Robust Portfolio Optimization

Anthony, Lucas

Shahid, Maira

Singh, Gurjus

Vaidya, Jay

School of Professional Studies, Northwestern University

MSDS 460-DL-SEC 56: Decision Analytics

Dr. Irene Tsapara

Abstract

Portfolio optimization is an important analytical framework in the investment management profession. Optimization is frequently sought with one of two goals in mind: optimizing expected return for a given unit of expected risk or optimizing expected risk for a given basket of securities. Classical Mean-Variance Portfolio Optimization, as established by Harry Markowitz in 1952, is the bedrock of modern portfolio theory, but its application is often untenable in practice as sensitivities to small changes in inputs, such as expected returns of a given security, and numerically ill-conditioned covariance matrices can lead to dramatic changes in recommended portfolio allocations and can promote large errors. Constructing minimum-variance portfolios using Hierarchical Risk Parity (HRP) algorithms, as formulated by Marcos Lopez de Prado, offers the opportunity to create more robust portfolio allocations subject to lower levels of recommended allocation variation. Furthermore, robustness of optimal portfolio selection can be established through simulation to determine those portfolios that provide the greatest protection against losses across simulated returns scenarios. Using a minimax regret decision criterion, defining regret with respect to negative expected portfolio returns, we establish an optimal robust portfolio selection strategy. Our developed optimal portfolio selection strategy is demonstrated using historical data drawn from 29 of 30 Dow Jones Industrial Average components as of November 13, 2020, for which full sets of returns data are available throughout our entire scrutiny period beginning January 1, 2017.

Keywords: Robust, Portfolio Optimization, Monte Carlo, Minimax Regret, Mean-Variance Optimization, Hierarchical Risk Parity

Introduction & Problem Setting

The primary objective of this project is to determine how to develop, and then select the best alternative among, a set of optimized portfolio allocations, using Monte Carlo simulation, in order to achieve a robust result that minimizes the expected maximum drawdown of a portfolio of securities. The final outcome of this line of inquiry is the identification of one set of portfolio weights for our basket of securities that is expected to best achieve this result, given the uncertainty inherent in forward-looking returns, correlation and covariance estimates.

Our analysis is undertaken as a point-in-time study and should not be construed as a time-series evaluation of the best methods for re-weighting portfolios across time. Rather, our method, once validated, could become the basis for such studies in the future. The basket of securities we use in our analysis is that of 29 of 30 equity constituents of the Dow Jones Industrial Average as of November 13, 2020, for which returns data is available during our entire scrutiny period.

The inputs for our analysis are the daily logarithmic returns of our chosen basket of securities beginning on January 1, 2017 through November 13, 2020, and the corresponding derived correlation and covariance matrices for this basket. Data used in our analysis is drawn from publicly available daily adjusted closing prices for each security provided by Yahoo! Finance. Furthermore, the Adjusted Daily Closing prices used in our analysis account for all applicable stock splits and dividend distributions in conformance with the Center for Research in Security Prices (CRSP) standards (Yahoo, 2020).

Our approach to the problem of portfolio optimization under uncertainty requires three distinct phases: First, historical returns information for the chosen basket of securities is obtained, scrutinized for conformance to the normal distribution, and then used to produce 1,000 simulated periods of returns using Monte Carlo Simulation techniques; Second, each array of simulated returns are used to identify an optimal portfolio allocation, resulting in 1,000 sets optimal portfolio allocations, which are then stored in tuples; Third, using each of the 1,000 tuples, each one of which represents an optimal portfolio allocation, the expected portfolio returns are calculated for each portfolio across the 1,000 simulated returns profiles generated by the Monte Carlo simulation in phase two. By scrutinizing the performance of each of the 1,000 optimal portfolios, across 1,000 simulated returns profiles, we then select the one portfolio that demonstrates the maximum robustness across the entire range of simulations; or, stated differently, we select the one portfolio that minimizes the maximum expected drawdown (i.e. regret) across our simulations.

The optimal portfolios generated for our study are produced using the Hierarchical Risk Parity approach to produce minimum-variance portfolio allocations. As will be detailed in the Literature Review and Methodology sections of this paper, we first began our analysis using the Classical Mean-Variance portfolio optimization technique, as developed by Harry Markowitz, but quickly determined that practical difficulties in application rendered this approach sub-optimal.

Literature Review

Classical mean-variance portfolio optimization (MVO), as first detailed by Harry Markowitz in his seminal 1952 paper, *Portfolio Selection*, is the point of departure for the field of portfolio optimization. MVO seeks to construct optimal portfolio allocations among a basket of securities by maximizing the expected return of the portfolio for a given unit of risk, with risk defined in terms of expected portfolio variance (Markowitz, 1952, p. 82). In using Markowitz's method, optimal portfolios are solved for using quadratic programming methods in general and Markowitz's Critical Line Algorithm in particular (Lopez de Prado, 2018, p. 222). Markowitz defined the parameters of his model in the following equations (1952, p. 81):

1. Expected portfolio return = E = $\sum_{i=1}^{N} Xi\ \mu\ i$
    a. Where X_i is the percentage of the investor's assets which are allocated to the ith security and μ_i is the expected return of the ith security

2. Expected portfolio variance = V = $\sum_{i=1}^{N} \sum_{i=1}^{N} Xi\ Xj\ \sigma ij$
    a. Where σ_{ij} is the covariance of the ith and jth securities and covariance is expressed as the correlation of returns R_i and R_j multiplied by the standard deviation of R_i and R_j, respectively
        i. Formally, $\sigma_{ij} = \rho_{ij}\ \sigma_i\ \sigma_j$

3. Sum of the security allocations = $\sum_{i=1}^{N} Xi = 1$
    a. Where the sum of the allocations to *N* securities is equal to 1, or 100% of an investor's available assets

4. Minimum security allocation: $X_i \geq 0$
    a. Where "short sales" are not permitted and the allocation to each security is greater than or equal to 0

Using quadratic programming, Markowitz's method seeks to find the greatest level of expected return per unit of risk for a given portfolio construction by using the aforementioned inputs and altering the weights of the securities in the portfolio. This method offered two major breakthroughs: first Markowitz "…quantified return and risk of a security, using the statistical measures of its expected return and standard deviation"; and second, he "suggested that investors should consider return and risk together, and determine the allocation of funds among investment alternatives on the basis of their return-risk trade-off" (Kolm, Tütüncü, & Fabozzi, 2013, p. 356). While the riskiest investor may choose to maximize return, ignoring risk, and a conservative investor may be most interested in minimizing variance at the expense of the return achieved, the essence of modern portfolio optimization is to find the balance between the two; to select the portfolio that will provide optimal return at a specified risk tolerance.

Markowitz's method, however, is not without its detractions. The method is premised on the assumption that expected mean returns and covariance among securities are known values. Furthermore, as has been demonstrated by Michaud, small deviations in forecasted returns can cause the CLA optimization method to produce significantly different portfolio allocations

(Michaud, 1998). Additionally, Bailey and Lopez de Prado have found that the required inversion of large covariance matrices used in quadratic programming methods is "…prone to large errors when the covariance matrix is numerically ill-conditioned, that is, when it has a high condition number" (2012).

The sensitivity of results of the MVO method to the issues detailed above has come to be known colloquially as Markowitz's curse (Lopez de Prado, 2018, p. 222). Over the past sixty years, much work has been done to expand and attempt to improve upon Markowitz's work. The two primary lines of inquiry to this end are, first, approaches to improve mean estimated returns for the securities in question, and second, approaches to address instability concerns and input sensitivities.

In addressing returns estimates, some researchers have added the application of Bayesian Estimators, such as the James-Stein approach which uses the grand mean of historic returns as "…this is the equivalent to assuming that you cannot confidently decide that one stock has an expected return different from that of any other security" (Jobson & Korkie, 1981). Others, still, have applied the Black–Litterman model which is an "…approach where the estimate of expected returns is calculated as a weighted average of the market equilibrium (e.g., the CAPM equilibrium) and the investor's views" (Kolm, Tütüncü, & Fabozzi, 2013).

In order to address stability concerns, proposals, such as that by Ledoit and Wolf, have been put forward with respect to estimating a more stable covariance matrix for security returns. Ledoit and Wolf recommend dampening the variation effects of sample covariance matrices by using "…an optimally weighted average of two existing estimators: the sample covariance matrix and single-index covariance matrix", in a process known as shrinkage (2003).

These proposed remedies, though beneficial in some respects, do not, however, sufficiently mitigate the issues of input sensitivity and errors stemming from numerical difficulties when inverting sizable covariance matrices, particularly in context of our goal of robust portfolio optimization. One potential solution that has been explored to remedy the aforementioned issues is to apply the Hierarchical Risk Parity (HRP) method for determining optimal portfolio allocations.

In contrast to MVO, the HRP method shifts the focus from classical areas of mathematics, such as geometry, linear algebra and calculus, to that of more modern techniques such as graph theory and machine learning. According to Lopez de Prado, a strong proponent of HRP methods:

> One reason for the instability of quadratic optimizers is that the vector space is modelled as a complete (fully connected) graph, where every node is a potential candidate to substitute another. In algorithmic terms, inverting the matrix means evaluating the partial correlations across the complete graph…[if you examine] the relationships implied by a covariance matrix of 50 x 50, that is 50 nodes and 1225 edges. This complex structure magnifies small estimation errors, leading to incorrect solutions. Intuitively, it would be desirable to drop unnecessary edges. (Lopez de Prado, 2018, p. 224)

HRP seeks to overcome the instability of quadratic optimizers by running an algorithm in three distinct stages: tree clustering, quasi-diagonalization and recursive bisection. In such a way, a hierarchy is introduced in a basket of securities being scrutinized wherein portfolio weights are no longer competing with every other security for allocations, but rather compete within generated clusters for allocations. The clustering of securities, in essence, removes edges from the vector space connecting securities and produces a more stable outcome. Furthermore, HRP is applied without requiring the inversion of a covariance matrix, further enhancing its appeal. The practical steps required to implement the HRP method, and how we arrived at our decision to employ this method, will be detailed in the Methodology section of this paper.

Methodology

In order to achieve our goal of deriving a robust, optimal portfolio allocation for our basket of securities, the first step is to determine the appropriate method for developing optimal portfolios. Once a method of optimization has been selected, we then simulate the returns for our basket of securities over N iterations, each iteration having the same number of observations, for each security, as the historical dataset upon which our analysis is predicated. For each of the N sets of simulated returns, we then apply our method of portfolio optimization to generate one set of optimal portfolio allocations (one for each of the simulated sets of returns). Using the N sets of optimal portfolio allocations, we then determine the expected return, for each optimal portfolio, in each of the N simulated returns scenarios. In practice, this results in N x N data points of expected returns; each optimal portfolio allocation, 1 to N, having N data points of expected returns. Finally, we apply our decision criterion, the minimax regret approach, whereby the single optimal portfolio is identified as that which produces the minimum level of regret (expressed as expected portfolio drawdown) across all simulated returns scenarios. In other words, the optimal portfolio is that which produces the minimum level of maximum regret across the simulations.

Our initial approach to generate the optimal portfolio allocations for our analysis was to use the Markowitz MVO method. Using Python as our programming language of choice, we set out to test the MVO method on our basket of securities using the Scientific Python (SciPy) optimization package, specifically utilizing the Sequential Least Squares Quadratic Programming (SLSQP) optimizer. In order to use this method, we first defined functions to calculate the expected return and standard deviation (the square root of variance) of a portfolio given the inputs of an array of expected returns for the basket of securities, the covariance matrix of returns for the securities and an initial set of starting weights, or allocations, for each security (equal weightings were used as a starting point). In this manner, the function we set the optimizer to maximize is the ratio of expected return to expected volatility of the portfolio, ensuring that the result produces the maximum expected return, per unit of expected volatility. To capture the required constraints, we then specified in the optimizer that portfolio weights must sum to one, meaning all capital must be allocated, and that all weights must fall between zero and one, inclusive, ensuring that no “short sales” could occur. Finally, we specify in the optimizer that the vector of portfolio weights, or allocations, act as the decision variables to be altered in order to achieve our optimal result.

To test our MVO formulation we first generated a set of 50,000 simulated portfolio allocations for our basket of securities, not using the optimization function, and ran these allocations through the functions we defined to calculate expected portfolio returns and standard deviations, based on the historical data, plotting the resultant 50,000 data points by placing expected standard deviation ("expected volatility") on the x-axis and expected return on the y-axis. The ratio of the two, known as the Sharpe Ratio, is expressed chromatically on the side of the graphic:

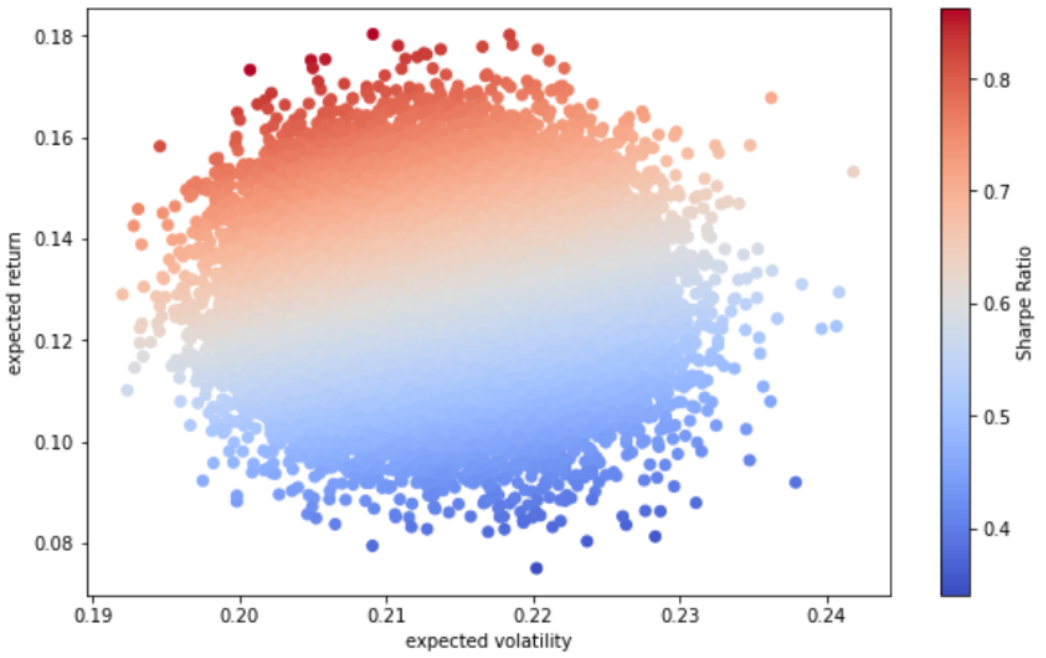


Figure 1. Scatterplot of 50,000 randomly weighted portfolios using historical returns and covariance matrix.

The next step was to overlay, on top of this scatterplot, the calculated Efficient Frontier of the MVO method. The Efficient Frontier is a concept that describes the shape of the parabolic line formed by connecting the points along a series of optimal portfolio allocations that are determined by varying the expected return of the portfolio and solving for the minimum expected volatility for a given level of return (Sharpe, 1994). By generating a loop for our optimizer, we calculated the efficient frontier for our historical data and overlaid it on our scatterplot:

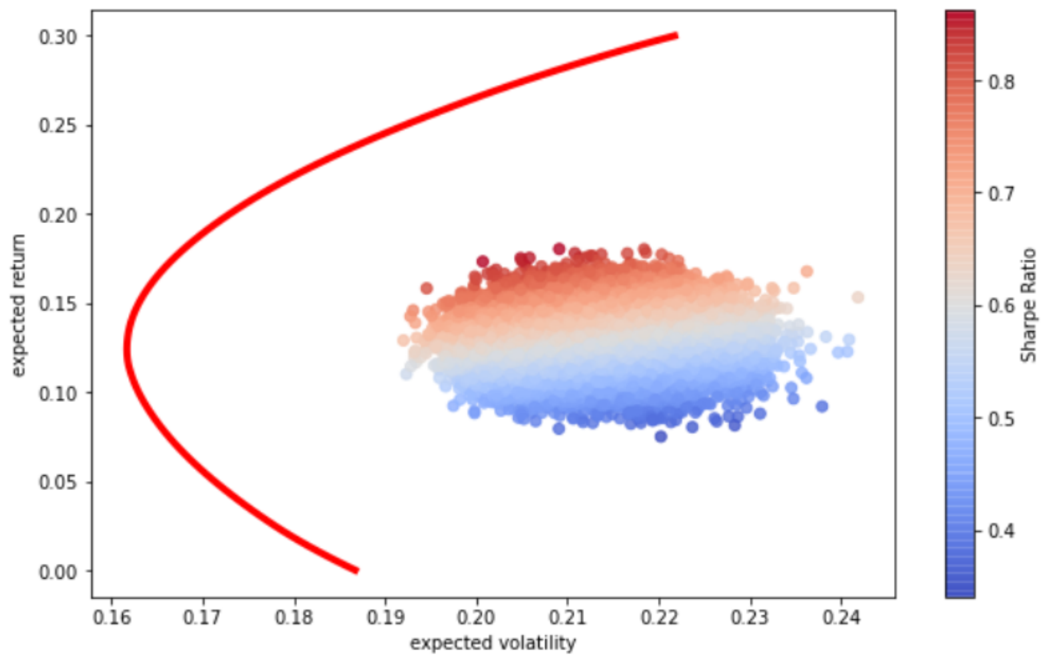


Figure 2. Efficient frontier overlaid on top of scatterplot of 50,000 randomly weighted portfolios.

When analyzing the visualization of the efficient frontier, plotted in red, we were immediately struck by the distance between the efficient frontier and the scatterplot of the randomized portfolios (Fig. 2). Having generated 50,000 randomized sets of portfolio weights, the gap between the two should not have been so dramatic. In theory, the efficient frontier should closely shadow the outer left edge of the randomized portfolios. Alerted to this discrepancy, we searched our code for errors. After an exhaustive process of examining our codebase, we determined that the source of error lay elsewhere.

In researching the issue, we learned about a set of pitfalls, known to some as “Markowitz’s Curse”, which are detailed in the Literature Review section of this paper. Specifically, quadratic portfolio optimization methods used to calculate the efficient frontier utilize covariance matrices; in a covariance matrix the condition number “…is the absolute value of the ratio between its maximal and minimal eigenvalues… [and at] some point the condition number is so high that numerical errors make the inverse matrix too unstable” (Lope de Prado, 2016). Investigating the issue further, we also learned that computational memory limitations within Python caused rounding errors of small numbers, impacting precision and further deteriorating results of quadratic optimization. An attempt was made to use Python packages such as Numerical Python (NumPy) to circumvent such computational limitations. Unfortunately, we determined that even Python’s extended precision, “long double”, numerical format was constrained by the operational limitations of the system being used; specifically, we could extend from 64-bit to 96-bit or 128-bit storage precision, but the additional memory was only used to “pad” figures with additional zero’s, useful when porting between systems with different operational limitations, but would not enhance the accuracy of our calculations (NumPy User Guide). As such, numerical errors inherent in covariance calculations would propagate, leading to incorrect conclusions and fooling the system into believing covariance between certain securities was zero or near-zero.

In a further attempt to find a method in which the extended precision we required for calculating quadratically optimized portfolios would be afforded, and recognizing the limitations of Python in this regard, we worked to rewrite our program in the programming language R. We hoped that R, which is based in C++, would be able to handle the extended precision required for our calculations. Unfortunately, R provided very similar results – we are left to surmise that the limitations of our hardware could not be easily overcome. Without a computer of x86 architecture at our disposal, we determined to seek other options for generating our portfolio optimizations.

While looking into the aforementioned issue we discovered the Hierarchical Risk Parity algorithm. This algorithm seemed to offer several advantages that improved on the deficiencies of Markowitz's MVO method, the first being that HRP does not require the calculation of the inverse of the covariance matrix. Additionally, HRP has been found to produce portfolios with better out-of-sample risk protection as compared to traditional portfolio optimization approaches (Kolanovic, Lau, Lee, & Krishnamachari, 2017). Given our focus on developing robust portfolio optimizations, HRP was deemed to offer strong potential.

Once again, our goal in applying HRP was to use Monte Carlo Simulations to generate N sets of simulated returns for our basket of securities and based on those returns generate N optimal portfolios. Out of the N optimal portfolios, each portfolio would then be tested across the N sets of simulated returns, whichever portfolio produced the minimum level of maximum regret (identified as maximum portfolio drawdown across simulations) would be considered the optimal portfolio for our purposes.

The HRP method is comprised of three major stages: tree clustering, quasi-diagonalization and recursive bisection. Each stage will be detailed in turn (Lopez de Prado, 2018, p. 224-227):

1. Tree Clustering – the beginning input is an N*x*N correlation matrix of the securities in our basket
    a. A distance measure for the correlation matrix is calculated, where ρ_{ij} is the correlation between the ith and jth securities
        i. Distance = $d_{ij} = \sqrt{\frac{1}{2}(1-\rho ij)}$
    b. Using the distance measure, the correlation matrix is recoded as a distance matrix, this distance matrix is then used to compute the Euclidean distance between any two column vectors where each column vector represents the correlations of a given security with the other securities in the basket
        i. Euclidean Distance = $d\tilde{}_{ij} = \sqrt{\sum_{n=1}^{N}\left(d_{n,1}-d_{n,j}\right)^2}$
    c. Next, columns of vectors are paired together so as to minimize the distance between columns. The distance between the paired columns is calculated and then the individual columns that composed the original pairing are dropped, leaving only the vector of distances for the paired columns. This procedure is applied recursively until N - 1 clusters are formed from the original dataset

2. Quasi-Diagonalization – at this stage of the process the rows and columns of the covariance matrix are reorganized to align the largest values along the diagonal of the matrix
    a. The benefit of this approach is that similar investments are placed together and dissimilar investments are placed further apart

3. Recursive Bisection – the weights of the portfolio are determined at this stage through an inverse-variance allocation algorithm; weights are split in inverse proportion to a subset's variance – when beginning this process with a diagonalized covariance matrix, the result is a minimum-variance portfolio allocation
    a. It is at this stage that the HRP method guarantees that $0 \leq w_i \leq 1$ and that the $\sum_{n=1}^{N} w_i = 1$; at each iteration the algorithm splits the weights received from the previous, higher level in the hierarchy, with the initial level representing the entire basket, thus ensuring the total weights sum to 1

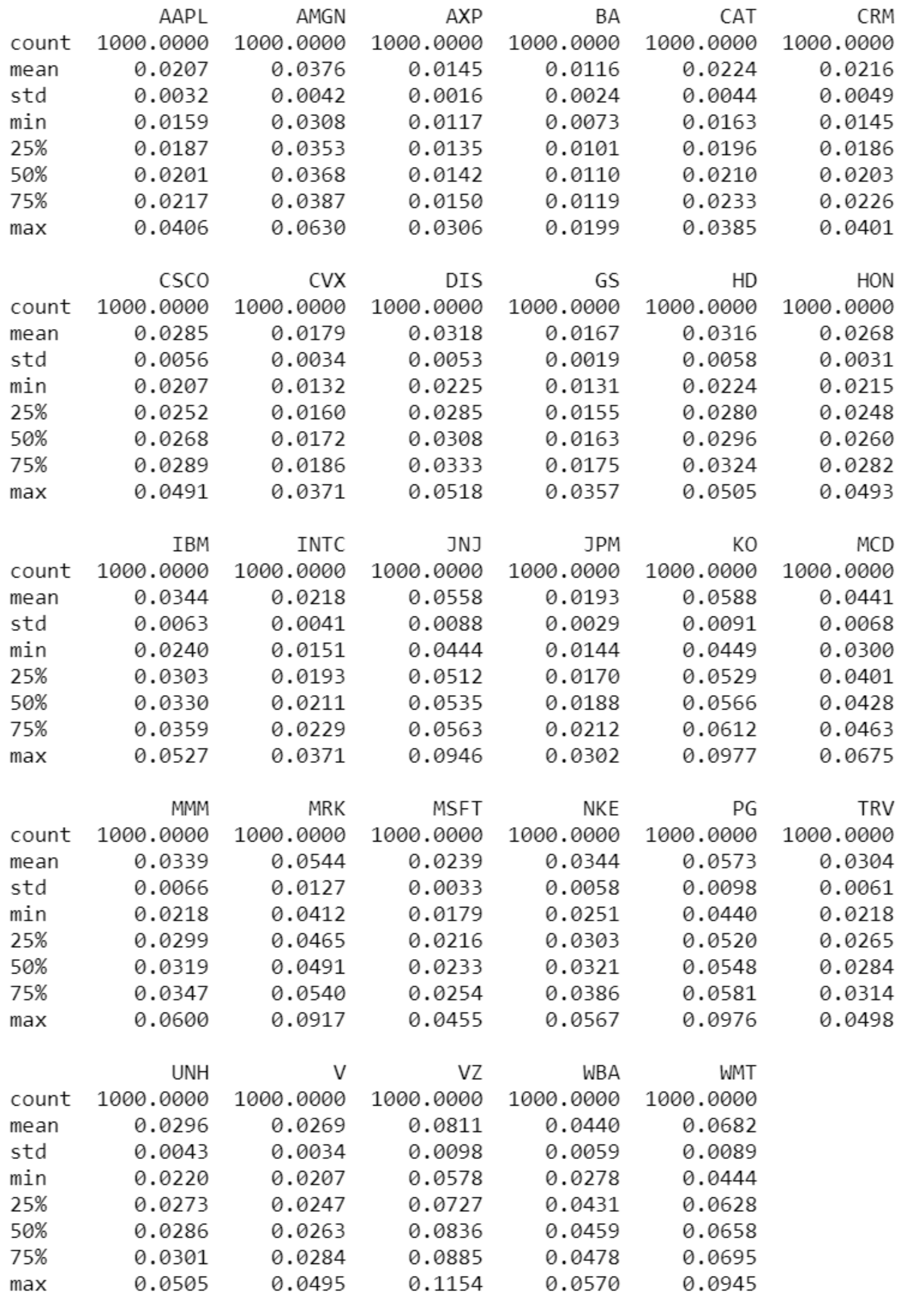
Following a validation of the HRP method on test data, in our Python development environment, we began the process of deploying the algorithm to produce the optimal portfolios required for our work on robust portfolio optimizations. The first step in this process was to choose a number for N, the number of sets of simulated returns we'd use to produce optimal portfolios and then against which we'd test each optimal portfolio. As a matter of computational practicality, we chose N = 1,000, generating 1,000 sets of simulated returns using SciPy's multivariate normal random variable package, which permits simulated data to be created using an overlayed covariance matrix, drawn from historical data, that also generates random-normal fluctuations in corresponding covariances for securities in the basket.

The 1,000 sets of securities returns, each comprised of 974 daily observations, were then used to generate 1,000 unique optimal portfolio allocations through the HRP method. For each unique portfolio, the expected portfolio return was next calculated across 1,000 sets of simulated returns. The result of this process was 1,000,000 data points, or 1,000 sets of 1,000 expected returns.

Finally, the minimum expected return was identified for each tuple of expected portfolio returns. From here, we identified the one portfolio with the minimum expected drawdown (or the least negative return), which represented our single optimal portfolio that minimized maximum regret from the original 1,000 portfolios generated.

Simulation and Results Analysis

To generate a final product, or recommendation, from our simulated securities baskets, we applied the HRP method to generate 1,000 optimized portfolios. In the optimized portfolios, each security contains a positive, finite value, between 0 and 1, the sum of which is 1, in conformance with our enforced constraints of non-negativity and total allocation of available assets. Unlike the MVO method, the distribution of allocations between securities, across the simulations, demonstrated stable patterns. The results of the HRP method demonstrate stark contrasts to the recommended allocations derived from the MVO method in which allocations to only five securities accounted for the entirety of available assets. The following descriptive statistics provide the distribution of HRP allocations across our 1,000 simulations:

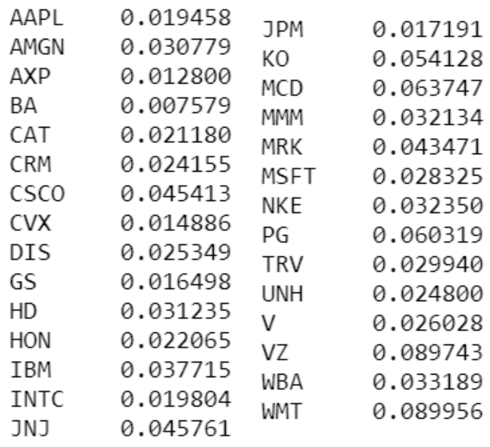


```
            AAPL       AMGN        AXP         BA        CAT        CRM
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0207     0.0376     0.0145     0.0116     0.0224     0.0216
std       0.0032     0.0042     0.0016     0.0024     0.0044     0.0049
min       0.0159     0.0308     0.0117     0.0073     0.0163     0.0145
25%       0.0187     0.0353     0.0135     0.0101     0.0196     0.0186
50%       0.0201     0.0368     0.0142     0.0110     0.0210     0.0203
75%       0.0217     0.0387     0.0150     0.0119     0.0233     0.0226
max       0.0406     0.0630     0.0306     0.0199     0.0385     0.0401

            CSCO        CVX        DIS         GS         HD        HON
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0285     0.0179     0.0318     0.0167     0.0316     0.0268
std       0.0056     0.0034     0.0053     0.0019     0.0058     0.0031
min       0.0207     0.0132     0.0225     0.0131     0.0224     0.0215
25%       0.0252     0.0160     0.0285     0.0155     0.0280     0.0248
50%       0.0268     0.0172     0.0308     0.0163     0.0296     0.0260
75%       0.0289     0.0186     0.0333     0.0175     0.0324     0.0282
max       0.0491     0.0371     0.0518     0.0357     0.0505     0.0493

             IBM       INTC        JNJ        JPM         KO        MCD
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0344     0.0218     0.0558     0.0193     0.0588     0.0441
std       0.0063     0.0041     0.0088     0.0029     0.0091     0.0068
min       0.0240     0.0151     0.0444     0.0144     0.0449     0.0300
25%       0.0303     0.0193     0.0512     0.0170     0.0529     0.0401
50%       0.0330     0.0211     0.0535     0.0188     0.0566     0.0428
75%       0.0359     0.0229     0.0563     0.0212     0.0612     0.0463
max       0.0527     0.0371     0.0946     0.0302     0.0977     0.0675

             MMM        MRK       MSFT        NKE         PG        TRV
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0339     0.0544     0.0239     0.0344     0.0573     0.0304
std       0.0066     0.0127     0.0033     0.0058     0.0098     0.0061
min       0.0218     0.0412     0.0179     0.0251     0.0440     0.0218
25%       0.0299     0.0465     0.0216     0.0303     0.0520     0.0265
50%       0.0319     0.0491     0.0233     0.0321     0.0548     0.0284
75%       0.0347     0.0540     0.0254     0.0386     0.0581     0.0314
max       0.0600     0.0917     0.0455     0.0567     0.0976     0.0498

             UNH          V         VZ        WBA        WMT
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0296     0.0269     0.0811     0.0440     0.0682
std       0.0043     0.0034     0.0098     0.0059     0.0089
min       0.0220     0.0207     0.0578     0.0278     0.0444
25%       0.0273     0.0247     0.0727     0.0431     0.0628
50%       0.0286     0.0263     0.0836     0.0459     0.0658
75%       0.0301     0.0284     0.0885     0.0478     0.0695
max       0.0505     0.0495     0.1154     0.0570     0.0945
```

Table 1. Descriptive, distributional statistics of 1,000 HRP-generated optimal portfolio allocations.

The final, recommended optimal portfolio, identified among our set of 1,000 simulated optimal portfolios as that which minimized the maximum regret (or drawdown) held the following allocations:

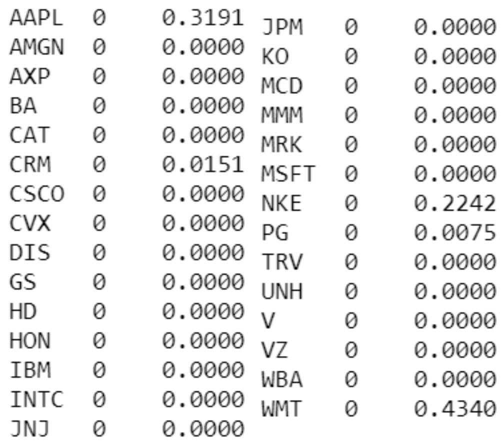
Optimal HRP Portfolio Allocations



```
AAPL    0.019458   JPM    0.017191
AMGN    0.030779   KO     0.054128
AXP     0.012800   MCD    0.063747
BA      0.007579   MMM    0.032134
CAT     0.021180   MRK    0.043471
CRM     0.024155   MSFT   0.028325
CSCO    0.045413   NKE    0.032350
CVX     0.014886   PG     0.060319
DIS     0.025349   TRV    0.029940
GS      0.016498   UNH    0.024800
HD      0.031235   V      0.026028
HON     0.022065   VZ     0.089743
IBM     0.037715   WBA    0.033189
INTC    0.019804   WMT    0.089956
JNJ     0.045761
```

Expected Portfolio Std. Dev. = .1878
Expected Portfolio Return = .1334
Sharpe Ratio = 0.71

Optimal MVO Portfolio Allocations



```
AAPL  0    0.3191   JPM   0    0.0000
AMGN  0    0.0000   KO    0    0.0000
AXP   0    0.0000   MCD   0    0.0000
BA    0    0.0000   MMM   0    0.0000
CAT   0    0.0000   MRK   0    0.0000
CRM   0    0.0151   MSFT  0    0.0000
CSCO  0    0.0000   NKE   0    0.2242
CVX   0    0.0000   PG    0    0.0075
DIS   0    0.0000   TRV   0    0.0000
GS    0    0.0000   UNH   0    0.0000
HD    0    0.0000   V     0    0.0000
HON   0    0.0000   VZ    0    0.0000
IBM   0    0.0000   WBA   0    0.0000
INTC  0    0.0000   WMT   0    0.4340
JNJ   0    0.0000
```

Expected Portfolio Std. Dev. = .2127
Expected Portfolio Return = .2808
Sharpe Ratio = 1.32

Table 2. Optimal portfolio allocations generated using minimax regret decision criterion.

For illustrative purposes, we have included the allocations recommended by the optimal MVO portfolio as a contrast to the HRP optimal portfolio. As can be seen, the HRP and MVO optimal portfolios have an expected portfolio standard deviation of .1878 and .2127, respectively, based on the historical data for our basket of securities. Expected returns for HRP and MVO are .1334 and .2808, respectively. The HRP portfolio has demonstrated a tradeoff of lower expected return for a lower expected variance. The MVO portfolio, due to the numerical precision errors detailed earlier, has recommended a portfolio allocation with significant inherent idiosyncratic, or company-specific, risks and high concentration in a small subset of securities. The expected Sharpe Ratio's of 0.71 and 1.32, respectively, compare to the current Sharpe Ratio of 0.36 for the Dow Jones Industrial Average (Morningstar, 2020).

In addition to the reduction in idiosyncratic risk provided by the HRP method, it can be noted that the method offers strong potential applications to a wide breadth of securities across a broad spectrum of asset classes. By clustering together securities with similar risk profiles and splitting allocations amongst them, the HRP method offers strong cross-asset potential. The results of the clustering of like-securities in our optimal portfolio are shown below:

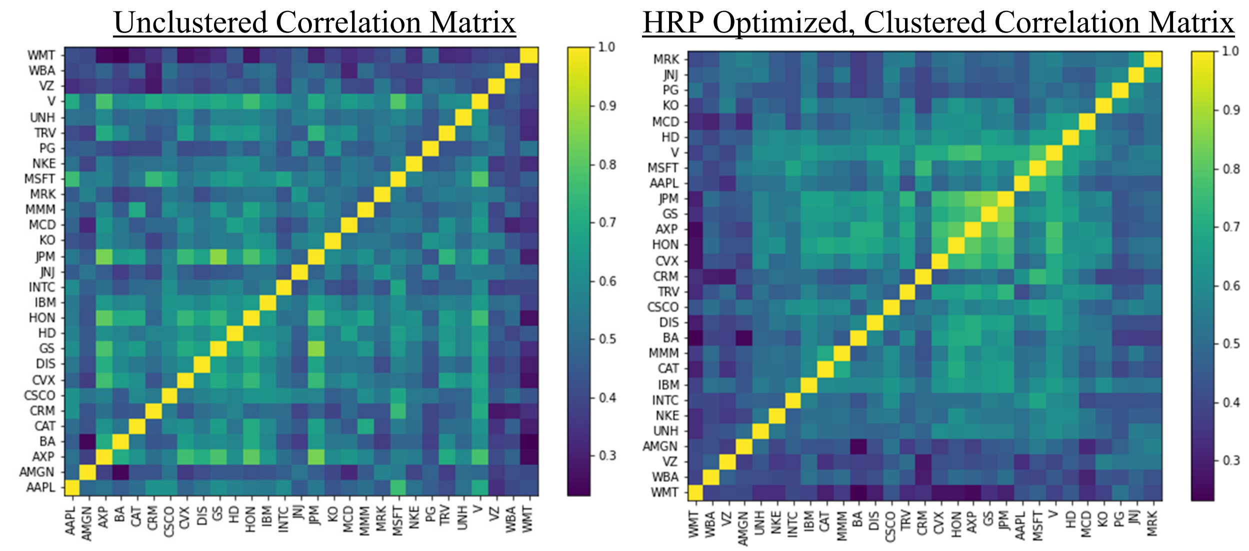


Figure 3. Correlation matrices of the basket of securities under analysis, before and after HRP clustering.

The optimized correlation matrix on the right displays, visually, the reorienting of the basket of securities to align the highest correlations for each sub-group along the diagonal of the matrix. The results are intuitively appealing to a market practitioner; for example, numerous clusters of securities operating in similar lines of business have been formed. Microsoft (MSFT), Apple (AAPL) and Visa (V), all of which are considered technology-focused firms, can be seen to form a cluster; banking and finance firms including American Express (AXP), Goldman Sachs (GS) and J.P. Morgan (JPM) are clustered; consumer products companies Coca-Cola (KO) and Procter & Gamble (PG) form another cluster; the same pattern can be seen for big-box retailers (Walmart and Walgreens Boots Alliance) and pharmaceutical companies (Merck and Johnson & Johnson). Furthermore, the HRP methodology ensures that securities with highly correlative characteristics are grouped together and allocations are split amongst the group before new clusters are formed, ensuring no significant over-representation of any one security or sub-group.

The results of our analysis indicate that the generation of minimum-variance portfolios using the HRP method, and selection of a single "robust" allocation using a minimax regret decision crtierion, offers strong potential gains, with respect to portfolio stability, for investors. We believe robust allocations, such as that recommended by our procedure outlined above, offer both better potential out-of-sample risk protection and reduction of idiosyncratic risk as compared to traditional MVO methods.

Conclusions and Future Study

In his 1952 paper, Markowitz remarked on the importance of diversification, "…a rule of behavior which does not imply the superiority of diversification must be rejected both as hypothesis and maxim" (Markowitz, 1952, p. 77). Following this line of thinking, we must turn a critical eye towards Markowitz's own method of mean-variance optimization. Though theoretically rigorous and elegant, MVO offers several drawbacks in application, notably: the requirement that correlation matrices be positive-definite for inversion; the susceptibility to errors during covariance matrix inversion due to propagation of numerical errors; and the finding that as correlations between investments increase, so to do recommended concentrations in fewer securities (Lopez de Prado, 2018). For these reasons, we must reject the temptation to ascribe superiority to MVO results based on naïve measures such as expected portfolio returns. Susceptibility to idiosyncratic risks and instability of results based on modest changes in inputs render MVO difficult to apply in practice.

Hierarchical Risk Parity, on the other hand, demonstrates appealing characteristics of stability of recommended allocations and robust performance in a broad range of scenarios. By imposing a hierarchy on sub-groups of securities, HRP produces both numerically and intuitively appealing results with respect to recommended allocations and expected risk profiles.

Our study detailed in this paper focused on a specific application of the HRP method for portfolio construction at a given point in time. In order to further the usefulness of this approach, however, we propose that additional analyses on the viability of applying HRP to time-series studies, engaged in active rebalancing of portfolios during specified intervals, be undertaken. Given our experience applying the HRP method to date, we are confident that such analyses are likely to find practical use of HRP methods for a range of inter- and intra-day investment activities.

References

Bailey, D., & Lopez de Prado, M. (2012). Balanced baskets: A new approach to trading and hedging risks. *Journal of Investment Strategies*, *1(4)*, 21-62.

Hilpisch, Y. (2018). *Python for finance: mastering data-driven finance* (2nd ed.). O'Reilly Media, Inc.

Jobson, J.D., & Korkie, R. (1981). Putting Markowitz theory to work. *Journal of Portfolio Management*, *7(4)*, 70-74. https://jpm.pm-research.com/content/7/4/70

Kolanovic, M., Lau, A., Lee, T., & Krishnamachari, R. (2017). Cross asset portfolios of tradable risk premia indices. Hierarchical risk parity: Enhancing returns at target volatility. White paper, Global Quantitative & Derivatives Strategy. J.P. Morgan.

Kolm, P., Tütüncü, R., & Fabozzi, F. (2013). 60 years of portfolio optimization: Practical challenges and current trends. *European Journal of Operational Research*, *234(2)*, 356-371. https://doi.org/10.1016/j.ejor.2013.10.060

Ledoit, O., & Wolf, R. (2003). Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, *10(5)*, 603-621.

Lopez de Prado, M. (2018). *Advances in financial machine learning*. Hoboken, NJ: John Wiley & Sons, Inc.

Lopez de Prado, M. (2016). Building diversified portfolios that outperform out of sample. *Journal of Portfolio Management*, *42(2)*, 56-69. https://jpm.pmresearch.com/content/42/4/59

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*, 77-91.

Michaud, R. (1998). *Efficient asset management: A practical guide to stock portfolio optimization and asset allocation*. Boston, MA: Harvard Business School Press.

Morningstar, Inc. (n.d.) Retrieved November 20, 2020, https://www.morningstar.com/etfs/arcx/dia/risk

NumPy User Guide: Data types. (n.d.). Retrieved from https://numpy.org/devdocs/user/basics.types.html

Sharpe, W.F. (1994). The Sharpe ratio. *Journal of Portfolio Management*, *21(1)*, 49-58. https://doi-org.turing.library.northwestern.edu/10.3905/jpm.1994.409501

What is the adjusted close? | Yahoo Help - SLN28256. (n.d.). Retrieved November 19, 2020, from https://help.yahoo.com/kb/SLN28256.html

Appendix A

Jupyter Notebook: Hierarchical Risk Parity Algorithm

A1. Import the required packages used in the notebook.

```
# import the required packages

import matplotlib.pyplot as mpl
import scipy
from scipy import stats
import scipy.cluster.hierarchy as sch
import numpy
import random, numpy as np
import pandas as pd
from datetime import datetime
from pandas_datareader import data as web
from datetime import datetime
from scipy.cluster.hierarchy import ClusterWarning
from warnings import simplefilter
import sys
import statsmodels.api as sm
```

A2. Capture the data and generate a dataframe for the basket of securities to be used for analysis.

```
# Create a dataframe of returns data for the selected securities

symbols =  ["AAPL", "AMGN", "AXP", "BA", "CAT", "CRM", "CSCO", "CVX", "DIS", "GS",
            "HD", "HON", "IBM", "INTC", "JNJ", "JPM", "KO", "MCD", "MMM", "MRK",
            "MSFT", "NKE", "PG", "TRV", "UNH", "V", "VZ", "WBA", "WMT"]

# Set the starting date for stock data collection
stock_startdate = '2017-01-01'

# Set the ending date for stock data collection
stock_enddate = '2020-11-13'

# Generate a dataframe to store the adjusted closing price of the stocks
df = pd.DataFrame()

# Store the adjusted close price of the stocks into the dataframe
for symbol in symbols:
    df[symbol] = web.DataReader(symbol, data_source='yahoo', start=stock_startdate, end=stock_enddate)['Adj Close']

# View the created dataframe
df
```

```
                AAPL        AMGN        AXP         BA          CAT         CRM         CSCO        CVX         DIS          GS          ...  MRK         MSFT
Date
2017-01-03      27.277639   134.517654  70.771912   145.533676  84.248009   70.540001   26.905218   99.251274   101.584358   226.374527  ...  53.849270   58.673244
2017-01-04      27.247108   136.427490  71.932106   147.063477  83.871544   72.800003   26.745281   99.226013   102.886719   227.836365  ...  53.831364   58.410725
2017-01-05      27.385668   136.525650  71.045456   147.146927  83.360641   72.790001   26.807476   98.796509   102.829262   226.140228  ...  53.813457   58.410725
2017-01-06      27.690971   139.916946  71.186943   147.508514  83.396454   73.800003   26.860790   98.400665   104.361458   229.495026  ...  53.956696   58.917015
2017-01-09      27.944603   141.755325  71.554817   146.785309  82.795929   73.959999   26.816360   97.558479   103.767731   227.611450  ...  54.699753   58.729496
```

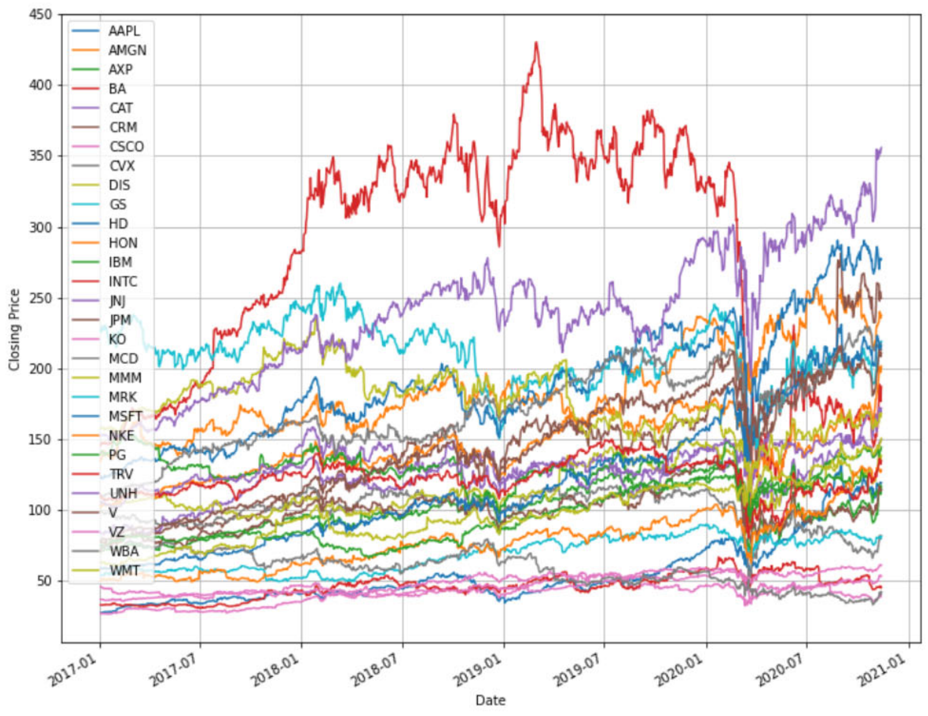
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-11-09	116.320000	233.414444	117.370003	179.360001	170.820007	259.279999	38.200001	78.248627	142.589996	214.929993	...	80.500000	217.819733
2020-11-10	115.970001	239.969910	115.949997	188.690002	172.300003	247.660004	38.720001	81.865410	142.110001	217.470001	...	81.110001	210.459000
2020-11-11	119.489998	239.066055	111.120003	182.149994	169.460007	254.179993	39.330002	81.264252	137.820007	218.050003	...	81.059998	215.984543
2020-11-12	119.209999	235.539993	111.099998	176.720001	169.130005	249.419998	38.669998	79.500206	135.520004	214.509995	...	79.860001	214.877441
2020-11-13	119.260002	237.360001	114.989998	187.110001	171.710007	249.509995	41.400002	81.825989	138.360001	219.080002	...	81.089996	215.944641

975 rows × 29 columns

A3. Visualize the data and engage in EDA to determine if log-returns are approximately normally distributed.

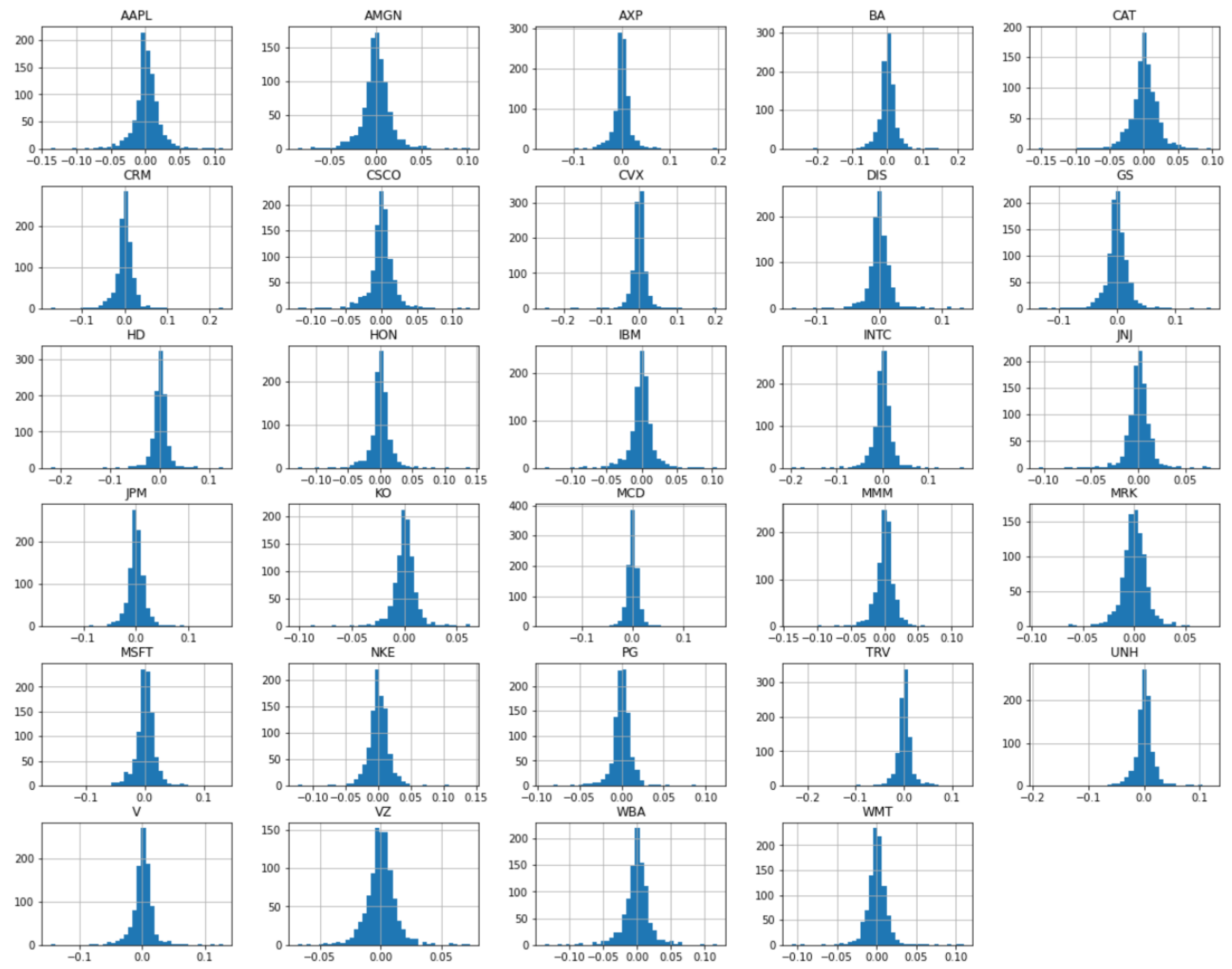
```
# Generate a plot displaying the daily adjusted closing price of the securities in our basket over the scrutiny period
ax = df.plot(grid = True,figsize=(12,10))
ax.set_xlabel("Date")
ax.set_ylabel("Closing Price")
```

```
Text(0, 0.5, 'Closing Price')
```



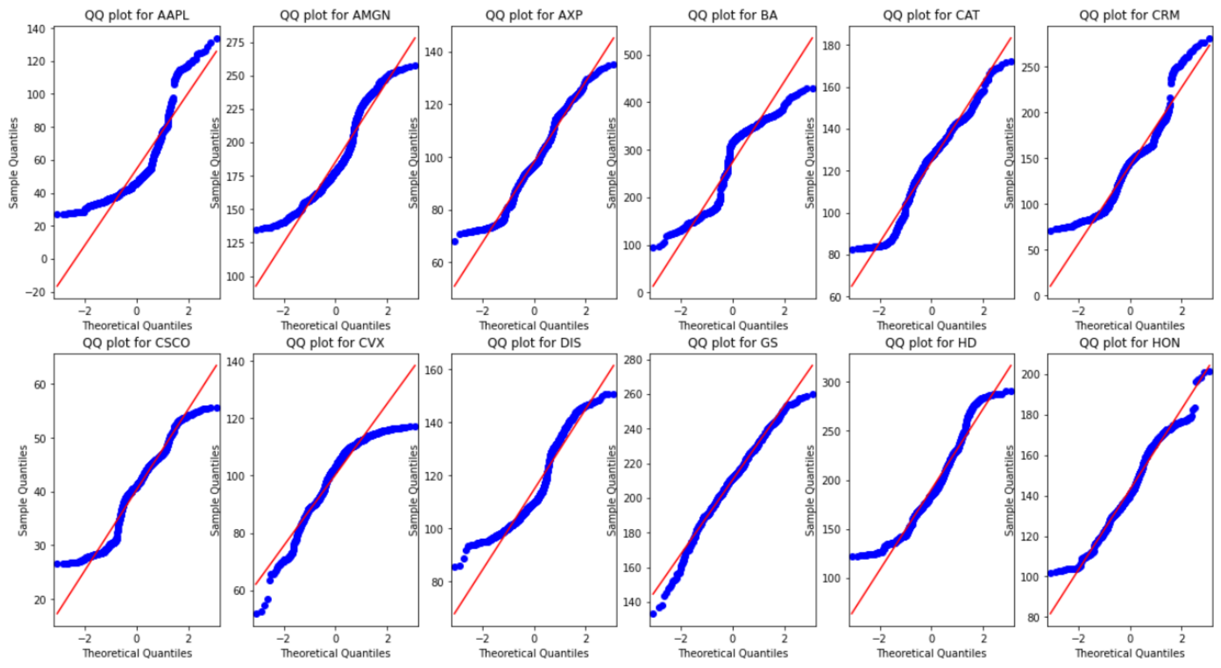
```
# Generate a matrix of daily log returns
rets = np.log(df / df.shift(1))
```

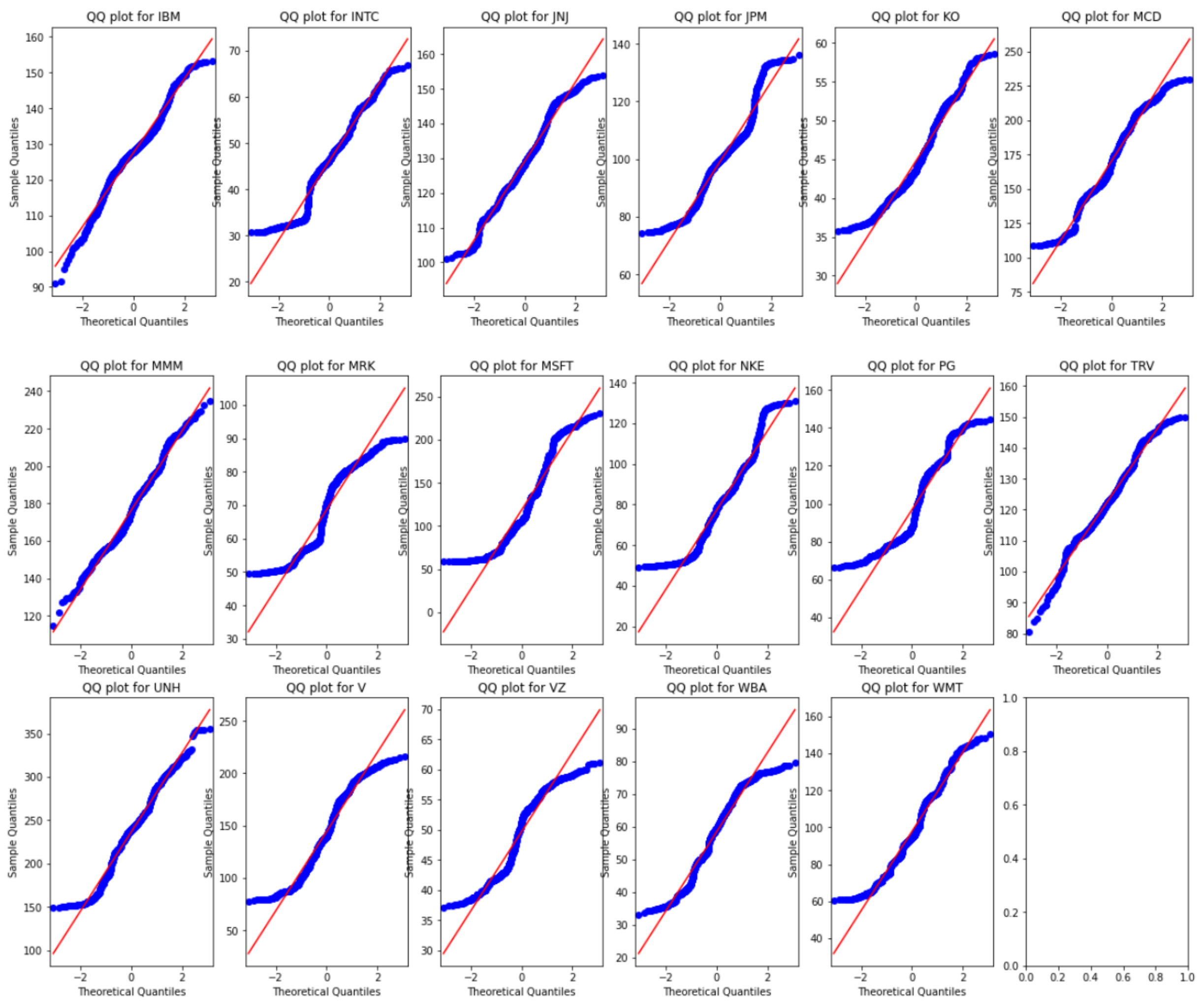
```
# Generate histogram of daily log returns for the securities in the basket
rets.hist(bins=40, figsize=(20, 16))
```



```
# Check if the data is normally distributed using qq-plots
# The plots appear to indicate that the data is approximately normal, with departures from normality occuring in the tails
# This generally conforms to anecdotal observations of security price movements
fig, ax = mpl.subplots(5, 6, figsize=(20, 28))
sm.qqplot(df['AAPL'], line='s', ax=ax[0,0])
ax[0,0].set_title('QQ plot for AAPL')
sm.qqplot(df["AMGN"], line='s', ax=ax[0,1])
ax[0,1].set_title('QQ plot for AMGN')
sm.qqplot(df["AXP"], line='s', ax=ax[0,2])
ax[0,2].set_title('QQ plot for AXP')
sm.qqplot(df["BA"], line='s', ax=ax[0,3])
ax[0,3].set_title('QQ plot for BA')
sm.qqplot(df["CAT"], line='s', ax=ax[0,4])
ax[0,4].set_title('QQ plot for CAT')
sm.qqplot(df["CRM"], line='s', ax=ax[0,5])
ax[0,5].set_title('QQ plot for CRM')

sm.qqplot(df["CSCO"], line='s', ax=ax[1,0])
ax[1,0].set_title('QQ plot for CSCO')
sm.qqplot(df["CVX"], line='s', ax=ax[1,1])
ax[1,1].set_title('QQ plot for CVX')
sm.qqplot(df["DIS"], line='s', ax=ax[1,2])
ax[1,2].set_title('QQ plot for DIS')
sm.qqplot(df["GS"], line='s', ax=ax[1,3])
ax[1,3].set_title('QQ plot for GS')
sm.qqplot(df["HD"], line='s', ax=ax[1,4])
ax[1,4].set_title('QQ plot for HD')
sm.qqplot(df["HON"], line='s', ax=ax[1,5])
ax[1,5].set_title('QQ plot for HON')
```

```
sm.qqplot(df["IBM"], line='s', ax=ax[2,0])
ax[2,0].set_title('QQ plot for IBM')
sm.qqplot(df["INTC"], line='s', ax=ax[2,1])
ax[2,1].set_title('QQ plot for INTC')
sm.qqplot(df["JNJ"], line='s', ax=ax[2,2])
ax[2,2].set_title('QQ plot for JNJ')
sm.qqplot(df["JPM"], line='s', ax=ax[2,3])
ax[2,3].set_title('QQ plot for JPM')
sm.qqplot(df["KO"], line='s', ax=ax[2,4])
ax[2,4].set_title('QQ plot for KO')
sm.qqplot(df["MCD"], line='s', ax=ax[2,5])
ax[2,5].set_title('QQ plot for MCD')

sm.qqplot(df["MMM"], line='s', ax=ax[3,0])
ax[3,0].set_title('QQ plot for MMM')
sm.qqplot(df["MRK"], line='s', ax=ax[3,1])
ax[3,1].set_title('QQ plot for MRK')
sm.qqplot(df["MSFT"], line='s', ax=ax[3,2])
ax[3,2].set_title('QQ plot for MSFT')
sm.qqplot(df["NKE"], line='s', ax=ax[3,3])
ax[3,3].set_title('QQ plot for NKE')
sm.qqplot(df["PG"], line='s', ax=ax[3,4])
ax[3,4].set_title('QQ plot for PG')
sm.qqplot(df["TRV"], line='s', ax=ax[3,5])
ax[3,5].set_title('QQ plot for TRV')

sm.qqplot(df["UNH"], line='s', ax=ax[4,0])
ax[4,0].set_title('QQ plot for UNH')
sm.qqplot(df["V"], line='s', ax=ax[4,1])
ax[4,1].set_title('QQ plot for V')
sm.qqplot(df["VZ"], line='s', ax=ax[4,2])
ax[4,2].set_title('QQ plot for VZ')
sm.qqplot(df["WBA"], line='s', ax=ax[4,3])
ax[4,3].set_title('QQ plot for WBA')
sm.qqplot(df["WMT"], line='s', ax=ax[4,4])
ax[4,4].set_title('QQ plot for WMT')

mpl.show()
```





A4. Define the functions that will compose the Hierarchical Risk Parity algorithm (Lopez de Prado, 2018).

```
# Define function to compute the inverse variance portfolio

def getIVP(cov, **kargs):
    ivp = 1./np.diag(cov)
    ivp/=ivp.sum()
    return ivp
```

```
# Define function to compute the variance per cluster

def getClusterVar(cov, cItems):
    cov_=cov.loc[cItems, cItems] # matrix slice
    w_=getIVP(cov_).reshape(-1,1)
    cVar=np.dot(np.dot(w_.T,cov_),w_)[0,0]
    return cVar
```

```
# Define function to sort clustered items by Euclidean distance

def getQuasiDiag(link):
    link = link.astype(int)
    sortIx = pd.Series([link[-1,0], link[-1,1]])
    numItems = link[-1,3] # number of original items
    while sortIx.max() >= numItems:
        sortIx.index = range(0, sortIx.shape[0]*2, 2) # make space
        df0 = sortIx[sortIx>=numItems] # find clusters
        i = df0.index; j = df0.values-numItems
        sortIx[i]=link[j,0] # item 1
        df0 = pd.Series(link[j,1], index= i+1)
        sortIx = sortIx.append(df0) # item 2
        sortIx = sortIx.sort_index() # re-sort
        sortIx.index = range(sortIx.shape[0]) # re-index
    return sortIx.tolist()
```

```
# Define function to compute HRP allocation using recursive bi-section

def getRecBipart(cov, sortIx):
    w = pd.Series(1, index=sortIx)
    cItems = [sortIx] # initialize all items in one cluster
    while len(cItems) > 0:
        cItems = [i[j:k] for i in cItems for j,k in ((0, len(i)//2), (len(i)//2, len(i))) if len(i) > 1] # bi-section
        for i in range(0, len(cItems), 2): # parse in pairs
            cItems0 = cItems[i] # cluster 1
            cItems1 = cItems[i+1] # cluster 2
            cVar0 = getClusterVar(cov, cItems0)
            cVar1 = getClusterVar(cov, cItems1)
            alpha = 1-cVar0/(cVar0+cVar1)
            w[cItems0]*=alpha # weight 1
            w[cItems1]*=1-alpha # weight 2
    return w
```

```
# Define function to generate a distance matrix based on correlation, where 0<=d[i,j]<=1
# This is a proper distance metric

def correlDist(corr):
    dist = ((1-corr)/2.)**.5 # distance matrix
    return dist
```

```
# Define function to generate a heatmap of the correlation matrix

def plotCorrMatrix(path, corr, labels=None):
    if labels is None:labels = []
    mpl.figure(figsize = (9, 7))
    mpl.pcolor(corr)
    mpl.colorbar()
    mpl.yticks(np.arange(.5, corr.shape[0]+.5), labels)
    mpl.xticks(np.arange(.5, corr.shape[0]+.5), labels, rotation = 90)
    mpl.savefig(path)
    mpl.clf(); mpl.close() # reset pylab
    return
```

```
# Define a function to direct the HRP process (indidvidual steps outlined above) and return a list of portfolio allocations

def getHRP(cov, corr):
    # Construct a hierarchical portfolio
    dist = correlDist(corr)
    link = sch.linkage(dist, 'single')
    sortIx = getQuasiDiag(link)
    sortIx = corr.index[sortIx].tolist()
    hrp = getRecBipart(cov, sortIx)
    return hrp.sort_index()
```

A5. Test the HRP algorithm on the historical data and test that the computed distance matrix is suitable for use in SciPy package.

```
# Test the getHRP function on the original dataset (the ouput is the HRP portfolio, sorted by name, not clusters)
getHRP(rets.cov(), rets.corr())
```

```
AAPL    0.020487
AMGN    0.036827
AXP     0.016234
BA      0.010150
CAT     0.021030
CRM     0.017697
CSCO    0.041283
CVX     0.017680
DIS     0.043422
GS      0.018915
HD      0.028640
HON     0.030124
IBM     0.033478
INTC    0.022130
JNJ     0.052233
JPM     0.023432
KO      0.051969
MCD     0.045735
MMM     0.030859
MRK     0.046997
MSFT    0.024326
NKE     0.029517
PG      0.055727
TRV     0.026953
UNH     0.027423
V       0.027995
VZ      0.086819
WBA     0.047148
WMT     0.064771
dtype: float64
```

```
# We note that Scipy presents a cluster warning; we use the following code to validate that our data
# for the distance matrix used in the HRP algorithm is a valid distance matrix recognized by Scipy.
# Given that this check returns "true" we will add code to ignore the warning posted so it does not recur.
scipy.spatial.distance.is_valid_dm(correlDist(rets.corr()))
```

```
True
```

```
simplefilter("ignore", ClusterWarning)
```

A6. Generate simulated returns data using the SciPy multivariate_normal function to permit the overlay of a covariance matrix. Then generate a loop to calculate the HRP “optimal” portfolio allocations for each set of simulated returns.

```
# Use the vectors of returns and and the covariance matrix for the historical data to generate a new, randomized
# set of vectors based on the normal distribution; 1000 sets of simulated data, each matching the length of the original
# dataframe are produced in this step

rand_norm_rets = scipy.stats.multivariate_normal.rvs(mean = rets.mean(),
                                                     cov = rets.cov(),
                                                     size = (len(df)*1000),
                                                     random_state = 12345)
```

```
# Generate a dataframe from the simulated data and then check the length of the new dataframe to confirm it is equal to
# the number of original observations in the 'df' frame multiplied by the size of the simulated observations,
# which is currently set to 1000

rand_norm_rets_df = pd.DataFrame(rand_norm_rets, columns = symbols)
len(rand_norm_rets_df)
```

```
975000
```

```
# Generate an empty list; then create a loop for the getHRP function to run on every slice of simulated data. Append the
# results of each loop to the empty list. This will result in a list of lists.

HRP_ports = []

for x in range(1,1001):
    HRP_rand = getHRP(rand_norm_rets_df[(x*len(df)-len(df)):(len(df)*x)].cov(),
                      rand_norm_rets_df[(x*len(df)-len(df)):(len(df)*x)].corr())
    HRP_ports.append(HRP_rand)
```

A7. Define functions to calculate expected portfolio returns and volatility.

```
# Define the function for calculating expected portfolio return; To do so, expected, or mean, daily log returns are
# multiplied by the average number of trading days in a year, 252
def port_ret(weights, returnsdata):
    return np.sum(returnsdata.mean() * weights) * 252

# Define a function for calculating expected portolio volatility; Note: the dot function is used for matrix multiplication
def port_vol(weights, returnsdata):
    return np.sqrt(np.dot(weights.T, np.dot(returnsdata.cov() * 252, weights)))
```

A8. Calculate expected returns for each HRP portfolio, across each of the 1,000 sets of simulated returns. Save the returns to tuples. Then determine the location of the “optimal” portfolio by applying the minimax regret decision criterion (i.e. the portfolio with the maximum of minimum expected returns, across the simulations).

```
# For each set of weights generated and stored in the HRP_ports list, determine the expected return under every simulated
# set of returns scenarios. Each of the 1000 portfolios generated using the HRP alogirthm will be tested against 1000
# returns scnearios and each will be saved to a tuple in the HRP_tups_returns empty list. This list will contain 1000
# tuples, each tuple containing 1000 expected return data points for a given set of portfolio weights
HRP_tups_returns = []

for portfolio in HRP_ports:
    HRP_ports_returns = []
    for simulated_slice in range(1,1001):
        current_return = port_ret(portfolio, rand_norm_rets_df[(simulated_slice*len(df)-len(df)):(len(df)*simulated_slice)])
        HRP_ports_returns.append(current_return)
    HRP_tups_returns.append(tuple(HRP_ports_returns))
```

```
# Transform the tuples into arrays for easier numerical operations
HRP_array_returns = np.array(HRP_tups_returns)
```

```
# Loop through each array and find the minimum value in each array
min_returns_list = []

for rets in HRP_array_returns:
    min_returns_list.append(rets.min())
```

```
# Determine the index location of the MAX of these MIN returns
min_returns_list.index(max(min_returns_list))
```

```
538
```

```
# Display the figure; this figure representes the highest MINIMUM return across all of the simulated portfolios
# In other words, this data points represents the minimization of maximum regret experienced across the simulations
min_returns_list[min_returns_list.index(max(min_returns_list))]
```

```
-0.13207104540798617
```

A9. Display the allocations of the optimal portfolio, as identified by the minimax regret criterion. Then probe distributional characteristics of this portfolio.

```
# Display the portfolio weights for the identified optimal portfolio
HRP_ports[min_returns_list.index(max(min_returns_list))]
```

```
AAPL    0.019458
AMGN    0.030779
AXP     0.012800
BA      0.007579
CAT     0.021180
CRM     0.024155
CSCO    0.045413
CVX     0.014886
DIS     0.025349
GS      0.016498
HD      0.031235
HON     0.022065
IBM     0.037715
INTC    0.019804
JNJ     0.045761
JPM     0.017191
KO      0.054128
MCD     0.063747
MMM     0.032134
MRK     0.043471
MSFT    0.028325
NKE     0.032350
PG      0.060319
TRV     0.029940
UNH     0.024800
V       0.026028
VZ      0.089743
WBA     0.033189
WMT     0.089956
dtype: float64
```

```
# This is the mean of ALL simulated returns for the weights that comprise the 'optimal' portfolio
np.mean(HRP_tups_returns[min_returns_list.index(max(min_returns_list))])
```

```
0.12809077407233166
```

```
# Calculate the expected portfolio volatility for the identified optimal portfolio - based on the simulated returns that
# generated the optimal portfolio

port_vol(HRP_ports[min_returns_list.index(max(min_returns_list))],
         rand_norm_rets_df[(min_returns_list.index(max(min_returns_list))
                           *len(df)-len(df)):(len(df)*min_returns_list.index(max(min_returns_list)))])
```

```
0.19389576179862342
```

```
# Calculate the expected portfolio volatility for the identified optimal portfolio - based on the historical returns data
port_vol(HRP_ports[min_returns_list.index(max(min_returns_list))], np.log(df / df.shift(1)))
```

```
0.18776149804636064
```

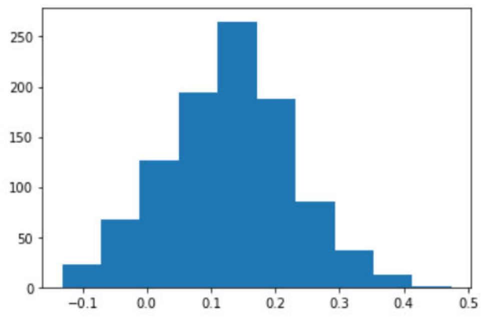
```
# Calculate the expected portfolio return for the identified optimal portfolio - based on the historical returns data
port_ret(HRP_ports[min_returns_list.index(max(min_returns_list))], np.log(df / df.shift(1)))
```

```
0.1333831946214797
```

A10. Generate a histogram of the expected returns of the optimal portfolio across the 1,000 simulations.

```
# Generate a historgram of the expected returns from the optimal portfolio
mpl.hist(HRP_tups_returns[min_returns_list.index(max(min_returns_list))])
```

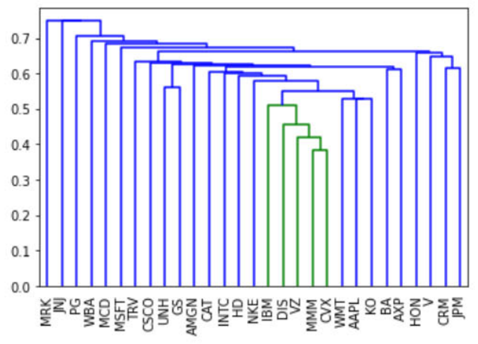
```
(array([ 23.,  67., 126., 194., 265., 188.,  86.,  37.,  13.,   1.]),
 array([-0.13207105, -0.0714549 , -0.01083875,  0.0497774 ,  0.11039355,
         0.1710097 ,  0.23162585,  0.292242  ,  0.35285815,  0.4134743 ,
         0.47409045]),
 <a list of 10 Patch objects>)
```



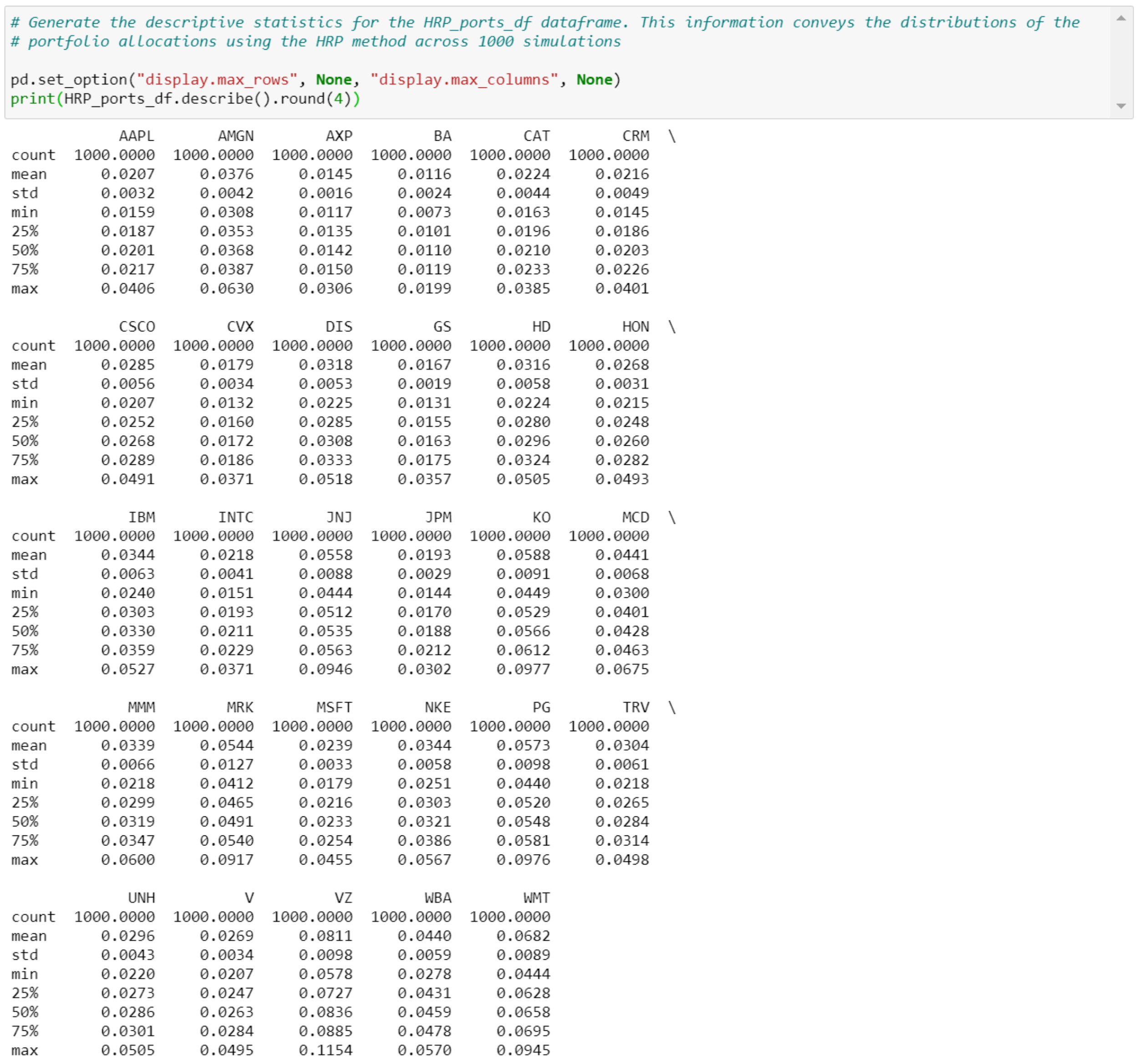
A11. Generate the correlation heatmaps of the pre- and post-clustered basket of securities. Also generate a dendrogram.

```
# Generate the correlation heat maps, before and after the HRP algorithm is run, for the optimal portfolio
# Both the original heat map and the heat map of the clustered data will be saved to your current working directory

# 1) find the correlation matrix of the optimal portfolio, based on the simulated data, before any transofrmations occur;
# This is accomplished by taking a slice of the simulated data over the range determined by the location of the optimal
# portfolio identified above
opt_corr = rand_norm_rets_df[(min_returns_list.index(max(min_returns_list))*len(df)-len(df)):
                   (len(df)*min_returns_list.index(max(min_returns_list)))].corr()

# 2) compute and plot correl matrix
plotCorrMatrix('HRP_corr.png', opt_corr, labels=opt_corr.columns)

# 3) cluster
dist = correlDist(opt_corr)
link = sch.linkage(dist, 'single')
sortIx = getQuasiDiag(link)
sortIx = opt_corr.index[sortIx].tolist() # recover labels
df0 = opt_corr.loc[sortIx, sortIx] # reorder

# 4) compute the clustered correl matrix
plotCorrMatrix('HRP_corr_opt.png', df0, labels=df0.columns)
```

```
# Produce and view the dendrogram
dn = sch.dendrogram(link, labels=sortIx, leaf_rotation=90, get_leaves = True)
```



A12. Generate descriptive statistics for the distribution of allocations across securities for the 1,000 HRP portfolios generated.



```
# Generate the descriptive statistics for the HRP_ports_df dataframe. This information conveys the distributions of the
# portfolio allocations using the HRP method across 1000 simulations

pd.set_option("display.max_rows", None, "display.max_columns", None)
print(HRP_ports_df.describe().round(4))
```

```
            AAPL       AMGN        AXP         BA        CAT        CRM  \
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0207     0.0376     0.0145     0.0116     0.0224     0.0216
std       0.0032     0.0042     0.0016     0.0024     0.0044     0.0049
min       0.0159     0.0308     0.0117     0.0073     0.0163     0.0145
25%       0.0187     0.0353     0.0135     0.0101     0.0196     0.0186
50%       0.0201     0.0368     0.0142     0.0110     0.0210     0.0203
75%       0.0217     0.0387     0.0150     0.0119     0.0233     0.0226
max       0.0406     0.0630     0.0306     0.0199     0.0385     0.0401

            CSCO        CVX        DIS         GS         HD        HON  \
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0285     0.0179     0.0318     0.0167     0.0316     0.0268
std       0.0056     0.0034     0.0053     0.0019     0.0058     0.0031
min       0.0207     0.0132     0.0225     0.0131     0.0224     0.0215
25%       0.0252     0.0160     0.0285     0.0155     0.0280     0.0248
50%       0.0268     0.0172     0.0308     0.0163     0.0296     0.0260
75%       0.0289     0.0186     0.0333     0.0175     0.0324     0.0282
max       0.0491     0.0371     0.0518     0.0357     0.0505     0.0493

             IBM       INTC        JNJ        JPM         KO        MCD  \
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0344     0.0218     0.0558     0.0193     0.0588     0.0441
std       0.0063     0.0041     0.0088     0.0029     0.0091     0.0068
min       0.0240     0.0151     0.0444     0.0144     0.0449     0.0300
25%       0.0303     0.0193     0.0512     0.0170     0.0529     0.0401
50%       0.0330     0.0211     0.0535     0.0188     0.0566     0.0428
75%       0.0359     0.0229     0.0563     0.0212     0.0612     0.0463
max       0.0527     0.0371     0.0946     0.0302     0.0977     0.0675

             MMM        MRK       MSFT        NKE         PG        TRV  \
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0339     0.0544     0.0239     0.0344     0.0573     0.0304
std       0.0066     0.0127     0.0033     0.0058     0.0098     0.0061
min       0.0218     0.0412     0.0179     0.0251     0.0440     0.0218
25%       0.0299     0.0465     0.0216     0.0303     0.0520     0.0265
50%       0.0319     0.0491     0.0233     0.0321     0.0548     0.0284
75%       0.0347     0.0540     0.0254     0.0386     0.0581     0.0314
max       0.0600     0.0917     0.0455     0.0567     0.0976     0.0498

             UNH          V         VZ        WBA        WMT
count  1000.0000  1000.0000  1000.0000  1000.0000  1000.0000
mean      0.0296     0.0269     0.0811     0.0440     0.0682
std       0.0043     0.0034     0.0098     0.0059     0.0089
min       0.0220     0.0207     0.0578     0.0278     0.0444
25%       0.0273     0.0247     0.0727     0.0431     0.0628
50%       0.0286     0.0263     0.0836     0.0459     0.0658
75%       0.0301     0.0284     0.0885     0.0478     0.0695
max       0.0505     0.0495     0.1154     0.0570     0.0945
```

Appendix B

Jupyter Notebook: Mean-Variance Optimization

Appendix B Note: Areas of functional overlap shared with Appendix A, such as harvesting data and generating dataframes for processing, have been omitted to avoid redundancy. Only key functions in the MVO process are displayed in this appendix.

B1. Define functions for calculating expected portfolio return and volatility. Then generate 50,000 randomly weighted portfolios and calculate expected returns and volatility for each (Hilpisch, 2018).

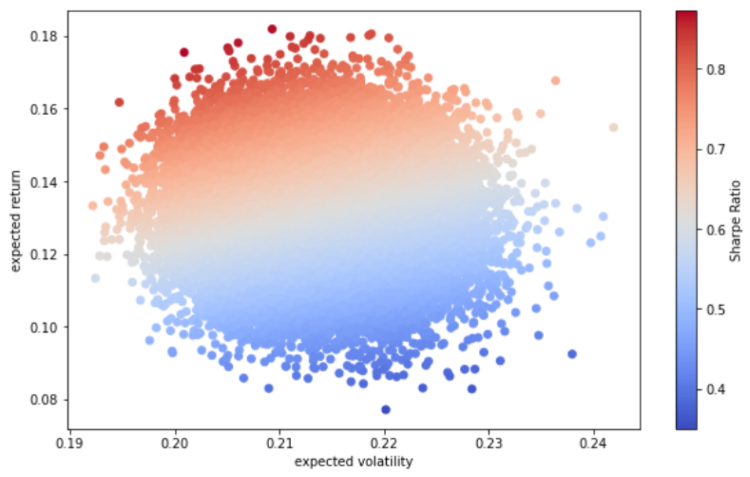
```python
# Define a function for portfolio return; annualize the daily log returns by taking the mean and multipluing by 252
# average trading days in a year
def port_ret(weights, returnsdata):
    return np.sum(returnsdata.mean() * weights) * 252

# Define a function for portolio volatility; Note: the dot function is used for matrix multiplication
def port_vol(weights, returnsdata):
    return np.sqrt(np.dot(weights.T, np.dot(returnsdata.cov() * 252, weights)))
```

```python
# Initialize empty lists that will evnetually contain portfolio return and volatility data for the simulated portfolios
p_returns = []
p_vols = []
p_weights = []

# Set the seed for the Numpy random number generator to ensure reproducible results
np.random.seed(seed=1234)

# Generate a loop to 1) assign random weights to the assets in the 'rets' dataframe and then ensure the sum of the weights
# is 1 by dividing each weight by the sum of the weights; 2) pass the simulated, random weights to the above defined frame
# to generate return and volatility information for this simulated vector of weights. Append the new data to the empty lists
for p in range(50000):
    weights = np.random.random(num_assets)
    weights /= np.sum(weights)
    p_returns.append(port_ret(weights, rets))
    p_vols.append(port_vol(weights, rets))
    p_weights.append(weights)

# Convert each of the filled lists to a numpy array
p_returns = np.array(p_returns)
p_vols = np.array(p_vols)
```

B2. Generate a scatterplot of the 50,000 randomly weighted portfolios.

```python
# Plot the results of this initial work
# For simplicity, the risk-free rate needed for Sharpe ratio is currently considered equal to zero
# Given the historically low yield on short dated treasuries and other govt. instruments, this is a reasonable assumption

plt.figure(figsize=(10, 6))
plt.scatter(p_vols, p_returns, c=p_returns/p_vols, marker='o', cmap='coolwarm')
plt.xlabel('expected volatility'),
plt.ylabel('expected return')
plt.colorbar(label='Sharpe Ratio')
```

```
<matplotlib.colorbar.Colorbar at 0x26c5755e448>
```



B3. Define a function to calculate the ratio of expected portfolio return to expected portfolio volatility. Use the SLSQP quadratic optimization package to engage in MVO.

```python
# import SciPy's optimization package
import scipy.optimize as sco
```

```python
# Define a new function for the sharpe ratio to be minimized in our optimization
# Because the scipy optimize function is primarily setup to minimize, we add a negative sign to the function below
# defining the Sharpe ratio. In this way, we can now ask scipy to minmize the resultant negative value, which is
# functionally the same as finding the maximum sharpe ratio when optimization is applied

def sharpe_function_to_minimize(weights, returnsdata):
    return -port_ret(weights, returnsdata) / port_vol(weights, returnsdata)

# set constraint - which will be added to optimization function - requiring portfolio weights to sum to 1
constraint_1 = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

# Set boundaries - which will be added to the optimization function - for the parameters requiring
# all weights fall between 0 and 1
boundaries = tuple((0, 1) for x in range(num_assets))

# Create a vector of equal weights for initialization purposes
equal_weights = np.array(num_assets * [1. / num_assets,])
```

B4. Calculate the optimal portfolio, using MVO, for the historical data. This portfolio has the highest expected Sharpe ratio among all alternatives.

```python
# Initialize the optimization using the SLSQP (Sequential Least Squares Programming)
# Documentation: http://www.pyopt.org/reference/optimizers.slsqp.html

port_opt = sco.minimize(fun = sharpe_function_to_minimize, x0 = equal_weights, args = rets, method='SLSQP',
                        bounds=boundaries, constraints=constraint_1)

# Show the output of the SciPy optimization
port_opt
```

```
      fun: -1.354085543392713
      jac: array([-2.47657299e-05,  3.03608328e-01,  6.21221855e-01,  1.11386460e+00,
        2.63867214e-01, -1.00672245e-04,  7.10631147e-01,  1.24681616e+00,
        5.90875983e-01,  1.26079175e+00,  2.20937058e-01,  1.61669552e-01,
        1.19472423e+00,  1.05776010e+00,  3.12800497e-01,  6.29693955e-01,
        2.45492250e-01,  4.84158248e-02,  8.26779723e-01,  2.43059680e-01,
        7.03901052e-04, -2.33009458e-04,  6.71523809e-03,  6.73378453e-01,
        1.43956035e-01,  6.57524765e-02,  1.93789631e-01,  1.55914201e+00,
       -1.95905566e-04])
  message: 'Optimization terminated successfully.'
     nfev: 219
      nit: 7
     njev: 7
   status: 0
  success: True
        x: array([3.37792129e-01, 3.13619031e-16, 4.57601079e-16, 0.00000000e+00,
       7.54604712e-16, 5.68799403e-02, 0.00000000e+00, 0.00000000e+00,
       5.27667645e-17, 0.00000000e+00, 5.96046921e-16, 3.08604596e-16,
       0.00000000e+00, 0.00000000e+00, 5.23927147e-16, 1.47654783e-16,
       2.83850905e-16, 9.26118719e-16, 2.03050738e-16, 5.83151691e-16,
       1.49301011e-01, 7.29444985e-02, 8.43997181e-16, 0.00000000e+00,
       2.53621993e-16, 3.93267233e-16, 1.02083733e-15, 0.00000000e+00,
       3.83082421e-01])
```

```python
# Print portfolio weights
print(port_opt['x'].round(3))

# Print the optimized portfolio expected return
print(port_ret(port_opt['x'], rets).round(5))

# Print the optimized portfolio expected volatility
print(port_vol(port_opt['x'], rets).round(5))

# Print the optimized portfolio Sharpe Ratio
print(port_ret(port_opt['x'], rets) / port_vol(port_opt['x'], rets))
```

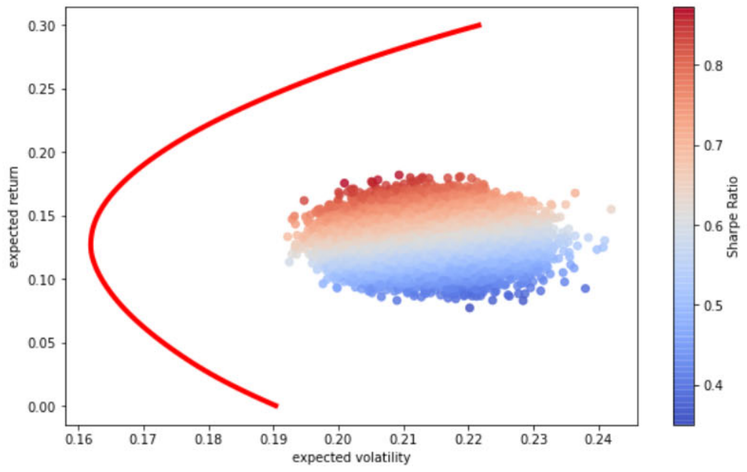
```
[0.338 0.    0.    0.    0.    0.057 0.    0.    0.    0.    0.    0.
 0.    0.    0.    0.    0.    0.    0.    0.    0.149 0.073 0.    0.
 0.    0.    0.    0.    0.383]
0.30216
0.22314
1.354085543392713
```

B5. Generate a loop for the SLSQP optimizer in order to calculate the efficient frontier. Plot the frontier overlayed on the 50,000 randomly weighted portfolios.

```python
# Now we go about calculating the efficient frontier, this is done in the first constraint in "constraint_2" by forcing
# the portfolio return generated by the optimization to be equal to zero after subtracting the target return. By optimizing
# for the minimized volatility under this condition, we are finding the best return/risk ratio for each given level of
# return specific by the "trets" variable below

constraint_2 = ({'type': 'eq', 'fun': lambda x: port_ret(x, rets) - tret},
               {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

boundaries_2 = tuple((0,1) for x in weights)

trets = np.linspace(0, .30, 100)
tvols = []

for tret in trets:
    res = sco.minimize(fun = port_vol, x0 = equal_weights, args = rets, method='SLSQP',
                       bounds = boundaries_2, constraints = constraint_2)
    tvols.append(res['fun'])
tvols = np.array(tvols)
```

```
# Plot the resultant optimizations that generated the efficient frontier, overlayed on top of the randomized portfolios
# generated earlier

plt.figure(figsize=(10,6))
plt.scatter(p_vols, p_returns, c = p_returns/p_vols, marker = 'o', alpha = 0.8, cmap = 'coolwarm')
plt.plot(tvols, trets, 'r', lw=4.0)
plt.xlabel('expected volatility'),
plt.ylabel('expected return')
plt.colorbar(label='Sharpe Ratio')
```

```
<matplotlib.colorbar.Colorbar at 0x26c58b5e6c8>
```



B6. Generate simulated data using the SciPy multivariate_normal function.

```
# Use the vectors of returns and standard deviations for the securities to generate a new, randomized vector based on the
# normal distribution using mean returns and the covariance matrix as inputs

rand_norm_rets = scipy.stats.multivariate_normal.rvs(mean = rets.mean(),
                                                     cov = rets.cov(),
                                                     size = (len(df)*1000),
                                                     random_state = 12345)
```

```
# Ensure the length of the list is equal to the number of observations per simulation (969 as of 11/05/2020) multiplied
# by the number of simulations. The number of simulations is represented by the range in the function above.
len(rand_norm_rets)
```

```
975000
```

```
# Generating a dataframe from the simulated tuples; this df will then be used to calculate optimal portfolios for each
# simulation
rand_norm_df = pd.DataFrame(rand_norm_rets, columns = symbols)
rand_norm_df
```

B7. Generate a loop over the simulated returns data to produce 1,000 MVO portfolios.

```
# Set constraints and boundaries for this optimization. Unlike the previous optimization where we were generating the entire
# efficient frontier, in this instance we only need to impose the constraint that all portfolio weights sum to 1 since we
# are only looking for the single point for each optimization that maximizes the Sharpe Ratio
# Boundaries stay the same: we specify that all portfolio weights must be between 0 and 1

constraint_4 = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

boundaries_4 = tuple((0,1) for x in weights)

port_rets_info = []

for x in range(1,1001):
    port_opt_rand_tups = sco.minimize(fun = sharpe_function_to_minimize, x0 = equal_weights,
                                      args = rand_norm_df[(x*len(df)-len(df)):(len(df)*x)],
                                      method='SLSQP', bounds=boundaries_4, constraints=constraint_4)
    port_rets_info.append(port_opt_rand_tups)
```

B8. Locate the “optimal” MVO portfolio using the minimax regret decision criterion. Calculate the expected return and volatility of this portfolio. Display the recommended allocations.

```
# Display the portfolio weights for this portfolio
opt_port_allocations[min_returns_list.index(max(min_returns_list))]
```

```
array([3.19143208e-01, 4.74541738e-17, 1.12196629e-15, 3.32170408e-15,
       2.48729531e-16, 1.51102611e-02, 1.60276930e-15, 2.65195174e-15,
       0.00000000e+00, 1.39889185e-15, 0.00000000e+00, 8.94866592e-16,
       9.46433958e-16, 2.47477955e-15, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.09436657e-16, 0.00000000e+00,
       5.59746477e-15, 2.24179350e-01, 7.52785089e-03, 1.64909183e-15,
       0.00000000e+00, 4.28917156e-16, 0.00000000e+00, 5.23092312e-15,
       4.34039329e-01])
```

```
# Calculate the expected portfolio volatility for the identified optimal portfolio - based on the historical returns data
port_vol(opt_port_allocations[min_returns_list.index(max(min_returns_list))], np.log(df / df.shift(1)))
```

```
0.2126217606955506
```

```
# Calculate the expected portfolio return for the identified optimal portfolio - based on the historical returns data
port_ret(opt_port_allocations[min_returns_list.index(max(min_returns_list))], np.log(df / df.shift(1)))
```

```
0.2808385899688251
```

```
# Display the weights for the "optimal" portfolio
pd.DataFrame(opt_port_allocations[min_returns_list.index(max(min_returns_list))].round(4), index = symbols).stack()
```

```
AAPL  0    0.3191
AMGN  0    0.0000
AXP   0    0.0000
BA    0    0.0000
CAT   0    0.0000
CRM   0    0.0151
CSCO  0    0.0000
CVX   0    0.0000
DIS   0    0.0000
GS    0    0.0000
HD    0    0.0000
HON   0    0.0000
IBM   0    0.0000
INTC  0    0.0000
JNJ   0    0.0000
JPM   0    0.0000
KO    0    0.0000
MCD   0    0.0000
MMM   0    0.0000
MRK   0    0.0000
MSFT  0    0.0000
NKE   0    0.2242
PG    0    0.0075
TRV   0    0.0000
UNH   0    0.0000
V     0    0.0000
VZ    0    0.0000
WBA   0    0.0000
WMT   0    0.4340
dtype: float64
```