Monte-Carlo Simulation for Portfolio Risk Analysis

Introduction

Financial markets are inherently uncertain, and future returns cannot be forecasted with certainty. Traditional risk metrics, such as volatility or beta, summarize risk using historical averages but often fail to capture the **full distribution of potential outcomes** or the **asymmetry commonly observed in real-world returns**.

Monte Carlo simulation overcomes these limitations by:

- Modelling asset returns as random variables drawn from a specified distribution (e.g., multivariate normal or Student-t)
- Preserving empirical volatility and correlation structures across assets
- Generating a wide range of potential future scenarios rather than relying on a single forecast

This probabilistic framework is particularly valuable for **portfolio risk analysis**, as it enables us to:

- Capture cross-asset dependencies and interactions
- Assess tail risk and extreme loss scenarios
- Estimate the likelihood of breaching specific loss thresholds
- Stress-test portfolios under adverse volatility and correlation regimes

Motivation of project

The primary goal of this project is to **demonstrate probabilistic thinking in risk management** and develop a practical understanding of **tail risk, dependency structures, and scenario-based analysis**. Through this exercise, we aim to:

- Translate statistical assumptions about returns into **actionable portfolio risk insights**
- Quantify potential losses under a variety of simulated scenarios
- Illustrate the importance of stress-testing portfolios beyond simple historical averages

Ultimately, this project emphasizes the value of **scenario analysis** and **risk-aware decision-making** in portfolio management.

Applying Monte-Carlo Simulation to Portfolio Risk Management

In practice, a portfolio manager constructs a portfolio by selecting assets in a specific composition and implements a strategy based on this allocation. Typically, the strategy is **backtested** on historical data and **validated** on out-of-sample test data to evaluate portfolio performance over time. Through this process, the manager can assess metrics such as **Alpha, Beta, and returns**, gaining insight into how the portfolio might perform on unseen data—essentially, a proxy for future performance.

However, traditional backtesting falls short in addressing the **uncertainty of potential losses**. A portfolio is exposed to both **systematic and unsystematic risks**, and backtesting on a single historical path does not provide a complete picture of possible adverse outcomes. Questions such as “how severe could drawdowns be?” or “what is the probability of large losses over the holding period?” remain unanswered.

This is where **Monte Carlo simulation** becomes invaluable. Instead of evaluating the portfolio on a single historical path for each asset, Monte Carlo simulates **multiple potential price paths** using the historical covariance and volatility of assets. In the real world, we only see one path for any given asset… but if we were to consider the fact that there are other worlds in the whole universe, then it makes sense that maybe, in the other worlds, the same asset may exhibit a different price path! By running thousands of simulated scenarios, we can observe the range of potential portfolio outcomes and estimate **probabilistic measures of loss**, such as **Value at Risk (VaR)** and **Conditional Value at Risk (CVaR)**. This provides the portfolio manager with a **quantitative sense of risk**, allowing for informed decisions about potential losses and tail events.

In summary, while quantitative and systematic strategies aim to **maximize returns and Sharpe ratios**, Monte Carlo simulation allows managers to **quantify the underlying risk** of these strategies, providing a forward-looking, probabilistic framework to assess potential losses and portfolio behavior under uncertainty.

What is Monte Carlo Simulation

Monte Carlo simulation is a **probabilistic technique used to model uncertainty and quantify risk**. It evaluates the behaviour of a system or portfolio by replacing uncertain factors with **random variables drawn from specified probability distributions**, and then repeatedly simulating outcomes to estimate their likelihood.

In the context of **portfolio risk analysis**, Monte Carlo simulation is used to model **possible portfolio values** based on the **volatility and correlation of underlying assets**. By simulating thousands (or even millions) of potential outcomes, we can **quantify the impact of risk and uncertainty** on expected portfolio performance. Mathematically, for a portfolio with n assets, the portfolio value V_t at time t can be approximated as:

$$V_t = \sum_{i=1}^{n} w_i S_i(t)$$

where:

- w_i = weight of asset i in the portfolio
- $S_i(t)$ = simulated value of asset i at time t

Each $S_i(t)$ is modelled as a **random variable**, often using a **geometric Brownian motion**:

$$S_i(t+\Delta t) = S_i(t) \cdot \exp\left(\left(\mu_i - \frac{1}{2}\sigma_i^2\right)\Delta t + \sigma_i \sqrt{\Delta t}\, Z_i\right)$$

where:

- μ_i = expected return of asset i
- σ_i = volatility of asset i
- $Z_i \sim N(0,1)$ = standard normal random variable

After simulating many paths, **Value at Risk (VaR)** and **Conditional Value at Risk (CVaR)** can be computed to summarize potential losses in the tail of the distribution.

Covariance and its relation to Monte Carlo Simulation

Covariance measures how two random variables move together. For two asset returns R_A and R_B, covariance is defined as:

$$\text{Cov}(R_A, R_B) = \mathbb{E}[(R_A - \mu_A)(R_B - \mu_B)]$$

where μ_A and μ_B are the expected returns of the assets.

- **Positive covariance**: Returns of the two assets move in the same direction
- **Negative covariance**: Returns move in opposite directions

In Monte Carlo simulations, covariance is **essential for modeling realistic joint movements of assets**, ensuring that simulated paths reflect **empirical correlations**. This is critical in **portfolio risk management**, because:

- Covariance-informed diversification can **reduce overall portfolio risk** by combining assets that do not move perfectly together.
- Ignoring covariance can **underestimate tail risk**, as simultaneous adverse movements are more likely than independent movements.

In short, we can summarise the Monte-Carlo Simulation to a series of 5-steps as outlined below:

- **Define uncertainties**: Identify random variables (e.g., asset returns) and assign probability distributions.

- **Generate random scenarios**: Draw random samples from the distributions to simulate asset values.
- **Compute portfolio value**: Calculate portfolio outcomes for each scenario.
- **Repeat simulations**: Run the process thousands of times to build the **empirical distribution of portfolio outcomes**
- **Estimate risk metrics**: Extract metrics such as **VaR** and **CVaR** from the simulated distribution to assess tail risk.

Dataset and Duration of Analysis

For our Monte Carlo simulation, the **portfolio is composed of the following stock tickers**:

- 'BNS.TO', 'GOOGL', 'XOM', 'NIO', 'KO', 'PEP', 'ARCC', 'IBM', 'AGNC', 'LCID'

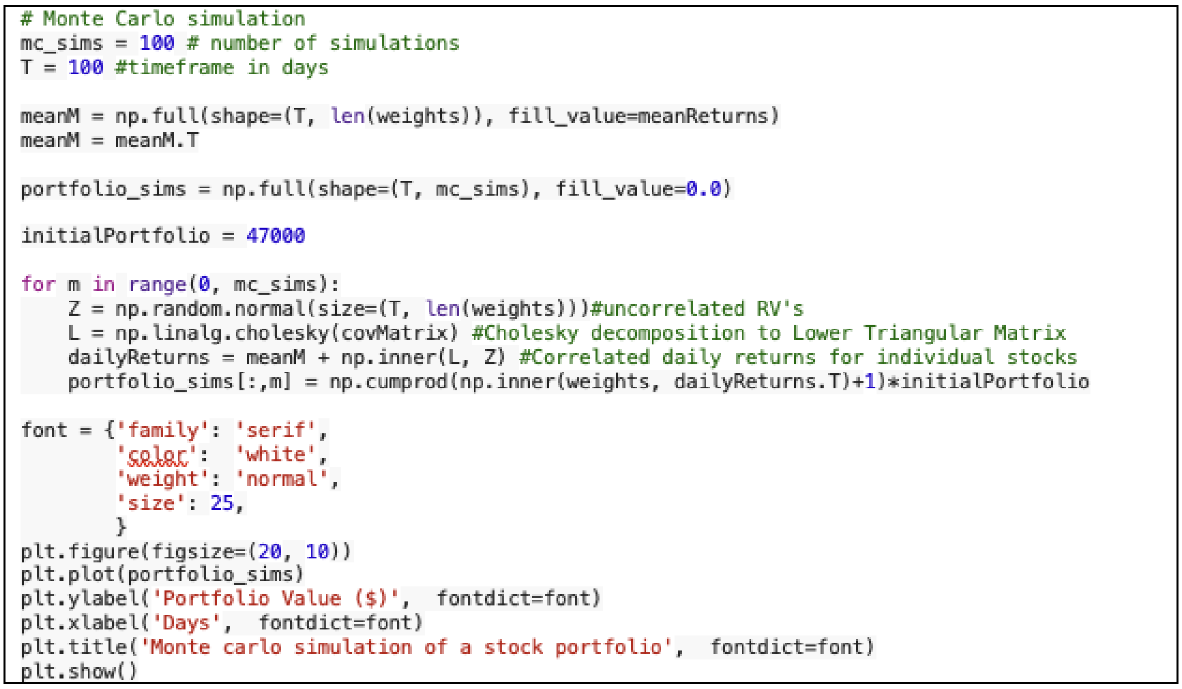
These historical price series can be retrieved from **Yahoo Finance**. A well-constructed portfolio typically includes a **diverse mix of asset classes**, such as stocks, bonds, commodities, futures, and derivatives. The **allocation** across these assets is determined by a fund manager, who follows a strict investment mandate aligned with the firm's objectives and risk tolerance.

For this analysis, we focus on the **equity portion** represented by the selected tickers. The **risk assessment** will be based on the **252 trading-day performance** of these stocks, corresponding to a one-year horizon in standard financial practice.

For this analysis, we will assume an initial portfolio investment value of **$1,000,000**. Given that we need to derive our covariance matrix, we will need to retrieve the historical prices of our assets and for that, we will be using a period from **2022 to 2024**. A medium period of 2-3 years provides a **balance** between statistical stability and relevance, as we do not underestimate / overestimate market noises or include outdated regimes that are non-representative of current market correlations and volatilities.

Monte-Carlo Simulation

Once we import the data from YahooFinance, compute the relevant metrics like assets' mean, standard deviation and covariances, we define our Monte-Carlo simulation function as follow:



```
# Monte Carlo simulation
mc_sims = 100 # number of simulations
T = 100 #timeframe in days

meanM = np.full(shape=(T, len(weights)), fill_value=meanReturns)
meanM = meanM.T

portfolio_sims = np.full(shape=(T, mc_sims), fill_value=0.0)

initialPortfolio = 47000

for m in range(0, mc_sims):
    Z = np.random.normal(size=(T, len(weights)))#uncorrelated RV's
    L = np.linalg.cholesky(covMatrix) #Cholesky decomposition to Lower Triangular Matrix
    dailyReturns = meanM + np.inner(L, Z) #Correlated daily returns for individual stocks
    portfolio_sims[:,m] = np.cumprod(np.inner(weights, dailyReturns.T)+1)*initialPortfolio

font = {'family': 'serif',
        'color':  'white',
        'weight': 'normal',
        'size': 25,
        }
plt.figure(figsize=(20, 10))
plt.plot(portfolio_sims)
plt.ylabel('Portfolio Value ($)',  fontdict=font)
plt.xlabel('Days',  fontdict=font)
plt.title('Monte carlo simulation of a stock portfolio',  fontdict=font)
plt.show()
```

Let us walk through the steps taken in the code above.

We begin with initialising the number of independent MC paths we want to simulate. Each path represents a *possible future evolution* of the portfolio. For each simulation, we define the duration or time horizon (measured in trading days).

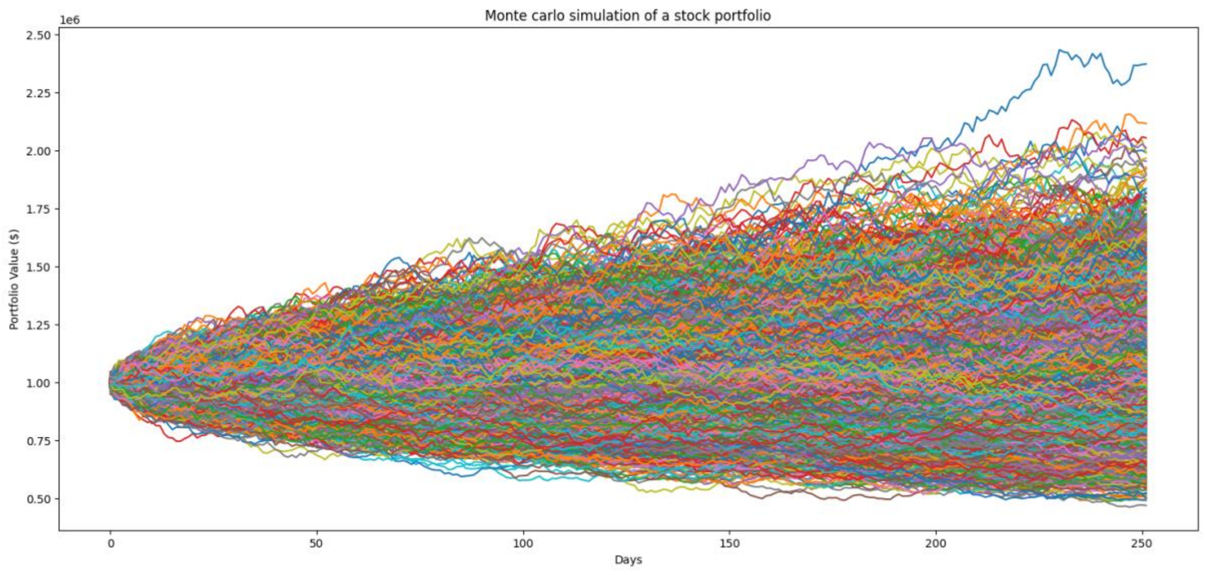
- `meanReturns` is assumed to be a vector of **expected daily returns** for each asset.
- `np.full(...)` replicates the mean returns across all `T` days.
- The shape before transposing is `(T, number_of_assets)` `.T` transposes it to `(number_of_assets, T)` so dimensions align later. Conceptually, this assumes **stationary expected returns** across the simulation horizon. Each asset has the same expected return every day.

Next, we initialize our portfolio simulation to be `np.full(shape=(T, mc_sims), fill_value=0.0)`. This pre-allocates a matrix to store portfolio values, where rows → time (`T` days) and columns → simulation number (`mc_sims`).

Finally, we iterate through our simulation. For each simulation, we start by generating **random shocks** Z from independent standard normal variables, representing uncorrelated daily market movements for each asset. These shocks are the raw inputs before introducing realistic correlations between assets.

To incorporate **empirical correlations**, we apply the **Cholesky decomposition** to the covariance matrix of asset returns, factorizing it as $\Sigma = LL^T$, where L is a lower triangular matrix. Multiplying the random shocks Z by Ltransforms them into **correlated shocks**, preserving the historical volatility and cross-asset correlations. Adding the expected returns vector shifts these simulated returns from zero-mean to expected-return-adjusted values.

Finally, we compute the **portfolio value path** by calculating daily portfolio returns as the weighted sum of asset returns, converting them to gross returns, and scaling by the initial portfolio value. Repeating this process across all simulation runs produces a matrix of portfolio value trajectories, each representing a possible path the portfolio could follow under the simulated market conditions. This produces the chart below.



Where each line represents the outcome of one simulated portfolio path based on the randomized price action of the underlying assets.

From our simulation, we see that as we project our timeline further into the future, the **dispersion** of portfolio value paths increases markedly. This widening spread reflects the compounding nature of uncertainty: small variations in returns at earlier periods are amplified over time, leading to increasingly divergent outcomes across simulation paths.

One interesting observation from the simulation is that we can roughly define upper and lower bounds for the portfolio's potential value at each future horizon. These bounds are not deterministic limits, but **probabilistic envelopes** derived from the distribution of simulated outcomes (for example, percentile bands). The upper bound represents optimistic but plausible scenarios driven by sustained positive return realizations, while the lower bound captures adverse scenarios in which negative shocks and drawdowns compound over time.

As the time horizon extends, these bounds move further apart, highlighting the trade-off between return potential and risk. While longer investment horizons increase the probability of achieving higher absolute returns, they also expose the portfolio to a wider range of unfavorable outcomes. In the area of risk management, we are more concerned about the downside risks rather than the upside risks, and this is where we will study the losses exhibited by our simulated portfolios.

Risk Metrics and Analysis

Once the Monte Carlo simulation generates a large number of potential portfolio value paths, we can quantify **portfolio risk** using a set of statistical measures. The most commonly used metrics include **Value-at-Risk (VaR)**, **Conditional Value-at-Risk (CVaR)**, **maximum drawdowns**, and **loss distributions**.

Value-at-Risk (VaR)

Value-at-Risk at a confidence level α (e.g., 95% or 99%) represents the maximum loss not exceeded with probability α over a given horizon. Mathematically, if L denotes portfolio losses:

$$\mathrm{VaR}_\alpha = \inf\{l \in \mathbb{R} : P(L \le l) \ge \alpha\}$$

In practice, for simulated losses $L_1, L_2, \dots, L_N$, VaR can be estimated as the α-percentile of the loss distribution:

$$\mathrm{VaR}_\alpha \approx \mathrm{Percentile}_\alpha(L_1, L_2, \dots, L_N)$$

This metric provides a threshold for worst-case losses under normal market conditions.

Conditional Value-at-Risk (CVaR)

While VaR indicates the threshold, it does not capture the magnitude of losses beyond that threshold. **Conditional Value-at-Risk**, also known as **Expected Shortfall**, is defined as the expected loss given that the loss exceeds VaR:

$$\mathrm{CVaR}_\alpha = \mathbb{E}[L \mid L \ge \mathrm{VaR}_\alpha]$$

CVaR is particularly useful in understanding tail risk, highlighting how severe losses could be in extreme market scenarios.

Maximum Drawdown and Loss Distribution

Maximum drawdown (MDD) measures the largest observed decline from a portfolio's peak to trough during a given period:

$$\mathrm{MDD} = \max_{t \in [0,T]} \frac{Peak_t - V_t}{Peak_t}$$

where V_t is the portfolio value at time t.

Loss distribution provides a probabilistic view of potential losses, showing not only the average or extreme losses but also the frequency and severity of losses across all simulated scenarios.

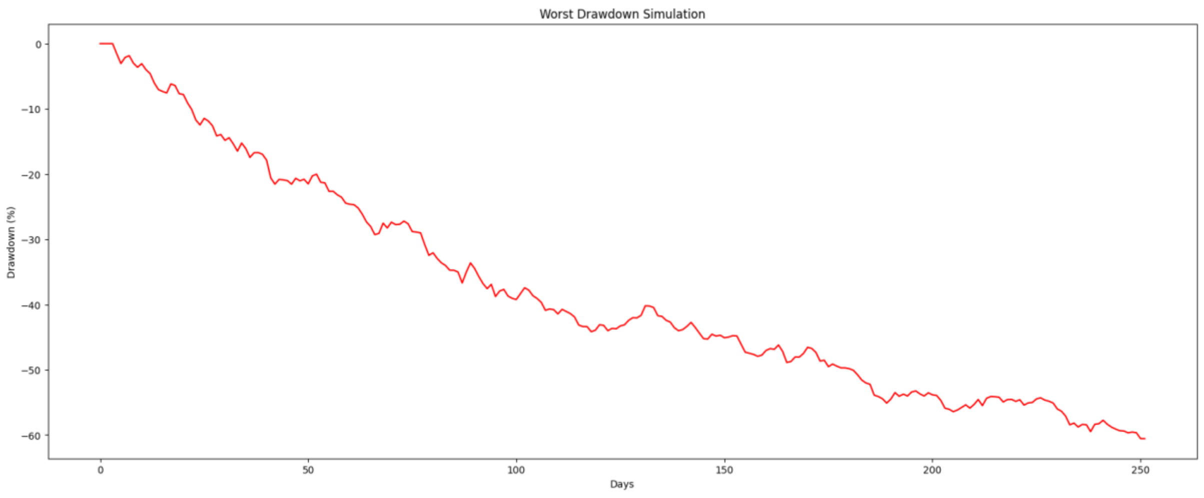
Computing the VaR and cVaR of one complete Monte-Carlo Simulation, we got the following values for a portfolio with initial investment of $1,000,000, **simulated 10,000 times over 252 trading days**.

	Value
VaR at 95% Confidence Interval	$296,981.38
CVaR at 95% Confidence Interval	$355,329.25

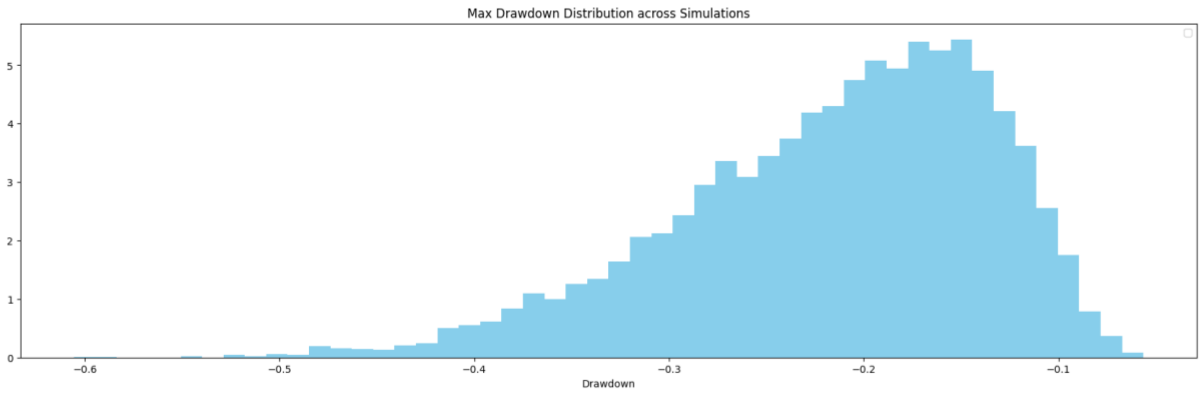
From there, we can generate the following report:

*Given an initial portfolio value of $1,000,000, the 95% VaR indicates that, over a horizon of **252** trading days, we are **95%** confident that the portfolio will not incur losses greater than **$296,981.38**. The 95% CVaR provides a more conservative risk measure by estimating the average loss conditional on outcomes worse than the 95% VaR threshold.*
*In other words, if the portfolio performance falls into the worst **5%** of simulated scenarios, the expected loss is approximately **$410,648.28**. Under such extreme scenarios, the portfolio value would be expected to decline to approximately: $1,000,000−$410,648.28=$589,351.72*

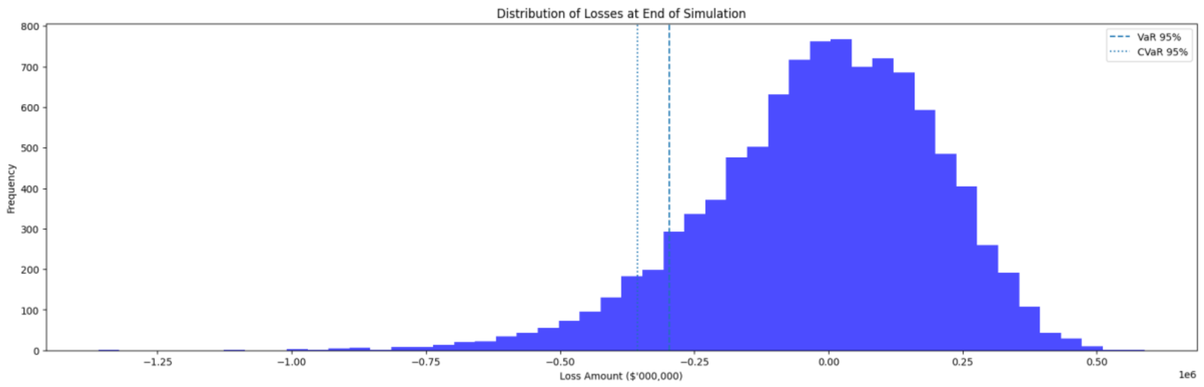
When we plot the drawdown of our worst simulation, we get the following chart:



This shows that in the worst-case scenario, our portfolio is trending downwards across the entire forecasted duration. As for all max-drawdown durations, we can visualize it by plotting the max drawdowns across all simulations into a histogram;



And finally, for all the simulated outcomes, we can visualize the distribution of losses with a histogram as follow:



Taking the outputs from the same Monte-Carlo Simulation, we get the following values

	Value
VaR at 95% Confidence Interval	$296,981.38
CVaR at 95% Confidence Interval	$355,329.25
Mean Loss	$13,896.64

Median Loss	$4,392.81

Once again, we can report the above information as follow:

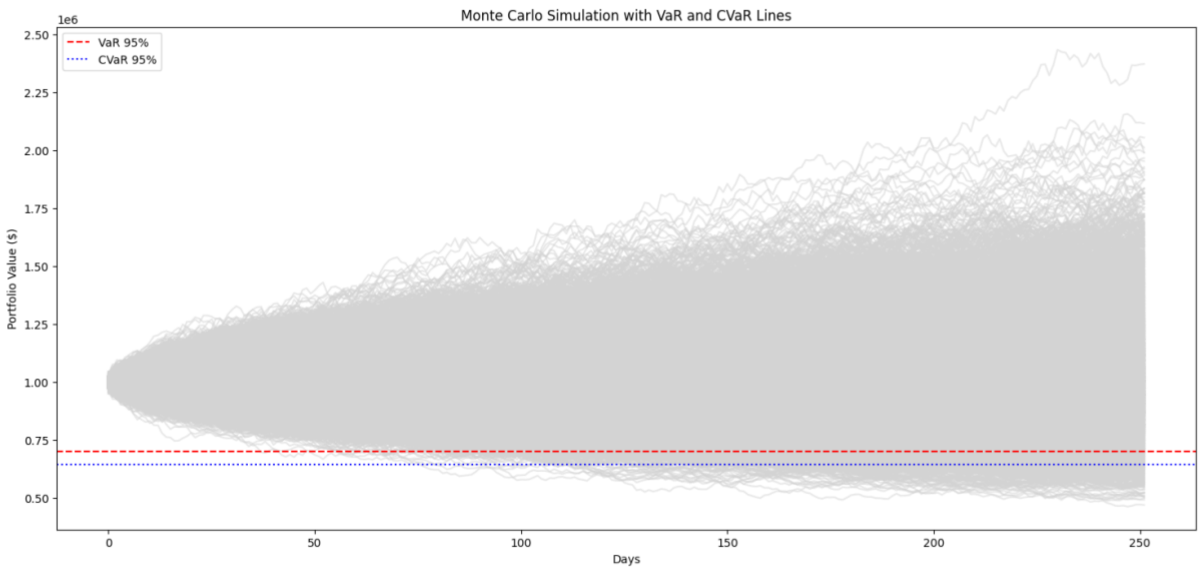
The 5% Value at Risk (VaR) of **$296,981.38** indicates that, over a one-tradable-year horizon, 95% of simulated outcomes resulted in losses less than approximately **29.7%** of the initial portfolio value of **$1,000,000**.

The 5% Conditional Value at Risk (CVaR) of **$355,329.25** represents the average loss conditional on losses exceeding the VaR threshold. This highlights the severity of tail risk: when losses are extreme, they tend to be materially larger than the VaR cutoff.

The gap between VaR and CVaR (**~$58,000**) suggests a fat-tailed loss distribution, where adverse outcomes deteriorate quickly beyond the 5% threshold.

The mean loss ($13,896.64) and median loss ($4,392.81) are both small relative to the portfolio size, implying that most simulated outcomes cluster around modest losses or near breakeven, with downside risk concentrated in relatively rare but severe events.

We can plot the two lines onto our Monte-Carlo visualization for a better representation of the thresholds.



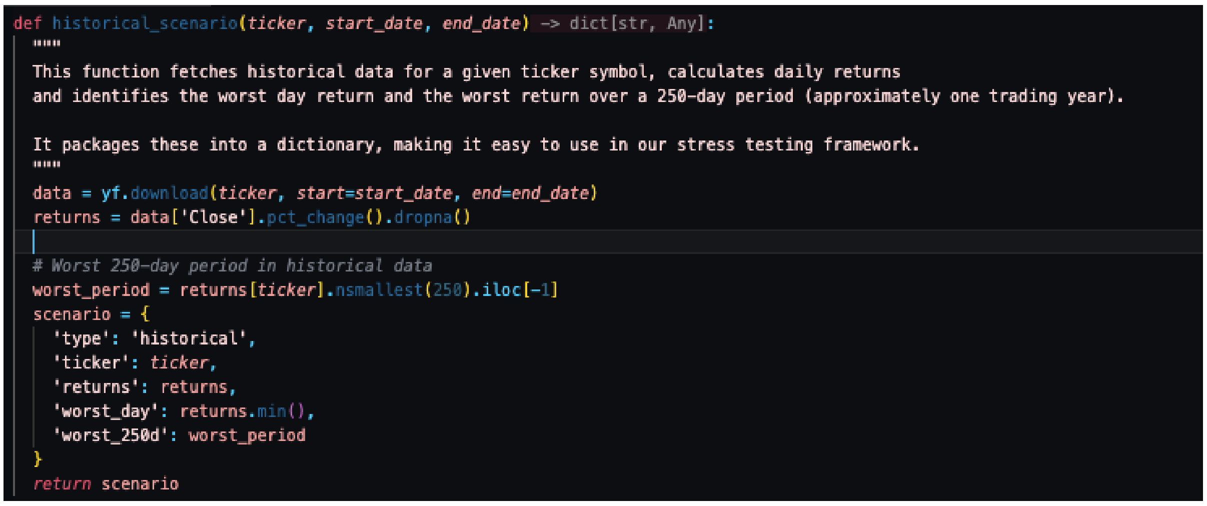
Stress Testing and Scenario Analysis

Stress testing provides a structured framework for assessing portfolio resilience under extreme but plausible market conditions. Unlike traditional risk measures such as **Value-at-Risk (VaR)**, which are designed to capture losses under typical market fluctuations, stress testing explicitly focuses on low-probability, high-impact events that may materially impair portfolio performance. By examining portfolio behaviour outside normal market regimes, stress testing helps identify vulnerabilities that may not be visible through standard risk metrics alone. This is a step further from just simulating random paths to quantify risks, as we are now deliberately introducing shocks to assess the resiliency of portfolios on top of the simulation.

In this next section, we will explore three complementary approaches: replicating historically observed crisis events, designing hypothetical "what-if" scenarios based on adverse market assumptions, and generating a broad distribution of outcomes through Monte Carlo simulations (which we have done above). Together, these methods provide multiple perspectives on tail risk and enhance our understanding of how portfolios may respond under severe market stress.

Historical scenarios

Historical scenario analysis involves replaying realised market shocks observed during past stress periods (e.g. financial crises, policy shocks, or sharp drawdowns) and applying them to the current portfolio. This approach preserves the empirical dependence structure, volatility clustering, and nonlinear dynamics present during real crises, providing insight into how the portfolio would have behaved under known adverse conditions.

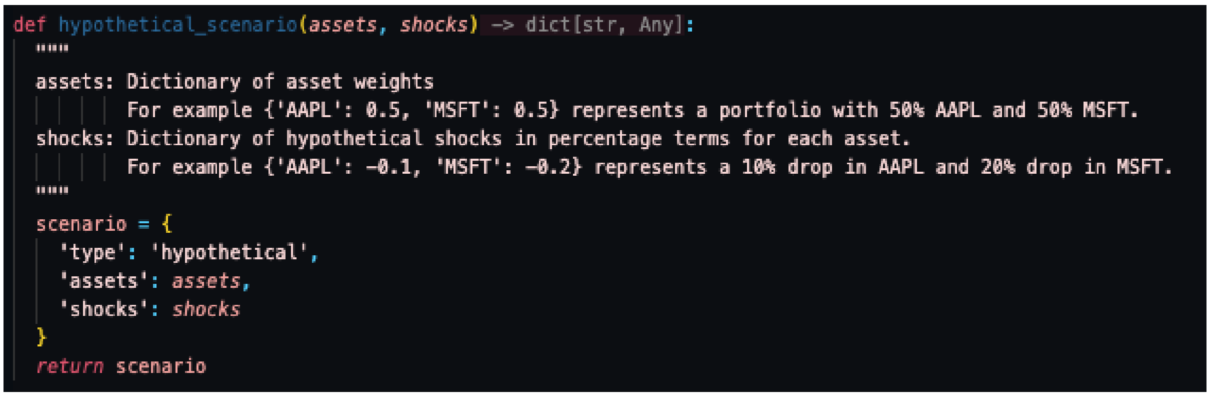


```python
def historical_scenario(ticker, start_date, end_date) -> dict[str, Any]:
    """
    This function fetches historical data for a given ticker symbol, calculates daily returns
    and identifies the worst day return and the worst return over a 250-day period (approximately one trading year).

    It packages these into a dictionary, making it easy to use in our stress testing framework.
    """
    data = yf.download(ticker, start=start_date, end=end_date)
    returns = data['Close'].pct_change().dropna()

    # Worst 250-day period in historical data
    worst_period = returns[ticker].nsmallest(250).iloc[-1]
    scenario = {
        'type': 'historical',
        'ticker': ticker,
        'returns': returns,
        'worst_day': returns.min(),
        'worst_250d': worst_period
    }
    return scenario
```

We defined the function for historical scenario as shown above. It fetches historical data for a given ticker symbol, calculates daily returns and identifies the worst day return and the worst return over a 250-day period (approximately one trading year). It packages these into a dictionary, making it easy to use in our stress testing framework.

Hypothetical scenarios

Hypothetical scenarios are forward-looking stress tests constructed by imposing user-defined shocks to selected assets or risk factors, such as interest rate shifts, equity sell-offs, or volatility spikes. These scenarios allow targeted “what-if” analysis, enabling the evaluation of portfolio sensitivity to specific risks that may not have occurred historically but are considered plausible given current market conditions.

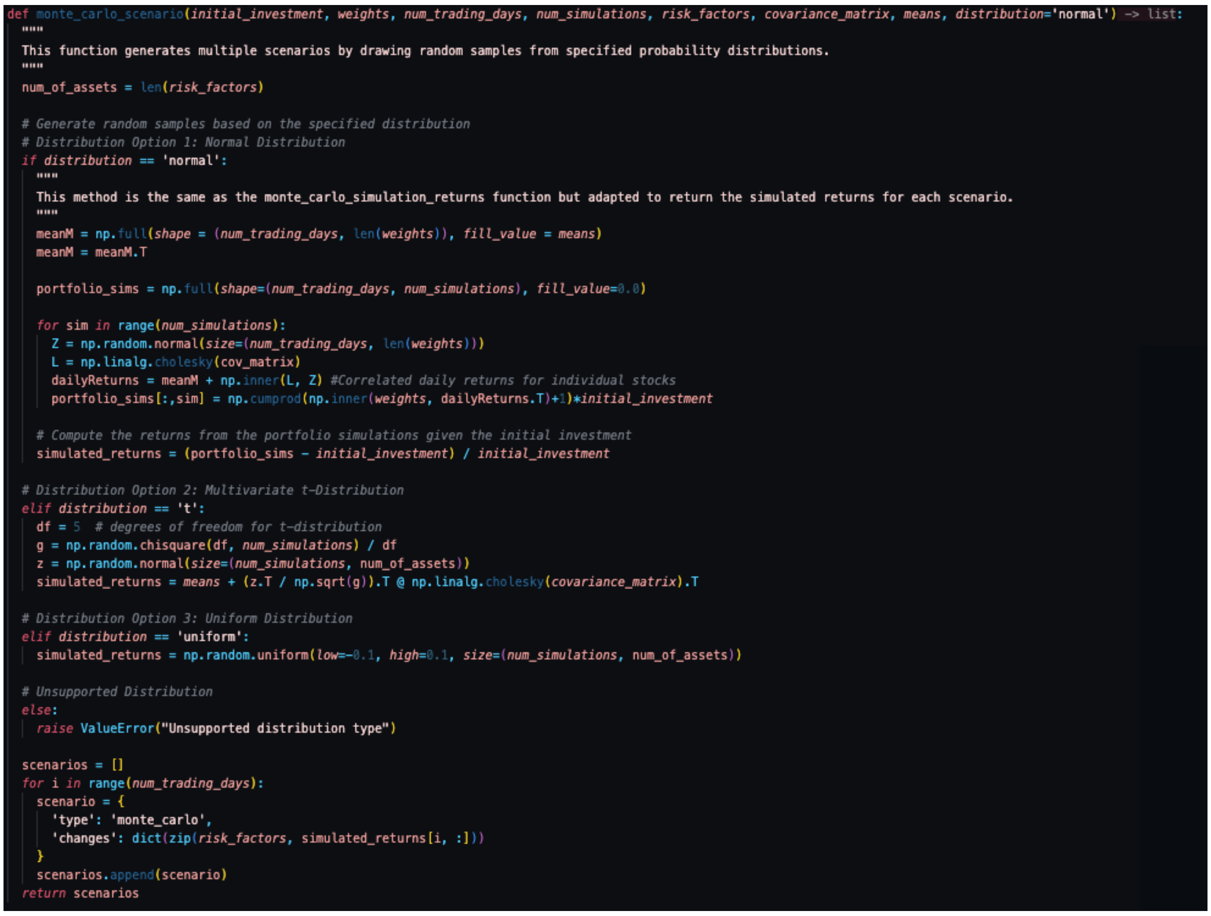


```python
def hypothetical_scenario(assets, shocks) -> dict[str, Any]:
    """
    assets: Dictionary of asset weights
            For example {'AAPL': 0.5, 'MSFT': 0.5} represents a portfolio with 50% AAPL and 50% MSFT.
    shocks: Dictionary of hypothetical shocks in percentage terms for each asset.
            For example {'AAPL': -0.1, 'MSFT': -0.2} represents a 10% drop in AAPL and 20% drop in MSFT.
    """
    scenario = {
        'type': 'hypothetical',
        'assets': assets,
        'shocks': shocks
    }
    return scenario
```

We define the function for hypothetical scenario as shown above. It returns a dictionary containing the asset composition and their respective shocks in percentage decimal form.

Monte Carlo scenario

Monte Carlo scenario analysis defines a stochastic framework in which asset returns are simulated across many correlated random paths based on assumed statistical properties and dependence structures. By generating a large distribution of potential portfolio outcomes, this approach enables the assessment of tail risk, loss dispersion, and uncertainty beyond single-point estimates, complementing deterministic stress scenarios with probabilistic insight.



```python
def monte_carlo_scenario(initial_investment, weights, num_trading_days, num_simulations, risk_factors, covariance_matrix, means, distribution='normal') -> list:
  """
  This function generates multiple scenarios by drawing random samples from specified probability distributions.
  """
  num_of_assets = len(risk_factors)

  # Generate random samples based on the specified distribution
  # Distribution Option 1: Normal Distribution
  if distribution == 'normal':
    """
    This method is the same as the monte_carlo_simulation_returns function but adapted to return the simulated returns for each scenario.
    """
    meanM = np.full(shape = (num_trading_days, len(weights)), fill_value = means)
    meanM = meanM.T

    portfolio_sims = np.full(shape=(num_trading_days, num_simulations), fill_value=0.0)

    for sim in range(num_simulations):
      Z = np.random.normal(size=(num_trading_days, len(weights)))
      L = np.linalg.cholesky(cov_matrix)
      dailyReturns = meanM + np.inner(L, Z) #Correlated daily returns for individual stocks
      portfolio_sims[:,sim] = np.cumprod(np.inner(weights, dailyReturns.T)+1)*initial_investment

    # Compute the returns from the portfolio simulations given the initial investment
    simulated_returns = (portfolio_sims - initial_investment) / initial_investment

  # Distribution Option 2: Multivariate t-Distribution
  elif distribution == 't':
    df = 5  # degrees of freedom for t-distribution
    g = np.random.chisquare(df, num_simulations) / df
    z = np.random.normal(size=(num_simulations, num_of_assets))
    simulated_returns = means + (z.T / np.sqrt(g)).T @ np.linalg.cholesky(covariance_matrix).T

  # Distribution Option 3: Uniform Distribution
  elif distribution == 'uniform':
    simulated_returns = np.random.uniform(low=-0.1, high=0.1, size=(num_simulations, num_of_assets))

  # Unsupported Distribution
  else:
    raise ValueError("Unsupported distribution type")

  scenarios = []
  for i in range(num_trading_days):
    scenario = {
      'type': 'monte_carlo',
      'changes': dict(zip(risk_factors, simulated_returns[i, :]))
    }
    scenarios.append(scenario)
  return scenarios
```

We defined our Monte-Carlo scenario as shown above, there was some quick adaption to the original method in the previous section but the logic is the same. Over here, we opened the consideration to include multivariate t-distribution as well as uniform distribution, on top of our standard normal distribution.

Classes for Assets and Portfolio

Instead of sticking to dataframes, we will be defining classes to model the behaviour and characteristics of our assets (Stock, Bond, Option) and portfolio.

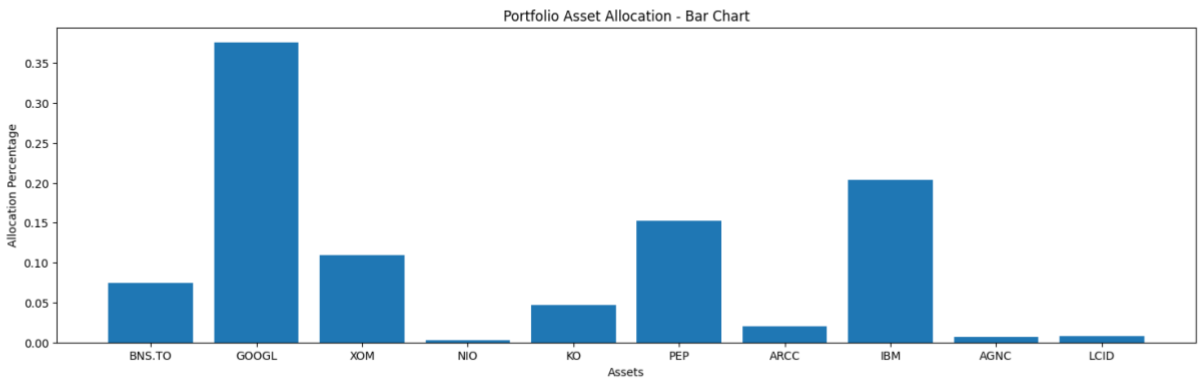
Class	Description
Stock	- Represents an equity position defined by a ticker symbol and number of shares held. - Stores market attributes such as current price and volatility (either user-provided or fetched dynamically). - Includes logic to retrieve the latest market price via yfinance if not explicitly supplied. - Provides methods to compute the market value of the position and to retrieve the ticker identifier.

Bond	- Models a fixed-income instrument using key contractual parameters: maturity date, coupon rate, face value, and yield to maturity. - Uses a simplified valuation approach based on discounting the face value by the yield to maturity. - Identified by a name attribute to allow clear distinction within portfolio allocations
Option	- Represents a derivative contract linked to an underlying asset, with defined strike price, expiration date, and option type (call or put). - Designed to be pricing-model agnostic, with option value intended to be set externally (e.g. via Black–Scholes or other models). - Provides a placeholder valuation interface through get_value() for integration into the portfolio.
Portfolio	- Acts as a container for heterogeneous assets (stocks, bonds, options, etc.). - Supports dynamic asset aggregation through an add_asset method. - Computes total portfolio value by summing the individual values of all constituent assets. - Calculates asset allocation weights based on current market values, with logic to distinguish between asset types for clear reporting.

While we may have defined python classes for a range of asset classes, we will be sticking to the initial stock tickers highlighted at the start of our project, namely:

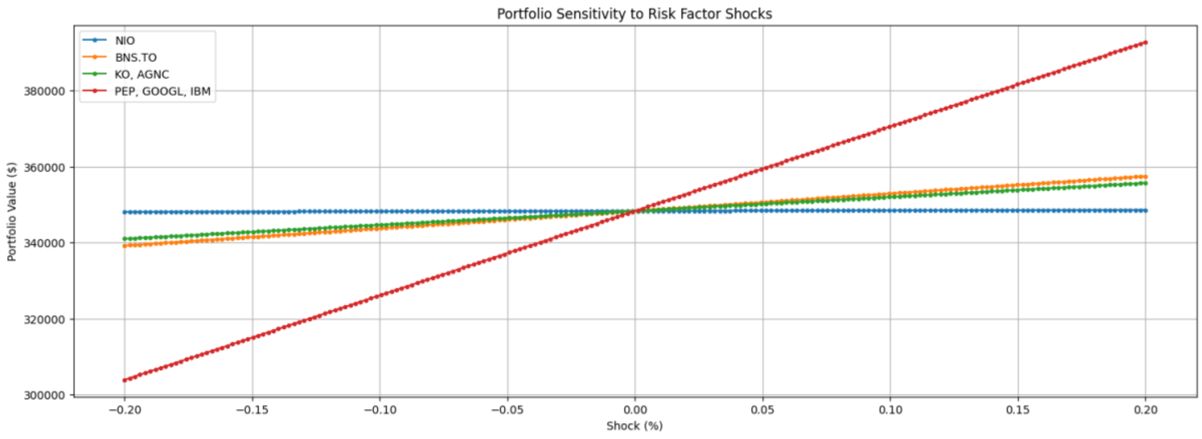
- 'BNS.TO', 'GOOGL', 'XOM', 'NIO', 'KO', 'PEP', 'ARCC', 'IBM', 'AGNC', 'LCID'

Instantiating our portfolio class and adding our stocks (with randomized share counts for now), we end up with a portfolio breakdown as shown below.



Sensitivity Analysis

Now we need to define a function that performs the sensitivity analysis portion. This function applys a range of shocks to specified risk factor(s) and measure the resulting impact on the portfolio value. Given a portfolio, our chosen risk factor(s), and a corresponding set of shock magnitudes as inputs, the function revalues the portfolio under each scenario. The resulting output provide insight into how sensitive the portfolio is to changes in that particular risk factor(s) and help identify potential sources of vulnerability.



As shown above, applying a shock range of −20% to +20% across four separate runs—each corresponding to a different risk factor—produces the observed fluctuations in portfolio value. The resulting response is linear across the shock range, as an identical proportional shock is applied uniformly to each identified risk factor.

While this provides a useful first-order sensitivity check, it is a simplified representation of market behaviour. For a more realistic analysis, the framework can be extended to randomise shock magnitudes across risk factors at each iteration, rather than applying the same deterministic shock to all factors simultaneously. This allows for more heterogeneous and plausible

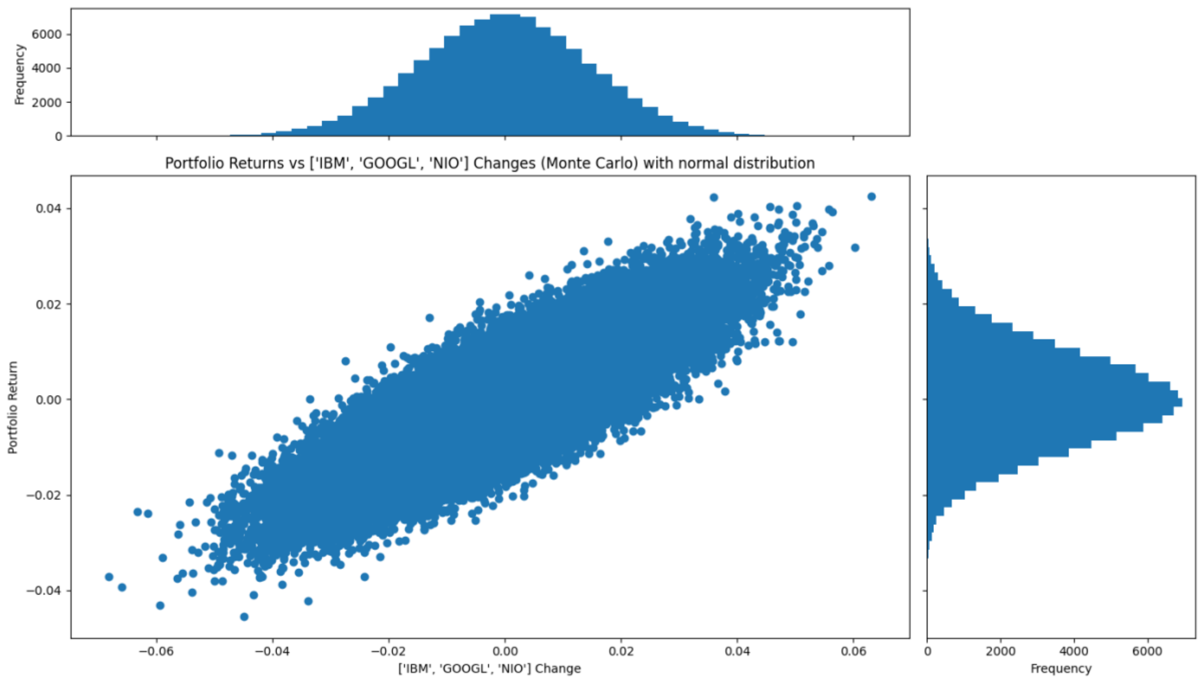
Stress Testing our Portfolio

With the sensitivity function defined, we can now proceed to stress test the portfolio under a range of scenarios—historical, hypothetical, or Monte Carlo. For each scenario, asset values are adjusted according to the specified shocks or realised historical returns, after which the portfolio is revalued. This process allows us to assess the impact of extreme market conditions on overall portfolio performance and identify potential sources of risk under stressed environments.

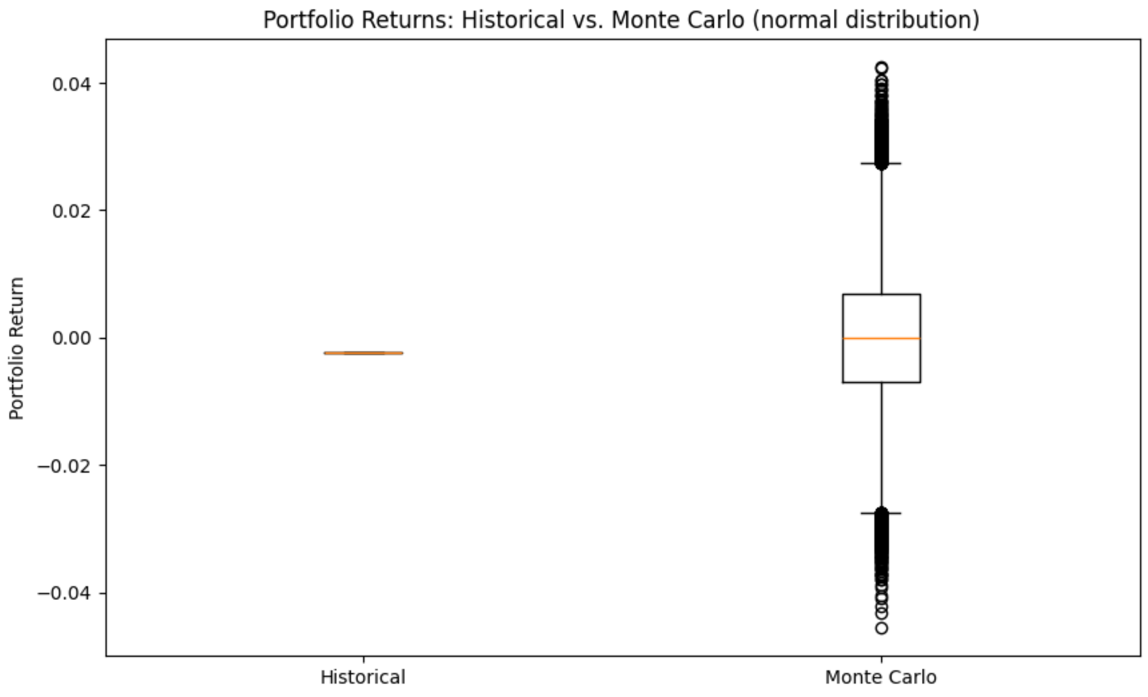
We defined the necessary functions for backtesting our portfolio and running the relevant scenarios (the codes can be found in the Jupyter notebook). Using the following values as an example, we ran our portfolio backtesting procedure.

Variable	Value
num_of_simulations	100,000
shock_factor	'IBM'
asset_weights	portfolio.get_asset_allocation()
shocks	{asset: np.random.uniform(-0.3, 0) for asset in asset_weights}
risk_factors	'IBM', 'GOOGL', 'NIO'
covariance_matrix	cov_matrix (Defined and computed earlier in the script)
means	mean_return (Defined and computed earlier in the script)
initial_investment	portfolio.get_total_value()
num_trading_days	252

By running on our three scenarios, we got the following results.



And a box plot with the following results



Summary

In conclusion, combining Monte Carlo simulation with scenario-based stress testing provides a powerful framework for understanding portfolio risk. Historical scenarios reveal how portfolios would have responded to actual past market shocks, hypothetical scenarios allow testing of forward-looking “what-if” events, and Monte Carlo simulations capture a wide range of possible outcomes under stochastic, correlated market movements. Together, these methods enable a comprehensive assessment of portfolio sensitivity, tail risk, and resilience under extreme conditions, providing valuable insights for risk management and informed decision-making.