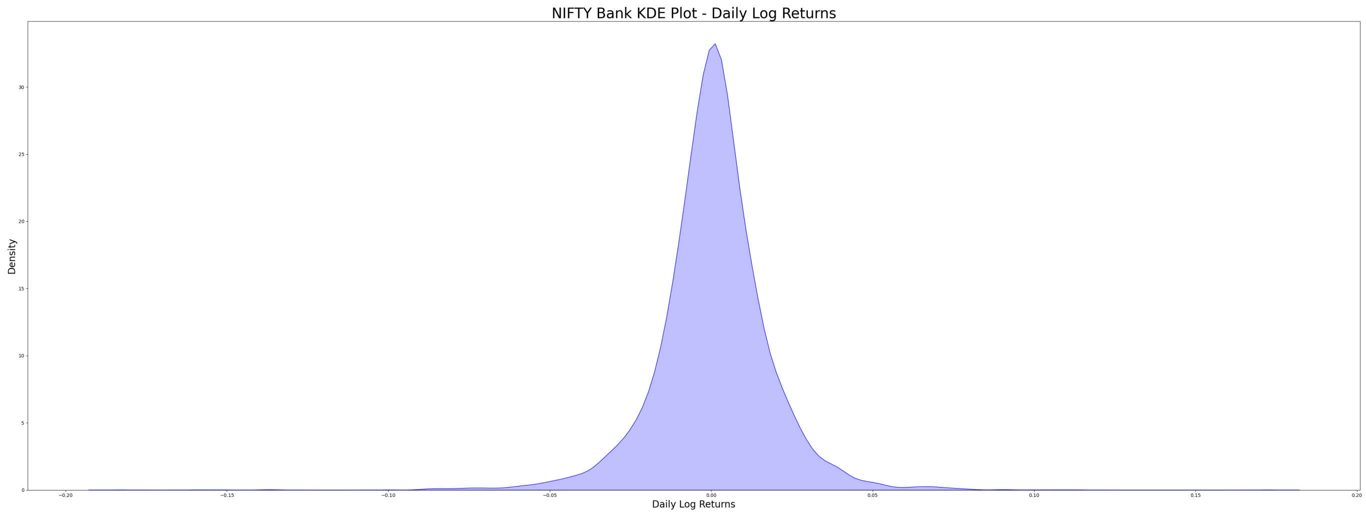
Statistical diagnostics indicate that NIFTY 50 daily log returns are strongly non-Gaussian, exhibiting pronounced fat tails, negative skewness, and persistent volatility clustering. Normality is decisively rejected by both Jarque–Bera and Anderson–Darling tests, while autocorrelation in squared returns confirms time-varying variance dynamics. Accordingly, Gaussian Monte-Carlo is inappropriate as a standalone risk model, extreme observations are retained for machine learning, volatility-aware features are emphasized, and downside-focused risk measures such as CVaR are prioritized.

2.2.2. EDA and Tests on NIFTYBank data

```
SUMMARY STATISTICS
NIFTYBank dataset - Top 5 and Bottom 5 rows:
   SNo.        date  closing_value  daily_closing_pct_change  \
0     1  2001-01-02         1015.7                  0.038654
1     2  2001-01-03         1043.7                  0.027567
2     3  2001-01-04         1034.6                 -0.008719
3     4  2001-01-05         1046.9                  0.011889
4     5  2001-01-08         1033.9                 -0.012418

   daily_log_closing_value  instrument_token trading_symbol
0                 0.037926            260105     NIFTY BANK
1                 0.027194            260105     NIFTY BANK
2                -0.008757            260105     NIFTY BANK
3                 0.011819            260105     NIFTY BANK
4                -0.012495            260105     NIFTY BANK
      SNo.        date  closing_value  daily_closing_pct_change  \
6203  6204  2025-12-26       59011.35                 -0.002910
6204  6205  2025-12-29       58932.35                 -0.001339
6205  6206  2025-12-30       59171.25                  0.004054
6206  6207  2025-12-31       59581.85                  0.006939
6207  6208  2026-01-01       59711.55                  0.002177

      daily_log_closing_value  instrument_token trading_symbol
6203                -0.002915            260105     NIFTY BANK
6204                -0.001340            260105     NIFTY BANK
6205                 0.004046            260105     NIFTY BANK
6206                 0.006915            260105     NIFTY BANK
6207                 0.002174            260105     NIFTY BANK
NIFTYBank summary statistics:
              SNo.  closing_value  daily_closing_pct_change  \
count  6208.000000    6208.000000               6208.000000
mean   3104.500000   18086.633346                  0.000820
std    1792.239567   15660.122095                  0.017698
min       1.000000     743.700000                 -0.167340
25%    1552.750000    5036.062500                 -0.007354
50%    3104.500000   11843.000000                  0.000830
75%    4656.250000   27454.600000                  0.009163
max    6208.000000   59777.200000                  0.188146

       daily_log_closing_value  instrument_token
count              6208.000000      6.208000e+03
mean                  0.000662      2.601050e+05
std                   0.017737      9.247868e-12
min                  -0.183130      2.601050e+05
25%                  -0.007381      2.601050e+05
50%                   0.000830      2.601050e+05
75%                   0.009121      2.601050e+05
```

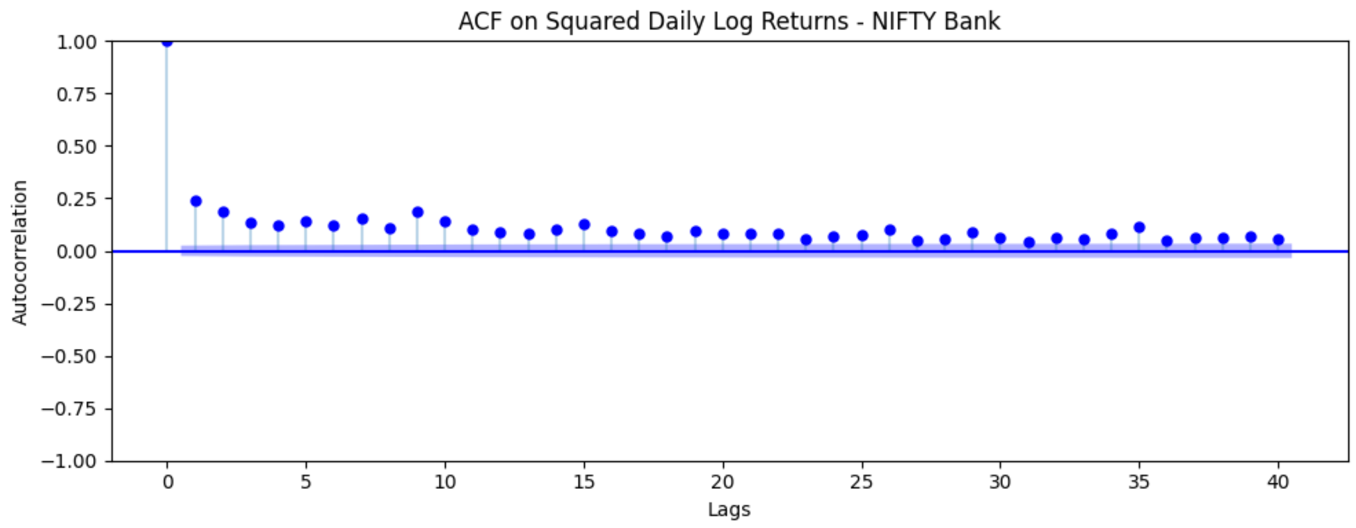
```
max                       0.172394      2.601050e+05
========================================================
EDA
```





```
========================================================
STATISTICAL TESTS
1. Kurtosis Test

Kurtosis = 9.35.
Outcome:
    Distribution is Leptokurtic (fat tails) - meaning there is a higher probability of
     ↪  extreme outcomes and volatility clustering is likely present. Thus, our project
     ↪  will adopt the below approach:
    1. Monte-Carlo: During implementation, we will incorporate fat-tailed innovations
     ↪  (e.g., Student-t) to preserve tail behavior.
    2. XGBoost: No winsorization of extremes. Outliers shall be retained as-is as they
     ↪  represent genuine market behavior rather than pure noise.
    3. Risk Metrics: Conditional Value at Risk (CVaR) - over VaR - will be the preferred
     ↪  metric of choice to ensure sensitivity to tail risk.

Kurtosis sanity check:
Pass

2. Skewness Test

Skewness = -0.44.
```

```
Outcome:
    Distribution is negatively skewed (left-skewed) - indicating a heavier left tail and
     ↪  a higher frequency of large negative returns relative to positive extremes.
    This highlights pronounced downside asymmetry and elevated crash risk. Accordingly,
     ↪  the project will adopt the following approach:
    1. Monte-Carlo: Consider asymmetric or skew-aware innovations (e.g., skew-t), or
     ↪  focus explicitly on left-tail behavior in simulations.
    2. XGBoost: Preserve negative return extremes without winsorization, and prioritize
     ↪  downside-sensitive features (e.g., downside volatility, drawdown-related
     ↪  measures).
    3. Risk Metrics: Emphasize left-tail CVaR as the primary risk metric to ensure
     ↪  sensitivity to downside tail risk.

Skewness sanity check:
Pass

3. Normality Rejection Tests (Jarque-Bera and Anderson-Darling)
3.1. Jarque-Bera Test: H0: Returns are normally distributed
Jarque-Bera Test Results: JB-Statistic = 22781.63 and P-Value = 0.0
Reject H0: Returns are not normally distributed. Accordingly, the project will adopt the
 ↪  following approach:
1. Monte-Carlo: Gaussian Monte-Carlo is explicitly rejected.
2. XGBoost: Extreme observations will be retained without winsorization and feature
 ↪  engineering will explicitly capture nonlinear and regime-dependent behavior,
 ↪  including rolling volatility and downside-sensitive features.
3. Risk Metrics: CVaR will be emphasized over VaR to ensure sensitivity to tail risk.

3.2. Anderson-Darling Test: H0: Returns are normally distributed
Anderson-Darling Test Results: AD-Statistic = 88.19 and AD-Critical Value @5%
 ↪  Significance = 0.786
Reject H0: Returns are not normally distributed. Project approach remains same as
 ↪  outlined in JB above.

4. Volatility Clustering Tests (ACF and Ljung-Box)
4.1. ACF on squared log returns
```



```
4.2. Ljung-Box on squared log returns
LB Test:
        lb_stat  lb_pvalue
10  1566.225637        0.0
```

```
20  2115.801638        0.0
40  2758.723772        0.0
```

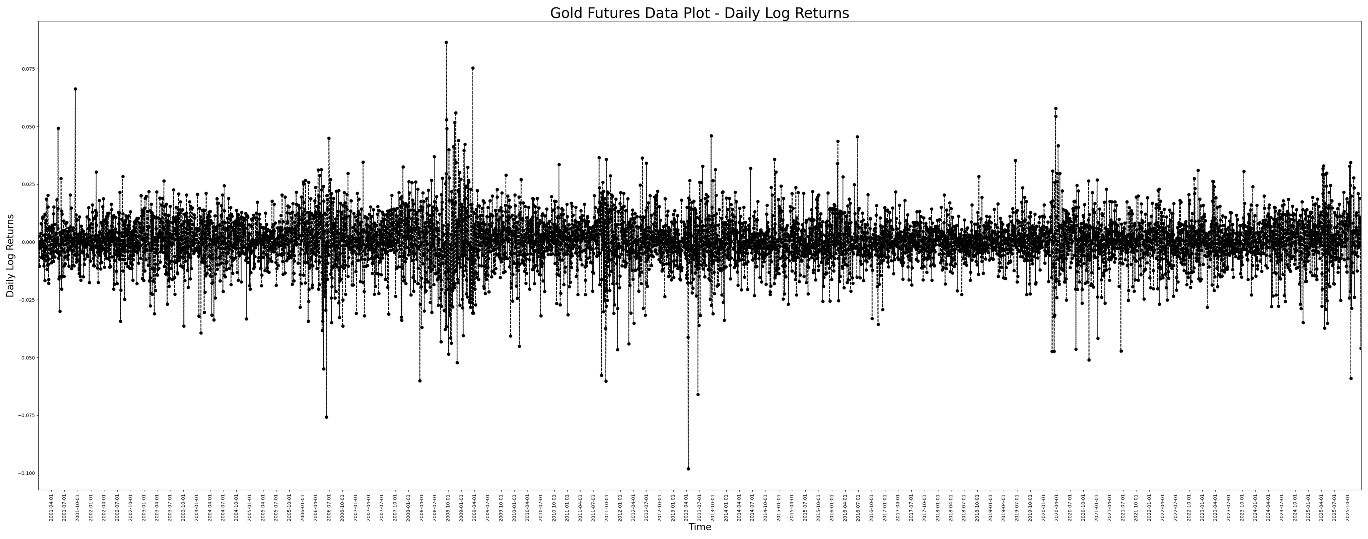
Conclusion - NIFTY Bank

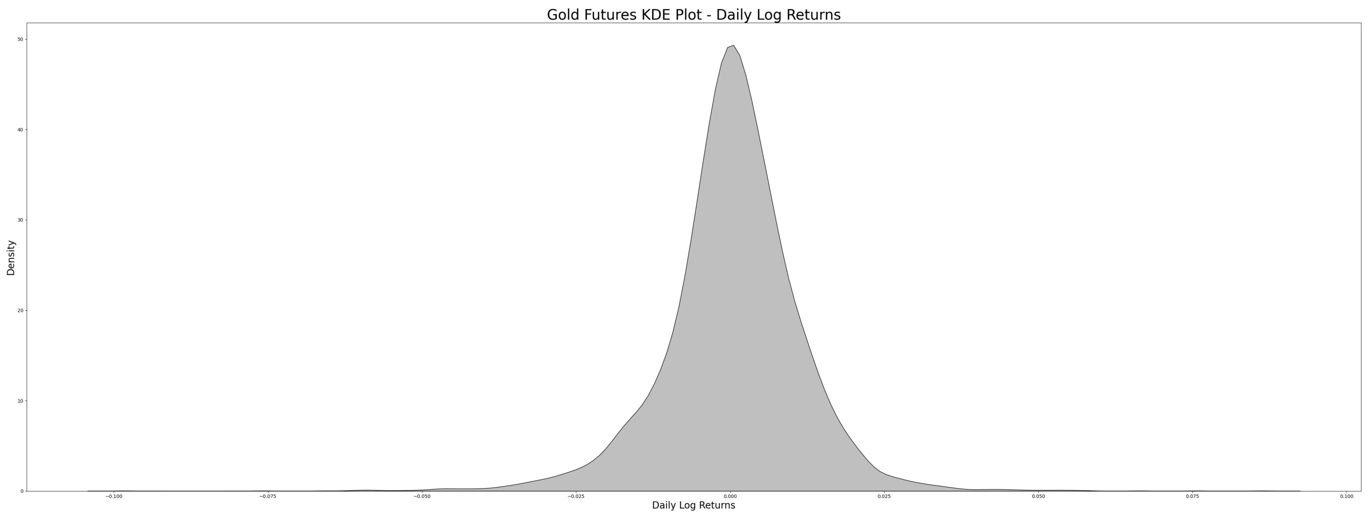
Statistical diagnostics indicate that NIFTY Bank daily log returns are strongly non-Gaussian, characterized by pronounced fat tails, clear negative skewness, and highly persistent volatility clustering. Excess kurtosis (9.35) confirms a materially elevated probability of extreme outcomes, while negative skewness (-0.44) highlights asymmetric downside risk and heightened crash sensitivity. Normality is decisively rejected by both Jarque–Bera and Anderson–Darling tests, eliminating Gaussian assumptions. Autocorrelation in squared returns, reinforced by Ljung–Box test rejections across multiple horizons, provides strong evidence of time-varying and persistent volatility dynamics. Accordingly, Gaussian Monte Carlo is unsuitable as a standalone risk framework; simulations must incorporate fat-tailed and potentially skewed innovations, extreme observations should be preserved for machine learning, volatility- and downside-aware features are central to XGBoost modeling, and left-tail-focused risk measures such as CVaR are prioritized over VaR to accurately capture systemic and crash-related risk embedded in NIFTY Bank returns.

2.2.3. EDA and Tests on Gold Futures (proxy for Gold spot) data

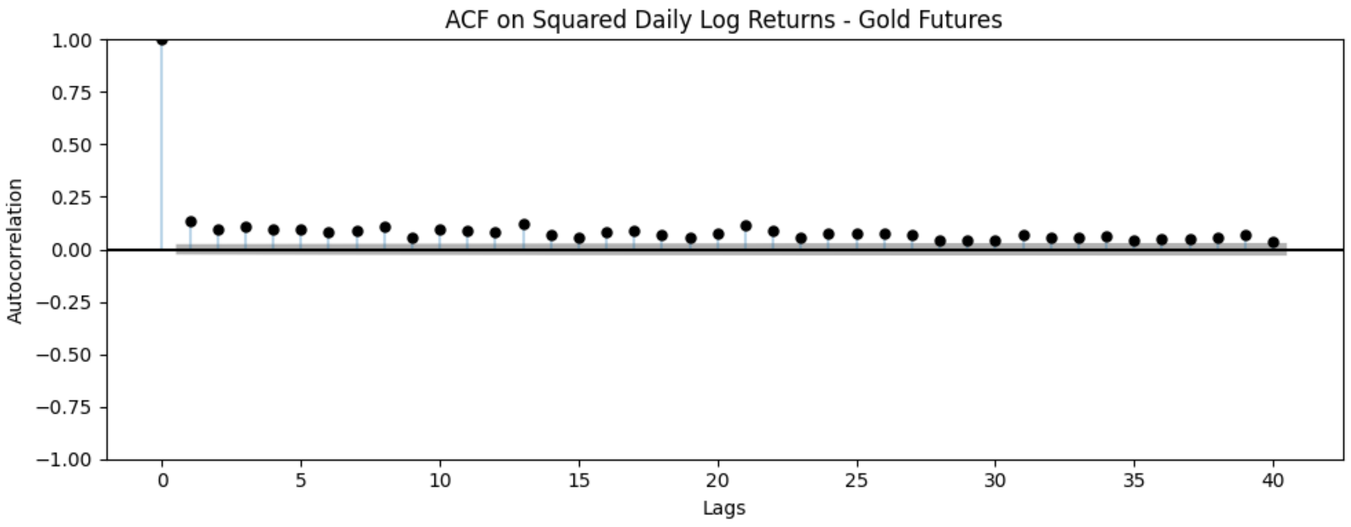
```
SUMMARY STATISTICS
Gold Futures dataset - Top 5 and Bottom 5 rows:
   SNo.        date  closing_value  daily_closing_pct_change  \
0     1  2001-01-03     268.000000                 -0.001490   
1     2  2001-01-04     267.299988                 -0.002612   
2     3  2001-01-05     268.000000                  0.002619   
3     4  2001-01-08     268.000000                  0.000000   
4     5  2001-01-09     267.500000                 -0.001866   

   daily_log_closing_value                    exhange_name trading_symbol
0                -0.001491  COMEX GOLD FUTURES (CME GROUP)           GC=F
1                -0.002615  COMEX GOLD FUTURES (CME GROUP)           GC=F
2                 0.002615  COMEX GOLD FUTURES (CME GROUP)           GC=F
3                 0.000000  COMEX GOLD FUTURES (CME GROUP)           GC=F
4                -0.001867  COMEX GOLD FUTURES (CME GROUP)           GC=F
      SNo.        date  closing_value  daily_closing_pct_change  \
6268  6269  2025-12-24    4480.600098                 -0.000491   
6269  6270  2025-12-26    4529.100098                  0.010824   
6270  6271  2025-12-29    4325.100098                 -0.045042   
6271  6272  2025-12-30    4370.100098                  0.010404   
6272  6273  2025-12-31    4325.600098                 -0.010183   

      daily_log_closing_value                    exhange_name trading_symbol
6268                -0.000491  COMEX GOLD FUTURES (CME GROUP)           GC=F
6269                 0.010766  COMEX GOLD FUTURES (CME GROUP)           GC=F
6270                -0.046088  COMEX GOLD FUTURES (CME GROUP)           GC=F
6271                 0.010351  COMEX GOLD FUTURES (CME GROUP)           GC=F
6272                -0.010235  COMEX GOLD FUTURES (CME GROUP)           GC=F
Gold Futures summary statistics:
              SNo.  closing_value  daily_closing_pct_change  \
count  6273.000000    6273.000000               6273.000000   
mean   3137.000000    1264.945783                  0.000504   
std    1811.003451     725.167493                  0.011000   
min       1.000000     255.100006                 -0.093538   
25%    1569.000000     662.900024                 -0.004762   
50%    3137.000000    1253.099976                  0.000529   
75%    4705.000000    1690.800049                  0.006348   
max    6273.000000    4529.100098                  0.090277   
```

```
       daily_log_closing_value
count              6273.000000
mean                  0.000443
std                   0.011012
min                  -0.098206
25%                  -0.004773
50%                   0.000529
75%                   0.006328
max                   0.086432
========================================================
EDA
Date datatype:object
```





```
========================================================
STATISTICAL TESTS
1. Kurtosis Test

Kurtosis = 5.14.
Outcome:
    Distribution is Leptokurtic (fat tails) - meaning there is a higher probability of
     ↪  extreme outcomes and volatility clustering is likely present. Thus, our project
     ↪  will adopt the below approach:
    1. Monte-Carlo: During implementation, we will incorporate fat-tailed innovations
     ↪  (e.g., Student-t) to preserve tail behavior.
    2. XGBoost: No winsorization of extremes. Outliers shall be retained as-is as they
     ↪  represent genuine market behavior rather than pure noise.
    3. Risk Metrics: Conditional Value at Risk (CVaR) - over VaR - will be the preferred
     ↪  metric of choice to ensure sensitivity to tail risk.

Kurtosis sanity check:
Pass

2. Skewness Test

Skewness = -0.32.
Outcome:
    Distribution is negatively skewed (left-skewed) - indicating a heavier left tail and
     ↪  a higher frequency of large negative returns relative to positive extremes.
    This highlights pronounced downside asymmetry and elevated crash risk. Accordingly,
     ↪  the project will adopt the following approach:
    1. Monte-Carlo: Consider asymmetric or skew-aware innovations (e.g., skew-t), or
     ↪  focus explicitly on left-tail behavior in simulations.
    2. XGBoost: Preserve negative return extremes without winsorization, and prioritize
     ↪  downside-sensitive features (e.g., downside volatility, drawdown-related
     ↪  measures).
    3. Risk Metrics: Emphasize left-tail CVaR as the primary risk metric to ensure
     ↪  sensitivity to downside tail risk.

Skewness sanity check:
Pass

3. Normality Rejection Tests (Jarque-Bera and Anderson-Darling)
3.1. Jarque-Bera Test: H0: Returns are normally distributed
Jarque-Bera Test Results: JB-Statistic = 6998.23 and P-Value = 0.0
Reject H0: Returns are not normally distributed. Accordingly, the project will adopt the
 ↪  following approach:
1. Monte-Carlo: Gaussian Monte-Carlo is explicitly rejected.
2. XGBoost: Extreme observations will be retained without winsorization and feature
 ↪  engineering will explicitly capture nonlinear and regime-dependent behavior,
 ↪  including rolling volatility and downside-sensitive features.
3. Risk Metrics: CVaR will be emphasized over VaR to ensure sensitivity to tail risk.

3.2. Anderson-Darling Test: H0: Returns are normally distributed
Anderson-Darling Test Results: AD-Statistic = 57.62 and AD-Critical Value @5%
 ↪  Significance = 0.786
Reject H0: Returns are not normally distributed. Project approach remains same as
 ↪  outlined in JB above.

4. Volatility Clustering Tests (ACF and Ljung-Box)
4.1. ACF on squared log returns
```



```
4.2. Ljung-Box on squared log returns
LB Test:
        lb_stat      lb_pvalue
10   601.575222  8.095121e-123
20  1019.233950  3.085422e-203
40  1538.148177  5.682211e-297
```

Conclusion - Gold Futures (COMEX Proxy for Spot Gold)

Statistical diagnostics indicate that Gold Futures daily log returns are distinctly non-Gaussian, exhibiting moderate but economically meaningful fat tails, mild negative skewness, and persistent—though comparatively smoother—volatility clustering. Excess kurtosis (5.14) confirms an elevated probability of extreme outcomes relative to a normal distribution, while negative skewness (-0.32) highlights asymmetric downside risk, albeit less severe than in equity indices. Normality is decisively rejected by both Jarque–Bera and Anderson–Darling tests, ruling out Gaussian assumptions for simulation-based risk modeling. Autocorrelation in squared returns, reinforced by Ljung–Box test rejections across multiple horizons, provides strong evidence of time-varying volatility dynamics. Accordingly, Gaussian Monte Carlo is inappropriate for Gold Futures, simulations must incorporate fat-tailed (and potentially skewed) innovations, extreme observations should be retained for machine learning models, volatility-aware and downside-sensitive features are emphasized in XGBoost, and CVaR is preferred over VaR to ensure adequate capture of tail risk—even in an asset traditionally perceived as a defensive or stabilizing component.

2.2.4. Cross-Asset Pearson Correlation Tests

As established in the introduction to this section, we perform cross-asset Pearson Correlation tests to justify the use of multivariate risk modelling in our Monte-Carlo simulations. Hence, this sub-section starts with consolidating the 3 separate datasets into one master dataset (this will also act as the master dataset for implementation of Monte-Carlo approaches) and then, performs the Pearson Correlation tests on each of the 3 observations (NIFTY50, NIFTY Bank and Gold Futures Returns) separately.

```
Number of records in raw NIFTY50 dataset: 6218
Number of records in raw NIFTYBank dataset: 6208
Number of records in raw Gold Futures dataset: 6273
Final Master for Bootstrap Monte-Carlo:
            NIFTY50_RETURNS  NIFTYBank_RETURNS  GOLD_FUTURES_RETURNS
date
2001-01-03         0.015139           0.027194             -0.001491
2001-01-04         0.012621          -0.008757             -0.002615
2001-01-05         0.014878           0.011819              0.002615
2001-01-08        -0.013655          -0.012495              0.000000
2001-01-09         0.001832          -0.005820             -0.001867
```

```
Loaded 5979 records to csv.
NaN_s in final dataset: 0
PEARSON CORRELATION - NIFTY Bank and NIFTY 50
Strong positive linear correlation between NIFTY Bank and NIFTY 50 returns. Result:
 ↪  0.8478377523287053
========================================================
PEARSON CORRELATION - NIFTY Bank and Gold Futures
No/ negligible linear correlation between NIFTY Bank and Gold Futures returns. Result:
 ↪  0.043326812721422955
========================================================
PEARSON CORRELATION - NIFTY 50 and Gold Futures
No/ negligible linear correlation between NIFTY 50 and Gold Futures returns. Result:
 ↪  0.06372749770224348
```

2.2.5. Cross-Asset Pearson Tests Conclusion

The Pearson correlation analysis indicates a strong and statistically meaningful positive linear relationship between NIFTY Bank and NIFTY 50 returns, reflecting the close structural and sectoral linkages between the banking index and the broader equity market. This result is expected, given the significant weight of financial institutions within the NIFTY 50 and the shared exposure of both indices to macroeconomic and market-wide risk factors.

In contrast, the Pearson correlation tests reveal no statistically significant linear correlation between Gold Futures and either NIFTY Bank or NIFTY 50 returns. This suggests that, over the sample period, gold exhibits weak or negligible linear co-movement with Indian equity markets, consistent with its widely documented role as a diversification asset and potential hedge during periods of equity market stress.

Implications for Multivariate Risk Modelling During Monte-Carlo Implementation:

Despite the absence of statistically significant linear correlation between gold and equity returns, the results strongly support the use of multivariate risk modeling for portfolio construction.

- The strong equity–equity correlation between NIFTY Bank and NIFTY 50 makes a multivariate framework essential, as portfolio risk is materially influenced by their joint behavior and covariance structure.
- The near-zero linear correlation between gold and equities does not imply independence, nor does it preclude co-movement during extreme market conditions. Financial asset relationships are often non-linear, time-varying, and regime-dependent, particularly during periods of stress.
- Multivariate risk models—whether Gaussian or bootstrap-based—allow the portfolio construction process to incorporate cross-asset dependence, diversification effects, and joint tail behavior, which cannot be captured through univariate risk estimation.

Accordingly, modeling asset returns within a multivariate Monte-Carlo framework is both methodologically justified and practically necessary. This approach ensures that portfolio risk estimates reflect the observed dependence structure among assets while allowing diversification benefits from assets such as gold to be appropriately quantified within the optimization process.

2.2.6. EDA and Tests Conclusion:

(NIFTY 50, NIFTY Bank, Gold Futures)

Across equities and commodities, the statistical evidence consistently shows that **asset returns deviate materially from Gaussian assumptions**. All three assets exhibit fat tails, downside asymmetry, and time-varying volatility, with the strength of these effects differing by asset class. NIFTY Bank displays the most severe tail and crash risk, NIFTY 50 shows structurally non-Gaussian behavior typical of broad equity markets, and Gold—while comparatively more stable—still exhibits meaningful departures from normality. These findings directly shape the modeling strategy for this project.

Implications for Monte-Carlo Simulation

- **Gaussian Monte-Carlo is rejected as a standalone approach** across all assets, as it fails to capture observed tail risk and volatility dynamics.
- Simulations will incorporate **fat-tailed innovations** (e.g., Student-t), with flexibility to allow for **asymmetry where downside risk is pronounced**, particularly for equity indices.
- Monte-Carlo outputs will be interpreted as **risk-scenario generators rather than precise distributional forecasts**, recognizing regime dependence and structural breaks in financial markets.
- Gold simulations, while still non-Gaussian, will reflect **relatively lower tail severity**, supporting its role as a stabilizing portfolio component without assuming normality.

Implications for XGBoost and Machine Learning

- **Extreme observations are retained** across all assets; no winsorization or truncation is applied, as extremes represent genuine market behavior.
- Feature engineering emphasizes **volatility-aware and downside-sensitive variables**, including rolling volatility, lagged returns, and drawdown-related measures.
- Volatility clustering evidenced by ACF and Ljung-Box tests justifies the use of **lagged and rolling features** to capture persistence and regime effects.
- XGBoost is positioned as a **complementary, data-driven alternative** to simulation models, particularly suited to capturing nonlinearities and cross-asset differences in risk behavior.

Implications for Risk Metrics and Portfolio Evaluation

- **CVaR is prioritized over VaR** as the primary risk metric across all assets due to its sensitivity to tail losses.
- Downside-focused risk evaluation is emphasized, especially for NIFTY Bank, where crash risk and tail dependence are structurally higher.
- Risk comparisons across assets explicitly account for **differences in tail behavior**, rather than relying solely on volatility or standard deviation.
- Gold's contribution to portfolios is assessed not by assumed normality, but by its **empirically observed tail characteristics and volatility persistence**, reinforcing its diversification role on a risk-adjusted basis.

Overall Project Positioning

Taken together, the diagnostics support a **hybrid risk-modeling framework**:
- **Non-Gaussian Monte-Carlo** for scenario-based risk exploration,
- **XGBoost** for capturing nonlinear, volatility-driven dynamics, and
- **CVaR-focused evaluation** for consistent, downside-aware risk assessment.

This integrated approach ensures that portfolio risk is evaluated using models and metrics that are **aligned with observed market behavior**, rather than imposed theoretical assumptions.

Section 3: Feature Engineering for XGBoost

Based on the findings from section 2, this section focusses on undertaking feature engineering - across all 3 datasets - to generate analysis-ready files. This section starts by outlining guiding principles for identification of the engineered features and then proceeds to implement the same.

XGBoost Feature Engineering Guiding Principles

Features Identified Based on Statistical Tests:

- **Return-based Features:** Statistical diagnosis suggest Non-Gaussian returns and volatility clustering meaning that simple linear/constant-variance assumptions are insufficient. Based on this finding, return-based features need to include lagged and rolling features to capture persistence and regime shifts and thus, the following features will be implemented:
  - Lagged Returns:
    - Lag 1: r_{t-1}
    - Lag 2: r_{t-2}
    - Lag 3: r_{t-3}
    - Lag 4: r_{t-4}
    - Lag 5: r_{t-5}
    - Lag 10: r_{t-10}
    - Lag 20: r_{t-20}
  - Rolling Mean Returns:
    - Rolling Mean of Returns over:
      - 5 days
      - 10 days
      - 20 days
  - Return Momentum:
    - Cumulative Returns over:
      - 5 days
      - 10 days
      - 20 days
  - Volatility Persistence Proxies:
    - Squared Returns: r_t^2
    - Lagged Squared Returns:
      - Squared Lag 1: r_{t-1}^2
      - Squared Lag 5: r_{t-5}^2
      - Squared Lag 10: r_{t-10}^2
    - Rolling Mean of Squared Returns over:
      - 5 days
      - 10 days
- **Volatility Clustering Features:** Statistical diagnosis - directly motivated by ACF and Ljung-Box tests - suggest presence of time-varying volatility (with strength varying across the 3 asset classes). Hence, the following features will be implemented (with strong focus on volatility ratios - which will help XGBoost identify and handle volatility regime shifts):
  - Rolling Volatility:
    - Rolling standard deviation of returns over:
      - 5 days
      - 10 days
      - 20 days
      - 40 days
      - 60 days
  - Lagged Volatility (over 20 days):
    - Lag 1: $\sigma_{20,t-1}$
    - Lag 5: $\sigma_{20,t-5}$
    - Lag 10: $\sigma_{20,t-10}$
  - Volatility Ratios (for Regime Detection):

    - Short v/s Medium (near-term stress): $\sigma_{5,t}/\sigma_{20,t}$
    - Medium v/s Long (regime shift): $\sigma_{20,t}/\sigma_{60,t}$

- **Downside Sensitive Features:** Statistical diagnosis identified negative skew across all 3 asset classes. This explicilty warrants the presence of dedicated downside sensitive features for XGBoost to be correctly modelled. Hence, the following features will be implemented:
  - Negative Return Indicators:
    - Binary Downside Flag: $I(r_t < 0)$
    - Severe Downside Flag: $I(r_t < Q^{(20)}_{0.05,t})$ where $Q^{(20)}_{0.05,t}$ is the 0.05 (5%) rolling quantile over a 20-day window.
  - Downside Volatility:
    - Rolling downside standard deviation (returns < 0) over:
      - 10 days
      - 20 days
  - Rolling Drawdowns:
    - Maximum drawdown over:
      - 10 days
      - 20 days
      - 60 days
- **Tail-Risk Proxies:** With the statistical diagnosis credibly establishing that all 3 datasets are Leptokurtic, it is imperative for the final dataset to have features that can allow XGBoost to learn tail dynamics indirectly. Hence, the following features will be implemented:
  - Rolling Quantiles:
    - 1% rolling quantile (20-day window)
    - 5% rolling quantile (20-day window)
  - Extreme Move Indicators:
    - Absolute Return Percentile: $I(|r_t| > Q^{(20)}_{0.95,t}(|r|))$ where $Q^{(20)}_{0.95,t}(|r|)$ is the 0.95 (95%) rolling quantile over a 20-day window.
  - Volatility Adjusted Returns:
    - Standardized Return: r_t/σ_{20}

Features Identified to Boost Model Performance: - **Regime Identification Features:** XGBoost performs best when regimes are learned, not imposed. Hence, the following features will be implemented: - *Volatility Regime Flags:* - High volatility regime: $\sigma_{20} > rollingmedian(\sigma_{20})$ - Stress regime: $\sigma_5 > \sigma_{20} AND r_t < 0$ - *Trend + Volatility Interaction:* - Return x Volatility: $r_t * \sigma_{20}$ - Downside x Volatility: $min(r_t, 0) * \sigma_{20}$ - **Calendar Features:** While these are not core drivers, they will provide critical controls to the model and hence, the following features will be implemented: - *Day-of-the-week: One-hot encoded* - *Separate month identifier column* - *End-of-month flag*

In total, 165 (55x3) features have been engineered for the XGBoost model - which is a reasonable count for XGBoost given the long daily history and the need to capture volatility regimes and tail behavior. Tree-based ensembles handle correlated predictors well, and feature importance diagnostics will be used post-fit to validate contribution and prune if needed (revised list of features shall be included (in the Appendix) - if pruning is conducted). Additionally, each feature family has been designed based on: - fat tails - negative skew - volatility clustering Which have been empirically demonstrated in our statistical diagnostics.

```
Feature engineered master data:
    SNo.       date  closing_value  daily_closing_pct_change  \
59    60 2001-03-28         1206.2                  0.024200
60    61 2001-03-29         1195.0                 -0.009285
61    62 2001-03-30         1148.2                 -0.039163
62    63 2001-04-02         1138.0                 -0.008883
63    64 2001-04-03         1149.2                  0.009842
```

```
    daily_log_closing_value  instrument_token trading_symbol exhange_name  \
59                 0.023912          256265.0       NIFTY 50         None
60                -0.009329          256265.0       NIFTY 50         None
61                -0.039951          256265.0       NIFTY 50         None
62                -0.008923          256265.0       NIFTY 50         None
63                 0.009794          256265.0       NIFTY 50         None

    return_lag_1  return_lag_2  ...  day_of_week  dow_0  dow_1  dow_2  dow_3  \
59      0.013851      0.000172  ...            2      0      0      1      0
60      0.023912      0.013851  ...            3      0      0      0      1
61     -0.009329      0.023912  ...            4      0      0      0      0
62     -0.039951     -0.009329  ...            0      1      0      0      0
63     -0.008923     -0.039951  ...            1      0      1      0      0

    dow_4  dow_5  dow_6  month_identifier  end_of_month_flag
59      0      0      0                 3                  0
60      0      0      0                 3                  0
61      1      0      0                 3                  0
62      0      0      0                 4                  0
63      0      0      0                 4                  0

[5 rows x 62 columns]
Loaded 18255 records to csv.
```

Section 4: Portfolio Construction Rule - Maximizing Sharpe Ratio

A **portfolio construction rule** defines how expected returns and risk estimates are translated into portfolio weights and hence, in the pipeline, this is **the most critical** step since it establishes a common portfolio contruction baseline for both Monte-Carlo and XGBoost implementations - there by - allowing the risk profiles (generated from each) to dictate the evaluation. For this analysis, we have considered **"Maximizing Sharpe"** as the portfolio construction rule. However, there are other alternatives such as:

- **Mean Variance Optimization (Markowitz):** Focuses on maximizing portfolio variance subject to a target return (or vice versa).

  Key characteristics:

  - Requires selecting a target return r
  - Highly sensitive to estimation error in expected returns
  - Foundational but fragile in practice

  Objective:

$$min_w(w^T\Sigma w) \text{ subject to: } w^T\mu >= r; \quad \Sigma_i w_i = 1$$

- **Maximize Sharpe Ratio (Tangency Portfolio) - Selected Rule:** Focuses on maximizing portfolio risk-adjusted return by choosing weights that yeild the highest expected excess return per unit of total volatility (Sharpe Ratio).

  Key characteristics:

  - Maximizes risk-adjusted return
  - Produces a unique optimal portfolio without arbitrary targets

  Objective:

$$max_w(\frac{(w^T\mu - r_f)}{\sqrt{w^T\Sigma w}}) \text{ subject to: } w_i >= 0; \quad \Sigma_i w_i = 1$$

- **Minimum Variance Portfolio (MVP):** Focuses on minimizing total portfolio variance without regard to expected returns, producing the least volatile - fully invested portfolio.

  Key characteristics:

  - Ignores expected returns entirely
  - Extremely stable

  Objective:

  $$min_w(w^T\Sigma w) \text{ subject to: } \Sigma_i w_i = 1$$

- **Risk Parity/ Equal Risk Contribution (ERC):** Focuses on allocating weights so that each asset contributes equally to total portfolio risk, promoting balanced diversification across assets.

  Key characteristics:

  - Uses only the covariance structure
  - Popular in multi-asset institutional portfolios

  Objective:

  $$RiskContribution_i = w_i \frac{(\Sigma w)_i}{\sqrt{w^T\Sigma w}} \text{ equal for all } i$$

- **Minimum Diversification Ratio:** Focuses on minimizing the ratio of weighted individual asset volatilities to overall portfolio volatility, thereby exploiting diversification benefits from imperfect correlations.

  Key characteristics:

  - Maximizes diversification benefit
  - Closely related to risk parity

  Objective:

  $$max_w(\frac{w^T\sigma}{\sqrt{w^T\Sigma w}}) \quad \text{where: } \sigma \text{ is the vector of individual asset volatilities.}$$

- **CVaR (Expected Shortfall) Minimization:** Focuses on minimizing expected losses in the worst-case tail of the return distribution, focusing on downside risk rather than overall variance.

  Key characteristics:

  - Focuses on downside tail risk
  - Suitable for drawdown-sensitive investors

  Objective:

  $$min_w CVaR_\alpha(w)$$

- **Utility-Based Optimization (CRRA/ CARA):** Focuses on minimizing the investor's expected utility of portfolio returns, explicitly trading off return and risk according to a specified risk-aversion preference..

  Key characteristics:

  - Requires specifying investor risk aversion
  - Difficult to compare across models due to subjective parameters

  Objective:

  $$max_w \mathbb{E}[U(r_p)]$$

Implementation Pipeline: Maximize Sharpe Ratio 1. Compute portfolio expected return (scalar) as the dot product of weights and expected returns:

$$r_p = w^T \mu$$

2. Compute portfolio volatility (scalar) using the quadratic form:

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

3. Define the objective as negative Sharpe (so a minimizer can be used):

$$min_w - \frac{(r_p - r_f)}{(\sigma_p + \epsilon)}$$

4. Define the fully-invested constraint:

$$\Sigma_i w_i = 1$$

5. Define long-only bounds:

$$0 <= w_i <= 1$$

6. Initialize starting weights:

$$w_0 = \frac{1}{n}\mathbf{1} \text{ where } n = \text{number of assets}$$

7. Solve the constrained nonlinear optimization using scipy.optimize.minimize(method="SLSQP") and return optimal weights. 8. Validate solver success and revert to starting weights if failed.

Section 6: Gaussian and Non-Gaussian Monte-Carlo

This section implements two Monte-Carlo approaches: Gaussian and Non-Gaussian and covers Step 5A of the proposed methodology (see Section 1: Introduction and Summary). At each rebalance date, portfolio weights are constructed using risk estimates derived from a bootstrap-based Monte-Carlo (Non-Gaussian), as well as a Gaussian Monte-Carlo, and out-of-sample performance evaluation is conducted on realized portfolio returns over the next holding period - which are derived from expected return vector μ and covariance matrix Σ - outputs of this section.

While our statistical tests and EDA plots (please see Section 2.2 above) suggest to employ a bootstrap (historical simulation) Monte-Carlo approach - as a non-Gaussian, non-parametric risk estimation method - we have also included a Gaussian Monte-Carlo approach - for comparative purposes. The recommendation of a Non-Gaussian Monte-Carlo approach is motivated by the empirically observed leptokurtic and left-skewed nature of asset return distributions, as well as the need to preserve joint dependence and tail co-movement across assets - to retain the empirical joint return structure. Unlike parametric Monte-Carlo methods, the bootstrap approach does not impose any assumed distributional form on returns, instead resampling directly from the empirical joint return distribution observed in the historical data.

A **key point of difference between the Gaussian and Non-Gaussian (Bootstrap) approaches to Monte-Carlo** is that while Bootstrap MC performs SRSWR on the return indices to pull historical return vectors and then compute the simulation expected return vector and covariance matrix, Gaussian Monte-Carlo generates synthetic return vectors from a fitted parametric multivariate normal distribution using estimated mean and covariance parameters (from the raw returns matrix).

One **important point to note** is that since our research objective requires evalauation of risk-adjusted portfolios, our implementation approach for Monte-Carlo will focus on simulating returns - rather than forecasting prices and hence, our implementation of Monte-Carlo does not incorporate Geometric Browninan Motion (GBM) mechanics.

6.1. Non-Gaussian (Bootstrap) Monte-Carlo Function

This section focuses on the implementation of a bootstrap Monte-Carlo. Bootstrap Monte-Carlo is a non-parametric framework which relies on SRSWR to generate samples from the underlying empirical distribution. This section creates a function which will be later called during the walk-forward portfolio construction. The steps followed for implementation of bootstrap MC are as follows: 1. Allocate number of simulation runs (200,000 in our case) and horizon days (21 in our case since we are doing monthly runs). 2. Generate returns matrix from consolidated master data - this should be of size (lookback window, number of assets) where lookback window = 756 (~3-years) and number of assets = 3. 3. Initialize random number generator engine. 4. Leverage the random number generator engine to generate a (200000x21) shape matrix which will contain integers that will act as index references to the returns matrix generated above. 5. Pass the random integer matrix to the returns matrix to fetch the corresponding returns. These are the simulation returns (in a 3D matrix of shape (200000x21x3)) which will be used for mu and covariance. 6. Aggregate the simulated returns across the 21 day period to generate a matrix of shape (200000x3) - to convert daily returns to monthly. 7. Generate expected returns vector = average of all 200000 return values - shape (3,). 8. Generate covariance matrix (shape (3x3)):

$$\Sigma = \begin{bmatrix} \mathrm{Var}(x) & \mathrm{Cov}(x,y) & \mathrm{Cov}(x,z) \\ \mathrm{Cov}(y,x) & \mathrm{Var}(y) & \mathrm{Cov}(y,z) \\ \mathrm{Cov}(z,x) & \mathrm{Cov}(z,y) & \mathrm{Var}(z) \end{bmatrix} \text{ where } x = \text{NIFTY50returns}, \ y = \text{NIFTYBankreturns and } z = \text{GoldFutures}$$

6.2. Gaussian Monte-Carlo Function

This section focuses on the implementation of a Gaussian Monte-Carlo framework. Unlike the bootstrap Monte-Carlo approach described in Section 6.1, which resamples directly from the empirical return distribution, the Gaussian Monte-Carlo approach is parametric and assumes that asset returns follow a multivariate normal distribution characterized by an estimated mean vector and covariance matrix. This section creates a function that will be called during the walk-forward portfolio construction to generate risk inputs under the Gaussian assumption. The steps followed for the implementation of Gaussian Monte-Carlo are as follows:

1. Allocate the number of simulation runs (200,000 in our case) and the forward horizon in trading days (21 in our case, corresponding to a monthly holding period).
2. Generate a returns matrix from the consolidated master data - this should be of size (lookback window, number of assets) where the lookback window = 756 (~3-years) and the number of assets = 3.
3. **Estimate the daily expected return vector and daily covariance matrix from the historical returns matrix. These parameters define the multivariate normal distribution assumed for daily asset returns.**
4. Initialize random number generator engine.
5. **Leverage the random number generator engine to generate and draw synthetic daily return vectors from the estimated multivariate normal distribution. This will result in a 3D matrix of shape (200000x21x3) comprising of synthetically generated returns that follow a normal distribution.**
6. Aggregate the simulated returns across the 21 day period to generate a matrix of shape (200000x3) - to convert daily returns to monthly.
7. Generate expected returns vector = average of all 200000 return values - shape (3,).
8. Generate covariance matrix (shape (3x3)):

$$\Sigma = \begin{bmatrix} \mathrm{Var}(x) & \mathrm{Cov}(x,y) & \mathrm{Cov}(x,z) \\ \mathrm{Cov}(y,x) & \mathrm{Var}(y) & \mathrm{Cov}(y,z) \\ \mathrm{Cov}(z,x) & \mathrm{Cov}(z,y) & \mathrm{Var}(z) \end{bmatrix} \text{ where } x = \text{NIFTY50returns}, \ y = \text{NIFTYBankreturns and } z = \text{GoldFu}$$

Steps 3 and 5 above are the main points where the implementation approaches differ - between Gaussian and Non-Gaussian approaches.

Additionally, the implementation of Gaussian Monte-Carlo requires the generation of correlated Gaussian random vectors with covariance matrix Σ. Among the available methods for generating correlated Gaussian random vectors, we adopt the Cholesky decomposition due to its numerical efficiency and lower constant-factor computational cost when the covariance matrix is positive definite, which is typically the case in low-dimensional portfolio risk estimation.

Available methods: 1. Cholesky - Requires covariance matrix to be positive definite - Fastest and cleanest - Exact decomposition

$$\begin{aligned} \Sigma &= LL^\top \\ \text{Where:} & \\ L:&\ \text{Cholesky factor of the covariance matrix } \Sigma \end{aligned}$$

2. Eigen Decomposition ('eigh')

- Can handle positive semi-definite matrices
- More robust when eigenvalues are very small or near-zero
- Slightly slower than Cholesky

$$\begin{aligned} \Sigma &= Q\Lambda Q^\top \\ \text{Where:} & \\ Q:&\ \text{eigenvectors} \\ \Lambda:&\ \text{diagonal matrix of eigenvalues} \end{aligned}$$

3. Singluar Vector Decomposition ('svd')

- The most numerically robust method
- Works even when covariance is poorly conditioned
- Slowest among the three

$$\begin{aligned} \Sigma &= USV^\top \\ \text{Where:} & \\ U:&\ \text{left singular vectors} \\ S:&\ \text{diagonal matrix of singular values (square roots of eigenvalues)} \\ V:&\ \text{right singular vectors} \end{aligned}$$

Section 7: Extreme Gradient Boosting (XGBoost)

XGBoost is a supervised, non-parametric, tree-ensemble method that can capture non-linear relationships and high-order interactions without imposing linear functional form assumptions or a Gaussian error structure. As an optimized implementation of gradient-boosted decision trees, it builds an additive sequence of regression trees, where each successive tree is trained to reduce the remaining prediction error of the current ensemble through gradient- and Hessian-based updates of the loss function.

To improve computational efficiency, XGBoost is trained within a CUDA-enabled WSL environment, enabling GPU acceleration relative to CPU-only training.

Key points of note: - XGBoost selects candidate splits by maximizing a regularized gain criterion and supports multiple tree growth policies (commonly depth-wise/level-wise), enabling flexible partitioning of the feature space. - Because decision trees learn piecewise non-linear mappings, XGBoost can adapt to complex local structure and interactions that are difficult to represent with linear models. - Unlike bagging-based ensembles, boosting does not rely on bootstrap resampling; however, XGBoost can optionally subsample rows and columns as regularization. - For time-series applications, model training and validation must preserve chronology (e.g., walk-forward splits) and all engineered features must be constructed using past-only information to prevent leakage. - With appropriate feature engineering (e.g., lags, rolling statistics, volatility proxies), XGBoost can learn patterns consistent with regime shifts and volatility clustering present in financial return series.

XGBoost's learning mechanism can be summarized as: - Gradient $\rightarrow$ update direction - Hessian $\rightarrow$ update magnitude - Regularization $\rightarrow$ complexity control (reduces overfitting) - Closed-form leaf weights $\rightarrow$

efficient optimization (dramatically reduces computational burden) - for example: - learning is decomposed into independent quadratic subproblems - each leaf is solved analytically - regularization is applied directly to the solution

The implementation pipeline is as follows: - Step 1: Data load and preparation - Step 2: Generating dependent variables: forward realized risk indicators (realized volatility, realized covariance and realized correlation)

Objective: In this step, we measure how risk is actually realized after time t (ex-post) over a forward window $(t+1, t+2, ..., t+H)$ using three indicators (**realized volatility per asset**, **realized cross-asset correlation** and **realized covariance matrix - derived from volatility and correlation**), which will be subsequently fed to the model so it can learn how observed risk conditions translate into future realized risk. This will then be used by the model to forecast risk dynamics - without attempting to forecast point estimates of returns and hence, these 3 indicators will make up the dependent variables for the supervised learning dataset. - Step 3: Generate supervised (model ready) dataset

Objective: In this step, we consolidate the output (from step 2 above) with the feature engineered dataset such that the output of step 2 will be the dependent variables (that the model will forecast) based on the features (independent variables) already generated (in section 3 above). - Step 4: Walkforward and backtest

Objective: In this step (implemented in the subsequent section 8), we implement the xgboost model to generate forecast estimates (for the risk indicators) for each horizon window - while keeping the portfolio construction rule constant. Outputs from this step will allow us to capture the out-of-sample (oos) performance of the model - which we will use to measure against the oos performance of the MC approaches - to drive final conclusions.

Section 8: Walkforward Backtest Function (monthly rebalancing + monthly holding)

This section focuses on the implementation of a walk-forward backtest program with monthly rebalancing and monthly holding periods. In summary, at each rebalancing date T (month-end), a trailing window of historical daily returns available up to T are captured and passed to the mc_bootstrap, mc_gaussian and ml_xgb functions to estimate the one-period-ahead portfolio risk inputs, namely the expected return vector μ and the covariance matrix Σ.

These risk inputs are then passed to the maximize_sharpe function above (section 4: portfolio construction rule) to obtain the portfolio weights at time T. The resulting weights are held constant over the subsequent month and applied to the realized daily asset returns to compute out-of-sample portfolio returns for that holding period.

At the next rebalance date, the procedure is repeated using an updated historical window, thereby generating a time series of realized portfolio returns produced under a consistent, forward-looking decision framework.

8.1. Walkforward Backtest Program - Monte-Carlo

This section focuses on the development of the walkforward backtest program for both Gaussian and Bootstrap approaches to Monte-Carlo. The ouputs from this program will be:

1. Optimized portfolio weights - generated at each rebalance date using the portfolio construction rule defined in Section 4, with risk inputs obtained from both Gaussian and Bootstrap Monte-Carlo approaches.
2. Realized out-of-sample monthly portfolio returns - obtained by applying the optimized weights to the realized daily asset returns over the subsequent holding period and aggregating them to a monthly frequency.

```
List of assets: ['NIFTY50_RETURNS', 'NIFTYBank_RETURNS', 'GOLD_FUTURES_RETURNS'], Count:
 ↪  3
Rebalance Dates: DatetimeIndex(['2001-01-31', '2001-02-28', '2001-03-31', '2001-04-30',
               '2001-05-31', '2001-06-30', '2001-07-31', '2001-08-31',
               '2001-09-30', '2001-10-31',
               ...
               '2025-03-31', '2025-04-30', '2025-05-31', '2025-06-30',
               '2025-07-31', '2025-08-31', '2025-09-30', '2025-10-31',
               '2025-11-30', '2025-12-31'],
              dtype='datetime64[ns]', name='date', length=300, freq='ME')
Data is monotonic increasing
Gaussian and Bootstrap MC implementation complete. Data loaded into csv files.
Total time taken by MC walkforward backtest program: 1.52 minutes.
```

8.2. Walkforward Backtest Program - XGBoost

This section focuses on the development of the walkforward backtest program for the XGBoost ML model. The ouputs from this program will be:

1. Optimized portfolio weights - generated at each rebalance date using the portfolio construction rule defined in Section 4, with risk inputs obtained from both Gaussian and Bootstrap Monte-Carlo approaches.
2. Realized out-of-sample monthly portfolio returns - obtained by applying the optimized weights to the realized daily asset returns over the subsequent holding period and aggregating them to a monthly frequency.

Implementation Pipeline: 1. Generate end-of-month rebalance dates 2. Verify monotonic time ordering in all datasets 3. For each rebalance date t: Select rolling horizon window (e.g., last 36 months, ending at $t-1$) 4. Train XGBoost models (volatility + correlation) using only data $< t$ 5. Predict $\hat{\sigma}_t$ and $\hat{\rho}_t$ at t 6. Reconstruct covariance matrix:

$$\hat{\Sigma}_t = D\hat{\sigma}_t \, R\hat{\rho}_t \, D\hat{\sigma}_t$$

$$\begin{aligned} &\text{Where:} \\ D &= \text{volatility vector} \\ R &= \text{correlation matrix} \end{aligned}$$

7. Obtain expected returns μ_t (historical proxy or separate model) 8. Apply portfolio construction rule $\rightarrow weights_t$ 9. Apply weights to realized returns in $(t, t+1]$ 10. Convert log returns $\rightarrow$ simple returns 11. Store: - $weights_t$ - realized OOS $return_{t+1}$ 12. Roll forward and repeat

```
============================================================
STEP 1
Data load complete and variables initialized.
============================================================
STEP 2
Shape of realized volatility dataframe: (5572, 3)
Records in realized correlation matrix dictionary: 5572
Records in realized covariance matrix dictionary: 5572
Step 2: Generate dependent variables dataset complete.
============================================================
STEP 3
No duplicates found.
Count of common dates: 5572
Number of records in final depepndent dataset: 5572
Number of records in final independent dataset: 5572
```

```
All dates in the dependent and independent datasets are same
```

```
Rebalance Dates - Independent Dataset: [Timestamp('2001-03-31 00:00:00'),
 ↪  Timestamp('2001-04-30 00:00:00'), Timestamp('2001-05-31 00:00:00'),
 ↪  Timestamp('2001-06-30 00:00:00'), Timestamp('2001-07-31 00:00:00'),
 ↪  Timestamp('2001-08-31 00:00:00'), Timestamp('2001-09-30 00:00:00'),
 ↪  Timestamp('2001-10-31 00:00:00'), Timestamp('2001-11-30 00:00:00'),
 ↪  Timestamp('2001-12-31 00:00:00'), Timestamp('2002-01-31 00:00:00'),
 ↪  Timestamp('2002-02-28 00:00:00'), Timestamp('2002-03-31 00:00:00'),
 ↪  Timestamp('2002-04-30 00:00:00'), Timestamp('2002-05-31 00:00:00'),
 ↪  Timestamp('2002-06-30 00:00:00'), Timestamp('2002-07-31 00:00:00'),
 ↪  Timestamp('2002-08-31 00:00:00'), Timestamp('2002-09-30 00:00:00'),
 ↪  Timestamp('2002-10-31 00:00:00'), Timestamp('2002-11-30 00:00:00'),
 ↪  Timestamp('2002-12-31 00:00:00'), Timestamp('2003-01-31 00:00:00'),
 ↪  Timestamp('2003-02-28 00:00:00'), Timestamp('2003-03-31 00:00:00'),
 ↪  Timestamp('2003-04-30 00:00:00'), Timestamp('2003-05-31 00:00:00'),
 ↪  Timestamp('2003-06-30 00:00:00'), Timestamp('2003-07-31 00:00:00'),
 ↪  Timestamp('2003-08-31 00:00:00'), Timestamp('2003-09-30 00:00:00'),
 ↪  Timestamp('2003-10-31 00:00:00'), Timestamp('2003-11-30 00:00:00'),
 ↪  Timestamp('2003-12-31 00:00:00'), Timestamp('2004-01-31 00:00:00'),
 ↪  Timestamp('2004-02-29 00:00:00'), Timestamp('2004-03-31 00:00:00'),
 ↪  Timestamp('2004-04-30 00:00:00'), Timestamp('2004-05-31 00:00:00'),
 ↪  Timestamp('2004-06-30 00:00:00'), Timestamp('2004-07-31 00:00:00'),
 ↪  Timestamp('2004-08-31 00:00:00'), Timestamp('2004-09-30 00:00:00'),
 ↪  Timestamp('2004-10-31 00:00:00'), Timestamp('2004-11-30 00:00:00'),
 ↪  Timestamp('2004-12-31 00:00:00'), Timestamp('2005-01-31 00:00:00'),
 ↪  Timestamp('2005-02-28 00:00:00'), Timestamp('2005-03-31 00:00:00'),
 ↪  Timestamp('2005-04-30 00:00:00'), Timestamp('2005-05-31 00:00:00'),
 ↪  Timestamp('2005-06-30 00:00:00'), Timestamp('2005-07-31 00:00:00'),
 ↪  Timestamp('2005-08-31 00:00:00'), Timestamp('2005-09-30 00:00:00'),
 ↪  Timestamp('2005-10-31 00:00:00'), Timestamp('2005-11-30 00:00:00'),
 ↪  Timestamp('2005-12-31 00:00:00'), Timestamp('2006-01-31 00:00:00'),
 ↪  Timestamp('2006-02-28 00:00:00'), Timestamp('2006-03-31 00:00:00'),
 ↪  Timestamp('2006-04-30 00:00:00'), Timestamp('2006-05-31 00:00:00'),
 ↪  Timestamp('2006-06-30 00:00:00'), Timestamp('2006-07-31 00:00:00'),
 ↪  Timestamp('2006-08-31 00:00:00'), Timestamp('2006-09-30 00:00:00'),
 ↪  Timestamp('2006-10-31 00:00:00'), Timestamp('2006-11-30 00:00:00'),
 ↪  Timestamp('2006-12-31 00:00:00'), Timestamp('2007-01-31 00:00:00'),
 ↪  Timestamp('2007-02-28 00:00:00'), Timestamp('2007-03-31 00:00:00'),
 ↪  Timestamp('2007-04-30 00:00:00'), Timestamp('2007-05-31 00:00:00'),
 ↪  Timestamp('2007-06-30 00:00:00'), Timestamp('2007-07-31 00:00:00'),
 ↪  Timestamp('2007-08-31 00:00:00'), Timestamp('2007-09-30 00:00:00'),
 ↪  Timestamp('2007-10-31 00:00:00'), Timestamp('2007-11-30 00:00:00'),
 ↪  Timestamp('2007-12-31 00:00:00'), Timestamp('2008-01-31 00:00:00'),
 ↪  Timestamp('2008-02-29 00:00:00'), Timestamp('2008-03-31 00:00:00'),
 ↪  Timestamp('2008-04-30 00:00:00'), Timestamp('2008-05-31 00:00:00'),
 ↪  Timestamp('2008-06-30 00:00:00'), Timestamp('2008-07-31 00:00:00'),
 ↪  Timestamp('2008-08-31 00:00:00'), Timestamp('2008-09-30 00:00:00'),
 ↪  Timestamp('2008-10-31 00:00:00'), Timestamp('2008-11-30 00:00:00'),
 ↪  Timestamp('2008-12-31 00:00:00'), Timestamp('2009-01-31 00:00:00'),
 ↪  Timestamp('2009-02-28 00:00:00'), Timestamp('2009-03-31 00:00:00'),
 ↪  Timestamp('2009-04-30 00:00:00'), Timestamp('2009-05-31 00:00:00'),
 ↪  Timestamp('2009-06-30 00:00:00'), Timestamp('2009-07-31 00:00:00'),
 ↪  Timestamp('2009-08-31 00:00:00'), Timestamp('2009-09-30 00:00:00'),
 ↪  Timestamp('2009-10-31 00:00:00'), Timestamp('2009-11-30 00:00:00'),
 ↪  Timestamp('2009-12-31 00:00:00'), Timestamp('2010-01-31 00:00:00'),
 ↪  Timestamp('2010-02-28 00:00:00'), Timestamp('2010-03-31 00:00:00'),
 ↪  Timestamp('2010-04-30 00:00:00'), Timestamp('2010-05-31 00:00:00'),
 ↪  Timestamp('2010-06-30 00:00:00'), Timestamp('2010-07-31 00:00:00'),
 ↪  Timestamp('2010-08-31 00:00:00'), Timestamp('2010-09-30 00:00:00'),
 ↪  Timestamp('2010-10-31 00:00:00'), Timestamp('2010-11-30 00:00:00'),
All dates - inside independndent dataset - are monotonic increasing.
All dates - inside dependent dataset - are monotonic increasing.
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2004-07-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2004-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.03189163  0.03169231]
 [-0.03189163  1.          0.80736238]
 [ 0.03169231  0.80736238  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2004-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00952163 0.         0.        ]
 [0.         0.01052258 0.        ]
 [0.         0.         0.01289043]]
Resulting covariance matrix for run date DatetimeIndex(['2004-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.99455149e-03 -7.02963947e-05  8.55766660e-05]
 [-7.02963947e-05  2.43594480e-03  2.40924577e-03]
 [ 8.55766660e-05  2.40924577e-03  3.65559229e-03]]
Expected returns vector for run date DatetimeIndex(['2004-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.00998153 0.0006778  0.01452313]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-07-30'],
 ↪ dtype='datetime64[ns]')
[4.76601900e-01 9.71445147e-17 5.23398100e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2004-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2004-08-30'], dtype='datetime64[ns]'):
[[ 1.          0.16748965 -0.03044136]
 [ 0.16748965  1.          0.72850579]
 [-0.03044136  0.72850579  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2004-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00775328 0.         0.        ]
 [0.         0.00888719 0.        ]
 [0.         0.         0.01424942]]
Resulting covariance matrix for run date DatetimeIndex(['2004-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.26237885e-03  2.42357774e-04 -7.06261745e-05]
 [ 2.42357774e-04  1.65862521e-03  1.93737585e-03]
 [-7.06261745e-05  1.93737585e-03  4.26396358e-03]]
Expected returns vector for run date DatetimeIndex(['2004-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.01103913 0.00362237 0.01722895]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-08-30'],
 ↪ dtype='datetime64[ns]')
[0.6202327 0.        0.3797673]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2004-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2004-09-29'], dtype='datetime64[ns]'):
[[ 1.          0.02439177 -0.00188402]
 [ 0.02439177  1.          0.72382087]
 [-0.00188402  0.72382087  1.        ]]
```

```
Volatility diagonal vector for run date DatetimeIndex(['2004-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0094327  0.         0.        ]
 [0.         0.0090239  0.        ]
 [0.         0.         0.01456414]]
Resulting covariance matrix for run date DatetimeIndex(['2004-09-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.69054092e-03  3.94481949e-05 -4.91768488e-06]
 [ 3.94481949e-05  1.54718342e-03  1.80743838e-03]
 [-4.91768488e-06  1.80743838e-03  4.03016660e-03]]
Expected returns vector for run date DatetimeIndex(['2004-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.01171336 0.00422589 0.01861805]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-09-29'],
 ↪ dtype='datetime64[ns]')
[5.34591943e-01 2.67581096e-16 4.65408057e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2004-10-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2004-10-29'], dtype='datetime64[ns]'):
[[ 1.         -0.29830104 -0.01128205]
 [-0.29830104  1.          0.69686806]
 [-0.01128205  0.69686806  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2004-10-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00775342 0.         0.        ]
 [0.         0.00746324 0.        ]
 [0.         0.         0.01480485]]
Resulting covariance matrix for run date DatetimeIndex(['2004-10-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.14219482e-03 -3.27966313e-04 -2.46058902e-05]
 [-3.27966313e-04  1.05829989e-03  1.46297150e-03]
 [-2.46058902e-05  1.46297150e-03  4.16449037e-03]]
Expected returns vector for run date DatetimeIndex(['2004-10-29'],
 ↪ dtype='datetime64[ns]'):
[0.01050466 0.00802624 0.02344821]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-10-29'],
 ↪ dtype='datetime64[ns]')
[5.14548418e-01 2.62376926e-17 4.85451582e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2004-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2004-11-29'], dtype='datetime64[ns]'):
[[1.         0.1914458  0.00424252]
 [0.1914458  1.         0.7388494 ]
 [0.00424252 0.7388494  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2004-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01080309 0.         0.        ]
 [0.         0.00986277 0.        ]
 [0.         0.         0.01935005]]
Resulting covariance matrix for run date DatetimeIndex(['2004-11-29'],
 ↪ dtype='datetime64[ns]'):
[[2.45084243e-03 4.28362985e-04 1.86239994e-05]
 [4.28362985e-04 2.04275747e-03 2.96111984e-03]
 [1.86239994e-05 2.96111984e-03 7.86291012e-03]]
Expected returns vector for run date DatetimeIndex(['2004-11-29'],
 ↪  dtype='datetime64[ns]'):
[0.01241353 0.00742752 0.02086346]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-11-29'],
 ↪  dtype='datetime64[ns]')
[0.59409824 0.         0.40590176]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2004-12-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2004-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.40753001 -0.35239601]
 [-0.40753001  1.          0.92483711]
 [-0.35239601  0.92483711  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2004-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00730887 0.         0.        ]
 [0.         0.01668487 0.        ]
 [0.         0.         0.02475252]]
Resulting covariance matrix for run date DatetimeIndex(['2004-12-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00096155 -0.00089455 -0.00114755]
 [-0.00089455  0.00501093  0.00687512]
 [-0.00114755  0.00687512  0.01102837]]
Expected returns vector for run date DatetimeIndex(['2004-12-30'],
 ↪  dtype='datetime64[ns]'):
[0.0144731  0.00749787 0.02474917]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2004-12-30'],
 ↪  dtype='datetime64[ns]')
[8.08638361e-01 1.55322796e-16 1.91361639e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-01-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-01-28'], dtype='datetime64[ns]'):
[[ 1.         -0.10339632 -0.03355383]
 [-0.10339632  1.          0.8472656 ]
 [-0.03355383  0.8472656   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-01-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00731199 0.         0.        ]
 [0.         0.01329047 0.        ]
 [0.         0.         0.01826031]]
Resulting covariance matrix for run date DatetimeIndex(['2005-01-28'],
 ↪  dtype='datetime64[ns]'):
[[ 1.01583773e-03 -1.90912489e-04 -8.51214724e-05]
 [-1.90912489e-04  3.35609456e-03  3.90680341e-03]
 [-8.51214724e-05  3.90680341e-03  6.33533785e-03]]
Expected returns vector for run date DatetimeIndex(['2005-01-28'],
 ↪  dtype='datetime64[ns]'):
[0.01309787 0.00880802 0.02988222]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-01-28'],
 ↪  dtype='datetime64[ns]')
[0.66737517 0.         0.33262483]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-02-25'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-02-25'], dtype='datetime64[ns]'):
[[1.         0.34899196 0.33924821]
 [0.34899196 1.         0.79267156]
 [0.33924821 0.79267156 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-02-25'],
 ↪ dtype='datetime64[ns]'):
[[0.00787755 0.         0.        ]
 [0.         0.01024472 0.        ]
 [0.         0.         0.0191669 ]]
Resulting covariance matrix for run date DatetimeIndex(['2005-02-25'],
 ↪ dtype='datetime64[ns]'):
[[0.00130317 0.00059146 0.00107567]
 [0.00059146 0.00220404 0.00326862]
 [0.00107567 0.00326862 0.00771477]]
Expected returns vector for run date DatetimeIndex(['2005-02-25'],
 ↪ dtype='datetime64[ns]'):
[0.01114387 0.00790482 0.02702485]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-02-25'],
 ↪ dtype='datetime64[ns]')
[0.49516644 0.         0.50483356]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-03-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-03-29'], dtype='datetime64[ns]'):
[[ 1.         -0.14700441 -0.16764084]
 [-0.14700441  1.          0.8800416 ]
 [-0.16764084  0.8800416   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-03-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00670364 0.         0.        ]
 [0.         0.01314754 0.        ]
 [0.         0.         0.01866437]]
Resulting covariance matrix for run date DatetimeIndex(['2005-03-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00085384 -0.00024617 -0.00039853]
 [-0.00024617  0.0032843   0.00410312]
 [-0.00039853  0.00410312  0.00661881]]
Expected returns vector for run date DatetimeIndex(['2005-03-29'],
 ↪ dtype='datetime64[ns]'):
[0.01069379 0.00731986 0.02625275]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-03-29'],
 ↪ dtype='datetime64[ns]')
[6.87021970e-01 1.47126235e-16 3.12978030e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-04-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2005-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.33741054 -0.08215115]
 [-0.33741054  1.          0.7689622 ]
 [-0.08215115  0.7689622   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00685344 0.         0.        ]
```

```
 [0.         0.00800689 0.        ]
 [0.         0.         0.01003744]]
Resulting covariance matrix for run date DatetimeIndex(['2005-04-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00089242 -0.00035179 -0.00010737]
 [-0.00035179  0.0012181   0.00117421]
 [-0.00010737  0.00117421  0.00191425]]
Expected returns vector for run date DatetimeIndex(['2005-04-29'],
 ↪  dtype='datetime64[ns]'):
[0.00965532 0.00658055 0.02470283]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-04-29'],
 ↪  dtype='datetime64[ns]')
[3.69219825e-01 9.54097912e-18 6.30780175e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-05-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-05-27'], dtype='datetime64[ns]'):
[[ 1.         -0.21070772 -0.21350293]
 [-0.21070772  1.          0.49291578]
 [-0.21350293  0.49291578  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-05-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00755827 0.         0.        ]
 [0.         0.00858691 0.        ]
 [0.         0.         0.01252317]]
Resulting covariance matrix for run date DatetimeIndex(['2005-05-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00114255 -0.00027351 -0.00040418]
 [-0.00027351  0.0014747   0.00106012]
 [-0.00040418  0.00106012  0.0031366 ]]
Expected returns vector for run date DatetimeIndex(['2005-05-27'],
 ↪  dtype='datetime64[ns]'):
[0.00983325 0.00574626 0.02199161]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-05-27'],
 ↪  dtype='datetime64[ns]')
[0.50045348 0.         0.49954652]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-06-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-06-28'], dtype='datetime64[ns]'):
[[1.         0.28473955 0.06472434]
 [0.28473955 1.         0.62010998]
 [0.06472434 0.62010998 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-06-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00685816 0.         0.        ]
 [0.         0.00949032 0.        ]
 [0.         0.         0.01458395]]
Resulting covariance matrix for run date DatetimeIndex(['2005-06-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00089365 0.00035212 0.000123  ]
 [0.00035212 0.00171126 0.00163072]
 [0.000123   0.00163072 0.00404114]]
Expected returns vector for run date DatetimeIndex(['2005-06-28'],
 ↪  dtype='datetime64[ns]'):
[0.00728028 0.0088533  0.02537689]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-06-28'],
 ↪ dtype='datetime64[ns]')
[0.23691296 0.         0.76308704]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-07-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2005-07-29'], dtype='datetime64[ns]'):
[[1.         0.0421659  0.13884144]
 [0.0421659  1.         0.77471483]
 [0.13884144 0.77471483 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-07-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0074241  0.         0.        ]
 [0.         0.00978677 0.        ]
 [0.         0.         0.0133961 ]]
Resulting covariance matrix for run date DatetimeIndex(['2005-07-29'],
 ↪ dtype='datetime64[ns]'):
[[1.21258075e-03 6.74011898e-05 3.03783482e-04]
 [6.74011898e-05 2.10718057e-03 2.23451086e-03]
 [3.03783482e-04 2.23451086e-03 3.94802098e-03]]
Expected returns vector for run date DatetimeIndex(['2005-07-29'],
 ↪ dtype='datetime64[ns]'):
[0.00986406 0.00999009 0.02519914]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-07-29'],
 ↪ dtype='datetime64[ns]')
[0.34452917 0.         0.65547083]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-08-30'], dtype='datetime64[ns]'):
[[1.         0.05424073 0.03794676]
 [0.05424073 1.         0.80204821]
 [0.03794676 0.80204821 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00707293 0.         0.        ]
 [0.         0.01373803 0.        ]
 [0.         0.         0.0158704 ]]
Resulting covariance matrix for run date DatetimeIndex(['2005-08-30'],
 ↪ dtype='datetime64[ns]'):
[[5.00262761e-04 5.27046724e-05 4.25953032e-05]
 [5.27046724e-05 1.88733535e-03 1.74869040e-03]
 [4.25953032e-05 1.74869040e-03 2.51869648e-03]]
Expected returns vector for run date DatetimeIndex(['2005-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.01023748 0.013708   0.03084114]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-08-30'],
 ↪ dtype='datetime64[ns]')
[4.73712714e-01 9.01188846e-16 5.26287286e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2005-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-09-29'], dtype='datetime64[ns]'):
[[1.         0.20366056 0.15374599]
 [0.20366056 1.         0.87050474]
 [0.15374599 0.87050474 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-09-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00865224 0.         0.        ]
 [0.         0.01415647 0.        ]
 [0.         0.         0.01840004]]
Resulting covariance matrix for run date DatetimeIndex(['2005-09-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00149723 0.00049891 0.00048953]
 [0.00049891 0.00400811 0.00453497]
 [0.00048953 0.00453497 0.00677123]]
Expected returns vector for run date DatetimeIndex(['2005-09-29'],
 ↪  dtype='datetime64[ns]'):
[0.01011131 0.01310456 0.028407  ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-09-29'],
 ↪  dtype='datetime64[ns]')
[0.40073856 0.         0.59926144]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-10-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-10-28'], dtype='datetime64[ns]'):
[[ 1.         -0.02779221 -0.14780809]
 [-0.02779221  1.          0.83320004]
 [-0.14780809  0.83320004  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-10-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00793675 0.         0.        ]
 [0.         0.0136278  0.        ]
 [0.         0.         0.01553609]]
Resulting covariance matrix for run date DatetimeIndex(['2005-10-28'],
 ↪  dtype='datetime64[ns]'):
[[ 8.81887921e-04 -4.20842432e-05 -2.55158863e-04]
 [-4.20842432e-05  2.60003748e-03  2.46970364e-03]
 [-2.55158863e-04  2.46970364e-03  3.37918145e-03]]
Expected returns vector for run date DatetimeIndex(['2005-10-28'],
 ↪  dtype='datetime64[ns]'):
[0.00948814 0.01531433 0.02968532]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-10-28'],
 ↪  dtype='datetime64[ns]')
[0.47471481 0.         0.52528519]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2005-11-29'], dtype='datetime64[ns]'):
[[ 1.          0.27676392 -0.02688768]
 [ 0.27676392  1.          0.79750067]
 [-0.02688768  0.79750067  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2005-11-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01042058 0.         0.        ]
 [0.         0.01181324 0.        ]
 [0.         0.         0.01530515]]
Resulting covariance matrix for run date DatetimeIndex(['2005-11-29'],
 ↪  dtype='datetime64[ns]'):
[[ 2.06318216e-03  6.47327696e-04 -8.14772668e-05]
 [ 6.47327696e-04  2.65150146e-03  2.73962654e-03]
 [-8.14772668e-05  2.73962654e-03  4.45070435e-03]]
Expected returns vector for run date DatetimeIndex(['2005-11-29'],
 ↪  dtype='datetime64[ns]'):
[0.00954739 0.01307662 0.02659878]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-11-29'],
 ↪  dtype='datetime64[ns]')
[3.17271835e-01 2.51426484e-16 6.82728165e-01]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2005-12-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2005-12-30'], dtype='datetime64[ns]'):
[[1.00000000e+00 4.18477058e-01 9.95730981e-04]
 [4.18477058e-01 1.00000000e+00 7.20444679e-01]
 [9.95730981e-04 7.20444679e-01 1.00000000e+00]]
Volatility diagonal vector for run date DatetimeIndex(['2005-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01307825 0.         0.        ]
 [0.         0.01033983 0.        ]
 [0.         0.         0.01533529]]
Resulting covariance matrix for run date DatetimeIndex(['2005-12-30'],
 ↪  dtype='datetime64[ns]'):
[[2.73665152e-03 9.05429989e-04 3.19524194e-06]
 [9.05429989e-04 1.71059399e-03 1.82778912e-03]
 [3.19524194e-06 1.82778912e-03 3.76273779e-03]]
Expected returns vector for run date DatetimeIndex(['2005-12-30'],
 ↪  dtype='datetime64[ns]'):
[0.01028665 0.0128113  0.02620849]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2005-12-30'],
 ↪  dtype='datetime64[ns]')
[0.24553627 0.         0.75446373]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2006-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.07851285 -0.10115814]
 [-0.07851285  1.          0.72412813]
 [-0.10115814  0.72412813  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-01-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01359378 0.         0.        ]
 [0.         0.0093754  0.        ]
 [0.         0.         0.01550308]]
Resulting covariance matrix for run date DatetimeIndex(['2006-01-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00332623 -0.00018011 -0.00038374]
 [-0.00018011  0.00158217  0.0018945 ]
 [-0.00038374  0.0018945   0.00432622]]
Expected returns vector for run date DatetimeIndex(['2006-01-30'],
 ↪  dtype='datetime64[ns]'):
[0.0088607  0.01443964 0.02546507]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-01-30'],
 ↪  dtype='datetime64[ns]')
[0.24362045 0.06290293 0.69347662]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-02-27'], dtype='datetime64[ns]'):
[[1.         0.10925083 0.38180327]
 [0.10925083 1.         0.61453247]
 [0.38180327 0.61453247 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01042162 0.         0.        ]
 [0.         0.00960306 0.        ]
 [0.         0.         0.00994698]]
Resulting covariance matrix for run date DatetimeIndex(['2006-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00238942 0.00024054 0.00087074]
 [0.00024054 0.00202881 0.00129142]
 [0.00087074 0.00129142 0.00217673]]
Expected returns vector for run date DatetimeIndex(['2006-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.00876993 0.01737625 0.02655816]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-02-27'],
 ↪ dtype='datetime64[ns]')
[2.08166817e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-03-30'], dtype='datetime64[ns]'):
[[1.         0.37572238 0.21620342]
 [0.37572238 1.         0.69752938]
 [0.21620342 0.69752938 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01363892 0.         0.        ]
 [0.         0.01416245 0.        ]
 [0.         0.         0.01799189]]
Resulting covariance matrix for run date DatetimeIndex(['2006-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00297632 0.0011612  0.00084887]
 [0.0011612  0.0032092  0.00284379]
 [0.00084887 0.00284379 0.00517933]]
Expected returns vector for run date DatetimeIndex(['2006-03-30'],
 ↪ dtype='datetime64[ns]'):
[0.00937619 0.01784224 0.02630037]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-03-30'],
 ↪ dtype='datetime64[ns]')
[0.01753121 0.13092865 0.85154014]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-04-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2006-04-28'], dtype='datetime64[ns]'):
[[1.         0.65555924 0.50333256]
 [0.65555924 1.         0.86887223]
 [0.50333256 0.86887223 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-04-28'],
 ↪ dtype='datetime64[ns]'):
[[0.0205266  0.         0.        ]
```

```
 [0.         0.03055987 0.        ]
 [0.         0.         0.02784621]]
Resulting covariance matrix for run date DatetimeIndex(['2006-04-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00758414 0.00740207 0.00517858]
 [0.00740207 0.01681031 0.01330902]
 [0.00517858 0.01330902 0.01395741]]
Expected returns vector for run date DatetimeIndex(['2006-04-28'],
 ↪ dtype='datetime64[ns]'):
[0.0115362  0.02315375 0.02771831]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-04-28'],
 ↪ dtype='datetime64[ns]')
[2.08166817e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-05-30'], dtype='datetime64[ns]'):
[[1.         0.40412897 0.23428956]
 [0.40412897 1.         0.88189971]
 [0.23428956 0.88189971 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0229884  0.         0.        ]
 [0.         0.03388897 0.        ]
 [0.         0.         0.03006972]]
Resulting covariance matrix for run date DatetimeIndex(['2006-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01162627 0.00692643 0.00356298]
 [0.00692643 0.02526616 0.01977105]
 [0.00356298 0.01977105 0.01989214]]
Expected returns vector for run date DatetimeIndex(['2006-05-30'],
 ↪ dtype='datetime64[ns]'):
[0.01386739 0.02495956 0.02486735]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-05-30'],
 ↪ dtype='datetime64[ns]')
[0.33995944 0.         0.66004056]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-06-29'], dtype='datetime64[ns]'):
[[1.         0.3791278  0.18354172]
 [0.3791278  1.         0.80595136]
 [0.18354172 0.80595136 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.02166261 0.         0.        ]
 [0.         0.01997098 0.        ]
 [0.         0.         0.02159695]]
Resulting covariance matrix for run date DatetimeIndex(['2006-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0089161  0.00311637 0.00163152]
 [0.00311637 0.00757796 0.00660472]
 [0.00163152 0.00660472 0.00886214]]
Expected returns vector for run date DatetimeIndex(['2006-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.01024711 0.01922085 0.01709022]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-06-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.11022302e-16]
=========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-07-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-07-28'], dtype='datetime64[ns]'):
[[1.         0.08663093 0.13374218]
 [0.08663093 1.         0.58614647]
 [0.13374218 0.58614647 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-07-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01220684 0.         0.        ]
 [0.         0.0185617  0.        ]
 [0.         0.         0.02265017]]
Resulting covariance matrix for run date DatetimeIndex(['2006-07-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00327815 0.00043183 0.00081352]
 [0.00043183 0.00757981 0.00542148]
 [0.00081352 0.00542148 0.01128667]]
Expected returns vector for run date DatetimeIndex(['2006-07-28'],
 ↪ dtype='datetime64[ns]'):
[0.01041103 0.01635643 0.01378585]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-07-28'],
 ↪ dtype='datetime64[ns]')
[0.49956623 0.50043377 0.        ]
=========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-08-30'], dtype='datetime64[ns]'):
[[1.         0.45577037 0.39523196]
 [0.45577037 1.         0.78319454]
 [0.39523196 0.78319454 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01373788 0.         0.        ]
 [0.         0.01193744 0.        ]
 [0.         0.         0.01591376]]
Resulting covariance matrix for run date DatetimeIndex(['2006-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00396332 0.00156963 0.00181453]
 [0.00156963 0.00299255 0.00312445]
 [0.00181453 0.00312445 0.0053182 ]]
Expected returns vector for run date DatetimeIndex(['2006-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.01069786 0.01500751 0.01304409]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-08-30'],
 ↪ dtype='datetime64[ns]')
[3.07371513e-02 9.69262849e-01 1.89366285e-17]
=========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-09-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2006-09-29'], dtype='datetime64[ns]'):
[[1.         0.59078199 0.50618047]
 [0.59078199 1.         0.74131095]
 [0.50618047 0.74131095 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01551034 0.         0.        ]
 [0.         0.0093065  0.        ]
 [0.         0.         0.01416727]]
Resulting covariance matrix for run date DatetimeIndex(['2006-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00457085 0.00162028 0.00211333]
 [0.00162028 0.00164561 0.00185706]
 [0.00211333 0.00185706 0.00381352]]
Expected returns vector for run date DatetimeIndex(['2006-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.00882134 0.01380946 0.01428335]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-09-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.66533454e-16]
==========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-10-30'], dtype='datetime64[ns]'):
[[1.         0.31565803 0.16964611]
 [0.31565803 1.         0.61976308]
 [0.16964611 0.61976308 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01166924 0.         0.        ]
 [0.         0.00875815 0.        ]
 [0.         0.         0.01489257]]
Resulting covariance matrix for run date DatetimeIndex(['2006-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00245108 0.00058069 0.00053068]
 [0.00058069 0.00138069 0.00145506]
 [0.00053068 0.00145506 0.0039922 ]]
Expected returns vector for run date DatetimeIndex(['2006-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.00650448 0.01493869 0.0177331 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-10-30'],
 ↪ dtype='datetime64[ns]')
[1.14925430e-17 8.66235529e-01 1.33764471e-01]
==========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2006-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.1129688  -0.05702907]
 [-0.1129688   1.          0.82075071]
 [-0.05702907  0.82075071  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01076837 0.         0.        ]
 [0.         0.01568943 0.        ]
 [0.         0.         0.02405564]]
Resulting covariance matrix for run date DatetimeIndex(['2006-11-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00231916 -0.00038172 -0.00029546]
 [-0.00038172  0.00492317  0.00619534]
 [-0.00029546  0.00619534  0.01157348]]
Expected returns vector for run date DatetimeIndex(['2006-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00685021 0.01457725 0.01658385]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-11-29'],
 ↪ dtype='datetime64[ns]')
[3.35783859e-01 6.64216141e-01 3.74655289e-18]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2006-12-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2006-12-29'], dtype='datetime64[ns]'):
[[1.         0.03192501 0.22088122]
 [0.03192501 1.         0.77007085]
 [0.22088122 0.77007085 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2006-12-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01179756 0.         0.        ]
 [0.         0.01179199 0.        ]
 [0.         0.         0.02012235]]
Resulting covariance matrix for run date DatetimeIndex(['2006-12-29'],
 ↪ dtype='datetime64[ns]'):
[[2.64446535e-03 8.43847088e-05 9.96284063e-04]
 [8.43847088e-05 2.64196835e-03 3.47176190e-03]
 [9.96284063e-04 3.47176190e-03 7.69326843e-03]]
Expected returns vector for run date DatetimeIndex(['2006-12-29'],
 ↪ dtype='datetime64[ns]'):
[0.00830558 0.01574454 0.02104148]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2006-12-29'],
 ↪ dtype='datetime64[ns]')
[0.18834469 0.72760079 0.08405452]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-01-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-01-29'], dtype='datetime64[ns]'):
[[1.         0.17404902 0.1936879 ]
 [0.17404902 1.         0.75330335]
 [0.1936879  0.75330335 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-01-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01237838 0.         0.        ]
 [0.         0.01340031 0.        ]
 [0.         0.         0.02065711]]
Resulting covariance matrix for run date DatetimeIndex(['2007-01-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00275804 0.00051966 0.00089147]
 [0.00051966 0.00323223 0.00375342]
 [0.00089147 0.00375342 0.00768089]]
Expected returns vector for run date DatetimeIndex(['2007-01-29'],
 ↪ dtype='datetime64[ns]'):
[0.00677969 0.01363537 0.01789212]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-01-29'],
 ↪ dtype='datetime64[ns]')
[0.         0.63869626 0.36130374]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-02-27'], dtype='datetime64[ns]'):
[[1.         0.36097714 0.26462391]
 [0.36097714 1.         0.89978367]
 [0.26462391 0.89978367 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.017517   0.         0.        ]
 [0.         0.0281127  0.        ]
 [0.         0.         0.02780817]]
Resulting covariance matrix for run date DatetimeIndex(['2007-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00644375 0.00373303 0.00270695]
 [0.00373303 0.0165968  0.01477176]
 [0.00270695 0.01477176 0.01623918]]
Expected returns vector for run date DatetimeIndex(['2007-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.00842761 0.01678049 0.0187555 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-02-27'],
 ↪ dtype='datetime64[ns]')
[0.15042861 0.         0.84957139]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-03-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2007-03-30'], dtype='datetime64[ns]'):
[[ 1.          0.1446811  -0.08279023]
 [ 0.1446811   1.          0.86123639]
 [-0.08279023  0.86123639  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00767453 0.         0.        ]
 [0.         0.01728903 0.        ]
 [0.         0.         0.02346553]]
Resulting covariance matrix for run date DatetimeIndex(['2007-03-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00111907  0.00036474 -0.00028328]
 [ 0.00036474  0.0056793   0.00663861]
 [-0.00028328  0.00663861  0.01046199]]
Expected returns vector for run date DatetimeIndex(['2007-03-30'],
 ↪ dtype='datetime64[ns]'):
[0.00925754 0.01440501 0.01514624]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-03-30'],
 ↪ dtype='datetime64[ns]')
[0.73045198 0.1399785  0.12956951]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-04-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-04-27'], dtype='datetime64[ns]'):
[[ 1.          0.16623917 -0.04976974]
 [ 0.16623917  1.          0.71747643]
 [-0.04976974  0.71747643  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-04-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01015897 0.         0.        ]
```

```
 [0.         0.01080721 0.        ]
 [0.         0.         0.01586632]]
Resulting covariance matrix for run date DatetimeIndex(['2007-04-27'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00185768  0.00032853 -0.0001444 ]
 [ 0.00032853  0.00210232  0.00221447]
 [-0.0001444   0.00221447  0.00453132]]
Expected returns vector for run date DatetimeIndex(['2007-04-27'],
 ↪ dtype='datetime64[ns]'):
[0.00655366 0.01540654 0.01335292]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-04-27'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 3.76499329e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-05-30'], dtype='datetime64[ns]'):
[[1.         0.27139232 0.28221354]
 [0.27139232 1.         0.81622964]
 [0.28221354 0.81622964 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01028099 0.         0.        ]
 [0.         0.01369391 0.        ]
 [0.         0.         0.01751093]]
Resulting covariance matrix for run date DatetimeIndex(['2007-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00221968 0.00080238 0.00106694]
 [0.00080238 0.00393799 0.00411025]
 [0.00106694 0.00411025 0.00643929]]
Expected returns vector for run date DatetimeIndex(['2007-05-30'],
 ↪ dtype='datetime64[ns]'):
[0.00997256 0.01681914 0.0127809 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-05-30'],
 ↪ dtype='datetime64[ns]')
[2.94972470e-01 7.05027530e-01 4.52636994e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-06-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2007-06-29'], dtype='datetime64[ns]'):
[[1.         0.19961381 0.31668901]
 [0.19961381 1.         0.7501933 ]
 [0.31668901 0.7501933  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00830409 0.         0.        ]
 [0.         0.01120572 0.        ]
 [0.         0.         0.01367455]]
Resulting covariance matrix for run date DatetimeIndex(['2007-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00144811 0.00039007 0.00075519]
 [0.00039007 0.00263693 0.00241405]
 [0.00075519 0.00241405 0.00392686]]
Expected returns vector for run date DatetimeIndex(['2007-06-29'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00922529 0.02282782 0.02251008]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-06-29'],
 ↪ dtype='datetime64[ns]')
[0.11032807 0.8247439  0.06492803]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-07-30'], dtype='datetime64[ns]'):
[[1.         0.38623774 0.41822675]
 [0.38623774 1.         0.87496853]
 [0.41822675 0.87496853 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01008308 0.         0.        ]
 [0.         0.0192652  0.        ]
 [0.         0.         0.02341929]]
Resulting covariance matrix for run date DatetimeIndex(['2007-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00223671 0.00165061 0.00217271]
 [0.00165061 0.00816525 0.00868485]
 [0.00217271 0.00868485 0.01206619]]
Expected returns vector for run date DatetimeIndex(['2007-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.0087767  0.0219035  0.02313514]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-07-30'],
 ↪ dtype='datetime64[ns]')
[0.02031724 0.95184738 0.02783538]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-08-30'], dtype='datetime64[ns]'):
[[1.         0.17438468 0.13676921]
 [0.17438468 1.         0.82719535]
 [0.13676921 0.82719535 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00671413 0.         0.        ]
 [0.         0.01418521 0.        ]
 [0.         0.         0.01949395]]
Resulting covariance matrix for run date DatetimeIndex(['2007-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00072127 0.00026574 0.00028642]
 [0.00026574 0.00321952 0.00365985]
 [0.00028642 0.00365985 0.00608023]]
Expected returns vector for run date DatetimeIndex(['2007-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.0089688  0.02111508 0.02221993]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-08-30'],
 ↪ dtype='datetime64[ns]')
[0.42154779 0.57845221 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-09-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2007-09-27'], dtype='datetime64[ns]'):
[[1.         0.46380982 0.33273   ]
```

```
 [0.46380982 1.         0.8973856 ]
 [0.33273    0.8973856  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00832849 0.         0.        ]
 [0.         0.02502577 0.        ]
 [0.         0.         0.03181172]]
Resulting covariance matrix for run date DatetimeIndex(['2007-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00131791 0.00183674 0.00167494]
 [0.00183674 0.01189949 0.01357398]
 [0.00167494 0.01357398 0.01922773]]
Expected returns vector for run date DatetimeIndex(['2007-09-27'],
 ↪ dtype='datetime64[ns]'):
[0.00786413 0.02072381 0.02103541]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-09-27'],
 ↪ dtype='datetime64[ns]')
[0.11736281 0.88263719 0.        ]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-10-30'], dtype='datetime64[ns]'):
[[1.         0.29335567 0.06875455]
 [0.29335567 1.         0.8733021 ]
 [0.06875455 0.8733021  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01565989 0.         0.        ]
 [0.         0.01935965 0.        ]
 [0.         0.         0.02392341]]
Resulting covariance matrix for run date DatetimeIndex(['2007-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00490464 0.00177873 0.00051516]
 [0.00177873 0.00749592 0.00808938]
 [0.00051516 0.00808938 0.01144659]]
Expected returns vector for run date DatetimeIndex(['2007-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.00871875 0.02181377 0.02286082]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-10-30'],
 ↪ dtype='datetime64[ns]')
[1.11022302e-16 1.00000000e+00 0.00000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-11-29'], dtype='datetime64[ns]'):
[[1.         0.12473392 0.08080572]
 [0.12473392 1.         0.85718453]
 [0.08080572 0.85718453 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01103936 0.         0.        ]
 [0.         0.01658541 0.        ]
 [0.         0.         0.01801629]]
Resulting covariance matrix for run date DatetimeIndex(['2007-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00231548 0.00043392 0.00030535]
```

```
 [0.00043392 0.00522644 0.00486653]
 [0.00030535 0.00486653 0.00616715]]
Expected returns vector for run date DatetimeIndex(['2007-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00993086 0.02459108 0.02632666]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-11-29'],
 ↪ dtype='datetime64[ns]')
[0.27454301 0.34221664 0.38324035]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2007-12-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2007-12-28'], dtype='datetime64[ns]'):
[[ 1.         -0.11057048  0.07289034]
 [-0.11057048  1.          0.8358562 ]
 [ 0.07289034  0.8358562   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2007-12-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01268809 0.         0.        ]
 [0.         0.02787368 0.        ]
 [0.         0.         0.02863643]]
Resulting covariance matrix for run date DatetimeIndex(['2007-12-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00305876 -0.00074299  0.0005032 ]
 [-0.00074299  0.01476189  0.01267647]
 [ 0.0005032   0.01267647  0.01558085]]
Expected returns vector for run date DatetimeIndex(['2007-12-28'],
 ↪ dtype='datetime64[ns]'):
[0.00756579 0.02123756 0.02048784]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2007-12-28'],
 ↪ dtype='datetime64[ns]')
[0.46263674 0.49964305 0.03772021]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2008-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.24649005 -0.10805596]
 [-0.24649005  1.          0.83629149]
 [-0.10805596  0.83629149  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01217816 0.         0.        ]
 [0.         0.02533727 0.        ]
 [0.         0.         0.02349504]]
Resulting covariance matrix for run date DatetimeIndex(['2008-01-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00296615 -0.00152115 -0.00061835]
 [-0.00152115  0.01283955  0.00995689]
 [-0.00061835  0.00995689  0.01104034]]
Expected returns vector for run date DatetimeIndex(['2008-01-30'],
 ↪ dtype='datetime64[ns]'):
[0.00981192 0.02183325 0.01832575]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-01-30'],
 ↪ dtype='datetime64[ns]')
[5.97748773e-01 4.02251227e-01 7.08187299e-18]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2008-02-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-02-28'], dtype='datetime64[ns]'):
[[ 1.         -0.03045964  0.01609748]
 [-0.03045964  1.          0.90917635]
 [ 0.01609748  0.90917635  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-02-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01757423 0.         0.        ]
 [0.         0.02744956 0.        ]
 [0.         0.         0.03539665]]
Resulting covariance matrix for run date DatetimeIndex(['2008-02-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00555937 -0.00026449  0.00018025]
 [-0.00026449  0.01356261  0.01590077]
 [ 0.00018025  0.01590077  0.02255261]]
Expected returns vector for run date DatetimeIndex(['2008-02-28'],
 ↪ dtype='datetime64[ns]'):
[0.01442554 0.01927565 0.0187271 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-02-28'],
 ↪ dtype='datetime64[ns]')
[0.61441067 0.38558933 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2008-03-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-03-28'], dtype='datetime64[ns]'):
[[1.         0.26001599 0.37805134]
 [0.26001599 1.         0.83731055]
 [0.37805134 0.83731055 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.0163474  0.         0.        ]
 [0.         0.01706408 0.        ]
 [0.         0.         0.0258587 ]]
Resulting covariance matrix for run date DatetimeIndex(['2008-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00534475 0.00145065 0.00319622]
 [0.00145065 0.00582366 0.00738935]
 [0.00319622 0.00738935 0.01337345]]
Expected returns vector for run date DatetimeIndex(['2008-03-28'],
 ↪ dtype='datetime64[ns]'):
[0.01492268 0.01909666 0.01488225]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-03-28'],
 ↪ dtype='datetime64[ns]')
[3.76025138e-01 6.23974862e-01 3.31124031e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2008-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-04-29'], dtype='datetime64[ns]'):
[[1.         0.19649619 0.21215242]
 [0.19649619 1.         0.84303033]
 [0.21215242 0.84303033 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01351331 0.         0.        ]
 [0.         0.01296574 0.        ]
 [0.         0.         0.01902103]]
Resulting covariance matrix for run date DatetimeIndex(['2008-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00346958 0.00065413 0.00103609]
 [0.00065413 0.0031941  0.00395028]
 [0.00103609 0.00395028 0.0068742 ]]
Expected returns vector for run date DatetimeIndex(['2008-04-29'],
 ↪  dtype='datetime64[ns]'):
[0.01484372 0.01741137 0.0088888 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-04-29'],
 ↪  dtype='datetime64[ns]')
[3.86821666e-01 6.13178334e-01 8.63348900e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-05-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2008-05-30'], dtype='datetime64[ns]'):
[[1.         0.0606136  0.07604823]
 [0.0606136  1.         0.91326022]
 [0.07604823 0.91326022 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-05-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01509433 0.         0.        ]
 [0.         0.01871943 0.        ]
 [0.         0.         0.02601122]]
Resulting covariance matrix for run date DatetimeIndex(['2008-05-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00478462 0.00035966 0.00062702]
 [0.00035966 0.00735876 0.00933828]
 [0.00062702 0.00933828 0.01420825]]
Expected returns vector for run date DatetimeIndex(['2008-05-30'],
 ↪  dtype='datetime64[ns]'):
[0.01319922 0.02179725 0.01511068]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-05-30'],
 ↪  dtype='datetime64[ns]')
[0.40715809 0.59284191 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-06-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-06-27'], dtype='datetime64[ns]'):
[[ 1.         -0.14341262 -0.19919936]
 [-0.14341262  1.          0.9308148 ]
 [-0.19919936  0.9308148   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-06-27'],
 ↪  dtype='datetime64[ns]'):
[[0.01413126 0.         0.        ]
 [0.         0.02943346 0.        ]
 [0.         0.         0.04338052]]
Resulting covariance matrix for run date DatetimeIndex(['2008-06-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00439324 -0.0013123  -0.0026865 ]
 [-0.0013123   0.01905923  0.02614701]
 [-0.0026865   0.02614701  0.04140113]]
Expected returns vector for run date DatetimeIndex(['2008-06-27'],
 ↪  dtype='datetime64[ns]'):
[0.01549718 0.01855659 0.00924488]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-06-27'],
 ↪  dtype='datetime64[ns]')
[0.74707132 0.25292868 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.33979374 -0.35221928]
 [-0.33979374  1.          0.92814761]
 [-0.35221928  0.92814761  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-07-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01569173 0.         0.        ]
 [0.         0.01569118 0.        ]
 [0.         0.         0.03224783]]
Resulting covariance matrix for run date DatetimeIndex(['2008-07-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00492461 -0.00167329 -0.00356463]
 [-0.00167329  0.00492426  0.00939298]
 [-0.00356463  0.00939298  0.02079845]]
Expected returns vector for run date DatetimeIndex(['2008-07-30'],
 ↪  dtype='datetime64[ns]'):
[0.01534427 0.0119052  0.00057194]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-07-30'],
 ↪  dtype='datetime64[ns]')
[0.55159081 0.44840919 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-08-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2008-08-29'], dtype='datetime64[ns]'):
[[ 1.         -0.27664337 -0.28978264]
 [-0.27664337  1.          0.8985666 ]
 [-0.28978264  0.8985666   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-08-29'],
 ↪  dtype='datetime64[ns]'):
[[0.03038582 0.         0.        ]
 [0.         0.02290006 0.        ]
 [0.         0.         0.03289914]]
Resulting covariance matrix for run date DatetimeIndex(['2008-08-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.01846596 -0.00384997 -0.00579372]
 [-0.00384997  0.01048826  0.01353947]
 [-0.00579372  0.01353947  0.02164707]]
Expected returns vector for run date DatetimeIndex(['2008-08-29'],
 ↪  dtype='datetime64[ns]'):
[ 0.01552745  0.01232443 -0.00129314]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-08-29'],
 ↪  dtype='datetime64[ns]')
[0.44290876 0.55709124 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-09-29'], dtype='datetime64[ns]'):
[[ 1.         -0.09337237 -0.17751245]
 [-0.09337237  1.          0.92982864]
 [-0.17751245  0.92982864  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-09-29'],
 ↪  dtype='datetime64[ns]'):
[[0.02401764 0.         0.        ]
 [0.         0.04001014 0.        ]
 [0.         0.         0.04961003]]
Resulting covariance matrix for run date DatetimeIndex(['2008-09-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.01153694 -0.00179452 -0.00423018]
 [-0.00179452  0.03201623  0.03691241]
 [-0.00423018  0.03691241  0.04922309]]
Expected returns vector for run date DatetimeIndex(['2008-09-29'],
 ↪  dtype='datetime64[ns]'):
[0.01312051 0.01164207 0.00242383]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-09-29'],
 ↪  dtype='datetime64[ns]')
[0.75010906 0.24989094 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-10-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-10-29'], dtype='datetime64[ns]'):
[[1.         0.25148109 0.24109384]
 [0.25148109 1.         0.90661049]
 [0.24109384 0.90661049 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-10-29'],
 ↪  dtype='datetime64[ns]'):
[[0.02498204 0.         0.        ]
 [0.         0.04026426 0.        ]
 [0.         0.         0.04341961]]
Resulting covariance matrix for run date DatetimeIndex(['2008-10-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01123384 0.00455329 0.0047073 ]
 [0.00455329 0.0291818  0.02852981]
 [0.0047073  0.02852981 0.03393472]]
Expected returns vector for run date DatetimeIndex(['2008-10-29'],
 ↪  dtype='datetime64[ns]'):
[0.01398812 0.00797514 0.00013361]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-10-29'],
 ↪  dtype='datetime64[ns]')
[1.0000000e+00 3.2959746e-16 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2008-11-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2008-11-28'], dtype='datetime64[ns]'):
[[1.         0.21613148 0.12437241]
 [0.21613148 1.         0.831644  ]
 [0.12437241 0.831644   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-11-28'],
 ↪  dtype='datetime64[ns]'):
[[0.02216669 0.         0.        ]
 [0.         0.02480481 0.        ]
 [0.         0.         0.0281443 ]]
Resulting covariance matrix for run date DatetimeIndex(['2008-11-28'],
 ↪  dtype='datetime64[ns]'):
[[0.0103186  0.00249559 0.00162943]
```

```
 [0.00249559 0.01292085 0.01219223]
 [0.00162943 0.01219223 0.01663414]]
Expected returns vector for run date DatetimeIndex(['2008-11-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.01107005  0.0020339  -0.00281052]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-11-28'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 0.0000000e+00 6.8984061e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2008-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2008-12-30'], dtype='datetime64[ns]'):
[[1.         0.19896425 0.08353894]
 [0.19896425 1.         0.87242264]
 [0.08353894 0.87242264 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2008-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.02090165 0.         0.        ]
 [0.         0.0239837  0.        ]
 [0.         0.         0.02390865]]
Resulting covariance matrix for run date DatetimeIndex(['2008-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00786382 0.00179533 0.00075144]
 [0.00179533 0.01035392 0.00900473]
 [0.00075144 0.00900473 0.01028922]]
Expected returns vector for run date DatetimeIndex(['2008-12-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01408968 -0.00092302 -0.00494044]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2008-12-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.80411242e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-01-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-01-30'], dtype='datetime64[ns]'):
[[1.         0.13540301 0.12054805]
 [0.13540301 1.         0.66602737]
 [0.12054805 0.66602737 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01725493 0.         0.        ]
 [0.         0.02053632 0.        ]
 [0.         0.         0.04344205]]
Resulting covariance matrix for run date DatetimeIndex(['2009-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00327506 0.00052778 0.00099398]
 [0.00052778 0.00463915 0.00653609]
 [0.00099398 0.00653609 0.02075933]]
Expected returns vector for run date DatetimeIndex(['2009-01-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01536011 -0.00065238 -0.00237665]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-01-30'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-02-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-02-27'], dtype='datetime64[ns]'):
[[1.         0.13473837 0.09768663]
 [0.13473837 1.         0.94212717]
 [0.09768663 0.94212717 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.02362417 0.         0.        ]
 [0.         0.02235118 0.        ]
 [0.         0.         0.03724539]]
Resulting covariance matrix for run date DatetimeIndex(['2009-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01116203 0.00142291 0.00171907]
 [0.00142291 0.0099915  0.01568601]
 [0.00171907 0.01568601 0.02774439]]
Expected returns vector for run date DatetimeIndex(['2009-02-27'],
 ↪ dtype='datetime64[ns]'):
[ 0.01441334 -0.00396314 -0.0069973 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-02-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.70870262e-16 1.09036093e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-03-30'], dtype='datetime64[ns]'):
[[1.         0.05321001 0.0171504 ]
 [0.05321001 1.         0.86625385]
 [0.0171504  0.86625385 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01621625 0.         0.        ]
 [0.         0.02351742 0.        ]
 [0.         0.         0.03505616]]
Resulting covariance matrix for run date DatetimeIndex(['2009-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00420747 0.00032468 0.00015599]
 [0.00032468 0.0088491  0.01142666]
 [0.00015599 0.01142666 0.01966295]]
Expected returns vector for run date DatetimeIndex(['2009-03-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01599579 -0.00392822 -0.01054284]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-03-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.04083409e-17 1.67977732e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-04-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.21834317 -0.21389012]
 [-0.21834317  1.          0.91173148]
 [-0.21389012  0.91173148  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-04-29'],
 ↪ dtype='datetime64[ns]'):
```

```
[[0.00777094 0.         0.        ]
 [0.         0.03997022 0.        ]
 [0.         0.         0.04763997]]
Resulting covariance matrix for run date DatetimeIndex(['2009-04-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00114736 -0.00128856 -0.00150449]
 [-0.00128856  0.03035475  0.03298591]
 [-0.00150449  0.03298591  0.04312176]]
Expected returns vector for run date DatetimeIndex(['2009-04-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.01474316 -0.00426986 -0.00937116]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-04-29'],
 ↪ dtype='datetime64[ns]')
[9.95812226e-01 4.18777431e-03 3.00258105e-16]
==================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-05-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-05-29'], dtype='datetime64[ns]'):
[[1.         0.33454186 0.3396036 ]
 [0.33454186 1.         0.74832988]
 [0.3396036  0.74832988 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-05-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01117568 0.         0.        ]
 [0.         0.01909748 0.        ]
 [0.         0.         0.02114339]]
Resulting covariance matrix for run date DatetimeIndex(['2009-05-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00274771 0.00157081 0.0017654 ]
 [0.00157081 0.0080237  0.00664762]
 [0.0017654  0.00664762 0.00983494]]
Expected returns vector for run date DatetimeIndex(['2009-05-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.01211372 -0.00141014 -0.00209725]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-05-29'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
==================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-06-29'], dtype='datetime64[ns]'):
[[1.         0.47159716 0.54433846]
 [0.47159716 1.         0.92956281]
 [0.54433846 0.92956281 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01018569 0.         0.        ]
 [0.         0.01762324 0.        ]
 [0.         0.         0.02953152]]
Resulting covariance matrix for run date DatetimeIndex(['2009-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00228246 0.00186239 0.0036022 ]
 [0.00186239 0.00683273 0.01064322]
 [0.0036022  0.01064322 0.01918644]]
```

```
Expected returns vector for run date DatetimeIndex(['2009-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.01714164 0.00965801 0.0114973 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-06-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 2.88657986e-15 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-07-30'], dtype='datetime64[ns]'):
[[1.         0.38446349 0.40890169]
 [0.38446349 1.         0.92962515]
 [0.40890169 0.92962515 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00887667 0.         0.        ]
 [0.         0.01525661 0.        ]
 [0.         0.         0.02058277]]
Resulting covariance matrix for run date DatetimeIndex(['2009-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0016547  0.00109341 0.00156889]
 [0.00109341 0.00488805 0.00613041]
 [0.00156889 0.00613041 0.00889666]]
Expected returns vector for run date DatetimeIndex(['2009-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.01691722 0.00821166 0.01406178]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-07-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 6.78168459e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-08-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-08-28'], dtype='datetime64[ns]'):
[[1.         0.15645243 0.1754643 ]
 [0.15645243 1.         0.8268612 ]
 [0.1754643  0.8268612  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-08-28'],
 ↪ dtype='datetime64[ns]'):
[[0.0097311  0.         0.        ]
 [0.         0.01048147 0.        ]
 [0.         0.         0.01453475]]
Resulting covariance matrix for run date DatetimeIndex(['2009-08-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00132572 0.00022341 0.00034745]
 [0.00022341 0.00153806 0.00176356]
 [0.00034745 0.00176356 0.00295762]]
Expected returns vector for run date DatetimeIndex(['2009-08-28'],
 ↪ dtype='datetime64[ns]'):
[0.01678121 0.0098441  0.01206889]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-08-28'],
 ↪ dtype='datetime64[ns]')
[0.83679912 0.07770014 0.08550073]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-09-29'], dtype='datetime64[ns]'):
```

```
[[1.         0.21597454 0.25166836]
 [0.21597454 1.         0.87871855]
 [0.25166836 0.87871855 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0119416  0.         0.        ]
 [0.         0.01247049 0.        ]
 [0.         0.         0.01861972]]
Resulting covariance matrix for run date DatetimeIndex(['2009-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00270944 0.00061109 0.00106321]
 [0.00061109 0.00295475 0.00387668]
 [0.00106321 0.00387668 0.00658718]]
Expected returns vector for run date DatetimeIndex(['2009-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.01682238 0.00770252 0.00834099]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-09-29'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 6.9388939e-17 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-10-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-10-30'], dtype='datetime64[ns]'):
[[ 1.         -0.01396044  0.04970887]
 [-0.01396044  1.          0.81421447]
 [ 0.04970887  0.81421447  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0142509  0.         0.        ]
 [0.         0.01438088 0.        ]
 [0.         0.         0.01941033]]
Resulting covariance matrix for run date DatetimeIndex(['2009-10-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.42161643e-03 -2.00274098e-05  9.62514080e-05]
 [-2.00274098e-05  1.44766866e-03  1.59094570e-03]
 [ 9.62514080e-05  1.59094570e-03  2.63732659e-03]]
Expected returns vector for run date DatetimeIndex(['2009-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.01919878 0.00692597 0.00689659]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-10-30'],
 ↪ dtype='datetime64[ns]')
[8.86808224e-01 1.13191776e-01 2.00531750e-18]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2009-11-12'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2009-11-12'], dtype='datetime64[ns]'):
[[1.         0.24271126 0.10998505]
 [0.24271126 1.         0.70534533]
 [0.10998505 0.70534533 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-11-12'],
 ↪ dtype='datetime64[ns]'):
[[0.01386266 0.         0.        ]
 [0.         0.01030266 0.        ]
 [0.         0.         0.01339009]]
```

```
Resulting covariance matrix for run date DatetimeIndex(['2009-11-12'],
 ↪  dtype='datetime64[ns]'):
[[0.00249825 0.00045064 0.0002654 ]
 [0.00045064 0.00137988 0.00126496]
 [0.0002654  0.00126496 0.00233083]]
Expected returns vector for run date DatetimeIndex(['2009-11-12'],
 ↪  dtype='datetime64[ns]'):
[0.01966598 0.00368593 0.00409127]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-11-12'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 5.55111512e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2009-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2009-12-30'], dtype='datetime64[ns]'):
[[1.         0.37494749 0.37546462]
 [0.37494749 1.         0.81115913]
 [0.37546462 0.81115913 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2009-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01347495 0.         0.        ]
 [0.         0.01174866 0.        ]
 [0.         0.         0.01637211]]
Resulting covariance matrix for run date DatetimeIndex(['2009-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00254204 0.00083103 0.00115965]
 [0.00083103 0.00193243 0.00218437]
 [0.00115965 0.00218437 0.00375264]]
Expected returns vector for run date DatetimeIndex(['2009-12-30'],
 ↪  dtype='datetime64[ns]'):
[0.01878801 0.00321238 0.00331691]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2009-12-30'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 3.05311332e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2010-01-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2010-01-29'], dtype='datetime64[ns]'):
[[1.         0.50120711 0.45181763]
 [0.50120711 1.         0.95041162]
 [0.45181763 0.95041162 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-01-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01518469 0.         0.        ]
 [0.         0.01297201 0.        ]
 [0.         0.         0.01512002]]
Resulting covariance matrix for run date DatetimeIndex(['2010-01-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00415035 0.00177706 0.00186721]
 [0.00177706 0.00302892 0.0033554 ]
 [0.00186721 0.0033554  0.00411507]]
Expected returns vector for run date DatetimeIndex(['2010-01-29'],
 ↪  dtype='datetime64[ns]'):
[0.01747835 0.00345611 0.00322488]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-01-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 1.11022302e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-02-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2010-02-26'], dtype='datetime64[ns]'):
[[1.         0.52584296 0.48744446]
 [0.52584296 1.         0.81026918]
 [0.48744446 0.81026918 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-02-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00933822 0.         0.        ]
 [0.         0.00691628 0.        ]
 [0.         0.         0.00856777]]
Resulting covariance matrix for run date DatetimeIndex(['2010-02-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00183125 0.0007132  0.00081899]
 [0.0007132  0.00100453 0.0010083 ]
 [0.00081899 0.0010083  0.00154154]]
Expected returns vector for run date DatetimeIndex(['2010-02-26'],
 ↪ dtype='datetime64[ns]'):
[0.01533426 0.00062348 0.00126209]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-02-26'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 2.22044605e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-03-30'], dtype='datetime64[ns]'):
[[1.         0.29917538 0.25690132]
 [0.29917538 1.         0.88280821]
 [0.25690132 0.88280821 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00848947 0.         0.        ]
 [0.         0.00818926 0.        ]
 [0.         0.         0.01102971]]
Resulting covariance matrix for run date DatetimeIndex(['2010-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00144142 0.00041599 0.00048111]
 [0.00041599 0.00134128 0.0015948 ]
 [0.00048111 0.0015948  0.00243309]]
Expected returns vector for run date DatetimeIndex(['2010-03-30'],
 ↪ dtype='datetime64[ns]'):
[0.01567558 0.00328754 0.00544816]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-03-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 4.99600361e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-04-29'], dtype='datetime64[ns]'):
[[1.         0.00862789 0.08921842]
 [0.00862789 1.         0.916767  ]
 [0.08921842 0.916767   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01119316 0.         0.        ]
 [0.         0.01409857 0.        ]
 [0.         0.         0.01855866]]
Resulting covariance matrix for run date DatetimeIndex(['2010-04-29'],
 ↪  dtype='datetime64[ns]'):
[[2.50573785e-03 2.72309437e-05 3.70667015e-04]
 [2.72309437e-05 3.97539601e-03 4.79745327e-03]
 [3.70667015e-04 4.79745327e-03 6.88847622e-03]]
Expected returns vector for run date DatetimeIndex(['2010-04-29'],
 ↪  dtype='datetime64[ns]'):
[0.01615352 0.00451327 0.00734163]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-04-29'],
 ↪  dtype='datetime64[ns]')
[0.98553728 0.         0.01446272]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2010-05-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2010-05-28'], dtype='datetime64[ns]'):
[[ 1.         -0.62445343 -0.64801371]
 [-0.62445343  1.          0.94938576]
 [-0.64801371  0.94938576  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-05-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00909565 0.         0.        ]
 [0.         0.01154248 0.        ]
 [0.         0.         0.01289234]]
Resulting covariance matrix for run date DatetimeIndex(['2010-05-28'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00182008 -0.0014423  -0.00167175]
 [-0.0014423   0.00293103  0.00310811]
 [-0.00167175  0.00310811  0.00365667]]
Expected returns vector for run date DatetimeIndex(['2010-05-28'],
 ↪  dtype='datetime64[ns]'):
[0.01652337 0.00286449 0.0069547 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-05-28'],
 ↪  dtype='datetime64[ns]')
[0.66847843 0.         0.33152157]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2010-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-06-29'], dtype='datetime64[ns]'):
[[1.         0.12857533 0.03544483]
 [0.12857533 1.         0.89249504]
 [0.03544483 0.89249504 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01059437 0.         0.        ]
 [0.         0.00858102 0.        ]
 [0.         0.         0.01132013]]
Resulting covariance matrix for run date DatetimeIndex(['2010-06-29'],
 ↪  dtype='datetime64[ns]'):
[[2.35705495e-03 2.45465935e-04 8.92685857e-05]
```

```
 [2.45465935e-04 1.54631331e-03 1.82060407e-03]
 [8.92685857e-05 1.82060407e-03 2.69105086e-03]]
Expected returns vector for run date DatetimeIndex(['2010-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.01765975 0.00082177 0.00272355]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-06-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 2.77555756e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-07-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2010-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.10480077  0.05042398]
 [-0.10480077  1.          0.86658072]
 [ 0.05042398  0.86658072  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00550172 0.         0.        ]
 [0.         0.00719829 0.        ]
 [0.         0.         0.01132574]]
Resulting covariance matrix for run date DatetimeIndex(['2010-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 6.65917462e-04 -9.13094289e-05  6.91234266e-05]
 [-9.13094289e-05  1.13993980e-03  1.55427521e-03]
 [ 6.91234266e-05  1.55427521e-03  2.82199263e-03]]
Expected returns vector for run date DatetimeIndex(['2010-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.01896    0.00188532 0.00125339]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-07-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 1.77635684e-15]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-08-30'], dtype='datetime64[ns]'):
[[ 1.         -0.15400553 -0.22181995]
 [-0.15400553  1.          0.80933917]
 [-0.22181995  0.80933917  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00786174 0.         0.        ]
 [0.         0.00923675 0.        ]
 [0.         0.         0.0127637 ]]
Resulting covariance matrix for run date DatetimeIndex(['2010-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00098891 -0.00017893 -0.00035614]
 [-0.00017893  0.00136508  0.00152667]
 [-0.00035614  0.00152667  0.00260659]]
Expected returns vector for run date DatetimeIndex(['2010-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.01670505 0.00086712 0.0022926 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-08-30'],
 ↪ dtype='datetime64[ns]')
[0.96422258 0.         0.03577742]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-09-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-09-28'], dtype='datetime64[ns]'):
[[1.         0.24977553 0.17630115]
 [0.24977553 1.         0.94041401]
 [0.17630115 0.94041401 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-09-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01034268 0.         0.        ]
 [0.         0.01100883 0.        ]
 [0.         0.         0.01185074]]
Resulting covariance matrix for run date DatetimeIndex(['2010-09-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00224639 0.00059723 0.00045379]
 [0.00059723 0.00254508 0.00257647]
 [0.00045379 0.00257647 0.00294924]]
Expected returns vector for run date DatetimeIndex(['2010-09-28'],
 ↪ dtype='datetime64[ns]'):
[0.01798761 0.00144727 0.00488007]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-09-28'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.66533454e-16 1.38777878e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-10-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2010-10-29'], dtype='datetime64[ns]'):
[[1.         0.49897259 0.34604055]
 [0.49897259 1.         0.90982258]
 [0.34604055 0.90982258 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-10-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01295396 0.         0.        ]
 [0.         0.01297069 0.        ]
 [0.         0.         0.01907298]]
Resulting covariance matrix for run date DatetimeIndex(['2010-10-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0033561  0.00167677 0.00170993]
 [0.00167677 0.00336478 0.00450162]
 [0.00170993 0.00450162 0.00727557]]
Expected returns vector for run date DatetimeIndex(['2010-10-29'],
 ↪ dtype='datetime64[ns]'):
[0.0180634  0.00050084 0.00427839]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-10-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 2.22044605e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-11-29'], dtype='datetime64[ns]'):
[[1.         0.23610272 0.15733707]
 [0.23610272 1.         0.86636204]
 [0.15733707 0.86636204 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.009559   0.         0.        ]
```

```
 [0.         0.01106735 0.        ]
 [0.         0.         0.01910307]]
Resulting covariance matrix for run date DatetimeIndex(['2010-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00191886 0.00052454 0.00060335]
 [0.00052454 0.00257221 0.0038465 ]
 [0.00060335 0.0038465  0.00766348]]
Expected returns vector for run date DatetimeIndex(['2010-11-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.01712215 -0.00302076  0.00101037]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-11-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 4.71844785e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2010-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2010-12-30'], dtype='datetime64[ns]'):
[[1.         0.10957211 0.31236646]
 [0.10957211 1.         0.84356505]
 [0.31236646 0.84356505 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2010-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01126132 0.         0.        ]
 [0.         0.01200779 0.        ]
 [0.         0.         0.01982685]]
Resulting covariance matrix for run date DatetimeIndex(['2010-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00240953 0.00028152 0.00132514]
 [0.00028152 0.00273955 0.00381583]
 [0.00132514 0.00381583 0.00746898]]
Expected returns vector for run date DatetimeIndex(['2010-12-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01858647 -0.00267296  0.00109832]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2010-12-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 2.77555756e-17 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-01-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-01-28'], dtype='datetime64[ns]'):
[[1.         0.06599005 0.01631451]
 [0.06599005 1.         0.91546226]
 [0.01631451 0.91546226 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-01-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00813965 0.         0.        ]
 [0.         0.01384776 0.        ]
 [0.         0.         0.0186291 ]]
Resulting covariance matrix for run date DatetimeIndex(['2011-01-28'],
 ↪ dtype='datetime64[ns]'):
[[1.19257061e-03 1.33886388e-04 4.45291256e-05]
 [1.33886388e-04 3.45169005e-03 4.25093547e-03]
 [4.45291256e-05 4.25093547e-03 6.24678197e-03]]
Expected returns vector for run date DatetimeIndex(['2011-01-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.01775203 -0.00331557 -0.0007067 ]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-01-28'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 2.49800181e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-02-25'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2011-02-25'], dtype='datetime64[ns]'):
[[1.         0.04948828 0.04510704]
 [0.04948828 1.         0.92535251]
 [0.04510704 0.92535251 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-02-25'],
 ↪ dtype='datetime64[ns]'):
[[0.00901903 0.         0.        ]
 [0.         0.01089738 0.        ]
 [0.         0.         0.01315985]]
Resulting covariance matrix for run date DatetimeIndex(['2011-02-25'],
 ↪ dtype='datetime64[ns]'):
[[0.0017082  0.00010214 0.00011243]
 [0.00010214 0.00249381 0.00278676]
 [0.00011243 0.00278676 0.00363681]]
Expected returns vector for run date DatetimeIndex(['2011-02-25'],
 ↪ dtype='datetime64[ns]'):
[ 0.01297991 -0.00342864 -0.00297242]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-02-25'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-03-30'], dtype='datetime64[ns]'):
[[1.         0.37722111 0.22873926]
 [0.37722111 1.         0.88538766]
 [0.22873926 0.88538766 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01006725 0.         0.        ]
 [0.         0.01104307 0.        ]
 [0.         0.         0.01607445]]
Resulting covariance matrix for run date DatetimeIndex(['2011-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00131754 0.00054518 0.00048121]
 [0.00054518 0.00158534 0.00204316]
 [0.00048121 0.00204316 0.00335904]]
Expected returns vector for run date DatetimeIndex(['2011-03-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01304882 -0.00536857 -0.00202889]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-03-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 2.22044605e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-04-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2011-04-26'], dtype='datetime64[ns]'):
[[1.         0.3410576  0.29266706]
```

```
 [0.3410576  1.         0.9394933 ]
 [0.29266706 0.9394933  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-04-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00947538 0.         0.        ]
 [0.         0.01142759 0.        ]
 [0.         0.         0.01702566]]
Resulting covariance matrix for run date DatetimeIndex(['2011-04-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00179566 0.0007386  0.00094429]
 [0.0007386  0.0026118  0.0036558 ]
 [0.00094429 0.0036558  0.00579746]]
Expected returns vector for run date DatetimeIndex(['2011-04-26'],
 ↪ dtype='datetime64[ns]'):
[ 0.01366148 -0.0011142   0.00739463]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-04-26'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-05-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-05-27'], dtype='datetime64[ns]'):
[[1.         0.10632948 0.1168078 ]
 [0.10632948 1.         0.90284783]
 [0.1168078  0.90284783 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-05-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00724923 0.         0.        ]
 [0.         0.01003775 0.        ]
 [0.         0.         0.01302169]]
Resulting covariance matrix for run date DatetimeIndex(['2011-05-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00115613 0.00017022 0.00024258]
 [0.00017022 0.00221664 0.00259622]
 [0.00024258 0.00259622 0.00373042]]
Expected returns vector for run date DatetimeIndex(['2011-05-27'],
 ↪ dtype='datetime64[ns]'):
[ 0.01526608 -0.0037696   0.00348896]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-05-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 1.66533454e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-06-29'], dtype='datetime64[ns]'):
[[1.         0.05953135 0.08389884]
 [0.05953135 1.         0.92467499]
 [0.08389884 0.92467499 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00862707 0.         0.        ]
 [0.         0.01067265 0.        ]
 [0.         0.         0.01241206]]
Resulting covariance matrix for run date DatetimeIndex(['2011-06-29'],
 ↪ dtype='datetime64[ns]'):
[[1.04196942e-03 7.67378450e-05 1.25774107e-04]
```

```
 [7.67378450e-05 1.59467734e-03 1.71487832e-03]
 [1.25774107e-04 1.71487832e-03 2.15682832e-03]]
Expected returns vector for run date DatetimeIndex(['2011-06-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.01384826 -0.00321733  0.00589693]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-06-29'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 1.9971004e-16 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-07-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2011-07-29'], dtype='datetime64[ns]'):
[[ 1.         -0.02641723  0.03326391]
 [-0.02641723  1.          0.9283762 ]
 [ 0.03326391  0.9283762   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-07-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01993378 0.         0.        ]
 [0.         0.01485168 0.        ]
 [0.         0.         0.01703228]]
Resulting covariance matrix for run date DatetimeIndex(['2011-07-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00834446 -0.00016424  0.00023717]
 [-0.00016424  0.00463202  0.00493164]
 [ 0.00023717  0.00493164  0.00609207]]
Expected returns vector for run date DatetimeIndex(['2011-07-29'],
 ↪ dtype='datetime64[ns]'):
[0.01204197 0.00240222 0.01395153]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-07-29'],
 ↪ dtype='datetime64[ns]')
[3.55810689e-01 9.15066634e-17 6.44189311e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-08-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2011-08-30'], dtype='datetime64[ns]'):
[[1.         0.45152169 0.3385368 ]
 [0.45152169 1.         0.95499909]
 [0.3385368  0.95499909 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.02435881 0.         0.        ]
 [0.         0.01611173 0.        ]
 [0.         0.         0.01945267]]
Resulting covariance matrix for run date DatetimeIndex(['2011-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01186704 0.00354411 0.00320827]
 [0.00354411 0.00519176 0.00598624]
 [0.00320827 0.00598624 0.00756813]]
Expected returns vector for run date DatetimeIndex(['2011-08-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.01355985 -0.0002988   0.00968021]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-08-30'],
 ↪ dtype='datetime64[ns]')
[6.55785773e-01 3.46944695e-18 3.44214227e-01]
```

```
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-09-29'], dtype='datetime64[ns]'):
[[1.         0.3243708  0.05396705]
 [0.3243708  1.         0.85117173]
 [0.05396705 0.85117173 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01686127 0.         0.        ]
 [0.         0.01635176 0.        ]
 [0.         0.         0.02117223]]
Resulting covariance matrix for run date DatetimeIndex(['2011-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00540174 0.00169922 0.00036605]
 [0.00169922 0.00508022 0.00559889]
 [0.00036605 0.00559889 0.008517  ]]
Expected returns vector for run date DatetimeIndex(['2011-09-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.01843506 -0.00302257  0.00422802]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-09-29'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-10-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-10-28'], dtype='datetime64[ns]'):
[[1.         0.48974746 0.23564309]
 [0.48974746 1.         0.88967389]
 [0.23564309 0.88967389 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-10-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01403791 0.         0.        ]
 [0.         0.01521586 0.        ]
 [0.         0.         0.0185756 ]]
Resulting covariance matrix for run date DatetimeIndex(['2011-10-28'],
 ↪ dtype='datetime64[ns]'):
[[0.0037442  0.00198758 0.00116749]
 [0.00198758 0.00439892 0.00477775]
 [0.00116749 0.00477775 0.00655601]]
Expected returns vector for run date DatetimeIndex(['2011-10-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.01357274 -0.00034522  0.00545797]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-10-28'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 2.08166817e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2011-11-29'], dtype='datetime64[ns]'):
[[1.         0.3349939  0.16059555]
 [0.3349939  1.         0.9308812 ]
 [0.16059555 0.9308812  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01582398 0.         0.        ]
```

```
 [0.         0.01604765 0.        ]
 [0.         0.         0.0226788 ]]
Resulting covariance matrix for run date DatetimeIndex(['2011-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00500797 0.00170135 0.00115265]
 [0.00170135 0.00515054 0.00677572]
 [0.00115265 0.00677572 0.01028656]]
Expected returns vector for run date DatetimeIndex(['2011-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.01768048 0.01024798 0.01387972]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-11-29'],
 ↪ dtype='datetime64[ns]')
[0.80379614 0.         0.19620386]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2011-12-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2011-12-30'], dtype='datetime64[ns]'):
[[1.         0.30980885 0.16298239]
 [0.30980885 1.         0.84663731]
 [0.16298239 0.84663731 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2011-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00973339 0.         0.        ]
 [0.         0.01202446 0.        ]
 [0.         0.         0.01750071]]
Resulting covariance matrix for run date DatetimeIndex(['2011-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00161056 0.00061641 0.00047196]
 [0.00061641 0.00245799 0.00302878]
 [0.00047196 0.00302878 0.00520667]]
Expected returns vector for run date DatetimeIndex(['2011-12-30'],
 ↪ dtype='datetime64[ns]'):
[0.01386675 0.00853323 0.01081546]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2011-12-30'],
 ↪ dtype='datetime64[ns]')
[0.89715117 0.         0.10284883]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-01-30'], dtype='datetime64[ns]'):
[[1.         0.13460591 0.13645047]
 [0.13460591 1.         0.93831146]
 [0.13645047 0.93831146 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01464944 0.         0.        ]
 [0.         0.0132072  0.        ]
 [0.         0.         0.01910096]]
Resulting covariance matrix for run date DatetimeIndex(['2012-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00429212 0.00052087 0.00076363]
 [0.00052087 0.0034886  0.00473416]
 [0.00076363 0.00473416 0.00729693]]
Expected returns vector for run date DatetimeIndex(['2012-01-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00887021 0.00494743 0.00429802]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-01-30'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 1.2490009e-16 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-02-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-02-28'], dtype='datetime64[ns]'):
[[1.         0.30939409 0.19197686]
 [0.30939409 1.         0.90975702]
 [0.19197686 0.90975702 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-02-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01077271 0.         0.        ]
 [0.         0.01370857 0.        ]
 [0.         0.         0.01989979]]
Resulting covariance matrix for run date DatetimeIndex(['2012-02-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00243708 0.00095951 0.00086425]
 [0.00095951 0.00394642 0.00521177]
 [0.00086425 0.00521177 0.00831603]]
Expected returns vector for run date DatetimeIndex(['2012-02-28'],
 ↪ dtype='datetime64[ns]'):
[0.01097029 0.00976225 0.01368987]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-02-28'],
 ↪ dtype='datetime64[ns]')
[0.71950448 0.         0.28049552]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-03-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2012-03-30'], dtype='datetime64[ns]'):
[[1.         0.30069333 0.30980545]
 [0.30069333 1.         0.91639316]
 [0.30980545 0.91639316 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01044594 0.         0.        ]
 [0.         0.00948015 0.        ]
 [0.         0.         0.0118898 ]]
Resulting covariance matrix for run date DatetimeIndex(['2012-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00207324 0.00056577 0.00073108]
 [0.00056577 0.00170759 0.00196257]
 [0.00073108 0.00196257 0.00268598]]
Expected returns vector for run date DatetimeIndex(['2012-03-30'],
 ↪ dtype='datetime64[ns]'):
[0.01010741 0.01025637 0.01880158]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-03-30'],
 ↪ dtype='datetime64[ns]')
[0.11148448 0.         0.88851552]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-04-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-04-27'], dtype='datetime64[ns]'):
[[1.         0.36319882 0.36831868]
```

```
 [0.36319882 1.         0.92422628]
 [0.36831868 0.92422628 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-04-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01052327 0.         0.        ]
 [0.         0.00984672 0.        ]
 [0.         0.         0.01520012]]
Resulting covariance matrix for run date DatetimeIndex(['2012-04-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00232552 0.00079033 0.0012372 ]
 [0.00079033 0.00203612 0.00290493]
 [0.0012372  0.00290493 0.00485192]]
Expected returns vector for run date DatetimeIndex(['2012-04-27'],
 ↪ dtype='datetime64[ns]'):
[0.00953819 0.00731704 0.01654915]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-04-27'],
 ↪ dtype='datetime64[ns]')
[0.24123036 0.         0.75876964]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-05-30'], dtype='datetime64[ns]'):
[[ 1.         -0.23325384 -0.27312756]
 [-0.23325384  1.          0.93502933]
 [-0.27312756  0.93502933  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0133155  0.         0.        ]
 [0.         0.01070103 0.        ]
 [0.         0.         0.01625368]]
Resulting covariance matrix for run date DatetimeIndex(['2012-05-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00372336 -0.00069796 -0.00124135]
 [-0.00069796  0.00240475  0.00341524]
 [-0.00124135  0.00341524  0.00554782]]
Expected returns vector for run date DatetimeIndex(['2012-05-30'],
 ↪ dtype='datetime64[ns]'):
[0.00895663 0.00347912 0.01092219]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-05-30'],
 ↪ dtype='datetime64[ns]')
[5.17262178e-01 1.38777878e-17 4.82737822e-01]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-06-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2012-06-29'], dtype='datetime64[ns]'):
[[ 1.         -0.05033623 -0.09293203]
 [-0.05033623  1.          0.91057426]
 [-0.09293203  0.91057426  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00984829 0.         0.        ]
 [0.         0.00867196 0.        ]
 [0.         0.         0.01087872]]
Resulting covariance matrix for run date DatetimeIndex(['2012-06-29'],
 ↪ dtype='datetime64[ns]'):
```

```
[[ 2.03676403e-03 -9.02772156e-05 -2.09085266e-04]
 [-9.02772156e-05  1.57926234e-03  1.80397350e-03]
 [-2.09085266e-04  1.80397350e-03  2.48527980e-03]]
Expected returns vector for run date DatetimeIndex(['2012-06-29'],
 ↪  dtype='datetime64[ns]'):
[ 0.00453174 -0.00553414 -0.00225604]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-06-29'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 5.55111512e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2012-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.04608499 -0.08929133]
 [-0.04608499  1.          0.91943687]
 [-0.08929133  0.91943687  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-07-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00925277 0.         0.        ]
 [0.         0.00845146 0.        ]
 [0.         0.         0.01115885]]
Resulting covariance matrix for run date DatetimeIndex(['2012-07-30'],
 ↪  dtype='datetime64[ns]'):
[[ 1.79788731e-03 -7.56801898e-05 -1.93606447e-04]
 [-7.56801898e-05  1.49997167e-03  1.82092775e-03]
 [-1.93606447e-04  1.82092775e-03  2.61492080e-03]]
Expected returns vector for run date DatetimeIndex(['2012-07-30'],
 ↪  dtype='datetime64[ns]'):
[ 0.00675684 -0.00259885  0.00056281]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-07-30'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 5.55111512e-17 4.02455846e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2012-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-08-30'], dtype='datetime64[ns]'):
[[1.         0.12018967 0.13217346]
 [0.12018967 1.         0.91208774]
 [0.13217346 0.91208774 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-08-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00854194 0.         0.        ]
 [0.         0.00958932 0.        ]
 [0.         0.         0.01337114]]
Resulting covariance matrix for run date DatetimeIndex(['2012-08-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00109447 0.00014767 0.00022644]
 [0.00014767 0.00137932 0.00175422]
 [0.00022644 0.00175422 0.00268181]]
Expected returns vector for run date DatetimeIndex(['2012-08-30'],
 ↪  dtype='datetime64[ns]'):
[ 0.00609207 -0.00461194  0.00040837]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-08-30'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 5.64132074e-15 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-09-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2012-09-28'], dtype='datetime64[ns]'):
[[1.         0.10254704 0.04639428]
 [0.10254704 1.         0.95011193]
 [0.04639428 0.95011193 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-09-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00791199 0.         0.        ]
 [0.         0.00747872 0.        ]
 [0.         0.         0.01037656]]
Resulting covariance matrix for run date DatetimeIndex(['2012-09-28'],
 ↪ dtype='datetime64[ns]'):
[[1.18939281e-03 1.15289480e-04 7.23697664e-05]
 [1.15289480e-04 1.06269289e-03 1.40090512e-03]
 [7.23697664e-05 1.40090512e-03 2.04578568e-03]]
Expected returns vector for run date DatetimeIndex(['2012-09-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.00726011 -0.00460892 -0.00024636]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-09-28'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-10-26'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-10-26'], dtype='datetime64[ns]'):
[[ 1.         -0.0731976  -0.09975982]
 [-0.0731976   1.          0.91930187]
 [-0.09975982  0.91930187  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-10-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00931132 0.         0.        ]
 [0.         0.00791234 0.        ]
 [0.         0.         0.01090798]]
Resulting covariance matrix for run date DatetimeIndex(['2012-10-26'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00164731 -0.00010246 -0.00019251]
 [-0.00010246  0.0011895   0.00150751]
 [-0.00019251  0.00150751  0.0022607 ]]
Expected returns vector for run date DatetimeIndex(['2012-10-26'],
 ↪ dtype='datetime64[ns]'):
[ 0.00751499 -0.00385188 -0.00101684]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-10-26'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 7.49400542e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.1573213  -0.13103133]
 [-0.1573213   1.          0.87988812]
 [-0.13103133  0.87988812  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00767965 0.         0.        ]
```

```
 [0.         0.00775872 0.        ]
 [0.         0.         0.00909797]]
Resulting covariance matrix for run date DatetimeIndex(['2012-11-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00117954 -0.00018748 -0.0001831 ]
 [-0.00018748  0.00120395  0.0012422 ]
 [-0.0001831   0.0012422   0.00165546]]
Expected returns vector for run date DatetimeIndex(['2012-11-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.00652336 -0.00182414  0.00043749]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-11-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 7.91033905e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2012-12-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2012-12-28'], dtype='datetime64[ns]'):
[[1.         0.09060326 0.22853109]
 [0.09060326 1.         0.82047904]
 [0.22853109 0.82047904 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2012-12-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00801824 0.         0.        ]
 [0.         0.00611744 0.        ]
 [0.         0.         0.00827473]]
Resulting covariance matrix for run date DatetimeIndex(['2012-12-28'],
 ↪ dtype='datetime64[ns]'):
[[1.35013605e-03 9.33279282e-05 3.18418033e-04]
 [9.33279282e-05 7.85883275e-04 8.72187994e-04]
 [3.18418033e-04 8.72187994e-04 1.43789504e-03]]
Expected returns vector for run date DatetimeIndex(['2012-12-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.00569382 -0.00132938  0.00087003]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2012-12-28'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.38777878e-17 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.09419551 -0.05270758]
 [-0.09419551  1.          0.89725769]
 [-0.05270758  0.89725769  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0081385  0.         0.        ]
 [0.         0.0059751  0.        ]
 [0.         0.         0.00888757]]
Resulting covariance matrix for run date DatetimeIndex(['2013-01-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.25846909e-03 -8.70308768e-05 -7.24359660e-05]
 [-8.70308768e-05  6.78334205e-04  9.05313483e-04]
 [-7.24359660e-05  9.05313483e-04  1.50078986e-03]]
Expected returns vector for run date DatetimeIndex(['2013-01-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.00691809 -0.00153753  0.00253604]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-01-30'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 3.55618313e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-02-27'], dtype='datetime64[ns]'):
[[1.         0.01920599 0.06276681]
 [0.01920599 1.         0.9322232 ]
 [0.06276681 0.9322232  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00752283 0.         0.        ]
 [0.         0.0104929  0.        ]
 [0.         0.         0.01428198]]
Resulting covariance matrix for run date DatetimeIndex(['2013-02-27'],
 ↪ dtype='datetime64[ns]'):
[[9.05486372e-04 2.42567709e-05 1.07899468e-04]
 [2.42567709e-05 1.76161409e-03 2.23523719e-03]
 [1.07899468e-04 2.23523719e-03 3.26359786e-03]]
Expected returns vector for run date DatetimeIndex(['2013-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.00800613 0.00097667 0.00496631]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-02-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 1.66533454e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-03-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2013-03-28'], dtype='datetime64[ns]'):
[[1.         0.13649631 0.0427935 ]
 [0.13649631 1.         0.85383487]
 [0.0427935  0.85383487 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.02517522 0.         0.        ]
 [0.         0.01021873 0.        ]
 [0.         0.         0.0146062 ]]
Resulting covariance matrix for run date DatetimeIndex(['2013-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01204204 0.00066718 0.00029898]
 [0.00066718 0.00198403 0.00242137]
 [0.00029898 0.00242137 0.00405348]]
Expected returns vector for run date DatetimeIndex(['2013-03-28'],
 ↪ dtype='datetime64[ns]'):
[ 0.00557307 -0.00083874  0.00185449]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-03-28'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.24181545 -0.39696723]
 [-0.24181545  1.          0.88871253]
 [-0.39696723  0.88871253  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01198102 0.         0.        ]
 [0.         0.00953096 0.        ]
 [0.         0.         0.01459269]]
Resulting covariance matrix for run date DatetimeIndex(['2013-04-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00301444 -0.00057987 -0.00145748]
 [-0.00057987  0.00190762  0.00259569]
 [-0.00145748  0.00259569  0.00447188]]
Expected returns vector for run date DatetimeIndex(['2013-04-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.00506734 -0.00189857  0.00050746]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-04-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.00995601e-14 6.71684930e-15]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-05-30'], dtype='datetime64[ns]'):
[[1.         0.49082172 0.53225207]
 [0.49082172 1.         0.91962194]
 [0.53225207 0.91962194 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01758169 0.         0.        ]
 [0.         0.01052496 0.        ]
 [0.         0.         0.01351708]]
Resulting covariance matrix for run date DatetimeIndex(['2013-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00618232 0.0018165  0.00252983]
 [0.0018165  0.0022155  0.00261663]
 [0.00252983 0.00261663 0.00365423]]
Expected returns vector for run date DatetimeIndex(['2013-05-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.00028019 -0.00124146  0.00171176]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-05-30'],
 ↪ dtype='datetime64[ns]')
[4.77741079e-01 3.95516953e-16 5.22258921e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-06-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2013-06-28'], dtype='datetime64[ns]'):
[[1.         0.19458893 0.31371427]
 [0.19458893 1.         0.79871047]
 [0.31371427 0.79871047 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-06-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01439208 0.         0.        ]
 [0.         0.01087612 0.        ]
 [0.         0.         0.01973115]]
Resulting covariance matrix for run date DatetimeIndex(['2013-06-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00455691 0.0006701  0.0019599 ]
```

```
 [0.0006701  0.00260238 0.00377084]
 [0.0019599  0.00377084 0.008565  ]]
Expected returns vector for run date DatetimeIndex(['2013-06-28'],
 ↪ dtype='datetime64[ns]'):
[-0.0015163  -0.00036127  0.00266132]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-06-28'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 4.16333634e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-07-30'], dtype='datetime64[ns]'):
[[ 1.          0.02374953 -0.03102352]
 [ 0.02374953  1.          0.83605361]
 [-0.03102352  0.83605361  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00991482 0.         0.        ]
 [0.         0.01406676 0.        ]
 [0.         0.         0.02126468]]
Resulting covariance matrix for run date DatetimeIndex(['2013-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.96607241e-03  6.62466550e-05 -1.30817123e-04]
 [ 6.62466550e-05  3.95747655e-03  5.00169387e-03]
 [-1.30817123e-04  5.00169387e-03  9.04373050e-03]]
Expected returns vector for run date DatetimeIndex(['2013-07-30'],
 ↪ dtype='datetime64[ns]'):
[-0.0058583  -0.00224515  0.00038338]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-07-30'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-08-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2013-08-30'], dtype='datetime64[ns]'):
[[1.         0.1306207  0.18589056]
 [0.1306207  1.         0.88324535]
 [0.18589056 0.88324535 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01711169 0.         0.        ]
 [0.         0.01862684 0.        ]
 [0.         0.         0.03486672]]
Resulting covariance matrix for run date DatetimeIndex(['2013-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00556339 0.00079104 0.00210724]
 [0.00079104 0.00659223 0.01089897]
 [0.00210724 0.01089897 0.02309808]]
Expected returns vector for run date DatetimeIndex(['2013-08-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00245609 -0.00333486 -0.00568271]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-08-30'],
 ↪ dtype='datetime64[ns]')
[6.9388939e-17 0.0000000e+00 1.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-09-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-09-27'], dtype='datetime64[ns]'):
[[ 1.         -0.27784914 -0.35063344]
 [-0.27784914  1.          0.89467067]
 [-0.35063344  0.89467067  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01376774 0.         0.        ]
 [0.         0.011302   0.        ]
 [0.         0.         0.02040721]]
Resulting covariance matrix for run date DatetimeIndex(['2013-09-27'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00398057 -0.00090792 -0.0020688 ]
 [-0.00090792  0.00268244  0.00433333]
 [-0.0020688   0.00433333  0.00874554]]
Expected returns vector for run date DatetimeIndex(['2013-09-27'],
 ↪ dtype='datetime64[ns]'):
[-0.00287718 -0.00485326 -0.01005749]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-09-27'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 5.55111512e-17 1.00000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-10-30'], dtype='datetime64[ns]'):
[[1.         0.05335522 0.06834674]
 [0.05335522 1.         0.93585491]
 [0.06834674 0.93585491 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00895939 0.         0.        ]
 [0.         0.01024803 0.        ]
 [0.         0.         0.01661661]]
Resulting covariance matrix for run date DatetimeIndex(['2013-10-30'],
 ↪ dtype='datetime64[ns]'):
[[1.44487182e-03 8.81796101e-05 1.83151765e-04]
 [8.81796101e-05 1.89039799e-03 2.86855909e-03]
 [1.83151765e-04 2.86855909e-03 4.97001047e-03]]
Expected returns vector for run date DatetimeIndex(['2013-10-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00578629 -0.00579237 -0.01159159]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-10-30'],
 ↪ dtype='datetime64[ns]')
[8.9338259e-17 0.0000000e+00 1.0000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2013-11-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2013-11-29'], dtype='datetime64[ns]'):
[[1.         0.12122227 0.06176267]
 [0.12122227 1.         0.89290857]
 [0.06176267 0.89290857 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01406108 0.         0.        ]
```

```
 [0.         0.00911523 0.        ]
 [0.         0.         0.01749718]]
Resulting covariance matrix for run date DatetimeIndex(['2013-11-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00415199 0.00032628 0.0003191 ]
 [0.00032628 0.00174484 0.00299063]
 [0.0003191  0.00299063 0.00642918]]
Expected returns vector for run date DatetimeIndex(['2013-11-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00780815 -0.00313103 -0.00661164]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-11-29'],
 ↪  dtype='datetime64[ns]')
[1.66533454e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2013-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2013-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.08728718 -0.19247787]
 [-0.08728718  1.          0.9047367 ]
 [-0.19247787  0.9047367   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2013-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00964581 0.         0.        ]
 [0.         0.00821958 0.        ]
 [0.         0.         0.01283708]]
Resulting covariance matrix for run date DatetimeIndex(['2013-12-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00195388 -0.00014533 -0.0005005 ]
 [-0.00014533  0.00141879  0.00200474]
 [-0.0005005   0.00200474  0.0034606 ]]
Expected returns vector for run date DatetimeIndex(['2013-12-30'],
 ↪  dtype='datetime64[ns]'):
[-0.01003593 -0.00347206 -0.00696393]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2013-12-30'],
 ↪  dtype='datetime64[ns]')
[2.77555756e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2014-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.27920669 -0.22330597]
 [-0.27920669  1.          0.90339375]
 [-0.22330597  0.90339375  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-01-30'],
 ↪  dtype='datetime64[ns]'):
[[0.0098764  0.         0.        ]
 [0.         0.00886152 0.        ]
 [0.         0.         0.01503783]]
Resulting covariance matrix for run date DatetimeIndex(['2014-01-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00136561 -0.00034211 -0.00046431]
 [-0.00034211  0.00109937  0.00168538]
 [-0.00046431  0.00168538  0.00316591]]
Expected returns vector for run date DatetimeIndex(['2014-01-30'],
 ↪  dtype='datetime64[ns]'):
[-0.01168829 -0.00401513 -0.00599736]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-01-30'],
 ↪ dtype='datetime64[ns]')
[1.38777878e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-02-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-02-26'], dtype='datetime64[ns]'):
[[ 1.         -0.2860077  -0.16701534]
 [-0.2860077   1.          0.89460808]
 [-0.16701534  0.89460808  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-02-26'],
 ↪ dtype='datetime64[ns]'):
[[0.01053836 0.         0.        ]
 [0.         0.0065721  0.        ]
 [0.         0.         0.01271659]]
Resulting covariance matrix for run date DatetimeIndex(['2014-02-26'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00166586 -0.00029713 -0.00033573]
 [-0.00029713  0.00064789  0.0011215 ]
 [-0.00033573  0.0011215   0.00242567]]
Expected returns vector for run date DatetimeIndex(['2014-02-26'],
 ↪ dtype='datetime64[ns]'):
[-0.00904349 -0.00214942 -0.00624197]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-02-26'],
 ↪ dtype='datetime64[ns]')
[1.94289029e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-03-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-03-26'], dtype='datetime64[ns]'):
[[ 1.         -0.03139389  0.07291167]
 [-0.03139389  1.          0.91758221]
 [ 0.07291167  0.91758221  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-03-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00863321 0.         0.        ]
 [0.         0.00834007 0.        ]
 [0.         0.         0.01442384]]
Resulting covariance matrix for run date DatetimeIndex(['2014-03-26'],
 ↪ dtype='datetime64[ns]'):
[[ 1.11798481e-03 -3.39061328e-05  1.36188860e-04]
 [-3.39061328e-05  1.04335059e-03  1.65571923e-03]
 [ 1.36188860e-04  1.65571923e-03  3.12070810e-03]]
Expected returns vector for run date DatetimeIndex(['2014-03-26'],
 ↪ dtype='datetime64[ns]'):
[-0.00908285 -0.00082783 -0.00540924]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-03-26'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.70575359e-14 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-04-29'], dtype='datetime64[ns]'):
[[1.         0.13929395 0.06056601]
```

```
 [0.13929395 1.         0.87315881]
 [0.06056601 0.87315881 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00815585 0.         0.        ]
 [0.         0.00973807 0.        ]
 [0.         0.         0.01431903]]
Resulting covariance matrix for run date DatetimeIndex(['2014-04-29'],
 ↪ dtype='datetime64[ns]'):
[[9.31250608e-04 1.54882553e-04 9.90238379e-05]
 [1.54882553e-04 1.32762035e-03 1.70454191e-03]
 [9.90238379e-05 1.70454191e-03 2.87048304e-03]]
Expected returns vector for run date DatetimeIndex(['2014-04-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00916105 -0.00154363 -0.00516837]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-04-29'],
 ↪ dtype='datetime64[ns]')
[2.35922393e-16 4.74806882e-01 5.25193118e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-05-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-05-30'], dtype='datetime64[ns]'):
[[ 1.         -0.02107771 -0.14792544]
 [-0.02107771  1.          0.84836584]
 [-0.14792544  0.84836584  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0084109  0.         0.        ]
 [0.         0.00958229 0.        ]
 [0.         0.         0.0142023 ]]
Resulting covariance matrix for run date DatetimeIndex(['2014-05-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.20263476e-03 -2.88791414e-05 -3.00395213e-04]
 [-2.88791414e-05  1.56094553e-03  1.96272891e-03]
 [-3.00395213e-04  1.96272891e-03  3.42899255e-03]]
Expected returns vector for run date DatetimeIndex(['2014-05-30'],
 ↪ dtype='datetime64[ns]'):
[-0.01017274 -0.00147469 -0.00441862]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-05-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.11022302e-15 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-06-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-06-27'], dtype='datetime64[ns]'):
[[ 1.         -0.04511911 -0.09793016]
 [-0.04511911  1.          0.87243438]
 [-0.09793016  0.87243438  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-06-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00853985 0.         0.        ]
 [0.         0.00898804 0.        ]
 [0.         0.         0.01401523]]
Resulting covariance matrix for run date DatetimeIndex(['2014-06-27'],
 ↪ dtype='datetime64[ns]'):
```

```
[[ 1.38565290e-03 -6.58005595e-05 -2.22700269e-04]
 [-6.58005595e-05  1.53491289e-03  2.08810198e-03]
 [-2.22700269e-04  2.08810198e-03  3.73210745e-03]]
Expected returns vector for run date DatetimeIndex(['2014-06-27'],
 ↪ dtype='datetime64[ns]'):
[-0.01058218  0.00043093 -0.00158576]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-06-27'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 3.05311332e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.33046931 -0.43096885]
 [-0.33046931  1.          0.8468231 ]
 [-0.43096885  0.8468231   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00760742 0.         0.        ]
 [0.         0.00795357 0.        ]
 [0.         0.         0.01154455]]
Resulting covariance matrix for run date DatetimeIndex(['2014-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00109958 -0.00037991 -0.00071914]
 [-0.00037991  0.00120193  0.00147736]
 [-0.00071914  0.00147736  0.00253226]]
Expected returns vector for run date DatetimeIndex(['2014-07-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00849755  0.00132147 -0.00162703]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-07-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 9.99200722e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-08-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-08-28'], dtype='datetime64[ns]'):
[[ 1.         -0.56900257 -0.5954448 ]
 [-0.56900257  1.          0.87622523]
 [-0.5954448   0.87622523  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-08-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00587203 0.         0.        ]
 [0.         0.0080974  0.        ]
 [0.         0.         0.01187865]]
Resulting covariance matrix for run date DatetimeIndex(['2014-08-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.0007241  -0.00056816 -0.0008722 ]
 [-0.00056816  0.00137693  0.0017699 ]
 [-0.0008722   0.0017699   0.00296315]]
Expected returns vector for run date DatetimeIndex(['2014-08-28'],
 ↪ dtype='datetime64[ns]'):
[-0.01058812  0.00256705 -0.00079344]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-08-28'],
 ↪ dtype='datetime64[ns]')
[7.80625564e-17 1.00000000e+00 5.55111512e-17]
```

```
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-09-29'], dtype='datetime64[ns]'):
[[1.         0.10227144 0.15848839]
 [0.10227144 1.         0.78675073]
 [0.15848839 0.78675073 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00921515 0.         0.        ]
 [0.         0.00843167 0.        ]
 [0.         0.         0.01339074]]
Resulting covariance matrix for run date DatetimeIndex(['2014-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0013587  0.00012714 0.00031291]
 [0.00012714 0.00113749 0.00142127]
 [0.00031291 0.00142127 0.00286899]]
Expected returns vector for run date DatetimeIndex(['2014-09-29'],
 ↪ dtype='datetime64[ns]'):
[-0.01317929  0.00594394  0.00375868]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-09-29'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-10-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-10-30'], dtype='datetime64[ns]'):
[[ 1.         -0.30374676 -0.22160156]
 [-0.30374676  1.          0.9016369 ]
 [-0.22160156  0.9016369   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0137384  0.         0.        ]
 [0.         0.00622605 0.        ]
 [0.         0.         0.0104168 ]]
Resulting covariance matrix for run date DatetimeIndex(['2014-10-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00264241 -0.00036374 -0.00044399]
 [-0.00036374  0.00054269  0.00081867]
 [-0.00044399  0.00081867  0.00151914]]
Expected returns vector for run date DatetimeIndex(['2014-10-30'],
 ↪ dtype='datetime64[ns]'):
[-0.01153743  0.00592213  0.00294576]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-10-30'],
 ↪ dtype='datetime64[ns]')
[1.94289029e-16 1.00000000e+00 0.00000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2014-11-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2014-11-28'], dtype='datetime64[ns]'):
[[ 1.         -0.09616989 -0.03220156]
 [-0.09616989  1.          0.9079845 ]
 [-0.03220156  0.9079845   1.        ]]
```

```
Volatility diagonal vector for run date DatetimeIndex(['2014-11-28'],
 ↪  dtype='datetime64[ns]'):
[[0.013301   0.         0.        ]
 [0.         0.00822125 0.        ]
 [0.         0.         0.01111062]]
Resulting covariance matrix for run date DatetimeIndex(['2014-11-28'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00389216 -0.00023136 -0.00010469]
 [-0.00023136  0.00148696  0.00182464]
 [-0.00010469  0.00182464  0.00271581]]
Expected returns vector for run date DatetimeIndex(['2014-11-28'],
 ↪  dtype='datetime64[ns]'):
[-0.01308551  0.00442676  0.00363705]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-11-28'],
 ↪  dtype='datetime64[ns]')
[8.48930301e-17 1.00000000e+00 3.99680289e-15]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2014-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2014-12-30'], dtype='datetime64[ns]'):
[[1.         0.05348951 0.14150611]
 [0.05348951 1.         0.90288919]
 [0.14150611 0.90288919 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2014-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01019849 0.         0.        ]
 [0.         0.01052444 0.        ]
 [0.         0.         0.0136742 ]]
Resulting covariance matrix for run date DatetimeIndex(['2014-12-30'],
 ↪  dtype='datetime64[ns]'):
[[1.66414710e-03 9.18593424e-05 3.15742490e-04]
 [9.18593424e-05 1.77222033e-03 2.07900391e-03]
 [3.15742490e-04 2.07900391e-03 2.99174097e-03]]
Expected returns vector for run date DatetimeIndex(['2014-12-30'],
 ↪  dtype='datetime64[ns]'):
[-0.0124969   0.00811863  0.00956831]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2014-12-30'],
 ↪  dtype='datetime64[ns]')
[4.16333634e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-01-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2015-01-27'], dtype='datetime64[ns]'):
[[1.         0.03316185 0.1153048 ]
 [0.03316185 1.         0.86055112]
 [0.1153048  0.86055112 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-01-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00723301 0.         0.        ]
 [0.         0.00817499 0.        ]
 [0.         0.         0.0136104 ]]
Resulting covariance matrix for run date DatetimeIndex(['2015-01-27'],
 ↪  dtype='datetime64[ns]'):
[[9.41696206e-04 3.52953703e-05 2.04319599e-04]
 [3.52953703e-05 1.20294894e-03 1.72348501e-03]
 [2.04319599e-04 1.72348501e-03 3.33437528e-03]]
Expected returns vector for run date DatetimeIndex(['2015-01-27'],
 ↪  dtype='datetime64[ns]'):
[-0.0093191   0.00871348  0.01216366]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-01-27'],
 ↪  dtype='datetime64[ns]')
[8.9338259e-17 0.0000000e+00 1.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-02-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2015-02-27'], dtype='datetime64[ns]'):
[[ 1.         -0.10064611 -0.15010172]
 [-0.10064611  1.          0.86143333]
 [-0.15010172  0.86143333  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-02-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00786147 0.         0.        ]
 [0.         0.0088535  0.        ]
 [0.         0.         0.01331828]]
Resulting covariance matrix for run date DatetimeIndex(['2015-02-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00117425 -0.0001331  -0.0002986 ]
 [-0.0001331   0.0014893   0.00192992]
 [-0.0002986   0.00192992  0.00337015]]
Expected returns vector for run date DatetimeIndex(['2015-02-27'],
 ↪  dtype='datetime64[ns]'):
[-0.00922818  0.00746955  0.00866028]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-02-27'],
 ↪  dtype='datetime64[ns]')
[0.         0.45410983 0.54589017]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-03-30'], dtype='datetime64[ns]'):
[[1.         0.07458649 0.02301494]
 [0.07458649 1.         0.75743341]
 [0.02301494 0.75743341 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-03-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01252552 0.         0.        ]
 [0.         0.00997266 0.        ]
 [0.         0.         0.01313188]]
Resulting covariance matrix for run date DatetimeIndex(['2015-03-30'],
 ↪  dtype='datetime64[ns]'):
[[2.98088318e-03 1.77019163e-04 7.19260290e-05]
 [1.77019163e-04 1.88962354e-03 1.88467204e-03]
 [7.19260290e-05 1.88467204e-03 3.27647881e-03]]
Expected returns vector for run date DatetimeIndex(['2015-03-30'],
 ↪  dtype='datetime64[ns]'):
[-0.00982873  0.00659383  0.00646907]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-03-30'],
 ↪  dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.60982339e-15]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-04-29'], dtype='datetime64[ns]'):
[[1.         0.12852512 0.07604013]
 [0.12852512 1.         0.86148846]
 [0.07604013 0.86148846 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01014181 0.         0.        ]
 [0.         0.01076485 0.        ]
 [0.         0.         0.01538928]]
Resulting covariance matrix for run date DatetimeIndex(['2015-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00195427 0.0002666  0.00022549]
 [0.0002666  0.00220176 0.00271162]
 [0.00022549 0.00271162 0.00449977]]
Expected returns vector for run date DatetimeIndex(['2015-04-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00916416  0.00640874  0.00564463]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-04-29'],
 ↪  dtype='datetime64[ns]')
[1.90819582e-17 1.00000000e+00 1.11022302e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-05-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2015-05-29'], dtype='datetime64[ns]'):
[[ 1.          0.02391038 -0.06314961]
 [ 0.02391038  1.          0.91483319]
 [-0.06314961  0.91483319  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-05-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00768627 0.         0.        ]
 [0.         0.01005317 0.        ]
 [0.         0.         0.01354192]]
Resulting covariance matrix for run date DatetimeIndex(['2015-05-29'],
 ↪  dtype='datetime64[ns]'):
[[ 1.18157477e-03  3.69517563e-05 -1.31460891e-04]
 [ 3.69517563e-05  2.02132451e-03  2.49089395e-03]
 [-1.31460891e-04  2.49089395e-03  3.66767301e-03]]
Expected returns vector for run date DatetimeIndex(['2015-05-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00848753  0.00572537  0.00584531]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-05-29'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 4.75710110e-01 5.24289890e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-06-29'], dtype='datetime64[ns]'):
[[ 1.         -0.22129458 -0.16326189]
 [-0.22129458  1.          0.89418793]
 [-0.16326189  0.89418793  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00772195 0.         0.        ]
```

```
 [0.         0.00886774 0.        ]
 [0.         0.         0.01131384]]
Resulting covariance matrix for run date DatetimeIndex(['2015-06-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00131183 -0.00033338 -0.00031379]
 [-0.00033338  0.00173001  0.00197367]
 [-0.00031379  0.00197367  0.00281607]]
Expected returns vector for run date DatetimeIndex(['2015-06-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00642929  0.00899668  0.00964197]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-06-29'],
 ↪ dtype='datetime64[ns]')
[1.65666092e-16 8.92317773e-01 1.07682227e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2015-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.12123573 -0.25561678]
 [-0.12123573  1.          0.91977805]
 [-0.25561678  0.91977805  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00895246 0.         0.        ]
 [0.         0.01422271 0.        ]
 [0.         0.         0.01895024]]
Resulting covariance matrix for run date DatetimeIndex(['2015-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00168308 -0.00032417 -0.00091068]
 [-0.00032417  0.00424799  0.00520594]
 [-0.00091068  0.00520594  0.00754135]]
Expected returns vector for run date DatetimeIndex(['2015-07-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00736559  0.006632    0.00694907]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-07-30'],
 ↪ dtype='datetime64[ns]')
[0.         0.94458754 0.05541246]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2015-08-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-08-28'], dtype='datetime64[ns]'):
[[1.         0.11123716 0.16114008]
 [0.11123716 1.         0.91492766]
 [0.16114008 0.91492766 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-08-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01085543 0.         0.        ]
 [0.         0.01442575 0.        ]
 [0.         0.         0.02106585]]
Resulting covariance matrix for run date DatetimeIndex(['2015-08-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00223897 0.00033097 0.00070014]
 [0.00033097 0.00395394 0.00528272]
 [0.00070014 0.00528272 0.00843163]]
Expected returns vector for run date DatetimeIndex(['2015-08-28'],
 ↪ dtype='datetime64[ns]'):
[-0.00936498  0.0073814   0.0074204 ]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-08-28'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 2.22044605e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2015-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-09-29'], dtype='datetime64[ns]'):
[[1.         0.03457641 0.01349713]
 [0.03457641 1.         0.80490208]
 [0.01349713 0.80490208 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00820101 0.         0.        ]
 [0.         0.00812138 0.        ]
 [0.         0.         0.01154089]]
Resulting covariance matrix for run date DatetimeIndex(['2015-09-29'],
 ↪ dtype='datetime64[ns]'):
[[1.27787606e-03 4.37553448e-05 2.42718006e-05]
 [4.37553448e-05 1.25318078e-03 1.43339573e-03]
 [2.42718006e-05 1.43339573e-03 2.53065181e-03]]
Expected returns vector for run date DatetimeIndex(['2015-09-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00955889  0.00533413  0.00606768]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-09-29'],
 ↪ dtype='datetime64[ns]')
[1.80411242e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2015-10-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2015-10-30'], dtype='datetime64[ns]'):
[[ 1.         -0.14169706 -0.05101946]
 [-0.14169706  1.          0.82480663]
 [-0.05101946  0.82480663  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00800813 0.         0.        ]
 [0.         0.00750118 0.        ]
 [0.         0.         0.00973704]]
Resulting covariance matrix for run date DatetimeIndex(['2015-10-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.15434153e-03 -1.53212307e-04 -7.16088104e-05]
 [-1.53212307e-04  1.01281770e-03  1.08437924e-03]
 [-7.16088104e-05  1.08437924e-03  1.70658067e-03]]
Expected returns vector for run date DatetimeIndex(['2015-10-30'],
 ↪ dtype='datetime64[ns]'):
[-0.0111138   0.0042422   0.00514429]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-10-30'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2015-11-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-11-27'], dtype='datetime64[ns]'):
[[ 1.         -0.26037642 -0.16267948]
 [-0.26037642  1.          0.88642341]
 [-0.16267948  0.88642341  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-11-27'],
 ↪  dtype='datetime64[ns]'):
[[0.01014568 0.         0.        ]
 [0.         0.00889053 0.        ]
 [0.         0.         0.01167578]]
Resulting covariance matrix for run date DatetimeIndex(['2015-11-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00226456 -0.00051669 -0.00042396]
 [-0.00051669  0.00173891  0.00202431]
 [-0.00042396  0.00202431  0.00299912]]
Expected returns vector for run date DatetimeIndex(['2015-11-27'],
 ↪  dtype='datetime64[ns]'):
[-0.01059355  0.00448614  0.0048518 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-11-27'],
 ↪  dtype='datetime64[ns]')
[3.46944695e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2015-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2015-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.44312659 -0.40172502]
 [-0.44312659  1.          0.88543153]
 [-0.40172502  0.88543153  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2015-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.01052431 0.         0.        ]
 [0.         0.01069133 0.        ]
 [0.         0.         0.01445662]]
Resulting covariance matrix for run date DatetimeIndex(['2015-12-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.0019937  -0.00089748 -0.00110017]
 [-0.00089748  0.00205748  0.00246335]
 [-0.00110017  0.00246335  0.00376189]]
Expected returns vector for run date DatetimeIndex(['2015-12-30'],
 ↪  dtype='datetime64[ns]'):
[-0.01252172  0.0026553   0.00288702]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2015-12-30'],
 ↪  dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-01-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2016-01-29'], dtype='datetime64[ns]'):
[[ 1.         -0.39677683 -0.32804772]
 [-0.39677683  1.          0.90773469]
 [-0.32804772  0.90773469  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-01-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01511037 0.         0.        ]
 [0.         0.01302796 0.        ]
 [0.         0.         0.01602623]]
Resulting covariance matrix for run date DatetimeIndex(['2016-01-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00456647 -0.00156217 -0.00158882]
```

```
 [-0.00156217  0.00339455  0.0037905 ]
 [-0.00158882  0.0037905   0.0051368 ]]
Expected returns vector for run date DatetimeIndex(['2016-01-29'],
 ↪  dtype='datetime64[ns]'):
[-0.01207305  0.0025751   0.00135384]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-01-29'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-02-26'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-02-26'], dtype='datetime64[ns]'):
[[ 1.         -0.11188203 -0.04802052]
 [-0.11188203  1.          0.82204467]
 [-0.04802052  0.82204467  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-02-26'],
 ↪  dtype='datetime64[ns]'):
[[0.01180999 0.         0.        ]
 [0.         0.01124672 0.        ]
 [0.         0.         0.01599233]]
Resulting covariance matrix for run date DatetimeIndex(['2016-02-26'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00278952 -0.00029721 -0.00018139]
 [-0.00029721  0.00252977  0.00295708]
 [-0.00018139  0.00295708  0.00511509]]
Expected returns vector for run date DatetimeIndex(['2016-02-26'],
 ↪  dtype='datetime64[ns]'):
[-0.0107909   0.00116386 -0.00107719]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-02-26'],
 ↪  dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-03-30'], dtype='datetime64[ns]'):
[[ 1.         -0.24201667 -0.26827234]
 [-0.24201667  1.          0.81761295]
 [-0.26827234  0.81761295  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-03-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00893022 0.         0.        ]
 [0.         0.00935104 0.        ]
 [0.         0.         0.01310908]]
Resulting covariance matrix for run date DatetimeIndex(['2016-03-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00143548 -0.00036378 -0.0005653 ]
 [-0.00036378  0.00157395  0.00180407]
 [-0.0005653   0.00180407  0.00309326]]
Expected returns vector for run date DatetimeIndex(['2016-03-30'],
 ↪  dtype='datetime64[ns]'):
[-6.58863527e-03 -8.33112831e-05 -2.14652631e-03]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-03-30'],
 ↪  dtype='datetime64[ns]')
[0.00000000e+00 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-04-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2016-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.21279702 -0.23397478]
 [-0.21279702  1.          0.94550216]
 [-0.23397478  0.94550216  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.0091125  0.         0.        ]
 [0.         0.00874756 0.        ]
 [0.         0.         0.01176033]]
Resulting covariance matrix for run date DatetimeIndex(['2016-04-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00174379 -0.00035621 -0.00052656]
 [-0.00035621  0.00160692  0.00204262]
 [-0.00052656  0.00204262  0.00290441]]
Expected returns vector for run date DatetimeIndex(['2016-04-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00614252  0.00202807  0.00100845]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-04-29'],
 ↪  dtype='datetime64[ns]')
[1.94289029e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-05-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-05-27'], dtype='datetime64[ns]'):
[[ 1.         -0.52948725 -0.43441066]
 [-0.52948725  1.          0.84692669]
 [-0.43441066  0.84692669  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-05-27'],
 ↪  dtype='datetime64[ns]'):
[[0.01166821 0.         0.        ]
 [0.         0.00833466 0.        ]
 [0.         0.         0.01104782]]
Resulting covariance matrix for run date DatetimeIndex(['2016-05-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00285909 -0.00108135 -0.00117598]
 [-0.00108135  0.0014588   0.00163768]
 [-0.00117598  0.00163768  0.00256314]]
Expected returns vector for run date DatetimeIndex(['2016-05-27'],
 ↪  dtype='datetime64[ns]'):
[-0.00174045  0.00161548 -0.00027437]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-05-27'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2016-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-06-29'], dtype='datetime64[ns]'):
[[ 1.         -0.38770047 -0.47950554]
 [-0.38770047  1.          0.87639624]
 [-0.47950554  0.87639624  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00829804 0.         0.        ]
```

```
 [0.         0.00653674 0.        ]
 [0.         0.         0.00971522]]
Resulting covariance matrix for run date DatetimeIndex(['2016-06-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00130829 -0.00039956 -0.00073447]
 [-0.00039956  0.00081185  0.00105747]
 [-0.00073447  0.00105747  0.00179333]]
Expected returns vector for run date DatetimeIndex(['2016-06-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00234062  0.00287363  0.00171124]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-06-29'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-07-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2016-07-29'], dtype='datetime64[ns]'):
[[ 1.         -0.44530338 -0.39717835]
 [-0.44530338  1.          0.85288829]
 [-0.39717835  0.85288829  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-07-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00710793 0.         0.        ]
 [0.         0.00684831 0.        ]
 [0.         0.         0.00932273]]
Resulting covariance matrix for run date DatetimeIndex(['2016-07-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.0011115  -0.00047688 -0.00057902]
 [-0.00047688  0.00103178  0.00119796]
 [-0.00057902  0.00119796  0.00191209]]
Expected returns vector for run date DatetimeIndex(['2016-07-29'],
 ↪ dtype='datetime64[ns]'):
[0.00330028 0.0042038  0.00456782]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-07-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 2.77555756e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-08-30'], dtype='datetime64[ns]'):
[[1.         0.21359791 0.3367036 ]
 [0.21359791 1.         0.91755289]
 [0.3367036  0.91755289 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00748019 0.         0.        ]
 [0.         0.00797873 0.        ]
 [0.         0.         0.01111348]]
Resulting covariance matrix for run date DatetimeIndex(['2016-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00111906 0.00025496 0.00055981]
 [0.00025496 0.0012732  0.00162722]
 [0.00055981 0.00162722 0.00247019]]
Expected returns vector for run date DatetimeIndex(['2016-08-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00182253 0.00601098 0.01010604]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-08-30'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-09-29'], dtype='datetime64[ns]'):
[[ 1.         -0.2682932  -0.18913147]
 [-0.2682932   1.          0.90719831]
 [-0.18913147  0.90719831  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01001575 0.         0.        ]
 [0.         0.00948573 0.        ]
 [0.         0.         0.01216412]]
Resulting covariance matrix for run date DatetimeIndex(['2016-09-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00170536 -0.00043332 -0.00039172]
 [-0.00043332  0.00152964  0.00177952]
 [-0.00039172  0.00177952  0.00251542]]
Expected returns vector for run date DatetimeIndex(['2016-09-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00025353  0.00782079  0.01412182]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-09-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-10-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2016-10-28'], dtype='datetime64[ns]'):
[[ 1.         -0.50105786 -0.53176355]
 [-0.50105786  1.          0.79997075]
 [-0.53176355  0.79997075  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-10-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00885618 0.         0.        ]
 [0.         0.0097971  0.        ]
 [0.         0.         0.01398454]]
Resulting covariance matrix for run date DatetimeIndex(['2016-10-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00149021 -0.00082601 -0.00125131]
 [-0.00082601  0.00182368  0.00208244]
 [-0.00125131  0.00208244  0.00371578]]
Expected returns vector for run date DatetimeIndex(['2016-10-28'],
 ↪ dtype='datetime64[ns]'):
[0.00134678 0.00635325 0.01199125]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-10-28'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2016-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.18752247 -0.36065608]
```

```
 [-0.18752247  1.          0.85904169]
 [-0.36065608  0.85904169  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00986386 0.         0.        ]
 [0.         0.01242128 0.        ]
 [0.         0.         0.01174472]]
Resulting covariance matrix for run date DatetimeIndex(['2016-11-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00204321 -0.00048249 -0.00087741]
 [-0.00048249  0.00324005  0.00263174]
 [-0.00087741  0.00263174  0.00289671]]
Expected returns vector for run date DatetimeIndex(['2016-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00148062 0.00379913 0.00750881]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-11-29'],
 ↪ dtype='datetime64[ns]')
[7.2858386e-17 0.0000000e+00 1.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2016-12-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2016-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.27075812 -0.26917744]
 [-0.27075812  1.          0.86649108]
 [-0.26917744  0.86649108  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2016-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00796354 0.         0.        ]
 [0.         0.0072777  0.        ]
 [0.         0.         0.0102603 ]]
Resulting covariance matrix for run date DatetimeIndex(['2016-12-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00114152 -0.00028246 -0.00039589]
 [-0.00028246  0.00095337  0.00116464]
 [-0.00039589  0.00116464  0.00189493]]
Expected returns vector for run date DatetimeIndex(['2016-12-30'],
 ↪ dtype='datetime64[ns]'):
[0.00180984 0.00422093 0.00804395]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2016-12-30'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.32379591 -0.18827483]
 [-0.32379591  1.          0.88241184]
 [-0.18827483  0.88241184  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00648914 0.         0.        ]
 [0.         0.00708991 0.        ]
 [0.         0.         0.00911344]]
Resulting covariance matrix for run date DatetimeIndex(['2017-01-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[[ 0.00071585 -0.00025325 -0.00018928]
 [-0.00025325  0.00085454  0.00096927]
 [-0.00018928  0.00096927  0.00141193]]
Expected returns vector for run date DatetimeIndex(['2017-01-30'],
 ↪ dtype='datetime64[ns]'):
[0.00241524 0.00379144 0.00715073]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-01-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 3.33066907e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-02-27'], dtype='datetime64[ns]'):
[[1.         0.13427283 0.03947796]
 [0.13427283 1.         0.88834274]
 [0.03947796 0.88834274 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00798192 0.         0.        ]
 [0.         0.00740533 0.        ]
 [0.         0.         0.00979622]]
Resulting covariance matrix for run date DatetimeIndex(['2017-02-27'],
 ↪ dtype='datetime64[ns]'):
[[1.40164292e-03 1.74607453e-04 6.79115214e-05]
 [1.74607453e-04 1.20645712e-03 1.41777258e-03]
 [6.79115214e-05 1.41777258e-03 2.11125233e-03]]
Expected returns vector for run date DatetimeIndex(['2017-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.0031243  0.00609306 0.01222718]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-02-27'],
 ↪ dtype='datetime64[ns]')
[7.97972799e-17 3.46944695e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-03-30'], dtype='datetime64[ns]'):
[[ 1.         -0.12526844 -0.04111363]
 [-0.12526844  1.          0.75149822]
 [-0.04111363  0.75149822  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00681667 0.         0.        ]
 [0.         0.0051644  0.        ]
 [0.         0.         0.00630495]]
Resulting covariance matrix for run date DatetimeIndex(['2017-03-30'],
 ↪ dtype='datetime64[ns]'):
[[ 8.36404634e-04 -7.93790551e-05 -3.18062345e-05]
 [-7.93790551e-05  4.80078272e-04  4.40455667e-04]
 [-3.18062345e-05  4.40455667e-04  7.15544155e-04]]
Expected returns vector for run date DatetimeIndex(['2017-03-30'],
 ↪ dtype='datetime64[ns]'):
[0.00208349 0.00690783 0.01352316]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-03-30'],
 ↪ dtype='datetime64[ns]')
[5.55111512e-17 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-04-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2017-04-28'], dtype='datetime64[ns]'):
[[ 1.          0.2332473  -0.04929648]
 [ 0.2332473   1.          0.82754612]
 [-0.04929648  0.82754612  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-04-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00701918 0.         0.        ]
 [0.         0.00575317 0.        ]
 [0.         0.         0.008518  ]]
Resulting covariance matrix for run date DatetimeIndex(['2017-04-28'],
 ↪ dtype='datetime64[ns]'):
[[ 1.03464688e-03  1.97801589e-04 -6.18955574e-05]
 [ 1.97801589e-04  6.95078997e-04  8.51641502e-04]
 [-6.18955574e-05  8.51641502e-04  1.52368452e-03]]
Expected returns vector for run date DatetimeIndex(['2017-04-28'],
 ↪ dtype='datetime64[ns]'):
[0.00197177 0.00700485 0.01237054]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-04-28'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 2.77555756e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-05-30'], dtype='datetime64[ns]'):
[[ 1.         -0.00635356  0.00903681]
 [-0.00635356  1.          0.87737983]
 [ 0.00903681  0.87737983  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0070265 0.        0.       ]
 [0.        0.0041945 0.       ]
 [0.        0.        0.0067676]]
Resulting covariance matrix for run date DatetimeIndex(['2017-05-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.03680714e-03 -3.93238304e-06  9.02419026e-06]
 [-3.93238304e-06  3.69470504e-04  5.23024069e-04]
 [ 9.02419026e-06  5.23024069e-04  9.61807766e-04]]
Expected returns vector for run date DatetimeIndex(['2017-05-30'],
 ↪ dtype='datetime64[ns]'):
[0.00259549 0.00756081 0.01289188]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-05-30'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-06-29'], dtype='datetime64[ns]'):
[[ 1.         -0.25921607 -0.05551588]
 [-0.25921607  1.          0.81722099]
 [-0.05551588  0.81722099  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00696037 0.         0.        ]
```

```
 [0.         0.00578931 0.        ]
 [0.         0.         0.00726816]]
Resulting covariance matrix for run date DatetimeIndex(['2017-06-29'],
 ↪ dtype='datetime64[ns]'):
[[ 6.29807356e-04 -1.35788915e-04 -3.65104755e-05]
 [-1.35788915e-04  4.35709535e-04  4.47027721e-04]
 [-3.65104755e-05  4.47027721e-04  6.86740761e-04]]
Expected returns vector for run date DatetimeIndex(['2017-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.00378999 0.00723921 0.0122759 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-06-29'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-07-20'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2017-07-20'], dtype='datetime64[ns]'):
[[ 1.         -0.14386329 -0.12514277]
 [-0.14386329  1.          0.89595056]
 [-0.12514277  0.89595056  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-07-20'],
 ↪ dtype='datetime64[ns]'):
[[0.00570014 0.         0.        ]
 [0.         0.00705725 0.        ]
 [0.         0.         0.00833159]]
Resulting covariance matrix for run date DatetimeIndex(['2017-07-20'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00064983 -0.00011574 -0.00011886]
 [-0.00011574  0.0009961   0.0010536 ]
 [-0.00011886  0.0010536   0.00138831]]
Expected returns vector for run date DatetimeIndex(['2017-07-20'],
 ↪ dtype='datetime64[ns]'):
[0.00184748 0.00562493 0.01150427]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-07-20'],
 ↪ dtype='datetime64[ns]')
[1.52655666e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-08-30'], dtype='datetime64[ns]'):
[[ 1.         -0.07518411 -0.04319582]
 [-0.07518411  1.          0.91568768]
 [-0.04319582  0.91568768  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00780388 0.         0.        ]
 [0.         0.00656081 0.        ]
 [0.         0.         0.0072649 ]]
Resulting covariance matrix for run date DatetimeIndex(['2017-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.21800945e-03 -7.69881366e-05 -4.89791895e-05]
 [-7.69881366e-05  8.60884289e-04  8.72899592e-04]
 [-4.89791895e-05  8.72899592e-04  1.05557491e-03]]
Expected returns vector for run date DatetimeIndex(['2017-08-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00288318 0.00678545 0.01333802]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-08-30'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-09-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2017-09-29'], dtype='datetime64[ns]'):
[[1.         0.27968478 0.27945137]
 [0.27968478 1.         0.75710046]
 [0.27945137 0.75710046 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0056723  0.         0.        ]
 [0.         0.00529302 0.        ]
 [0.         0.         0.01013079]]
Resulting covariance matrix for run date DatetimeIndex(['2017-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0006435  0.00016794 0.00032117]
 [0.00016794 0.00056032 0.00081195]
 [0.00032117 0.00081195 0.00205266]]
Expected returns vector for run date DatetimeIndex(['2017-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.00355832 0.00541379 0.0115869 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-09-29'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-10-30'], dtype='datetime64[ns]'):
[[1.         0.09230921 0.02821162]
 [0.09230921 1.         0.85156471]
 [0.02821162 0.85156471 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00697021 0.         0.        ]
 [0.         0.0053561  0.        ]
 [0.         0.         0.00638565]]
Resulting covariance matrix for run date DatetimeIndex(['2017-10-30'],
 ↪ dtype='datetime64[ns]'):
[[1.02025997e-03 7.23699746e-05 2.63692870e-05]
 [7.23699746e-05 6.02442991e-04 6.11632448e-04]
 [2.63692870e-05 6.11632448e-04 8.56307549e-04]]
Expected returns vector for run date DatetimeIndex(['2017-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.00449249 0.00543971 0.01260502]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-10-30'],
 ↪ dtype='datetime64[ns]')
[2.49800181e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2017-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2017-11-29'], dtype='datetime64[ns]'):
[[1.         0.01282629 0.11883701]
```

```
 [0.01282629 1.         0.85296226]
 [0.11883701 0.85296226 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-11-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00598488 0.         0.        ]
 [0.         0.00613541 0.        ]
 [0.         0.         0.00951712]]
Resulting covariance matrix for run date DatetimeIndex(['2017-11-29'],
 ↪  dtype='datetime64[ns]'):
[[6.08919034e-04 8.00660620e-06 1.15069847e-04]
 [8.00660620e-06 6.39934510e-04 8.46696250e-04]
 [1.15069847e-04 8.46696250e-04 1.53978511e-03]]
Expected returns vector for run date DatetimeIndex(['2017-11-29'],
 ↪  dtype='datetime64[ns]'):
[0.00497705 0.00636998 0.01151897]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-11-29'],
 ↪  dtype='datetime64[ns]')
[0. 0. 1.]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2017-12-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2017-12-26'], dtype='datetime64[ns]'):
[[1.         0.38448593 0.21425185]
 [0.38448593 1.         0.939852  ]
 [0.21425185 0.939852   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2017-12-26'],
 ↪  dtype='datetime64[ns]'):
[[0.00695961 0.         0.        ]
 [0.         0.00851941 0.        ]
 [0.         0.         0.01106516]]
Resulting covariance matrix for run date DatetimeIndex(['2017-12-26'],
 ↪  dtype='datetime64[ns]'):
[[0.0005328  0.00025077 0.00018149]
 [0.00025077 0.00079838 0.00097458]
 [0.00018149 0.00097458 0.00134682]]
Expected returns vector for run date DatetimeIndex(['2017-12-26'],
 ↪  dtype='datetime64[ns]'):
[0.00423059 0.00536724 0.01048394]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2017-12-26'],
 ↪  dtype='datetime64[ns]')
[1.92012205e-16 2.58921031e-17 1.00000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2018-01-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2018-01-30'], dtype='datetime64[ns]'):
[[1.         0.18112423 0.14888763]
 [0.18112423 1.         0.91269755]
 [0.14888763 0.91269755 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-01-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00769965 0.         0.        ]
 [0.         0.00846316 0.        ]
 [0.         0.         0.01090017]]
```

```
Resulting covariance matrix for run date DatetimeIndex(['2018-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00106712 0.00021245 0.00022492]
 [0.00021245 0.00128925 0.00151553]
 [0.00022492 0.00151553 0.00213865]]
Expected returns vector for run date DatetimeIndex(['2018-01-30'],
 ↪ dtype='datetime64[ns]'):
[0.0042625  0.00718973 0.01052402]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-01-30'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-02-27'], dtype='datetime64[ns]'):
[[ 1.         -0.05940539 -0.07156257]
 [-0.05940539  1.          0.89613593]
 [-0.07156257  0.89613593  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00760551 0.         0.        ]
 [0.         0.00916167 0.        ]
 [0.         0.         0.01127025]]
Resulting covariance matrix for run date DatetimeIndex(['2018-02-27'],
 ↪ dtype='datetime64[ns]'):
[[ 1.09903155e-03 -7.86470420e-05 -1.16547084e-04]
 [-7.86470420e-05  1.59478766e-03  1.75806834e-03]
 [-1.16547084e-04  1.75806834e-03  2.41335305e-03]]
Expected returns vector for run date DatetimeIndex(['2018-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.00157965 0.00610384 0.00945392]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-02-27'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-03-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2018-03-28'], dtype='datetime64[ns]'):
[[ 1.         -0.03774338 -0.06753445]
 [-0.03774338  1.          0.8581863 ]
 [-0.06753445  0.8581863   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00852907 0.         0.        ]
 [0.         0.00694919 0.        ]
 [0.         0.         0.00952726]]
Resulting covariance matrix for run date DatetimeIndex(['2018-03-28'],
 ↪ dtype='datetime64[ns]'):
[[ 1.16391924e-03 -3.57928531e-05 -8.78041362e-05]
 [-3.57928531e-05  7.72659671e-04  9.09083673e-04]
 [-8.78041362e-05  9.09083673e-04  1.45229991e-03]]
Expected returns vector for run date DatetimeIndex(['2018-03-28'],
 ↪ dtype='datetime64[ns]'):
[0.00205626 0.00481954 0.00800325]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-03-28'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2018-04-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-04-27'], dtype='datetime64[ns]'):
[[1.         0.12805384 0.19893561]
 [0.12805384 1.         0.83193058]
 [0.19893561 0.83193058 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-04-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00661794 0.         0.        ]
 [0.         0.00616229 0.        ]
 [0.         0.         0.00920753]]
Resulting covariance matrix for run date DatetimeIndex(['2018-04-27'],
 ↪  dtype='datetime64[ns]'):
[[6.56956861e-04 7.83337066e-05 1.81831682e-04]
 [7.83337066e-05 5.69606926e-04 7.08048996e-04]
 [1.81831682e-04 7.08048996e-04 1.27167809e-03]]
Expected returns vector for run date DatetimeIndex(['2018-04-27'],
 ↪  dtype='datetime64[ns]'):
[0.00210012 0.0044489  0.00846286]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-04-27'],
 ↪  dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2018-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-05-30'], dtype='datetime64[ns]'):
[[1.         0.06967662 0.13065809]
 [0.06967662 1.         0.863253  ]
 [0.13065809 0.863253   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-05-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00635003 0.         0.        ]
 [0.         0.00568624 0.        ]
 [0.         0.         0.00782725]]
Resulting covariance matrix for run date DatetimeIndex(['2018-05-30'],
 ↪  dtype='datetime64[ns]'):
[[8.46781670e-04 5.28332602e-05 1.36376926e-04]
 [5.28332602e-05 6.78999229e-04 8.06847772e-04]
 [1.36376926e-04 8.06847772e-04 1.28658279e-03]]
Expected returns vector for run date DatetimeIndex(['2018-05-30'],
 ↪  dtype='datetime64[ns]'):
[0.00148682 0.00686582 0.00975443]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-05-30'],
 ↪  dtype='datetime64[ns]')
[0.00000000e+00 8.32667268e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2018-06-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2018-06-29'], dtype='datetime64[ns]'):
[[ 1.          0.09601825 -0.04761372]
 [ 0.09601825  1.          0.75511718]
 [-0.04761372  0.75511718  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00557021 0.         0.        ]
 [0.         0.00544103 0.        ]
 [0.         0.         0.00695221]]
Resulting covariance matrix for run date DatetimeIndex(['2018-06-29'],
 ↪ dtype='datetime64[ns]'):
[[ 6.51571387e-04  6.11118761e-05 -3.87209075e-05]
 [ 6.11118761e-05  6.21701068e-04  5.99842993e-04]
 [-3.87209075e-05  5.99842993e-04  1.01499725e-03]]
Expected returns vector for run date DatetimeIndex(['2018-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.0009774  0.00501545 0.00928181]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-06-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.11022302e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-07-30'], dtype='datetime64[ns]'):
[[1.         0.3844513  0.31221986]
 [0.3844513  1.         0.87612885]
 [0.31221986 0.87612885 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00676563 0.         0.        ]
 [0.         0.00609712 0.        ]
 [0.         0.         0.00833825]]
Resulting covariance matrix for run date DatetimeIndex(['2018-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00096125 0.00033304 0.00036988]
 [0.00033304 0.00078067 0.00093538]
 [0.00036988 0.00093538 0.00146006]]
Expected returns vector for run date DatetimeIndex(['2018-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.00013327 0.00539172 0.00882951]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-07-30'],
 ↪ dtype='datetime64[ns]')
[0.0000000e+00 4.4408921e-16 1.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-08-30'], dtype='datetime64[ns]'):
[[ 1.         -0.01946941  0.15758699]
 [-0.01946941  1.          0.83319569]
 [ 0.15758699  0.83319569  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00527944 0.         0.        ]
 [0.         0.00800357 0.        ]
 [0.         0.         0.01049717]]
Resulting covariance matrix for run date DatetimeIndex(['2018-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 4.73831850e-04 -1.39853451e-05  1.48466790e-04]
 [-1.39853451e-05  1.08897155e-03  1.19001410e-03]
 [ 1.48466790e-04  1.19001410e-03  1.87324100e-03]]
Expected returns vector for run date DatetimeIndex(['2018-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.0013921  0.00641857 0.00956364]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-08-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 2.49800181e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-09-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2018-09-28'], dtype='datetime64[ns]'):
[[ 1.         -0.45610076 -0.45846611]
 [-0.45610076  1.          0.90623283]
 [-0.45846611  0.90623283  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-09-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00848231 0.         0.        ]
 [0.         0.01367012 0.        ]
 [0.         0.         0.01387141]]
Resulting covariance matrix for run date DatetimeIndex(['2018-09-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00151094 -0.00111062 -0.00113282]
 [-0.00111062  0.00392432  0.00360871]
 [-0.00113282  0.00360871  0.00404073]]
Expected returns vector for run date DatetimeIndex(['2018-09-28'],
 ↪ dtype='datetime64[ns]'):
[0.00021718 0.00909097 0.01228589]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-09-28'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-10-30'], dtype='datetime64[ns]'):
[[ 1.         -0.14593764  0.06311385]
 [-0.14593764  1.          0.81594676]
 [ 0.06311385  0.81594676  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00770631 0.         0.        ]
 [0.         0.00858895 0.        ]
 [0.         0.         0.00889591]]
Resulting covariance matrix for run date DatetimeIndex(['2018-10-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.12835737e-03 -1.83530275e-04  8.22082086e-05]
 [-1.83530275e-04  1.40163244e-03  1.18452995e-03]
 [ 8.22082086e-05  1.18452995e-03  1.50360688e-03]]
Expected returns vector for run date DatetimeIndex(['2018-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.00014526 0.0072085  0.00874755]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-10-30'],
 ↪ dtype='datetime64[ns]')
[1.38777878e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-11-29'], dtype='datetime64[ns]'):
[[1.         0.27079585 0.2572864 ]
 [0.27079585 1.         0.90560615]
 [0.2572864  0.90560615 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00606316 0.         0.        ]
 [0.         0.00877259 0.        ]
 [0.         0.         0.00966541]]
Resulting covariance matrix for run date DatetimeIndex(['2018-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00069848 0.00027367 0.00028648]
 [0.00027367 0.00146221 0.00145895]
 [0.00028648 0.00145895 0.00177498]]
Expected returns vector for run date DatetimeIndex(['2018-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00029109 0.00562952 0.00878764]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-11-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 7.84095011e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2018-12-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2018-12-28'], dtype='datetime64[ns]'):
[[ 1.         -0.29881376 -0.21248287]
 [-0.29881376  1.          0.85728294]
 [-0.21248287  0.85728294  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2018-12-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00514465 0.         0.        ]
 [0.         0.00692407 0.        ]
 [0.         0.         0.00733608]]
Resulting covariance matrix for run date DatetimeIndex(['2018-12-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00055582 -0.00022353 -0.00016841]
 [-0.00022353  0.0010068   0.00091447]
 [-0.00016841  0.00091447  0.00113018]]
Expected returns vector for run date DatetimeIndex(['2018-12-28'],
 ↪ dtype='datetime64[ns]'):
[0.00236292 0.00774095 0.01085047]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2018-12-28'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 8.32667268e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.09112138 -0.0536349 ]
 [-0.09112138  1.          0.81694365]
 [-0.0536349   0.81694365  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00574857 0.         0.        ]
 [0.         0.00620503 0.        ]
 [0.         0.         0.00762815]]
Resulting covariance matrix for run date DatetimeIndex(['2019-01-30'],
 ↪  dtype='datetime64[ns]'):
[[ 6.27876094e-04 -6.17558741e-05 -4.46869197e-05]
 [-6.17558741e-05  7.31546596e-04  7.34698293e-04]
 [-4.46869197e-05  7.34698293e-04  1.10558432e-03]]
Expected returns vector for run date DatetimeIndex(['2019-01-30'],
 ↪  dtype='datetime64[ns]'):
[0.00379875 0.00788803 0.01215996]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-01-30'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2019-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-02-27'], dtype='datetime64[ns]'):
[[ 1.         -0.14034933 -0.09177331]
 [-0.14034933  1.          0.84257454]
 [-0.09177331  0.84257454  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-02-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00648974 0.         0.        ]
 [0.         0.00643078 0.        ]
 [0.         0.         0.00899202]]
Resulting covariance matrix for run date DatetimeIndex(['2019-02-27'],
 ↪  dtype='datetime64[ns]'):
[[ 6.31751828e-04 -8.78603779e-05 -8.03327574e-05]
 [-8.78603779e-05  6.20324520e-04  7.30837519e-04]
 [-8.03327574e-05  7.30837519e-04  1.21284712e-03]]
Expected returns vector for run date DatetimeIndex(['2019-02-27'],
 ↪  dtype='datetime64[ns]'):
[0.00363849 0.00865051 0.01423271]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-02-27'],
 ↪  dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2019-03-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2019-03-29'], dtype='datetime64[ns]'):
[[ 1.         -0.02101501  0.02978639]
 [-0.02101501  1.          0.90588027]
 [ 0.02978639  0.90588027  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-03-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00605583 0.         0.        ]
 [0.         0.00651104 0.        ]
 [0.         0.         0.00942315]]
Resulting covariance matrix for run date DatetimeIndex(['2019-03-29'],
 ↪  dtype='datetime64[ns]'):
[[ 6.96789331e-04 -1.57437228e-05  3.22954788e-05]
 [-1.57437228e-05  8.05478858e-04  1.05601707e-03]
 [ 3.22954788e-05  1.05601707e-03  1.68712061e-03]]
Expected returns vector for run date DatetimeIndex(['2019-03-29'],
 ↪  dtype='datetime64[ns]'):
[0.00071314 0.01168592 0.01775868]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-03-29'],
 ↪ dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-04-26'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-04-26'], dtype='datetime64[ns]'):
[[ 1.         -0.13658053 -0.10075997]
 [-0.13658053  1.          0.94224685]
 [-0.10075997  0.94224685  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-04-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00678117 0.         0.        ]
 [0.         0.01123464 0.        ]
 [0.         0.         0.01380462]]
Resulting covariance matrix for run date DatetimeIndex(['2019-04-26'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00096567 -0.00021851 -0.00019808]
 [-0.00021851  0.00265056  0.00306879]
 [-0.00019808  0.00306879  0.00400192]]
Expected returns vector for run date DatetimeIndex(['2019-04-26'],
 ↪ dtype='datetime64[ns]'):
[6.32907374e-05 1.04782051e-02 1.63745052e-02]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-04-26'],
 ↪ dtype='datetime64[ns]')
[1.11022302e-16 2.77555756e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-05-30'], dtype='datetime64[ns]'):
[[1.         0.15666957 0.10881647]
 [0.15666957 1.         0.8939023 ]
 [0.10881647 0.8939023  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00930593 0.         0.        ]
 [0.         0.00842194 0.        ]
 [0.         0.         0.0092979 ]]
Resulting covariance matrix for run date DatetimeIndex(['2019-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.001299   0.00018418 0.00014123]
 [0.00018418 0.00106394 0.00104997]
 [0.00014123 0.00104997 0.00129676]]
Expected returns vector for run date DatetimeIndex(['2019-05-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00109599  0.01037652  0.01466129]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-05-30'],
 ↪ dtype='datetime64[ns]')
[3.64291930e-17 1.38777878e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-06-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2019-06-28'], dtype='datetime64[ns]'):
[[ 1.          0.00771347 -0.1634087 ]
 [ 0.00771347  1.          0.82277691]
 [-0.1634087   0.82277691  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-06-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00703448 0.         0.        ]
 [0.         0.00740759 0.        ]
 [0.         0.         0.00965587]]
Resulting covariance matrix for run date DatetimeIndex(['2019-06-28'],
 ↪ dtype='datetime64[ns]'):
[[ 1.08864675e-03  8.84262452e-06 -2.44186281e-04]
 [ 8.84262452e-06  1.20719145e-03  1.29471082e-03]
 [-2.44186281e-04  1.29471082e-03  2.05118679e-03]]
Expected returns vector for run date DatetimeIndex(['2019-06-28'],
 ↪ dtype='datetime64[ns]'):
[0.00107915 0.00959534 0.01442115]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-06-28'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.38777878e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-07-30'], dtype='datetime64[ns]'):
[[ 1.          0.02300461 -0.08540464]
 [ 0.02300461  1.          0.92171383]
 [-0.08540464  0.92171383  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00872503 0.         0.        ]
 [0.         0.00968124 0.        ]
 [0.         0.         0.01410978]]
Resulting covariance matrix for run date DatetimeIndex(['2019-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.52252374e-03  3.88635847e-05 -2.10280437e-04]
 [ 3.88635847e-05  1.87452841e-03  2.51812610e-03]
 [-2.10280437e-04  2.51812610e-03  3.98171998e-03]]
Expected returns vector for run date DatetimeIndex(['2019-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.00073043 0.00885909 0.01368271]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-07-30'],
 ↪ dtype='datetime64[ns]')
[7.2858386e-17 0.0000000e+00 1.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-08-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2019-08-30'], dtype='datetime64[ns]'):
[[1.         0.11324013 0.2760419 ]
 [0.11324013 1.         0.95906699]
 [0.2760419  0.95906699 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0100346  0.         0.        ]
 [0.         0.01562031 0.        ]
 [0.         0.         0.02478353]]
Resulting covariance matrix for run date DatetimeIndex(['2019-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00191317 0.00033724 0.00130434]
```

```
 [0.00033724 0.00463589 0.00705432]
 [0.00130434 0.00705432 0.01167024]]
Expected returns vector for run date DatetimeIndex(['2019-08-30'],
 ↪ dtype='datetime64[ns]'):
[0.0005879  0.00615128 0.01017717]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-08-30'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 1.11022302e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-09-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-09-27'], dtype='datetime64[ns]'):
[[1.         0.12155892 0.29428843]
 [0.12155892 1.         0.85327357]
 [0.29428843 0.85327357 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00794842 0.         0.        ]
 [0.         0.00932256 0.        ]
 [0.         0.         0.0165913 ]]
Resulting covariance matrix for run date DatetimeIndex(['2019-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00113719 0.00016213 0.00069857]
 [0.00016213 0.00156438 0.00237562]
 [0.00069857 0.00237562 0.00495488]]
Expected returns vector for run date DatetimeIndex(['2019-09-27'],
 ↪ dtype='datetime64[ns]'):
[0.00308815 0.00544261 0.00755114]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-09-27'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 3.74700271e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-10-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2019-10-30'], dtype='datetime64[ns]'):
[[ 1.         -0.10245693 -0.05199076]
 [-0.10245693  1.          0.83684182]
 [-0.05199076  0.83684182  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00705921 0.         0.        ]
 [0.         0.00600427 0.        ]
 [0.         0.         0.00881309]]
Resulting covariance matrix for run date DatetimeIndex(['2019-10-30'],
 ↪ dtype='datetime64[ns]'):
[[ 6.97654422e-04 -6.07974987e-05 -4.52833618e-05]
 [-6.07974987e-05  5.04717565e-04  6.19954598e-04]
 [-4.52833618e-05  6.19954598e-04  1.08738816e-03]]
Expected returns vector for run date DatetimeIndex(['2019-10-30'],
 ↪ dtype='datetime64[ns]'):
[0.00213726 0.00711596 0.00991167]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-10-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 2.77555756e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-11-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2019-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.03719082 -0.05566068]
 [-0.03719082  1.          0.87555301]
 [-0.05566068  0.87555301  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0052753  0.         0.        ]
 [0.         0.0058816  0.        ]
 [0.         0.         0.00764913]]
Resulting covariance matrix for run date DatetimeIndex(['2019-11-29'],
 ↪ dtype='datetime64[ns]'):
[[ 5.84405573e-04 -2.42324901e-05 -4.71658307e-05]
 [-2.42324901e-05  7.26457495e-04  8.27197773e-04]
 [-4.71658307e-05  8.27197773e-04  1.22869364e-03]]
Expected returns vector for run date DatetimeIndex(['2019-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00345417 0.00784246 0.01021226]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-11-29'],
 ↪ dtype='datetime64[ns]')
[4.16333634e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2019-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2019-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.38280362 -0.39069146]
 [-0.38280362  1.          0.85823649]
 [-0.39069146  0.85823649  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2019-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00542742 0.         0.        ]
 [0.         0.00712493 0.        ]
 [0.         0.         0.00819826]]
Resulting covariance matrix for run date DatetimeIndex(['2019-12-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00050077 -0.00025165 -0.00029553]
 [-0.00025165  0.000863    0.00085223]
 [-0.00029553  0.00085223  0.00114259]]
Expected returns vector for run date DatetimeIndex(['2019-12-30'],
 ↪ dtype='datetime64[ns]'):
[0.00468352 0.00844468 0.01121435]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2019-12-30'],
 ↪ dtype='datetime64[ns]')
[0.27061472 0.         0.72938528]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-01-30'], dtype='datetime64[ns]'):
[[1.         0.22052191 0.17532554]
 [0.22052191 1.         0.88833445]
 [0.17532554 0.88833445 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01256192 0.         0.        ]
```

```
 [0.         0.01054761 0.        ]
 [0.         0.         0.01169057]]
Resulting covariance matrix for run date DatetimeIndex(['2020-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00236703 0.00043828 0.00038621]
 [0.00043828 0.00166878 0.00164307]
 [0.00038621 0.00164307 0.00205004]]
Expected returns vector for run date DatetimeIndex(['2020-01-30'],
 ↪ dtype='datetime64[ns]'):
[0.00618468 0.00856258 0.01172423]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-01-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.11022302e-16 1.00000000e+00]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-02-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2020-02-28'], dtype='datetime64[ns]'):
[[ 1.         -0.12270421 -0.19614449]
 [-0.12270421  1.          0.96252549]
 [-0.19614449  0.96252549  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-02-28'],
 ↪ dtype='datetime64[ns]'):
[[0.02594335 0.         0.        ]
 [0.         0.0500166  0.        ]
 [0.         0.         0.05861667]]
Resulting covariance matrix for run date DatetimeIndex(['2020-02-28'],
 ↪ dtype='datetime64[ns]'):
[[ 0.01413421 -0.00334364 -0.00626387]
 [-0.00334364  0.05253486  0.0592607 ]
 [-0.00626387  0.0592607   0.07215419]]
Expected returns vector for run date DatetimeIndex(['2020-02-28'],
 ↪ dtype='datetime64[ns]'):
[0.00514889 0.00776805 0.01010661]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-02-28'],
 ↪ dtype='datetime64[ns]')
[0.22040607 0.         0.77959393]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-03-30'], dtype='datetime64[ns]'):
[[ 1.         -0.04199451 -0.06731369]
 [-0.04199451  1.          0.91407657]
 [-0.06731369  0.91407657  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-03-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01705093 0.         0.        ]
 [0.         0.03310461 0.        ]
 [0.         0.         0.04444361]]
Resulting covariance matrix for run date DatetimeIndex(['2020-03-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00523322 -0.00042668 -0.00091819]
 [-0.00042668  0.01972647  0.02420766]
 [-0.00091819  0.02420766  0.03555422]]
Expected returns vector for run date DatetimeIndex(['2020-03-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00315019 0.00639149 0.00837659]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-03-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.07541179 -0.03440368]
 [-0.07541179  1.          0.92805523]
 [-0.03440368  0.92805523  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01139149 0.         0.        ]
 [0.         0.0221948  0.        ]
 [0.         0.         0.03427706]]
Resulting covariance matrix for run date DatetimeIndex(['2020-04-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00246555 -0.00036226 -0.00025524]
 [-0.00036226  0.00935957  0.01341474]
 [-0.00025524  0.01341474  0.02232342]]
Expected returns vector for run date DatetimeIndex(['2020-04-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.00390458 -0.00186293 -0.0044061 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-04-29'],
 ↪ dtype='datetime64[ns]')
[1.23165367e-16 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-05-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2020-05-29'], dtype='datetime64[ns]'):
[[ 1.         -0.1821374  -0.29955667]
 [-0.1821374   1.          0.93893915]
 [-0.29955667  0.93893915  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-05-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01020498 0.         0.        ]
 [0.         0.01312239 0.        ]
 [0.         0.         0.02380693]]
Resulting covariance matrix for run date DatetimeIndex(['2020-05-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00218698 -0.0005122  -0.00152832]
 [-0.0005122   0.00361614  0.00615989]
 [-0.00152832  0.00615989  0.01190217]]
Expected returns vector for run date DatetimeIndex(['2020-05-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.00355732  0.00155018 -0.00229712]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-05-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 5.55111512e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-06-29'], dtype='datetime64[ns]'):
[[ 1.         -0.01634067 -0.08585903]
```

```
 [-0.01634067  1.          0.81170231]
 [-0.08585903  0.81170231  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.01091523 0.         0.        ]
 [0.         0.0096762  0.        ]
 [0.         0.         0.01773137]]
Resulting covariance matrix for run date DatetimeIndex(['2020-06-29'],
 ↪  dtype='datetime64[ns]'):
[[ 2.14455897e-03 -3.10655989e-05 -2.99111759e-04]
 [-3.10655989e-05  1.68531773e-03  2.50677980e-03]
 [-2.99111759e-04  2.50677980e-03  5.65922575e-03]]
Expected returns vector for run date DatetimeIndex(['2020-06-29'],
 ↪  dtype='datetime64[ns]'):
[ 0.00383243 -0.00015181 -0.0068528 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-06-29'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 6.81174142e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2020-07-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2020-07-27'], dtype='datetime64[ns]'):
[[ 1.          0.06027466 -0.07832952]
 [ 0.06027466  1.          0.89541703]
 [-0.07832952  0.89541703  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-07-27'],
 ↪  dtype='datetime64[ns]'):
[[0.0153338  0.         0.        ]
 [0.         0.00820466 0.        ]
 [0.         0.         0.01518291]]
Resulting covariance matrix for run date DatetimeIndex(['2020-07-27'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00376201  0.00012133 -0.00029178]
 [ 0.00012133  0.00107706  0.00178468]
 [-0.00029178  0.00178468  0.00368833]]
Expected returns vector for run date DatetimeIndex(['2020-07-27'],
 ↪  dtype='datetime64[ns]'):
[ 0.00482389  0.002088   -0.00396523]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-07-27'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 1.11022302e-15 1.73409515e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2020-08-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-08-27'], dtype='datetime64[ns]'):
[[1.         0.1645126  0.03908699]
 [0.1645126  1.         0.9038018 ]
 [0.03908699 0.9038018  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-08-27'],
 ↪  dtype='datetime64[ns]'):
[[0.01331659 0.         0.        ]
 [0.         0.03274099 0.        ]
 [0.         0.         0.01865998]]
Resulting covariance matrix for run date DatetimeIndex(['2020-08-27'],
 ↪  dtype='datetime64[ns]'):
```

```
[[0.00372396 0.00150627 0.00020397]
 [0.00150627 0.02251142 0.01159566]
 [0.00020397 0.01159566 0.00731209]]
Expected returns vector for run date DatetimeIndex(['2020-08-27'],
 ↪ dtype='datetime64[ns]'):
[ 0.00679191  0.00307901 -0.00446633]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-08-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 2.22044605e-16 9.21845370e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-09-29'], dtype='datetime64[ns]'):
[[1.         0.14799882 0.23401123]
 [0.14799882 1.         0.81959784]
 [0.23401123 0.81959784 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01306242 0.         0.        ]
 [0.         0.01023895 0.        ]
 [0.         0.         0.01836003]]
Resulting covariance matrix for run date DatetimeIndex(['2020-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00204752 0.00023753 0.00067346]
 [0.00023753 0.00125803 0.00184889]
 [0.00067346 0.00184889 0.00404509]]
Expected returns vector for run date DatetimeIndex(['2020-09-29'],
 ↪ dtype='datetime64[ns]'):
[ 0.00398272  0.00387397 -0.00204238]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-09-29'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 3.3029135e-15 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-10-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2020-10-30'], dtype='datetime64[ns]'):
[[1.         0.01767085 0.02587229]
 [0.01767085 1.         0.77666503]
 [0.02587229 0.77666503 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01597819 0.         0.        ]
 [0.         0.00881661 0.        ]
 [0.         0.         0.01923991]]
Resulting covariance matrix for run date DatetimeIndex(['2020-10-30'],
 ↪ dtype='datetime64[ns]'):
[[3.82953705e-03 3.73402941e-05 1.19304427e-04]
 [3.73402941e-05 1.16598896e-03 1.97619388e-03]
 [1.19304427e-04 1.97619388e-03 5.55260987e-03]]
Expected returns vector for run date DatetimeIndex(['2020-10-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.00357039  0.00367114 -0.00471441]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-10-30'],
 ↪ dtype='datetime64[ns]')
[1.0000000e+00 0.0000000e+00 6.3371403e-16]
```

```
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-11-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2020-11-27'], dtype='datetime64[ns]'):
[[ 1.         -0.00201246 -0.02100285]
 [-0.00201246  1.          0.8507356 ]
 [-0.02100285  0.8507356   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-11-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01349405 0.         0.        ]
 [0.         0.0101793  0.        ]
 [0.         0.         0.01245338]]
Resulting covariance matrix for run date DatetimeIndex(['2020-11-27'],
 ↪ dtype='datetime64[ns]'):
[[ 2.36716271e-03 -3.59361597e-06 -4.58829320e-05]
 [-3.59361597e-06  1.34703688e-03  1.40198458e-03]
 [-4.58829320e-05  1.40198458e-03  2.01612647e-03]]
Expected returns vector for run date DatetimeIndex(['2020-11-27'],
 ↪ dtype='datetime64[ns]'):
[ 0.00293532  0.0028791  -0.00448311]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-11-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 5.27355937e-15 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2020-12-23'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2020-12-23'], dtype='datetime64[ns]'):
[[1.         0.14282629 0.12219014]
 [0.14282629 1.         0.88544452]
 [0.12219014 0.88544452 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2020-12-23'],
 ↪ dtype='datetime64[ns]'):
[[0.01544878 0.         0.        ]
 [0.         0.01370892 0.        ]
 [0.         0.         0.0201496 ]]
Resulting covariance matrix for run date DatetimeIndex(['2020-12-23'],
 ↪ dtype='datetime64[ns]'):
[[0.00381863 0.00048398 0.00060858]
 [0.00048398 0.00300695 0.00391337]
 [0.00060858 0.00391337 0.0064961 ]]
Expected returns vector for run date DatetimeIndex(['2020-12-23'],
 ↪ dtype='datetime64[ns]'):
[ 0.00133616  0.00537476 -0.00065513]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2020-12-23'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.94289029e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-01-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2021-01-29'], dtype='datetime64[ns]'):
[[1.         0.12976937 0.16876517]
 [0.12976937 1.         0.95161003]
 [0.16876517 0.95161003 1.        ]]
```

```
Volatility diagonal vector for run date DatetimeIndex(['2021-01-29'],
 ↪  dtype='datetime64[ns]'):
[[0.011625   0.         0.        ]
 [0.         0.01752096 0.        ]
 [0.         0.         0.02603402]]
Resulting covariance matrix for run date DatetimeIndex(['2021-01-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00216225 0.00042291 0.00081722]
 [0.00042291 0.00491175 0.0069451 ]
 [0.00081722 0.0069451  0.01084432]]
Expected returns vector for run date DatetimeIndex(['2021-01-29'],
 ↪  dtype='datetime64[ns]'):
[ 0.00195275  0.00537765 -0.00090656]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-01-29'],
 ↪  dtype='datetime64[ns]')
[2.32452946e-16 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2021-02-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2021-02-26'], dtype='datetime64[ns]'):
[[ 1.         -0.17069428 -0.15936644]
 [-0.17069428  1.          0.90818524]
 [-0.15936644  0.90818524  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-02-26'],
 ↪  dtype='datetime64[ns]'):
[[0.00964161 0.         0.        ]
 [0.         0.01303231 0.        ]
 [0.         0.         0.01614642]]
Resulting covariance matrix for run date DatetimeIndex(['2021-02-26'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00195217 -0.00045041 -0.00052101]
 [-0.00045041  0.00356666  0.00401321]
 [-0.00052101  0.00401321  0.00547485]]
Expected returns vector for run date DatetimeIndex(['2021-02-26'],
 ↪  dtype='datetime64[ns]'):
[ 0.00045142  0.00372576 -0.00278068]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-02-26'],
 ↪  dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2021-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-03-30'], dtype='datetime64[ns]'):
[[ 1.         -0.00498437 -0.03762835]
 [-0.00498437  1.          0.9177807 ]
 [-0.03762835  0.9177807   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-03-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00801266 0.         0.        ]
 [0.         0.01198595 0.        ]
 [0.         0.         0.0194696 ]]
Resulting covariance matrix for run date DatetimeIndex(['2021-03-30'],
 ↪  dtype='datetime64[ns]'):
[[ 1.21985050e-03 -9.09521293e-06 -1.11532733e-04]
 [-9.09521293e-06  2.72959788e-03  4.06932230e-03]
 [-1.11532733e-04  4.06932230e-03  7.20224129e-03]]
Expected returns vector for run date DatetimeIndex(['2021-03-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00127978  0.00606171  0.00210992]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-03-30'],
 ↪ dtype='datetime64[ns]')
[6.9388939e-18 1.0000000e+00 0.0000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-04-29'], dtype='datetime64[ns]'):
[[1.         0.27887845 0.12643389]
 [0.27887845 1.         0.88089049]
 [0.12643389 0.88089049 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00876091 0.         0.        ]
 [0.         0.00946218 0.        ]
 [0.         0.         0.01610168]]
Resulting covariance matrix for run date DatetimeIndex(['2021-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00138157 0.00041613 0.00032104]
 [0.00041613 0.00161159 0.00241578]
 [0.00032104 0.00241578 0.00466675]]
Expected returns vector for run date DatetimeIndex(['2021-04-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00099705  0.00739118  0.00183629]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-04-29'],
 ↪ dtype='datetime64[ns]')
[1.38777878e-17 1.00000000e+00 1.94289029e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-05-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2021-05-28'], dtype='datetime64[ns]'):
[[1.         0.19191848 0.28781167]
 [0.19191848 1.         0.79003584]
 [0.28781167 0.79003584 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-05-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01107335 0.         0.        ]
 [0.         0.00488169 0.        ]
 [0.         0.         0.00822686]]
Resulting covariance matrix for run date DatetimeIndex(['2021-05-28'],
 ↪ dtype='datetime64[ns]'):
[[0.002575   0.00021786 0.00055061]
 [0.00021786 0.00050045 0.0006663 ]
 [0.00055061 0.0006663  0.00142131]]
Expected returns vector for run date DatetimeIndex(['2021-05-28'],
 ↪ dtype='datetime64[ns]'):
[ 1.16453005e-04  5.89275895e-03 -9.40228545e-05]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-05-28'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-06-29'], dtype='datetime64[ns]'):
[[1.         0.19647183 0.10749105]
 [0.19647183 1.         0.84139872]
 [0.10749105 0.84139872 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00600595 0.         0.        ]
 [0.         0.00645523 0.        ]
 [0.         0.         0.0125584 ]]
Resulting covariance matrix for run date DatetimeIndex(['2021-06-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00072143 0.00015234 0.00016215]
 [0.00015234 0.0008334  0.0013642 ]
 [0.00016215 0.0013642  0.00315427]]
Expected returns vector for run date DatetimeIndex(['2021-06-29'],
 ↪ dtype='datetime64[ns]'):
[0.00225828 0.00792491 0.00155313]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-06-29'],
 ↪ dtype='datetime64[ns]')
[2.77555756e-17 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-07-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2021-07-30'], dtype='datetime64[ns]'):
[[ 1.          0.18669695 -0.02037495]
 [ 0.18669695  1.          0.75810826]
 [-0.02037495  0.75810826  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00938651 0.         0.        ]
 [0.         0.00600081 0.        ]
 [0.         0.         0.00931456]]
Resulting covariance matrix for run date DatetimeIndex(['2021-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.58591940e-03  1.89288483e-04 -3.20653404e-05]
 [ 1.89288483e-04  6.48175842e-04  7.62739845e-04]
 [-3.20653404e-05  7.62739845e-04  1.56169986e-03]]
Expected returns vector for run date DatetimeIndex(['2021-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.00186201 0.00802483 0.00135842]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-07-30'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-08-30'], dtype='datetime64[ns]'):
[[ 1.         -0.17581454 -0.44095474]
 [-0.17581454  1.          0.74430722]
 [-0.44095474  0.74430722  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00949433 0.         0.        ]
```

```
 [0.         0.00639175 0.        ]
 [0.         0.         0.01007393]]
Resulting covariance matrix for run date DatetimeIndex(['2021-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.0017127  -0.00020272 -0.00080133]
 [-0.00020272  0.00077624  0.00091059]
 [-0.00080133  0.00091059  0.0019282 ]]
Expected returns vector for run date DatetimeIndex(['2021-08-30'],
 ↪ dtype='datetime64[ns]'):
[ 0.00325263  0.00646486 -0.00030568]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-08-30'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-09-29'], dtype='datetime64[ns]'):
[[ 1.         -0.07370305 -0.15177181]
 [-0.07370305  1.          0.73785001]
 [-0.15177181  0.73785001  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00752255 0.         0.        ]
 [0.         0.00795932 0.        ]
 [0.         0.         0.01053847]]
Resulting covariance matrix for run date DatetimeIndex(['2021-09-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.07518533e-03 -8.38454795e-05 -2.28605711e-04]
 [-8.38454795e-05  1.20366338e-03  1.17591221e-03]
 [-2.28605711e-04  1.17591221e-03  2.11012690e-03]]
Expected returns vector for run date DatetimeIndex(['2021-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.00300437 0.00757563 0.0008921 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-09-29'],
 ↪ dtype='datetime64[ns]')
[2.08166817e-17 1.00000000e+00 1.85526864e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-10-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2021-10-29'], dtype='datetime64[ns]'):
[[1.         0.26467398 0.28541133]
 [0.26467398 1.         0.8734898 ]
 [0.28541133 0.8734898  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-10-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00994275 0.         0.        ]
 [0.         0.01122997 0.        ]
 [0.         0.         0.01313389]]
Resulting covariance matrix for run date DatetimeIndex(['2021-10-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00187831 0.0005615  0.00070815]
 [0.0005615  0.00239613 0.00244784]
 [0.00070815 0.00244784 0.00327748]]
Expected returns vector for run date DatetimeIndex(['2021-10-29'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.00209161 0.00973116 0.00466112]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-10-29'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.38777878e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.01139164  0.05270559]
 [-0.01139164  1.          0.90510821]
 [ 0.05270559  0.90510821  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00857646 0.         0.        ]
 [0.         0.01162099 0.        ]
 [0.         0.         0.01548013]]
Resulting covariance matrix for run date DatetimeIndex(['2021-11-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.61822332e-03 -2.49781348e-05  1.53943635e-04]
 [-2.49781348e-05  2.97104489e-03  3.58212699e-03]
 [ 1.53943635e-04  3.58212699e-03  5.27195543e-03]]
Expected returns vector for run date DatetimeIndex(['2021-11-29'],
 ↪ dtype='datetime64[ns]'):
[0.00296264 0.01102405 0.00560967]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-11-29'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2021-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2021-12-30'], dtype='datetime64[ns]'):
[[1.         0.17781252 0.05571446]
 [0.17781252 1.         0.8384198 ]
 [0.05571446 0.8384198  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2021-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00862319 0.         0.        ]
 [0.         0.01008153 0.        ]
 [0.         0.         0.01190802]]
Resulting covariance matrix for run date DatetimeIndex(['2021-12-30'],
 ↪ dtype='datetime64[ns]'):
[[1.26410942e-03 2.62788063e-04 9.72578033e-05]
 [2.62788063e-04 1.72783371e-03 1.71110506e-03]
 [9.72578033e-05 1.71110506e-03 2.41061783e-03]]
Expected returns vector for run date DatetimeIndex(['2021-12-30'],
 ↪ dtype='datetime64[ns]'):
[0.00223803 0.00825398 0.00101942]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2021-12-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.55507732e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2022-01-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-01-28'], dtype='datetime64[ns]'):
[[ 1.         -0.27893779 -0.3023403 ]
 [-0.27893779  1.          0.94891775]
 [-0.3023403   0.94891775  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-01-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00777066 0.         0.        ]
 [0.         0.01579732 0.        ]
 [0.         0.         0.0202173 ]]
Resulting covariance matrix for run date DatetimeIndex(['2022-01-28'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00096613 -0.00054786 -0.00075997]
 [-0.00054786  0.00399288  0.00484903]
 [-0.00075997  0.00484903  0.00653983]]
Expected returns vector for run date DatetimeIndex(['2022-01-28'],
 ↪  dtype='datetime64[ns]'):
[0.00177751 0.00878034 0.00063013]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-01-28'],
 ↪  dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-02-25'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-02-25'], dtype='datetime64[ns]'):
[[ 1.         -0.08302591 -0.12069208]
 [-0.08302591  1.          0.91485316]
 [-0.12069208  0.91485316  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-02-25'],
 ↪  dtype='datetime64[ns]'):
[[0.01318068 0.         0.        ]
 [0.         0.01371337 0.        ]
 [0.         0.         0.01839445]]
Resulting covariance matrix for run date DatetimeIndex(['2022-02-25'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00364834 -0.00031515 -0.0006145 ]
 [-0.00031515  0.00394919  0.0048462 ]
 [-0.0006145   0.0048462   0.00710547]]
Expected returns vector for run date DatetimeIndex(['2022-02-25'],
 ↪  dtype='datetime64[ns]'):
[0.00069798 0.00871067 0.00278482]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-02-25'],
 ↪  dtype='datetime64[ns]')
[8.32667268e-17 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-03-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-03-30'], dtype='datetime64[ns]'):
[[1.         0.1139351  0.05829563]
 [0.1139351  1.         0.85381502]
 [0.05829563 0.85381502 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-03-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00906912 0.         0.        ]
 [0.         0.01129294 0.        ]
 [0.         0.         0.01482823]]
Resulting covariance matrix for run date DatetimeIndex(['2022-03-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00156273 0.00022171 0.00014895]
 [0.00022171 0.00242308 0.00271653]
 [0.00014895 0.00271653 0.00417765]]
Expected returns vector for run date DatetimeIndex(['2022-03-30'],
 ↪  dtype='datetime64[ns]'):
[0.00148765 0.00907505 0.0036185 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-03-30'],
 ↪  dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-04-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2022-04-29'], dtype='datetime64[ns]'):
[[1.         0.06167909 0.14874539]
 [0.06167909 1.         0.91182733]
 [0.14874539 0.91182733 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.0092061  0.         0.        ]
 [0.         0.0137756  0.        ]
 [0.         0.         0.01627118]]
Resulting covariance matrix for run date DatetimeIndex(['2022-04-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00169505 0.00015644 0.00044562]
 [0.00015644 0.00379534 0.00408764]
 [0.00044562 0.00408764 0.00529503]]
Expected returns vector for run date DatetimeIndex(['2022-04-29'],
 ↪  dtype='datetime64[ns]'):
[0.00222814 0.00853366 0.0010709 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-04-29'],
 ↪  dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 1.83880688e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-05-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-05-27'], dtype='datetime64[ns]'):
[[1.         0.16417898 0.10593555]
 [0.16417898 1.         0.92090577]
 [0.10593555 0.92090577 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-05-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00893534 0.         0.        ]
 [0.         0.01142528 0.        ]
 [0.         0.         0.01434797]]
Resulting covariance matrix for run date DatetimeIndex(['2022-05-27'],
 ↪  dtype='datetime64[ns]'):
[[0.00167664 0.00035198 0.00028521]
 [0.00035198 0.00274128 0.00317024]
 [0.00028521 0.00317024 0.00432315]]
Expected returns vector for run date DatetimeIndex(['2022-05-27'],
 ↪  dtype='datetime64[ns]'):
[0.00187035 0.00765617 0.00146309]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-05-27'],
 ↪  dtype='datetime64[ns]')
[2.08166817e-16 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-06-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-06-29'], dtype='datetime64[ns]'):
[[1.         0.23922575 0.21857385]
 [0.23922575 1.         0.86930639]
 [0.21857385 0.86930639 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00875211 0.         0.        ]
 [0.         0.00883739 0.        ]
 [0.         0.         0.01054324]]
Resulting covariance matrix for run date DatetimeIndex(['2022-06-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00153199 0.00037006 0.00040338]
 [0.00037006 0.00156199 0.00161995]
 [0.00040338 0.00161995 0.0022232 ]]
Expected returns vector for run date DatetimeIndex(['2022-06-29'],
 ↪  dtype='datetime64[ns]'):
[ 0.0002559   0.00606052 -0.00024864]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-06-29'],
 ↪  dtype='datetime64[ns]')
[1.66533454e-16 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-07-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2022-07-29'], dtype='datetime64[ns]'):
[[ 1.          0.07738204 -0.00852731]
 [ 0.07738204  1.          0.93839055]
 [-0.00852731  0.93839055  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-07-29'],
 ↪  dtype='datetime64[ns]'):
[[0.0080774  0.         0.        ]
 [0.         0.00896627 0.        ]
 [0.         0.         0.01181525]]
Resulting covariance matrix for run date DatetimeIndex(['2022-07-29'],
 ↪  dtype='datetime64[ns]'):
[[ 9.13420584e-04  7.84605538e-05 -1.13934245e-05]
 [ 7.84605538e-05  1.12551571e-03  1.39176626e-03]
 [-1.13934245e-05  1.39176626e-03  1.95440152e-03]]
Expected returns vector for run date DatetimeIndex(['2022-07-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00203513  0.00465821 -0.00200018]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-07-29'],
 ↪  dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2022-08-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2022-08-30'], dtype='datetime64[ns]'):
[[1.         0.15550983 0.07254574]
 [0.15550983 1.         0.86855966]
 [0.07254574 0.86855966 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-08-30'],
 ↪  dtype='datetime64[ns]'):
```

```
[[0.00917888 0.         0.        ]
 [0.         0.00973638 0.        ]
 [0.         0.         0.01315846]]
Resulting covariance matrix for run date DatetimeIndex(['2022-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00176929 0.00029185 0.000184  ]
 [0.00029185 0.00199074 0.0023368 ]
 [0.000184   0.0023368  0.00363604]]
Expected returns vector for run date DatetimeIndex(['2022-08-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00299814  0.00853456  0.00300315]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-08-30'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2022-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-09-29'], dtype='datetime64[ns]'):
[[1.         0.04222246 0.07004637]
 [0.04222246 1.         0.91071051]
 [0.07004637 0.91071051 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01028963 0.         0.        ]
 [0.         0.01003105 0.        ]
 [0.         0.         0.01309316]]
Resulting covariance matrix for run date DatetimeIndex(['2022-09-29'],
 ↪ dtype='datetime64[ns]'):
[[1.79989892e-03 7.40864047e-05 1.60427397e-04]
 [7.40864047e-05 1.71057405e-03 2.03338735e-03]
 [1.60427397e-04 2.03338735e-03 2.91432281e-03]]
Expected returns vector for run date DatetimeIndex(['2022-09-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00466289  0.0090532   0.0048959 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-09-29'],
 ↪ dtype='datetime64[ns]')
[1.04083409e-16 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2022-10-27'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2022-10-27'], dtype='datetime64[ns]'):
[[ 1.          0.19426072 -0.07348736]
 [ 0.19426072  1.          0.7752279 ]
 [-0.07348736  0.7752279   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-10-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01060425 0.         0.        ]
 [0.         0.00584214 0.        ]
 [0.         0.         0.00521344]]
Resulting covariance matrix for run date DatetimeIndex(['2022-10-27'],
 ↪ dtype='datetime64[ns]'):
[[ 2.13655047e-03  2.28659973e-04 -7.71916385e-05]
 [ 2.28659973e-04  6.48480867e-04  4.48620421e-04]
 [-7.71916385e-05  4.48620421e-04  5.16418885e-04]]
```

```
Expected returns vector for run date DatetimeIndex(['2022-10-27'],
 ↪ dtype='datetime64[ns]'):
[-0.00471942  0.00667943  0.00233614]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-10-27'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 5.55111512e-17]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2022-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2022-11-29'], dtype='datetime64[ns]'):
[[1.         0.23629516 0.24102516]
 [0.23629516 1.         0.8698647 ]
 [0.24102516 0.8698647  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.0107299  0.         0.        ]
 [0.         0.00690115 0.        ]
 [0.         0.         0.00761523]]
Resulting covariance matrix for run date DatetimeIndex(['2022-11-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00230261 0.00034995 0.00039389]
 [0.00034995 0.00095252 0.0009143 ]
 [0.00039389 0.0009143  0.00115984]]
Expected returns vector for run date DatetimeIndex(['2022-11-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00523522  0.00692726  0.00347274]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-11-29'],
 ↪ dtype='datetime64[ns]')
[2.22044605e-16 1.00000000e+00 7.77156117e-16]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2022-12-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2022-12-30'], dtype='datetime64[ns]'):
[[ 1.         -0.09594343  0.02768251]
 [-0.09594343  1.          0.86611092]
 [ 0.02768251  0.86611092  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2022-12-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00699325 0.         0.        ]
 [0.         0.00820199 0.        ]
 [0.         0.         0.01118639]]
Resulting covariance matrix for run date DatetimeIndex(['2022-12-30'],
 ↪ dtype='datetime64[ns]'):
[[ 9.29205239e-04 -1.04560310e-04  4.11460626e-05]
 [-1.04560310e-04  1.27817879e-03  1.50985855e-03]
 [ 4.11460626e-05  1.50985855e-03  2.37757316e-03]]
Expected returns vector for run date DatetimeIndex(['2022-12-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00409092  0.00724982  0.00375548]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2022-12-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 4.16333634e-16]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2023-01-30'], dtype='datetime64[ns]')
```

```
Correlation matrix for run date DatetimeIndex(['2023-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.22399907 -0.18121527]
 [-0.22399907  1.          0.82582682]
 [-0.18121527  0.82582682  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00762293 0.         0.        ]
 [0.         0.00801675 0.        ]
 [0.         0.         0.01003733]]
Resulting covariance matrix for run date DatetimeIndex(['2023-01-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00110407 -0.00026009 -0.00026344]
 [-0.00026009  0.0012211   0.00126258]
 [-0.00026344  0.00126258  0.00191421]]
Expected returns vector for run date DatetimeIndex(['2023-01-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00350136  0.00567813  0.00262971]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-01-30'],
 ↪ dtype='datetime64[ns]')
[0. 1. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2023-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-02-27'], dtype='datetime64[ns]'):
[[ 1.         -0.24835405 -0.35731113]
 [-0.24835405  1.          0.91683447]
 [-0.35731113  0.91683447  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01092927 0.         0.        ]
 [0.         0.01481122 0.        ]
 [0.         0.         0.01344563]]
Resulting covariance matrix for run date DatetimeIndex(['2023-02-27'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00250843 -0.00084425 -0.00110265]
 [-0.00084425  0.00460682  0.00383426]
 [-0.00110265  0.00383426  0.00379648]]
Expected returns vector for run date DatetimeIndex(['2023-02-27'],
 ↪ dtype='datetime64[ns]'):
[-0.00224258  0.00457563  0.00081174]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-02-27'],
 ↪ dtype='datetime64[ns]')
[2.08166817e-17 1.00000000e+00 1.80411242e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2023-03-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-03-29'], dtype='datetime64[ns]'):
[[1.         0.11417131 0.23669359]
 [0.11417131 1.         0.8634727 ]
 [0.23669359 0.8634727  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-03-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00959849 0.         0.        ]
 [0.         0.00569797 0.        ]
 [0.         0.         0.00928176]]
```

```
Resulting covariance matrix for run date DatetimeIndex(['2023-03-29'],
 ↪  dtype='datetime64[ns]'):
[[1.19770356e-03 8.11752343e-05 2.74134287e-04]
 [8.11752343e-05 4.22069318e-04 5.93666703e-04]
 [2.74134287e-04 5.93666703e-04 1.11996481e-03]]
Expected returns vector for run date DatetimeIndex(['2023-03-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00216875  0.00474329  0.00123653]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-03-29'],
 ↪  dtype='datetime64[ns]')
[1.29746474e-15 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-04-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2023-04-28'], dtype='datetime64[ns]'):
[[1.         0.55648643 0.49339393]
 [0.55648643 1.         0.83403796]
 [0.49339393 0.83403796 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-04-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00850109 0.         0.        ]
 [0.         0.00594512 0.        ]
 [0.         0.         0.00791014]]
Resulting covariance matrix for run date DatetimeIndex(['2023-04-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00151764 0.00059062 0.00069674]
 [0.00059062 0.00074223 0.00082366]
 [0.00069674 0.00082366 0.00131398]]
Expected returns vector for run date DatetimeIndex(['2023-04-28'],
 ↪  dtype='datetime64[ns]'):
[-0.0003998   0.01218188  0.01314669]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-04-28'],
 ↪  dtype='datetime64[ns]')
[3.03142927e-16 9.44317840e-01 5.56821601e-02]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-05-30'], dtype='datetime64[ns]'):
[[ 1.         -0.25938019 -0.31654909]
 [-0.25938019  1.          0.84324962]
 [-0.31654909  0.84324962  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-05-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00770914 0.         0.        ]
 [0.         0.00536147 0.        ]
 [0.         0.         0.00697863]]
Resulting covariance matrix for run date DatetimeIndex(['2023-05-30'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00118862 -0.00021442 -0.0003406 ]
 [-0.00021442  0.00057491  0.00063102]
 [-0.0003406   0.00063102  0.00097403]]
Expected returns vector for run date DatetimeIndex(['2023-05-30'],
 ↪  dtype='datetime64[ns]'):
[-0.00064605  0.00958075  0.01102125]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-05-30'],
 ↪  dtype='datetime64[ns]')
[5.55111512e-17 4.70394593e-01 5.29605407e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-06-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-06-28'], dtype='datetime64[ns]'):
[[ 1.          0.00740786 -0.16306534]
 [ 0.00740786  1.          0.8072747 ]
 [-0.16306534  0.8072747   1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-06-28'],
 ↪  dtype='datetime64[ns]'):
[[0.0079884  0.         0.        ]
 [0.         0.0061606  0.        ]
 [0.         0.         0.00745258]]
Resulting covariance matrix for run date DatetimeIndex(['2023-06-28'],
 ↪  dtype='datetime64[ns]'):
[[ 1.14866157e-03  6.56217446e-06 -1.74743403e-04]
 [ 6.56217446e-06  6.83153784e-04  6.67150247e-04]
 [-1.74743403e-04  6.67150247e-04  9.99737865e-04]]
Expected returns vector for run date DatetimeIndex(['2023-06-28'],
 ↪  dtype='datetime64[ns]'):
[-0.001607    0.01094372  0.01445214]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-06-28'],
 ↪  dtype='datetime64[ns]')
[1.38777878e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-07-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-07-28'], dtype='datetime64[ns]'):
[[1.         0.30394775 0.31994313]
 [0.30394775 1.         0.80250502]
 [0.31994313 0.80250502 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-07-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00593487 0.         0.        ]
 [0.         0.00530844 0.        ]
 [0.         0.         0.00661902]]
Resulting covariance matrix for run date DatetimeIndex(['2023-07-28'],
 ↪  dtype='datetime64[ns]'):
[[0.0007749  0.00021067 0.0002765 ]
 [0.00021067 0.00061995 0.00062034]
 [0.0002765  0.00062034 0.00096385]]
Expected returns vector for run date DatetimeIndex(['2023-07-28'],
 ↪  dtype='datetime64[ns]'):
[-0.00269316  0.01006365  0.0123992 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-07-28'],
 ↪  dtype='datetime64[ns]')
[4.42354486e-17 1.13109319e-01 8.86890681e-01]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-08-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-08-30'], dtype='datetime64[ns]'):
[[ 1.         -0.00363192  0.00226763]
 [-0.00363192  1.          0.85358614]
 [ 0.00226763  0.85358614  1.        ]]
```

```
Volatility diagonal vector for run date DatetimeIndex(['2023-08-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00614987 0.         0.        ]
 [0.         0.0054543  0.        ]
 [0.         0.         0.00694181]]
Resulting covariance matrix for run date DatetimeIndex(['2023-08-30'],
 ↪ dtype='datetime64[ns]'):
[[ 5.67314094e-04 -1.82739675e-06  1.45211768e-06]
 [-1.82739675e-06  4.46241204e-04  4.84786575e-04]
 [ 1.45211768e-06  4.84786575e-04  7.22831040e-04]]
Expected returns vector for run date DatetimeIndex(['2023-08-30'],
 ↪ dtype='datetime64[ns]'):
[-0.00412102  0.00861433  0.012235  ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-08-30'],
 ↪ dtype='datetime64[ns]')
[0.00000000e+00 1.11022302e-16 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2023-09-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2023-09-29'], dtype='datetime64[ns]'):
[[ 1.          0.02632957 -0.13965774]
 [ 0.02632957  1.          0.94195038]
 [-0.13965774  0.94195038  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00837413 0.         0.        ]
 [0.         0.00696295 0.        ]
 [0.         0.         0.00730056]]
Resulting covariance matrix for run date DatetimeIndex(['2023-09-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.40252087e-03  3.07048276e-05 -1.70761863e-04]
 [ 3.07048276e-05  9.69653290e-04  9.57651483e-04]
 [-1.70761863e-04  9.57651483e-04  1.06596377e-03]]
Expected returns vector for run date DatetimeIndex(['2023-09-29'],
 ↪ dtype='datetime64[ns]'):
[-0.00268746  0.00765288  0.00968489]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-09-29'],
 ↪ dtype='datetime64[ns]')
[8.32667268e-17 5.55111512e-17 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2023-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-10-30'], dtype='datetime64[ns]'):
[[1.         0.1589013  0.16256741]
 [0.1589013  1.         0.77692628]
 [0.16256741 0.77692628 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.0075788  0.         0.        ]
 [0.         0.00525747 0.        ]
 [0.         0.         0.00711841]]
Resulting covariance matrix for run date DatetimeIndex(['2023-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00109133 0.0001203  0.00016664]
 [0.0001203  0.00052518 0.00055245]
 [0.00016664 0.00055245 0.00096276]]
Expected returns vector for run date DatetimeIndex(['2023-10-30'],
 ↪  dtype='datetime64[ns]'):
[-0.00315795  0.00791877  0.01210196]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-10-30'],
 ↪  dtype='datetime64[ns]')
[1.38777878e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-11-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2023-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.05150098 -0.25777161]
 [-0.05150098  1.          0.84674078]
 [-0.25777161  0.84674078  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-11-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00940092 0.         0.        ]
 [0.         0.00688592 0.        ]
 [0.         0.         0.00958392]]
Resulting covariance matrix for run date DatetimeIndex(['2023-11-29'],
 ↪  dtype='datetime64[ns]'):
[[ 1.67916864e-03 -6.33433576e-05 -4.41267926e-04]
 [-6.33433576e-05  9.00900699e-04  1.06171770e-03]
 [-4.41267926e-04  1.06171770e-03  1.74518003e-03]]
Expected returns vector for run date DatetimeIndex(['2023-11-29'],
 ↪  dtype='datetime64[ns]'):
[-0.00032453  0.00639945  0.0096716 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-11-29'],
 ↪  dtype='datetime64[ns]')
[0. 0. 1.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2023-12-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2023-12-29'], dtype='datetime64[ns]'):
[[1.         0.44219026 0.43290788]
 [0.44219026 1.         0.83886766]
 [0.43290788 0.83886766 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2023-12-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00765104 0.         0.        ]
 [0.         0.00926576 0.        ]
 [0.         0.         0.01254023]]
Resulting covariance matrix for run date DatetimeIndex(['2023-12-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00111223 0.00059561 0.00078918]
 [0.00059561 0.00163123 0.00185197]
 [0.00078918 0.00185197 0.00298789]]
Expected returns vector for run date DatetimeIndex(['2023-12-29'],
 ↪  dtype='datetime64[ns]'):
[0.00150491 0.00557884 0.00631286]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2023-12-29'],
 ↪  dtype='datetime64[ns]')
[3.12250226e-17 0.00000000e+00 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2024-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-01-30'], dtype='datetime64[ns]'):
[[ 1.         -0.04020902  0.01584966]
 [-0.04020902  1.          0.81051469]
 [ 0.01584966  0.81051469  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-01-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00703065 0.         0.        ]
 [0.         0.00756862 0.        ]
 [0.         0.         0.01095666]]
Resulting covariance matrix for run date DatetimeIndex(['2024-01-30'],
 ↪  dtype='datetime64[ns]'):
[[ 9.88601848e-04 -4.27923167e-05  2.44187805e-05]
 [-4.27923167e-05  1.14567980e-03  1.34426749e-03]
 [ 2.44187805e-05  1.34426749e-03  2.40096793e-03]]
Expected returns vector for run date DatetimeIndex(['2024-01-30'],
 ↪  dtype='datetime64[ns]'):
[0.00126439 0.00684503 0.00844329]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-01-30'],
 ↪  dtype='datetime64[ns]')
[1.56125113e-17 1.66533454e-15 1.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2024-02-28'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-02-28'], dtype='datetime64[ns]'):
[[1.         0.36531058 0.49578437]
 [0.36531058 1.         0.82393348]
 [0.49578437 0.82393348 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-02-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00822105 0.         0.        ]
 [0.         0.00629269 0.        ]
 [0.         0.         0.00776051]]
Resulting covariance matrix for run date DatetimeIndex(['2024-02-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00108137 0.00030238 0.00050609]
 [0.00030238 0.00063357 0.00064378]
 [0.00050609 0.00064378 0.00096361]]
Expected returns vector for run date DatetimeIndex(['2024-02-28'],
 ↪  dtype='datetime64[ns]'):
[0.00264691 0.00746599 0.00730233]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-02-28'],
 ↪  dtype='datetime64[ns]')
[0.00000000e+00 1.00000000e+00 8.32667268e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2024-03-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2024-03-28'], dtype='datetime64[ns]'):
[[ 1.         -0.14873192  0.02328703]
 [-0.14873192  1.          0.88308823]
 [ 0.02328703  0.88308823  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-03-28'],
 ↪  dtype='datetime64[ns]'):
[[0.00979843 0.         0.        ]
```

```
 [0.         0.00652512 0.        ]
 [0.         0.         0.00967142]]
Resulting covariance matrix for run date DatetimeIndex(['2024-03-28'],
 ↪ dtype='datetime64[ns]'):
[[ 1.53614883e-03 -1.52148983e-04  3.53086282e-05]
 [-1.52148983e-04  6.81234230e-04  8.91666374e-04]
 [ 3.53086282e-05  8.91666374e-04  1.49658064e-03]]
Expected returns vector for run date DatetimeIndex(['2024-03-28'],
 ↪ dtype='datetime64[ns]'):
[0.004915   0.00653815 0.00468635]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-03-28'],
 ↪ dtype='datetime64[ns]')
[7.35522754e-16 1.00000000e+00 0.00000000e+00]
============================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-04-29'], dtype='datetime64[ns]'):
[[1.         0.00982244 0.01213215]
 [0.00982244 1.         0.79048014]
 [0.01213215 0.79048014 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.00837519 0.         0.        ]
 [0.         0.00678028 0.        ]
 [0.         0.         0.00972718]]
Resulting covariance matrix for run date DatetimeIndex(['2024-04-29'],
 ↪ dtype='datetime64[ns]'):
[[1.33273241e-03 1.05977875e-05 1.87790279e-05]
 [1.05977875e-05 8.73472169e-04 9.90556496e-04]
 [1.87790279e-05 9.90556496e-04 1.79774259e-03]]
Expected returns vector for run date DatetimeIndex(['2024-04-29'],
 ↪ dtype='datetime64[ns]'):
[0.00635336 0.00723615 0.00697566]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-04-29'],
 ↪ dtype='datetime64[ns]')
[0.25843619 0.74156381 0.        ]
============================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-05-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-05-30'], dtype='datetime64[ns]'):
[[1.         0.20206282 0.24811433]
 [0.20206282 1.         0.93327606]
 [0.24811433 0.93327606 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00946018 0.         0.        ]
 [0.         0.01122154 0.        ]
 [0.         0.         0.01766371]]
Resulting covariance matrix for run date DatetimeIndex(['2024-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00161091 0.00038611 0.00074629]
 [0.00038611 0.00226661 0.00332979]
 [0.00074629 0.00332979 0.00561612]]
Expected returns vector for run date DatetimeIndex(['2024-05-30'],
 ↪ dtype='datetime64[ns]'):
[0.00593453 0.00742116 0.00839533]
```

```
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-05-30'],
 ↪ dtype='datetime64[ns]')
[0.15807724 0.84192276 0.        ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-06-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2024-06-28'], dtype='datetime64[ns]'):
[[1.         0.39519382 0.4190869 ]
 [0.39519382 1.         0.67914629]
 [0.4190869  0.67914629 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-06-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01141775 0.         0.        ]
 [0.         0.00557403 0.        ]
 [0.         0.         0.00702434]]
Resulting covariance matrix for run date DatetimeIndex(['2024-06-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00273767 0.00052818 0.00070585]
 [0.00052818 0.00065247 0.00055842]
 [0.00070585 0.00055842 0.00103617]]
Expected returns vector for run date DatetimeIndex(['2024-06-28'],
 ↪ dtype='datetime64[ns]'):
[0.004617   0.00554682 0.00545776]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-06-28'],
 ↪ dtype='datetime64[ns]')
[5.54244151e-16 1.00000000e+00 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-07-30'], dtype='datetime64[ns]'):
[[1.         0.4156265  0.50236881]
 [0.4156265  1.         0.8852821 ]
 [0.50236881 0.8852821  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00848467 0.         0.        ]
 [0.         0.00953937 0.        ]
 [0.         0.         0.00932657]]
Resulting covariance matrix for run date DatetimeIndex(['2024-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00115183 0.00053824 0.00063606]
 [0.00053824 0.00145599 0.00126021]
 [0.00063606 0.00126021 0.00139176]]
Expected returns vector for run date DatetimeIndex(['2024-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.00636064 0.00732007 0.00758913]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-07-30'],
 ↪ dtype='datetime64[ns]')
[0.00993433 0.         0.99006567]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-08-26'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2024-08-26'], dtype='datetime64[ns]'):
[[1.         0.28598282 0.34780937]
```

```
 [0.28598282 1.         0.86376899]
 [0.34780937 0.86376899 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-08-26'],
 ↪ dtype='datetime64[ns]'):
[[0.0081021  0.         0.        ]
 [0.         0.00861949 0.        ]
 [0.         0.         0.01052216]]
Resulting covariance matrix for run date DatetimeIndex(['2024-08-26'],
 ↪ dtype='datetime64[ns]'):
[[0.00091902 0.00027961 0.00041512]
 [0.00027961 0.00104014 0.00109676]
 [0.00041512 0.00109676 0.00155002]]
Expected returns vector for run date DatetimeIndex(['2024-08-26'],
 ↪ dtype='datetime64[ns]'):
[0.00683014 0.00849365 0.00762924]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-08-26'],
 ↪ dtype='datetime64[ns]')
[2.16244814e-01 7.83755186e-01 1.10565602e-16]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-09-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-09-27'], dtype='datetime64[ns]'):
[[1.         0.09538785 0.07632987]
 [0.09538785 1.         0.87001443]
 [0.07632987 0.87001443 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00818097 0.         0.        ]
 [0.         0.00783887 0.        ]
 [0.         0.         0.01103493]]
Resulting covariance matrix for run date DatetimeIndex(['2024-09-27'],
 ↪ dtype='datetime64[ns]'):
[[0.00127164 0.00011623 0.00013093]
 [0.00011623 0.00116751 0.00142989]
 [0.00013093 0.00142989 0.00231363]]
Expected returns vector for run date DatetimeIndex(['2024-09-27'],
 ↪ dtype='datetime64[ns]'):
[0.00810153 0.00652529 0.0056773 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-09-27'],
 ↪ dtype='datetime64[ns]')
[0.71884943 0.28115057 0.        ]
=======================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2024-10-25'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-10-25'], dtype='datetime64[ns]'):
[[ 1.         -0.09962071 -0.07905012]
 [-0.09962071  1.          0.89843786]
 [-0.07905012  0.89843786  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-10-25'],
 ↪ dtype='datetime64[ns]'):
[[0.01137333 0.         0.        ]
 [0.         0.01047522 0.        ]
 [0.         0.         0.01145172]]
Resulting covariance matrix for run date DatetimeIndex(['2024-10-25'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00232835 -0.00021364 -0.00018532]
```

```
 [-0.00021364  0.00197514  0.00193997]
 [-0.00018532  0.00193997  0.00236055]]
Expected returns vector for run date DatetimeIndex(['2024-10-25'],
 ↪  dtype='datetime64[ns]'):
[0.01018914 0.00578597 0.00482121]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-10-25'],
 ↪  dtype='datetime64[ns]')
[8.18131335e-01 1.81868665e-01 1.17721373e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2024-11-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2024-11-29'], dtype='datetime64[ns]'):
[[ 1.         -0.14499882 -0.07520898]
 [-0.14499882  1.          0.87592036]
 [-0.07520898  0.87592036  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-11-29'],
 ↪  dtype='datetime64[ns]'):
[[0.00979463 0.         0.        ]
 [0.         0.00755994 0.        ]
 [0.         0.         0.00846417]]
Resulting covariance matrix for run date DatetimeIndex(['2024-11-29'],
 ↪  dtype='datetime64[ns]'):
[[ 0.00201463 -0.00022547 -0.00013094]
 [-0.00022547  0.00120021  0.00117703]
 [-0.00013094  0.00117703  0.00150448]]
Expected returns vector for run date DatetimeIndex(['2024-11-29'],
 ↪  dtype='datetime64[ns]'):
[0.01004607 0.00394358 0.00248406]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-11-29'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 1.31838984e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪  DatetimeIndex(['2024-12-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2024-12-30'], dtype='datetime64[ns]'):
[[1.         0.41740462 0.31101659]
 [0.41740462 1.         0.86547875]
 [0.31101659 0.86547875 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2024-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.00853592 0.         0.        ]
 [0.         0.00876894 0.        ]
 [0.         0.         0.01088687]]
Resulting covariance matrix for run date DatetimeIndex(['2024-12-30'],
 ↪  dtype='datetime64[ns]'):
[[0.0015301  0.00065611 0.00060695]
 [0.00065611 0.00161478 0.0017351 ]
 [0.00060695 0.0017351  0.002489  ]]
Expected returns vector for run date DatetimeIndex(['2024-12-30'],
 ↪  dtype='datetime64[ns]'):
[0.00937405 0.0055706  0.00549039]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2024-12-30'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 2.77555756e-17 6.20749544e-18]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-01-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-01-30'], dtype='datetime64[ns]'):
[[ 1.          0.10198499 -0.00418356]
 [ 0.10198499  1.          0.84305483]
 [-0.00418356  0.84305483  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-01-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00876978 0.         0.        ]
 [0.         0.00835658 0.        ]
 [0.         0.         0.00839327]]
Resulting covariance matrix for run date DatetimeIndex(['2025-01-30'],
 ↪ dtype='datetime64[ns]'):
[[ 1.38436316e-03  1.34532183e-04 -5.54291955e-06]
 [ 1.34532183e-04  1.25698416e-03  1.06435920e-03]
 [-5.54291955e-06  1.06435920e-03  1.26804593e-03]]
Expected returns vector for run date DatetimeIndex(['2025-01-30'],
 ↪ dtype='datetime64[ns]'):
[0.00825028 0.00429282 0.0047468 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-01-30'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-02-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-02-27'], dtype='datetime64[ns]'):
[[1.         0.14005393 0.10059798]
 [0.14005393 1.         0.85921925]
 [0.10059798 0.85921925 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.0088064  0.         0.        ]
 [0.         0.00889141 0.        ]
 [0.         0.         0.01218567]]
Resulting covariance matrix for run date DatetimeIndex(['2025-02-27'],
 ↪ dtype='datetime64[ns]'):
[[0.0014735  0.00020836 0.00020511]
 [0.00020836 0.00150209 0.0017688 ]
 [0.00020511 0.0017688  0.00282132]]
Expected returns vector for run date DatetimeIndex(['2025-02-27'],
 ↪ dtype='datetime64[ns]'):
[0.0103165  0.00425956 0.00149512]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-02-27'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 1.31838984e-16 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-03-28'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2025-03-28'], dtype='datetime64[ns]'):
[[1.         0.392874   0.41796044]
 [0.392874   1.         0.90430975]
 [0.41796044 0.90430975 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.01829917 0.         0.        ]
```

```
 [0.         0.01262153 0.        ]
 [0.         0.         0.01338513]]
Resulting covariance matrix for run date DatetimeIndex(['2025-03-28'],
 ↪ dtype='datetime64[ns]'):
[[0.00636234 0.00172405 0.0019451 ]
 [0.00172405 0.00302676 0.00290272]
 [0.0019451  0.00290272 0.00340407]]
Expected returns vector for run date DatetimeIndex(['2025-03-28'],
 ↪ dtype='datetime64[ns]'):
[0.00979403 0.0020863  0.00028527]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-03-28'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 0.00000000e+00 1.83645618e-16]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-04-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-04-29'], dtype='datetime64[ns]'):
[[ 1.         -0.10959726 -0.28132194]
 [-0.10959726  1.          0.90123272]
 [-0.28132194  0.90123272  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-04-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01737732 0.         0.        ]
 [0.         0.01108933 0.        ]
 [0.         0.         0.01025984]]
Resulting covariance matrix for run date DatetimeIndex(['2025-04-29'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00603943 -0.00042239 -0.00100313]
 [-0.00042239  0.00245947  0.00205075]
 [-0.00100313  0.00205075  0.00210529]]
Expected returns vector for run date DatetimeIndex(['2025-04-29'],
 ↪ dtype='datetime64[ns]'):
[0.01174601 0.0026963  0.00194766]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-04-29'],
 ↪ dtype='datetime64[ns]')
[1.00000000e+00 5.55111512e-17 0.00000000e+00]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-05-30'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2025-05-30'], dtype='datetime64[ns]'):
[[ 1.         -0.44407389 -0.42476285]
 [-0.44407389  1.          0.88374943]
 [-0.42476285  0.88374943  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-05-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01076052 0.         0.        ]
 [0.         0.00683802 0.        ]
 [0.         0.         0.00669266]]
Resulting covariance matrix for run date DatetimeIndex(['2025-05-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00231578 -0.0006535  -0.0006118 ]
 [-0.0006535   0.00093517  0.00080889]
 [-0.0006118   0.00080889  0.00089583]]
Expected returns vector for run date DatetimeIndex(['2025-05-30'],
 ↪ dtype='datetime64[ns]'):
```

```
[0.01302131 0.00422453 0.00400196]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-05-30'],
 ↪ dtype='datetime64[ns]')
[7.21274488e-01 2.78725512e-01 1.84982990e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-06-27'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-06-27'], dtype='datetime64[ns]'):
[[ 1.         -0.06227224 -0.0110902 ]
 [-0.06227224  1.          0.87623268]
 [-0.0110902   0.87623268  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-06-27'],
 ↪ dtype='datetime64[ns]'):
[[0.01093362 0.         0.        ]
 [0.         0.00537102 0.        ]
 [0.         0.         0.00646167]]
Resulting covariance matrix for run date DatetimeIndex(['2025-06-27'],
 ↪ dtype='datetime64[ns]'):
[[ 2.74951209e-03 -8.41091229e-05 -1.80208736e-05]
 [-8.41091229e-05  6.63500958e-04  6.99437172e-04]
 [-1.80208736e-05  6.99437172e-04  9.60322199e-04]]
Expected returns vector for run date DatetimeIndex(['2025-06-27'],
 ↪ dtype='datetime64[ns]'):
[0.0147863  0.00590527 0.00487952]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-06-27'],
 ↪ dtype='datetime64[ns]')
[7.22297141e-01 2.77702859e-01 1.65919777e-17]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-07-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-07-30'], dtype='datetime64[ns]'):
[[ 1.         -0.14790609 -0.1556226 ]
 [-0.14790609  1.          0.90407377]
 [-0.1556226   0.90407377  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-07-30'],
 ↪ dtype='datetime64[ns]'):
[[0.00945676 0.         0.        ]
 [0.         0.00614822 0.        ]
 [0.         0.         0.00599354]]
Resulting covariance matrix for run date DatetimeIndex(['2025-07-30'],
 ↪ dtype='datetime64[ns]'):
[[ 0.00169918 -0.00016339 -0.00016759]
 [-0.00016339  0.00071821  0.00063298]
 [-0.00016759  0.00063298  0.00068253]]
Expected returns vector for run date DatetimeIndex(['2025-07-30'],
 ↪ dtype='datetime64[ns]'):
[0.01542077 0.00825696 0.00738654]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-07-30'],
 ↪ dtype='datetime64[ns]')
[0.537612 0.462388 0.      ]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-08-29'], dtype='datetime64[ns]')
Dates across actual datasets match
Correlation matrix for run date DatetimeIndex(['2025-08-29'], dtype='datetime64[ns]'):
[[ 1.         -0.05191574  0.06072681]
```

```
 [-0.05191574  1.          0.89820975]
 [ 0.06072681  0.89820975  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-08-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01039295 0.         0.        ]
 [0.         0.00521555 0.        ]
 [0.         0.         0.00647729]]
Resulting covariance matrix for run date DatetimeIndex(['2025-08-29'],
 ↪ dtype='datetime64[ns]'):
[[ 1.40417380e-03 -3.65831469e-05  5.31442229e-05]
 [-3.65831469e-05  3.53625229e-04  3.94470488e-04]
 [ 5.31442229e-05  3.94470488e-04  5.45419000e-04]]
Expected returns vector for run date DatetimeIndex(['2025-08-29'],
 ↪ dtype='datetime64[ns]'):
[0.01605236 0.00525068 0.00386551]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-08-29'],
 ↪ dtype='datetime64[ns]')
[9.24859069e-01 7.51409310e-02 5.66320421e-18]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-09-29'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-09-29'], dtype='datetime64[ns]'):
[[ 1.         -0.01754101  0.0992156 ]
 [-0.01754101  1.          0.91508466]
 [ 0.0992156   0.91508466  1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-09-29'],
 ↪ dtype='datetime64[ns]'):
[[0.01960173 0.         0.        ]
 [0.         0.00499428 0.        ]
 [0.         0.         0.00538896]]
Resulting covariance matrix for run date DatetimeIndex(['2025-09-29'],
 ↪ dtype='datetime64[ns]'):
[[ 6.91609724e-03 -3.09096684e-05  1.88647970e-04]
 [-3.09096684e-05  4.48970936e-04  4.43314561e-04]
 [ 1.88647970e-04  4.43314561e-04  5.22736968e-04]]
Expected returns vector for run date DatetimeIndex(['2025-09-29'],
 ↪ dtype='datetime64[ns]'):
[0.01753572 0.00458413 0.00223294]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-09-29'],
 ↪ dtype='datetime64[ns]')
[1. 0. 0.]
========================================================
Commencing model training and walkforward backtest for rebalance date:
 ↪ DatetimeIndex(['2025-10-30'], dtype='datetime64[ns]')
Correlation matrix for run date DatetimeIndex(['2025-10-30'], dtype='datetime64[ns]'):
[[1.         0.16506505 0.04178775]
 [0.16506505 1.         0.85286856]
 [0.04178775 0.85286856 1.        ]]
Volatility diagonal vector for run date DatetimeIndex(['2025-10-30'],
 ↪ dtype='datetime64[ns]'):
[[0.01171511 0.         0.        ]
 [0.         0.00518608 0.        ]
 [0.         0.         0.00516476]]
Resulting covariance matrix for run date DatetimeIndex(['2025-10-30'],
 ↪ dtype='datetime64[ns]'):
[[2.33314331e-03 1.70486156e-04 4.29827239e-05]
```

```
 [1.70486156e-04 4.57221417e-04 3.88346696e-04]
 [4.29827239e-05 3.88346696e-04 4.53469891e-04]]
Expected returns vector for run date DatetimeIndex(['2025-10-30'],
 ↪  dtype='datetime64[ns]'):
[0.02085095 0.00524959 0.0028932 ]
Resulting weights (from maximize sharpe rule) for run date DatetimeIndex(['2025-10-30'],
 ↪  dtype='datetime64[ns]')
[1.00000000e+00 1.04083409e-16 0.00000000e+00]
XGBoost implementation complete. Data loaded into csv files.
Total time taken by XGB walkforward backtest program: 102.09 minutes.
```

Section 9: Out-of-Sample Performance Evaluation

This section focuses on evaluating the Monte-Carlo and XGBoost models on the following out-of-sample (OOS) parameters:

1. Rolling realized volatility (12Months)
2. Rolling model-implied volatility (under fixed weights w^T)
3. Regime-controlled Sharpe and Volatility
4. Regime-controlled Sharpe versus CVaR Trade-off
5. Tail Loss Frequency
6. Expected Shortfall Duration
7. Maximum Drawdown
8. Monthly Turnover

While standard OOS predictive accuracy metrics such as RMSE, MAE, and R-squared have been computed for the XGBoost models, these metrics are **not used for out-of-sample portfolio performance evaluation**. This is because the objective of this study is to assess the effectiveness of conditional risk estimation, specifically second-moment processes such as volatility and correlation, rather than the accuracy of point forecasts of returns (a first-moment process). Since metrics such as RMSE and MAE are designed to evaluate point-estimate forecasts, their use in evaluating portfolio outcomes driven by risk forecasts would not be methodologically aligned with the objectives of this study.

To recap, our modelling approach followed the below methodology:

1. **Statistical evaluation of the underlying log returns** - to determine applicability of gaussian models as well as guide feature enginieering (for Monte-Carlo and XGBoost approaches respectively).
2. **Portfolio construction rule design (Maximize Sharpe)** - to standardize the portfolio construction mechanism across all approaches - for an apple-to-apple comparison.
3. **Setup of Gaussian and Non-Gaussian (Bootstrap) Monte-Carlo** - while all our statistical tests empirically established Non-Gaussian Monte-Carlo as the choice of approach, Gaussian Monte-Carlo was also included - purely for this study. To credibly establish the need for upfront statistical tests - and conforming to the same during real-world applications.
4. **Setup of XGBoost** - to define the dependent and independent datasets for model training.
5. **Setup of walkforward backtest** - to define (and standardize) the walkforward backtest mechanism - across all 3 approaches - to enforce an apple-to-apple comparison (1 month walkforward - holding the weights constant - and 3 years backtest)

This section evaluates the three portfolio construction approaches across complementary dimensions of risk-adjusted outcomes, tail-risk realism, and portfolio stability:

1. **Risk-adjusted performance comparison:** This sub-section will leverage OOS parameters 1-4 above, to provide insights on what level of return efficiency remains - given each model's risk controls.
2. **Tail risk and drawdown analysis:** This sub-section will leverage OOS parameters 5-7 above, to provide insights on which approach captures risk better (across all 3).
3. **Portfolio stability:** This sub-section will leverage OOS parameter 8 above, to provide insights on which approach results in a more stable - risk resilient - portfolio.

9.1. Risk-adjusted Performance Comparison (Gaussian Monte-Carlo v/s Bootstrap Monte-Carlo v/s XGBoost)

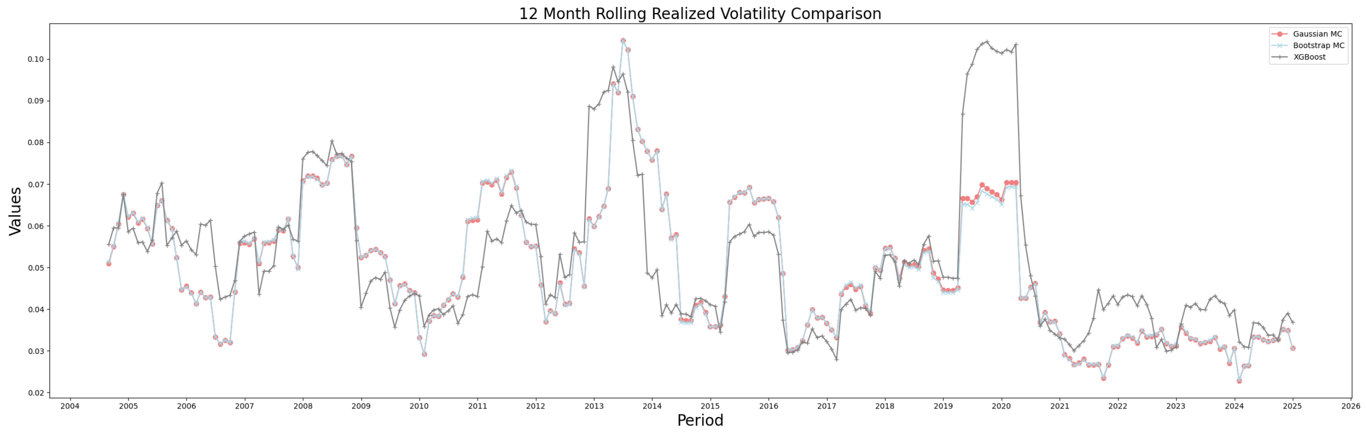
This sub-section evaluates the residual risk-adjusted performance of all three portfolio construction approaches, conditional on prevailing market regimes. The objective is not to identify the approach that maximizes returns, but to assess the level of return efficiency that remains after each model's risk estimation and control mechanisms have been applied.

The following out-of-sample metrics are employed:

1. Rolling realized volatility (12-month window)
2. Rolling model-implied volatility (under fixed weights w)
3. Regime-controlled Sharpe and Volatility
4. Regime-controlled Sharpe versus CVaR Trade-off

Metrics 1 and 2 focus on differences in risk recognition and risk pricing behavior across approaches and metrics 3 and 4 identify how this translates to Sharpe. Since our study objective is "risk-adjusted portfolio construction" we want the ideal approach to be one that exhibits timely and regime-consistent risk sensitivity, rather than unconditional risk minimization - which metrics 1 and 2 will shed light on - and translate the risk awareness into stable excess returns - which will answer the question: Had an investor invested $100 in the portfolio, what kind of excess returns would they have received - given each approach's risk identification profile? In a real world scenario - this model would then ideally be used to benchmark multiple portfolios.

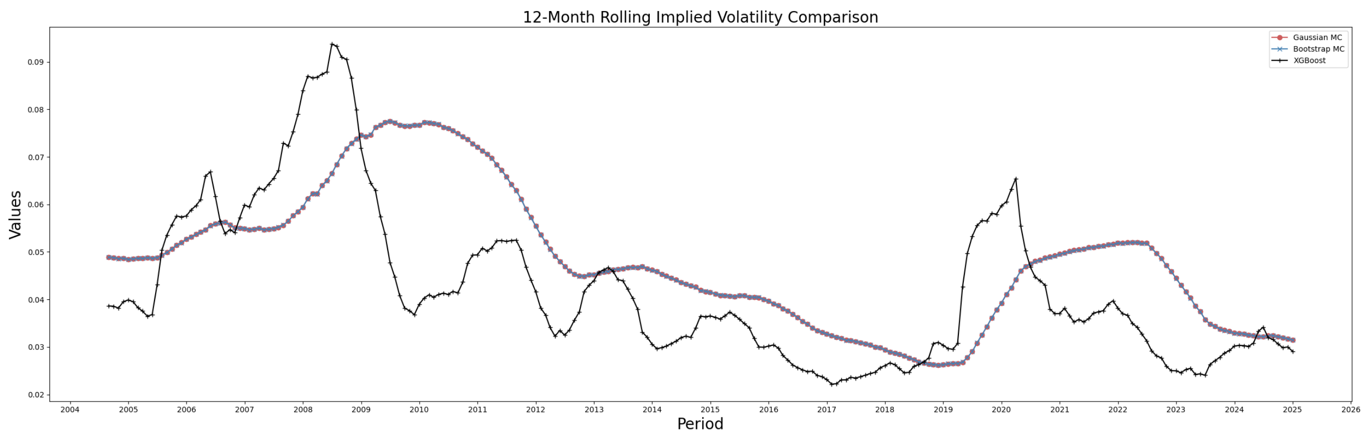
Sharpe ratios, and volatility, are evaluated conditionally across calm and stress regimes to avoid masking regime-specific risk–return trade-offs, thereby enabling a more nuanced assessment of how each approach balances return efficiency against downside risk within its respective risk estimation framework.



Analysis Conclusion - Rolling Realized Volatility

The rolling 12-month realized volatility profiles of the Gaussian and Bootstrap Monte-Carlo approaches exhibit closely aligned and relatively smooth dynamics across the sample, reflecting gradual risk absorption and mean-reverting behavior. Bootstrap Monte-Carlo displays marginally higher realized volatility during stress periods, consistent with its ability to incorporate empirical distributional features beyond the Gaussian assumption. However, the overall similarity in volatility trajectories indicates that both Monte-Carlo frameworks primarily smooth risk over time, normalizing portfolio variability as shocks are absorbed into the rolling window.

In contrast, the XGBoost-based approach exhibits sharper transitions and greater persistence in realized volatility once higher-risk regimes are identified. This behavior reflects the model's regime-conditional design, whereby portfolio allocations adjust discretely in response to changes in volatility and correlation states rather than continuously. As a result, XGBoost maintains elevated realized volatility for longer durations during and following stress events, avoiding rapid re-risking during transitional market phases. Importantly, this volatility profile does not indicate excessive risk-taking, but rather a structurally more conservative response that prioritizes sustained risk awareness over volatility compression, reinforcing the model's suitability for robust risk identification and capture.



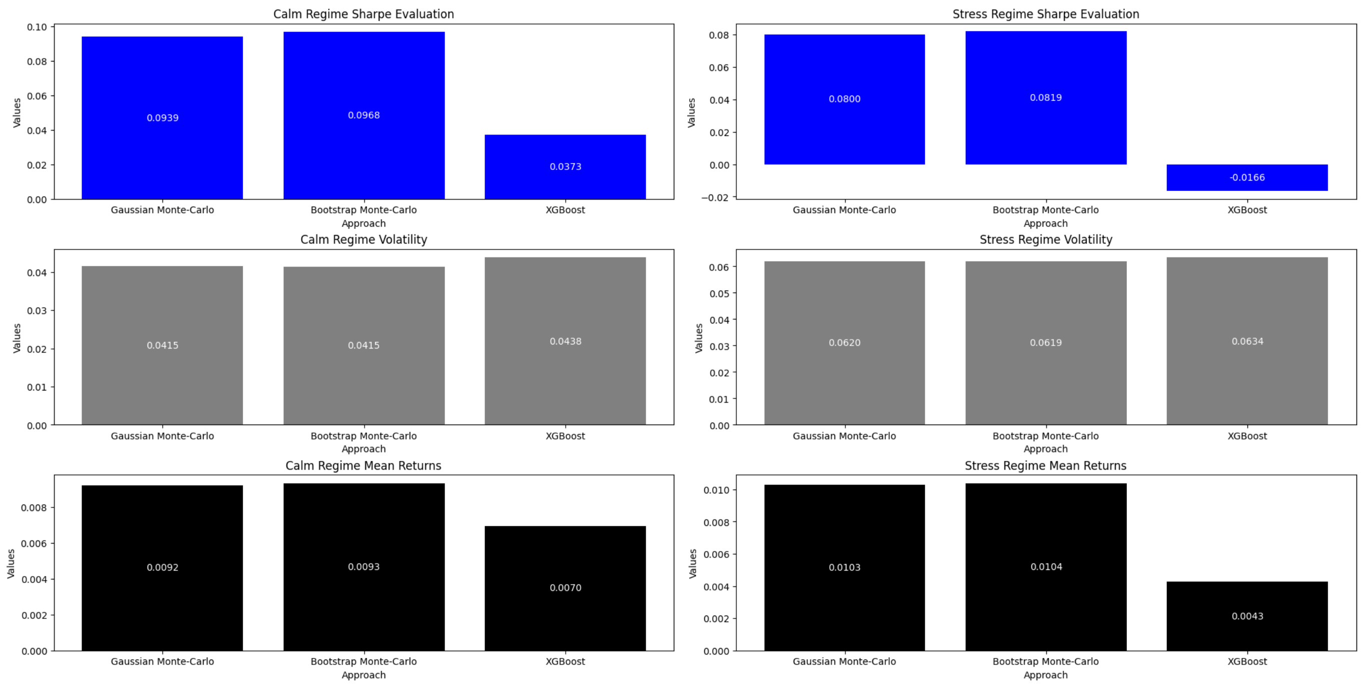
Analysis Conclusion - Rolling Implied Volatility

The implied- and realized-volatility profiles show that all three approaches respond to major stress regimes, with the XGBoost-based framework exhibiting sharper and higher-amplitude risk responses around the 2008–09 financial crisis and the 2019–20 COVID episode. Under fixed equal weights, XGBoost produces more abrupt increases in model-implied volatility, consistent with stronger regime sensitivity embedded in its volatility and correlation estimates. In contrast, both Monte-Carlo approaches display smoother and more persistent volatility plateaus, indicative of slower risk normalization and stronger averaging effects inherent in distribution-based covariance estimation.

Importantly, differences in spike magnitude should be interpreted as differences in implied risk assigned to the same equal-weight portfolio, rather than as definitive evidence of superior or inferior estimation of "true" risk. Realized volatility can converge across approaches even when implied volatility diverges, as realized outcomes reflect each model's endogenous allocation choices and the trailing-window construction used in realized-risk measurement.

Overall, the results indicate that XGBoost reacts more discontinuously to regime shifts, while Monte-Carlo models adjust risk more gradually, consistent with their respective modelling assumptions. The discontinuous response observed under XGBoost reflects a more decisive regime classification mechanism, whereas Monte-Carlo approaches produce smoother, continuously adjusted risk estimates driven by distributional averaging.

```
Number of calm years (out of total evaluation of 21 years): 110
Number of stress years (out of total evaluation of 21 years): 111
```



Analysis Conclusion - Regime-Controlled Sharpe and Volatility

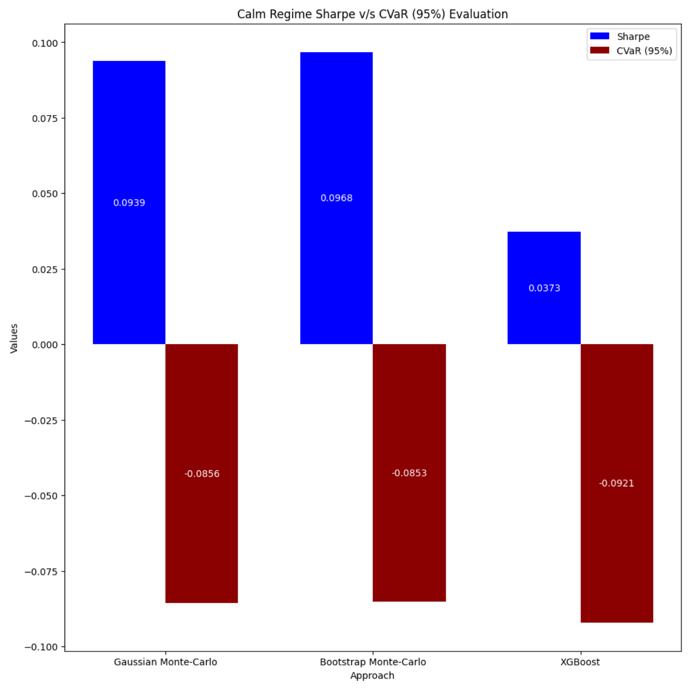
The regime-controlled Sharpe and volatility analysis reinforces the structural differences observed in the implied- and realized-volatility assessments, highlighting how each modelling framework translates its risk perception into realized portfolio outcomes under distinct market conditions.

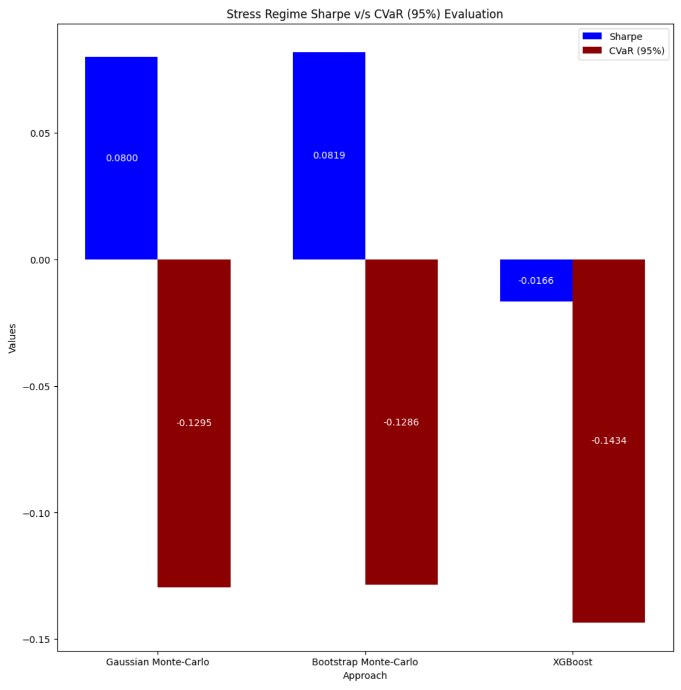
Across both calm and stress regimes, the Gaussian and Bootstrap Monte-Carlo approaches exhibit closely aligned Sharpe ratios and realized volatility levels, with Bootstrap Monte-Carlo delivering a marginal Sharpe advantage in both regimes. This modest improvement remains consistent with the Bootstrap framework's ability to relax Gaussian distributional assumptions while still benefiting from the smoothing and averaging effects inherent in Monte-Carlo covariance estimation. Importantly, the similarity in regime-level volatility between the two Monte-Carlo variants indicates that their risk identification and normalization dynamics remain broadly comparable, even during stress periods.

In contrast, the XGBoost-based approach reports materially lower Sharpe ratios across both regimes, accompanied by slightly higher realized volatility, particularly in calm market conditions. When interpreted alongside the implied-volatility analysis, this outcome reflects a structural feature of the XGBoost framework rather than inferior execution. The model exhibits a more discontinuous and regime-sensitive risk perception, leading to conservative allocations that persist even when realized volatility appears subdued. As a result, XGBoost forgoes return re-expansion during calm regimes in favor of maintaining heightened risk awareness.

During stress regimes, realized volatility under XGBoost converges toward Monte-Carlo levels despite higher implied-risk signals, indicating that the model's endogenous allocation adjustments partially offset elevated risk perceptions. This convergence underscores the distinction between ex-ante risk identification and ex-post realized outcomes: while XGBoost assigns higher implied risk to stress environments, its portfolio construction dampens realized volatility through defensive positioning, leading to Sharpe suppression without disproportionate volatility escalation.

Overall, the regime-controlled results are fully consistent with the broader volatility diagnostics. Monte-Carlo approaches—particularly Bootstrap Monte-Carlo—deliver smoother risk normalization and marginally superior Sharpe performance through continuous, distribution-driven adjustment. By contrast, the XGBoost framework demonstrates a more decisive and conservative regime classification, prioritizing robust risk identification and persistence over return maximization. The resulting Sharpe suppression should therefore be interpreted not as a weakness, but as evidence that XGBoost emphasizes regime awareness and downside protection, aligning with the study's objective of risk-aware portfolio construction.





Analysis Conclusion - Regime-Controlled Sharpe v/s CVaR

The regime-wise Sharpe–CVaR comparison complements the implied and realized volatility diagnostics by explicitly characterizing how each modelling framework trades off return efficiency against tail-risk exposure under different market conditions. Together, these results clarify not only how much risk each approach realizes, but why that risk materializes given the model's underlying risk perception and allocation behavior.

Calm Regime: During low-stress market conditions, both Gaussian and Bootstrap Monte-Carlo approaches exhibit comparable and well-contained tail risk, with nearly identical CVaR magnitudes at the 95% confidence level. Bootstrap Monte-Carlo delivers a marginal Sharpe advantage over its Gaussian counterpart without a corresponding deterioration in CVaR, consistent with its modestly improved return extraction observed in the regime-controlled Sharpe and volatility analysis. This outcome aligns with the implied-volatility findings, where both Monte-Carlo models assign similar risk levels to calm regimes and normalize risk smoothly through distribution-driven averaging.

In contrast, XGBoost reports a materially lower Sharpe ratio alongside a more severe CVaR even in calm regimes. When interpreted jointly with the implied-volatility analysis, this reflects a conservative risk stance rather than poor execution. XGBoost assigns elevated implied risk to transitional and ostensibly calm periods and, as a result, maintains defensive positioning that suppresses upside participation. The higher calm-regime CVaR therefore arises not from excessive risk-taking, but from a reluctance to re-risk aggressively following prior stress, consistent with the model's regime-persistent design.

Stress Regime: Under stressed market conditions, differences across approaches become more pronounced. Both Monte-Carlo frameworks continue to generate positive Sharpe ratios while containing CVaR losses at levels materially smaller than those observed under XGBoost. Bootstrap Monte-Carlo again marginally outperforms the Gaussian specification on Sharpe without increasing tail exposure, reinforcing its relative robustness under adverse distributional shifts. These results are consistent with the realized-volatility analysis, where Monte-Carlo portfolios exhibit smoother volatility responses and faster normalization following stress shocks.

XGBoost, by contrast, records the deepest CVaR losses and a negative Sharpe ratio in stress regimes. Importantly, this outcome must be interpreted in conjunction with the implied-volatility findings: XGBoost assigns substantially higher ex-ante risk to stress environments and reacts more discontinuously to regime shifts. While realized volatility ultimately converges toward Monte-Carlo levels due to defensive reallocation, return suppression dominates, resulting in inferior Sharpe performance and more severe CVaR. This pattern reflects heightened sensitivity to tail events rather than an inability to manage realized volatility.

Overall Interpretation: Taken together, the regime-controlled Sharpe–CVaR results reinforce a clear methodological distinction across approaches. Bootstrap Monte-Carlo offers the most balanced risk–return profile, consistently delivering marginal Sharpe improvements without exacerbating tail risk, while Gaussian Monte-Carlo provides stable and predictable outcomes through continuous risk averaging. XGBoost, in contrast, systematically prioritizes tail-risk identification and regime persistence over return efficiency, leading to lower Sharpe ratios and deeper CVaR losses across regimes. Importantly, these outcomes are fully consistent with the model's implied- and realized-volatility behavior and should not be interpreted as underperformance. Rather, they reflect a design philosophy centered on decisive regime classification and conservative risk management, aligning closely with the study's objective of evaluating risk-aware—rather than return-maximizing—portfolio construction frameworks.

9.2. Tail Risk and Drawdown Analysis (Gaussian Monte-Carlo v/s Bootstrap Monte-Carlo v/s XGBoost)

This sub-section evaluates the effectiveness of each portfolio construction approach in capturing and controlling extreme downside risk. Unlike the preceding risk-adjusted performance analysis, which focuses on residual return efficiency after risk controls, the emphasis here is explicitly on loss severity, tail exposure, and capital drawdowns under adverse market conditions.

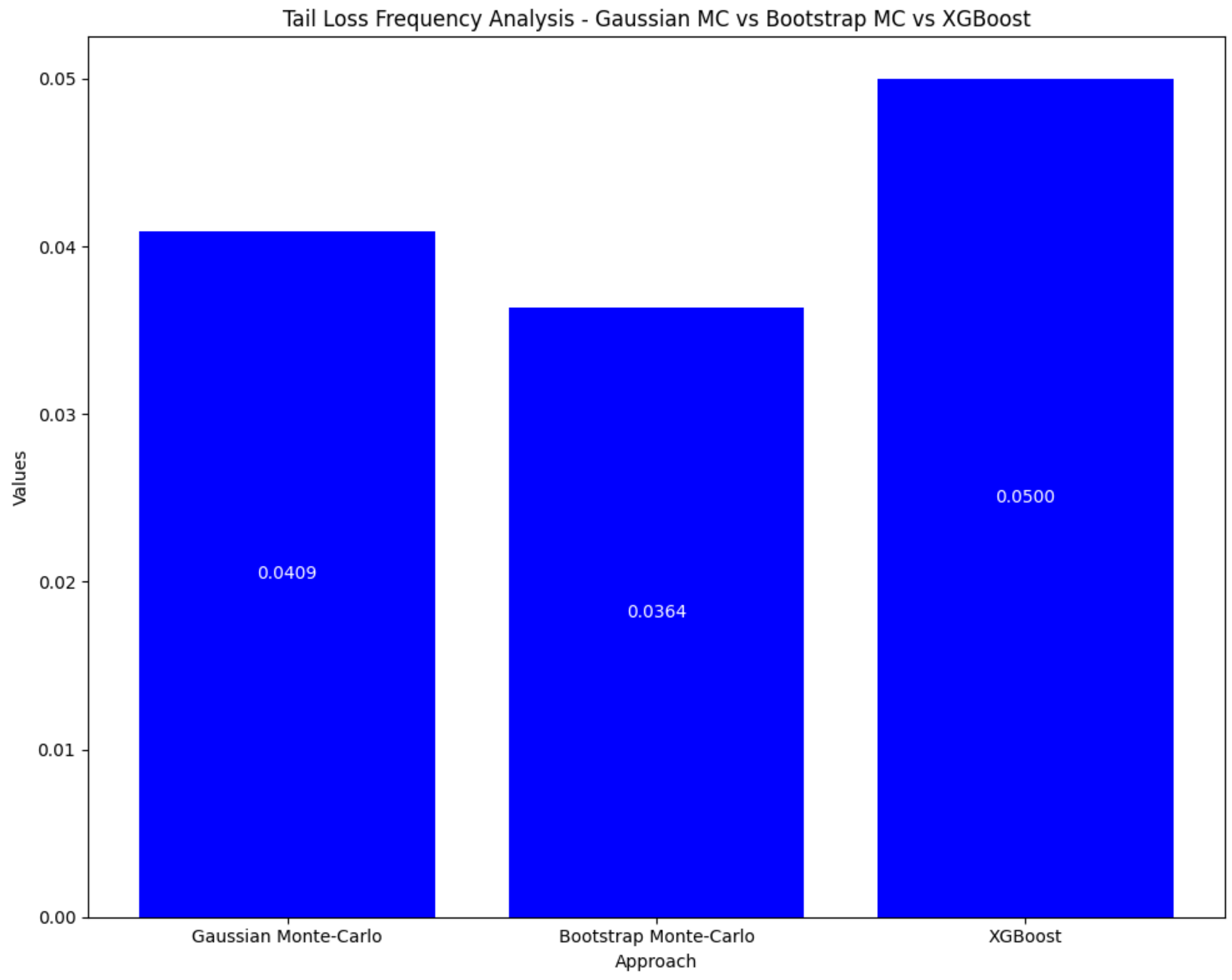
The objective is to assess how well each modelling framework identifies and mitigates persistent and recurring extreme outcomes, rather than how efficiently it converts risk into returns.

The following out-of-sample metrics are employed: 1. Tail Loss Frequency – to measure how often realized portfolio returns breach a predefined downside threshold (e.g., the empirical 5th percentile), capturing the recurrence of extreme negative events 2. Expected Shortfall Duration – to evaluate the average length of consecutive periods spent in the loss tail once a tail event is triggered, reflecting the persistence of downside stress 3. Maximum Drawdown – to quantify peak-to-trough capital erosion over the full evaluation horizon

Tail Loss Frequency captures the incidence of extreme losses, providing insight into how frequently each modelling approach exposes the portfolio to tail-risk realizations. Expected Shortfall Duration complements this by measuring the temporal clustering of tail events, highlighting whether losses are isolated shocks or part of sustained adverse regimes. Together, these metrics provide a more complete characterization of downside risk than single-period loss magnitude alone.

Maximum drawdown is included as a path-dependent measure of capital preservation, capturing the cumulative impact of prolonged or repeated tail-risk realizations on investor wealth.

All tail-risk measures are computed using historical (empirical) distributions to avoid imposing parametric assumptions that may understate non-normality, skewness, or fat-tailed behavior in realized portfolio returns. Maximum drawdown is evaluated over the complete out-of-sample period to reflect real-world investor experience under sustained market stress.



Analysis Conclusion - Tail Loss Frequency

Tail Loss Frequency (TLF) measures how often the portfolio experiences extreme downside outcomes, defined here as monthly returns falling at or below a rolling 36-month empirical 5th percentile threshold. Unlike magnitude-based tail metrics, TLF focuses on the recurrence of tail events, providing insight into how frequently each portfolio construction approach exposes capital to severe downside risk relative to recent historical conditions.

The observed TLF results across the three approaches are as follows: 1. **Bootstrap Monte-Carlo records the lowest tail loss frequency**, with tail events occurring in 3.6% of evaluated months (8 out of 220). 2. **Gaussian Monte-Carlo follows closely**, with a TLF of 4.1% (9 tail months). 3. **XGBoost exhibits**

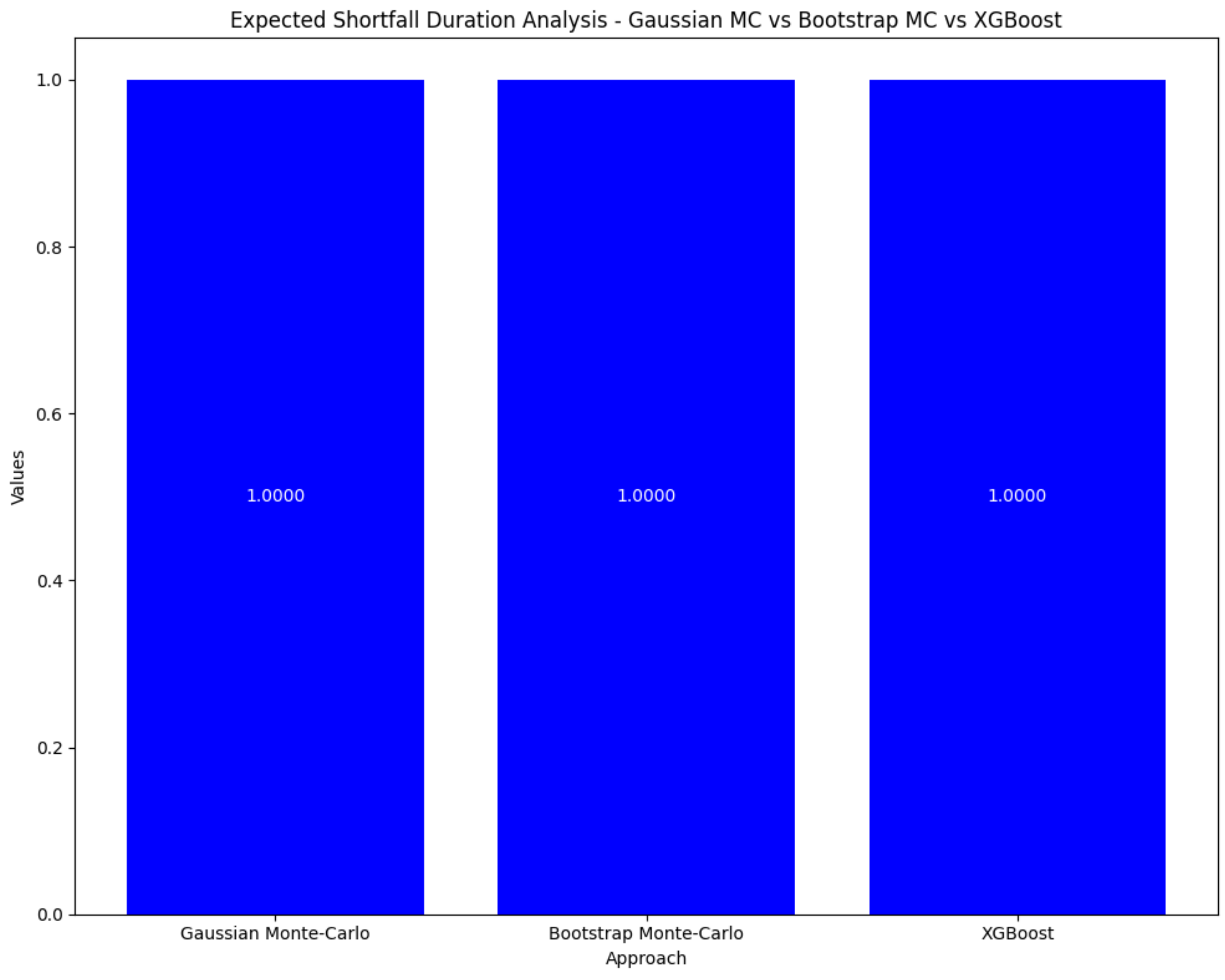
the highest tail loss frequency at 5.0% (11 tail months), closely aligned with the nominal tail probability implied by the 5th percentile threshold.

These results indicate that the Bootstrap Monte-Carlo approach is the most effective at limiting repeated exposure to extreme downside outcomes, while XGBoost allows tail events to occur more frequently over the out-of-sample period.

From a portfolio construction perspective, this finding has direct economic relevance. Frequent tail losses tend to depress cumulative performance and increase volatility, both of which negatively impact risk-adjusted return metrics such as the Sharpe ratio. The lower TLF observed under Bootstrap Monte-Carlo is therefore consistent with its superior Sharpe performance in this study, as fewer extreme loss episodes reduce drawdowns and stabilize return distributions over time. Conversely, XGBoost's higher tail loss frequency helps explain its relatively weaker Sharpe ratio, as more frequent tail breaches introduce additional downside volatility that is not sufficiently offset by higher returns.

Importantly, these results do not imply that XGBoost fails to identify risk. Prior regime-controlled volatility assessments indicate that XGBoost is more responsive to changes in market conditions. However, this responsiveness translates into greater willingness to re-enter risk, which increases the frequency of tail exposures at the portfolio level. In contrast, Bootstrap Monte-Carlo embeds a more conservative, distribution-aware allocation mechanism that prioritizes tail avoidance, even at the cost of slower re-risking.

In the context of this study's objective—risk-adjusted portfolio construction with a stronger emphasis on downside risk control—the TLF results support the conclusion that Bootstrap Monte-Carlo provides the most robust balance between risk containment and return generation. By minimizing the recurrence of extreme losses, it creates a return profile that is both more stable and more efficient on a risk-adjusted basis, making it the preferred approach for risk-aware portfolio construction within the scope of this analysis.



Analysis Conclusion - Expected Shortfall Duration

Expected Shortfall Duration (ESD) measures the persistence of tail losses once they occur. Specifically, it captures the average length (in months) of consecutive tail-loss episodes, conditional on a breach of the rolling 36-month empirical 5th percentile threshold. While Tail Loss Frequency (TLF) focuses on how often extreme downside events occur, ESD provides complementary insight into how long the portfolio remains in a stressed tail state before recovering above the threshold.

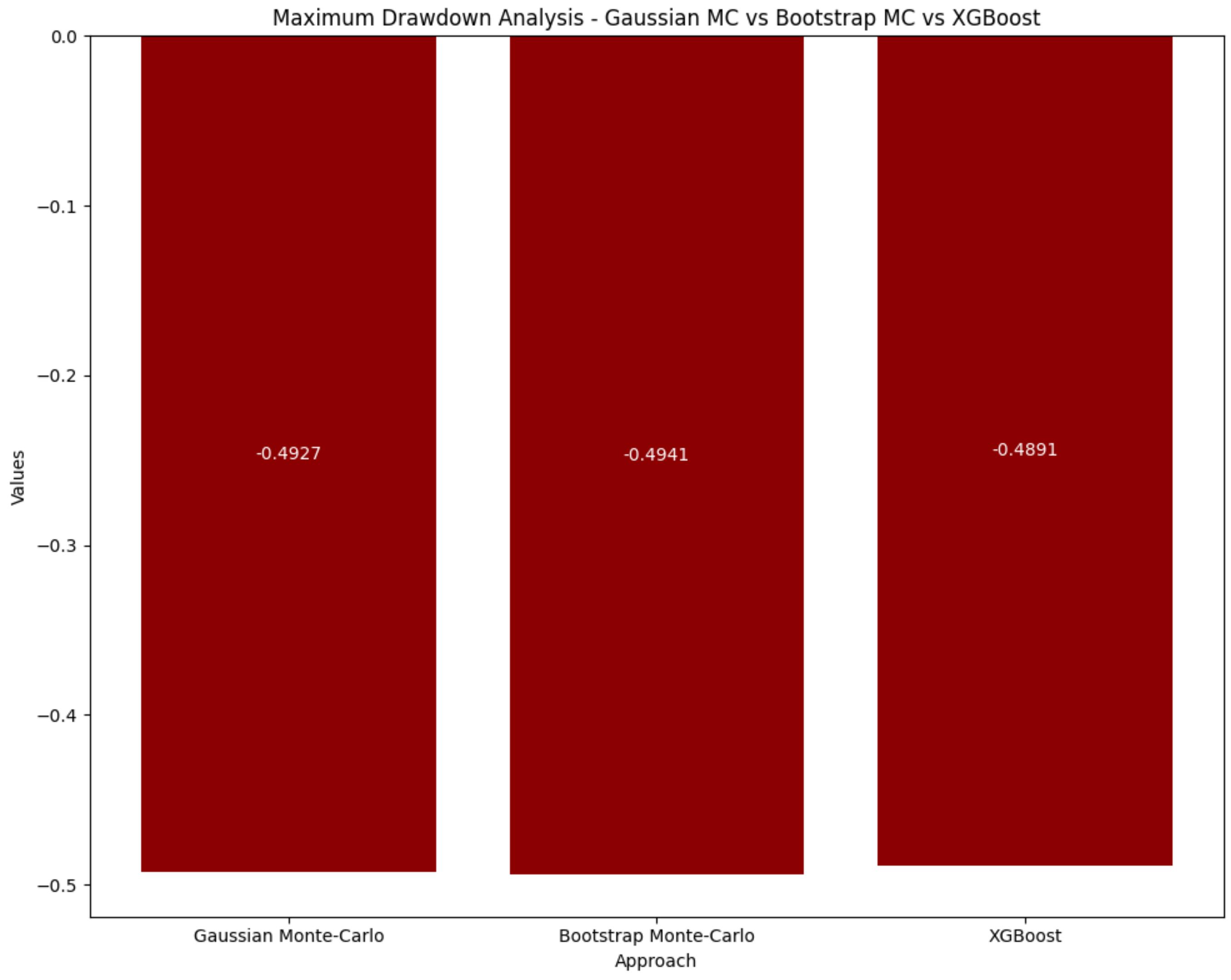
The results (across all three approaches) indicate that, across all three approaches, tail losses manifest as isolated monthly events rather than persistent multi-month drawdown episodes. In other words, although tail losses do occur, none of the models exhibit meaningful clustering of extreme downside outcomes over consecutive months within the out-of-sample period.

From a risk management perspective, this is an important finding. Persistent tail episodes—where losses remain in the extreme tail for multiple consecutive months—are typically associated with prolonged capital impairment, liquidity stress, and elevated investor drawdown risk. An ESD of 1.0 suggests that, once a tail loss is realized, the portfolio recovers above the tail threshold in the following month, limiting the duration of stress exposure. This behavior is consistent with effective rebalancing and risk-reset mechanisms embedded within each approach.

The uniformity of ESD across models also implies that differences in downside risk across approaches are driven primarily by tail loss frequency and magnitude, rather than persistence. This aligns closely with the Tail Loss Frequency findings, where Bootstrap Monte-Carlo demonstrated fewer tail events overall, and XGBoost exhibited more frequent tail breaches. However, once a tail event occurs, none of the approaches display a tendency to remain trapped in extended tail-loss regimes.

It is also important to interpret this result in the context of the rolling, adaptive nature of the tail threshold used in this study. Because the 5th percentile threshold is recalibrated over a rolling 36-month window, prolonged stress periods are partially absorbed into the reference distribution, reducing the likelihood of repeated consecutive breaches. As a result, ESD acts here as a diagnostic confirming the absence of tail clustering rather than a primary discriminator between models.

In summary, the Expected Shortfall Duration analysis suggests that all three portfolio construction approaches are effective at limiting the persistence of extreme downside risk. While Bootstrap Monte-Carlo remains superior in terms of reducing the frequency of tail losses, and XGBoost exhibits higher tail recurrence, none of the models expose the portfolio to prolonged tail-loss episodes. Consequently, model differentiation in this study arises from how often and how severely tail events occur, rather than from extended drawdown persistence—an insight that reinforces the complementary roles of TLF, ESD, and maximum drawdown metrics in a comprehensive tail-risk evaluation framework.



Analysis Conclusion - Maximum Drawdown

Maximum Drawdown (MDD) measures the worst peak-to-trough loss experienced by the portfolio over the out-of-sample period. Unlike tail loss frequency, which focuses on how often extreme losses occur, MDD captures the maximum cumulative damage caused by adverse return sequences. As a path-dependent metric, it reflects not only the severity of individual losses but also the clustering and persistence of downside shocks over time.

The observed maximum drawdowns across the three portfolio construction approaches are as follows: 1. **XGBoost** records the lowest maximum drawdown, at approximately -48.9%. 2. **Gaussian Monte-Carlo** follows closely, with a maximum drawdown of -49.3%. 3. **Bootstrap Monte-Carlo** exhibits the deepest drawdown, at roughly -49.4%.

While the absolute differences across approaches are modest, the ranking is economically meaningful. XGBoost's marginally lower drawdown indicates that, despite experiencing tail losses more frequently (as evidenced by its higher TLF), these losses tend to be less damaging in aggregate, preventing the portfolio from reaching as deep a cumulative trough. In contrast, Bootstrap Monte-Carlo, while effective at reducing the frequency of tail events, is more vulnerable to prolonged or clustered stress episodes, which compound over time into a slightly deeper peak-to-trough loss.

From a portfolio construction perspective, this highlights an important distinction between tail frequency control and drawdown containment. Bootstrap Monte-Carlo emphasizes distributional robustness and tail avoidance at the monthly level, but once adverse conditions persist, its slower re-risking behavior can delay recovery, allowing cumulative losses to deepen. Gaussian Monte-Carlo displays similar behavior, reflecting its reliance on fixed parametric assumptions that may under-adapt during extended stress regimes.

XGBoost's relatively lower MDD aligns with earlier regime-sensitivity findings. Its adaptive, feature-driven allocation framework allows it to adjust exposures dynamically as conditions evolve, limiting the total depth of cumulative losses even if it re-enters risk more frequently. This suggests that XGBoost manages drawdown severity more effectively than it manages tail event recurrence, reinforcing the view that it prioritizes responsiveness over conservatism.

In the broader context of this study, the MDD results complement the tail loss frequency and expected shortfall duration analyses. Bootstrap Monte-Carlo excels at minimizing repeated tail exposures, while XGBoost performs better at limiting the maximum capital impairment once stress unfolds. Consequently, the choice between approaches depends on the investor's primary risk objective: minimizing the occurrence of extreme losses versus constraining the worst-case cumulative drawdown. For strictly risk-aware portfolio construction with an emphasis on tail avoidance, Bootstrap Monte-Carlo remains preferable, whereas for investors more concerned with containing peak-to-trough losses and enabling faster recovery, XGBoost offers a modest advantage.

9.3. Portfolio Stability Analysis (Gaussian Monte-Carlo v/s Bootstrap Monte-Carlo v/s XGBoost)

This sub-section evaluates the stability and consistency of portfolio decisions generated by each portfolio construction approach over time. While the preceding analyses focus on return efficiency, downside protection, and drawdown behavior, the emphasis here is on the temporal robustness and implementability of the resulting asset allocations.

The objective is to assess whether observed risk-adjusted and tail-risk outcomes are achieved through stable, regime-consistent portfolio adjustments or through frequent and potentially noisy reallocation decisions that may undermine real-world applicability.

The analysis employs a single out-of-sample metric:

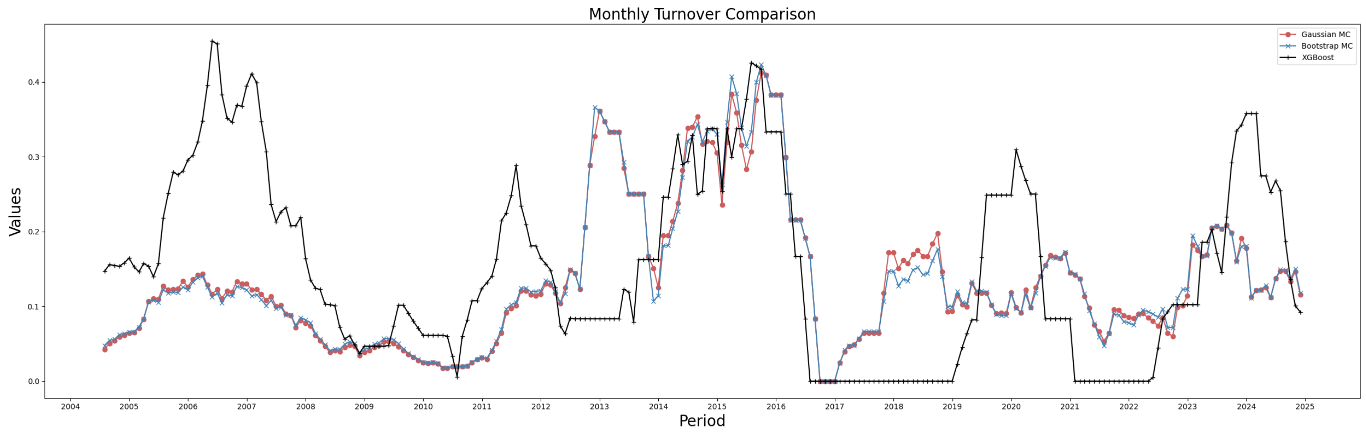
1. Monthly Turnover – to quantify the magnitude of portfolio reallocation between consecutive monthly rebalancing periods, capturing the stability of allocation decisions implied by each modelling framework

Monthly turnover measures the total absolute change in asset weights from one rebalancing period to the next and serves as a proxy for the consistency of the model's risk assessment over time. Lower turnover indicates smoother and more persistent portfolio allocations, suggesting stable risk estimates and controlled responses to evolving market conditions. Conversely, higher turnover may reflect sensitivity to short-term fluctuations, estimation noise, or frequent reassessment of risk regimes.

In the context of risk-aware portfolio construction, turnover does not represent a cost-efficiency metric but rather a diagnostic of decision stability. A model that frequently reshuffles portfolio weights may still achieve favorable risk-adjusted or tail-risk outcomes, yet such performance may rely on fragile or over-reactive signals that are difficult to sustain outside a backtesting environment.

Evaluating turnover alongside risk-adjusted performance and tail-risk metrics enables a more holistic assessment of each approach, distinguishing between models that achieve risk resilience through measured structural adaptation and those that rely on frequent tactical repositioning.

Monthly turnover is computed consistently across all approaches using out-of-sample portfolio weights generated at each monthly rebalance, ensuring comparability of stability characteristics under identical evaluation conditions.



Analysis Conclusion - Monthly Turnover

Monthly turnover evaluates the stability and consistency of portfolio allocation decisions over time by measuring the extent of rebalancing required between consecutive periods. Unlike risk-adjusted performance metrics, which focus on outcome efficiency, or tail-risk measures, which emphasize loss severity and persistence, turnover captures the behavioral and structural stability of the underlying portfolio construction process. As such, it provides insight into whether observed risk outcomes are achieved through smooth regime adaptation or frequent tactical repositioning.

To improve interpretability and focus on persistent allocation behavior rather than month-to-month noise, portfolio turnover is analyzed using a 12-month rolling average of half-turnover, allowing for clearer identification of sustained differences in allocation stability across modelling approaches.

The observed turnover dynamics across the three approaches reveal a clear and consistent ranking: 1. **XGBoost** exhibits the highest and most variable turnover across the sample, with pronounced and persistent spikes during periods of heightened market uncertainty. 2. **Bootstrap Monte-Carlo** demonstrates moderate turnover, closely tracking Gaussian Monte-Carlo but with slightly higher responsiveness during regime transitions. 3. **Gaussian Monte-Carlo** consistently displays the lowest and smoothest turnover profile, indicating the most stable allocation behavior over time.

The elevated and episodic turnover observed for XGBoost reflects its adaptive, feature-driven framework, which frequently reassesses risk conditions and reallocates capital accordingly. While this responsiveness allows XGBoost to adjust exposures rapidly as regimes evolve, it also results in sustained periods of heightened rebalancing intensity, particularly around major structural transitions. This behavior is consistent with earlier findings showing higher tail-loss frequency but improved drawdown containment, suggesting that XGBoost prioritizes continuous adjustment over allocation persistence.

In contrast, both Monte-Carlo–based approaches exhibit substantially smoother turnover profiles, indicating greater stability in portfolio construction decisions. Gaussian Monte-Carlo, in particular, benefits from fixed parametric assumptions that dampen allocation volatility, resulting in gradual and measured rebalancing. Bootstrap Monte-Carlo, while sharing this overall stability, displays marginally higher turnover during stressed periods, reflecting its sensitivity to empirical distributional shifts without fully abandoning allocation persistence.

From a portfolio construction perspective, these results highlight an important trade-off between adaptability and stability. XGBoost's higher turnover underscores its strength in responding quickly to evolving risk environments but raises concerns around potential over-reactivity and reduced implementability in real-world settings. The Monte-Carlo approaches, especially Gaussian Monte-Carlo, achieve risk resilience through more stable and predictable allocation paths, aligning closely with objectives centered on robustness and behavioral consistency.

In the broader context of this study, the turnover analysis complements the risk-adjusted performance and tail-risk findings. Bootstrap Monte-Carlo emerges as a balanced approach, offering improved tail control relative to Gaussian Monte-Carlo while maintaining moderate allocation stability. XGBoost, meanwhile, demonstrates superior adaptability at the cost of higher decision instability. Consequently, for risk-aware portfolio construction where stability and robustness are prioritized alongside tail-risk mitigation, the Monte-Carlo frameworks—particularly Bootstrap Monte-Carlo—remain more suitable, whereas XGBoost may be preferable in settings where rapid regime adaptation is valued despite higher rebalancing intensity.

Section 10: Conclusion

1. Gaussian Monte-Carlo: strengths, weaknesses, and role in this study

Strengths

- **Stability and smooth allocation behavior**
  Gaussian Monte-Carlo produces the smoothest allocation paths and the lowest portfolio turnover across the evaluation horizon. This reflects high allocation persistence and strong implementation stability, attributes that are valuable from a governance and execution perspective.
- **Predictable and interpretable risk dynamics**
  The model responds to changes in market conditions in a gradual and averaged manner, making its behavior easy to interpret and suitable as a baseline risk framework.
- **Competitive regime-conditioned performance**
  When evaluated within explicitly defined calm and stress regimes, Gaussian Monte-Carlo exhibits performance broadly comparable to Bootstrap Monte-Carlo, reinforcing its value as a stable benchmark.

Weaknesses

- **Distributional misspecification risk**
  Given the empirically observed non-Gaussian nature of returns, the Gaussian assumption exposes the model to systematic understatement of tail risk, particularly during extreme market conditions.
- **Delayed recognition of regime shifts**
  The same smoothing properties that improve stability also limit responsiveness around regime transitions, where risk often changes abruptly rather than continuously.

Summary
Gaussian Monte-Carlo serves as a robust and interpretable benchmark with strong stability properties, but its reliance on restrictive distributional assumptions limits its suitability as the primary risk model when empirical data rejects normality.

2. Bootstrap Monte-Carlo: strengths, weaknesses, and alignment with risk-adjusted portfolio construction

Strengths

- **Superior tail-risk control in frequency terms**
  Bootstrap Monte-Carlo exhibits the lowest tail loss frequency among the three approaches, indicating fewer realized breaches of extreme downside thresholds. This directly supports improved risk-adjusted performance by reducing repeated tail events that mechanically degrade Sharpe ratios.
- **Best balance of Sharpe and CVaR across regimes**
  Across calm and stress regimes, Bootstrap Monte-Carlo delivers a marginally higher Sharpe ratio than Gaussian Monte-Carlo without worsening Conditional Value-at-Risk. This indicates improved efficiency without compensating through additional downside exposure.
- **Empirical distribution fidelity with controlled adaptiveness**
  The bootstrap framework captures fat tails and skewness present in the data while maintaining smoother allocation dynamics than machine-learning-based approaches. This places it in an effective middle ground between realism and implementability.

- **Consistent performance across diagnostics**
  The model demonstrates coherent behavior across volatility analysis, regime-conditioned risk-adjusted metrics, tail-risk diagnostics, and turnover assessments, reinforcing its robustness as a portfolio construction engine.

Weaknesses

- **Slightly deeper maximum drawdown**
  Bootstrap Monte-Carlo exhibits a marginally larger maximum drawdown than the other approaches. While the difference is modest, it reflects exposure to path-dependent drawdown accumulation during prolonged stress periods.
- **Lack of explicit regime classification**
  The model adapts implicitly through rolling estimation rather than explicitly identifying or classifying market regimes.

Summary
Bootstrap Monte-Carlo offers the most defensible balance between statistical realism, downside protection, and portfolio stability, making it the most appropriate primary approach for risk-adjusted portfolio construction in this study.

3. XGBoost: strengths, weaknesses, and its role as a regime-aware risk engine

Strengths

- **Strong regime sensitivity and risk recognition**
  XGBoost responds sharply and discontinuously around major market stress events, effectively capturing regime transitions and shifts in risk dynamics.
- **High modelling flexibility and control**
  The framework allows explicit control over feature engineering, hyperparameters, and functional form, enabling targeted modelling of nonlinear interactions, volatility clustering, and regime persistence.
- **Improved drawdown containment**
  XGBoost achieves the shallowest maximum drawdown among the three approaches, indicating effective capital preservation during severe market declines.

Weaknesses

- **Inferior risk-adjusted performance metrics**
  Despite its regime awareness, XGBoost delivers materially lower Sharpe ratios and more severe CVaR outcomes across both calm and stress regimes, indicating that risk recognition does not automatically translate into improved risk-adjusted returns.
- **Highest tail loss frequency**
  The model experiences tail events more frequently, consistent with a more active and reactive risk posture.
- **Elevated turnover and implementation costs**
  XGBoost produces the highest and most volatile turnover profile, particularly during uncertain market phases, raising concerns around transaction costs and operational robustness.

Summary
XGBoost functions best as a regime identification and risk signalling framework rather than as a standalone portfolio construction engine within this setup.

4. Final synthesis and recommendation

Across volatility diagnostics, regime-conditioned risk-adjusted performance, tail-risk behavior, drawdown containment, and portfolio stability, Bootstrap Monte-Carlo emerges as the approach most closely aligned with the objective of risk-adjusted portfolio construction. Its empirical distribution consistency, lowest tail loss frequency, balanced Sharpe–CVaR trade-off, and moderate turnover collectively position it as the most robust primary risk engine. Gaussian Monte-Carlo remains a valuable benchmark characterized by stability and interpretability but is constrained by distributional assumptions. XGBoost, while offering superior regime sensitivity and drawdown containment, incurs meaningful costs in Sharpe, CVaR, tail frequency, and turnover, making it better suited as a complementary regime-aware overlay rather than the core portfolio construction mechanism.