HUDSON & THAMES

hudsonthames.org

THE MODERN GUIDE TO PORTFOLIO OPTIMIZATION

V 1.0

HUDSON AND THAMES

WE ARE HUDSON & THAMES

Our mission is to promote the scientific method within investment management by codifying frameworks, algorithms, and best practices to build the world's first central repository of ready to use implementations and intellectual property.

"PortfolioLab provides access to the latest cutting edges methods. PortfolioLab is thus essential for quants who want to be ahead of the technology rather than being replaced by it."

DOCTOR THOMAS RAFFINOT

Lead Data Scientist at AXA Investment Managers

1.0

MEAN-VARIANCE OPTIMIZATION WITH PORTFOLIOLAB

INTRODUCTION

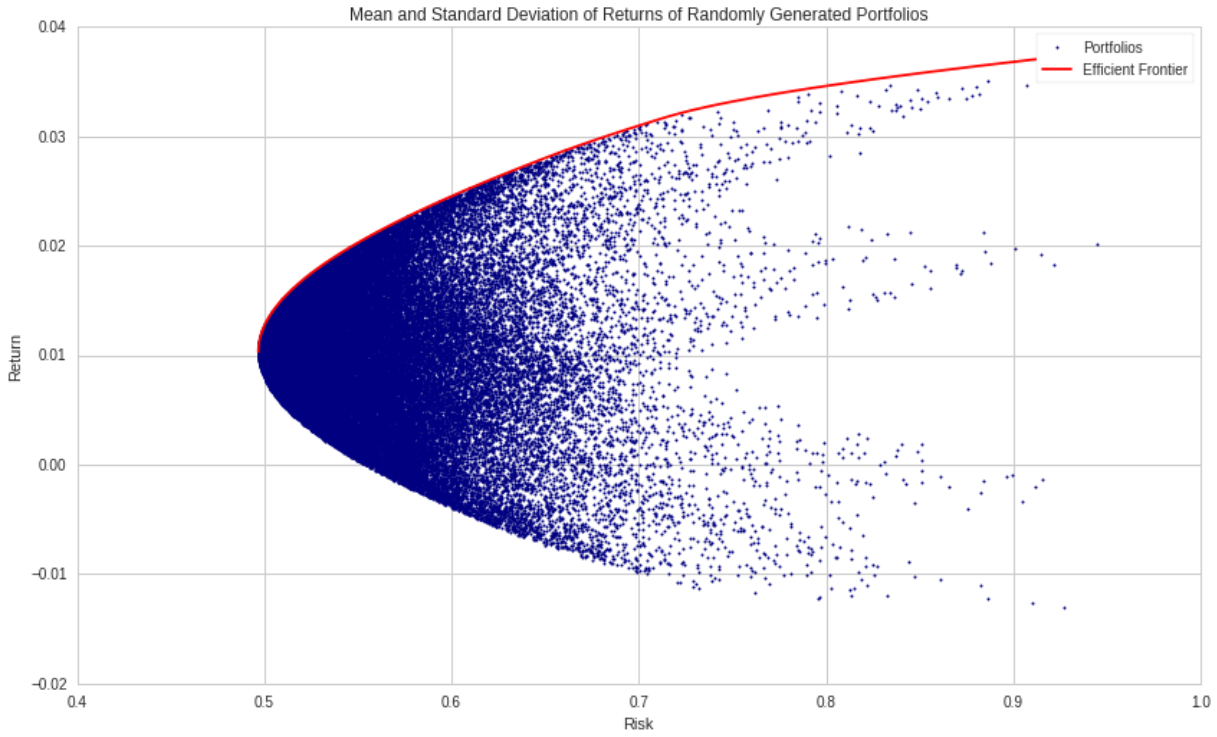


Image Credits: Abasi, Margenot and Granizo-Mackenzie, 2020

For a long while, investors worked under the assumption that the risk and return relationship of a portfolio was linear, meaning that if an investor wanted higher returns, they would have to take on a higher level of risk. This assumption changed when in 1952, Harry Markowitz introduced Modern Portfolio Theory (MPT). MPT introduced the notion that the diversification of a portfolio can inherently decrease the risk of a portfolio. Simply put, this meant that investors could increase their returns while also reducing their risk. Markowitz's work on MPT was groundbreaking in the world of asset allocation, eventually earning him a Nobel prize for his work in 1990.

Throughout this article, we will explore Markowitz's Modern Portfolio Theory and work through a full implementation in the PortfolioLab library.

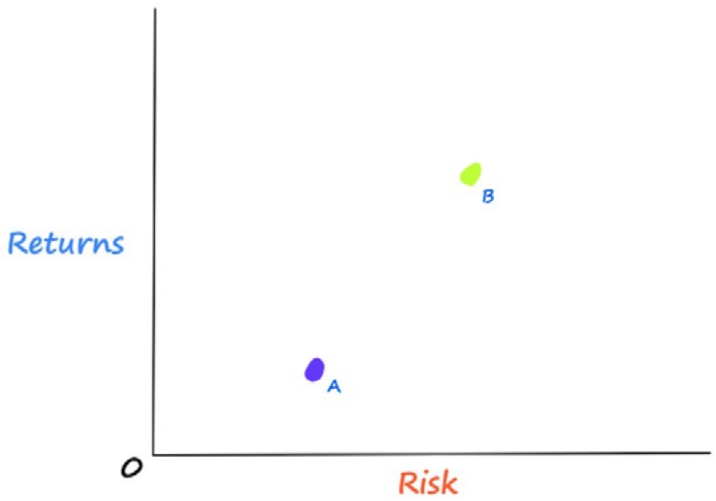
UNDERSTANDING MODERN PORTFOLIO THEORY

To understand the intuition behind MPT, we must understand that it is a theory at heart. Meaning that it relies on various assumptions which may not always be realistic. **It provides a general framework for establishing a range of reasonable expectations of which investors can use to inform their decisions.**

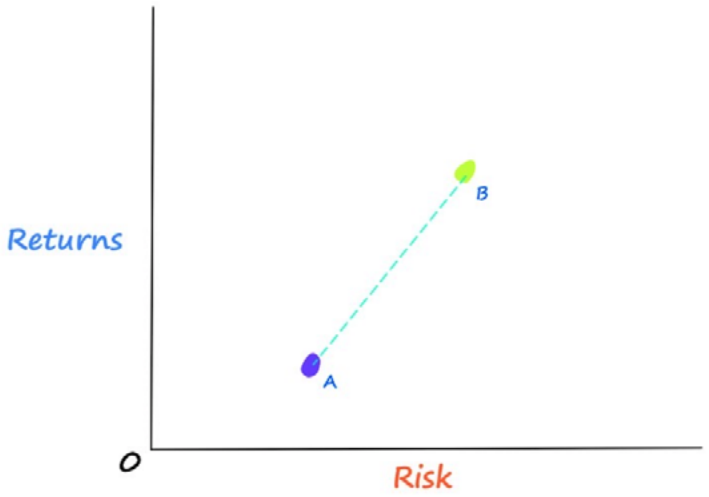
Modern Portfolio Theory is based on the following assumptions:

1. Investors wish to maximize their utility with a certain level of capital
2. Investors have access to fair and correct information of risk and returns
3. The markets are efficient
4. Investors always try to minimize risk and maximize returns
5. Investors prefer the highest returns for a specific level of risk
6. Investors base their decisions upon expected returns and the risk of assets

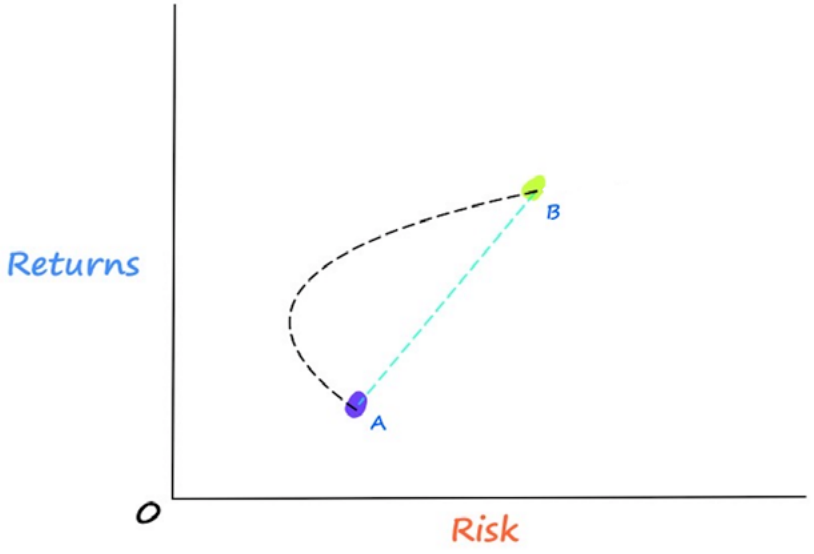
Now that we are familiar with our working assumptions, we can begin to understand the intuition behind MPT. Let's start by imagining we have two stocks in a portfolio, Stock A and Stock B.



From the image above, we can see how Stock A has a lower expected returns level and a lower risk level than Stock B. Prior to MPT, the possible portfolio combinations between these two stocks was assumed to be linear.



Assuming that there is a linear relationship between our stocks, we would think that all of our possible portfolio combinations would fall somewhere on the dotted line shown above. According to Markowitz, we would be wrong. Markowitz's theory implies that there is not a linear relationship between our stocks, and that it actually takes a curved shape called the efficient frontier. In which the efficient frontier is the true representation of all our possible portfolio combinations.



So you may now be asking, how is this possible? This efficient frontier is all made possible through a beautiful concept called correlation. To understand how correlation gives us this curved shape representing our portfolio combinations, we first need to understand how our portfolio returns and portfolio risk are calculated.

Portfolio Returns and Risk

In short, the expected return of our portfolio is the proportional sum of our individual assets' expected returns.

$$E(r_p) = \sum_{i=1}^{N} w_i * E(r_i)$$

where w_i is the weight of the i^{th} asset of our portfolio and $E(r_i)$ represents the expected return of the i^{th} asset of our portfolio.

This calculation of our portfolio's expected return follows the same intuition which would make us believe that the relationship between our two stocks would be linear. The part which makes this relationship non-linear comes when calculating our portfolio's risk.

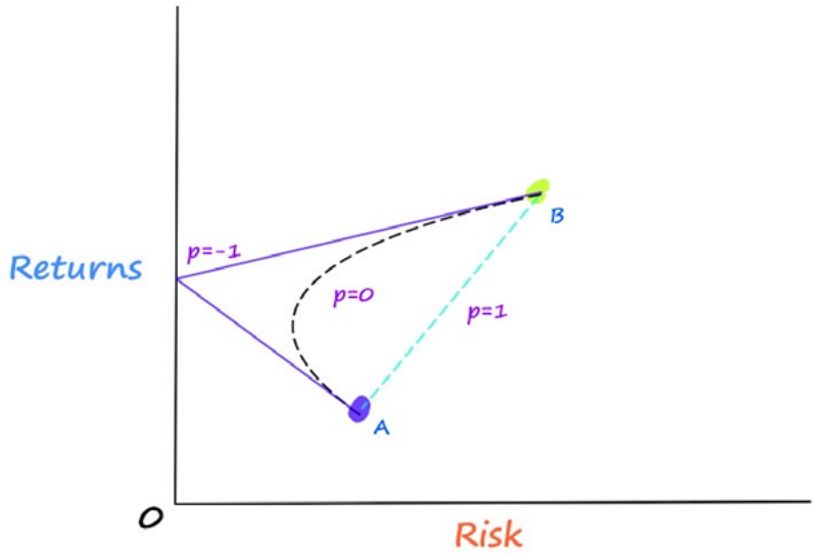
Now that we can calculate our portfolio's expected returns, we can see how to calculate our portfolio's risk, which is represented as our portfolio's variance.

$$\sigma_p^2 = w_A^2\sigma_A^2 + w_B^2\sigma_B^2 + 2w_Aw_B\sigma_A\sigma_B\rho_{A,B}$$

where $\rho_{A,B}$ represents our correlation between stocks and σ represents the standard deviation for each stock. Now that we can see how our risk equation is affected by the correlation between our two stocks, we can begin to understand why we get an efficient frontier shape for our possible portfolio combinations.

The correlation of our two assets is a number between -1 and 1 which shows us how our two stocks are related. If our correlation is positive, the returns of our two assets tend to move in the same direction, and vice versa. To calculate the correlation of Stock A and Stock B, it would look like this:

$$corr_{A,B} = \rho_{A,B} = \frac{\sigma_{A,B}}{\sigma_A\sigma_B}$$



Looking at the picture above, it showcases three different scenarios. When the correlation between our two stocks is -1, there is the largest reduction in our portfolio risk. Meanwhile, as the correlation between our stocks increases, our portfolio risk increases as well.

While this was intended as a quick primer on Modern Portfolio Theory, we hope this gives you a basic understanding of the topic for which you can further read upon. There are lots of great articles and papers written on the subject that go into the depth of the topic. The rest of this article will be geared towards implementing this portfolio optimization technique through the PortfolioLab library.

SUPPORTED PORTFOLIO ALLOCATION SOLUTIONS

The MPT or Mean-Variance Optimization encompasses a large variety of portfolio solutions with different objectives and constraints. The MeanVarianceOptimization class provides some of these commonly encountered portfolio problems out-of-the-box:

Inverse Variance

For this solution, the diagonal of the covariance matrix is used for weights allocation.

$$w_i = \frac{\sum^{-1}}{\sum_{j=1}^{N}(\sum_{j,j})^{-1}}$$

where w_i is the weight allocated to the i^{th} asset in a portfolio, $\sum_{i,i}$ is the i^{th} element on the main diagonal of the covariance matrix of elements in a portfolio and N is the number of elements in a portfolio.

Minimum Volatility

For this solution, the objective is to generate a portfolio with the least variance. The following optimization problem is being solved.

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimise}} \quad & w^T \sum w \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, .., N \end{aligned}$$

Maximum Sharpe Ratio

For this solution, the objective is (as the name suggests) to maximise the Sharpe Ratio of your portfolio.

$$\begin{aligned} \underset{\mathbf{w}}{\text{maximise}} \quad & \frac{\mu^T w - R_f}{(w^T \sum w)^{1/2}} \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, . . , N \end{aligned}$$

where y refer to the set of unscaled weights, κ is the scaling factor and the other symbols refer to their usual meanings.

The process of deriving this optimization problem from the standard maximising Sharpe ratio problem is described in the notes IEOR 4500 Maximizing the Sharpe ratio from Columbia University.

Efficient Risk

For this solution, the objective is to minimise risk given a target return value by the investor. Note that the risk value for such a portfolio will not be the minimum, which is achieved by the minimum-variance solution. However, the optimiser will find the set of weights which efficiently allocate risk constrained by the provided target return, hence the name “efficient risk”.

$$\begin{aligned} \min_{\mathbf{w}} \quad & w^T \sum w \\ \text{s.t.} \quad & \mu^T w = \mu_t \\ & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, . . , N \end{aligned}$$

where μ_t is the target portfolio return set by the investor and the other symbols refer to their usual meanings.

Efficient Return

For this solution, the objective is to maximise the portfolio return given a target risk value by the investor. This is very similar to the **efficient_risk** solution. The optimiser will find the set of weights which efficiently try to maximise return constrained by the provided target risk, hence the name "efficient return".

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mu^T w \\ \text{s.t.} \quad & w^T \sum w = \sigma_t^2 \\ & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, .., N \end{aligned}$$

where σ_t^2 is the target portfolio risk set by the investor and the other symbols refer to their usual meanings.

Maximum Return – Minimum Volatility

This is often referred to as **quadratic risk utility**. The objective function consists of both the portfolio return and the risk. Thus, minimising the objective relates to minimising the risk and correspondingly maximising the return. Here, is the risk-aversion parameter which models the amount of risk the user is willing to take. A higher value means the investor will have high defense against risk at the expense of lower returns and keeping a lower value will place higher emphasis on maximising returns, neglecting the risk associated with it.

$$\begin{aligned} \min_{\mathbf{w}} \quad & \lambda * w^T \sum w - \mu^T w \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, .., N \end{aligned}$$

Maximum Diversification

Maximum diversification portfolio tries to diversify the holdings across as many assets as possible. In the 2008 paper, Toward Maximum Diversification, the diversification ratio, D, of a portfolio, is defined as:

$$D = \frac{w^T \sigma}{\sqrt{w^T \sum w}}$$

where σ is the vector of volatilities and $\sum$ is the covariance matrix. The term in the denominator is the volatility of the portfolio and the term in the numerator is the weighted average volatility of the assets.

More diversification within a portfolio decreases the denominator and leads to a higher diversification ratio. The corresponding objective function and the constraints are:

$$\begin{aligned} \max_{\mathbf{w}} \quad & D \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, .., N \end{aligned}$$

You can read more about maximum diversification portfolio in the following article on the website Flirting with Models: Maximizing Diversification.

Maximum Decorrelation

For this solution, the objective is to minimise the correlation between the assets of a portfolio

$$\begin{aligned} \min_{\mathbf{w}} \quad & w^T A w \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j = 1 \\ & w_j \geq 0, j = 1, .., N \end{aligned}$$

where **A** is the correlation matrix of assets. The Maximum Decorrelation portfolio is closely related to Minimum Variance and Maximum Diversification, but applies to the case where an investor believes all assets have similar returns and volatility, but heterogeneous correlations. It is a Minimum Variance optimization that is performed on the correlation matrix rather than the covariance matrix, $\sum$.

You can read more on maximum decorrelation portfolio in the following article: Max Decorrelation Portfolio.

MEAN-VARIANCE OPTIMIZATION WITH PORTFOLIOLAB

In this section, we will show users how to optimize their portfolio using several mean-variance Optimization (MVO) solutions provided through the PortfolioLab Python library. Official documentation can be found at this link. The mean-variance Optimization class from PortfolioLab utilizes techniques based on Harry Markowtiz's methods for calculating efficient frontier solutions.

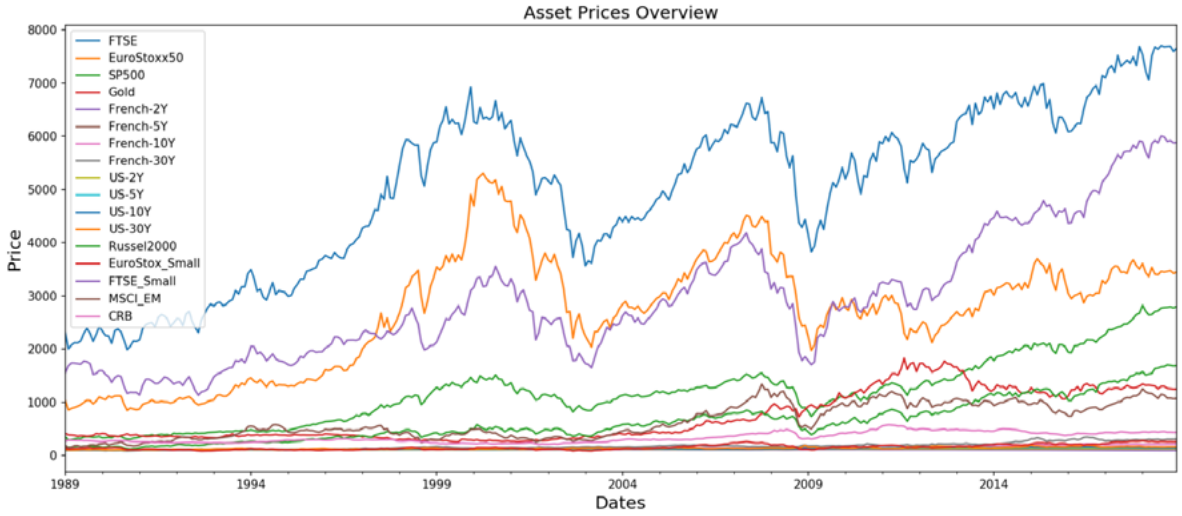
Through the PortfolioLab library, users can generate optimal portfolio solutions for different objective functions, including:

```
# importing our required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from portfoliolab.modern_portfolio_theory import MeanVarianceOptimization
```

The Data

In this tutorial, we will be working with the historical closing prices for 17 assets. The portfolio consists of diverse set of assets ranging from commodities to bonds and each asset exhibits different risk-return characteristics.

```
# preparing our data

raw_prices = pd.read_csv('../Sample-Data/assetalloc.csv', sep=';', parse_dates=True,

                         index_col='Dates')

stock_prices = raw_prices.sort_values(by='Dates')


# Taking a quick look at our most recent 5000 data points

stock_prices.resample('M').last().plot(figsize=(17,7))

plt.ylabel('Price', size=15)

plt.xlabel('Dates', size=15)

plt.title('Stock Data Overview', size=15)

plt.show()
```



Calculating some Example Portfolios

We will first work through building an optimal portfolio under the inverse-variance solution string. It is one of the simplest yet powerful allocation strategy which outperforms a lot of complex Optimization objectives.

In order to calculate the asset weights of the inverse-variance portfolio, we will need to create a new MeanVarianceOptimization object and use the allocate method. More information on inverse-variance solutions can be found here.

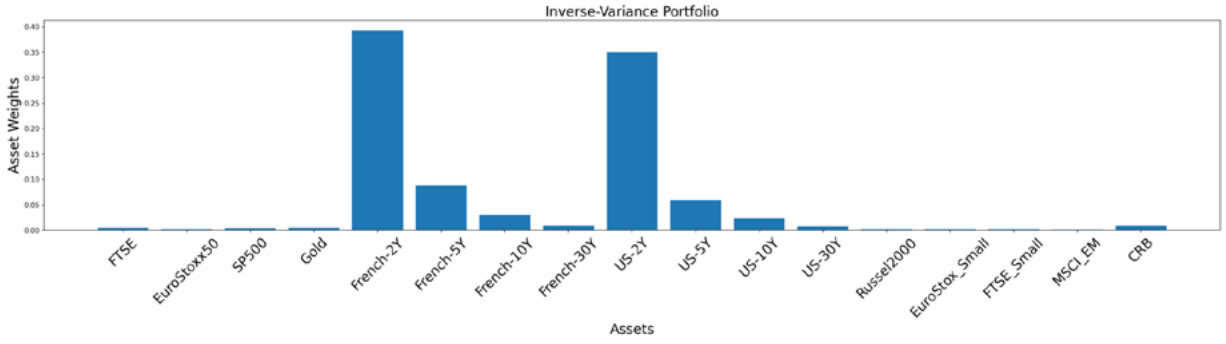
Note that the allocate method requires three parameters to run:

1. **asset_names** (a list of strings containing the asset names)
2. **asset_prices** (a dataframe of historical asset prices – daily close)
3. **solution** (the type of solution/algorithm to use to calculate the weights)

Instead of providing historical asset prices, users can also provide the expected asset returns along with a covariance matrix of asset returns. There is also much more customizability within the allocation method in which we will explore later in this article.

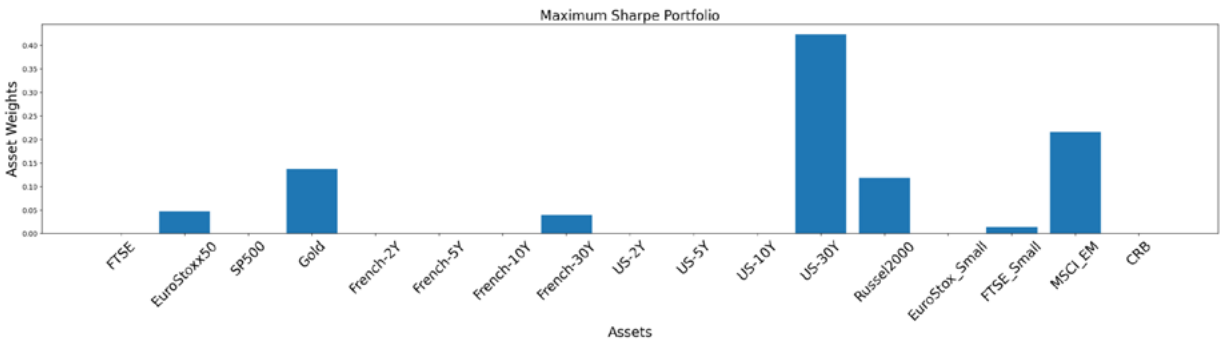
For simplicity, we will show users how to create an optimal portfolio under the Inverse Variance portfolio solution. If you wish to create a different optimized portfolio, all that must be changed is the 'solution' parameter string. The corresponding solution strings and portfolio solutions can be found at the official documentation.

```
# creating our portfolio weights under the correct objective function

mvoIV = MeanVarianceOptimization()

mvoIV.allocate(asset_names=stock_prices.columns,

               asset_prices=stock_prices,

               solution='inverse_variance')

# plotting our optimal portfolio

IV_weights = mvoIV.weights

y_pos = np.arange(len(IV_weights.columns))

plt.figure(figsize=(25,7))

plt.bar(list(IV_weights.columns), IV_weights.values[0])

plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('Inverse-Variance Portfolio', size=20)

plt.show()
```



Let us create another portfolio – the **maximum Sharpe Ratio** portfolio. This is sometimes also referred to as the tangency portfolio.

```
# Importing ReturnsEstimation class from PortfolioLab

mvoMS = MeanVarianceOptimization()

mvoMS.allocate(asset_names=stock_prices.columns,

               asset_prices=stock_prices,

               solution='max_sharpe')

# plotting our optimal portfolio

MS_weights = mvoMS.weights

y_pos = np.arange(len(MS_weights.columns))

plt.figure(figsize=(25,7))

plt.bar(list(MS_weights.columns), MS_weights.values[0])

plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('Maximum Sharpe Portfolio', size=20)

plt.show()
```



Custom Input from Users

While PortfolioLab provides many of the required calculations when constucting optimal portfolios, users also have the choice to provide custom input for their calculations. Instead of providing the raw historical closing prices for their assets, users can input a covariance matrix of asset returns and the expected asset returns to calculate their optimal portfolio. If you would like to learn more about the customizbility within PortfolioLab's MVO implementation, please refer to the official documentation.

he following parameters in the allocate() method are utilized in order to do so:

1. covariance_matrix': (pd.DataFrame/NumPy matrix) A covariance matrix of asset returns
2. 'expected_asset_returns: (list) A list of mean asset returns

To make some of the necessary calculations, we will make use of the ReturnsEstimators class provided by PortfolioLab.

```
# Importing ReturnsEstimation class from PortfolioLab
from portfoliolab.estimators import ReturnsEstimators
# Calculating our asset returns in order to calculate our covariance matrix
returns = ReturnsEstimators.calculate_returns(stock_prices)
# Calculating our covariance matrix
cov = returns.cov()
```

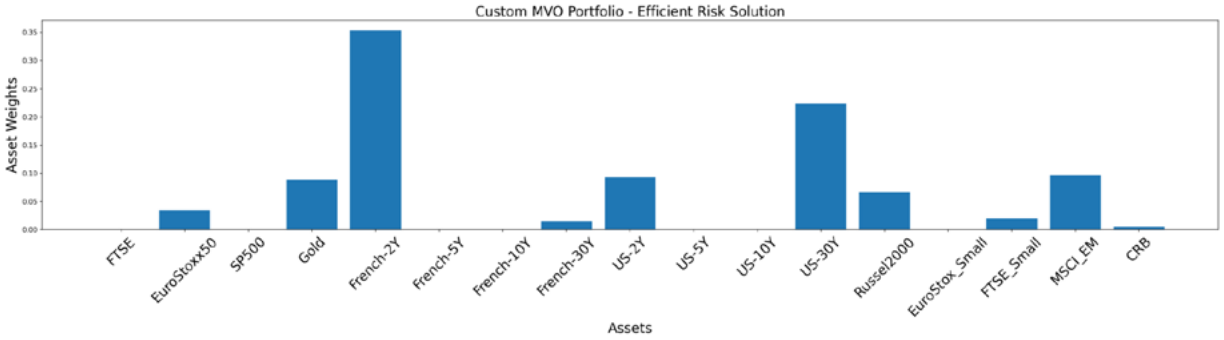
We also calculate the expected returns i.e. mean asset returns

```
# Calculating our mean asset returns
from portfoliolab.estimators import ReturnsEstimators
mean_returns = ReturnsEstimators.calculate_mean_historical_returns(stock_prices)
```

```
FTSE              0.264280
EuroStoxx50       0.433313
SP500             0.335210
Gold              0.260800
French-2Y        -0.000044
French-5Y         0.027392
French-10Y        0.064993
French-30Y        0.167084
US-2Y             0.010230
US-5Y             0.032652
US-10Y            0.064161
US-30Y            0.174726
Russel2000        0.481756
EuroStox_Small    0.460265
FTSE_Small        0.449934
MSCI_EM           0.792658
CRB               0.143230
dtype: float64
```

Having calculated all the required input matrices, we create an **efficient risk** portfolio using these custom inputs. An efficient risk portfolio tries to efficiently manage risk for a given target return by the investor. Here we are specifying a target return of 0.2

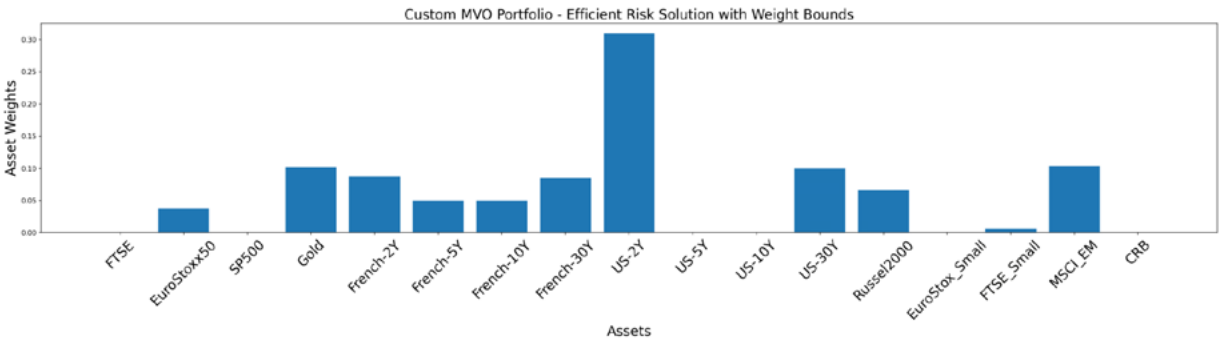
```
# From here, we can now create our portfolio

mvo_custom = MeanVarianceOptimization()

mvo_custom.allocate(asset_names=stock_prices.columns,

                    expected_asset_returns=mean_returns,

                    covariance_matrix=cov,

                    target_returns=0.2,

                    solution='efficient_risk')

# plotting our optimal portfolio

custom_weights = mvo_custom.weights

y_pos = np.arange(len(custom_weights.columns))

plt.figure(figsize=(25,7))

plt.bar(list(custom_weights.columns), custom_weights.values[0])

plt.xticks(y_pos, rotation=45, size=20)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('Custom MVO Portfolio - Efficient Risk Solution', size=20)

plt.show()
```



Additionally, a lot of investors and portfolio managers like to place specific weight bounds on some assets in their portfolio. For instance, in the above portfolio, one may want to limit the weights assigned to French-2Y and US-30Y. Let us try to decrease their allocations and diversify the portfolio a little. We place a maximum bound on their weights and a minimum bound on French-5Y and French-10Y. **Note that the indexing starts from 0.**

```
mvo_custom_bounds = MeanVarianceOptimization()

mvo_custom_bounds.allocate(asset_prices=stock_prices,

                           weight_bounds=["weights[4] <= 0.2", "weights[11] <= 0.1",

                            "weights[5] >= 0.05", "weights[6] >= 0.05"],

                           solution='efficient_risk')


# plotting our optimal portfolio

custom_bounds_weights = mvo_custom_bounds.weights

y_pos = np.arange(len(custom_bounds_weights.columns))
```

```
plt.figure(figsize=(25,7))

plt.bar(list(custom_bounds_weights.columns), custom_bounds_weights.values[0])

plt.xticks(y_pos, rotation=45, size=20)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('Custom MVO Portfolio - Efficient Risk Solution with Weight Bounds', size=20)

plt.tight_layout()

plt.savefig("mvo_custom_weight_bounds.png", dpi=150)

plt.show()
```



You can see how the weight bounds help us achieve a more diversified portfolio and prevent the weights from becoming too concentrated on only some assets.

CUSTOM PORTFOLIO ALLOCATION WITH A CUSTOM OBJECTIVE FUNCTION

PortfolioLab also provides a way for users to create their custom portfolio problem. **This includes complete flexibility to specify the input, optimization variables, objective function, and the corresponding constraints**. In this section, we will work through how to build a custom portfolio allocation with a custom objective function.

The Quadratic Optimizer

In order to solve our mean-variance objective functions, PortfolioLab uses cvxpy instead of the more frequently used scipy.optimize. This choice was made for a few reasons in particular:

1. The documentation of cvxpy is more understandable than that of scipy
2. cvxpy's code is much more readable and easier to understand
3. cvxpy raises clear error notifications if the problem we are attempting to solve is not convex and the required conditions are not met

Working with cvxpy through the PortfolioLab library, we can create our own convex optimization function to solve for portfolio allocation. In order to build this custom portfolio, we must follow the general steps:

1. Specifying our input variables not related to cvxpy
2. Specifying our cvxpy specific variables
3. Specifying our objective function
4. (Optional) Specifying the constrains for our optimization problem

Step-1: Specifying our variables not related to cvxpy

In this step we must specify our input variables not related to cvxpy (i.e. not defined as cvxpy variable objects). This can include anything ranging from raw asset prices data to historical returns to integer or string variables. All data types are supported in this step – int, float, str, Numpy matrices/lists, Python lists, Pandas DataFrame).

```
non_cvxpy_variables = {
    'asset_prices': stock_prices,
    'num_assets': stock_prices.shape[1],
    'covariance': stock_prices.cov(),
    'asset_names': stock_prices.columns,
    'expected_returns': mean_returns
}
```

Step-2: Specifying our cvxpy specific variables

The second step is to specify the cvxpy specific variables which are declared in the syntax required by cvxpy. You can include as many new variables as you need by initialising a simple Python list with each declaration being a string. **Each of these variables should be a cvxpy.Variable object**.

```
cvxpy_variables = [
        'risk = cp.quad_form(weights, covariance)',
        'portfolio_return = cp.matmul(weights, expected_returns)'
]
```

Step-3: Specifying our objective function

The third step is to specify the objective function for our portfolio optimization problem. You need to simply pass a string form of the Python code for the objective function.

```
custom_obj = 'cp.Minimize(risk)'
```

Step-4: Specifying the constraints for our optimization problem

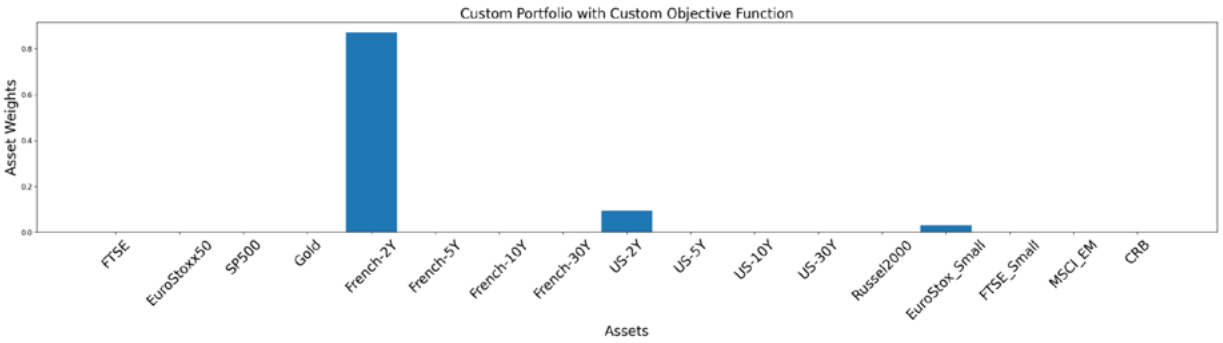
This is an optional step which requires you to specify the constraints for your optimization problem. Similar to how we specified cvxpy variables, the constraints need to be specified as a Python list with each constraint being a string representation.

```
constraints = ['cp.sum(weights) == 1', 'weights >= 0', 'weights <= 1']
```

Piecing our four parts together, we can now build our custom portfolio allocation.

```
mvo_custom_portfolio = MeanVarianceOptimization()
mvo_custom_portfolio.allocate_custom_objective(non_cvxpy_variables=non_cvxpy_variables,
                                               cvxpy_variables=cvxpy_variables,
                                               objective_function=custom_obj,
                                               constraints=constraints)
# plotting our optimal portfolio
mvo_custom_weights = mvo_custom_portfolio.weights
y_pos = np.arange(len(mvo_custom_weights.columns))
```

```
plt.figure(figsize=(25,7))

plt.bar(list(mvo_custom_weights.columns), mvo_custom_weights.values[0])

plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('Custom Portfolio with Custom Objective Function', size=20)

plt.show()
```



CONCLUSIONS

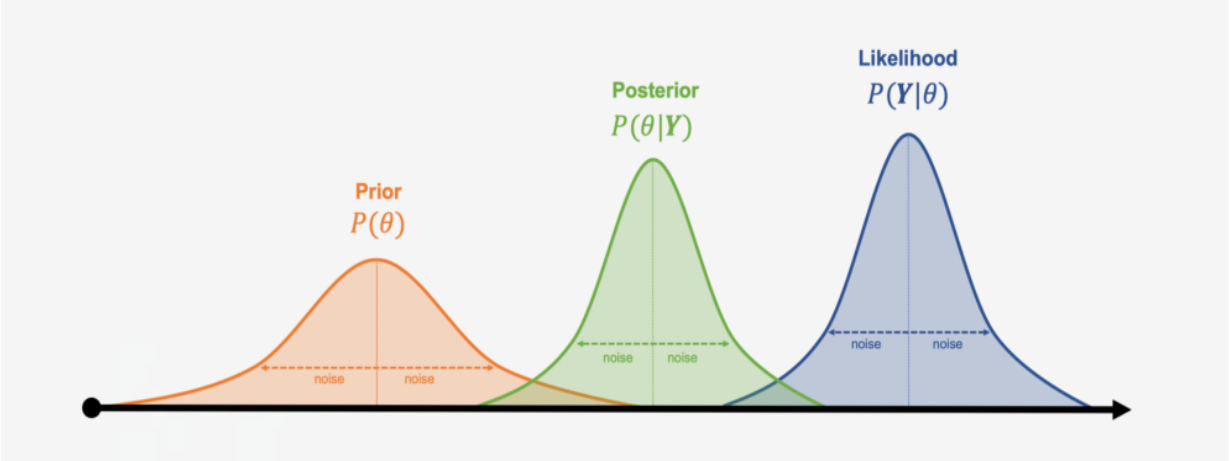
Throughout this article, we explored the intuition behind Harry Markowitz's Modern Portfolio Theory and the relationship behind portfolio risk and return and how it relates to the efficient frontier. PortfolioLab's **MeanVarianceOptimization** class provides users with the ability to create some standard portfolios out-of-the-box. At the same time, a lot of flexibility is provided for someone wanting to create their own Optimization problems with custom objective, data and constraints. If you want to read more about the literature and delve deeper into the theory, here are some links to good online resources which might be useful.

REFERENCES

1. Markowitz, Harry. "Portfolio Selection." The Journal of Finance, vol. 7, no. 1, 1952, p. 77. doi:10.2307/2975974
2. Faber, Nathan. "Maximizing Diversification." Flirting with Models, Newfound Research, 3 Dec. 2018, blog.thinknewfound.com/2018/12/maximizing-diversification/.
3. Author, et al. "Max Decorrelation Portfolio." Systematic Edge, 12 May 2013, systematicedge.wordpress.com/2013/05/12/max-decorrelation-portfolio

2.0

BAYESIAN PORTFOLIO OPTIMIZATION: INTRODUCING THE BLACK-LITTERMAN MODEL



The Black-Litterman (BL) model is one of the many successfully used portfolio allocation models out there. Developed by Fischer Black and Robert Litterman at Goldman Sachs, it combines Capital Asset Pricing Theory (CAPM) with Bayesian statistics and Markowitz's modern portfolio theory (Mean-Variance Optimization) to produce efficient estimates of the portfolio weights.

Before getting into the nitty-gritty of the algorithm it is important to understand the motivations behind developing it and why is it favored by practitioners in the industry. For a long while, investors worked under the assumption that the risk and return relationship of a portfolio was linear, meaning that if an investor wanted higher returns, they would have to take on a higher level of risk.

This assumption changed when in 1952, Harry Markowitz introduced Modern Portfolio Theory (MPT), which introduced the notion that the diversification of a portfolio can inherently decrease the risk of a portfolio. Simply put, this meant that investors could increase their returns while also reducing their risk. Markowitz's work on MPT was groundbreaking in the world of asset allocation, eventually earning him a Nobel prize for his work in 1990.

However, investors soon encountered a major problem – even though it had a sound theory supporting it, MPT failed to produce favorable results in practice. The weights produced by the models did not match with the investors' knowledge and failed to account for a lot of unexpected market variables. In short, after years of research, one can list down the following shortcomings of MPT:

1. **High Input Sensitivity**: MPT relies on the past performance of the market (returns and covariance matrix) to get an estimate of the portfolio holdings in the current market conditions. A big problem with this approach is that the performance of the past never provides a guarantee for the market situations that arise in the future. Small estimation errors in the past data leads to highly erroneous mean-variance portfolios.

2. **Highly Concentrated Portfolios**: The mean-variance procedure is found to be greedy and generates highly concentrated portfolios with only a few securities receiving the majority of allocations.

3. **Neglecting Investor Knowledge**: Finally, mean-variance optimization does not take into account an investor's personal knowledge of the market conditions and intuition. In my opinion, this point can be considered to be the most important of the previously listed points. Rather than just relying on historical data, the models should also incorporate an investor's own views of the market which is a very important asset. MVO, instead, chooses to use only historical data with a neglect for valuable market knowledge.

Reasons like the above and the underperformance of MPT in general enticed researchers to start looking for ways to improve it and come up with better models that can be used practically. The BL algorithm is one such model and although it has its own set of drawbacks, there are some unique features that make it an important part of a quant's toolkit.

The rest of the article is structured as follows: In the first section, we explore an in-depth step-by-step understanding of the Black-Litterman algorithm. In the next section, we will see it in action by using PortfolioLab's BL implementation and replicate the results of the original paper. This will be followed by a conclusion and references for further reading. You can click below on each item to go to the respective section.

BEHIND THE SCENES: INTO THE MATH OF BLACK-LITTERMAN

As mentioned previously, the Black-Litterman model relies heavily on Bayesian theory to generate its portfolios. We all know the classic Bayes formula equation:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

where, without going into much details, P(A) is the prior distribution of A, P(B|A) is the likelihood distribution of B given A and P(A|B) is the articleerior distribution of A. All 3 of these together – prior, likelihood, and articleerior – make up a Bayesian model and this is exactly what makes up the Black-Litterman model too.

It has always been said that the Black-Litterman model is based on Bayesian theory but the literature on how it is exactly connected to the Bayes equation above is very sparse. In my opinion, it is much easier to get a hang of BL when it is broken down into the individual Bayesian elements of prior, likelihood, and articleerior.

In this section, we will try to present it in a proper Bayesian framework with the appropriate derivations wherever necessary and hopefully, this will make it easier and more intuitive to understand.

THE PRIOR: IMPLIED EXCESS EQUILIBRIUM RETURNS

The model uses market equilibrium returns as the prior. But what exactly are these “equilibrium” returns? In simple words, they are the set of returns obtained by holding a CAPM market portfolio – one which theoretically contains all the assets in the universe. You will be familiar with the famous CAPM equation:

$$E(r_i) = r_f + \beta_i(E(r_{market}) \ - \ r_f)$$

where,

- r_f is the risk-free rate of return
- $E(r_{market})$ is the expected rate of return of the market portfolio
- β_i is a market uncertainty variable which quantifies the variance in the return of the i^{th} asset. It is given by $\beta_i = \frac{\sigma_{i,market}}{\sigma^2_{market}}$ where$\sigma_{i,market}$ is the covariance between the i^{th} asset and market portfolio and σ^2_{market} is the variance of the market portfolio.

Rearranging the above equation we get,

$$\Pi = \beta_i(E(r_{market}) \ - \ r_f)$$

where, Π is the excess equilibrium returns that we are interested in calculating. They are returns in excess of the risk-free market return.

The problem is how to find these returns? Fischer Black and Robert Litterman came up with a very cool trick to calculate them: **reverse Optimization.** Theoretically, in the CAPM universe all investors hold the market portfolio which contains every risky asset in the universe. Hence, one can use the market capitalisation of the assets for getting the appropriate weights allocations of the market portfolio.

$$w^i_{market} = \frac{MC_i}{\sum_{j=1}^{N} MC_j}$$

where N is the number of assets in the portfolio and MC_i is the market-cap of the i^{th} asset.

Now, for any type of portfolio, an investor’s risk utility is given by the following equation:

$$R \equiv w^T_{market}\Pi \ - \ \frac{\delta}{2} w^T_{alignat*}\Sigma w_{market}$$

where,

- R is the risk utility.
- w_{market} is the vector of market portfolio weights – already calculated as discussed above.
- Π is the vector of excess equilibrium weights.
- δ is the risk aversion parameter.
- Σ is the covariance matrix of asset returns.

The goal of any portfolio Optimization problem is to find the set of weights that minimise the risk-utlity for an investor. Now, the hessian of R is given by:

$$\frac{\partial^2 R}{\partial w_{market}^2} = -\delta\Sigma < 0$$

This quantity will be negative because Σ is positive-definite which also leads to the R being a concave function. Under no constraints, the risk utility will have a global maximum which will give us a least value of R. We differentiate R w.r.t w and equate it to 0

$$\frac{\partial R}{\partial w_{market}} = \Pi \ - \ \delta\Sigma w_{market} = 0$$

$$w_{market} = (\delta\Sigma)^{-1}\Pi$$

Remember that we already have a way of calculating the market weights using the market-cap of the assets, which allows us to **reverse** the above equation and find the desired equilibrium returns

$$\Pi = \delta\Sigma w_{market}$$

This gives us our first distribution of the BL Bayesian model - D_{prior}

$$\boxed{D_{prior} \equiv N(\Pi, \tau\Sigma)}$$

where Π and Σ carry their usual meanings and τ is a constant of proportionality. The value of τ has been a source of great confusion among practitioners since the original paper also does not clarify its exact significance and what values does it take. We will not go into a lot of details and you will find a lot of papers and articles which explore its significance. It suffices to say that its value is always close to 0 and different papers use different values depending on the use-case.

THE LIKELIHOOD: MARKET VIEWS

The second step is to nail the likelihood distribution. In my opinion, this is one of the most important and ingenious aspects of the BL algorithm and you will soon see why. As we discussed earlier, one of the criticisms of the mean-variance theory was its inability to incorporate investor knowledge. This is important since it allows the weights to be influenced by the ever-changing market factors which an investor can observe and feed into the model.

For the purpose of this section, let us assume that we have a portfolio comprising of the following assets – Apple, Google, Microsoft, and Tesla – and we have some important market views about some of them. There are 3 main components that will allow investors to incorporate them into the model:

Views Vector: Q

The views vector contains information about the specific values pertaining to the market views. Let me explain this using an example. In the Black-Litterman world, there are two types of views:

- **Absolute Views**: These are views that correspond to the scenario where an investor has assumptions on specific values of the market factors. Considering our example portfolio, let's say we observe the current market conditions and develop the following views:
    - Apple will yield a monthly expected return of 10%.
    - Microsoft will only give a monthly return of 2%
- **Relative Views**: As the name suggests, these other type of views rely on relative performance comparisons between the portfolio assets. So, rather than specific estimates on the performance of Google and Tesla, we have the following view:
    - Google will outperform Tesla (in terms of monthly expected returns) by 6%

All of these views are then sent as input to the BL model through a $Kx1$ vector Q, where K refers to the number of views

$$\begin{matrix} \textbf{View - 1} \\ \textbf{View - 2} \\ \textbf{View - 3} \end{matrix} \begin{bmatrix} 0.1 \\ 0.02 \\ 0.06 \end{bmatrix}$$

Pick Matrix: P

With the view vector giving information about the specific view values, the pick matrix tells the model which assets from our portfolio are involved in each of these views. For N assets in our portfolio, it is a KxN matrix which looks like this:

	Apple	Google	Microsoft	Tesla
View - 1	1	0	0	0
View - 2	0	0	1	0
View - 3	0	1	0	-1

Each row represents a view and the columns represent the assets in our portfolio. When only absolute views are involved, the asset involved in the view receives +1 while all others are 0. However, for relative views, the asset with better-expected performance (in our case Google) receives +1 while the other -1. This helps the model distinguish the outperforming assets from the underperforming ones. However, all this is well and good when there are at most 2 assets in our views, but what happens when there are more than 2 assets involved e.g. one might want to compare assets in one sector with those in another sector.

Let's say we also have the following view:

- Apple and Microsoft together will outperform Google and Tesla by 5%

An obvious choice is to assign +1 to Apple-Microsoft, -1 to Google-Tesla, and then divide it equally among them. So, individually, Apple and Microsoft get +0.5 while Google and Tesla receive -0.5. The problem is there is no universally accepted method for specifying the pick values and different researchers have proposed different ways of doing so.

Litterman (2003, pg 82) uses a percentage value for the pick matrix while Satchell and Scowcroft (2000) employ an equal weighting scheme which is what we just described and also shown in the image above. However, we prefer the market-capitalization based assignment proposed in Idzorek (2004) – the weightings are assigned in proportion to the market-cap of the asset divided by the total market-cap of the sector/group. Based on this method, we get the following pick values: (I have considered the approximate market-cap of these assets at the time of writing)

$$p_{AAPL} = \frac{MC_{AAPL}}{MC_{AAPL} + MC_{MSFT}} = 0.541$$

$$p_{MSFT} = \frac{MC_{MSFT}}{MC_{AAPL} + MC_{MSFT}} = 0.459$$

$$p_{GOOGL} = \frac{MC_{GOOGL}}{MC_{GOOGL} + MC_{TSLA}} = 0.745$$

$$p_{TSLA} = \frac{MC_{TSLA}}{MC_{GOOGL} + MC_{TSLA}} = 0.255$$

This is a better approach than the equal weighting scheme since it takes into account the relative importance of the assets within the outperforming and underperforming groups. So, even though the Apple-Microsoft group outperforms the other 2 assets, Apple with its larger market-cap has a more profound effect than Microsoft and the market-capitalisation based scheme helps the model capture this information.

Omega Matrix: Ω

The final component is to take into account the error in investor judgment because, in the real world, no one can have market views with 100% confidence. Hence, a variance factor comes into play here which the model uses to adjust the final weights. Out of the 3 components discussed here – Q, P and Ω – the omega matrix can be considered the most important and also one of the most complex aspects of the BL model. It is a KxK diagonal matrix where the diagonal values represent the variance in the corresponding views.

$$\Omega = \begin{bmatrix} \omega_1 & 0 & 0 & . & . & 0 \\ 0 & \omega_2 & 0 & . & . & 0 \\ 0 & 0 & . & . & . & 0 \\ 0 & . & . & . & . & 0 \\ 0 & . & . & . & . & 0 \\ 0 & . & . & . & . & \omega_k \end{bmatrix}$$

Thus one can think of the investor's expected return vector $E_{investor}(r)$ to be described by Q and an error component ϵ

$$\begin{aligned} E_{investor}(r) &\sim Q \ + \ \epsilon \\ \epsilon &\sim \ N(0, \Omega) \end{aligned}$$

The error in views has 0 mean indicating no bias in an investor's assumptions towards any particular assets in the portfolio. Another assumption is that all the error terms are uncorrelated and independent of each other – the reason for having a diagonal matrix.

Now an obvious question is how to create this matrix? Although the original paper does not give any details about a particular method, there has been a lot of research in this area and different papers have proposed varying ways of dealing with it. In this section, we will describe the most commonly used omega calculation method which was first proposed in He-Litterman (1999)

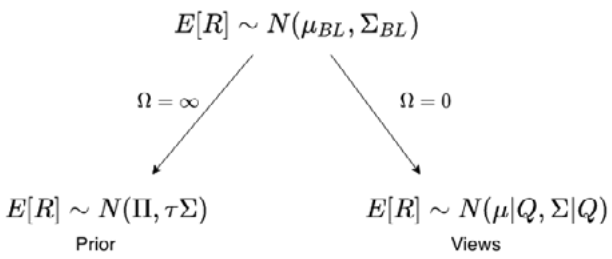
$$\Omega = diag(P \tau \Sigma P^T)$$

where, $diag()$ indicates creating a diagonal matrix from the diagonal elements of the target matrix and τ is a constant of proportionality whose values range from 0 to 1 (In the paper the authors use a $\tau = 0.05$). The assumption here is that the variance in the view errors will be proportional to prior variance and thus can be estimated only from the prior.

Combining all these components gives us our likelihood distribution $D_{likelihood}$:

$$\boxed{D_{likelihood} \equiv N(Q, \Omega)}$$

The BL articleerior portfolio always oscillates between two extremes – one scenario where the investor has 100% confidence in the views ($\Omega = 0$) and the other where he/she has no confidence in the specified views ($\Omega = \inf$).



When the investor has zero confidence in their views, the prior distribution dominates, and the articleerior portfolio is shifted towards the prior. On the other hand, when the views are specified with 100% confidence, the articleerior distribution of the parameters is dominated by only the views, neglecting any prior knowledge.

THE POSTERIOR: THE MASTER FORMULA

Combining both the prior and likelihood gives us the following Black-Litterman articleerior distribution

$$\boxed{\begin{aligned} &D_{BL} \equiv N(\mu_{BL}, \Sigma_{BL}) \\ &\mu_{BL} = ((\tau\Sigma)^{-1} + P^T\Omega^{-1}P)^{-1}((\tau\Sigma)^{-1}\Pi + P^T\Omega^{-1}Q) \\ &\Sigma_{BL} = ((\tau\Sigma)^{-1} + P^T\Omega^{-1}P)^{-1} \end{aligned}}$$

The above set of formulae for the mean and covariance is known as the Master Formula. It looks quite complex but the calculation of the equations is a simple application of the Bayes Rule. In this section we will be going into the details of its derivation and it will involve some math, so feel free to skip over to the next section if you want.

In the Black-Litterman world, the first assumption is that market returns R are normally distributed with a true hidden mean μ and covariance Σ.

$$R \sim N(\mu, \Sigma)$$

Of course, these true hidden parameters can never be known with certainty but we can always try to predict their underlying distribution and ultimately the distribution of returns. With respect to this, let us treat the mean

as a random variable $\mathbf{E(r)}$. This gives us the following two assumptions:

Without any constraints, the prior distribution acts as the sampling distribution for $\mathbf{E(r)}$

$$\mathbf{E(r)} \sim D_{prior} \equiv N(\Pi, \tau\Sigma)$$

Conditioned upon the prior, one can get sample mean vectors from the likelihood distribution.

$$\mathbf{PE(r)} \mid \mathbf{E(r)} \sim D_{likelihood} \equiv N(Q, \Omega)$$

Note that P here refers to the pick matrix and acts as an indexing matrix to select those assets of $\mathbf{E(r)}$ for which the investor has views.

For a specific mean vector $e(r)$ sampled from $\mathbf{E(r)}$, the probability density functions of prior and likelihood are a direct application of the pdf of multivariate normal distribution:

$$f_{\mathbf{PE(r)}|\mathbf{E(r)}}(Pe(r)) = \frac{|\Omega|^{-1/2}}{(2\pi)^{N/2}} exp(-\frac{1}{2}(Pe(r) - Q)^T(\Omega)^{-1}(Pe(r) - Q))$$

$$f_{\mathbf{PE(r)}|\mathbf{E(r)}}(Pe(r)) = \frac{|\Omega|^{-1/2}}{(2\pi)^{N/2}} exp(-\frac{1}{2}(Pe(r) - Q)^T(\Omega)^{-1}(Pe(r) - Q))$$

The articleerior distribution $\mathbf{E(r)} \mid \mathbf{PE(r)}$ can be calculated using the Bayes Rule. The equation needs to be reformatted so that the left hand side is proportional to a conditiona pdf $f_{\mathbf{E(r)}|\mathbf{PE(r)}}$ times a factor. The conditional aspect of the equation will be the required pdf and will give us the values for the articleerior mean and covariance.

$$f_{\mathbf{E(r)}|\mathbf{PE(r)}}(e(r)) = \frac{f_{\mathbf{E(r)}}(e(r)) f_{\mathbf{PE(r)}|\mathbf{E(r)}}(Pe(r))}{f_{\mathbf{PE(r)}}(Pe(r))}$$

$$\frac{f_{\mathbf{E(r)}}(e(r)) f_{\mathbf{PE(r)}|\mathbf{E(r)}}(Pe(r))}{\int f_{\mathbf{E(r)}}(e`(r)) f_{\mathbf{PE(r)}|\mathbf{E(r)}}(Pe`(r))\, de`(r)}$$

$$\propto exp(-\frac{1}{2}((e(r) - \Pi)^T (\tau\Sigma)^{-1} (e(r) - \Pi) + (Pe(r) - Q)^T (\Omega)^{-1} (Pe(r) - Q)))$$

All the other terms $-(2\pi)^{-N/2}$, $|\tau\Sigma|^{-1/2}$, $|\Omega|^{-1/2}$ and the integral denominator – do not contribute to the final conditional pdf terms and can be ignored as as a proportionality factor. Let us focus on the term within the exponential and break it down to get the desired form.

$$\begin{aligned} f_{\mathbf{E(r)}|\mathbf{PE(r)}}(e(r)) \propto exp(-\frac{1}{2}(&e(r)^T (\tau\Sigma)^{-1} e(r) - 2\Pi^T (\tau\Sigma)^{-1} e(r) + \Pi^T (\tau\Sigma)^{-1} \Pi \\ &+ e(r)^T P^T \Omega^{-1} P e(r) - 2Q^T \Omega^{-1} P e(r) + Q^T \Omega^{-1} Q)) \end{aligned}$$

Notice that $e(r)^T (\tau\Sigma)^{-1} \Pi$ is symmetric with $\Pi^T (\tau\Sigma)^{-1} e(r)$ and $e(r)^T P^T \Omega^{-1} Q$ is symmetric with $Q^T \Omega^{-1} P e(r)$. We can combine them together to simplify the equation further,

$$\begin{aligned} f_{\mathbf{E(r)}|\mathbf{PE(r)}}(e(r)) &\propto exp(-\frac{1}{2}(e(r)^T (\tau\Sigma)^{-1} e(r) - 2\Pi^T (\tau\Sigma)^{-1} e(r) + \Pi^T (\tau\Sigma)^{-1} \Pi \\ &\quad + e(r)^T P^T \Omega^{-1} P e(r) - 2Q^T \Omega^{-1} P e(r) + Q^T \Omega^{-1} Q)) \\ &\propto exp(-\frac{1}{2}(e(r)^T \left[(\tau\Sigma)^{-1} + P^T \Omega^{-1} P\right] e(r) - 2\left[\Pi^T (\tau\Sigma)^{-1} + Q^T \Omega^{-1} P\right] e(r) \\ &\quad + \Pi^T (\tau\Sigma)^{-1} \Pi + Q^T \Omega^{-1} Q)) \end{aligned}$$

To make it easy to follow the steps, let us make the following substitutions

$$\begin{aligned} A &= (\tau\Sigma)^{-1} \Pi + P^T \Omega^{-1} Q \\ B &= (\tau\Sigma)^{-1} + P^T \Omega^{-1} P \\ C &= Q^T \Omega^{-1} Q + \Pi^T (\tau\Sigma)^{-1} \Pi \end{aligned}$$

Continuing from where we left off,

$$f_{\mathbf{E(r)|PE(r)}}(e(r)) \propto exp(-\frac{1}{2}(e(r)^T Be(r) - 2A^T e(r) + C))$$

We make use of the following facts: B is symmetric with $B = B^T$ and$B^{-1}B = I$, where $I$is the identity matrix.

$$\begin{aligned}
&\propto exp(-\frac{1}{2}((Be(r))^T B^{-1} Be(r) - 2A^T B^{-1} Be(r) + C)) \\
&\propto exp(-\frac{1}{2}((Be(r))^T B^{-1} Be(r) - 2A^T B^{-1} Be(r) + C - A^T B^{-1} A + A^T B^{-1} A)) \\
&\propto exp(-\frac{1}{2}((Be(r) - A)^T B^{-1} (Be(r) - A) + C - A^T B^{-1} A)) \\
&\propto exp(-\frac{1}{2}((e(r) - B^{-1} A)^T B (e(r) - B^{-1} A) + C - A^T B^{-1} A)) \\
&\propto exp(-\frac{1}{2}((e(r) - B^{-1} A)^T B (e(r) - B^{-1} A)) exp(-\frac{1}{2}(C - A^T B^{-1} A)) \\
f_{\mathbf{E(r)|PE(r)}}(e(r)) &\propto exp(-\frac{1}{2}((e(r) - B^{-1} A)^T B (e(r) - B^{-1} A))
\end{aligned}$$

If we compare the above result with the pdf equation for a multivariate normal distribution, then the mean and covariance reveal themselves

$$\begin{aligned}
\mu &= B^{-1} A = ((\tau\Sigma)^{-1} + P^T \Omega^{-1} P)^{-1} ((\tau\Sigma)^{-1} \Pi + P^T \Omega^{-1} Q) \\
\Sigma &= B^{-1} = ((\tau\Sigma)^{-1} + P^T \Omega^{-1} P)^{-1}
\end{aligned}$$

If we compare the above result with the pdf equation for a multivariate normal distribution, then the mean and covariance reveal themselves

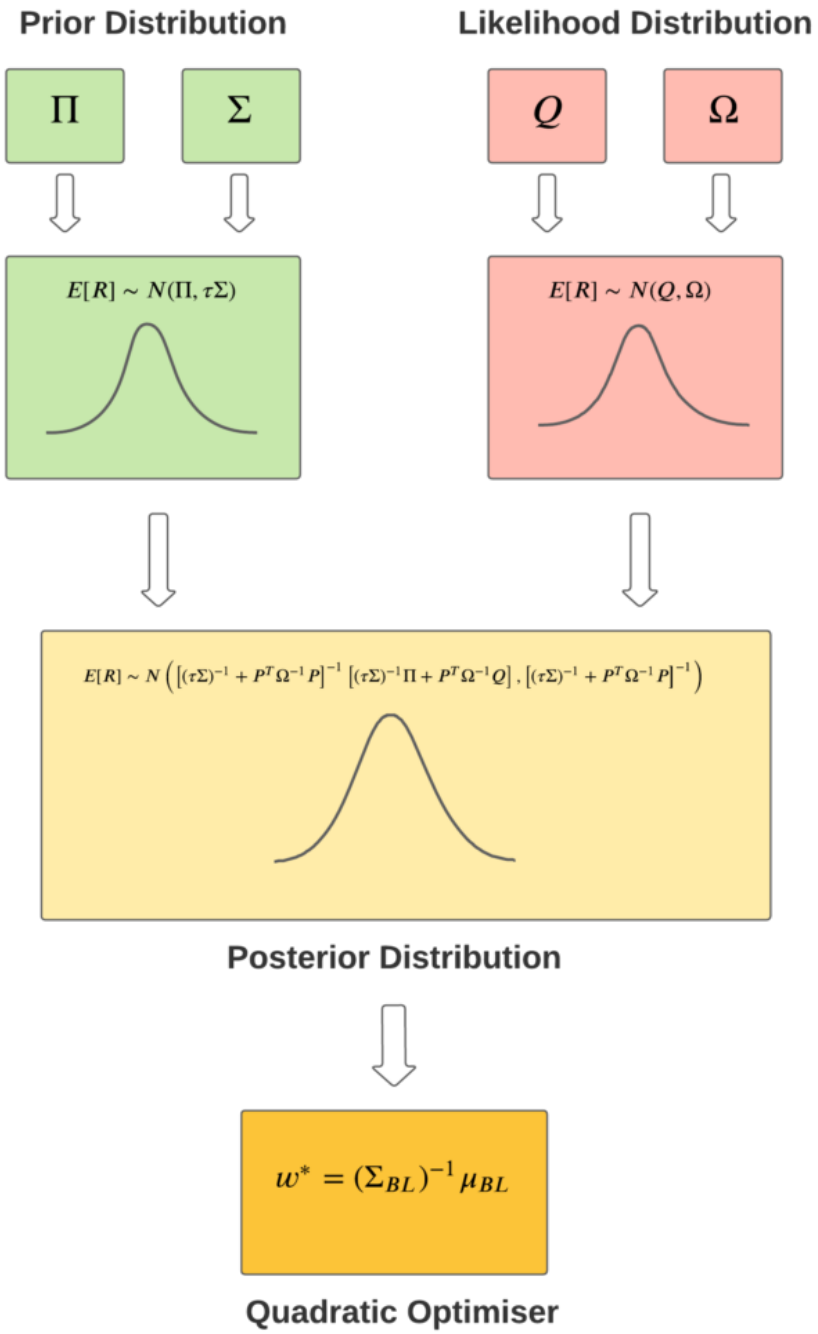
Optimal Portfolio Weights

The final weights can be found by simply plugging the articleerior mean and covariance into a standard mean-

$$w^* = (\Sigma_{BL})^{-1} \mu_{BL}$$

Note that the new mean and covariance can be passed into any custom Optimization problem and the above equation is true only for a no constraints problem.

This completes the working of the Black-Litterman algorithm. You can see that the views and prior distributions play a very important role in how the final BL portfolio is formed. The following graphic represents the Black-Litterman model in its entirety.



IN PRACTICE: BLACK-LITTERMAN USING PORTFOLIOLAB

This completes the working of the Black-Litterman algorithm. You can see that the views and prior distributions play a very important role in how the final BL portfolio is formed. The following graphic represents the Black-Litterman model in its entirety. Now that we have looked at the underlying theory of the BL model, let us run it on real-world financial data. The Black-Litterman model is available in the portfolio Optimization module of PortfolioLab. It is the latest offering from Hudson and Thames – a library dedicated for portfolio Optimization. You can install it from PyPi

```
pip install portfoliolab
```

Next up we import the Black-Litterman implementation:

```
from portfoliolab.bayesian import VanillaBlackLitterman
```

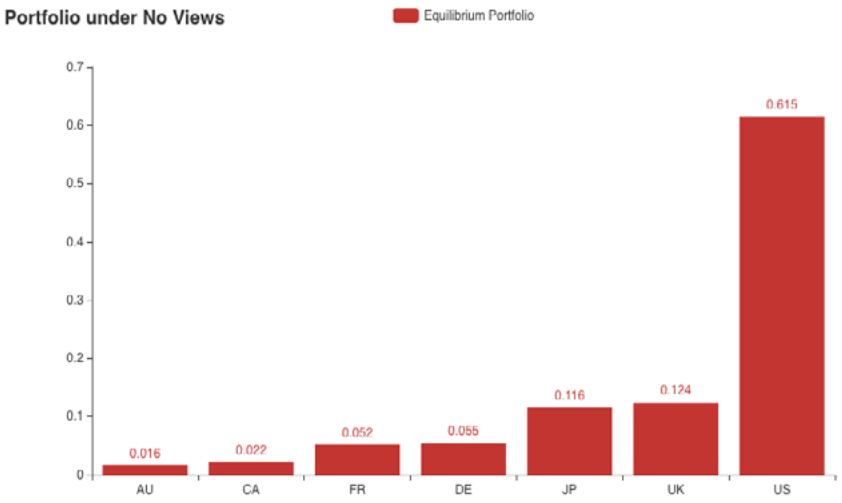
We will be using the model to replicate results from the He-Litterman paper – The Intuition Behind Black-Litterman Model Portfolios. In this paper, the authors test the BL algorithm on a market consisting of equity indexes of seven major industrial countries. It is a small dataset but helps to understand the working of the

```
countries = ['AU', 'CA', 'FR', 'DE', 'JP', 'UK', 'US']

# Table 1 of the He-Litterman paper: Correlation matrix

correlation = pd.DataFrame([

    [1.000, 0.488, 0.478, 0.515, 0.439, 0.512, 0.491],

    [0.488, 1.000, 0.664, 0.655, 0.310, 0.608, 0.779],

    [0.478, 0.664, 1.000, 0.861, 0.355, 0.783, 0.668],

    [0.515, 0.655, 0.861, 1.000, 0.354, 0.777, 0.653],

    [0.439, 0.310, 0.355, 0.354, 1.000, 0.405, 0.306],

    [0.512, 0.608, 0.783, 0.777, 0.405, 1.000, 0.652],

    [0.491, 0.779, 0.668, 0.653, 0.306, 0.652, 1.000]

], index=countries, columns=countries)
```

```
# Table 2 of the He-Litterman paper: Volatilities
volatilities = pd.DataFrame([0.160, 0.203, 0.248, 0.271, 0.210, 0.200, 0.187],
                            index=countries, columns=["vol"])
covariance = volatilities.dot(volatilities.T) * correlation
```

The market equilibrium weights are calculated using the market-cap of the indices.

```
# Table 2 of the He-Litterman paper: Market-capitalised weights
market_weights = pd.DataFrame([0.016, 0.022, 0.052, 0.055, 0.116, 0.124, 0.615],
                              index=countries, columns=["CapWeight"])
```



Under no market views, the investor holds the market portfolio shown above. Now let us start introducing some views.

View-1: Germany vs Rest of Europe

The first view explored by the authors states that German equities will outperform the rest of the European equities by 5%. We have one German equity in our portfolio – DE – and two European equities – FR and UK. We create the views vector and the pick list:

```
# Q
views = [0.05]


# P
pick_list = [
        {
            "DE": 1.0,
            "FR": -market_weights.loc["FR"]/(market_weights.loc["FR"] + \
                                             market_weights.loc["UK"]),
            "UK": -market_weights.loc["UK"] / (market_weights.loc["FR"] + \
                                               market_weights.loc["UK"])
        }
]
```

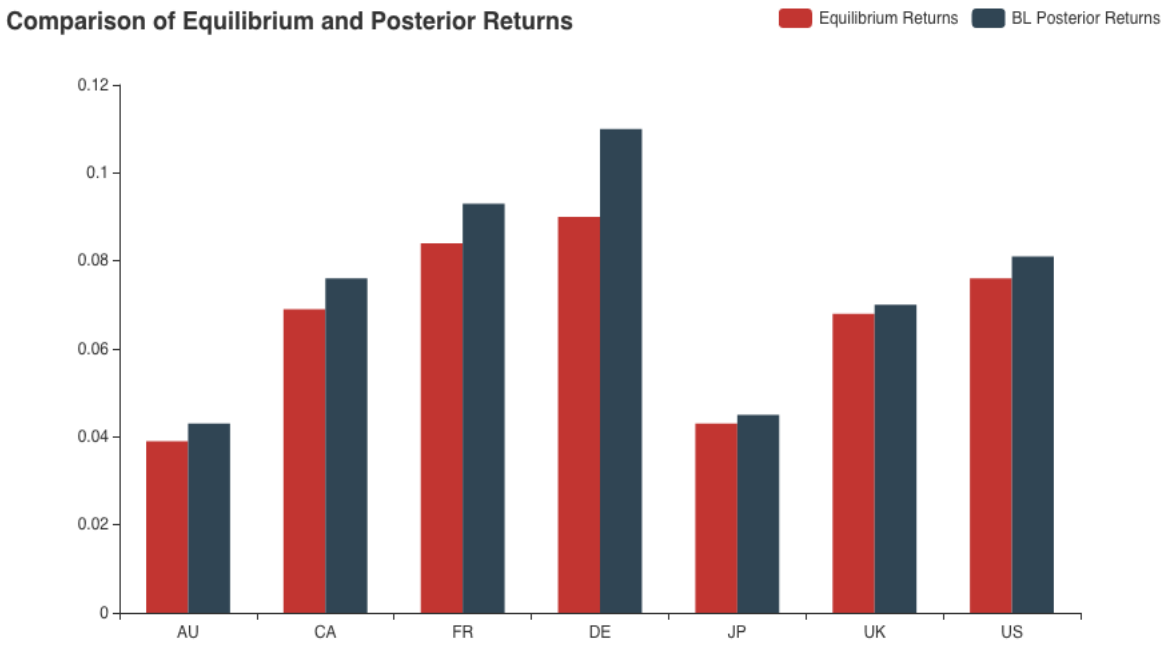
The pick list parameter in **VanillaBlackLitterman** class expects list of dictionaries specifying the assets participating in each view and the corresponding pick values. Internally, the code will convert this representation to the pick matrix P mentioned previously. Here we are using the capitalisation of the assets to calculate the respective pick values.

Finally it is time to allocate weights using the allocate() method.

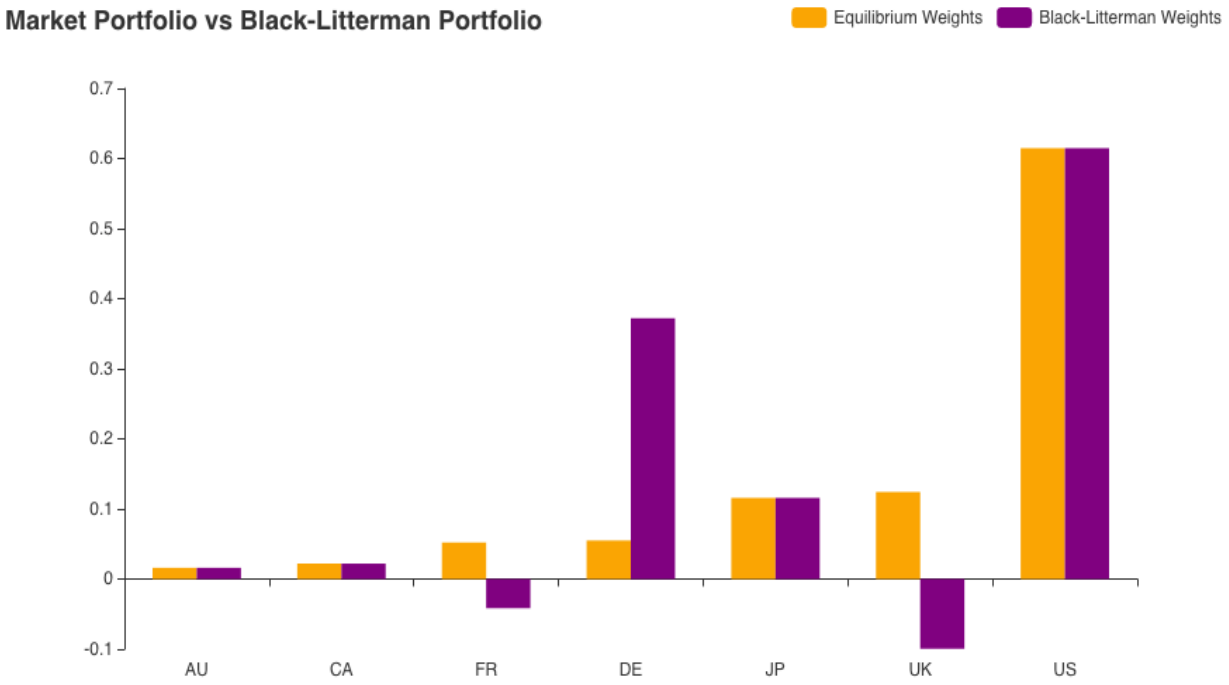
```
bl = VanillaBlackLitterman()

bl.allocate(covariance=covariance, market_capitalised_weights=market_weights,

            investor_views=views, pick_list=pick_list,

            asset_names=covariance.columns, tau=0.05,

            risk_aversion=2.5)
```

You can access the final results using four main class variables – **implied_equilibrium_weights, articleerior_expected_returns, articleerior_covariance and weights**.

Let's look at a comparison between the equilibrium and BL articleerior returns for View-1



One can notice that based on the positive view specified, the model has increased the returns of German equity relative to the others. However, note that since the market portfolio already had slightly higher returns for DE as compared to FR and UK, the relative increase due to the view is not so significant. The weight distribution, however, tells a different story.



Literally, all assets not involved in the view have their weights unchanged. On the other hand, the BL model increased the allocations for DE and made the weights of European equities negative! It is straight up telling us to short these assets. You can observe how a small view of 5% had such a visible effect on our portfolio. Now let us complicate this further by adding some more views.

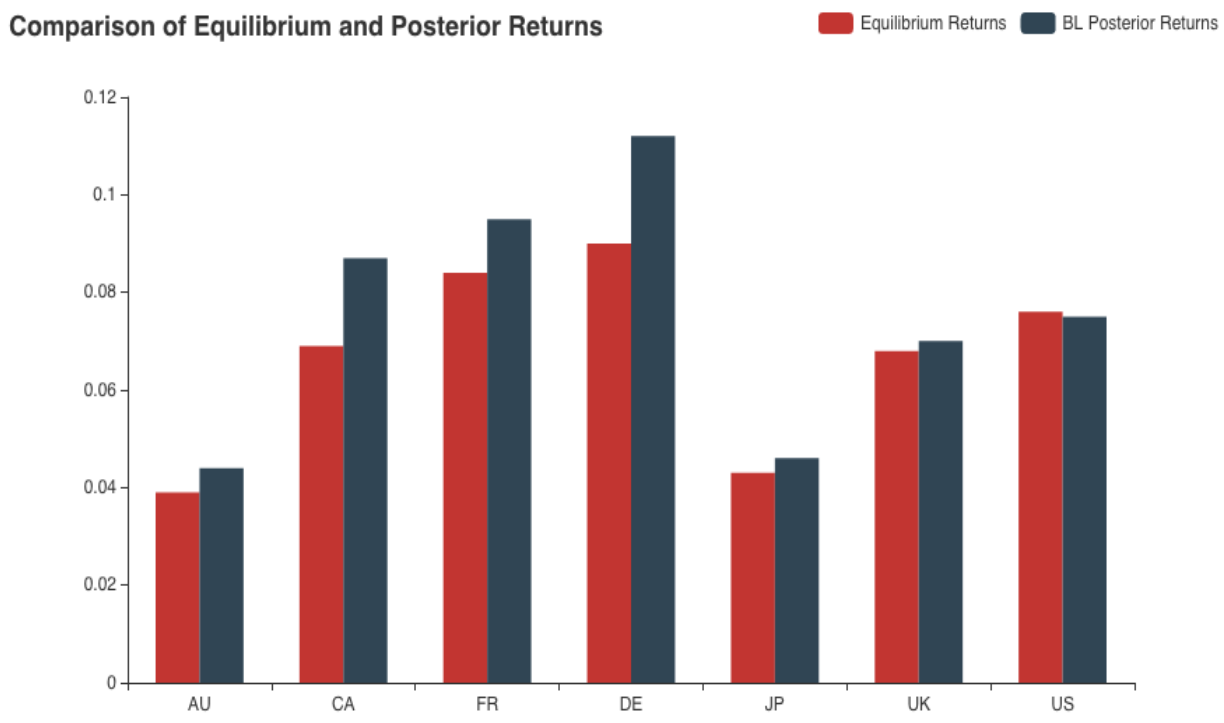
View-2: Canada vs US

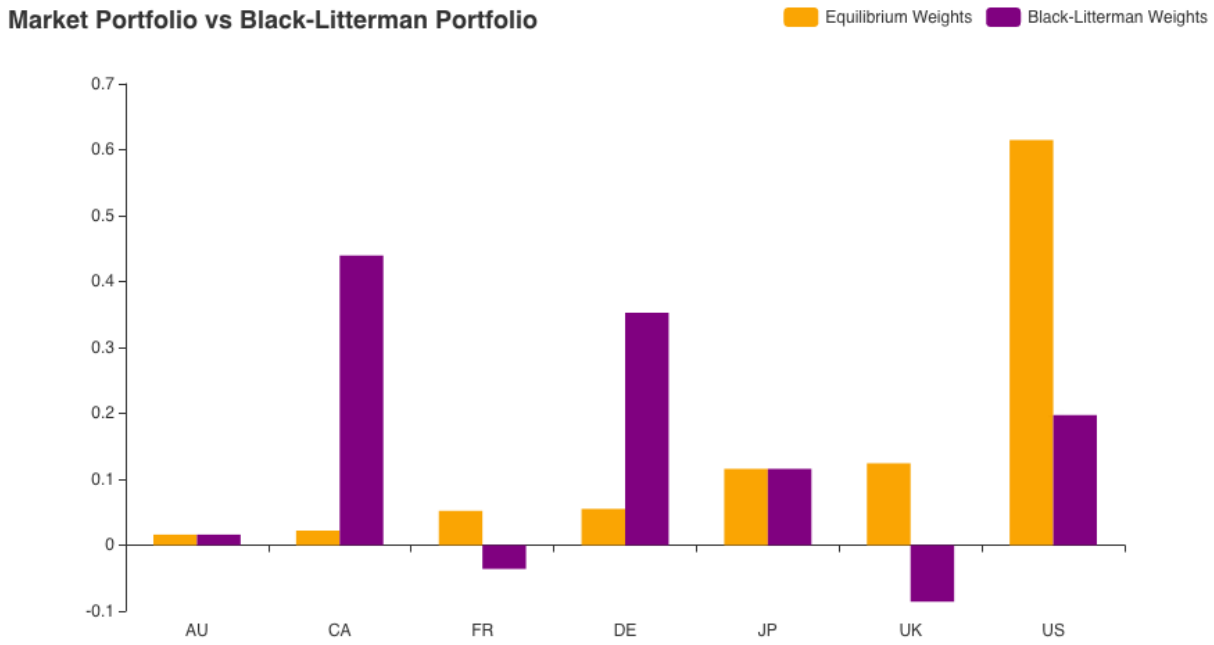
For this view, the investor thinks that Canadian equities will outperform US equity by 3% annually.

```
# Q
views = [0.05 , 0.03]


# P
pick_list = [
    {
        "DE": 1.0,
        "FR": -market_weights.loc["FR"]/(market_weights.loc["FR"] + \
                                         market_weights.loc["UK"]),
            "UK": -market_weights.loc["UK"] / (market_weights.loc["FR"] + \
                                               market_weights.loc["UK"])
    },
    {
        "CA": 1,
        "US": -1
    }
]
```

```
# Allocate
bl = VanillaBlackLitterman()
bl.allocate(covariance=covariance,
            market_capitalised_weights=market_weights, investor_views=views,
            pick_list=pick_list, asset_names=covariance.columns,
            tau=0.05, risk_aversion=2.5)
```

Note that we are also keeping the first view and adding a layer of complexity by introducing View-2. Let us see how our final portfolio looks like.





The result above is intuitively very easy to understand. The annualized returns on Canadian equities have increased further while those for the US have gone down a bit. Returns for DE, FR, and UK stay in conjunction with the first view. However, the most visible change can be seen in the weight distribution of the portfolio. Canadian equities see a 95% increase in the allocation while that of the US gets reduced by almost 67%. The German and European equities' allocations remains unaffected.

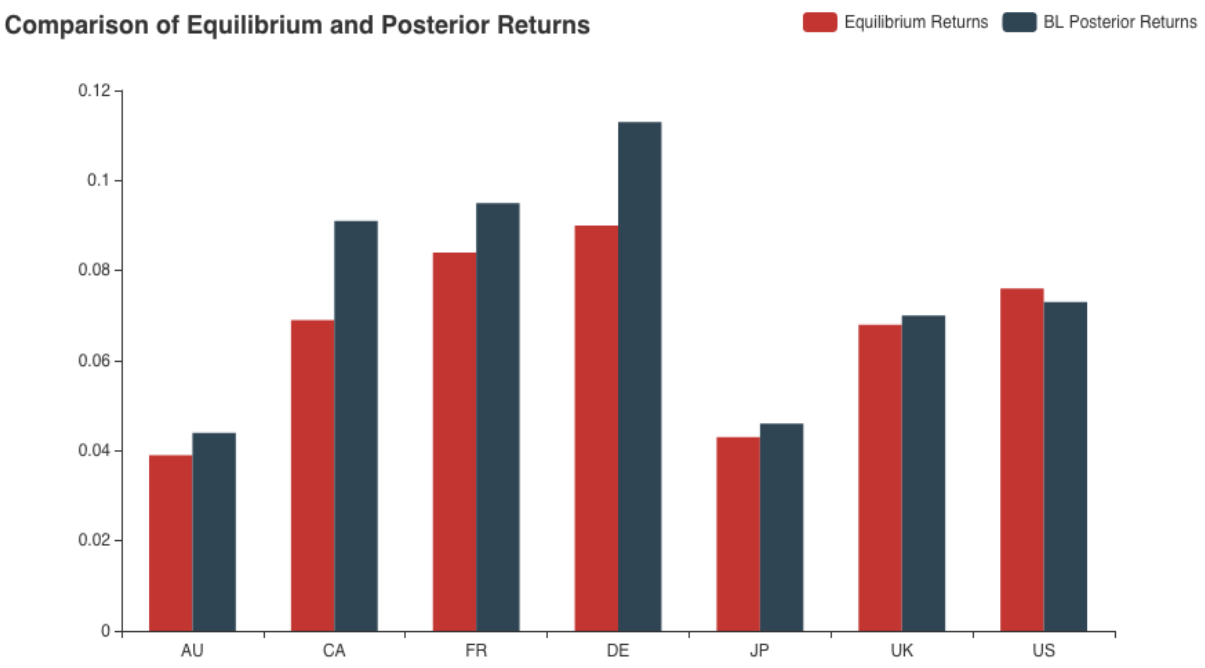
View-3: More Bullish Canada vs the US

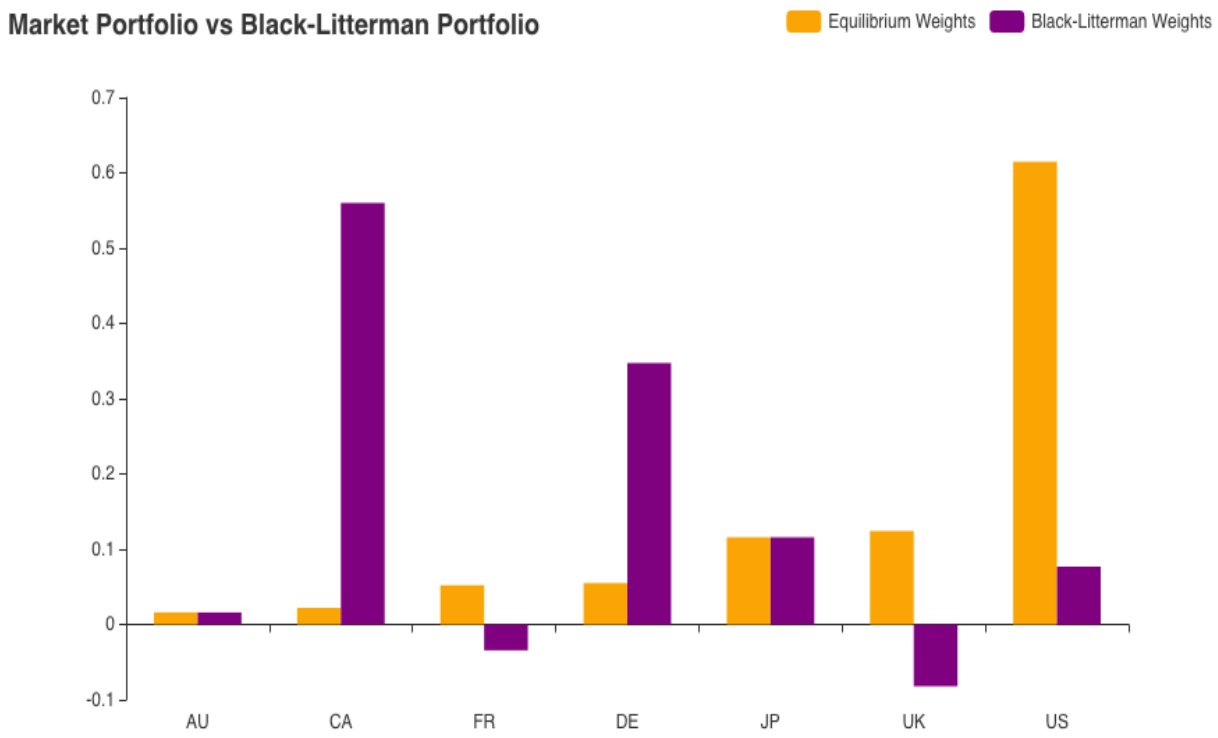
As the last example, the authors decide to make View-2 more bullish. Instead of a 3% return, the investor now expects CA to outperform the US by 4%.

```
# Q
views = [0.05 , 0.04]


# P
pick_list = [
    {
        "DE": 1.0,
        "FR": -market_weights.loc["FR"]/(market_weights.loc["FR"] + \
                                         market_weights.loc["UK"]),
            "UK": -market_weights.loc["UK"] / (market_weights.loc["FR"] + \
                                               market_weights.loc["UK"])
    },
    {
        "CA": 1,
        "US": -1
    }
]
```

```
# Allocate
bl = VanillaBlackLitterman()
bl.allocate(covariance=covariance,
            market_capitalised_weights=market_weights, investor_views=views,
            pick_list=pick_list, asset_names=covariance.columns,
            tau=0.05, risk_aversion=2.5)
```

Note that the P does not change here since we are only changing the view value.





The returns remain almost unchanged w.r.t View-2 return distribution. However, one can observe that a more bullish view on CA has led the BL model to increase its overall portfolio allocation to 56%! And correspondingly, the US allocations have been reduced to a meager 7% of the total holdings.

CONCLUSIONS

This article explores the intuition, mathematical formulations, and the practical uses of the Black-Litterman model, which is regarded by many as an important portfolio allocation algorithm developed in its time. Despite all the complexity, it is a simple application of the Bayes Rule and by formulating the model in the form of prior, likelihood, and articleerior distributions, the working and intuition behind the steps becomes much easier to understand.

In our opinion, one of the main reasons BL works so nicely is that it helps investors and analysts incorporate their own beliefs and views of the market – something which is ever-changing with the market conditions. This creates a model that is robust and less sensitive to the changing market dynamics and at the same time also handles variance in these views efficiently.

However, not everything is foolproof and the Black-Litterman model does suffer from some limitations:

- **Assumption of normality** – At the end of the day, even Black-Litterman could not escape the simplicity of normal assumption of the market returns. Assuming a normal distribution for the returns makes the model constrained and not entirely flexible to adapt to the true market parameters.

- **Views constrained to returns** – As mentioned previously, the ability to allow investors to specifytheir own views is one of the most noteworthy characteristics of the BL model. However, it is only really constrained to views on expected returns. The model can really be improved by allowing views on diverse market parameters like correlations, covariances, and variance between the returns, etc…

- **Limitations of the market portfolio** – The equilibrium portfolio plays an important role in the BL model. Hence, any changes in the market portfolio will affect the final articleerior BL weights. The problem is that the practicality the market portfolio is very hard to define. There are many risky assets trading in the world but they do not represent the entirety of the market securities. As a result, although in theory one can hold an equilibrium portfolio, in the practical world, it is very easy to get less than optimal BL allocations because of an improper market portfolio.

- **Sensitivity to investor inputs** – Although all portfolio Optimization models suffer from sensitivity to input data, BL is affected more so because of the added parameter of investor beliefs. As you saw in the bullish view of Canada vs the US, increasing the belief by just 1% led to such a dramatic change in the final portfolio allocations. Thus, one needs to be extra careful when trying to input their views as they can sway the final allocations in either direction very easily.

REFERENCES

1. Black, F. and Litterman, R., 1990. Asset allocation: combining investor views with market equilibrium., Goldman Sachs Fixed Income Research, 115.
2. Tetyana Polovenko, 2017. Black-Litterman Model
3. Idzorek, T., 2007. A step-by-step guide to the Black-Litterman model: Incorporating user-specified confidence levels. In Forecasting expected returns in the financial markets (pp. 17-38). Academic Press.
4. Meucci, A., 2006. Beyond Black-Litterman in practice: A five-step recipe to input views on non-normal markets. Available at SSRN 872577.
5. Walters, C.F.A., 2014. The Black-Litterman model in detail. Available at SSRN 1314585.
6. Narrow Margin, 2012. The Parameter Tau in Idzorek's Version of the Black-Litterman Model
7. He, G. and Litterman, R., 2002. The intuition behind Black-Litterman model portfolios. Available at SSRN 334304.

3.0

IMPLEMENTING THEORY-IMPLIED CORRELATION MATRIX WITH PORTFOLIOLAB

INTRODUCTION

Traditionally, correlation matrices have always played a large role in finance. They have been used in tasks ranging from portfolio management to risk management and are calculated based on historical empirical observations. Although they are used so frequently, these correlation matrices often have poor predictive power and prove to be unreliable estimators. Additionally, there are also factor-based correlation matrices, which also do not perform well due to their non-hierarchical structure.

In a 2019 paper written by Marcos López de Prado, he introduced the Theory-Implied Correlation (TIC) algorithm. It demonstrated a new machine learning approach to estimate correlation matrices based on economic theory, rather than historical observations or being factor-based. The TIC algorithm estimates a forward-looking correlation matrix implied by a proposed hierarchical structure of the assets and is computed in three main steps:

1. **Fitting our tree graph structure based on the empirical correlation matrix**
2. **Deriving our correlation matrix from the linkage object**
3. **De-noising the correlation matrix**

Today, we will be exploring the TIC algorithm and look at how to leverage PortfolioLab's implementation of the same.

HOW THE TIC ALGORITHM WORKS?

In this section, we will be going through a quick summary of the steps involved in computing the TIC matrix.

Fitting our Tree Graph Structure

In this algorithm, the tree graph represents the economic theory and hierarchical structure of our assets. This tree structure is fit based upon our empirical correlation matrix. Essentially, this structure will tell us which assets are closely related to each other and the relative distance between them. This results in a binary tree using an agglomerative clustering technique which can be visualized through a dendrogram. However, this is not the final tree graph we are looking for. The dendrogram always has two items per cluster while our actual tree graph could have one or more leaves per branch. Additionally, the dendrogam will always have $N - 1$ clusters while the tree graph may have an unlimited number of levels and our dendrogram incorporates the notion of distance but the tree graph does not.

The general steps for this part of the algorithm are as follows:

1. If there is no top level of the tree (tree root), this level is added so that all variables are included in one general cluster.

2. The empirical correlation matrix is transformed into a matrix of distances using the above formula:

$$d(i,j) = \sqrt{0.5 * (1 - \rho(i,j))}$$

3. For each level of the tree, the elements are grouped by elements from the higher level. The algorithm iterates from the lowest to the highest level of the tree.

4. A linkage object is created for these grouped elements based on their distance matrix. Each link in the linkage object is an array representing a cluster of two elements and has the following data as elements.

    - ID of the first element in a cluster
    - ID of the second element in a cluster
    - Distance between the elements
    - Number of atoms (simple elements from the portfolio and not clusters) inside

5. A linkage object is transformed to reflect the previously created clusters.

6. A transformed local linkage object is added to the global linkage object.

7. Distance matrix is adjusted to the newly created clusters – elements that are now in the new clusters are replaced by the clusters in the distance matrix. The distance from the new clusters to the rest of the elements in the distance matrix is calculated as a weighted average of distances of two elements in a cluster to the other elements. The weight is the number of atoms in an element. So, the formula is:

$$DistanceCluster = \frac{Distance_1 * NumAtoms_1 + Distance_2 * NumAtoms_2}{NumAtoms_1 + NumAtoms_2}$$

The linkage object, representing a dendrogram of all elements in a portfolio is the result of the first step of the algorithm. It sequentially clusters two elements together, while measuring how closely together the two elements are, until all elements are subsumed within the same cluster.

The final object looks something like this:

```
array([[18.        , 22.        ,  0.0811119 ,  2.        ],
       [ 1.        ,  6.        ,  0.08149536,  2.        ],
       [16.        , 24.        ,  0.09326276,  3.        ],
       [ 7.        , 25.        ,  0.1363472 ,  4.        ],
       [ 4.        , 26.        ,  0.17810389,  5.        ],
       [ 0.        , 14.        ,  0.1792156 ,  2.        ],
       [ 5.        , 19.        ,  0.21662725,  2.        ],
       [17.        , 28.        ,  0.23330552,  3.        ],
       [12.        , 23.        ,  0.24874012,  3.        ],
       [ 8.        ,  9.        ,  0.25724575,  2.        ],
       [27.        , 30.        ,  0.2655236 ,  8.        ],
       [31.        , 33.        ,  0.28691389, 11.        ],
       [32.        , 34.        ,  0.34482744, 13.        ],
       [ 3.        , 35.        ,  0.35005852, 14.        ],
       [15.        , 36.        ,  0.37701548, 15.        ],
       [10.        , 37.        ,  0.48135409, 16.        ],
       [ 2.        , 20.        ,  0.56979437,  2.        ],
       [13.        , 38.        ,  0.7405772 , 17.        ],
       [11.        , 21.        ,  0.99148687,  2.        ],
       [29.        , 39.        ,  1.00362904,  4.        ],
       [41.        , 42.        ,  1.34475143,  6.        ],
       [40.        , 43.        ,  2.64162061, 23.        ]])
```

Deriving the Correlation Matrix

We can now derive a correlation matrix from our linkage object returned by the first step.

1. One by one, the clusters (each represented by a link in the linkage object) are decomposed to lists of atoms contained in each of the two elements of the cluster.

2. The elements on the main diagonal of the resulting correlation matrix are set to 1s.
   The off-diagonal correlations between the variables are computed as

$$\rho(i,j) = 1 - 2 * d^2(i,j)$$

De-Noising the Correlation Matrix

We can now derive a correlation matrix from our linkage object returned by the first step.

1. The eigenvalues and eigenvectors of the correlation matrix are calculated.

2. Marcenko-Pastur distribution is fit to the eigenvalues of the correlation matrix and the maximum theoretical eigenvalue is calculated.

3. This maximum theoretical eigenvalue is set as a threshold and all the eigenvalues above the threshold are shrinked.

4. The de-noised correlation matrix is calculated back from the eigenvectors and the new eigenvalues.

USING PORTFOLIOLAB'S TIC IMPLEMENTATION

In this section, we will go through a working example of using the TIC implementation provided by PortfolioLab and test it on a portfolio of assets.

```
import portfoliolab as pl

import numpy as np

import pandas as pd

import seaborn as sns

import myplotlib.pyplot as plt
```

Specifying the Economic Structure a.k.a Tree Graph

The TIC algorithm combines both an economic theory and empirical data. The economic theory is represented by a tree graph and our empirical data is represented by an empirical correlation matrix.

As mentioned previously, the tree graph embeds important information about the hierarchy of the assets in our portfolio and this comes from external market knowledge. Let us look at how to create the graph file.

An example of a hierarchical structure for the financial instruments is the MSCI's Global Industry Classification Standard (GICS) that classifies investment universes in terms of four nested levels – sub-industry, industry, industry group, and sector. Investors can also add more diverse levels in the tree such as business ties, supply chain, competitors, industrial sectors, shared ownership, etc… For this article, we are using a dataset of ETFs and have created an example dataframe representing the tree graph:

```
# Getting the tree graph for ETFs

etf_tree = pd.read_csv("classification_tree.csv")

# Having a loot at the tree structure of ETFs

print(etf_tree)
```

The proposed tree structure has four levels:

1. Ticker of the ETF
2. Sector of the ETF
    - XXXX10 – General/No sector
    - XXXX20 – Materials
    - XXXX30 – Energy
    - XXXX40 – Financial
    - XXXX50 – Technology
    - XXXX60 – Utilities
3. Region of the ETF
    - XX10 – Emerging markets
    - XX20 – Europe
    - XX30 – US and Canada
    - XX40 – Asia
4. Type of the ETF
    - 10 – Equity ETF
    - 20 – Bond ETF

	TICKER	SECTOR	REGION	TYPE
0	EEM	101010	1010	10
1	EWG	102010	1020	10
2	TIP	203010	2030	20
3	EWJ	104010	1040	10
4	EFA	103010	1030	10
5	IEF	203010	2030	20
6	EWQ	102010	1020	10
7	EWU	102010	1020	10
8	XLB	103020	1030	10
9	XLE	103030	1030	10
10	XLF	103040	1030	10
11	LQD	203020	2030	20
12	XLK	103050	1030	10
13	XLU	103060	1030	10
14	EPP	104010	1040	10
15	FXI	104010	1040	10
16	VGK	102010	1020	10
17	VPL	104010	1040	10
18	SPY	103010	1030	10
19	TLT	203010	2030	20
20	BND	203010	2030	20
21	CSJ	203010	2030	20
22	DIA	103010	1030	10

The columns are ordered bottom-up, with the leftmost column corresponding to the terminal leaves, and the rightmost column corresponding to the tree's root. This means that if you converted this dataframe into a tree, then the topmost level would be the type of the asset (in this case, ETF) with the subsequent nested levels being region, sector and finally the asset itself.

Calculating the Empirical Correlation Matrix

Now we calculate the series of returns from the raw prices and finally get the empirical correlations.

```
# Getting the price data for ETFs
etf_tree = pd.read_csv('stock_prices.csv', parse_dates=True, index_col='Date')
# Class with returns calculation function
ret_est = pl.estimators.ReturnsEstimators()
# Calculating returns
etf_returns = ret_est.calculate_returns(etf_prices)
# Now the emperical correlation matrix can be calculated
etf_corr = etf_returns.corr())
```

Calculating the TIC Matrix

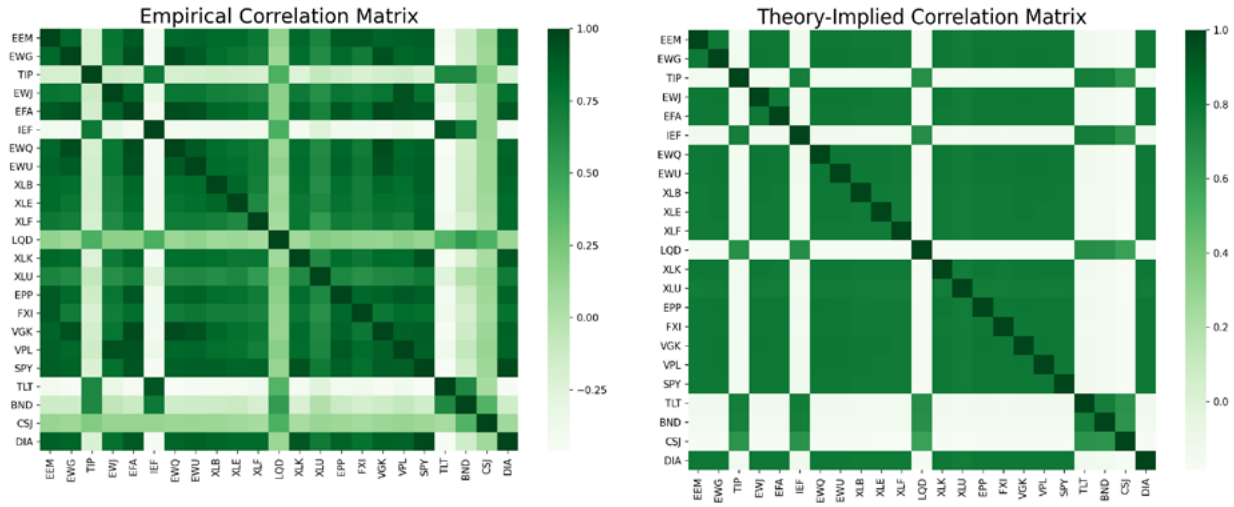
Equipped with the tree graph and the empirical correlation matrix, we merge them using the TIC algorithm to generate the new correlation matrix.

```
# Calculating the relation of sample length T to the number of variables N
# It's used for de-noising the TIC matrix
tn_relation = etf_returns.shape[0] / etf_returns.shape[1]


# Class with TIC function
tic = pl.estimators.TheoryImpliedCorrelation()


# Calculating theory-implied correlation matrix
etf_tic = tic.tic_correlation(etf_tree, etf_corr, tn_relation)


# Setting the indexes of the theory-implied correlation matrix
etf_tic = pd.DataFrame(etf_tic, index=etf_corr.index, columns=etf_corr.index)
```

Visualising the TIC Matrix

We can visualize the difference between the Empirical correlation matrix and the Theory-implied correlation matrix using heatmaps

```
# Plotting the heatmap of the matrix

sns.heatmap(etf_corr, cmap="Greens")

plt.title('Empirical Correlation Matrix')

plt.show()


sns.heatmap(etf_tic, cmap="Greens")

plt.title('Theory-Implied Correlation Matrix')

plt.show()
```



One can observe that the TIC matrix is less noisy and is more smoother.**The external market views are incorporated into the empirical values to adjust the noise and get better estimates of the underlying correlations.**

However, you might be wondering – if there is such a large visible difference between the two matrices, can we trust the new correlations? Does the TIC algorithm remove much more than just noise and lead to loss of the true correlations between the assets?.

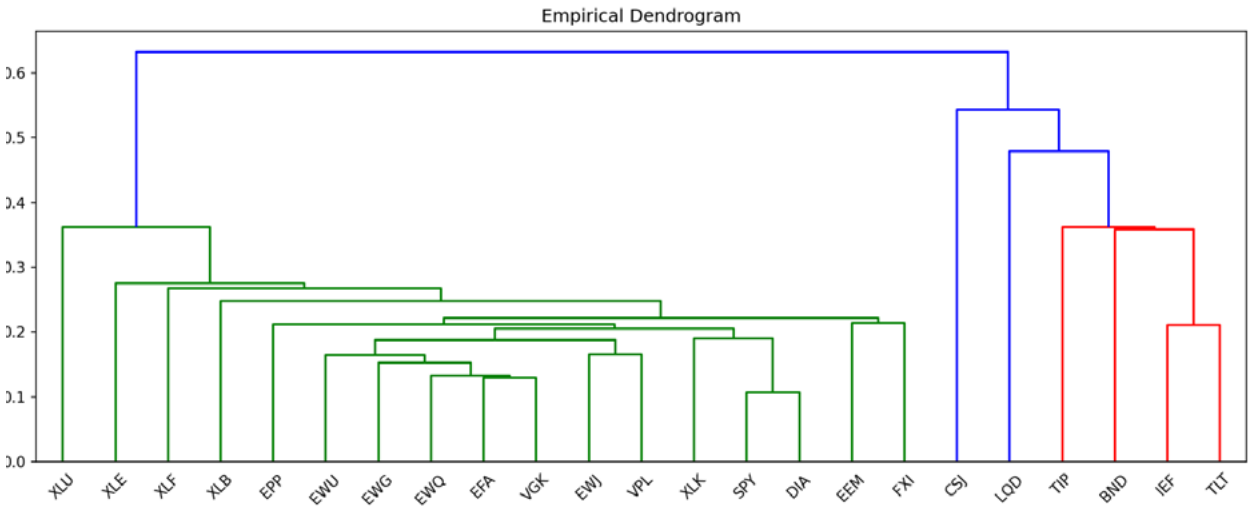
One can test the similarity of the two matrices using the correlation matrix distance introduced by Herdin and Bonek in a paper **AMIMO Correlation Matrix based Metric for Characterizing Non-Stationarity** available here.

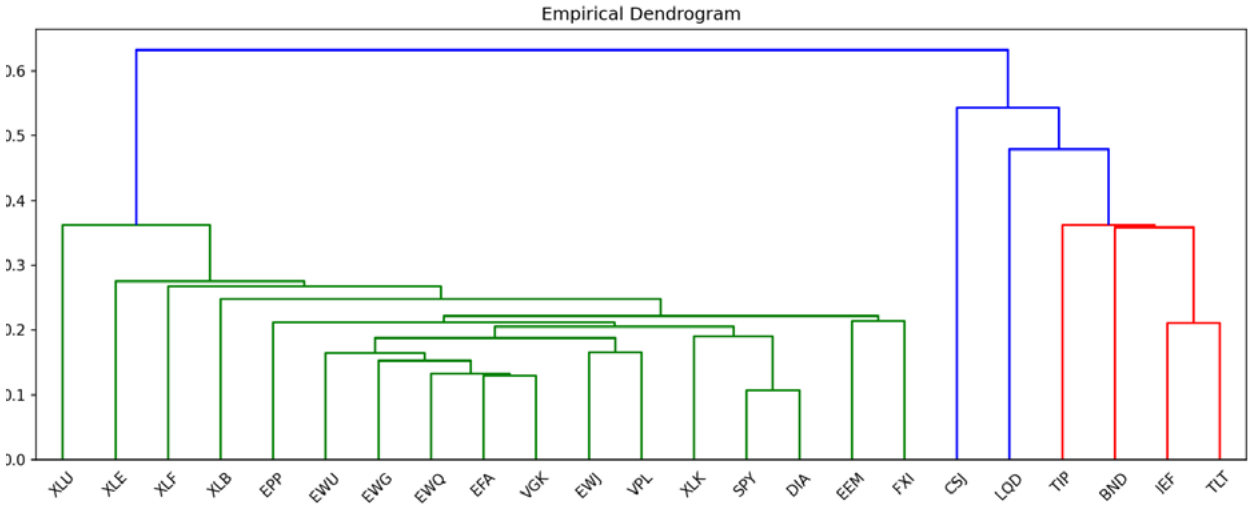
```
# Calculating the correlation matrix distance

distance = tic.corr_dist(etf_corr, etf_tic)

# Printing the result

print('The distance between empirical and the theory-implied correlation matrices is',

      distance)
```

```
The distance between empirical and the theory-implied correlation matrices is

0.035661302090136404
```

The correlation matrices are different but are not too far apart. **This shows that the theory-implied correlation matrix blended theory-implied views with empirical ones.**

You can see that the dendrograms generated from the respective correlation matrices are also visibly different from each other. The TIC dendrogram structure appears to be less complex and a simple hierarchy as compared to the empirical one.





CONCLUSION

Throughout this tutorial article, we learned about the intuition behind the Theory-Implied Correlation algorithm and how to use PortfolioLab's implementation of the same. By combining external market views with empirical observations, the TIC algorithm removes the underlying noise from traditional correlation measurements and generates better estimates of the portfolio correlations. Here is an excerpt taken directly from the original paper:

For over half a century, most asset managers have used historical correlation matrices (empirical or factor-based) to develop investment strategies and build diversified portfolios. When financial variables undergo structural breaks, historical correlation matrices do not reflect the correct state of the financial system, hence leading to wrong investment decisions. TIC matrices open the door to the development of forward-looking investment strategies and portfolios.

The following links provide a more detailed exploration of the algorithm and other sources for further reading.

REFERENCES

1. **Prado, Marcos López De. "Estimation of Theory-Implied Correlation Matrices." SSRN Electronic Journal, 2019, doi:10.2139/ssrn.3484152**
2. Herdin, M., et al. "Correlation Matrix Distance, a Meaningful Measure for Evaluation of Non-Stationary MIMO Channels." 2005 IEEE 61st Vehicular Technology Conference, doi:10.1109/vetecs.2005.1543265.

4.0

INTRODUCING ONLINE PORTFOLIO SELECTION

INTRODUCTION

Online Portfolio Selection is an algorithmic trading strategy that sequentially allocates capital among a group of assets to maximize the final returns of the investment.

Traditional theories for portfolio selection, such as Markowitz's Modern Portfolio Theory, optimize the balance between the portfolio's risks and returns. However, OLPS is founded on the capital growth theory, which solely focuses on maximizing the returns of the current portfolio.

Through these walkthroughs of different portfolio selection strategies, we hope to introduce a set of different selection tools available for everyone. Most of the works will be based on Dr. Bin Li and Dr. Steven Hoi's book, Online Portfolio Selection: Principles and Algorithms, and further recent papers will be implemented to assist the development and understanding of these unique portfolio selection strategies.

Problem Formulation

- **Price:** p_t
- **Price Relative:** $x_t = \frac{p_t}{p_{t-1}}$
    - Ratio of the current time's price to the last time's price.
    - Asset i's price relative at time t is $x_{t,i}$.
- **Portfolio Weight:** b_t
    - Represent the allocation of capital to a particular asset.
    - Assume that the weights are non-negative and that the sum of the weights is one, which will simulate a long-only, no leverage environment.
- **Portfolio Return:** $S_t = S_0 \prod_{i=0}^{t} x_i \cdot b_i$
    - Cumulative product of all previous returns.
    - S_0 represents the initial capital, and each product of price relative and portfolio weights represent the increase in capital for a particular time period.

Data

We will use the ETF data included in the PortfolioLab library for analysis. This includes 23 ETF's with closing prices from 2008 to 2016.

Date	EEM	EWG	TIP	EWJ	EFA	IEF	EWQ	EWU	XLB
2008-01-02	49.273335	35.389999	106.639999	52.919998	78.220001	87.629997	37.939999	47.759998	41.299999
2008-01-03	49.716667	35.290001	107.000000	53.119999	78.349998	87.809998	37.919998	48.060001	42.049999
2008-01-04	48.223331	34.599998	106.970001	51.759998	76.570000	88.040001	36.990002	46.919998	40.779999
2008-01-07	48.576668	34.630001	106.949997	51.439999	76.650002	88.199997	37.259998	47.060001	40.220001
2008-01-08	48.200001	34.389999	107.029999	51.320000	76.220001	88.389999	36.970001	46.400002	39.599998

Buy and Hold

Buy and Hold is a strategy where an investor invests in an initial portfolio and never rebalances it. The portfolio weights, however, change as time goes by because the underlying assets change in prices.

Returns for Buy and Hold can be calculated by multiplying the initial weight and the cumulative product of relative returns.

$$S_n(BAH(b_1)) = b_1 \cdot \left(\bigodot_{t=1}^{n} x_t \right)$$

Because we didn't specify a weight when we allocated to the strategy, initial weights are uniformly distributed across all the ETFs for time 0

Date	EEM	EWG	TIP	EWJ	EFA	IEF	EWQ	EWU	XLB	XLE	...
2016-06-27	0.029938	0.029988	0.049053	0.038726	0.030994	0.057853	0.026079	0.027566	0.050344	0.037951	...
2016-06-28	0.029835	0.029540	0.049666	0.038889	0.030689	0.058910	0.025715	0.026721	0.049112	0.037086	...
2016-06-29	0.030271	0.029711	0.049119	0.038906	0.031040	0.058051	0.026115	0.027466	0.048818	0.037569	...
2016-06-30	0.030720	0.029810	0.048588	0.039013	0.031286	0.057224	0.026305	0.027893	0.049138	0.037890	...
2016-07-01	0.030727	0.029944	0.048208	0.038302	0.031445	0.056796	0.026561	0.028361	0.049441	0.037823	...

The weights change over time because the ETF prices fluctuate.

With the crash in 2009, Buy and Hold strategies have returned almost the same returns from the initial allocation. The results would vary depending on the weights of stock. Generally, this strategy is a great starting point as a benchmark comparison, because we can compare the historical performances of active and passive investing.

Best Stock

Best Stock strategy chooses the best performing asset in hindsight.

The best performing asset is determined with an argmax equation stated below. The portfolio selection strategy searches for the asset that increases the most in the price for the given time period.

$$b_0 = \underset{b \in \Delta_m}{\arg\max}\, b \cdot \left(\bigodot_{t=1}^{n} x_t \right)$$

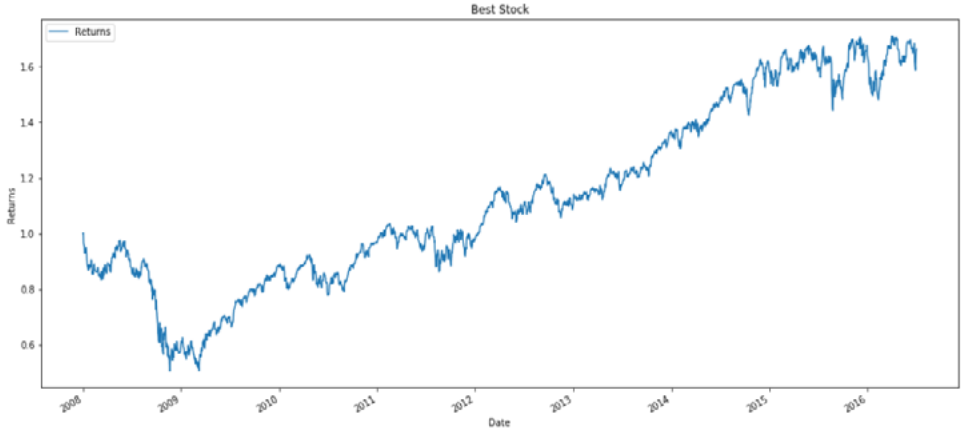
Once the initial portfolio has been determined, the final weights can be represented as buying and holding the initial weight.

$$S_n(BEST) = \max_{b \in \Delta_m} b \cdot \left(\bigodot_{t=1}^{n} x_t \right) = S_n(BAH(b_0))$$

Date	EWQ	EWU	XLB	XLE	XLF	LQD	XLK	XLU	EPP	FXI	VGK	VPL	SPY	TLT	BND	CSJ	DIA
2008-01-02	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2008-01-03	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2008-01-04	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2008-01-07	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2008-01-08	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2016-06-27	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2016-06-28	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2016-06-29	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2016-06-30	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

If we examine all of the weights assigned to best stock, we notice that all the weights are set to 0 except for XLK. For the given period and price data, XLK was the best performing asset, so the portfolio strategy chooses to allocate all of its own weight on XLK.

Welcome to Part 1 of the series of articles on optimal stopping problems for statistical arbitrage. And today we are going to talk about the Ornstein-Uhlenbeck model application to optimal stopping problems in pairs trading.



As seen with the above graph, we are directly tracking XLK, the best performing ETF during this period. The same exact graph can be replicated by buying only XLK from the beginning using the Buy and Hold strategy since we are not rebalancing the portfolio.

Constant Rebalanced Portfolio

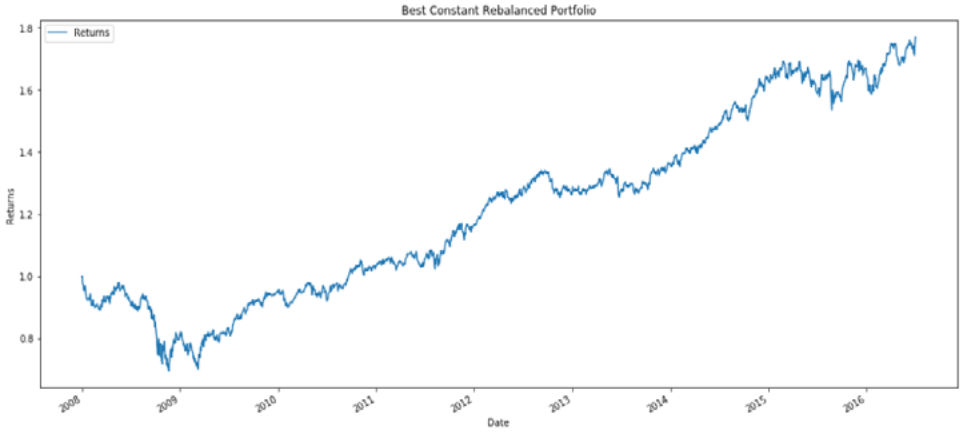
Constant Rebalanced Portfolio rebalances to a certain portfolio weight every time period. This particular weight can be set by the user, and if there are no inputs, it will automatically allocate equal weights to all assets. The total returns for a CRP can be calculated by taking the cumulative product of the weight and relative returns matrix.

$$S_n(CRP(b)) = \prod_{t=1}^{n} b^\top x_t$$

Date	IEF	EWQ	EWU	XLB	XLE	XLF	LQD	XLK	XLU	EPP	FXI	VGK	VPL	SPY	TLT	BND	CSJ
2008-01-02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2008-01-03	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2008-01-04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2008-01-07	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2008-01-08	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2016-06-27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2016-06-28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2016-06-29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2016-06-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0
2016-07-01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.625272	0.0	0.0	0.0	0.0	0.0	0.0	0.374728	0.0	0.0

The weights are the same from the beginning to the end. BCRP returns weights of 0.625272 to XLK, which was the best performing ETF, and 0.374728 to TLT, the second-best performing ETF.

Interestingly, TLT tracks US Treasury bonds of 20+ years and XLK tracks the S&P 500 technology sector. The portfolio weights returned by BCRP is the inverse of the popularized 60/40 portfolio of equity to bonds. For the assets of this dataset, there is a strong indication of a passive mean reversion with these two assets that allow the BCRP to produce returns that are higher than the Best Stock of XLK.

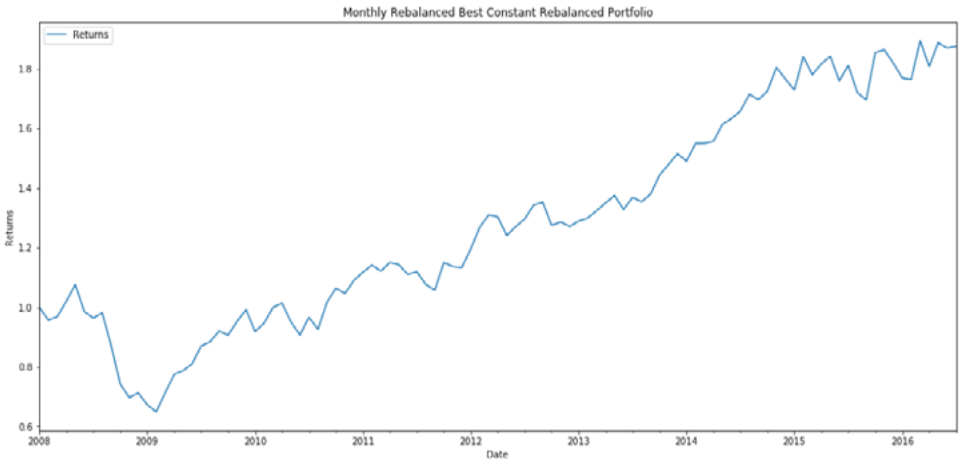


BCRP achieves higher returns than any other portfolio that we have seen so far. However, this is primarily due to the fact that we had the data of the complete market sequence. This is impossible to implement in the real market, but it is a good benchmark to keep in hand as we implement other strategies. For other portfolios and the given dataset, we should look to perform better than the 1.8 given with BCRP strategy.

On another note, our strategies can be resampled to a given time period if we want to simulate a portfolio manager's monthly rebalancing or if we want to minimize our transaction costs. We'll examine a case where we monthly rebalance our portfolio.

Date	IEF	EWQ	EWU	XLB	XLE	XLF	LQD	XLK	XLU	EPP	FXI	VGK	VPL	SPY	TLT	BND	CSJ	DIA
2008-01-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2008-02-29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2008-03-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2008-04-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2008-05-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2016-03-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2016-04-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2016-05-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2016-06-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0
2016-07-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89725	0.0	0.0	0.0	0.0	0.0	0.0	0.10275	0.0	0.0	0.0

The weights have changed to 0.89725 of XLK and 0.10275 TLT as we are now resampling on a monthly basis.

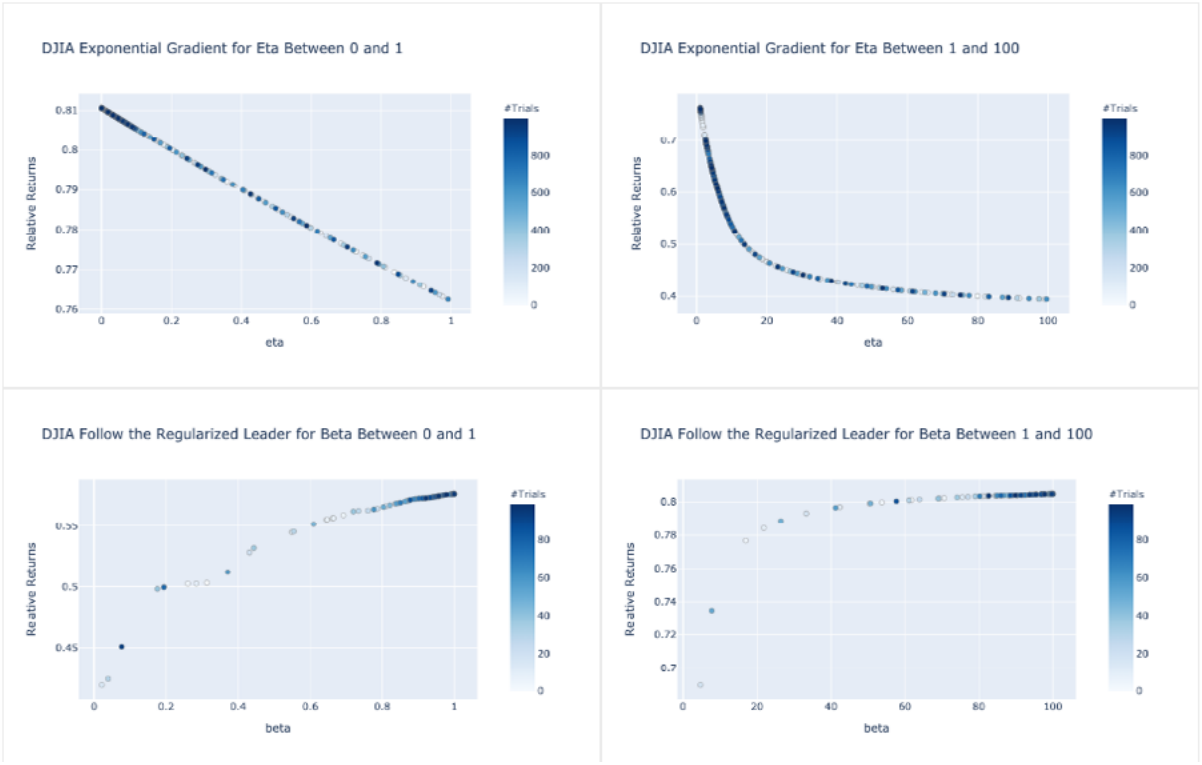


The graph is smoother compared to the daily rebalanced portfolio, and we actually achieve higher returns than the daily rebalanced one by a marginal amount. The monthly rebalancing most likely reduces market noise and follows the traditional mean reversion method to account for higher returns.

4.1

ONLINE PORTFOLIO SELECTION: MOMENTUM

INTRODUCTION



Today we will be exploring the second chapter of our portfolio selection module, momentum.

Momentum strategies have been a popular quantitative strategy in recent decades as the simple but powerful trend-following allows investors to exponentially increase their returns. This module will implement two types of momentum strategies with one following the best-performing assets in the last period and the other following the Best Constant Rebalanced Portfolio until the last period.

The focus of this article will be on introducing the applications of these popular strategies with the ease of using PortfolioLab's newest module. We hope that the newfound interests from these topics will spur more development as well as assist the research process for many others in the field.

EXPONENTIAL GRADIENT

Exponential Gradient strategies track the best performing stock with a learning rate, η, but also regularizes the new portfolio weight to prevent drastic changes from the previous portfolio.

$$b_{t+1} = \underset{b \in \Delta_m}{\arg\max}\, \eta \log b \cdot x_t - R(b, b_t)$$

As a review, b_t is the portfolio weights, and x_t is the price relatives for time t.

Exponential Gradients have an extremely efficient computational time that scales with the number of assets, and broadly speaking, there are three update methods to iteratively update the selection of portfolio weights.

Multiplicative Update

David Helmbold first proposed a regularization term that adopts relative entropy.

$$R(b, b_t) = \sum_{i=1}^{m} b_i \log \frac{b_i}{b_{t,i}}$$

Using log's first order taylor expansion of b_i

$$\log b \cdot x_t \approx \log(b_t \cdot x_t) + \frac{x_t}{b_t \cdot x_t}(b - b_t)$$

Multiplicative update algorithm can be stated as the following.

$$b_{t+1} = b_t \cdot \exp\left(\eta \frac{x_t}{b_t \cdot x_t}\right)$$

Gradient Projection

Instead of relative entropy, gradient projection adopts an L2-regularization term.

$$R(b, b_t) = \frac{1}{2} \sum_{i=1}^{m} (b_i - b_{t,i})^2$$

Gradient projection can then be iteratively updated with the following equation.

$$b_{t+1} = b_t + \eta \cdot \left(\frac{x_t}{b_t \cdot x_t} - \frac{1}{m} \sum_{j=1}^{m} \frac{x_t}{b_t \cdot x_t}\right)$$

Expectation Maximization

Lastly, expectation maximization uses χ^2 regularization.

$$R(b - b_t) = \frac{1}{2} \sum_{i=1}^{m} \frac{(b_i - b_{t,i})^2}{b_{t,i}}$$

Then the corresponding update rule becomes

$$b_{t+1} = b_t \cdot \left(\eta \cdot \left(\frac{x_t}{b_t \cdot x_t} - 1\right) + 1\right)$$

Follow the Leader

The biggest drawback of using Exponential Gradient is the failure to look at the changes before the latest period. Follow the Leader mediates this shortfall by directly tracking the Best Constant Rebalanced Portfolio; therefore, FTL looks at the whole history of the data and calculates the portfolio weights that would have had the maximum returns.

$$b_{t+1} = b_t^\star = \underset{b \in \Delta_m}{\arg\max} \sum_{\tau=1}^{t} \log(b \cdot x_\tau)$$

Follow the Regularized Leader

Follow the Regularized Leader adds an additional regularization term to prevent rapid changes each period.

β acts similarly to Exponential Gradient's η.

$$b_{t+1} = \underset{b \in \Delta_m}{\arg\max} \sum_{\tau=1}^{t} \log(b \cdot x_\tau) - \frac{\beta}{2} R(b)$$

DATA

We will be using 6 different datasets, and if you would like to learn more about each dataset, exploratory analysis is available here.

1. 36 NYSE Stocks from 1962 to 1984 by Cover

2. 30 DJIA Stocks from 2001 to 2003 by Borodin

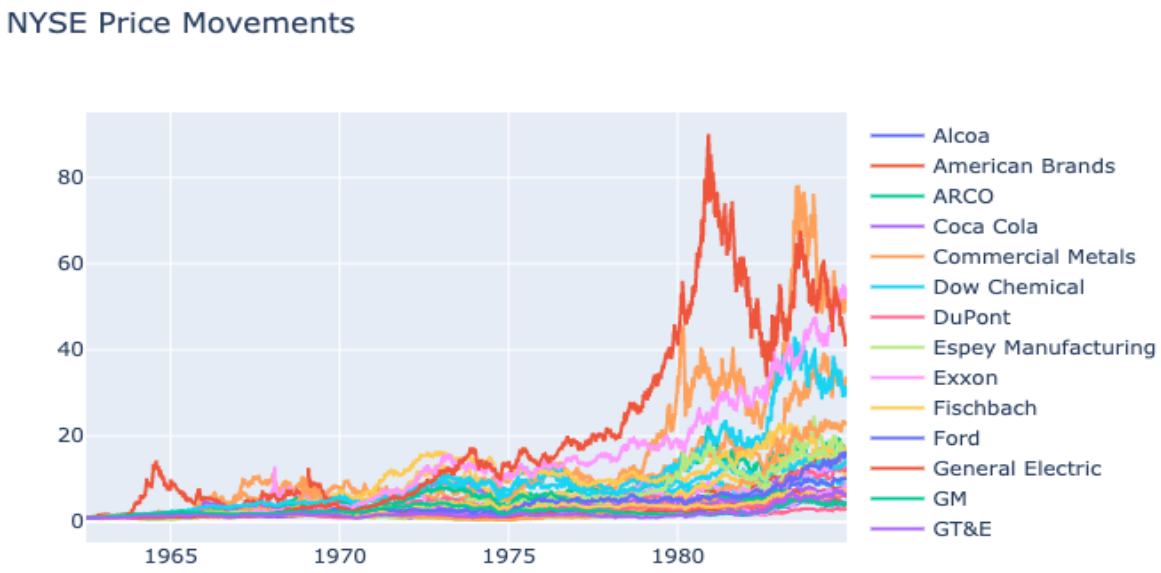
3. 88 TSE Stocks from 1994 to 1998 by Borodin

4. 25 Largest S&P500 Stocks from 1998 to 2003 by Borodin

5. 23 MSCI Developed Market Indices from 1993 to 2020 by Alex Kwon

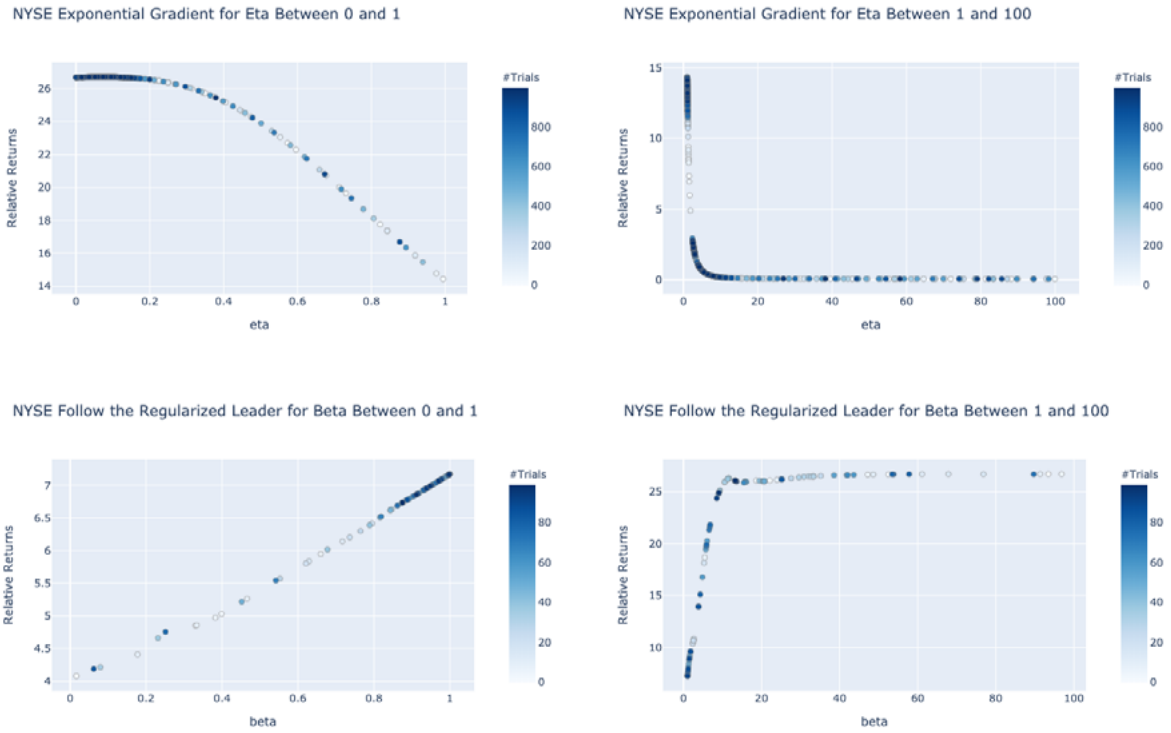
6. 44 Largest US Stocks by from 2011 to 2020 by Alex Kwon

NYSE: 1961-1984



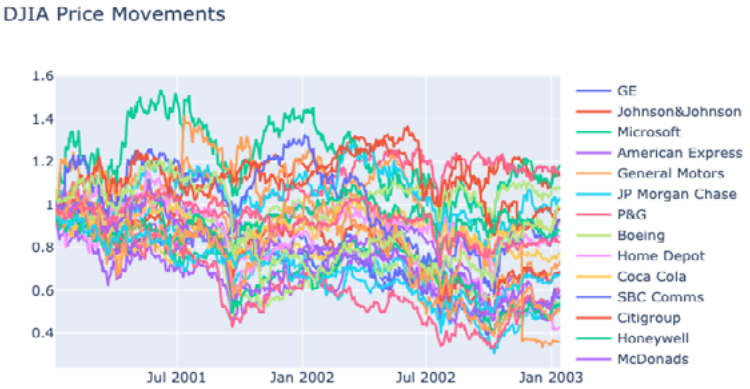
Generally, most assets in this dataset increased by a significant amount. The notable outperforming companies are American Brands and Commercial Metals, and the least performing stock, DuPont, still ended with 2.9 times returns as no stocks in this list decreased in value.

We will be using Optuna and dividing the parameters into two sets. The first set will include parameter values between 0 and 1 to emphasize the importance of adhering to the previous portfolio weights, and the second set will examine values between 1 and 100, which will present significant returns if changing weights benefit the overall returns.



From Helmbold's original paper, η of 0.05 was suggested for Exponential Gradient, and through our tuning, we discovered that the best parameter was 0.0736, which is very similar to the suggested η value.

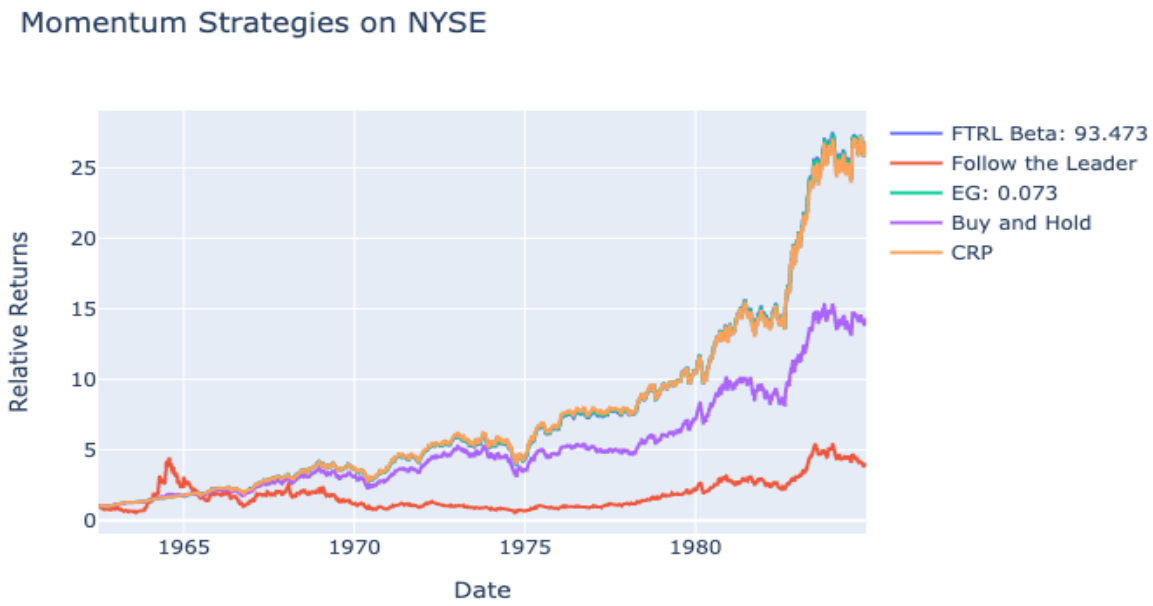
Follow the Regularized Leader's β with the highest returns indicate similar returns to the highest returns by Exponential Gradient.



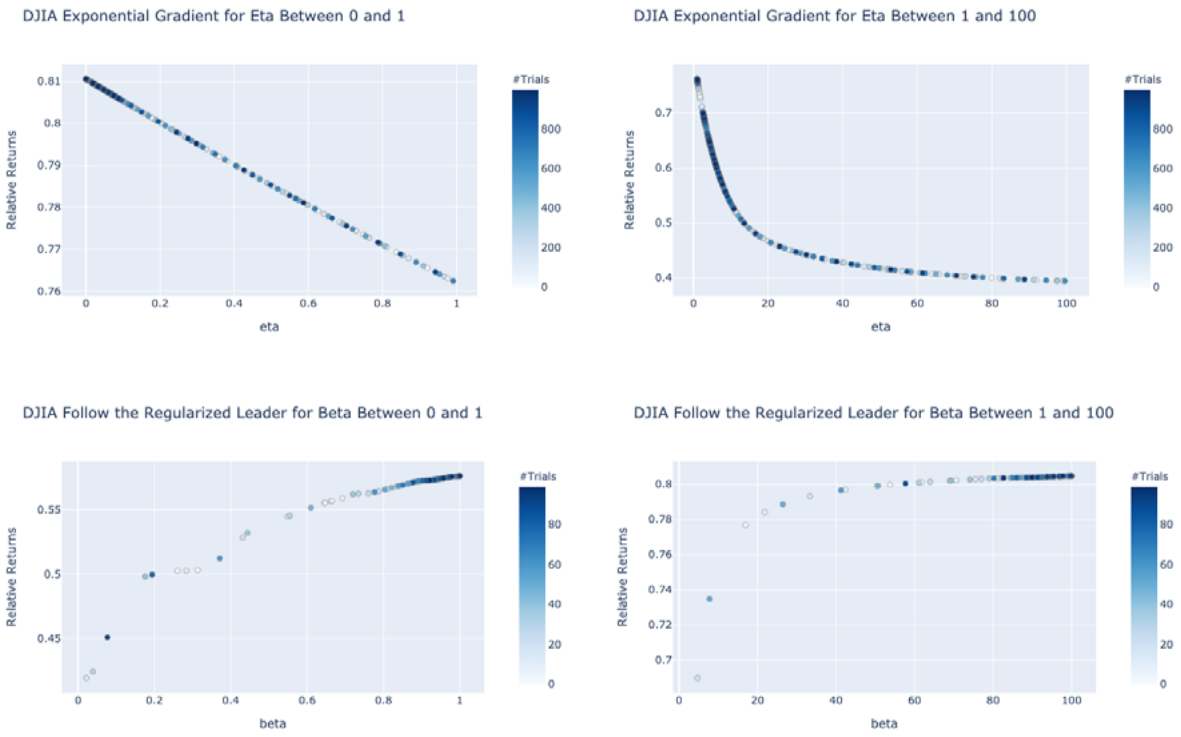
A simple buy and hold returned 12, whereas the best performing Exponential Gradient returned 26.7.

For a dataset where all stocks increase in value and a uniform buy and hold strategy is profitable, a low EG value was adequate. Huge deviations from the original portfolio weight were not necessary and blindly following the best performing asset often decreased the returns as the momentum for these sets of stocks did not continue on a daily basis.

DJIA: 2001-2003

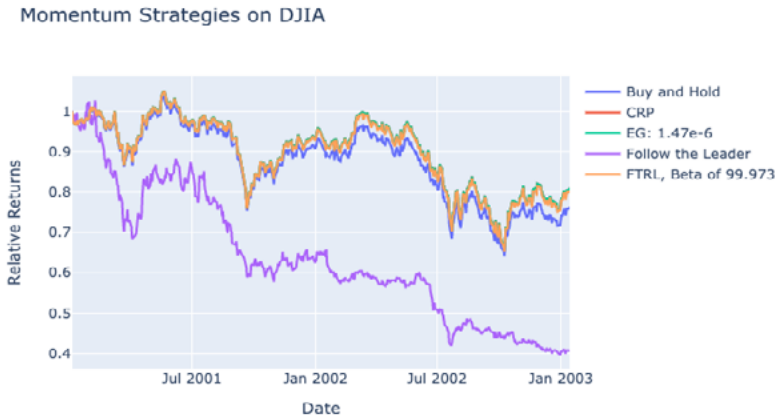


DJIA from 2001 to 2003 provides strikingly different patterns compared to the previous NYSE data. Only 5 companies increased in price as most declined at a steady rate
The ideal η for Exponential Gradient was a value close to 0, and ideal β was close to 100. These parameters



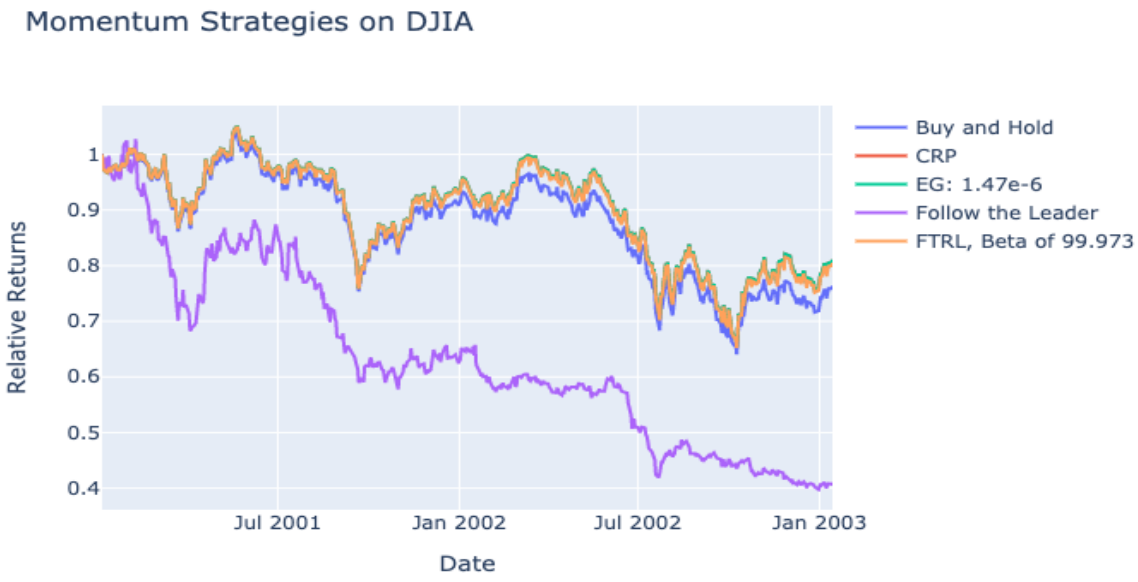
suggest that the weights should not change from its original set value. The regularization term in FTRL prevents the strategy from tracking the BCRP, and the learning rate in EG prevents the strategy from following the Best Stock.

For a market with a general downtrend, momentum strategies fail to make a significant effect on our returns.



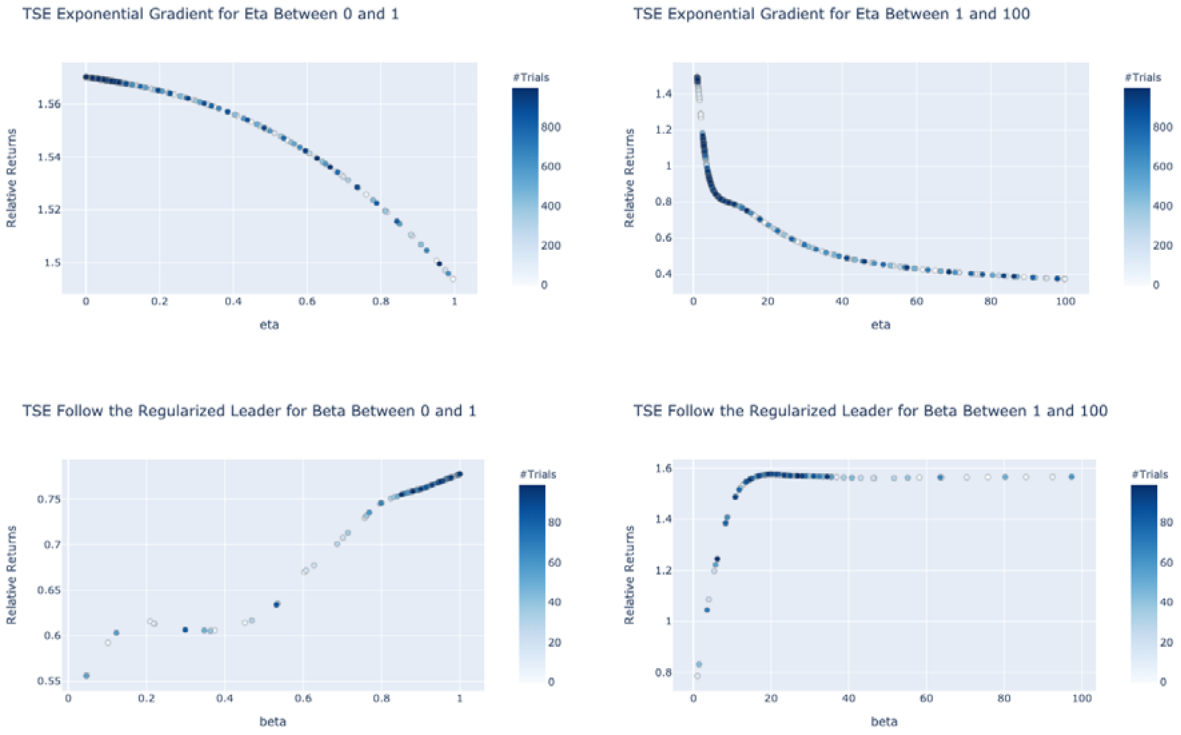
This is in line with the underlying concept for momentum where a strong trend in a direction is necessary to reap the rewards. With our current problem formulation that prevents shorting assets, a momentum strategy cannot produce meaningful returns in this environment. It is not surprising that both Exponential Gradient and Follow the Leader fail to have higher returns than a simple buy and hold strategy.

TSE: 1994-1998



The Toronto Stock Exchange data includes a collection that may be unfamiliar to most researchers. It is a unique universe with half of the stocks decreasing in value. With a combination of both overperforming and underperforming stocks, selection strategies need to identify the ups and downs to have profitable returns.

TSE's ideal η is close to 0, and ideal β is 19.826. A magnitude of β was necessary to lift FTRL returns, whereas η of 0 was better for EG.



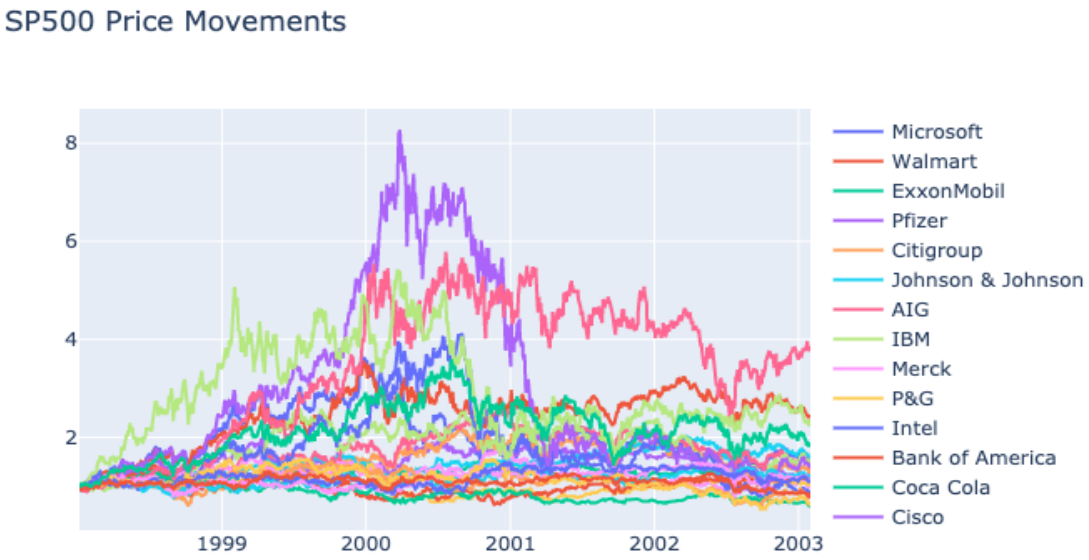
The highest returns for FTRL and EG were 1.57, which was achieved by both CRP and buy and hold. Accounting for transaction cost and rebalancing, momentum strategy was ineffective for TSE as well. The presence of such a volatile but performing best stock in TSE would suggest that the momentum strategies would follow these trends. However, following the TSE price movements graph, the changes in prices are not predictive and extremely sudden. With every increase, a slight decrease follows, which represents a mean reversion trend and not a momentum one.

SP500: 1998-2003

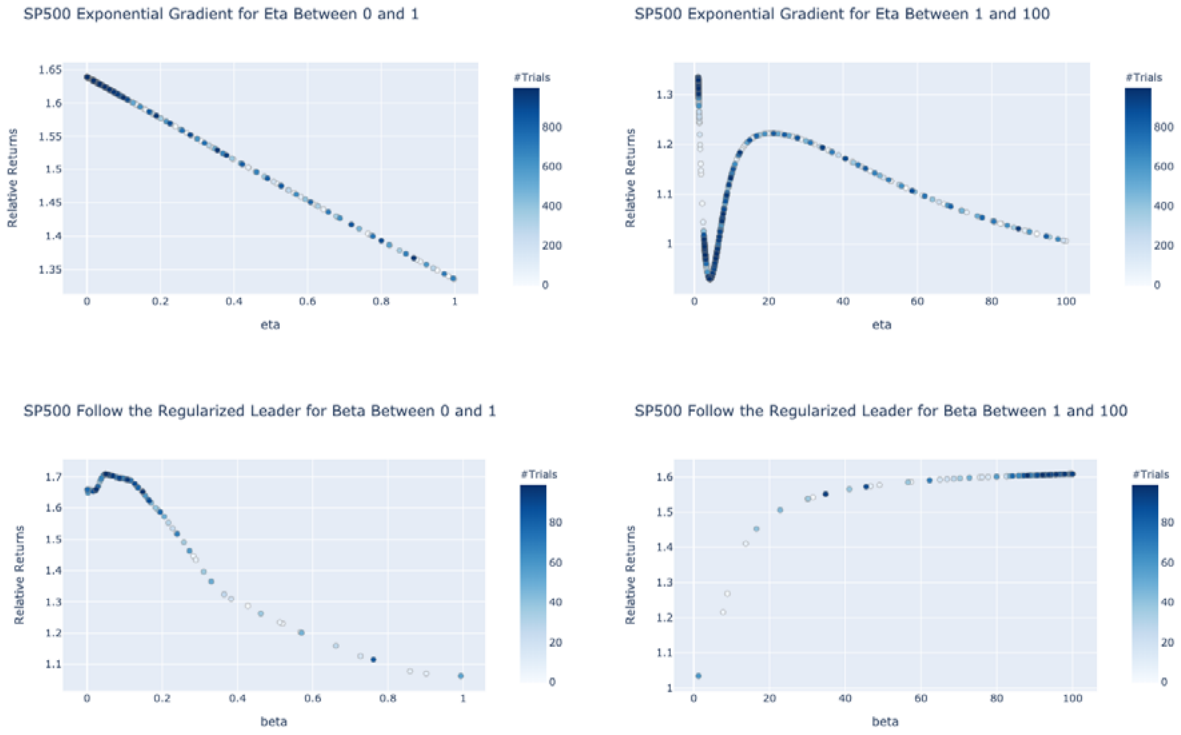
This dataset also includes the bear and bull run during turbulent times. It is longer than the DJIA data by 3 years and includes many companies that are familiar to us.

SP500 during this time goes through the bear market in 2000, and in the long run, all but 5 companies increase in value.

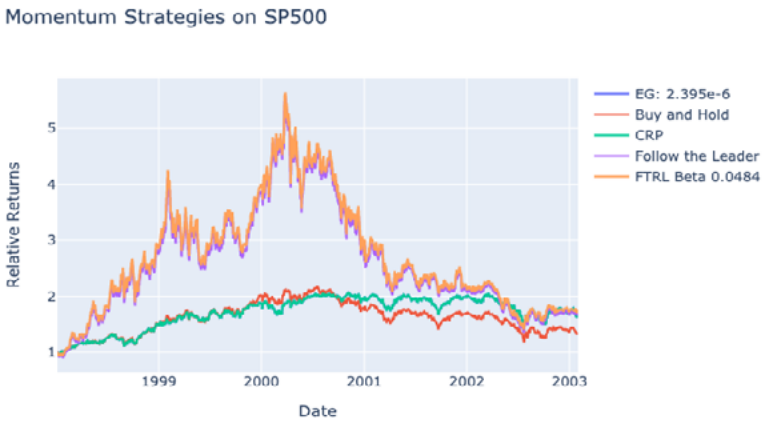
Ideal η is 0, and β is 0.048. For FTRL, we see an increase in returns for higher values of β, and in fact, the



returns for β of 100 is very similar to the best performing FTRL. This is in line with the results displayed below as buy and hold returned the same amount as FTRL.



SP500 during this time represents a tale of two periods. The first half has a momentum rally with FTRL returning the highest returns; however, after the peak from 2000, the portfolio rapidly decreases and converges to the rest of the strategies.

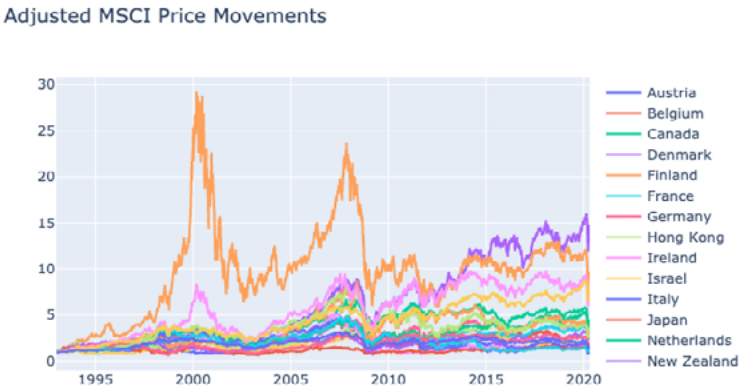


Follow the Leader, CRP, and EG all have similar returns that are marginally higher than a buy and hold, and this is another example where a lack of clear direction and continuous momentum hinders the ability to effectively predict the change in prices.

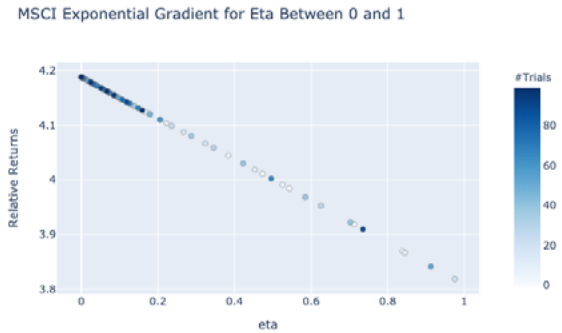
MSCI: 1993-2020

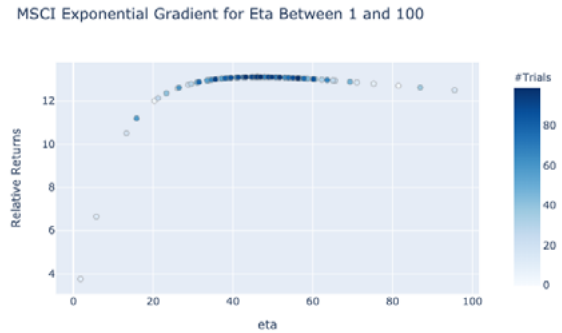
Different from traditional assets, the world indexes capture much more than just the price changes of individual companies. With an overarching representation of the countries' market states, these market indexes will present a different idea for applications of online portfolio selection strategies.

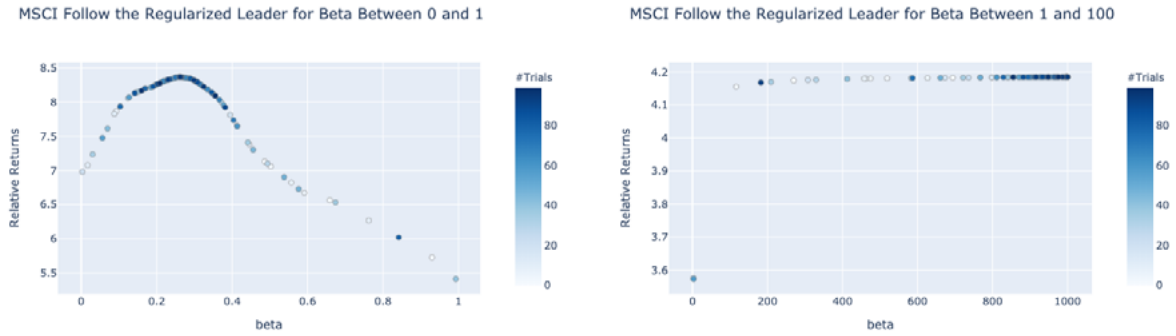
Finland is not the first country to come in mind with metrics like these, but the rise and fall of Finland around the 2000s puts every other country aside. Most countries show movements that are strongly correlated with each other.



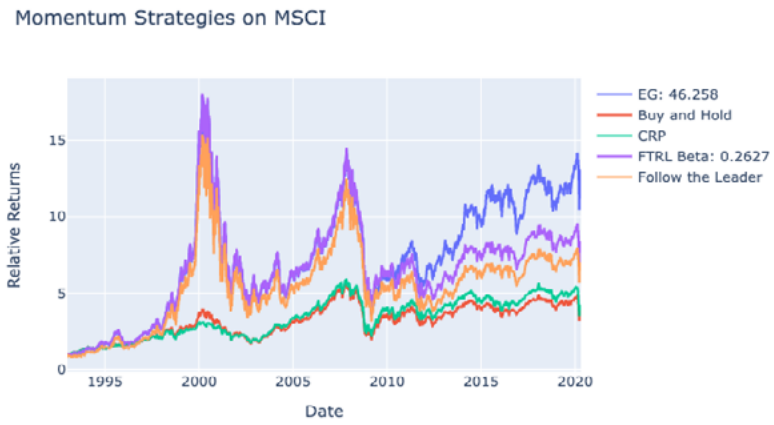
Ideal η is 46.25, and β is 0.2627. We see η and β values that are different from previous datasets. Both values indicate a measure of slight regularization to adhere to previous portfolio weights, but also emphasize the need to follow the trending asset.







For the MSCI dataset, we see significantly higher returns of momentum strategies near the 2000s. Primarily because of Finland's rapid increase from the late 1990s, EG and FTL captured and put all of its weights onto Finland. If there is an asset that performs far better for a long period, momentum strategies are effective.



A higher η and lower β can blindly follow the performing asset and produce higher returns. However, the trend-following strategy is extremely volatile and can also create major drawbacks to the portfolio over time.

US Equity: 2011-2020

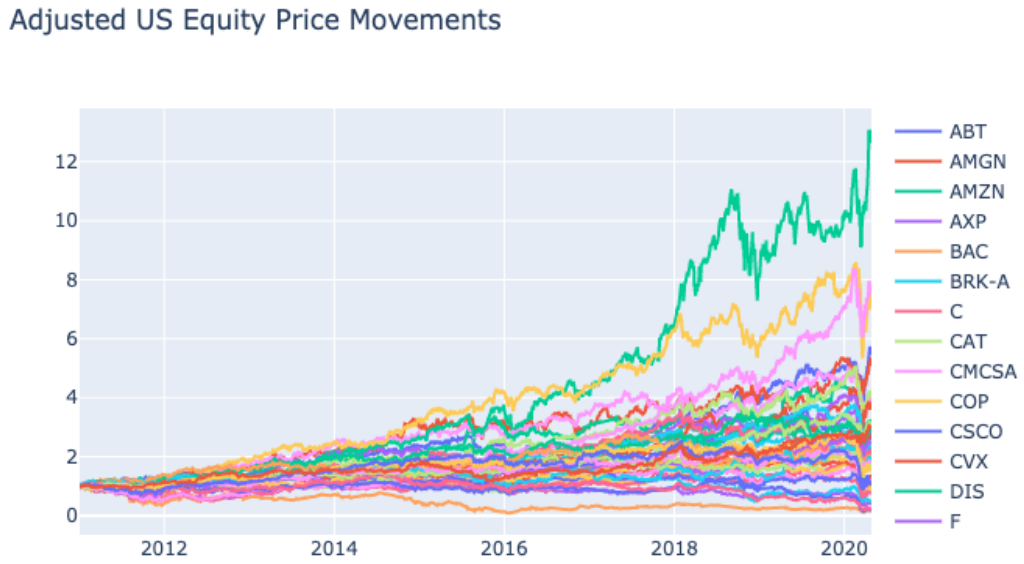
For a more recent dataset, we collected the 44 largest US stocks based on market capitalization according to a Financial Times report.

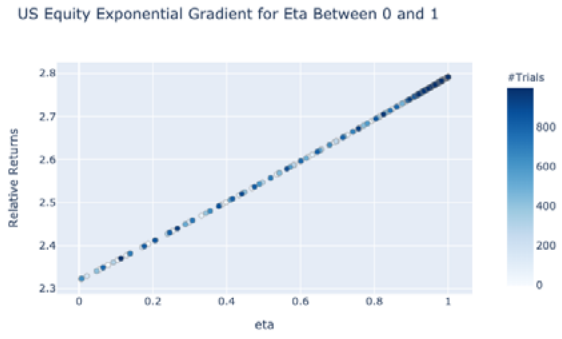
Although included in the original report, we did not include United Technologies and Kraft Foods due to M&A

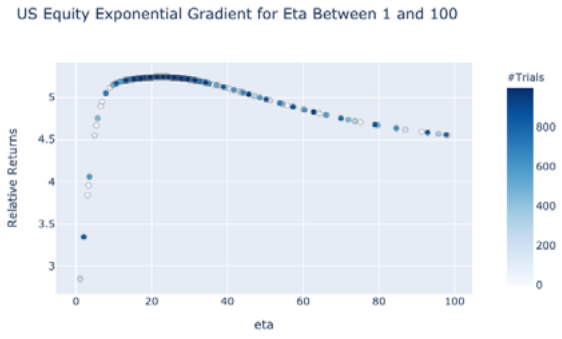
and also excluded Hewlett-Packard because of the company split in 2015.

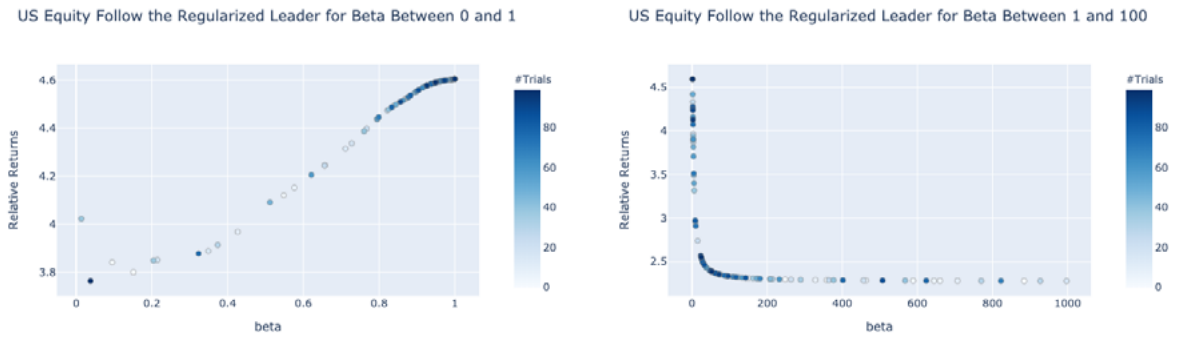
This dataset will be particularly interesting because it also includes the recent market impact by the coronavirus as well. With 10 years of continuous bull run after the financial crisis in 2008, we can examine which strategy was the most robust to the rapidly changing market paradigm in the last month.

Ideal η is 22.67, and β is 0.9999.

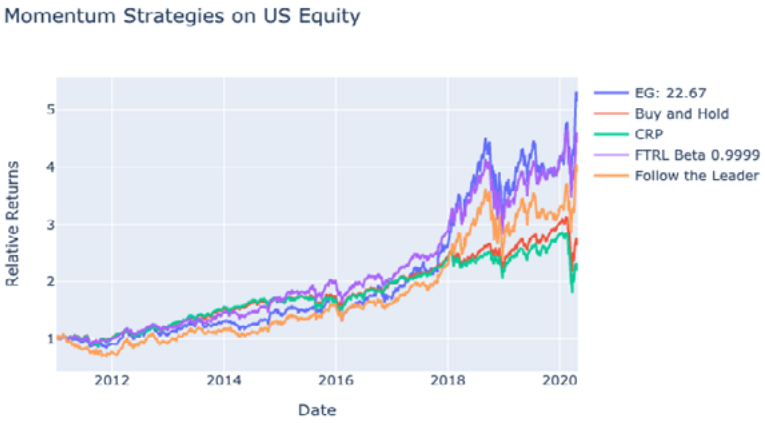








For US Equity, momentum strategies also outperform buy and hold. Starting from 2018, the gap between the benchmarks becomes larger as momentum catches onto Amazon's rapidly growing prices. There is still an insurmountable gap between Amazon's



performance and other strategies as Amazon has been performing incredibly well in the last decade. Moreover, the 10-year bull run allowed our strategies to progress well without too many drawdowns during the period.

CONCLUSION

We were able to explore the momentum functionalities of PortfolioLab's newest Online Portfolio Selection module. Readers were exposed to a basic introduction to the momentum strategies and will be able to replicate results using the simple methods of the new module.

4.2

ONLINE PORTFOLIO SELECTION: MEAN REVERSION

INTRODUCTION

Mean Reversion is an effective quantitative strategy based on the theory that prices will revert back to its historical mean. A basic example of mean reversion follows the benchmark of Constant Rebalanced Portfolio. By setting a predetermined allocation of weight to each asset, the portfolio shifts its weights from increasing to decreasing ones. This module will implement four types of mean reversion strategies:

1. Passive Aggressive Mean Reversion
2. Confidence Weighted Mean Reversion
3. Online Moving Average Reversion
4. Robust Median Reversion.

Through this article, the importance of hyperparameters is highlighted, as the choices greatly affect the outcome of returns. Many of the hyperparameters for traditional research has been chosen by looking at the data in hindsight, and fundamental analysis of each dataset and market structure is required to profitably implement this strategy in a real-time market scenario.

We will be introducing the applications of these popular strategies with the ease of using PortfolioLab's newest module. We hope that the newfound interests from these topics will spur more development as well as assist the research process for many others in the field.

PASSIVE AGGRESSIVE MEAN REVERSION

Passive Aggressive Mean Reversion

Passive Aggressive Mean Reversion alternates between a passive and aggressive approach to the current market conditions. The strategy can effectively prevent a huge loss and maximize returns by setting a threshold for mean reversion.

PAMR takes in a variable ϵ, a threshold for the market condition. If the portfolio returns for the period are below ϵ, then PAMR will passively keep the previous portfolio, whereas if the returns are above the threshold, the portfolio will actively rebalance to the less performing assets.

In a way, ϵ can be interpreted as the maximum loss for the portfolio. It is most likely that the asset that decreased in prices for the period will bounce back, but there are cases where some companies plummet in value. PAMR is an effective algorithm that will prevent huge losses in blindly following these assets. Three different methods of PAMR have been implemented in the module with different levels of aggressiveness towards mean reversion.

CONFIDENCE WEIGHTED MEAN REVERSION

Extending from PAMR, Confidence Weighted Mean Reversion looks at the autocovariance across all assets. Instead of focusing on a single asset's deviation from the original price, CWMR takes in second-order information about the portfolio vector as well to formulate a set of weights.

For CWMR, we introduce Σ, a measure of anti-confidence in the portfolio weights, where a smaller value represents higher confidence for the corresponding portfolio weights.

The problem can be interpreted as maximizing the portfolio confidence by minimizing Σ given a confidence interval θ determined by the threshold, ϵ. Two different solutions to the problem were implemented in the module with the standard deviation and variance method.

ONLINE MOVING AVERAGE REVERSION

Traditional mean reversion techniques have an underlying assumption that the next price relative is inversely proportional to the latest price relative; however, mean reversion trends are not limited to a single period. Unlike traditional reversion methods that rely on windows of just one, OLMAR looks to revert to the moving average of price data.

OLMAR proposes two different moving average methods: Simple Moving Average and Exponential Moving Average. From these moving average methods, the strategy predicts the next period's price relative, and the strategy iteratively updates its new weights.

ROBUST MEDIAN REVERSION

Robust Median Reversion extends the previous Online Moving Average Reversion by introducing the L1 median of the specified windows. Instead of reverting to a moving average, RMR reverts to the L1 median estimator, which proves to be a more effective method of predicting the next period's price because financial data is inherently noisy and contains many outliers.

Calculating the L1 median is computationally inefficient, so the Modified Weiszfeld Algorithm is used to speed up the process.

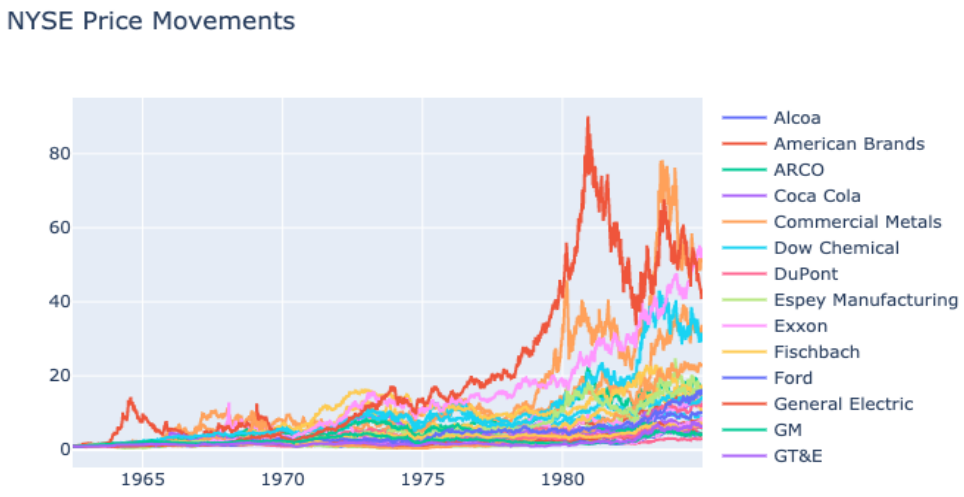
DATA

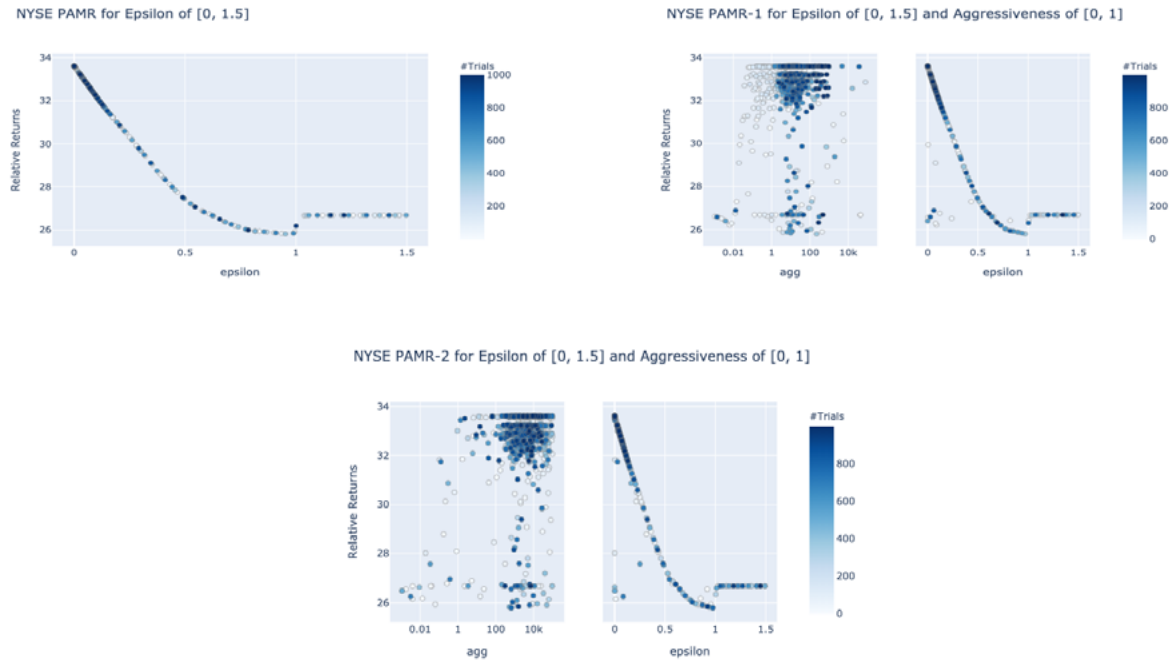
We will be using 6 different datasets, and if you would like to learn more about each dataset, exploratory analysis is available here.

1. 36 NYSE Stocks from 1962 to 1984 by Cover
2. 30 DJIA Stocks from 2001 to 2003 by Borodin
3. 88 TSE Stocks from 1994 to 1998 by Borodin
4. 25 Largest S&P500 Stocks from 1998 to 2003 by Borodin
5. 23 MSCI Developed Market Indices from 1993 to 2020 by Alex Kwon
6. 44 Largest US Stocks by from 2011 to 2020 by Alex Kwon

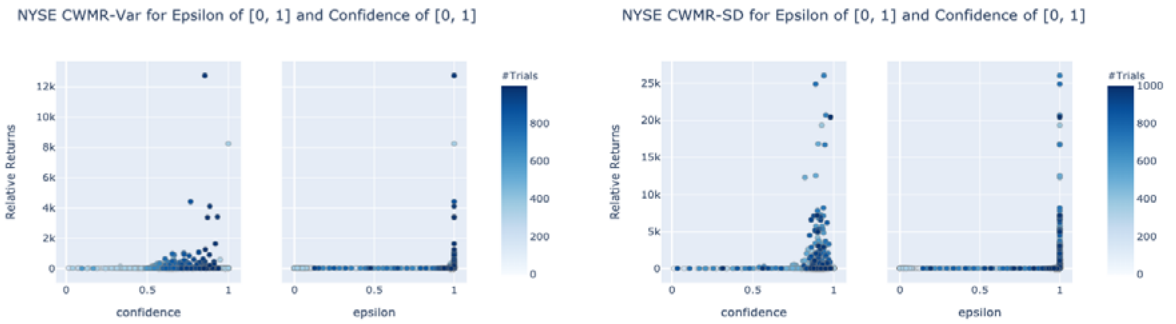
We will be using Optuna to examine the effects of parameters.

NYSE: 1962-1984

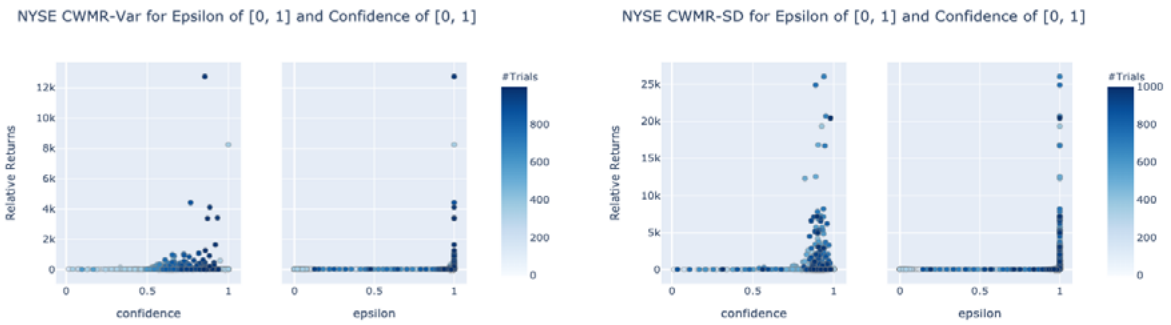




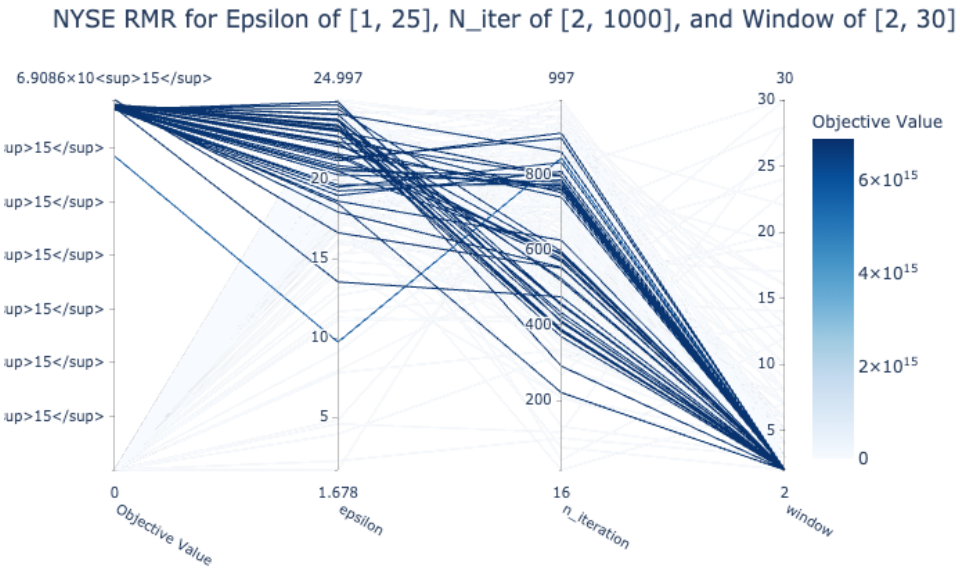
PAMR results on NYSE indicate the dependency on epsilon value over agg(aggressiveness). A low value of epsilon, one that is close to 0, is ideal in this strategy and indicates the preference for passive mean reversion.



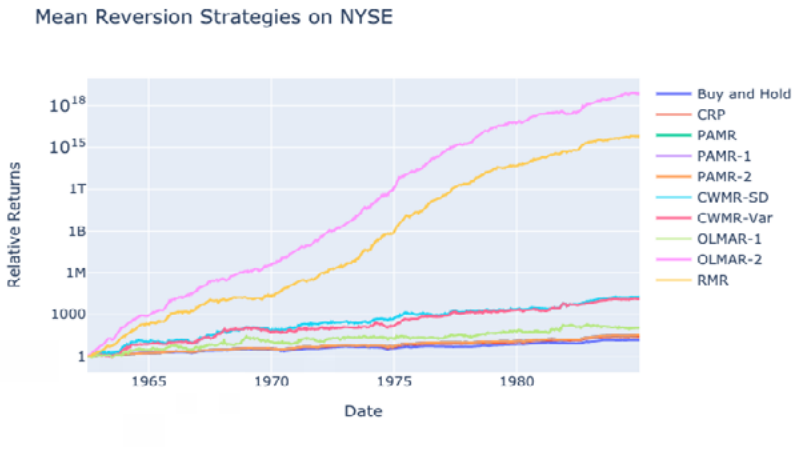
CWMR results vary over the intervals of confidence and epsilon. The majority of the trials returned around the same magnitude; however, for certain parameters, CWMR produced extremely high returns. The high returns have a common parameter of a high epsilon value of 1.



OLMAR's returns are exponentially higher than other mean reversion techniques so far with magnitudes of 10^{18}. OLMAR returns are more dependent on the alpha and window parameters that choose the moving averages rather than epsilon. Interestingly, a window of 23 was ideal for OLMAR-1, and an alpha of 0.35 was ideal for OLMAR-2. A tendency of mean reversion for a longer period appears for the NYSE dataset proven by the relatively lower value of alpha and window of closer to three weeks.

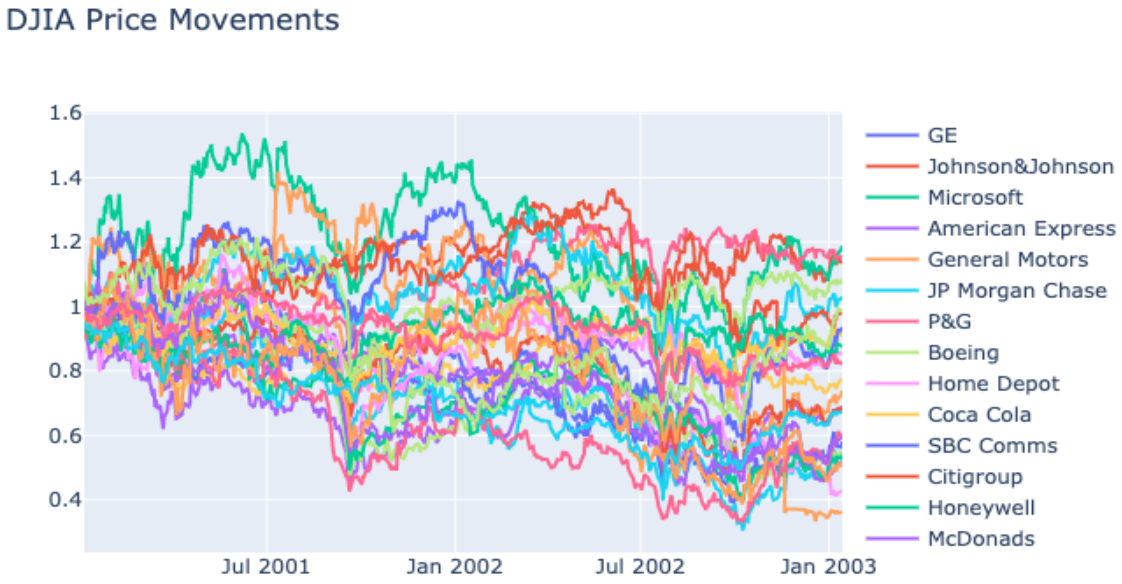


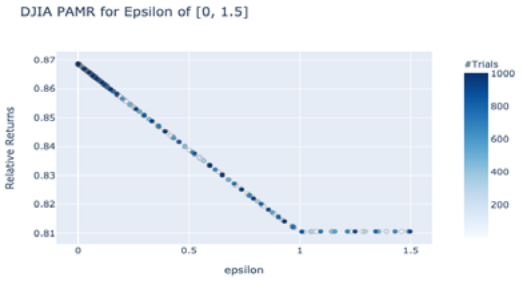
From the graph, we can notice that darker lines indicate higher returns. The majority of the dark line goes through a high epsilon and low window value. The range for the number of iterations and epsilon is wide, but the results were primarily dependent on the window value of 2.

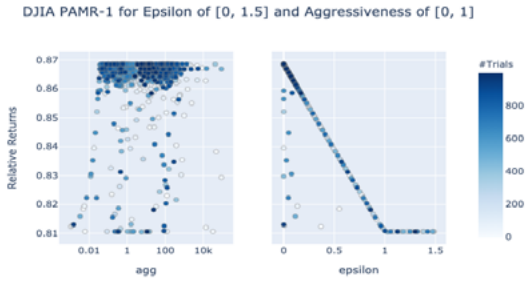


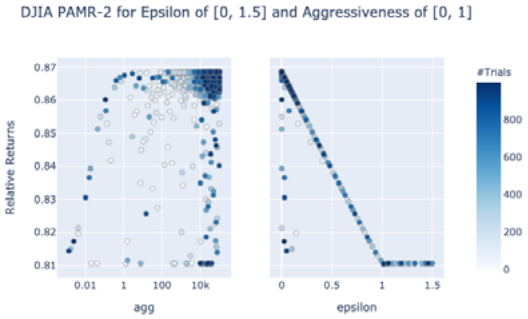
OLMAR-2 and RMR returns are significantly higher than any other strategies with astronomical returns of 10^{18}.

DJIA: 2001-2003

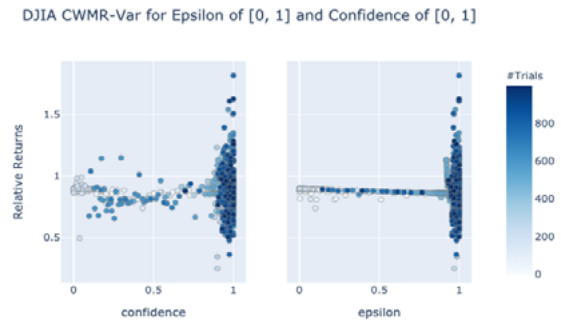


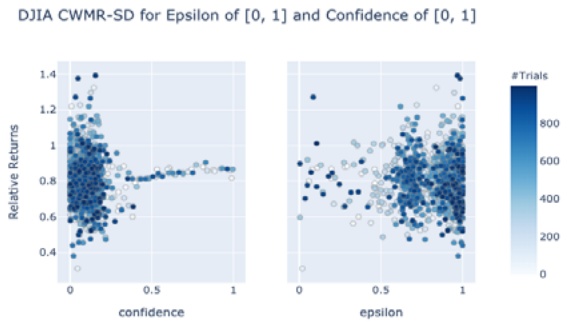






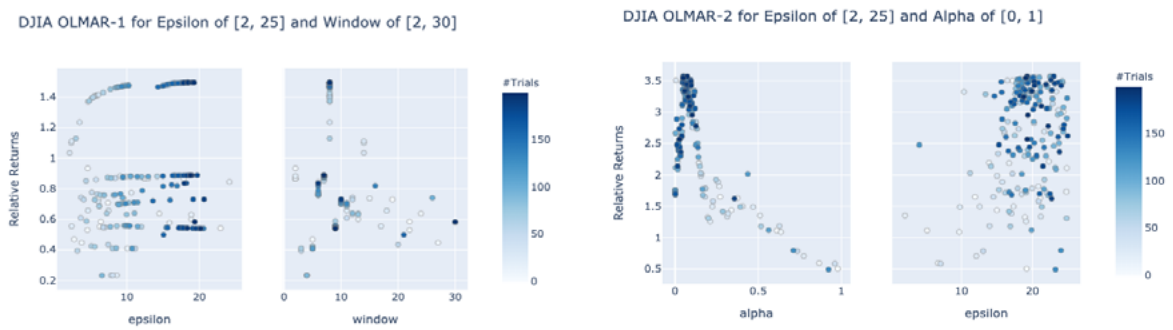
Similarly to NYSE, DJIA data is dependent on the epsilon parameter. A lower value of epsilon, closer to 0,





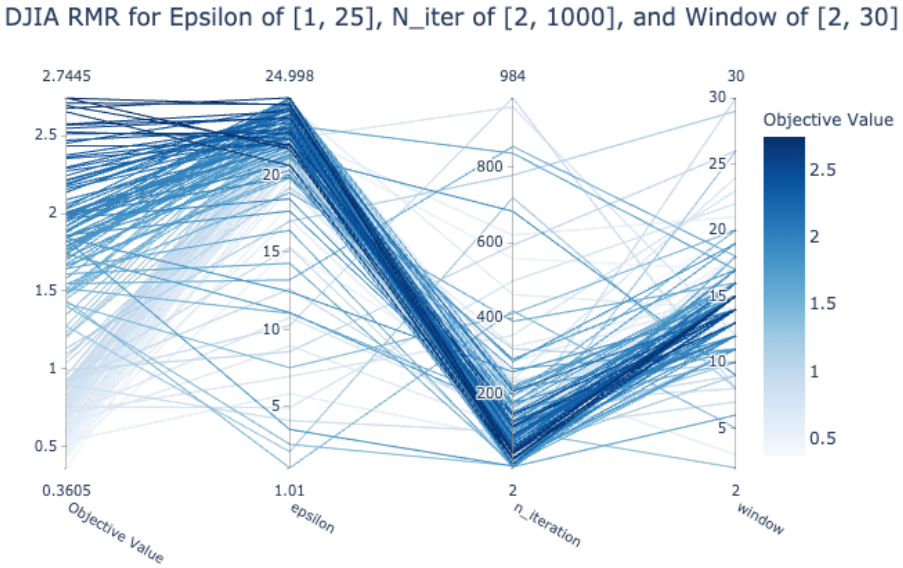
produced higher returns regardless of aggressiveness.

CWMR produces a similar graph to NYSE's. Epsilon of 1 continues to be the ideal parameter; however,

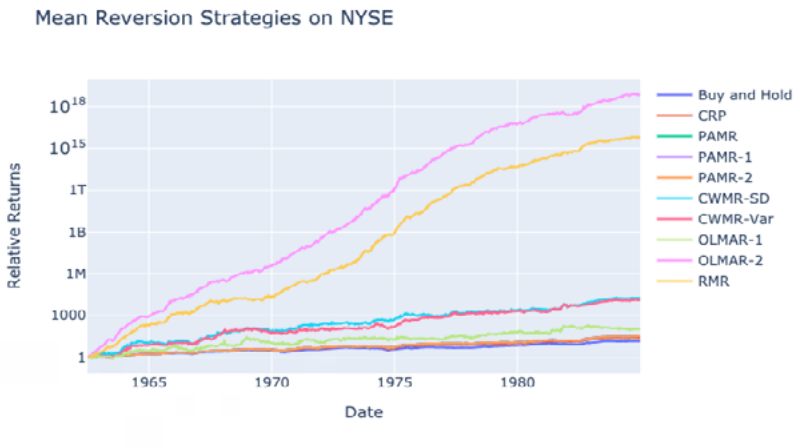


confidence is different for CWMR-Var and CWMR-SD. The best confidence value for the variance method is 1, whereas the standard deviation's parameter is closer to 0.

OLMAR parameters suggest a similar trend where the epsilon parameter does not significantly affect results.

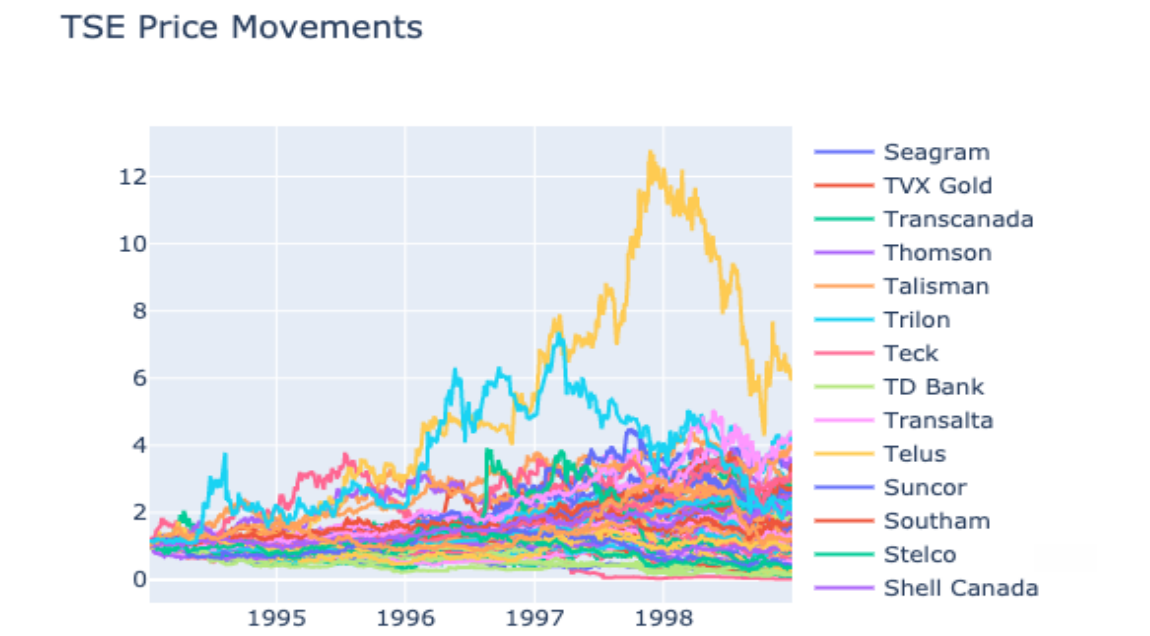


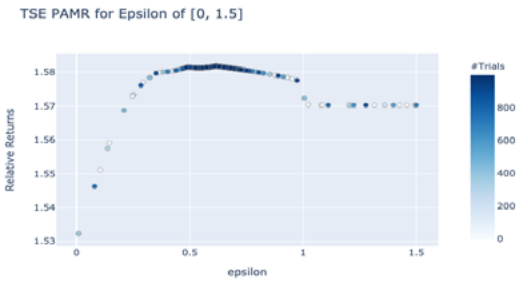
A window of 8 represents a shorter mean reversion trend with the DJIA dataset, and a low alpha value of 0.1 indicates less importance on the current weights as a parameter for price prediction.

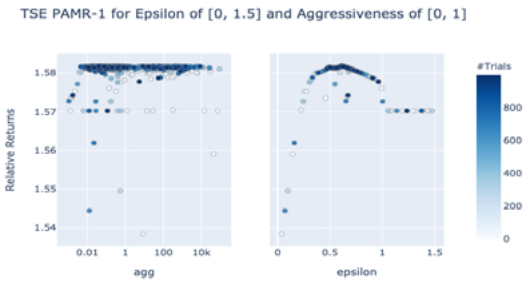


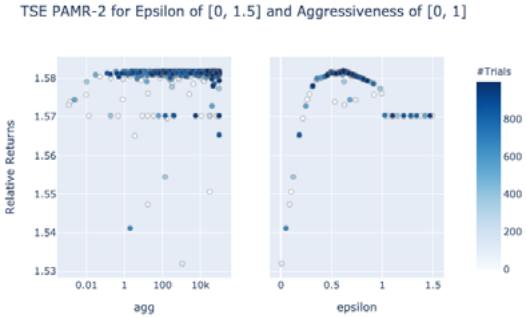
Higher returns were achieved from high epsilon, a lower number of iterations, and a window range of 12 to 16. This is in contrast to the window range suggested by OLMAR-1, where 8 had the highest returns. A higher window value for RMR compared to OLMAR-1 indicate that the noise in price data is more effectively reduced with a wider range of window.

TSE: 1994-1998

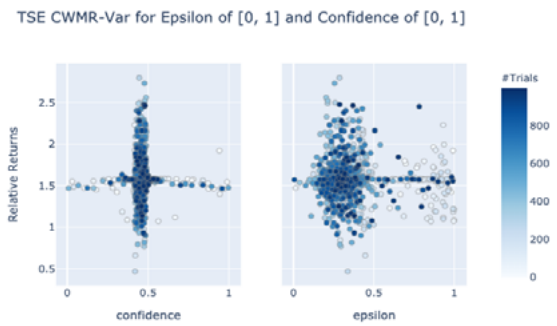


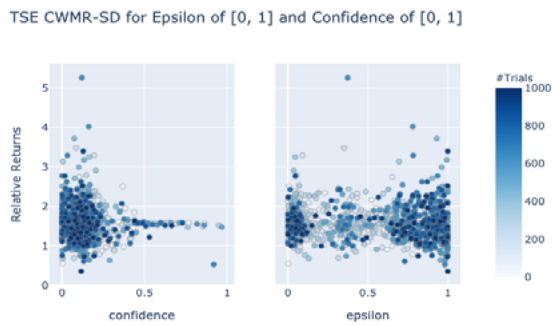




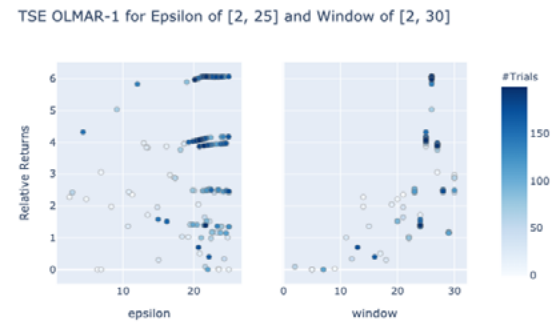


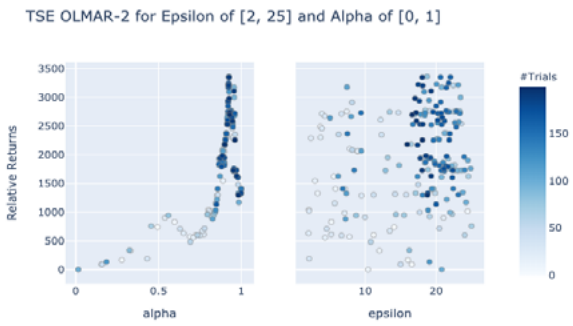
Mean reversion strategies perform significantly better than other benchmarks. For a period of a continuous downtrend, OLMAR-2 and RMR each have triple and double returns when most other strategies are stagnant around 1. Mean reversion provides promising returns for a long bear market.



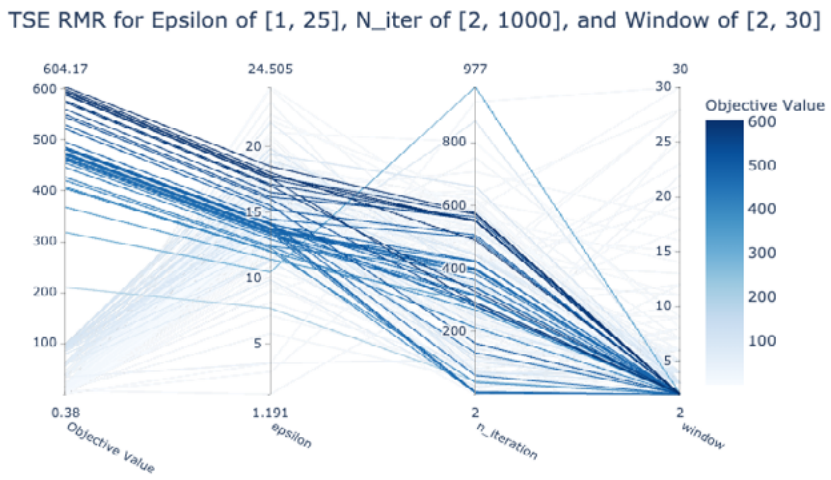


TSE suggests a different ideal epsilon value from the two previous datasets. Instead of 0, the best epsilon value is 0.6. The strategy performed an aggressive strategy for portfolio returns that were above 0.6 and performed a passive strategy for periods with returns less than 0.6. The returns are also relatively less sensitive to epsilon value as models with epsilon value between 0.4 and 1 have similar returns.



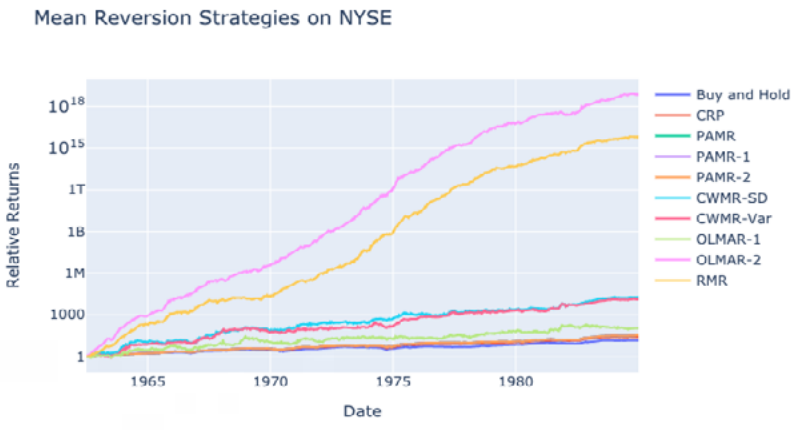


CWMR parameters are very different from the previous datasets. For the standard deviation method, epsilon values have minimal effect on returns as the highest drive was the low value of confidence around 0.1. For the variance method, a confidence value of 0.5 displayed significantly higher returns with epsilon not having as



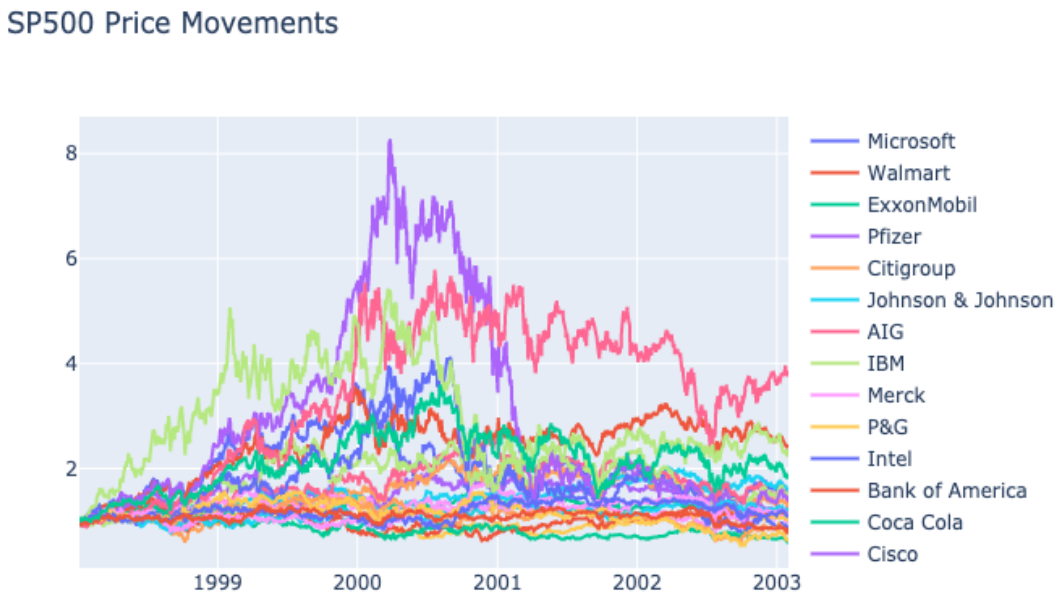
much effect as it did on the standard deviation method.

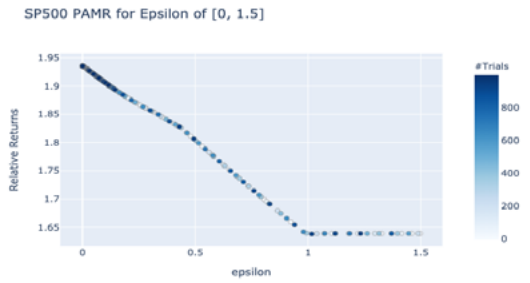
TSE also represents a longer mean reversion window of 23 and an alpha of 0.9. The difference between the two parameters is noteworthy as a high value of alpha indicates a stronger weighting for recent prices; however, the

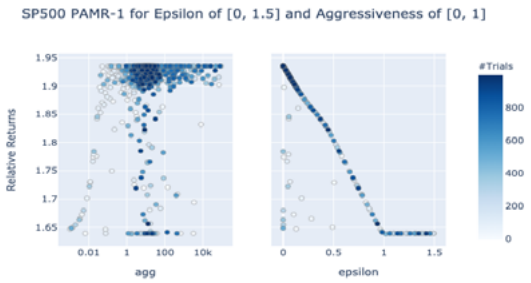


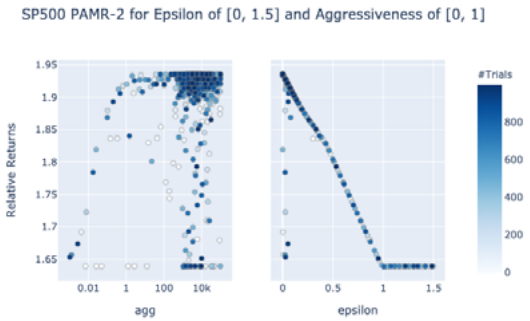
high window value of 23 suggests that the periods are reliant on a more longer-term.
RMR results indicate that the most important parameter is the number of windows. A window of 2 had the highest returns regardless of epsilon and number of iterations. This is in line with the OLMAR-2 strategy that had the best returns with a high alpha value. A shorter window frame was a better indicator of future prices

SP500: 1998-2003



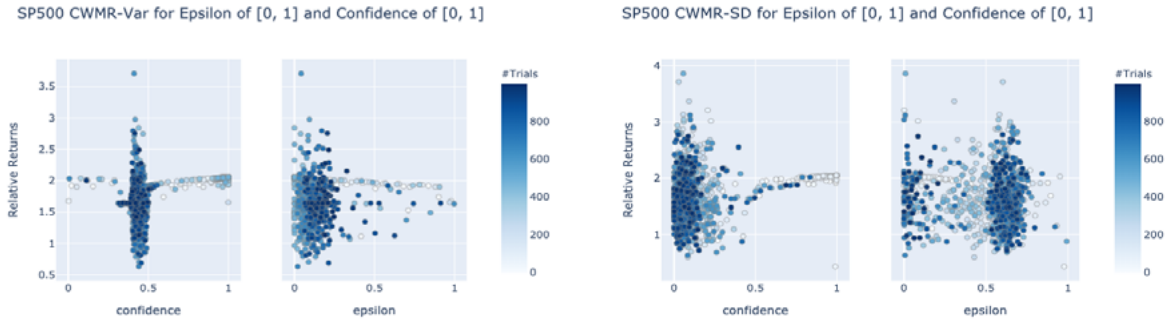




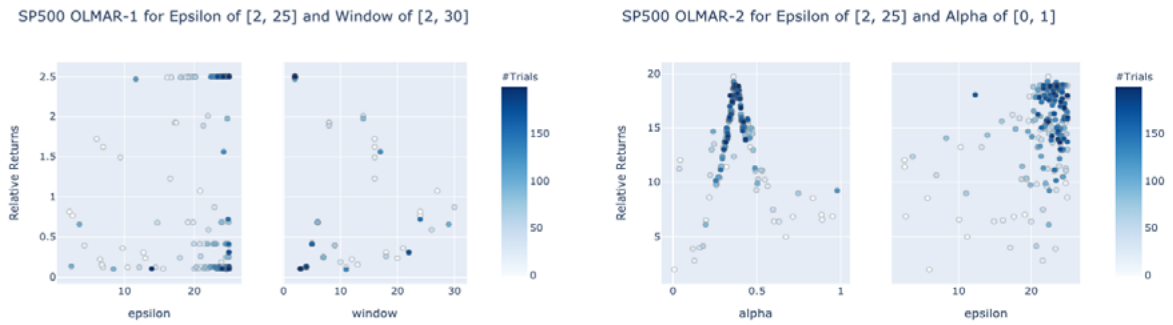


for the TSE dataset.

OLMAR-2 and RMR continue to lead in returns with OLMAR-1 being a distant third. Shorter mean reversion trends proved to be a successful strategy for TSE.

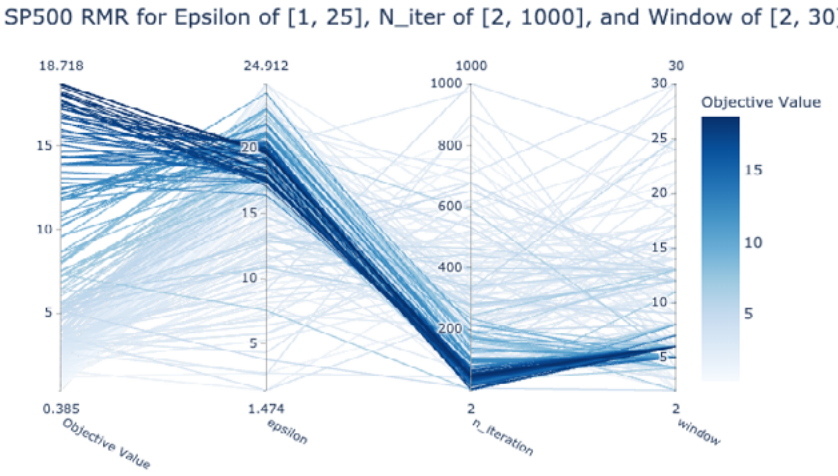


Epsilon of 0 proved to have the highest return as aggressiveness continues to have minimal impact on our returns.

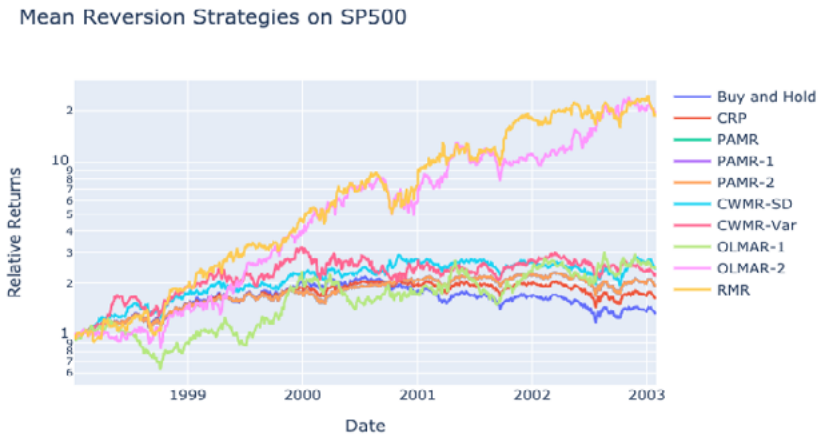


CWMR-Var had its highest and lowest returns at a confidence value of 0.45. CWMR continues to be sensitive to its hyperparameters as even the slightest deviations of parameters return vastly different returns for its models.

For SP500, a window of 2 and an alpha of 0.4 produced the highest returns. OLMAR-1 returns vary for most

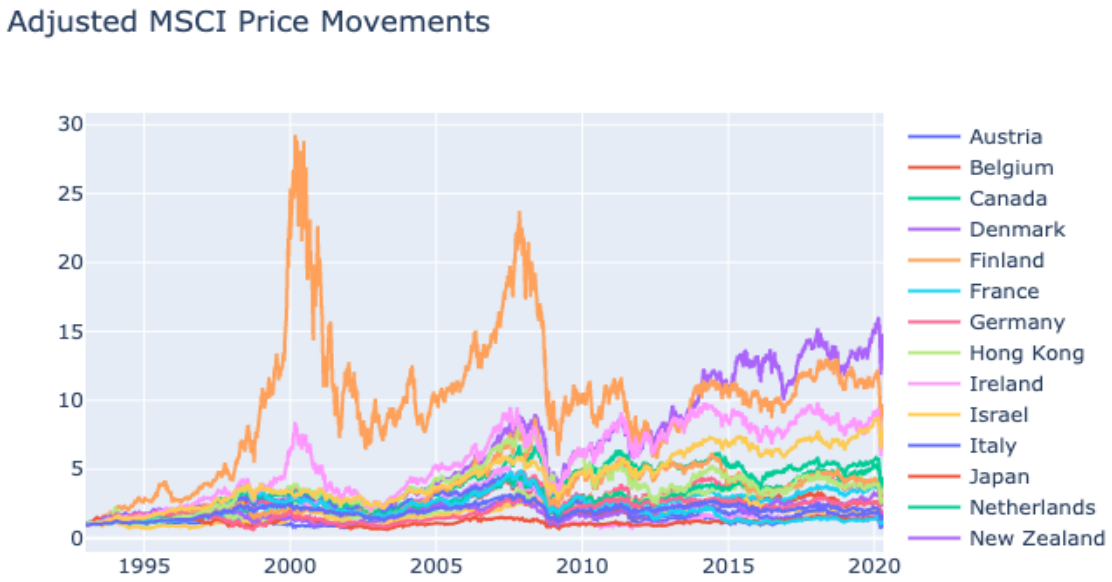


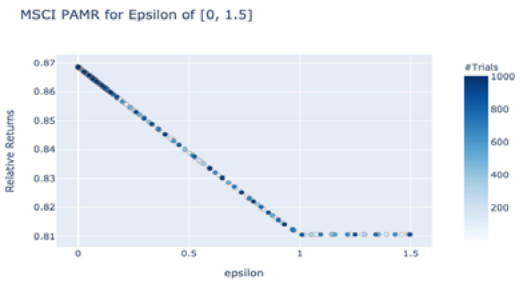
parameter choices and do not indicate a strong trend in any direction; however, OLMAR-2 has a clear alpha value that maximizes returns and is more robust to different parameter choices in the range between 0.3 and 0.5.

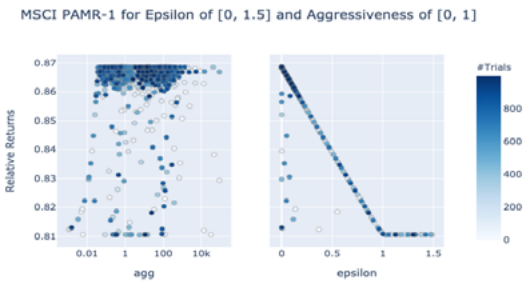


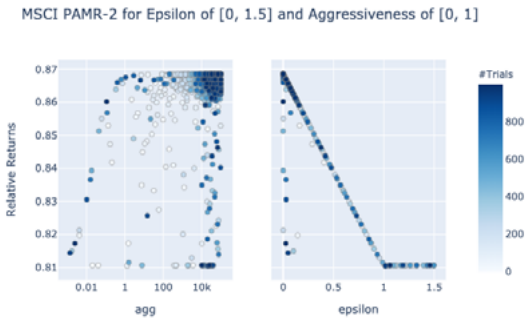
The darkest lines for RMR pass through high epsilon, a low number of iterations, and a window range between 4 and 8.

MSCI: 1993-2020

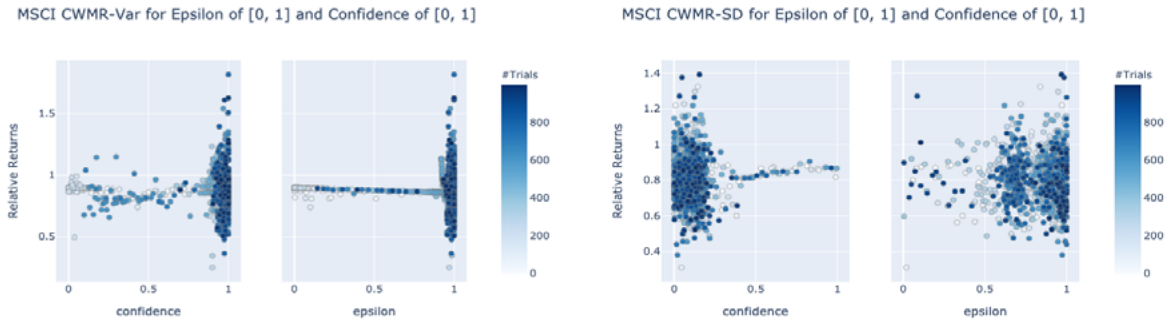




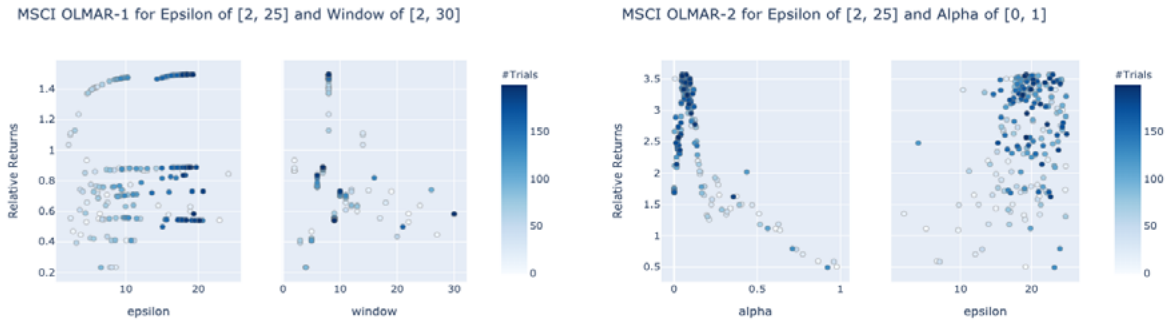




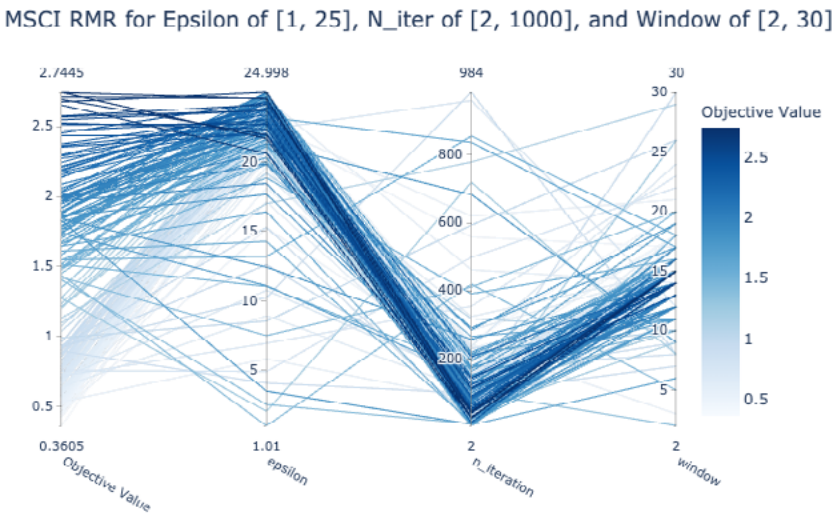
OLMAR-2 and RMR produce the highest returns as the rest of the strategies fail to increase by more than 2 times the original value.



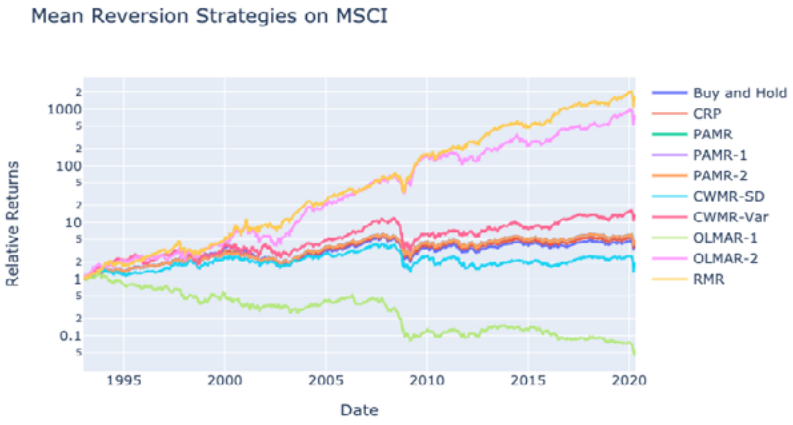
Aggressive mean reversion was less preferred for MSCI with a low epsilon value preferred.



For both sets, epsilon of 1 had the highest returns with low confidence for CWMR-SD and high confidence for CWMR-Var. CWMR continues to provide models that are extremely sensitive to hyperparameter choices.

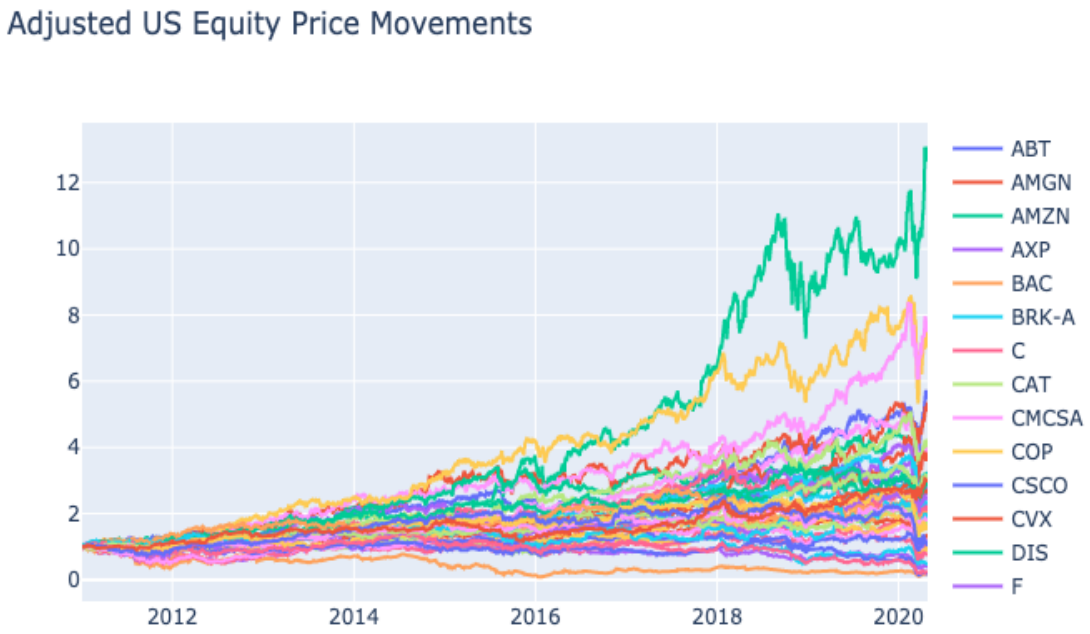


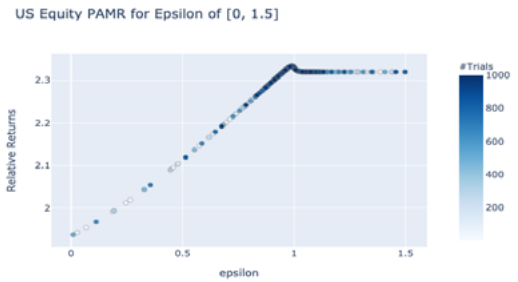
A window of 8 was optimal for mean reversion prediction for OLMAR-1, and OLMAR-2 had an optimal alpha value close to 0.1.

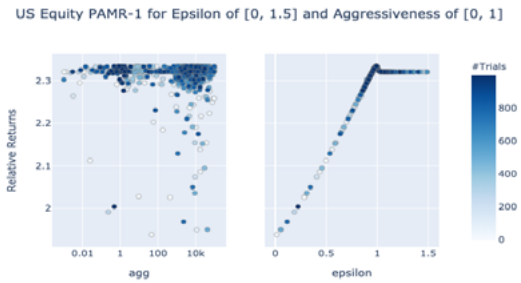


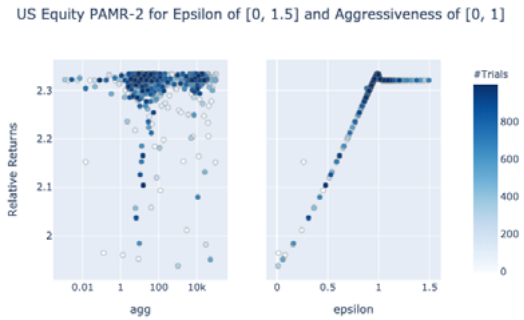
RMR values were highest for high epsilon, a low number of iteration, and a window range of 12 to 15. The relatively higher optimal window value is similar to the optimal parameter for OLMAR-2. Both indicate a longer mean reversion expectancy regarding price predictions.

US Equity: 2011-2020

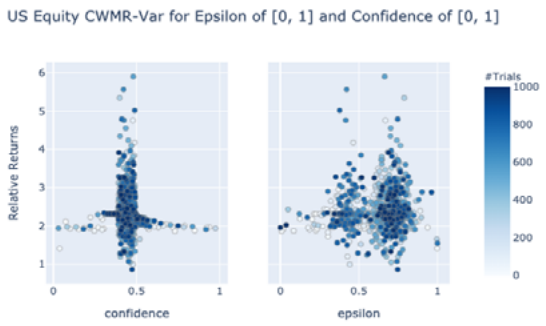


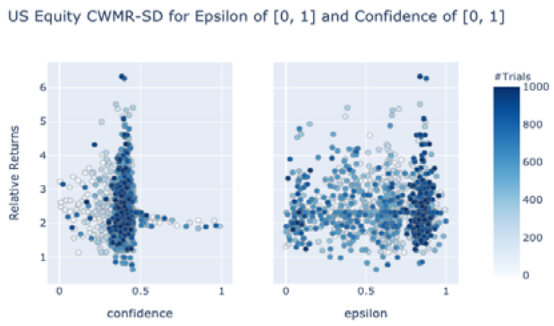




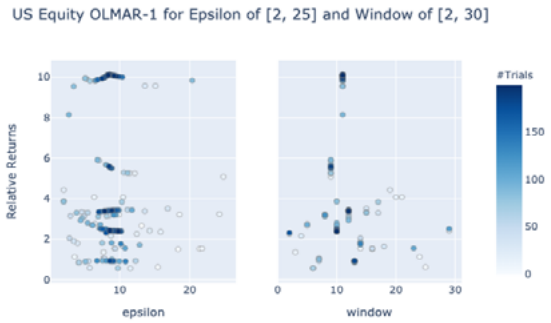


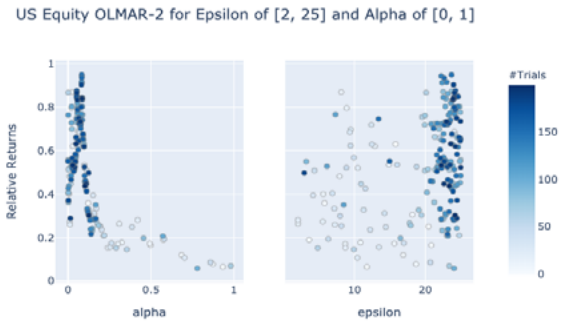
Not too surprisingly, OLMAR-2 and RMR produce the highest returns by more than 100 times in returns compared to other strategies. OLMAR-1, which typically represented a robust model, actually failed to have



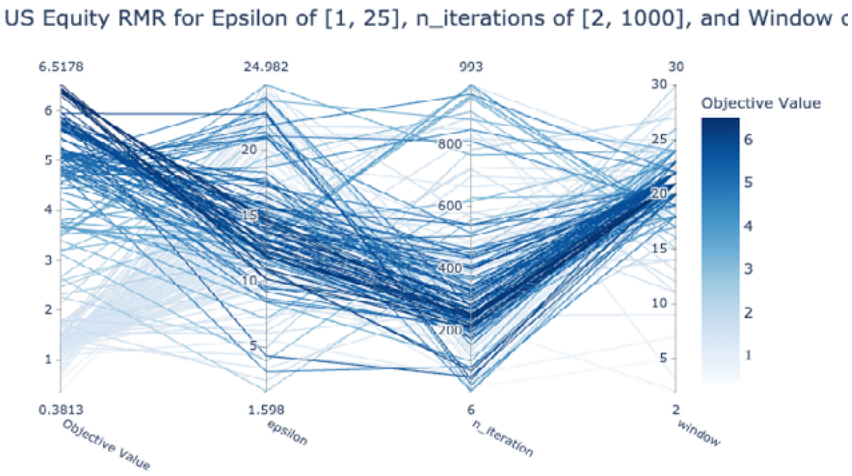


positive returns and approached a portfolio value closer to 0.

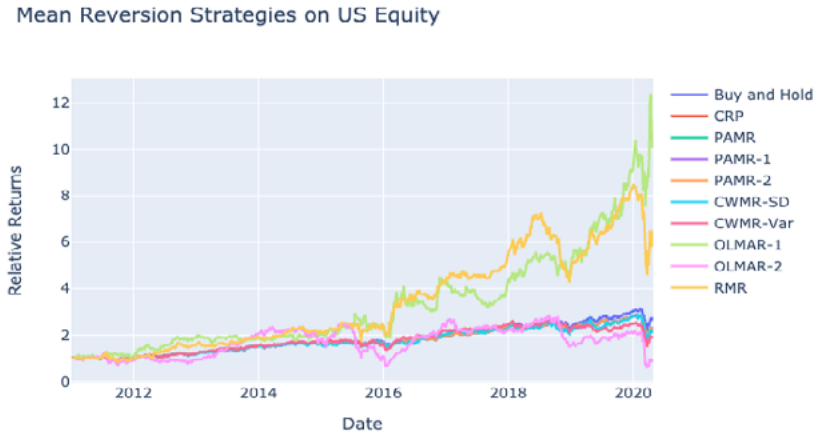




For SP500, a window of 2 and an alpha of 0.4 produced the highest returns. OLMAR-1 returns vary for most parameter choices and do not indicate a strong trend in any direction; however, OLMAR-2 has a clear alpha value that maximizes returns and is more robust to different parameter choices in the range between 0.3 and 0.5.



The darkest lines for RMR pass through high epsilon, a low number of iterations, and a window range between 4 and 8.



CONCLUSION

Mean reversion can be formulated in multiple methods. From a simple constant rebalanced portfolio to a complex robust median reversion, the strategy can employ endless techniques to find the most profitable set of portfolio weights. The article covered a wide range of mean reversion strategies employed in PortfolioLab's newest Online Portfolio Selection module, and readers will be able to replicate results using the simple methods of the new module.

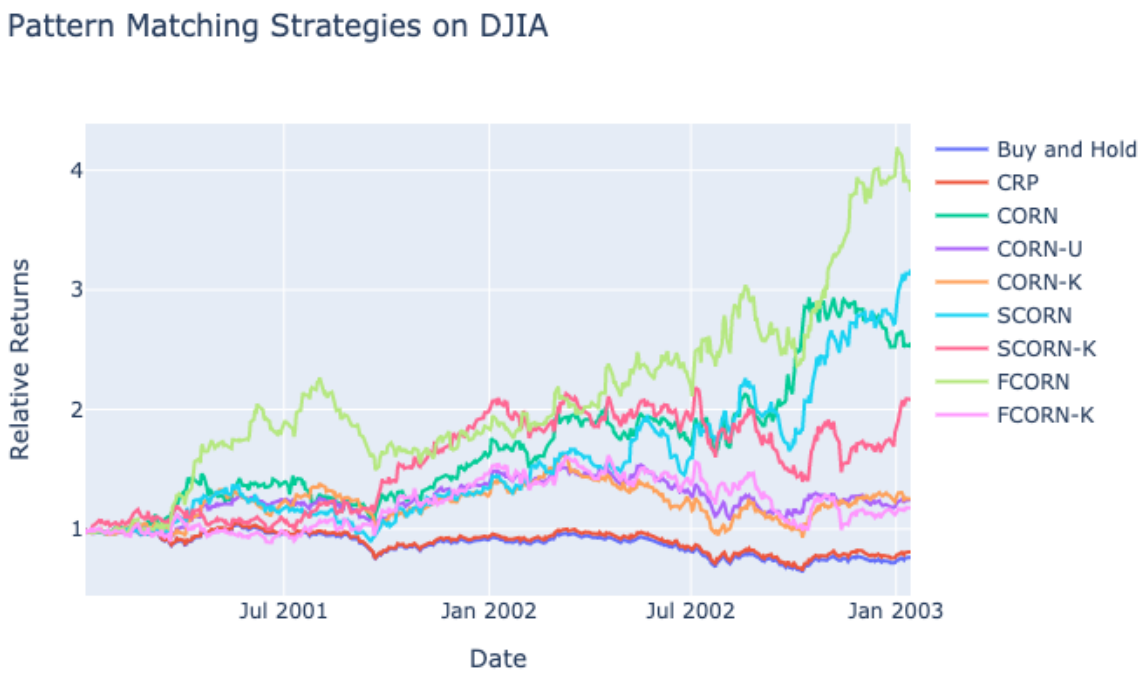
REFERENCES

1. Li, B. and Hoi, S.C.H., 2018. Online portfolio selection: principles and algorithms. Crc Press.
2. Li, B., Zhao, P., Hoi, S.C. and Gopalkrishnan, V., 2012. PAMR: Passive aggressive mean reversion strategy for portfolio selection. Machine learning, 87(2), pp.221-258.
3. Li, B., Hoi, S.C., Zhao, P. and Gopalkrishnan, V., 2013. Confidence weighted mean reversion strategy for online portfolio selection. ACM Transactions on Knowledge Discovery from Data (TKDD), 7(1), pp.1-38.
4. Li, B. and Hoi, S.C., 2012. On-line portfolio selection with moving average reversion. arXiv preprint arXiv:1206.4626.
5. Huang, D., Zhou, J., Li, B., Hoi, S.C. and Zhou, S., 2016. Robust median reversion strategy for online portfolio selection. IEEE Transactions on Knowledge and Data Engineering, 28(9), pp.2480-2493.

4.3

ONLINE PORTFOLIO SELECTION: PATTERN MATCHING

INTRODUCTION



Pattern matching locates similarly acting historical market windows and makes future predictions based on the similarity. They combine the strengths of both momentum and mean reversion by exploiting the statistical correlations of the current market window to the past.

In the following article, we will examine three variations of Correlation Driven Nonparametric Strategies strategies:

1. Correlation Driven Nonparametric Learning
2. Symmetric Correlation Driven Nonparametric Learning
3. Functional Correlation Driven Nonparametric Learning

These strategies can further be improved by using the top-K selection process from the Universal Portfolio. By using an ensemble method, the aggregated portfolio reduces variability and can effectively choose the strategy with the highest returns.

We will be introducing the applications of these popular strategies with the ease of using PortfolioLab's newest module. We hope that the newfound interests from these topics will spur more development as well as assist the research process for many others in the field.

Correlation Driven Nonparametric Learning (CORN)

Correlation Driven Nonparametric Learning strategies look at historical market sequences to identify similarly correlated periods. Existing pattern matching strategies attempt to exploit and identify the correlation between different market windows by using the Euclidean distance to measure the similarity between two market windows. However, the traditional Euclidean distance between windows does not effectively capture the linear relation. CORN utilizes the Pearson correlation coefficient instead of Euclidean distances to capture the whole market direction.

Correlation Driven Nonparametric Learning – Uniform (CORN-U)

Because the CORN strategies are dependent on the parameters, we propose a more generic one that takes an ensemble approach to reduce variability. One possible CORN ensemble is the CORN-U method.

CORN-U generates a set of experts with different window sizes and the same ρ value. After all the expert's weights are calculated, weights are evenly distributed among all experts to represent the strategy as a universal portfolio.

Correlation Driven Nonparametric Learning – K (CORN-K)

CORN-K further improves the CORN-U by generating more parameters of experts. There is more variability as different ranges of window and ρ value are considered to create more options.

The most important part of the CORN-K, however, is the capital allocation method. Unlike CORN-U, which uniformly distributes capital among all the experts, CORN-K selects the top-k best performing experts until the last period and equally allocate capital among them. This prunes the experts that have less optimal returns and puts more weight on the performing ones.

Symmetric Correlation Driven Nonparametric Learning (SCORN)

Market symmetry is a concept that the markets have mirrored price movements. Increasing price trends represent a mirror of a decreasing trend. This gives us an intuitional understanding that if the price movements are strongly negatively correlated, the optimal portfolio weights should minimize the returns or the losses from those periods as it is most likely that the optimal portfolio weights would be the inverse.

Introduced recently in a Journal of Financial Data Science paper by Yang Wang and Dong Wang in 2019, SCORN identifies positively correlated windows and negatively correlated windows.

The positively correlated windows are identified the same way as for CORN, and the negatively correlated windows are identified as any that have a correlation value below the threshold, ρ.

The strategy, therefore, maximizes the returns for periods that are considered similar and minimize the losses over periods that are considered the opposite.

Functional Correlation Driven Nonparametric Learning (FCORN)

FCORN further extends the SCORN by introducing a concept of an activation function. Applying the concept to the previous CORN algorithms, the activation function for the SCORN can be considered as a piecewise function. For any value between the positive and negative value of the threshold, we discount the importance of the period by placing a constant of 0.

Instead of completely neglecting windows with correlation with absolute value less than the threshold, FCORN introduces a sigmoid function that places a set of different weights depending on the correlation to the current market window. By replacing with such a variable, it is possible for us to place different importance on the correlated periods. One that has higher correlation will have higher weights of importance whereas ones that are less correlated will have less importance on it.

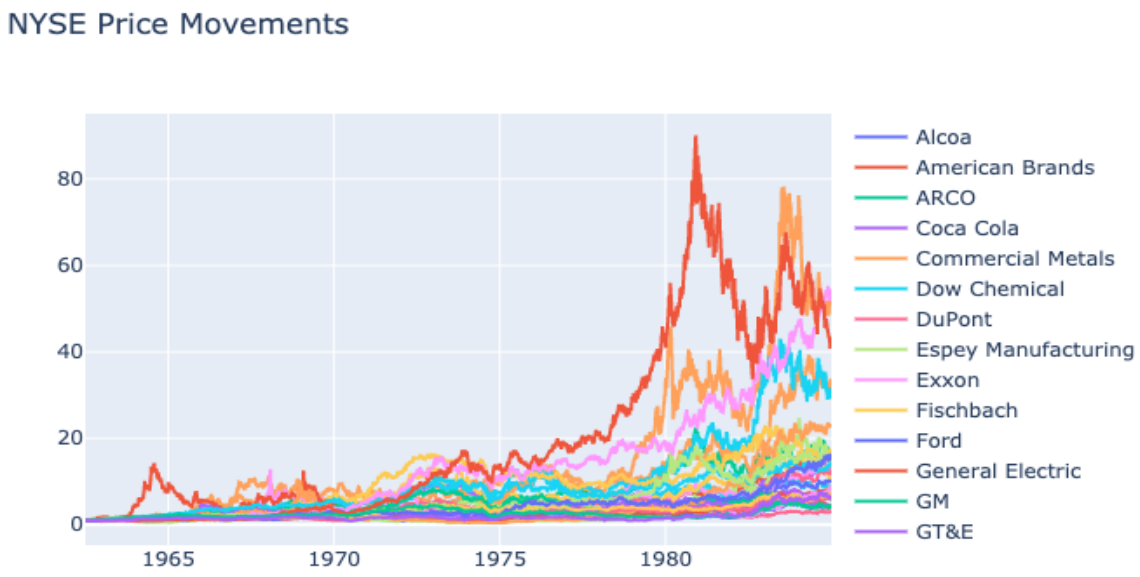
Data

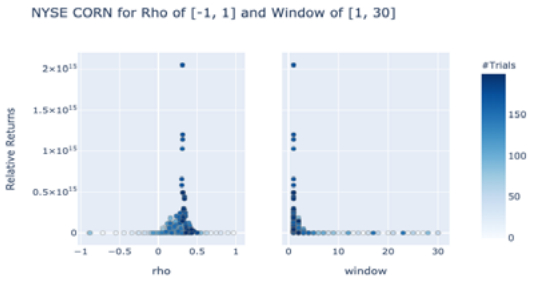
We will be using 6 different datasets, and if you would like to learn more about each dataset, exploratory analysis is available here.

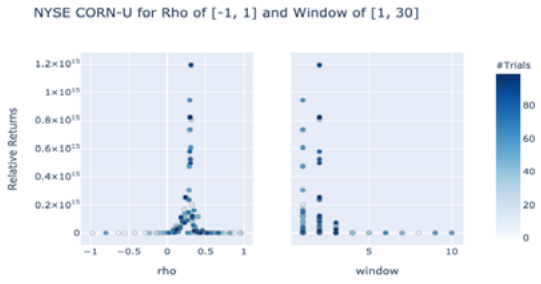
1. 36 NYSE Stocks from 1962 to 1984 by Cover
2. 30 DJIA Stocks from 2001 to 2003 by Borodin
3. 88 TSE Stocks from 1994 to 1998 by Borodin
4. 25 Largest S&P500 Stocks from 1998 to 2003 by Borodin
5. 23 MSCI Developed Market Indices from 1993 to 2020 by Alex Kwon
6. 44 Largest US Stocks from 2011 to 2020 by Alex Kwon

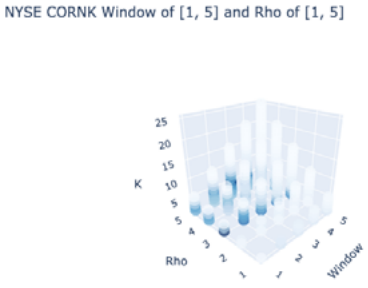
We will be using **Optuna** to examine the effects of parameters.

NYSE: 1962-1984



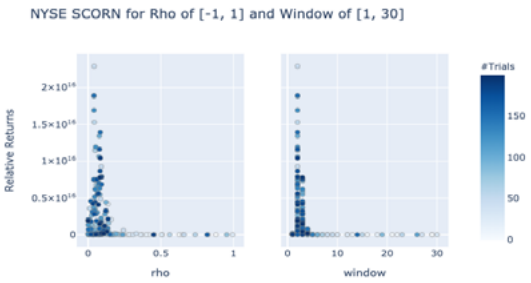


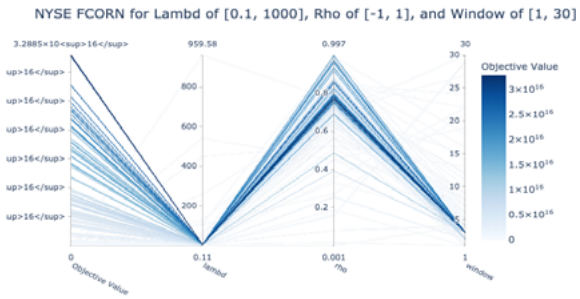


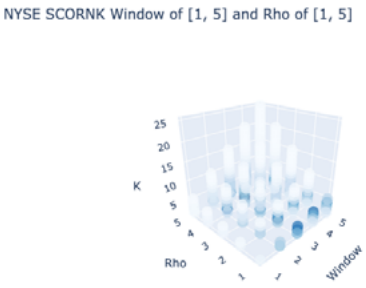


There is an overwhelming concentration of parameters near a window of 1, which indicates that these pattern-matching strategies perform the best for a single period window. The optimal value of rho was near 0.4, and interestingly, we can see that returns are primarily dependent on the window length. It is certainly possible to reach extremely high returns with the right parameters, but a rho range between 0.2 and 0.4 was adequate for the most parts.

For the CORN-K, darker colors indicate higher returns. This ensemble strategy is primarily dependant on a low k value. The highest returns are from a window of 1 and rho of 3. This is in line with our CORN analysis that NYSE had a pattern matching the trend for a single period window, and rho of 3 covered value of $\frac{1}{3}$, which is in range of the original optimal rho value.



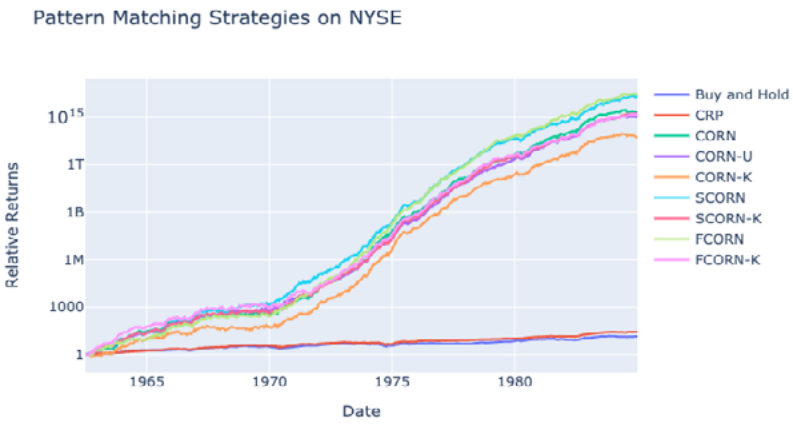




The SCORN algorithm actually provides a binary classification for the correlation threshold. Because the optimal rho is close to 0, the strategy indicates an activation function of 1 for correlated periods and -1 for inversely correlated periods. The market window is also slightly longer for SCORN as we see a price trend with a period of 2 corresponding to the highest returns.

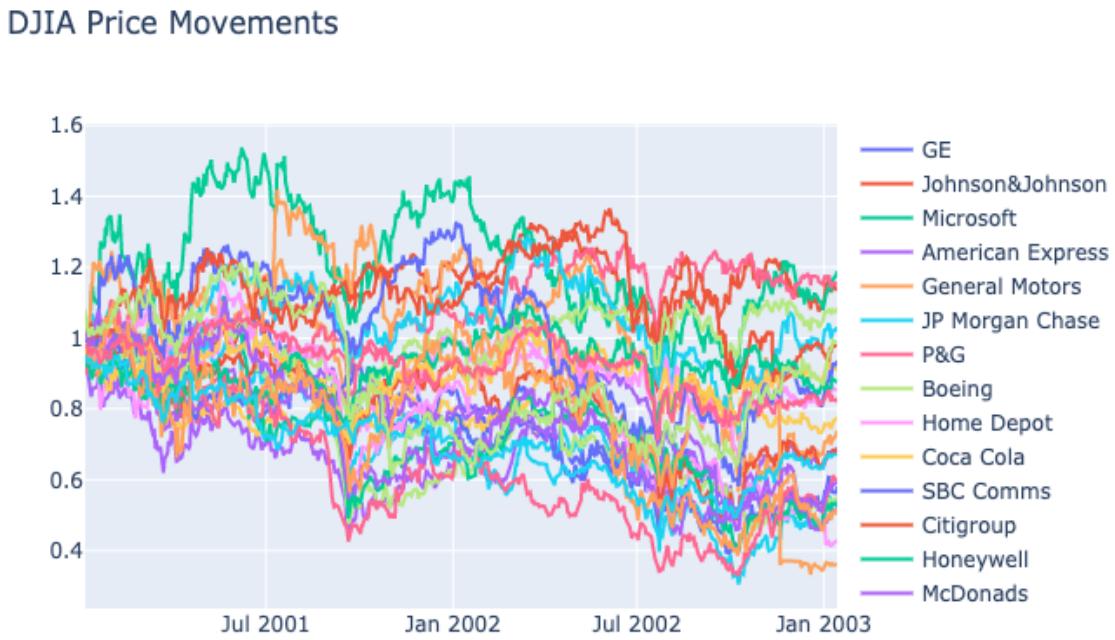
Looking at the SCORN-K graph, we notice that a longer window was more optimal for SCORN, which is in line with our SCORN analysis. Because a lower rho and slightly longer window had higher returns, we see that the window of 3 and rho of 1 has the darkest spots as rho of 1 indicates an expert with a rho correlation value of 0.

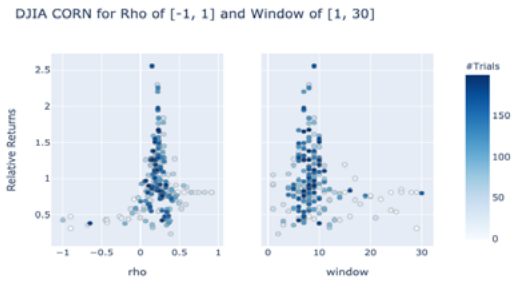
The results for FCORN are completely different from the previous SCORN or CORN. There is a much higher optimal rho value, one that is closer to 0.8. The optimal window continues to stay the same as SCORN with a value of 3. With the lambd value of 1, a rho of 0.8 allows the strategy to effectively identify the periods with high similarity but also incorporate the information from periods that are less correlated.

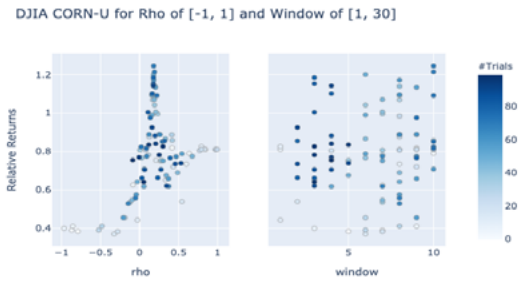


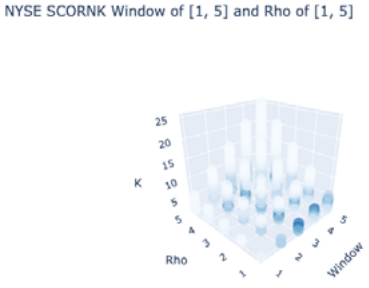
The results articleed for NYSE are close to unbelievable as returns are in the magnitude of 10^{18}. Highest returns are articleed from the resulting FCORN and SCORN algorithms, and the lowest returns were articleed by the CORN-K with still an astonishing 10^{13}. Further analysis of other datasets is needed to justify the usage of these algorithms as the NYSE may be an outlier.

DJIA: 2001-2003



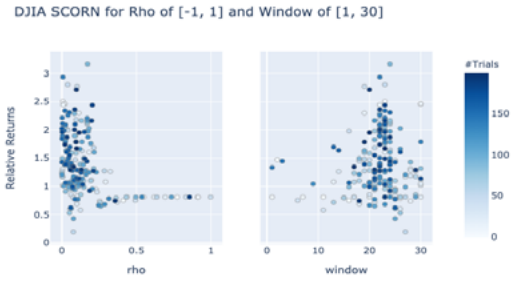


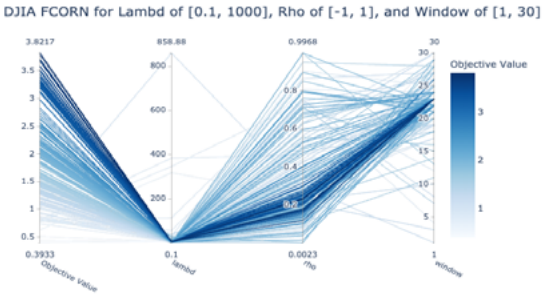


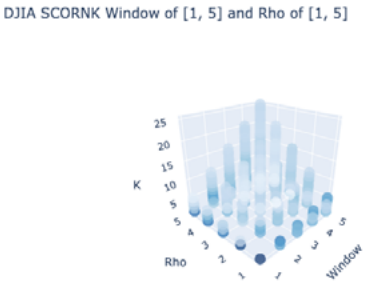


The optimal rho value for DJIA is similar to NYSE's 0.4; however, the biggest difference in the parameter value is the optimal window. We see the highest returns with a window 7, 8, or 9. This is longer than any of the other window parameters that we have seen so far.

Knowing that the optimal window for CORN is closer to 10, our current range of window of 1 to 5 for the CORN-K tests does not encompass the full potential. Different from NYSE's CORN-K results, a lower value of K does not guarantee higher returns. In this case, a high rho value of 5 was the most important parameter as the range of correlation was needed to capture all possible variations of the strategy.



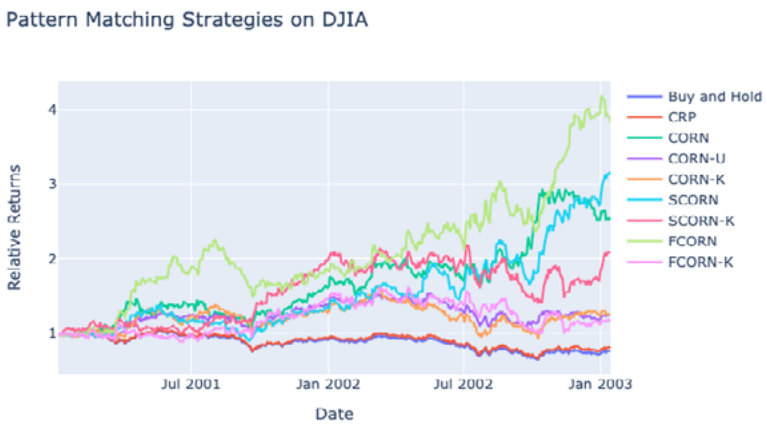




Similarly to the above CORN value, we see that the optimal window for SCORN is almost double of CORN's window value. Window closer to 22 and 23 had the highest returns. For SCORN, rho of 0 continues to have the highest returns as the binary classification of historical market windows proves to be more effective.

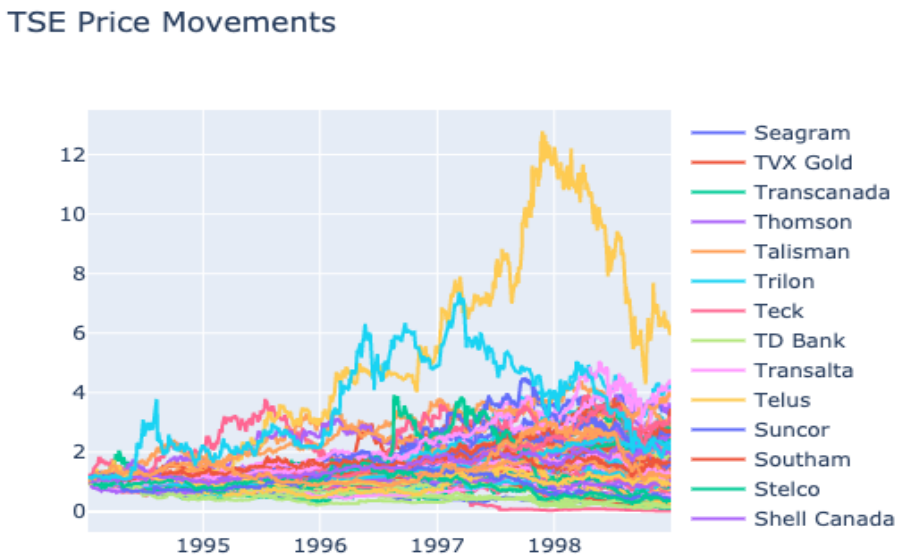
Window range of 1 to 5 is suboptimal for SCORN-K as the optimal window range for SCORN is much higher at 22. Because the SCORN continues to prefer a binary classification regarding its similarity threshold, rho of 1 is sufficient to capture the highest returns as rho of 1 indicates a threshold at 0.

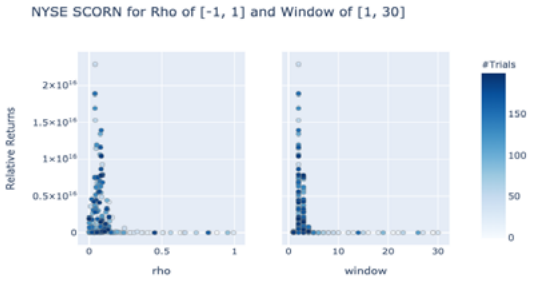
FCORN also has higher returns for window value near 22, and we actually see a different optimal rho value near 0.2. It is difficult, however, to say that there is only one best rho range as we see dark lines pass through a significant range of rho. Lambd of 1 continues to provide the highest returns for FCORN values.

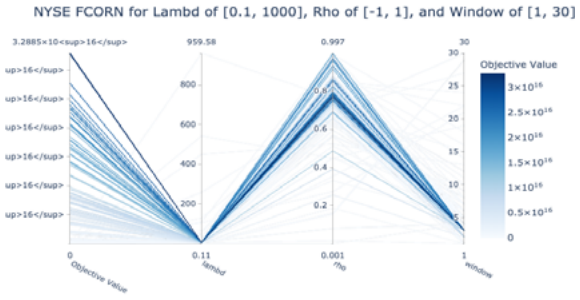


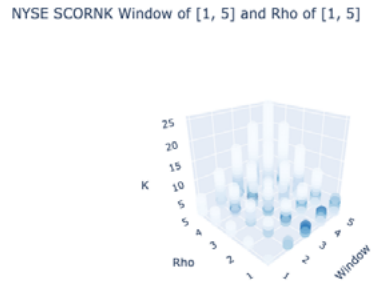
CORN performs significantly better than the benchmarks strategies for DJIA. This is a market with a continuous downtrend, and the current problem formulation only allows for a long-only portfolio. CORN is able to capture the patterns in these strategies as we see double and triple returns. SCORN-K and FCORN-K have lower returns due to the incorrect window range set by the initial testing phase. Preliminary results indicate that these CORN strategies can be applied for a market with a general bear run as well.

TSE: 1994-1998



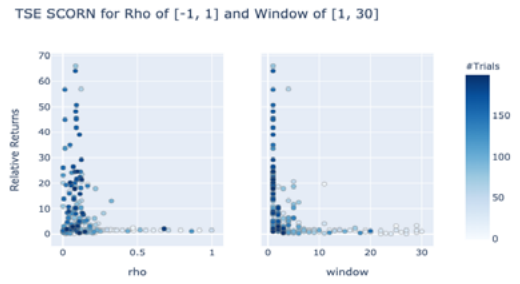


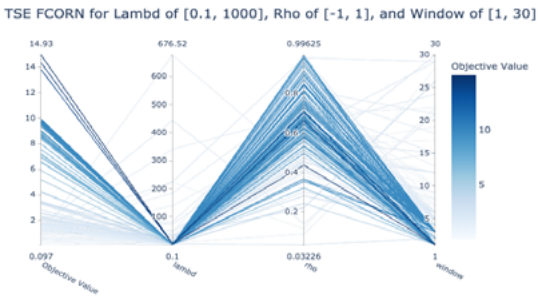


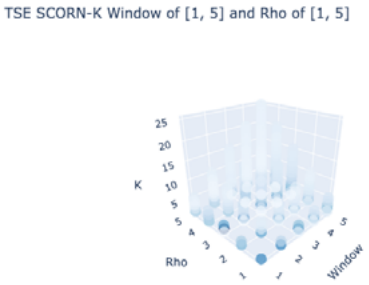


The optimal rho for TSE is lower than the previous datasets with rho of 0.1, and the distribution of the window value for CORN does not indicate a clear, explainable pattern.

CORN-K represents the same pattern as the CORN strategy with a rho of 1 being the most optimal. All values of the window return similar darkness as window ranges fail to make a significant impact on returns.



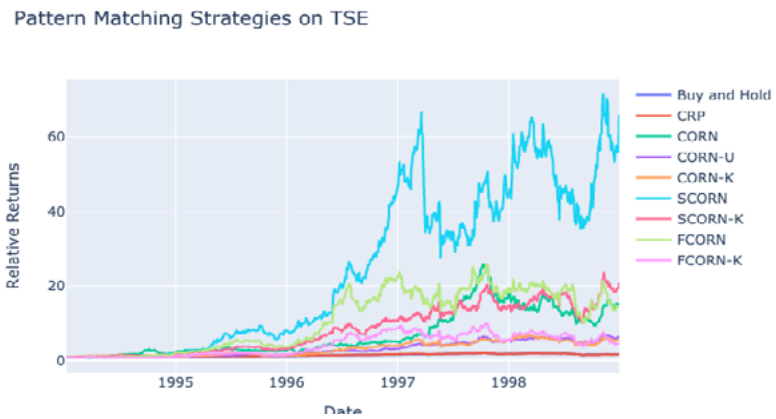




Unlike for CORN, SCORN has a clear pattern with the optimal window value of 1. There is a higher concentration of values for a window of 1, and the best rho value is slightly off from 0 at 0.1. In fact, SCORN leaves out the market periods with a correlation near 0 by having the threshold slightly above the neutral value.

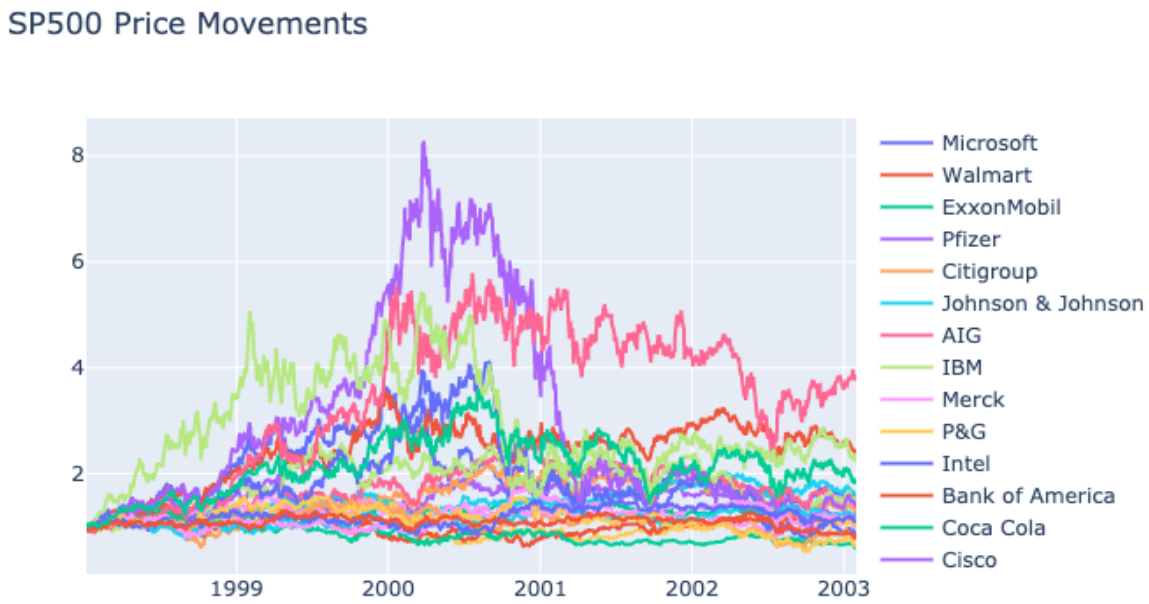
As the optimal value for the window is in the lower range at 1, the darker circles appear near a window of 1 for SCORN-K. Interestingly, the highest returns appear for a rho of 3 instead of 1. This is primarily because the optimal rho value is not exactly at 0, but a value between 0 and $\frac{1}{3}$.

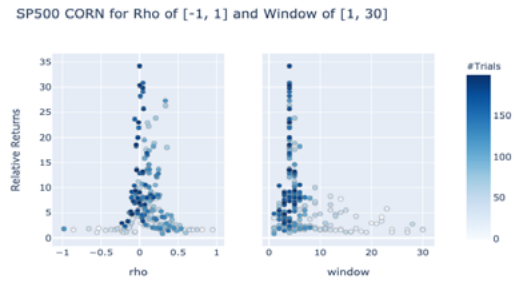
FCORN for DJIA displays similar patterns to the previous SP500. There is a wide range of optimal rho and a clear preference for low lambda and window value. If we had to pinpoint the optimal rho, a value of 0.7 would represent the median, which is in line with the previous optimal rho for FCORN.

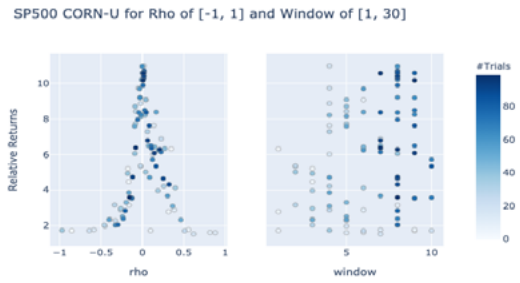


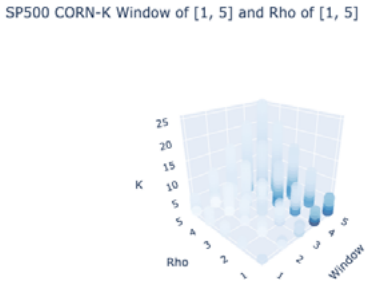
CORN performs significantly better than the benchmarks strategies for DJIA. This is a market with a continuous downtrend, and the current problem formulation only allows for a long-only portfolio. CORN is able to capture the patterns in these strategies as we see double and triple returns. SCORN-K and FCORN-K have lower returns due to the incorrect window range set by the initial testing phase. Preliminary results indicate that these CORN

SP500: 1998-2003



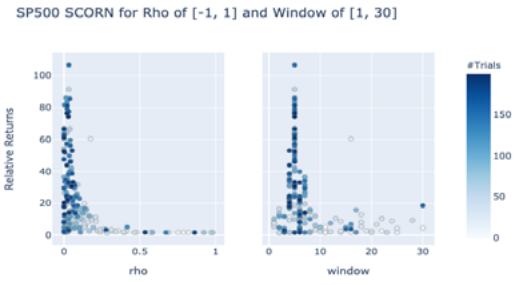


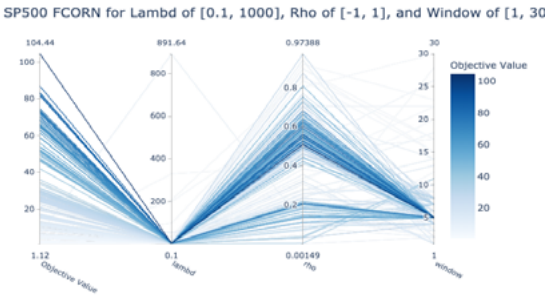


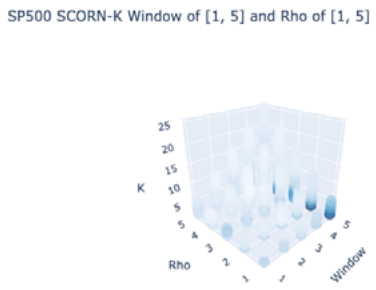


CORN strategies applied to SP500 indicate an interesting pattern as the optimal rho is right at 0. The strategy designated a time period as similar if the coefficient was non-negative. The optimal window value also has a clear result at 4 with the highest results reaching 35.

In line with the CORN applications, CORN-K displays the highest returns with an ideal rho of 1 and a window of 4.



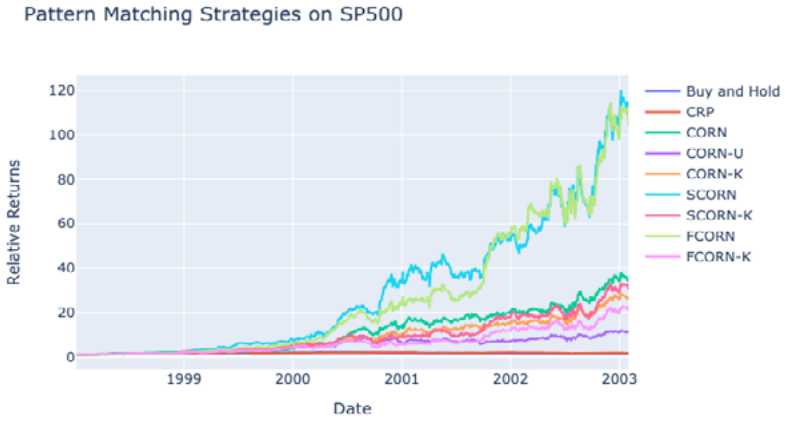




As with other datasets, ideal rho is close to 0, and the best window value is at 5. Results for SCORN in fact reach close to 100, triple the results of CORN.

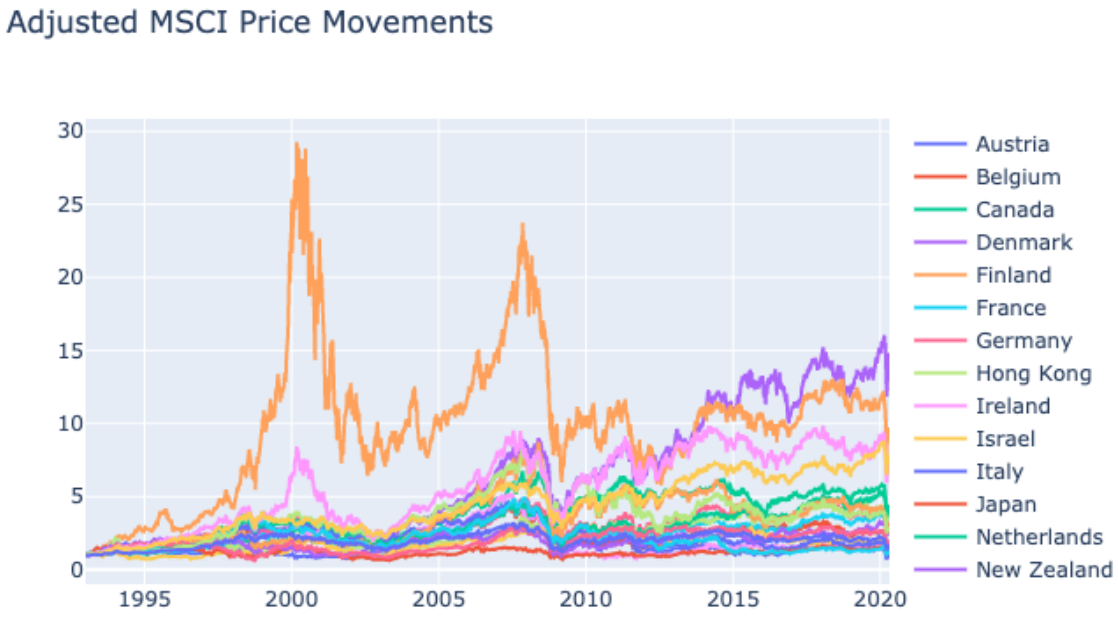
Any rho value that has a window of 5 displayed significantly higher returns for SCORN-K.

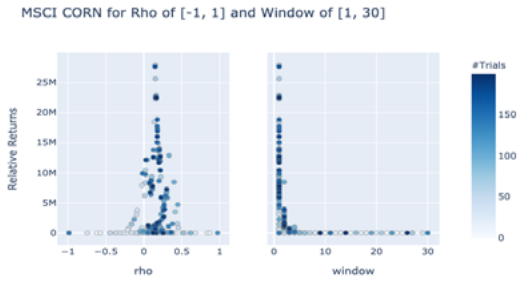
FCORN-K has the same optimal window value as SCORN-K at 5. Lambd continues to have higher returns at a lower value. The ranges for rho, however, are more divided as we see a clear segment at 0.2 and 0.6. It is difficult to explain how a rho of 0.2 would produce higher returns, but the value of 0.6 is in line with the previous results.

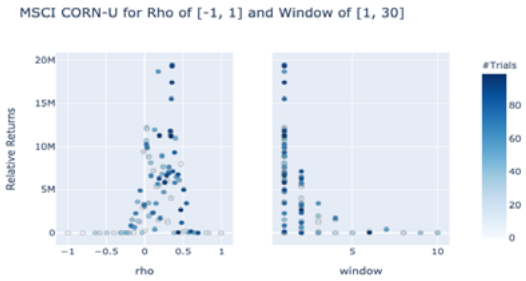


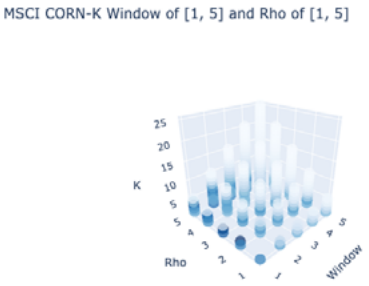
The tuned values for SCORN and FCORN reach returns of 120, and the more realistic application of SCORN-K and FCORN-K have returns that are close to 30. In an ideal scenario, the correct parameters should be identified to have high returns, but even if we do not have the exact values, the relatively high returns of SCORN-K and FCORN-K indicate a possibility of applications to a real trading environment.

MSCI: 1993-2020



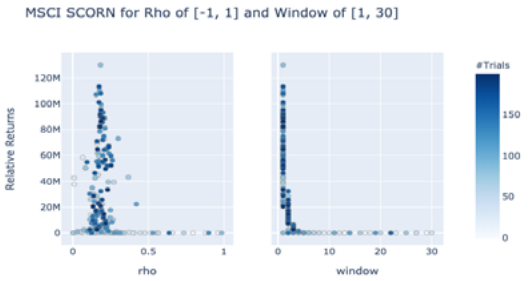


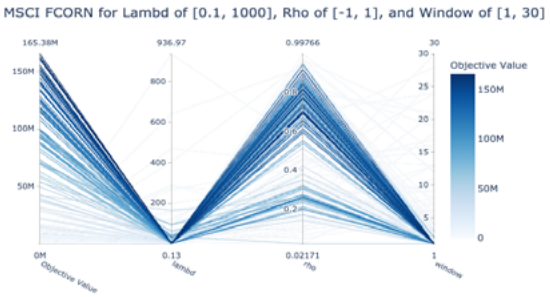


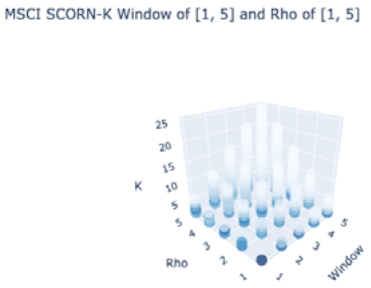


The returns for MSCI are also extremely high with applications with CORN. With a rho of 0.2, we see results that reach well above 10^8; however, looking at the graph for CORN closely, we see that there is a slight segmentation between the 9 and 7 million returns. This might indicate that the results are extremely sensitive to your parameter choice.

With a rho value of 3 being the highest for CORN-K, we see a trend with the optimal selection of rho. For datasets with optimal rho value near 0.2. Rho of 3 represents the highest returns as the strategy can alternate between a rho of 0 and $\frac{1}{3}$.

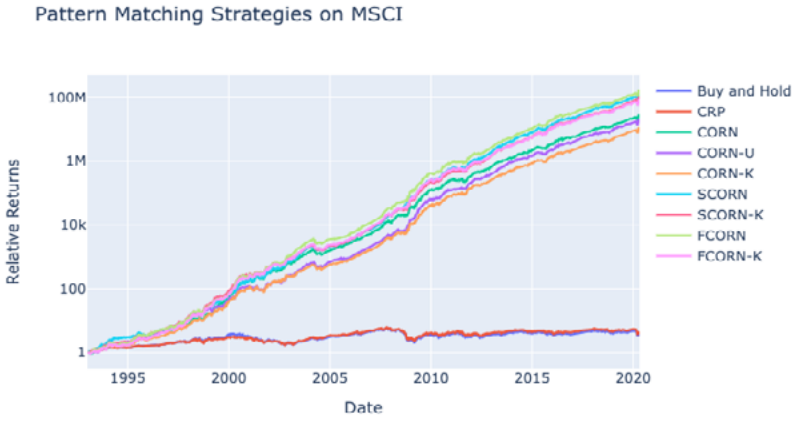






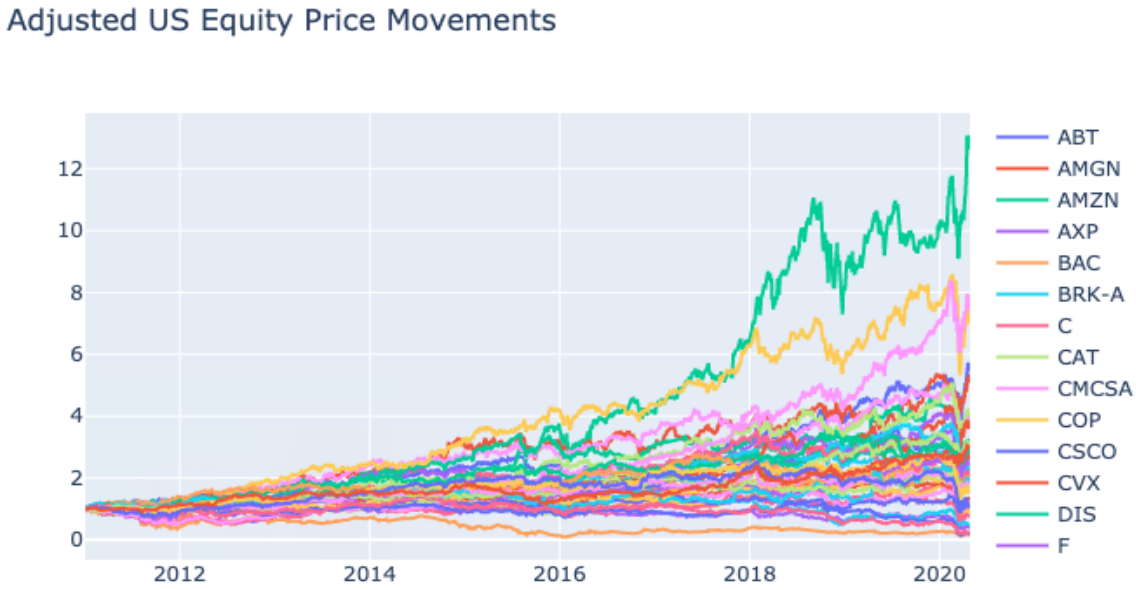
The SCORN results for MSCI are significantly different from previous sets. Rho of 0 is no longer the optimal as we see the value shifted to 0.2. Moreover, we see that the optimal window is 1, but if you take a closer look at the bottom values of window 1, there is a huge gap between 0 and 20M. A more detailed analysis of this phenomenon is required, but it is likely that for a window value of 1 a wider range of rho is accepted to produce similar returns.

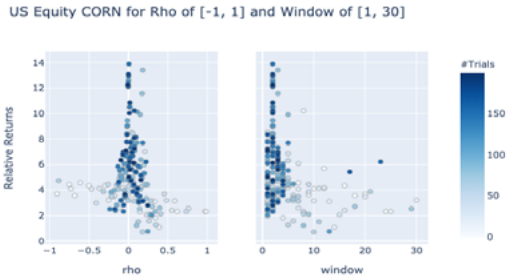
In general, SCORN-K graph shows that as long as k is 1 most combinations have high returns. This most likely happens because the ideal window value is strictly 1, and therefore any combination of portfolios that have a parameter of 1 will have high returns.

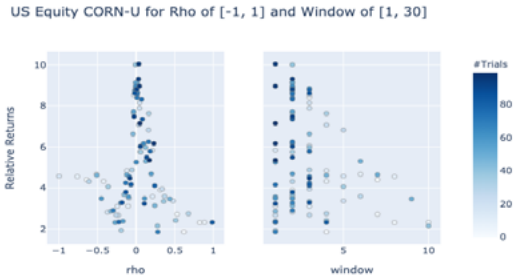


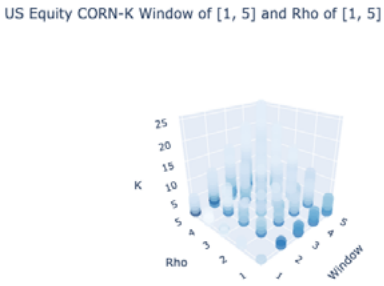
Other than the overfitting SCORN and FCORN graphs, the other k-aggregated algorithms perform significantly better than the benchmarks. FCORN-K in fact is able to almost match the other two with SCORN-K and CORN-K closely following behind.

US Equity: 2011-2020



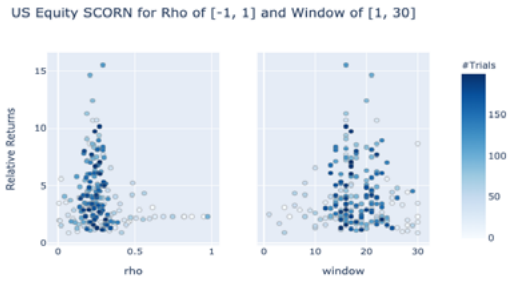


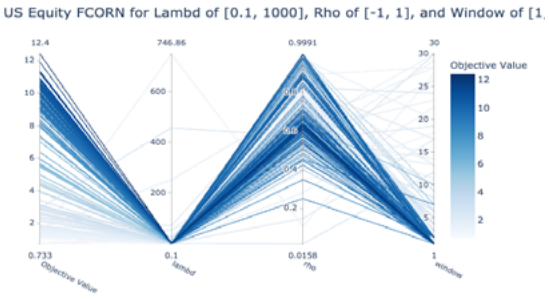


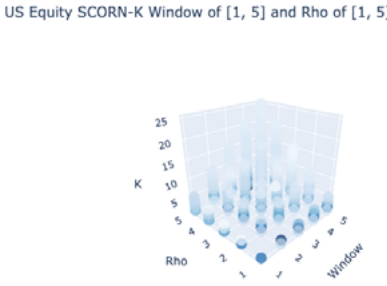


The CORN data with US Equity provides the most interesting results as this is directly applicable to a more modern timeline. Highest returns typically congregated around rho of 1 and a window of 2. In a way, the rho of 0 implies that any past history window with a non-negative correlation should be incorporated in the calculation, and a window of 2 indicates that a price trend should be looked at in a 2-day rolling window for correlation calculations.

CORN-K provides a similar analysis as CORN with a window of 2 and rho of 1 having the highest returns. Based on the darkness of the circles for all rho values of 1, we can also notice that most parameters with k values of 1 had significant returns.

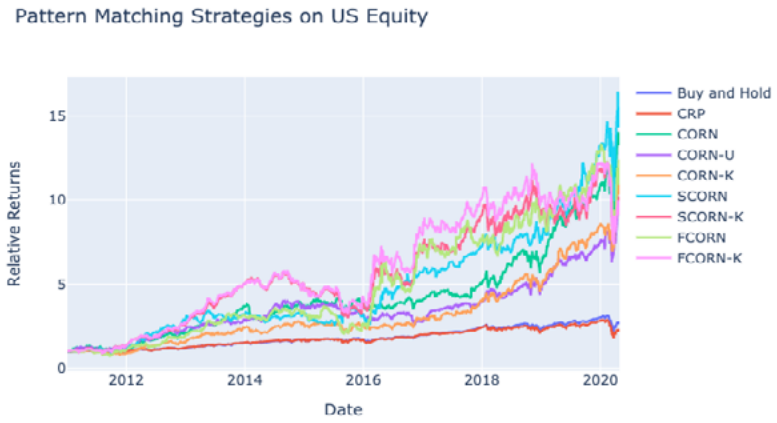






There is less of a distinct pattern for SCORN. Rho of 0.2 is similar to MSCI but different from the other datasets. The window range is also much longer with a value closer to 16 being optimal. From the initial look, a more rigorous analysis should be required to determine the parameters and apply this strategy in a real trading environment.

The results for SCORN-K unfortunately do not mean much as we examined from SCORN that the optimal window is around 16, which is not covered in this range of values. The most important note from this graph is that k must be either 1 or 2 to have the highest returns for any ensemble of portfolios.



This US Equity dataset includes the recent downturn from COVID-19. Particularly, SCORN has its returns drop from 14 to 9 initially but regains its value back to 16. This is actually higher than the returns before the crash. Exact portfolio weights should be examined to determine the logic and pathway of the portfolio selection method, but these results indicate a possibility of applying pattern-matching strategies in a real trading environment

CONCLUSIONS

Pattern matching strategies can be formulated in numerous ways with different thresholds and parameters. The original CORN development by Li, Hoi, and Gopalkrishnan was further improved with studies from Yang Wang and Dong Wang. This notebook covered a wide range of CORN strategies employed in PortfolioLab's newest Online Portfolio Selection module, and readers will be able to replicate results using the simple methods of the new module.

REFERENCES

1. Li, B. and Hoi, S.C.H., 2018. Online portfolio selection: principles and algorithms. Crc Press.
2. Li, B., Hoi, S.C. & Gopalkrishnan, V., 2011. CORN: Correlation-Driven Nonparametric Learning Approach for Portfolio Selection. ACM Transactions on Intelligent Systems and Technology, 2(3), pp.1–29.
3. Wang, Y. & Wang, D., 2019. Market Symmetry and Its Application to Pattern-Matching-Based Portfolio Selection. The Journal of Financial Data Science.

5.0

NETWORKS WITH MINIMUM SPANNING TREES

Network analysis can provide interesting insights into the dynamics of the market, and the continually changing behaviour. A Minimum Spanning Tree (MST) is a useful method of analyzing complex networks, for aspects such as risk management, portfolio design, and trading strategies. For example, Onnela et al. (2003) notices that the optimal Markowitz portfolio is found at the outskirts of the tree [4]. Analysing the Tree structure, as a representation of the market, can give us an idea about the stability and state of the market.

A Minimum Spanning Tree (MST) is a graph consisting of the fewest number of edges needed for all nodes to be connected by some path – where the combination of edge weights sum to the smallest total possible.

MST strongly shrinks during a stock crisis [2]. Properties such as skewness are positive during times of market crises (such as 1987, early 2000's and 2008) and skewness and kurtosis have stabilised after 2000's [1]. Analysing the Tree structure, as a representation of the market, can give us an idea about the stability and state of the market and predict how volatility shocks will propagate through a network.

MST VISUALISATIONS

Visualisations are created using Plotly's Dash, which creates a mini Flask server to serve the frontend components. This section gives examples of how to create and run the MST visualisations, and examples of optional functionality such as adding colours to represent industry and node size to represent, for example market cap.

Example Code Snippet

Given a dataframe of log returns, you can call generate_mst_server to generate a MST server.

```
import pandas as pd
import numpy as np


# Import Visualisations method
from mlfinlab.networks.visualisations import generate_mst_server


# Import Data
input_dataframe = pd.read_csv('../Sample-Data/stock_prices.csv')


# Calculate log returns
input_dataframe = np.log(input_dataframe).diff()


# Remove first row of NaNs
log_return_df = input_dataframe.iloc[1:]


# Create and run the server
server = generate_mst_server(log_return_df)
server.run_server()
```

To create and run the server in the Jupyter notebook, pass in the parameter jupyter=True and call run_server with the parameter mode as 'inline', 'external' or 'jupyterlab'.

```
server = generate_mst_server(log_return_df, jupyter=True)
server.run_server(mode='inline')
```

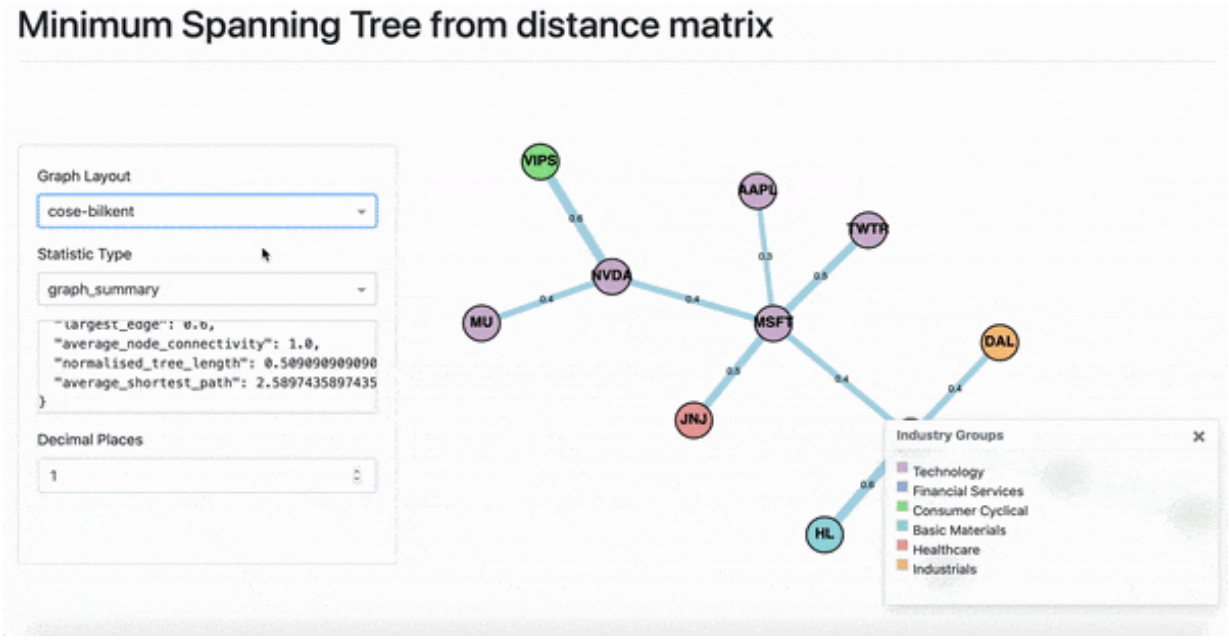
The inline mode allows you to interact with the Dash interface within a cell of Jupyter notebook.

MST Layouts

On the side panel, the top dropdown menu contains three different layouts designed to layout the MST. These are namely cose-bilkent, cola and spread, selected as optimal layouts to view the MST. Cola and spread are physics simulation based, and cose-bilkent is a spring embedder layout. These are external layouts, and more information can be found on the Dash documentation.

Adding Industry Groups

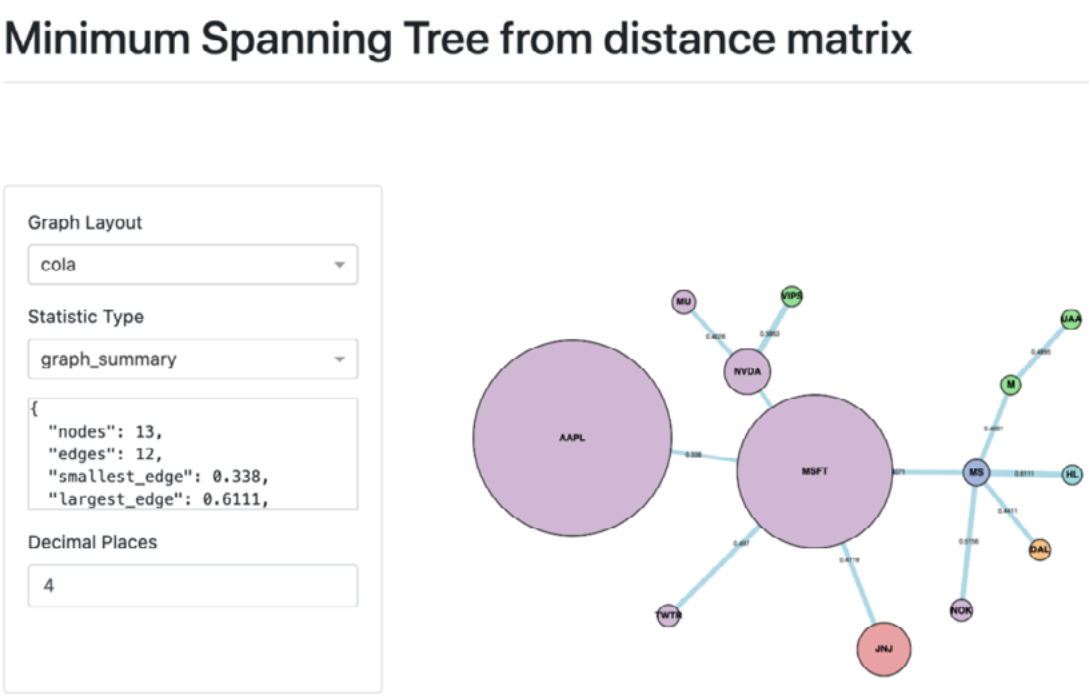
According to Marti et al., the "MST provides a taxonomy which is well compatible with the sector classification provided by an outside institution" [2] An example of 13 stocks from August 15th 2020 until August 10th 2020 for daily closing prices is shown below.



The colours correspond to the Industry Groups legend. However, you can add different categories such as instrument type. The legend generates the colours and the matching categories supplied. Colours must be set by calling `set_node_groups` on class `MST` before the `MST` object is passed to the `DashGraph` class.

Adding Market Cap

To add the market cap, we call `set_node_size()` in the `MST` class before passing it to the `Dash Graph`. The market cap is a list corresponding to the index of the original dataframe. The market cap supplied below is measured in billions, at the time of writing (24th August 2020).

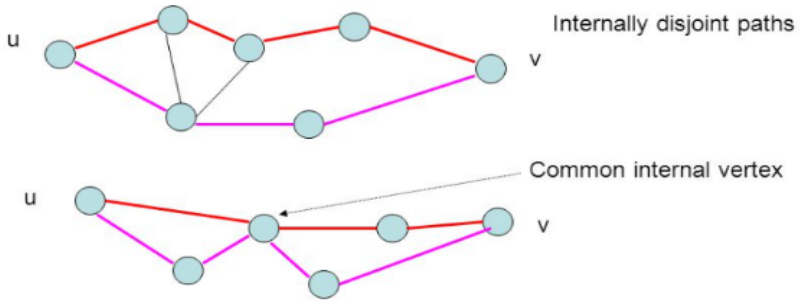


MST Statistics

Analysing the MST for topological features such as centrality, node and edge strength and normalised tree length, can be used to indicate the state of the market and understand changes over time. For example, properties such as a rising normalized tree length over the last 30 years [1], indicates growing correlation between stock returns. The statistics on the left panel of the UI, can be used for analysing the properties of the MST.

Average Node Connectivity

Average Node Connectivity is defined as “the maximum number of internally disjoint paths connecting” given a pair of vertices for all pairs in the graph [3]. Two paths u, v are internally disjoint if they don’t share a common internal vertex.



In the example above, the value is 1, since for an MST there are no loops. However, Average Node Connectivity is useful for understanding the properties of PMFG or other types of graphs.

Normalised Tree Length

Normalised Tree Length is positively correlated with stock market returns [1] and negatively correlated with tail risk and return volatility [1]. Thus, Normalised Tree Length is perhaps one of the most important statistics to analyse which financial instruments increase market returns but decrease tail risk and return volatility.

$$L(t) = \frac{1}{N-1} \sum_{d_t^{i,j} \in T^t} d_t^{i,j}$$

Normalised Tree length at time t, is the sum of all edges in the distance matrix divided by the number of edges in the MST. It is defined as follows [4] :

Where L(t) is the Normalised Tree length, t is the time at which the tree is constructed, N-1 is the number of edges in the MST and D is the distance matrix.

Average Shortest Path

The Shortest Path of two nodes i and j is defined as the path which gives the minimum possible sum of the weights of the edges. The average shortest path is the average of the shortest paths for every combination of nodes in the graph. This gives an indication of how closely connected the nodes are in the graph overall.

Average Degree Connectivity

The average degree connectivity is the "average nearest neighbor degree of nodes with degree k". The algorithm is essentially the same as the k-nearest neighbours algorithm, and returns a dictionary of degree k and value of the average connectivity. It is defined as [5]:

$$k_{nn,i}^{w} = \frac{1}{s_i} \sum_{j \in N(i)} w_{i,j} k_j$$

Where si is the weighted degree of node i, wij is the weight of the edge that links i and j, and N(i) are the neighbors of node i.

Average Neighbour Degree

The Average Neighbour Degree calculates the average number of neighbours of each node, weighted by the edge weight value.

Betweenness Centrality

Betweenness Centrality measures the "fraction of the shortest paths passing through a vertex" [6]. In a social network, a node with a high betweenness centrality value suggests the node is playing a "bridge spanning" role [7].

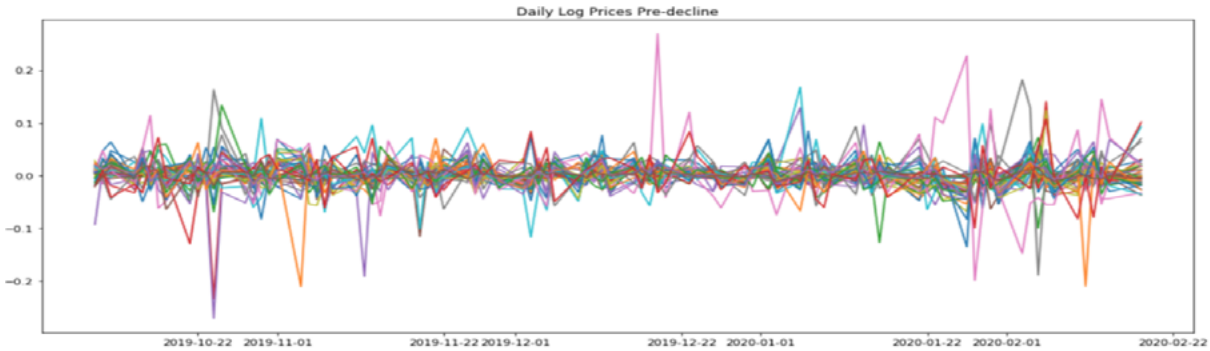
COVID EFFECT ON MST

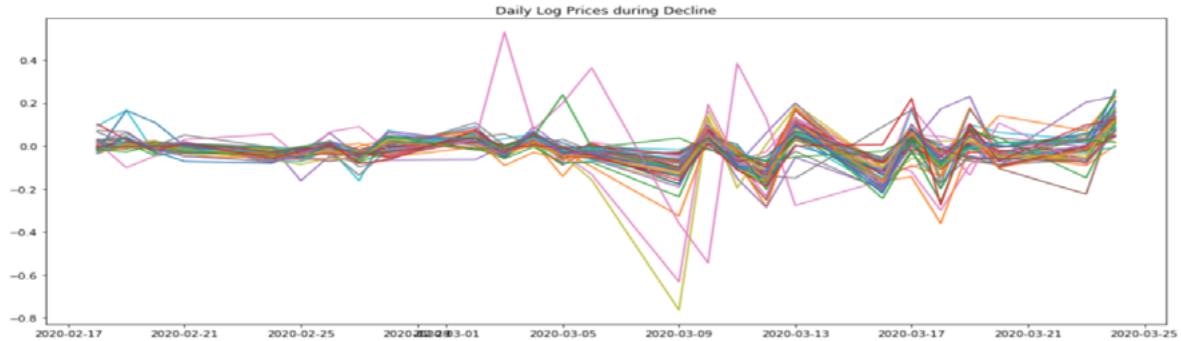
Now we can explore the effect of COVID-19 on the MST of 48 stocks. The stock data is based on daily closing prices and retrieved from Yahoo Finance (**https://finance.yahoo.com**).

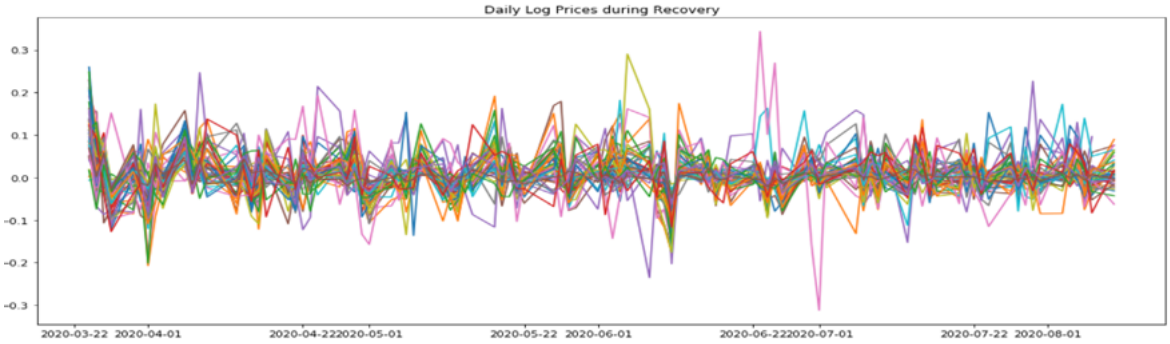
The data is split into 3 periods:

1. 9th October 2019 until 18th February 2020
2. 18th February until the 24th of March 2020
3. 24th March 2020 until 13th August 2020

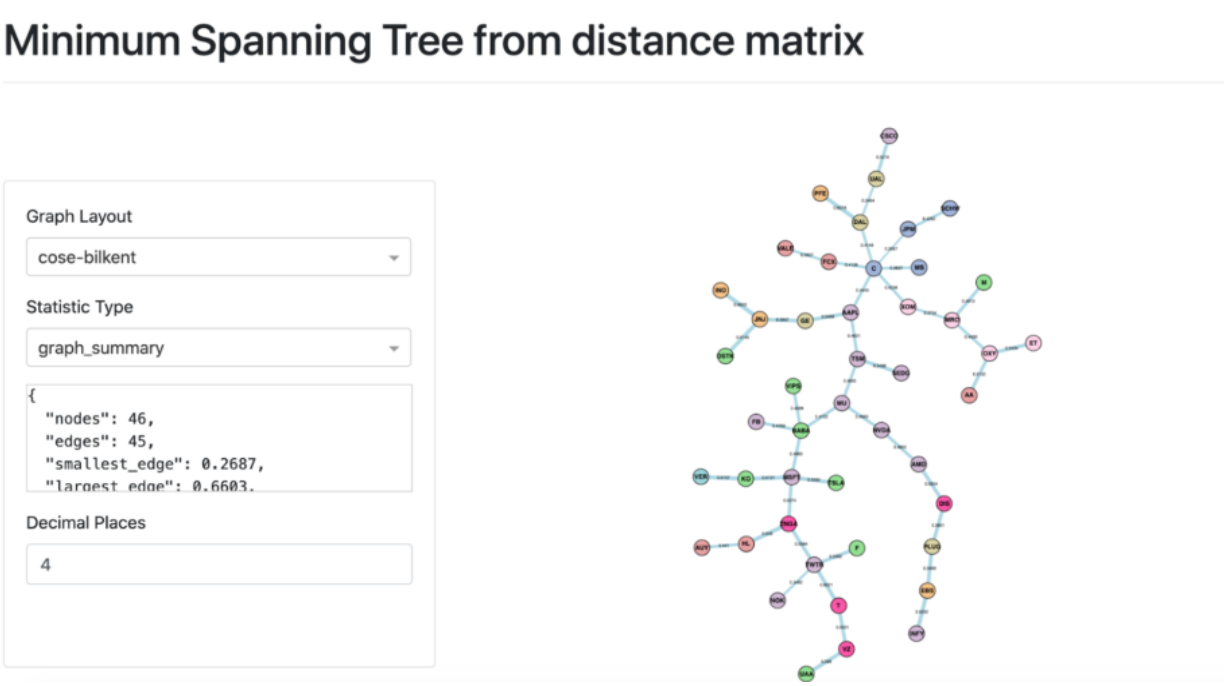
The following graphs show the Daily Log Prices during the pre-decline, decline and article-decline recovery periods.



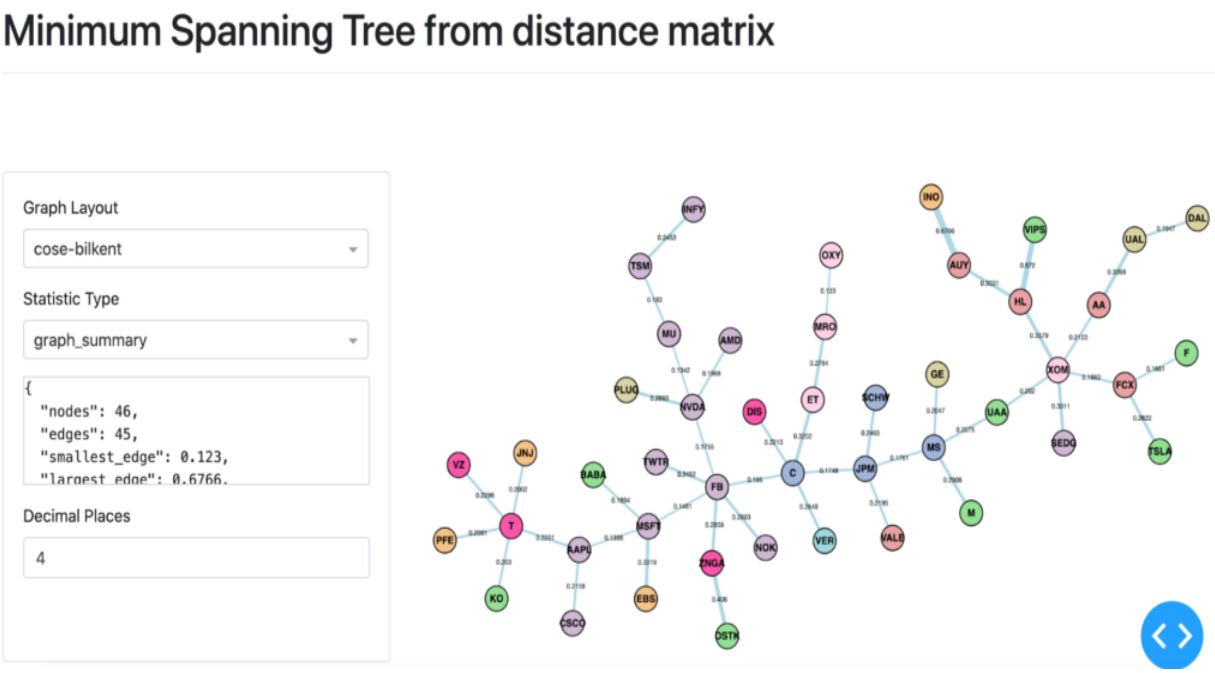




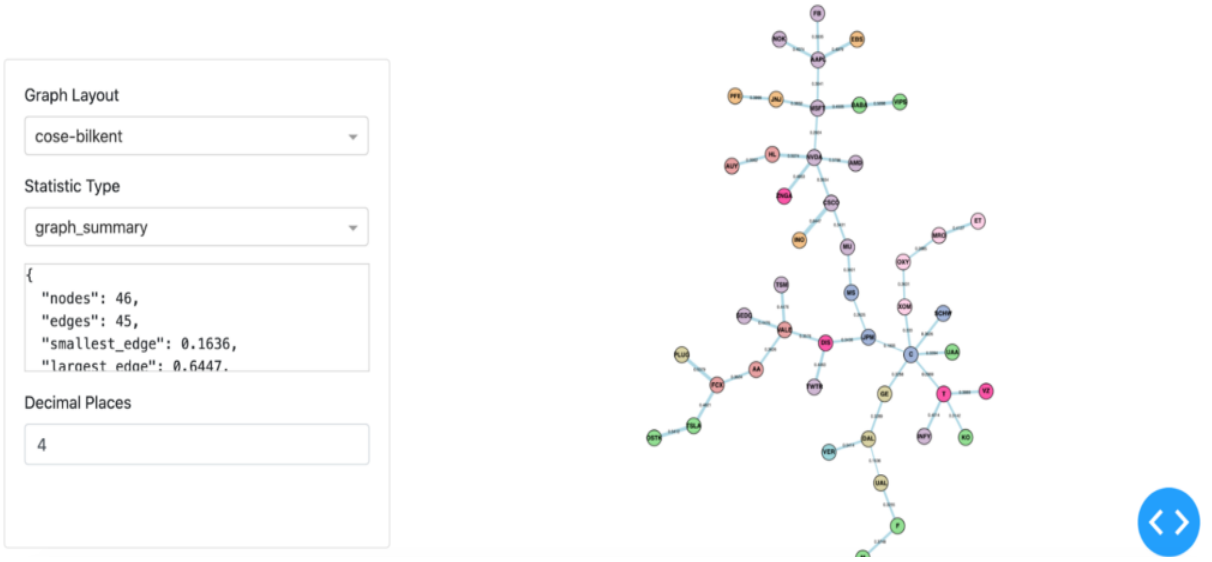
Analysis based on Distance Matrix



Pre-Decline MST based on distance matrix.



The above shows the MST based on the distance matrix during the decline period. Facebook (FB) changes from degree 1 in the pre-decline period, to degree 6 during the decline, and back to degree 1 during the article-decline recovery period. JPMorgan Chase (JPM) and MS (Morgan Stanley) become a part of the backbone of the MST during the decline period. Both JPM and MS become a connecting node in the article-decline recovery period, acting as bridges between the major branches in the MST.



The general trend throughout the periods is that the distances between the nodes in the same category decreased during the Decline period, and did not recover to the pre-decline levels.

This is the case for example, for the financial sector stocks. Another example is the sector group Energy (in a light pink colour). A similar trend occurs amongst Technology stocks (shown in light purple).

Analysis of Statistics

Statistic Type	Pre-decline	Decline	Post-decline
Normalised Tree Lenght	0.504	0.250	0.380
Average Shortest Path	6.224	5.533	6.293

Degree	Pre-decline	Decline	Post-decline
1	3.05	3.885	3.545
2	2.536	2.722	2.654
3	2.542	2.778	2.467
4	2.167	2.5	2.25

Both Normalised Tree Length and Average Shortest Path decrease during the Decline period, suggesting a shrinkage of the overall MST. The second table shows the Average Degree Connectivity (rounded to 3 decimal places). Degrees 1, 2, 3, 4 and 6 show a change during decline but returns closer to pre-decline levels. The central node with 6 degrees changes from C (Citigroup) to FB (Facebook) back to C (Citigroup).

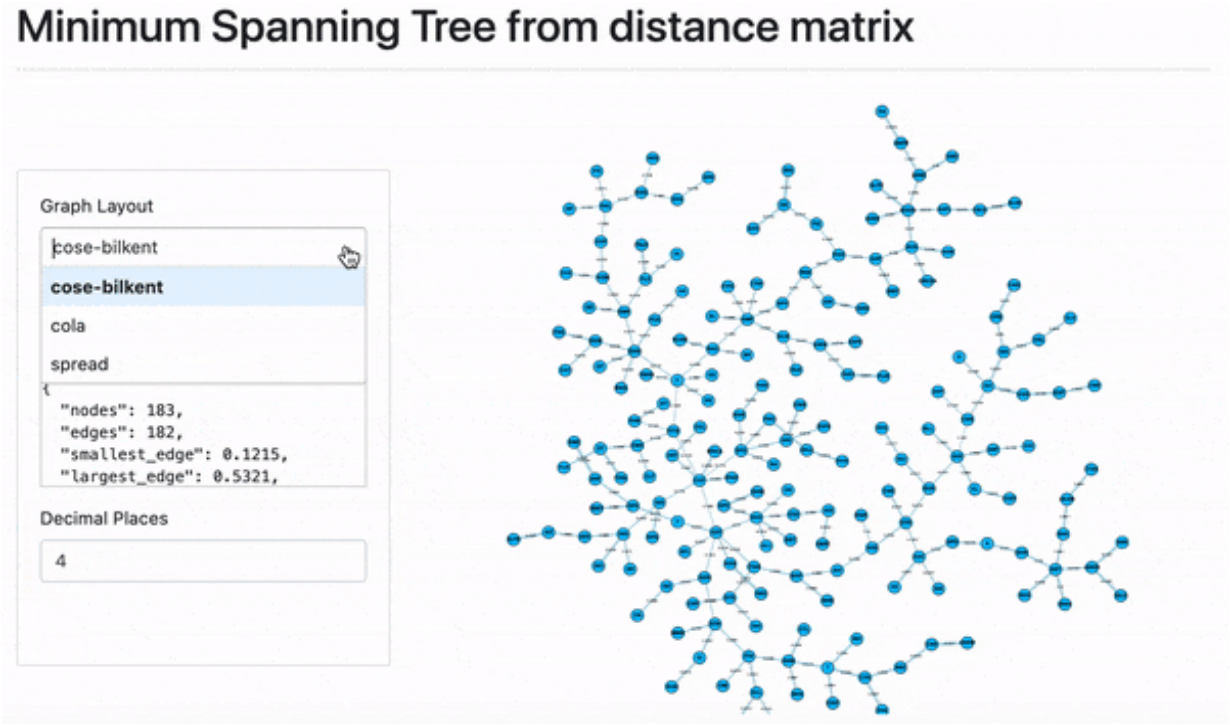
It is clear that the Covid-19 related decline has shrunk the MST. The market (based on the above stocks) has become increasingly correlated article-decline compared to pre-decline, despite much of the prices of stocks having recovered to pre-decline levels. The article-decline recovery market has also become more volatile (as shown by the log return figure for this period). Investors' preference for "safer" assets such as technology stocks (perceived as generally more pandemic resilient), was not obvious by looking at the MST.

PRIM'S VS. KRUSKAL'S IN LARGER DATASET

According to Huang et al. [8], Prim's has a time complexity of O(n) and Kruskal's has a time complexity of O(n2). For more than 100 stocks in a dataset to build an MST, Prim's was recommended for time efficiency.

To test the difference between Prim's and Kruskal's, one year's worth of data for 183 stocks for daily closing prices was used to generate the MST.

Due to the nature of the respective algorithms, Prim's is recommended for sample sizes larger than 100, and Kruskal's for small sample sizes or when space complexity is more important (Huang et al. 2009).



Running the MST constructor repeatedly showed that Prim's was on average around 40ms quicker than Kruskal's per loop. For 20 consecutive runs, Kruskal's took 11.9 s with 738 ms per loop and Prim's took 10.9 s with 1.05 s per loop. This was consistent with the hypothesis in the work of Huang et al. [8], where Prim's outperforms Kruskal's in Time Complexity when the sample sizes were larger than 100 nodes.

LIMITATIONS

The main limitation of MST is the instability which stems from the input data (correlation based matrices). Clusters from the MST are known to be unstable and large changes can occur even with small changes in the data input. A variety of different input data forms have been proposed to address this, as well as other network methods such as PMFG (Planar Maximally Filtered Graph), DST (based on dynamic correlation), or ALMST (Average Linkage MST). However, MST and its features continue to serve as a useful tool to analyse complex networks.

CONCLUSION

The article reviewed how to generate MST visualisations using DashGraph and MST classes. For example, how to add colours based on industry group and add node sizes to indicate market cap. The MST Dash interface allows for interaction with the graphs, as well as different layouts (cose-bilkent, cola and spread) to better view the MST. The article reviewed the statistics in the left panel, with an emphasis on the Normalised Tree Length statistic, which is correlated with market returns and negatively correlated with tail risk and returns volatility [1]. The statistics can be used to analyse the MST and its properties across different market states.

An analysis of MST's during the Covid-related market decline and recovery was done, by splitting the period into pre-decline, decline and article-decline recovery periods. During the decline, the MST shrunk, as is expected during crises [2]. Prim's was faster than Kruskal's for a larger dataset of 183 stocks, as expected since the dataset exceeded a 100 nodes [8].

REFERENCES

1. Huang, Wei-Qiang, et al. "Dynamic asset trees in the US stock market: Structure variation and market phenomena." Chaos, Solitons & Fractals 94 (2017): 44-53..
2. Marti, Gautier, et al. "A review of two decades of correlations, hierarchies, networks and clustering in financial markets." arXiv preprint arXiv:1703.00485 (2017).
3. Beineke, Lowell W., Ortrud R. Oellermann, and Raymond E. Pippert. "The average connectivity of a graph." Discrete mathematics 252.1-3 (2002): 31-45.
4. Onnela, J-P., et al. "Dynamics of market correlations: Taxonomy and portfolio analysis." Physical Review E 68.5 (2003): 056110.
5. Barrat, Alain, et al. "The architecture of complex weighted networks." Proceedings of the national academy of sciences 101.11 (2004): 3747-3752.
6. Jia, Yuntao, et al. "Edge v. node parallelism for graph centrality metrics." GPU Computing Gems Jade Edition. Morgan Kaufmann, 2012. 15-28.
7. Hansen, Derek, Ben Shneiderman, and Marc A. Smith. Analyzing social media networks with NodeXL: Insights from a connected world. Morgan Kaufmann, 2010.
8. Huang, Feixue, Pengfei Gao, and Yu Wang. "Comparison of Prim and Kruskal on Shanghai and Shenzhen 300 Index hierarchical structure tree." 2009 International Conference on Web Information Systems and Mining. IEEE, 2009.

6.0

THE HIERARCHICAL RISK PARITY ALGORITHM: AN INTRODUCTION

Portfolio Optimization has always been a hot topic of research in financial modelling and rightly so – a lot of people and companies want to create and manage an optimal portfolio which gives them good returns. There is an abundance of mathematical literature dealing with this topic such as the classical Markowitz mean variance Optimization, Black-Litterman models and many more. Specifically, Harry Markowitz developed a special algorithm called the Critical Line Algorithm (CLA) for this purpose which proved to be one of the many algorithms which could be used in practical settings. However, the algorithm comes with its own set of caveats –

1. The CLA algorithm involves the inversion of a positive-definite covariance matrix and this makes it unstable to the volatility of the stock-market – the inverse of the covariance matrix can change significantly for small changes in market correlations.

2. A second drawback of CLA (and in-fact many methods in the previous literature) is that it is dependent on the estimation of stock-returns. In practical scenario, stock returns are very hard to estimate with sufficient accuracy and this makes it hard to quantify the results of this algorithm.

3. CLA considers the correlations between the returns of all the assets in a portfolio and this leads to a very large correlation dependency connected graph where each asset is connected to all the other assets. Hence, when the algorithm calculates the inverse of the correlations, it is a computationally expensive task as all the edges of a connected graph are considered.

4. Finally, not all the assets in a portfolio are correlated with each other and considering all possible correlations between all the assets in the portfolio is impractical. Assets can be grouped into categories depending on their liquidity, size, industry and region and it is only these stocks within each group that compete with each other for allocations within a portfolio. However, such a sense of hierarchy is lost in a correlation matrix where all assets are potential replacements for one another. This is a very important point since this is how many asset managers manage a portfolio – by comparing stocks sharing some similarities with each other and rebalancing them within individual groups.

All these disadvantages make CLA and other similar allocation algorithms unsuitable for practical applications and this is where Hierarchical Risk Parity (HRP) comes into the picture as it tries to address the above mentioned points and improve upon them. In the following sections, we will understand its algorithm in detail and finally end with a performance comparison with CLA.

HOW HIERARCHICAL RISK PARITY ACTUALLY WORKS?

The algorithm can be broken down into 3 major steps:

- **Hierarchical Tree Clustering**
- **Matrix Seriation**
- **Recursive Bisection**

Lets understand each step in detail. Note that we will use real stock data in the form of a .csv file containing stock-returns indexed by date. For a more detailed approach along with the code and the data file, please refer to this online research notebook and this Github repository for other cool notebooks.

Hierarchical Tree Clustering

This step breaks down the assets in our portfolio into different hierarchical clusters using the famous Hierarchical Tree Clustering algorithm. Specifically, we calculate the tree clusters based on the TxN matrix of stock returns where T represents the timeseries of the data and N represents the number of stocks in our portfolio. Note that this method combines the items into a cluster rather than breaking down a cluster into individual items i.e. it does an agglomerative clustering. Lets understand how the clusters are formed in a step-by-step manner:

1. Given a TxN matrix of stock returns, calculate the correlation of each stock's returns with the other stocks which gives us an NxN matrix of these correlations,

2. The correlation matrix is converted to a correlation-distance matrix D, where,

$$(i, j) = \sqrt{0.5 * (1 - \rho(i, j))}$$

3. Now, we calculate another distance matrix $\overline{D}$ where,

$$\overline{D}(i, j) = \sqrt{\sum_{k=1}^{N} (D(k, i) - D(k, j))^2}$$

   It is formed by taking the Euclidean distance between all the columns in a pair-wise manner.

4. A quick explanation regarding the difference between D and $\overline{D}$ – for two assets i and j, $D(i, j)$ is the distance between the two assets while $\overline{D}(i, j)$ indicates the closeness in similarity of these assets with the rest of the portfolio. This becomes obvious when we look at the formula for calculating $\overline{D}$ – we sum over the squared difference of distances of i and j from the other stocks. Hence, a lower value means that assets i and j are similarly correlated with the other stocks in our portfolio.

5. We start forming clusters of assets using these distances in a recursive manner. Let us denote the set of clusters as U. The first cluster (i^*, j^*) is calculated as,

$$U[1] = argmin_{(i,j)} \overline{D}(i, j)$$

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

7. We remove the columns and rows corresponding to the new cluster – in this case we remove rows and columns for stocks a and b. For calculating the distance of an asset i outside this cluster, we use the following formula

	(a,b)	c	d	e
(a,b)	0	**21**	**31**	**21**
c	**21**	0	*28*	*39*
d	**31**	*28*	0	*43*
e	**21**	*39*	*43*	0

Using the above formula we calculate distances for c, d and e from cluster (a, b).

$$\overline{D}(c, U[1]) = min(\overline{D}(c, a), \overline{D}(c, b)) = min(21, 30) = 21$$

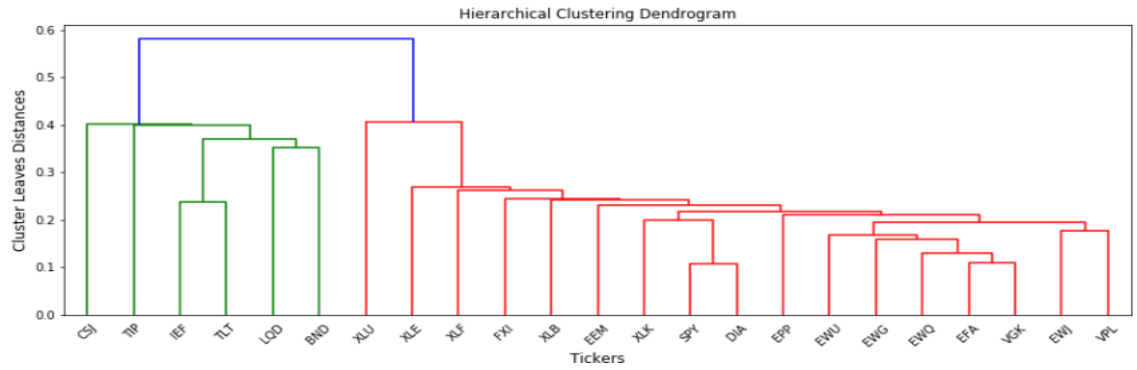
$$\overline{D}(d, U[1]) = min(\overline{D}(d, a), \overline{D}(d, b)) = min(31, 34) = 31$$

$$\overline{D}(e, U[1]) = min(\overline{D}(e, a), \overline{D}(e, b)) = min(23, 21) = 21$$

8. In this way, we go on recursively combining assets into clusters and updating the distance matrix until we are left with one giant cluster of stocks as shown in the following image where we finally combine d with ((a, b), c, e).

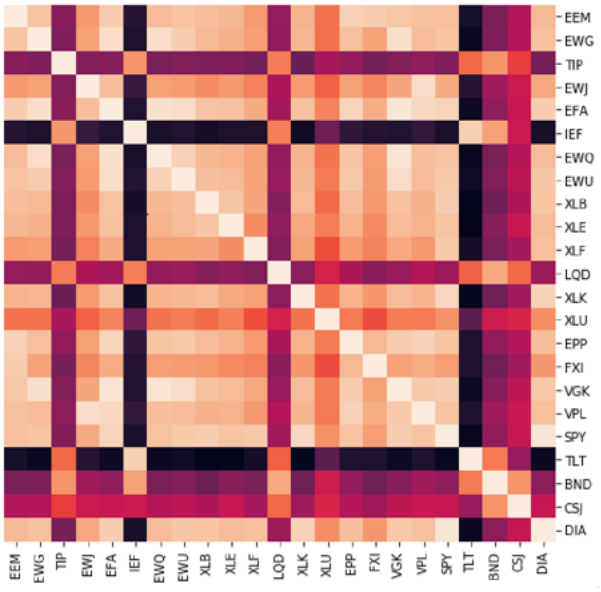
	((a,b),c,e)	d
((a,b),c,e)	0	**28**
d	**28**	0

9. Finally, in hierarchical clustering, the clusters are always visualised in the form of a nice cluster diagram called dendrogram. Below is the image of the hierarchical clusters for our stock data,

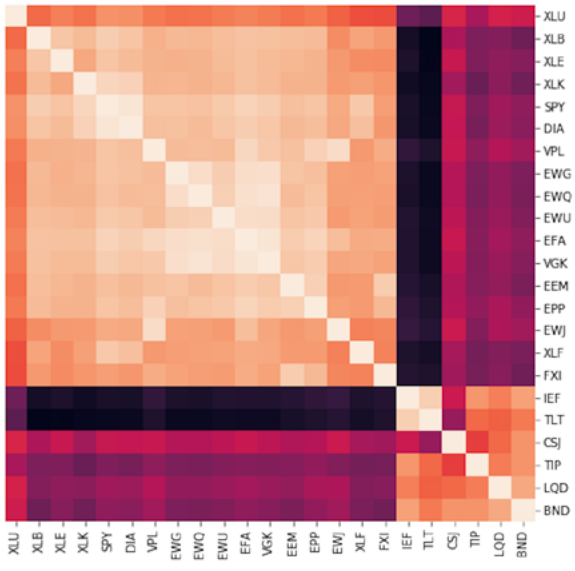


Matrix Seriation

TIn the **original paper** of this algorithm, this step is identified as Quasi-Diagonalisation but it is nothing more than a simple seriation algorithm. Matrix seriation is a very old statistical technique which is used to rearrange the data to show the inherent clusters clearly. Using the order of hierarchical clusters from the previous step, we rearrange the rows and columns of the covariance matrix of stocks so that similar investments are placed together and dissimilar investments are placed far apart. This rearranges the original covariance matrix of stocks so that larger covariances are placed along the diagonal and smaller ones around this diagonal and since the off-diagonal elements are not completely zero, this is called a **quasi-diagonal covariance matrix**



Unclustered Correlations



Clustered Correlations

The above image shows the seriated correlation matrix for our data. We see that before seriation, the asset clusters are broken down into small sub-sections and it is only after quasi-diagonalising/seriating the matrix, does the clustering structure become more evident.

Recursive Bisection

This is the final and the most important step of this algorithm where the actual weights are assigned to the assets in our portfolio.

1. We initialise the weights of the assets,

$$W_i = 1, \forall i = 1, \ldots, N$$

2. At the end of the tree-clustering step, we were left with one giant cluster with subsequent clusters nested within each other. We now break each cluster into two sub-clusters by starting with the topmost cluster and traversing in a top-down manner. This is where Hierarchical Risk Parity makes use of Step-2 to quasi-diagonalise the covariance matrix and uses this new matrix for recursing into the clusters

3. Hierarchical tree clustering forms a binary tree where each cluster has a left and right child cluster V_1 and V_2. For each of these sub-clusters, we calculate its variance,

$$V_{adj} = w^T V w$$

where,

$$w = \frac{diag[V]^{-1}}{trace(diag[V]^{-1})}$$

This step exploits the property that for a diagonal covariance matrix, the inverse-variance allocations are the most optimal. Since, we are dealing with a quasi-diagonal matrix, it makes sense to use the inverse-allocation weights to calculate the variance for this subcluster.

4. A weighting factor is calculated based on the new covariance matrix

$$\alpha_1 = 1 - \frac{V_1}{V_1 + V_2}; \alpha_2 = 1 - \alpha_1$$

You might be wondering how do we arrive at the above formula for the weighting factor. It comes from the classical portfolio Optimization theory where we have the following Optimization objective,

$$min \frac{1}{2} w^T \sigma w$$

$$s.t. \, e^T w = 1; e = 1^T$$

where w are the portfolio weights and σ is the covariance of the portfolio. For a minimum variance portfolio, the solution to the above equation comes out to be,

$$w = \frac{\sigma^{-1} e}{e^T \sigma^{-1} e}$$

And if σ is diagonal,

$$w = \frac{\sigma_{n,n}^{-1}}{\sum_{i=1} N\sigma_{i,i}^{-1}}$$

Since, we only have N = 2 (2 subclusters) this equation becomes,

$$w = \frac{1/\sigma_1}{1/\sigma_1 + 1/\sigma 2} = 1 - \frac{\sigma_1}{\sigma_1 + \sigma_2}$$

which is the same formula used by Hierarchical Risk Parity.

5.The weights of stocks in the left and right subclusters are then updated respectively,

$$W_1 = \alpha_1 * W_1$$

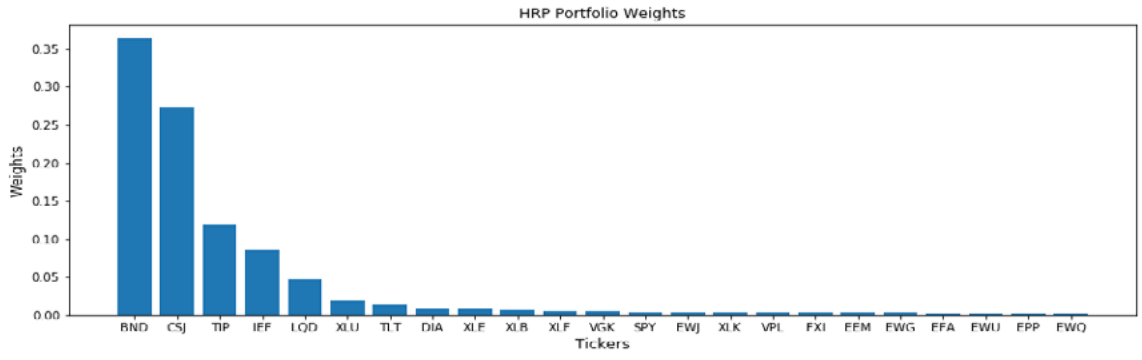
$$W_2 = \alpha_2 * W_2$$

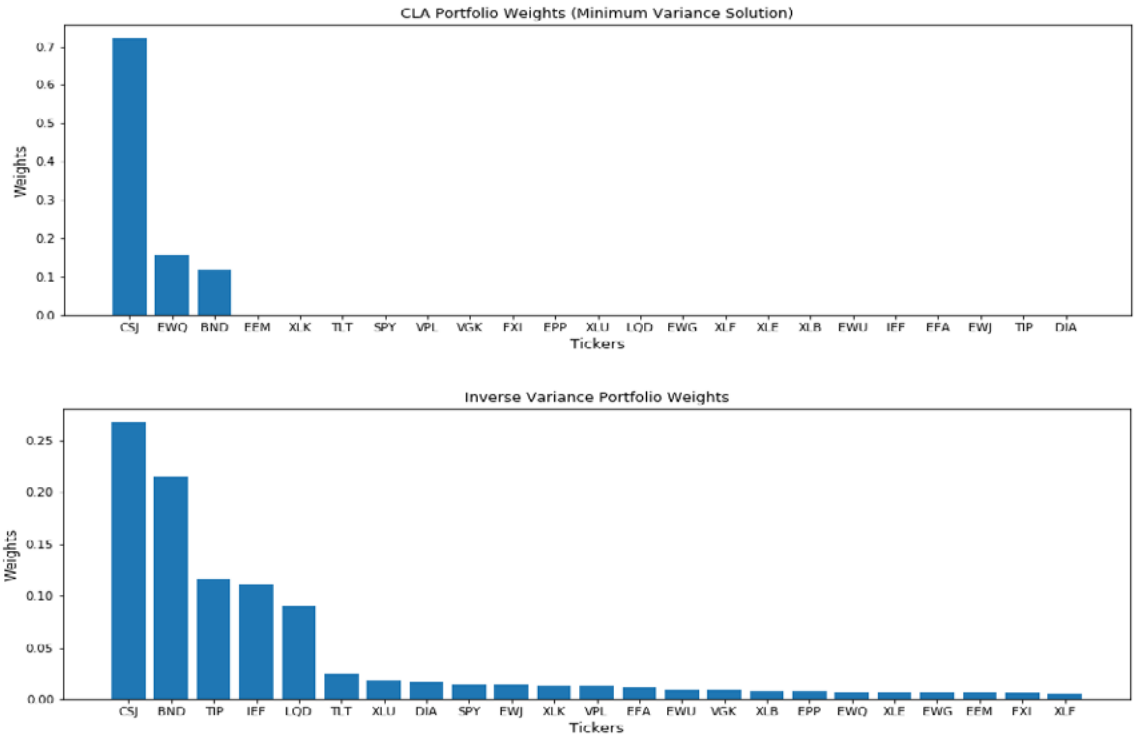
6. We recursively execute steps 2-5 on V_1 and V_2 until all the weights are assigned to the stocks.

Notice how the weights are assigned in a top-down manner based on the variance within a sub-cluster. The main advantage of such a hierarchical weight assignment is that only assets within the same group compete for allocation with each other rather than competing with all the assets in the portfolio.

Performance of Hierarchical Risk Parity

Having understood the working of the Hierarchical Risk Parity algorithm in detail, we now compare its performance with other allocation algorithms. In particular, we will compare it with 2 algorithms – the Inverse-Variance Allocation (IVP) and Critical Line Algorithm (CLA). We will be using PortfolioLab which already has these algorithms implemented in its portfolio Optimization module. In the original paper, the author uses monte-carlo simulations to compare the performances of these algorithms, however, we will use a different method to do so.





First, let's look at the weight allocations of the 3 algorithms for our normal portfolio,

We can see a major difference in how these 3 algorithms allocate their weights to the portfolio,

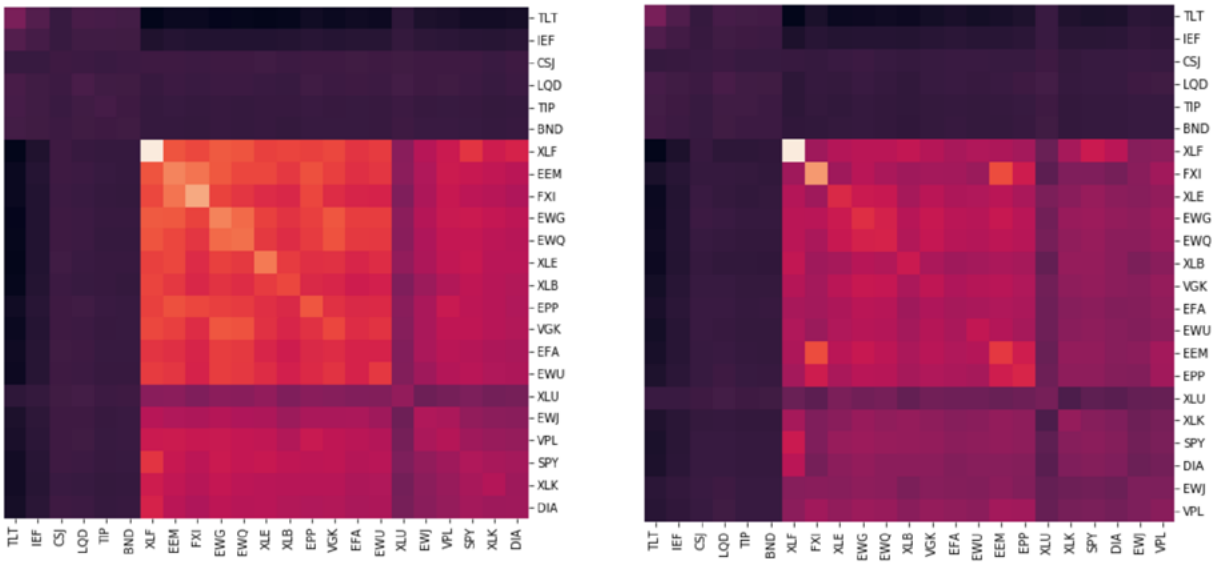
- CLA concentrates literally 99% of the holdings on the top-3 investments and assigns zero weight to all other assets. The reason behind CLA's extreme concentration is its goal of minimising the variance of the portfolio. This makes it take a very conservative approach in allocating weights and it places emphasis on only a few of the assets.

- Inverse variance (IVP) has assigned non-zero weights to all the assets and except the top 5 holdings, its weight allocations are distributed almost uniformly. Hence, it tends to take a safer approach and tries to allocate weights almost equally to the assets.

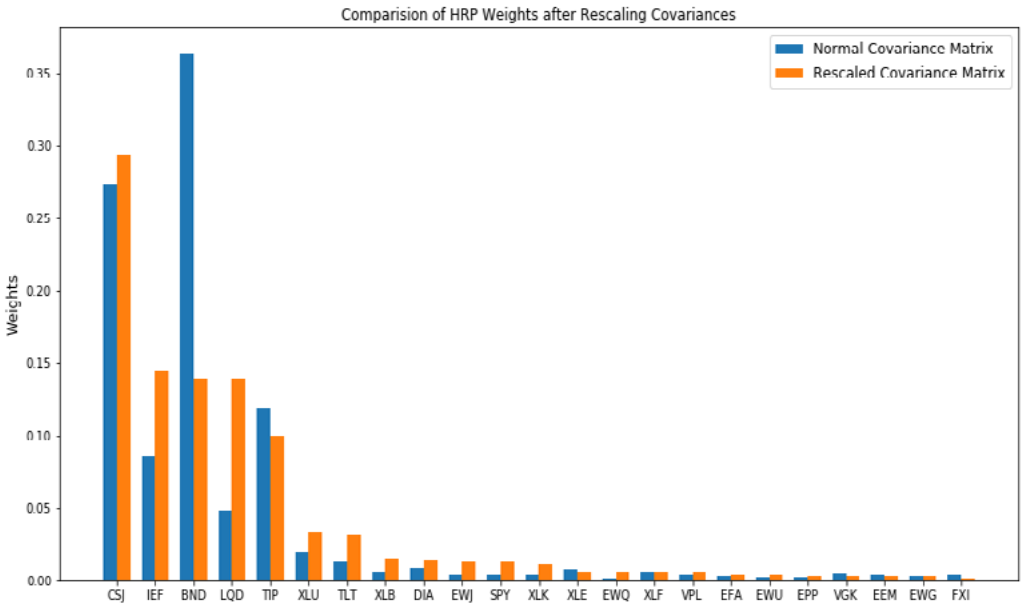
- HRP, on the other hand, tries to find a middle ground between CLA and IVP allocations. It places more emphasis on the top 5 holdings/assets just like IVP but assigns lesser but non-uniform weights to the rest of the stocks.

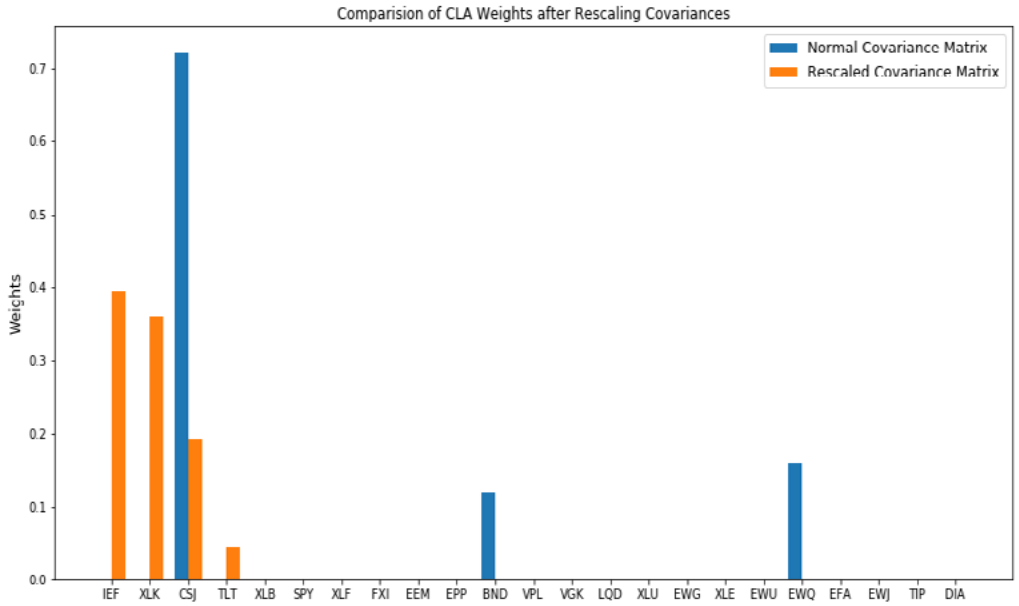
- Another important fact is that both the CLA and HRP weights have very little difference in their standard deviations, $\sigma^2_{HRP} = 0.12$ and $\sigma^2_{CLA} = 0.15$. However, CLA has discarded half of the investment universe in favor of a minor risk reduction while HRP did not. Since, CLA has placed its emphasis on only a few of the assets, it is prone to much more negative impact by random shocks than HRP – something which we will see below.

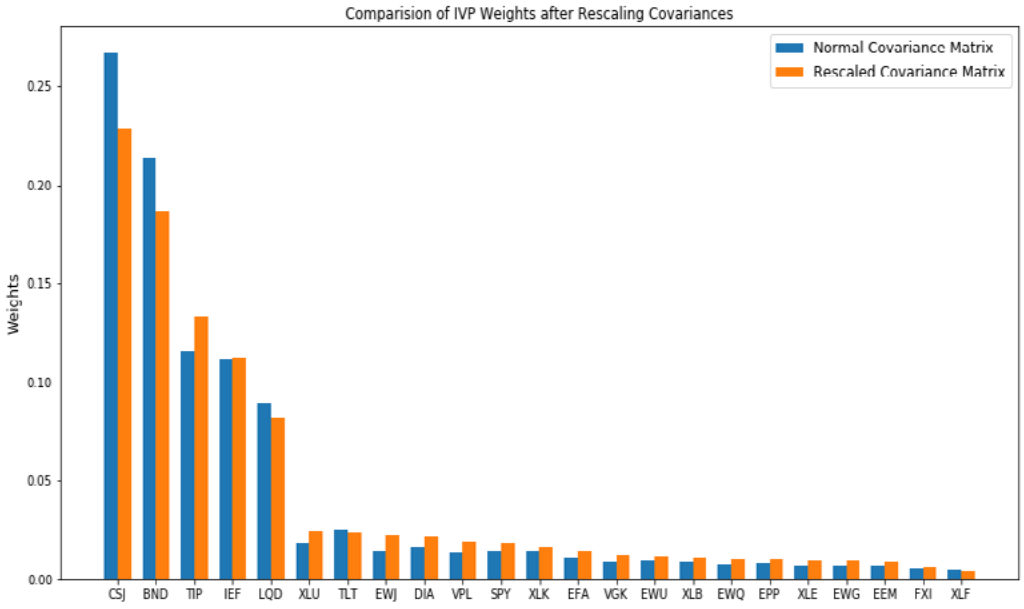
Now, using the same stocks, we will rescale the covariance matrix of these stocks by adding some random noise from a normal distribution.



n the above image, we see that the covariances do not change drastically and the relative covariances are still the same, however there is a visible change in the values. **Such a rescaling of covariances mimics the volatile nature of the stock-market and we get to see how the allocations of these algorithms are affected with such a sudden and small change in the relationships between the stocks in our portfolio.**







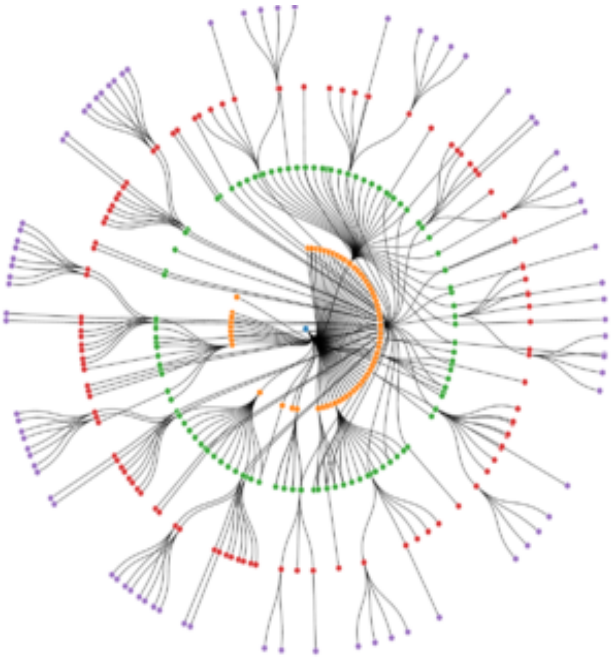
We observe that rescaling of the variances has led to a rebalancing of the portfolios for all the three strategies. Let's analyse the differences in this rebalancing

REFERENCES

1. López de Prado, Marcos, Building Diversified Portfolios that Outperform Out-of-Sample (May 23, 2016). Journal of Portfolio Management, 2016; https://doi.org/10.3905/jpm.2016.42.4.059. , Available at SSRN: https://ssrn.com/abstract=2708678 or http://dx.doi.org/10.2139/ssrn.2708678
2. Marti, Gautier. Hierarchical Risk Parity - Implementation & Experiments (Part I), 2 Oct. 2018, gmarti.gitlab.io/qfin/2018/10/02/hierarchical-risk-parity-part-1.htm.
3. What Is Seriation?, www.atgc-montpellier.fr/permutmatrix/manual/SeriationCorps.htm.
4. Rcpp. Hierarchical Risk Parity Implementation in Rcpp and OpenMP, gallery.rcpp.org/articles/hierarchical-risk-parity/.

7.0

HIERARCHICAL RISK PARITY WITH PORTFOLIOLAB



In 2016, Dr. Marcos Lopez de Prado introduced the Hierarchical Risk Parity (HRP) algorithm for portfolio optimization. Prior to this, Harry Markowitz's Modern Portfolio Theory (MPT) was used as an industry-wide benchmark for portfolio optimization. MPT was an amazing accomplishment in the field of portfolio optimization and risk management, earning Harry Markowitz a Nobel Prize for his work. However, even though it is mathematically sound, it fails to produce optimal portfolios which can perform in real-world scenarios. This can be mainly attributed to two different reasons:

1. It involves the estimation of returns for a given set of assets. In real life, accurately estimating returns for a set of assets is very difficult, and even small errors in estimation can cause sub-optimal performance.

2. Mean-variance optimization methods involve the inversion of a covariance matrix for a set of assets. This matrix inversion makes the algorithm susceptible to market volatility and can heavily change the results for small changes in the correlations.

The HRP algorithm aims to solve some of these issues with a new approach based on the notion of hierarchy. This algorithm is computed in three main steps:

1. Hierarchical Clustering – breaks down our assets into hierarchical clusters

2. Quasi-Diagonalization – reorganizes the covariance matrix, placing similar assets together

3. Recursive Bisection – weights are assigned to each asset in our portfolio

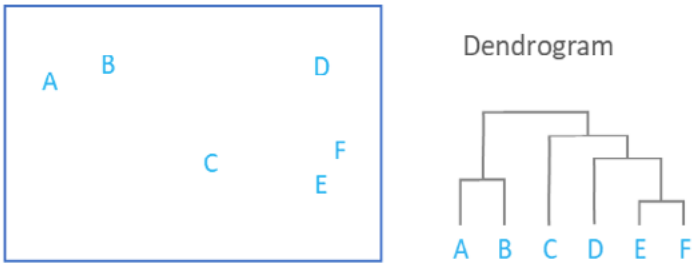
Throughout this article, we will explore the intuition behind HRP and also learn to apply it using the PortfolioLab library. Please keep in mind that this is a tutorial styled article and if you would like to learn more about the theory behind this algorithm, please refer to this article.

HOW THE HIERARCHICAL RISK PARITY ALGORITHM WORKS?

In this section, we will go through each step of the HRP algorithm quickly and explain the intuition behind them.

Hierarchical Clustering

Hierarchical clustering is used to place our assets into clusters suggested by the data and not by previously defined metrics. This ensures that the assets in a specific cluster maintain similarity. The objective of this step is to build a hierarchical tree in which our assets are all clustered on different levels. Conceptually, this may be difficult for some to understand, which is why we can visualize this tree through a dendrogram.



The previous image shows the hierarchical clustering process results through a dendrogram. As the square containining our assets A-F showcases the similarity between each other, we can understand how the assets are clustered. Keep in mind that we are using agglomerative clustering, which assumes each data point to be an individual cluster at the start.

First, the assets E and F are clustered together as they are the most similar. This is followed by the clustering of assets A and B. From this point, the clustering algorithm then includes asset D (and subsequently asset C) into the first clustering pair of assets E and F. Finally, the asset pair A and B is then clustered with the rest of the assets in the last step.

So you now may be asking, how does the algorithm know which assets to cluster together? Of course, we can visually see the distance between each asset, but our algorithm cannot. There are a few widely used methods for calculating the measure of distance/similarity within our algorithm:

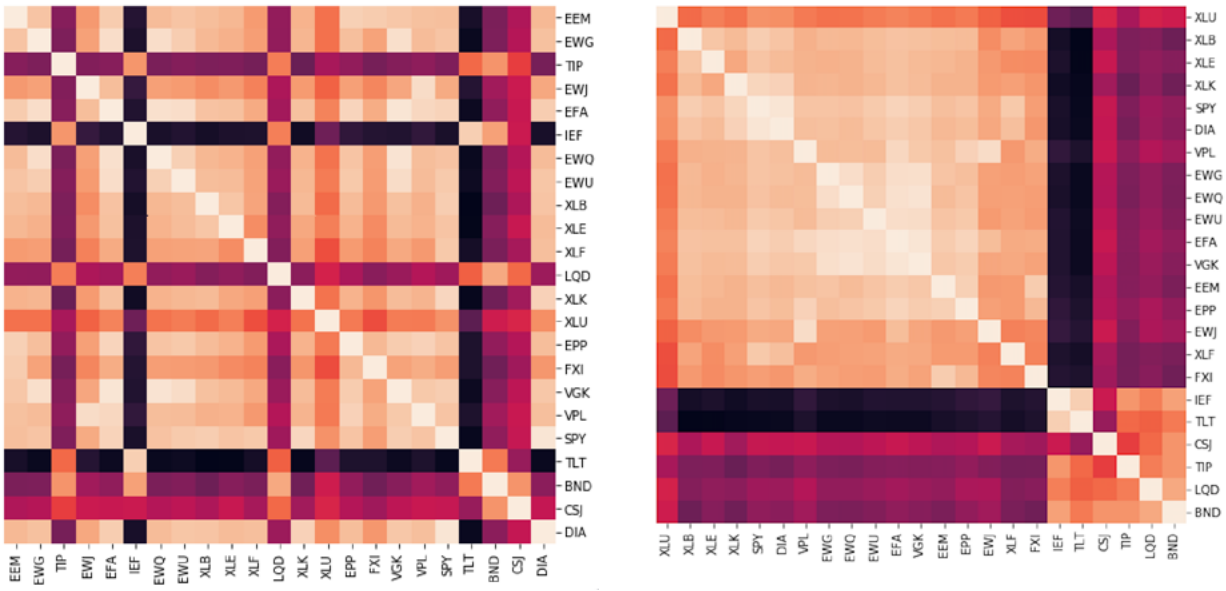
- **Single Linkage** – the distance between two clusters it the minimum distance between any two points in the clusters
- **Complete Linkage** – the distance between two clusters is the maximum of the distance between any two points in the clusters

- **Average Linkage** – the distance between two clusters is the average of the distance between any two points in the clusters

Thankfully, we can easily implement each linkage algorithm within the PortfolioLab library, allowing us to quickly compare the results to each other.

Quasi-Diagonalization

Once our assets are all clustered into a hierarchical tree, we can now perform our quasi-diagonalization step in our algorithm. From our previous step, we clustered all our assets into a hierarchical tree based on similarity defined through our chosen distance measure. In this step, we rearrange the rows and columns of the covariance matrix of assets so that we place similar assets together and dissimilar assets further apart. Once completed, this step rearranges our covariance matrix in a way so that the larger covariances in our matrix are placed along the diagonal, with the smaller ones spread around this diagonal. Because the off-diagonal elements are not completely zero, this is called our quasi-diagonal covariance matri



From the images shown above, we can see how the unclustered matrix shows asset clusters in small sub-sections of our matrix, while after quasi-diagonalization our clustering structure becomes much more visible.

At this point in our algorithm, all of our assets have been clustered in a hierarchical tree and our covariance matrix has been rearranged accordingly.

Recursive Bisection

Recursive bisection is the final and most important step in our algorithm. In this step, the actual portfolio weights are assigned to our assets in a top-down recursive manner.

At the end of our first step, we were left with our large hierarchical tree with one giant cluster and subsequent clusters nested within each other. By performing this step, we break each cluster into sub-clusters by starting with our largest cluster and moving down our tree in a top-down manner. Recursive bisection makes use of our quasi-diagonalized covariance matrix for recursing into the clusters under the assumption that for a diagonal matrix, the inverse-variance allocation is the most optimal allocation.

One of the main advantages to having this step in our algorithm is that our assets are only competing for weight allocation within the same cluster, leading us to developing a much more robust portfolio.

USING PORTFOLIOLAB'S HRP IMPLEMENTATION

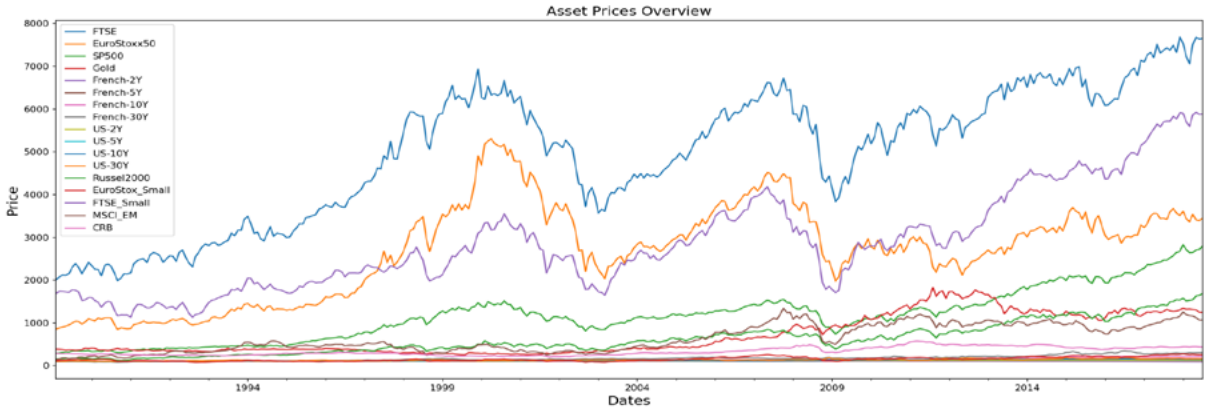
In this section, we will go through a working example of using the Hierarchical Risk Parity implementation provided by PortfolioLab and test it on a portfolio of assets.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import mlfinlab as ml

import matplotlib.patches as mpatches

from portfoliolab.clustering import HierarchicalRiskParity
```

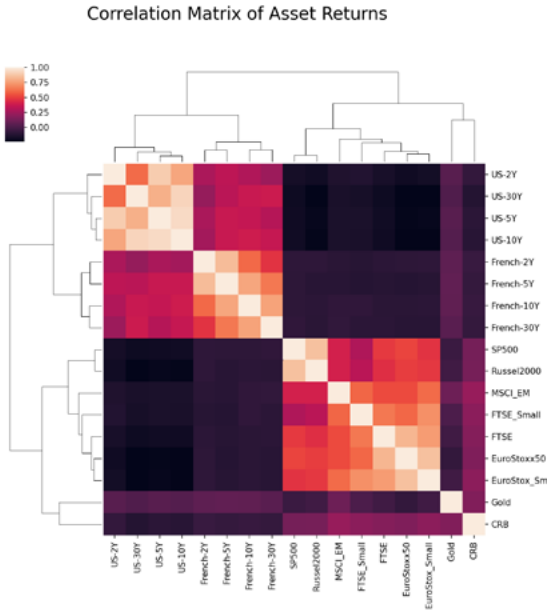
Choosing the Dataset

In this example, we will be working with historical closing-price data for 17 assets. The portfolio consists of diverse set of assets ranging from commodities to bonds.

```
# reading in our data

raw_prices = pd.read_csv('assetalloc.csv', sep=';', parse_dates=True, index_col='Dates')

stock_prices = raw_prices.sort_values(by='Dates')

stock_prices.head()stock_prices.resample('M').last().plot(figsize=(17,7))

plt.ylabel('Price', size=15)

plt.xlabel('Dates', size=15)

plt.title('Stock Data Overview', size=15)

plt.show()
```



The specific set of securities have very good clustering structure which is important for hierarchical based algorithms. If you look at the visual representation of the correlation matrix below, the inherent clusters can be clearly identified. Hierarchical clustering based algorithms achieve the best performance when the data can be divided easily into clusters.



Calculating the Optimal Weight Allocations

Now that we have our data loaded in, we can make use of the HierarchicalRiskParity class from PortfolioLab to construct our optimal portfolio. First we must instantiate our class and then run the allocate() method to optimize our portfolio.

The allocate method requires only two parameters to run its default solution:

1. **asset_names** (a list of strings containing the asset names)
2. **asset_prices** (a dataframe of historical asset prices – daily close)

Note: The list of asset names is not a necessary parameter. If your input data is in the form of a dataframe, it will use the column names as the default asset names.

Users are also given the option to customize many different parameters in the allocate() method, some of these include:

1. The type of linkage algorithm being used
2. Pass a custom distance matrix
3. Build a long/short portfolio

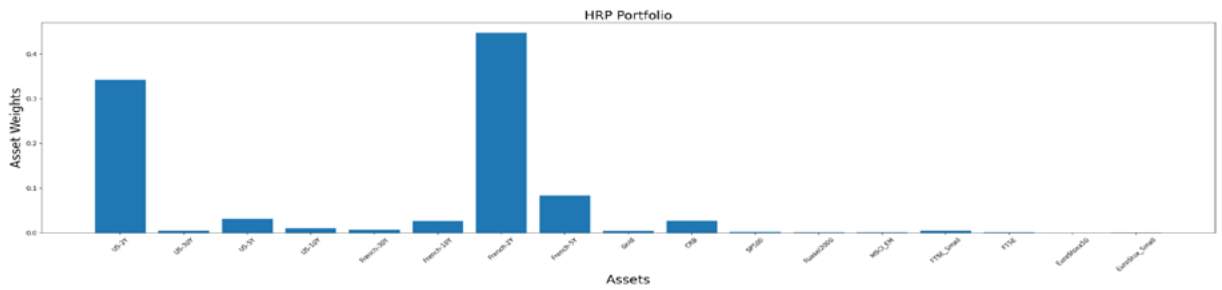
PortfolioLab currently supports all four linkage algorithms discussed in this article, although the default method described in the original HRP paper is the Single Linkage algorithm.

Additionally, instead of providing raw historical closing prices, users can choose to input their own asset returns and a covariance matrix. We will explore this in more detail later but for now, we will only be working with the two required parameters as well as specifying our linkage algorithm of choice.

```
# constructing our HRP portfolio - Single Linkage
hrp = HierarchicalRiskParity()
hrp.allocate(asset_names=stock_prices.columns,
             asset_prices=stock_prices,
             linkage='single')
```

```
# plotting our optimal portfolio
hrp_weights = hrp.weights
y_pos = np.arange(len(hrp_weights.columns))
plt.figure(figsize=(25,7))
plt.bar(list(hrp_weights.columns), hrp_weights.values[0])
plt.xticks(y_pos, rotation=45, size=10)
```

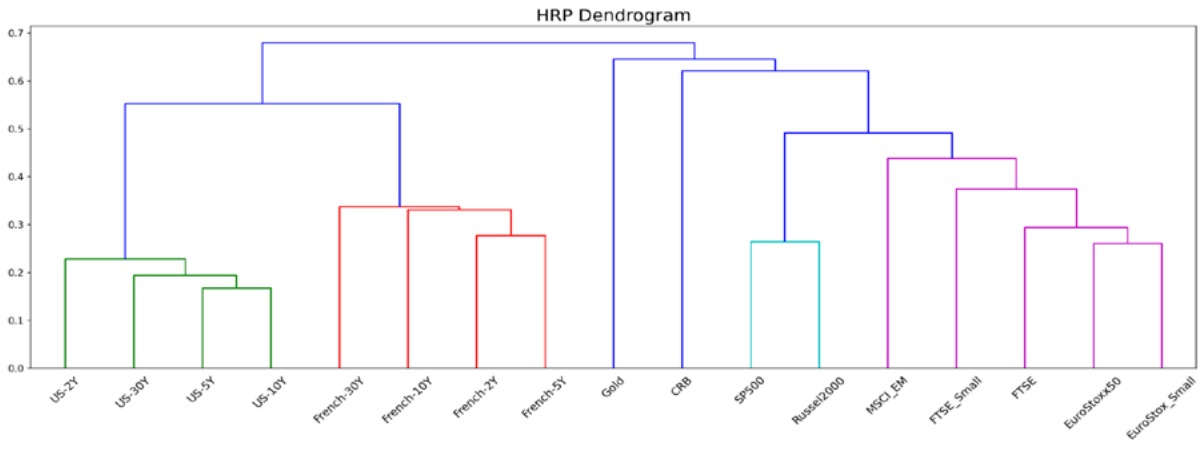
```
plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('HRP Portfolio', size=20)

plt.tight_layout()

plt.savefig('HRP Portfolio Weights')

plt.show()
```



Plotting the Clusters

Having allocated the weights, let us look at the tree structure generated by the hierarchical clustering step. This is visualised in the form of a dendrogram as shown below.

```
# plotting dendrogram of HRP portfolio

plt.figure(figsize=(17,7))

hrp.plot_clusters(stock_prices.columns)

plt.title('HRP Dendrogram', size=18)

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()
```



In this graph, the different colors of the tree structure represent the clusters that the stocks belong to.

USING CUSTOM INPUT WITH PORTFOLIOLAB

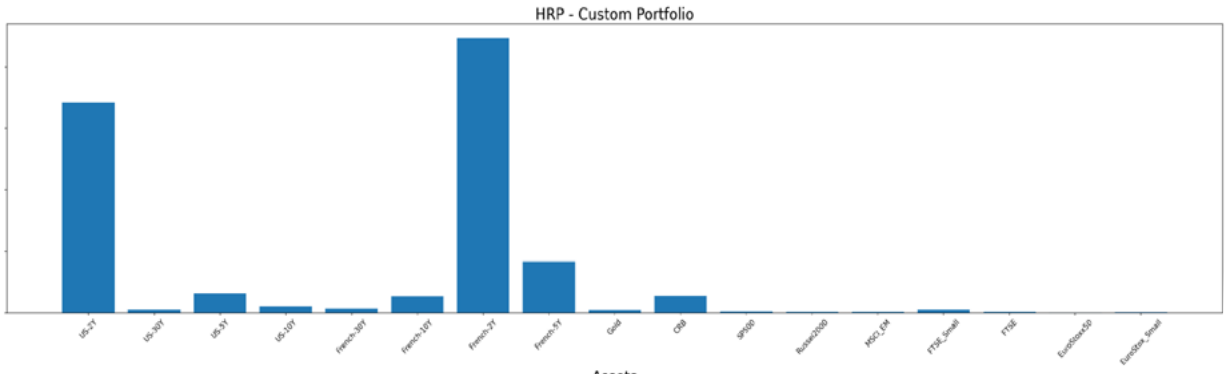
PortfolioLab also provides users with a lot of customizability when it comes to creating their optimal portfolios. Instead of providing the raw historical closing prices for assets, users can instead input asset returns, a covariance matrix of asset returns, a distance matrix, and side weights.

In this section, we will make use of the following parameters in the allocate() method to construct a custom use case:

1. **asset_returns** – (pd.DataFrame/numpy matrix) User supplied matrix of asset returns
2. **covariance_matrix** – (pd.DataFrame/numpy matrix) User supplied covariance matrix of asset returns
3. **distance_matrix** – (pd.DataFrame/numpy matrix) User supplied distance matrix

We will be constructing our first custom portfolio using the asset_returns and covariance_matrix parameters. To make some of the necessary calculations, we will make use of the ReturnsEstimators class provided by PortfolioLab.

```
# importing ReturnsEstimation class from PortfolioLab
from portfoliolab.estimators import ReturnsEstimators


# calculating our asset returns
returns = ReturnsEstimators.calculate_returns(stock_prices)


# calculating our covariance matrix
cov = returns.cov()


# constructing our first custom portfolio
hrp_custom = HierarchicalRiskParity()
hrp_custom.allocate(asset_names=stock_prices.columns,
                    asset_returns=returns,
                    covariance_matrix=cov)


# plotting our optimal portfolio
hrp_custom_weights = hrp_custom.weights
y_pos = np.arange(len(hrp_custom_weights.columns))


plt.figure(figsize=(25,7))
plt.bar(list(hrp_custom_weights.columns), hrp_custom_weights.values[0])
plt.xticks(y_pos, rotation=45, size=10)
plt.xlabel('Assets', size=20)
plt.ylabel('Asset Weights', size=20)
plt.title('HRP - Custom Portfolio', size=20)
plt.tight_layout()
plt.show()
```



You can notice we get the same weight allocations as the ones when we instead passed raw asset prices. You can use your own pre-calculated covariance matrices and asset returns or rely on our inbuilt calculation methods for calculating weights.

Passing a Custom Distance Matrix

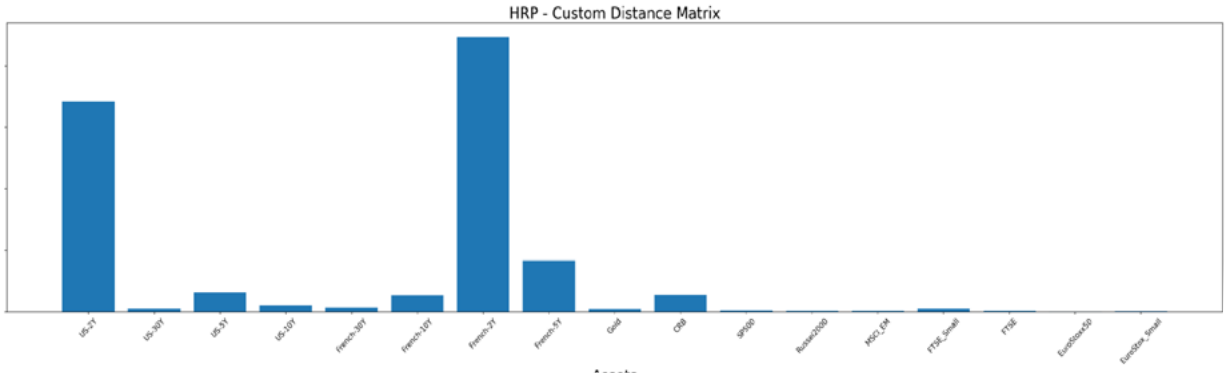
Note that the Hierarchical Risk Parity algorithm proposed in the original paper uses the following distance matrix:

$$D = \sqrt{0.5 * (1 - \rho)}$$

where ρ refers to the correlation matrix of asset returns. Users can choose to use the HRP method as it is or can tweak its performance by passing their our own distance matrix to calculate our optimal portfolio. All other steps of the algorithm will stay the same.

```
from portfoliolab.estimators import RiskEstimators


# create our own distance matrix

corr = RiskEstimators.cov_to_corr(cov)

distance = np.sqrt((1 - corr).round(5) / 2)


# allocate weights

hrp_distance = HierarchicalRiskParity()

hrp_distance.allocate(asset_names=stock_prices.columns, distance_matrix=distance,

                      covariance_matrix=cov)
```

```
# plotting our optimal portfolio

hrp_distance_weights = hrp_distance.weights

y_pos = np.arange(len(hrp_distance_weights.columns))


plt.figure(figsize=(25,7))

plt.bar(list(hrp_distance_weights.columns), hrp_distance_weights.values[0])

plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('HRP - Custom Distance Matrix', size=20)

plt.show()
```



BUILDING A LONG-SHORT PORTFOLIO

This is an extra feature we have added to our HRP implementation. By default, shorting of assets is not allowed in the original algorithm. But with PortfolioLab's implementation, you can pass a side_weights parameter to short some assets in your portfolio.

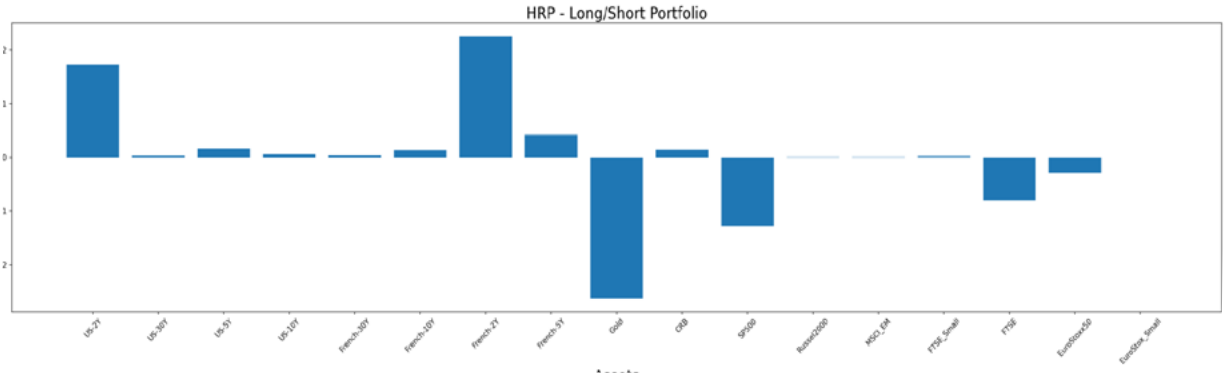
In the following example, we will short the first four stocks in our dataset. All other steps remain the same – the only difference being the addition of the side_weights parameter to indicate which stocks we would like to short and long (-1 indicates shorting a stock and 1 indicates going long on a stock).

```
side_weights = pd.Series([1]*stock_prices.shape[1], index=stock_prices.columns)

# short the first 4 stocks

side_weights.loc[stock_prices.columns[:4]] = -1

print(side_weights)
```

```
FTSE             -1
EuroStoxx50      -1
SP500            -1
Gold             -1
French-2Y         1
French-5Y         1
French-10Y        1
French-30Y        1
US-2Y             1
US-5Y             1
US-10Y            1
US-30Y            1
Russel2000        1
EuroStox_Small    1
FTSE_Small        1
MSCI_EM           1
CRB               1
dtype: int64
```

```
# calculating optimal weights

hrp_ls = HierarchicalRiskParity()

hrp_ls.allocate(asset_names=stock_prices.columns,

                asset_prices=stock_prices,

                side_weights=side_weights)


# plotting our optimal portfolio

hrp_ls_weights = hrp_ls.weights

y_pos = np.arange(len(hrp_ls_weights.columns))

plt.figure(figsize=(25,7))

plt.bar(list(hrp_ls_weights.columns), hrp_ls_weights.values[0])
```

```
plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('HRP - Long/Short Portfolio', size=20)

plt.tight_layout()

plt.savefig('HRP Long-Short Portfolio')

plt.show()
```

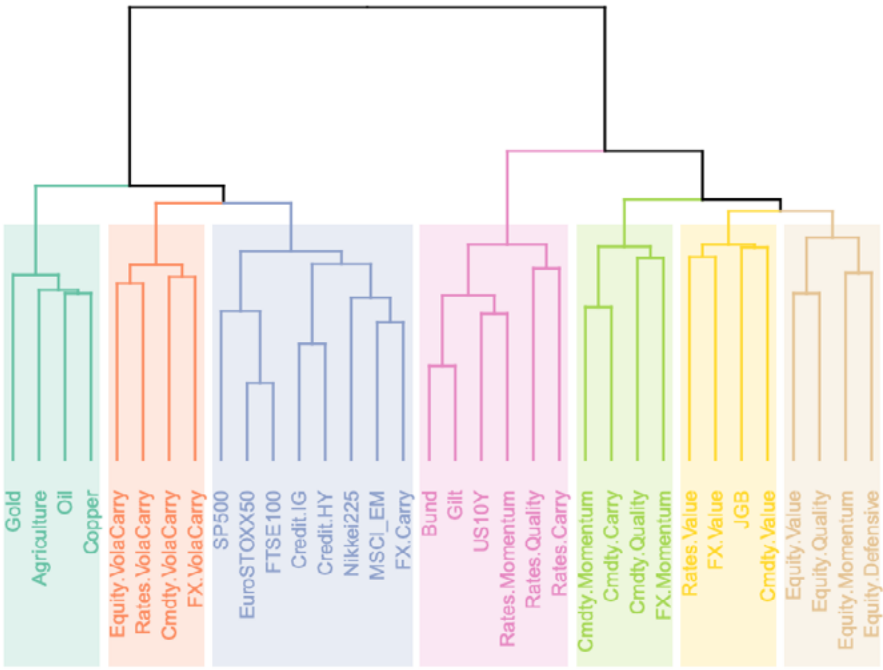


CONCLUSION

Through this article, we learned the intuition behind the Hierarchical Risk Parity algorithm and also saw how we can utilize PortfolioLab's implementation to apply this technique out-of-the-box. HRP is a powerful algorithm that can produce robust risk-parity portfolios which avoids many of the limitations of traditional mean-variance Optimization methods.

8.0

BEYOND RISK PARITY: THE HIERARCHICAL EQUAL RISK CONTRIBUTION



Dendrogram based on Ward's method. (Lohre, H., Rother, C. and Schäfer, K.A., 2020)

As diversification is the only free lunch in finance, the Hierarchical Equal Risk Contribution Portfolio (HERC) aims at diversifying capital allocation and risk allocation. Briefly, the principle is to retain the correlations that really matter and once the assets are hierarchically clustered, a capital allocation is estimated. HERC allocates capital within and across the "right" number of clusters of assets at multiple hierarchical levels. This Top-Down recursive division is based on the shape of the dendrogram, the optimal number of clusters and follows an Equal Risk Contribution allocation. In contrary to modern portfolio optimization techniques, HERC portfolios are diversified and outperform out-of-sample.

– Dr. Thomas Raffinot

Ever since Prof. Lopez de Prado's seminal paper on Hierarchical Risk Parity (HRP), there has been an abundance of research on using hierarchical clustering strategies for portfolio allocation. This was further stimulated by the fact that HRP showed better performance on out-of-sample data, suggesting that the use of hierarchy identified by the clustering step is indeed helpful in achieving an optimal weight allocation – something which is ignored by traditional Optimization algorithms. The Hierarchical Risk parity algorithm can be broken down into three steps:

1. Hierarchical tree clustering

2. Quasi-Diagnalisation

3. Recursive bisection

The use of tree clustering to group assets and assign weights is not new. In his 2011 PhD thesis – **Asset Clusters and Asset Networks in Financial Risk Management and Portfolio Optimization** – Dr. Jochen Papenbrock proposes a cluster-based waterfall approach where the assets are clustered in a hierarchical tree and at each bisection, the weights are split equally until there are no more bisections.

In 2017, Thomas Raffinot created the Hierarchical Clustering based Asset Allocation (HCAA) algorithm. It is similar to HRP and Dr. Papenbrock's waterfall approach and uses hierarchical clustering to allocate the weights. It consists of four main steps:

1. Hierarchical tree clustering

2. Selecting optimal number of clusters

3. Allocation of capital across clusters

4. Allocation of capital within clusters

This was followed by another paper by Dr. Raffinot in 2018, "The Hierarchical Equal Risk Contribution Portfolio" which proposed the Hierarchical Equal Risk Contribution (HERC) algorithm. Quoting directly from the paper – "HERC merges and enhances the machine learning approach of HCAA and the Top-down recursive bisection of HRP". It can be broken down into four major steps:

1. Hierarchical tree clustering

2. Selecting optimal number of clusters

3. Top-down recursive bisection

4. Naive risk parity within the clusters

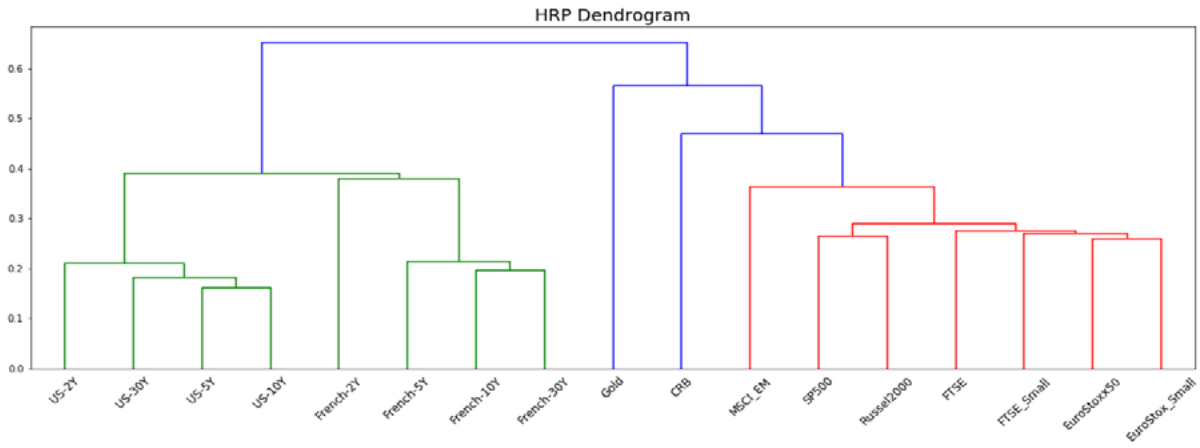
You can clearly observe that steps from both HRP and HCAA have been combined to create the HERC algorithm. In this article, we will be covering a more theoretical explanation of each of these steps, understand the motivations behind them and finally look at a quick performance comparison of HERC and HRP.

CRITICISMS OF HIERARCHICAL RISK PARITY

Portfolios generated by HRP exhibit better out-of-sample performance than other traditional portfolio allocation algorithms. However, the algorithm – and in general any hierarchical based allocation strategy – comes with its own set of drawbacks.

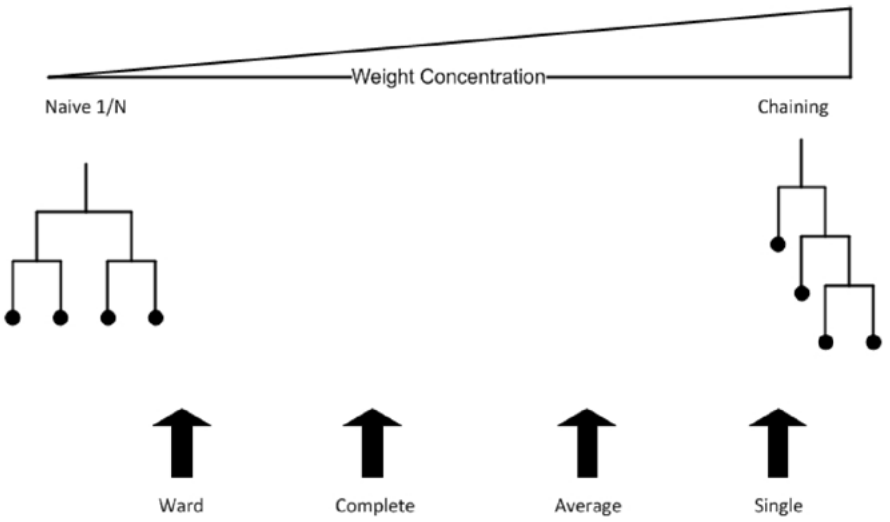
Depth of Hierarchical Tree

The hierarchical tree clustering algorithm identifies and segregates the assets in our portfolio into different clusters. At the end of the step, you are left with a tree which can be visualised in the form of a dendrogram, as shown below:



Starting with each asset being an individual cluster, the clustering algorithm uses something called **linkage** to determine how to combine the clusters. There are different linkage methods which can be used – **single, ward, average** and **complete**. HRP employs single linkage clustering which builds the tree based on the distance between the two closest points in these clusters. This results in a chaining effect and makes the tree very deep and wide, preventing any dense clusters from being formed and affecting the weight allocations.

This has been explained in detail in Dr. Papenbrock's previously mentioned thesis through the following image:



The chaining effect of single linkage results in large weights to be allocated to few assets and an unequal distribution of the portfolio. Hence, using other linkage methods is important for optimal allocation – something which is incorporated in HERC. By default, it uses Ward linkage but users can specify any other methods to work with it.

Identifying the Optimal Number of Clusters

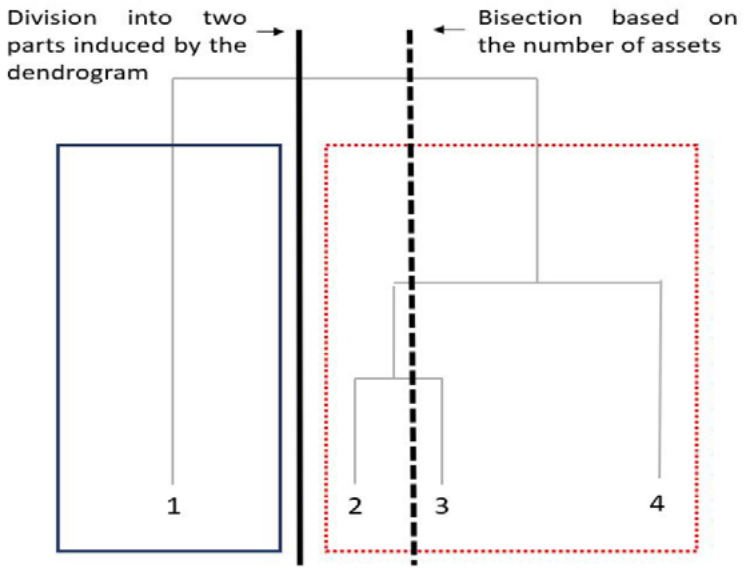
The hierarchical tree in the previous image starts with one big cluster and successively breaks into two new clusters at each level, with the end result being that each asset is in its own individual cluster. There are some problems with this approach:

- Forming such large trees makes the algorithm computationally slow for very large datasets.
- Letting the tree grow completely on our data also leads to the problem of overfitting.
  This can result in very small inaccuracies in the data leading to large estimation errors in the portfolio weights.

All these problems indicate a need to enforce an early stopping criteria for the tree and a way to calculate the exact number of clusters required. We will later talk about how this is handled by the HERC algorithm.

Not Following the Dendrogram Structure

During recursive bisection, the weights trickle down the tree until all the assets at the bottom-most level are assigned the respective weights. The problem here is that HRP does not stay true to the dendrogram structure, and instead it chooses to bisect the tree based on the number of assets.



The above figure is taken from HERC paper and illustrates the importance of following the tree structure. If you look closely, dividing the tree based on number of assets wrongly separates assets {1, 2} and {3, 4} while the correct cluster composition should have been {1} and {2, 3, 4}. Thomas Raffinot modified the recursive bisection of HRP to stay true to the natural dendrogram structure and we will later see how this affects the portfolio weights.

Variance as a Measure of Risk

In the recursive bisection step, HRP uses the variance of the clusters to calculate the weight allocations. This ensures that assets in clusters with minimum variance/volatility receive higher weights. We all know that estimates of risk in the form of the covariance matrix are always prone to errors and is one of the major factors why portfolio optimization methods fail to perform in the real world. At the same time, even if we are able to obtain an almost-accurate estimate of the covariances of the assets, it is extremely sensitive to small changes in the market conditions and ultimately result in serious estimation errors.

Also, while variance is a very simple and popular representation of risk used in the investing world, it can underestimate the true risk of a portfolio which is why there are many other important risk metrics used by investment managers that can try to ascertain the true risk of a portfolio/asset.

HERC modifies the recursive bisection of HRP to allow investors to use alternative risk measures – Expected Shortfall (CVaR), Conditional Drawdown at Risk (CDaR) and Standard Deviation. In Dr. Raffinot's words:

> Hierarchical Equal Risk Contribution portfolios based on downside risk measures achieve statistically better risk-adjusted performances, especially those based on the Conditional Drawdown at Risk.

ENTER HIERARCHICAL EQUAL RISK CONTRIBUTION

Enter Hierarchical Equal Risk Contribution

Having understood the limitations of HRP, in this section, we will go over each of the steps of the HERC algorithm in a detailed manner.
Hierarchical Tree Clustering

This step breaks down the assets in our portfolio into different hierarchical clusters using the famous Hierarchical Tree Clustering algorithm. Specifically, we calculate the tree clusters based on the TxN matrix of stock returns where T represents the time-series of the data and N represents the number of stocks in our portfolio. Note that this method combines the items into a cluster rather than breaking down a cluster into individual items i.e. it does an agglomerative clustering. Let's understand how the clusters are formed in a step-by-step manner:

1. Given a TxN matrix of stock returns, calculate the correlation of each stock's returns with the other stocks which gives us an NxN matrix of these correlations,

2. The correlation matrix is converted to a correlation-distance matrix D, where,

$$(i, j) = \sqrt{0.5 * (1 - \rho(i, j))}$$

3. Now, we calculate another distance matrix $\overline{D}$ where,

$$\overline{D}(i, j) = \sqrt{\sum_{k=1}^{N} (D(k, i) - D(k, j))^2}$$

It is formed by taking the Euclidean distance between all the columns in a pair-wise manner.

4. A quick explanation regarding the difference between D and $\overline{D}$ – for two assets i and j,

$D(i, j)$ is the distance between the two assets while $\overline{D}(i, j)$ indicates the closeness in similarity of these assets with the rest of the portfolio. This becomes obvious when we look at the

formula for calculating $\overline{D}$ – we sum over the squared difference of distances of i and j from the other stocks. Hence, a lower value means that assets i and j are similarly correlated with the other stocks in our portfolio.

5. We start forming clusters of assets using these distances in a recursive manner. Let us denote the set of clusters as U. The first cluster (i^*, j^*) is calculated as,

$$U[1] = argmin_{(i,j)} \overline{D}(i, j)$$

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

In the above figure, stocks a and b have the minimum distance value and so we combine these two into a cluster.

6. We now update the distance matrix D by calculating the distances of other items from the newly formed cluster. This step is called linkage clustering and there are different ways of doing this. Hierarchical Risk Parity uses single linkage clustering which means the distances between two clusters is defined by a single element pair – those two elements which are closest to each other.

7. We remove the columns and rows corresponding to the new cluster – in this case we remove rows and columns for stocks a and b. For calculating the distance of an asset i outside this cluster, we use the following formula

	(a,b)	c	d	e
(a,b)	0	**21**	**31**	**21**
c	**21**	0	*28*	*39*
d	**31**	*28*	0	*43*
e	**21**	*39*	*43*	0

Using the above formula we calculate distances for c, d and e from cluster (a, b).

$$\overline{D}(c, U[1]) = min(\overline{D}(c, a), \overline{D}(c, b)) = min(21, 30) = 21$$

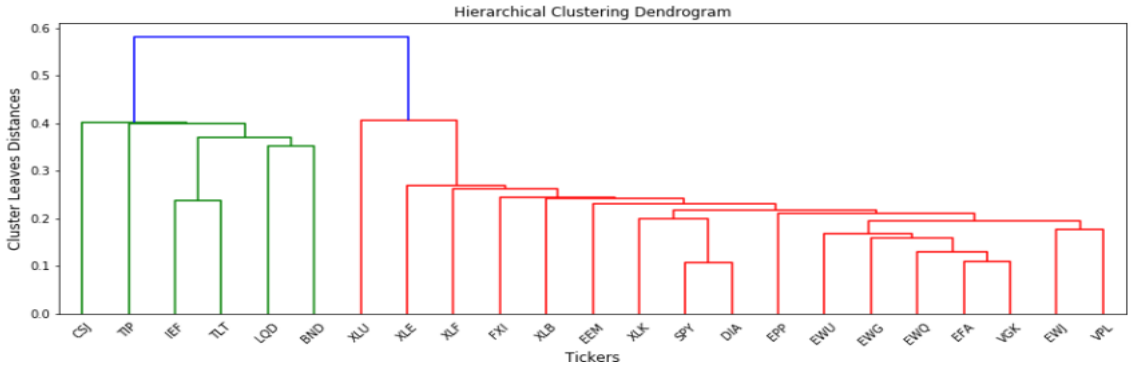
$$\overline{D}(d, U[1]) = min(\overline{D}(d, a), \overline{D}(d, b)) = min(31, 34) = 31$$

$$\overline{D}(e, U[1]) = min(\overline{D}(e, a), \overline{D}(e, b)) = min(23, 21) = 21$$

8. In this way, we go on recursively combining assets into clusters and updating the distance matrix until we are left with one giant cluster of stocks as shown in the following image where we finally combine d with ((a, b), c, e).

	((a,b),c,e)	d
((a,b),c,e)	0	28
d	28	0

9. Finally, in hierarchical clustering, the clusters are always visualised in the form of a nice cluster diagram called dendrogram. Below is the image of the hierarchical clusters for our stock data,



Selecting the Optimal Number of Clusters

Selecting the Optimal Number of Clusters

This is where HERC deviates from the traditional HRP algorithm. The first step grows the tree to its maximum depth and now it is time to prune it by selecting the required number of clusters. Dr. Raffinot has used the famous Gap Index method for this purpose. It was developed by Tibshirani et. al at Stanford University and is a widely used statistic in clustering applications.

Let's say you have a data with k clusters –$C_1, C_2, C_3, \ldots\ldots, C_k$. Then the sum of pairwise distances for a cluster, C_r is given by,

$$D_r = \sum_{i,i' \in C_r} d_{ii'}$$

where $d_{ii'}$ is the Euclidean distance between two data points i and i'. Based on this, the authors calculate the within-cluster sum of squares around the cluster means – W_k (cluster inertia)

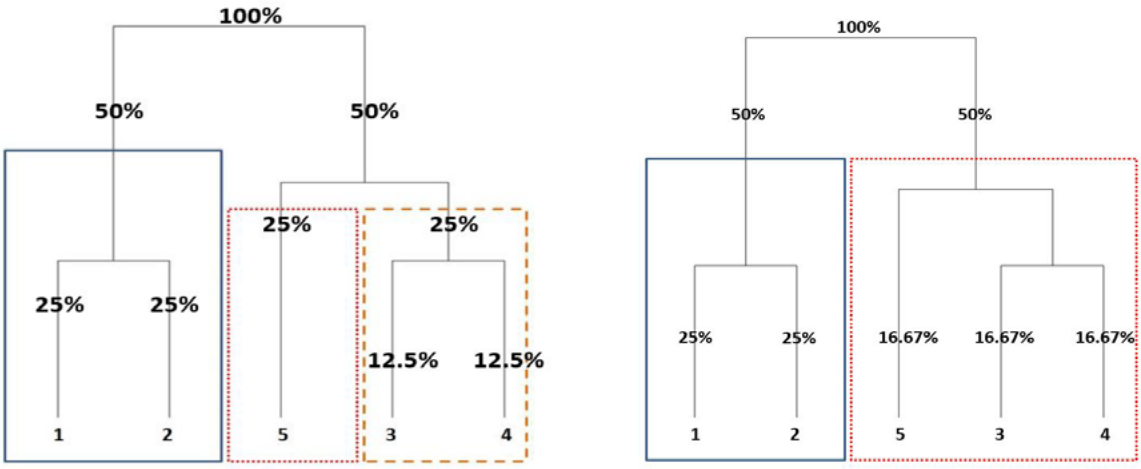
$$W_k = \sum_{r=1}^{k} \frac{D_r}{2n_r}$$

where n_r is the number of data points in the r^{th} cluster. Having calculated the cluster inertia, the Gap statistic is given by the following equation,

$$Gap_n(k) = E_n^*[log(W_k)] \; - \; log(W_k)$$

where E^* is an expectation from some reference distribution. According to the authors, the optimal value of clusters –k^* – will be the one maximising $Gap_n(k)$.

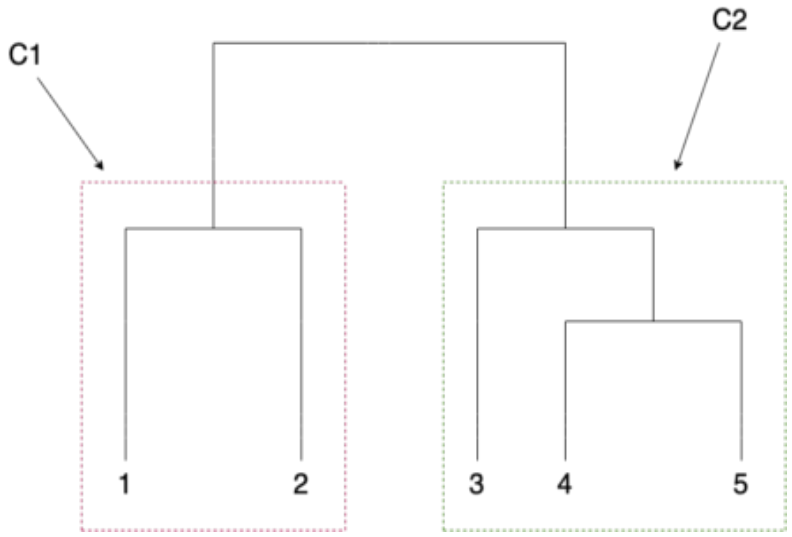
I will not go into the further details about the method and its mathematical motivations, but if you want to delve deeper, we highly suggest you read the original paper – Estimating the Number of Clusters in a Data Set



Pruning the hierarchical tree to the optimal number of clusters not only reduces overfitting but also helps achieve better weight allocations. The above image is taken from HERC paper and illustrates the importance of number of clusters on weight allocations.

Top-down Recursive Bisection

Having found the required number of clusters, this step calculates weights for each of them. Let us use the following diagram as a reference



1. At the top of the tree, we have one big cluster and its weight is 1.

2. We now descend through the dendrogram structure and successively assign weights at each level of the tree. At each point, the tree always bisects into two sub-clusters, let's say – C_1 and C_2 The respective cluster weights are given by the following formulae,

$$CW_{C_1} = \frac{RC_{C_1}}{RC_{C_1} + RC_{C_2}}$$

$$CW_{C_2} = 1 \ - \ CW_{C_1}$$

RC_{C_1} and RC_{C_2} refers to the risk contribution of C_1 and C_2 respectively. Any traditional measure of risk can be used here and HERC currently supports the following ones – **Variance, Standard Deviation, Expected Shortfall (CVaR)** and **Conditional Drawdown at Risk (CDaR)**. The RC for a cluster is the additive risk contribution of all individual assets in that cluster. For example, from

the above figure, if $C_1 = \{1, 2\}$ and $C_2 = \{3, 4, 5\}$, then,

$$RC_{C_1} = RC_1 + RC_2$$

$$RC_{C_2} = RC_3 + RC_4 + RC_5$$

where RC_1 is the risk associated with the 1^{st} asset and so on.

3. Recurse through the tree until all the clusters have been assigned weights.

Naive Risk Parity within Clusters

The last step is to calculate the final asset weights. Let us calculate the weights of assets residing in C_2 i.e. assets 3, 4 and 5.

1. The first step is to calculate the naive **risk parity** weights, W_{NRP}, which uses the inverse-risk allocation to assign weights to assets in a cluster.

$$W^i_{NRP} = \frac{\frac{1}{RC_i}}{\sum_{k=3}^{5} \frac{1}{RC_k}}, i \in \{3, 4, 5\}$$

2. Multiply the risk parity weights of assets with the weight of the cluster in which they reside.

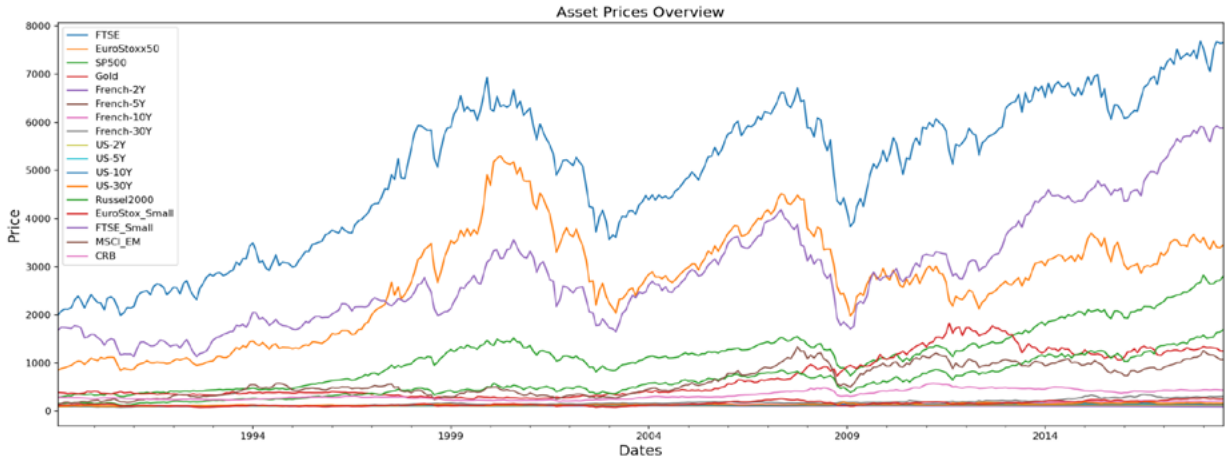
$$W^i_{final} = W^i_{NRP} * CW_{C_2}, i \in \{3, 4, 5\}$$

In this way, the final weights are calculated for assets in all the other clusters.

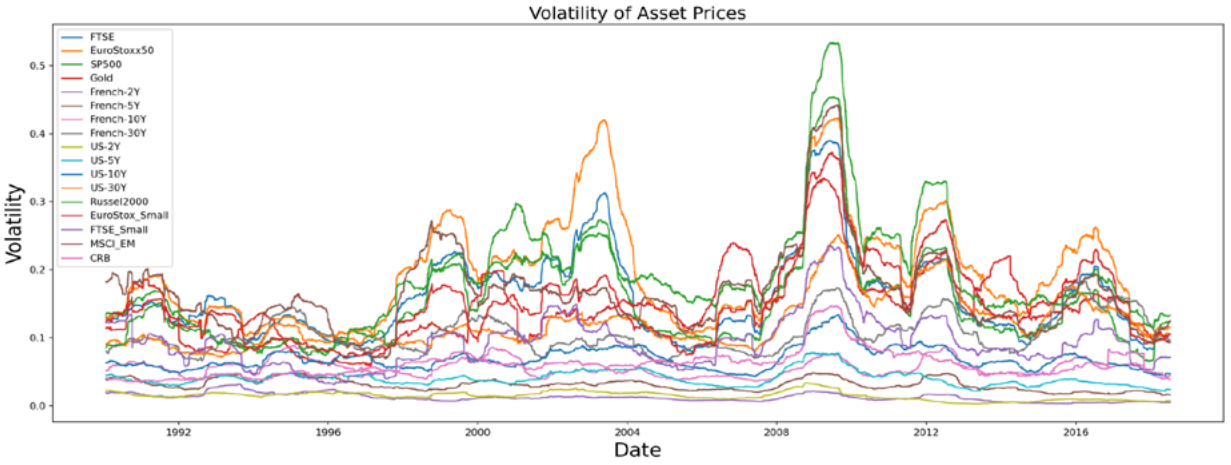
A QUICK COMPARISON OF HRP AND HERC

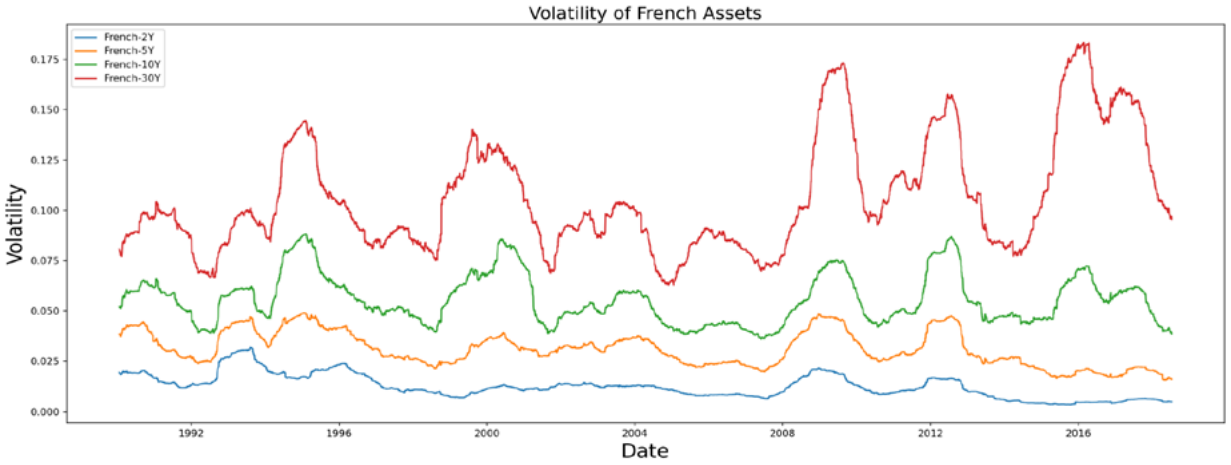
If you are reading this section, then you have already made it through the difficult parts of the article and all that is left is to see the algorithm in action. In this section, we will do a quick comparison of the differences between HRP and HERC by running both the algorithms on real asset data and observing the respective weight allocations. We will be using PortfolioLab's HERC implementation for this example.

We will be using a multi-asset dataset of 17 assets from different asset classes (bonds and commodities) and exhibiting different risk-return characteristics.



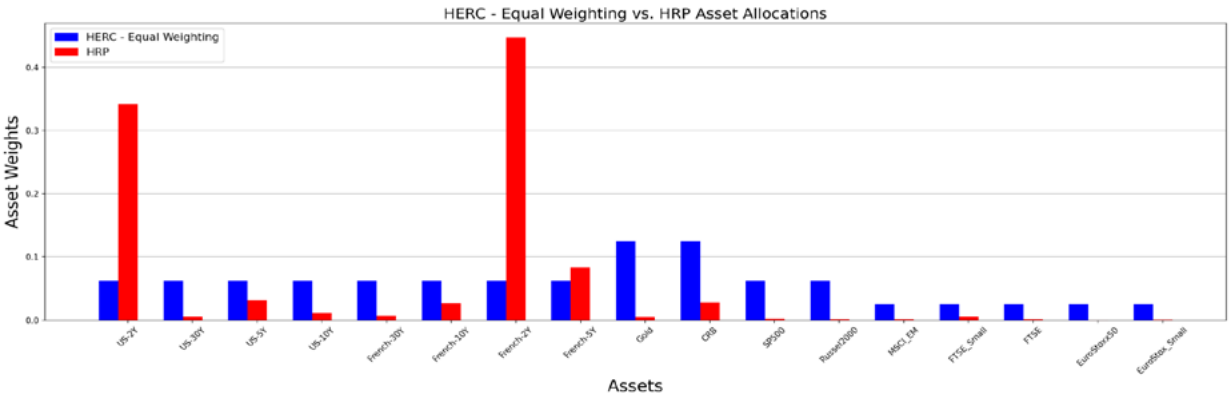
We also give a quick look at the volatilities of these assets,
In particular let us focus on the volatilities of the US and French bonds,





HERC Equal Weighting vs HRP

In his paper, Dr. Raffinot concludes with the following statement – "Hierarchical 1/N is very difficult to beat". Apart from the different risk measures mentioned earlier, HERC can also allocate using the equal weighting metric where the risk is allocated equally among assets in the clusters. Here we have compared HERC's equal weighting allocations with those of HRP's minimum variance. By default, HERC uses Ward clustering (which is what Raffinot has used in his paper) and HRP employs the single linkage clustering.

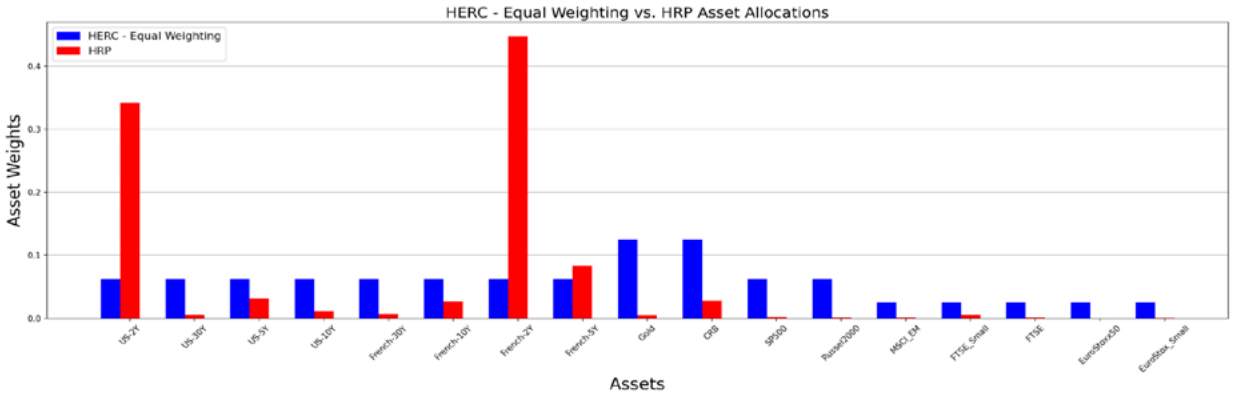


• HRP has allocated most of the weights to US-2Y and French-2Y because these bonds exhibit the least amount of volatility (look at the plot of volatilities). However, the other assets have received lower weights and one can say that the HRP portfolio is sparsely distributed.

• On the other hand, HERC allocated weights equally to all the assets in a cluster. All US bonds, being part of the same cluster, got the same weights and this is true for French bonds too (except French-2Y). Although similar to traditional equal weight portfolios, a major difference in HERC's equal weighting scheme is that only the assets in a cluster compete for these allocations. Instead of all the 17 assets receiving equal weights, the weights are divided unequally among the clusters and then divided equally within them. This ensures that the portfolio remains well diversified by assigned weights to all assets and at the same time making sure that only items within a cluster compete for these allocations.

HERC Minimum Variance vs HRP

In the following comparison, we will change HERC's risk measure to variance. All other parameters stay the same.



While the HRP portfolio remains unchanged, the HERC weights undergo a significant change and are more in sync with the former. You can observe that it has increased the weights for US and French bonds, while decreasing allocations for other assets. Furthermore, there are slight differences between both the portfolios – HERC decreases the weight of US-2Y and spreads it across the cluster to the other US assets. At the same time, it slightly increased the allocation for French-2Y asset as compared to the HRP one.

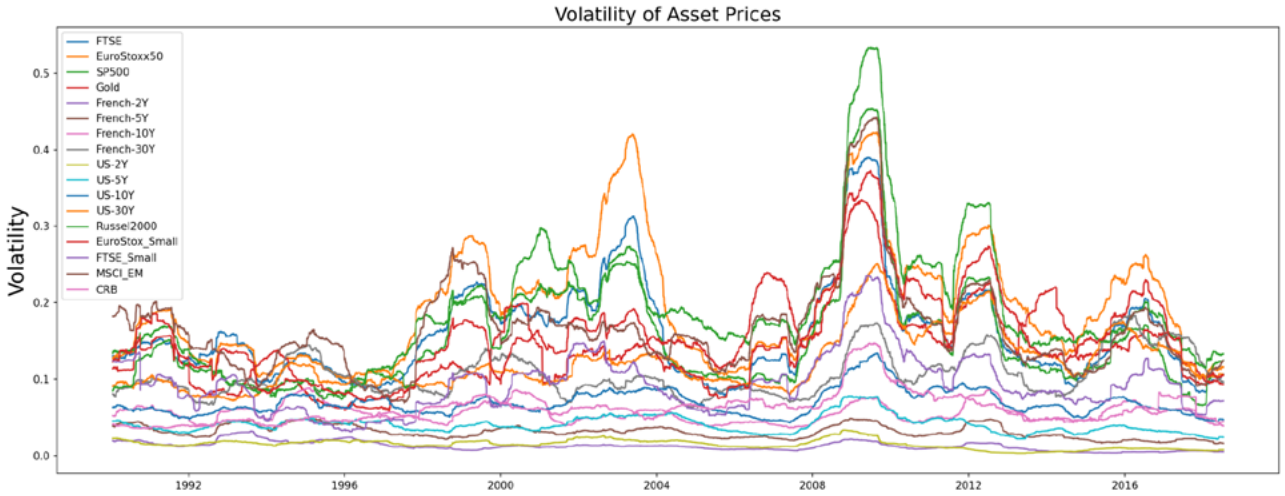
Note that this was a very rudimentary comparison of these two algorithms and in no way a final one. For an in-depth analysis, one should resort to more statistically adjusted methods like Monte Carlo simulations, generating bootstrapped samples of time-series returns etc… However, the purpose of this section was to provide a quick glimpse of how small differences in the working of these algorithms affect the final portfolios generated by them.

CONCLUSION

By building upon the notion of hierarchy introduced by Hierarchical Risk Parity and enhancing the machine learning approach of Hierarchical Clustering based Asset allocation, the Hierarchical Equal Risk Contribution aims at diversifying capital and risk allocation. Selection of appropriate number of clusters and the addition of different risk measures like CVaR and CDaR help in generating better risk-adjusted portfolios with good out-of-sample performance. In my opinion this algorithm is an important addition to the growing list of hierarchical clustering based allocation strategies.

9.0

IMPLEMENTING THE HERC ALGORITHM WITH PORTFOLIOLAB



Harry Markowitz's Modern Portfolio Theory (MPT) was seen as an amazing accomplishment in portfolio optimization, earning him a Nobel Prize for his work. it is based on the hypothesis that investors can optimize their portfolios based on a given level of risk. While this theory works very well mathematically, it fails to translate to real-world investing. This can be mainly attributed to two different reasons:

1. MPT involves the estimation of returns for a given set of assets. Although, accurately estimating returns for a set of assets is very difficult, in which small errors in estimation can cause sub-optimal performance.

2. Traditional optimization methods involve the inversion of a covariance matrix for a set of assets. This matrix inversion leads the algorithm to be susceptible to market volatility and can heavily change the results for small changes in the correlations.

In 2016, Dr. Marcos Lopez de Prado introduced an alternative method for portfolio optimization, the Hierarchical Risk Parity (HRP) algorithm. This algorithm introduced the notion of hierarchy and can be computed in three main steps:

1. **Hierarchical Clustering** – breaks down our assets into hierarchical clusters

2. **Quasi-Diagonalization** – reorganizes the covariance matrix, placing similar assets together

3. **Recursive Bisection** – weights are assigned to each asset in our portfolio

The Hierarchical Risk Parity algorithm laid the foundation for application of hierarchical clustering for asset allocation. In 2017, Thomas Raffinot built off this algorithm and this notion of hierarchy, creating the Hierarchical Clustering Asset Allocation algorithm. This algorithm consists of four main steps:

1. **Hierarchical Clustering**
2. **Selecting the optimal number of clusters**
3. **Capital is allocated across clusters**
4. **Capital is allocated within clusters**

However, in 2018, Raffinot developed the Hierarchical Equal Risk Contribution (HERC) algorithm, combining the machine learning approach of the HCAA algorithm with the recursive bisection approach from the HRP algorithm. The HERC algorithm aims to diversify capital and risk allocations throughout the portfolio and is computed in four main steps:

1. **Hierarchical Clustering**
2. **Selecting the optimal number of clusters**
3. **Recursive Bisection**
4. **Implement Naive Risk Parity within clusters for weight allocations**

Today, we will be exploring the HERC algorithm implemented through the PortfolioLab library. **Please keep in mind that this a tutorial style article and a more detailed explanation of the algorithm and its steps is available here** – Beyond Risk Parity: The Hierarchical Equal Risk Contribution Algorithm

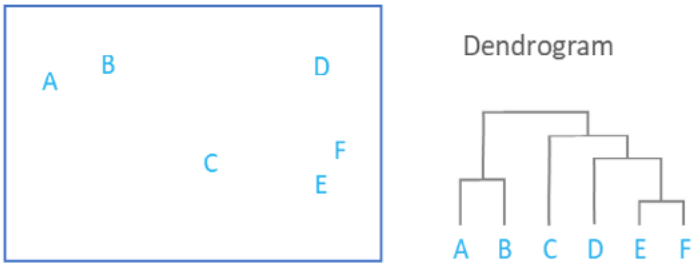
HOW THE HIERARCHICAL EQUAL RISK CONTRIBUTION ALGORITHM WORKS?

In this section, we will go through a quick summary of the steps of the HERC algorithm.

Hierarchical Clustering

Hierarchical clustering is used to place our assets into clusters suggested by the data and not by previously defined metrics. This ensures that the assets in a specific cluster maintain similarity. The objective of this step is to build a hierarchical tree in which our assets are all clustered on different levels. Conceptually, this may be difficult for some to understand, which is why we can visualize this tree through a dendrogram.

The previous image shows the hierarchical clustering process results through a dendrogram.



As the square containining our assets A-F showcases the similarity between each other, we can understand how the assets are clustered. Keep in mind that we are using agglomerative clustering, which assumes each data point to be an individual cluster at the start.

First, the assets E and F are clustered together as they are the most similar. This is followed by the clustering of assets A and B. From this point, the clustering algorithm then includes asset D (and subsequently asset C) into the first clustering pair of assets E and F. Finally, the asset pair A and B is then clustered with the rest of the assets in the last step.

So you now may be asking, how does the algorithm know which assets to cluster together? Of course, we can visually see the distance between each asset, but our algorithm cannot. There are a few widely used methods for calculating the measure of distance/similarity within our algorithm:

- **Single Linkage** – the distance between two clusters it the minimum distance between any two points in the clusters
- **Complete Linkage** – the distance between two clusters is the maximum of the distance between any two points in the clusters
- **Average Linkage** – the distance between two clusters is the average of the distance between any two points in the clusters

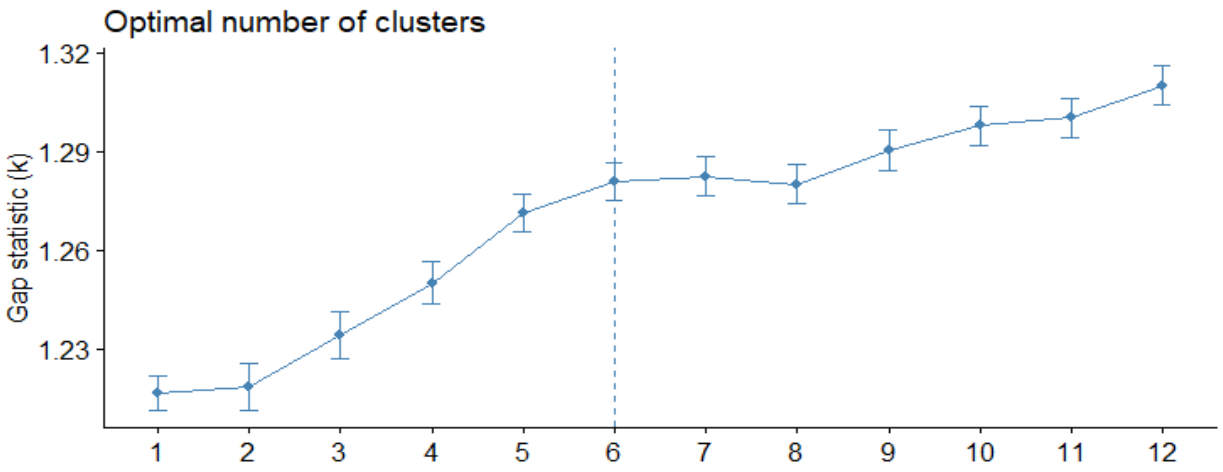
Thankfully, we can easily implement each linkage algorithm within the PortfolioLab library, allowing us to quickly compare the results to each other.

Selecting the Optimal Number of Clusters

Once our assets are all clustered in a hierarchical tree, we run into the problem of not knowing the optimal number of clusters for our weight allocations. We can use the Gap statistic for calculating our optimal number of clusters.

The Gap statistic is used as a measure for calculating our optimal number of clusters. It compares the total within intra-cluster variation for different values of k (with k being the number of clusters) with their expected values under null reference distribution of the data. Informally, this means that we should select the number of clusters which maximizes our Gap statistic before the rate of change begins to slow down.

For example, in the following graph, the optimal number of clusters selected would be 6.



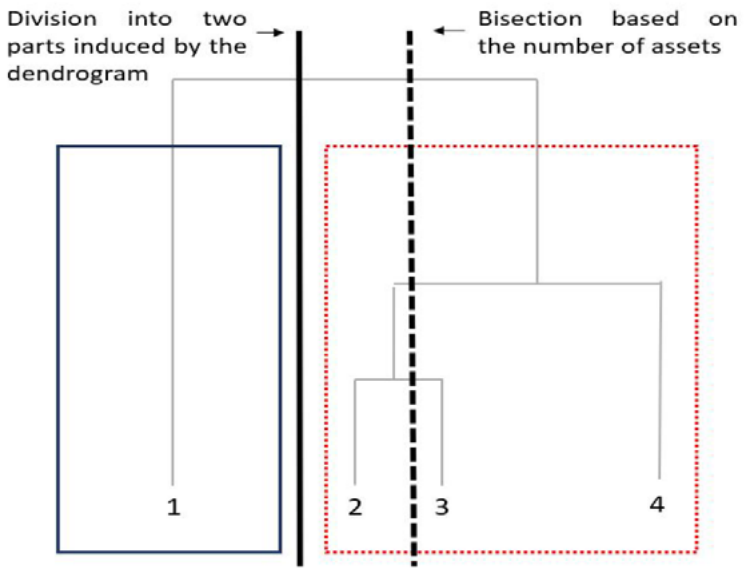
At this stage in the algorithm, we have clustered all our assets into a hierarchical tree and selected the optimal number of clusters using the Gap statistic. We will now move onto recursive bisection.

Recursive Bisection

In this step, portfolio weights for each of the tree clusters are calculated. Based on our dendrogram structure, the recursive bisection step recursively divides our tree following an Equal Risk Contribution allocation. An Equal Risk Contribution allocation makes sure that each asset contributes equally to the portfolio's volatility.

For example, consider a portfolio with asset allocations of 50% in stocks and 50% in bonds. While the asset allocation of this portfolio is equal with respect to our asset classes, our risk allocation is not. Depending on the nature of our stocks and bonds, our risk allocation for this portfolio can be near 90% portfolio risk in stocks and only 10% portfolio risk in bonds. By following an Equal Risk Contribution allocation method, this ensures that the recursive bisection step in our algorithm will distribute portfolio weights equally in terms of risk allocation and not in terms of asset allocation.
In the following image, we can see how recursive bisection splits the dendrogram by structure, and not based

on assets. This is a key part of the HERC algorithm and is one of the reasons as to how HERC distinguishes itself from the HRP algorithm which breaks down the tree based on the number of assets



Naive Risk Parity for Weight Allocations

Having calculated the cluster weights in the previous step, this step calculates the final asset weights. Within the same cluster, an initial set of weights – w_{NRP} – is calculated using the naive risk parity allocation. In this approach, assets are allocated weights in proportion to the inverse of their respective risk; higher risk assets will receive lower portfolio weights, and lower risk assets will receive higher weights. Here the risk can be quantified in different ways – variance, CVaR, CDaR, max daily loss etc… The final weights are given by the following equation:

$$w^i_{final} = w^i_{NRP} * C^i,\ i \in Clusters$$

where, w^i_{NRP} refers to naive risk parity weights of assets in the i^{th} cluster and C^i is the weight of the i^{th} cluster calculated in Step-3.

USING PORTFOLIOLAB'S HERC IMPLEMENTATION

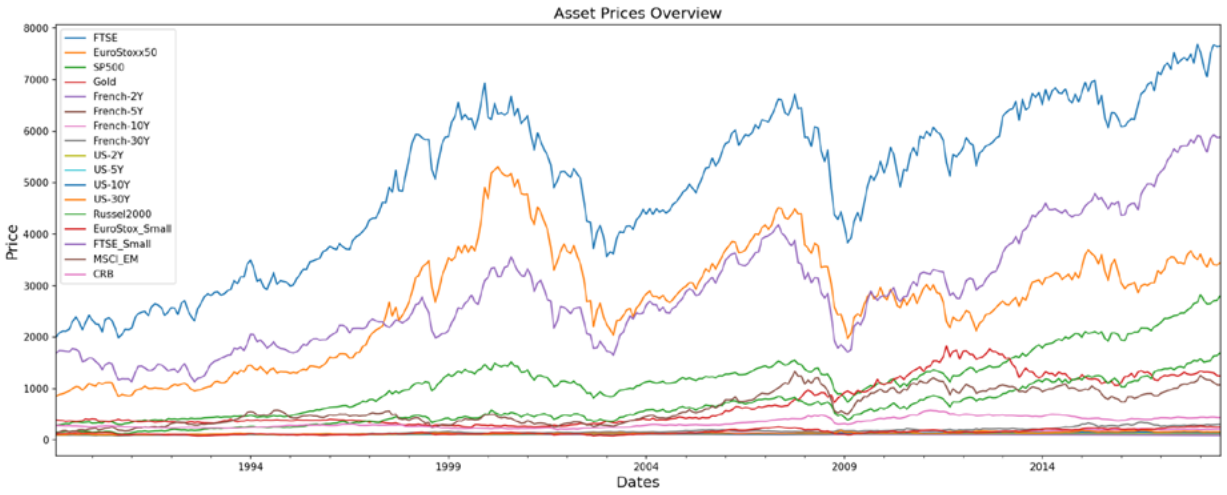
In the example below, we simulate a pair of co-moving assets using an implementation from **ArbitrageLab**. The first plot shows the co-moving assets whilst the bottom one illustrates how a mean-reverting spread is created by going long the first asset and short 60% in the second. It is this spread that can be traded without needing to make any forecasts about the future.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import mlfinlab as ml

import matplotlib.patches as mpatches

from portfoliolab.clustering import HierarchicalEqualRiskContribution
```

Choosing the Dataset

In this example, we will be working with historical closing-price data for 17 assets. The portfolio consists of diverse set of assets ranging from commodities to bonds and each asset exhibits different risk-return characteristics. You can download the dataset CSV file from our research repository here

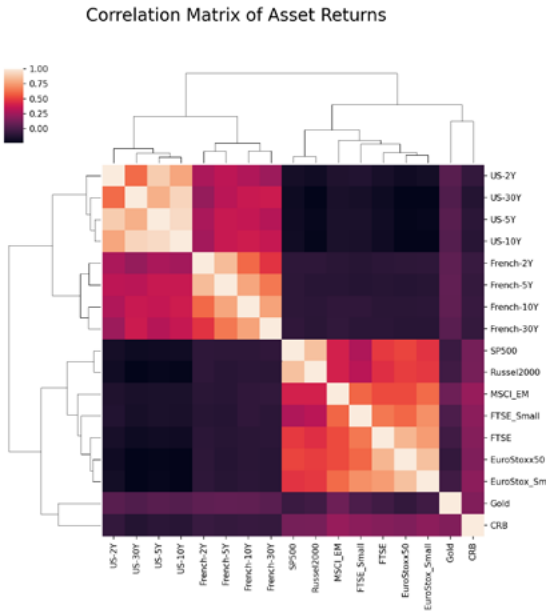
```
# reading in our data

raw_prices = pd.read_csv('assetalloc.csv', sep=';', parse_dates=True, index_col='Dates')

stock_prices = raw_prices.sort_values(by='Dates')

stock_prices.head()stock_prices.resample('M').last().plot(figsize=(17,7))

plt.ylabel('Price', size=15)

plt.xlabel('Dates', size=15)

plt.title('Stock Data Overview', size=15)

plt.show()
```



The specific set of assets was chosen for two reasons:

1. This is the same dataset used by Dr. Raffinot in the original HERC paper. By testing our implementation on the same data, we wanted to make sure that the results are in sync with the original results. This ensures the correctness of PortfolioLab's implementation.

2. The specific set of securities have very good clustering structure which is important for hierarchical based algorithms. If you look at the visual representation of the correlation matrix below, the inherent clusters can be clearly identified.



Calculating the Optimal Weight Allocations

Now that we have our data loaded in, we can make use of the HierarchicalEqualRiskContribution class from PortfolioLab to construct our optimized portfolio. First we must instantiate our class and then run the allocate() method to optimize our portfolio.

Keep in mind that PortfolioLab currently supports the following metrics for calculating weight allocations:

1. **'variance'**: The variance of the clusters is used as a risk metric
2. **'standard_deviation'**: The standard deviation of the clusters is used as a risk metric
3. **'equal_weighting'**: All clusters are weighted equally with respect to the risk
4. **'expected_shortfall'**: The expected shortfall of the clusters is used as a risk metric
5. **'conditional_drawdown_risk'**: The conditional drawdown at risk of the clusters is used as a risk metric

PortfolioLab also supports all four linkage algorithms discussed in this post, though the default method is set as the Ward Linkage algorithm.

The **allocate()** method for the **HierarchicalEqualRiskContribution** object requires three parameters to run:

1. **asset_names** (a list of strings containing the asset names)
2. **asset_prices** (a dataframe of historical asset prices – daily close)
3. **risk_measure** (the type of risk representation to use – PortfolioLab currently supports 5 different solutions)

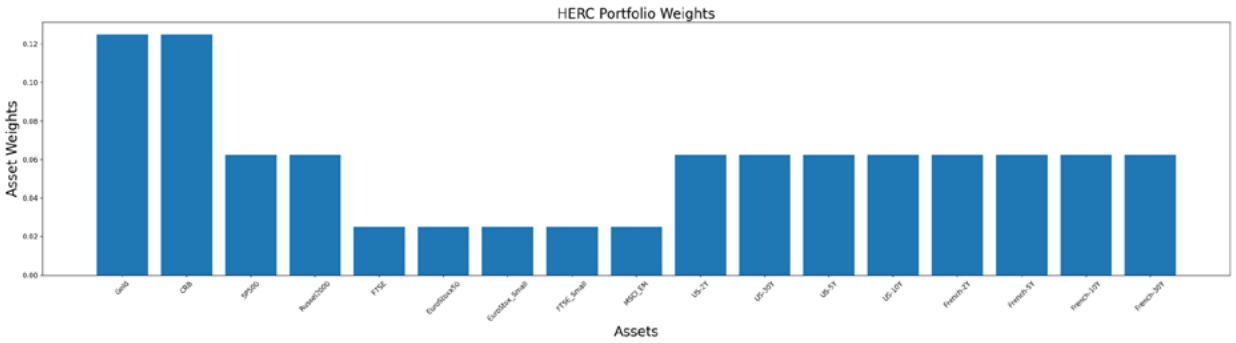
Note: The list of asset names is not a necessary parameter. If your input data is in the form of a dataframe, it will use the column names as the default asset names.

Users can also specify:

1. **The type of linkage algorithm (shown below)**
2. **The confidence level used for calculating expected shortfall and conditional drawdown at risk**
3. **The optimal number of clusters for clustering**

For simplicity, we will only be working with the three required parameters and also specifying our linkage algorithm of choice. **In the example shown below, we are using an equal_weighting solution and the Ward Linkage algorithm.**

```
herc = HierarchicalEqualRiskContribution()


herc.allocate(asset_names=stock_prices.columns,
              asset_prices=stock_prices,
              risk_measure="equal_weighting",
              linkage="ward")


# plotting our optimal portfolio


herc_weights = herc.weights
y_pos = np.arange(len(herc_weights.columns))
plt.figure(figsize=(25,7))
plt.bar(list(herc_weights.columns), herc_weights.values[0])
plt.xticks(y_pos, rotation=45, size=10)
plt.xlabel('Assets', size=20)
plt.ylabel('Asset Weights', size=20)
plt.title('HERC Portfolio Weights', size=20)


plt.show()
```

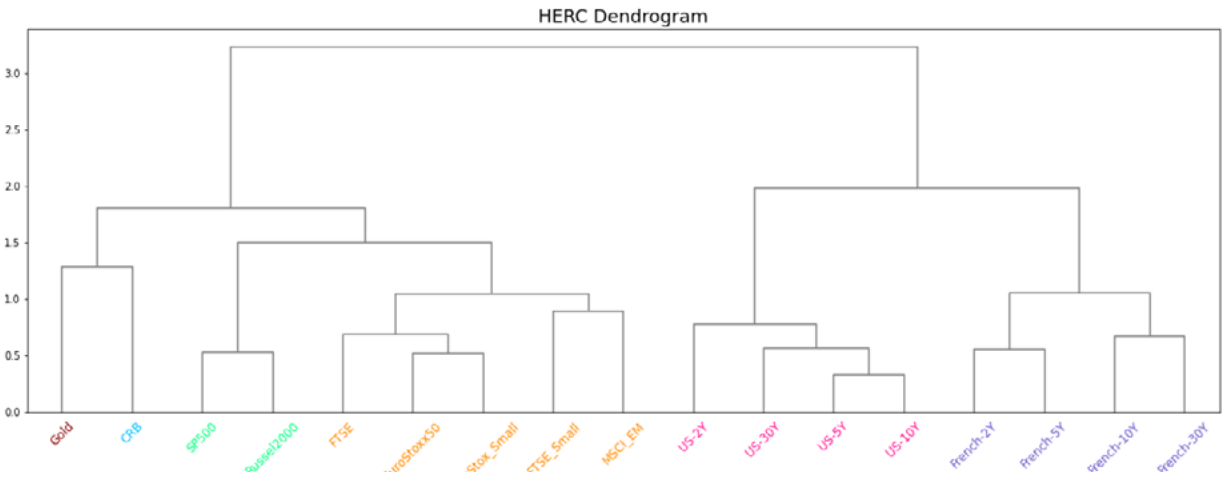


We can observe from the above plot that many assets have been assigned equal weights. However, there is still a visible difference in the allocations. Let us see the reason behind this by visualising the tree structure.

Plotting the Clusters

The clusters can be visualised by calling the plot_clusters() method. This will plot the dendrogram tree which is the standard way of visualising hierarchical clusters.

```
plt.figure(figsize=(17,7))

herc.plot_clusters(assets=stock_prices.columns)

plt.title('HERC Dendrogram', size=18)

plt.xticks(rotation=45)

plt.show()
```



In this graph, the different colors in the x-labels represent the clusters that the stocks belong to. **Through visual analysis, we can count that the optimal number of clusters determined by the Gap statistic is 6 for our dataset.**

We can also confirm this by printing out our optimal number of clusters through the 'optimal_num_clusters' attribute.

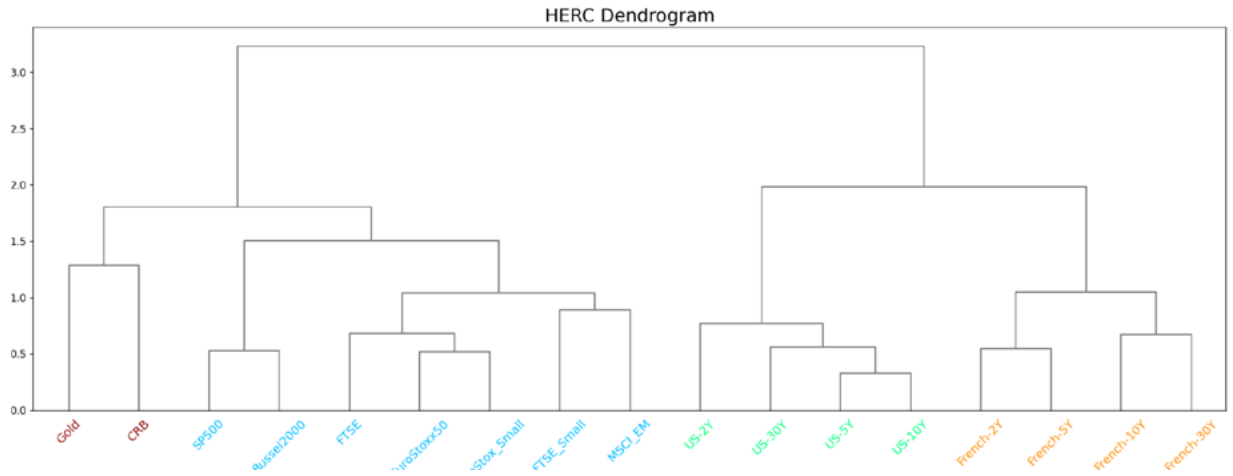
```
print("Optimal Number of Clusters: " + str(herc.optimal_num_clusters))
```

```
Optimal Number of Clusters: 6
```

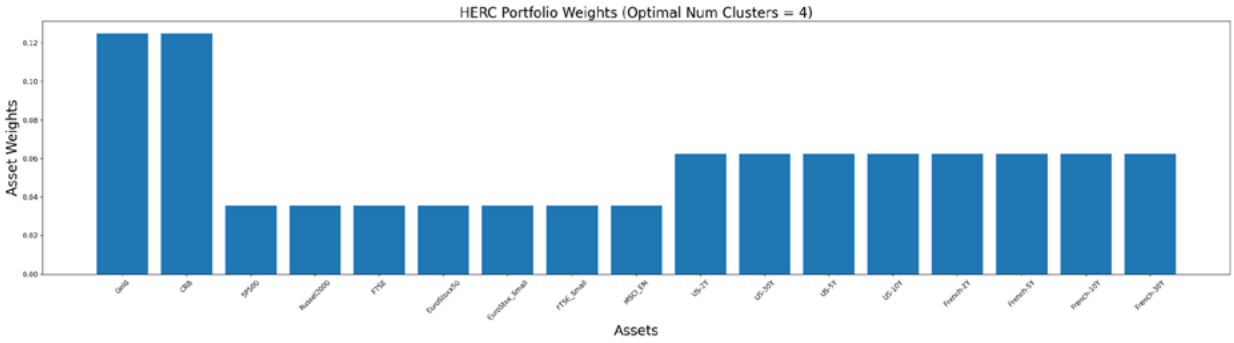
This shows why some assets received similar weights. At each point of bisection, the left and right clusters are assigned equal weights. This results in both the US and French clusters getting the same weight allocation which is then distributed equally among all assets. Hence, all the US and French assets have the same weight allocations.

Changing the optimal number of clusters can lead to very different weight allocations. You can play around with this number by passing different values through the **optimal_num_clusters** parameter and understanding how the weights change. For example, in the following code snippet, we look at the clusters identified by the algorithm when the optimal number of clusters is 4.

```
herc = HierarchicalEqualRiskContribution()


herc.allocate(asset_prices=stock_prices,

              optimal_num_clusters=4,

              risk_measure="equal_weighting")


# Plot clusters


plt.figure(figsize=(17,7))

herc.plot_clusters(assets=stock_prices.columns)

plt.title('HERC Dendrogram', size=18)

plt.xticks(rotation=45)

plt.show()


# Plotting optimal portfolio


herc_weights = herc.weights

y_pos = np.arange(len(herc_weights.columns))

plt.figure(figsize=(25,7))

plt.bar(list(herc_weights.columns), herc_weights.values[0])

plt.xticks(y_pos, rotation=45, size=10)

plt.xlabel('Assets', size=20)

plt.ylabel('Asset Weights', size=20)

plt.title('HERC Portfolio Weights', size=20)


plt.show()
```



You can observe that Gold and CRB have now been grouped into one cluster. Similarly, SP500 and Russel2000 are now in the same cluster as the other assets like FTSE and MSCI. Due to this, the weight allocations have also changed slightly, as can be observed in the following figure.



USING CUSTOM INPUT WITH PORTFOLIOLAB

PortfolioLab also provides users with a lot of customizability when it comes to creating their optimal portfolios. Instead of providing the raw historical closing prices for the assets, users can input the asset returns, a covariance matrix of asset returns, and expected asset returns to calculate their optimal portfolio.

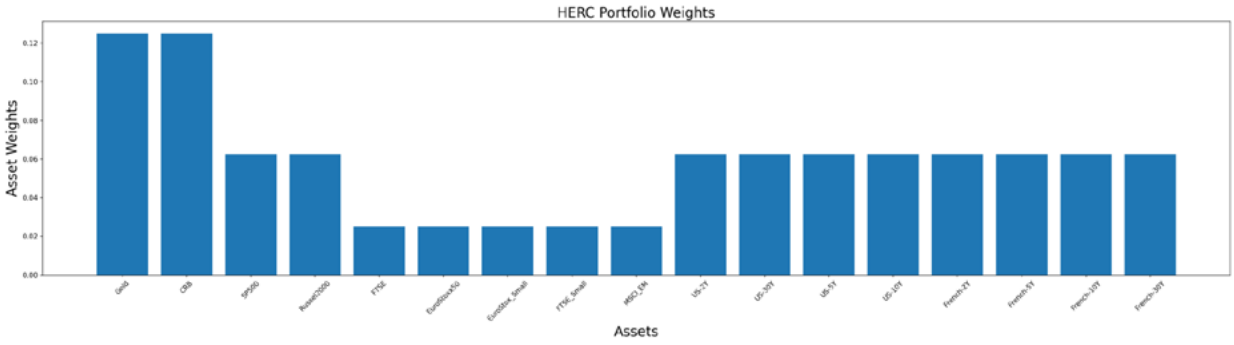
The following parameters in the allocate() method are utilized in order to construct a custom use case:

1. **'asset_returns'**: (pd.DataFrame/NumPy matrix) A matrix of asset returns
2. **'covariance_matrix'**: (pd.DataFrame/NumPy matrix) A covariance matrix of asset returns

In this example, we will be constructing the same optimized portfolio as the first example, utilizing an **equal_weighting** solution with the Ward Linkage algorithm. As we already know the optimal number of clusters, we will also be passing that in as a parameter to save on computation time.

To make some of the necessary calculations, we will make use of the ReturnsEstimators class provided by PortfolioLab.

```
# importing ReturnsEstimation class from PortfolioLab
from portfoliolab.estimators import ReturnsEstimators


# calculating our asset returns
returns = ReturnsEstimators.calculate_returns(stock_prices)


# calculating our covariance matrix
cov = returns.cov()


# from here, we can now create our portfolio
herc_custom = HierarchicalEqualRiskContribution()


herc_custom.allocate(asset_returns=returns,
                     covariance_matrix=cov,
                     risk_measure='equal_weighting',
                     linkage='ward',
                     optimal_num_clusters=6)
```



You can observe that these are exactly the same portfolio we got when passing raw asset prices. You can further play around with the different risk measures mentioned previously and generate different types of portfolios.

CONCLUSION

Through this article, we learned the intuition behind Thomas Raffinot's Hierarchical Equal Risk Contribution portfolio optimization algorithm and also saw how we can utilize PortfolioLab's implementation to apply this technique out-of-the-box. HERC is a powerful algorithm that can produce robust portfolios which avoids many of the problems seen with Modern Portfolio Theory and Hierarchical Risk Parity.

We want to thank everybody in Hudson&Thames team for making this book possible!

ADITYA VYAS
Co-Author of MlFinLab and PortfolioLab

ILLYA BARZIY
Quantitatve Research Team Lead at Hudson & Thames

VALERIIA PERVUSHYNA
Quantitative Researcher at Hudson & Thames

URI LEE
Researcher at Hudson & Thames


HUDSON
AND THAMES