Beating Mr. Market? A Monte Carlo Simulation of a Value Stock Portfolio vs. The S&P 500: A Quantitative Approach That Leverages Matrix Theory

Edward E. Daisey

23 February 2025

Abstract

This paper presents a Monte Carlo simulation that compares two portfolio strategies. The first portfolio—the value portfolio—is constructed following the investment philosophy of Warren Buffet. This portfolio focuses on companies with strong fundamentals such as high intrinsic value, desirable size, low volatility, and durable business strength, while simultaneously avoiding companies with excessive growth expectations and high dividend yield. Ten companies not included in the S&P 500 were chosen after a detailed review of their annual 10-K and quarterly 10-Q financial reports. The second portfolio—the S&P 500 portfolio—is a commonly recommended portfolio for retail investors as it provides broad market exposure, has a very low expense ratio relative to actively managed funds, and has proven long-term performance. The portfolio focuses on one stock, VOO, an exchange-traded fund that directly tracks the S&P 500. Each portfolio is allocated \$10,000. The simulation was conducted over a ten year period (approximately 2,520 trading days). Asset prices are modeled using a log-normal model to capture the stochastic nature of price movements and the compounding effect of returns. The results of the Monte Carlo simulation indicate that the value portfolio presented in this paper outperforms the S&P 500 portfolio.

Disclaimer: This paper does not constitute financial advice. This paper is for academic purposes only.

1 Introduction

Warren Buffet's investment approach emphasizes discovering & analyzing companies whose intrinsic value exceeds their market price. Value is determined by several factors. This includes a low price relative to strong fundamentals (e.g., a price-to-earnings ratio below 15), desirable size (i.e., financially stable mid-to-large-cap companies with scalable operations), low volatility, and durable business strength (i.e., consistent earnings, an economic moat, low debt, and trustworthy management). Companies with overinflated growth expectations or excessive dividend payouts to shareholders are generally avoided. For a detailed discussion of Buffet's investment philosophy, see the following sources in the Bibliography section [1–3].

Passive investing into index funds has become popular with retail investors.[4] To minimize the effect of overpricing due to the popularity of index funds, this investigation constructed the value portfolio using 10 companies not included in the S&P 500, where the S&P 500 captures 500 of the largest companies on stock exchanges in the United States of America. The portfolio includes: Cinemark Holdings (CNK), Hain Celestial Group (HAIN), Lamar Advertising Company (LAMR), Macerich Company (MAC), Mistras Group (MG), Peoples Bancorp (PEBO), Preformed Line Products Company (PLPC), Signet Jewelers (SIG), TreeHouse Foods (THS), and Valaris Limited (VAL). The companies were selected following thorough review of their regulatory filings (10-K and 10-Q).[5] The S&P 500 portfolio is represented by VOO, which is regarded as an accurate marker for the S&P 500 index.[6]

Both portfolios were initially allocated \$10,000. The asset prices were allowed to evolve over 2,520 trading days (approximately 10 years of investing). A log-normal model is used because it ensures that the prices remain positive and captures the compounding effects of returns.[7] Fundamental analysis predicts an annual return of 15.52% (yielding a daily log-return (μ) around 0.0005725) and daily volatility (σ) is 0.0126 based on historical data.[8] VOO typically exhibits a slightly lower daily log-return (approximately 0.0004141—a conservative estimate)[9] with a daily volatility that is approximately 0.00968.[10] Parameter calculations may be found in Appendix A. The results of the Monte Carlo simulation indicate that the value portfolio presented in this paper outperforms the S&P 500 portfolio.

2 Methods

2.1 Mathematical Models For Both Portfolios

Let us start with the construction of the mathematical model for the value portfolio. Let

$$\mathbf{p}(t) = \begin{bmatrix} P_1(t) \\ P_2(t) \\ \vdots \\ P_{10}(t) \end{bmatrix}$$

denote the price vector of the 10 selected companies affiliated with the value portfolio at time t (days). At $t = 0$ days (prices determined on 22 February 2025), it follows that:

$$\mathbf{p}(0) = \begin{bmatrix} \$27.50 \\ \$4.18 \\ \$121.91 \\ \$19.82 \\ \$9.89 \\ \$31.78 \\ \$134.78 \\ \$52.75 \\ \$30.58 \\ \$41.78 \end{bmatrix}.$$

Given a \$1,000 allocation per company, the number of whole shares purchased for company i is computed as:

$$s_i = \left\lfloor \frac{\$1000}{P_i(0)} \right\rfloor.$$

Furthermore, the value of the portfolio is defined by the following equation:

$$V(t) = \sum_{i=1}^{10} s_i \, P_i(t) = \mathbf{s}^\top \mathbf{p}(t),$$

where $\mathbf{s}$ is the vector capturing the number of whole shares purchased.

Let us now create the mathematical model for the S&P 500 portfolio. Let $Q(t)$ be the price of VOO at time t (days). Recalling that the entirety of the initial \$10,000 is to be allocated to VOO, and with an initial price of $Q(0) = \$550.0$, the whole number of shares purchased is:

$$r = \left\lfloor \frac{\$10,000}{\$550.0} \right\rfloor.$$

The S&P 500 portfolio value is then:

$$V_{\text{S\&P}}(t) = r \cdot Q(t).$$

2.2 Linearity of the Valuation Function

Linearity ensures efficiency, reliability, and consistency in portfolio valuations. The valuation functions for both portfolios are linear transformations. The value portfolio is a linear transform from $\mathbb{R}^{10}$ (ten stocks) to $\mathbb{R}^1$ (one final portfolio value). The S&P 500 portfolio is a linear transform from $\mathbb{R}^1$ (one stock) to $\mathbb{R}^1$ (one final portfolio value). A worked out example may be found in Appendix A. For a general valuation function defined by:

$$V(\mathbf{x}(t)) = \mathbf{w}^\top \mathbf{x}(t),$$

where $\mathbf{w}$ is a vector (either $\mathbf{s}$ or r) and $\mathbf{x}(t)$ represents the vector associated with asset price, the following two properties hold:

$$V(\mathbf{x}(t) + \mathbf{y}(t)) = V(\mathbf{x}(t)) + V(\mathbf{y}(t)),$$

$$V(c\,\mathbf{x}(t)) = c\,V(\mathbf{x}(t)).$$

Theorem. The function $V(\mathbf{x}(t)) = \mathbf{w}^\top \mathbf{x}(t)$ is a linear transform.
Proof. By the distributive property of the dot product,

$$V(\mathbf{x}(t) + \mathbf{y}(t)) = \mathbf{w}^\top(\mathbf{x}(t) + \mathbf{y}(t)) = \mathbf{w}^\top \mathbf{x}(t) + \mathbf{w}^\top \mathbf{y}(t) = V(\mathbf{x}(t)) + V(\mathbf{y}(t)).$$

In a similar fashion for any scalar c,

$$V(c\,\mathbf{x}(t)) = \mathbf{w}^\top(c\,\mathbf{x}(t)) = c\,(\mathbf{w}^\top \mathbf{x}(t)) = c\,V(\mathbf{x}(t)). \quad \square$$

2.3 Asset Price Evolution and the Log-normal Model

The prices associated with the assets are assumed to evolve according to a log-normal process:

$$P_i(t+1) = P_i(t)\,\exp\big(r_i(t)\big).$$

The daily log-return, $r_i(t)$, is sampled from a Gaussian (normal) distribution, $N(\mu, \sigma^2)$.

A log-normal model is used because it ensures that the positive prices and captures the exponential compounding effects of the returns. Based on fundamental analysis & historical leads to a daily log-return around 0.0005725 (μ) and daily volatility around 0.0126 (σ), whereas the S&P 500 portfolio typically exhibits a slightly lower daily log-return (approximately 0.0004141 (μ)) with a daily volatility that is approximately 0.00968 (σ).

3 Results

A PCG64 pseudorandom number generating algorithm was used because it is efficient for a large number of runs and has excellent statistical properties.[11] Monte Carlo simulations were conducted over 2,520 trading days with 100 independent runs to simulate 100 different manifestations of reality. For each run, daily log-returns for each of the assets are sampled using the appropriate μ and σ values. Asset prices are updated multiplicatively. The portfolio value is computed via the dot product described earlier in this paper. The portfolio outperformed the S&P 500. Figures and tables capturing points of interest that also support this claim may be found in Appendix B.

4 Discussion

The combination of Matrix Theory with the Monte Carlo simulation outlined in this paper and in the code (see Appendix C) offers a simple, powerful framework for analyzing portfolios. Since the portfolio valuation can be expressed as the dot product of (1) a vector that captures the number of shares purchased, and (2) a vector that captures the price of the asset, this simplifies computations in addition to leveraging key properties of linear transformations such as additivity and distributivity.

The results suggest that a portfolio constructed following Buffet's principles can outperform the S&P 500 over a ten year period of time. Future avenues of research may extend this analysis by incorporating the reinvestment of dividends, incorporating transaction costs, in addition to having volatility changing as a function of time. Moreover, exploring the impact of fractional shares as opposed to whole shares can further refine this model.

References

[1] S. Wu, K. Kuehn, and J. Jiang, "The Value Perspective: The Case of Warren Buffett and his investment behavior towards Apple, Walmart, and Amazon," *Global Journal of Accounting and Finance*, vol. 3, no. 1, pp. 73–87, 2019. Link.

[2] E. Chirkova, "Why is It that I am not Warren Buffett?," *American Journal of Economics*, vol. 2, no. 6, pp. 115–121, 2012. doi: Link..

[3] R. G. Hagstrom, *The Warren Buffett Way: Investment Strategies of the World's Greatest Investor*, 3rd ed., Wiley, 2013.

[4] UCI Paul Merage School of Business, "The Dominance of Passive Investing and Its Effect on Financial Markets," University of California, Irvine, Oct. 2024. Online. Available: Link.

[5] ShareSeer, "ShareSeer Financial Analytics,". Online. Available: Link.

[6] Vanguard, "Vanguard S&P 500 ETF (VOO) Fund Overview," Vanguard Group, 2024. Online. Available: Link.

[7] M. Cudina, "Binomial Tree and Lognormality Lecture Slides," Department of Mathematics, University of Texas at Austin. Online. Available: Link.

[8] J. Y. Campbell, A. W. Lo, and A. C. MacKinlay, "The Econometrics of Financial Markets," Princeton University Press, 1997.

[9] J. M. Gruber and S. J. Richards, "The Long-term Return on the Original S&P 500 Firms," Rodney L. White Center for Financial Research, The Wharton School, University of Pennsylvania, Working Paper No. 04-29, 2004. Online. Available: Link.

[10] W. B. Hao and R. Watts, "A Historical Perspective on Factor Index Performance across Macroeconomic Cycles," S&P Dow Jones Indices, November 2024. Online. Available: Link.

[11] M. E. O'Neill, "PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation," Harvey Mudd College, Technical Report HMC-CS-2014-0905, 2014. Available: `Link`.

Appendix A: Parameter Calculations & Example

Section I. Parameter Calculations for the Portfolios

(1) Value Portfolio Portfolio:
Annual Return: Approximately 15.52%
Daily Log-Return:

$$\mu_{\text{daily}} = \frac{\ln(1.1552)}{252} \approx 0.0005725.$$

Annual Volatility: Approximately 20%
Daily Volatility:

$$\sigma_{\text{daily}} = \frac{0.20}{\sqrt{252}} \approx 0.0126.$$

(2) S&P 500 Portfolio:
Annual Return: Approximately 11%
Daily Log-Return:

$$\mu_{\text{daily}} = \frac{\ln(1.11)}{252} \approx 0.0004141.$$

Annual Volatility: Approximately 15.38%
Daily Volatility:

$$\sigma_{\text{daily}} = \frac{0.1538}{\sqrt{252}} \approx 0.00968.$$

Section II. Example

Suppose we are interested in purchasing 3 stocks with a total investment of \$3,000. We decide to invest in Cinemark Holdings (CNK), Lamar Advertising (LAMR), and Signet Jewelers (SIG).

Step 1: Definition of the Price Vector

The initial stock prices are:

$$\mathbf{p}(0) = \begin{bmatrix} \$27.45 \\ \$121.91 \\ \$52.75 \end{bmatrix}.$$

Step 2: Compute Shares Purchased

Each stock is allocated \$1,000, so the number of whole shares purchased is:

$$\mathbf{s} = \begin{bmatrix} \left\lfloor \frac{\$1{,}000}{\$27.45/\text{Share}} \right\rfloor \\ \left\lfloor \frac{\$1{,}000}{\$121.91/\text{Share}} \right\rfloor \\ \left\lfloor \frac{\$1{,}000}{\$52.75/\text{Share}} \right\rfloor \end{bmatrix} = \begin{bmatrix} 36 \\ 8 \\ 18 \end{bmatrix} \text{ Shares.}$$

Step 3: Compute the Value of the Portfolio

The total portfolio value is computed using the dot product at $t = 0$ days:

$$\begin{aligned} V(0) = &(36 \text{ Shares} \times \$27.45/\text{Share}) \\ &+ (8 \text{ Shares} \times \$121.91/\text{Share}) \\ &+ (18 \text{ Shares} \times \$52.75/\text{Share}) = \$2,912.98. \end{aligned}$$

The portfolio valuation function is a linear transformation from $\mathbb{R}^3$ to $\mathbb{R}$.

Appendix B: Table & Figures

Metric	Value Portfolio	S&P 500 (VOO)
Final Value (Mean)	$51,131.91	$29,491.18
Final Value (Std. Dev.)	$11,322.09	$15,983.19
Outperformance Frequency (%)	100.00	-

Table 1. Monte Carlo Simulation Results (100 Runs, 10 Years). This table represents an example Monte Carlo simulation output comparing a Value Portfolio with the S&P 500 (VOO) over a 10-year period.

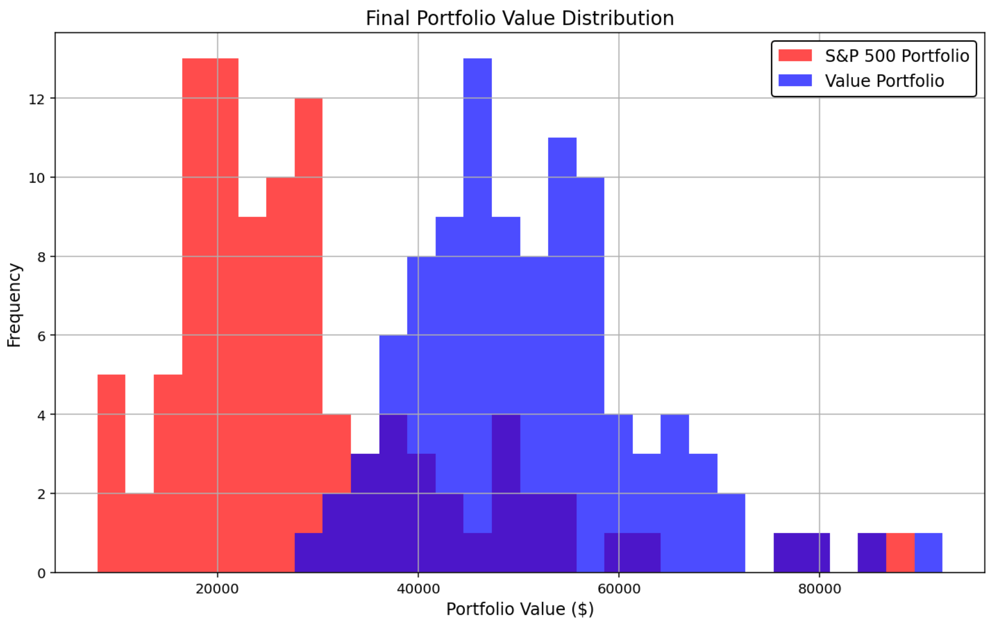

Figure 1. This figures compares the final portfolio values after 10 years of the 100 simulations.

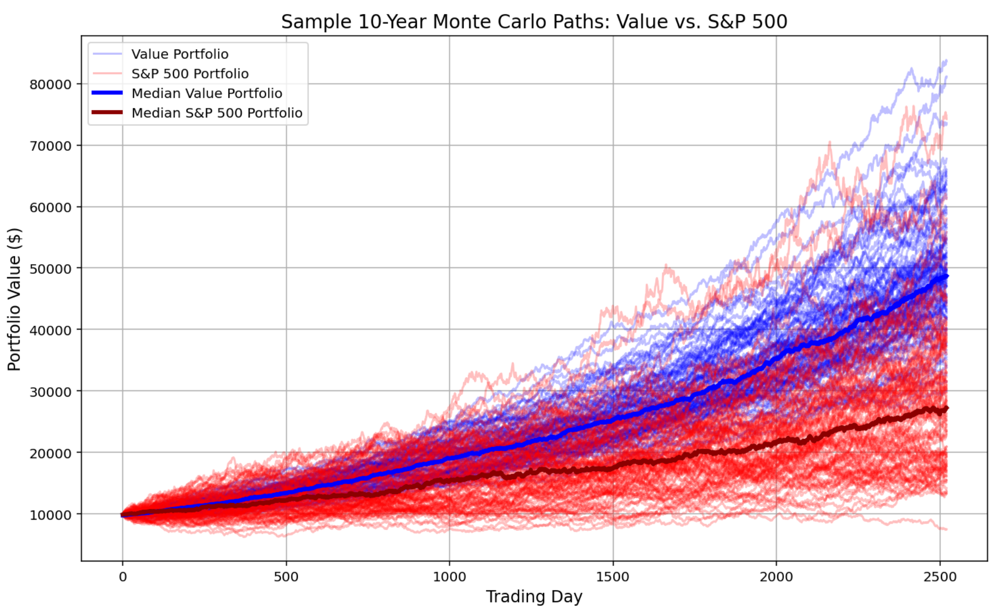

Figure 2. This figures captures 100 Monte Carlo simulation runs, with the median run in bold.

Appendix C: Source Code

The following Python code simulates the evolution of two investment portfolios (Value Portfolio vs. S&P 500 Portfolio) using a Monte Carlo simulation with log-normal asset price modeling. You can download the full source code from: `Monte Carlo Simulation Source Code`.

```python
# ##############################################################################
#       Author: Edward E. Daisey
#        Class: Matrix Theory
#    Professor: Dr. Cutrone
#         Date: 23 February 2025
#        Title: Monte Carlo Simulation for a Value Portfolio vs. The S&P 500.
# Description: This code presents two portfolios over a ten year period of time.
#               This corresponds to approximately 2520 trading days. The first
#               portfolio -- the value portfolio -- hold ten value stocks
#               outside the S&P 500.  The second portfolio -- the S&P 500 one --
#               is comprised of VOO.  Both portfolios have an inital allocation
#               of $10,000.  This code uses a log-normal price process driven by
#               a PCG64-based pseudo-random number generator for daily returns.
#               Floor rounding is used to ensure that only whole shares are
#               purchased -- this paper assumes no fractional shares.
# ##############################################################################


# ################################# Packages ##################################
import numpy as np
import matplotlib.pyplot as plt
import time
from numpy.random import PCG64, Generator
# #############################################################################


# ############################### Constants ###################################
TOTAL_DAYS = 2520        # Represents approximately 10 years of trading.
NUM_SIMULATIONS = 100    # Total number of Monte Carlo simulations.
SEED_VALUE = 10000       # Initial Seed.

# ############### Value Portfolio ###############
# Below is the list of stocks in the value portfolio. Each stock is allocated
# $1,000 in order to purchase whole shares in the company.
value_tickers = [
    "CNK",  # Cinemark Holdings
    "HAIN", # Hain Celestial Group
    "LAMR", # Lamar Advertising Company
    "MAC",  # Macerich Company
    "MG",   # Mistras Group
    "PEBO", # Peoples Bancorp
    "PLPC", # Preformed Line Products
    "SIG",  # Signet Jewelers
    "THS",  # TreeHouse Foods
    "VAL"   # Valaris Limited
]

# Current prices (USD) for the aforementioned stocks. Taken on 22 February 2025.
value_current_prices = np.array([27.45, 4.18, 121.91, 19.82, 9.89,
                                 31.78, 134.32, 52.75, 30.58, 41.78])
```

```python

# Daily log-return and volatility for the value portfolio.
MU_VALUE = 0.0005725    # Feel Free To Change THis To Match Your Portfilio!
SIGMA_VALUE = 0.0126
# ##############################################

# ############### S&P 500 Portfolio ###############
# S&P 500 simulation details. We use VOO at $550.
sp500_ticker = "VOO"
sp500_current_price = 550.0

# Typical daily log-return and volatility for the S&P 500.
MU_SP500 = 0.0004141
SIGMA_SP500 = 0.00968
################################################

# Rationale for mu (daily log-return) and sigma (daily volatility) for
# both portfolios:
# --------------------------------------------------------------------------
# 1) Value portfolio (mu=0.0005725, sigma=0.0126):
# Annual return ~ 15.52% --> mu_daily ~ ln(1.1552) / 252 ~ 0.0005725.
# Annual vol ~ 20% --> sigma_daily ~ 0.20 / sqrt(252) ~ 0.0126.
# 2) S&P 500 (mu=0.0004141, sigma=0.00968):
# Annual return ~ 11% --> mu_daily ~ ln(1.11) / 252 ~ 0.0004141.
# Annual vol closer to ~15.38% --> sigma_daily ~ 0.1538 / sqrt(252) ~ 0.00968.
# ##############################################################################


# ############################## Function 1 ##################################
# Function Name: SimulateStockPath
# Function Purpose: Simulate a single stock path over TOTAL_DAYS using daily
#                   log-returns.
# Function Input:
#     initial_price (float) - The starting stock price.
#          total_days (int) - The number of trading days to simulate.
#                mu (float) - The average daily log-return.
#             sigma (float) - The standard deviation of the daily log-return.
#           rng (Generator) - PCG64-based random number generator (rng) instance.
# Function Output:
#     A numpy array of length total_days + 1 representing daily stock prices.
def SimulateStockPath( initial_price, total_days, mu, sigma, rng ):
    daily_returns = rng.normal( loc = mu, scale = sigma, size = total_days)
    log_cumsum    = np.cumsum( daily_returns )

    path          = np.zeros( total_days + 1 )
    path[0]       = initial_price
    path[1:]      = initial_price * np.exp( log_cumsum )
    return path
# ##############################################################################


# ############################## Function 2 ##################################
# Function Name: SimulateValuePortfolio
# Function Purpose: Model a 10-stock value portfolio, each allocated $1,000,
#                   using floor rounding to purchase whole shares.
# Function Input:
```

```
#     prices (numpy array) - The current prices for the 10 value stocks.
#         total_days (int) - The number of trading days to simulate.
#               mu (float) - The average daily log-return for these stocks.
#            sigma (float) - The daily volatility for these stocks.
#          rng (Generator) - PCG64-based random number generator instance.
# Function Output:
#     A tuple: (portfolio_values, stock_paths)
#     - portfolio_values (1D array): length total_days+1, daily portfolio value.
#     -      stock_paths (2D array): shape (num_assets, total_days+1) with each
#                                    asset's path.
def SimulateValuePortfolio( prices, total_days, mu, sigma, rng ):
    # Determine Number of Assets (n = 10) & Shares:
    num_assets = len( prices )
    shares     = np.floor( 1000.0 / prices )

    # Initialize Stock Price Paths & Generate Daily Random Log-Returns For Each
    Stock:
    stock_paths       = np.zeros( ( num_assets, total_days + 1 ) )
    daily_returns_all = rng.normal(  loc = mu,
                                   scale = sigma,
                                    size = ( num_assets, total_days ) )

    # Compute Cumulativ Log-Returns:
    log_cumsum_all    = np.cumsum( daily_returns_all, axis = 1 )

    # Simulate Stock Price Paths:
    for i in range( num_assets ):
        stock_paths[ i, 0 ]  = prices[ i ]
        stock_paths[ i, 1: ] = prices[ i ] * np.exp( log_cumsum_all[ i ] )

    # Compute Total Portfolio Value at Each Time Step:
    portfolio_values = np.sum( shares[ :, None ] * stock_paths, axis = 0 )
    return portfolio_values, stock_paths
# ##############################################################################


# ############################## Function 3 ##################################
# Function Name: SimulateSP500Portfolio
# Function Purpose: Model a S&P 500 (i.e., VOO) portfolio, allocated $10,000,
#                   using floor rounding to purchase whole shares.
# Function Input:
#     current_price (float) - The initial price of VOO (i.e., $550).
#          total_days (int) - The number of trading days to simulate.
#          mu_bench (float) - The average daily log-return for VOO.
#       sigma_bench (float) - The daily volatility for VOO.
#           rng (Generator) - PCG64-based random number generator instance.
# Function Output:
#     A tuple: (portfolio_values, sp500_path)
#     - portfolio_values (1D array): length total_days+1, daily portfolio value.
#     -       sp500_path (1D array): length total_days+1, simulated daily
#                                    VOO prices.
def SimulateSP500Portfolio( current_price, total_days, mu_bench, sigma_bench, rng
    ):
    # Determine Number of Shares:
    shares_sp500     = np.floor( 10000.0 / current_price )

```

```
    # Simulate VOO Price Path:
    sp500_path       = SimulateStockPath( current_price, total_days, mu_bench, sigma_bench, rng )

    # Compute Portfolio Value Over Time:
    portfolio_values = sp500_path * shares_sp500
    return portfolio_values, sp500_path
# ##############################################################################


# ############################## Function 4 ##################################
# Function Name: MonteCarloSim (Particularly Geometric Brownian Motion-based Monte Carlo)
# Function Purpose: Run multiple simulations for the value portfolio vs. S&P 500.
# Function Input:
#                    num_sims (int) - Number of Monte Carlo runs.
#                  total_days (int) - Number of trading days to simulate.
#       value_prices (numpy array) - Prices for the value stocks.
#    mu_value, sigma_value (float) - Log-return & volatility for the value stocks.
#               sp500_price (float) - The initial VOO price.
#    mu_sp500, sigma_sp500 (float) - Log-return & volatility for VOO.
# Function Output:
#    A tuple: (final_value_vals, final_sp500_vals)
#    - final_value_vals (1D array): final day values for the value portfolio across runs.
#    - final_sp500_vals (1D array): final day values for VOO across runs.
def MonteCarloSim(    num_sims, total_days,
                  value_prices,   mu_value,  sigma_value,
                   sp500_price,   mu_sp500,  sigma_sp500 ):

    # Initializing Storage For Final Portfolio Values:
    final_value_vals = np.zeros( num_sims )
    final_sp500_vals = np.zeros( num_sims )

    # Random Number Generator:
    rng = Generator( PCG64( SEED_VALUE ) )

    # Monte Carlo Loop & Simulating Portfolios:
    for i in range( num_sims ):  # Each iteration represent one 10-year stock market scenario.
        val_port, _   = SimulateValuePortfolio( value_prices, total_days, mu_value,
                                                sigma_value, rng )
        sp500_port, _ = SimulateSP500Portfolio(  sp500_price, total_days, mu_sp500,
                                   sigma_bench = sigma_sp500, rng = rng )

        # Storing Final Portfolio Values:
        final_value_vals[ i ] =   val_port[ -1 ]
        final_sp500_vals[ i ] = sp500_port[ -1 ]

    return final_value_vals, final_sp500_vals
# ##############################################################################
```

```python
# ############################## Function 5 ##################################
# Function Name: Main
# Function Purpose: Coordinate the simulation, display results, and plotting of data.
def Main():
    # Run Monte Carlo Sim:
    final_val, final_sp5 = MonteCarloSim(
             NUM_SIMULATIONS, TOTAL_DAYS,
        value_current_prices, MU_VALUE, SIGMA_VALUE,
         sp500_current_price, MU_SP500, SIGMA_SP500
    )

    # Compute & Output Key Statistics
    mean_val          = np.mean( final_val )
    std_val           = np.std(  final_val )
    mean_sp5          = np.mean( final_sp5 )
    std_sp5           = np.std(  final_sp5 )
    median_sp5_final = np.median( final_sp5 )
    beat_count        = np.sum( final_val > median_sp5_final )
    beat_pct          = ( beat_count / NUM_SIMULATIONS ) * 100
    print( f"\n===================== Monte Carlo Results ({NUM_SIMULATIONS} Runs, 10 Years) =====================")
    print( f"{'Metric':<35}{'Value Portfolio':>20}{'S&P 500':>25}" )
    print( "-" * 85)
    print( f"{'Final Value (Mean)':<35}{mean_val:>20.2f}{mean_sp5:>25.2f}" )
    print( f"{'Final Value (Std. Dev.)':<35}{std_val:>20.2f}{std_sp5:>25.2f}" )
    print( f"{'Outperformance Frequency (%)':<35}{beat_pct:>20.2f}\n" )
    # Note: Outperformance Frequency (%) measures how often the Value Portfolio
    #       ends with a higher final value than the *median* of the S&P 500
    #       portfolio.  E.g., if 100 simulations, then a 99% outperformance
    #       frequency means that the Value Portfolio ended higher than the median
    #       S&P 500 final value in 99 out of 100 runs (or 99% of the time).
    # Example Output:
    # ================= Monte Carlo Results (100 Runs, 10 Years) ================
    # Metric                              Value Portfolio                  S&P 500
    # ---------------------------------------------------------------------------
    # Final Value (Mean)                         51131.91                 29491.18
    # Final Value (Std. Dev.)                      11322.09 15893.19
    # Outperformance Frequency (%)                 100.00

    # Histogram:
    plt.figure( figsize = ( 12, 7 ) )
    plt.title( 'Final Portfolio Value Distribution', fontsize = 14 )
    plt.xlabel( 'Portfolio Value ($)', fontsize = 12 )
    plt.ylabel( 'Frequency', fontsize = 12 )
    bin_edges = np.linspace( min( final_val.min(), final_sp5.min()),
                             max( final_val.max(), final_sp5.max()),
                             31)  # 30 Bins
    plt.hist(final_sp5, bins = bin_edges, alpha = 0.7, label = 'S&P 500 Portfolio', color = 'red' )
    plt.hist(final_val, bins = bin_edges, alpha = 0.7, label = 'Value Portfolio', color = 'blue' )
    plt.legend( fontsize = 12,   loc = 'upper right', edgecolor = 'black',
                fancybox = True, framealpha = 1, title_fontsize = 12 )
    plt.grid( True )
```

```python
    plt.show()

    # Monte Carlo Plot:
    plt.figure( figsize=( 12, 7 ) )
    plt.title( 'Sample 10-Year Monte Carlo Paths: Value vs. S&P 500', fontsize = 14 )
    plt.xlabel( 'Trading Day', fontsize = 12 )
    plt.ylabel( 'Portfolio Value ($)', fontsize = 12 )

    # Random Number Generator For Monte Carlo Plot:
    rng_sample = Generator( PCG64( int( time.time() ) ) )

    # Initialize Lists For Storing Portfolio Paths For Plot:
    sample_val_paths = []
    sample_sp5_paths = []

    # Simulate & Plot Monte Carlo Paths For Plot:
    for _ in range( NUM_SIMULATIONS ):
        val_port, _ = SimulateValuePortfolio( value_current_prices, TOTAL_DAYS, MU_VALUE, SIGMA_VALUE, rng_sample )
        sp5_port, _ = SimulateSP500Portfolio(  sp500_current_price, TOTAL_DAYS, MU_SP500, SIGMA_SP500, rng_sample )
        sample_val_paths.append( val_port )
        sample_sp5_paths.append( sp5_port )
    for path in sample_val_paths:
        plt.plot( path, color = 'blue', alpha = 0.25,
                  label = 'Value Portfolio' if 'Value Portfolio' not in plt.gca().get_legend_handles_labels()[ 1 ] else "" )
    for path in sample_sp5_paths:
        plt.plot( path, color = 'red', alpha = 0.25,
                  label = 'S&P 500 Portfolio' if 'S&P 500 Portfolio' not in plt.gca().get_legend_handles_labels()[ 1 ] else "" )

    # Compute & Plot Median Price Path For Plot & Finalize Monte Carlo Plot:
    median_val_path = np.median( sample_val_paths, axis = 0 )
    median_sp5_path = np.median( sample_sp5_paths, axis = 0 )
    plt.plot( median_val_path, color = 'blue',    linewidth = 3, label = 'Median Value Portfolio' )
    plt.plot( median_sp5_path, color = '#8B0000', linewidth = 3, label = 'Median S&P 500 Portfolio' )
    plt.legend()
    plt.grid( True )
    plt.show()
# ############################## Main Execution ##############################
if __name__ == "__main__":
    Main()
# ##########################################################################
```