RISK ANALYSIS AND PERFORMANCE EVALUATION IN ASSET MANAGEMENT

Robust Portfolio Optimization: An Empirical Analysis of the Risk-Adjusted Performance of Equity Strategies Constructed with Classical, Bayesian and Machine-Learning Techniques.

Paul C. McAteer*

**MS (NYU, Stern School of Business), MBA (IE Business School)*

Pre-Print: 30h June 2020

Abstract

This study reviews the empirical evidence over the last decade of the risk-adjusted outperformance of US equity portfolios constructed with robust optimization techniques. The performance of such portfolios is compared to a market-weighted index, a naively diversified (equal-weighted) strategy, Maximal Sharpe Ratio and Global Minimum Variance portfolios constructed within the classical Markowitz optimization framework, a Risk Parity Portfolio and a portfolio optimized with Random Forest techniques. The results confirm that the utilization of robust covariance and return estimators in the portfolio design process yielded significant relative outperformance on a risk-adjusted basis. The paper provides detailed code in Python to facilitate investors' practical implementation of the strategies and to enable academics to easily replicate and interrogate the results.

Key words: Portfolio optimization; Robust estimators; Parameter estimation error; Black-Litterman; Ledoit-Wolf; Machine Learning; Random Forest; Portfolio risk analysis; Portfolio performance analysis; Portfolio construction; Risk Parity; Markowitz; Efficient Frontier

1. Introduction

In spite of the theoretically resilient underpinnings of robust portfolio optimization techniques, prospective (and existing) users of the Black-Litterman and Ledoit-Wolf procedures – which produce robust return and covariance matrix estimates respectively – continue to confront uncertainties regarding the intuition behind the models, their practical implementation and their merit, that is, their capacity to generate out-performance. With respect to the challenges of both comprehension and application, it is instructive to merely conduct a brief survey of the promises of enlightenment contained in the titles of papers published since Black-Litterman's original pioneering work of 1991: "The Intuition Behind Black-Litterman Model Portfolios" (He and Litterman, 1999); "A Demystification of the Black-Litterman Model" (Satchell and Scowcroft, 2000); "A Step-by-Step Guide to the Black-Litterman Model" (Izadorek, 2004); "The Black-Litterman Model Explained" (Cheung, 2010); "Deconstructing Black-Litterman" (Michaud, 2013) and "Reconstructing the Black-Litterman Model" (Walters, 2014).

e-mail: pcm353@stern.nyu.edu

This paper provides a concise synthesis of the conceptual foundations of the robust portfolio optimization techniques without sacrificing analytical rigour. I situate the Ledoit-Wolf and Black-Litterman optimization procedures within the broader theoretical context of Harry Markowitz's Modern Portfolio Theory and progress to discuss novel alternative approaches to diversification and optimization, namely Risk Parity and Random Forest. I provide a detailed computational framework in open-source code which will enable the reader to construct the portfolios, re-specify model parameters and backtest performance using standard metrics. Finally, I utilize this framework to examine the evidence in the US equity market over the last decade as to whether robust estimation techniques have indeed proved capable of producing portfolios which generate relative risk-adjusted outperformance. Performance will be compared to two market benchmarks, a market-weighted index (MW), and an equal-weighted (EW) index, as well as common alternative strategies, namely Maximal Sharpe Ratio (MSR) and Global Minimum Variance (GMV) portfolios constructed within the classical Markowitz optimization framework, a Risk Parity Portfolio (Equal Risk Contribution - ERC) and a portfolio optimized with Random Forest (RF) techniques. The guiding objective is to provide clarity on model construction, implementation, and value.

2. Literature Review

Since the publication in 1952 of Harry Markowitz's seminal work, *Portfolio Selection* [1], the mean-variance methodology has been the dominant solution to the portfolio selection problem. The optimal portfolio is formed by the rational investor who allocates wealth to assets within her investable universe such that she maximizes expected (mean) return for a given risk level, represented by portfolio variance and estimated by the sample covariance matrix of historic asset returns. The set of optimal portfolios for all risk levels defines the efficient frontier. Merton (1972) [2] allowed for the relaxation of the short selling constraint within the context of the classical Mean-Variance Optimization solution.

Academics and practitioners have since confronted multiple challenges related to the practical application of the model, particularly, the sensitivity of the "optimal" portfolio to the estimation error of expected return and volatility. Michaud (1989) [3] contended that Mean-Variance Optimization gave rise to error-maximizing and under-performing portfolios, stating that "The main problem with MVO is its tendency to maximize the effects of errors in the input assumptions [which]… can yield results that are inferior to those of simple equal-weighting schemes" The latter comment on underperformance references earlier work undertaken by Jobson and Korkie (1981) [4]. Michaud further observes that MVO tends to produce unintuitive, concentrated portfolios noting that the model "significantly over-weights those securities that have large estimated returns, negative correlations and small variances". From the perspective of inferential statistics Stein (1956) [5] insisted on the "Inadmissibility of the Usual Estimator of the Mean of a Multivariate Normal Distribution". Best and Grauer (1991) [6] highlighted the extreme sensitivity of portfolio design to changes in the mean return vector. Similarly Chopra (1993) [7] together with Ziemba (1993) [8] demonstrated that small changes to the mean values of variances can result in radically different "optimal" portfolios.

Given the described issues with the estimator inputs, many academics came to focus on Bayes-Stein shrinkage estimation, a technique formulated by Stein (1956) [5] and further developed by James and Stein (1961) [9]. In essence, these estimators are generally formed by shrinking an observed prior estimate of the population mean towards an updated estimator, which incorporates some additional information, in order to obtain a posterior estimate, which is a weighted average of the two. The weights are determined by some shrinkage factor. The updated estimated value may draw on properties of the statistical distribution of the observed data or incorporate exogenous information. This paper leverages the Black-Litterman model (1991,1992) [10] [11] which seeks to provide robust estimates of security returns and the Ledoit-Wolf (2013, 2014) [12] [13] shrinkage technique which aims to generate robust estimates of the covariance matrix. The former produces a weighted average of security returns implied by market equilibrium and the investor's subjective expectations. The latter generates a posterior covariance matrix which is a weighted average of the observed sample covariance matrix and a covariance matrix obtained by using Elton and Gruber's (1973, 1978) [14] [15] constant correlation model in which the correlation coefficients are equal to the mean of the sample correlation coefficients.

In the aftermath of the Global Financial Crisis, risk management came to rival performance management as a driving objective of portfolio optimization. This increased the theoretical and practical interest in the risk parity portfolio, defined as a strategy which seeks to constrain each asset such that they contribute equally to portfolio volatility. Risk Parity portfolios gained favor as the academic literature and its proponents in the Hedge Fund industry proliferated. Noteworthy contributions to the academic discourse include papers by Roncalli et al. (2009, 2012) [16] [17]. The advocacy of Ray Dalio and the performance of the Bridgewater "All Weather" asset allocation strategy further helped increase the popularity of so-called Equal Risk Contribution strategies.

Traditionally portfolio optimization has focused on the ex-ante optimal portfolio based on estimates of future risk and returns. Novel machine learning techniques applied to the portfolio selection problem tend to rely on identifying the ex-post optimal portfolios over an historical time series which serve as a dependent (or "target") variable, and which one then seeks to explain as a function of a large number of independent (or "feature") variables. Breiman developed the concept of the Random Forest (2001) [18], a supervised machine learning algorithm based on ensemble learning, which combines multiple Classification and Regression Trees (CART) (Breiman et al.,1984) [19] using Bagging (Breiman, 1996) [20]. Bagging is a process which aggregates the results of multiple decision trees trained on random subsets of the features and bootstrapped[1] samples of the training data to grow a forest of "random" trees. He posited that ensembles of decision trees could produce highly accurate predictions of target variables whilst handling a large number of input variables without overfitting. The random forest algorithm can be used for both regression and classification tasks. Yang (2013) [21] demonstrated the application of the technique to modelling portfolio risk whilst Khaidem et al (2016) [22] applied it to stock price prediction using technical indicators as the feature variables.

[1] In the jargon, resampling with replacement is referred to as bootstrapping. The term "Bagging" derives from the practice of both *B*ootstrapping and *Agg*regating the results.

3. Theory of Optimal Portfolio Construction

Traditional portfolio optimization theory adheres to the notion that the objective of a rational investor is to select the portfolio which minimizes risk for any given level of expected return amongst the set of all possible portfolios. The set of risk minimizing portfolios for varying required levels of return are described as optimal. The set of all possible portfolios is called the feasible set. Expected portfolio return is the weighted average of the expected returns of portfolio constituents. Portfolio risk refers to the dispersion of expected portfolio returns, represented by their historic standard deviation, under the assumption that these returns are normally distributed. Alternative definitions of risk incorporate the assumption of investors' aversion to semi-variance, negative skewness, and positive excess kurtosis. Hodges (1997) [23] formulated an Adjusted Sharpe Ratio risk measures which incorporate the third and fourth moments of non-normal return distributions. Harlow (1991) [24] employed lower partial moments as a downside risk measure in portfolio selection. Whilst such risk measures have theoretical and intuitive appeal, the co-movement of the higher moments and the lower partial moments has proved difficult to estimate and the expected diversification effect within such portfolios has consequently proved vulnerable to significant estimation error. This paper therefore retains a return-variance optimization criterion which solves for the asset allocation, w*, that maximizes a utility function of the form:

$$\mu_\Pi - \frac{\gamma}{2} V_\Pi$$

Where μ_Π is portfolio return, V_Π is portfolio variance and $\gamma > 0$ represents the degree of risk aversion.

This is the starting point of the classical Markowitz mean-variance optimization solution, which will be described in detail. I will then proceed to describe enhancements to the model which address its well-documented deficiencies by providing robust estimates for security returns and the variance-covariance matrix.

3.1. Canonical Markowitz Framework for Mean-Variance Optimization (MVO)

The true excess returns[2] of the constituent securities in a portfolio are assumed to have a normal distribution, denoted by:

$$r \sim N(\mu, \sigma^2)$$

Where μ is the expected excess return and σ^2 the variance.

The expected excess return of a portfolio is the weighted sum of the expected excess return on each constituent asset:

$$\mu_\Pi = \sum_{i=1}^{N} w_i \, \mu_i$$

which is written in matrix form as follows:

$$\mu_\Pi = \boldsymbol{w}'\boldsymbol{\mu}$$

Where $\boldsymbol{w}'$is the transpose of the asset weight vector and $\boldsymbol{\mu}$ is the vector of expected returns.

The variance of a portfolio is determined by the weights, variances and covariances on the constituent assets. For a portfolio of n assets, we obtain the generalized expression for the variance of the portfolio returns:

$$V_\Pi = \sum_{i=1}^{N} w_i^2 \, \sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j \, \rho_{ij} \, \sigma_i \sigma_j$$

Where ρ_{ij} is the correlation between assets i and j.

Employing matrix notation, portfolio variance is compactly represented a quadratic form of the covariance matrix and the portfolio weights as follows:

$$V_\Pi = \boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}$$

Where $\boldsymbol{\Sigma}$ is an $N \times N$ covariance matrix given by:

$$\begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1N} \\ \vdots & \ddots & \vdots \\ \sigma_{N1} & \cdots & \sigma_{NN} \end{bmatrix}$$

We denote the joint return distribution of the portfolio returns as the following multivariate normal distribution:

$$R_P \sim N(\boldsymbol{w}'\boldsymbol{\mu}, \boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w})$$

Given this parametrization of portfolio variance and excess return, we can formulate the mean-variance optimization problem as an unconstrained quadratic optimization problem which maximizes investor utility, U, in the decision variable w:

$$\underset{w}{\operatorname{argmax}} \, U = \boldsymbol{w}'\boldsymbol{\mu} - \frac{1}{2}\gamma \, \boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}$$

Subject to:

$$\boldsymbol{w} \cdot \boldsymbol{i} = \boldsymbol{1}$$

The optimal weights $\boldsymbol{w^*}$ are found by determining the stationary point of the objective function, which requires equating the partial derivatives of the weight variables to zero. The first order condition is represented thus:

$$\nabla U(\boldsymbol{w}^*) = \frac{\partial V(\boldsymbol{w}^*)}{\partial \boldsymbol{w}} = \mu - \frac{1}{2} \cdot 2\,\gamma\, \boldsymbol{\Sigma}\boldsymbol{w}^* = 0$$

Which simplifies to:

$$\boldsymbol{\mu} - \gamma\,\boldsymbol{\Sigma}\boldsymbol{w}^* = 0$$

Which yields the equivalent expression:

$$\boldsymbol{\mu} = \gamma\,\boldsymbol{\Sigma}\boldsymbol{w}^*$$

[2] Excess Return refers to the return in excess of the risk-free rate.

Which implies the following candidate solution for the so-called market portfolio:

$$\boldsymbol{w}^* = \frac{1}{\gamma}\,\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$$

Finally, we examine the Hessian Matrix of second partial derivatives to determine if it is negative definite and so confirm we have found a (unique) maximum at the stationary point:

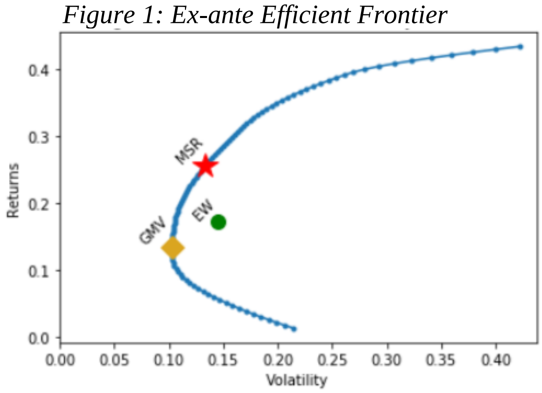
$$\nabla^2 U(\boldsymbol{w}^*) = \mathrm{H}U(\boldsymbol{w}^*) = -\gamma\,\boldsymbol{\Sigma} < 0$$

The market portfolio is the asset allocation solution which maximises expected excess return per unit of risk, that is, it provides the optimal asset weights to maximise the Sharpe ratio:

$$\max_{w} \frac{\boldsymbol{w}'\boldsymbol{\mu}}{\sqrt{\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}}}$$

This Maximal Sharpe Ratio (MSR) portfolio is visible on the ex-ante efficient frontier depicted in Figure 1 along with the Global Minimum Variance and Equal-Weighted portfolio. Conceptually, the Global Minimum Variance portfolio can be considered a special variant of the MSR where the expected return for each constituent security is equalised, and asset weights are purely a function of the covariance matrix. The EW "naively diversified" portfolio, is dominated by both the MSR and GMV portfolios.

Figure 1: Ex-ante Efficient Frontier



3.2. Achieving Robust Return Estimates with the Black-Litterman Procedure

The Black-Litterman procedure is a Bayesian shrinkage method, which incorporates (1) The *asset returns implied by market equilibrium*, denoted by Π; and (2) The *subjective expectations of asset returns*, formed by a "link" matrix P expressing bearishness or bullishness and a vector Q expressing expected relative or absolute returns for these positions. The result is a vector of *posterior expected returns*, denoted by $\hat{\mu}_{BL}$.

The vector of implied equilibrium excess returns is obtained by a process of reverse-optimization, using the observed market capitalizations of securities for weights, the observed sample variance-covariance matrix and the aggregate risk aversion of market participants, denoted by δ. δ is derived from observed market data in the following manner:

If: $\Pi_i = \beta_i\left[E(R_M) - r_f\right]$

Then, equivalently:

$$\Pi_i = \frac{CoV_{i,M}}{Var_{M,M}}\left[E(R_M) - r_f\right]$$

$$= \frac{\left[E(R_M) - r_f\right]}{Var_{M,M}}\,CoV_{i,M}$$

The first term, $\left[E(R_M) - r_f\right]/Var_{M,M}$, is δ ,the market price of risk. Under the assumption that rational investors will seek to maximize the risk-return tradeoff on all assets, then the market portfolio will be formed by rational investors maximizing their utility function in the weight variable. w_λ denotes asset weights under conditions of market equilibrium.

$$\underset{w}{\operatorname{argmax}}\left\{\boldsymbol{w}'\boldsymbol{\Pi} - \frac{1}{2}\,\delta\,\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}\right\} = \boldsymbol{w_\lambda}$$

Assuming therefore that market capitalization weights are the product of market participants' aggregate efforts to maximize utility and are thus optimal, and given furthermore that both the sample covariance matrix and the average risk aversion level are observable, the derivation of the vector of implied equilibrium excess returns is trivial:

$$\boldsymbol{\Pi} = \delta\, \boldsymbol{\Sigma}\, \boldsymbol{w_\lambda}$$

This formula moreover supplies further intuition vis-à-vis the market price of risk. Pre-multiplying both sides of the previous equation by the transpose of the weights of the market in equilibrium gives expected market return as a function of expected market variance and the risk coefficient:

$$\boldsymbol{w'_\lambda \Pi} = \delta\, \boldsymbol{w'_\lambda \Sigma\, w_\lambda}$$

Restating in terms of δ :

$$\delta = \frac{\boldsymbol{w'_\lambda \Pi}}{\boldsymbol{w'_\lambda \Sigma\, w_\lambda}} = \frac{\boldsymbol{w'_\lambda \Pi}}{\sigma^2_{mkt}}$$

$$= \frac{\boldsymbol{w'_{mkt} \Pi}}{\sigma_{mkt}} \times \frac{1}{\sigma_{mkt}}$$

$$= Sharpe\ Ratio_{mkt} \times \frac{1}{\sigma_{mkt}}$$

The vector of *posterior expected returns*, $\boldsymbol{\hat{\mu}_{BL}}$, will be a function of the degree of confidence in the subjective expected returns relative to the degree of confidence in the market-implied expected returns. Essentially, $\boldsymbol{\hat{\mu}_{BL}}$ can be considered as type of complex weighted average of subjective and market-implied expected returns where the weights are determined by the level of confidence in one expected return relative to the other.

For market implied returns, if *uncertainty* is captured by the dispersion or variance of asset returns in the market equilibrium model, then, intuitively, the inverse of the sample variance-covariance matrix[3] will reflect the degree of *certainty*. The greater the magnitude of variability, the smaller the inverse of $\boldsymbol{\Sigma}$. A bounded scalar parameter[4] τ may be applied to $\boldsymbol{\Sigma}$ to adjust for estimation error. One approach is to set τ = 1/T= T^{-1}, where T is the number of historical periods used. Generally, τ is close to zero. The prior equilibrium distribution therefore is:

$$\mu_{prior} \sim N\ (\boldsymbol{\Pi},\ \tau\Sigma)$$

[3] Black and Litterman assume that the variance of the estimate $\Sigma\pi$ is proportional to the sample covariance matrix of the excess returns Σ with a coefficient of proportionality τ i.e. $\Sigma\pi = \tau\Sigma$

[4] $0 < \tau < 1$

The confidence factor for market-implied returns is therefore:

$$(\tau\boldsymbol{\Sigma})^{-1}$$

Having obtained the *prior*, the equilibrium vector of excess return, the investors' K views on N assets are now described by (1) a $K \times N$ matrix of bullish or bearish (long or short) positions denoted by $\boldsymbol{P}$[5], where K refers to the number of views and N to the number of assets in the investment universe; and (2) a K-element column vector of subjective expected returns on these positions, $\boldsymbol{Q}$. By way of example, we assume 3 views in an investment universe of 4 securities. The first is of the relative outperformance of asset A versus asset B ; the second and third is the belief that assets B and C will return 3% on average. We hold no views on Asset D. The position matrix, P, would be of the form:

$$\boldsymbol{P} = \begin{pmatrix} & \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{D} \\ \boldsymbol{View\ 1} & 1 & -1 & 0 & 0 \\ \boldsymbol{View\ 2} & 0 & 1 & 0 & 0 \\ \boldsymbol{View\ 3} & 0 & 0 & 1 & 0 \end{pmatrix}$$

The first row incorporates the relative positions, the second row and third rows, the absolute positions.

The Q vector of expected returns will be of the form:

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{View\ 1} & 10\% \\ \boldsymbol{View\ 2} & 2\% \\ \boldsymbol{View\ 3} & 1\% \end{pmatrix}$$

The general forms of the P matrix and Q vector are:

$$\boldsymbol{P} = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{k1} & \cdots & p_{kn} \end{pmatrix}$$

$$\boldsymbol{Q} = \begin{pmatrix} Q_1 \\ \vdots \\ Q_k \end{pmatrix}$$

$\boldsymbol{\Omega}$ models uncertainty in the views space. The uncertainty of the views is represented by a random,

[5] For relative views, the sum of the weights will equal 0 while absolute views equal 1

independent, normally distributed error term vector ($\boldsymbol{\varepsilon}$). Views under uncertainty will thus have the form of a $\boldsymbol{Q}$ vector and $\boldsymbol{\varepsilon}$ vector:

$$\begin{matrix} Q_1 \\ \vdots \\ Q_k \end{matrix} + \begin{matrix} \varepsilon_1 \\ \vdots \\ \varepsilon_k \end{matrix}$$

The error term has mean of 0 and a covariance matrix $\boldsymbol{\Omega}$. The distribution of error terms is thus:

$$\begin{matrix} \varepsilon_1 \\ \vdots \\ \varepsilon_k \end{matrix} \sim N\left[\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \left(\boldsymbol{\Omega} = \begin{pmatrix} \omega_{1,1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \omega_{k,k} \end{pmatrix}\right)\right]$$

The structure of the view-uncertainty matrix $\boldsymbol{\Omega}$ is inherited from the sample covariance matrix $\boldsymbol{\Sigma}$ and the $\boldsymbol{P}$ matrix which identifies the asset positioning on the views vector $\boldsymbol{Q}$. $\boldsymbol{\Omega}$ is a diagonal covariance matrix with off-diagonal positions set to zero under the assumption that the views are independent of one another. The variance of the views is formed in the following manner:

$$\boldsymbol{\Omega} = diag\ P(\tau\boldsymbol{\Sigma})P^T$$

The diagonal matrix Ω is therefore populated in the following manner:

$$\boldsymbol{\Omega} = \begin{pmatrix} P_1(\tau\boldsymbol{\Sigma})P_1^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & P_k(\tau\boldsymbol{\Sigma})P_k^T \end{pmatrix}$$

The views distribution is:

$$r_{views} \sim N\,(\mathbf{Q}, \boldsymbol{\Omega})$$

The confidence factor for subjective expected returns is seen below, where the transpose of the P matrix simply links the confidence Ω^{-1} to vector Q:

$$(\mathrm{P}'\boldsymbol{\Omega}^{-1})$$

We have now gathered the necessary inputs to calculate the vector of posterior expected returns, $\hat{\mu}_{BL}$ also referred to as the Combined Return Vector:

$$\hat{\mu}_{BL} = [(\tau\Sigma)^{-1} + \mathrm{P}'\Omega^{-1}\mathrm{P}]^{-1}\,[(\tau\Sigma)^{-1}\Pi + \mathrm{P}'\Omega^{-1}\mathrm{Q}]$$

Where:

- $\hat{\mu}_{BL}$ is the Combined Return Vector (N-element vector where N refers to the assets in the investable universe);
- τ is a scalar;
- Σ is the sample covariance matrix of excess returns (N x N matrix).
- Π is the Implied Equilibrium Return Vector (N x 1 column vector).
- Q is the View Vector (K x 1 column vector, where K refers to the subjective views on the N assets).
- P is a matrix that identifies the asset positions related to the K views in the view vector (K x N matrix).
- Ω is a diagonal covariance matrix of error terms of the subjective views where the elements represent the uncertainty in each view (K x K matrix).

It should be apparent that $\hat{\mu}_{BL}$ is a confidence-weighted average of the expected returns implied by market equilibrium Π and the expected returns implied by the investor's views Q, where $(\tau\Sigma)^{-1}$ and $P\Omega^{-1}$ represent confidence in estimates of the market equilibrium and views respectively. We multiply the second term $[(\tau\Sigma)^{-1}\Pi + \mathrm{P}'\Omega^{-1}\mathrm{Q}]$ in the master formula by the first term $[(\tau\Sigma)^{-1} + \mathrm{P}'\Omega^{-1}\mathrm{P}]^{-1}$ to ensure that the sum of all weights is equal to 1.

3.3. Achieving Robust Estimates of the Covariance Matrix with the Ledoit-Wolf Shrinkage Method

The shrinkage technique for covariance matrix estimation involves shrinking (1) an *unbiased, high-variance, unstructured estimate* toward (2) a *biased, low-variance, structured estimate*. In the context of Ledoit-Wolf model, the objective is to obtain the optimal weighted average of a sample covariance matrix and a shrinkage target, based on a constant correlation structure:

$$\hat{\Sigma}_{LW} = w\hat{\Sigma}_{CC} + (1-w)\hat{\Sigma}_S$$

The shrinkage intensity is determined by the shrinkage constant, the weight w applied to the shrinkage target. The optimal shrinkage constant $\boldsymbol{w}^*$is derived by minimization of a quadratic loss function, which in a matrix setting is the squared Frobenius norm analogous with the squared error loss function. We are thus seeking to minimize here the quadratic measure of distance between the true (Σ) and inferred $\left(\mathrm{w}\,\hat{\Sigma}_{CC} + (1-\mathrm{w})\,\hat{\Sigma}_S\right)$ covariance matrices:

$$\mathrm{L(w)} = \left\| \left(\mathrm{w}\,\hat{\Sigma}_{CC} + (1-\mathrm{w})\hat{\Sigma}_S\right) - \Sigma \right\|_F^2$$

Which gives rise to the expected loss function:

$$\mathrm{E(L(w))} = \sum_{i=1}^{N}\sum_{j=1}^{N} E\left(w\,\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} + (1-\mathrm{w})s_{ij} - \sigma_{ij}\right)^2$$

Where: $\bar{\mathrm{r}}$ is the mean of sample correlations, s_{ii} *and* s_{jj} are the sample variances and σ_{ij} is the true covariance between elements i and j.

Noting that $\mathrm{E}(x^2) = \mathrm{Var}(x) + [\mathrm{E}(x)]^2$; for any random variable x; we can rewrite

$$\mathrm{E(L(w))} = \sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(w\,\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} + (1-\mathrm{w})s_{ij}\right) + \left[E\left(w\,\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} + (1-\mathrm{w})s_{ij} - \sigma_{ij}\right)\right]^2$$

Which simplifies to:

$$\begin{aligned}\mathrm{E(L(w))} = \sum_{i=1}^{N}\sum_{j=1}^{N} & w^2\, Var\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right) \\ & + (1-\mathrm{w})^2\, Var\left(s_{ij}\right) \\ & + 2w(1-w)Cov\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}, s_{ij}\right) \\ & + w^2\,(\phi_{ij} - \sigma_{ij})^2\end{aligned}$$

Where: ϕ_{ij} is the constant covariance term for elements ij formed by the average correlation in the population $\bar{\varrho}$ and the square root of the population variance terms $\sqrt{\sigma_{i\,i}\sigma_{jj}}$.

Taking the first derivative of the expected loss function with respect to *w* gives:

$$\begin{aligned}\frac{\mathrm{d\,E(L(w)}}{\mathrm{d\,w}} = 2\sum_{i=1}^{N}\sum_{j=1}^{N} & w\, Var\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right) \\ & - (1-\mathrm{w})\, Var\left(s_{ij}\right) \\ & + (1-2w)Cov\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}, s_{ij}\right) \\ & + w\,(\phi_{ij} - \sigma_{ij})^2\end{aligned}$$

Setting the first derivative to zero and solving for w*, yields:

$$w^* = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(s_{ij}\right) - Cov\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}, s_{ij}\right)}{\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} - s_{ij}\right) + (\phi_{ij} - \sigma_{ij})^2}$$

Notice that the terms in the numerator represent the sum of the variances of the entries of the sample covariance matrix and sum of the covariances of the entries of the constant correlation covariance matrix with the entries of the sample covariance matrix. Notice also that the denominator contains the population terms ϕ_{ij} and σ_{ij}. Ledoit and Wolf show that w* can be shown to be proportional to a constant $\hat{\kappa}$ divided by time T:

$$w^* = \frac{\hat{\kappa}}{\mathrm{T}}$$

It follows from this relation that:

$$\begin{aligned}\kappa &= Tw^* \\ &= \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(\sqrt{T}\, s_{ij}\right) - Cov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right), \left(\sqrt{T}\, s_{ij}\right)\right]}{\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} - s_{ij}\right) + (\phi_{ij} - \sigma_{ij})^2}\end{aligned}$$

Taking the first term in the numerator, Ledoit and Wolf contend that standard asymptotic theory, under the assumptions of iid data and finite fourth moments provides consistent estimators for π:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(\sqrt{T}\, s_{ij}\right) \to \sum_{i=1}^{N}\sum_{j=1}^{N} \mathrm{Asy}Var\left(\sqrt{T}\, s_{ij}\right) \to \pi$$

Where π represents the sum of asymptotic variances of the entries of the sample covariance matrix scaled by $\sqrt{T}$.

Similarly:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} Cov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right),\ \left(\sqrt{T}\, s_{ij}\right)\right]$$

$$\rightarrow \sum_{i=1}^{N}\sum_{j=1}^{N} AsyCov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right),\ \left(\sqrt{T}\, s_{ij}\right)\right]$$
$$\rightarrow \rho$$

Where ρ represents the sum of asymptotic covariances of the entries of the shrinkage target with the entries of the sample covariance matrix scaled by $\sqrt{T}$.

The authors prove that a consistent estimator of $\hat{\pi}_{\mathrm{ij}}$ will be found by first finding the product of the deviations of the returns on securities i and j from their average returns at each time t and then taking sum of the squared differences of this product and the sample variance over total time T:

$$\hat{\pi}_{\mathrm{ij}} = \frac{1}{T}\sum_{t=1}^{T}\left\{(\mathrm{y}_{i,t} - \bar{\mathrm{y}}_{i,})(\mathrm{y}_{j,t} - \bar{\mathrm{y}}_{j,}) - s_{ij}\right\}^{2}$$

Then the consistent estimator for π is:

$$\hat{\pi} = \sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\pi}_{\mathrm{ij}}$$

A consistent estimator of ρ is proven to be found by splitting it into its diagonal and off-diagonal elements. By definition:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} AsyCov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right),\ \left(\sqrt{T}\, s_{ij}\right)\right]$$

$$= \sum_{i=1}^{N} AsyVar\left[\sqrt{T}\, s_{ii}\right]$$
$$+ \sum_{i=1}^{N}\sum_{\substack{j=1 \\ \mathrm{j}\neq\mathrm{i}}}^{N} AsyCov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right),\ \left(\sqrt{T}\, s_{ij}\right)\right]$$

Which implies on the diagonal for element i:

$$AsyVar\left[\sqrt{T}\, s_{ii}\right] = \frac{1}{T}\sum_{t=1}^{T}\left\{(\mathrm{y}_{i,t} - \bar{\mathrm{y}}_{i,}) - s_{ii}\right\}^{2}$$
$$= \hat{\pi}_{\mathrm{ij}}$$

And on the off-diagonal for elements i,j:

$$AsyCov\left[\left(\sqrt{T}\ \bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}}\right),\ \left(\sqrt{T}\, s_{ij}\right)\right]$$

$$= \frac{\bar{\mathrm{r}}}{2}\sqrt{\frac{s_{jj}}{ss_{ii}}}\, AsyCov\left[\sqrt{T}\, s_{ii},\ \sqrt{T}\, s_{ij}\right]$$
$$+ \sqrt{\frac{s_{ii}}{ss_{jj}}}\, AsyCov\left[\sqrt{T}\, s_{jj},\ \sqrt{T}\, s_{ij}\right]$$

$$= \frac{\bar{\mathrm{r}}}{2}\sqrt{\frac{s_{jj}}{ss_{ii}}}\, \hat{\varphi}_{ii,ij} + \sqrt{\frac{s_{ii}}{ss_{jj}}}\, \hat{\varphi}_{jj,ij}$$

Where $\varphi_{ii,ij}$ and $\varphi_{jj,ij}$ are:

$$\varphi_{ii,ij} = \frac{1}{T}\sum_{t=1}^{T}\left\{(\mathrm{y}_{i,t} - \bar{\mathrm{y}}_{i,})^{2} - s_{ii}\right\}\left\{(\mathrm{y}_{i,t} - \bar{\mathrm{y}}_{i,})(\mathrm{y}_{j,t} - \bar{\mathrm{y}}_{j,}) - s_{ij}\right\}^{2}$$
$$\varphi_{jj,ij} = \frac{1}{T}\sum_{t=1}^{T}\left\{(\mathrm{y}_{j,t} - \bar{\mathrm{y}}_{j,})^{2} - s_{jj}\right\}\left\{(\mathrm{y}_{i,t} - \bar{\mathrm{y}}_{i,})(\mathrm{y}_{j,t} - \bar{\mathrm{y}}_{j,}) - s_{ij}\right\}^{2}$$

Then the consistent estimator for ρ is:

$$\hat{\rho} = \sum_{i=1}^{N}\hat{\pi}_{\mathrm{ii}} + \sum_{i=1}^{N}\sum_{\substack{j=1 \\ \mathrm{j}\neq\mathrm{i}}}^{N}\frac{\bar{\mathrm{r}}}{2}\sqrt{\frac{s_{jj}}{ss_{ii}}}\, \hat{\varphi}_{ii,ij} + \sqrt{\frac{s_{ii}}{ss_{jj}}}\, \hat{\varphi}_{jj,ij}$$

Finally, turning to the denominator terms:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} Var\left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} - s_{ij}\right) = 0\,\frac{1}{T}$$

And:

$$\gamma = \sum_{i=1}^{N}\sum_{j=1}^{N} (\phi_{ij} - \sigma_{ij})^{2}$$

Where γ is the misspecification of the population shrinkage target, for which the consistent estimator is its sample counterpart :

$$\hat{\gamma} = \sum_{i=1}^{N}\sum_{j=1}^{N} \left(\bar{\mathrm{r}}\sqrt{s_{ii}s_{jj}} - s_{ij}\right)^2$$

Collecting the three consistent estimator terms over T gives the optimal shrinkage constant w^*:

$$w^* = \frac{(\hat{\pi} - \hat{\rho})\,/\,\hat{\gamma}}{T} = \frac{\hat{\kappa}}{\mathrm{T}}$$

4. Diversification by other means: The Risk Parity Portfolio.

The objective of a Risk Parity Portfolio is that all constituent assets contribute equally to portfolio risk. More precisely the weighted marginal risk contribution (variously referred to as component risk, the dollar risk contribution or simply the risk contribution) for every asset must be the same:

$$\mathrm{w}_i \frac{\partial \sigma_P}{\partial \mathrm{w}_i} = \mathrm{w}_j \frac{\partial \sigma_P}{\partial \mathrm{w}_j}$$

Equivalently and somewhat more intuitively, the risk contribution can be expressed as a function of covariance with the portfolio:

$$\mathrm{RC}_i = \frac{\mathrm{w}_i}{\sigma_P} \operatorname{Cov}[\mathrm{R}_i, \mathrm{R}_P]$$

$$= \frac{\mathrm{w}_i\,(\boldsymbol{\Sigma}\mathbf{w})_i}{\sqrt{\mathbf{w}'\boldsymbol{\Sigma}\mathbf{w}}}$$

The sum of these risk contributions must add up to give total portfolio risk:

$$\sigma_P = \sum_{i=1}^{N} \mathrm{RC}_i$$

Since the portfolio volatility is the sum of contributions, the relative contribution of asset i to portfolio volatility is defined as:

$$\mathrm{RRC}_i = \frac{\mathrm{RC}_i}{\sigma_P}$$

The sum of these relative risk contributions must equal 1:

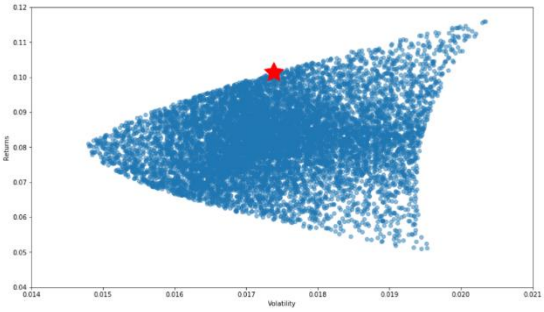
$$1 = \sum_{i=1}^{N} RRC$$

No analytical expression is generally available for the asset weights which equalize the risk contributions. Numerical methods are employed such that asset weights produce a portfolio where each holding has the following relative contribution to portfolio risk:

$$RRC_i = \frac{1}{N}$$

5. Optimizing portfolios with Random Forest Regression techniques

Figure 3: Ex-post Efficient Frontier



We employ a Random Forest Regressor to predict the optimal portfolio weights which will give the maximum Sharpe Ratio. This weights variable is known as the target. The historical sample data of these optimal portfolios is obtained by calculating the portfolio risk and return associated with 1 million randomly generated weight vectors in each month of the sample period and then identifying the one which produces the highest Sharpe ratio. We are effectively constructing the ex-post efficient frontier and finding the ex-post optimal portfolio using the daily realized volatility and return in each month. See Figure 3 above which shows the ex-post efficient frontier, the set of feasible portfolios and the realized risk and return of the optimal portfolio.

The predictor (or "feature" variable) inputs to the Random Forest regressor are the following high frequency price-related technical indicators:

(i) Relative Strength Indicator.

$$RSI = 100 - \frac{100}{1 + RS}$$

$$RS = \frac{Average\ Gain\ Over\ past\ 14\ days}{Average\ Loss\ Over\ past\ 14\ days}$$

(ii) Percentage Price Oscillator

$$PPO = \frac{12\ period\ EMA - 26\ period\ EMA}{26\ period\ EMA} \times 100$$

$EMA = Exponential\ moving\ average$

(iii) Exponentially Weighted Moving Average.

$$\hat{\sigma}_{t+1} = \sqrt{\lambda\sigma_t^2 + (1 - \lambda)\mu_t^2}$$

$\lambda = Decay\ Factor\ for\ 14\ days$
$\mu^2 = Squared\ Daily\ Return$
$\sigma^2 = Daily\ Variance$

(iv) Short-term percentage price volatility

$$\hat{\sigma}_{t+1} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\mu_t^2}$$

$m = 14\ (days)$

(v) Rate of Change.

$$ROC = \frac{(P_t - P_{t-n})}{P_{t-n}} \times 100$$

$P_t = Closing\ Price$
$P_{t-n} = Closing\ Price\ 10\ days\ ago$

The algorithm for the Random Forest Regression is as follows:

1) Draw a bootstrap sample B_1 of size N from the training data. The training data in our model is 70% of the total dataset.

2) Randomly select a subset m_1 of T features where $m_1 < T$. The features in our model are high frequency technical indicators relating to closing price data.

3) From this subset, select the most informative feature to form the root node of the decision tree by identifying the feature with the lowest sum of squared error across the branches.

4) The sum of squared error is calculated as the sum of squared differences between each individual target value and the expected (mean) target value at each branch for that category. The target values in our model are the optimal weights which resulted in the ex-post maximal Sharpe ratio in the bootstrap sample. For example, to calculate the SSE of an RSI input:

$$\sum_{RSI>s}(\bar{y}_{RSI>s} - y^n)^2 + \sum_{RSI<s}(\bar{y}_{RSI<s} - y^n)^2$$

5) Note that we just use the threshold method to convert numerical feature data (the technical indicator) into categories (values of the technical indicator above/below threshold *s*). The threshold level will impact the SSE. The general expression for the objective function is therefore the minimization of the sum of squared error via the feature and threshold variables. $x_m^{(n)} < s$ refers to the numerical value of the m^th attribute of the n^th data point:

$$\min_s\left(\sum_{x_m^{(n)}<s}\min_y(\bar{y} - y^n)^2 + \sum_{x_m^{(n)}\geq s}\min_y(\bar{y} - y^n)^2\right)$$

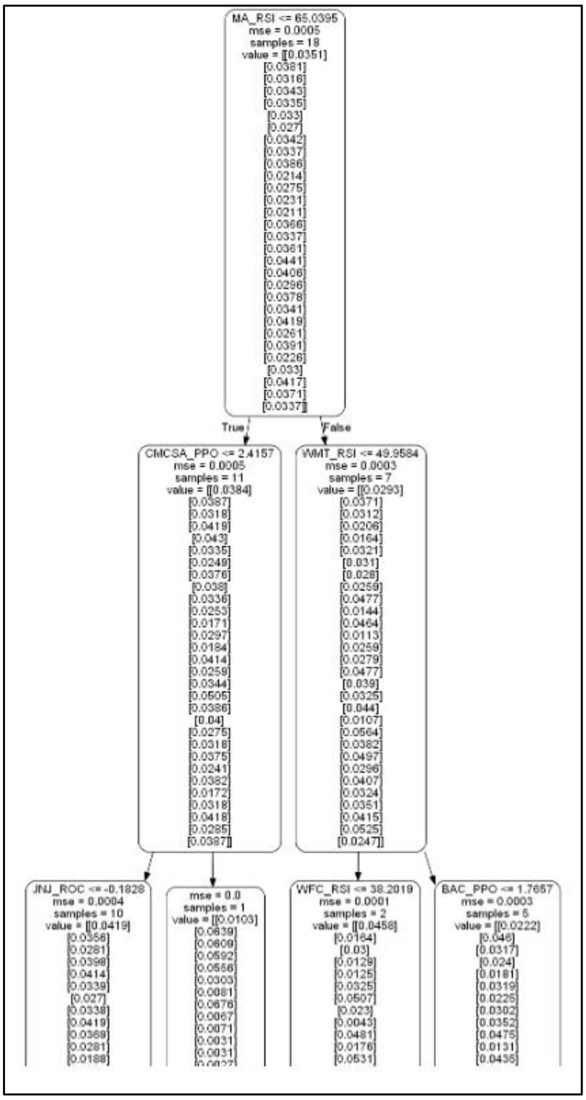
6) Having obtained the best variable/split point among the m_1, the root node is split into two daughter nodes.

7) Grow the Random Tree, RT_1, by recursively repeating steps 2-6 for the remaining elements of m_1 until the minimum node size is reached.

8) Populate the Random Forest with additional trees $RT_{(2...n)}$ by repeating steps 1-7 *n* number of times

The average at each leaf node of each tree will give the expected target values determined by the (limited) input variables used to build that tree. The average values of all the leaf nodes in the forest will give the expected target values for all the input variables used to build that forest. This forest therefore will predict the optimal (Sharpe Ratio-maximizing) asset weights for the month, taking all the current technical indicator levels as model input values.

Figure 3: Root and daughter nodes of constituent decison tree in Random Forest



6. Investment Strategy Design

We limit the investment universe to the 30 largest securities in the S&P 500 by market capitalization with available price data over the sample period. Portfolios are optimized and rebalanced at the beginning of every month. We analyze the performance of 9 strategies in total:

- We introduce two benchmark portfolios, the equal-weighted (EW) and cap-weighted (CW) indices.
- We construct two Global Minimum Variance (GMV) portfolios formed by the optimal security weights, for which the expected return corresponds to the target minimum volatility on the ex-ante efficient frontier, having been supplied with some covariance matrix. This obviates the need to forecast returns. In the first case, which we call GMV-Sample, the covariance matrix is formed by the sample volatilities and correlations; in the second case, which we call GMV-Shrink, we incorporate robust estimates of the covariance matrix by employing the Ledoit-Wolf procedure. In both cases, the sampling period is 12 months.
- We further construct two Maximal Sharpe Ratio (MSR) portfolios formed by the optimal security weights which maximize expected return per unit of volatility on the ex-ante efficient frontier, having been supplied with a vector of mean returns and some covariance matrix. In the first case, which we call MSR-Sample, the covariance matrix is formed by the sample volatilities and correlations; in the second case, which we call MSR-Shrink, we use the shrunk covariance matrix. In both cases, the sampling period is again 12 months.
- The Black-Litterman portfolio is constructed by drawing on the analyst consensus for each security's 12-month price target, obtained from Marketbeat.com. To minimize the importance of stale estimates and overweight more recent estimates, we calculate the exponential weighted moving average of analysts' price objectives using a lambda of 0.8. To ensure that only high conviction bets are included, the P Matrix is composed of 3 views. The first view over-weights the security with the highest expected return and under-weights the security with the lowest expected return. The corresponding input for this view in the Q vector will be the expected difference in return between these two assets. The second view over-weights the security with the second highest expect return and underweights the security with the second lowest return. Again, the corresponding input for this view in the Q vector will be the expected relative difference in return. The same procedure is employed to form the remaining view on the assets with the third highest and third lowest returns. Views are updated every six months and the portfolio is rebalanced every month.
- The Risk Parity Portfolio is built using the sample covariance matrix and is rebalanced and reoptimized every month.
- Finally, the portfolio optimized with Random Forest techniques builds the ex-post efficient frontier and identifies the portfolio with the ex-post maximal Sharpe ratio using the daily volatilities, correlations and returns in each given month. These weights of the portfolio with the maximal Sharpe ratio in each month are the target variables used to train the model. The feature variables are the Technical indicator values at the beginning of each month. The Random Forest portfolio therefore is rebalanced and re-optimized every month.

7. Performance Metrics

This study employs the following metrics:

(i) Sharpe Ratio.

The Sharpe Ratio measures the return achieved per unit of volatility incurred:

$$Sharpe\ Ratio\ = \frac{Annualized\ Return}{Ann.\ Standard\ Dev.}$$

(ii) Sortino Ratio.

The Sortino Ratio measures the return achieved per unit of downside volatility incurred:

$$Sortino\ Ratio\ = \frac{Annualized\ Return}{Ann. Semi - Deviation}$$

$$Semi\ Dev. = \sqrt{\frac{1}{n} \times \sum_{r_t < Mean}^{n} (Mean - r_t)^2}$$

(iii) Conditional Value at Risk.

Conditional Value at Risk, alternatively known as Expected Shortfall or Expected Tail Loss, refers to the mean loss of portfolio value given that a loss is occurring at or below a particular q-quantile (for example, 5% given a confidence level of 95%)

$$CVaR_{\alpha} = -\frac{1}{\alpha}\int_{0}^{\alpha} VaR_{\gamma}(X)dy$$

Where α is the threshold level of VaR and VaR_{γ} is the Value at Risk at the defined confidence level.

(iv) Modified Value at Risk.

Modified VaR, alternatively known as Cornish-Fisher VaR, permits the computation of the Value-at-Risk for non-normal with positive or negative skewness and fat tails that is, positive excess kurtosis.

Formally defined, if Gaussian VaR is:

$$VaR_{Gaussian} = \mu - z_g\, \sigma$$

Where: z_g is the z-score determined by the determined confidence level.

Then:

$$VaR_{Cornish\ Fisher} = \mu - z_{cf}\, \sigma$$

Where: z_{cf} is the adjusted z-score determined by z_g, and the observed skew (S) and kurtosis (K) of the distribution of returns:

$$z_{cf} = z_g + \frac{1}{6}\left(z_g^2 - 1\right)S + \frac{1}{24}\left(z_g^3 - 3z_g\right)K - \frac{1}{36}\left(2z_g^3 - 5z_g\right)S^2$$

(v) Maximum Drawdown.

Maximum drawdown is defined as the peak-to-trough decline of an investment during a specific period. It is usually quoted as a percentage of the peak value.

$$Max\ Drawdown = \frac{P - L}{P}$$

Where: P is the peak value before the largest drop in value and L is the lowest value before the new high is established.

8. Implementation in Python

The complete code to implement the risk analysis and performance evaluation of the described strategies is presented in order for the reader to verify the results, expand or modify the study and provide granularity in terms of strategy design and backtesting methodology.

8.1. Define Parameters for raw data import and storage

```
# Import the python libraries
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt

# Selection of Securities and Date Range
Securities = "MSFT AAPL AMZN GOOG NVDA BRK-A JNJ V PG JPM UNH MA INTC VZ HD T PFE MRK PEP WMT BAC XOM DIS KO CVX CSCO CMCSA WFC BA ADBE"
Start = "2016-06-30"
End = "2020-06-30"
# Select File Type for upload of Security Data
filetype =".csv"
# Specify Local Storage Location

path = r"C:\Users\delga\Desktop\NYU\CQF_Work\Portfolio_Management"
#Convert data parameters to string
```

```
Sec_Dates = Securities,Start, End

def convertTuple(tup):
    str =  '_'.join(tup)
    return str
conv = convertTuple(Sec_Dates)
print(conv)
# Converted data parameters + File type = Filename
filename = conv+filetype
print(filename)
# Join path, filename & filetype for single reference "File"
import os
File = os.path.join(path,filename)

print(File)
```

8.2. Import save and inspect raw data.

```
import yfinance a s yf
data = yf.download(Securities, start=Start, end=End)

# Save Data
data.to_csv(File)

# Inspect the first 5 lines of the saved CSV file
f =open(File,"r")
f.readlines()[:5]
```

8.3. Create dataframe to house daily prices. Clean data structure

```
#The filename passed to the pd.read_csv() function creates the daily price dataframe.
#Specified that the first two rows shall be handled as an headers.
#Specified that the first column shall be handled as an index.
#Specified that the index values are of type datetime
df_csv = pd.read_csv(File, header=[0,1], index_col=0, parse_dates=True,)
df_csv.info()


# Define string and substring to count securities in portfolio. Reduce Dataframe to daily adj close for 30 securities
string = Securities
substring = " "
Sec_count = string.count(substring)+1

df_csv = df_csv.iloc[:,0:Sec_count]
df_csv

# Create single level header from multilevel header
df_csv.columns = df_csv.columns.map('|'.join).str.strip('|')
print(df_csv.columns)

df_csv.columns = df_csv.columns.str.replace(r'Adj Close|$', '')

df_csv.columns = df_csv.columns.str.lstrip('|')  # strip suffix at the right end only.
df_csv.info()

# Identify null values in dataset

df_csv.isnull().any()

# Drop null values in dataset
df_csv = pd.DataFrame(df_csv.dropna().round(2))
df_csv.info()
```

8.4. Inspect asset prices and daily and monthly returns,

```
# Plot Daily Price Evolution
df_csv.plot(figsize=(12, 60), subplots=True);

# Calculate and plot daily returns

returns_daily = df_csv.pct_change()
returns_daily.plot(figsize=(12, 60), subplots=True);

# Calculate and plot monthly returns (from first day of each mth)
```

```
"""Date Offset
"""
prices_BOM = df_csv.resample("BMS").first()
prices_BOM

# Calculate monthly returns
ind_return = prices_BOM.pct_change()
ind_return

# Remove null values and format datetime index
ind_return = ind_return.dropna().round(4)
ind_return

ind_return.index = pd.to_datetime(ind_return.index, format="%Y%m").to_period('M')
ind_return

# plot monthly returns
ind_return.plot(figsize=(12, 60), subplots=True);
```

8.5. Construct cap-weighted benchmark,

```
#Import
ind_mktcap = pd.read_excel("mktcap_2008_2020.xlsx", sheet_name='Mkt_Cap', index_col=0, parse_dates=True)
ind_mktcap

#Slice by specified starting and ending dates
ind_mktcap =ind_mktcap.loc[Start:End]
ind_mktcap

#Date Format
ind_mktcap.index = pd.to_datetime(ind_mktcap.index, format="%Y%m").to_period('M')
ind_mktcap

# Compute and inspect price evolution of benchmark

total_mktcap = ind_mktcap.sum(axis="columns")
total_mktcap.plot(figsize=(12,6));

# Compute benchmark capweights
ind_capweight = ind_mktcap.divide(total_mktcap, axis="rows")
ind_capweight = ind_capweight.iloc[0:]
ind_capweight

#Check that sum to one
ind_capweight.sum(axis="columns")

# Compute monthly market return
total_market_return = (ind_capweight * ind_return).sum(axis="columns")
total_market_return

total_market_return.plot();

total_market_index = 1000*(1+total_market_return).cumprod()
total_market_index.plot(title="Market Cap Weighted Index");
```

8.6. Construct equal-weighted benchmark

```
n_ew = ind_return.shape[1]
w_ew = np.repeat(1/n_ew, n_ew)
ind_equalweight = ind_capweight.multiply(1/ind_capweight/n_ew, axis="rows")
ind_equalweight

# Calculate monthly return
total_eqweighted_return = (ind_equalweight * ind_return).sum(axis="columns")
total_eqweighted_return.plot();

# Calculate evolution of price of equal-weighted index
total_eqweighted_index = 1000*(1+total_eqweighted_return).cumprod()

total_eqweighted_index.plot(title="Equal Cap Weighted Index");
```

```
# Compare evolution of prices of cap-weighted and equal-weighted index
total_market_index.plot(title="Market Cap Weighted Index", label="Mkt-weighted", legend=True)
total_eqweighted_index.plot(title="Equal Cap Weighted Vs. Market Cap Weighted Indices", label="Eq-weighted", legend=True);
```

8.7. Programs to compute expected return vector and sample covariance matrix

```
def annualize_rets(r, periods_per_year):
    """
    Gives the annualized return. Takes a times series of returns and their periodicity as arguments
    """
    compounded_growth = (1+r).prod()
    n_periods = r.shape[0]
    return compounded_growth**(periods_per_year/n_periods)-1

def annualize_vol(r, periods_per_year):
    """
    Gives the annualized volatility. Takes a times series of returns and their periodicity as arguments.
    """
    return r.std()*(periods_per_year**0.5)

rf = 0.00
ann_factor = 12
er = annualize_rets(ind_return, ann_factor)
ev = annualize_vol(ind_return, ann_factor)
corr = ind_return.corr()
cov = ind_return.cov()
covmat_ann = cov*(ann_factor)
```

8.8. Programs to compute risk adjusted performance measures

```
def sharpe_ratio(r, riskfree_rate, periods_per_year):
    """
    Computes the annualized sharpe ratio of a set of returns
    """
    # convert the annual riskfree rate to per period
    rf_per_period = (1+riskfree_rate)**(1/periods_per_year)-1
    excess_ret = r - rf_per_period
    ann_ex_ret = annualize_rets(excess_ret, periods_per_year)
    ann_vol = annualize_vol(r, periods_per_year)
    return ann_ex_ret/ann_vol

import scipy.stats
def is_normal(r, level=0.01):
    """
    Applies the Jarque-Bera test to determine if a Series is normal or not
    Test is applied at the 1% level by default
    Returns True if the hypothesis of normality is accepted, False otherwise
    """
    if isinstance(r, pd.DataFrame):
        return r.aggregate(is_normal)
    else:
        statistic, p_value = scipy.stats.jarque_bera(r)
        return p_value > level

def drawdown(return_series: pd.Series):
    """Takes a time series of asset returns.
       returns a DataFrame with columns for
       the wealth index,
       the previous peaks, and
       the percentage drawdown
    """
    wealth_index = 1000*(1+return_series).cumprod()
```

```python
    previous_peaks = wealth_index.cummax()
    drawdowns = (wealth_index - previous_peaks)/previous_peaks
    return pd.DataFrame({"Wealth": wealth_index,
                         "Previous Peak": previous_peaks,
                         "Drawdown": drawdowns})

def semideviation(r):
    """
    Returns the semideviation aka negative semideviation of r
    r must be a Series or a DataFrame, else raises a TypeError
    """
    if isinstance(r, pd.Series):
        is_negative = r < 0
        return r[is_negative].std(ddof=0)
    elif isinstance(r, pd.DataFrame):
        return r.aggregate(semideviation)
    else:
        raise TypeError("Expected r to be a Series or DataFrame")

def var_historic(r, level=5):
    """
    Returns the historic Value at Risk at a specified level
    i.e. returns the number such that "level" percent of the returns
    fall below that number, and the (100-level) percent are above
    """
    if isinstance(r, pd.DataFrame):

        return r.aggregate(var_historic, level=level)
    elif isinstance(r, pd.Series):
        return -np.percentile(r, level)
    else:
        raise TypeError("Expected r to be a Series or DataFrame")


def cvar_historic(r, level=5):
    """
    Computes the Conditional VaR of Series or DataFrame
    """
    if isinstance(r, pd.Series):
        is_beyond = r <= -var_historic(r, level=level)
        return -r[is_beyond].mean()
    elif isinstance(r, pd.DataFrame):
        return r.aggregate(cvar_historic, level=level)
    else:
        raise TypeError("Expected r to be a Series or DataFrame")


from scipy.stats import norm
def var_gaussian(r, level=5, modified=False):
    """
    Returns the Parametric Gauusian VaR of a Series or DataFrame
    If "modified" is True, then the modified VaR is returned,
    using the Cornish-Fisher modification
    """
    # compute the Z score assuming it was Gaussian
    z = norm.ppf(level/100)
    if modified:
        # modify the Z score based on observed skewness and kurtosis
        s = skewness(r)
        k = kurtosis(r)
        z = (z +
                (z**2 - 1)*s/6 +
                (z**3 -3*z)*(k-3)/24 -
                (2*z**3 - 5*z)*(s**2)/36
            )
    return -(r.mean() + z*r.std(ddof=0))

def skewness(r):
    """
    Alternative to scipy.stats.skew()
    Computes the skewness of the supplied Series or DataFrame
    Returns a float or a Series
    """
```

```
        demeaned_r = r - r.mean()
        # use the population standard deviation, so set dof=0
        sigma_r = r.std(ddof=0)
        exp = (demeaned_r**3).mean()
        return exp/sigma_r**3


    def kurtosis(r):
        """
        Alternative to scipy.stats.kurtosis()
        Computes the kurtosis of the supplied Series or DataFrame
        Returns a float or a Series
        """
        demeaned_r = r - r.mean()
        # use the population standard deviation, so set dof=0
        sigma_r = r.std(ddof=0)
        exp = (demeaned_r**4).mean()
        return exp/sigma_r**4

    from scipy import stats
    for column in ind_return:
        stats.probplot(ind_return[column], dist="norm", plot=plt)
        plt.show()
```

8.9. Construct efficient frontier based on classical Markowitz model

```
# Define functions for portfolio return and volatility

def portfolio_return(weights, returns):
    """
    Computes the return on a portfolio from constituent returns and weights
    """
    return weights.T @ returns


def portfolio_vol(weights, covmat):
    """
    Computes the vol of a portfolio from a covariance matrix and constituent weights
    """
    vol = (weights.T @ covmat @ weights)**0.5
    return vol

# Program to return optimal weights for maximization of Sharpe ratio

from scipy.optimize import minimize

def msr(riskfree_rate, er, cov):
    """
    Returns the weights of the portfolio that gives you the maximum sharpe ratio
    given the riskfree rate, an expected returns vector and a covariance matrix
    """
    n = er.shape[0] # Input for initial guess
    init_guess = np.repeat(1/n, n) # Equal Weighting for init_guess
    bounds = ((0.0, 1.0),) * n # Minimum and maximum individual allocation (No shorting constraint)
    # Define the constraint: Sum of portfolio weights variable minus one must equal zero. ("Equality" Constraint)
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    def neg_sharpe(weights, riskfree_rate, er, cov):
        """
        Defining the objective function which we seek to minimize:
        The investor seeks weights to maximise Sharpe ratio (Excess Ret/Vol), for given return vector, cov matrix and rfr.
        Equivalent to minimizing the negative of this ratio.
        """
        r = portfolio_return(weights, er)
```

```
        vol = portfolio_vol(weights, cov)
        return -(r - riskfree_rate)/vol

    # Scipy optimize function takes obj fun; init guess, input args for obj fun, constraints on total weights, boundaries
    # for individual weights, the optimization method
    weights = minimize(neg_sharpe, init_guess,
                       args=(riskfree_rate, er, cov), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,),
                       bounds=bounds)
    return weights.x

# Program to return optimal weights to minimize vol for a given target return

def minimize_vol(target_return, er, cov):
    """
    Returns the optimal weights that achieve the target return
    given a set of expected returns and a covariance matrix
    """
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples!
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er: target_return - portfolio_return(weights,er)
    }
    weights = minimize(portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,return_is_target),
                       bounds=bounds)
    return weights.x

# Weighting scheme returning optimal weights for minimization of global min. variance

def gmv(cov):
    """
    Returns the weights of the Global Minimum Volatility portfolio
    given a covariance matrix
    """
    n = cov.shape[0]
    return msr(0, np.repeat(1, n), cov) #Exp. ret set to 1 for all securities

# Weighting scheme returning  equal weighted portfolio.
def weight_ew(r):
    """
    Returns the weights of the EW portfolio based on the asset returns "r" as a DataFrame
    """
    n = len(r.columns)
    ew = pd.Series(1/n, index=r.columns)
    return ew

def optimal_weights(n_points, er, cov):
    """
    Returns a list of weights that represent a grid of n_points on the efficient frontier given a range of
    target returns (from the lowest expected return to the highest expected return)
        """
        target_rs = np.linspace(er.min(), er.max(), n_points)
```

```python
        weights = [minimize_vol(target_return, er, cov) for target_return in target_rs]
        return weights

    def plot_ef(n_points, er, cov, style='.-', legend=False, show_cml=False, riskfree_rate=0, show_ew=False, show_gmv=False):
        """
        Plots the multi-asset efficient frontier using the "optimal weights" function
        """
        weights = optimal_weights(n_points, er, cov)
        rets = [portfolio_return(w, er) for w in weights]
        vols = [portfolio_vol(w, cov) for w in weights]
        ef = pd.DataFrame({
            "Returns": rets,
            "Volatility": vols
        })
        ax = ef.plot.line(x="Volatility", y="Returns", style=style, legend=legend)
        ax.set_title('Figure 1: Ex-Ante Efficient Frontier (June 2020)')
        plt.xlabel('Volatility')
        plt.ylabel('Returns')
        if show_cml:
            ax.set_xlim(left = 0)
            # get MSR
            w_msr = msr(riskfree_rate, er, cov)
            r_msr = portfolio_return(w_msr, er)
            vol_msr = portfolio_vol(w_msr, cov)
            # add CML
            cml_x = [vol_msr]
            cml_y = [r_msr]
            ax.plot(cml_x, cml_y, color='red', marker="*", linestyle='dashed', linewidth=2, markersize=18, label='msr')
            plt.annotate("MSR", xy=(vol_msr, r_msr), ha='right', va='bottom', rotation=45)
        if show_ew:
            n = er.shape[0]
            w_ew = np.repeat(1/n, n)
            r_ew = portfolio_return(w_ew, er)
            vol_ew = portfolio_vol(w_ew, cov)
            # add EW
            ax.plot([vol_ew], [r_ew], color='green', marker='o', markersize=10, label='ew')
            plt.annotate("EW", xy=(vol_ew, r_ew), horizontalalignment='right', verticalalignment='bottom', rotation=45)
        if show_gmv:
            w_gmv = gmv(cov)
            r_gmv = portfolio_return(w_gmv, er)
            vol_gmv = portfolio_vol(w_gmv, cov)
            # add GMV
            ax.plot([vol_gmv], [r_gmv], color='goldenrod', marker="D", markersize=12, label='gmv')
            plt.annotate("GMV", xy=(vol_gmv, r_gmv), horizontalalignment='right', verticalalignment='bottom', rotation=45)
            return ax

    # Display eff. frontier
    plot_ef(100, er, covmat_ann, style='.-', legend=False, show_cml=True, riskfree_rate=rf, show_ew=True, show_gmv=True);
```

8.10. Shrink Covariance Matrix

```python
def sample_cov(r, **kwargs):
    """
    Returns the sample covariance of the supplied returns
    """
    return r.cov()

def cc_cov(r, **kwargs):
    """
    Estimates a covariance matrix by using the Elton/Gruber Constant Correlation model
    """
```

```
    rhos = r.corr()
    n = rhos.shape[0]
    # this is a symmetric matrix with diagonals all 1
    rho_bar = (rhos.values.sum()-n)/(n*(n-1))
    ccor = np.full_like(rhos, rho_bar)
    np.fill_diagonal(ccor, 1.)
    sd = r.std()
    return pd.DataFrame(ccor * np.outer(sd, sd), index=r.columns, columns=r.columns)

def shrinkage_cov(r, delta=0.5, **kwargs):
    """
    Covariance estimator that shrinks between the Sample Covariance and the Constant Correlation Estimators
    """
    prior = cc_cov(r, **kwargs)
    sample = sample_cov(r, **kwargs)
    return delta*prior + (1-delta)*sample

# Reconstruct eff. frontier with shrunken covar. matrix
plot_ef(100, er, shrink_cov_ann, style='.-', legend=False, show_cml=True, riskfree_rate=rf, show_ew=True, show_gmv=True);
```

8.11. Design Risk Parity Portfolio

```
def risk_contribution(w,cov):
    """
    Compute the relative contributions to risk of the constituents of a portfolio, given a set of portfolio weights
    and a covariance matrix
    """
    total_portfolio_var = portfolio_vol(w,cov)**2
    # Marginal contribution of each constituent to portfolio variance
    marginal_contrib = cov@w

    # Relative contribution of each constituent to portfolio variance (risk)
    risk_contrib = np.multiply(marginal_contrib,w.T)/total_portfolio_var
    return risk_contrib

from scipy.optimize import minimize

def target_risk_contributions(target_risk, cov):
    """
    Returns a portfolio with constituent security weights such
    that their risk contributions to the portfolio are as close as possible to
    the target_risk contributions for a given the covariance matrix.
    """
    n = cov.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    def msd_risk(weights, target_risk, cov):
        """
        The objective function: Minimise the Sum of Squared Differences in the risk contributions to the portfolio
        and the target_risk contributions via the asset weights decision variable
        """
        w_contribs = risk_contribution(weights, cov)
        return ((w_contribs-target_risk)**2).sum()

    weights = minimize(msd_risk, init_guess,
                       args=(target_risk, cov), method='SLSQP',
                       options={'disp': False},
```

```
                       constraints=(weights_sum_to_1,),
                       bounds=bounds)
    return weights.x

def equal_risk_contributions(cov):
    """
    Returns the weights of the portfolio that equalizes the risk contributions
    of the constituents based on the given covariance matrix
    """
    n = cov.shape[0]
    return target_risk_contributions(target_risk=np.repeat(1/n,n), cov=cov)

def weight_erc(r, cov_estimator=sample_cov, **kwargs):
    """
    Produces the weights of the ERC portfolio given a returns series and covariance matrix strucrure.
    """
    est_cov = cov_estimator(r, **kwargs)
    return equal_risk_contributions(est_cov)

def target_risk_contributions(target_risk, cov):
    """
    Returns a portfolio with constituent security weights such
    that their risk contributions to the portfolio are as close as possible to
    the target_risk contributions for a given the covariance matrix.
    """
    n = cov.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    def msd_risk(weights, target_risk, cov):
        """
        The objective function: Minimise the Sum of Squared Differences in the risk contributions to the portfolio
        and the target_risk contributions via the asset weights decision variable
        """
        w_contribs = risk_contribution(weights, cov)
        return ((w_contribs-target_risk)**2).sum()

    weights = minimize(msd_risk, init_guess,
                       args=(target_risk, cov), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,),
                       bounds=bounds)
    return weights.x

def equal_risk_contributions(cov):
    """
    Returns the weights of the portfolio that equalizes the risk contributions
    of the constituents based on the given covariance matrix
    """
    n = cov.shape[0]
    return target_risk_contributions(target_risk=np.repeat(1/n,n), cov=cov)

def weight_erc(r, cov_estimator=sample_cov, **kwargs):
    """
    Produces the weights of the ERC portfolio given a returns series and covariance matrix strucrure.
    """
    est_cov = cov_estimator(r, **kwargs)
    return equal_risk_contributions(est_cov)

# RRC of ERC portfolio
RRC_erc = risk_contribution(equal_risk_contributions(cov), cov)
```

```
RRC_erc.plot.bar(title="Relative (%) Risk Contributions of an ERC portfolio");

# Portfolio composition of ERC strategy. (Numpy array)
weight_erc(ind_return, cov_estimator=sample_cov)

# Portfolio composition of ERC strategy. (DataFrame)
numpy_weight_erc = weight_erc(ind_return, cov_estimator=sample_cov)
df_weight_erc = pd.DataFrame(data=numpy_weight_erc, index=ind_return.columns,  columns=["ERC Asset Allocation"])
df_weight_erc

# Portfolio vol of ERC strategy
Port_vol_erc = portfolio_vol(weight_erc(ind_return), cov)
Port_vol_erc

# Risk Contribution ERC strategy

RC_erc = RRC_erc * Port_vol_erc
RC_erc.plot.bar(title="($) Risk Contributions of an ERC portfolio");
```

8.12. Design Black-Litterman Optimized Portfolio

```
# Lookback period

BL_per_beg_1 = Start
BL_per_end_1 = End


# Market inputs: rfr. exp returns vector, sample covariance matrix
rf_1 = 0.00
ann_factor_1 = 12
er_1 = annualize_rets(ind_return[BL_per_beg_1:BL_per_end_1] , ann_factor)
ev_1 = annualize_vol(ind_return[BL_per_beg_1:BL_per_end_1], ann_factor)

corr_1 = ind_return[BL_per_beg_1:BL_per_end_1].corr()
cov_1 = ind_return[BL_per_beg_1:BL_per_end_1].cov()

# Data for Views Vector, q

View_1 = 0.20
View_2 = 0.10
View_3 = 0.05

# Data for Pick Matrix, p

Long_1 = 'T'
Short_1 = 'JPM'
Long_2 = 'V'
Short_2 = 'GOOG'
Long_3 = 'UNH'
Short_3 = 'MA'

# Specify investable universe.
assets = list(ind_return.columns)
assets

# Calculate correlation matrix and convert to Dataframe
rho = corr_1
rho

# Calculate expected volatilities of securities
vols = pd.DataFrame(ev_1, columns=["Vols"])
vols

# Market weights (optimal assumimg market equilibrium)
w_eq = ind_capweight.loc[BL_per_end_1]
w_eq

# Define prior covariance matrix (sample annualised covar matrix here)
sigma_prior = vols.dot(vols.T) * rho

sigma_prior

# Compute Equilibrium-implied returns vector and convert to series

def implied_returns(delta, sigma, w):
    """
Obtain the implied expected returns by reverse engineering the weights
Inputs:

```

```
delta: Risk Aversion Coefficient (scalar)
sigma: Variance-Covariance Matrix (N x N) as DataFrame
    w: Market weights (N x 1) as Series
Returns an N x 1 vector of Returns as Series
    """
    ir = delta * sigma.dot(w).squeeze() # to get a series from a 1-column dataframe
    ir.name = 'Implied Returns'
    return ir

# Compute Pi and compare:
pi = implied_returns(delta=2.5, sigma=sigma_prior, w=w_eq)

# Populate views vector , Q: (X will outperform Y by Z%)
q = pd.Series([View_1]) # First view

# start with a single view and an empty Pick Matrix, to be overwritten with the specific pick(s) + view(s)
p = pd.DataFrame([0.]*len(assets), index=assets).T

# Pick 1
p.iloc[0][Long_1] = +1.
p.iloc[0][Short_1] = -1
(p*100).round(1)

# Add second view
view2 = pd.Series([View_2], index=[1])
q = q.append(view2)
pick2 = pd.DataFrame([0.]*len(assets), index=assets, columns=[1]).T
p = p.append(pick2)
p.iloc[1][Long_2]=+1
p.iloc[1][Short_2]=-1
np.round(p.T, 3)*100

# Add third view
view3 = pd.Series([View_3], index=[2])
q = q.append(view3)
pick3 = pd.DataFrame([0.]*len(assets), index=assets, columns=[2]).T
p = p.append(pick3)
p.iloc[2][Long_3]=+1
p.iloc[2][Short_3]=-1
np.round(p.T, 3)*100

# Calculate Omega as proportional to the variance of the prior
def proportional_prior(sigma, tau, p):
    """
        Returns the He-Litterman simplified Omega
        Inputs:
        sigma: N x N Covariance Matrix as DataFrame
        tau: a scalar
        p: a K x N DataFrame linking Q and Assets
        returns a P x P DataFrame, a Matrix representing Prior Uncertainties
        """
        helit_omega = p.dot(tau * sigma).dot(p.T)
        # Make a diag matrix from the diag elements of Omega
        return pd.DataFrame(np.diag(np.diag(helit_omega.values)),index=p.index, columns=p.index)

# Program to compute the posterior expected returns based on the original black litterman reference model

from numpy.linalg import inv

def bl(w_prior, sigma_prior, p, q,
                omega=None,
                delta=2.5, tau=.02):
        """
    # Computes the posterior expected returns based on the original black litterman reference model
    # W.prior must be an N x 1 vector of weights, a Series
    # Sigma.prior is an N x N covariance matrix, a DataFrame
    # P must be a K x N matrix linking Q and the Assets, a DataFrame
    # Q must be an K x 1 vector of views, a Series
    # Omega must be a K x K matrix a DataFrame, or None
```

```
# if Omega is None, we assume it is proportional to variance of the prior
# delta and tau are scalars
    """
    if omega is None:
        omega = proportional_prior(sigma_prior, tau, p)
    # Force w.prior and Q to be column vectors
    # How many assets?
    N = w_prior.shape[0]
    # How many views?
    K = q.shape[0]
    # First, reverse-engineer the weights to get pi
    pi = implied_returns(delta, sigma_prior,  w_prior)
    # Adjust (scale) Sigma by the uncertainty scaling factor
    sigma_prior_scaled = tau * sigma_prior
    # posterior estimate of the mean, use the "Master Formula"
    # we use the versions that do not require
    # Omega to be inverted (see previous section)
    # this is easier to read if we use '@' for matrixmult instead of .dot()
    #     mu_bl = pi + sigma_prior_scaled @ p.T @ inv(p @ sigma_prior_scaled @ p.T + omega) @ (q - p @ pi)
    mu_bl = pi + sigma_prior_scaled.dot(p.T).dot(inv(p.dot(sigma_prior_scaled).dot(p.T) + omega).dot(q - p.dot(pi).values))
    # posterior estimate of uncertainty of mu.bl
    #sigma_bl = sigma_prior + sigma_prior_scaled - sigma_prior_scaled @ p.T @ inv(p @ sigma_prior_scaled @ p.T + omega) @ p @ sigma_prior_scaled
    sigma_bl = sigma_prior + sigma_prior_scaled - sigma_prior_scaled.dot(p.T).dot(inv(p.dot(sigma_prior_scaled).dot(p.T) + omega)).dot(p).dot(sigma_prior_scaled)
    return (mu_bl, sigma_bl)

# Specify scalars

delta = 2.5
tau = 0.05

# Derive the Black Litterman Expected Returns
bl_mu, bl_sigma = bl(w_eq, sigma_prior, p, q, omega=None, delta=delta, tau= tau)
 (bl_mu*100).round(2)

 (bl_sigma*100).round(2)

# for convenience and readability, define the inverse of a dataframe

def inverse(d):
    """
    Invert the dataframe by inverting the underlying matrix
    """
    return pd.DataFrame(inv(d.values), index=d.columns, columns=d.index)

def w_msr(sigma, mu, scale=True):
    """
    Optimal (Tangent/Max Sharpe Ratio) Portfolio weights
    by using the Markowitz Optimization Procedure
    Mu is the vector of Excess expected Returns
    Sigma must be an N x N matrix as a DataFrame and Mu a column vector as a Series
    """
    w = inverse(sigma).dot(mu)
    if scale:
        w = w/sum(w) # fix: this assumes all w is +ve
    return w

# Optimal BL portfolio weights
bl_port = w_msr(bl_sigma,bl_mu)
bl_port.plot(kind='bar')

# Name BL optimal portfolio
alt_wstar = (w_msr(sigma=bl_sigma, mu=bl_mu,scale=True)*100).round(4)
alt_wstar

```

```
    # Transpose & Export for Backtesting purposes
    df_alt_wstar = pd.DataFrame(alt_wstar, columns=[ind_return.index[-1]]).T
    #df_alt_wstar.to_excel("BL_WEIGHTS4.5.xlsx", sheet_name=End)
    df_alt_wstar

    # Test: Market inputs should give market weights as output
    w_eq_check  = w_msr(delta*sigma_prior, pi, scale=False)
    w_eq_check

    # BL-implied Alpha : BL Exp Returns - Equilibrium Impl. Returns

    Exp_Active_ret = (((bl_mu) - (pi))*(100)).round(2)
    Exp_Active_ret.plot(kind='bar', title = "BL-implied Active Return");

    # Display the difference in Posterior and Prior weights
    Active_weight = np.round(wstar - w_eq/(1+tau), 3)*100

    Active_weight.plot(kind='bar', title = "BL-implied Active Weight");
```

8.13. Optimization with Random Forest

```
# Use Cleaned Closing Price Data
full_df = df_csv
full_df

# Resample the full DataFrame to monthly timeframe
monthly_df = full_df.resample('BMS').first()
# Calculate daily returns of stocks

returns_daily = full_df.pct_change()

# Calculate monthly returns of the stocks
returns_monthly = monthly_df.pct_change().dropna()
# Suffix to column name
returns_monthly.columns += '_RET'

print(returns_monthly.tail())

# Compute Daily covariance of stocks for each historical monthly period

# Create Empty dictionary for each month's daily covariances
covariances = {}

# Extract all dates relating to each trading day in the daily return times series
rtd_idx = returns_daily.index


for i in returns_monthly.index:
    # Mask daily returns for each month and year. Masks are an array of boolean values for which a condition
    is met.
    # In this instance, for each month-year of the monthly returns index, the mask identifies as "True" where
    # the index of daily returns has a matching month-year timestamp.
    # The resulting boolean arrays is used to isolate data in the original data array ie daily returns in
    each looped month

    mask = (rtd_idx.month == i.month) & (rtd_idx.year == i.year)

# The covariance calculation is performed on daily data in each monthly period
    covariances[i] = returns_daily[mask].cov()

covariances

# Obtain 1,000,000 potential portfolio performances for each month via random iterations of the weights vector.

portfolio_returns, portfolio_volatility, portfolio_weights = {}, {}, {}
```

```python

# For each key value (BOM date) in the covariances dictionary, return the covariance in that calendar month.
for date in sorted(covariances.keys()):
    cov = covariances[date]
    # Randomly iterate 1,000,000 times the weights vector for the 30 assets
    for portfolio in range(1000000):
        weights = np.random.random(cov.shape[0])
        weights /= np.sum(weights) # /= divides weights by their sum to normalize
        returns = np.dot(weights, returns_monthly.loc[date])
        volatility = np.sqrt(np.dot(weights.T, np.dot(cov, weights)))
        # The setdefault() method returns the value of the appended item with the specified key. (Like
        Vlookup)
        portfolio_returns.setdefault(date, []).append(returns)
        portfolio_volatility.setdefault(date, []).append(volatility)
        portfolio_weights.setdefault(date, []).append(weights)

print(portfolio_weights[date][0])

import matplotlib.pyplot as plt

# Plot efficient frontier for latest month of available data
date = sorted(covariances.keys())[-1]
latest_returns = portfolio_returns[date]
latest_vol = portfolio_volatility[date]
# define your figure then plot information in that space
plt.figure(figsize=(14,8))
plt.scatter(x=latest_vol, y= latest_returns, alpha=0.5, cmap='RdYlBu')
plt.axis([0.014, 0.030, 0.028, 0.10])


# Identify point on eff frontiier with maximal sharpe ratio in that month
max_sharpe_coord = max_sharpe_idxs[date]

# Place an red star on the point with the best Sharpe ratio
plt.scatter(x=latest_vol[max_sharpe_coord], y=latest_returns[max_sharpe_coord], marker=(5,1,0),color='r',s=1000)

# Label axes
plt.xlabel('Volatility')
plt.ylabel('Returns')

# Display

plt.show()

# Library to import technical indicators
import talib

# 1. Calculate exponentially-weighted moving average of daily returns
ewma_daily = returns_daily.ewm(span=14).mean()

# Resample daily returns to first business day of the month with the first day for that month
ewma_monthly = ewma_daily.resample('BMS').first()

# Shift ewma for the month by 1 month forward so we can use it as a feature for future predictions
ewma_monthly = ewma_monthly.shift(1).dropna()

# Rename Columns
ewma_monthly.columns += '_EWMA'

ewma_monthly

# 2. Calculate standard deviation of daily returns
sd_daily = returns_daily.apply(lambda colseries: talib.STDDEV(colseries,  timeperiod=14, nbdev=1))

```

```
# Resample daily returns to starting business day of the month with the first day for that month
sd_monthly = sd_daily.resample('BMS').first()

# Shift sd for the month by 1 month forward so we can use it as a feature for future predictions
sd_monthly = sd_monthly.shift(1).dropna()

# Rename Columns
sd_monthly.columns += '_SD'

sd_monthly

# 3. Calculate Rate of Change of Price
ROC_daily = full_df.apply(lambda colseries: talib.ROC(colseries, timeperiod=10))

# Resample daily ROC to starting business day of the month with the first day for that month
ROC_monthly = ROC_daily.resample('BMS').first()

# Shift sd for the month by 1 month forward so we can use it as a feature for future predictions
ROC_monthly = ROC_monthly.shift(1).dropna()

# Rename Columns
ROC_monthly.columns += '_ROC'


ROC_monthly

# 4. Calculate RSI
RSI_daily = full_df.apply(lambda colseries: talib.RSI(colseries, timeperiod=14))

# Resample daily RSI to starting business day of the month with the first day for that month
RSI_monthly = RSI_daily.resample('BMS').first()


# Shift sd for the month by 1 month forward so we can use it as a feature for future predictions
RSI_monthly = RSI_monthly.shift(1).dropna()

# Rename Columns
RSI_monthly.columns += '_RSI'


RSI_monthly

# 5. Calculate PPO
PPO_daily = full_df.apply(lambda colseries: talib.PPO(colseries, fastperiod=12, slowperiod=26, matype=0))

# Resample daily RSI to starting business day of the month with the first day for that month
PPO_monthly = PPO_daily.resample('BMS').first()

# Shift sd for the month by 1 month forward so we can use it as a feature for future predictions
PPO_monthly = PPO_monthly.shift(1).dropna()

# Rename Columns
PPO_monthly.columns += '_PPO'

PPO_monthly

# Collect Tech Indicators in Dataframe
Tech_Ind_df = pd.concat([ewma_monthly,sd_monthly, ROC_monthly, RSI_monthly, PPO_monthly], axis=1)
Tech_Ind_df = Tech_Ind_df.dropna()
Tech_Ind_df.info()

# Create features from Technical Indicators and targets from historically optimal security weights
targets_wt, features_ti = [], []

for date, row in Tech_Ind_df.iterrows():

```

```
        # Get the index number of the best sharpe ratio for each date
        best_idx = max_sharpe_idxs[date]
        # Use the maximal sharpe ratio for each date to find optimal portfolio weights on that date
        targets_wt.append(portfolio_weights[date][best_idx])
        # add Technical Indicators to features
        features_ti.append(Tech_Ind_df)

    # Convert list of target (optimal) weights to numpy array
    targets_wt_array = np.array(targets_wt)

    # Then to dataframe
    targets_wt_df = pd.DataFrame(data = targets_wt_array, columns= full_df.columns, index=Tech_Ind_df.index)

    targets_wt_df.info()

    # Create complete Dataframe of weights, returns and Tech Indicators
    ft_trg_df = pd.concat([Tech_Ind_df, returns_monthly, targets_wt_df], axis=1)
    ft_trg_df= ft_trg_df.dropna()

    # Calculate correlation matrix for complete dataframe
    Target_Feat_corr = ft_trg_df.corr()
    Target_Feat_corr

    # Plot heatmap of correlation matrix
    import seaborn as sns
    plt.figure(figsize=(14,8))
    sns.heatmap(Target_Feat_corr, annot=False, annot_kws = {"size": 11}, cmap='RdYlGn')
    plt.yticks(rotation=0, size = 1); plt.xticks(rotation=90, size = 1)  # fix ticklabel directions and size

    plt.tight_layout()  # fits plot area to the plot, "tightly"
    plt.show()  # show the plot

    # Create features and targets dataframes
    ret_names = returns_monthly.columns
    ft_names = Tech_Ind_df.columns
    tg_names = full_df.columns

    mret = ft_trg_df[ret_names]
    ft = ft_trg_df[ft_names]
    tg = ft_trg_df[tg_names]

    # Create training set + testing set for features and targets

    # Create a size for the training set that is 85% of the total number of samples
    train_size_1 = int(0.85 * ft.shape[0])

    # Apply the trainsize to obtain a (starting) chronological subset of the features data to train the algo

    train_features_1 = ft[:train_size_1]
    # Apply trainsize to obtain a (starting) chronological subset of the target data to train algo
    train_targets_1 = tg[:train_size_1]

    # Apply trainsize to obtain an (ending) chronological subset of the features data to test algo
    test_features_1 = ft[train_size_1:]
    # Apply trainsize to obtain an (ending) chronological subset of the targets data to test algo
    test_targets_1 = tg[train_size_1:]

    # Inspect dimensions
    print(train_features_1.shape, test_features_1.shape)
    print(train_targets_1.shape, test_targets_1.shape)

    # Specify model with default parameters
    rfr_1 = RandomForestRegressor(n_estimators=1000, random_state=42)
    # Run Model
```

```
rfr_1.fit(train_features_1, train_targets_1)
# Output Model Explanatory Power

print(rfr_1.score(train_features_1, train_targets_1))
print(rfr_1.score(test_features_1, test_targets_1))

# Specify hyperparameters to be tuned

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = [int(x) for x in np.linspace(start = 10, stop =150, num = 30)]
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 150, num = 20)]
max_depth.append(None)
# With Replacement?
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'bootstrap': bootstrap}
print(random_grid)

# Use the random grid to search for best hyperparameters using 10 fold cross validation and 100,000 iterations
# search across 10000 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rfr_1, param_distributions = random_grid, n_iter = 100,
      cv = 5, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(train_features_1, train_targets_1)

# Identify best hyparameters
rf_random.best_params_

# Re-specify model with tuned hyperparameters
rfr_random = RandomForestRegressor(n_estimators=1200, random_state=42, max_features=10, max_depth= 83)
# Run Model
rfr_random.fit(train_features_1, train_targets_1)
# Output Model Explanatory Power

print(rfr_random.score(train_features_1, train_targets_1))
print(rfr_random.score(test_features_1, test_targets_1))

# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

# Pull out one tree from the forest
tree = rfr_random.estimators_[6]

# Export the image to a dot file

export_graphviz(tree, out_file = 'tree.dot', feature_names = ft_names, rounded = True, precision = 4)
# Use dot file to create a graph

graph = pydotplus.graph_from_dot_file('tree.dot')
# Write graph to a png file
Image(graph.create_png())
# Save PNG
graph.write_png("tree_ex.png")

# Get security weight predictions from model on train and test
train_predictions_1 = rfr_random.predict(train_features_1)
test_predictions_1 = rfr_random.predict(test_features_1)
```

```python

    # Calculate and plot returns from our RF predictions
    test_returns_1 = np.sum(mret.iloc[train_size_1:] * test_predictions_1, axis=1)
    plt.plot(test_returns_1, label='algo')

    plt.legend()
    plt.show()

    # Generate portfolio return in test period
    test_returns_1

    # Calculate cumulative return of RF-optimized portfolio
    cash = 1000
    algo_cash =  [cash]  # set equal starting cash amounts
    for r in test_returns_1:
        cash *= 1 + r
        algo_cash.append(cash)

    print('algo returns:', (algo_cash[-1] - algo_cash[0]) / algo_cash[0])

    # Get feature importances from our random forest model
    importances_1 = rfr_random.feature_importances_

    # Get the index of importances from greatest importance to least
    sort_index = np.argsort(importances_1)[::-1]
    x = range(len(importances_1))

    # Create tick labels
    plt.figure(figsize=(16,10))
    labels = np.array(ft_names)[sort_index]
    plt.bar(x, importances_1[sort_index], tick_label=labels)

    # Rotate tick labels to vertical
    plt.xticks(rotation=90)
    plt.show()
```

8.14. Generate historic returns for strategies

```python
def weight_ew(r, cap_weights=None, max_cw_mult=None, microcap_threshold=None, **kwargs):
    """
    Returns the weights of the EW portfolio based on the asset returns "r" as a DataFrame
    If supplied a set of capweights and a capweight tether, it is applied and reweighted
    """
    n = len(r.columns)
    ew = pd.Series(1/n, index=r.columns)
    if cap_weights is not None:
        cw = cap_weights.loc[r.index[0]] # starting cap weight
        ## exclude microcaps
        if microcap_threshold is not None and microcap_threshold > 0:

            microcap = cw < microcap_threshold
            ew[microcap] = 0
            ew = ew/ew.sum()
        #limit weight to a multiple of capweight
        if max_cw_mult is not None and max_cw_mult > 0:
            ew = np.minimum(ew, cw*max_cw_mult)
            ew = ew/ew.sum() #reweight
    return ew

def weight_cw(r, cap_weights, **kwargs):
    """
    Returns the weights of the CW portfolio based on the time series of capweights
    """
    w = cap_weights.loc[r.index[11]]# Index number Must match Estimation Window!!!
    return w/w.sum()

def weight_gmv(r, cov_estimator=sample_cov, **kwargs):
    """
    Produces the weights of the GMV portfolio given a covariance matrix of the returns
    """
```

```
    est_cov = cov_estimator(r, **kwargs)
    return gmv(est_cov)

def weight_erc(r, cov_estimator=sample_cov, **kwargs):
    """
    Produces the weights of the ERC portfolio given a covariance matrix of the returns
    """
    est_cov = cov_estimator(r, **kwargs)
    return equal_risk_contributions(est_cov)

def weight_msr(r, cov_estimator=sample_cov, **kwargs):
    """
    Produces the weights of the MSR portfolio given a ret series and covariance matrix structure
    """
    est_cov = cov_estimator(r, **kwargs)
    exp_ret = annualize_rets(r, 12, **kwargs)
    return msr(0,exp_ret, est_cov)

# Create Security Weighting Scheme for Black-Litterman Portfolios
ind_blcap = pd.read_excel("mktcap_2008_2020.xlsx", sheet_name='BL_WTS', index_col=0, parse_dates=True)
ind_blcap =ind_blcap.loc[Start:End]
ind_blcap.index = pd.to_datetime(ind_blcap.index, format="%Y%m").to_period('M')
total_blcap = ind_blcap.sum(axis="columns")

ind_blweight = ind_blcap.divide(total_blcap, axis="rows")
ind_blweight = ind_blweight.iloc[0:]
ind_blweight

total_bl_return = (ind_blweight * ind_return).sum(axis="columns")
total_bl_return

total_bl_index = drawdown(total_bl_return).Wealth
total_bl_index.plot(title="BL Weighted Index");

# Specify Estimation Window
estimation_window=12

# MSR Returns (sample cov)
MSRr_sample = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_msr, cov_estimator=sample_cov)
# MSR Returns (shrink cov)
MSRr_shrink = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_msr, cov_estimator=shrinkage_cov)

# GMV Returns (sample cov)
GMVr_sample = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_gmv, cov_estimator=sample_cov)
# GMV Returns (shrink cov)
GMVr_shrink = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_gmv, cov_estimator=shrinkage_cov)

# ERC Returns (sample cov)
ERCr_sample = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_erc, cov_estimator=sample_cov)
# ERC Returns (shrink cov)
ERCr_shrink = backtest_ws(ind_return, estimation_window=estimation_window, weighting=weight_erc, cov_estimator=shrinkage_cov)

# Random Forest Strategy Returns
outsamp_test_ret = test_returns_1.iloc[-36:]
outsamp_test_ret

# Extract values, remove time-stamp
outsamp_tr_val = outsamp_test_ret.values
outsamp_tr_val

# Re-index
```

```python
outsamp_tr_ser = pd.Series(data=outsamp_tr_val)
new_index = ewr['2017-07':].index
outsamp_tr_dtser = pd.Series(data=outsamp_tr_val, index=new_index )
outsamp_tr_dtser

# Collect Out-of-Sample Returns in DataFrame
btr_outsample = pd.DataFrame({
                        "EW": ewr['2017-07':],
                        "CW": cwr['2017-07':],
                        "MSR-Sample": MSRr_sample['2017-07':],
                        "MSR-Shrink": MSRr_shrink['2017-07':],
                        "GMV-Sample": GMVr_sample['2017-07':],
                        "GMV-Shrink": GMVr_shrink['2017-07':],
                        "ERC": ERCr_sample['2017-07':],
                        "RF": outsamp_tr_dtser,
                        "B-L": total_bl_return['2017-07':]
                        })
# View DataFrame
btr_outsample

# Compute Cumulative Return
cum_ret_outsample = (1+btr_outsample).cumprod()
cum_ret_outsample

# Plot Compunded Return
(1+btr_outsample).cumprod().plot(figsize=(16,12), title="Strategies Cumulative Return");
```

8.15. Generate Performance Metrics

```python
def summary_stats(r, riskfree_rate=rf):
    """
    Return a DataFrame that contains aggregated summary stats for the returns in the columns of r
    """
    ann_r = r.aggregate(annualize_rets, periods_per_year=12)
    ann_vol = r.aggregate(annualize_vol, periods_per_year=12)
    ann_sr = r.aggregate(sharpe_ratio, riskfree_rate=riskfree_rate, periods_per_year=12)
    dd = r.aggregate(lambda r: drawdown(r).Drawdown.min())
    skew = r.aggregate(skewness)
    kurt = r.aggregate(kurtosis)
    ann_semi_dev = r.aggregate(semideviation) * math.sqrt(ann_factor)
    cf_var5 = r.aggregate(var_gaussian, modified=True)
    hist_cvar5 = r.aggregate(cvar_historic)
    rovol = ann_r/ann_vol
    ann_sortino = ann_r/ann_semi_dev
    rovar_cvar = ann_r/hist_cvar5
    rocvar_cfvar = ann_r/cf_var5
    radd = ann_r/-dd
    return pd.DataFrame({
        "Annualized Return": ann_r,
        "Annualized Volatility": ann_vol,
        "Ann. Semi-Dev.": ann_semi_dev,
        "Skewness": skew,
        "Kurtosis": kurt,
        "Modified VaR (5%)": cf_var5,
        "Historic CVaR (5%)": hist_cvar5,
        "Max Drawdown": dd,
        "Sharpe Ratio": ann_sr,
        "Sortino Ratio": ann_sortino,
          })

# Display Results
summary_stats(btr_outsample.dropna()).round(4)
```

9. Results

Figure 4: Compounded Return in Total Period

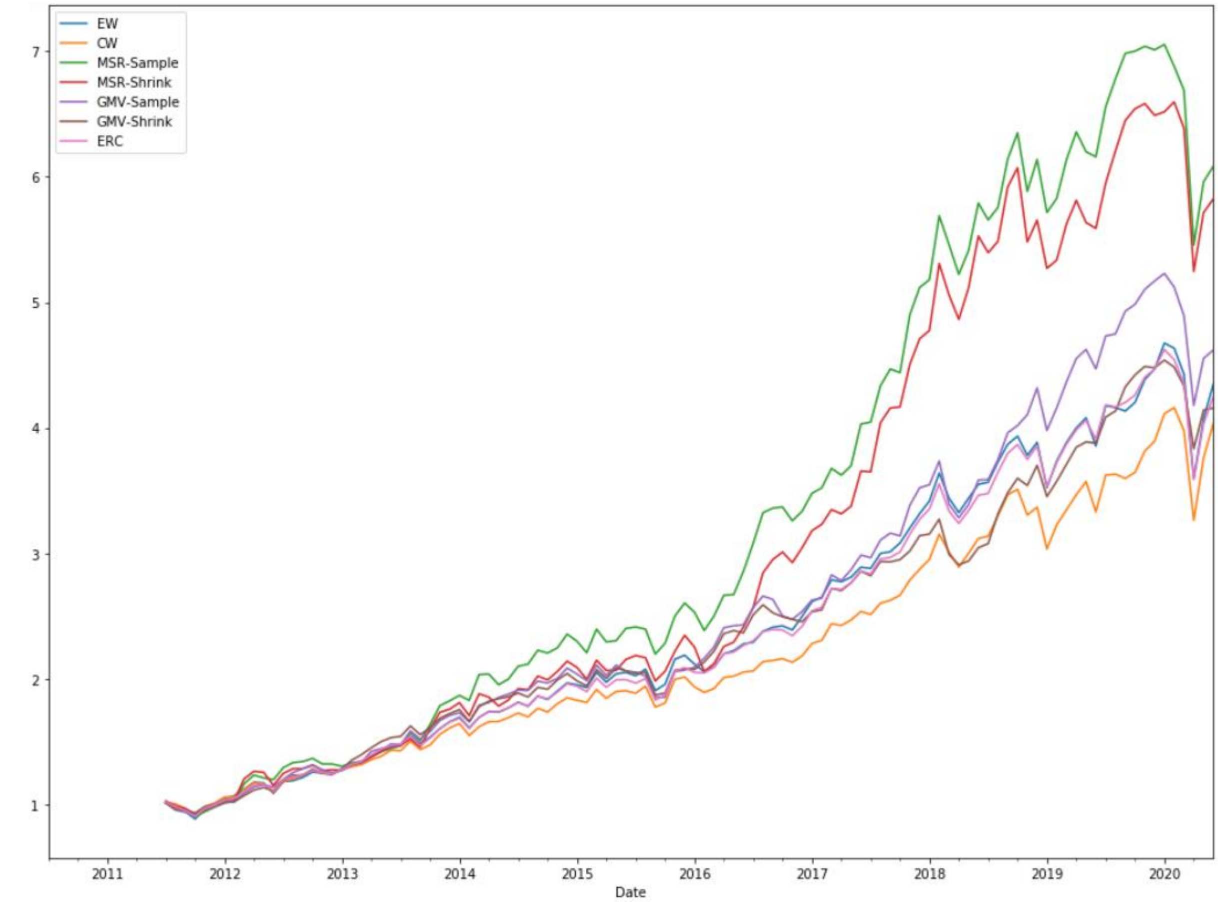


Figure 5: Performance in Total Period

	Annualized Return	Annualized Volatility	Ann. Semi-Dev.	Skewness	Kurtosis	Modified VaR (5%)	Historic CVaR (5%)	Max Drawdown	Sharpe Ratio	Sortino Ratio
EW	0.1775	0.1479	0.1267	-1.1178	7.1181	0.0643	0.0920	-0.2308	1.1997	1.4006
CW	0.1678	0.1490	0.1257	-0.8750	6.6867	0.0633	0.0918	-0.2157	1.1258	1.3348
MSR-Sample	0.2222	0.1656	0.1142	-0.5239	4.9560	0.0652	0.0857	-0.2266	1.3419	1.9462
MSR-Shrink	0.2163	0.1750	0.1227	-0.4748	4.6049	0.0699	0.0997	-0.2047	1.2359	1.7632
GMV-Sample	0.1855	0.1388	0.1072	-0.8462	4.8502	0.0581	0.0851	-0.2009	1.3368	1.7300
GMV-Shrink	0.1717	0.1251	0.0934	-0.7275	4.2430	0.0513	0.0770	-0.1553	1.3728	1.8380
ERC	0.1744	0.1372	0.1188	-1.1730	7.3293	0.0592	0.0845	-0.2239	1.2712	1.4680

Figure 6: Compounded Return in Out-of-Sample Period

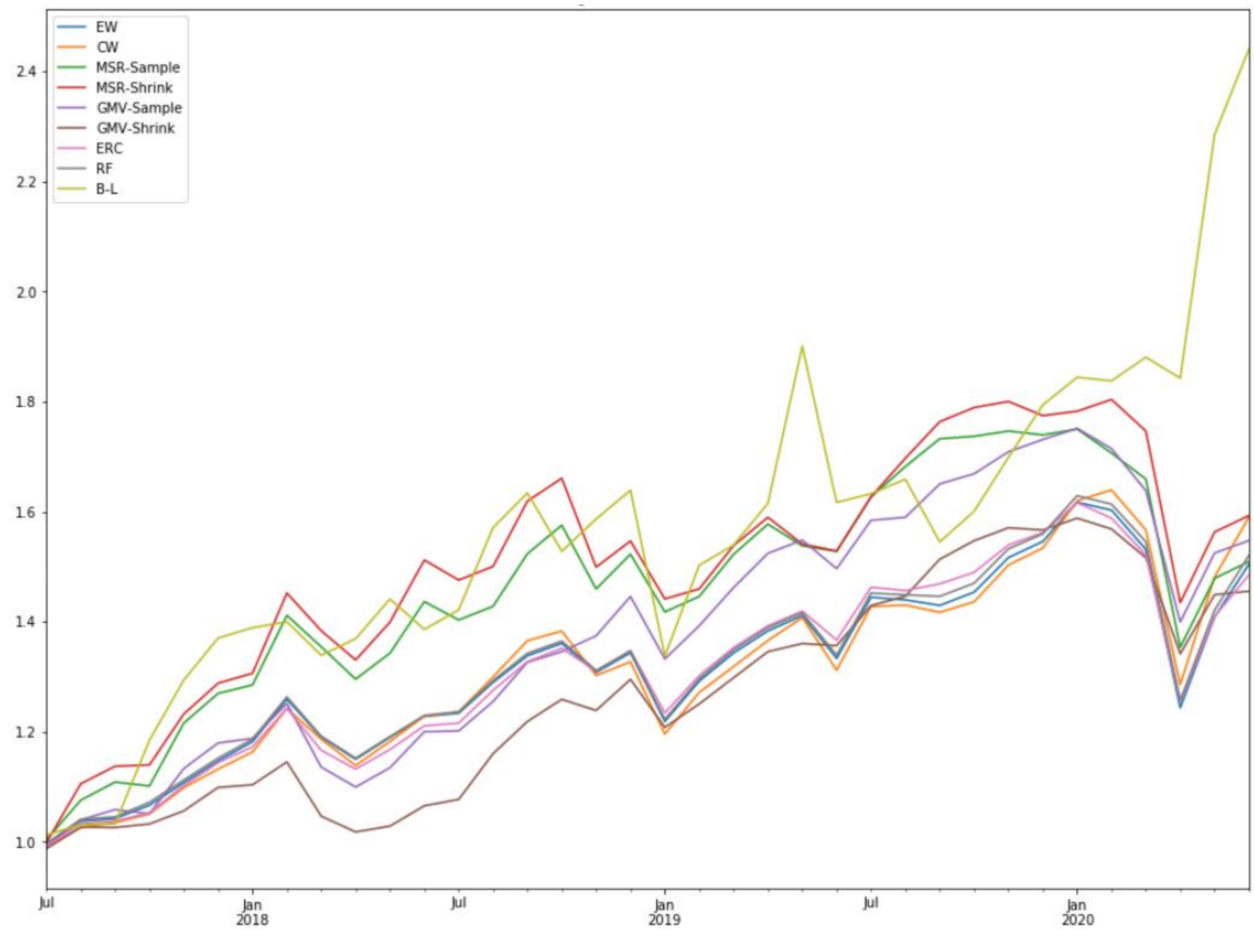


Figure 7: Performance in Out-of-Sample Period

	Annualized Return	Annualized Volatility	Ann. Semi-Dev.	Skewness	Kurtosis	Modified VaR (5%)	Historic CVaR (5%)	Max Drawdown	Sharpe Ratio	Sortino Ratio
EW	0.1458	0.1905	0.1783	-1.3372	6.6532	0.0910	0.1403	-0.2308	0.7655	0.8178
CW	0.1672	0.2000	0.1702	-0.9865	5.5801	0.0910	0.1390	-0.2157	0.8362	0.9822
MSR-Sample	0.1471	0.1912	0.1643	-1.1606	5.8530	0.0899	0.1289	-0.2266	0.7693	0.8949
MSR-Shrink	0.1678	0.1996	0.1682	-0.9945	5.0430	0.0914	0.1379	-0.2047	0.8408	0.9978
GMV-Sample	0.1567	0.1667	0.1502	-1.3291	5.1056	0.0790	0.1190	-0.2009	0.9396	1.0434
GMV-Shrink	0.1333	0.1412	0.1269	-1.1370	4.7364	0.0654	0.1006	-0.1553	0.9439	1.0504
ERC	0.1409	0.1762	0.1660	-1.3975	6.8729	0.0843	0.1298	-0.2239	0.7999	0.8489
RF	0.1499	0.1894	0.1785	-1.3476	6.6315	0.0903	0.1392	-0.2282	0.7916	0.8400
B-L	0.3464	0.2720	0.2045	-0.0809	4.7022	0.0984	0.1666	-0.1871	1.2734	1.6940

11. Conclusions

The GMV-Shrink portfolio is clearly the best performer over the total period. It suffers the lowest volatility and has the highest Sharpe ratio. The returns distribution is the least fat-tailed (kurtotic) and the second least negatively skewed resulting in the lowest Modified Value-at-Risk. It also achieves the lowest values for Conditional Value-at-Risk and Maximum Drawdown. Additionally, it achieves the lowest semi-deviation which gives it the second highest Sortino Ratio. The MSR-Sample Portfolio achieves the highest Sortino Ratio, due to a significantly higher annualized return though the investor would be obliged to assume higher dispersion of returns and significantly greater tail risk. The MSR-Shrink portfolio fails to outperform the MSR-Sample portfolio because, in the portfolio selection process, the higher mean assets returns are not adequately penalized by higher volatilities. Error maximization is more pronounced. The Equal-Weighted Portfolio outperforms the Cap-Weighted benchmark in terms of return per unit of risk, achieving superior Sharpe and Sortino Ratios. However, the investor in the EW strategy would be obliged to assume higher tail risk, as indicated by the greater values of Modified VaR, Conditional VaR and Maximum Drawdown. The performance of the Equal Risk Contribution (ERC) portfolio disappoints. It outperforms the Equal-Weighted (EW) and Cap-Weighted (CW) Indices in terms of Sortino and Sharpe Ratios though underperforms all other portfolios. Moreover, tail risk incurred is higher than that of EW and CW.

The starting 70% of the total data is used as the chronological subset to train the Random Forest model. The remaining data is the chronological subset used to test the model. The predicted portfolio weights in this out-of-sample test period are multiplied by actual security returns to generate the RF strategy returns which are then compared to those other strategies. The Black-Litterman (B-L) portfolio is constructed over this same period using the evolving explicit Price Targets available for all constituent securities. The GMV-Shrink Portfolio generates the second best Sharpe and Sortino ratios in this truncated period of elevated volatility. However, it is clearly and significantly outperformed by the Black-Litterman portfolio in these categories. Most notably, in terms of performance attribution analysis, as the Coronavirus crisis developed in 2020, the portfolio benefited from the strong returns resulting from the overweighting of Tech stocks and underweighting of Financials. In general, over the entire out-of-sample period the strong annualized return of B-L more than compensates for additional volatility and semi-deviation, resulting in the highest Sharpe and Sortino Ratios. Of additional note is that the B-L returns distribution has the lowest negative skew. The RF portfolio underperforms the Cap-Weighted Benchmark in terms of the Sharpe and Sortino Ratios and approximately equals the CW benchmark in terms of tail risk (Modified VaR, Conditional VaR, Max Drawdown).

We find evidence that both robust portfolio risk and return estimates produce portfolios capable of outperformance. It would be instructive to test the resilience of this tentative conclusion by expanding the study to encompass different time frames and international (non-US) equity markets. The under-performance of the RF portfolio should not necessarily be interpreted as a condemnation of the model but rather the feature variables (the specific technical indicators) used as inputs to the model. Further work should be done to see if volume-based or macroeconomic-orientated data could yield more favorable results.

References

[1] Markowitz, H. (1952). Portfolio selection. The Journal of Finance, 7(1), 77–91.

[2] Merton, R. C. (1972). An analytic derivation of the efficient portfolio frontier. Journal of Financial and Quantitative Analysis, 7(04), 1851–1872.

[3] Michaud, R. O. (1989). The Markowitz optimization enigma: Is "optimized" optimal? Financial Analysts Journal, 45(1), 31–42.

[4] Jobson, J.D., Korkie, B.M., (1981). Performance hypothesis testing with the Sharpe and Treynor measures. Journal of Finance 36, 889–908.

[5] Stein, C. (1956). Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, 399(1), 197–206.

[6] Best, M. J., & Grauer, R. R. (1991). On the sensitivity of mean-variance-efficient portfolios to changes in asset means: Some analytical and computational results. Review of Financial Studies, 4(2), 315–342.

[7] Chopra, V. K. (1993). Improving optimization. The. Journal of Investing, 2(3), 51–59.
[8] Chopra, V. K., & Ziemba,W. T. (1993). The effect of errors in means, variances, and covariances on optimal portfolio choice. Journal of Portfolio Management, 19(2), 6–11.
[9] James, W., and Stein, C., (1961). "Estimation with Quadratic Loss." Proceedings of the Fourth Berkeley Symposium, Vol. 1 (Berkeley, California: University of California Press), pp. 361-379
[10] Black F. and Litterman R. (1991). Asset Allocation Combining Investor Views with Market Equilibrium, Journal of Fixed Income, September 1991, Vol. 1, No. 2: pp. 7-18
[11] Black F. and Litterman R. (1991). Global Portfolio Optimization, Financial Analysts Journal, September 1992, pp. 28–43
[12] Ledoit, O., & Wolf, M. (2003). Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. Journal of Empirical Finance, 10(5), 603–621.
[13] Ledoit, O., & Wolf, M. (2004). A well-conditioned estimator for large-dimensional covariance matrices. Journal of Multivariate Analysis, 88(2), 365–411.
[14] Elton, E.J. and M.J. Gruber, (1973). Estimating the Dependence Structure of Share Prices – Implications for Portfolio Selection, Journal of Finance 28, 1203-1232.
[15] Elton, E.J., M.J. Gruber and T. Ulrich, (1978). Are Betas Best? Journal of Finance 23, 1375-1384.
[16] Maillard, S., Roncalli, T., Teiletche, J. (2010). On the properties of equally-weighted risk contributions portfolios. The Journal of Portfolio Management 36 (4) 60-70
[17] Bruder, B., and Roncalli, T. (2012). Managing Risk Exposures Using the Risk Budgeting Approach Working Paper (January 20, 2012).
[18] Breiman, L. (2001). Random Forests. Machine Learning 45, 5–32
[19] Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984) Classification and Regression Trees. Chapman and Hall, Wadsworth, New York
[20] Breiman, L. (1996a). Bagging predictors. Machine Learning 26(2), 123–140
[21] Yang, B.H., (2013): Modeling Portfolio Risk by Risk Discriminatory Trees and Random Forests. Published in: Journal of Risk Model Validation , Vol. 8, No. 1 (18 March 2014)
[22] Khaidem L., Saha S. & Roy Dey S. (2016). Predicting the direction of stock market prices using random forest. arXiv preprint arXiv:1605.00003.
[23] Hodges, S D (1997). A generalisation of the Sharpe ratio and its applications to valuation bounds and risk measures. Working paper of the Financial Options Research Centre, University of Warwick.
[24] Harlow, W. (1991). Asset Allocation in a Downside Risk Framework, Financial Analysts Journal 47(5), 28-40.